EX NO: 12 **FEATURE EXTRACTION FROM AUDIO**

DATE:

AIM:

To perform feature extraction from audio in Python.

ALGORITHM:

1. Import Libraries:
   - I. pydub for audio handling.
   - II. librosa for audio processing and feature extraction.
   - III. numpy for numerical operations.
   - IV. matplotlib.pyplot for plotting.
2. Convert MP3 to WAV:
   - I. Define mp3_to_wav(mp3_file, wav_file):
     - i. Load MP3 (mp3_file) and export to WAV (wav_file).
3. Extract Features:
   - I. Define extract_features(audio_file, sr=44100):
     - i. Load audio (audio_file) using librosa.load().
     - ii. Extract:
       1. MFCCs (librosa.feature.mfcc()).
       2. Chroma (librosa.feature.chroma_stft()).
       3. Mel-scaled spectrogram (librosa.feature.melspectrogram()).
       4. Spectral contrast (librosa.feature.spectral_contrast()).
       5. Tonnetz (librosa.feature.tonnetz()).
4. Plot Features:
   - I. Call extract_features() with audio file.
   - II. Plot features using matplotlib.pyplot:
     - i. MFCCs, Chroma, Mel-scaled Spectrogram, Spectral Contrast, Tonnetz.
     - ii. Display with color bars and titles.
   - III. Show plots with plt.show().

PROGRAM:

```
!pip install librosa
!pip install pydub
```

#mp3 to wav file converter

```
from pydub import AudioSegment

def mp3_to_wav(mp3_file, wav_file):
    # Load the MP3 file
    audio = AudioSegment.from_mp3(mp3_file)

    # Export the audio to WAV format
    audio.export(wav_file, format="wav")

# Replace 'input.mp3' and 'output.wav' with your input MP3 file path and desired output WAV file path
mp3_file = '/content/sample-6s.mp3'
wav_file = '/content/output.wav'
```

```
mp3_to_wav(mp3_file, wav_file)
```

#Feature extraction from audio

```
import librosa
import numpy as np
import matplotlib.pyplot as plt

def extract_features(audio_file, sr=44100):
    # Load the audio file
    y, sr = librosa.load(audio_file, sr=sr)

    # Extract features
    # MFCCs (Mel-frequency cepstral coefficients)
    mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=13)

    # Chroma feature
    chroma = librosa.feature.chroma_stft(y=y, sr=sr)

    # Mel-scaled spectrogram
    mel_spectrogram = librosa.feature.melspectrogram(y=y, sr=sr)

    # Spectral contrast
    spectral_contrast = librosa.feature.spectral_contrast(y=y, sr=sr)

    # Tonnetz
    tonnetz = librosa.feature.tonnetz(y=librosa.effects.harmonic(y), sr=sr)

    return mfccs, chroma, mel_spectrogram, spectral_contrast, tonnetz

# Example usage with your uploaded audio file path
audio_file = '/content/output.wav'  # Replace with the actual audio file path
mfccs, chroma, mel_spectrogram, spectral_contrast, tonnetz = extract_features(audio_file)

# Plot each feature separately
plt.figure(figsize=(10, 6))

# MFCCs
plt.subplot(2, 3, 1)
plt.imshow(mfccs, aspect='auto', origin='lower', cmap='viridis')
plt.colorbar()
plt.title('MFCCs')

# Chroma feature
plt.subplot(2, 3, 2)
plt.imshow(chroma, aspect='auto', origin='lower', cmap='viridis')
plt.colorbar()
plt.title('Chroma')

# Mel-scaled spectrogram
plt.subplot(2, 3, 3)
plt.imshow(mel_spectrogram, aspect='auto', origin='lower', cmap='viridis')
plt.colorbar()
```

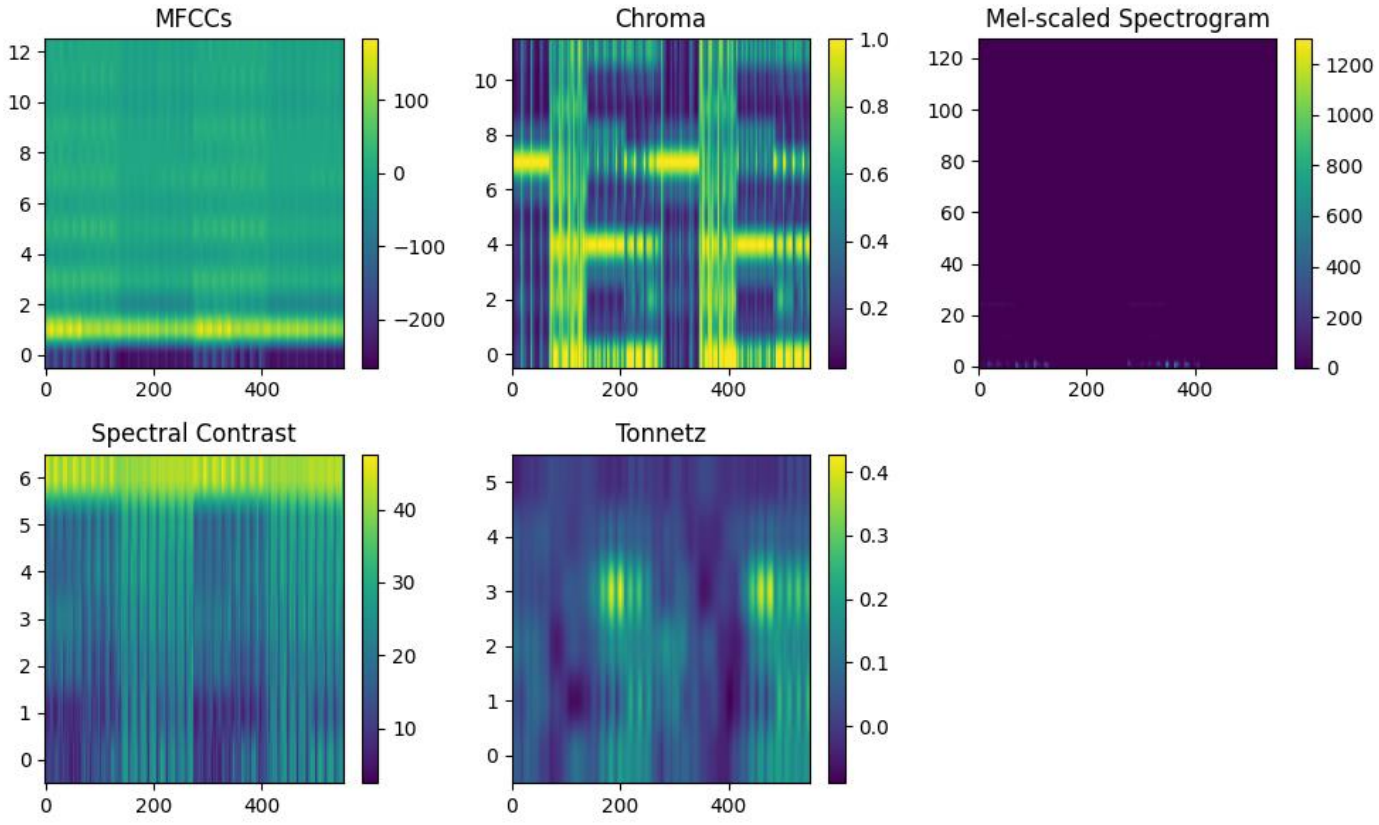
```
plt.title('Mel-scaled Spectrogram')

# Spectral contrast
plt.subplot(2, 3, 4)
plt.imshow(spectral_contrast, aspect='auto', origin='lower', cmap='viridis')
plt.colorbar()
plt.title('Spectral Contrast')

# Tonnetz
plt.subplot(2, 3, 5)
plt.imshow(tonnetz, aspect='auto', origin='lower', cmap='viridis')
plt.colorbar()
plt.title('Tonnetz')

plt.tight_layout()
plt.show()
```

OUTPUT:



RESULT:

Thus, the implementation of feature extraction if audio using python is executed successfully and output verified.

EX NO: 13 **SPEECH CLASSIFICATION**

DATE:

AIM:

To implement speech classification using Python.

ALGORITHM:

1. Setup and Load Data:
   - Import necessary libraries and set seed.
   - Define `DATASET_PATH` and download the 'mini_speech_commands.zip' dataset if not present.
   - Load audio commands and prepare training and validation datasets (`train_ds`, `val_ds`) using `tf.keras.utils.audio_dataset_from_directory`.

2. Prepare Spectrograms:
   - Define `get_spectrogram(waveform)` to convert waveforms to spectrograms.
   - Apply `get_spectrogram()` to examples from `train_ds` and visualize.

3. Build and Train Model:
   - Create a CNN model using `tf.keras.Sequential`.
   - Include layers: normalization, resizing, convolutional, pooling, dropout, and dense.
   - Compile model with optimizer, loss, and metrics.
   - Train model on `train_ds` with validation on `val_ds`.
   - Use early stopping to prevent overfitting.

4. Evaluate Model:
   - Evaluate trained model on test spectrogram dataset (`test_ds`) and generate predictions (`y_pred`).
   - Create confusion matrix (`confusion_mtx`) and plot it.

PROGRAM:

```
import os
import pathlib

import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import tensorflow as tf

from tensorflow.keras import layers
from tensorflow.keras import models
from IPython import display

# Set the seed value for experiment reproducibility.
seed = 42
tf.random.set_seed(seed)
np.random.seed(seed)
DATASET_PATH = 'data/mini_speech_commands'

data_dir = pathlib.Path(DATASET_PATH)
```

```
if not data_dir.exists():
  tf.keras.utils.get_file(
      'mini_speech_commands.zip',
      origin="http://storage.googleapis.com/download.tensorflow.org/data/mini_speech_commands.zip",
      extract=True,
      cache_dir='.', cache_subdir='data')
commands = np.array(tf.io.gfile.listdir(str(data_dir)))
commands = commands[(commands != 'README.md') & (commands != '.DS_Store')]
print('Commands:', commands)
train_ds, val_ds = tf.keras.utils.audio_dataset_from_directory(
    directory=data_dir,
    batch_size=64,
    validation_split=0.2,
    seed=0,
    output_sequence_length=16000,
    subset='both')

label_names = np.array(train_ds.class_names)
print()
print("label names:", label_names)
train_ds.element_spec
```

OUTPUT:

```
Downloading data from
http://storage.googleapis.com/download.tensorflow.org/data/mini_speech_commands.zip
182082353/182082353 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
Commands: ['down' 'go' 'no' 'left' 'right' 'up' 'yes' 'stop']
Found 8000 files belonging to 8 classes.
Using 6400 files for training.
Using 1600 files for validation.

label names: ['down' 'go' 'left' 'no' 'right' 'stop' 'up' 'yes']
(TensorSpec(shape=(None, 16000, None), dtype=tf.float32, name=None),
 TensorSpec(shape=(None,), dtype=tf.int32, name=None))
```

PROGRAM:

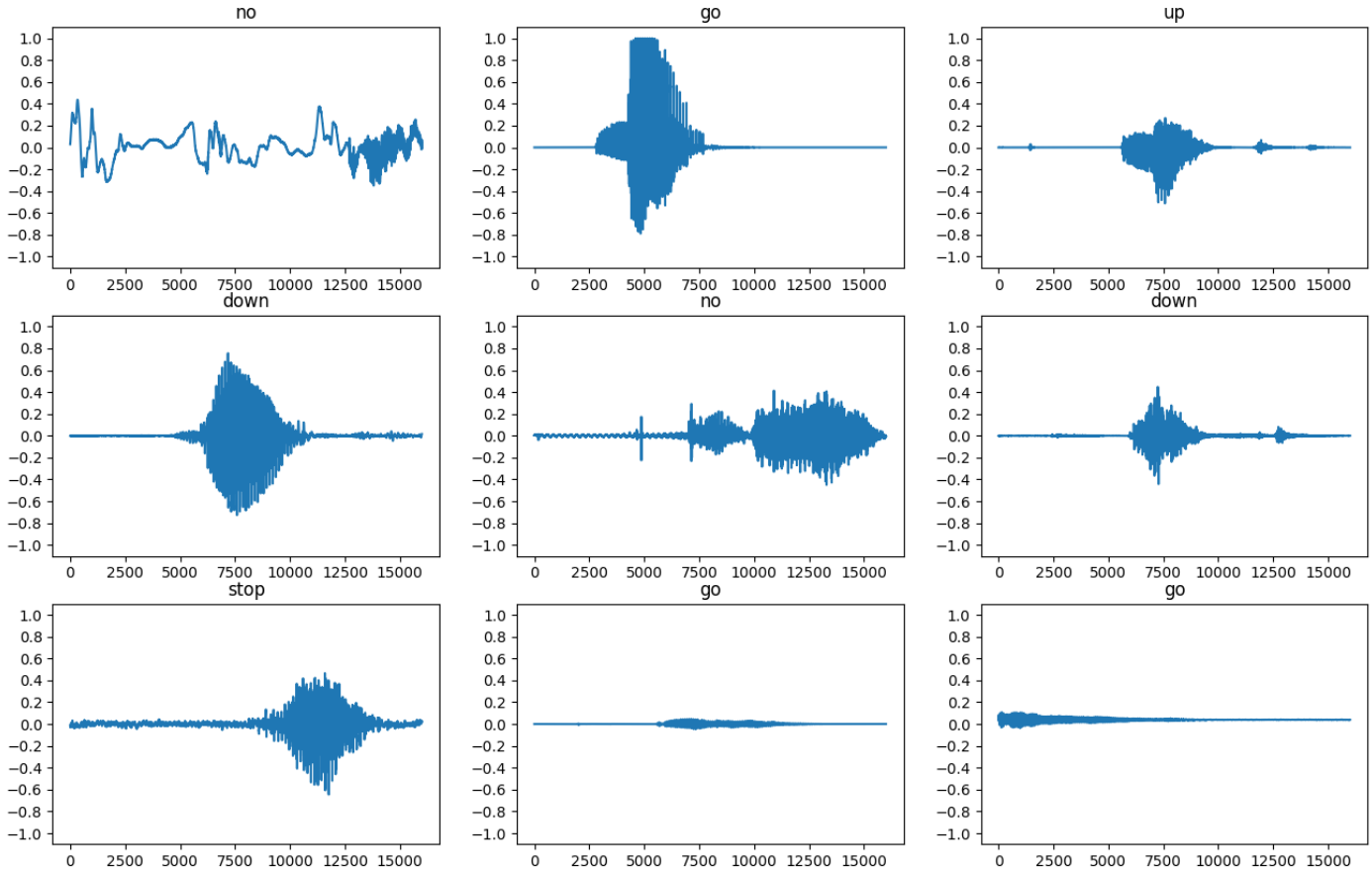
```
def squeeze(audio, labels):
  audio = tf.squeeze(audio, axis=-1)
  return audio, labels

train_ds = train_ds.map(squeeze, tf.data.AUTOTUNE)
val_ds = val_ds.map(squeeze, tf.data.AUTOTUNE)
test_ds = val_ds.shard(num_shards=2, index=0)
val_ds = val_ds.shard(num_shards=2, index=1)
for example_audio, example_labels in train_ds.take(1):
  print(example_audio.shape)
  print(example_labels.shape)
label_names[[1,1,3,0]]
plt.figure(figsize=(16, 10))
rows = 3
cols = 3
n = rows * cols
```

```
for i in range(n):
  plt.subplot(rows, cols, i+1)
  audio_signal = example_audio[i]
  plt.plot(audio_signal)
  plt.title(label_names[example_labels[i]])
  plt.yticks(np.arange(-1.2, 1.2, 0.2))
  plt.ylim([-1.1, 1.1])
```

OUTPUT:

```
(64, 16000)
(64,)
array(['go', 'go', 'no', 'down'], dtype='<U5')
```



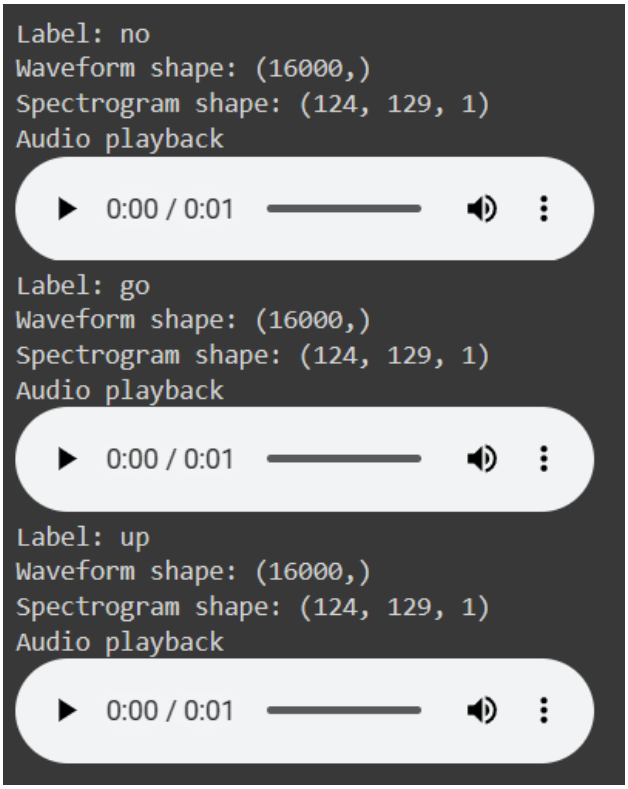
PROGRAM:

```
def get_spectrogram(waveform):
  # Convert the waveform to a spectrogram via a STFT.
  spectrogram = tf.signal.stft(
      waveform, frame_length=255, frame_step=128)
  # Obtain the magnitude of the STFT.
  spectrogram = tf.abs(spectrogram)
  # Add a `channels` dimension, so that the spectrogram can be used
  # as image-like input data with convolution layers (which expect
  # shape (`batch_size`, `height`, `width`, `channels`).
  spectrogram = spectrogram[..., tf.newaxis]
  return spectrogram
for i in range(3):
  label = label_names[example_labels[i]]
  waveform = example_audio[i]
```

```
  spectrogram = get_spectrogram(waveform)

  print('Label:', label)
  print('Waveform shape:', waveform.shape)
  print('Spectrogram shape:', spectrogram.shape)
  print('Audio playback')
  display.display(display.Audio(waveform, rate=16000))
```

OUTPUT:



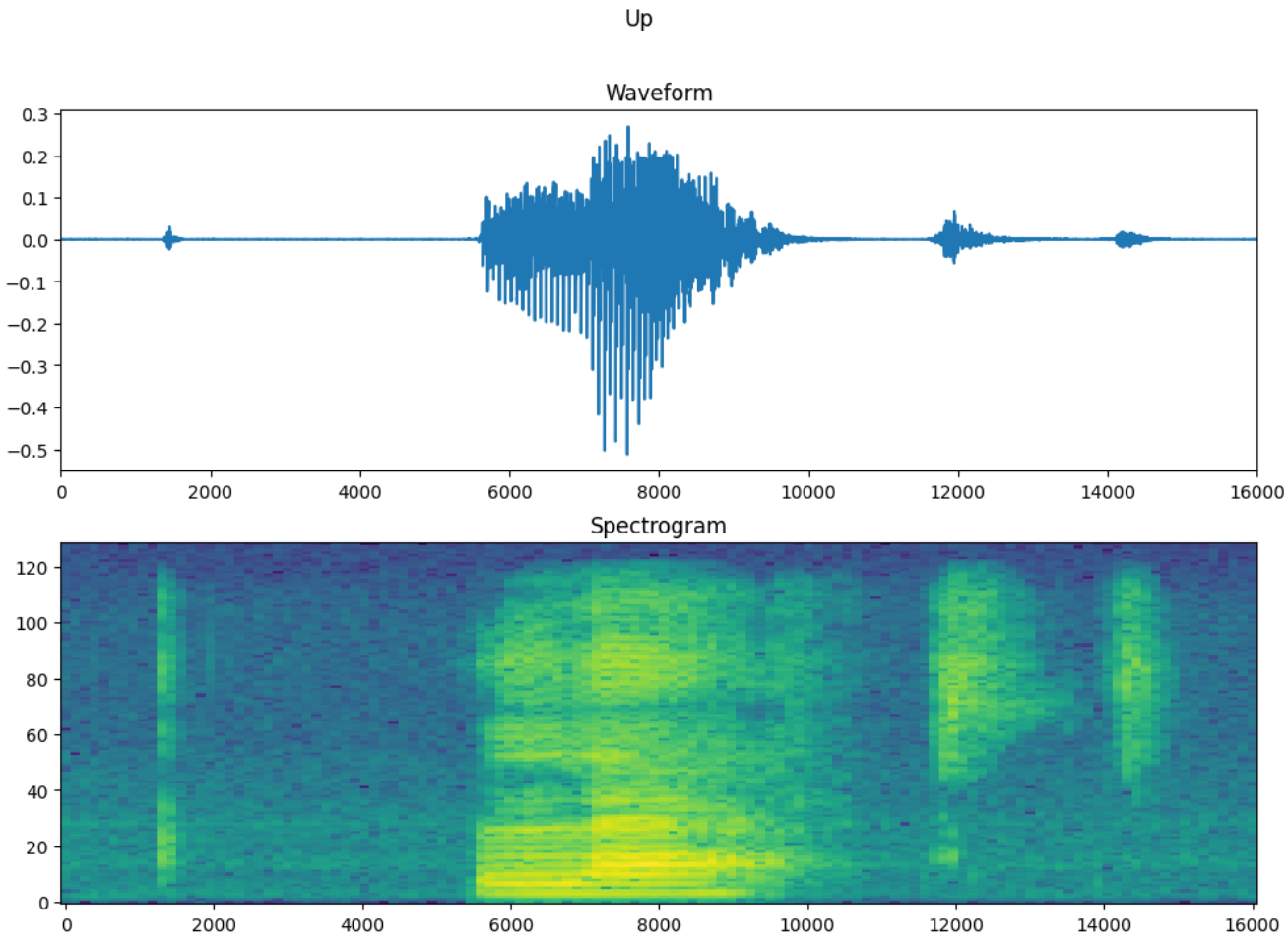
PROGRAM:

```
def plot_spectrogram(spectrogram, ax):
  if len(spectrogram.shape) > 2:
    assert len(spectrogram.shape) == 3
    spectrogram = np.squeeze(spectrogram, axis=-1)
  # Convert the frequencies to log scale and transpose, so that the time is
  # represented on the x-axis (columns).
  # Add an epsilon to avoid taking a log of zero.
  log_spec = np.log(spectrogram.T + np.finfo(float).eps)
  height = log_spec.shape[0]
  width = log_spec.shape[1]
  X = np.linspace(0, np.size(spectrogram), num=width, dtype=int)
  Y = range(height)
  ax.pcolormesh(X, Y, log_spec)
fig, axes = plt.subplots(2, figsize=(12, 8))
timescale = np.arange(waveform.shape[0])
axes[0].plot(timescale, waveform.numpy())
axes[0].set_title('Waveform')
axes[0].set_xlim([0, 16000])

plot_spectrogram(spectrogram.numpy(), axes[1])
```

```
axes[1].set_title('Spectrogram')
plt.suptitle(label.title())
plt.show()
```

OUTPUT:



PROGRAM:

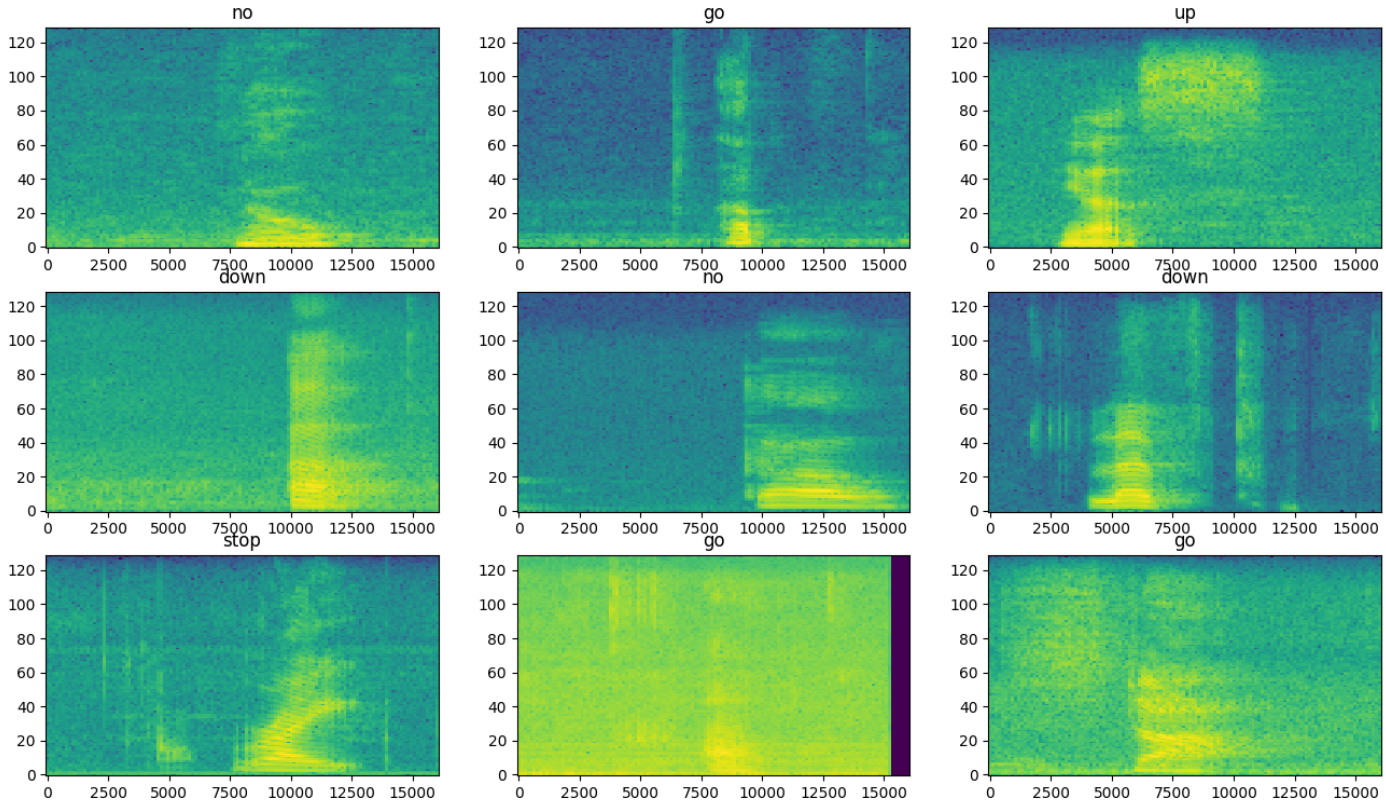
```
def make_spec_ds(ds):
  return ds.map(
      map_func=lambda audio,label: (get_spectrogram(audio), label),
      num_parallel_calls=tf.data.AUTOTUNE)
train_spectrogram_ds = make_spec_ds(train_ds)
val_spectrogram_ds = make_spec_ds(val_ds)
test_spectrogram_ds = make_spec_ds(test_ds)
for example_spectrograms, example_spect_labels in train_spectrogram_ds.take(1):
  break
rows = 3
cols = 3
n = rows*cols
fig, axes = plt.subplots(rows, cols, figsize=(16, 9))

for i in range(n):
    r = i // cols
    c = i % cols
    ax = axes[r][c]
    plot_spectrogram(example_spectrograms[i].numpy(), ax)
```

```
    ax.set_title(label_names[example_spect_labels[i].numpy()])

plt.show()
```

OUTPUT:



PROGRAM:

```
train_spectrogram_ds = train_spectrogram_ds.cache().shuffle(10000).prefetch(tf.data.AUTOTUNE)
val_spectrogram_ds = val_spectrogram_ds.cache().prefetch(tf.data.AUTOTUNE)
test_spectrogram_ds = test_spectrogram_ds.cache().prefetch(tf.data.AUTOTUNE)
input_shape = example_spectrograms.shape[1:]
print('Input shape:', input_shape)
num_labels = len(label_names)

# Instantiate the `tf.keras.layers.Normalization` layer.
norm_layer = layers.Normalization()
# Fit the state of the layer to the spectrograms
# with `Normalization.adapt`.
norm_layer.adapt(data=train_spectrogram_ds.map(map_func=lambda spec, label: spec))

model = models.Sequential([
    layers.Input(shape=input_shape),
    # Downsample the input.
    layers.Resizing(32, 32),
    # Normalize.
    norm_layer,
    layers.Conv2D(32, 3, activation='relu'),
    layers.Conv2D(64, 3, activation='relu'),
    layers.MaxPooling2D(),
    layers.Dropout(0.25),
```

```
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(num_labels),
])

model.summary()
```

OUTPUT:

```
Input shape: (124, 129, 1)
Model: "sequential"
```

Layer (type)	Output Shape	Param #
resizing (Resizing)	(None, 32, 32, 1)	0
normalization (Normalization)	(None, 32, 32, 1)	3
conv2d (Conv2D)	(None, 30, 30, 32)	320
conv2d_1 (Conv2D)	(None, 28, 28, 64)	18,496
max_pooling2d (MaxPooling2D)	(None, 14, 14, 64)	0
dropout (Dropout)	(None, 14, 14, 64)	0
flatten (Flatten)	(None, 12544)	0
dense (Dense)	(None, 128)	1,605,760
dropout_1 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 8)	1,032

```
Total params: 1,625,611 (6.20 MB)
Trainable params: 1,625,608 (6.20 MB)
Non-trainable params: 3 (16.00 B)
```

PROGRAM:

```
model.compile(
    optimizer=tf.keras.optimizers.Adam(),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=['accuracy'],
)
EPOCHS = 10
history = model.fit(
    train_spectrogram_ds,
    validation_data=val_spectrogram_ds,
    epochs=EPOCHS,
    callbacks=tf.keras.callbacks.EarlyStopping(verbose=1, patience=2),
)
```

OUTPUT:

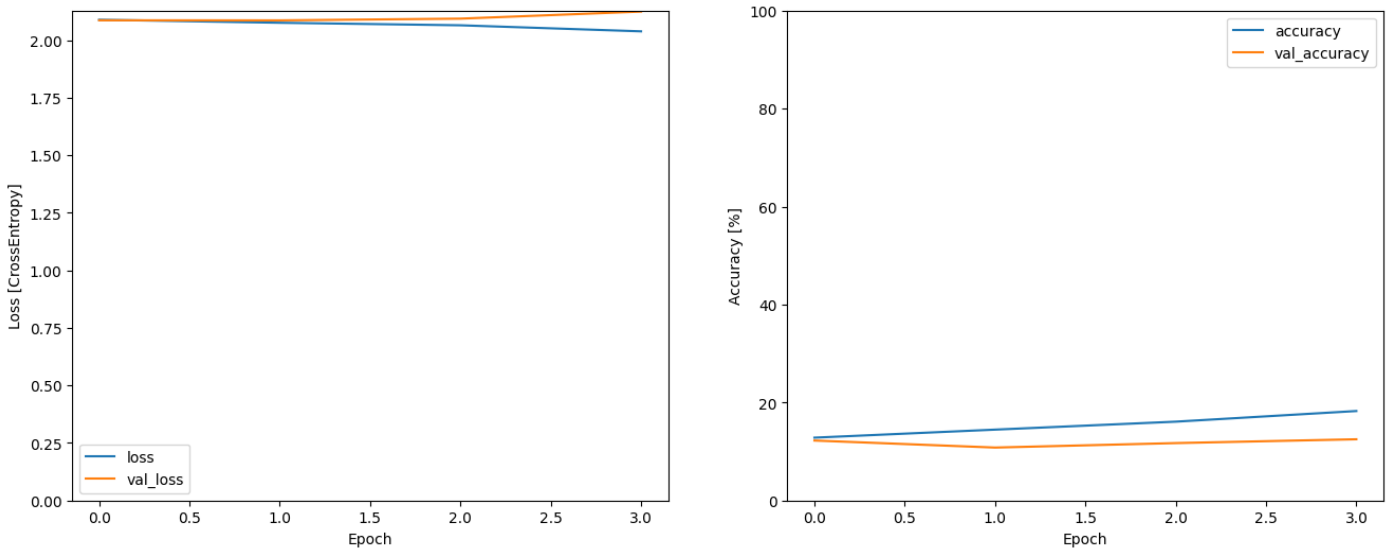
```
Epoch 1/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 30s 284ms/step - accuracy: 0.1325 - loss: 2.1041 - val_accuracy: 0.1224 - val_loss: 2.0859
Epoch 2/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 32s 323ms/step - accuracy: 0.1447 - loss: 2.0739 - val_accuracy: 0.1081 - val_loss: 2.0856
Epoch 3/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 39s 301ms/step - accuracy: 0.1707 - loss: 2.0622 - val_accuracy: 0.1172 - val_loss: 2.0931
Epoch 4/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 37s 259ms/step - accuracy: 0.1821 - loss: 2.0375 - val_accuracy: 0.1250 - val_loss: 2.1237
Epoch 4: early stopping
```

PROGRAM:

```
metrics = history.history
plt.figure(figsize=(16,6))
plt.subplot(1,2,1)
plt.plot(history.epoch, metrics['loss'], metrics['val_loss'])
plt.legend(['loss', 'val_loss'])
plt.ylim([0, max(plt.ylim())])
plt.xlabel('Epoch')
plt.ylabel('Loss [CrossEntropy]')

plt.subplot(1,2,2)
plt.plot(history.epoch, 100*np.array(metrics['accuracy']), 100*np.array(metrics['val_accuracy']))
plt.legend(['accuracy', 'val_accuracy'])
plt.ylim([0, 100])
plt.xlabel('Epoch')
plt.ylabel('Accuracy [%]')
```

OUTPUT:



PROGRAM:

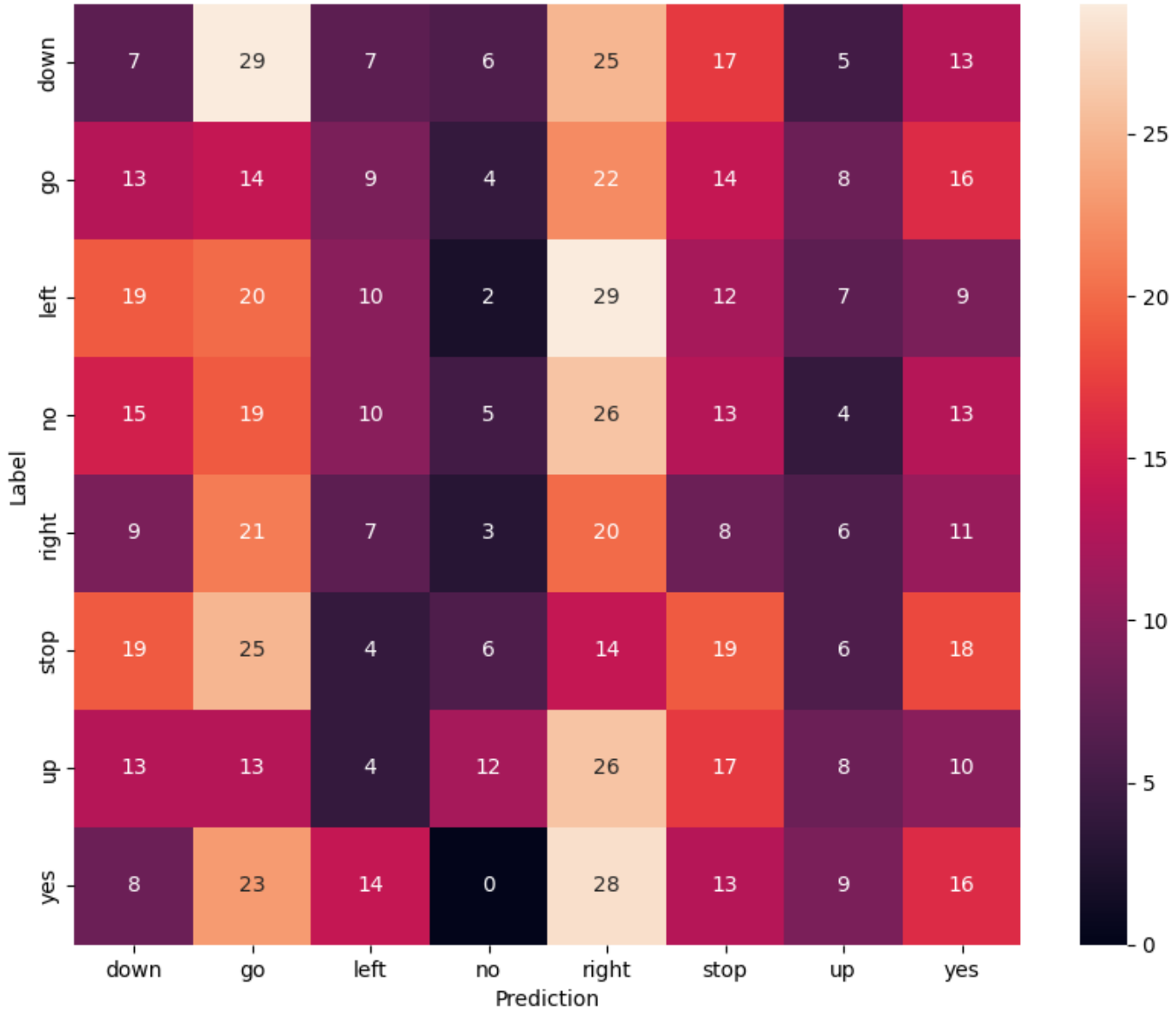
```
model.evaluate(test_spectrogram_ds, return_dict=True)

y_pred = model.predict(test_spectrogram_ds)
y_pred = tf.argmax(y_pred, axis=1)
y_true = tf.concat(list(test_spectrogram_ds.map(lambda s,lab: lab)), axis=0)
confusion_mtx = tf.math.confusion_matrix(y_true, y_pred)
```

```
plt.figure(figsize=(10, 8))
sns.heatmap(confusion_mtx,
            xticklabels=label_names,
            yticklabels=label_names,
            annot=True, fmt='g')
plt.xlabel('Prediction')
plt.ylabel('Label')
plt.show()
```

OUTPUT:

```
13/13 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3s 180ms/step - accuracy: 0.1159 - loss: 2.1145
{'accuracy': 0.11899038404226303, 'loss': 2.104218006134033}
```



RESULT:

Thus, the implementation of speech classification using python is executed successfully and output is verified.