Voice Emotion Classification

Import Modules

In [5]:
```
#importing the libery
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import librosa
import librosa.display
from IPython.display import Audio
import warnings
warnings.filterwarnings('ignore')
```

Load the DataSet

In [6]:
```
#loading the dataset
paths = []
labels = []
import os
for dirname, _, filenames in os.walk('input'):
    for filename in filenames:
        paths.append(os.path.join(dirname, filename))

        label = filename.split('_')[-1]

        label = label.split('.')[0]
        labels.append(label.lower())

print("Dataset Loaded")
```

```
Dataset Loaded
```

In [8]:
```
#check labels of the dataset
labels[:5]
```

Out[8]:
```
['fear', 'fear', 'fear', 'fear', 'fear']
```

In [9]:
```
#check dataset
df = pd.DataFrame()
df['speech'] = paths
df['label']= labels
df.head()
```

Out[9]:

	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

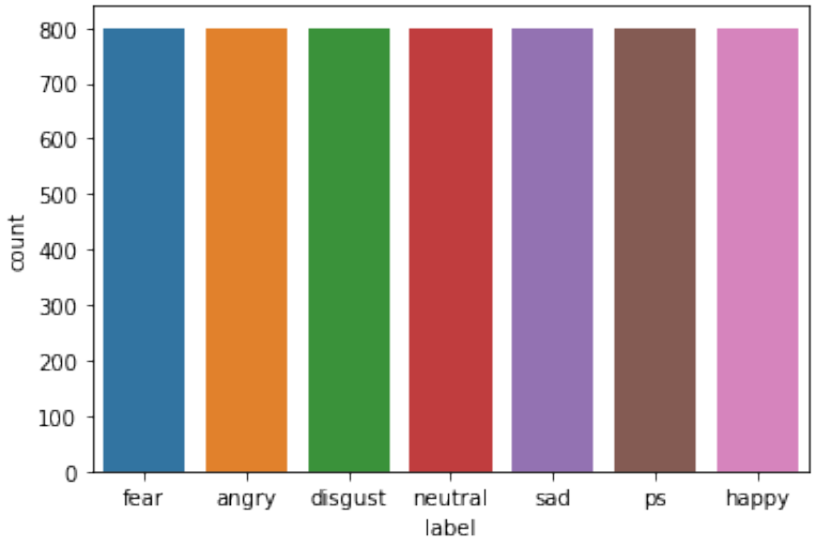
In [10]:
```
#count the value label
df['label'].value_counts()
```

Out[10]:
```
fear       800
angry      800
disgust    800
neutral    800
sad        800
ps         800
happy      800
Name: label, dtype: int64
```

Exploratory Data Analysis

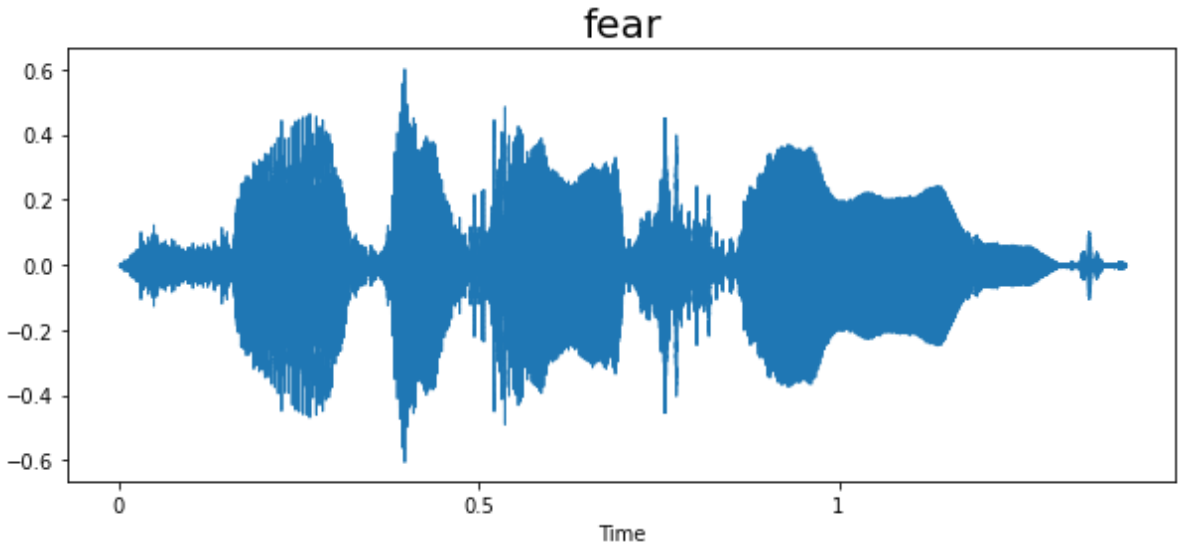
In [11]:
```
#count the value label
sns.countplot(df['label'])
```

Out[11]: <AxesSubplot:xlabel='label', ylabel='count'>



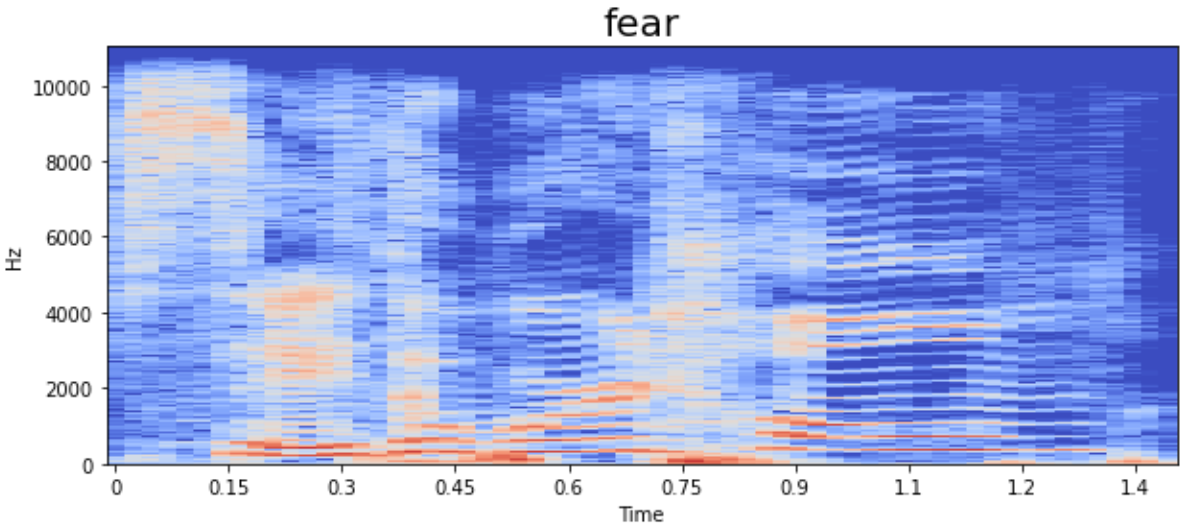
In [12]:
```
#plot waveplot
def waveplot(data, sr, emotion):
    plt.figure(figsize = (10, 4))
    plt.title(emotion, size= 20)
    librosa.display.waveshow(data, sr=sr)
    plt.show()

def spectogram(data, sr, emotion):
    x = librosa.stft(data)
    xdb = librosa.amplitude_to_db(abs(x))
    plt.figure(figsize = (10, 4))
    plt.title(emotion, size= 20)
    librosa.display.specshow(xdb, sr=sr, x_axis='time', y_axis='hz'
```

In [13]:
```
#plot for fear
emotion = 'fear'
path = np.array(df['speech'][df['label']==emotion])[0]
data, sampling_rate = librosa.load(path)
waveplot(data, sampling_rate, emotion)
spectogram(data, sampling_rate, emotion)
Audio(path)
```

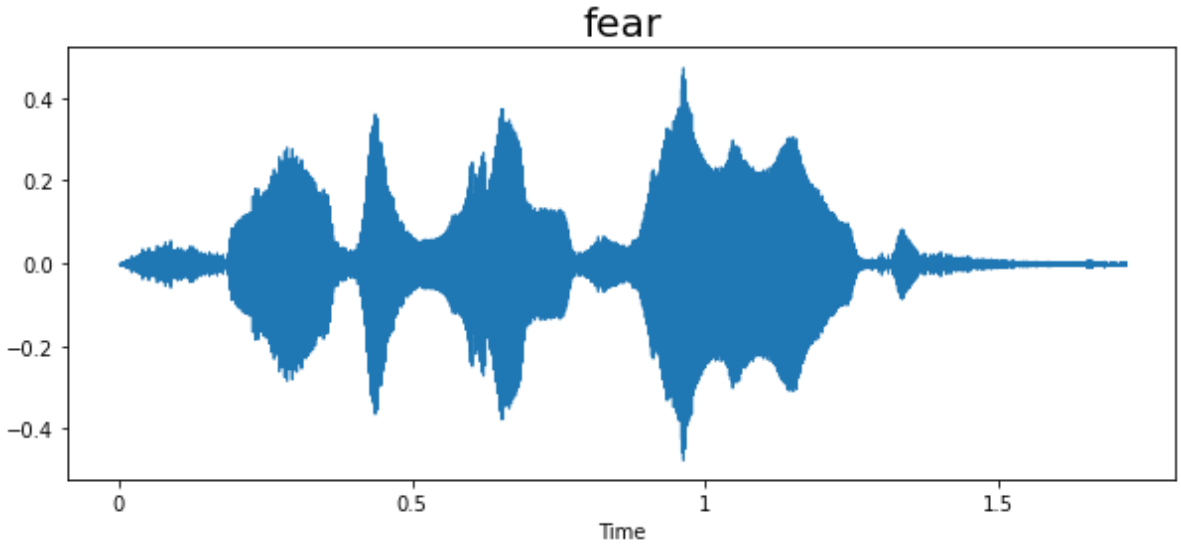


Out[13]:

-00:00

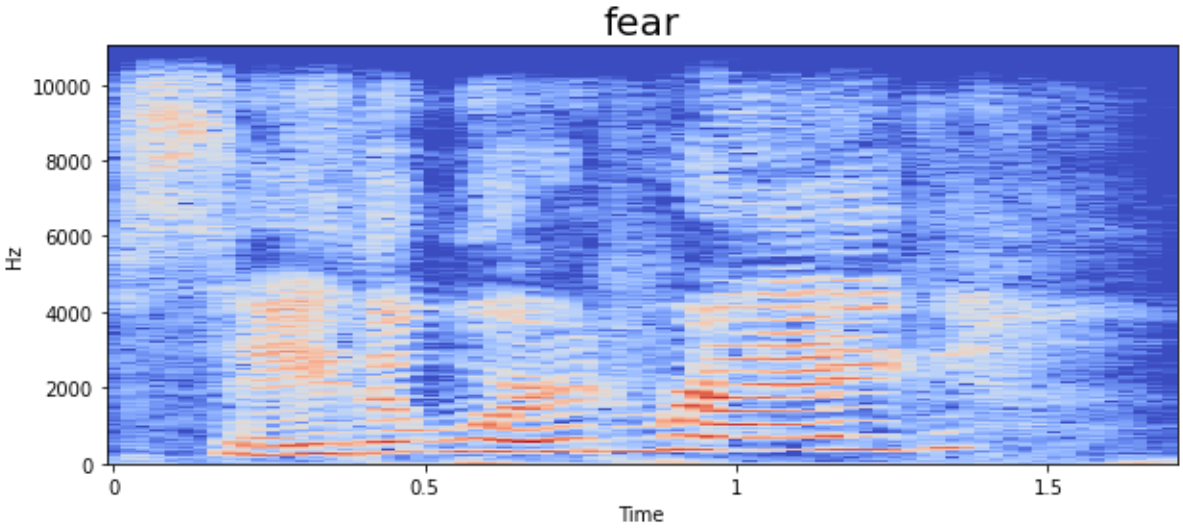


In [30]:
```
emotion = 'fear'
path = np.array(df['speech'][df['label']==emotion])[10]
data, sampling_rate = librosa.load(path)
waveplot(data, sampling_rate, emotion)
spectogram(data, sampling_rate, emotion)
Audio(path)
```

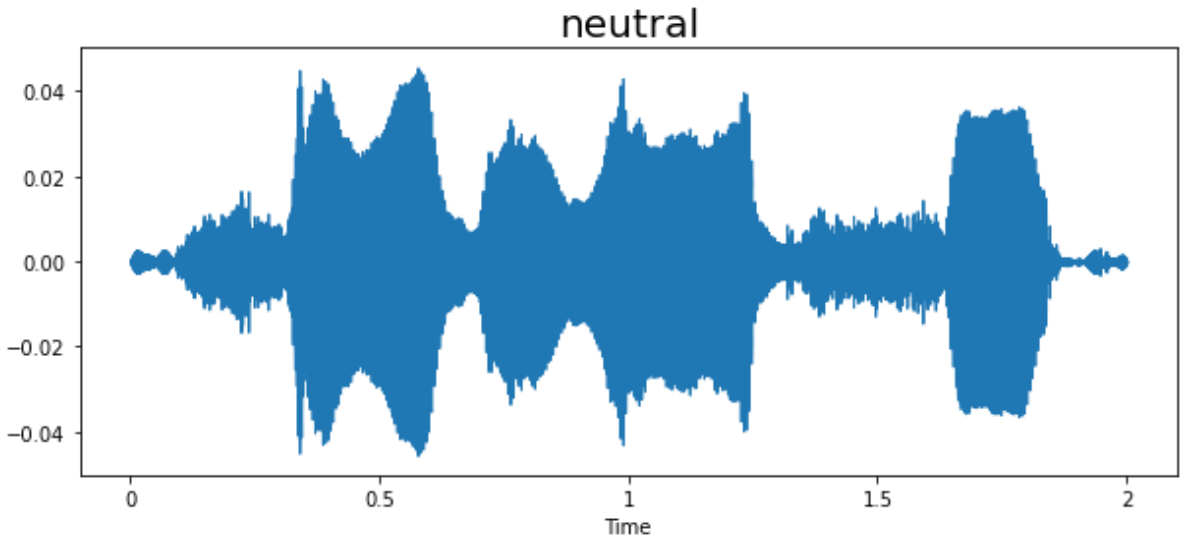


Out[30]:

-00:00

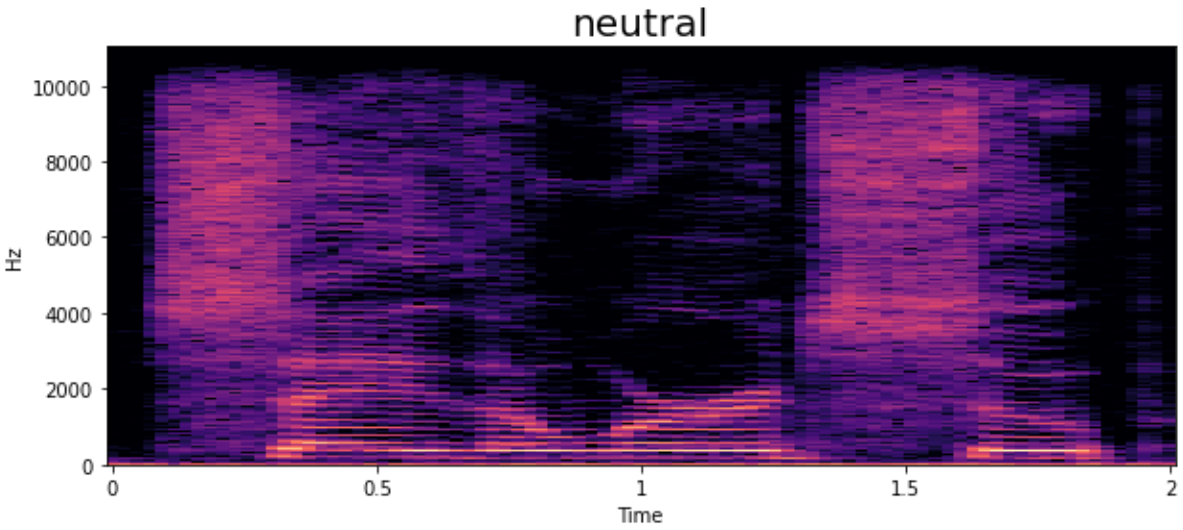


In [14]:
```
#plot neutral
emotion = 'neutral'
path = np.array(df['speech'][df['label']==emotion])[0]
data, sampling_rate = librosa.load(path)
waveplot(data, sampling_rate, emotion)
spectogram(data, sampling_rate, emotion)
Audio(path)
```



Out[14]:

-00:00



Feature Extraction

In [15]:
```
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, a
    return mfcc
```

In [16]:
```
extract_mfcc(df['speech'][0])
```

Out[16]:
```
array([-287.13037   ,   87.756935  ,   -4.139177  ,   24.081968  ,
       -16.696724  ,   12.970632  ,   10.522444  ,   -1.1463214 ,
        -0.73337686,   12.855532  ,  -19.147291  ,   -6.418063  ,
         4.9657683 ,   -2.6571155 ,  -10.655444  ,    4.9578815 ,
       -14.55586   ,   15.37587   ,   18.444935  ,   23.878317  ,
        31.495146  ,   17.326372  ,   -4.7648373 ,    1.7432437 ,
       -12.009847  ,    7.34574   ,   -3.2051265 ,   -7.171453  ,
       -11.410634  ,   -2.001994  ,   -5.610964  ,    4.5321946 ,
       -11.396625  ,   -8.892363  ,   -3.7391376 ,    4.8819685 ,
        -1.5599903 ,    2.465447  ,   11.59915   ,   11.042192  ]
,
      dtype=float32)
```

In [17]:
```
x_mfcc = df['speech'].apply(lambda x: extract_mfcc(x))
```

In [18]:
```
x_mfcc
```

Out[18]:
```
0       [-287.13037, 87.756935, -4.139177, 24.081968, ...
1       [-350.0836, 37.654167, -6.2928553, 17.09615, 4...
2       [-341.78152, 56.153652, -16.617884, 23.219698,...
3       [-309.17456, 24.854897, -8.00109, 10.065497, -...
4       [-347.12918, 49.69155, -27.524876, 22.730288, ...
                              ...
5595    [-376.5839, 63.9598, -3.0598662, 11.498796, -2...
5596    [-316.5801, 43.54606, -9.336959, -0.198444, -5...
5597    [-359.7638, 81.01536, -18.355762, 5.3012295, -...
5598    [-354.38315, 103.432144, -15.916284, -10.30884...
5599    [-391.15958, 56.44471, -1.0464002, 0.9587419, ...
Name: speech, Length: 5600, dtype: object
```

In [19]:
```
X= [x for x in x_mfcc]
X = np.array(X)
X.shape
```

Out[19]:
```
(5600, 40)
```

In [20]:
```
X = np.expand_dims(X, -1)
X.shape
```

Out[20]:
```
(5600, 40, 1)
```

In [21]:
```
from sklearn.preprocessing import OneHotEncoder
enc = OneHotEncoder()
y = enc.fit_transform(df[['label']])
```

In [22]:
```
y =y.toarray()
```

In [23]:
```
y.shape
```

Out[23]:
```
(5600, 7)
```

Create a LSTM model

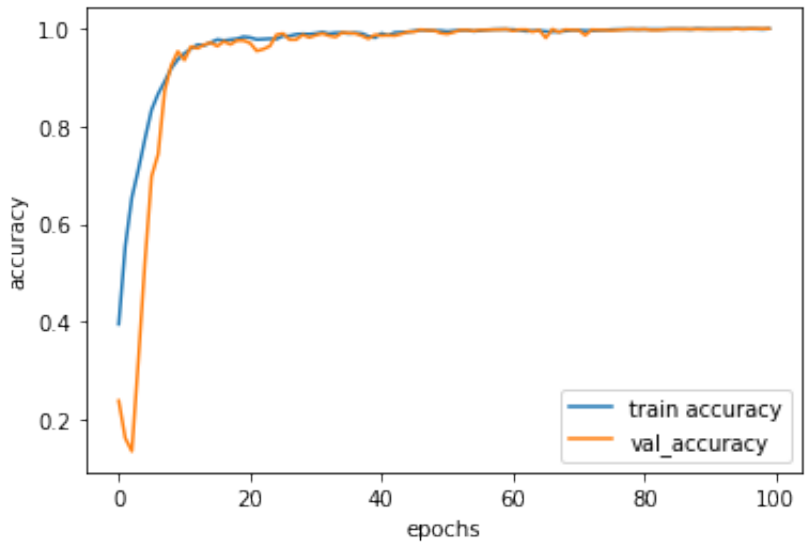
In [24]:
```
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout

model = Sequential([
    LSTM(123, return_sequences = False, input_shape=(40, 1)),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(32, activation = 'relu'),
    Dropout(0.2),
    Dense(7, activation='softmax'),

])

model.compile(loss = "categorical_crossentropy", optimizer = 'adam'
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 123)               61500
_________________________________________________________________
dense_3 (Dense)              (None, 64)                7936
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_4 (Dense)              (None, 32)                2080
_________________________________________________________________
dropout_3 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_5 (Dense)              (None, 7)                 231
=================================================================
Total params: 71,747
Trainable params: 71,747
Non-trainable params: 0
_________________________________________________________________
```

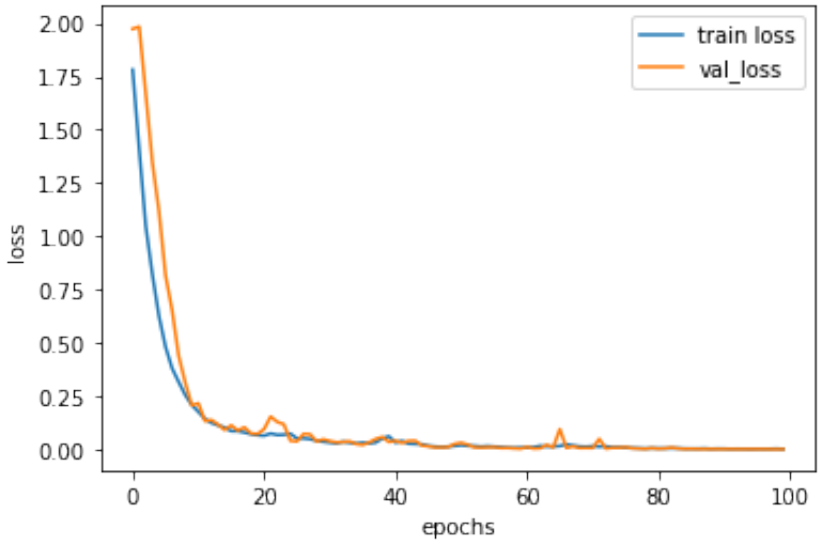
In [25]:
```
#train the model
history=model.fit(X, y, validation_split=0.2, epochs=100, batch_siz
```

```
Epoch 10/100
9/9 [==============================] - 0s 11ms/step - loss: 0.2081
- accuracy: 0.9371 - val_loss: 0.2065 - val_accuracy: 0.9536
Epoch 11/100
9/9 [==============================] - 0s 11ms/step - loss: 0.1764
- accuracy: 0.9491 - val_loss: 0.2161 - val_accuracy: 0.9357
Epoch 12/100
9/9 [==============================] - 0s 12ms/step - loss: 0.1429
- accuracy: 0.9594 - val_loss: 0.1331 - val_accuracy: 0.9634
Epoch 13/100
9/9 [==============================] - 0s 11ms/step - loss: 0.1240
- accuracy: 0.9672 - val_loss: 0.1376 - val_accuracy: 0.9598
Epoch 14/100
9/9 [==============================] - 0s 12ms/step - loss: 0.1127
- accuracy: 0.9658 - val_loss: 0.1165 - val_accuracy: 0.9670
Epoch 15/100
9/9 [==============================] - 0s 10ms/step - loss: 0.1023
- accuracy: 0.9717 - val_loss: 0.0911 - val_accuracy: 0.9714
Epoch 16/100
9/9 [==============================] - 0s 10ms/step - loss: 0.0860
```

In [26]:
```
epochs = list(range(100))
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()
```



In [27]:
```
epochs = list(range(100))
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()
```



In [ ]: