Importing Libraries

```
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')
```

Load the Dataset

```
paths = []
labels = []
import os
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())
print('Dataset is loaded')
```

```
Dataset is loaded
```

```
paths[:5]
```

```
['/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']
```

```
labels[:5]
```

```
['fear', 'fear', 'fear', 'fear', 'fear']
```

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

	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

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

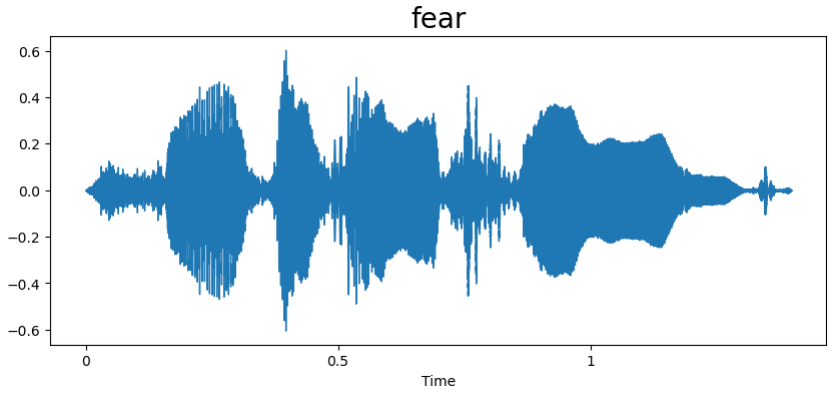
```
fear       800
angry      800
disgust    800
neutral    800
sad        800
ps         800
happy      800
Name: label, dtype: int64
```

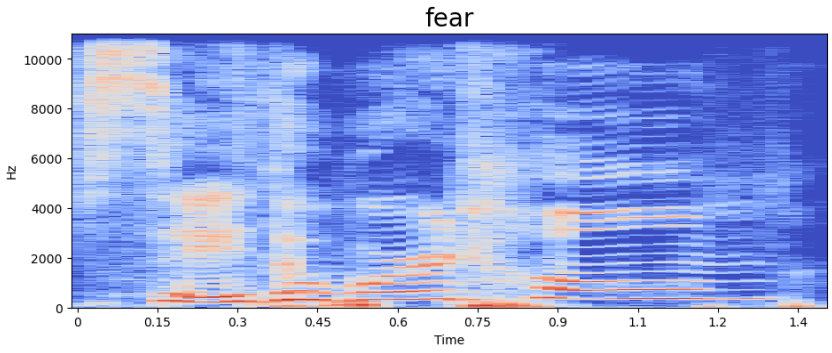
Exploratory Data Analysis

```
def waveshow(data, sr, emotion):
    plt.figure(figsize=(10,4))
    plt.title(emotion, size=20)
    librosa.display.waveshow(data, sr=sr)

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.show()
```

```
emotion = 'fear'
path = df['speech'][df['label']==emotion][0]
data, sampling_rate = librosa.load(path)
waveshow(data, sampling_rate, emotion)
spectogram(data, sampling_rate, emotion)
Audio(path)
```

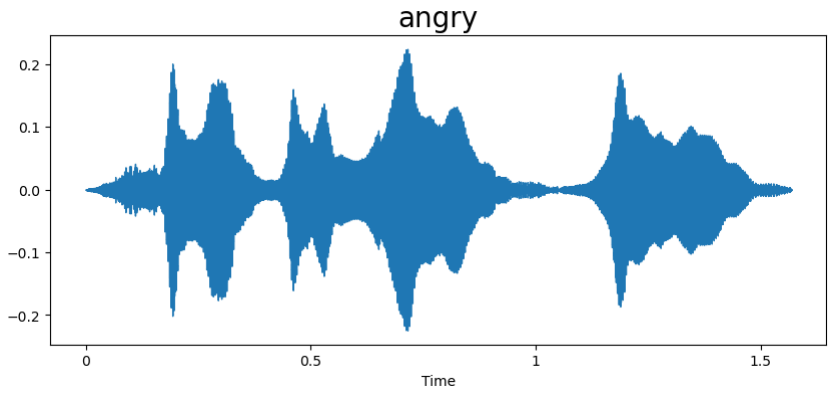


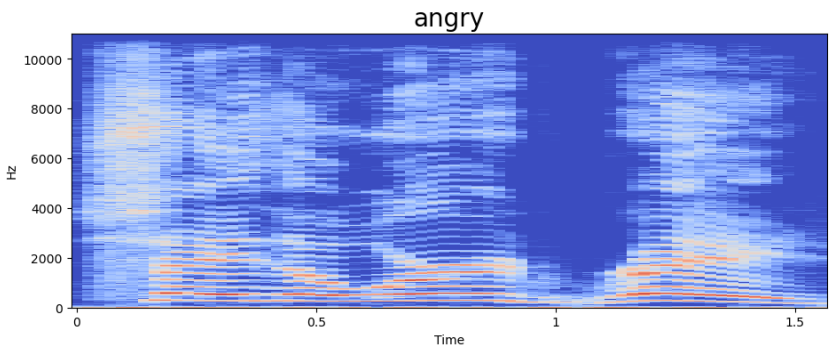


0:00 / 0:01

```
emotion = 'angry'
path = np.array(df['speech'][df['label']==emotion])[0]
data, sampling_rate = librosa.load(path)
waveshow(data, sampling_rate, emotion)
```

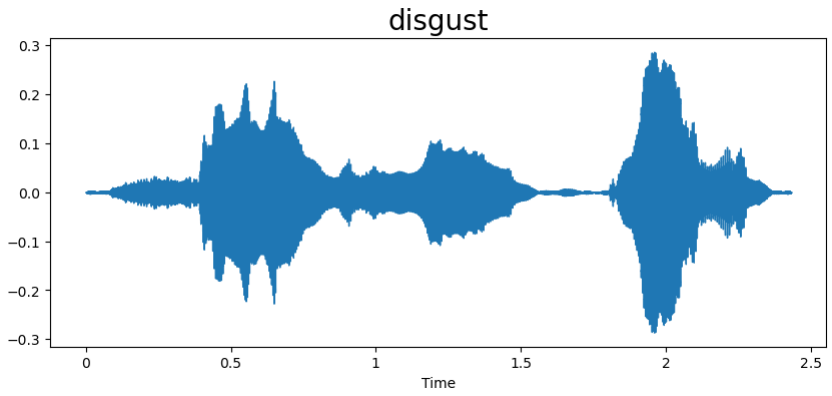
```
spectogram(data, sampling_rate, emotion)
Audio(path)
```

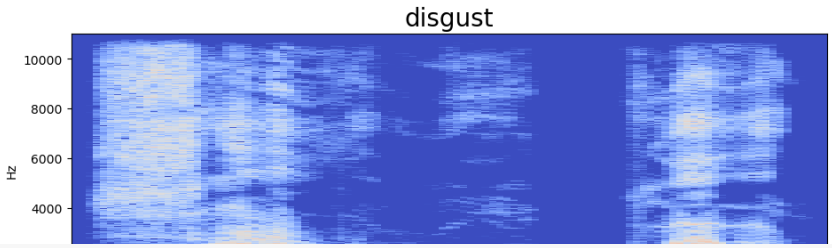




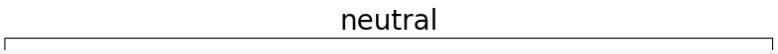
0:00 / 0:01

```
emotion = 'disgust'
path = np.array(df['speech'][df['label']==emotion])[0]
data, sampling_rate = librosa.load(path)
waveshow(data, sampling_rate, emotion)
spectogram(data, sampling_rate, emotion)
Audio(path)
```

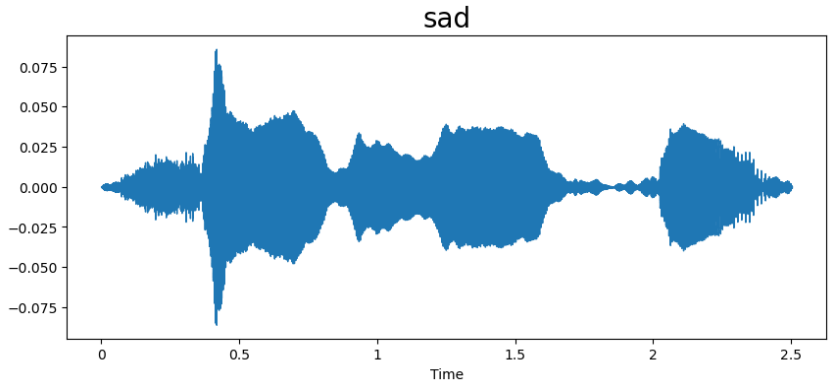


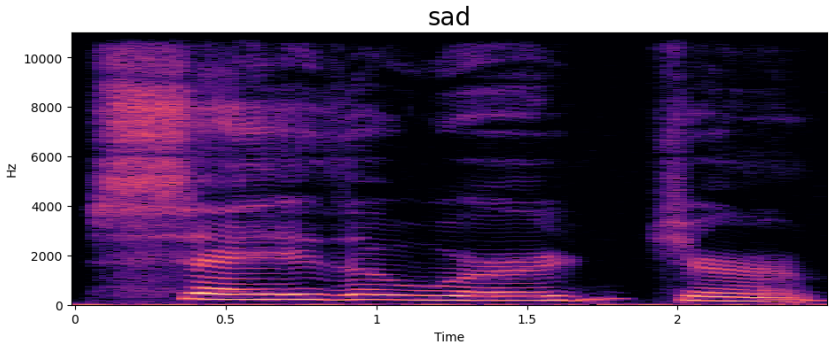



```
emotion = 'neutral'
path = np.array(df['speech'][df['label']==emotion])[0]
data, sampling_rate = librosa.load(path)
waveshow(data, sampling_rate, emotion)
spectogram(data, sampling_rate, emotion)
Audio(path)
```



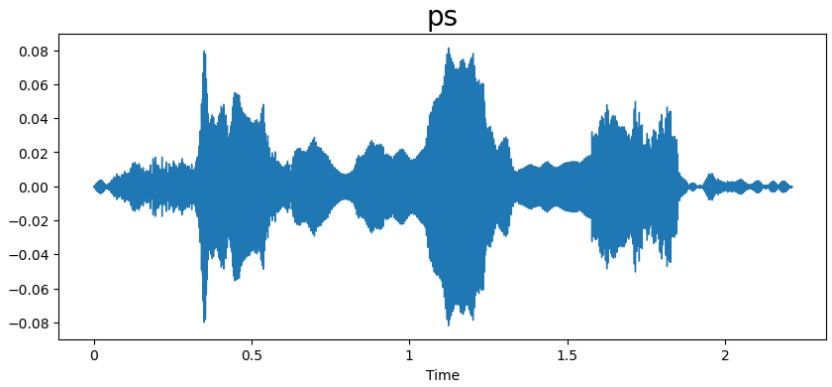

```
emotion = 'sad'
path = np.array(df['speech'][df['label']==emotion])[0]
data, sampling_rate = librosa.load(path)
waveshow(data, sampling_rate, emotion)
spectogram(data, sampling_rate, emotion)
Audio(path)
```

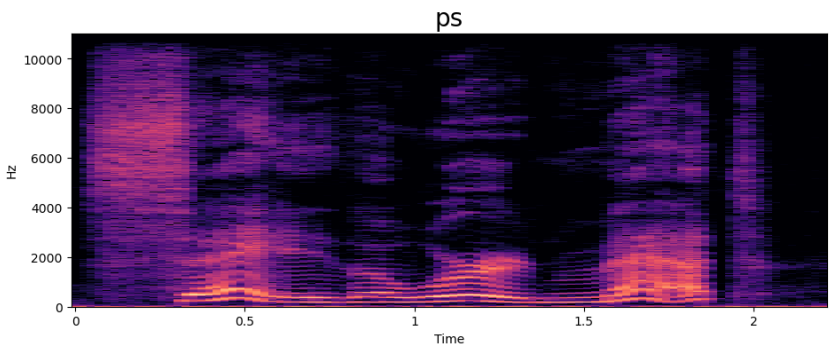




0:00 / 0:02

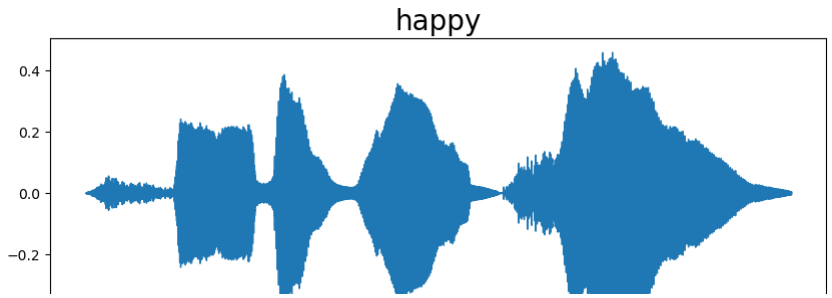
```
emotion = 'ps'
path = np.array(df['speech'][df['label']==emotion])[0]
data, sampling_rate = librosa.load(path)
waveshow(data, sampling_rate, emotion)
spectogram(data, sampling_rate, emotion)
Audio(path)
```





0:00 / 0:02

```
emotion = 'happy'
path = np.array(df['speech'][df['label']==emotion])[0]
data, sampling_rate = librosa.load(path)
waveshow(data, sampling_rate, emotion)
spectogram(data, sampling_rate, emotion)
Audio(path)
```



Extracting the Feature

Time

```
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
```

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

```
array([-285.73727  ,   85.78295  ,   -2.1689117,   22.12553  ,
       -14.757396 ,   11.051346 ,   12.412449 ,   -3.000262 ,
         1.0844985,   11.078272 ,  -17.41966  ,   -8.093213 ,
         6.5879736,   -4.2209525,   -9.15508  ,    3.52148  ,
       -13.186381 ,   14.078853 ,   19.66973  ,   22.725618 ,
        32.57464  ,   16.325033 ,   -3.8427281,    0.896297 ,
       -11.239263 ,    6.653462 ,   -2.5883696,   -7.7140164,
       -10.941657 ,   -2.4007547,   -5.2812867,    4.271157 ,
       -11.202216 ,   -9.024621 ,   -3.666985 ,    4.8697433,
        -1.6027987,    2.5600514,   11.454374 ,   11.233449 ],
      dtype=float32)
```

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

```
x_mfcc
```

```
0       [-285.73727, 85.78295, -2.1689117, 22.12553, -...
1       [-348.34332, 35.193233, -3.841328, 14.658875, ...
2       [-340.11435, 53.796444, -14.267782, 20.884031,...
3       [-306.63422, 21.259708, -4.4110823, 6.4871554,...
4       [-344.7548, 46.329193, -24.171413, 19.392921, ...
                              ...
5595    [-374.3952, 60.864998, 0.0250591, 8.431059, -2...
5596    [-313.9648, 39.847843, -5.6493053, -3.867575, ...
5597    [-357.54886, 77.88605, -15.224756, 2.194633, -...
5598    [-353.1474, 101.68391, -14.175898, -12.037376,...
5599    [-389.4595, 54.042767, 1.3469977, -1.4258989, ...
Name: speech, Length: 5600, dtype: object
```

```
x = [x for x in x_mfcc]
x = np.array(x)
x.shape
```

```
(5600, 40)
```

```
## Input split
x = np.expand_dims(x, -1)
x.shape
```

```
(5600, 40, 1)
```

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

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

```
y.shape
```

```
(5600, 7)
```

Create LSTM Model

```
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', metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 123)               61500

 dense (Dense)               (None, 64)                7936

 dropout (Dropout)           (None, 64)                0

 dense_1 (Dense)             (None, 32)                2080

 dropout_1 (Dropout)         (None, 32)                0

 dense_2 (Dense)             (None, 7)                 231

=================================================================
Total params: 71,747
Trainable params: 71,747
Non-trainable params: 0
_________________________________________________________________
```

```
# Training of the model
history = model.fit(x, y, validation_split=0.2, epochs=100, batch_size=512, shuffle=True)
```

```
Epoch 1/100
9/9 [==============================] - 7s 78ms/step - loss: 1.8162 - accuracy: 0.3364 - val_loss: 1.9743 - val_accuracy: 0.1732
Epoch 2/100
9/9 [==============================] - 0s 15ms/step - loss: 1.4929 - accuracy: 0.4848 - val_loss: 1.9196 - val_accuracy: 0.2036
Epoch 3/100
9/9 [==============================] - 0s 16ms/step - loss: 1.1340 - accuracy: 0.6330 - val_loss: 1.7172 - val_accuracy: 0.2295
Epoch 4/100
9/9 [==============================] - 0s 15ms/step - loss: 0.8212 - accuracy: 0.7254 - val_loss: 1.3802 - val_accuracy: 0.3545
Epoch 5/100
9/9 [==============================] - 0s 14ms/step - loss: 0.6316 - accuracy: 0.7815 - val_loss: 1.0565 - val_accuracy: 0.6286
Epoch 6/100
9/9 [==============================] - 0s 14ms/step - loss: 0.4860 - accuracy: 0.8377 - val_loss: 0.7609 - val_accuracy: 0.7107
Epoch 7/100
9/9 [==============================] - 0s 15ms/step - loss: 0.3757 - accuracy: 0.8779 - val_loss: 0.6118 - val_accuracy: 0.7688
Epoch 8/100
9/9 [==============================] - 0s 14ms/step - loss: 0.3082 - accuracy: 0.9009 - val_loss: 0.5348 - val_accuracy: 0.8027
Epoch 9/100
9/9 [==============================] - 0s 14ms/step - loss: 0.2517 - accuracy: 0.9183 - val_loss: 0.3633 - val_accuracy: 0.8938
Epoch 10/100
9/9 [==============================] - 0s 14ms/step - loss: 0.2365 - accuracy: 0.9203 - val_loss: 0.3635 - val_accuracy: 0.8911
Epoch 11/100
9/9 [==============================] - 0s 15ms/step - loss: 0.1974 - accuracy: 0.9413 - val_loss: 0.2431 - val_accuracy: 0.9295
Epoch 12/100
9/9 [==============================] - 0s 14ms/step - loss: 0.1628 - accuracy: 0.9513 - val_loss: 0.1847 - val_accuracy: 0.9491
Epoch 13/100
9/9 [==============================] - 0s 15ms/step - loss: 0.1395 - accuracy: 0.9585 - val_loss: 0.1626 - val_accuracy: 0.9545
Epoch 14/100
9/9 [==============================] - 0s 14ms/step - loss: 0.1214 - accuracy: 0.9683 - val_loss: 0.1708 - val_accuracy: 0.9527
Epoch 15/100
9/9 [==============================] - 0s 14ms/step - loss: 0.1171 - accuracy: 0.9674 - val_loss: 0.2242 - val_accuracy: 0.9312
Epoch 16/100
9/9 [==============================] - 0s 14ms/step - loss: 0.1118 - accuracy: 0.9683 - val_loss: 0.1338 - val_accuracy: 0.9598
Epoch 17/100
9/9 [==============================] - 0s 14ms/step - loss: 0.0954 - accuracy: 0.9737 - val_loss: 0.1105 - val_accuracy: 0.9679
Epoch 18/100
```

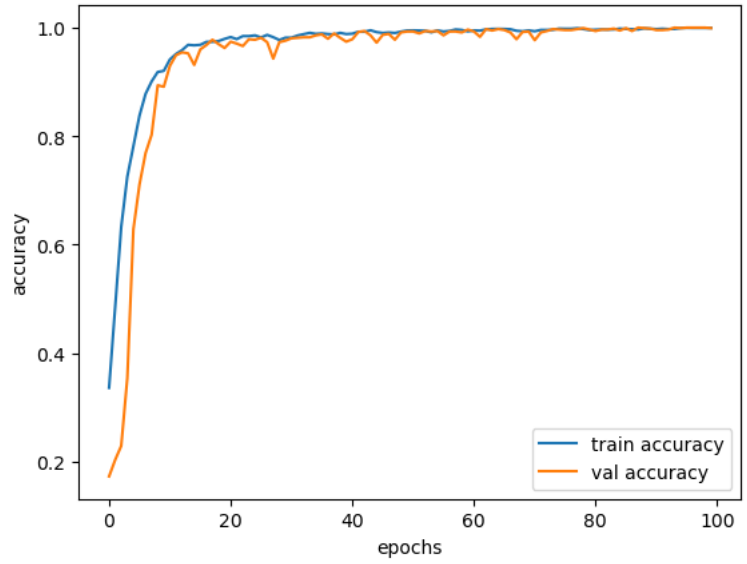
```
9/9 [==============================] - 0s 14ms/step - loss: 0.0888 - accuracy: 0.9739 - val_loss: 0.0772 - val_accuracy: 0.9777
Epoch 19/100
9/9 [==============================] - 0s 14ms/step - loss: 0.0841 - accuracy: 0.9752 - val_loss: 0.0923 - val_accuracy: 0.9696
Epoch 20/100
9/9 [==============================] - 0s 14ms/step - loss: 0.0751 - accuracy: 0.9792 - val_loss: 0.1228 - val_accuracy: 0.9625
Epoch 21/100
9/9 [==============================] - 0s 14ms/step - loss: 0.0758 - accuracy: 0.9826 - val_loss: 0.0897 - val_accuracy: 0.9741
Epoch 22/100
9/9 [==============================] - 0s 20ms/step - loss: 0.0751 - accuracy: 0.9788 - val_loss: 0.1051 - val_accuracy: 0.9705
Epoch 23/100
9/9 [==============================] - 0s 18ms/step - loss: 0.0648 - accuracy: 0.9844 - val_loss: 0.1051 - val_accuracy: 0.9661
Epoch 24/100
9/9 [==============================] - 0s 17ms/step - loss: 0.0616 - accuracy: 0.9844 - val_loss: 0.0693 - val_accuracy: 0.9786
Epoch 25/100
9/9 [==============================] - 0s 18ms/step - loss: 0.0548 - accuracy: 0.9857 - val_loss: 0.0671 - val_accuracy: 0.9777
Epoch 26/100
9/9 [==============================] - 0s 15ms/step - loss: 0.0587 - accuracy: 0.9819 - val_loss: 0.0598 - val_accuracy: 0.9812
Epoch 27/100
9/9 [==============================] - 0s 14ms/step - loss: 0.0493 - accuracy: 0.9866 - val_loss: 0.0731 - val_accuracy: 0.9732
Epoch 28/100
9/9 [==============================] - 0s 15ms/step - loss: 0.0604 - accuracy: 0.9824 - val_loss: 0.1773 - val_accuracy: 0.9429
Epoch 29/100
9/9 [==============================] - 0s 14ms/step - loss: 0.0765 - accuracy: 0.9772 - val_loss: 0.0944 - val_accuracy: 0.9732
```

Double-click (or enter) to edit

Result Plotting

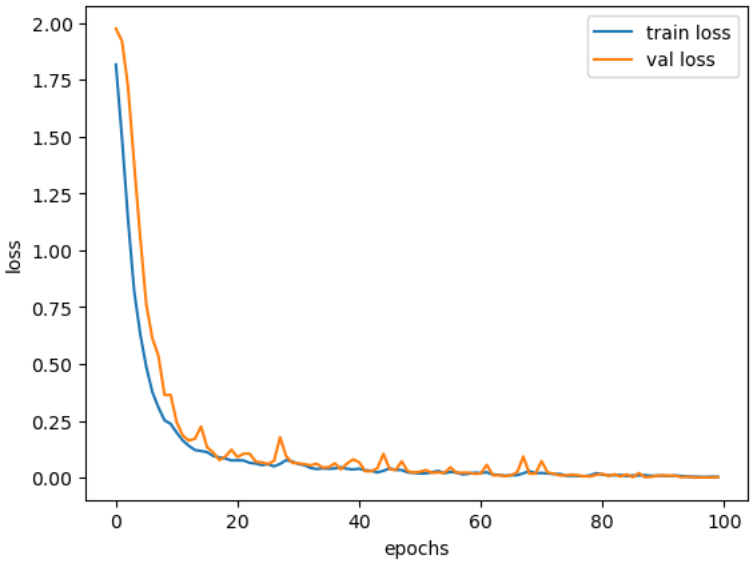
```
epochs = list(range(100))
acc = history.history['accuracy']
val_acc =  history.history['val_accuracy']

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()
```




```
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()
```


train loss
val loss
loss
epochs
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
20
40
60
80
100