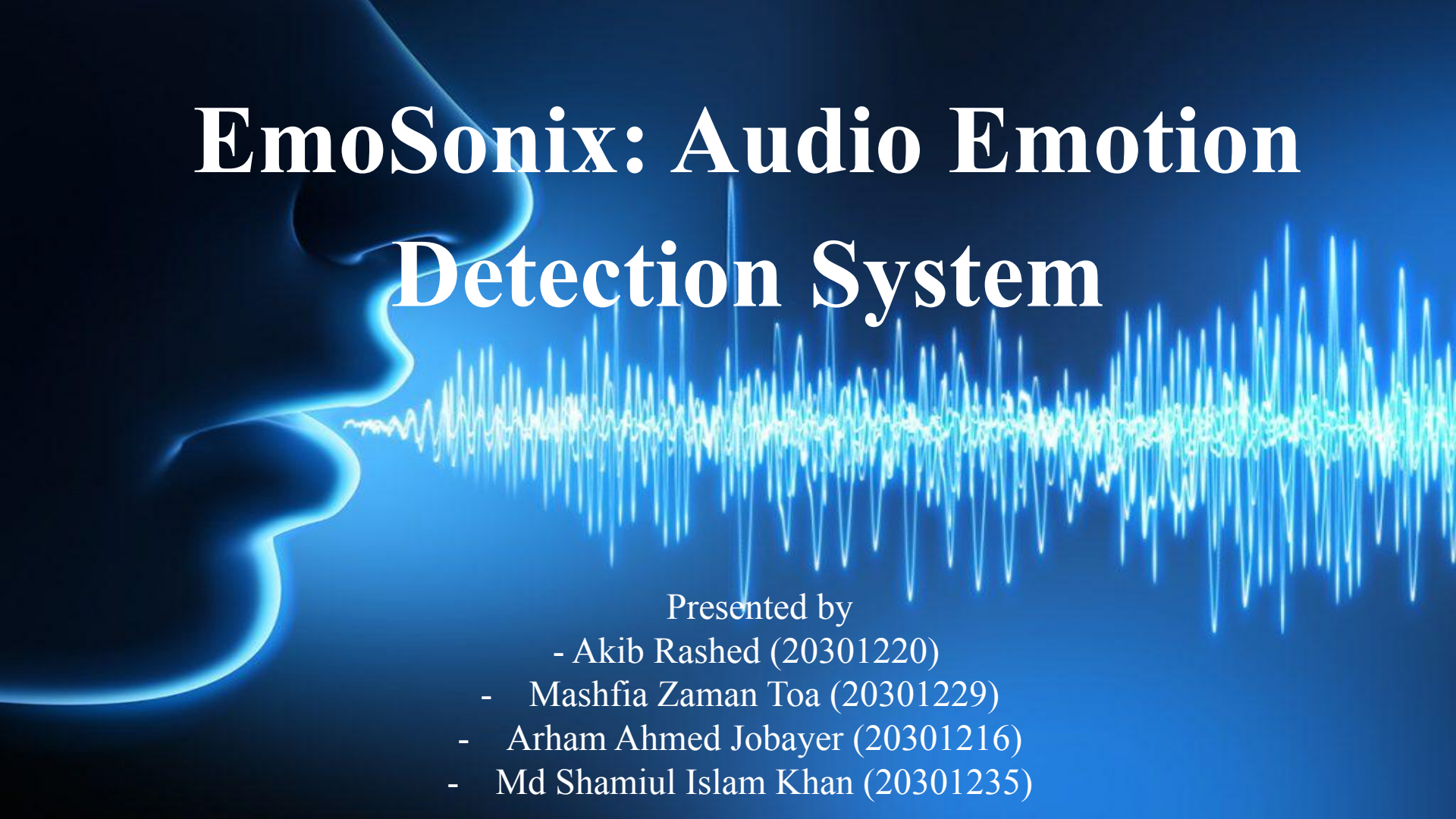
EmoSonix: Audio Emotion Detection System

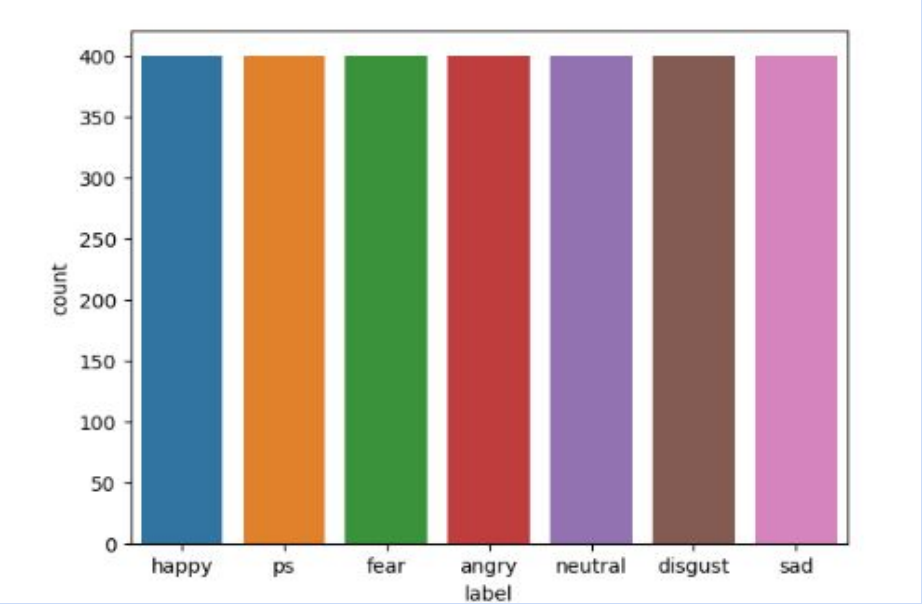


Presented by

- Akib Rashed (20301220)
- Mashfia Zaman Toa (20301229)
- Arham Ahmed Jobayer (20301216)
- Md Shamiul Islam Khan (20301235)

Previously Done work on Dataset

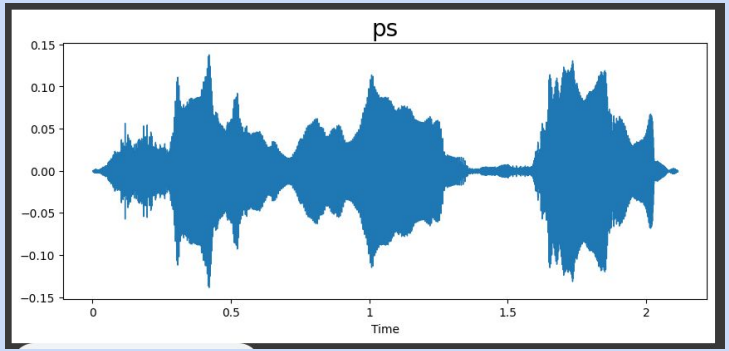
- Loaded the data and separated path and label
- Balanced dataset with 400 data point in each Category

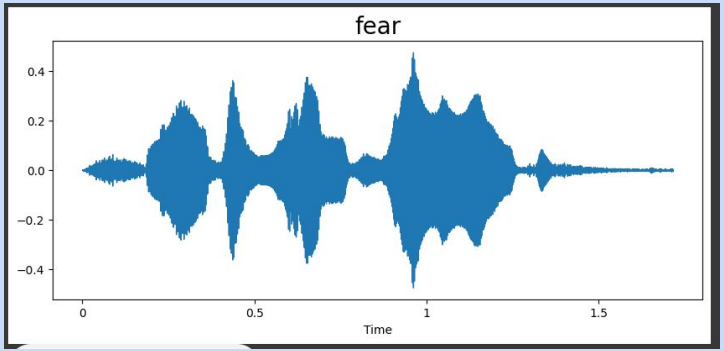


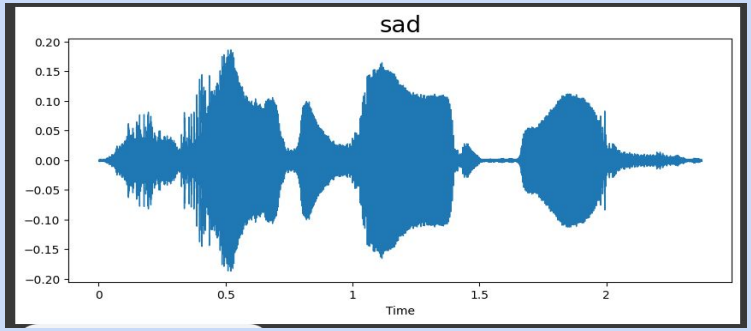
Data Wrangling

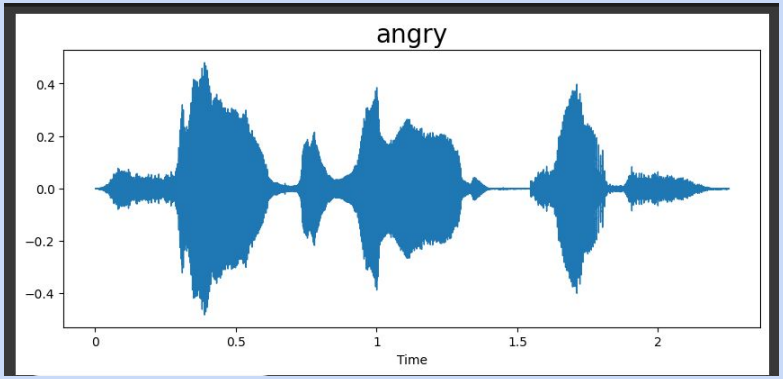
- Waveplot
  - Visualization
  - Comparison
  - Transient Detection
  - Noise and Distortion Analysis

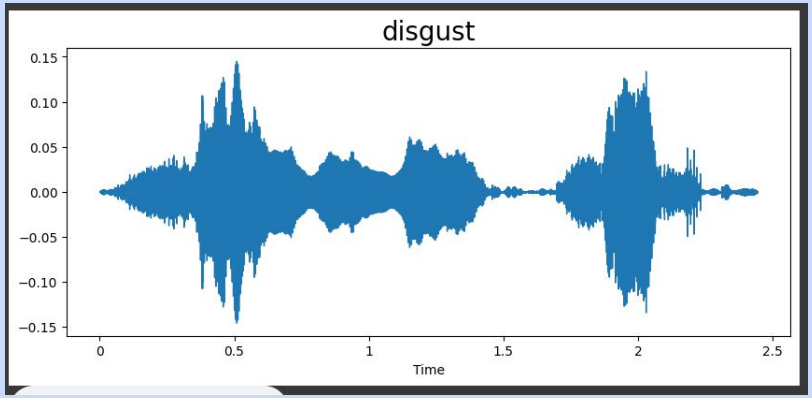
```
#waveform of data ar jonno
def waveplot(data, sr, emotion):
    plt.figure(figsize=(10,4))
    plt.title(emotion, size=20)
    librosa.display.waveshow(data, sr=sr)
    plt.show()
```

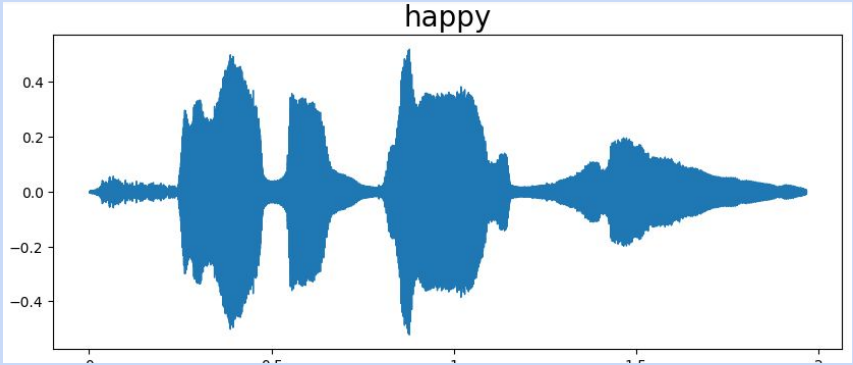





sad
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
0
0.5
1
1.5
2
Time


angry
0.4
0.2
0.0
−0.2
−0.4
0
0.5
1
1.5
2
Time

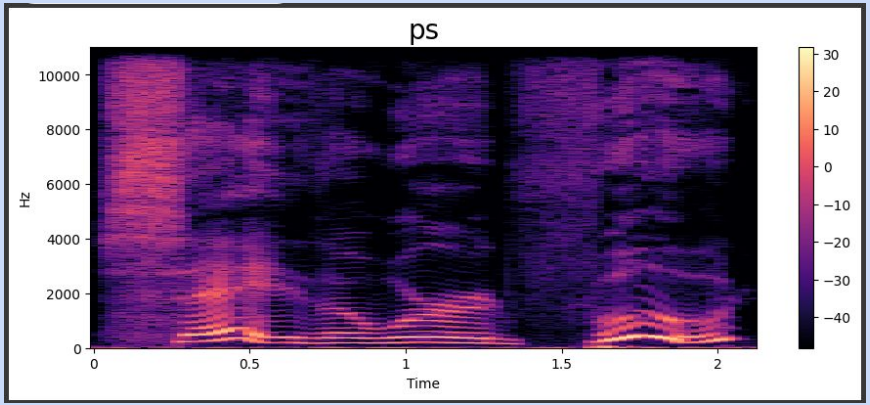

disgust
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
0
0.5
1
1.5
2
2.5
Time

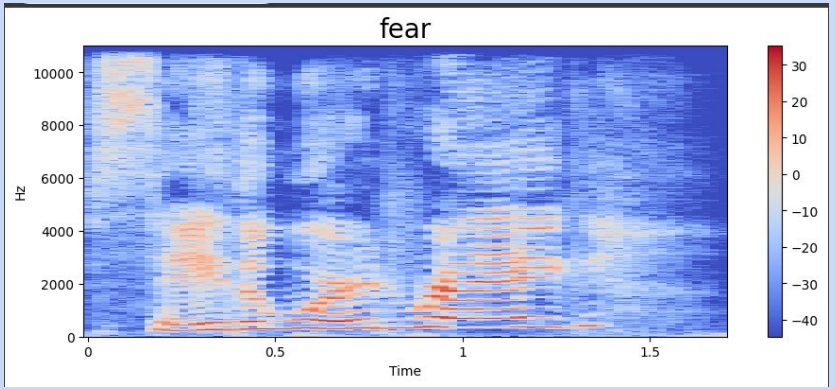

happy
0.4
0.2
0.0
−0.2
−0.4

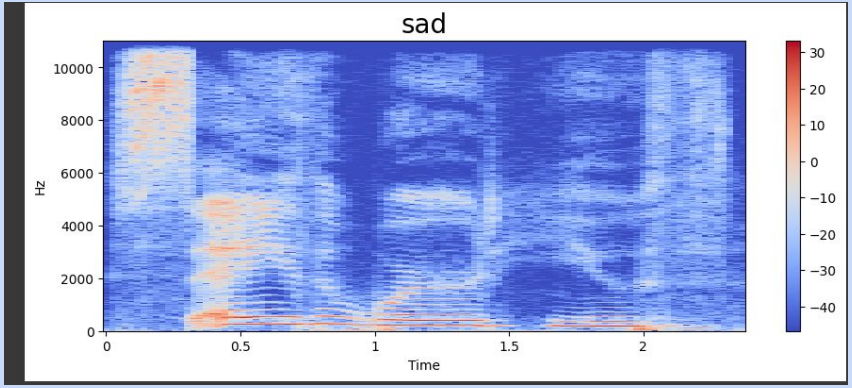
Spectrogram

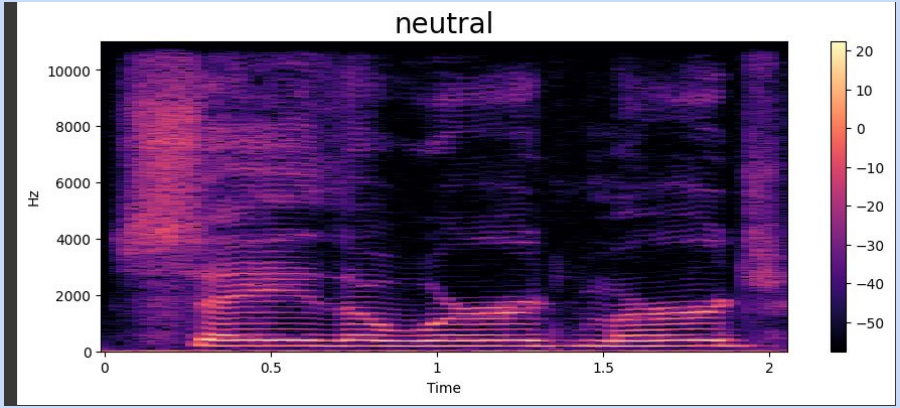
- Frequency Information
- Visualization of Features
- Better Discrimination
- Pattern Recognition
- Feature Extraction

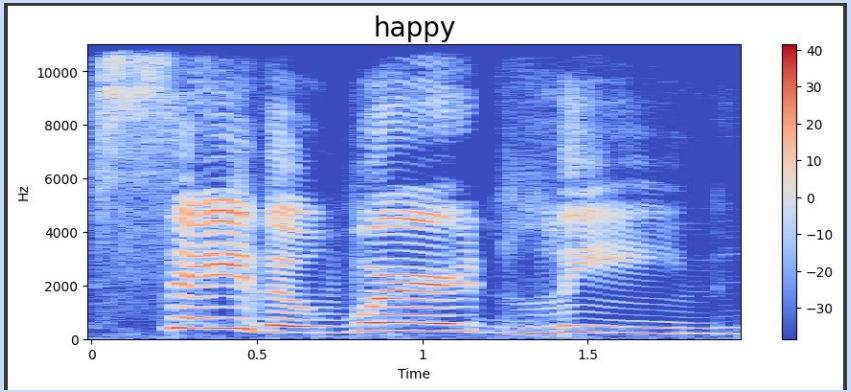
```
def spectogram(data, sr, emotion):
    x = librosa.stft(data)
    xdb = librosa.amplitude_to_db(abs(x))
    plt.figure(figsize=(11,4))
    plt.title(emotion, size=20)
    librosa.display.specshow(xdb, sr=sr, x_axis='time', y_axis='hz')
    plt.colorbar()
```

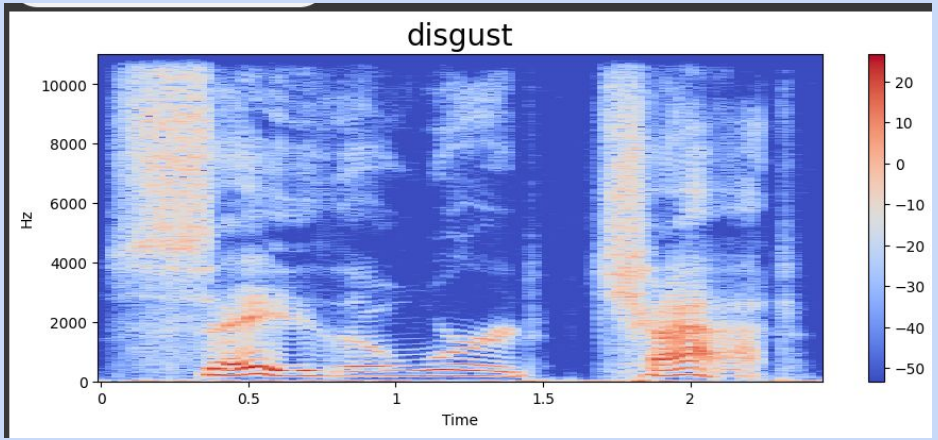




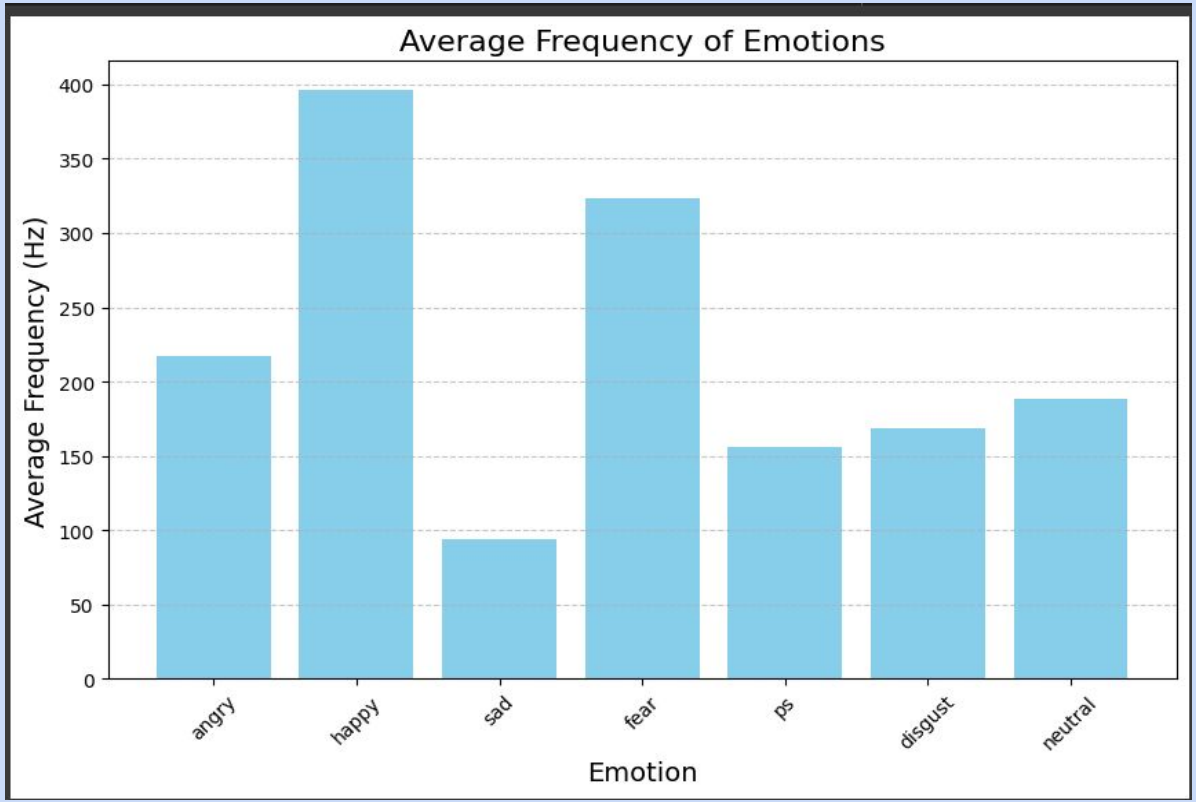

sad
Hz
Time
10000
8000
6000
4000
2000
0
0
0.5
1
1.5
2
30
20
10
0
−10
−20
−30
−40


neutral
Hz
Time
10000
8000
6000
4000
2000
0
0
0.5
1
1.5
2
20
10
0
−10
−20
−30
−40
−50


happy
Hz
Time
10000
8000
6000
4000
2000
0
0
0.5
1
1.5
40
30
20
10
0
−10
−20
−30


disgust
Hz
Time
10000
8000
6000
4000
2000
0
0
0.5
1
1.5
2
20
10
0
−10
−20
−30
−40
−50

Frequency Distribution based on Spectrogram



Feature Extraction

- To convert audio file into numeric format
- Mel-Frequency Cepstral Coefficients (MFCC)

```
array([-3.34945923e+02,  3.70755119e+01, -7.90456820e+00, -1.83790684e+00,
       -1.28784685e+01,  6.48743868e+00,  2.52078199e+00, -1.22021313e+01,
       -6.53204012e+00,  1.64398634e+00, -1.77280693e+01,  9.59110737e+00,
       -9.33374500e+00,  7.03533840e+00, -2.34031916e+00, -5.02835369e+00,
       -1.51206696e+00,  4.88031101e+00, -2.85840440e+00,  2.46732163e+00,
       -2.44669652e+00, -1.70657969e+00, -7.33406544e+00, -1.87550449e+00,
       -2.89007664e+00,  2.28449538e-01,  1.87676847e+00,  7.56279325e+00,
        1.12165041e+01,  1.49963121e+01,  1.19341040e+01,  1.10935335e+01,
        4.09022570e+00,  3.95253038e+00,  9.11194146e-01,  4.24853182e+00,
        3.31549644e+00,  4.12816429e+00, -2.26341224e+00, -1.07533288e+00,
       -1.64336324e+00,  3.98158669e+00, -1.11243165e+00, -8.54802370e-01,
       -4.10659170e+00, -8.08297634e-01, -2.51135379e-01,  2.69858575e+00,
       -1.66069591e+00, -1.15179658e+00,  1.13143539e+00,  1.59848392e+00,
       -1.75673831e+00, -6.32186159e-02, -1.91152751e+00,  1.58386374e+00,
        6.33633137e-01,  1.27641773e+00,  7.73862720e-01,  3.91387486e+00],
      dtype=float32)
```

Normalizing

- Improved Convergence
- Better Performance
- Prevention of Numerical Instabilities

```
] X_mfcc = df['speech'].apply(lambda x: extract_mfcc(x))
```

```
] X_mfcc
```

```
0       [-334.94592, 37.075512, -7.904568, -1.8379068,...
1       [-350.88992, 74.09218, -2.3219209, 7.8491626, ...
2       [-322.57584, 47.85948, -25.91174, 4.019515, -2...
3       [-289.10486, 78.74502, 2.2043347, -5.4672093, ...
4       [-325.0482, 52.04364, -21.359823, 3.8501308, -...
                              ...
2795    [-331.03552, 48.584187, -25.937048, -4.8479657...
2796    [-340.9328, 93.7364, -26.861814, 0.34102386, -...
2797    [-354.5709, 64.17273, -31.288214, 14.938522, -...
2798    [-305.5626, 70.34936, -42.098427, -7.0920105, ...
2799    [-389.8703, 94.92119, -6.0177207, -5.5790243, ...
Name: speech, Length: 2800, dtype: object
```

Encoding

- One Hot Encoded into an array

```
#output column
from sklearn.preprocessing import OneHotEncoder
enc = OneHotEncoder()
y = enc.fit_transform(df[['label']])
```

```
y = y.toarray()
```

```
y
```

```
array([[1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 1., 0.],
       [0., 0., 0., ..., 0., 1., 0.],
       [0., 0., 0., ..., 0., 1., 0.]])
```

LSTM on the encoded dataset

- Activation
  - Relu
  - Softmx
- Optimizer
  - Adam
- 50 epoch

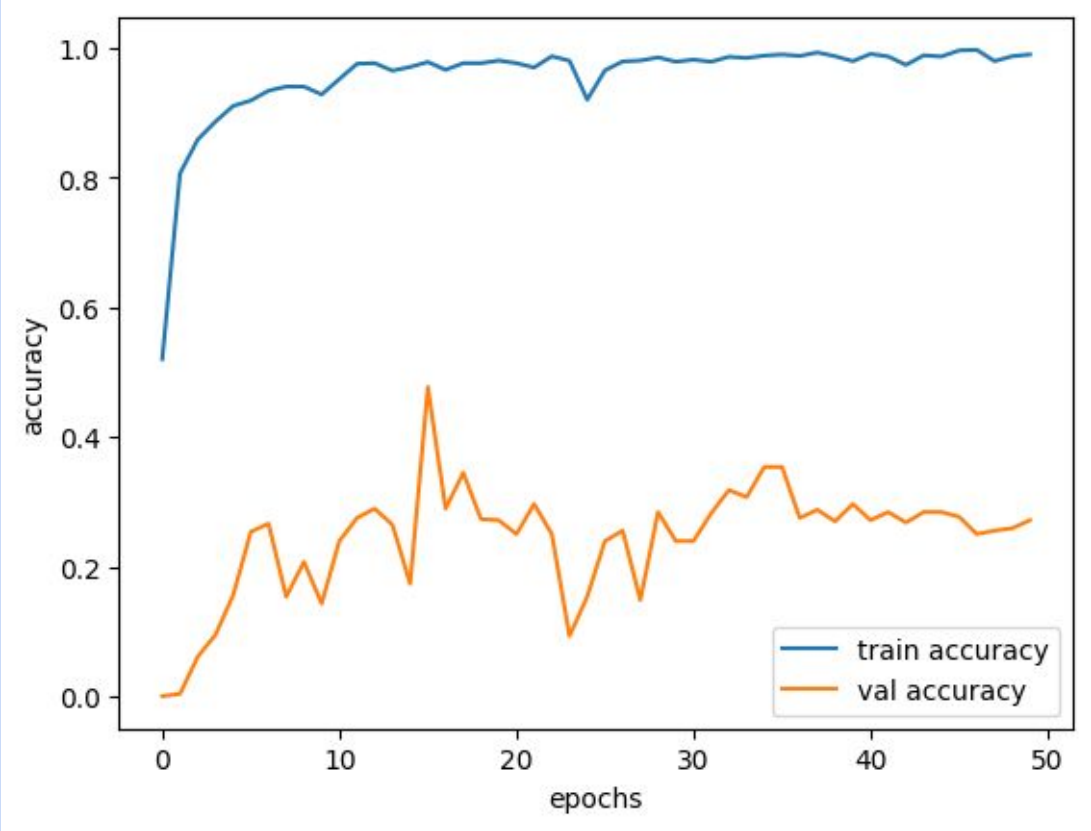
```
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout

model = Sequential([
    LSTM(256, return_sequences=False, input_shape=(60,1)),
    Dropout(0.2),
    Dense(128, activation='relu'),
    Dropout(0.2),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(7, activation='softmax')
])

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 256)               264192

 dropout (Dropout)           (None, 256)               0

 dense (Dense)               (None, 128)               32896

 dropout_1 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 64)                8256

 dropout_2 (Dropout)         (None, 64)                0

 dense_2 (Dense)             (None, 7)                 455
```

```
# Train the model
history = model.fit(X, y, validation_split=0.2, epochs=50, batch_size=64)
```

```
Epoch 22/50
35/35 [==============================] - 0s 9ms/step - loss: 0.1012 - accuracy: 0.9696 - val_loss: 5.2422 - val_accuracy: 0.2964
Epoch 23/50
35/35 [==============================] - 0s 9ms/step - loss: 0.0507 - accuracy: 0.9871 - val_loss: 5.5924 - val_accuracy: 0.2500
Epoch 24/50
35/35 [==============================] - 0s 9ms/step - loss: 0.0712 - accuracy: 0.9799 - val_loss: 6.6559 - val_accuracy: 0.0929
Epoch 25/50
35/35 [==============================] - 0s 9ms/step - loss: 0.2671 - accuracy: 0.9201 - val_loss: 5.9003 - val_accuracy: 0.1536
Epoch 26/50
35/35 [==============================] - 0s 9ms/step - loss: 0.1314 - accuracy: 0.9652 - val_loss: 5.3231 - val_accuracy: 0.2393
Epoch 27/50
35/35 [==============================] - 0s 9ms/step - loss: 0.0844 - accuracy: 0.9790 - val_loss: 5.2956 - val_accuracy: 0.2554
Epoch 28/50
35/35 [==============================] - 0s 9ms/step - loss: 0.0587 - accuracy: 0.9804 - val_loss: 6.8499 - val_accuracy: 0.1482
Epoch 29/50
35/35 [==============================] - 0s 9ms/step - loss: 0.0484 - accuracy: 0.9853 - val_loss: 6.1121 - val_accuracy: 0.2839
Epoch 30/50
35/35 [==============================] - 0s 9ms/step - loss: 0.0663 - accuracy: 0.9786 - val_loss: 5.8007 - val_accuracy: 0.2393
Epoch 31/50
35/35 [==============================] - 0s 9ms/step - loss: 0.0671 - accuracy: 0.9817 - val_loss: 6.2283 - val_accuracy: 0.2393
Epoch 32/50
35/35 [==============================] - 0s 9ms/step - loss: 0.0715 - accuracy: 0.9786 - val_loss: 6.1375 - val_accuracy: 0.2821
Epoch 33/50
35/35 [==============================] - 0s 9ms/step - loss: 0.0539 - accuracy: 0.9862 - val_loss: 5.0351 - val_accuracy: 0.3179
Epoch 34/50
35/35 [==============================] - 0s 9ms/step - loss: 0.0498 - accuracy: 0.9844 - val_loss: 5.6238 - val_accuracy: 0.3071
Epoch 35/50
35/35 [==============================] - 0s 9ms/step - loss: 0.0433 - accuracy: 0.9879 - val_loss: 5.1346 - val_accuracy: 0.3536
Epoch 36/50
35/35 [==============================] - 0s 10ms/step - loss: 0.0329 - accuracy: 0.9897 - val_loss: 5.1427 - val_accuracy: 0.3536
Epoch 37/50
```

Results


1.0
0.8
0.6
0.4
0.2
0.0
accuracy
0
10
20
30
40
50
epochs
train accuracy
val accuracy


THANK YOU!