Assignment #3
Speech Emotion Recognition

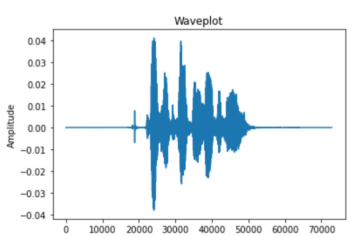
Seif El Din Wael Salem 18010832
Ziad Hassan Mahmoud 18010720
Mohamed Mostafa Badran 18011621

Download the Dataset and Understand the Format:

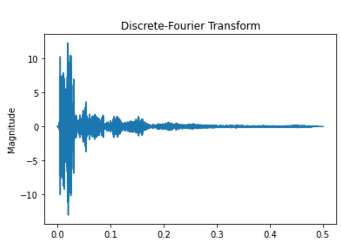
```
def load_plt_audio(path):
    audio_sig,sample_rate  = librosa.load(path)#dtrunc(ata.size/sample rate) = audio length
    #plot wave plot
    plt.plot(audio_sig)#reading of pressure wave form of peaks and troughs "negative peaks"
    plt.title('Waveplot')
    plt.xlabel('Time')
    plt.ylabel('Amplitude')
    plt.show()
    #plot frequencies
    spectrum = np.fft.fft(audio_sig)# we use Fast Fourier Transform not Fourier Transform as
our data is discrete
    frequencies = np.fft.fftfreq(len(spectrum))
    magnitude = np.abs(spectrum)
    plt.plot(frequencies[:int(len(magnitude)/2)],spectrum[:int(len(magnitude)/2)])#only plot
left half as right half is the same
    plt.title('Discrete-Fourier Transform')
    plt.xlabel('Frequency')
    plt.ylabel('Magnitude')
    plt.show()

    return  sample_rate, audio_sig
```

```
print("neutral")
path = '../input/speech-emotion-recognition-en/Ravdess/audio_speech_actors_01-24/Actor_01/03-01-
01-01-01-01-01.wav'
sample_rate, audio_sig =load_plt_audio(path)
IPython.display.Audio(rate= sample_rate,data= audio_sig)
```

```
neutral
```




```
/opt/conda/lib/python3.7/site-packages/matplotlib/cbook/__init__.py:1298: ComplexWarning: Ca
sting complex values to real discards the imaginary part
  return np.asarray(x, float)
```



0:00 / 0:03

Create the Feature Space:

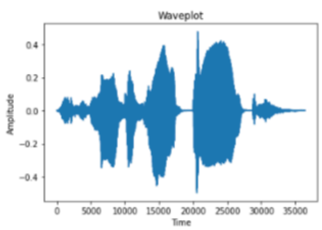
```
def load_audio(path):
    sig, sample_rate = librosa.load(path,offset=0.3)
    new_siganl = np.zeros(64745)
    new_siganl[:len(sig)] =  sig[:64745]
    return new_siganl,sample_rate
```

1D:

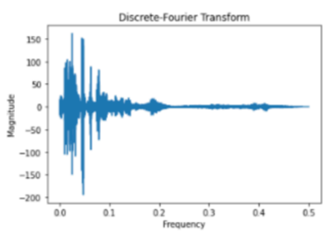
```
def zcr(data, frame_length=2048, hop_length=512):
    zcr = librosa.feature.zero_crossing_rate(y=data, frame_length=frame_length, hop_length=h
op_length)
    return np.squeeze(zcr)


def energy(data, frame_length=2048, hop_length=512):
    en = np.array([np.sum(np.power(np.abs(data[hop:hop+frame_length]), 2)) for hop in range
(0, data.shape[0], hop_length)])
    return en / frame_length


def rmse(data, frame_length=2048, hop_length=512):
    rmse = librosa.feature.rms(y=data, frame_length=frame_length, hop_length=hop_length)
    return np.squeeze(rmse)


def entropy_of_energy(data, frame_length=2048, hop_length=512):
    energies = energy(data, frame_length, hop_length)
    energies /= np.sum(energies)

    entropy = 0.0
    entropy -= energies * np.log2(energies)
    return entropy


def spc(data, sr, frame_length=2048, hop_length=512):
    spectral_centroid = librosa.feature.spectral_centroid(y=data, sr=sr, n_fft=frame_length,
hop_length=hop_length)
    return np.squeeze(spectral_centroid)


def spc_flux(data):
    isSpectrum = data.ndim == 1
    if isSpectrum:
        data = np.expand_dims(data, axis=1)
    X = np.c_[data[:, 0], data]
    af_Delta_X = np.diff(X, 1, axis=1)
    vsf = np.sqrt((np.power(af_Delta_X, 2).sum(axis=0))) / X.shape[0]

    return np.squeeze(vsf) if isSpectrum else vsf


def spc_rollof(data, sr, frame_length=2048, hop_length=512):
    spcrollof = librosa.feature.spectral_rolloff(y=data, sr=sr, n_fft=frame_length, hop_leng
th=hop_length)
    return np.squeeze(spcrollof)

def mfcc(data, sr, frame_length=2048, hop_length=512, flatten: bool = True):
    mfcc_feature = librosa.feature.mfcc(y=data, sr=sr)
    return np.squeeze(mfcc_feature.T) if not flatten else np.ravel(mfcc_feature.T)
```

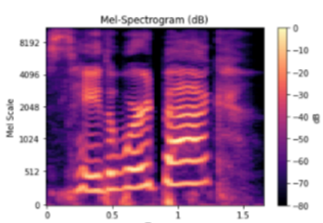
2D:

```
path = '../input/speech-emotion-recognition-en/Tess/YAF_fear/YAF_back_fear.wav'
samplerate, audio_sig =load_plt_audio(path)

#Compute a mel-scaled spectrogram.
mel_signal = librosa.feature.melspectrogram(y=audio_sig, sr=samplerate)
# gathering the absolute values for all values in our audio_stft
spectrogram = np.abs(mel_signal)
# Converting the amplitude to decibels
power_to_db = librosa.power_to_db(spectrogram, ref=np.max)


librosa.display.specshow(power_to_db, sr=samplerate, x_axis='time', y_axis='mel', cmap='magma')
plt.colorbar(label='dB')
plt.title('Mel-Spectrogram (dB)')
plt.xlabel('Time')
plt.ylabel('Mel Scale')
plt.show()
```




```
/opt/conda/lib/python3.7/site-packages/matplotlib/cbook/__init__.py:1298: ComplexWarning: Casting complex values to real discards the imaginary part
  return np.asarray(x, float)
```






```
for i in df.index:
    sig, sample_rate  = load_audio(df["Path"][i])
    mel_signal = librosa.feature.melspectrogram(y=sig, sr=sample_rate)
    spectrogram = np.abs(mel_signal)
    power_to_db = librosa.power_to_db(spectrogram, ref=np.max)
    X.append(power_to_db)
    Y.append( df['Emotion'][i])
```

Building the Model:

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, random_state=42, test_size=0.3, s
huffle=True)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((7922, 16256), (3396, 16256), (7922, 6), (3396, 6))
```

1D

```
def Res1D(layer_in, n_filters):
    x_skip = layer_in
    x = layers.Conv1D(n_filters, kernel_size=3, strides=1,padding="same")(layer_in)
    x = layers.BatchNormalization()(x)
    x = layers.Activation("LeakyReLU")(x)
    x = layers.Conv1D(n_filters, kernel_size=3, strides=1,padding="same")(x)
    x = layers.BatchNormalization()(x)
    x = layers.Add()([x, x_skip])
    x = layers.Activation("LeakyReLU")(x)
    return x
```

```
model = models.Sequential()
model.add(layers.Conv1D(512, kernel_size=5, strides=1,
                        padding="same", activation="relu",
                        input_shape=(X_train.shape[1], 1)))
model.add(layers.BatchNormalization())
model.add(layers.MaxPool1D(pool_size=5, strides=2, padding="same"))

model.add(layers.Conv1D(512, kernel_size=5, strides=1,
                        padding="same", activation="relu"))
model.add(layers.BatchNormalization())
model.add(layers.MaxPool1D(pool_size=5, strides=2, padding="same"))

model.add(layers.Conv1D(256, kernel_size=5, strides=1,
                        padding="same", activation="relu"))
model.add(layers.BatchNormalization())
model.add(layers.MaxPool1D(pool_size=5, strides=2, padding="same"))

model.add(layers.Conv1D(256, kernel_size=3, strides=1, padding='same', activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling1D(pool_size=5, strides = 2, padding = 'same'))

model.add(layers.Conv1D(128, kernel_size=3, strides=1, padding='same', activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling1D(pool_size=3, strides = 2, padding = 'same'))

model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.Dense(6, activation="softmax"))

model.compile(optimizer="rmsprop", loss="categorical_crossentropy", metrics=["acc", f_scor
e])
EPOCHS = 50
batch_size = 64
model.summary()
```

2D:

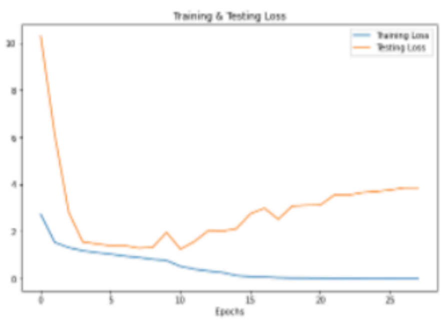
```
def Res2D(layer_in, n_filters):
    x_skip = layer_in
    x = layers.Conv2D(n_filters, kernel_size=(3,3), strides=(1,1),padding="same")(layer_in)
    x = layers.BatchNormalization()(x)
    x = layers.Activation("LeakyReLU")(x)
    x = layers.Conv2D(n_filters, kernel_size=(3,3), strides=(1,1),padding="same")(x)
    x = layers.BatchNormalization()(x)
    x = layers.Add()([x, x_skip])
    x = layers.Activation("LeakyReLU")(x)
    return x
```

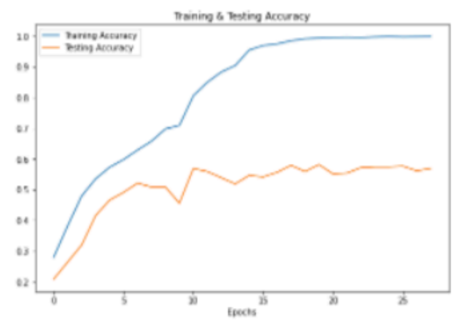
```
model_input = keras.Input((X_train.shape[1],X_train.shape[2],1),  name="audio")

x = layers.Conv2D(128, kernel_size=(3,3), strides=(3,3),padding="same")(model_input)
x = Res2D(x,128)
x = layers.MaxPool2D(pool_size=(3,3), strides=(3,3), padding="same")(x)

x = Res2D(x,128)
x = layers.MaxPool2D(pool_size=(3,3), strides=(3,3), padding="same")(x)

x = layers.Conv2D(256, kernel_size=(3,3), strides=(1,1),padding="same")(x)
x = Res2D(x,256)
x = layers.MaxPool2D(pool_size=(3,3), strides=(3,3), padding="same")(x)

x = Res2D(x,256)
x = layers.MaxPool2D(pool_size=(3,3), strides=(3,3), padding="same")(x)

x = Res2D(x,256)
x = layers.MaxPool2D(pool_size=(3,3), strides=(3,3), padding="same")(x)

x = Res2D(x,256)
x = layers.MaxPool2D(pool_size=(3,3), strides=(3,3), padding="same")(x)


x = Res2D(x,256)
x = layers.MaxPool2D(pool_size=(3,3), strides=(3,3), padding="same")(x)


x = layers.Conv2D(512, kernel_size=(3,3), strides=(3,3),padding="same")(x)
x = Res2D(x,512)
x = layers.MaxPool2D(pool_size=(3,3), strides=(3,3), padding="same")(x)
x = layers.Conv2D(512, kernel_size=(3,3), strides=(1,1),padding="same")(x)

x = layers.Flatten()(x)
model_output = layers.Dense(6, activation="softmax")(x)


model = keras.Model(model_input, model_output, name="2D-CNN")
model.summary()
model.compile(optimizer="rmsprop", loss="categorical_crossentropy", metrics=["acc", f_scor
e])
```

Big Picture:

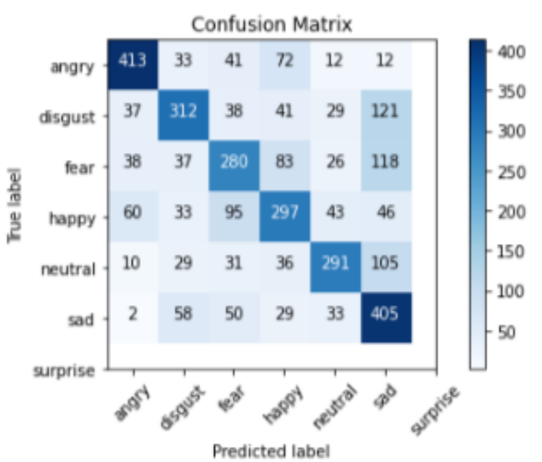
1D:

```
107/107 [==============================] - 14s 126ms/step - loss: 3.2303 - acc: 0.5883 -
f_score: 0.5892
Accuracy of our model on test data :  58.83392095565796 %
```



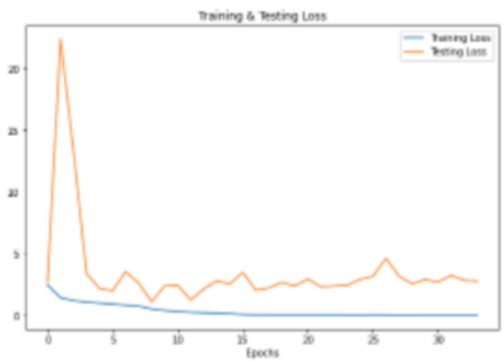



```
Confusion matrix, without normalization
[[413  33  41  72  12  12]
 [ 37 312  38  41  29 121]
 [ 38  37 280  83  26 118]
 [ 60  33  95 297  43  46]
 [ 10  29  31  36 291 105]
 [  2  58  50  29  33 405]]
```

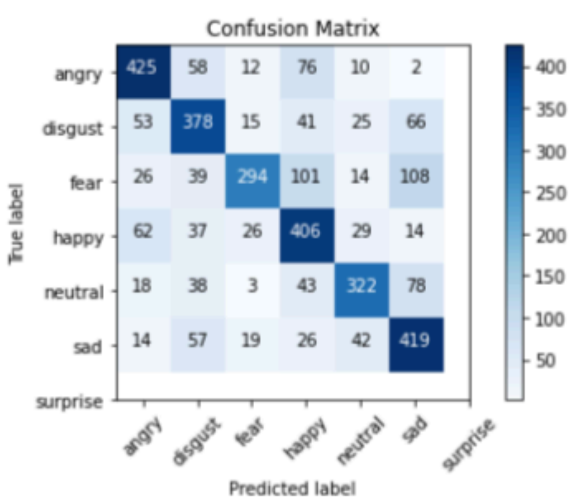


2D

```
107/107 [==============================] - 46s 426ms/step - loss: 2.2974 - acc: 0.6608 -
f_score: 0.6612
Accuracy of our model on test data :  66.07773900032043 %
```




```
Confusion matrix, without normalization
[[425  58  12  76  10   2]
 [ 53 378  15  41  25  66]
 [ 26  39 294 101  14 108]
 [ 62  37  26 406  29  14]
 [ 18  38   3  43 322  78]
 [ 14  57  19  26  42 419]]
```



the most confusing classes are fear and neutral