HAR LSTM Assignment

In [26]:
```
# Importing Libraries
import pandas as pd
import numpy as np
import scipy

from tensorflow.compat.v1 import ConfigProto
from tensorflow.compat.v1 import InteractiveSession

config = ConfigProto()
config.gpu_options.per_process_gpu_memory_fraction = 0.33
session = InteractiveSession(config=config)

import keras
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import BatchNormalization
from keras.layers import Dropout,Flatten,Conv1D
from keras.layers import Dense
from sklearn.metrics import accuracy_score

from keras.callbacks import EarlyStopping, CSVLogger, TensorBoard,ReduceLROnPl
ateau
```

```
C:\Users\family\Anaconda3\lib\site-packages\tensorflow\python\client\sessio
n.py:1735: UserWarning: An interactive session is already active. This can c
ause out-of-memory errors in some cases. You must explicitly call `Interacti
veSession.close()` to release resources held by the other session(s).
  warnings.warn('An interactive session is already active. This can '
```

In [27]:
```
# Activities are the class labels
# It is a 6 class classification
ACTIVITIES = {
    0: 'WALKING',
    1: 'WALKING_UPSTAIRS',
    2: 'WALKING_DOWNSTAIRS',
    3: 'SITTING',
    4: 'STANDING',
    5: 'LAYING',
}

# Utility function to print the confusion matrix
def confusion_matrix(Y_true, Y_pred):
    Y_true = pd.Series([ACTIVITIES[y] for y in np.argmax(Y_true, axis=1)])
    Y_pred = pd.Series([ACTIVITIES[y] for y in np.argmax(Y_pred, axis=1)])

    return pd.crosstab(Y_true, Y_pred, rownames=['True'], colnames=['Pred'])
```

Data

In [28]:
```
# Data directory
DATADIR = 'UCI_HAR_Dataset'
```

In [29]:
```
# 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 [30]:
```
# Utility function to read the data from csv file
def _read_csv(filename):
    return pd.read_csv(filename, delim_whitespace=True, header=None)

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

    for signal in SIGNALS:
        filename = f'UCI_HAR_Dataset/{subset}/Inertial Signals/{signal}_{subse
t}.txt'
        signals_data.append(
            _read_csv(filename).as_matrix()
        )

    # Transpose is used to change the dimensionality of the output,
    # aggregating the signals by combination of sample/timestep.
    # Resultant shape is (7352 train/2947 test samples, 128 timesteps, 9 signa
ls)
    return np.transpose(signals_data, (1, 2, 0))
```

In [31]:
```
def load_y(subset):
    """
    The objective that we are trying to predict is a integer, from 1 to 6,
    that represents a human activity. We return a binary representation of
    every sample objective as a 6 bits vector using One Hot Encoding
    (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.get_dummie
s.html)
    """
    filename = f'UCI_HAR_Dataset/{subset}/y_{subset}.txt'
    y = _read_csv(filename)[0]

    return pd.get_dummies(y).as_matrix()
```

In [32]:
```
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 [33]:
```
# Loading the train and test data
X_train, X_test, Y_train, Y_test = load_data()
```

```
C:\Users\family\Anaconda3\lib\site-packages\ipykernel_launcher.py:12: Future
Warning: Method .as_matrix will be removed in a future version. Use .values
instead.
  if sys.path[0] == '':
C:\Users\family\Anaconda3\lib\site-packages\ipykernel_launcher.py:11: Future
Warning: Method .as_matrix will be removed in a future version. Use .values
instead.
  # This is added back by InteractiveShellApp.init_path()
```

1. Simple 2 layer LSTM

In [79]:
```
# Initiliazing the sequential model
model = Sequential()

# Configuring the parameters

# 1st Layer of LSTM
model.add(LSTM(128, input_shape=(128, 9),return_sequences=True,kernel_initiali
zer='glorot_normal'))
# Adding a dropout layer
model.add(Dropout(0.2))

# 2nd Layer of LSTM
model.add(LSTM(64,kernel_initializer='glorot_normal'))
model.add(Dropout(0.5))

# Adding a dense output layer with sigmoid activation
model.add(Dense(6, activation='sigmoid'))
model.summary()

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

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_1 (LSTM)                (None, 128, 128)          70656     
_________________________________________________________________
dropout_5 (Dropout)          (None, 128, 128)          0         
_________________________________________________________________
lstm_2 (LSTM)                (None, 64)                49408     
_________________________________________________________________
dropout_6 (Dropout)          (None, 64)                0         
_________________________________________________________________
dense_5 (Dense)              (None, 6)                 390       
=================================================================
Total params: 120,454
Trainable params: 120,454

Non-trainable params: 0
_________________________________________________________________
```

In [80]:
```python
# Training the model
model.fit(X_train,
          Y_train,
          batch_size=75,
          validation_data=(X_test, Y_test),epochs=100)
```

```
WARNING:tensorflow:From C:\Users\family\Anaconda3\lib\site-packages\tensorfl
ow\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 futu
re version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
Train on 7352 samples, validate on 2947 samples
Epoch 1/100
7352/7352 [==============================] - 22s 3ms/step - loss: 1.2818 - a
ccuracy: 0.4702 - val_loss: 1.1241 - val_accuracy: 0.5073
Epoch 2/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.9431 - a
ccuracy: 0.5861 - val_loss: 0.8120 - val_accuracy: 0.6817
Epoch 3/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.7796 - a
ccuracy: 0.6733 - val_loss: 0.7196 - val_accuracy: 0.6899
Epoch 4/100
7352/7352 [==============================] - 21s 3ms/step - loss: 0.6075 - a
ccuracy: 0.7582 - val_loss: 0.5810 - val_accuracy: 0.7496
Epoch 5/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.4916 - a
ccuracy: 0.7904 - val_loss: 0.5486 - val_accuracy: 0.7706
Epoch 6/100
7352/7352 [==============================] - 21s 3ms/step - loss: 0.4150 - a
ccuracy: 0.8271 - val_loss: 0.5966 - val_accuracy: 0.7862
Epoch 7/100
7352/7352 [==============================] - 21s 3ms/step - loss: 0.3631 - a
ccuracy: 0.8912 - val_loss: 0.3594 - val_accuracy: 0.8731
Epoch 8/100
7352/7352 [==============================] - 21s 3ms/step - loss: 0.2398 - a
ccuracy: 0.9263 - val_loss: 0.3678 - val_accuracy: 0.8799
Epoch 9/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.2137 - a
ccuracy: 0.9344 - val_loss: 0.2750 - val_accuracy: 0.8965
Epoch 10/100
7352/7352 [==============================] - 21s 3ms/step - loss: 0.1792 - a
ccuracy: 0.9389 - val_loss: 0.2374 - val_accuracy: 0.9138
Epoch 11/100
7352/7352 [==============================] - 21s 3ms/step - loss: 0.2094 - a
ccuracy: 0.9229 - val_loss: 0.2595 - val_accuracy: 0.9046
Epoch 12/100
7352/7352 [==============================] - 21s 3ms/step - loss: 0.1605 - a
ccuracy: 0.9422 - val_loss: 0.5527 - val_accuracy: 0.8609
Epoch 13/100
7352/7352 [==============================] - 21s 3ms/step - loss: 0.1922 - a
ccuracy: 0.9317 - val_loss: 0.3882 - val_accuracy: 0.8697
Epoch 14/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1607 - a
ccuracy: 0.9403 - val_loss: 0.3212 - val_accuracy: 0.8972
Epoch 15/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1598 - a
ccuracy: 0.9438 - val_loss: 0.7885 - val_accuracy: 0.8056
Epoch 16/100

7352/7352 [==============================] - 22s 3ms/step - loss: 0.2009 - a
```

```
ccuracy: 0.9310 - val_loss: 0.4235 - val_accuracy: 0.8714
Epoch 17/100
7352/7352 [==============================] - 21s 3ms/step - loss: 0.1624 - a
ccuracy: 0.9436 - val_loss: 0.2887 - val_accuracy: 0.8996
Epoch 18/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1361 - a
ccuracy: 0.9475 - val_loss: 0.2749 - val_accuracy: 0.9128
Epoch 19/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1374 - a
ccuracy: 0.9497 - val_loss: 0.2438 - val_accuracy: 0.9203
Epoch 20/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1252 - a
ccuracy: 0.9501 - val_loss: 0.2404 - val_accuracy: 0.9253
Epoch 21/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1454 - a
ccuracy: 0.9452 - val_loss: 0.4725 - val_accuracy: 0.8741
Epoch 22/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1412 - a
ccuracy: 0.9501 - val_loss: 0.2898 - val_accuracy: 0.9091
Epoch 23/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1316 - a
ccuracy: 0.9516 - val_loss: 0.3423 - val_accuracy: 0.9063
Epoch 24/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1241 - a
ccuracy: 0.9501 - val_loss: 0.3153 - val_accuracy: 0.9067
Epoch 25/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1290 - a
ccuracy: 0.9513 - val_loss: 0.2757 - val_accuracy: 0.9070
Epoch 26/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1216 - a
ccuracy: 0.9531 - val_loss: 0.4003 - val_accuracy: 0.8789
Epoch 27/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1226 - a
ccuracy: 0.9535 - val_loss: 0.2877 - val_accuracy: 0.9108
Epoch 28/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1146 - a
ccuracy: 0.9574 - val_loss: 0.2977 - val_accuracy: 0.9097
Epoch 29/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1183 - a
ccuracy: 0.9559 - val_loss: 0.2960 - val_accuracy: 0.9125
Epoch 30/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1144 - a
ccuracy: 0.9561 - val_loss: 0.2954 - val_accuracy: 0.9111
Epoch 31/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1143 - a
ccuracy: 0.9553 - val_loss: 0.3298 - val_accuracy: 0.9148
Epoch 32/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1433 - a
ccuracy: 0.9523 - val_loss: 0.3059 - val_accuracy: 0.9118
Epoch 33/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1254 - a
ccuracy: 0.9524 - val_loss: 0.2731 - val_accuracy: 0.9203
Epoch 34/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1125 - a
ccuracy: 0.9548 - val_loss: 0.3178 - val_accuracy: 0.9233
Epoch 35/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1105 - a
ccuracy: 0.9548 - val_loss: 0.2981 - val_accuracy: 0.9203
Epoch 36/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1166 - a
ccuracy: 0.9517 - val_loss: 0.4157 - val_accuracy: 0.9013
Epoch 37/100
```

```
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1289 - a
ccuracy: 0.9509 - val_loss: 0.4628 - val_accuracy: 0.8884
Epoch 38/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.2523 - a
ccuracy: 0.9251 - val_loss: 0.3886 - val_accuracy: 0.8941
Epoch 39/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.2149 - a
ccuracy: 0.9329 - val_loss: 0.3416 - val_accuracy: 0.9084
Epoch 40/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1144 - a
ccuracy: 0.9553 - val_loss: 0.3218 - val_accuracy: 0.9125
Epoch 41/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1134 - a
ccuracy: 0.9523 - val_loss: 0.3552 - val_accuracy: 0.9026
Epoch 42/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1149 - a
ccuracy: 0.9539 - val_loss: 0.3364 - val_accuracy: 0.9121
Epoch 43/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1308 - a
ccuracy: 0.9505 - val_loss: 0.3073 - val_accuracy: 0.9179
Epoch 44/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1180 - a
ccuracy: 0.9527 - val_loss: 0.3540 - val_accuracy: 0.9067
Epoch 45/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1968 - a
ccuracy: 0.9321 - val_loss: 0.3570 - val_accuracy: 0.9030
Epoch 46/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1646 - a
ccuracy: 0.9433 - val_loss: 0.3478 - val_accuracy: 0.9087
Epoch 47/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1154 - a
ccuracy: 0.9543 - val_loss: 0.3401 - val_accuracy: 0.9111
Epoch 48/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1112 - a
ccuracy: 0.9550 - val_loss: 0.3364 - val_accuracy: 0.9080
Epoch 49/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1065 - a
ccuracy: 0.9570 - val_loss: 0.3581 - val_accuracy: 0.9070
Epoch 50/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1058 - a
ccuracy: 0.9542 - val_loss: 0.3661 - val_accuracy: 0.9111
Epoch 51/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1180 - a
ccuracy: 0.9521 - val_loss: 0.5082 - val_accuracy: 0.8935
Epoch 52/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1227 - a
ccuracy: 0.9484 - val_loss: 0.3639 - val_accuracy: 0.9203
Epoch 53/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1098 - a
ccuracy: 0.9506 - val_loss: 0.3505 - val_accuracy: 0.9213
Epoch 54/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1084 - a
ccuracy: 0.9551 - val_loss: 0.4031 - val_accuracy: 0.9189
Epoch 55/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1133 - a
ccuracy: 0.9544 - val_loss: 0.2353 - val_accuracy: 0.9287
Epoch 56/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1226 - a
ccuracy: 0.9523 - val_loss: 0.2549 - val_accuracy: 0.9213
Epoch 57/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1093 - a
ccuracy: 0.9542 - val_loss: 0.2458 - val_accuracy: 0.9243
```

```
Epoch 58/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1056 - a
ccuracy: 0.9558 - val_loss: 0.3888 - val_accuracy: 0.9040
Epoch 59/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1160 - a
ccuracy: 0.9553 - val_loss: 0.3312 - val_accuracy: 0.9057
Epoch 60/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1067 - a
ccuracy: 0.9531 - val_loss: 0.3546 - val_accuracy: 0.9121
Epoch 61/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1187 - a
ccuracy: 0.9483 - val_loss: 0.3328 - val_accuracy: 0.9216
Epoch 62/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1028 - a
ccuracy: 0.9572 - val_loss: 0.3358 - val_accuracy: 0.9101
Epoch 63/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1031 - a
ccuracy: 0.9563 - val_loss: 0.3775 - val_accuracy: 0.9108
Epoch 64/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1042 - a
ccuracy: 0.9557 - val_loss: 0.3862 - val_accuracy: 0.9148
Epoch 65/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1159 - a
ccuracy: 0.9501 - val_loss: 0.2362 - val_accuracy: 0.9298
Epoch 66/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1104 - a
ccuracy: 0.9546 - val_loss: 0.2757 - val_accuracy: 0.9253
Epoch 67/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1005 - a
ccuracy: 0.9551 - val_loss: 0.2939 - val_accuracy: 0.9203
Epoch 68/100
7352/7352 [==============================] - 21s 3ms/step - loss: 0.1233 - a
ccuracy: 0.9539 - val_loss: 0.3077 - val_accuracy: 0.9135
Epoch 69/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1055 - a
ccuracy: 0.9547 - val_loss: 0.3305 - val_accuracy: 0.9101
Epoch 70/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1032 - a
ccuracy: 0.9553 - val_loss: 0.3142 - val_accuracy: 0.9209
Epoch 71/100
7352/7352 [==============================] - 21s 3ms/step - loss: 0.0956 - a
ccuracy: 0.9577 - val_loss: 0.3167 - val_accuracy: 0.9206
Epoch 72/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1007 - a
ccuracy: 0.9542 - val_loss: 0.5418 - val_accuracy: 0.9013
Epoch 73/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1021 - a
ccuracy: 0.9566 - val_loss: 0.3876 - val_accuracy: 0.9138
Epoch 74/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1269 - a
ccuracy: 0.9548 - val_loss: 0.5927 - val_accuracy: 0.8880
Epoch 75/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1984 - a
ccuracy: 0.9342 - val_loss: 0.4757 - val_accuracy: 0.8619
Epoch 76/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1665 - a
ccuracy: 0.9422 - val_loss: 0.3287 - val_accuracy: 0.8972
Epoch 77/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1106 - a
ccuracy: 0.9558 - val_loss: 0.3224 - val_accuracy: 0.8999
Epoch 78/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1044 - a
```

```
ccuracy: 0.9593 - val_loss: 0.2954 - val_accuracy: 0.9158
Epoch 79/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.0958 - a
ccuracy: 0.9603 - val_loss: 0.3340 - val_accuracy: 0.9080
Epoch 80/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.0961 - a
ccuracy: 0.9611 - val_loss: 0.3241 - val_accuracy: 0.9135
Epoch 81/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1118 - a
ccuracy: 0.9539 - val_loss: 0.3146 - val_accuracy: 0.9036
Epoch 82/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1014 - a
ccuracy: 0.9569 - val_loss: 0.3208 - val_accuracy: 0.9111
Epoch 83/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.0970 - a
ccuracy: 0.9589 - val_loss: 0.3255 - val_accuracy: 0.9152
Epoch 84/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1031 - a
ccuracy: 0.9559 - val_loss: 0.3702 - val_accuracy: 0.9070
Epoch 85/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.0985 - a
ccuracy: 0.9572 - val_loss: 0.3227 - val_accuracy: 0.9169
Epoch 86/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.0995 - a
ccuracy: 0.9569 - val_loss: 0.3351 - val_accuracy: 0.9121
Epoch 87/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.0939 - a
ccuracy: 0.9606 - val_loss: 0.3435 - val_accuracy: 0.9111
Epoch 88/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.0947 - a
ccuracy: 0.9599 - val_loss: 0.3477 - val_accuracy: 0.9138
Epoch 89/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1158 - a
ccuracy: 0.9518 - val_loss: 0.3108 - val_accuracy: 0.9186
Epoch 90/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.0997 - a
ccuracy: 0.9570 - val_loss: 0.3305 - val_accuracy: 0.9172
Epoch 91/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1019 - a
ccuracy: 0.9561 - val_loss: 0.2361 - val_accuracy: 0.9155
Epoch 92/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.0964 - a
ccuracy: 0.9566 - val_loss: 0.2913 - val_accuracy: 0.9257
Epoch 93/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.2380 - a
ccuracy: 0.9120 - val_loss: 0.4270 - val_accuracy: 0.8697
Epoch 94/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1628 - a
ccuracy: 0.9314 - val_loss: 0.3257 - val_accuracy: 0.8867
Epoch 95/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1449 - a
ccuracy: 0.9430 - val_loss: 0.2255 - val_accuracy: 0.9094
Epoch 96/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1106 - a
ccuracy: 0.9548 - val_loss: 0.2556 - val_accuracy: 0.9118
Epoch 97/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1119 - a
ccuracy: 0.9562 - val_loss: 0.2538 - val_accuracy: 0.9145
Epoch 98/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1108 - a
ccuracy: 0.9518 - val_loss: 0.2391 - val_accuracy: 0.9104
Epoch 99/100
```

```
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1256 - a
ccuracy: 0.9489 - val_loss: 0.2442 - val_accuracy: 0.9253
Epoch 100/100
7352/7352 [==============================] - 21s 3ms/step - loss: 0.1193 - a
ccuracy: 0.9528 - val_loss: 0.2453 - val_accuracy: 0.9240
```

Out[80]:
```
<keras.callbacks.callbacks.History at 0x23dce5d21c8>
```

In [81]:
```python
# Confusion Matrix
print(confusion_matrix(Y_test, model.predict(X_test)))
```

```
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS
\
True
LAYING                 537        0         0        0                   0
SITTING                  0      394        95        0                   0
STANDING                 0       83       449        0                   0
WALKING                  0        0         0      470                  14
WALKING_DOWNSTAIRS       0        0         0        1                 419
WALKING_UPSTAIRS         0        0         0        5                  12

Pred                WALKING_UPSTAIRS
True
LAYING                             0
SITTING                            2
STANDING                           0
WALKING                           12
WALKING_DOWNSTAIRS                 0
WALKING_UPSTAIRS                 454
```

In [82]:
```python
score = model.evaluate(X_test, Y_test)
score
```

```
2947/2947 [==============================] - 9s 3ms/step
```

Out[82]:
```
[0.24532297056183483, 0.9239904880523682]
```

2. 1-D CNN followed by 2 layer Bidirectional LSTM

In [9]:
```python
n_classes=6

# Initiliazing the sequential model
model = Sequential()

# Configuring the parameters
#CNN
model.add(keras.layers.Conv1D(filters=32,kernel_size=(1),strides=1,padding='va
lid',activation='relu'))
model.add(Dropout(0.5))

# 1st Layer of LSTM
model.add(keras.layers.Bidirectional(LSTM(32,return_sequences=True,kernel_init
ializer='glorot_normal')))
# Adding a dropout layer
model.add(Dropout(0.5))

#LSTM
model.add(keras.layers.Bidirectional(LSTM(10 ,kernel_initializer='glorot_norma
```

```
l')))
model.add(Dropout(0.5))

#DENSE
model.add(Dense(28,activation='relu'))

# Adding a dense output layer with sigmoid activation
model.add(Dense(n_classes, activation='sigmoid'))

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

model.summary()
```

In [142]:

```
# Training the model
model.fit(X_train,
          Y_train,
          batch_size=100,
          validation_data=(X_test, Y_test),
          epochs=100)
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1460 - a
ccuracy: 0.9430 - val_loss: 0.3804 - val_accuracy: 0.8955
Epoch 2/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1587 - a
ccuracy: 0.9391 - val_loss: 0.4085 - val_accuracy: 0.8921
Epoch 3/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1525 - a
ccuracy: 0.9431 - val_loss: 0.3362 - val_accuracy: 0.8982
Epoch 4/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1392 - a
ccuracy: 0.9471 - val_loss: 0.3751 - val_accuracy: 0.8985
Epoch 5/100
7352/7352 [==============================] - 26s 3ms/step - loss: 0.1381 - a
ccuracy: 0.9456 - val_loss: 0.3014 - val_accuracy: 0.9030
Epoch 6/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1395 - a
ccuracy: 0.9434 - val_loss: 0.3216 - val_accuracy: 0.9141
Epoch 7/100
7352/7352 [==============================] - 26s 3ms/step - loss: 0.1481 - a
ccuracy: 0.9441 - val_loss: 0.3868 - val_accuracy: 0.8968
Epoch 8/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1582 - a
ccuracy: 0.9456 - val_loss: 0.3610 - val_accuracy: 0.9097
Epoch 9/100
7352/7352 [==============================] - 25s 3ms/step - loss: 0.1478 - a
ccuracy: 0.9457 - val_loss: 0.3466 - val_accuracy: 0.9077
Epoch 10/100
7352/7352 [==============================] - 26s 3ms/step - loss: 0.1367 - a
ccuracy: 0.9499 - val_loss: 0.3589 - val_accuracy: 0.9128
Epoch 11/100
7352/7352 [==============================] - 26s 3ms/step - loss: 0.1342 - a
ccuracy: 0.9491 - val_loss: 0.3594 - val_accuracy: 0.9104
Epoch 12/100
7352/7352 [==============================] - 26s 3ms/step - loss: 0.1291 - a
ccuracy: 0.9472 - val_loss: 0.3393 - val_accuracy: 0.9179
Epoch 13/100
```

```
7352/7352 [==============================] - 26s 3ms/step - loss: 0.1297 - a
ccuracy: 0.9499 - val_loss: 0.3080 - val_accuracy: 0.9104
Epoch 14/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1293 - a
ccuracy: 0.9499 - val_loss: 0.4544 - val_accuracy: 0.9033
Epoch 15/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1291 - a
ccuracy: 0.9529 - val_loss: 0.4845 - val_accuracy: 0.8996
Epoch 16/100
7352/7352 [==============================] - 26s 3ms/step - loss: 0.1383 - a
ccuracy: 0.9445 - val_loss: 0.3254 - val_accuracy: 0.9053
Epoch 17/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1319 - a
ccuracy: 0.9498 - val_loss: 0.4429 - val_accuracy: 0.8972
Epoch 18/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1257 - a
ccuracy: 0.9487 - val_loss: 0.3320 - val_accuracy: 0.9152
Epoch 19/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1153 - a
ccuracy: 0.9517 - val_loss: 0.3266 - val_accuracy: 0.9169
Epoch 20/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1213 - a
ccuracy: 0.9491 - val_loss: 0.3547 - val_accuracy: 0.9084
Epoch 21/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1306 - a
ccuracy: 0.9490 - val_loss: 0.3285 - val_accuracy: 0.9145
Epoch 22/100
7352/7352 [==============================] - 26s 3ms/step - loss: 0.1257 - a
ccuracy: 0.9513 - val_loss: 0.3349 - val_accuracy: 0.9084
Epoch 23/100
7352/7352 [==============================] - 26s 3ms/step - loss: 0.1516 - a
ccuracy: 0.9445 - val_loss: 0.4564 - val_accuracy: 0.9002
Epoch 24/100
7352/7352 [==============================] - 26s 3ms/step - loss: 0.1307 - a
ccuracy: 0.9467 - val_loss: 0.3754 - val_accuracy: 0.9131
Epoch 25/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1206 - a
ccuracy: 0.9512 - val_loss: 0.4352 - val_accuracy: 0.9063
Epoch 26/100
7352/7352 [==============================] - 26s 3ms/step - loss: 0.1247 - a
ccuracy: 0.9499 - val_loss: 0.3498 - val_accuracy: 0.9057
Epoch 27/100
7352/7352 [==============================] - 26s 3ms/step - loss: 0.1226 - a
ccuracy: 0.9513 - val_loss: 0.4139 - val_accuracy: 0.9009
Epoch 28/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1126 - a
ccuracy: 0.9551 - val_loss: 0.4298 - val_accuracy: 0.9060
Epoch 29/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1179 - a
ccuracy: 0.9528 - val_loss: 0.3607 - val_accuracy: 0.9091
Epoch 30/100
7352/7352 [==============================] - 26s 3ms/step - loss: 0.1177 - a
ccuracy: 0.9504 - val_loss: 0.3873 - val_accuracy: 0.9104
Epoch 31/100
7352/7352 [==============================] - 26s 3ms/step - loss: 0.1250 - a
ccuracy: 0.9504 - val_loss: 0.3745 - val_accuracy: 0.9165
Epoch 32/100
7352/7352 [==============================] - 26s 3ms/step - loss: 0.1180 - a
ccuracy: 0.9517 - val_loss: 0.3699 - val_accuracy: 0.9152
Epoch 33/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1178 - a
ccuracy: 0.9523 - val_loss: 0.2940 - val_accuracy: 0.9114
```

```
Epoch 34/100
7352/7352 [==============================] - 26s 3ms/step - loss: 0.1235 - a
ccuracy: 0.9502 - val_loss: 0.3252 - val_accuracy: 0.9158
Epoch 35/100
7352/7352 [==============================] - 26s 3ms/step - loss: 0.1190 - a
ccuracy: 0.9523 - val_loss: 0.3057 - val_accuracy: 0.9165
Epoch 36/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1141 - a
ccuracy: 0.9538 - val_loss: 0.3922 - val_accuracy: 0.9131
Epoch 37/100
7352/7352 [==============================] - 26s 3ms/step - loss: 0.1254 - a
ccuracy: 0.9521 - val_loss: 0.3283 - val_accuracy: 0.9220
Epoch 38/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.2240 - a
ccuracy: 0.9353 - val_loss: 0.4871 - val_accuracy: 0.9033
Epoch 39/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.2569 - a
ccuracy: 0.9158 - val_loss: 0.4095 - val_accuracy: 0.9094
Epoch 40/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1870 - a
ccuracy: 0.9353 - val_loss: 0.3674 - val_accuracy: 0.9111
Epoch 41/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1340 - a
ccuracy: 0.9484 - val_loss: 0.3900 - val_accuracy: 0.9087
Epoch 42/100
7352/7352 [==============================] - 26s 3ms/step - loss: 0.1293 - a
ccuracy: 0.9489 - val_loss: 0.4196 - val_accuracy: 0.9050
Epoch 43/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1345 - a
ccuracy: 0.9459 - val_loss: 0.3977 - val_accuracy: 0.9108
Epoch 44/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1238 - a
ccuracy: 0.9512 - val_loss: 0.4015 - val_accuracy: 0.9087
Epoch 45/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1207 - a
ccuracy: 0.9497 - val_loss: 0.3875 - val_accuracy: 0.9138
Epoch 46/100
7352/7352 [==============================] - 26s 3ms/step - loss: 0.1217 - a
ccuracy: 0.9494 - val_loss: 0.3926 - val_accuracy: 0.9084
Epoch 47/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1154 - a
ccuracy: 0.9508 - val_loss: 0.3513 - val_accuracy: 0.9148
Epoch 48/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1163 - a
ccuracy: 0.9524 - val_loss: 0.4061 - val_accuracy: 0.9114
Epoch 49/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1235 - a
ccuracy: 0.9509 - val_loss: 0.3548 - val_accuracy: 0.9141
Epoch 50/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1170 - a
ccuracy: 0.9536 - val_loss: 0.3650 - val_accuracy: 0.9189
Epoch 51/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1331 - a
ccuracy: 0.9491 - val_loss: 0.3452 - val_accuracy: 0.9155
Epoch 52/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1184 - a
ccuracy: 0.9509 - val_loss: 0.3585 - val_accuracy: 0.9172
Epoch 53/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1178 - a
ccuracy: 0.9532 - val_loss: 0.3812 - val_accuracy: 0.9196
Epoch 54/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1222 - a
```

```
ccuracy: 0.9513 - val_loss: 0.3649 - val_accuracy: 0.9135
Epoch 55/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1149 - a
ccuracy: 0.9512 - val_loss: 0.3980 - val_accuracy: 0.9138
Epoch 56/100
7352/7352 [==============================] - 26s 3ms/step - loss: 0.1417 - a
ccuracy: 0.9472 - val_loss: 0.3432 - val_accuracy: 0.9216
Epoch 57/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1148 - a
ccuracy: 0.9456 - val_loss: 0.3438 - val_accuracy: 0.9179
Epoch 58/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1137 - a
ccuracy: 0.9506 - val_loss: 0.4036 - val_accuracy: 0.9192
Epoch 59/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1424 - a
ccuracy: 0.9372 - val_loss: 0.4189 - val_accuracy: 0.9080
Epoch 60/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1265 - a
ccuracy: 0.9431 - val_loss: 0.3738 - val_accuracy: 0.9148
Epoch 61/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1366 - a
ccuracy: 0.9324 - val_loss: 0.3003 - val_accuracy: 0.8938
Epoch 62/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1267 - a
ccuracy: 0.9434 - val_loss: 0.4071 - val_accuracy: 0.9135
Epoch 63/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1210 - a
ccuracy: 0.9472 - val_loss: 0.3611 - val_accuracy: 0.9121
Epoch 64/100
7352/7352 [==============================] - 26s 3ms/step - loss: 0.1117 - a
ccuracy: 0.9502 - val_loss: 0.3111 - val_accuracy: 0.9186
Epoch 65/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1132 - a
ccuracy: 0.9521 - val_loss: 0.3706 - val_accuracy: 0.9169
Epoch 66/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1533 - a
ccuracy: 0.9423 - val_loss: 0.4784 - val_accuracy: 0.8996
Epoch 67/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1489 - a
ccuracy: 0.9460 - val_loss: 0.3952 - val_accuracy: 0.9080
Epoch 68/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1182 - a
ccuracy: 0.9525 - val_loss: 0.3491 - val_accuracy: 0.9101
Epoch 69/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1204 - a
ccuracy: 0.9513 - val_loss: 0.3071 - val_accuracy: 0.9189
Epoch 70/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1177 - a
ccuracy: 0.9512 - val_loss: 0.3403 - val_accuracy: 0.9097
Epoch 71/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1128 - a
ccuracy: 0.9535 - val_loss: 0.3318 - val_accuracy: 0.9128
Epoch 72/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1124 - a
ccuracy: 0.9570 - val_loss: 0.3738 - val_accuracy: 0.9158
Epoch 73/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1100 - a
ccuracy: 0.9532 - val_loss: 0.3994 - val_accuracy: 0.9111
Epoch 74/100
7352/7352 [==============================] - 26s 3ms/step - loss: 0.1124 - a
ccuracy: 0.9529 - val_loss: 0.4262 - val_accuracy: 0.9108
Epoch 75/100
```

```
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1068 - a
ccuracy: 0.9551 - val_loss: 0.4459 - val_accuracy: 0.9053
Epoch 76/100
7352/7352 [==============================] - 26s 3ms/step - loss: 0.1075 - a
ccuracy: 0.9531 - val_loss: 0.4066 - val_accuracy: 0.9141
Epoch 77/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1168 - a
ccuracy: 0.9520 - val_loss: 0.3940 - val_accuracy: 0.9128
Epoch 78/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1093 - a
ccuracy: 0.9567 - val_loss: 0.3772 - val_accuracy: 0.9165
Epoch 79/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1112 - a
ccuracy: 0.9539 - val_loss: 0.3633 - val_accuracy: 0.9179
Epoch 80/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1064 - a
ccuracy: 0.9581 - val_loss: 0.3295 - val_accuracy: 0.9158
Epoch 81/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1074 - a
ccuracy: 0.9551 - val_loss: 0.3802 - val_accuracy: 0.9155
Epoch 82/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1050 - a
ccuracy: 0.9580 - val_loss: 0.3834 - val_accuracy: 0.9209
Epoch 83/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1011 - a
ccuracy: 0.9558 - val_loss: 0.3930 - val_accuracy: 0.9162
Epoch 84/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1151 - a
ccuracy: 0.9543 - val_loss: 0.3322 - val_accuracy: 0.9206
Epoch 85/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1063 - a
ccuracy: 0.9547 - val_loss: 0.3533 - val_accuracy: 0.9209
Epoch 86/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1057 - a
ccuracy: 0.9559 - val_loss: 0.3469 - val_accuracy: 0.9145
Epoch 87/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1100 - a
ccuracy: 0.9539 - val_loss: 0.3275 - val_accuracy: 0.9131
Epoch 88/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1137 - a
ccuracy: 0.9524 - val_loss: 0.3467 - val_accuracy: 0.9223
Epoch 89/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1101 - a
ccuracy: 0.9559 - val_loss: 0.5188 - val_accuracy: 0.9125
Epoch 90/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1130 - a
ccuracy: 0.9553 - val_loss: 0.5026 - val_accuracy: 0.9091
Epoch 91/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1141 - a
ccuracy: 0.9532 - val_loss: 0.3782 - val_accuracy: 0.9209
Epoch 92/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1107 - a
ccuracy: 0.9561 - val_loss: 0.4357 - val_accuracy: 0.9128
Epoch 93/100
7352/7352 [==============================] - 26s 3ms/step - loss: 0.1074 - a
ccuracy: 0.9576 - val_loss: 0.5562 - val_accuracy: 0.9070
Epoch 94/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1097 - a
ccuracy: 0.9555 - val_loss: 0.4038 - val_accuracy: 0.9240
Epoch 95/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1141 - a
ccuracy: 0.9529 - val_loss: 0.3881 - val_accuracy: 0.9209
```

```
Epoch 96/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1170 - a
ccuracy: 0.9504 - val_loss: 0.5715 - val_accuracy: 0.8989
Epoch 97/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1187 - a
ccuracy: 0.9532 - val_loss: 0.4648 - val_accuracy: 0.9118
Epoch 98/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1075 - a
ccuracy: 0.9566 - val_loss: 0.4258 - val_accuracy: 0.9196
Epoch 99/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1032 - a
ccuracy: 0.9555 - val_loss: 0.4230 - val_accuracy: 0.9165
Epoch 100/100
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1358 - a
ccuracy: 0.9510 - val_loss: 0.4273 - val_accuracy: 0.9172
```

Out[142]:
```
<keras.callbacks.callbacks.History at 0x15d483e3e08>
```

In [143]:
```python
score = model.evaluate(X_test, Y_test)
score
```

```
2947/2947 [==============================] - 14s 5ms/step
```

Out[143]:
```
[0.42743501546085205, 0.917203962802887]
```

Observation ...

3. Divide and Conquer method

refer: https://www.mdpi.com/1424-8220/18/4/1055

refer: https://github.com/heeryoncho/sensors2018cnnhar

From paper:

```
    "Our approach is similar to [36] in that we perform a two-stage clas
sification where we classify
    abstract activities (e.g., dynamic and static) first and then classi
fy individual activities (e.g., walking,
    standing, etc.) next. However, we build one binary 1D CNN model at t
he first stage and two multi-class
    1D CNN models at the second stage. More importantly, we introduce te
st data sharpening in between
    the two-stage HAR, selectively at the prediction phase only, and thi
s differentiates our approach from
    the rest of the two-stage HAR approaches."
```

In [34]:
```python
# refer: https://www.mdpi.com/1424-8220/18/4/1055
# refer: https://github.com/heeryoncho/sensors2018cnnhar

from IPython.display import Image
Image(filename='divide and conquer.PNG')
```

Out[34]:

Out[34]:

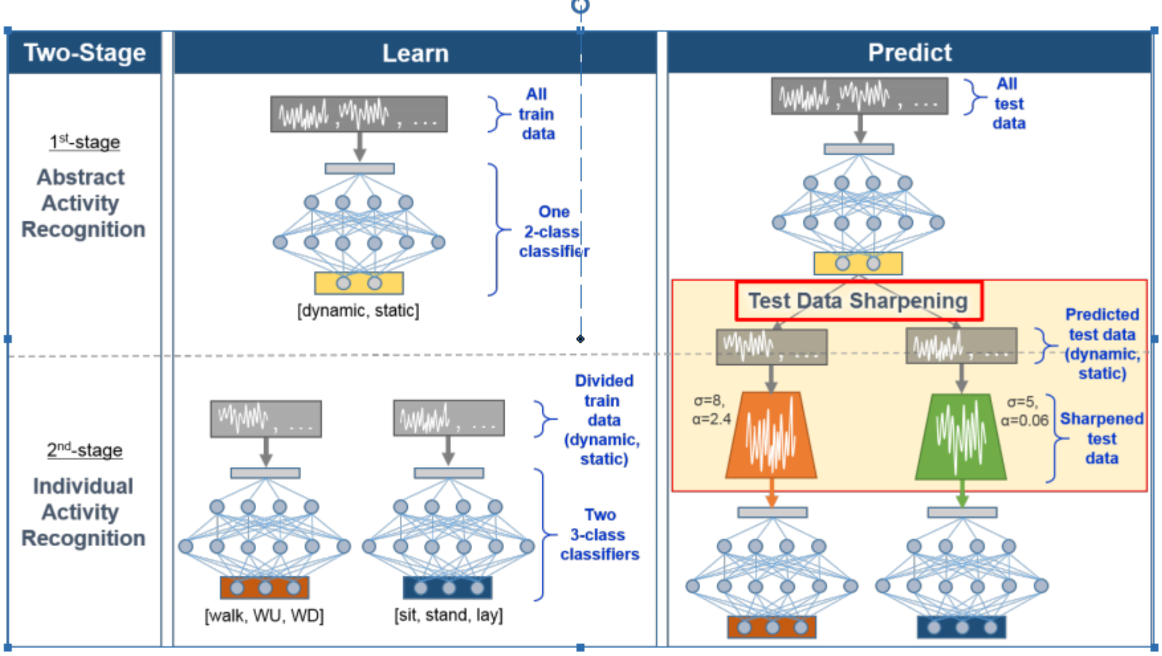


Figure 2. Overview of our divide and conquer-based 1D CNN HAR with test data sharpening. Our approach employs two-stage classifier learning during the learning phase and introduces test data sharpening during the prediction phase.

In [35]:

```
#utility function
# decoding y_tain(6 class) into {0,1,2,3,4,5,9}
def amax(num):
    return (np.argmax(num))

y_train_decode=pd.DataFrame(Y_train).apply(amax,axis=1)
y_test_decode=pd.DataFrame(Y_test).apply(amax,axis=1)

print("Before: ")
print(Y_train)
print("\nAfter: ")
print(y_train_decode)
```

```
Before: 
[[0 0 0 0 1 0]
 [0 0 0 0 1 0]
 [0 0 0 0 1 0]
 ...
 [0 1 0 0 0 0]
 [0 1 0 0 0 0]
 [0 1 0 0 0 0]]

After: 
0       4
1       4
2       4
3       4
4       4
       ..
7347    1
7348    1
7349    1
7350    1
7351    1
Length: 7352, dtype: int64
```

```
C:\Users\family\Anaconda3\lib\site-packages\numpy\core\fromnumeric.py:56: Fu
tureWarning:
```

```
The current behaviour of 'Series.argmax' is deprecated, use 'idxmax'
instead.
The behavior of 'argmax' will be corrected to return the positional
maximum in the future. For now, use 'series.values.argmax' or
'np.argmax(np.array(values))' to get the position of the maximum
row.
  return getattr(obj, method)(*args, **kwds)
```

In [36]:
```
# utility function
def binary_decode(num):
    if np.argmax(num)<3:
        return 0
    else:
        return 1
```

3.1 Model for static and dynamic binary classification

In [37]:
```
# model for static and dynamic binary classification
def train_2_class_classifiation():

    # mode configuration
    model = Sequential()
    model.add(Conv1D(128, 3, input_shape=(128, 9), activation='relu'))
    model.add(Conv1D(64, 3, activation='relu'))
    model.add(Flatten())
    model.add(Dense(2, activation='softmax'))
    model.add(Dropout(0.50))

    model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accur
acy'])

    # Summarize layers
    print(model.summary())
    return model

train_2_class_classifiation()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_9 (Conv1D)            (None, 126, 128)          3584
_________________________________________________________________
conv1d_10 (Conv1D)           (None, 124, 64)           24640
_________________________________________________________________
flatten_5 (Flatten)          (None, 7936)              0
_________________________________________________________________
dense_5 (Dense)              (None, 2)                 15874
_________________________________________________________________
dropout_5 (Dropout)          (None, 2)                 0
=================================================================
Total params: 44,098
Trainable params: 44,098
Non-trainable params: 0
_________________________________________________________________
None
```

Out[37]:
```
<keras.engine.sequential.Sequential at 0x2da5b4c8188>
```

In [38]:
```
# preparing data for static and dynamic binary classification

# this will convert y_train and y_test into {0,1}
y_train_binary= pd.DataFrame(Y_train).apply(binary_decode,axis=1)
y_test_binary= pd.DataFrame(Y_test).apply(binary_decode,axis=1)

# one hot encoding of decoded y_train and y_test
y_train_static_dynamic_oh = keras.utils.to_categorical(y_train_binary)
y_test_static_dynamic_oh = keras.utils.to_categorical(y_test_binary) # one hot
encoding of decoded y_test


# Traning tatic and dynamic  binary classification
binary_model = train_2_class_classifiation()
binary_model.fit(X_train, y_train_static_dynamic_oh,
             batch_size=32, epochs=15, verbose=2, validation_data=(X_test,y_te
st_static_dynamic_oh) )
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv1d_11 (Conv1D)           (None, 126, 128)          3584      
_________________________________________________________________
conv1d_12 (Conv1D)           (None, 124, 64)           24640     
_________________________________________________________________
flatten_6 (Flatten)          (None, 7936)              0         
_________________________________________________________________
dense_6 (Dense)              (None, 2)                 15874     
_________________________________________________________________
dropout_6 (Dropout)          (None, 2)                 0         
=================================================================
Total params: 44,098
Trainable params: 44,098
Non-trainable params: 0
_________________________________________________________________
None
Train on 7352 samples, validate on 2947 samples
Epoch 1/15
 - 2s - loss: 0.3890 - accuracy: 0.6094 - val_loss: 0.0746 - val_accuracy:
0.9969
Epoch 2/15
 - 1s - loss: 0.3816 - accuracy: 0.6158 - val_loss: 0.0745 - val_accuracy:
0.9973
Epoch 3/15
 - 1s - loss: 0.3803 - accuracy: 0.6100 - val_loss: 0.0634 - val_accuracy:
0.9969
Epoch 4/15
 - 1s - loss: 0.3737 - accuracy: 0.6243 - val_loss: 0.0735 - val_accuracy:
0.9959
Epoch 5/15
 - 1s - loss: 0.3781 - accuracy: 0.6117 - val_loss: 0.0671 - val_accuracy:
0.9990
Epoch 6/15
 - 1s - loss: 0.3767 - accuracy: 0.6124 - val_loss: 0.0768 - val_accuracy:
0.9959
Epoch 7/15

 - 1s - loss: 0.3768 - accuracy: 0.6163 - val_loss: 0.0770 - val_accuracy:
0.9946
Epoch 8/15
 - 1s - loss: 0.3727 - accuracy: 0.6306 - val_loss: 0.0796 - val_accuracy:
0.9956
```

```
0.9956
Epoch 9/15
 - 1s - loss: 0.3768 - accuracy: 0.6158 - val_loss: 0.0526 - val_accuracy:
0.9976
Epoch 10/15
 - 1s - loss: 0.3773 - accuracy: 0.6106 - val_loss: 0.0764 - val_accuracy:
0.9990
Epoch 11/15
 - 1s - loss: 0.3800 - accuracy: 0.6113 - val_loss: 0.0718 - val_accuracy:
0.9936
Epoch 12/15
 - 1s - loss: 0.3776 - accuracy: 0.6089 - val_loss: 0.0737 - val_accuracy:
0.9976
Epoch 13/15
 - 1s - loss: 0.3776 - accuracy: 0.6128 - val_loss: 0.0615 - val_accuracy:
0.9986
Epoch 14/15
 - 1s - loss: 0.3806 - accuracy: 0.5996 - val_loss: 0.0545 - val_accuracy:
0.9980
Epoch 15/15
 - 1s - loss: 0.3760 - accuracy: 0.6133 - val_loss: 0.0644 - val_accuracy:
0.9990
```

Out[38]: <keras.callbacks.callbacks.History at 0x2da5ad6ce88>

In [39]:
```
binary_model.evaluate(X_test,y_test_static_dynamic_oh)
```

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

Out[39]: [0.06440044102560151, 0.9989820122718811]

3.2 Model for Dynamic HAR

In [40]:
```
# refer: https://www.mdpi.com/1424-8220/18/4/1055
# refer: https://github.com/heeryoncho/sensors2018cnnhar

from IPython.display import Image
Image(filename='dynamic model.PNG')
```

Out[40]:

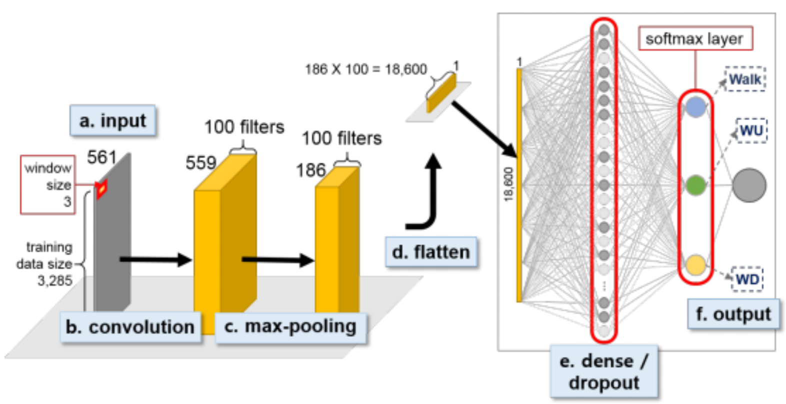


Figure 10. Second-stage 1D CNN for classifying dynamic activity, i.e., Walk, WU, and WD, for UCI HAR dataset.

In [41]:
```
# 1d CNN for dynamic HAR
def train_dyanamic():
    model = Sequential()
    model.add(Conv1D(100, 3, input_shape=(128, 9), activation='relu'))
```

```
    model.add(keras.layers.MaxPooling1D(3))
    model.add(Flatten())
    model.add(Dense(3, activation='softmax'))
    model.add(Dropout(0.5))

    adam = keras.optimizers.Adam(lr=0.0004, beta_1=0.9, beta_2=0.999, epsilon=
1e-08, decay=0.0)
    model.compile(loss='mean_squared_error', optimizer=adam, metrics=['accurac
y'])

    model.summary()

    return model
```

In [42]:
```
# preparing traning data for dynamic 3-class classification

# Index of dynamic activity
dynamic_0=np.where(y_train_decode==0)[0]
dynamic_1=np.where(y_train_decode==1)[0]
dynamic_2=np.where(y_train_decode==2)[0]

dynamic_index=np.concatenate((dynamic_0,dynamic_1,dynamic_2),axis=0)

x_train_dyanamic=X_train[dynamic_index]
y_train_dynamic=y_train_decode[dynamic_index]
print("Total number of dynamic_traning data:",dynamic_index.shape[0])


# preparing test data
# Index of dynamic activity
dynamic_0=np.where(y_test_decode==0)[0]
dynamic_1=np.where(y_test_decode==1)[0]
dynamic_2=np.where(y_test_decode==2)[0]

dynamic_index=np.concatenate((dynamic_0,dynamic_1,dynamic_2))

x_test_dyanamic=X_test[dynamic_index]
y_test_dynamic=y_test_decode[dynamic_index]
print("Total number of dynamic_testing data:",dynamic_index.shape[0])


# Convert to one hot encoding vector
y_train_dynamic_oh = keras.utils.to_categorical(y_train_dynamic)
y_test_dynamic_oh = keras.utils.to_categorical(y_test_dynamic)
```

```
Total number of dynamic_traning data: 3285
Total number of dynamic_testing data: 1387
```

In [43]:
```
# Early Stopper callback
early_stopper = EarlyStopping(monitor='val_loss',min_delta=1e-4,patience=10,ve
rbose=1,
                              mode='auto',baseline=None,restore_best_weights=T
rue)

dynamic_model=train_dyanamic()
dynamic_model.fit(x_train_dyanamic, y_train_dynamic_oh,
             batch_size=32, epochs=50, verbose=2, validation_data=(x_test_dyan
amic,y_test_dynamic_oh),callbacks=[early_stopper] )
```

```
Model: "sequential_7"
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_13 (Conv1D)           (None, 126, 100)          2800
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 42, 100)           0
_________________________________________________________________
flatten_7 (Flatten)          (None, 4200)              0
_________________________________________________________________
dense_7 (Dense)              (None, 3)                 12603
_________________________________________________________________
dropout_7 (Dropout)          (None, 3)                 0
=================================================================
Total params: 15,403
Trainable params: 15,403
Non-trainable params: 0
_________________________________________________________________
Train on 3285 samples, validate on 1387 samples
Epoch 1/50
 - 1s - loss: 0.3150 - accuracy: 0.4180 - val_loss: 0.1867 - val_accuracy:
0.6496
Epoch 2/50
 - 1s - loss: 0.2827 - accuracy: 0.4919 - val_loss: 0.1606 - val_accuracy:
0.6792
Epoch 3/50
 - 1s - loss: 0.2632 - accuracy: 0.5163 - val_loss: 0.1412 - val_accuracy:
0.7952
Epoch 4/50
 - 1s - loss: 0.2478 - accuracy: 0.5379 - val_loss: 0.1274 - val_accuracy:
0.8472
Epoch 5/50
 - 1s - loss: 0.2516 - accuracy: 0.5193 - val_loss: 0.1209 - val_accuracy:
0.8385
Epoch 6/50
 - 1s - loss: 0.2425 - accuracy: 0.5382 - val_loss: 0.1129 - val_accuracy:
0.8832
Epoch 7/50
 - 1s - loss: 0.2423 - accuracy: 0.5279 - val_loss: 0.1118 - val_accuracy:
0.8717
Epoch 8/50
 - 1s - loss: 0.2388 - accuracy: 0.5467 - val_loss: 0.1071 - val_accuracy:
0.9084
Epoch 9/50
 - 1s - loss: 0.2368 - accuracy: 0.5428 - val_loss: 0.1039 - val_accuracy:
0.9142
Epoch 10/50
 - 1s - loss: 0.2350 - accuracy: 0.5482 - val_loss: 0.1064 - val_accuracy:
0.9048
Epoch 11/50
 - 1s - loss: 0.2310 - accuracy: 0.5629 - val_loss: 0.1043 - val_accuracy:
0.9293
Epoch 12/50
 - 1s - loss: 0.2347 - accuracy: 0.5440 - val_loss: 0.1026 - val_accuracy:
0.9279
Epoch 13/50
 - 1s - loss: 0.2361 - accuracy: 0.5425 - val_loss: 0.1002 - val_accuracy:
0.9200
Epoch 14/50
 - 1s - loss: 0.2381 - accuracy: 0.5333 - val_loss: 0.1000 - val_accuracy:
0.9380

Epoch 15/50
 - 1s - loss: 0.2341 - accuracy: 0.5479 - val_loss: 0.1005 - val_accuracy:
```

```
0.9358
Epoch 16/50
 - 1s - loss: 0.2314 - accuracy: 0.5473 - val_loss: 0.0986 - val_accuracy:
0.9329
Epoch 17/50
 - 1s - loss: 0.2299 - accuracy: 0.5549 - val_loss: 0.0993 - val_accuracy:
0.9337
Epoch 18/50
 - 1s - loss: 0.2258 - accuracy: 0.5659 - val_loss: 0.1000 - val_accuracy:
0.9409
Epoch 19/50
 - 1s - loss: 0.2319 - accuracy: 0.5479 - val_loss: 0.0974 - val_accuracy:
0.9344
Epoch 20/50
 - 1s - loss: 0.2340 - accuracy: 0.5333 - val_loss: 0.0962 - val_accuracy:
0.9452
Epoch 21/50
 - 1s - loss: 0.2330 - accuracy: 0.5425 - val_loss: 0.0946 - val_accuracy:
0.9466
Epoch 22/50
 - 1s - loss: 0.2295 - accuracy: 0.5565 - val_loss: 0.0945 - val_accuracy:
0.9438
Epoch 23/50
 - 1s - loss: 0.2348 - accuracy: 0.5279 - val_loss: 0.0970 - val_accuracy:
0.9438
Epoch 24/50
 - 1s - loss: 0.2337 - accuracy: 0.5382 - val_loss: 0.0942 - val_accuracy:
0.9488
Epoch 25/50
 - 1s - loss: 0.2315 - accuracy: 0.5452 - val_loss: 0.0971 - val_accuracy:
0.9301
Epoch 26/50
 - 1s - loss: 0.2319 - accuracy: 0.5425 - val_loss: 0.0936 - val_accuracy:
0.9531
Epoch 27/50
 - 1s - loss: 0.2326 - accuracy: 0.5385 - val_loss: 0.0950 - val_accuracy:
0.9394
Epoch 28/50
 - 1s - loss: 0.2253 - accuracy: 0.5565 - val_loss: 0.0956 - val_accuracy:
0.9452
Epoch 29/50
 - 1s - loss: 0.2308 - accuracy: 0.5406 - val_loss: 0.0943 - val_accuracy:
0.9466
Epoch 30/50
 - 1s - loss: 0.2286 - accuracy: 0.5434 - val_loss: 0.0944 - val_accuracy:
0.9546
Epoch 31/50
 - 1s - loss: 0.2265 - accuracy: 0.5671 - val_loss: 0.0939 - val_accuracy:
0.9488
Epoch 32/50
 - 1s - loss: 0.2339 - accuracy: 0.5336 - val_loss: 0.0918 - val_accuracy:
0.9546
Epoch 33/50
 - 1s - loss: 0.2292 - accuracy: 0.5498 - val_loss: 0.0930 - val_accuracy:
0.9539
Epoch 34/50
 - 1s - loss: 0.2293 - accuracy: 0.5446 - val_loss: 0.0919 - val_accuracy:
0.9539
Epoch 35/50
 - 1s - loss: 0.2251 - accuracy: 0.5580 - val_loss: 0.0946 - val_accuracy:
0.9560
Epoch 36/50
```

```
 - 1s - loss: 0.2304 - accuracy: 0.5425 - val_loss: 0.0926 - val_accuracy:
0.9539
Epoch 37/50
 - 1s - loss: 0.2274 - accuracy: 0.5495 - val_loss: 0.0898 - val_accuracy:
0.9640
Epoch 38/50
 - 1s - loss: 0.2245 - accuracy: 0.5586 - val_loss: 0.0929 - val_accuracy:
0.9603
Epoch 39/50
 - 1s - loss: 0.2207 - accuracy: 0.5714 - val_loss: 0.0942 - val_accuracy:
0.9495
Epoch 40/50
 - 1s - loss: 0.2256 - accuracy: 0.5574 - val_loss: 0.0939 - val_accuracy:
0.9531
Epoch 41/50
 - 1s - loss: 0.2274 - accuracy: 0.5479 - val_loss: 0.0922 - val_accuracy:
0.9560
Epoch 42/50
 - 0s - loss: 0.2285 - accuracy: 0.5482 - val_loss: 0.0931 - val_accuracy:
0.9488
Epoch 43/50
 - 0s - loss: 0.2303 - accuracy: 0.5346 - val_loss: 0.0922 - val_accuracy:
0.9582
Epoch 44/50
 - 0s - loss: 0.2287 - accuracy: 0.5528 - val_loss: 0.0933 - val_accuracy:
0.9560
Epoch 45/50
 - 0s - loss: 0.2306 - accuracy: 0.5412 - val_loss: 0.0935 - val_accuracy:
0.9510
Epoch 46/50
 - 0s - loss: 0.2289 - accuracy: 0.5446 - val_loss: 0.0945 - val_accuracy:
0.9539
Epoch 47/50
 - 0s - loss: 0.2311 - accuracy: 0.5373 - val_loss: 0.0933 - val_accuracy:
0.9517
Restoring model weights from the end of the best epoch
Epoch 00047: early stopping
```

Out[43]: <keras.callbacks.callbacks.History at 0x2da5ac96e48>

In [44]:
```
dynamic_model.evaluate(x_test_dyanamic,y_test_dynamic_oh)
```

```
1387/1387 [==============================] - 0s 58us/step
```

Out[44]: [0.08977583711181257, 0.9639509916305542]

3.3 Model for static HAR

In [45]:
```
# refer: https://www.mdpi.com/1424-8220/18/4/1055
# refer: https://github.com/heeryoncho/sensors2018cnnhar

from IPython.display import Image
Image(filename='static model.PNG')
```

Out[45]:

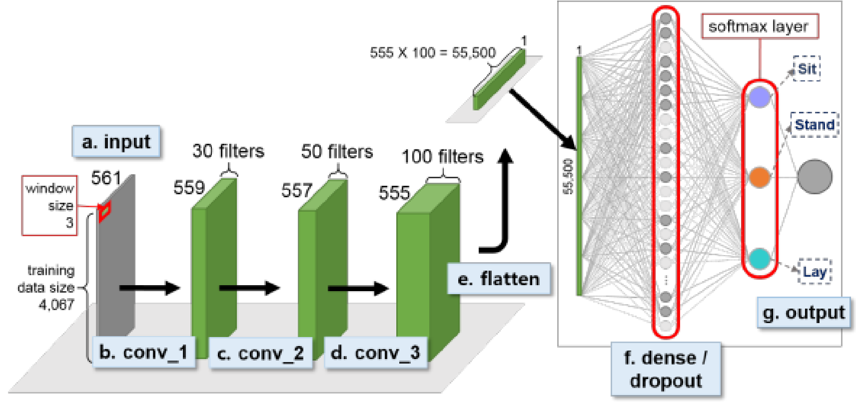


Figure 11. Second-stage 1D CNN for classifying static acitivity, i.e., Sit, Stand, and Lay, for UCI HAR dataset.

In [46]:
```
# model for static HAR
def train_static():

    model = Sequential()
    model.add(Conv1D(30, 3, input_shape=(128, 9), activation='relu'))
    model.add(Conv1D(50, 3, activation='relu'))
    model.add(Conv1D(100, 3, activation='relu'))
    model.add(Flatten())
    model.add(Dense(3, activation='softmax'))
    model.add(Dropout(0.50))

    adam = keras.optimizers.Adam(lr=0.0001, beta_1=0.9, beta_2=0.999, epsilon=
1e-08, decay=0.0)
    model.compile(loss='mean_squared_error', optimizer=adam, metrics=['accurac
y'])

    # Summarize layers
    print(model.summary())
    return model
```

In [47]:
```
# preparing data for STATIC 3-class classification

# Index of static activity train data
static_0=np.where(y_train_decode==3)[0]
static_1=np.where(y_train_decode==4)[0]
static_2=np.where(y_train_decode==5)[0]
static_index=np.concatenate((static_0, static_1, static_2),axis=0)

# X_train_static, y_train_static
x_train_static=X_train[static_index]
y_train_static=y_train_decode[static_index]
print("Total number of static traning data:",static_index.shape[0])


# Index of static activity test data
static_0=np.where(y_test_decode==3)[0]
static_1=np.where(y_test_decode==4)[0]
static_2=np.where(y_test_decode==5)[0]
static_index=np.concatenate((static_0, static_1, static_2) )
print("Total number of static traning data:",static_index.shape[0])

# X_test_static, y_test_static
x_test_static=X_test[static_index]
y_test_static=y_test_decode[static_index]
```

```
# labeling class_labes from {3,4,5} to {0,1,2} for traning
y_train_static=y_train_static.map({3:0, 4:1, 5:2})
y_test_static=y_test_static.map({3:0, 4:1, 5:2})

# Convert to one hot encoding vector
y_train_static_oh = keras.utils.to_categorical(y_train_static)
y_test_static_oh = keras.utils.to_categorical(y_test_static)
```

```
Total number of static_traning data: 4067
Total number of static_traning data: 1560
```

In [48]:

```
# Early Stopper callback
early_stopper = EarlyStopping(monitor='val_loss',min_delta=1e-4,patience=10,ve
rbose=1,
                              mode='auto',baseline=None,restore_best_weights=T
rue)

static_model=train_static()
static_model.fit(x_train_static, y_train_static_oh,
             batch_size=32, epochs=50, verbose=2, validation_data=(x_test_stat
ic,y_test_static_oh) ,callbacks=[early_stopper]  )
```

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv1d_14 (Conv1D)           (None, 126, 30)           840       
_________________________________________________________________
conv1d_15 (Conv1D)           (None, 124, 50)           4550      
_________________________________________________________________
conv1d_16 (Conv1D)           (None, 122, 100)          15100     
_________________________________________________________________
flatten_8 (Flatten)          (None, 12200)             0         
_________________________________________________________________
dense_8 (Dense)              (None, 3)                 36603     
_________________________________________________________________
dropout_8 (Dropout)          (None, 3)                 0         
=================================================================
Total params: 57,093
Trainable params: 57,093
Non-trainable params: 0

_________________________________________________________________
None
Train on 4067 samples, validate on 1560 samples
Epoch 1/50
 - 1s - loss: 0.2748 - accuracy: 0.4773 - val_loss: 0.1040 - val_accuracy:
0.8603
Epoch 2/50
 - 1s - loss: 0.2473 - accuracy: 0.5213 - val_loss: 0.0995 - val_accuracy:
0.8603
Epoch 3/50
 - 1s - loss: 0.2414 - accuracy: 0.5331 - val_loss: 0.1016 - val_accuracy:
0.8788
Epoch 4/50
 - 1s - loss: 0.2473 - accuracy: 0.5055 - val_loss: 0.0971 - val_accuracy:
0.8641
Epoch 5/50
 - 1s - loss: 0.2451 - accuracy: 0.5151 - val_loss: 0.0921 - val_accuracy:
0.8821
Epoch 6/50
 - 1s - loss: 0.2437 - accuracy: 0.5225 - val_loss: 0.0919 - val_accuracy:
```

```
0.8769
Epoch 7/50
 - 1s - loss: 0.2453 - accuracy: 0.5100 - val_loss: 0.0975 - val_accuracy:
0.8769
Epoch 8/50
 - 1s - loss: 0.2451 - accuracy: 0.5085 - val_loss: 0.0920 - val_accuracy:
0.8833
Epoch 9/50
 - 1s - loss: 0.2428 - accuracy: 0.5144 - val_loss: 0.0937 - val_accuracy:
0.8712
Epoch 10/50
 - 1s - loss: 0.2408 - accuracy: 0.5279 - val_loss: 0.0963 - val_accuracy:
0.8769
Epoch 11/50
 - 1s - loss: 0.2401 - accuracy: 0.5195 - val_loss: 0.0993 - val_accuracy:
0.8737
Epoch 12/50
 - 1s - loss: 0.2395 - accuracy: 0.5220 - val_loss: 0.0940 - val_accuracy:
0.8692
Epoch 13/50
 - 1s - loss: 0.2427 - accuracy: 0.5181 - val_loss: 0.0942 - val_accuracy:
0.8724
Epoch 14/50
 - 1s - loss: 0.2414 - accuracy: 0.5225 - val_loss: 0.0958 - val_accuracy:
0.8692
Epoch 15/50
 - 1s - loss: 0.2415 - accuracy: 0.5183 - val_loss: 0.0947 - val_accuracy:
0.8788
Epoch 16/50
 - 1s - loss: 0.2425 - accuracy: 0.5200 - val_loss: 0.0901 - val_accuracy:
0.8853
Epoch 17/50
 - 1s - loss: 0.2428 - accuracy: 0.5164 - val_loss: 0.0910 - val_accuracy:
0.8833
Epoch 18/50
 - 1s - loss: 0.2414 - accuracy: 0.5141 - val_loss: 0.0954 - val_accuracy:
0.8827
Epoch 19/50
 - 1s - loss: 0.2361 - accuracy: 0.5368 - val_loss: 0.0967 - val_accuracy:
0.8731
Epoch 20/50
 - 1s - loss: 0.2443 - accuracy: 0.5095 - val_loss: 0.0956 - val_accuracy:
0.8827
Epoch 21/50
 - 1s - loss: 0.2405 - accuracy: 0.5183 - val_loss: 0.0972 - val_accuracy:
0.8853
Epoch 22/50
 - 1s - loss: 0.2425 - accuracy: 0.5151 - val_loss: 0.0952 - val_accuracy:
0.8897
Epoch 23/50
 - 1s - loss: 0.2396 - accuracy: 0.5272 - val_loss: 0.0970 - val_accuracy:
0.8865
Epoch 24/50
 - 1s - loss: 0.2363 - accuracy: 0.5301 - val_loss: 0.0938 - val_accuracy:
0.8910
Epoch 25/50
 - 1s - loss: 0.2427 - accuracy: 0.5149 - val_loss: 0.0947 - val_accuracy:
0.8897
Epoch 26/50
 - 1s - loss: 0.2396 - accuracy: 0.5213 - val_loss: 0.0908 - val_accuracy:
0.8737
Restoring model weights from the end of the best epoch
```

```
Epoch 00026: early stopping
```

Out[48]:
```
<keras.callbacks.callbacks.History at 0x2da54126ac8>
```

In [49]:
```
static_model.evaluate(x_test_static,y_test_static_oh)
```

```
1560/1560 [==============================] - 0s 60us/step
```

Out[49]:
```
[0.09011286143691112, 0.8852564096450806]
```

Testing

In [50]:
```
# Execute this after executiong binary, static and dynamic classification tran
ing is complete
def testing(x_test, y_test):

    # Binary class prediction
    predict_y_test_binary=binary_model.predict_classes(x_test)


    # Static 1-D CNN 3-class prediction
    static_prediction_index = np.where(predict_y_test_binary==1)
    predict_y_test_static = static_model.predict_classes(x_test[static_predict
ion_index]) # for static
    predict_y_test_static = pd.Series(predict_y_test_static).map({0:3,1:4,2:5
})

     # Dynamic 1-D CNN 3-class prediction
    dynamic_prediction_index = np.where(predict_y_test_binary==0)
    predict_y_test_dynamic=  dynamic_model.predict_classes(x_test[dynamic_pred
iction_index]) # for dynamic

    # Modify the value of prediction od dynamic and static activity
    y_test_pred_generated = np.zeros((x_test.shape[0]))
    y_test_pred_generated[static_prediction_index] = predict_y_test_static
    y_test_pred_generated[dynamic_prediction_index] = predict_y_test_dynamic

    # converting final value to one hot encoding
    y_test_pred_generated_oh = keras.utils.to_categorical(y_test_pred_generate
d)

    # accuracy score
    accuracy = accuracy_score(y_test, y_test_pred_generated_oh)

    return accuracy
```

In [51]:
```
print("Train accuracy: ",testing(X_train,Y_train))
print("Test accuracy: ",testing(X_test,Y_test))
```

```
Train accuracy:  0.9610990206746464
Test accuracy:  0.9219545300305395
```

Test Sharpening

In [52]:
```python
# refer: https://www.mdpi.com/1424-8220/18/4/1055
# refer: https://github.com/heeryoncho/sensors2018cnnhar

from IPython.display import Image
Image(filename='test sharpening.PNG')
```

Out[52]:

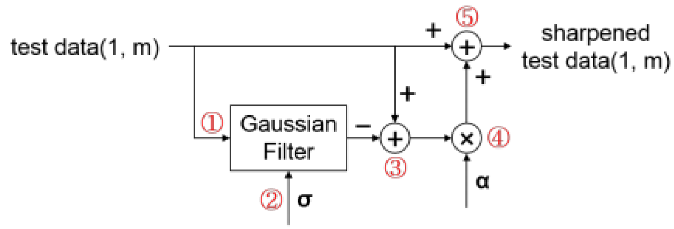


Figure 4. Test data sharpening using a Gaussian filter. Test data is first denoised using a Gaussian filter (①) using the σ parameter (②), and the denoised result is subtracted from the test data to obtain sharped details (③). The sharpened details are then amplified to some degree using α parameter (④) and added to the original test data to obtain sharpened test data (⑤).

$$\text{Denoised}(1,m) = \text{GaussianFilter}(\text{TestData}(1,m),\sigma) \quad (1)$$

$$\text{Detailed}(1,m) = \text{TestData}(1,m) - \text{Denoised}(1,m) \quad (2)$$

$$\text{Sharpened}(1,m) = \text{TestData}(1,m) + \alpha \times \text{Detailed}(1,m) \quad (3)$$

In [53]:
```python
def sharpened(data, sig=1,al=1):

    sharpend_data = []
    for i in (range((data.shape[0]))):

        x_t=data[i]
        denoised_data = scipy.ndimage.gaussian_filter(input=x_t , sigma=sig)
        detailed_data = x_t - denoised_data
        after_sharpend = x_t + (al*denoised_data)

        sharpend_data.append(after_sharpend)

    return np.array(sharpend_data)
```

In [54]:
```python
# refer: https://www.mdpi.com/1424-8220/18/4/1055
# refer: https://github.com/heeryoncho/sensors2018cnnhar

from IPython.display import Image
Image(filename='after sharpening.PNG')
```

Out[54]:

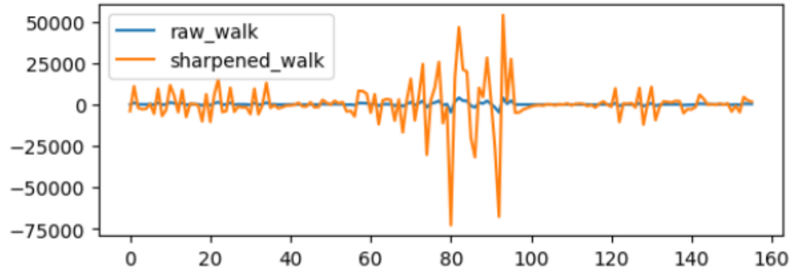


Figure 5. A sample activity data describing walking activity. Each number in the horizontal axis indicates various statistical features such as mean, standard deviation, minimum and maximum calculated from a fixed length time series data collected from multiple sensors. The blue line indicates data before sharpening and the orange line indicates data after sharpening.

In [55]:
```
X_test_sharpened=sharpened(data=X_test, sig=0.1, al=0.1)

testing(X_test_sharpened ,Y_test)
```

Out[55]: 0.9219545300305395

Finding right value of sigma and alpha for Test sharpening

In [56]:
```
# finding right value of sigma and alpha for Test sharpening
sigma_range = np.arange(5,10,1)
alpha_range = np.round(np.arange(0.01,0.31,0.01),3)

result_daframe = pd.DataFrame(np.zeros((sigma_range.shape[0],alpha_range.shape
[0])),index=sigma_range,columns=alpha_range)

for sig in sigma_range:
    for al in alpha_range:
        X_test_sharpened = sharpened(data = X_test, sig=sig  , al=al )
        result = testing(X_test_sharpened,Y_test)

        result_daframe.loc[sig][al]=result
```

In [57]:
```
result_daframe
```

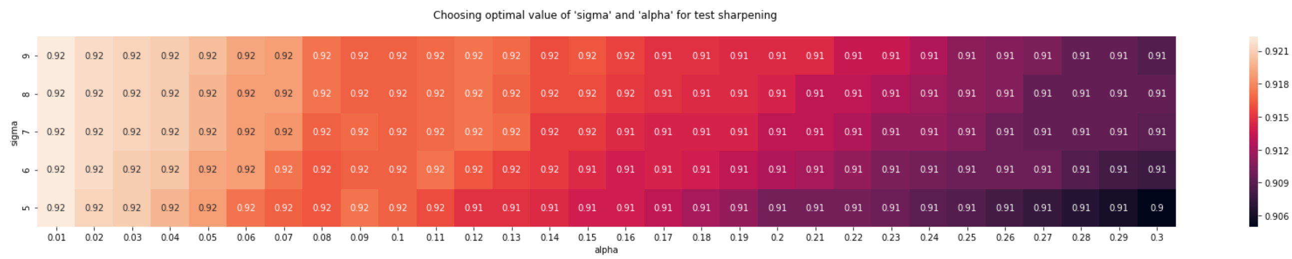
Out[57]:

	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09	0.10
5	0.922294	0.921276	0.920937	0.919919	0.918901	0.917204	0.916525	0.916186	0.917204	0.916525
6	0.922294	0.921615	0.920937	0.920597	0.919240	0.918901	0.917204	0.916186	0.916525	0.916525
7	0.922294	0.921615	0.921276	0.920937	0.919919	0.918901	0.918561	0.916525	0.916865	0.916525
8	0.922294	0.921615	0.921276	0.920937	0.919919	0.918901	0.918901	0.917204	0.916525	0.916525
9	0.922294	0.921615	0.921276	0.920937	0.920258	0.919240	0.918901	0.917204	0.916525	0.916525

5 rows × 30 columns

In [59]:
```
import seaborn as sn
import matplotlib.pyplot as plt
```

```
plt.figure(figsize=(30,4))
sn.heatmap(result_daframe,annot=True,square=True)
plt.title("Choosing optimal value of 'sigma' and 'alpha' for test sharpening
\n")
plt.ylabel("sigma")
plt.xlabel("alpha")
plt.ylim(0,len(result_daframe.index))
plt.show()
```



In [69]:
```
np.where(result_daframe==result_daframe.max(axis=0).max(axis=0))
```

Out[69]: (array([0, 1, 2, 3, 4], dtype=int64), array([0, 0, 0, 0, 0], dtype=int64))

In [72]:
```
# testing using optimal sigma and alpha value
X_test_sharpened=sharpened(data=X_test, sig=8, al=0.01)
testing(X_test_sharpened ,Y_test)
```

Out[72]: 0.9222938581608415

4. LSTM using keras callbacks

In [9]:
```
from keras.callbacks import EarlyStopping, CSVLogger, TensorBoard,ReduceLROnPl
ateau


# reduce_learning_rate callback
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.01,patience=10, min
_lr=0.001)

# CSVLogger callback
csv_log=CSVLogger(filename = "training_log.csv")

# Early Stopper callback
early_stopper = EarlyStopping(monitor='val_loss',min_delta=1e-4,patience=10,ve
rbose=1,
                              mode='auto',baseline=None,restore_best_weights=T
rue)
```

In [10]:
```
n_hidden=150

model = Sequential()
model.add(LSTM(n_hidden, kernel_initializer='glorot_normal',input_shape=(128,
9)))
model.add(Dropout(0.5))
model.add(Dense(6, activation='softmax'))
model.summary()


# Compiling the model
model.compile(loss='categorical_crossentropy',
```

```
                optimizer='adam',
                metrics=['accuracy'])
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_1 (LSTM)                (None, 150)               96000     
_________________________________________________________________
dropout_1 (Dropout)          (None, 150)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 906       
=================================================================
Total params: 96,906
Trainable params: 96,906
Non-trainable params: 0
_________________________________________________________________
```

In [11]:
```
# Training the model
model.fit(X_train,Y_train,
          batch_size=100,validation_data=(X_test, Y_test),epochs=100,callbacks
=[early_stopper,csv_log])
```

```
WARNING:tensorflow:From C:\Users\family\Anaconda3\lib\site-packages\keras\ba
ckend\tensorflow_backend.py:422: The name tf.global_variables is deprecated.
Please use tf.compat.v1.global_variables instead.

Train on 7352 samples, validate on 2947 samples
Epoch 1/100
7352/7352 [==============================] - 15s 2ms/step - loss: 1.3167 - a
ccuracy: 0.4498 - val_loss: 1.0624 - val_accuracy: 0.5684
Epoch 2/100
7352/7352 [==============================] - 14s 2ms/step - loss: 1.0655 - a
ccuracy: 0.5563 - val_loss: 0.9930 - val_accuracy: 0.6322
Epoch 3/100
7352/7352 [==============================] - 14s 2ms/step - loss: 0.7310 - a
ccuracy: 0.7130 - val_loss: 0.8851 - val_accuracy: 0.6837
Epoch 4/100
7352/7352 [==============================] - 15s 2ms/step - loss: 0.8096 - a
ccuracy: 0.6825 - val_loss: 0.7926 - val_accuracy: 0.6787
Epoch 5/100
7352/7352 [==============================] - 14s 2ms/step - loss: 0.7330 - a
ccuracy: 0.6994 - val_loss: 0.6850 - val_accuracy: 0.7418
Epoch 6/100
7352/7352 [==============================] - 15s 2ms/step - loss: 0.5707 - a
ccuracy: 0.7807 - val_loss: 0.5850 - val_accuracy: 0.7665
Epoch 7/100
7352/7352 [==============================] - 14s 2ms/step - loss: 0.6064 - a
ccuracy: 0.7865 - val_loss: 1.1314 - val_accuracy: 0.6071
Epoch 8/100
7352/7352 [==============================] - 14s 2ms/step - loss: 0.9855 - a
ccuracy: 0.6072 - val_loss: 0.7015 - val_accuracy: 0.7472
Epoch 9/100
7352/7352 [==============================] - 14s 2ms/step - loss: 0.6435 - a
ccuracy: 0.7614 - val_loss: 1.7106 - val_accuracy: 0.4635
Epoch 10/100
7352/7352 [==============================] - 14s 2ms/step - loss: 0.7652 - a
ccuracy: 0.7334 - val_loss: 0.5634 - val_accuracy: 0.7913
Epoch 11/100
7352/7352 [==============================] - 15s 2ms/step - loss: 0.4615 - a
ccuracy: 0.8392 - val_loss: 0.5001 - val_accuracy: 0.8263
Epoch 12/100
```

```
7352/7352 [==============================] - 14s 2ms/step - loss: 0.4161 - a
ccuracy: 0.8517 - val_loss: 0.4931 - val_accuracy: 0.8256
Epoch 13/100
7352/7352 [==============================] - 15s 2ms/step - loss: 0.3306 - a
ccuracy: 0.8845 - val_loss: 0.4509 - val_accuracy: 0.8347
Epoch 14/100
7352/7352 [==============================] - 15s 2ms/step - loss: 0.3069 - a
ccuracy: 0.9021 - val_loss: 0.3356 - val_accuracy: 0.8921
Epoch 15/100
7352/7352 [==============================] - 15s 2ms/step - loss: 0.2328 - a
ccuracy: 0.9215 - val_loss: 0.3051 - val_accuracy: 0.8941
Epoch 16/100
7352/7352 [==============================] - 15s 2ms/step - loss: 0.1974 - a
ccuracy: 0.9324 - val_loss: 0.3013 - val_accuracy: 0.8918
Epoch 17/100
7352/7352 [==============================] - 15s 2ms/step - loss: 0.1633 - a
ccuracy: 0.9406 - val_loss: 0.3015 - val_accuracy: 0.8958
Epoch 18/100
7352/7352 [==============================] - 15s 2ms/step - loss: 0.1456 - a
ccuracy: 0.9465 - val_loss: 0.3337 - val_accuracy: 0.8951
Epoch 19/100
7352/7352 [==============================] - 14s 2ms/step - loss: 0.1625 - a
ccuracy: 0.9399 - val_loss: 0.3177 - val_accuracy: 0.8972
Epoch 20/100
7352/7352 [==============================] - 15s 2ms/step - loss: 0.1664 - a
ccuracy: 0.9411 - val_loss: 0.3400 - val_accuracy: 0.8938
Epoch 21/100
7352/7352 [==============================] - 15s 2ms/step - loss: 0.1491 - a
ccuracy: 0.9430 - val_loss: 0.3109 - val_accuracy: 0.9023
Epoch 22/100
7352/7352 [==============================] - 9s 1ms/step - loss: 0.1593 - ac
curacy: 0.9404 - val_loss: 0.3561 - val_accuracy: 0.8972
Epoch 23/100
7352/7352 [==============================] - 8s 1ms/step - loss: 0.1533 - ac
curacy: 0.9483 - val_loss: 0.2875 - val_accuracy: 0.9040
Epoch 24/100
7352/7352 [==============================] - 8s 1ms/step - loss: 0.1579 - ac
curacy: 0.9372 - val_loss: 0.3630 - val_accuracy: 0.8870
Epoch 25/100
7352/7352 [==============================] - 13s 2ms/step - loss: 0.7281 - a
ccuracy: 0.7926 - val_loss: 1.5556 - val_accuracy: 0.4717
Epoch 26/100
7352/7352 [==============================] - 13s 2ms/step - loss: 0.9578 - a
ccuracy: 0.7088 - val_loss: 0.5580 - val_accuracy: 0.8334
Epoch 27/100
7352/7352 [==============================] - 10s 1ms/step - loss: 0.3733 - a
ccuracy: 0.8894 - val_loss: 0.4334 - val_accuracy: 0.8744
Epoch 28/100
7352/7352 [==============================] - 8s 1ms/step - loss: 0.3367 - ac
curacy: 0.8853 - val_loss: 0.4983 - val_accuracy: 0.8252
Epoch 29/100
7352/7352 [==============================] - 8s 1ms/step - loss: 0.3492 - ac
curacy: 0.8551 - val_loss: 0.3980 - val_accuracy: 0.8717
Epoch 30/100
7352/7352 [==============================] - 8s 1ms/step - loss: 1.1440 - ac
curacy: 0.6158 - val_loss: 1.4484 - val_accuracy: 0.4181
Epoch 31/100
7352/7352 [==============================] - 8s 1ms/step - loss: 1.5720 - ac
curacy: 0.4026 - val_loss: 1.3622 - val_accuracy: 0.4544
Epoch 32/100
7352/7352 [==============================] - 8s 1ms/step - loss: 1.1757 - ac
curacy: 0.5257 - val_loss: 1.0895 - val_accuracy: 0.5758
```

```
Epoch 33/100
7352/7352 [==============================] - 8s 1ms/step - loss: 1.5048 - ac
curacy: 0.3747 - val_loss: 1.3143 - val_accuracy: 0.4754
Restoring model weights from the end of the best epoch
Epoch 00033: early stopping
```

Out[11]:
```
<keras.callbacks.callbacks.History at 0x1d54b93fd48>
```

In [12]:
```
model.evaluate(X_test,Y_test)
```

```
2947/2947 [==============================] - 5s 2ms/step
```

Out[12]:
```
[0.2874682506673262, 0.9039701223373413]
```

Query:

I have tried various LSTM Architecture and also the architecure suggested by appliedAI team but still I am not getting desire result .

Shall I submit abover model`s result ?

In [ ]:

In [ ]: