In [0]:
```
# Importing Libraries
```

In [36]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call d
rive.mount("/content/drive", force_remount=True).
```

In [37]:
```
!pip3 install patool
import patoolib
patoolib.extract_archive('/content/drive/My Drive/HumanActivityRecognition.zip')
```

```
Requirement already satisfied: patool in /usr/local/lib/python3.6/dist-packages
(1.12)
patool: Extracting /content/drive/My Drive/HumanActivityRecognition.zip ...
patool: running /usr/bin/7z x -o./Unpack_osvr88i7 -- "/content/drive/My Drive/H
umanActivityRecognition.zip"
patool: ... /content/drive/My Drive/HumanActivityRecognition.zip extracted to `
HumanActivityRecognition1' (multiple files in root).
```

Out[37]:
```
'HumanActivityRecognition1'
```

In [0]:
```
import pandas as pd
import numpy as np
```

In [0]:
```
# 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 [0]:
```
# Data directory
DATADIR = 'UCI_HAR_Dataset'
```

In [0]:
```
# 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 [0]:
```
# 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'/content/HumanActivityRecognition/HAR/UCI_HAR_Dataset/{subs
        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 signals,
    return np.transpose(signals_data, (1, 2, 0))
```

In [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.ht
    """

    filename = f'/content/HumanActivityRecognition/HAR/UCI_HAR_Dataset/{subset}/y
    y = _read_csv(filename)[0]

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

In [0]:
```
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 [0]:
```
# Importing tensorflow
np.random.seed(42)
import tensorflow as tf
tf.set_random_seed(42)
```

In [0]:
```
# Configuring a session
session_conf = tf.ConfigProto(
    intra_op_parallelism_threads=1,
    inter_op_parallelism_threads=1
)
```

In [0]:
```
# Import Keras
from keras import backend as K
sess = tf.Session(graph=tf.get_default_graph(), config=session_conf)
K.set_session(sess)
```

In [0]:
```
# Importing libraries
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers.core import Dense, Dropout
```

In [0]:
```
# Utility function to count the number of classes
def _count_classes(y):
    return len(set([tuple(category) for category in y]))
```

In [50]:
```
# Loading the train and test data
X_train, X_test, Y_train, Y_test = load_data()
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:11: FutureWarning:
Method .as_matrix will be removed in a future version. Use .values instead.
  # This is added back by InteractiveShellApp.init_path()
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:13: FutureWarning:
Method .as_matrix will be removed in a future version. Use .values instead.
  del sys.path[0]
```

In [51]:
```
timesteps = len(X_train[0])
input_dim = len(X_train[0][0])
n_classes = _count_classes(Y_train)

print(timesteps)
print(input_dim)
print(len(X_train))
```

```
128
9
7352
```

- Defining the Architecture of CNN

In [52]:
```
# Initiliazing the sequential model
model = Sequential()
from keras.callbacks import EarlyStopping
from keras.callbacks import ModelCheckpoint
from keras.models import load_model
from keras.layers.normalization import BatchNormalization
from keras.layers import Dense, Activation, Flatten

from keras.layers.convolutional import Conv1D ,MaxPooling1D

model_BN = Sequential()
model_BN.add(Conv1D(32,3,activation='relu',padding='valid',input_shape=(timestep
model_BN.add(BatchNormalization())
model_BN.add(MaxPooling1D(pool_size=2))
model_BN.add(Conv1D(48,3, padding='valid', activation='relu'))
model_BN.add(BatchNormalization())
model_BN.add(MaxPooling1D(pool_size=2))
model_BN.add(Conv1D(64,3, padding='valid', activation='relu'))
model_BN.add(BatchNormalization())
model_BN.add(MaxPooling1D(pool_size=2))
model_BN.add(Conv1D(128,5,padding='valid',activation='relu'))
model_BN.add(MaxPooling1D(pool_size=4))
model_BN.add(Flatten())
model_BN.add(Dense(16, activation='relu'))
model_BN.add(Dense(n_classes, activation='softmax'))
model_BN.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_5 (Conv1D)            (None, 126, 32)           896
_________________________________________________________________
batch_normalization_4 (Batch (None, 126, 32)           128
_________________________________________________________________
max_pooling1d_5 (MaxPooling1 (None, 63, 32)            0
_________________________________________________________________
conv1d_6 (Conv1D)            (None, 61, 48)            4656
_________________________________________________________________
batch_normalization_5 (Batch (None, 61, 48)            192
_________________________________________________________________
max_pooling1d_6 (MaxPooling1 (None, 30, 48)            0
_________________________________________________________________
conv1d_7 (Conv1D)            (None, 28, 64)            9280
_________________________________________________________________
batch_normalization_6 (Batch (None, 28, 64)            256
_________________________________________________________________
max_pooling1d_7 (MaxPooling1 (None, 14, 64)            0
_________________________________________________________________
conv1d_8 (Conv1D)            (None, 10, 128)           41088
_________________________________________________________________
max_pooling1d_8 (MaxPooling1 (None, 2, 128)            0
_________________________________________________________________
flatten_2 (Flatten)          (None, 256)               0
_________________________________________________________________
dense_3 (Dense)              (None, 16)                4112
_________________________________________________________________
```

```
dense_4 (Dense)              (None, 6)                 102
=================================================================
Total params: 60,710
Trainable params: 60,422
Non-trainable params: 288
_________________________________________________________________
```

In [59]:
```
# Compiling the model
# https://machinelearningmastery.com/check-point-deep-learning-models-keras/
from keras.callbacks import *
filepath="/content/HumanActivityRecognition/HAR/epochs:{epoch:03d}-val_acc:{val_a
checkpoint = ModelCheckpoint(filepath, monitor='val_acc', verbose=1, save_best_o
callbacks_list = [checkpoint]



model_BN.compile(loss='categorical_crossentropy',optimizer='Adam', metrics=['acc
model_BN.fit(X_train,Y_train,batch_size=16,validation_data=(X_test, Y_test),epoc
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/15
7352/7352 [==============================] - 7s 889us/step - loss: 0.0994 - ac
c: 0.9596 - val_loss: 0.2548 - val_acc: 0.9267

Epoch 00001: val_acc improved from -inf to 0.92671, saving model to /content/Hu
manActivityRecognition/HAR/epochs:001-val_acc:0.927.hdf5
Epoch 2/15
7352/7352 [==============================] - 5s 684us/step - loss: 0.0814 - ac
c: 0.9657 - val_loss: 0.2947 - val_acc: 0.9335

Epoch 00002: val_acc improved from 0.92671 to 0.93349, saving model to /conten
t/HumanActivityRecognition/HAR/epochs:002-val_acc:0.933.hdf5
Epoch 3/15
7352/7352 [==============================] - 5s 711us/step - loss: 0.0981 - ac
c: 0.9645 - val_loss: 0.3924 - val_acc: 0.9304

Epoch 00003: val_acc did not improve from 0.93349
Epoch 4/15
7352/7352 [==============================] - 5s 694us/step - loss: 0.0962 - ac
c: 0.9627 - val_loss: 0.3372 - val_acc: 0.9355

Epoch 00004: val_acc improved from 0.93349 to 0.93553, saving model to /conten
t/HumanActivityRecognition/HAR/epochs:004-val_acc:0.936.hdf5
Epoch 5/15
7352/7352 [==============================] - 5s 743us/step - loss: 0.0716 - ac
c: 0.9717 - val_loss: 0.3508 - val_acc: 0.9301

Epoch 00005: val_acc did not improve from 0.93553
Epoch 6/15
7352/7352 [==============================] - 5s 697us/step - loss: 0.0890 - ac
c: 0.9640 - val_loss: 0.4023 - val_acc: 0.9186

Epoch 00006: val_acc did not improve from 0.93553
Epoch 7/15
7352/7352 [==============================] - 5s 676us/step - loss: 0.0829 - ac
c: 0.9680 - val_loss: 0.4176 - val_acc: 0.9335

Epoch 00007: val_acc did not improve from 0.93553
Epoch 8/15
7352/7352 [==============================] - 5s 697us/step - loss: 0.0741 - ac
c: 0.9690 - val_loss: 0.4678 - val_acc: 0.9348

Epoch 00008: val_acc did not improve from 0.93553
```

```
Epoch 9/15
7352/7352 [==============================] - 5s 691us/step - loss: 0.0765 - ac
c: 0.9693 - val_loss: 0.4397 - val_acc: 0.9264

Epoch 00009: val_acc did not improve from 0.93553
Epoch 10/15
7352/7352 [==============================] - 5s 671us/step - loss: 0.0655 - ac
c: 0.9736 - val_loss: 0.4522 - val_acc: 0.9382

Epoch 00010: val_acc improved from 0.93553 to 0.93824, saving model to /conten
t/HumanActivityRecognition/HAR/epochs:010-val_acc:0.938.hdf5
Epoch 11/15
7352/7352 [==============================] - 5s 649us/step - loss: 0.0702 - ac
c: 0.9724 - val_loss: 0.3408 - val_acc: 0.9389

Epoch 00011: val_acc improved from 0.93824 to 0.93892, saving model to /conten
t/HumanActivityRecognition/HAR/epochs:011-val_acc:0.939.hdf5
Epoch 12/15
7352/7352 [==============================] - 5s 695us/step - loss: 0.0701 - ac
c: 0.9717 - val_loss: 0.3625 - val_acc: 0.9420

Epoch 00012: val_acc improved from 0.93892 to 0.94197, saving model to /conten
t/HumanActivityRecognition/HAR/epochs:012-val_acc:0.942.hdf5
Epoch 13/15
7352/7352 [==============================] - 5s 654us/step - loss: 0.0656 - ac
c: 0.9721 - val_loss: 0.4574 - val_acc: 0.9291

Epoch 00013: val_acc did not improve from 0.94197
Epoch 14/15
7352/7352 [==============================] - 5s 690us/step - loss: 0.0701 - ac
c: 0.9724 - val_loss: 0.7031 - val_acc: 0.9036

Epoch 00014: val_acc did not improve from 0.94197
Epoch 15/15
7352/7352 [==============================] - 5s 674us/step - loss: 0.0606 - ac
c: 0.9735 - val_loss: 0.5487 - val_acc: 0.9111

Epoch 00015: val_acc did not improve from 0.94197
```

Out[59]:
```
<keras.callbacks.History at 0x7efbbf7ca710>
```

In [0]:
```python
model_BN.load_weights("/content/HumanActivityRecognition/HAR/epochs:012-val_acc:0
```

In [0]:
```python
Y_true = pd.Series([ACTIVITIES[y] for y in np.argmax(Y_test, axis=1)])
Y_pred = pd.Series([ACTIVITIES[y] for y in np.argmax(model_BN.predict(X_test), ax
```

In [62]:
```
pd.crosstab(Y_true, Y_pred, rownames=['True'], colnames=['Pred'])
```

Out[62]:

Pred True	LAYING	SITTING	STANDING	WALKING	WALKING_DOWNSTAIRS	WALKI
LAYING	537	0	0	0	0	
SITTING	22	407	60	0	0	
STANDING	0	19	512	1	0	
WALKING	0	0	1	471	11	
WALKING_DOWNSTAIRS	0	0	0	0	419	
WALKING_UPSTAIRS	0	2	0	12	27	

In [63]:
```
score = model_BN.evaluate(X_test, Y_test)
```

```
2947/2947 [==============================] - 0s 161us/step
```

In [64]:
```
score
```

Out[64]:
```
[0.3624597953069845, 0.9419748897183576]
```

- With a CNN architecture with Batch Normalization and Maxpooling we got 94.19% accuracy and a loss of 0.3624 from our best model