

speech-emotion-recognition

January 31, 2024

0.1 Import Modules

```
[30]: import pandas as pd
import numpy as np
import os
import seaborn as sns
import matplotlib.pyplot as plt
import librosa
import librosa.display
from IPython.display import Audio
import warnings
warnings.filterwarnings('ignore')
```

0.2 Load the Dataset

```
[31]: paths = []
labels = []
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        paths.append(os.path.join(dirname, filename))
        label = filename.split('_')[-1]
        label = label.split('.')[0]
        labels.append(label.lower())
    if len(paths) == 2800:
        break
print('Dataset is Loaded')
```

Dataset is Loaded

```
[32]: len(paths)
```

```
[32]: 2800
```

```
[33]: paths[:5]
```

```
[33]: ['/kaggle/input/toronto-emotional-speech-set-tess/TESS Toronto emotional speech
set data/YAF_fear/YAF_home_fear.wav',
'/kaggle/input/toronto-emotional-speech-set-tess/TESS Toronto emotional speech
```

```

set data/YAF_fear/YAF_youth_fear.wav',
'/kaggle/input/toronto-emotional-speech-set-tess/TESS Toronto emotional speech
set data/YAF_fear/YAF_near_fear.wav',
'/kaggle/input/toronto-emotional-speech-set-tess/TESS Toronto emotional speech
set data/YAF_fear/YAF_search_fear.wav',
'/kaggle/input/toronto-emotional-speech-set-tess/TESS Toronto emotional speech
set data/YAF_fear/YAF_pick_fear.wav']

```

```
[34]: labels[:5]
```

```
[34]: ['fear', 'fear', 'fear', 'fear', 'fear']
```

```

[35]: ## Create a dataframe
df = pd.DataFrame()
df['speech'] = paths
df['label'] = labels
df.head()

```

```

[35]:
           speech label
0  /kaggle/input/toronto-emotional-speech-set-tes...  fear
1  /kaggle/input/toronto-emotional-speech-set-tes...  fear
2  /kaggle/input/toronto-emotional-speech-set-tes...  fear
3  /kaggle/input/toronto-emotional-speech-set-tes...  fear
4  /kaggle/input/toronto-emotional-speech-set-tes...  fear

```

```
[36]: df['label'].value_counts()
```

```

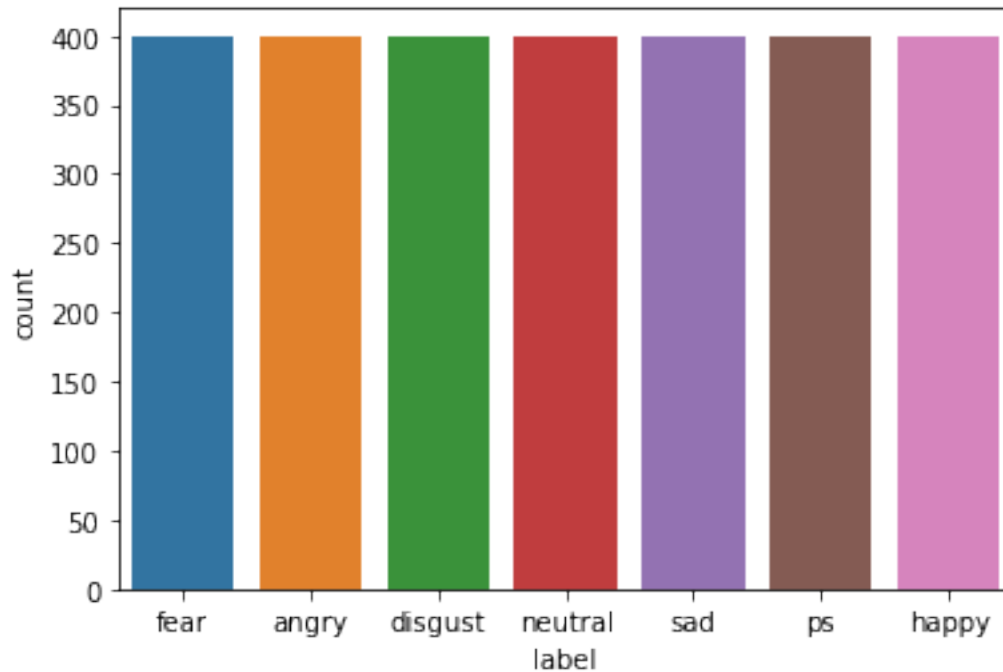
[36]: fear      400
      angry     400
      disgust   400
      neutral   400
      sad       400
      ps        400
      happy     400
      Name: label, dtype: int64

```

0.3 Exploratory Data Analysis

```
[37]: sns.countplot(df['label'])
```

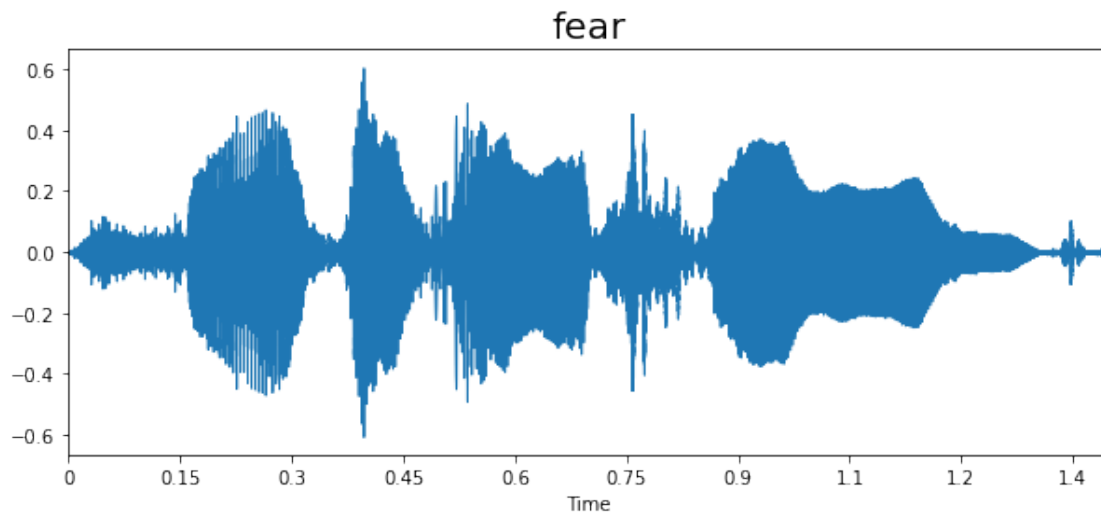
```
[37]: <AxesSubplot:xlabel='label', ylabel='count'>
```



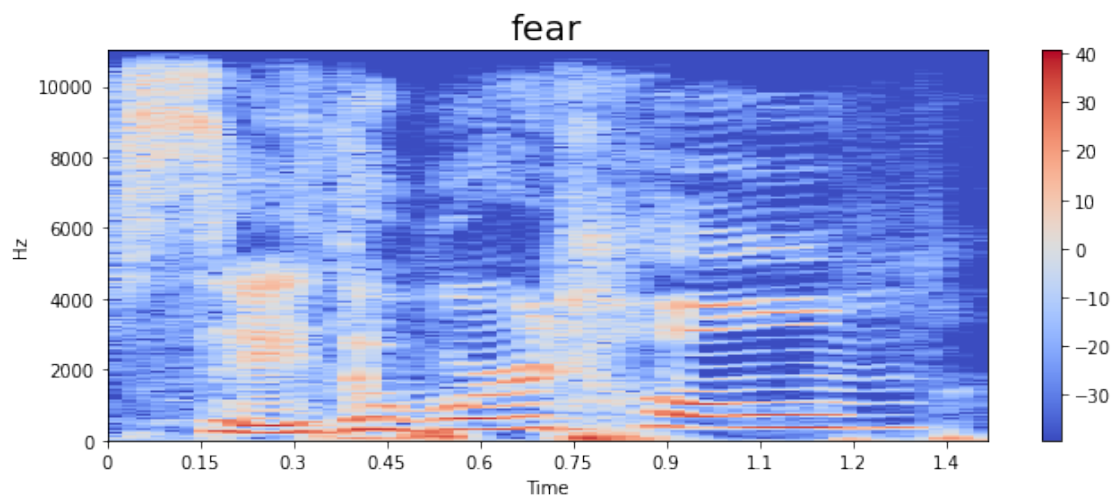
```
[38]: def waveplot(data, sr, emotion):
    plt.figure(figsize=(10,4))
    plt.title(emotion, size=20)
    librosa.display.waveplot(data, sr=sr)
    plt.show()

    def spectrogram(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()
```

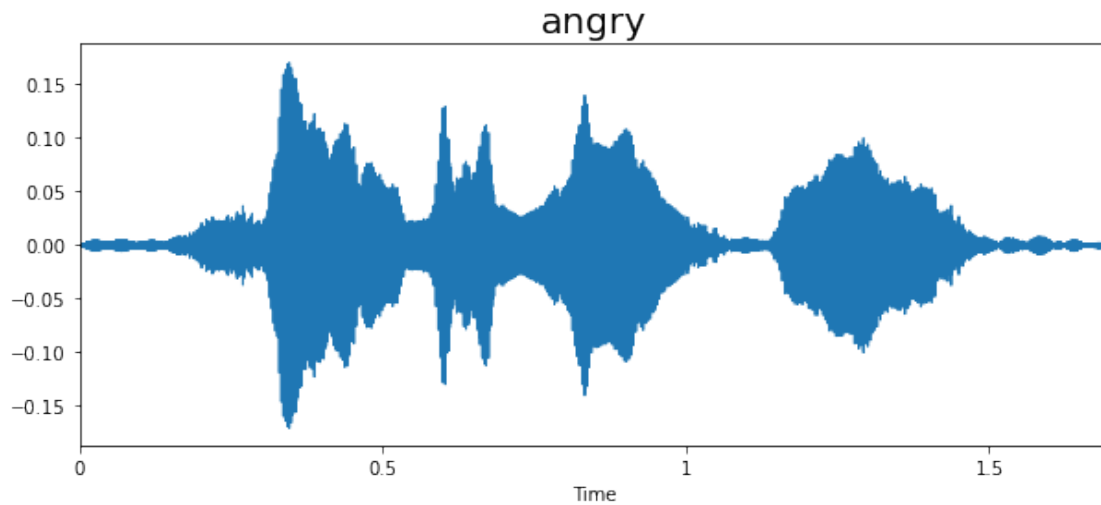
```
[39]: emotion = 'fear'
path = np.array(df['speech'][df['label']==emotion])[0]
data, sampling_rate = librosa.load(path)
waveplot(data, sampling_rate, emotion)
spectrogram(data, sampling_rate, emotion)
Audio(path)
```



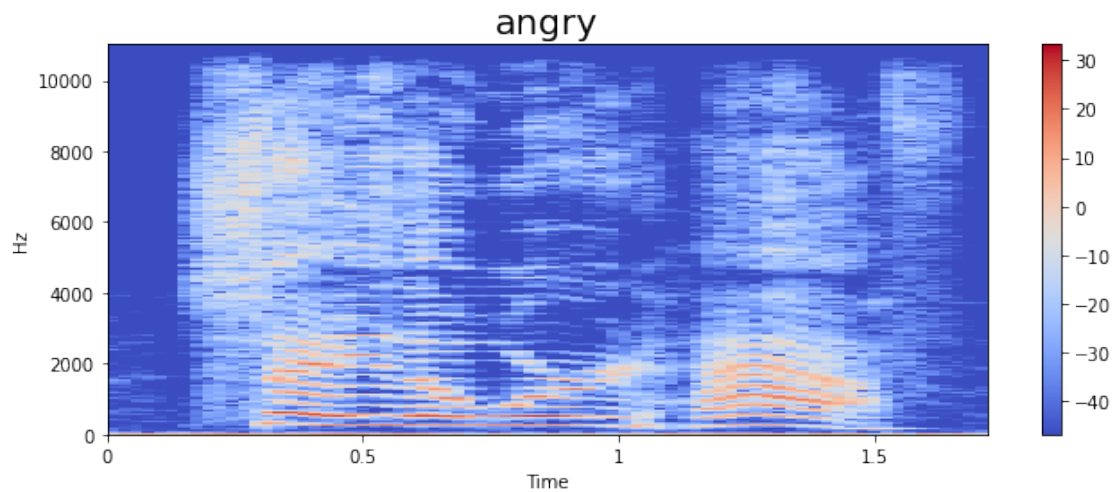
[39]: <IPython.lib.display.Audio object>



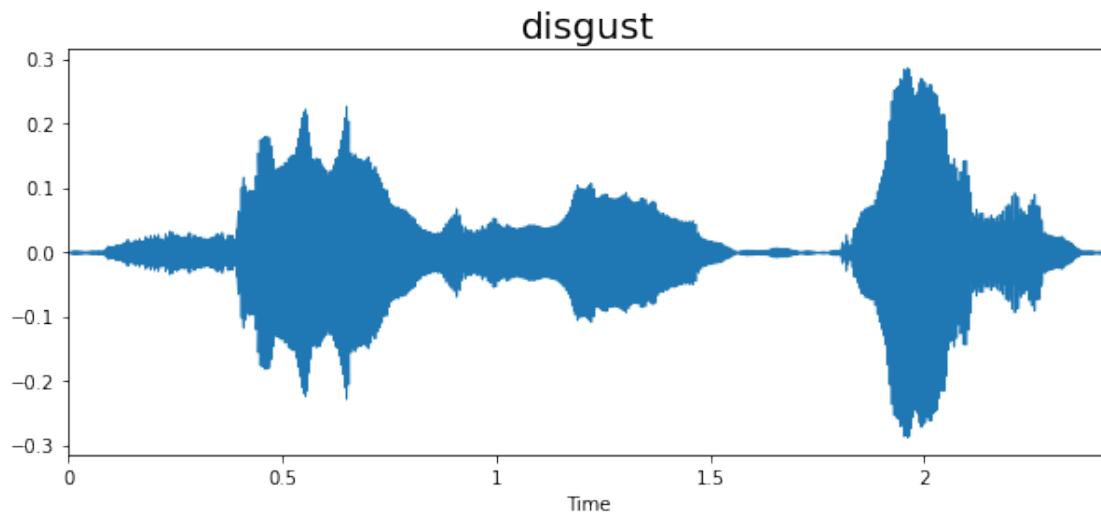
```
[40]: emotion = 'angry'
path = np.array(df['speech'][df['label']==emotion])[1]
data, sampling_rate = librosa.load(path)
waveplot(data, sampling_rate, emotion)
spectrogram(data, sampling_rate, emotion)
Audio(path)
```



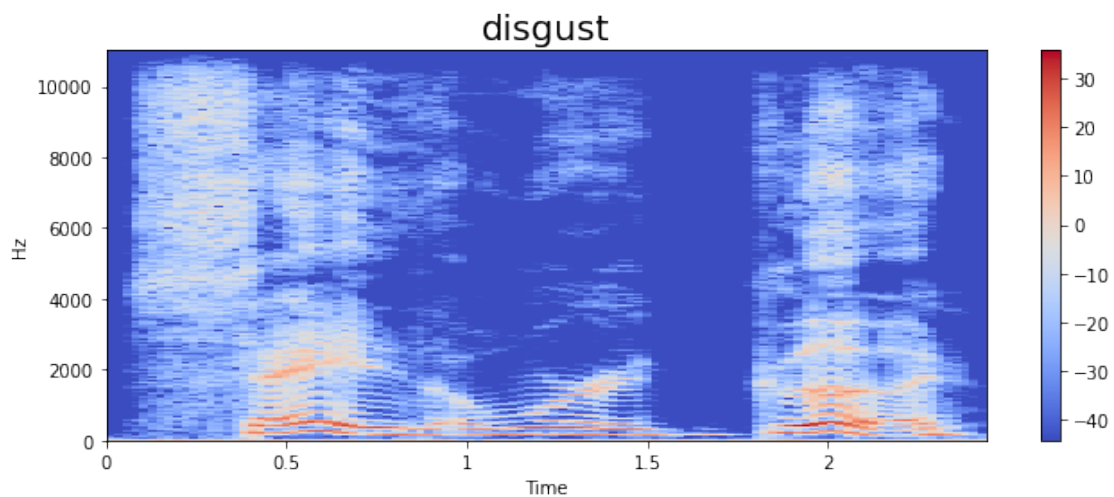
[40]: <IPython.lib.display.Audio object>



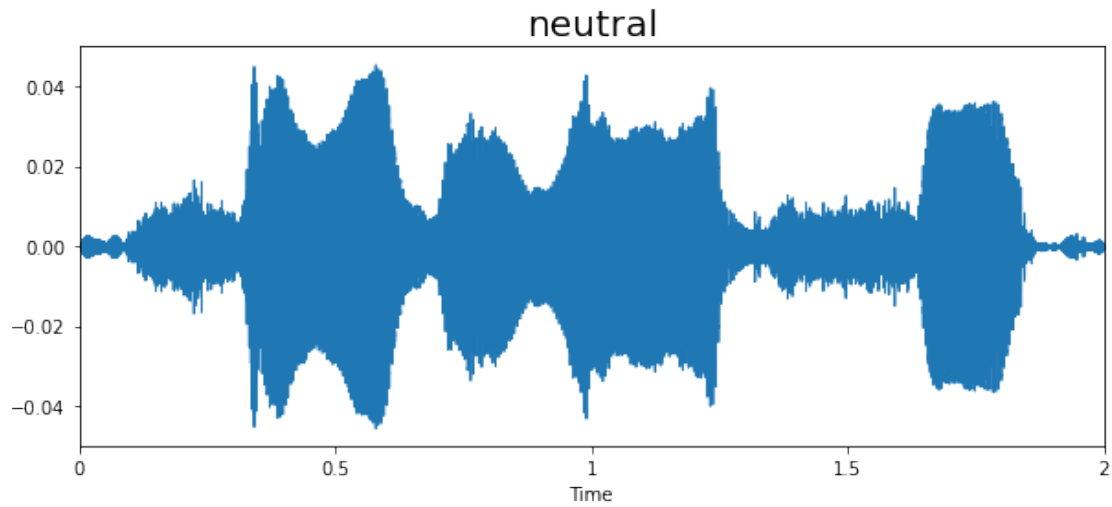
```
[41]: emotion = 'disgust'
path = np.array(df['speech'][df['label']==emotion])[0]
data, sampling_rate = librosa.load(path)
waveplot(data, sampling_rate, emotion)
spectrogram(data, sampling_rate, emotion)
Audio(path)
```



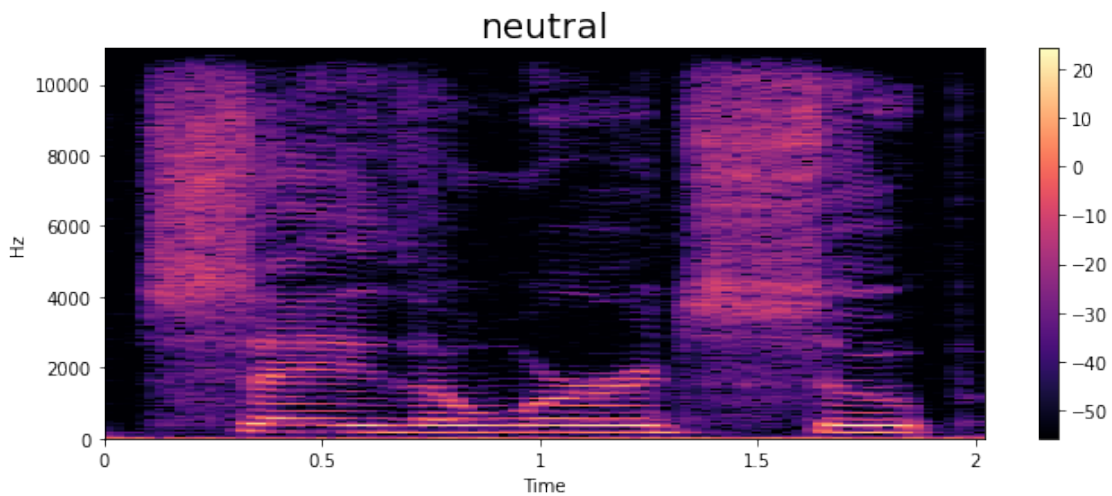
[41]: <IPython.lib.display.Audio object>



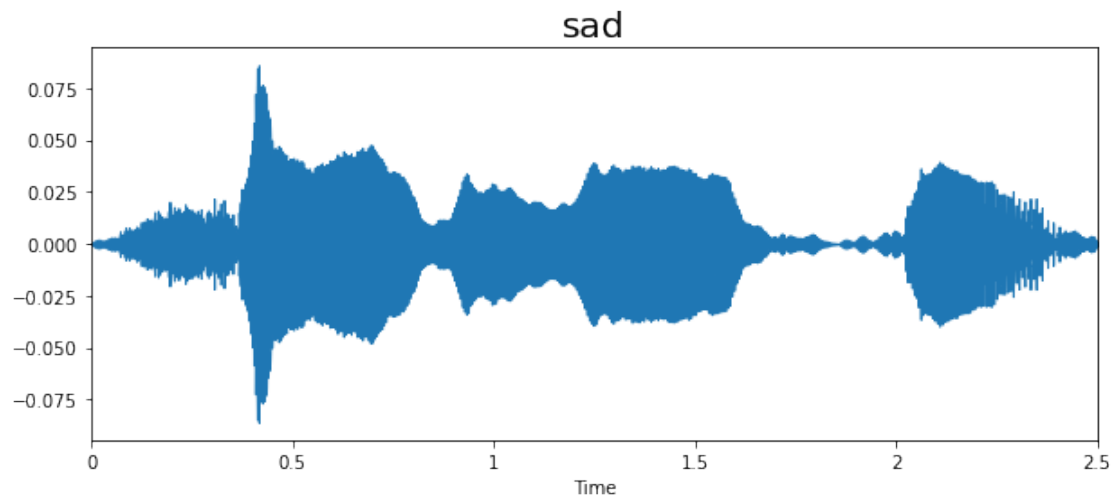
```
[42]: emotion = 'neutral'
path = np.array(df['speech'][df['label']==emotion])[0]
data, sampling_rate = librosa.load(path)
waveplot(data, sampling_rate, emotion)
spectrogram(data, sampling_rate, emotion)
Audio(path)
```



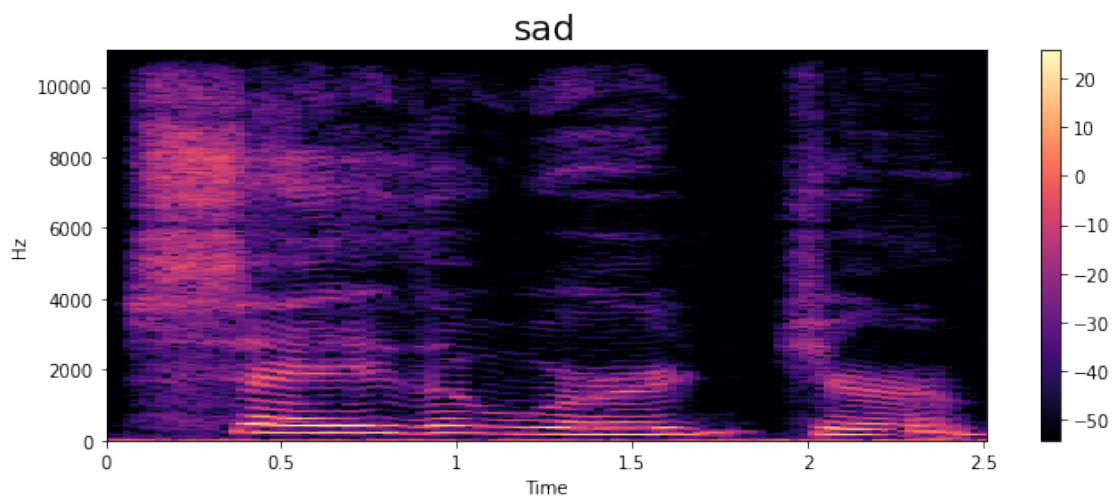
[42]: <IPython.lib.display.Audio object>



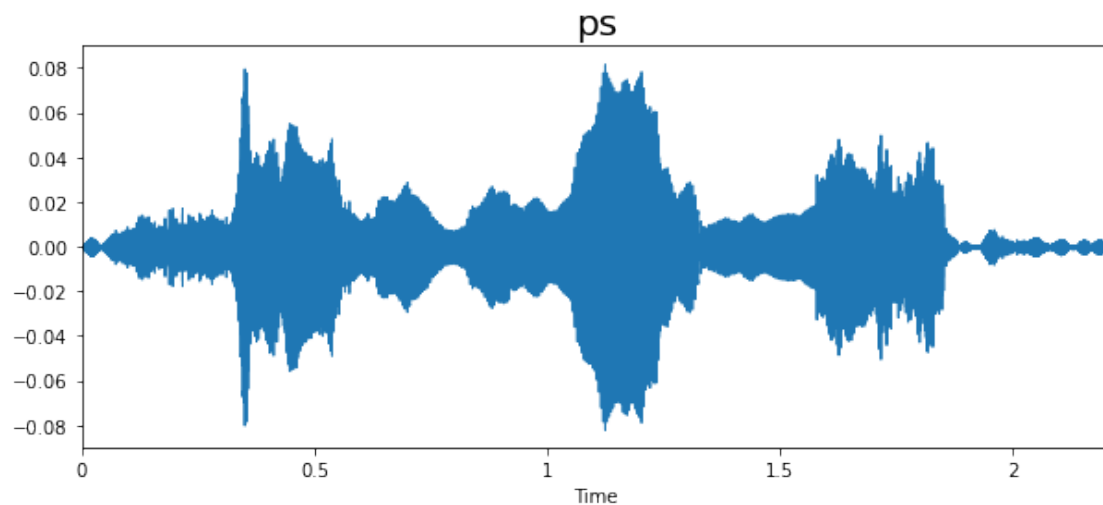
```
[43]: emotion = 'sad'
path = np.array(df['speech'][df['label']==emotion])[0]
data, sampling_rate = librosa.load(path)
waveplot(data, sampling_rate, emotion)
spectrogram(data, sampling_rate, emotion)
Audio(path)
```



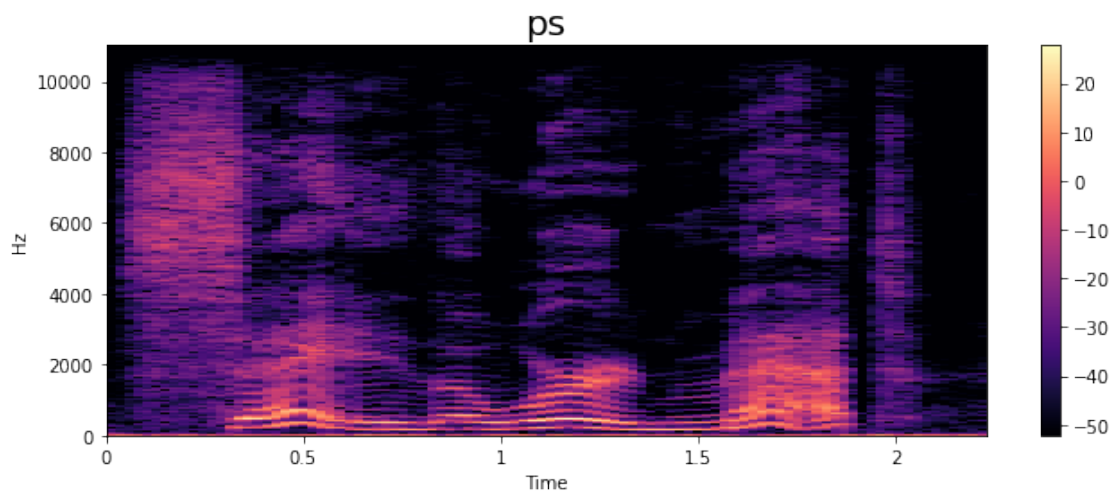
[43]: <IPython.lib.display.Audio object>



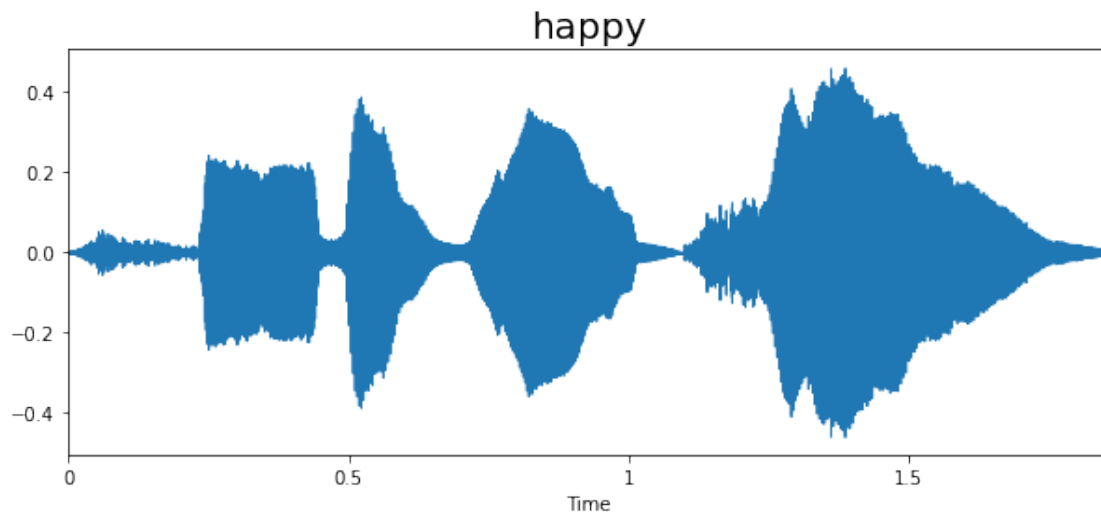
```
[44]: emotion = 'ps'
path = np.array(df['speech'][df['label']==emotion])[0]
data, sampling_rate = librosa.load(path)
waveplot(data, sampling_rate, emotion)
spectrogram(data, sampling_rate, emotion)
Audio(path)
```

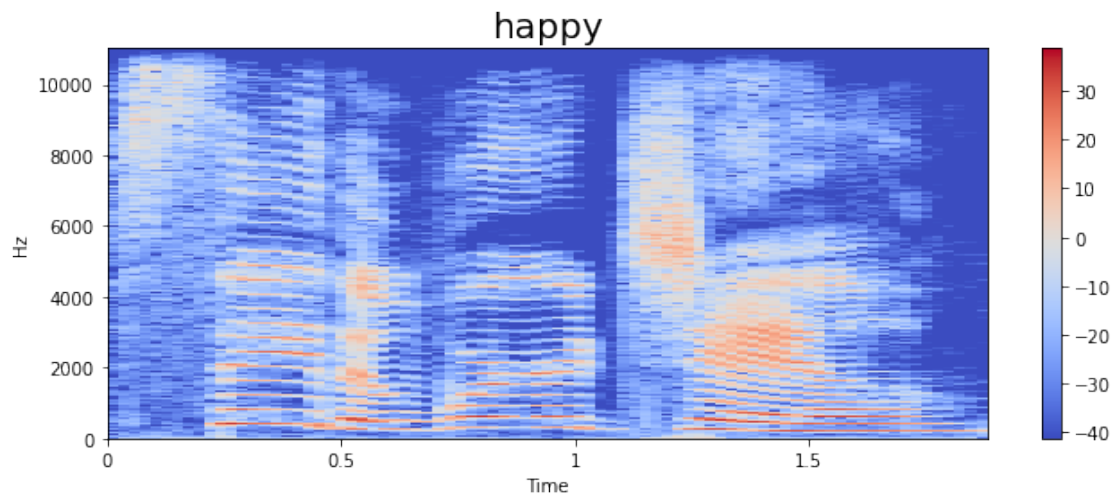
[44]: <IPython.lib.display.Audio object>



```
[45]: emotion = 'happy'
path = np.array(df['speech'][df['label']==emotion])[0]
data, sampling_rate = librosa.load(path)
waveplot(data, sampling_rate, emotion)
spectrogram(data, sampling_rate, emotion)
Audio(path)
```



[45]: <IPython.lib.display.Audio object>



0.4 Feature Extraction

```
[46]: def extract_mfcc(filename):  
    y, sr = librosa.load(filename, duration=3, offset=0.5)  
    mfcc = np.mean(librosa.feature.mfcc(y=y, sr=sr, n_mfcc=40).T, axis=0)  
    return mfcc
```

```
[47]: extract_mfcc(df['speech'][0])
```

```
[47]: array([-285.2542   ,  86.24267   , -2.7735834 ,  22.61731   ,
          -15.214631  ,  11.602871  ,  11.931779  , -2.5318177 ,
           0.65986294,  11.62756   , -17.814924  , -7.5654893 ,
           6.2167835 , -3.7255652 , -9.563306   ,  3.899267   ,
          -13.657834 ,  14.420068 ,  19.243341 ,  23.024492 ,
           32.129776 ,  16.585697 , -4.137755   ,  1.2746525 ,
          -11.517016 ,  7.0145273 , -2.8494127 , -7.415011 ,
          -11.150621 , -2.1190548 , -5.4515266 ,  4.473824 ,
          -11.377713 , -8.931878 , -3.8482094 ,  4.950994 ,
          -1.7254968 ,  2.659218 ,  11.390564 ,  11.3327265 ],
        dtype=float32)
```

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

```
[49]: X_mfcc
```

```
[49]: 0      [-285.2542, 86.24267, -2.7735834, 22.61731, -1...
1      [-348.23337, 35.60242, -4.365128, 15.534869, 6...
2      [-339.50308, 54.41241, -14.795754, 21.566118, ...
3      [-306.92944, 21.973307, -5.1588626, 7.6269317,...
4      [-344.88586, 47.05694, -24.83122, 20.24406, 1...

...

2795    [-374.1317, 61.859463, -0.41998756, 9.31088, -...
2796    [-314.12222, 40.262157, -6.7909045, -3.2963052...
2797    [-357.65854, 78.49201, -15.684815, 3.644915, -...
2798    [-352.78336, 102.219765, -14.560364, -11.48181...
2799    [-389.80002, 54.120773, 0.8988281, -0.6595729,...
Name: speech, Length: 2800, dtype: object
```

```
[50]: X = [x for x in X_mfcc]
X = np.array(X)
X.shape
```

```
[50]: (2800, 40)
```

```
[51]: ## input split
X = np.expand_dims(X, -1)
X.shape
```

```
[51]: (2800, 40, 1)
```

```
[52]: from sklearn.preprocessing import OneHotEncoder
enc = OneHotEncoder()
y = enc.fit_transform(df[['label']])
```

```
[53]: y = y.toarray()
```

```
[54]: y.shape
```

```
[54]: (2800, 7)
```

0.5 Create the LSTM Model

```
[101]: from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout

model = Sequential([
    LSTM(256, return_sequences=False, input_shape=(40,1)),
    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_24"

Layer (type)	Output Shape	Param #
lstm_30 (LSTM)	(None, 256)	264192
dropout_62 (Dropout)	(None, 256)	0
dense_72 (Dense)	(None, 128)	32896
dropout_63 (Dropout)	(None, 128)	0
dense_73 (Dense)	(None, 64)	8256
dropout_64 (Dropout)	(None, 64)	0
dense_74 (Dense)	(None, 7)	455
Total params: 305,799		
Trainable params: 305,799		
Non-trainable params: 0		

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

```
Epoch 1/50
35/35 [=====] - 2s 18ms/step - loss: 1.0892 - accuracy:
0.6201 - val_loss: 2.0684 - val_accuracy: 0.2946
Epoch 2/50
35/35 [=====] - 0s 7ms/step - loss: 0.3742 - accuracy:
0.8598 - val_loss: 2.4078 - val_accuracy: 0.2054
Epoch 3/50
35/35 [=====] - 0s 8ms/step - loss: 0.1671 - accuracy:
0.9487 - val_loss: 1.9055 - val_accuracy: 0.4446
Epoch 4/50
35/35 [=====] - 0s 8ms/step - loss: 0.1672 - accuracy:
0.9442 - val_loss: 2.7364 - val_accuracy: 0.3179
Epoch 5/50
35/35 [=====] - 0s 7ms/step - loss: 0.1107 - accuracy:
0.9683 - val_loss: 1.8414 - val_accuracy: 0.5607
Epoch 6/50
35/35 [=====] - 0s 8ms/step - loss: 0.1082 - accuracy:
0.9585 - val_loss: 2.7303 - val_accuracy: 0.4679
Epoch 7/50
35/35 [=====] - 0s 9ms/step - loss: 0.1275 - accuracy:
0.9656 - val_loss: 1.2245 - val_accuracy: 0.7232
Epoch 8/50
35/35 [=====] - 0s 7ms/step - loss: 0.0765 - accuracy:
0.9786 - val_loss: 2.9995 - val_accuracy: 0.3893
Epoch 9/50
35/35 [=====] - 0s 8ms/step - loss: 0.0577 - accuracy:
0.9795 - val_loss: 3.5068 - val_accuracy: 0.3679
Epoch 10/50
35/35 [=====] - 0s 8ms/step - loss: 0.0531 - accuracy:
0.9826 - val_loss: 2.3169 - val_accuracy: 0.4786
Epoch 11/50
35/35 [=====] - 0s 8ms/step - loss: 0.0357 - accuracy:
0.9888 - val_loss: 3.8930 - val_accuracy: 0.4321
Epoch 12/50
35/35 [=====] - 0s 7ms/step - loss: 0.0554 - accuracy:
0.9835 - val_loss: 1.5915 - val_accuracy: 0.6607
Epoch 13/50
35/35 [=====] - 0s 7ms/step - loss: 0.0684 - accuracy:
0.9790 - val_loss: 3.4805 - val_accuracy: 0.4589
Epoch 14/50
35/35 [=====] - 0s 7ms/step - loss: 0.0453 - accuracy:
0.9835 - val_loss: 2.4777 - val_accuracy: 0.4661
Epoch 15/50
35/35 [=====] - 0s 8ms/step - loss: 0.0874 - accuracy:
```

0.9737 - val_loss: 4.4744 - val_accuracy: 0.2446
 Epoch 16/50
 35/35 [=====] - 0s 8ms/step - loss: 0.0530 - accuracy:
 0.9853 - val_loss: 2.9993 - val_accuracy: 0.5232
 Epoch 17/50
 35/35 [=====] - 0s 6ms/step - loss: 0.0421 - accuracy:
 0.9839 - val_loss: 4.3298 - val_accuracy: 0.4714
 Epoch 18/50
 35/35 [=====] - 0s 8ms/step - loss: 0.0514 - accuracy:
 0.9857 - val_loss: 2.2161 - val_accuracy: 0.5946
 Epoch 19/50
 35/35 [=====] - 0s 7ms/step - loss: 0.0310 - accuracy:
 0.9897 - val_loss: 3.7546 - val_accuracy: 0.4071
 Epoch 20/50
 35/35 [=====] - 0s 7ms/step - loss: 0.0301 - accuracy:
 0.9897 - val_loss: 2.7526 - val_accuracy: 0.5036
 Epoch 21/50
 35/35 [=====] - 0s 8ms/step - loss: 0.0342 - accuracy:
 0.9875 - val_loss: 4.7068 - val_accuracy: 0.2839
 Epoch 22/50
 35/35 [=====] - 0s 8ms/step - loss: 0.0295 - accuracy:
 0.9893 - val_loss: 3.4425 - val_accuracy: 0.4054
 Epoch 23/50
 35/35 [=====] - 0s 7ms/step - loss: 0.0213 - accuracy:
 0.9933 - val_loss: 2.8260 - val_accuracy: 0.5607
 Epoch 24/50
 35/35 [=====] - 0s 7ms/step - loss: 0.0255 - accuracy:
 0.9933 - val_loss: 4.4797 - val_accuracy: 0.4696
 Epoch 25/50
 35/35 [=====] - 0s 7ms/step - loss: 0.0236 - accuracy:
 0.9915 - val_loss: 4.2527 - val_accuracy: 0.4143
 Epoch 26/50
 35/35 [=====] - 0s 7ms/step - loss: 0.0223 - accuracy:
 0.9920 - val_loss: 3.5158 - val_accuracy: 0.4429
 Epoch 27/50
 35/35 [=====] - 0s 7ms/step - loss: 0.0245 - accuracy:
 0.9929 - val_loss: 3.9560 - val_accuracy: 0.4661
 Epoch 28/50
 35/35 [=====] - 0s 7ms/step - loss: 0.0238 - accuracy:
 0.9902 - val_loss: 4.4557 - val_accuracy: 0.3893
 Epoch 29/50
 35/35 [=====] - 0s 8ms/step - loss: 0.0702 - accuracy:
 0.9772 - val_loss: 3.5628 - val_accuracy: 0.3839
 Epoch 30/50
 35/35 [=====] - 0s 8ms/step - loss: 0.0165 - accuracy:
 0.9951 - val_loss: 3.8458 - val_accuracy: 0.4089
 Epoch 31/50
 35/35 [=====] - 0s 8ms/step - loss: 0.0080 - accuracy:

0.9978 - val_loss: 3.8509 - val_accuracy: 0.4339
 Epoch 32/50
 35/35 [=====] - 0s 7ms/step - loss: 0.0091 - accuracy:
 0.9964 - val_loss: 3.8585 - val_accuracy: 0.4786
 Epoch 33/50
 35/35 [=====] - 0s 6ms/step - loss: 0.0146 - accuracy:
 0.9960 - val_loss: 4.4246 - val_accuracy: 0.3554
 Epoch 34/50
 35/35 [=====] - 0s 7ms/step - loss: 0.0075 - accuracy:
 0.9969 - val_loss: 4.4920 - val_accuracy: 0.3911
 Epoch 35/50
 35/35 [=====] - 0s 7ms/step - loss: 0.0072 - accuracy:
 0.9982 - val_loss: 3.6941 - val_accuracy: 0.4232
 Epoch 36/50
 35/35 [=====] - 0s 7ms/step - loss: 0.0244 - accuracy:
 0.9933 - val_loss: 2.6108 - val_accuracy: 0.5000
 Epoch 37/50
 35/35 [=====] - 0s 7ms/step - loss: 0.0115 - accuracy:
 0.9969 - val_loss: 3.3635 - val_accuracy: 0.5750
 Epoch 38/50
 35/35 [=====] - 0s 7ms/step - loss: 0.0519 - accuracy:
 0.9853 - val_loss: 5.5903 - val_accuracy: 0.2554
 Epoch 39/50
 35/35 [=====] - 0s 7ms/step - loss: 0.0369 - accuracy:
 0.9906 - val_loss: 3.8724 - val_accuracy: 0.4018
 Epoch 40/50
 35/35 [=====] - 0s 7ms/step - loss: 0.0122 - accuracy:
 0.9964 - val_loss: 4.3779 - val_accuracy: 0.4250
 Epoch 41/50
 35/35 [=====] - 0s 7ms/step - loss: 0.0124 - accuracy:
 0.9973 - val_loss: 3.4232 - val_accuracy: 0.4893
 Epoch 42/50
 35/35 [=====] - 0s 7ms/step - loss: 0.0095 - accuracy:
 0.9969 - val_loss: 4.3362 - val_accuracy: 0.3804
 Epoch 43/50
 35/35 [=====] - 0s 8ms/step - loss: 0.0089 - accuracy:
 0.9978 - val_loss: 3.9718 - val_accuracy: 0.4911
 Epoch 44/50
 35/35 [=====] - 0s 7ms/step - loss: 0.0058 - accuracy:
 0.9987 - val_loss: 3.5679 - val_accuracy: 0.5018
 Epoch 45/50
 35/35 [=====] - 0s 7ms/step - loss: 0.0074 - accuracy:
 0.9982 - val_loss: 4.0037 - val_accuracy: 0.4607
 Epoch 46/50
 35/35 [=====] - 0s 7ms/step - loss: 0.0030 - accuracy:
 0.9991 - val_loss: 4.6531 - val_accuracy: 0.3982
 Epoch 47/50
 35/35 [=====] - 0s 8ms/step - loss: 0.0076 - accuracy:

```

0.9982 - val_loss: 5.2379 - val_accuracy: 0.3571
Epoch 48/50
35/35 [=====] - 0s 7ms/step - loss: 0.0082 - accuracy:
0.9973 - val_loss: 4.3685 - val_accuracy: 0.4357
Epoch 49/50
35/35 [=====] - 0s 7ms/step - loss: 0.0155 - accuracy:
0.9964 - val_loss: 4.8508 - val_accuracy: 0.3804
Epoch 50/50
35/35 [=====] - 0s 6ms/step - loss: 0.0079 - accuracy:
0.9982 - val_loss: 5.0355 - val_accuracy: 0.3750

```

```

[ ]: # best val accuracy: 72.32
     # use checkpoint to save the best val accuracy model
     # adjust learning rate for slow convergence

```

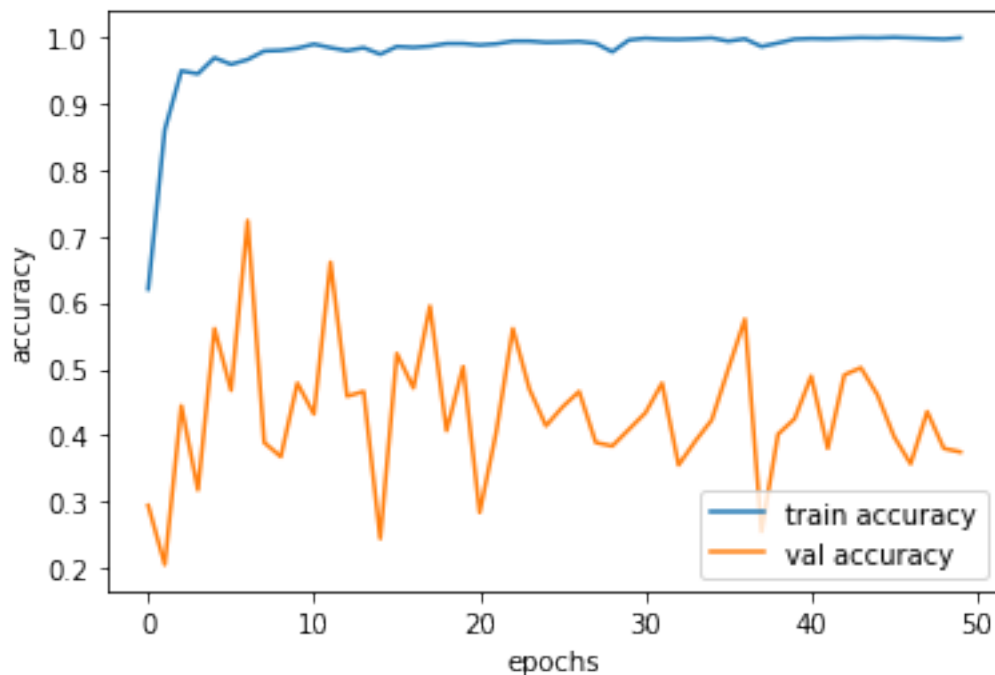
0.6 Plot the results

```

[103]: epochs = list(range(50))
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

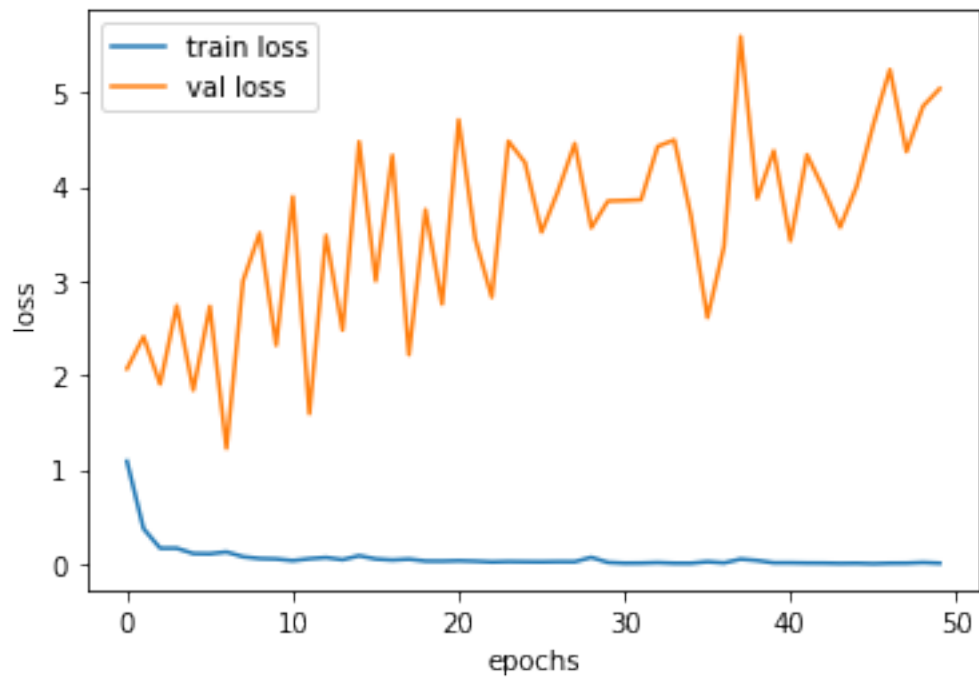
plt.plot(epochs, acc, label='train accuracy')
plt.plot(epochs, val_acc, label='val accuracy')
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.legend()
plt.show()

```




```
[104]: loss = history.history['loss']
val_loss = history.history['val_loss']

plt.plot(epochs, loss, label='train loss')
plt.plot(epochs, val_loss, label='val loss')
plt.xlabel('epochs')
plt.ylabel('loss')
plt.legend()
plt.show()
```



```
[ ]:
```

```
[ ]:
```