<u>Pattern Recognition</u>

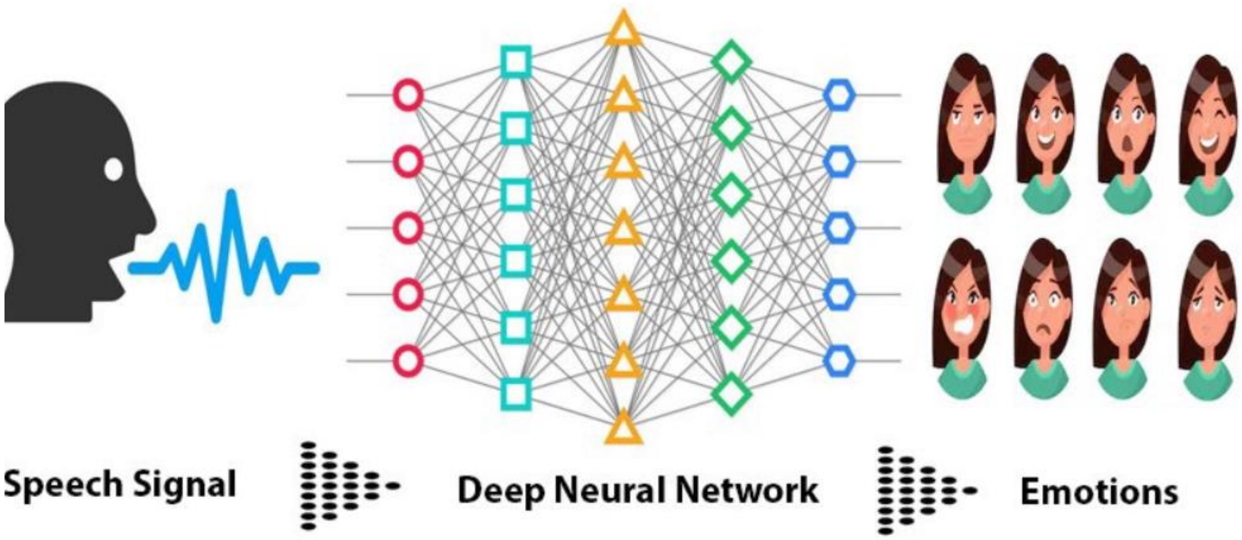
<u>Assignment 3</u>

<u>Speech Emotion Recognition</u>

Made By:
Salma Barakat
Yasmine Ashraf Eshra
Habiba Bakr

1)Introduction:

Speech Emotion Recognition (SER) is a field of artificial intelligence that focuses on detecting and interpreting emotions conveyed through human speech. The goal of SER is to develop algorithms and techniques that can accurately identify the emotional state of a speaker based on their speech signal.



- **Dataset used**:

CREMA-D is a data set of 7,442 original clips from 91 actors, Actors spoke from a selection of 12 sentences. The sentences were presented using one of six different emotions (Anger, Disgust, Fear, Happy, Neutral and Sad) and four different emotion levels (Low, Medium, High and Unspecified). The filenames of each CREMA-D audio files are following a naming convention consists of 4 blocks, separated by underscores.

2- Downloading the Dataset and Understanding the Format:

CREMA dataset consists of 7442 audios, we classify them according to the emotions as follows:

```python
# classify crema dataset into classes according to their emotions
audio_path = []
audio_emotion = []
path = '/kaggle/input/speech-emotion-recognition-en/Crema/'
directory_path = os.listdir(path)
# collects all the audio filename in the variable 'path'
# directory_path = sorted(os.listdir(path))
print(len(directory_path))
for audio in directory_path:
    if audio.endswith('zip'):
      continue
    else:
      audio_path.append(path + "/"+audio)
      emotion = audio.split('_')
      if emotion[2] == 'SAD':
          audio_emotion.append("sad")
      elif emotion[2] == 'ANG':
          audio_emotion.append("angry")
      elif emotion[2] == 'DIS':
          audio_emotion.append("disgust")
      elif emotion[2] == 'NEU':
          audio_emotion.append("neutral")
      elif emotion[2] == 'HAP':
          audio_emotion.append("happy")
      elif emotion[2] == 'FEA':
          audio_emotion.append("fear")
      else:
          audio_emotion.append("unknown")
#dataframe for labels
emotion_dataset = pd.DataFrame(audio_emotion, columns=['Emotions'])
#dataframe for audios
audio_path_dataset = pd.DataFrame(audio_path, columns=['Path'])
#dataframe for audio with its labels
dataset = pd.concat([audio_path_dataset, emotion_dataset], axis= 1)
print(dataset)
```

```
7442
                                                   Path Emotions
0     /kaggle/input/speech-emotion-recognition-en/Cr...  disgust
1     /kaggle/input/speech-emotion-recognition-en/Cr...    happy
2     /kaggle/input/speech-emotion-recognition-en/Cr...    happy
3     /kaggle/input/speech-emotion-recognition-en/Cr...  disgust
4     /kaggle/input/speech-emotion-recognition-en/Cr...  disgust
...                                                 ...      ...
7437  /kaggle/input/speech-emotion-recognition-en/Cr...    angry
7438  /kaggle/input/speech-emotion-recognition-en/Cr...    angry
7439  /kaggle/input/speech-emotion-recognition-en/Cr...    angry
7440  /kaggle/input/speech-emotion-recognition-en/Cr...      sad
7441  /kaggle/input/speech-emotion-recognition-en/Cr...      sad

[7442 rows x 2 columns]
```

3)Load audio and listen to each class:

- we load data using librosa library to obtain waveform and sampling rate of audios.
- we get maximum length of audio to pad all audios to reach same length.

```
#load audios
audio_arrays = []
# signal variable contains the waveform as 1-dimensional NumPy array
# sr is the sampling rate of the audio
for i in dataset['Path']:
    x, sr = librosa.load(i)
    audio_arrays.append(x)
```

```
max=0
for i in range(7442):
    if len(audio_arrays[i])>max:
        max = len(audio_arrays[i])
print(max)
```

```
for i in range(7442):
  if len(audio_arrays[i]) < max:
    difference = max - len(audio_arrays[i])
    # padding array using CONSTANT mode
    audio_arrays[i] = np.pad(audio_arrays[i], (0, difference), 'constant')
```

```
dataset['Arrays'] = audio_arrays
```

- we made a function to play a specified audio and plot its waveform.

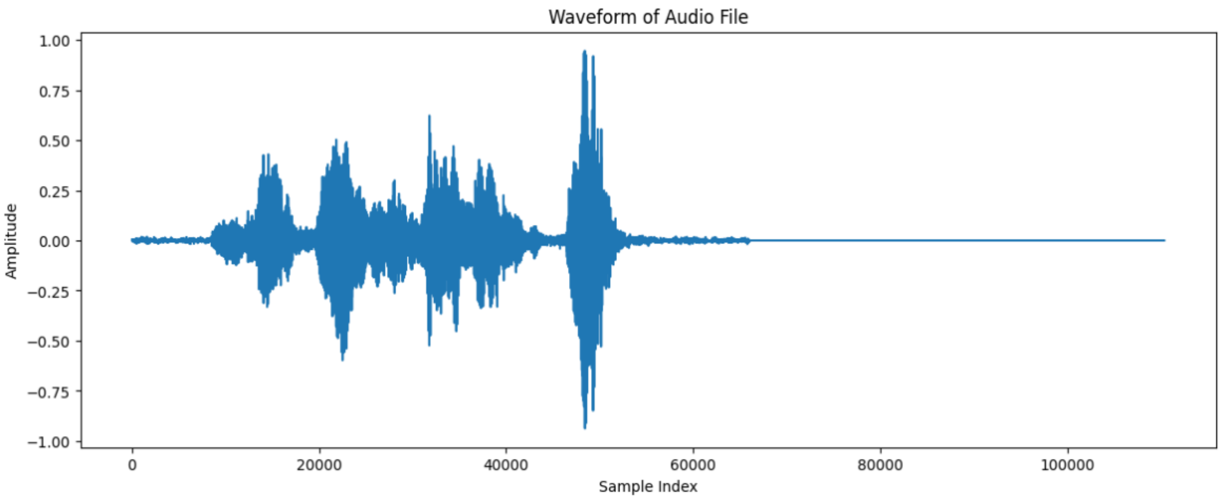
```
#function for listen and plot waveform audio
def lisandplot(y,emotion):
  audio_file_path=dataset[dataset['Emotions']==emotion]['Path'].iloc[y]
  array=dataset[dataset['Emotions']==emotion]['Arrays'].iloc[y]
  name=audio_file_path.split("/")
  print("audio name:",name[5],"\n")
  audio=Audio(audio_file_path, autoplay=True)
  display(audio)
  # Plot the waveform
  plt.figure(figsize=(14, 5))
  plt.plot(array)
  plt.xlabel('Sample Index')
  plt.ylabel('Amplitude')
  plt.title('Waveform of Audio File')
  plt.show()
```

- play and plot waveform of first audio in class angry

```
#angry
lisandplot(0,'angry')
```

```
audio name: 1079_TSI_ANG_XX.wav
```

0:02 / 0:02

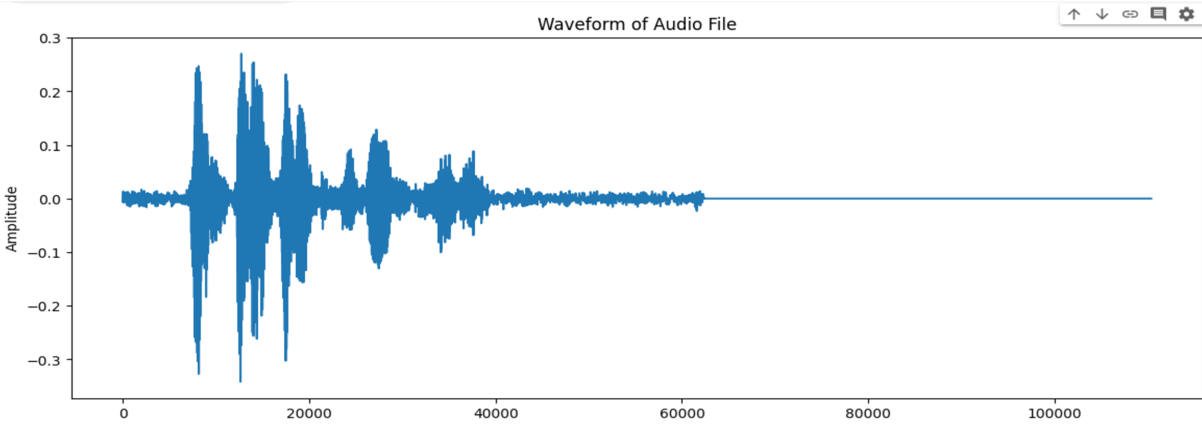


- Play and plot waveform of second audio in class disgust

```
#disgust
lisandplot(1,'disgust')
```

```
audio name: 1079_WSI_DIS_XX.wav
```

0:02 / 0:02



4) Create the Feature Space:

In 1D: (Time domain):

- Zero crossing rate (ZCR) represents the rate at which the signal changes from positive to negative or vice versa. It is computed by counting the number of times the signal crosses the zero-axis in a given each frame. It is a useful feature for analyzing the temporal structure of speech or audio signals, and can be useful component of a feature set for speech or audio recognition tasks.
- Energy: It represents the total magnitude of the signal in a given time window and can be computed by summing the squared amplitude of the signal over the window. can provide useful information about the overall level of loudness or intensity of the speech or audio signal, as well as the presence of specific events or sounds.

In 2D: (Mel_spectogram):

- To compute the Mel spectrogram of a speech or audio signal, we typically break up the signal into short overlapping windows, and compute the power spectrum of each window using a Fourier transform. We then apply a set of Mel filters to the power spectrum to obtain the Mel spectrogram. The resulting Mel spectrogram values are often organized into a matrix that can be used as input to a machine learning model

```
def find_FS(audio):
  #audio in time domain
  # Zero crossing rate
  zcr = librosa.feature.zero_crossing_rate(audio).T

  # MelSpectogram
  #n_ftt -> length of fft window
  #sr -> sampling rate
  mel =librosa.feature.melspectrogram(y=audio, sr=sr)

  # Energy
  frame_length=2048
  hop_length=512
  energy = np.array([np.sum(np.power(np.abs(audio[hop:hop+frame_length]), 2)) for hop in range(0, audio.shape[0], hop_length)])
  normalized_energy = energy/frame_length

  feature_space=np.append(zcr,normalized_energy)
  return mel, feature_space
```

```
def getFeatures(data):
  y = []
  melSpec=[]
  time=[]
  dp_melSpec=[]
  for i in range(len(data)):
        mel,dp,features=find_FS(data[i]);
        melSpec.append(mel)     #melspectogram (second feature)
        time.append(features)   #zero crossing rate &energy(first feature)
  return np.array(melSpec), np.array(time)
```

4- Building the Model:

- We split the data into 70% training and validation and 30% testing.
- We use 5% of the training and validation data for validation.

```
#70% train & validation ,30% test
xtrain, xtest, ytrain, ytest = train_test_split(audio_arrays,audio_emotion, test_size=0.3,stratify=audio_emotion,random_state=42)
#95% train & 5% validation
xtrain, xvalid, ytrain, yvalid = train_test_split(xtrain, ytrain, test_size=0.05,stratify=ytrain,random_state=42)
```

- Then, we get the feature space of each of the training data, validation data and test data.

```
xtrain_spec, xtrain_time  = getFeatures(xtrain)
xvalid_spec, xvalid_time  = getFeatures(xvalid)
xtest_spec, xtest_time  = getFeatures(xtest)
```

- We convert the labels from categorical data to numerical data and fix the shapes of data to use them in the model.

```
encoder  = OneHotEncoder()
ytrain=shape(np.array(ytrain))
ytest=shape(np.array(ytest))
yvalid=shape(np.array(yvalid))
```

```
def shape(arr):
  arr = arr.reshape((arr.shape[0], 1))
  arr = encoder.fit_transform(np.array(arr).reshape(-1,1)).toarray()
  return arr
```

```
xtrain_time = xtrain_time.reshape((xtrain_time.shape[0],xtrain_time.shape[1], 1))
xtest_time = xtest_time.reshape((xtest_time.shape[0],xtest_time.shape[1], 1))
xvalid_time = xvalid_time.reshape((xvalid_time.shape[0],xvalid_time.shape[1],1))
xtrain_spec = xtrain_spec.reshape((xtrain_spec.shape[0],xtrain_spec.shape[1],xtrain_spec.shape[2], 1))
xvalid_spec = xvalid_spec.reshape((xvalid_spec.shape[0],xvalid_spec.shape[1],xvalid_spec.shape[2], 1))
xtest_spec = xtest_spec.reshape((xtest_spec.shape[0],xtest_spec.shape[1],xtest_spec.shape[2], 1))
```

1D CNN Model: Time Domain Feature Space:

The model consists of:

- 1st layer: 512 filters with size (5, 5) and strides = 1, then a maxPool layer of size (2, 2) and strides = 2.

- 2nd layer: 128 filters with size (5, 5) and strides = 1, then a maxPool layer of size (2, 2) and strides = 2.

- A drop out layer = 0.2 to reduce the overfit.

- 3rd layer: 64 filters with size (5, 5) and strides = 1, then a maxPool layer of size (2, 2) and strides = 2.

- Flattening the output to 1D vector.

- A fully connected layer with 64 units and using RelU activation function.

- A drop out layer = 0.2 to reduce the overfit.

```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d_8 (Conv1D)           (None, 432, 512)          3072

 max_pooling1d_7 (MaxPooling  (None, 216, 512)         0
 1D)

 conv1d_9 (Conv1D)           (None, 216, 128)          327808

 max_pooling1d_8 (MaxPooling  (None, 108, 128)         0
 1D)

 dropout_13 (Dropout)        (None, 108, 128)          0

 conv1d_10 (Conv1D)          (None, 108, 64)           41024

 max_pooling1d_9 (MaxPooling  (None, 54, 64)           0
 1D)

 flatten_11 (Flatten)        (None, 3456)              0

 dense_24 (Dense)            (None, 64)                221248

 dropout_14 (Dropout)        (None, 64)                0

 dense_25 (Dense)            (None, 32)                2080

 dense_26 (Dense)            (None, 6)                 198

=================================================================
Total params: 595,430
Trainable params: 595,430
Non-trainable params: 0
_________________________________________________________________
```

- A fully connected layer with 32 units and using RelU activation function.

- Output layer with 6 units (corresponding to the number of classes) and softmax activation.

2D CNN Model: Mel Spectogram Feature Space:

The model consists of:

- 1st layer: 256 filters with size (3, 3) and strides = 1, then a maxPool layer of size (2, 2) and strides = 2.

- 2nd layer: 128 filters with size (4, 4) and strides = 1, then a maxPool layer of size (2, 2) and strides = 2.

- 3rd layer: 64 filters with size (4, 4) and strides = 1, then a maxPool layer of size (2, 2) and strides = 2.

- Flattening the output to 1D vector.

- A fully connected layer with 64 units and using RelU activation function.

- A drop out layer = 0.3 to reduce the overfit.

- Output layer with 6 units (corresponding to the number of classes) and softmax activation.

```
Model: "sequential_14"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_38 (Conv2D)          (None, 128, 216, 256)     2560

 max_pooling2d_38 (MaxPoolin  (None, 64, 108, 256)     0
 g2D)

 conv2d_39 (Conv2D)          (None, 64, 108, 128)      524416

 max_pooling2d_39 (MaxPoolin  (None, 32, 54, 128)      0
 g2D)

 conv2d_40 (Conv2D)          (None, 32, 54, 64)        131136

 max_pooling2d_40 (MaxPoolin  (None, 16, 27, 64)       0
 g2D)

 flatten_14 (Flatten)        (None, 27648)             0

 dense_31 (Dense)            (None, 64)                1769536

 dropout_17 (Dropout)        (None, 64)                0

 dense_32 (Dense)            (None, 6)                 390

=================================================================
Total params: 2,428,038
Trainable params: 2,428,038
Non-trainable params: 0
_________________________________________________________________
```

5-The big picture:

For 1D model:

We ran the model on 30 epochs and it produced test accuracy of 45.59 %

```
Epoch 28/30
155/155 [==============================] - 2s 12ms/step - loss: 1.1704 - accuracy: 0.5277 - val_loss: 1.4030 - val_accuracy: 0.4368
Epoch 29/30
155/155 [==============================] - 2s 12ms/step - loss: 1.1550 - accuracy: 0.5331 - val_loss: 1.4528 - val_accuracy: 0.4444
Epoch 30/30
155/155 [==============================] - 2s 11ms/step - loss: 1.1527 - accuracy: 0.5388 - val_loss: 1.4066 - val_accuracy: 0.4559
```

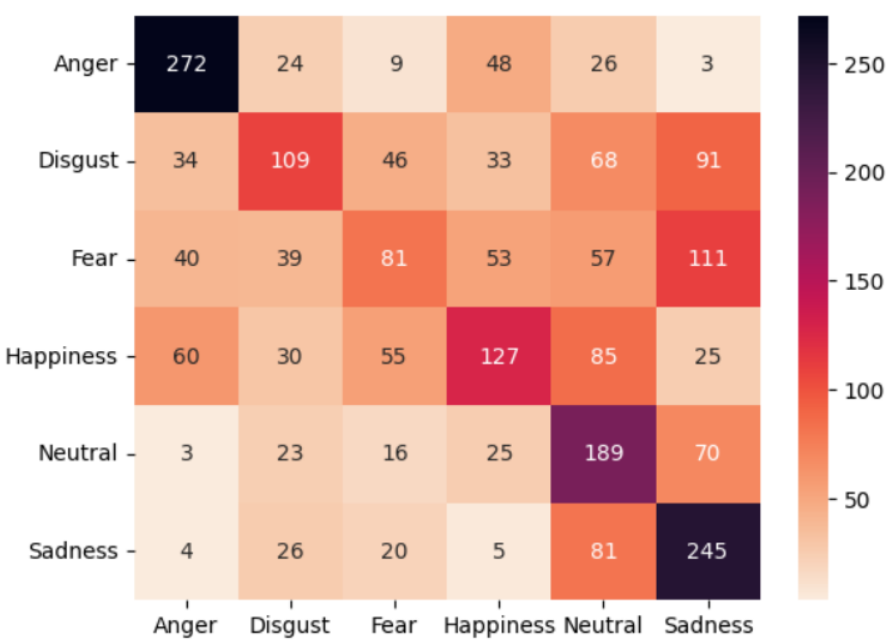
Computing the accuracy and F-Score for the 1D model and plotting the confusion matrices and finding the most confusing classes:

```
y_pred_time = model_1d.predict(xtest_time)
matrix_time = confusion_matrix(ytest.argmax(axis=1), y_pred_time.argmax(axis=1))
ax = sns.heatmap(matrix_time, annot=True, fmt="d", cmap = 'rocket_r',
                 xticklabels = ['Anger', 'Disgust', 'Fear', 'Happiness', 'Neutral', 'Sadness'],
                 yticklabels = ['Anger', 'Disgust', 'Fear', 'Happiness', 'Neutral', 'Sadness'])
print(classification_report(ytest.argmax(axis=1), y_pred_time.argmax(axis=1)))
```

```
70/70 [==============================] - 0s 5ms/step
              precision    recall  f1-score   support

           0       0.66      0.71      0.68       382
           1       0.43      0.29      0.34       381
           2       0.36      0.21      0.27       381
           3       0.44      0.33      0.38       382
           4       0.37      0.58      0.45       326
           5       0.45      0.64      0.53       381

    accuracy                           0.46      2233
   macro avg       0.45      0.46      0.44      2233
weighted avg       0.45      0.46      0.44      2233
```



For 2D model:

We ran the model on 25 epochs and it produced test accuracy of 48.66 %

```
155/155 [==============================] - 32s 205ms/step - loss: 1.0894 - accuracy: 0.5762 - val_loss: 1.7375 - val_accuracy: 0.4406
Epoch 24/25
155/155 [==============================] - 32s 205ms/step - loss: 1.1036 - accuracy: 0.5734 - val_loss: 1.8604 - val_accuracy: 0.4330
Epoch 25/25
155/155 [==============================] - 32s 207ms/step - loss: 1.1033 - accuracy: 0.5728 - val_loss: 1.7876 - val_accuracy: 0.4866
```

```
70/70 [==============================] - 3s 38ms/step
              precision    recall  f1-score   support

           0       0.58      0.64      0.61       382
           1       0.44      0.35      0.39       381
           2       0.45      0.24      0.31       381
           3       0.38      0.40      0.39       382
           4       0.40      0.61      0.48       326
           5       0.52      0.57      0.54       381

    accuracy                           0.46      2233
   macro avg       0.46      0.47      0.45      2233
weighted avg       0.46      0.46      0.45      2233
```