Pattern Recognition

Assignment 3

Speech Emotion Recognition

Done By:

Shimaa Ahmed 5556

Fagr Hesham 5886

Raghda Sallam 5837

Heidi Zamzam 5742

Mayar Abuzeid 5773

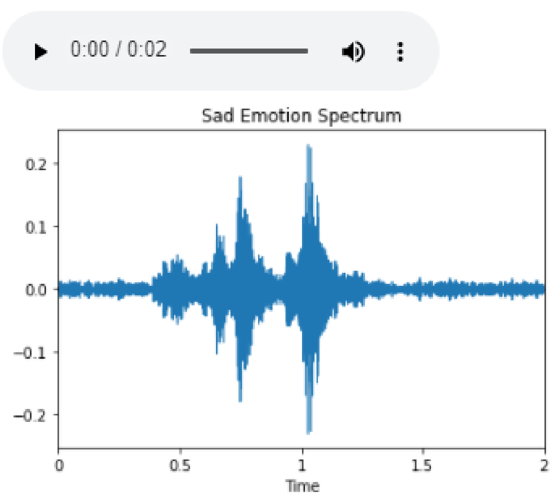
Link for our code:

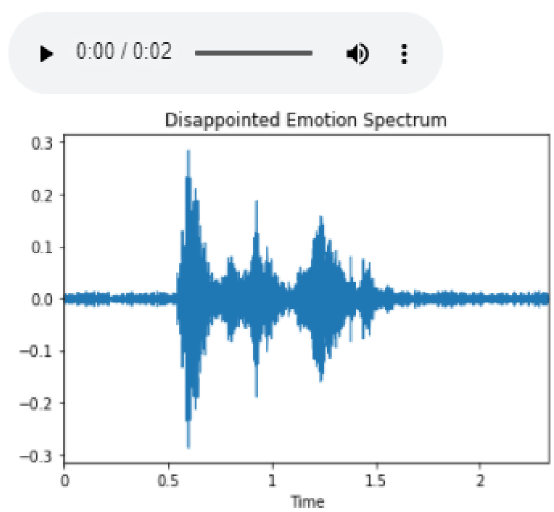
https://colab.research.google.com/drive/1oBIPyoCSmaZGtwLmAn-atphKVXnjzwzy?usp=sharing

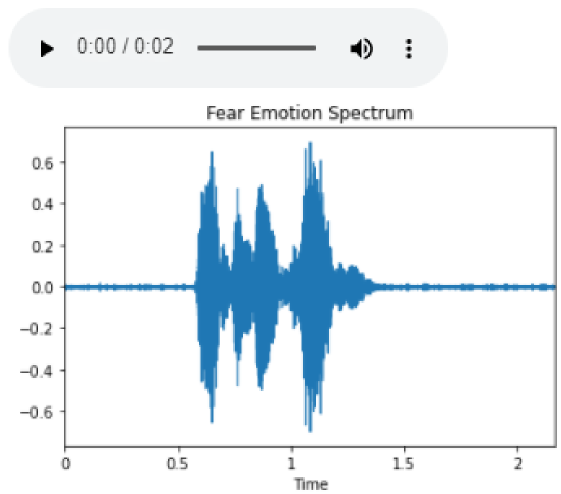
Code Explanation:

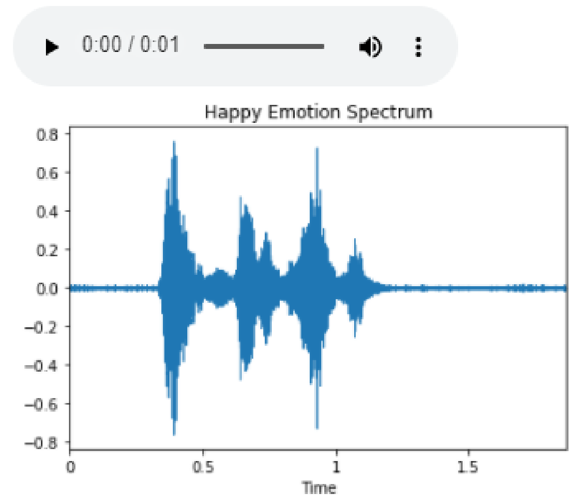
1. Download the Dataset

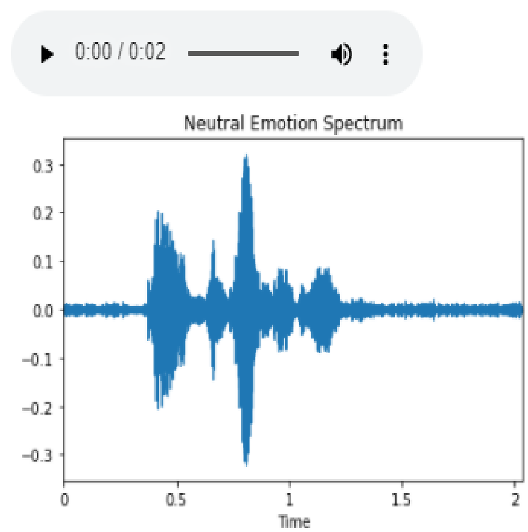
- First the Crema dataset is downloaded using the librosa.load function which returns the audio signals and the sampling frequency.
- Then we divided the audio signals into 5 sound states which are: `'SAD', 'DISAPPOINTED', 'FEAR', 'HAPPY', 'NEUTRAL','ANGRY'` where each state has a different label vector which has a 1 in the the corresponding state and 0 in the remaining states,one of these label vectors are appended to our main label vector y according to which sound it is.
- Audios for each state can be listened to and Spectrums are plotted as shown below.

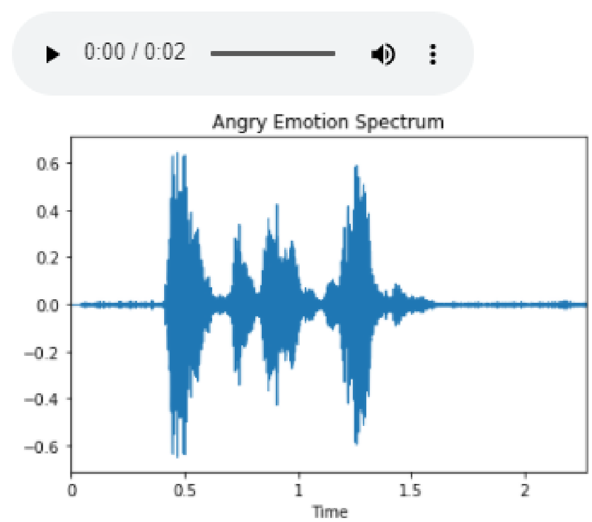












2. Creating Feature Spaces for 1D

- In this section two feature spaces are created from the audio one for Zero-crossing rate and the other is for the Energy.

1-Zero-Crossing Rate

- Zero-Crossing is the rate the signal crosses the x-axis i.e the wave changes from +ve to -ve and vice versa. A very simple way for measuring smoothness of a signal is to calculate the number of zero-crossing within a segment of that signal. A voice signal oscillates slowly - for example, a 100 Hz signal will cross zero 100 per second - whereas an unvoiced fricative can have 3000 zero crossing per second.

- The zero crosses are brought using the function librosa.feature.zero_crossing_rate.
- Splitting and reshaping zero-crossing dataset. The data is splitted 70 **%** for the training set and 30% for the test set which is done using train_test_split function.This function is used again for the training and validation to have a 5**%** validation.
- 1D Model for Zero-crossing.The following layers are used Convolution,Maxpooling,Dropout,Flatten and Dense.
- Next the validation loss and training loss are plotted for Zero-crossing.
- Accuracy of test set of Zero-crossing is calculated.
- Confusion matrix of Zero-crossing is calculated and plotted.
- F-score of zero crossing is calculated using the built-in function f1_score which takes in as input the true and predicted labels.

Screenshots:

Splitting of Zero-crossing

```
print(zeroTraining.shape)
print(yzeroTraining.shape)
print(zeroValidation.shape)
print(yzeroValidation.shape)
```

```
(4948, 216, 1)
(4948, 6)
(261, 216, 1)
(261, 6)
```

Zero-crossing 1D Model

1D Models for Zero Crossing

```
[ ] input_layer=Input(shape=(zeroTraining.shape[1],1))
    model=Conv1D(64,kernel_size=(3), strides=1,padding = 'same')(input_layer)
    model=Conv1D(64,kernel_size=(3), strides=1,padding = 'same')(model)
    model=MaxPooling1D(pool_size=(2), padding='same')(model)
    model=tf.keras.layers.BatchNormalization(axis=-1, momentum=0.99,epsilon=0.001)(model)
    model = tf.keras.layers.Dropout(.5)(model)

    model=Conv1D(128,kernel_size=(3), strides=1,padding = 'same')(model)
    model=Conv1D(128,kernel_size=(3), strides=1,padding = 'same')(model)
    model=MaxPooling1D(pool_size=(2),padding='same')(model)
    model=tf.keras.layers.BatchNormalization(axis=-1, momentum=0.99,epsilon=0.001)(model)
    model = tf.keras.layers.Dropout(.9)(model)


    model=Conv1D(256, kernel_size=(3), strides=1,padding = 'same')(model)
    model=Conv1D(256, kernel_size=(3), strides=1,padding = 'same')(model)
    model=MaxPooling1D(pool_size=(2), padding='same')(model)
    model=tf.keras.layers.BatchNormalization(axis=-1, momentum=0.99,epsilon=0.001)(model)
    model = tf.keras.layers.Dropout(.9)(model)

    flat=Flatten()(model)
    model=Dense(4096, activation='relu')(flat)
    model=tf.keras.layers.BatchNormalization(axis=-1, momentum=0.99,epsilon=0.001)(model)
    model = tf.keras.layers.Dropout(.9)(model)
    model=Dense(1000, activation='relu')(model)
    model=tf.keras.layers.BatchNormalization(axis=-1, momentum=0.99,epsilon=0.001)(model)
    model = tf.keras.layers.Dropout(.9)(model)
```

```
model=Conv1D(256, kernel_size=(3), strides=1,padding = 'same')(model)
model=Conv1D(256, kernel_size=(3), strides=1,padding = 'same')(model)
model=MaxPooling1D(pool_size=(2), padding='same')(model)
model=tf.keras.layers.BatchNormalization(axis=-1, momentum=0.99,epsilon=0.001)(model)
model = tf.keras.layers.Dropout(.9)(model)

flat=Flatten()(model)
model=Dense(4096, activation='relu')(flat)
model=tf.keras.layers.BatchNormalization(axis=-1, momentum=0.99,epsilon=0.001)(model)
model = tf.keras.layers.Dropout(.9)(model)
model=Dense(1000, activation='relu')(model)
model=tf.keras.layers.BatchNormalization(axis=-1, momentum=0.99,epsilon=0.001)(model)
model = tf.keras.layers.Dropout(.9)(model)
model=Dense(6, activation='softmax')(model)
main_model = Model(input_layer, model)
epochs=80
learning_rate = 0.001
decay_rate = learning_rate / epochs
momentum = 0.8
sgd = SGD(lr=learning_rate, momentum=momentum, decay=decay_rate, nesterov=False)
main_model.compile(loss='categorical_crossentropy',optimizer=sgd,metrics='accuracy')
main_model.summary()
history = main_model.fit(x = zeroTraining, y =yzeroTraining,batch_size=32, verbose = 1,validation_data=(zeroValidation, np.array(yzeroValidation)) ,epochs=epochs)
print("History = "+str(history.history))
```

Output

```
Epoch 64/80
155/155 [==============================] - 68s 436ms/step - loss: 1.2028 - accuracy: 0.5192 - val_loss: 1.5774 - val_accuracy: 0.3793
Epoch 65/80
155/155 [==============================] - 67s 430ms/step - loss: 1.1439 - accuracy: 0.5561 - val_loss: 1.6816 - val_accuracy: 0.3985
Epoch 66/80
155/155 [==============================] - 67s 435ms/step - loss: 1.1734 - accuracy: 0.5325 - val_loss: 1.6518 - val_accuracy: 0.3908
Epoch 67/80
155/155 [==============================] - 66s 429ms/step - loss: 1.1492 - accuracy: 0.5375 - val_loss: 1.7020 - val_accuracy: 0.3793
Epoch 68/80
155/155 [==============================] - 67s 435ms/step - loss: 1.1213 - accuracy: 0.5669 - val_loss: 1.7372 - val_accuracy: 0.3525
Epoch 69/80
155/155 [==============================] - 66s 428ms/step - loss: 1.1184 - accuracy: 0.5540 - val_loss: 1.7096 - val_accuracy: 0.3946
Epoch 70/80
155/155 [==============================] - 66s 428ms/step - loss: 1.0917 - accuracy: 0.5708 - val_loss: 1.6772 - val_accuracy: 0.3870
Epoch 71/80
155/155 [==============================] - 66s 428ms/step - loss: 1.0745 - accuracy: 0.5846 - val_loss: 1.7750 - val_accuracy: 0.3793
Epoch 72/80
155/155 [==============================] - 66s 428ms/step - loss: 1.0577 - accuracy: 0.5837 - val_loss: 1.7730 - val_accuracy: 0.3831
Epoch 73/80
155/155 [==============================] - 66s 428ms/step - loss: 1.0540 - accuracy: 0.5927 - val_loss: 1.7919 - val_accuracy: 0.3985
Epoch 74/80
155/155 [==============================] - 67s 429ms/step - loss: 1.0330 - accuracy: 0.6026 - val_loss: 1.8344 - val_accuracy: 0.3487
Epoch 75/80
155/155 [==============================] - 67s 429ms/step - loss: 1.0027 - accuracy: 0.6136 - val_loss: 1.8468 - val_accuracy: 0.3946
Epoch 76/80
155/155 [==============================] - 66s 429ms/step - loss: 1.0059 - accuracy: 0.6208 - val_loss: 1.9560 - val_accuracy: 0.3755
Epoch 77/80
155/155 [==============================] - 66s 429ms/step - loss: 0.9892 - accuracy: 0.6230 - val_loss: 1.8479 - val_accuracy: 0.3755
Epoch 78/80
155/155 [==============================] - 67s 430ms/step - loss: 0.9518 - accuracy: 0.6417 - val_loss: 1.9702 - val_accuracy: 0.3678
Epoch 79/80
155/155 [==============================] - 67s 431ms/step - loss: 0.9500 - accuracy: 0.6357 - val_loss: 1.9512 - val_accuracy: 0.4023
Epoch 80/80
155/155 [==============================] - 66s 429ms/step - loss: 0.9908 - accuracy: 0.6180 - val_loss: 1.9484 - val_accuracy: 0.4061
History = {'loss': [1.7901586294174194, 1.786758303642273, 1.7839961051940918, 1.7810691595077515, 1.7772750854492188, 1.771883845329284
```

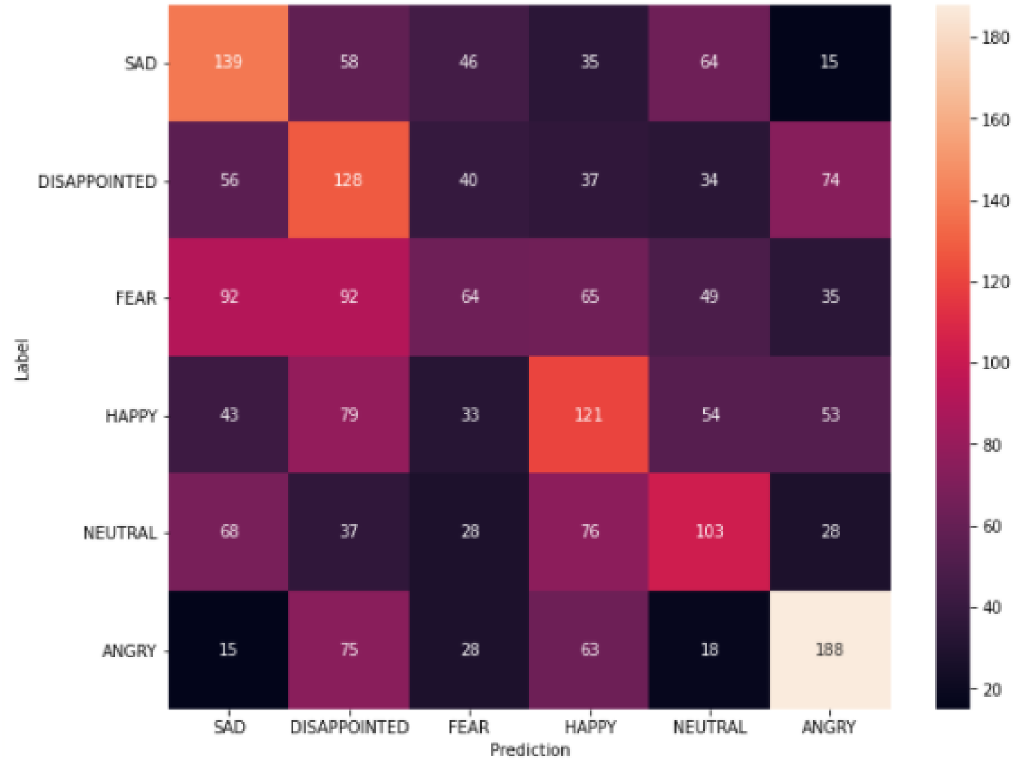
Plot of validation and training loss for zero crossing

```python
metrics = history.history
plt.plot(history.epoch, metrics['loss'], metrics['val_loss'])
plt.legend(['loss', 'val_loss'])
plt.show()
```

Test set accuracy of zero crossing

```
Test set accuracy of Zero Crossing: 33%
```

Confusion Matrix of Zero-crossing



F-score of Zero-crossing

```
array([0.36103896, 0.30548926, 0.20125786, 0.31025641, 0.31117825,
       0.48205128])
```

2-Energy

- Energy is the sum of squares of the signal values, normalized by the respective frame length.
- The energy is brought using the function librosa.feature.rms.
- Splitting and reshaping Energy dataset. The data is splitted 70 **%** for the training set and 30% for the test set which is done using train_test_split function.This function is used again for the training and validation to have a 5**%** validation.
- 1D Model for Zero-crossing.The following layers are used Convolution,Maxpooling,Dropout,Flatten and Dense.
- Next the validation loss and training loss are plotted for Energy.
- Accuracy of test set of Energy is calculated.

- Confusion matrix of Energy is calculated and plotted.
- F-score of Energy is calculated using the built-in function f1_score which takes in as input the true and predicted labels.

Screenshots:

Splitting of Energy

```
print(energyTraining.shape)
print(yenergyTraining.shape)
print(energyValidation.shape)
print(yenergyValidation.shape)
```

```
(4948, 216, 1)
(4948, 6)
(261, 216, 1)
(261, 6)
```

Screenshots of Energy 1D Model

1D Model for Energy

```
input_layer=Input(shape=(energyTraining.shape[1],1))
model=Conv1D(216, kernel_size=(5), strides=1, padding = 'same')(input_layer)
model=Conv1D(216,kernel_size=(5), strides=1,padding = 'same')(model)
model=Conv1D(216,kernel_size=(5), strides=1,padding = 'same')(model)
model=MaxPooling1D(pool_size=(3), padding='same')(model)
model = tf.keras.layers.Dropout(.4)(model)
model=Conv1D(108,kernel_size=(5),strides=1,padding = 'same')(model)
model=Conv1D(108,kernel_size=(5),strides=1,padding = 'same')(model)
model=Conv1D(108,kernel_size=(5),strides=1,padding = 'same')(model)
model=MaxPooling1D(pool_size=(3), padding='same')(model)
model = tf.keras.layers.Dropout(.5)(model)
model=Conv1D(54, kernel_size=(5),strides=1,padding = 'same')(model)
model=Conv1D(54, kernel_size=(5),strides=1,padding = 'same')(model)
model=Conv1D(54, kernel_size=(5),strides=1,padding = 'same')(model)
model=MaxPooling1D(pool_size=(3), padding='same')(model)
model = tf.keras.layers.Dropout(.4)(model)
model=Conv1D(27, kernel_size=(5),strides=1,padding = 'same')(model)
model=Conv1D(27, kernel_size=(5),strides=1,padding = 'same')(model)
model=Conv1D(27, kernel_size=(5),strides=1,padding = 'same')(model)
model=MaxPooling1D(pool_size=(3), padding='same')(model)
model = tf.keras.layers.Dropout(.3)(model)
model=Conv1D(13, kernel_size=(5),strides=1,padding = 'same')(model)
model=Conv1D(13, kernel_size=(5),strides=1,padding = 'same')(model)
model=Conv1D(13, kernel_size=(5),strides=1,padding = 'same')(model)
model=MaxPooling1D(pool_size=(3), padding='same')(model)
model = tf.keras.layers.Dropout(.2)(model)
```

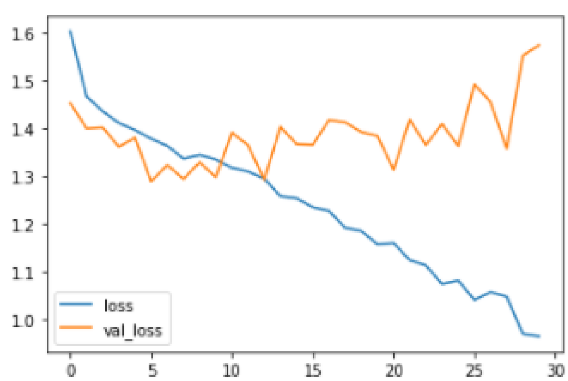
```python
flat=Flatten()(model)
model=Dense(10000, activation='relu')(flat)
model=Dense(6, activation='softmax')(model)
main_model = Model(input_layer, model)
opt = tf.keras.optimizers.Adam(learning_rate=0.001)
main_model.compile(loss='categorical_crossentropy', optimizer=opt,metrics='accuracy')
main_model.summary()
history = main_model.fit(x = energyTraining, y =yenergyTraining,batch_size=50,verbose = 1,validation_data=(energyValidation, np.array(yenergyValidation)),epochs=30)
print("History = "+str(history.history))
```

Output

```
99/99 [==============================] - 55s 555ms/step - loss: 1.2292 - accuracy: 0.5028 - val_loss: 1.4051 - val_accuracy: 0.4215
Epoch 15/30
99/99 [==============================] - 55s 555ms/step - loss: 1.2420 - accuracy: 0.5093 - val_loss: 1.3670 - val_accuracy: 0.4215
Epoch 16/30
99/99 [==============================] - 55s 554ms/step - loss: 1.2238 - accuracy: 0.5061 - val_loss: 1.3656 - val_accuracy: 0.4751
Epoch 17/30
99/99 [==============================] - 55s 553ms/step - loss: 1.2080 - accuracy: 0.5067 - val_loss: 1.4172 - val_accuracy: 0.4138
Epoch 18/30
99/99 [==============================] - 55s 557ms/step - loss: 1.1740 - accuracy: 0.5253 - val_loss: 1.4124 - val_accuracy: 0.4138
Epoch 19/30
99/99 [==============================] - 55s 556ms/step - loss: 1.1428 - accuracy: 0.5420 - val_loss: 1.3921 - val_accuracy: 0.4789
Epoch 20/30
99/99 [==============================] - 55s 557ms/step - loss: 1.1444 - accuracy: 0.5386 - val_loss: 1.3841 - val_accuracy: 0.4674
Epoch 21/30
99/99 [==============================] - 55s 554ms/step - loss: 1.1032 - accuracy: 0.5557 - val_loss: 1.3138 - val_accuracy: 0.4789
Epoch 22/30
99/99 [==============================] - 54s 550ms/step - loss: 1.0833 - accuracy: 0.5690 - val_loss: 1.4181 - val_accuracy: 0.4215
Epoch 23/30
99/99 [==============================] - 54s 550ms/step - loss: 1.1025 - accuracy: 0.5565 - val_loss: 1.3649 - val_accuracy: 0.4751
Epoch 24/30
99/99 [==============================] - 55s 553ms/step - loss: 1.0246 - accuracy: 0.6009 - val_loss: 1.4090 - val_accuracy: 0.4751
Epoch 25/30
99/99 [==============================] - 55s 555ms/step - loss: 1.0174 - accuracy: 0.6052 - val_loss: 1.3629 - val_accuracy: 0.4713
Epoch 26/30
99/99 [==============================] - 55s 556ms/step - loss: 1.0017 - accuracy: 0.6065 - val_loss: 1.4918 - val_accuracy: 0.4636
Epoch 27/30
99/99 [==============================] - 55s 554ms/step - loss: 1.0168 - accuracy: 0.6006 - val_loss: 1.4547 - val_accuracy: 0.4521
Epoch 28/30
99/99 [==============================] - 55s 555ms/step - loss: 1.0038 - accuracy: 0.6015 - val_loss: 1.3574 - val_accuracy: 0.4559
Epoch 29/30
99/99 [==============================] - 55s 555ms/step - loss: 0.9493 - accuracy: 0.6331 - val_loss: 1.5515 - val_accuracy: 0.4061
Epoch 30/30
99/99 [==============================] - 55s 556ms/step - loss: 0.9287 - accuracy: 0.6240 - val_loss: 1.5730 - val_accuracy: 0.4061
History = {'loss': [1.6018946170806885, 1.466667890548706, 1.435476541519165, 1.4109262228012085, 1.3962897062301636, 1.378288269042968
```

Plot of validation and training loss for zero crossing

```python
metrics = history.history
plt.plot(history.epoch, metrics['loss'], metrics['val_loss'])
plt.legend(['loss', 'val_loss'])
plt.show()
```



Test set accuracy of Energy

```
Test set accuracy of Energy: 43%
```

Confusion Matrix of Energy

F-score of Energy

```
array([0.5011655 , 0.29765886, 0.30107527, 0.36437247, 0.46293888,
       0.6005291 ])
```

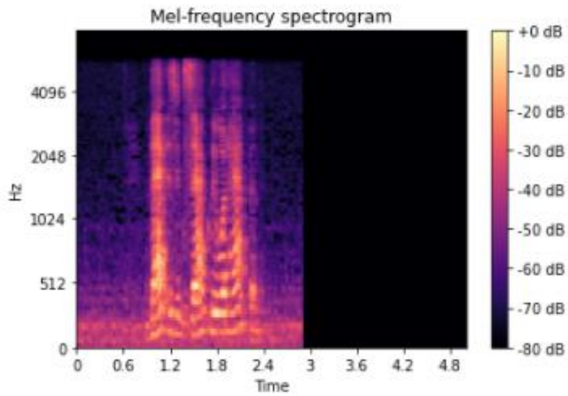
3. Creating Feature Space for 2D

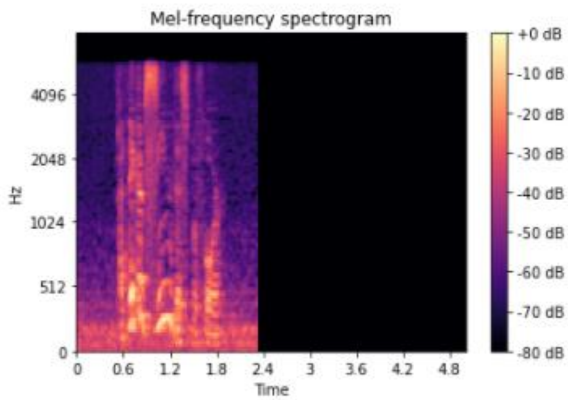
Mel Spectogram

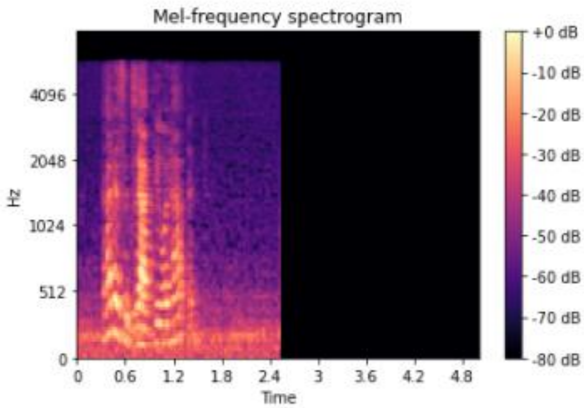
- A spectrogram is a detailed view of audio, able to represent time, frequency, and amplitude all on one graph. A spectrogram can visually reveal broadband, electrical, or intermittent noise in audio, and can allow you to easily isolate those audio problems by sight. Because of its profound level of detail, a spectrogram is particularly useful in post production.
- The mel spectrogram is brought using the function
  mel(y=sound[i], sr=freq[i]).ssss
- This is done for 2D matrix and the padding with zeros here done for both rows and colomns.
- Extraction of Spectogram is then done.
- Splitting and reshaping zero-crossing dataset. The data is splitted 70 **%** for the training set and 30% for the test set which is done using train_test_split function.This function is used again for the training and validation to have a 5**%** validation.
- 2D Model of Spectogramis implemented which contains the following Convolution,Maxpooling,Dropout,Flatten and Dense.
- Next the validation loss and training loss are plotted for Mel-spectogram.
- Accuracy of test set of Mel-spectogram is calculated.
- Confusion matrix of Mel-spectogram is calculated and plotted.
- F-score of Mel-spectogram is calculated using the built-in function f1_score which takes in as input the true and predicted labels.

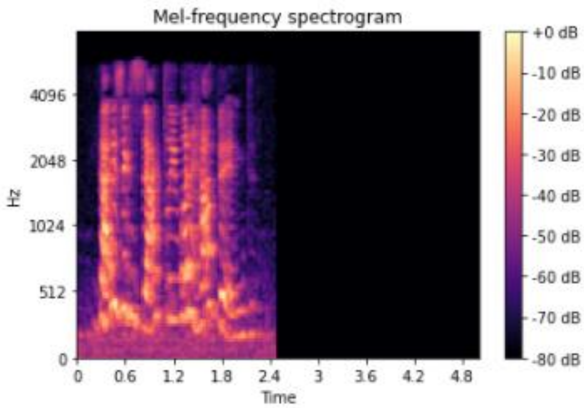
Screenshots:

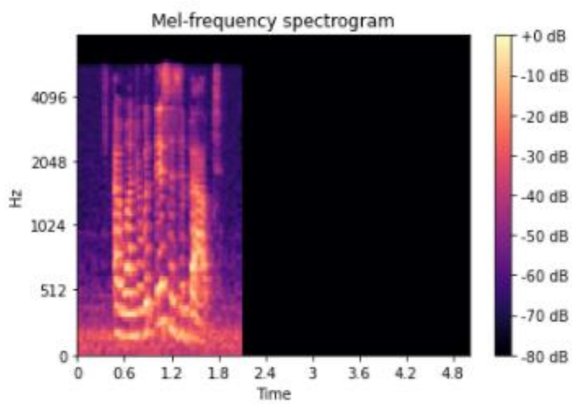
Screenshots of 8 mel Spectograms

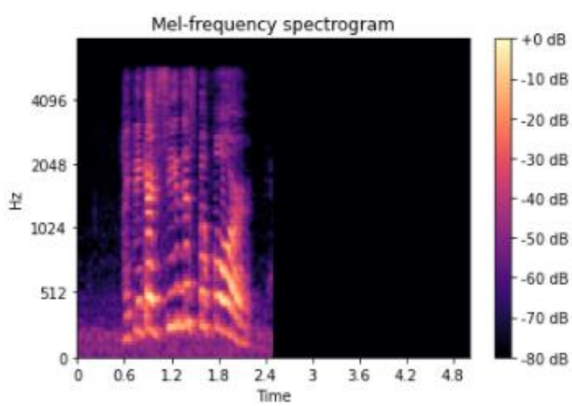


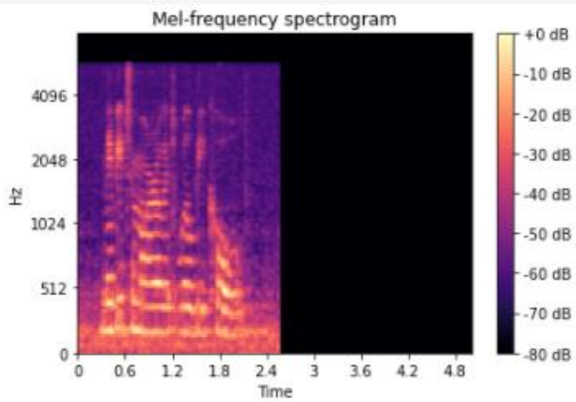


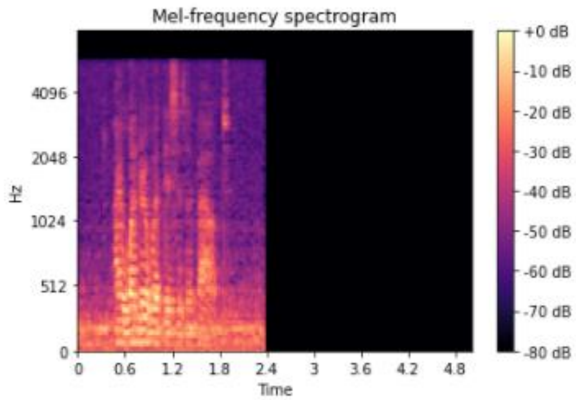

Mel-frequency spectrogram
Hz
4096
2048
1024
512
0
Time
0 0.6 1.2 1.8 2.4 3 3.6 4.2 4.8
+0 dB
-10 dB
-20 dB
-30 dB
-40 dB
-50 dB
-60 dB
-70 dB
-80 dB


Mel-frequency spectrogram
Hz
4096
2048
1024
512
0
Time
0 0.6 1.2 1.8 2.4 3 3.6 4.2 4.8
+0 dB
-10 dB
-20 dB
-30 dB
-40 dB
-50 dB
-60 dB
-70 dB
-80 dB


Mel-frequency spectrogram
Hz
4096
2048
1024
512
0
Time
0 0.6 1.2 1.8 2.4 3 3.6 4.2 4.8
+0 dB
-10 dB
-20 dB
-30 dB
-40 dB
-50 dB
-60 dB
-70 dB
-80 dB


Mel-frequency spectrogram
Hz
4096
2048
1024
512
0
Time
0 0.6 1.2 1.8 2.4 3 3.6 4.2 4.8
+0 dB
-10 dB
-20 dB
-30 dB
-40 dB
-50 dB
-60 dB
-70 dB
-80 dB





Splitting of Mel Spectogram

```
print(np.array(melTraining).shape)
print(np.array(melValidation).shape)
print(np.array(ymelTraining).shape)
print(np.array(ymelValidation).shape)
```

```
(4948, 128, 216)
(261, 128, 216)
(4948, 6)
(261, 6)
```

Screenshots of Mel spectrogram 2D Model

Models of 2D

```
#Reshaping
MelTrainArray = np.array(melTraining)
MelTrainArray=MelTrainArray.reshape((MelTrainArray.shape[0], MelTrainArray.shape[1], MelTrainArray.shape[2], 1))
MelValArray = np.array(melValidation)
melValArray = MelValArray.reshape(MelValArray.shape[0],MelValArray.shape[1], MelValArray.shape[2], 1)

print(melValArray.shape)
input_layer=Input(shape=(MelValArray.shape[1],MelValArray.shape[2],1))

model=Conv2D(8, kernel_size=(3,3),strides=1, padding = 'same')(input_layer)
model=Conv2D(8, kernel_size=(3,3),strides=1, padding = 'same')(model)

model=MaxPooling2D(pool_size=(2, 2), strides=(2,2),padding = 'valid')(model)

model=Conv2D(16, kernel_size=(3,3), strides=1,padding = 'valid')(model)
model=Conv2D(16, kernel_size=(3,3), strides=1,padding = 'valid')(model)

model=MaxPooling2D(pool_size=(2, 2), strides=(2,2),padding = 'valid')(model)

model=Conv2D(32, kernel_size=(3,3), strides=1,padding = 'valid')(model)
model=Conv2D(32, kernel_size=(3,3), strides=1,padding = 'valid')(model)

model=MaxPooling2D(pool_size=(2, 2), strides=(2,2),padding = 'valid')(model)

model=Conv2D(64, kernel_size=(3,3), strides=1,padding = 'valid')(model)
model=Conv2D(64, kernel_size=(3,3), strides=1,padding = 'valid')(model)
```

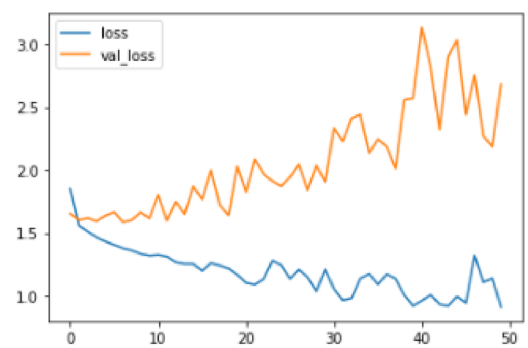
```
model=MaxPooling2D(pool_size=(2, 2), strides=(2,2))(model)

model=Flatten()(model)
model=Dense(120, activation='relu')(model)
model=Dense(6, activation='softmax')(model)
# opt = tf.keras.optimizers.Adam(learning_rate=0.001)
opt = tf.keras.optimizers.SGD(learning_rate=0.001)
main_model = Model(input_layer, model)
main_model.compile(loss='categorical_crossentropy', optimizer=opt,metrics='accuracy')
main_model.summary()
history = main_model.fit(x = MelTrainArray, y = np.array(ymelTraining),batch_size=8,verbose = 1,validation_data=(MelValArray, np.array(ymelValidation)),epochs=50)
print("History = "+str(history.history))
```

Output

```
619/619 [==============================] - 160s 258ms/step - loss: 1.0304 - accuracy: 0.6157 - val_loss: 2.1860 - val_accuracy: 0.4291
Epoch 38/50
619/619 [==============================] - 159s 258ms/step - loss: 1.1314 - accuracy: 0.5853 - val_loss: 2.0127 - val_accuracy: 0.4521
Epoch 39/50
619/619 [==============================] - 159s 257ms/step - loss: 1.0239 - accuracy: 0.6066 - val_loss: 2.5562 - val_accuracy: 0.4023
Epoch 40/50
619/619 [==============================] - 160s 258ms/step - loss: 0.9299 - accuracy: 0.6470 - val_loss: 2.5688 - val_accuracy: 0.4444
Epoch 41/50
619/619 [==============================] - 160s 258ms/step - loss: 0.8770 - accuracy: 0.6560 - val_loss: 3.1316 - val_accuracy: 0.3870
Epoch 42/50
619/619 [==============================] - 160s 258ms/step - loss: 0.9924 - accuracy: 0.6263 - val_loss: 2.8174 - val_accuracy: 0.3946
Epoch 43/50
619/619 [==============================] - 159s 257ms/step - loss: 0.9244 - accuracy: 0.6480 - val_loss: 2.3199 - val_accuracy: 0.4215
Epoch 44/50
619/619 [==============================] - 159s 257ms/step - loss: 0.9145 - accuracy: 0.6495 - val_loss: 2.9012 - val_accuracy: 0.4368
Epoch 45/50
619/619 [==============================] - 160s 258ms/step - loss: 0.9801 - accuracy: 0.6435 - val_loss: 3.0311 - val_accuracy: 0.3831
Epoch 46/50
619/619 [==============================] - 159s 257ms/step - loss: 0.8868 - accuracy: 0.6715 - val_loss: 2.4406 - val_accuracy: 0.3908
Epoch 47/50
619/619 [==============================] - 159s 257ms/step - loss: 1.1389 - accuracy: 0.6233 - val_loss: 2.7544 - val_accuracy: 0.3448
Epoch 48/50
619/619 [==============================] - 160s 258ms/step - loss: 1.1080 - accuracy: 0.5812 - val_loss: 2.2703 - val_accuracy: 0.3640
Epoch 49/50
619/619 [==============================] - 159s 258ms/step - loss: 1.0687 - accuracy: 0.6133 - val_loss: 2.1857 - val_accuracy: 0.4023
Epoch 50/50
619/619 [==============================] - 160s 258ms/step - loss: 0.9199 - accuracy: 0.6453 - val_loss: 2.6798 - val_accuracy: 0.3755
History = {'loss': [1.851027488708496, 1.5579965114593506, 1.510033369064331, 1.464797019958496, 1.4318299293518066, 1.4053725004196167, 1.
```

Plot of validation and training loss for Mel Spectogram

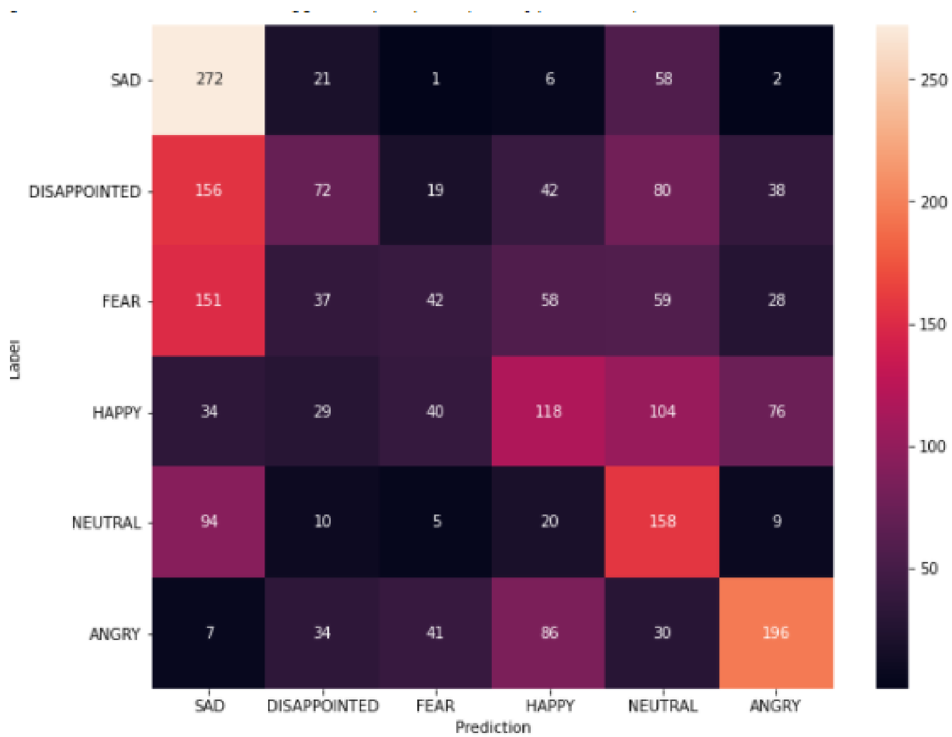
```python
metrics = history.history
plt.plot(history.epoch, metrics['loss'], metrics['val_loss'])
plt.legend(['loss', 'val_loss'])
plt.show()
```



Test set accuracy of Mel Spectogram

Test set accuracy of Spectogram: 38%

Confusion Matrix of Mel Spectogram



F-score of Mel Spectogram

```
array([0.50651769, 0.23606557, 0.16061185, 0.32284542, 0.40254777,
       0.52759085])
```