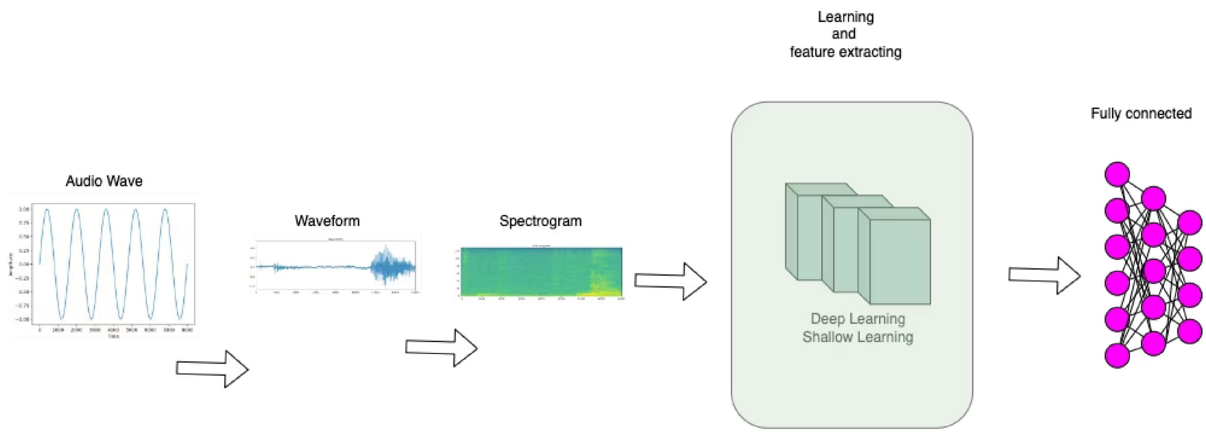
Audio Classification Using Deep Learning



Librosa is a Python package for analyzing and extracting features from audio signals. It is commonly used in tasks related to music and audio processing. Here's a brief overview of the librosa library:

Librosa is an open-source package for music and audio analysis. It provides tools for tasks such as:

Loading audio files

Extracting various audio features

Performing spectral analysis

Time-domain and frequency-domain manipulation

Displaying spectrograms and other visualizations

In [ ]:

```

```

In [1]:

```
### Install librosa package
pip install librosa
```

In [2]:

```
import matplotlib.pyplot as plt
%matplotlib inline
```

In [3]:

```
filename='dog_bark.wav'
```

In [4]:

```
### install ipython
pip install ipython
```

In [5]:

```
import IPython.display as ipd
```

`IPython.display` module to play an audio file specified by `filename`. The `ipd.Audio` class generates an audio player widget, facilitating seamless playback within an IPython environment, like Jupyter Notebooks

In [6]:
```
### Dog Sound ####
plt.figure(figsize=(14,5))
ipd.Audio(filename)
```

Out[6]:

0:00 / 0:00

```
<Figure size 1400x500 with 0 Axes>
```

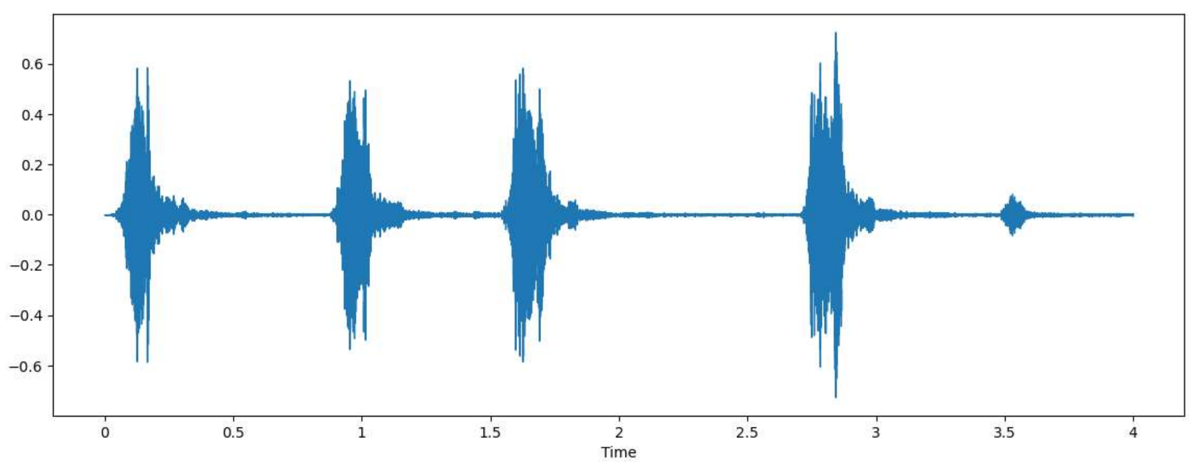
In [ ]:
```

```

In [7]:
```
import IPython.display as ipd
import librosa
import librosa.display
import numpy as np
```

In [8]:
```
### Dog Sound
plt.figure(figsize=(14,5))
data,sample_rate=librosa.load(filename)
librosa.display.waveshow(data,sr=sample_rate)
ipd.Audio(filename)
```

Out[8]:

0:00 / 0:00



In [9]:
```
data
```

Out[9]:
```
array([ 1.4551915e-10, -8.7311491e-11, -1.1641532e-10, ...,
        3.6435030e-04,  2.6052771e-04, -2.1291785e-04], dtype=float32)
```

In [10]:
```
sample_rate ### >22 khz
```
Out[10]:
```
22050
```

Scipy

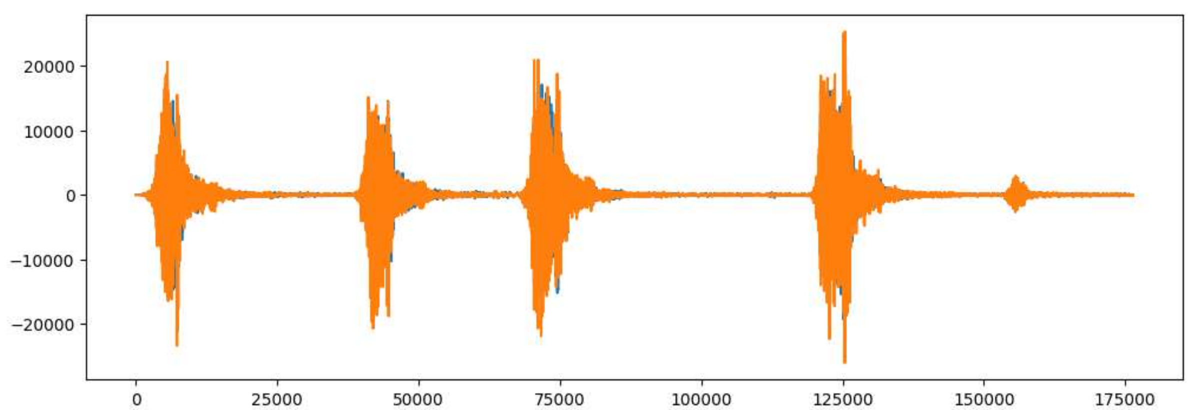
In the audio domain, SciPy is often used for tasks such as reading and writing audio files, filtering, convolution, and basic signal analysis. It provides functions to manipulate and process audio signals, making it a valuable tool in audio-related scientific and engineering applications.

In [11]:
```
from scipy.io import wavfile as wav # convert audio into stirio signal
wav_sample_rate,scipy_audio=wav.read(filename)
```

In [12]:
```
wav_sample_rate
```
Out[12]:
```
44100
```

In [13]:
```
scipy_audio
```
Out[13]:
```
array([[  0,   0],
       [  0,   0],
       [  0,   0],
       ...,
       [  6,  -5],
       [  7, -17],
       [ -5, -21]], dtype=int16)
```

In [14]:
```
# Original audio with 2 channels
plt.figure(figsize=(12, 4))
plt.plot(scipy_audio)
```
Out[14]:
```
[<matplotlib.lines.Line2D at 0x208b834f690>,
 <matplotlib.lines.Line2D at 0x208b836b850>]
```



What is the difference between librosa & scipy

Librosa and SciPy are both Python libraries used in audio signal processing, but they serve different purposes. Librosa is specialized for music and audio analysis, offering tools for feature extraction, spectral analysis, and more. SciPy, on the other hand, is a general-purpose scientific computing library with signal processing capabilities,

including functions for reading and writing audio files, filtering, and basic signal analysis. In summary, Librosa is tailored for music-specific tasks, while SciPy provides broader functionality for scientific computing, including some audio signal processing features.

Here we are using Librosa

In [ ]:

```python

```

In [15]:

```python
import pandas as pd

metadata=pd.read_csv('UrbanSound8K/metadata/UrbanSound8K.csv')
metadata.head(10)
```

Out[15]:

	slice_file_name	fsID	start	end	salience	fold	classID	class
0	100032-3-0-0.wav	100032	0.000000	0.317551	1	5	3	dog_bark
1	100263-2-0-117.wav	100263	58.500000	62.500000	1	5	2	children_playing
2	100263-2-0-121.wav	100263	60.500000	64.500000	1	5	2	children_playing
3	100263-2-0-126.wav	100263	63.000000	67.000000	1	5	2	children_playing
4	100263-2-0-137.wav	100263	68.500000	72.500000	1	5	2	children_playing
5	100263-2-0-143.wav	100263	71.500000	75.500000	1	5	2	children_playing
6	100263-2-0-161.wav	100263	80.500000	84.500000	1	5	2	children_playing
7	100263-2-0-3.wav	100263	1.500000	5.500000	1	5	2	children_playing
8	100263-2-0-36.wav	100263	18.000000	22.000000	1	5	2	children_playing
9	100648-1-0-0.wav	100648	4.823402	5.471927	2	10	1	car_horn

In [16]:

```python
### check whether the dataset is imblanced
metadata['class'].value_counts()
```

Out[16]:

```
class
dog_bark            1000
children_playing    1000
air_conditioner     1000
street_music        1000
engine_idling       1000
jackhammer          1000
drilling            1000
siren                929
car_horn             429
gun_shot             374
Name: count, dtype: int64
```

In [ ]:

```python

```

Data Preprocessing

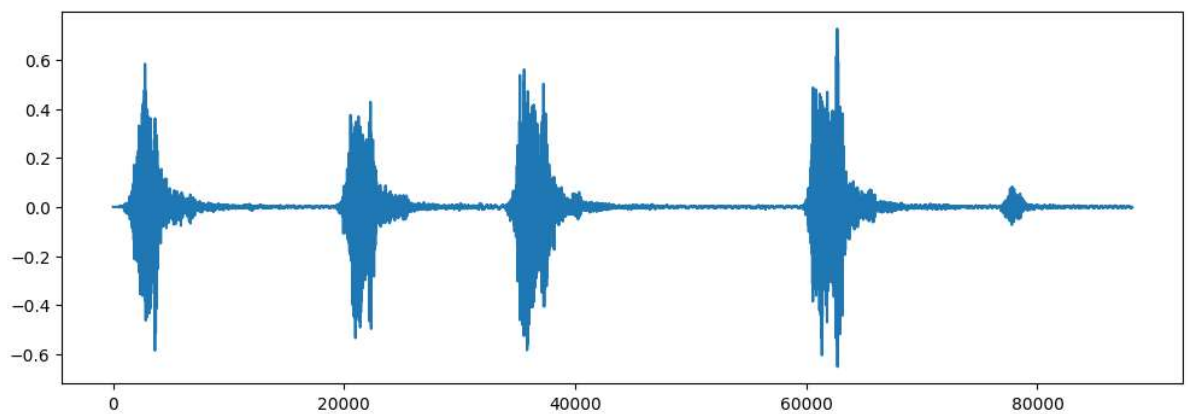
In [17]:
```
### Let's read a sample audio using librosa
import librosa
audio_file_path='dog_bark.wav'
librosa_audio_data,librosa_sample_rate=librosa.load(audio_file_path)
```

In [18]:
```
print(librosa_audio_data) # librosa convert audio data into mono channel
```

```
[ 1.4551915e-10 -8.7311491e-11 -1.1641532e-10 ...  3.6435030e-04
  2.6052771e-04 -2.1291785e-04]
```

In [19]:
```
### Lets plot the librosa audio data
import matplotlib.pyplot as plt
# Original audio with 1 channel
plt.figure(figsize=(12,4))
plt.plot(librosa_audio_data)
```

Out[19]:
```
[<matplotlib.lines.Line2D at 0x208bca354d0>]
```



In [ ]:
```

```

Extract Features

Using Mel-Frequency Cepstral Coefficients

Mel-Frequency Cepstral Coefficients (MFCCs) are numerical representations of the power spectrum of a sound signal, commonly used in speech and audio processing. They capture the spectral characteristics of a signal by converting the frequency-domain information into a set of coefficients. The process involves applying a series of steps, including framing the signal, applying the Fast Fourier Transform (FFT), mapping the resulting spectrum onto the mel scale, taking the logarithm of the powers, and finally applying the Discrete Cosine Transform (DCT) to obtain the MFCCs. These coefficients are widely employed in speech and audio analysis for tasks such as speech recognition and speaker identification.

In [20]:
```
mfccs = librosa.feature.mfcc(y=librosa_audio_data, sr=librosa_sample_rate,
mfccs.shape
```

Out[20]: (40, 173)

In [21]:
```
mfccs
```

Out[21]:
```
array([[-5.8003693e+02, -4.9177695e+02, -3.6617474e+02, ...,
        -5.0730090e+02, -5.1229175e+02, -5.2267572e+02],
       [ 3.0879444e+01,  1.1654912e+02,  1.7551926e+02, ...,
         9.0842987e+01,  9.2580032e+01,  8.8494919e+01],
       [ 1.7225260e+01,  3.9759499e+01, -5.0101824e+00, ...,
         2.7333740e+01,  2.7949635e+01,  3.1582390e+01],
       ...,
       [-3.7395463e+00, -4.9923792e+00,  4.4441938e+00, ...,
         2.7369065e+00,  1.6080571e+00,  2.7038860e+00],
       [-1.9384031e+00, -4.6504954e-01,  6.2187805e+00, ...,
         2.7966838e+00,  2.2690997e+00,  9.9648261e-01],
       [ 1.7400653e+00,  2.0404801e+00,  4.3179603e+00, ...,
         1.5787597e+00,  1.0261321e+00, -3.2630148e+00]], dtype=float32)
```

In [ ]:
```

```

In [22]:
```
### Extracting MFCC's For every audio file
import pandas as pd
import os
import librosa


audio_dataset_path='UrbanSound8K/audio/'
metadata=pd.read_csv('UrbanSound8K/metadata/UrbanSound8K.csv')
metadata.head()
```

Out[22]:

	slice_file_name	fsID	start	end	salience	fold	classID	class
0	100032-3-0-0.wav	100032	0.0	0.317551	1	5	3	dog_bark
1	100263-2-0-117.wav	100263	58.5	62.500000	1	5	2	children_playing
2	100263-2-0-121.wav	100263	60.5	64.500000	1	5	2	children_playing
3	100263-2-0-126.wav	100263	63.0	67.000000	1	5	2	children_playing
4	100263-2-0-137.wav	100263	68.5	72.500000	1	5	2	children_playing

In [23]:
```
def features_extractor(file):
    audio, sample_rate = librosa.load(file_name,res_type='kaiser_fast')
    mfccs_features = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=
    mfccs_scaled_features = np.mean(mfccs_features.T,axis=0)

    return mfccs_scaled_features
```

In [24]:
```
#pip install resampy
```

In [25]:
```
import numpy as np
from tqdm import tqdm
### Now we iterate through every audio file and extract features
### using Mel-Frequency Cepstral Coefficients
extracted_features=[]
for index_num,row in tqdm(metadata.iterrows()):
    file_name=os.path.join(os.path.abspath(audio_dataset_path),'fold'+str(
    final_class_labels=row["class"]
    data=features_extractor(file_name)
    extracted_features.append([data,final_class_labels])
```

```
3555it [05:07, 11.29it/s]C:\Users\HP\AppData\Local\Programs\Python\Python311
\Lib\site-packages\librosa\core\spectrum.py:257: UserWarning: n_fft=2048 is t
oo large for input signal of length=1323
  warnings.warn(
8325it [11:49, 16.96it/s]C:\Users\HP\AppData\Local\Programs\Python\Python311
\Lib\site-packages\librosa\core\spectrum.py:257: UserWarning: n_fft=2048 is t
oo large for input signal of length=1103
  warnings.warn(
C:\Users\HP\AppData\Local\Programs\Python\Python311\Lib\site-packages\librosa
\core\spectrum.py:257: UserWarning: n_fft=2048 is too large for input signal
of length=1523
  warnings.warn(
8732it [12:22, 11.77it/s]
```

In [26]:
```
extracted_features_df = pd.DataFrame(extracted_features,columns=['feature'
extracted_features_df.head()
```

Out[26]:

	feature	class
0	[-217.35526, 70.22338, -130.38527, -53.282898,...	dog_bark
1	[-424.09818, 109.34077, -52.919525, 60.86475, ...	children_playing
2	[-458.79114, 121.38419, -46.52066, 52.00812, -...	children_playing
3	[-413.89984, 101.66373, -35.42945, 53.036354, ...	children_playing
4	[-446.60352, 113.68541, -52.402214, 60.302044,...	children_playing

In [27]:
```
#extracted_features_df.to_csv('extractedfeatures.csv', index=False)
```

In [28]:
```
#import pandas as pd
#import numpy as np
```

In [29]:
```
#extracted_features_df=pd.read_csv("extractedfeatures.csv")
```

In [30]:
```
### Splite the dataset into independent and dependent dataset
x=np.array(extracted_features_df['feature'].tolist())
y=np.array(extracted_features_df['class'].tolist())

```

In [31]:
```
x.shape
```

Out[31]: (8732, 40)

In [32]:
```
y
```

Out[32]:
```
array(['dog_bark', 'children_playing', 'children_playing', ...,
       'car_horn', 'car_horn', 'car_horn'], dtype='<U16')
```

In [33]:
```
### Label Encoder
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder
labelencoder=LabelEncoder()
y=to_categorical(labelencoder.fit_transform(y))
```

In [34]:
```
y
```

Out[34]:
```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.],
       ...,
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.]], dtype=float32)
```

Split the data in two parts

In [35]:
```
from sklearn.model_selection import train_test_split
```

In [36]:
```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, r
```

In [37]:
```
X_train.shape
```

Out[37]:
```
(6985, 40)
```

In [38]:
```
y_train.shape
```

Out[38]:
```
(6985, 10)
```

In [ ]:
```

```

Model Creation

In [1]:
```
import tensorflow as tf
```

In [40]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Dropout,Activation,Flatten
from tensorflow.keras.optimizers import Adam
from sklearn import metrics
```

In [41]:
```
# No of classes
num_labels=y.shape[1]
```

In [42]:
```

```

In [43]:
```
model=Sequential()

### First Layer
model.add(Dense(100,input_shape=(40,)))
model.add(Activation('relu'))
model.add(Dropout(0.5)) # Why Dropout is use

### Second Layer
model.add(Dense(200))
model.add(Activation('relu'))
model.add(Dropout(0.5))

### Third Layer
model.add(Dense(100))
model.add(Activation('relu'))
model.add(Dropout(0.5))

### Final Layer
model.add(Dense(num_labels))
model.add(Activation('softmax'))

```

In [44]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense (Dense)               (None, 100)               4100      
                                                                 
 activation (Activation)     (None, 100)               0         
                                                                 
 dropout (Dropout)           (None, 100)               0         
                                                                 
 dense_1 (Dense)             (None, 200)               20200     
                                                                 
 activation_1 (Activation)   (None, 200)               0         
                                                                 
 dropout_1 (Dropout)         (None, 200)               0         
                                                                 
 dense_2 (Dense)             (None, 100)               20100     
                                                                 
 activation_2 (Activation)   (None, 100)               0         
                                                                 
 dropout_2 (Dropout)         (None, 100)               0         
                                                                 
 dense_3 (Dense)             (None, 10)                1010      
                                                                 
 activation_3 (Activation)   (None, 10)                0         
                                                                 
=================================================================
Total params: 45,410
Trainable params: 45,410
Non-trainable params: 0
_________________________________________________________________
```

In [45]:
```
model.compile(loss='categorical_crossentropy',metrics=['accuracy'],optimiz
```

In [46]:
```
#### Training my model
from tensorflow.keras.callbacks import ModelCheckpoint
from datetime import datetime

num_epochs = 500
num_batch_size=32
checkpoint = ModelCheckpoint(filepath='saved_models/audio_classification.h

start = datetime.now()

model.fit(X_train,y_train,batch_size=num_batch_size,epochs=num_epochs,vali

duration = datetime.now() - start
print("Training completed in time ", duration)
```

```
Epoch 1/700
212/219 [============================>.] - ETA: 0s - loss: 10.6375 - accura
cy: 0.1324
Epoch 1: val_loss improved from inf to 2.28775, saving model to saved_model
s\audio_classification.hdf5
219/219 [==============================] - 4s 9ms/step - loss: 10.4159 - ac
curacy: 0.1321 - val_loss: 2.2877 - val_accuracy: 0.1059
Epoch 2/700
215/219 [============================>.] - ETA: 0s - loss: 2.5094 - accurac
y: 0.1366
Epoch 2: val_loss improved from 2.28775 to 2.27706, saving model to saved_m
odels\audio_classification.hdf5
219/219 [==============================] - 1s 7ms/step - loss: 2.5090 - acc
uracy: 0.1363 - val_loss: 2.2771 - val_accuracy: 0.1122
Epoch 3/700
219/219 [==============================] - ETA: 0s - loss: 2.3251 - accurac
y: 0.1463
Epoch 3: val_loss improved from 2.27706 to 2.23326, saving model to saved_m
odels\audio_classification.hdf5
```

In [47]:
```
test_accuracy = model.evaluate(X_test,y_test,verbose=0)
print(test_accuracy[1])
```

```
0.8202633261680603
```

In [48]:
```
filename="dog_bark.wav"
prediction_feature=features_extractor(filename)
prediction_feature=prediction_feature.reshape(1,-1)
predictions=model.predict(prediction_feature)
predicted_classes = np.argmax(predictions, axis=1)
print("Predicted Classes:", predicted_classes)
```

```
1/1 [==============================] - 0s 202ms/step
Predicted Classes: [9]
```

In [ ]:
```

```

Testing Some Test Audio Data

In [ ]:

```

```

In [49]:

```python
filename="testAudio/dog_bark.wav"
audio,sample_rate = librosa.load(filename,res_type='kaiser_fast')
mfccs_features = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=40)
mfccs_scaled_features = np.mean(mfccs_features.T, axis=0)

print(mfccs_scaled_features)
mfccs_scaled_features=mfccs_scaled_features.reshape(1,-1)
print(mfccs_scaled_features)
print(mfccs_scaled_features.shape)
predicted_label=model.predict(mfccs_scaled_features)
predicted_label = np.argmax(predicted_label, axis=1)
print(predicted_label)
predicted_class= labelencoder.inverse_transform(predicted_label)
print(f"Predicted Class: {predicted_class}")
```

```
[-3.96273438e+02  1.35678085e+02 -8.87105179e+00 -1.31443491e+01
 -3.03524661e+00 -9.20888007e-01 -9.90265751e+00  4.11157370e+00
 -3.45679484e-02 -2.76059484e+00  3.53937483e+00  5.03969812e+00
  8.68413353e+00  1.26352654e+01  1.20249009e+00  5.19995809e-01
  2.37442183e+00  8.39730740e+00  1.67275083e+00 -7.17254162e+00
 -2.43528509e+00  1.59860241e+00  4.82548237e+00  8.36232185e+00
  3.71345329e+00  1.06405444e-01 -9.77238536e-01  3.11135091e-02
 -2.76709723e+00 -3.47207761e+00  1.95038319e+00  3.71639514e+00
  4.09950876e+00  3.32178473e+00  4.86096144e+00  1.85095692e+00
 -4.40172404e-02 -1.72012842e+00  6.98764801e-01  1.07319450e+00]
[[-3.96273438e+02  1.35678085e+02 -8.87105179e+00 -1.31443491e+01
  -3.03524661e+00 -9.20888007e-01 -9.90265751e+00  4.11157370e+00
  -3.45679484e-02 -2.76059484e+00  3.53937483e+00  5.03969812e+00
   8.68413353e+00  1.26352654e+01  1.20249009e+00  5.19995809e-01
   2.37442183e+00  8.39730740e+00  1.67275083e+00 -7.17254162e+00
  -2.43528509e+00  1.59860241e+00  4.82548237e+00  8.36232185e+00
   3.71345329e+00  1.06405444e-01 -9.77238536e-01  3.11135091e-02
  -2.76709723e+00 -3.47207761e+00  1.95038319e+00  3.71639514e+00
   4.09950876e+00  3.32178473e+00  4.86096144e+00  1.85095692e+00
  -4.40172404e-02 -1.72012842e+00  6.98764801e-01  1.07319450e+00]]
(1, 40)
1/1 [==============================] - 0s 40ms/step
[3]
Predicted Class: ['dog_bark']
```

In [ ]:

```

```

In [ ]:

```

```