SPEECH EMOTION RECOGNITION

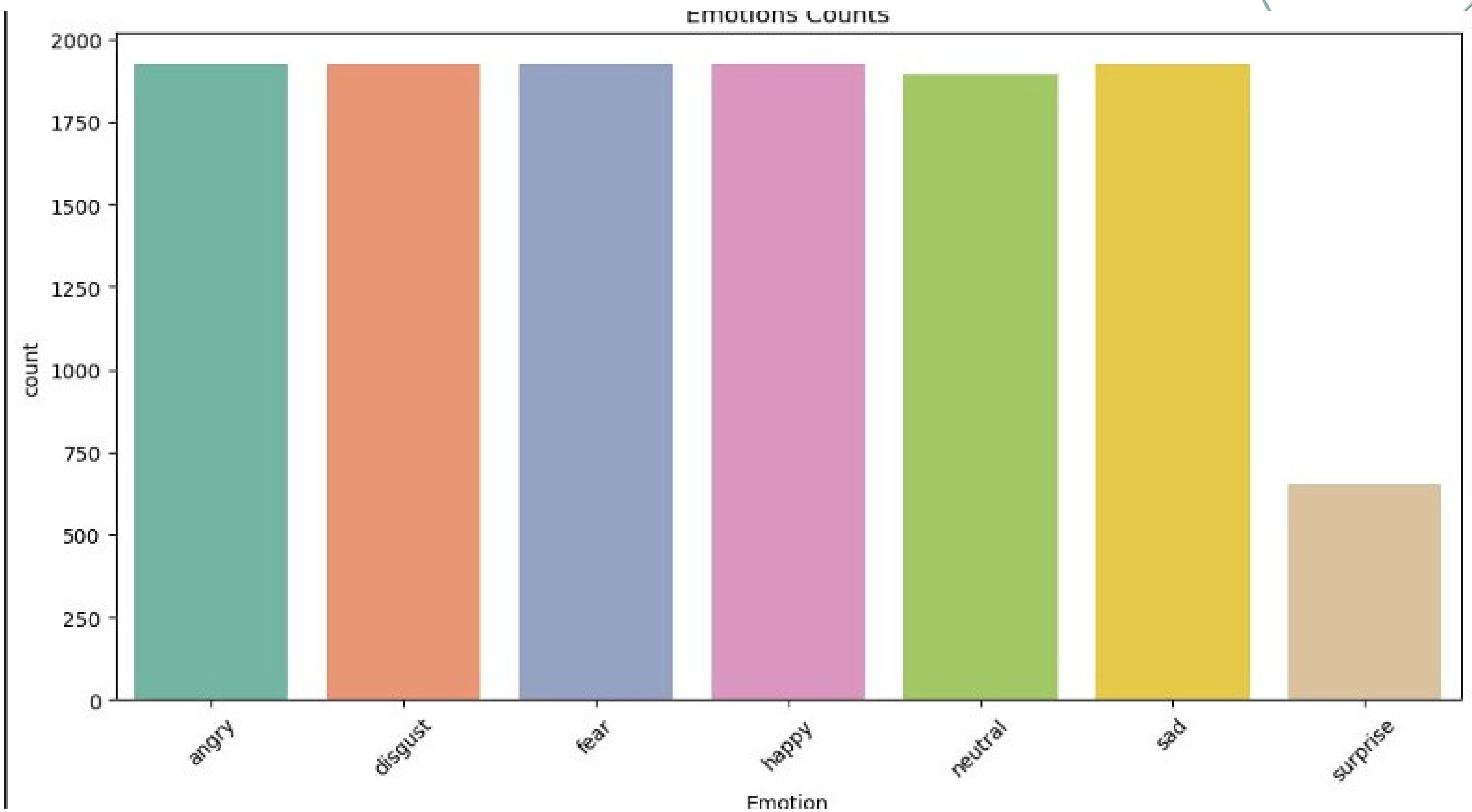
By Anshu Ghate,
Lyu Haozhi,
Wang Xilun

PROBLEM STATEMENT

Using theories and concepts in class for speech emotion recognition. The goal is to classify the emotion of each utterance in a conversation
We will focus on the CNN model.

DATASET

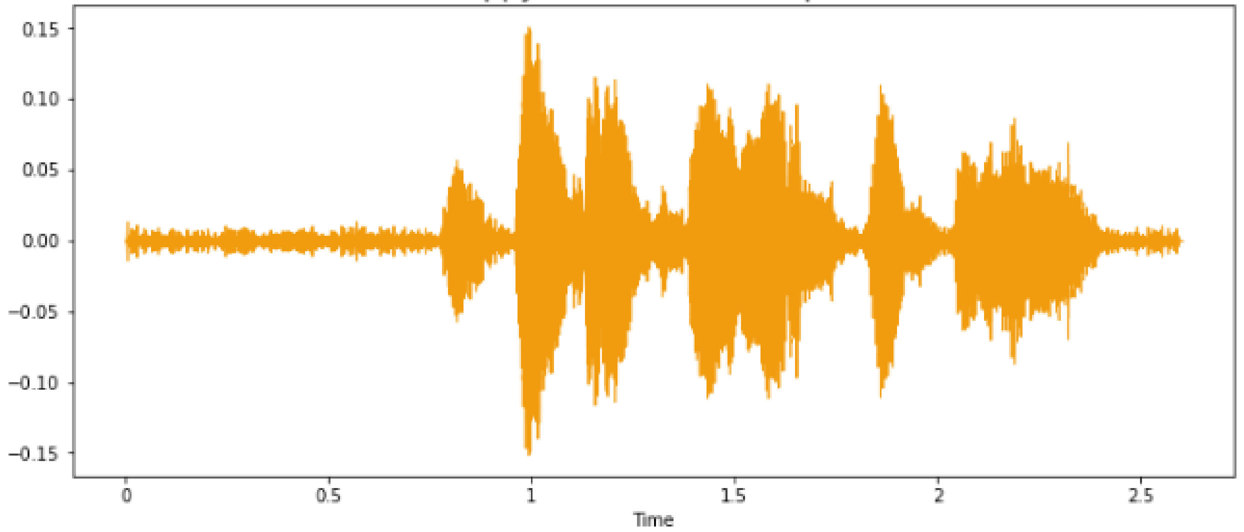
https://www.kaggle.com/datasets/dmitrybabko/speech-emotion-recognition-en



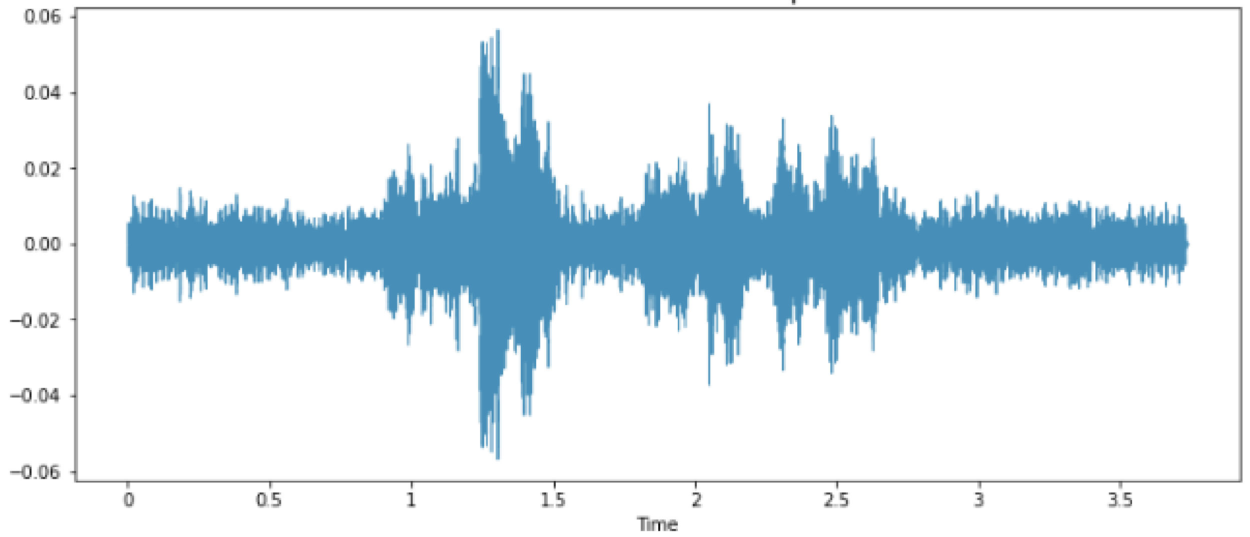
We are using a dataset of audio recordings we found on kaggle which consists of different people saying a particular sentence using various emotions such as happiness, fear, anger etc

SAMPLE AUDIO

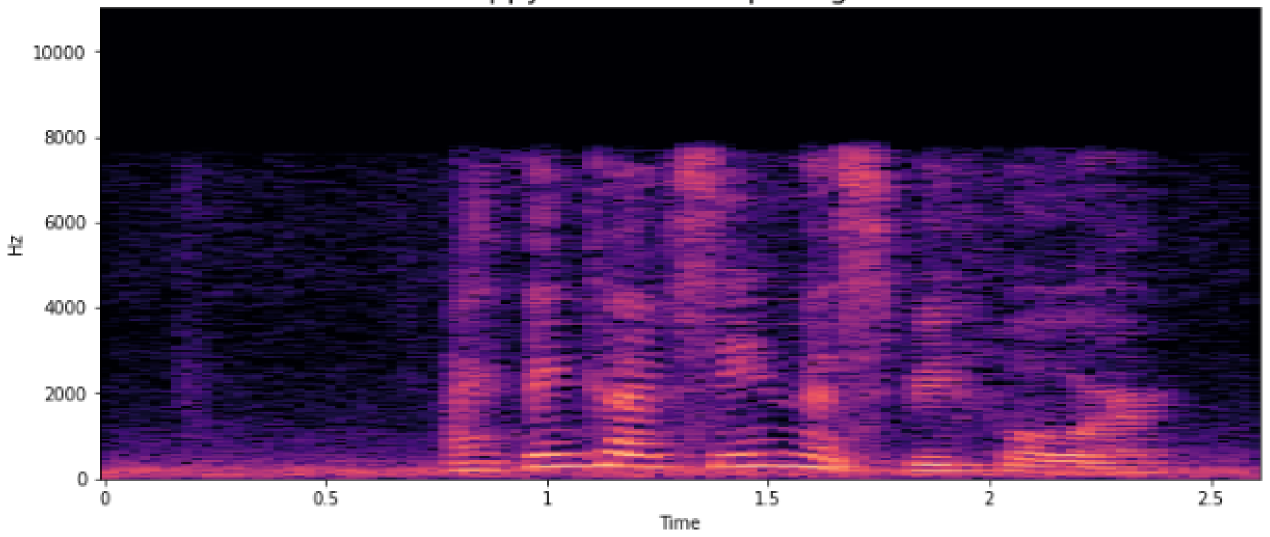
happy emotion for waveplot



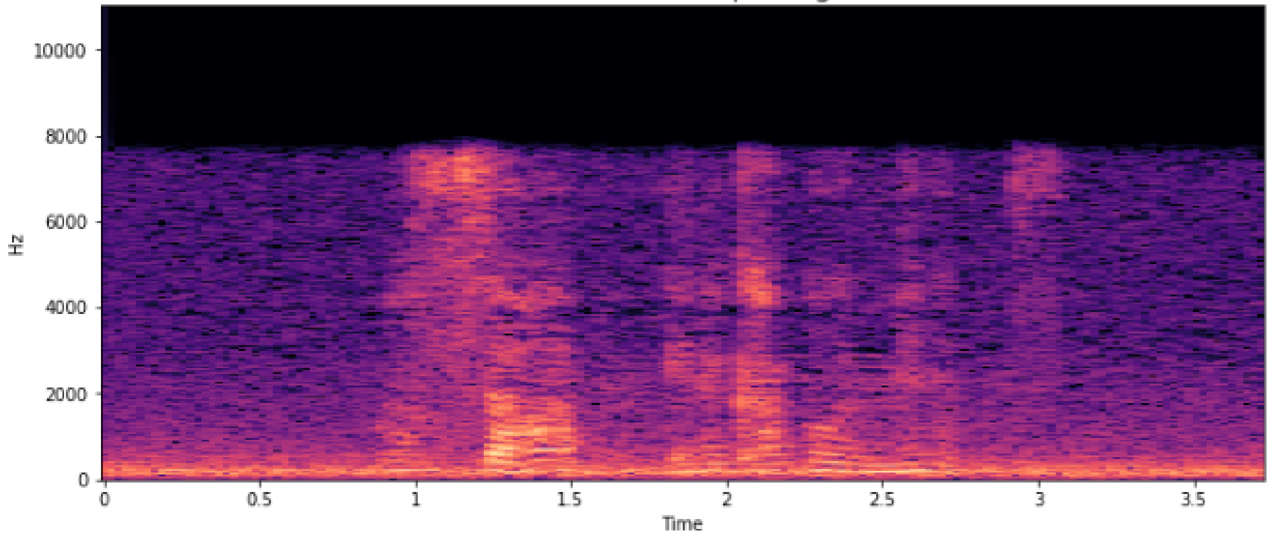
sad emotion for waveplot



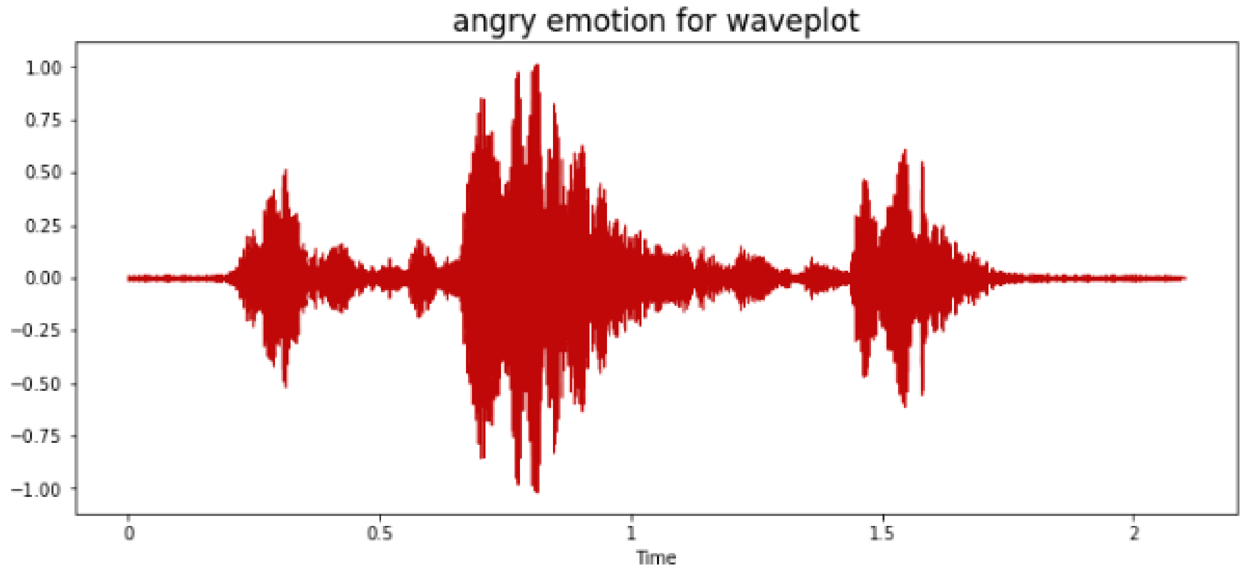
happy emotion for spectogram

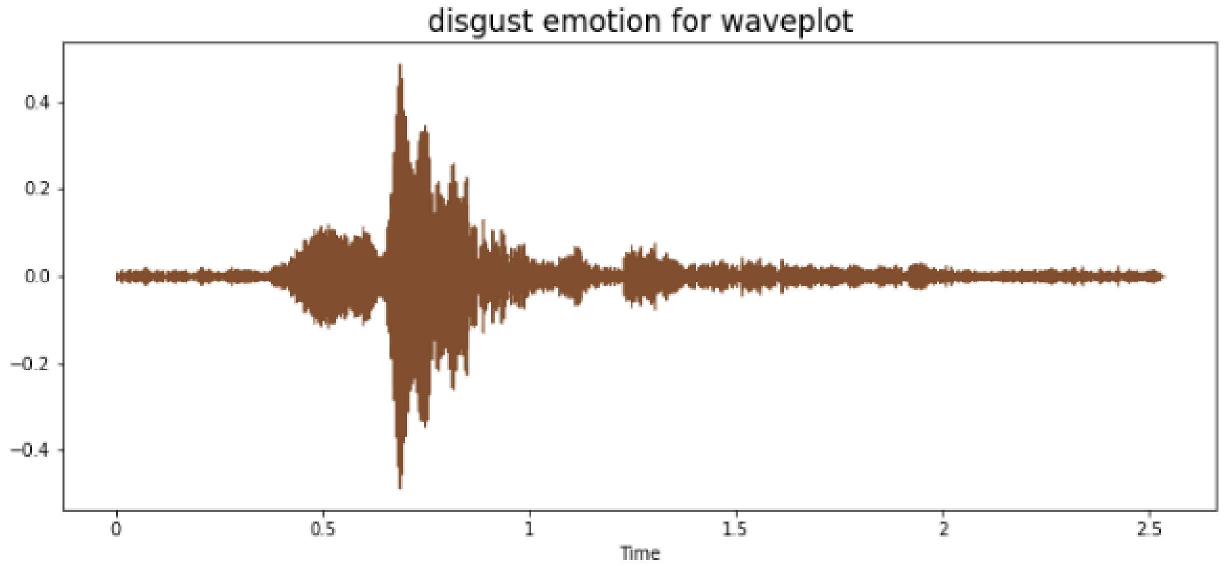


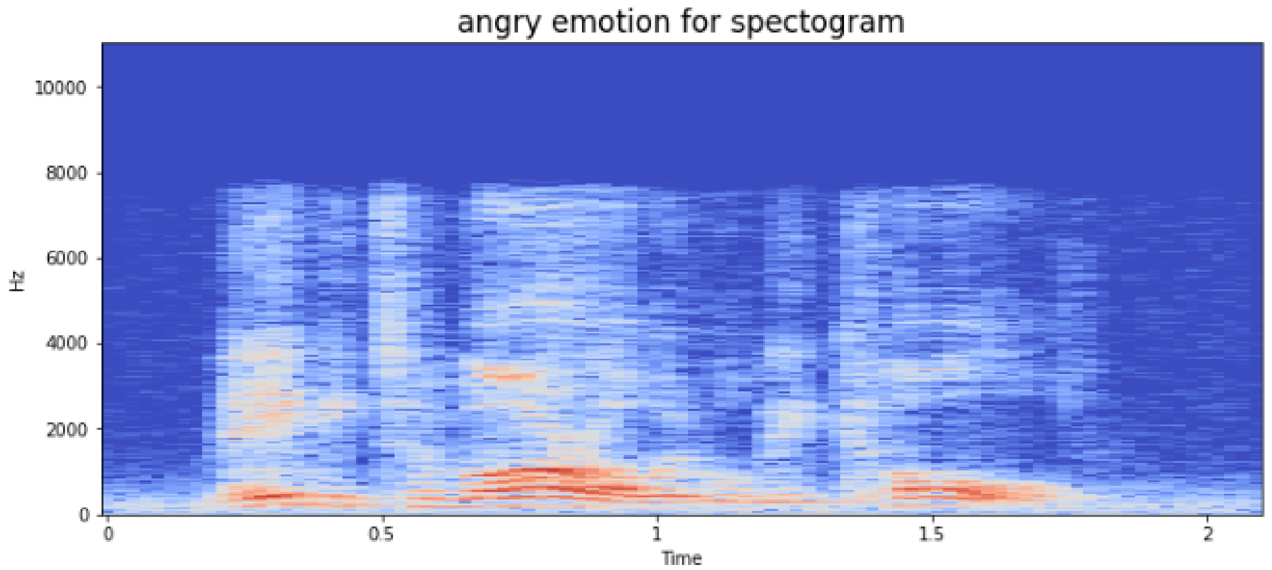
sad emotion for spectogram

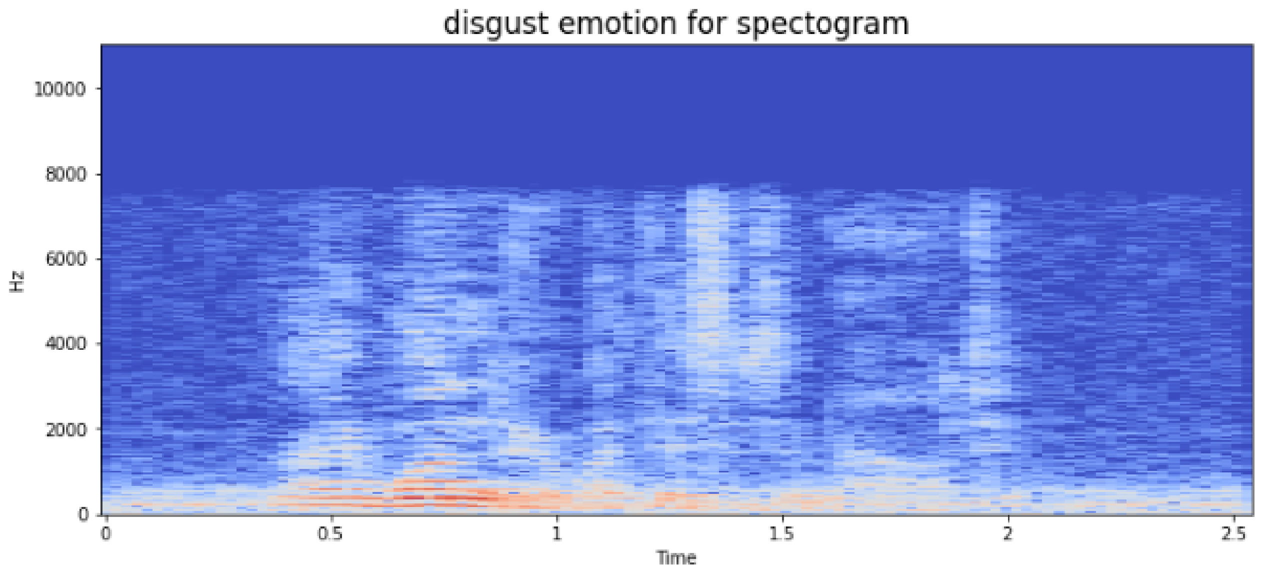


SAMPLE AUDIO



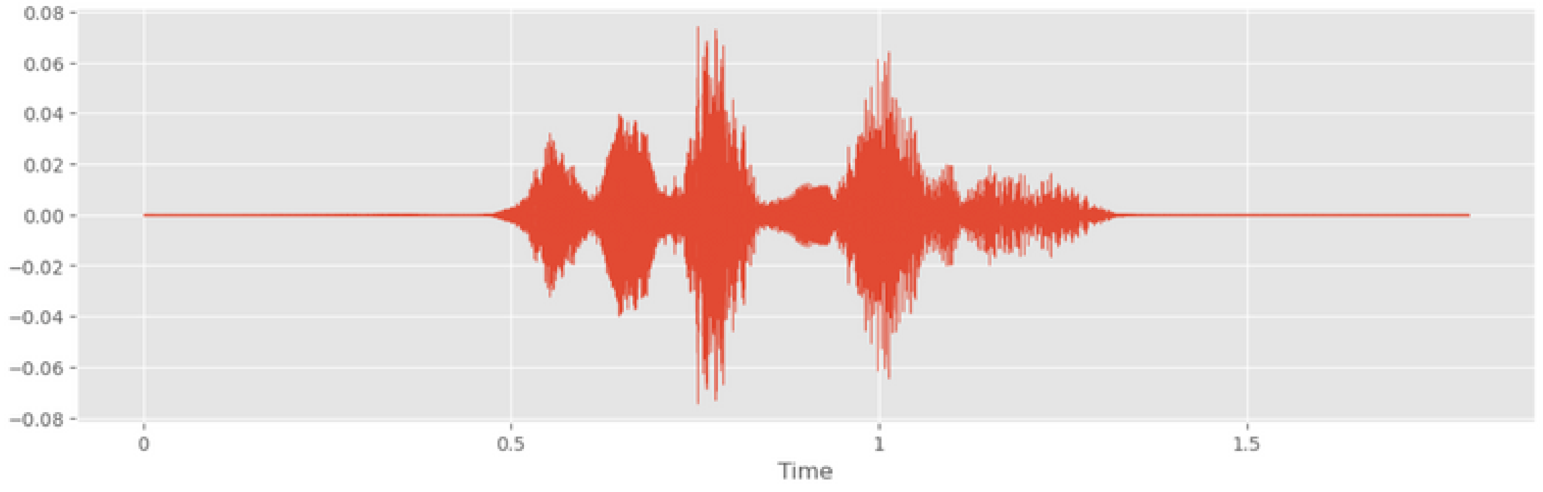






DATA AUGMENTATION

The training data can be made more diverse by using methods like noise injection, stretching, shifting, and pitching.




```
def stretch(data, rate=0.8):
  """Stretching data with some rate."""
  return
librosa.effects.time_stretch(data,
rate=1.0/rate)
```

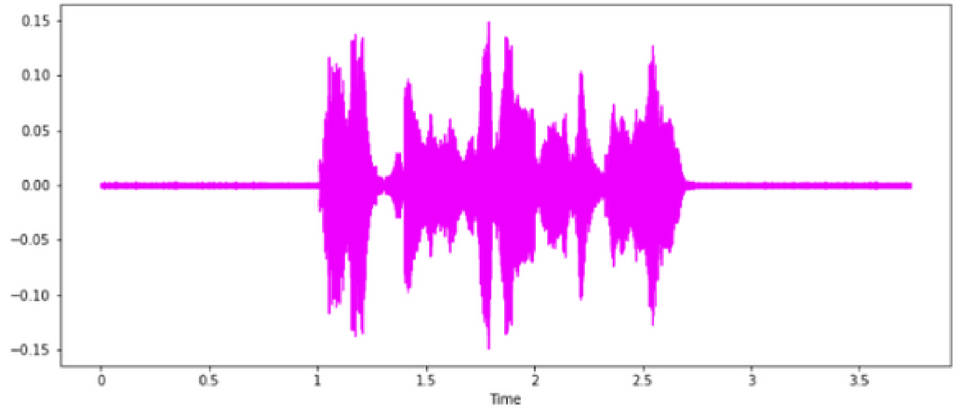
```
stretched_data = stretch(data,
rate=0.5)
plt.figure(figsize=(14,4))
librosa.display.waveshow(y=stretched
_data, sr=sampling_rate)
Audio(stretched_data,
rate=sampling_rate
```

Noised Audio

In [30]:
```
noised_audio=add_noise(data)
plt.figure(figsize=(12,5))
librosa.display.waveshow(noised_audio,sr,color='#EE00FF')
IPython.display.Audio(noised_audio,rate=sr)
```

Out[30]:

0:03 / 0:03



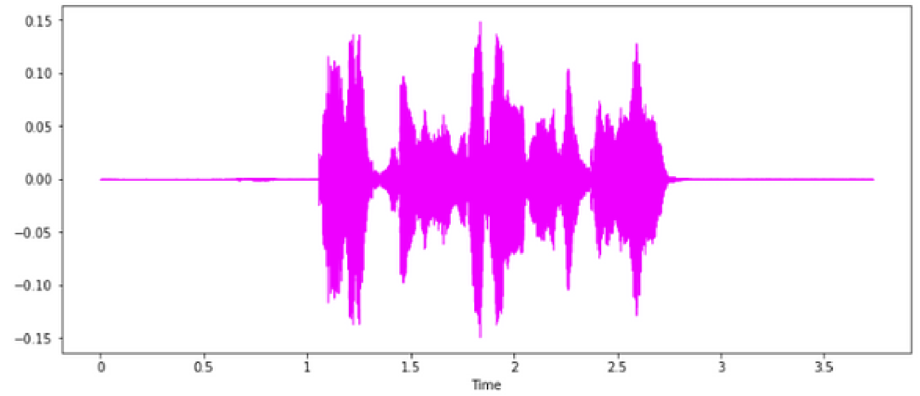
1.Noised/3.shifted

Shifted Audio

In [32]:
```
shifted_audio=shifting(data)
plt.figure(figsize=(12,5))
librosa.display.waveshow(shifted_audio,sr,color='#EE00FF')
IPython.display.Audio(shifted_audio,rate=sr)
```

Out[32]:

0:00 / 0:03

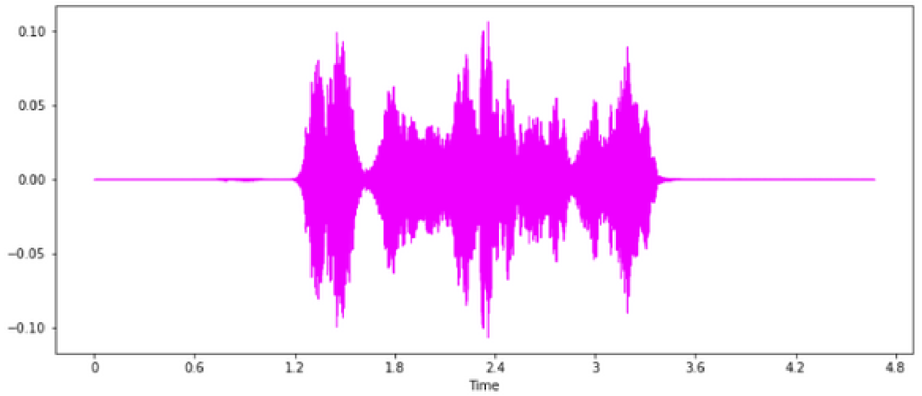


Streched Audio

In [31]:
```
stretched_audio=streching(data)
plt.figure(figsize=(12,5))
librosa.display.waveshow(stretched_audio,sr,color='#EE00FF')
IPython.display.Audio(stretched_audio,rate=sr)
```

Out[31]:

0:03 / 0:04



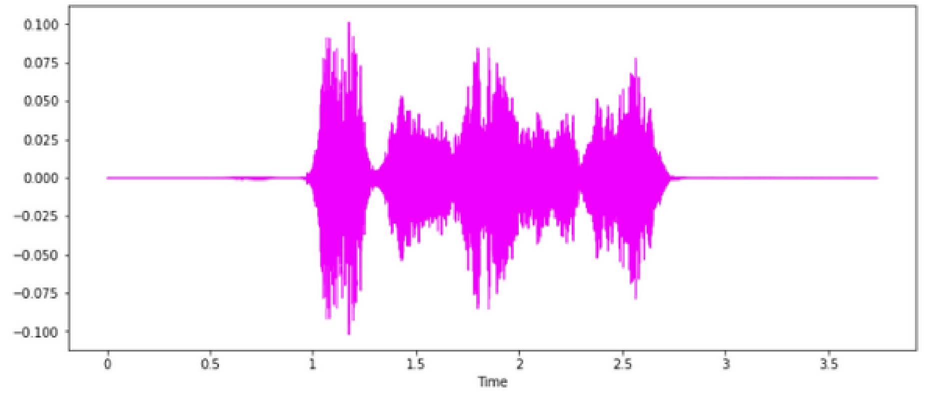
2.Stretched/4.pitched

Pitched Audio

In [33]:
```
pitched_audio=pitching(data,sr)
plt.figure(figsize=(12,5))
librosa.display.waveshow(pitched_audio,sr,color='#EE00FF')
IPython.display.Audio(pitched_audio,rate=sr)
```

Out[33]:

0:03 / 0:03



FEATURE EXTRACTION

After testing various features, it was decided that the three most important features to use are ZCR, RMS, and MFCC.

The features are saved as a DataFrame for further processing using the specified file path of "./features.csv".

```
[ ] # Fill NaN with 0
    extracted_df = extracted_df.fillna(0)
    print(extracted_df.isna().any())
    extracted_df.shape
```

```
0         False
1         False
2         False
3         False
4         False
          ...
2372      False
2373      False
2374      False
2375      False
labels    False
Length: 2377, dtype: bool
(48648, 2377)
```

```
[ ] extracted_df = pd.read_csv(features_path)
    print(extracted_df.shape)
```

```
(48648, 2377)
```

Let's save our features as DataFrame for further processing:

```
[ ] features_path = "./features.csv"
```

01 - DATA PRE-PROCESSING

The data has a x_train(35026, 2376, 1)

02 - PROBLEM

The problem at hand is to classify the emotions present in a given voice message.

03 - ALGORITHM

the algorithm model used in this project is a 1-dimensional Convolutional Neural Network (CNN).

PROBLEM AND ALGORITHM

WHY THIS MODEL

an effective deep learning algorithm for processing sequential data

CALLBACKS USED

EarlyStopping and ReduceLROnPlateau

LAYERS

sequence of Conv1D layers followed by BatchNormalization, MaxPooling1D, and Dense layers.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, 2376, 512)         3072
_________________________________________________________________
batch_normalization (BatchNo (None, 2376, 512)         2048
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 1188, 512)         0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 1188, 512)         1311232
_________________________________________________________________
batch_normalization_1 (Batch (None, 1188, 512)         2048
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 594, 512)          0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 594, 256)          655616
_________________________________________________________________
batch_normalization_2 (Batch (None, 594, 256)          1024
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 297, 256)          0
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 297, 256)          196864
_________________________________________________________________
batch_normalization_3 (Batch (None, 297, 256)          1024
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 149, 256)          0
_________________________________________________________________
conv1d_4 (Conv1D)            (None, 149, 128)          98432
_________________________________________________________________
batch_normalization_4 (Batch (None, 149, 128)          512
_________________________________________________________________
max_pooling1d_4 (MaxPooling1 (None, 75, 128)           0
_________________________________________________________________
flatten (Flatten)            (None, 9600)              0
_________________________________________________________________
dense (Dense)                (None, 512)               4915712
_________________________________________________________________
batch_normalization_5 (Batch (None, 512)               2048
_________________________________________________________________
dense_1 (Dense)              (None, 7)                 3591
=================================================================
Total params: 7,193,223
Trainable params: 7,188,871
Non-trainable params: 4,352
_________________________________________________________________
```

Tuning for Training

```
[ ] early_stop=EarlyStopping(monitor='val_acc',mode='auto',patience=5,restore_best_weights=True)
    lr_reduction=ReduceLROnPlateau(monitor='val_acc',patience=3,verbose=1,factor=0.5,min_lr=0.00001)
```

```
EPOCH=50
BATCH_SIZE=64
```

RESULS OBTAINED

2.3255 TEST LOSS

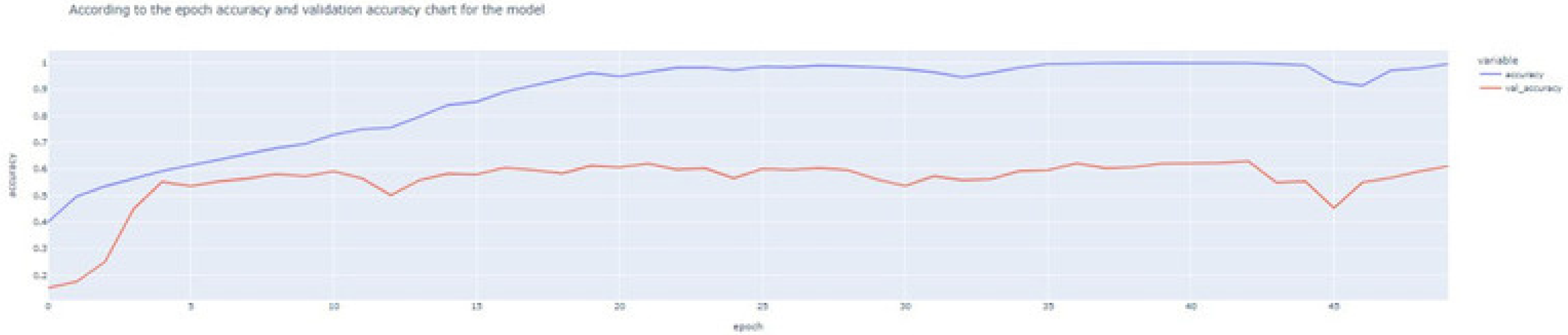
0.5745 TEST ACCURACY

```
Test Loss: 2.3255960941314697
Test Accuracy: 0.5745992660522461
```

```
Epoch 1/50
137/137 [==============================] - 32s 172ms/step - loss: 1.6120 - accuracy: 0.4103 - val_loss: 4.4827 - val_accuracy: 0.1449
Epoch 2/50
137/137 [==============================] - 23s 167ms/step - loss: 1.2921 - accuracy: 0.4967 - val_loss: 3.9161 - val_accuracy: 0.2055
Epoch 3/50
137/137 [==============================] - 23s 167ms/step - loss: 1.1814 - accuracy: 0.5426 - val_loss: 2.4522 - val_accuracy: 0.2816
Epoch 4/50
137/137 [==============================] - 23s 166ms/step - loss: 1.1130 - accuracy: 0.5726 - val_loss: 1.6588 - val_accuracy: 0.4018
Epoch 5/50
137/137 [==============================] - 23s 166ms/step - loss: 1.0375 - accuracy: 0.6011 - val_loss: 1.1303 - val_accuracy: 0.5971
Epoch 6/50
137/137 [==============================] - 23s 166ms/step - loss: 0.9781 - accuracy: 0.6190 - val_loss: 1.5690 - val_accuracy: 0.4789
Epoch 7/50
137/137 [==============================] - 23s 166ms/step - loss: 0.9409 - accuracy: 0.6351 - val_loss: 1.2642 - val_accuracy: 0.5498
Epoch 8/50
137/137 [==============================] - 23s 167ms/step - loss: 0.8901 - accuracy: 0.6566 - val_loss: 1.2182 - val_accuracy: 0.5735
Epoch 9/50
137/137 [==============================] - 23s 166ms/step - loss: 0.8311 - accuracy: 0.6879 - val_loss: 1.3287 - val_accuracy: 0.5334
Epoch 10/50
137/137 [==============================] - 23s 166ms/step - loss: 0.7686 - accuracy: 0.7120 - val_loss: 1.2578 - val_accuracy: 0.5766
Epoch 11/50
137/137 [==============================] - 23s 167ms/step - loss: 0.7157 - accuracy: 0.7253 - val_loss: 1.3222 - val_accuracy: 0.5324
Epoch 12/50
137/137 [==============================] - 23s 166ms/step - loss: 0.6253 - accuracy: 0.7651 - val_loss: 1.3258 - val_accuracy: 0.5673
Epoch 13/50
137/137 [==============================] - 23s 166ms/step - loss: 0.5783 - accuracy: 0.7817 - val_loss: 1.4528 - val_accuracy: 0.5478
Epoch 14/50
137/137 [==============================] - 23s 166ms/step - loss: 0.5304 - accuracy: 0.8015 - val_loss: 1.3841 - val_accuracy: 0.6053
Epoch 15/50
137/137 [==============================] - 23s 166ms/step - loss: 0.4273 - accuracy: 0.8427 - val_loss: 1.3334 - val_accuracy: 0.5920
Epoch 16/50
137/137 [==============================] - 23s 166ms/step - loss: 0.3868 - accuracy: 0.8641 - val_loss: 1.6700 - val_accuracy: 0.5581
Epoch 17/50
137/137 [==============================] - 23s 167ms/step - loss: 0.3458 - accuracy: 0.8764 - val_loss: 1.5918 - val_accuracy: 0.5416
Epoch 18/50
137/137 [==============================] - 23s 166ms/step - loss: 0.2501 - accuracy: 0.9174 - val_loss: 1.5673 - val_accuracy: 0.5755
Epoch 19/50
137/137 [==============================] - 23s 166ms/step - loss: 0.1878 - accuracy: 0.9387 - val_loss: 1.7243 - val_accuracy: 0.5817
Epoch 20/50
137/137 [==============================] - 23s 167ms/step - loss: 0.1521 - accuracy: 0.9540 - val_loss: 1.6290 - val_accuracy: 0.5755
Epoch 21/50
137/137 [==============================] - 23s 166ms/step - loss: 0.1472 - accuracy: 0.9532 - val_loss: 1.7361 - val_accuracy: 0.5714
Epoch 22/50
137/137 [==============================] - 23s 166ms/step - loss: 0.0999 - accuracy: 0.9711 - val_loss: 1.9359 - val_accuracy: 0.5704
Epoch 23/50
137/137 [==============================] - 23s 166ms/step - loss: 0.0648 - accuracy: 0.9833 - val_loss: 1.6852 - val_accuracy: 0.5807
Epoch 24/50
137/137 [==============================] - 23s 166ms/step - loss: 0.0596 - accuracy: 0.9858 - val_loss: 1.9695 - val_accuracy: 0.5755
Epoch 25/50
====================] - 23s 166ms/step - loss: 0.0526 - accuracy: 0.9866 - val_loss: 2.0585 - val_accuracy: 0.5766
```

Accuracy Charts

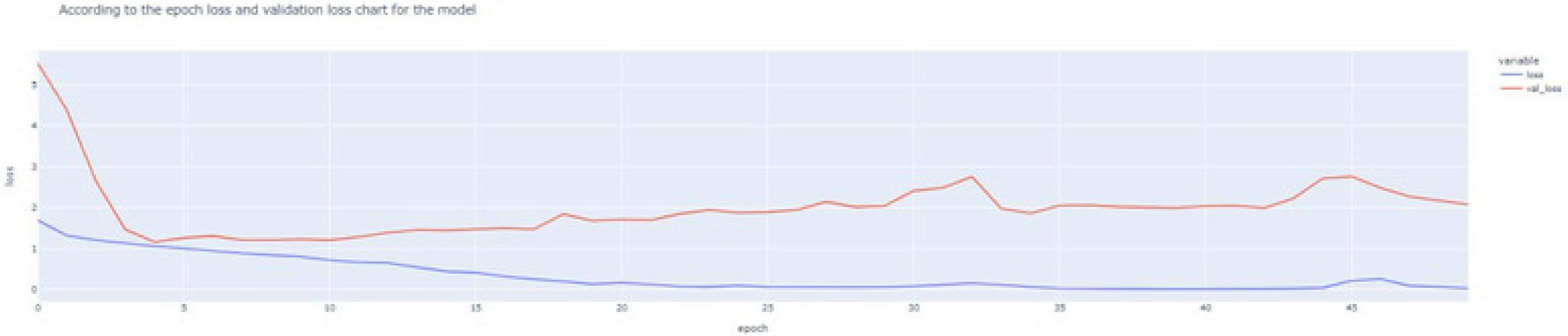
```
fig=px.line(history.history,y=['accuracy','val_accuracy'],
            labels={'index':'epoch','value':'accuracy'},
            title=f'According to the epoch accuracy and validation accuracy chart for the model')
fig.show()
```



Loss Charts

+ Code + Markdown

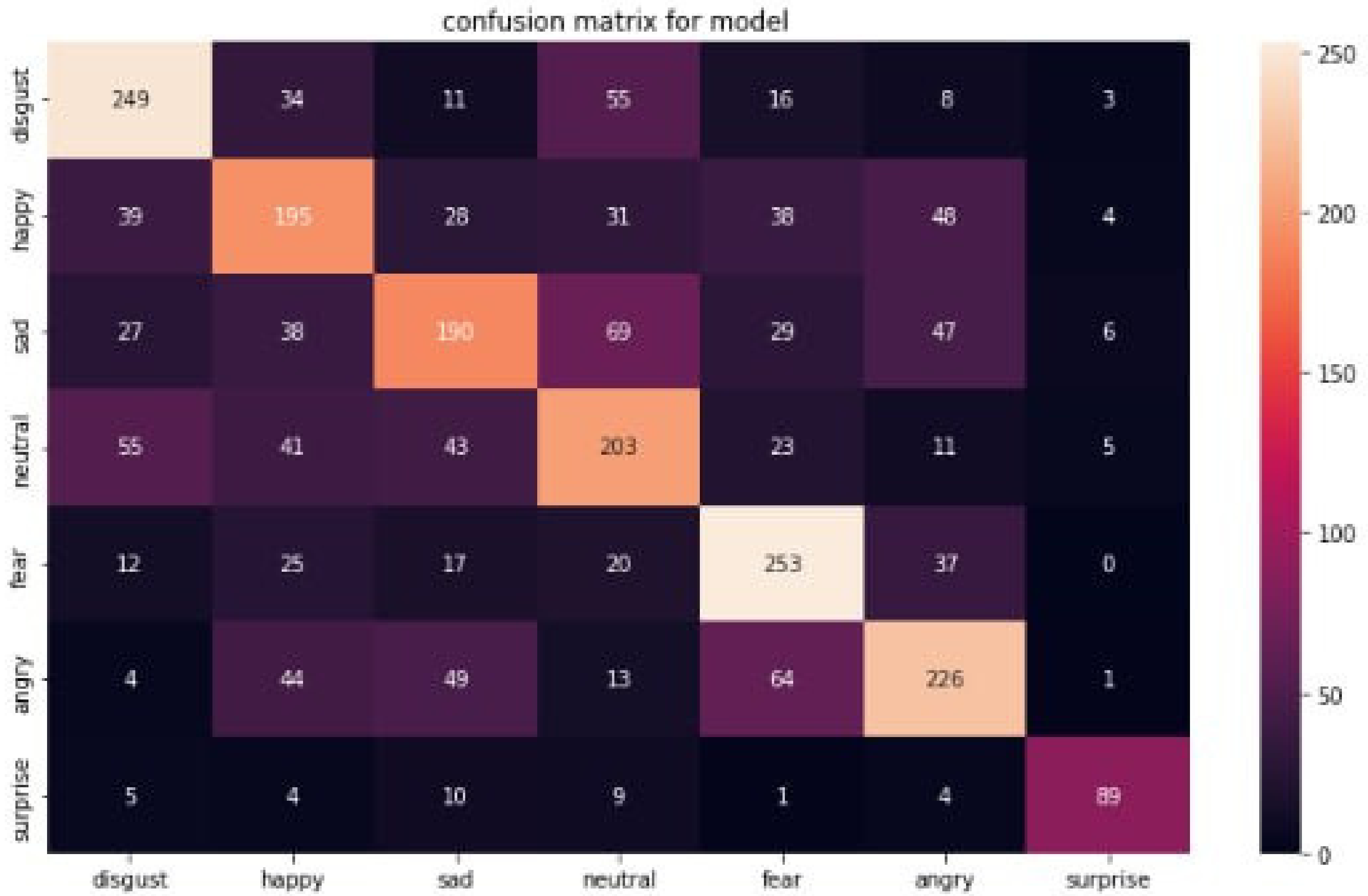
```
fig=px.line(history.history,y=['loss','val_loss'],
           labels={'index':'epoch','value':'loss'},
           title=f'According to the epoch loss and validation loss chart for the model')
fig.show()
```



Confusion Matrix

+ Code + Markdown

```
conf=confusion_matrix(y_check,y_pred)
cm=pd.DataFrame(
    conf,index=[i for i in emotion_names],
    columns=[i for i in emotion_names]
)
plt.figure(figsize=(12,7))
ax=sns.heatmap(cm,annot=True,fmt='d')
ax.set_title(f'confusion matrix for model ')
plt.show()
```



+ Code + Markdown

[53]:

```
print(f'Model Confusion Matrix\n',classification_report(y_check,y_pred,target_r
```

```
Model Confusion Matrix
               precision    recall  f1-score   support

     disgust       0.64      0.66      0.65       376
       happy       0.51      0.51      0.51       383
         sad       0.55      0.47      0.50       406
     neutral       0.51      0.53      0.52       381
        fear       0.60      0.70      0.64       364
       angry       0.59      0.56      0.58       401
    surprise       0.82      0.73      0.77       122

    accuracy                           0.58      2433
   macro avg       0.60      0.59      0.60      2433
weighted avg       0.58      0.58      0.58      2433
```

SUMMARY

Limitations -
a. Limited accuracy
b. Limited generalizability
c.Limited interpretability
d. Limited scalability

Conclusion -
a. Trying different models
b. Better feature extraction
c. Using transfer Learning

THANK YOU