In [66]:
```
# Author : Sahil Chitnis
# Dataset : https://drive.google.com/drive/folders/1s1-174qlu_ekiKcGXeutP5tvr
```

In [ ]:
```
# Install some libraries/packages required for Google Collab

#!pip install noisereduce
#!pip install torchaudio
#!apt install libasound2-dev portaudio19-dev libportaudio2 libportaudiocpp0 f
#!pip install pyaudio
```

In [142…
```
# Step0: Import all libraries required for this project

import librosa
import os
import numpy as np
import noisereduce as nr
import torchaudio
import tensorflow
import numpy
import torch
import random
from sklearn.preprocessing import LabelEncoder
import pandas as pd
import numpy as np
import os
from sklearn import metrics
import keras
from keras.models import Sequential
from keras.layers import Convolution2D, MaxPooling2D,MaxPooling1D,Conv1D
from keras.layers import Dense, Dropout, Activation, Flatten,LSTM,TimeDistrib
from keras.optimizers import Adam,SGD
from keras.utils import np_utils
from keras.callbacks import EarlyStopping, ModelCheckpoint
import random
from keras import optimizers
import datetime
import matplotlib.pyplot as plt
import noisereduce as nr
from keras.models import model_from_json
from sklearn.preprocessing import LabelEncoder
import IPython
import os
import pyaudio
```

In [143…
```
#!unzip ./Dataset_audio.zip
```

In [144…

```
# Step1 : Data Cleaning and Feature extraction step

# Step1.a) Data augmentation - Shift the signal to left/right by some %
#          This step is needed to increase samples
def time_shift(aud, shift_limit):
  sig,sr = aud
  _, sig_len = sig.shape
  shift_amt = int(random.random() * shift_limit * sig_len)
  return (sig.roll(shift_amt), sr)

# Step1.b) Transform data and extract features :
#
# a) Data Augmentation
# b) Noise reduction
# c) Data Trimming
# d) Perform Short-time Fourier transform to extract features
# e) Save features for future use

def compute_and_save_transform(file, name, activity, subject):

    # read the audio data
    audio_data, sample_rate = librosa.load(file)
    #aud = torchaudio.load(file)
    #data = []

    # Performing data augmentation
    #aud1 = time_shift(aud, 1)
    #for elem in aud1[0][0]:
    #  data.append(elem)
    #audio_data = np.array(data)



    # Perform noise reduction
    noise_audio = audio_data[0:35000]
    noise_removed_audio = nr.reduce_noise(audio_clip=audio_data, noise_clip=n



    # Trim the silence in the data by setting silence threshold to 20DB,
    # 512 samples per frame and 128 number of samples between analysis frames
    trimmed_audio, index = librosa.effects.trim(noise_removed_audio, top_db=2



    # Perform Short-time Fourier transform  to extract features
    stft_feat = np.abs(librosa.stft(trimmed_audio, n_fft=512, hop_length=256,


    # Save the features to folder "Transformed_Audio_Features"
    np.save("Transformed_Audio_Features/" + subject + "_" + name[:-4] + "_" +
```

In [ ]:

```
# Step1.c) Perform step1.b for all activities in all samples

activities = ['Writing', 'StandUp', 'Calling', 'Drinking', 'Clapping', 'Eatin
              'Exiting', 'Falling', 'OpeningPillContainer', 'LyingDown',
              'Reading', 'Sitting', 'SitStill', 'Sleeping', 'PickingObject',
              'Sweeping', 'UsingPhone', 'UseLaptop', 'WakeUp', 'Walking',
              'WashingHand', 'WatchingTV', 'WaterPouring', 'Entering']

samples = ['s01', 's02', 's03', 's04', 's05', 's06', 's07', 's08', 's09',
           's10', 's11', 's12', 's13', 's14', 's15', 's16', 's17']

for activity in activities:
    for sample in samples:
        innerDir = sample + "/" + activity
        for file in os.listdir("./Dataset_audio/" + innerDir):
            if(file.endswith(".wav")):
                compute_and_save_transform("Dataset_audio/" + innerDir + "/"
                print("Sample",sample, "Performing", activity, "in", file)
```

In [146…

```
# Step2) Split Activities1,2,3 of each sample as per
#        sample in 3 buckets into Training, Test, Validation based on

#  Taking 12 Training samples
train_samples = ['s07', 's16', 's09', 's13', 's04', 's11', 's15', 's01', 's12

# Taking 2 validation samples
validation_samples = ['s02', 's03']


# Taking 4 test samples
test_samples = ['s05', 's17']


def Split_data(path,sampleSize):


    Activities1 = ['Drinking', 'Eating', 'LyingDown', 'OpeningPillContainer',
                   'PickingObject', 'Reading', 'SitStill', 'Sitting',
                   'StandUp', 'UseLaptop', 'UsingPhone', 'WakeUp', 'Wa
                   'WaterPouring', 'Writing']

    Activities2 = ['Calling', 'Clapping', 'Falling', 'Sweeping', 'WashingHand

    Activities3 = ['Entering', 'Exiting']

    X_train = []
    Y_train = []
    X_test = []
    Y_test = []
    X_validation = []
    Y_validation = []
```

```
    # Split data into Training, Validation, Test
    for file in os.listdir(path):
        if int(file.split("__")[1].split("_")[0])!=1:
          a = (np.load(path + file)).T
          activityLabel = file.split('_')[-1].split(".")[0]
          # Split data from Activities2
          if(activityLabel in Activities2):
                if file.split("_")[0] in train_samples:
                  X_train.append(np.mean(a,axis=0))
                  Y_train.append(activityLabel)
                elif file.split("_")[0] in validation_samples:
                  X_validation.append(np.mean(a,axis=0))
                  Y_validation.append(activityLabel)
                else:
                  X_test.append(np.mean(a,axis=0))
                  Y_test.append(activityLabel)
          # Split data from Activities3
          elif(activityLabel in Activities3):
                activityLabel = "Activities3"
                if file.split("_")[0] in train_samples:
                  X_train.append(np.mean(a,axis=0))
                  Y_train.append(activityLabel)
                elif file.split("_")[0] in validation_samples:
                  X_validation.append(np.mean(a,axis=0))
                  Y_validation.append(activityLabel)
                else:
                  X_test.append(np.mean(a,axis=0))
                  Y_test.append(activityLabel)
          # Split from remaining activities
          else:
                activityLabel = "other"
                if file.split("_")[0] in train_samples:
                  X_train.append(np.mean(a,axis=0))
                  Y_train.append(activityLabel)
                elif file.split("_")[0] in validation_samples:
                  X_validation.append(np.mean(a,axis=0))
                  Y_validation.append(activityLabel)
                else:
                  X_test.append(np.mean(a,axis=0))
                  Y_test.append(activityLabel)

    # Change lists to numpy arrays
    X_train = np.array(X_train)
    Y_train = np.array(Y_train)
    X_test = np.array(X_test)
    Y_test = np.array(Y_test)
    X_validation = np.array(X_validation)
    Y_validation = np.array(Y_validation)

    return X_train,Y_train,X_validation,Y_validation,X_test,Y_test
```

```
# Step3) Functions to print output/data

# Re-sample the data
def reSample(data, samples):
    r = len(data)/samples #re-sampling ratio
    newdata = []
    for i in range(0,samples):
        newdata.append(data[int(i*r)])
    return np.array(newdata)

# Print ConfMatrix
def print_activity_data1(confMatrix):
        s = "ACTIVITY Confusion Matrix:\n"
        for i in range(len(confMatrix)):
            s += lb.inverse_transform([i])[0] + "\t|"
        print(s[:-1])
        for i in range(len(confMatrix)):
            s = ""
            for j in range(len(confMatrix)):
                s += str(confMatrix[i][j])
                s += "\t|"
            print(lb.inverse_transform([i])[0],"\t|", s[:-1])
        print()

def print_activity_data2(confMatrix):
        s = "ACTIVITY  Matrix:\n"
        for i in range(len(confMatrix)):
            s += lb.inverse_transform([i])[0] + "\t|"
        print(s[:-1])
        for i in range(len(confMatrix)):
            s = ""
            for j in range(len(confMatrix)):
                val = confMatrix[i][j]/float(sum(confMatrix[i]))
                s += str(round(val,2))
                s += "\t|"
            print(lb.inverse_transform([i])[0],"\t|", s[:-1])
        print()


# Display the result
def DisplayResult():
  predictions = [np.argmax(y) for y in result]
  expected = [np.argmax(y) for y in y_test]

  confMatrix = []
  num_labels=y_test[0].shape[0]
  for i in range(num_labels):
      r = []
      for j in range(num_labels):
          r.append(0)
      confMatrix.append(r)

  n_tests = len(predictions)
```

```
    for i in range(n_tests):
        confMatrix[expected[i]][predictions[i]] += 1

    # Print activity data
    print_activity_data1(confMatrix)
    print_activity_data2(confMatrix)
```

In [148…

```
featuresPath = "Transformed_Audio_Features/"

# Split data into Train, Validation, Test
X_train,Y_train,X_validation,Y_validation,X_test,Y_test  = Split_data(feature
```

In [149…

```
n_samples = len(Y_train)
print("Number of training samples: " + str(n_samples))
order = np.array(range(n_samples))

# Shuffle Training data
np.random.shuffle(order)
X_train = X_train[order]
Y_train = Y_train[order]


# Step4) Encode the labels for Y_train, Y_test and Y_validation

# Encode the labels
lb = LabelEncoder()

# Fit label encoder on Y_train, Y_test and Y_validation
y_train = np_utils.to_categorical(lb.fit_transform(Y_train))
y_test = np_utils.to_categorical(lb.fit_transform(Y_test))
y_validation = np_utils.to_categorical(lb.fit_transform(Y_validation))
num_labels = y_train.shape[1]
```

```
Number of training samples: 880
```

In [161…

```
# Step5.1) Build Model:
#        a)  1st layer : Dense with o/p = 256 neurons, i/p = 257and RELU acti
#        b)  2nd layer : Dense with o/p = 256 neurons and RELU activation
#        c)  3rd layer : Dense with o/p = 128 neurons and RELU activation
#        d)  4th layer : Dense with o/p = 128 neurons, RELU activation and Dr
#        d)  5th layer : Dense with o/p = 128 neurons, RELU activation and Dr
#        e)  6th layer : Dense with o/p = 8 neurons (ie = num_labels), Softma

num_labels = y_train.shape[1]
filter_size = 20

# Build Sequential model
model = Sequential()

#Layer1
```

```
model.add(Dense(256, input_shape=(257,)))
model.add(Activation('relu'))

#Layer2
model.add(Dense(256))
model.add(Activation('relu'))

#Layer3
model.add(Dense(128))
model.add(Activation('relu'))

#Layer4
model.add(Dense(128))
model.add(Activation('relu'))
model.add(Dropout(0.5))

#Layer5
model.add(Dense(128))
model.add(Activation('relu'))
model.add(Dropout(0.5))

#Layer6
model.add(Dense(num_labels))
model.add(Activation('softmax'))

model.summary()

# Step5.2) Compile Model:
#          a) Loss is categorical_crossentropy
#          b) Optimizer is Adam optimizer

#lr_rates = [2e-3, 3e-1, 3e-2, 3e-3]


# Step5.3) Fit the Model:
#          a) Batch size is 10
#          b) Epoch's is 100
lr_rate = 3e-3
opt = keras.optimizers.Adam(learning_rate=lr_rate)
model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimize
model.fit(X_train, y_train, batch_size=10, epochs=100, validation_data=(X_val
#    result = model.predict(X_test)
```

```
Model: "sequential_32"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_137 (Dense)            (None, 256)               66048
_________________________________________________________________
activation_137 (Activation)  (None, 256)               0
_________________________________________________________________
dense_138 (Dense)            (None, 256)               65792
_________________________________________________________________
activation_138 (Activation)  (None, 256)               0
```

```
_________________________________________________________________
dense_139 (Dense)            (None, 128)               32896
_________________________________________________________________
activation_139 (Activation)  (None, 128)               0
_________________________________________________________________
dense_140 (Dense)            (None, 128)               16512
_________________________________________________________________
activation_140 (Activation)  (None, 128)               0
_________________________________________________________________
dropout_46 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_141 (Dense)            (None, 128)               16512
_________________________________________________________________
activation_141 (Activation)  (None, 128)               0
_________________________________________________________________
dropout_47 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_142 (Dense)            (None, 8)                 1032
_________________________________________________________________
activation_142 (Activation)  (None, 8)                 0
=================================================================
Total params: 198,792
Trainable params: 198,792
Non-trainable params: 0
_________________________________________________________________
Train on 880 samples, validate on 146 samples
Epoch 1/100
880/880 [==============================] - 2s 2ms/step - loss: 1.1108 - accura
cy: 0.7091 - val_loss: 0.7038 - val_accuracy: 0.8014
Epoch 2/100
880/880 [==============================] - 1s 880us/step - loss: 0.8110 - accu
racy: 0.7557 - val_loss: 0.6278 - val_accuracy: 0.7808
Epoch 3/100
880/880 [==============================] - 1s 914us/step - loss: 0.8333 - accu
racy: 0.7636 - val_loss: 0.6095 - val_accuracy: 0.8082
Epoch 4/100
880/880 [==============================] - 1s 867us/step - loss: 0.8441 - accu
racy: 0.7614 - val_loss: 0.5670 - val_accuracy: 0.8082
Epoch 5/100
880/880 [==============================] - 1s 890us/step - loss: 0.8070 - accu
racy: 0.7693 - val_loss: 0.5642 - val_accuracy: 0.8082
Epoch 6/100
880/880 [==============================] - 1s 869us/step - loss: 0.7501 - accu
racy: 0.7739 - val_loss: 0.5632 - val_accuracy: 0.8151
Epoch 7/100
880/880 [==============================] - 1s 865us/step - loss: 0.7306 - accu
racy: 0.7773 - val_loss: 0.6151 - val_accuracy: 0.8151
Epoch 8/100
880/880 [==============================] - 1s 872us/step - loss: 0.7541 - accu
racy: 0.7784 - val_loss: 0.5905 - val_accuracy: 0.8082
Epoch 9/100
880/880 [==============================] - 1s 908us/step - loss: 0.8015 - accu
racy: 0.7636 - val_loss: 0.6070 - val_accuracy: 0.8151
Epoch 10/100
880/880 [==============================] - 1s 868us/step - loss: 0.8004 - accu
racy: 0.7682 - val_loss: 0.5265 - val_accuracy: 0.8219
Epoch 11/100
```

```
880/880 [==============================] - 1s 871us/step - loss: 0.7531 - accu
racy: 0.7841 - val_loss: 0.5150 - val_accuracy: 0.8082
Epoch 12/100
880/880 [==============================] - 1s 875us/step - loss: 0.7327 - accu
racy: 0.7852 - val_loss: 0.5331 - val_accuracy: 0.8151
Epoch 13/100
880/880 [==============================] - 1s 868us/step - loss: 0.7291 - accu
racy: 0.7852 - val_loss: 0.5423 - val_accuracy: 0.8288
Epoch 14/100
880/880 [==============================] - 1s 883us/step - loss: 0.7254 - accu
racy: 0.7852 - val_loss: 0.5186 - val_accuracy: 0.8151
Epoch 15/100
880/880 [==============================] - 1s 863us/step - loss: 0.7066 - accu
racy: 0.7898 - val_loss: 0.5140 - val_accuracy: 0.8151
Epoch 16/100
880/880 [==============================] - 1s 909us/step - loss: 0.6336 - accu
racy: 0.7932 - val_loss: 0.4793 - val_accuracy: 0.8356
Epoch 17/100
880/880 [==============================] - 1s 877us/step - loss: 0.7032 - accu
racy: 0.8023 - val_loss: 0.4179 - val_accuracy: 0.8288
Epoch 18/100
880/880 [==============================] - 1s 872us/step - loss: 0.6177 - accu
racy: 0.8068 - val_loss: 0.3927 - val_accuracy: 0.8562
Epoch 19/100
880/880 [==============================] - 1s 879us/step - loss: 0.6377 - accu
racy: 0.8182 - val_loss: 0.4904 - val_accuracy: 0.8219
Epoch 20/100
880/880 [==============================] - 1s 860us/step - loss: 0.7376 - accu
racy: 0.7977 - val_loss: 0.4909 - val_accuracy: 0.8014
Epoch 21/100
880/880 [==============================] - 1s 876us/step - loss: 0.6811 - accu
racy: 0.8000 - val_loss: 0.3955 - val_accuracy: 0.8904
Epoch 22/100
880/880 [==============================] - 1s 911us/step - loss: 0.5630 - accu
racy: 0.8273 - val_loss: 0.3790 - val_accuracy: 0.8425
Epoch 23/100
880/880 [==============================] - 1s 873us/step - loss: 0.6345 - accu
racy: 0.8068 - val_loss: 0.4693 - val_accuracy: 0.8219
Epoch 24/100
880/880 [==============================] - 1s 871us/step - loss: 0.5351 - accu
racy: 0.8364 - val_loss: 0.3512 - val_accuracy: 0.8767
Epoch 25/100
880/880 [==============================] - 1s 884us/step - loss: 0.5365 - accu
racy: 0.8386 - val_loss: 0.3620 - val_accuracy: 0.8836
Epoch 26/100
880/880 [==============================] - 1s 876us/step - loss: 0.5240 - accu
racy: 0.8523 - val_loss: 0.3337 - val_accuracy: 0.8973
Epoch 27/100
880/880 [==============================] - 1s 879us/step - loss: 0.5374 - accu
racy: 0.8534 - val_loss: 0.4294 - val_accuracy: 0.8630
Epoch 28/100
880/880 [==============================] - 1s 881us/step - loss: 0.5589 - accu
racy: 0.8409 - val_loss: 0.4014 - val_accuracy: 0.8699
Epoch 29/100
880/880 [==============================] - 1s 915us/step - loss: 0.5424 - accu
racy: 0.8443 - val_loss: 0.3879 - val_accuracy: 0.8767
Epoch 30/100
```

```
880/880 [==============================] - 1s 863us/step - loss: 0.5022 - accu
racy: 0.8682 - val_loss: 0.2903 - val_accuracy: 0.9041
Epoch 31/100
880/880 [==============================] - 1s 876us/step - loss: 0.4557 - accu
racy: 0.8705 - val_loss: 0.3571 - val_accuracy: 0.8836
Epoch 32/100
880/880 [==============================] - 1s 870us/step - loss: 0.4529 - accu
racy: 0.8648 - val_loss: 0.3619 - val_accuracy: 0.8767
Epoch 33/100
880/880 [==============================] - 1s 874us/step - loss: 0.4495 - accu
racy: 0.8636 - val_loss: 0.3686 - val_accuracy: 0.8767
Epoch 34/100
880/880 [==============================] - 1s 876us/step - loss: 0.4690 - accu
racy: 0.8784 - val_loss: 0.4533 - val_accuracy: 0.8767
Epoch 35/100
880/880 [==============================] - 1s 920us/step - loss: 0.5397 - accu
racy: 0.8477 - val_loss: 0.4869 - val_accuracy: 0.8493
Epoch 36/100
880/880 [==============================] - 1s 899us/step - loss: 0.5007 - accu
racy: 0.8477 - val_loss: 0.3672 - val_accuracy: 0.8904
Epoch 37/100
880/880 [==============================] - 1s 982us/step - loss: 0.4535 - accu
racy: 0.8648 - val_loss: 0.3399 - val_accuracy: 0.9041
Epoch 38/100
880/880 [==============================] - 1s 956us/step - loss: 0.4496 - accu
racy: 0.8636 - val_loss: 0.3387 - val_accuracy: 0.8973
Epoch 39/100
880/880 [==============================] - 1s 1ms/step - loss: 0.4299 - accura
cy: 0.8739 - val_loss: 0.4116 - val_accuracy: 0.8630
Epoch 40/100
880/880 [==============================] - 1s 982us/step - loss: 0.4207 - accu
racy: 0.8784 - val_loss: 0.2869 - val_accuracy: 0.8836
Epoch 41/100
880/880 [==============================] - 1s 962us/step - loss: 0.3893 - accu
racy: 0.8818 - val_loss: 0.4090 - val_accuracy: 0.8767
Epoch 42/100
880/880 [==============================] - 1s 989us/step - loss: 0.4027 - accu
racy: 0.8852 - val_loss: 0.4162 - val_accuracy: 0.8904
Epoch 43/100
880/880 [==============================] - 1s 937us/step - loss: 0.4272 - accu
racy: 0.8795 - val_loss: 0.4390 - val_accuracy: 0.8630
Epoch 44/100
880/880 [==============================] - 1s 930us/step - loss: 0.4030 - accu
racy: 0.8886 - val_loss: 0.4076 - val_accuracy: 0.8836
Epoch 45/100
880/880 [==============================] - 1s 970us/step - loss: 0.3946 - accu
racy: 0.8909 - val_loss: 0.4038 - val_accuracy: 0.8904
Epoch 46/100
880/880 [==============================] - 1s 914us/step - loss: 0.3782 - accu
racy: 0.8955 - val_loss: 0.4328 - val_accuracy: 0.8767
Epoch 47/100
880/880 [==============================] - 1s 989us/step - loss: 0.3826 - accu
racy: 0.8875 - val_loss: 0.4349 - val_accuracy: 0.8699
Epoch 48/100
880/880 [==============================] - 1s 879us/step - loss: 0.3940 - accu
racy: 0.8886 - val_loss: 0.3450 - val_accuracy: 0.8836
Epoch 49/100
```

```
880/880 [==============================] - 1s 873us/step - loss: 0.3584 - accu
racy: 0.8898 - val_loss: 0.3311 - val_accuracy: 0.8904
Epoch 50/100
880/880 [==============================] - 1s 869us/step - loss: 0.3372 - accu
racy: 0.9102 - val_loss: 0.4090 - val_accuracy: 0.8630
Epoch 51/100
880/880 [==============================] - 1s 927us/step - loss: 0.3356 - accu
racy: 0.8943 - val_loss: 0.3034 - val_accuracy: 0.8973
Epoch 52/100
880/880 [==============================] - 1s 976us/step - loss: 0.3744 - accu
racy: 0.8966 - val_loss: 0.5100 - val_accuracy: 0.8630
Epoch 53/100
880/880 [==============================] - 1s 957us/step - loss: 0.3660 - accu
racy: 0.8830 - val_loss: 0.4451 - val_accuracy: 0.8904
Epoch 54/100
880/880 [==============================] - 1s 889us/step - loss: 0.6084 - accu
racy: 0.8807 - val_loss: 0.4881 - val_accuracy: 0.8562
Epoch 55/100
880/880 [==============================] - 1s 881us/step - loss: 0.4173 - accu
racy: 0.8705 - val_loss: 0.3268 - val_accuracy: 0.9041
Epoch 56/100
880/880 [==============================] - 1s 884us/step - loss: 0.3574 - accu
racy: 0.8989 - val_loss: 0.4202 - val_accuracy: 0.9041
Epoch 57/100
880/880 [==============================] - 1s 897us/step - loss: 0.3474 - accu
racy: 0.9068 - val_loss: 0.5139 - val_accuracy: 0.8699
Epoch 58/100
880/880 [==============================] - 1s 890us/step - loss: 0.3393 - accu
racy: 0.9000 - val_loss: 0.5292 - val_accuracy: 0.8699
Epoch 59/100
880/880 [==============================] - 1s 913us/step - loss: 0.3250 - accu
racy: 0.9057 - val_loss: 0.5160 - val_accuracy: 0.9041
Epoch 60/100
880/880 [==============================] - 1s 918us/step - loss: 0.5519 - accu
racy: 0.8670 - val_loss: 0.3971 - val_accuracy: 0.8699
Epoch 61/100
880/880 [==============================] - 1s 890us/step - loss: 0.4464 - accu
racy: 0.8795 - val_loss: 0.3483 - val_accuracy: 0.8904
Epoch 62/100
880/880 [==============================] - 1s 899us/step - loss: 0.4299 - accu
racy: 0.8898 - val_loss: 0.3617 - val_accuracy: 0.8973
Epoch 63/100
880/880 [==============================] - 1s 892us/step - loss: 0.3654 - accu
racy: 0.9000 - val_loss: 0.3087 - val_accuracy: 0.9041
Epoch 64/100
880/880 [==============================] - 1s 887us/step - loss: 0.3370 - accu
racy: 0.8920 - val_loss: 0.3969 - val_accuracy: 0.8767
Epoch 65/100
880/880 [==============================] - 1s 896us/step - loss: 0.3034 - accu
racy: 0.9068 - val_loss: 0.4493 - val_accuracy: 0.8904
Epoch 66/100
880/880 [==============================] - 1s 942us/step - loss: 0.2979 - accu
racy: 0.9159 - val_loss: 0.3979 - val_accuracy: 0.8904
Epoch 67/100
880/880 [==============================] - 1s 904us/step - loss: 0.3096 - accu
racy: 0.9182 - val_loss: 0.3697 - val_accuracy: 0.9110
Epoch 68/100
```

```
880/880 [==============================] - 1s 917us/step - loss: 0.3157 - accu
racy: 0.9045 - val_loss: 0.4778 - val_accuracy: 0.8767
Epoch 69/100
880/880 [==============================] - 1s 898us/step - loss: 0.2817 - accu
racy: 0.9239 - val_loss: 0.4622 - val_accuracy: 0.8904
Epoch 70/100
880/880 [==============================] - 1s 903us/step - loss: 0.2707 - accu
racy: 0.9170 - val_loss: 0.4896 - val_accuracy: 0.8630
Epoch 71/100
880/880 [==============================] - 1s 889us/step - loss: 0.3144 - accu
racy: 0.9068 - val_loss: 0.4358 - val_accuracy: 0.8904
Epoch 72/100
880/880 [==============================] - 1s 950us/step - loss: 0.2918 - accu
racy: 0.9057 - val_loss: 0.5584 - val_accuracy: 0.8904
Epoch 73/100
880/880 [==============================] - 1s 903us/step - loss: 0.3326 - accu
racy: 0.9068 - val_loss: 0.4943 - val_accuracy: 0.8904
Epoch 74/100
880/880 [==============================] - 1s 906us/step - loss: 0.2761 - accu
racy: 0.9125 - val_loss: 0.5915 - val_accuracy: 0.8699
Epoch 75/100
880/880 [==============================] - 1s 964us/step - loss: 0.2726 - accu
racy: 0.9261 - val_loss: 0.4829 - val_accuracy: 0.8904
Epoch 76/100
880/880 [==============================] - 1s 986us/step - loss: 0.8553 - accu
racy: 0.8943 - val_loss: 0.4993 - val_accuracy: 0.8904
Epoch 77/100
880/880 [==============================] - 1s 910us/step - loss: 0.4503 - accu
racy: 0.8841 - val_loss: 0.5316 - val_accuracy: 0.8973
Epoch 78/100
880/880 [==============================] - 1s 1ms/step - loss: 0.3089 - accura
cy: 0.9080 - val_loss: 0.5781 - val_accuracy: 0.8836
Epoch 79/100
880/880 [==============================] - 1s 1ms/step - loss: 0.2886 - accura
cy: 0.9227 - val_loss: 0.6791 - val_accuracy: 0.8493
Epoch 80/100
880/880 [==============================] - 1s 1ms/step - loss: 0.3143 - accura
cy: 0.8989 - val_loss: 0.5002 - val_accuracy: 0.8767
Epoch 81/100
880/880 [==============================] - 1s 1ms/step - loss: 0.2570 - accura
cy: 0.9261 - val_loss: 0.6536 - val_accuracy: 0.8425
Epoch 82/100
880/880 [==============================] - 1s 922us/step - loss: 0.2936 - accu
racy: 0.9136 - val_loss: 0.5697 - val_accuracy: 0.9110
Epoch 83/100
880/880 [==============================] - 1s 935us/step - loss: 0.2644 - accu
racy: 0.9216 - val_loss: 0.5580 - val_accuracy: 0.8973
Epoch 84/100
880/880 [==============================] - 1s 994us/step - loss: 0.2634 - accu
racy: 0.9216 - val_loss: 0.5092 - val_accuracy: 0.8836
Epoch 85/100
880/880 [==============================] - 1s 919us/step - loss: 0.2500 - accu
racy: 0.9227 - val_loss: 0.5464 - val_accuracy: 0.8973
Epoch 86/100
880/880 [==============================] - 1s 913us/step - loss: 0.2848 - accu
racy: 0.9080 - val_loss: 0.5612 - val_accuracy: 0.9041
Epoch 87/100
```

```
880/880 [==============================] - 1s 907us/step - loss: 0.2577 - accu
racy: 0.9227 - val_loss: 0.4365 - val_accuracy: 0.8904
Epoch 88/100
880/880 [==============================] - 1s 910us/step - loss: 0.2858 - accu
racy: 0.9261 - val_loss: 0.4951 - val_accuracy: 0.8836
Epoch 89/100
880/880 [==============================] - 1s 903us/step - loss: 0.3086 - accu
racy: 0.9034 - val_loss: 0.5078 - val_accuracy: 0.8630
Epoch 90/100
880/880 [==============================] - 1s 957us/step - loss: 0.2987 - accu
racy: 0.9205 - val_loss: 0.9739 - val_accuracy: 0.8630
Epoch 91/100
880/880 [==============================] - 1s 902us/step - loss: 0.2623 - accu
racy: 0.9227 - val_loss: 0.6721 - val_accuracy: 0.8904
Epoch 92/100
880/880 [==============================] - 1s 905us/step - loss: 0.2788 - accu
racy: 0.9170 - val_loss: 0.7108 - val_accuracy: 0.8836
Epoch 93/100
880/880 [==============================] - 1s 912us/step - loss: 0.2497 - accu
racy: 0.9284 - val_loss: 0.7413 - val_accuracy: 0.9110
Epoch 94/100
880/880 [==============================] - 1s 908us/step - loss: 0.2521 - accu
racy: 0.9295 - val_loss: 0.8465 - val_accuracy: 0.9178
Epoch 95/100
880/880 [==============================] - 1s 903us/step - loss: 0.3224 - accu
racy: 0.9170 - val_loss: 0.5542 - val_accuracy: 0.9041
Epoch 96/100
880/880 [==============================] - 1s 954us/step - loss: 0.2511 - accu
racy: 0.9307 - val_loss: 0.5453 - val_accuracy: 0.8904
Epoch 97/100
880/880 [==============================] - 1s 911us/step - loss: 0.2389 - accu
racy: 0.9284 - val_loss: 0.3941 - val_accuracy: 0.9041
Epoch 98/100
880/880 [==============================] - 1s 884us/step - loss: 0.4467 - accu
racy: 0.8875 - val_loss: 0.4598 - val_accuracy: 0.8699
Epoch 99/100
880/880 [==============================] - 1s 891us/step - loss: 0.3520 - accu
racy: 0.8932 - val_loss: 0.5015 - val_accuracy: 0.8836
Epoch 100/100
880/880 [==============================] - 1s 887us/step - loss: 0.3386 - accu
racy: 0.8898 - val_loss: 0.3184 - val_accuracy: 0.9178
```

Out[161…

```
<keras.callbacks.callbacks.History at 0x7f932376dfd0>
```

In [162…

```
# Step5.4) Predict on test data using above Model
result = model.predict(X_test)

# Step5.5) Calculate accuracy
cnt = 0
for i in range(len(Y_test)):
    if(np.amax(result[i]) < 0.5):
      pred = np.argmax(result[i])
    else:
      pred = np.argmax(result[i])
    if np.argmax(y_test[i]) == pred:
        cnt+=1

acc = str(round( cnt*100 / float(len(Y_test)),2))
print("Accuracy: " + acc + "%")

# Step5.6) Display accuracy and ConfMatrix
DisplayResult()
```

```
Accuracy: 91.28%
ACTIVITY Confusion Matrix:
Activities3      |Calling         |Clapping        |Falling         |Sweeping
 |WashingHand     |WatchingTV      |other
Activities3      | 14     |0      |0       |0       |0       |0       |0       |0
Calling          | 0      |6      |0       |2       |0       |0       |0       |4
Clapping         | 0      |0      |12      |0       |0       |0       |0       |0
Falling          | 0      |0      |0       |11      |1       |0       |0       |0
Sweeping         | 0      |0      |0       |0       |6       |0       |0       |0
WashingHand      | 0      |0      |0       |0       |0       |4       |0       |2
WatchingTV       | 0      |0      |0       |0       |1       |0       |1       |4
other    | 0     |0       |0      |2       |3       |0       |0       |145

ACTIVITY  Matrix:
Activities3      |Calling         |Clapping        |Falling         |Sweeping
 |WashingHand     |WatchingTV      |other
Activities3      | 1.0    |0.0    |0.0     |0.0     |0.0     |0.0     |0.0     |0.0
Calling          | 0.0    |0.5    |0.0     |0.17    |0.0     |0.0     |0.0     |0.33
Clapping         | 0.0    |0.0    |1.0     |0.0     |0.0     |0.0     |0.0     |0.0
Falling          | 0.0    |0.0    |0.0     |0.92    |0.08    |0.0     |0.0     |0.0
Sweeping         | 0.0    |0.0    |0.0     |0.0     |1.0     |0.0     |0.0     |0.0
WashingHand      | 0.0    |0.0    |0.0     |0.0     |0.0     |0.67    |0.0     |0.33
WatchingTV       | 0.0    |0.0    |0.0     |0.0     |0.17    |0.0     |0.17    |0.67
other    | 0.0   |0.0     |0.0    |0.01    |0.02    |0.0     |0.0     |0.97
```

In [163…

```
# Step6) Save Model for future use

model_path = r"Audio_Classification_Model/"
model_name = "audio_CNN_model"



model_json = model.to_json()

with open(model_path + model_name +".json", "w") as json_file:
    json_file.write(model_json)
    print("SAVED MODEL !!! Model saved to folder ./" + model_path + " as " + r

# serialize weights to HDF5
model.save_weights(model_path + model_name + ".h5")
print("SAVED WEIGHTS !!! Serialized weights stored to folder ./" + model_path
```

```
SAVED MODEL !!! Model saved to folder ./Audio_Classification_Model/ as audio_C
NN_model.json
SAVED WEIGHTS !!! Serialized weights stored to folder ./Audio_Classification_M
odel/ as audio_CNN_model.h5
```

In [ ]: