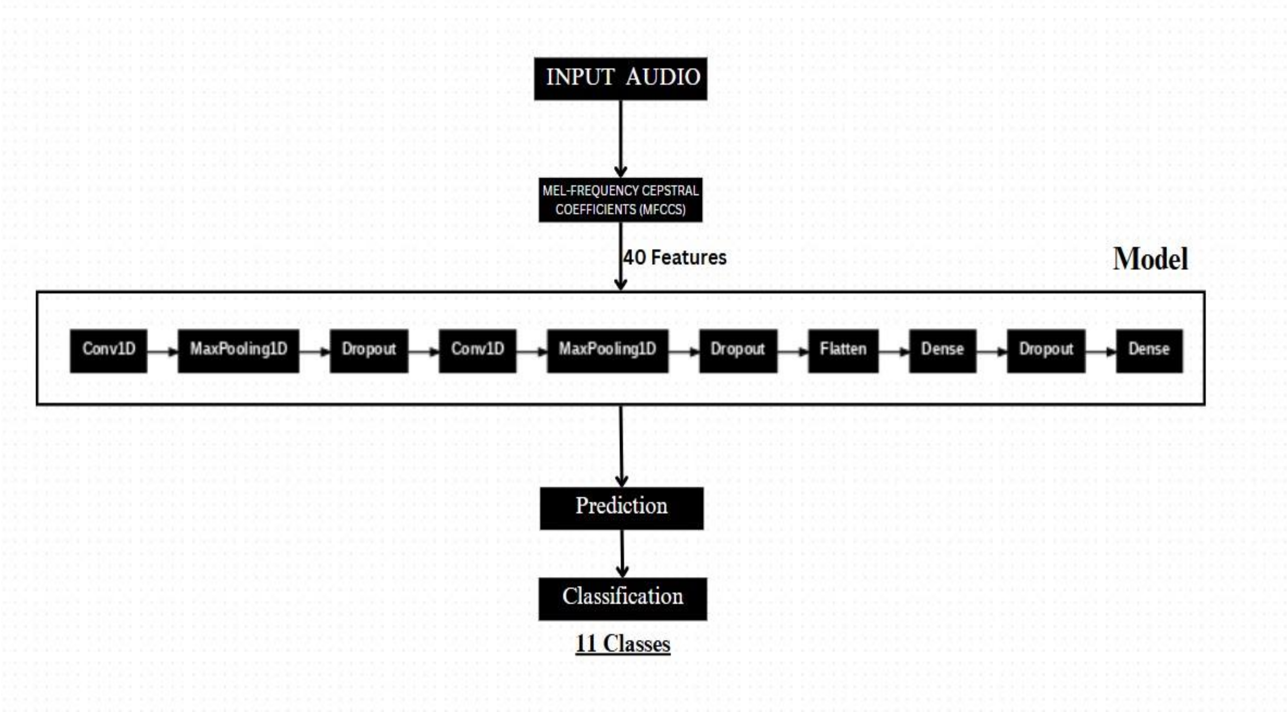
Architecture of Audio Classification



Introduction to the Audio Classification Model

1. Input Audio Processing: The system begins with raw audio input. These audio signals are processed to extract meaningful features that can be used for classification.
2. Feature Extraction: The audio data is transformed into Mel-Frequency Cepstral Coefficients (MFCCs), which are widely used for representing audio signals in machine learning tasks. The feature extraction process results in 40 MFCC features, capturing the essential characteristics of the input audio.
3. Model Architecture: The extracted MFCC features are passed through a Convolutional Neural Network (CNN). The CNN architecture is designed as follows:
    - Convolutional Layers (Conv1D): Two Conv1D layers are used to capture patterns in the sequential audio data.
    - MaxPooling Layers: MaxPooling layers follow each Conv1D layer, reducing the dimensionality and computational complexity while preserving key features.

- Dropout Layers: Dropout layers are integrated at multiple stages to mitigate overfitting and enhance generalization.
- Fully Connected Layers (Dense): After flattening the feature maps, Dense layers process the data and produce the final predictions.

4. Classification and Output: The model predicts the probabilities for each class, ultimately classifying the input audio into one of 11 predefined classes. This classification pipeline is suitable for various applications, including speech recognition, environmental sound classification, and other audio analysis tasks.

Repository Link

All the files and resources related to this project can be accessed in the Git repository at the following link:
GitHub Repository: Audio Classification

Contents of the Document

The subsequent pages contain the following sections:

1. **Data Exploration and Analysis (EDA):** A detailed exploration of the dataset used for the audio classification task, including statistical summaries and visualizations.
2. **CNN Implementation:** The actual Convolutional Neural Network (CNN) code used for modeling and training the classification pipeline.
3. **Model Output:** The results generated by the model, including performance metrics and predictions.

In [1]:
```
#####dataset url https://urbansounddataset.weebly.com/urbansound8k.html
```

In [2]:
```
import matplotlib.pyplot as plt
```

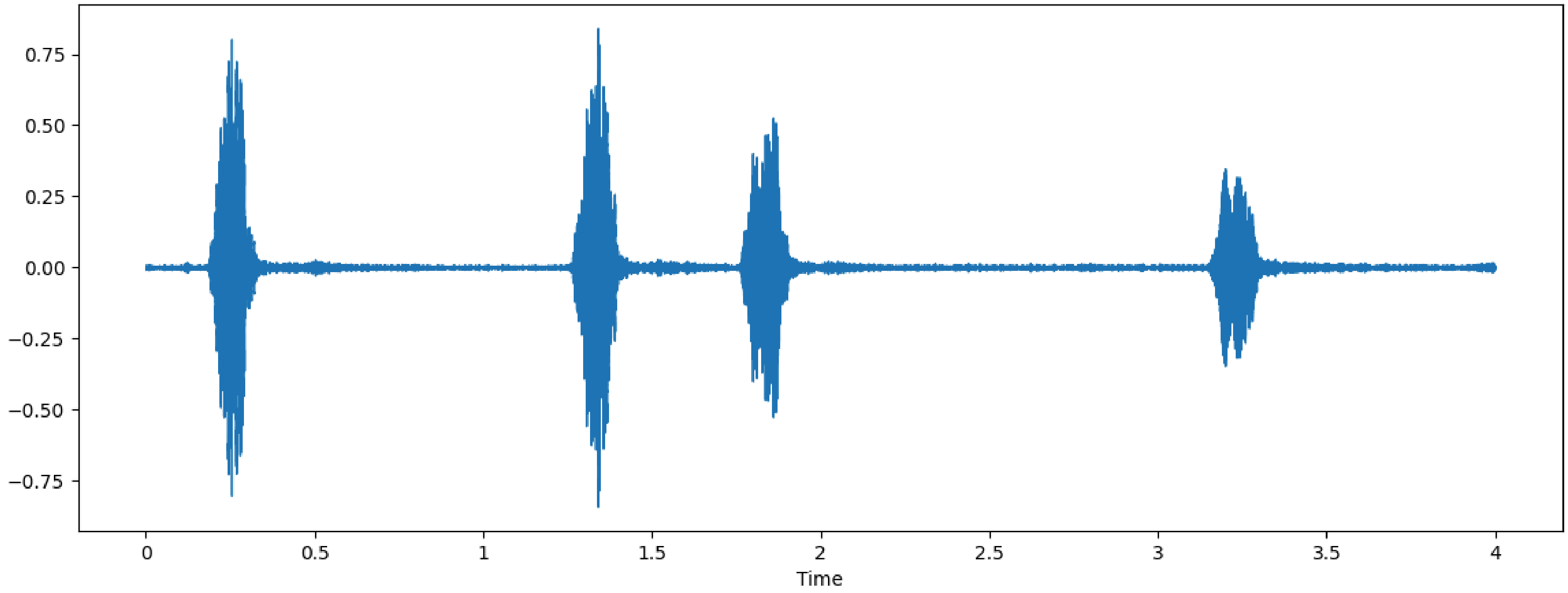
In [3]:
```
filename=r'C:\Users\pratik\Documents\Projects\audio_classification\UrbanSound8K\audio\fold1\193394-3-0-10.wav'
```

In [4]:
```
import IPython.display as ipd
import librosa
import librosa.display
```

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

Out[5]:

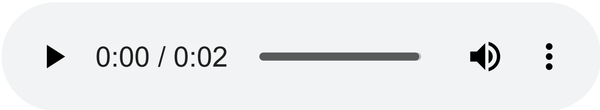
0:00 / 0:04

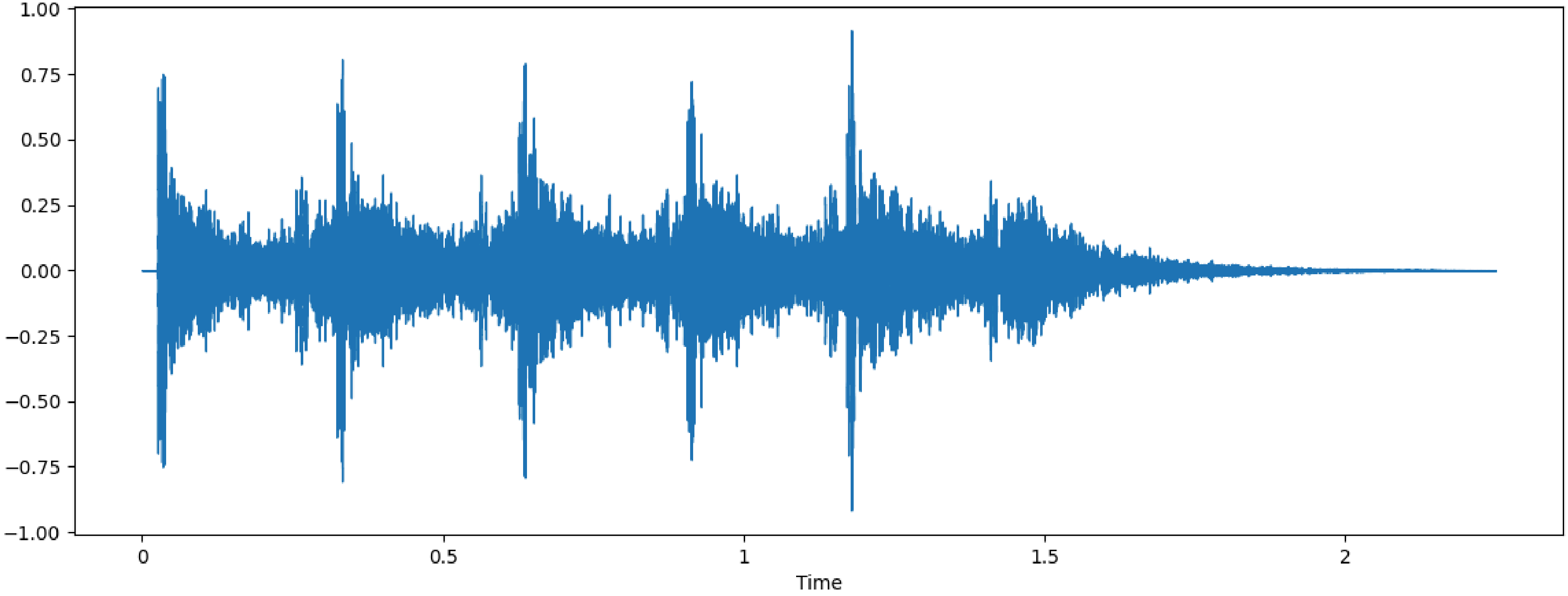


In [6]:

```
### Sound
filename=r'C:\Users\pratik\Documents\Projects\audio_classification\UrbanSound8K\audio\fold1\7061-6-0-0.wav'
plt.figure(figsize=(14,5))
data,sample_rate=librosa.load(filename)
librosa.display.waveshow(data,sr=sample_rate)
ipd.Audio(filename)
```

Out[6]:





In [7]:
```
sample_rate
```

Out[7]: 22050

In [8]:
```
from scipy.io import wavfile as wav
wave_sample_rate, wave_audio=wav.read(filename)
```

In [9]:
```
wave_sample_rate
```

Out[9]: 44100

In [10]:
```
wave_audio
```

Out[10]:
```
array([[0, 0],
       [0, 0],
       [0, 0],
       ...,
       [1, 0],
       [1, 1],
       [0, 0]], dtype=int16)
```

In [11]:
```
data
```

Out[11]:
```
array([-7.4505806e-09,  2.9802322e-08,  4.8428774e-08, ...,
        1.0127544e-05,  2.3271263e-05,  7.0009992e-06], dtype=float32)
```

In [12]:
```
import pandas as pd

metadata=pd.read_csv(r'C:\Users\pratik\Documents\Projects\audio_classification\UrbanSound8K\metadata\metadata2.csv')
metadata.head(10)
```

Out[12]:

	Unnamed: 0	slice_file_name	fsID	start	end	salience	fold	classID	class
0	0	100032-3-0-0.wav	100032.0	0.000000	0.317551	1.0	5	3.0	dog_bark
1	1	100263-2-0-117.wav	100263.0	58.500000	62.500000	1.0	5	2.0	children_playing
2	2	100263-2-0-121.wav	100263.0	60.500000	64.500000	1.0	5	2.0	children_playing
3	3	100263-2-0-126.wav	100263.0	63.000000	67.000000	1.0	5	2.0	children_playing
4	4	100263-2-0-137.wav	100263.0	68.500000	72.500000	1.0	5	2.0	children_playing
5	5	100263-2-0-143.wav	100263.0	71.500000	75.500000	1.0	5	2.0	children_playing
6	6	100263-2-0-161.wav	100263.0	80.500000	84.500000	1.0	5	2.0	children_playing
7	7	100263-2-0-3.wav	100263.0	1.500000	5.500000	1.0	5	2.0	children_playing
8	8	100263-2-0-36.wav	100263.0	18.000000	22.000000	1.0	5	2.0	children_playing
9	9	100648-1-0-0.wav	100648.0	4.823402	5.471927	2.0	10	1.0	car_horn

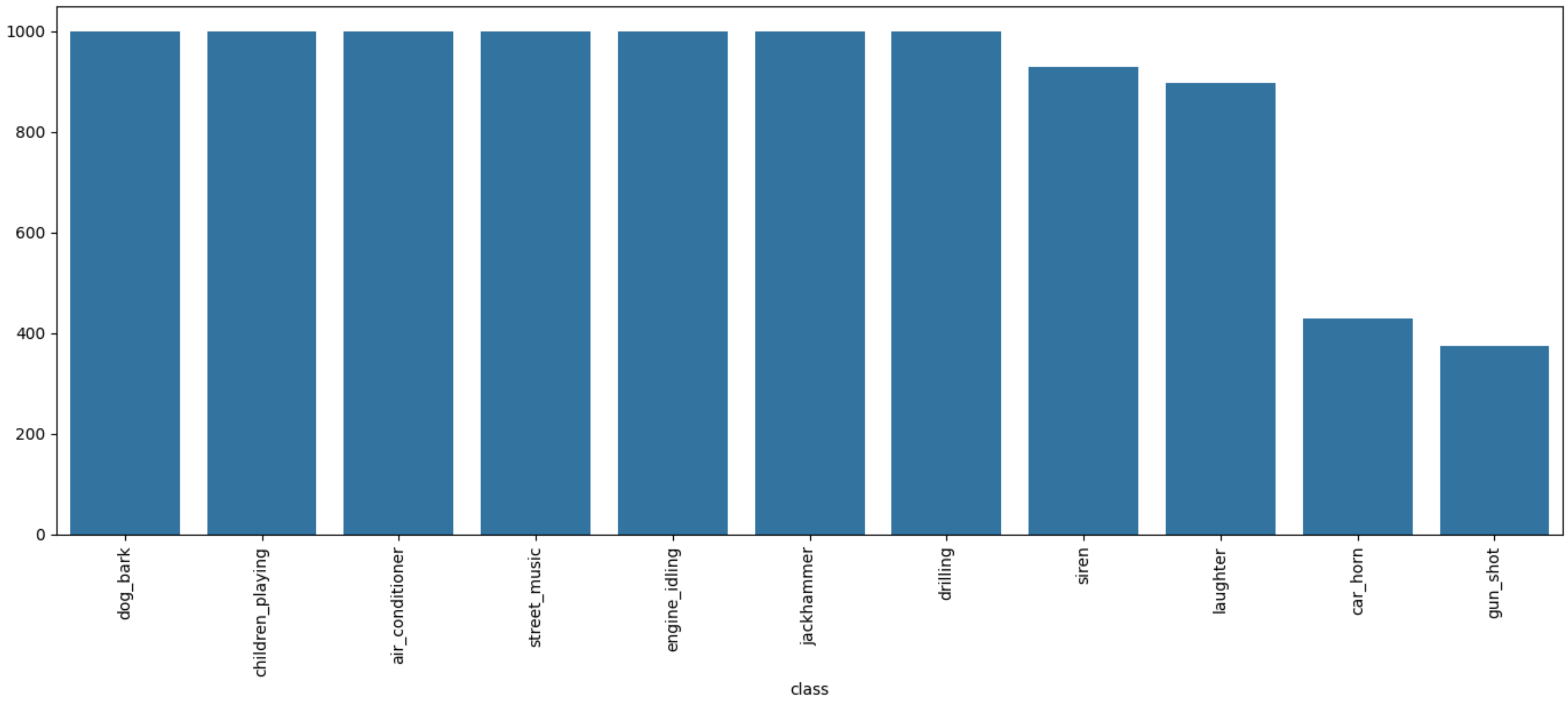
In [13]:
```
metadata['class'].value_counts()
```

Out[13]:

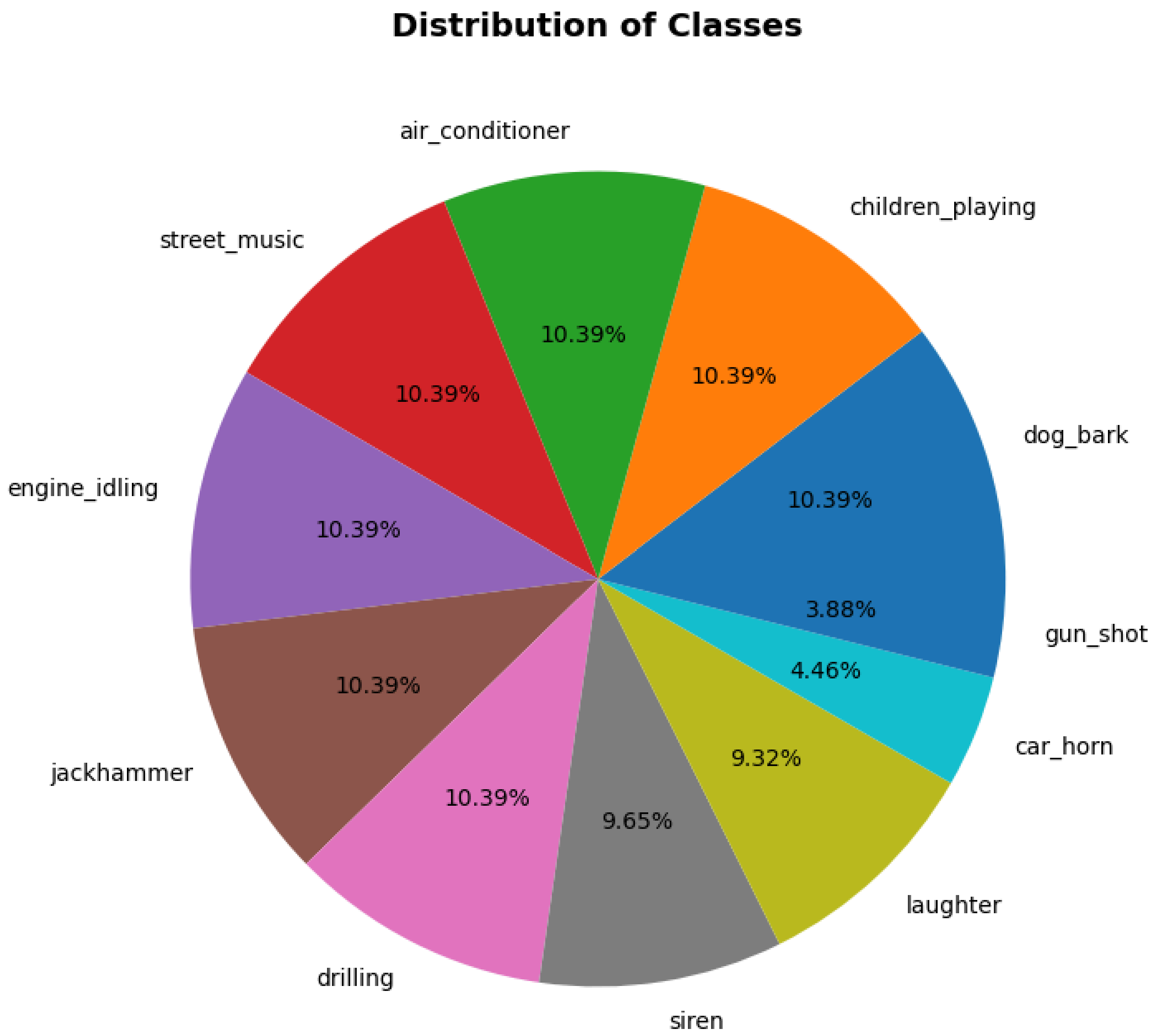
```
class
dog_bark            1000
children_playing    1000
air_conditioner     1000
street_music        1000
engine_idling       1000
jackhammer          1000
drilling            1000
siren                929
laughter             897
car_horn             429
gun_shot             374
Name: count, dtype: int64
```

In [16]:

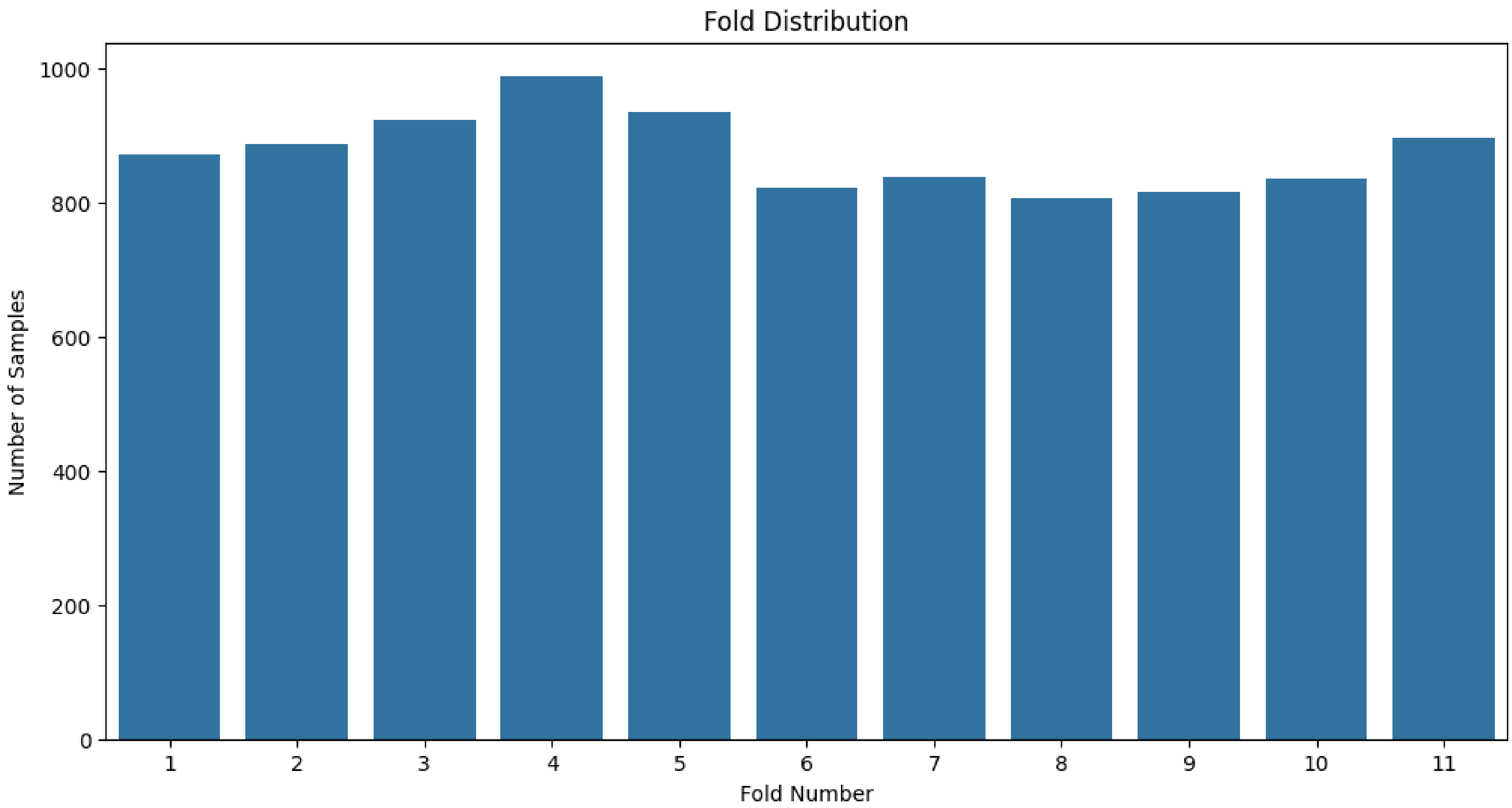
```
import seaborn as sns
plt.figure(figsize=(17, 6))
counts = metadata["class"].value_counts()
sns.barplot(x=counts.index, y=counts.values)
plt.xticks(rotation=90);
```



In [17]:
```
plt.figure(figsize=(17, 8))
plt.pie(counts, labels=counts.index, autopct="%.2f%%")
plt.title("Distribution of Classes", fontweight="black", size=14, pad=15)
plt.show()
```

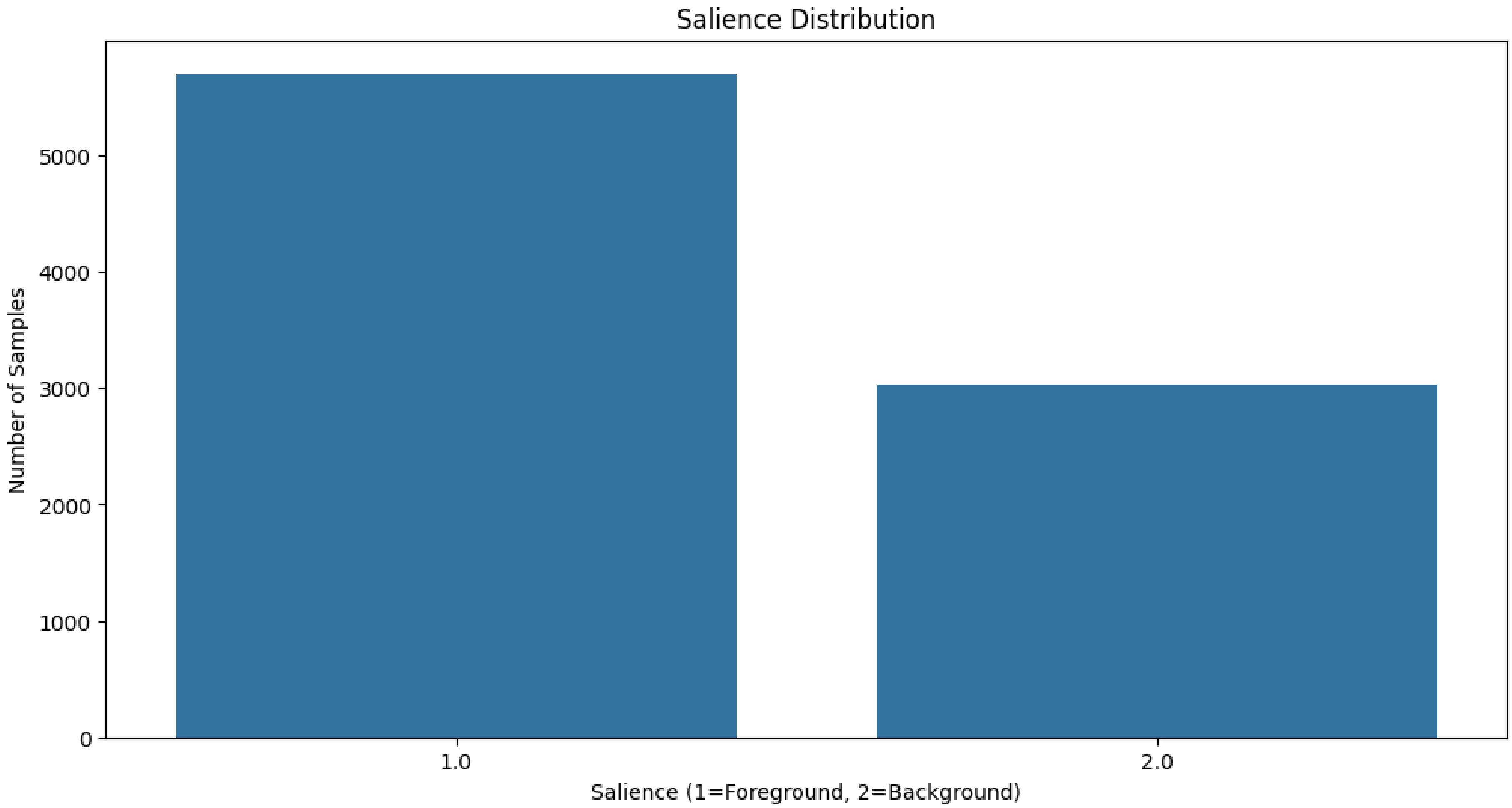

Distribution of Classes
air_conditioner
children_playing
street_music
dog_bark
engine_idling
gun_shot
car_horn
jackhammer
laughter
drilling
siren
10.39%
10.39%
10.39%
10.39%
10.39%
10.39%
10.39%
3.88%
4.46%
9.32%
9.65%

In [18]:
```
plt.figure(figsize=(12, 6))
sns.countplot(data=metadata, x='fold')
plt.title('Fold Distribution')
plt.xlabel('Fold Number')
plt.ylabel('Number of Samples')
plt.show()
```



In [19]:
```
plt.figure(figsize=(12, 6))
sns.countplot(data=metadata, x='salience')
plt.title('Salience Distribution')
```

```
plt.xlabel('Salience (1=Foreground, 2=Background)')
plt.ylabel('Number of Samples')
plt.show()
```



In [ ]:

In [ ]:

Audio Classification Data Preprocessing

In [50]:
```
import librosa
import numba
import numpy
import tensorflow as tf
print("NumPy version:", numpy.__version__)
print("Numba version:", numba.__version__)
print("Tensorflow version:",tf.__version__)
print("Librosa version:",librosa.__version__)
```

```
NumPy version: 1.26.4
Numba version: 0.60.0
Tensorflow version: 2.18.0
Librosa version: 0.10.2.post1
```

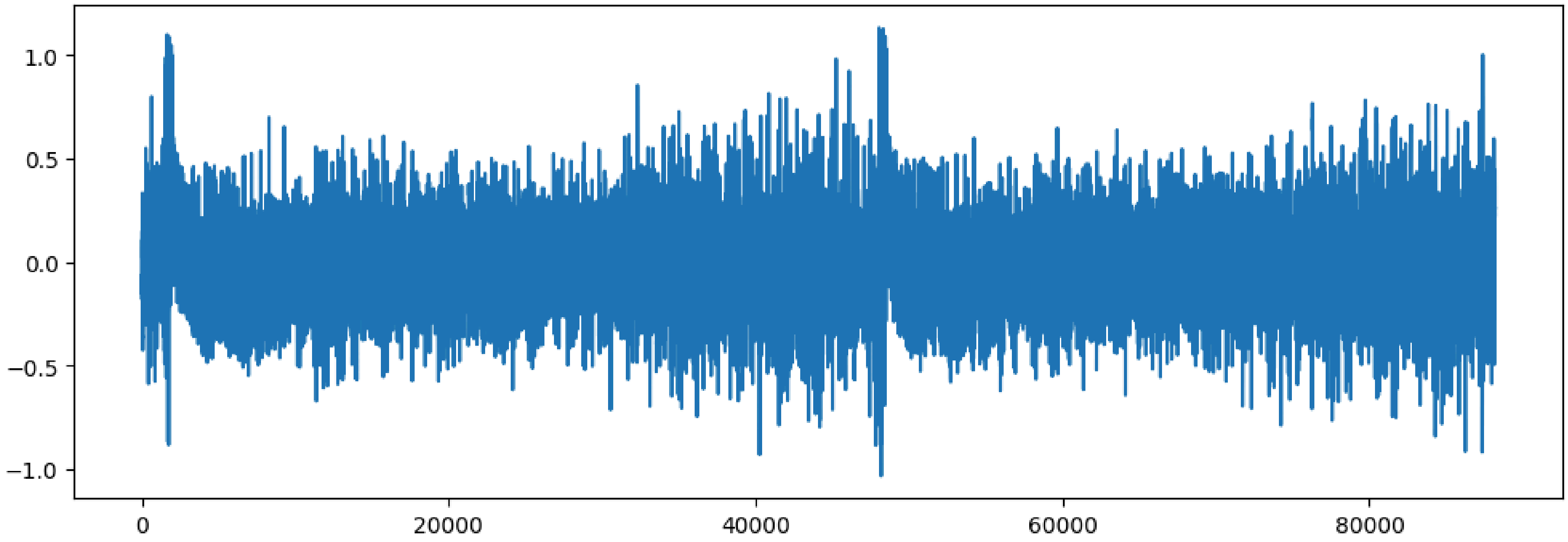
In [51]:
```
audio_file_path=r'C:\Users\Mihir\Audio_Classification\audio\fold4\135528-6-4-2.wav'
librosa_audio_data,librosa_sample_rate=librosa.load(audio_file_path)
```

In [52]:
```
print(librosa_audio_data)
```

```
[ 0.1017779   0.09252265 -0.01029476 ...  0.28079608  0.26662326
  0.22703297]
```

In [53]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 4))
plt.plot(librosa_audio_data)
```

Out[53]:
```
[<matplotlib.lines.Line2D at 0x2b1ffbf0d70>]
```



Observation

Here Librosa converts the signal to mono, meaning the channel will alays be 1

In [54]:
```
from scipy.io import wavfile as wav
wave_sample_rate, wave_audio = wav.read(audio_file_path)
```

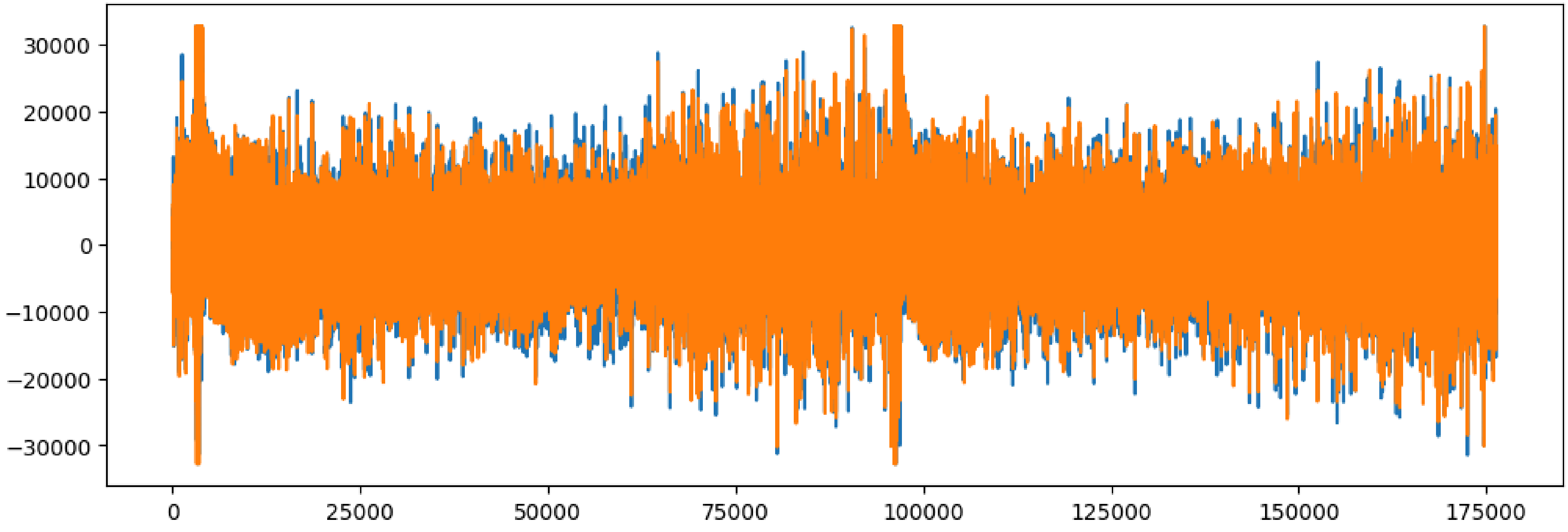
In [55]:
```
wave_audio
```

Out[55]:
```
array([[5318, 4033],
       [4459, 3127],
       [3304, 1978],
       ...,
       [8509, 7693],
       [7680, 6546],
       [6925, 5669]], dtype=int16)
```

In [56]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 4))
```

```
plt.plot(wave_audio)
```

Out[56]:
```
[<matplotlib.lines.Line2D at 0x2b1ffc50680>,
 <matplotlib.lines.Line2D at 0x2b1ffc53590>]
```



Extract Features

Here we will be using Mel-Frequency Cepstral Coefficients(MFCC) from the audio samples. The MFCC summarises the frequency distribution across the window size, so it is possible to analyse both the frequency and time characteristics of the sound. These audio representations will allow us to identify features for classification.

In [57]:
```
mfccs = librosa.feature.mfcc(y=librosa_audio_data, sr=librosa_sample_rate, n_mfcc=40)
print(mfccs.shape)
```

```
(40, 173)
```

In [58]:
```
mfccs
```

Out[58]:
```
array([[-9.0757378e+01, -6.4957870e+01,  2.5701323e+01, ...,
        -3.1466866e+01, -3.4667324e+01, -5.0364735e+01],
```

```
[ 1.7926303e+02,  1.9429706e+02,  1.0372595e+02, ...,
  1.8602203e+02,  1.8453941e+02,  1.6647110e+02],
[-3.8887314e+01, -4.5582771e+01,  3.7404199e+00, ...,
 -3.6085999e+01, -2.9083427e+01, -2.3078842e+01],
...,
[-1.1893963e+00, -9.9441922e-01,  4.5947514e+00, ...,
 -1.4260173e+00, -3.3234169e+00, -2.1230774e+00],
[-2.9536510e+00, -6.3421693e+00, -1.1170654e+00, ...,
 -2.6557617e+00, -9.8496598e-01, -6.0143051e+00],
[ 8.0155127e-02,  8.3574450e-01,  2.6181114e+00, ...,
 -1.2515843e+00,  2.3683584e+00, -1.1718901e+00]], dtype=float32)
```

In [59]:

```
import pandas as pd
import os
import librosa

audio_dataset_path='C:/Users/Mihir/Audio_Classification/audio/'
metadata=pd.read_csv('C:/Users/Mihir/Audio_Classification/metadata2.csv')
metadata.head()
```

Out[59]:

	Unnamed: 0	slice_file_name	fsID	start	end	salience	fold	classID	class
0	0	100032-3-0-0.wav	100032.0	0.0	0.317551	1.0	5	3.0	dog_bark
1	1	100263-2-0-117.wav	100263.0	58.5	62.500000	1.0	5	2.0	children_playing
2	2	100263-2-0-121.wav	100263.0	60.5	64.500000	1.0	5	2.0	children_playing
3	3	100263-2-0-126.wav	100263.0	63.0	67.000000	1.0	5	2.0	children_playing
4	4	100263-2-0-137.wav	100263.0	68.5	72.500000	1.0	5	2.0	children_playing

In [60]:

```
def features_extractor(file_name):
    audio, sample_rate = librosa.load(file_name, res_type='kaiser_fast')
    mfccs_features = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=40)
    mfccs_scaled_features = np.mean(mfccs_features,axis=1)

    return mfccs_scaled_features
```

In [61]:

```
import numpy as np
from tqdm import tqdm
extracted_features=[]
for index_num,row in tqdm(metadata.iterrows()):
    file_name = os.path.join(os.path.abspath(audio_dataset_path),'fold'+str(row["fold"])+'/',str(row["slice_file_nar
    final_class_labels=row["class"]
    data=features_extractor(file_name)
    extracted_features.append([data,final_class_labels])
```

```
3555it [02:16, 26.58it/s]c:\Users\Mihir\Audio_Classification\myenv\Lib\site-packages\librosa\core\spectrum.py:266: U
serWarning: n_fft=2048 is too large for input signal of length=1323
  warnings.warn(
8325it [05:08, 36.25it/s]c:\Users\Mihir\Audio_Classification\myenv\Lib\site-packages\librosa\core\spectrum.py:266: U
serWarning: n_fft=2048 is too large for input signal of length=1103
  warnings.warn(
c:\Users\Mihir\Audio_Classification\myenv\Lib\site-packages\librosa\core\spectrum.py:266: UserWarning: n_fft=2048 is
too large for input signal of length=1523
  warnings.warn(
9629it [05:40, 28.32it/s]
```

In [62]:

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

Out[62]:

	feature	class
0	[-217.35526, 70.22339, -130.38527, -53.282898,...	dog_bark
1	[-424.09818, 109.34077, -52.919525, 60.86475, ...	children_playing
2	[-458.79114, 121.38419, -46.520657, 52.00812, ...	children_playing
3	[-413.89984, 101.66371, -35.42945, 53.036354, ...	children_playing
4	[-446.60352, 113.68541, -52.402218, 60.302044,...	children_playing

In [63]:

```
X=np.array(extracted_features_df['feature'].tolist())
y=np.array(extracted_features_df['class'].tolist())
```

In [64]:
```
X.shape
```

Out[64]:
```
(9629, 40)
```

In [65]:
```
y
```

Out[65]:
```
array(['dog_bark', 'children_playing', 'children_playing', ...,
       'laughter', 'laughter', 'laughter'], dtype='<U16')
```

In [66]:
```
y.shape
```

Out[66]:
```
(9629,)
```

In [67]:
```
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder
labelencoder=LabelEncoder()
y=to_categorical(labelencoder.fit_transform(y))
```

In [68]:
```
y
```

Out[68]:
```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 1., 0., 0.],
       [0., 0., 0., ..., 1., 0., 0.],
       [0., 0., 0., ..., 1., 0., 0.]])
```

In [69]:
```
y.shape
```

Out[69]:
```
(9629, 11)
```

In [70]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=0)
```

In [71]:
```
X_train
```

Out[71]:
```
array([[-516.0193    ,   90.57641   ,   -5.9716887 , ...,    0.6649486 ,
           1.3720773 ,    0.726882  ],
```

```
       [ -91.93015   ,   27.93474   ,  -42.40329   , ...,    4.0705447 ,
          -2.0685377 ,    1.2096167 ],
       [-100.50485   ,   75.717445  , -136.156     , ...,   -1.6173779 ,
          -5.9498897 ,   -5.2066193 ],
       ...,
       [-427.01236   ,   92.62305   ,    3.1293974 , ...,    0.7426412 ,
           0.73349077,    0.71100914],
       [-145.75461   ,  136.26578   ,  -33.515522  , ...,    1.4681195 ,
          -2.00917   ,   -0.8821819 ],
       [-421.03134   ,  210.65454   ,    3.4906607 , ...,   -5.3888674 ,
          -3.3713605 ,   -1.5665114 ]], dtype=float32)
```

In [72]:
```
y
```

Out[72]:
```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 1., 0., 0.],
       [0., 0., 0., ..., 1., 0., 0.],
       [0., 0., 0., ..., 1., 0., 0.]])
```

In [73]:
```
X_train.shape
```

Out[73]:
```
(7703, 40)
```

In [74]:
```
X_test.shape
```

Out[74]:
```
(1926, 40)
```

In [75]:
```
y_train.shape
```

Out[75]:
```
(7703, 11)
```

In [76]:
```
y_test.shape
```

Out[76]:
```
(1926, 11)
```

Model Creation

In [77]:
```
import tensorflow as tf
print(tf.__version__)
```

```
2.18.0
```

In [78]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense, Dropout, BatchNormalization
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
```

In [79]:
```
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)
```

In [80]:
```
X_train.shape
```

Out[80]:
```
(7703, 40, 1)
```

In [81]:
```
model = Sequential()

model.add(Conv1D(64, kernel_size=3, activation='relu', input_shape=(40, 1)))
model.add(MaxPooling1D(pool_size=2))
model.add(Dropout(0.3))

model.add(Conv1D(128, kernel_size=3, activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Dropout(0.3))

model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(y_train.shape[1], activation='softmax'))
```

```
c:\Users\Mihir\Audio_Classification\myenv\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:107: UserWar
ning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `I
nput(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [82]:
```
model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv1d_2 (Conv1D)	(None, 38, 64)	256
max_pooling1d_2 (MaxPooling1D)	(None, 19, 64)	0
dropout_3 (Dropout)	(None, 19, 64)	0
conv1d_3 (Conv1D)	(None, 17, 128)	24,704
max_pooling1d_3 (MaxPooling1D)	(None, 8, 128)	0
dropout_4 (Dropout)	(None, 8, 128)	0
flatten_1 (Flatten)	(None, 1024)	0
dense_2 (Dense)	(None, 128)	131,200
dropout_5 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 11)	1,419

Total params: 157,579 (615.54 KB)

Trainable params: 157,579 (615.54 KB)

Non-trainable params: 0 (0.00 B)

In [83]:
```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

In [84]:
```
callbacks = [
    EarlyStopping(monitor='val_accuracy', patience=10, restore_best_weights=True, verbose=1)
]
```

In [85]:
```
history = model.fit(
    X_train,
    y_train,
```

```
    validation_data=(X_test, y_test),
    epochs=100,
    batch_size=32,
    callbacks=callbacks,
    verbose=1
)
```

```
Epoch 1/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.1832 - loss: 3.5540 - val_accuracy: 0.4803 - val
_loss: 1.6597
Epoch 2/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3726 - loss: 1.8006 - val_accuracy: 0.5924 - val
_loss: 1.3412
Epoch 3/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4668 - loss: 1.5488 - val_accuracy: 0.6630 - val
_loss: 1.1030
Epoch 4/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5274 - loss: 1.3870 - val_accuracy: 0.6973 - val
_loss: 1.0122
Epoch 5/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5704 - loss: 1.2650 - val_accuracy: 0.7160 - val
_loss: 0.9310
Epoch 6/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5981 - loss: 1.1831 - val_accuracy: 0.7440 - val
_loss: 0.8259
Epoch 7/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.6167 - loss: 1.1174 - val_accuracy: 0.7388 - val
_loss: 0.8150
Epoch 8/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.6421 - loss: 1.0689 - val_accuracy: 0.7814 - val
_loss: 0.7195
Epoch 9/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.6558 - loss: 1.0148 - val_accuracy: 0.7882 - val
_loss: 0.6888
Epoch 10/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.6651 - loss: 0.9749 - val_accuracy: 0.7970 - val
_loss: 0.6513
Epoch 11/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.6808 - loss: 0.9329 - val_accuracy: 0.8074 - val
_loss: 0.6130
Epoch 12/100
```

```
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.7009 - loss: 0.8671 - val_accuracy: 0.8120 - val
_loss: 0.5936
Epoch 13/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.7076 - loss: 0.8526 - val_accuracy: 0.8349 - val
_loss: 0.5453
Epoch 14/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.7178 - loss: 0.8234 - val_accuracy: 0.8385 - val
_loss: 0.5270
Epoch 15/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.7339 - loss: 0.7832 - val_accuracy: 0.8349 - val
_loss: 0.5244
Epoch 16/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.7414 - loss: 0.7550 - val_accuracy: 0.8359 - val
_loss: 0.5063
Epoch 17/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7499 - loss: 0.7298 - val_accuracy: 0.8489 - val
_loss: 0.4841
Epoch 18/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7574 - loss: 0.7204 - val_accuracy: 0.8499 - val
_loss: 0.4637
Epoch 19/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7679 - loss: 0.6792 - val_accuracy: 0.8525 - val
_loss: 0.4732
Epoch 20/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7684 - loss: 0.6692 - val_accuracy: 0.8593 - val
_loss: 0.4512
Epoch 21/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.7800 - loss: 0.6773 - val_accuracy: 0.8567 - val
_loss: 0.4408
Epoch 22/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.7752 - loss: 0.6708 - val_accuracy: 0.8577 - val
_loss: 0.4340
Epoch 23/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.7718 - loss: 0.6509 - val_accuracy: 0.8692 - val
_loss: 0.4164
Epoch 24/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7932 - loss: 0.6008 - val_accuracy: 0.8660 - val
_loss: 0.4064
Epoch 25/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8008 - loss: 0.5929 - val_accuracy: 0.8733 - val
_loss: 0.4121
```

```
Epoch 26/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.7938 - loss: 0.6135 - val_accuracy: 0.8744 - val
_loss: 0.3977
Epoch 27/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.7964 - loss: 0.5925 - val_accuracy: 0.8692 - val
_loss: 0.3973
Epoch 28/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.7972 - loss: 0.5963 - val_accuracy: 0.8790 - val
_loss: 0.3964
Epoch 29/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8032 - loss: 0.5568 - val_accuracy: 0.8712 - val
_loss: 0.4192
Epoch 30/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.7952 - loss: 0.5775 - val_accuracy: 0.8795 - val
_loss: 0.3900
Epoch 31/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8227 - loss: 0.5106 - val_accuracy: 0.8790 - val
_loss: 0.3695
Epoch 32/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8088 - loss: 0.5397 - val_accuracy: 0.8842 - val
_loss: 0.3645
Epoch 33/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8312 - loss: 0.5014 - val_accuracy: 0.8827 - val
_loss: 0.3666
Epoch 34/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8215 - loss: 0.5140 - val_accuracy: 0.8873 - val
_loss: 0.3700
Epoch 35/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8222 - loss: 0.5179 - val_accuracy: 0.8853 - val
_loss: 0.3662
Epoch 36/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8264 - loss: 0.4951 - val_accuracy: 0.8904 - val
_loss: 0.3528
Epoch 37/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8220 - loss: 0.5239 - val_accuracy: 0.8889 - val
_loss: 0.3543
Epoch 38/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8400 - loss: 0.4697 - val_accuracy: 0.8936 - val
_loss: 0.3606
Epoch 39/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8266 - loss: 0.5088 - val_accuracy: 0.8930 - val
```

```
_loss: 0.3558
Epoch 40/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8282 - loss: 0.4868 - val_accuracy: 0.8863 - val
_loss: 0.3586
Epoch 41/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8259 - loss: 0.5091 - val_accuracy: 0.8946 - val
_loss: 0.3497
Epoch 42/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8289 - loss: 0.4886 - val_accuracy: 0.8925 - val
_loss: 0.3475
Epoch 43/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8306 - loss: 0.4856 - val_accuracy: 0.8868 - val
_loss: 0.3537
Epoch 44/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8386 - loss: 0.4589 - val_accuracy: 0.8863 - val
_loss: 0.3657
Epoch 45/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8423 - loss: 0.4581 - val_accuracy: 0.8967 - val
_loss: 0.3363
Epoch 46/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1602s 7s/step - accuracy: 0.8485 - loss: 0.4401 - val_accuracy: 0.8884 - v
al_loss: 0.3504
Epoch 47/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.8322 - loss: 0.4658 - val_accuracy: 0.8956 - val
_loss: 0.3489
Epoch 48/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.8466 - loss: 0.4375 - val_accuracy: 0.8951 - val
_loss: 0.3415
Epoch 49/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.8387 - loss: 0.4538 - val_accuracy: 0.8951 - val
_loss: 0.3354
Epoch 50/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.8487 - loss: 0.4449 - val_accuracy: 0.9003 - val
_loss: 0.3292
Epoch 51/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.8456 - loss: 0.4441 - val_accuracy: 0.9003 - val
_loss: 0.3397
Epoch 52/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.8544 - loss: 0.4318 - val_accuracy: 0.9060 - val
_loss: 0.3311
Epoch 53/100
```

```
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.8484 - loss: 0.4402 - val_accuracy: 0.9065 - val
_loss: 0.3219
Epoch 54/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8412 - loss: 0.4407 - val_accuracy: 0.9008 - val
_loss: 0.3348
Epoch 55/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8567 - loss: 0.4155 - val_accuracy: 0.9045 - val
_loss: 0.3187
Epoch 56/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8442 - loss: 0.4388 - val_accuracy: 0.9050 - val
_loss: 0.3169
Epoch 57/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8613 - loss: 0.3923 - val_accuracy: 0.9029 - val
_loss: 0.3293
Epoch 58/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.8548 - loss: 0.4232 - val_accuracy: 0.9039 - val
_loss: 0.3223
Epoch 59/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.8459 - loss: 0.4480 - val_accuracy: 0.9013 - val
_loss: 0.3195
Epoch 60/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.8586 - loss: 0.4062 - val_accuracy: 0.9055 - val
_loss: 0.3222
Epoch 61/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.8628 - loss: 0.3806 - val_accuracy: 0.8993 - val
_loss: 0.3239
Epoch 62/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.8533 - loss: 0.4206 - val_accuracy: 0.9086 - val
_loss: 0.3127
Epoch 63/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8643 - loss: 0.4041 - val_accuracy: 0.9091 - val
_loss: 0.3156
Epoch 64/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8638 - loss: 0.4111 - val_accuracy: 0.9039 - val
_loss: 0.3283
Epoch 65/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8591 - loss: 0.4054 - val_accuracy: 0.9024 - val
_loss: 0.3213
Epoch 66/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8632 - loss: 0.4203 - val_accuracy: 0.9050 - val
_loss: 0.3236
```

```
Epoch 67/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8670 - loss: 0.4155 - val_accuracy: 0.9091 - val
_loss: 0.3249
Epoch 68/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8620 - loss: 0.4014 - val_accuracy: 0.9091 - val
_loss: 0.3189
Epoch 69/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8662 - loss: 0.3790 - val_accuracy: 0.9071 - val
_loss: 0.3153
Epoch 70/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8558 - loss: 0.4101 - val_accuracy: 0.9086 - val
_loss: 0.3082
Epoch 71/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8611 - loss: 0.4043 - val_accuracy: 0.9123 - val
_loss: 0.3145
Epoch 72/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8625 - loss: 0.3955 - val_accuracy: 0.9117 - val
_loss: 0.3220
Epoch 73/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8697 - loss: 0.3987 - val_accuracy: 0.9148 - val
_loss: 0.3126
Epoch 74/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8679 - loss: 0.3831 - val_accuracy: 0.9086 - val
_loss: 0.3134
Epoch 75/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8683 - loss: 0.3770 - val_accuracy: 0.9154 - val
_loss: 0.3060
Epoch 76/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8810 - loss: 0.3532 - val_accuracy: 0.9148 - val
_loss: 0.3059
Epoch 77/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8656 - loss: 0.3815 - val_accuracy: 0.9123 - val
_loss: 0.3082
Epoch 78/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8725 - loss: 0.3616 - val_accuracy: 0.9112 - val
_loss: 0.3258
Epoch 79/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8706 - loss: 0.3592 - val_accuracy: 0.9112 - val
_loss: 0.3214
Epoch 80/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8724 - loss: 0.3593 - val_accuracy: 0.9065 - val
```

```
_loss: 0.3360
Epoch 81/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8766 - loss: 0.3531 - val_accuracy: 0.9091 - val
_loss: 0.3245
Epoch 82/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8782 - loss: 0.3540 - val_accuracy: 0.9071 - val
_loss: 0.3201
Epoch 83/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8633 - loss: 0.3775 - val_accuracy: 0.9086 - val
_loss: 0.3199
Epoch 84/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8701 - loss: 0.3633 - val_accuracy: 0.9128 - val
_loss: 0.3125
Epoch 85/100
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8740 - loss: 0.3680 - val_accuracy: 0.9097 - val
_loss: 0.3220
Epoch 85: early stopping
Restoring model weights from the end of the best epoch: 75.
```

In [99]:

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

```
0.9153686165809631
```

In [101…

```
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_pred_classes
```

```
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```

Out[101…

```
array([5, 5, 0, ..., 0, 6, 8], dtype=int64)
```

Testing Some Test Audio Data

Steps

- Preprocess the new audio data
- predict the classes
- Invere transform your Predicted Label

In [102…
```
filename = r"C:\Users\Mihir\Audio_Classification\audio\fold4\52441-3-0-0.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, axis=1)

mfccs_scaled_features = mfccs_scaled_features.reshape(1, 40, 1)
print(mfccs_scaled_features.shape)

y_pred = model.predict(mfccs_scaled_features)
print(y_pred)

y_pred_classes = np.argmax(y_pred, axis=1)
print(y_pred_classes)

prediction_class = labelencoder.inverse_transform(y_pred_classes)
print(prediction_class)
```

```
(1, 40, 1)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
[[1.2825653e-07 5.1664272e-05 6.0751818e-02 9.2083097e-01 8.5027609e-04
  1.3739476e-05 4.7315522e-03 4.9767298e-08 2.3268769e-03 9.0271821e-03
  1.4158334e-03]]
[3]
['dog_bark']
```

In [103…
```
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
import seaborn as sns
```

In [104…
```
y_testlab=np.argmax(y_test, axis=1)
y_testlab.shape
```

Out[104…
```
(1926,)
```

In [105…
```
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_pred_classes.shape
```

```
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```

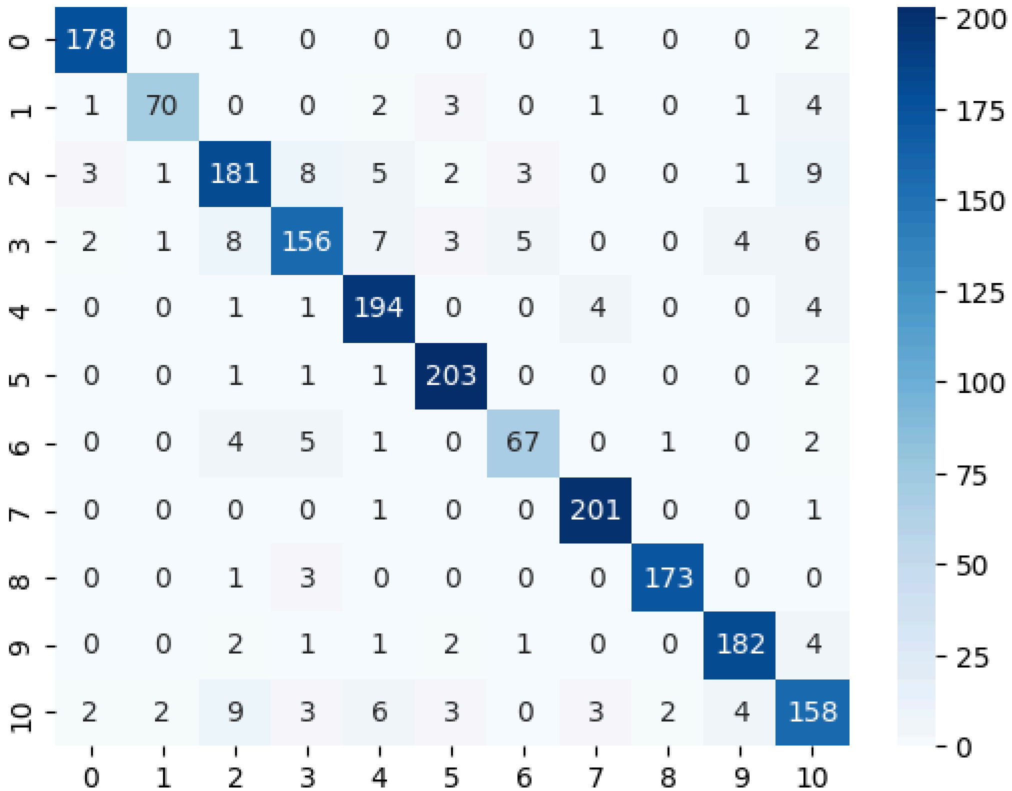
Out[105… (1926,)

In [106…
```
accuracy_score(y_testlab,y_pred_classes)
```

Out[106… 0.9153686396677051

In [107…
```
cm=confusion_matrix(y_testlab,y_pred_classes)
sns.heatmap(cm,annot=True,fmt='d',cmap='Blues')
```

Out[107… <Axes: >



In [108…
```
model.save(r'final_audio_model.keras')
```

In [109…
```
class_names = labelencoder.classes_
```

```
class_names.shape
```

Out[109…

```
(11,)
```

In [110…

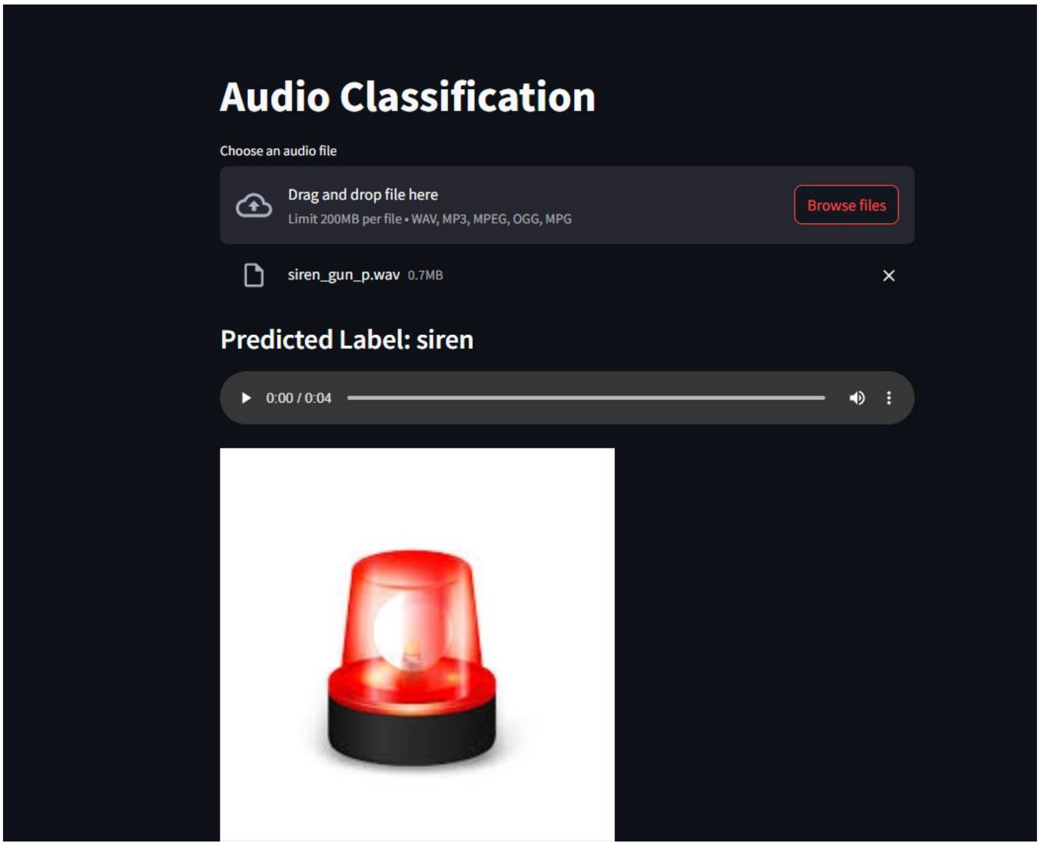
```
print(classification_report(y_testlab,y_pred_classes,target_names=class_names))
```

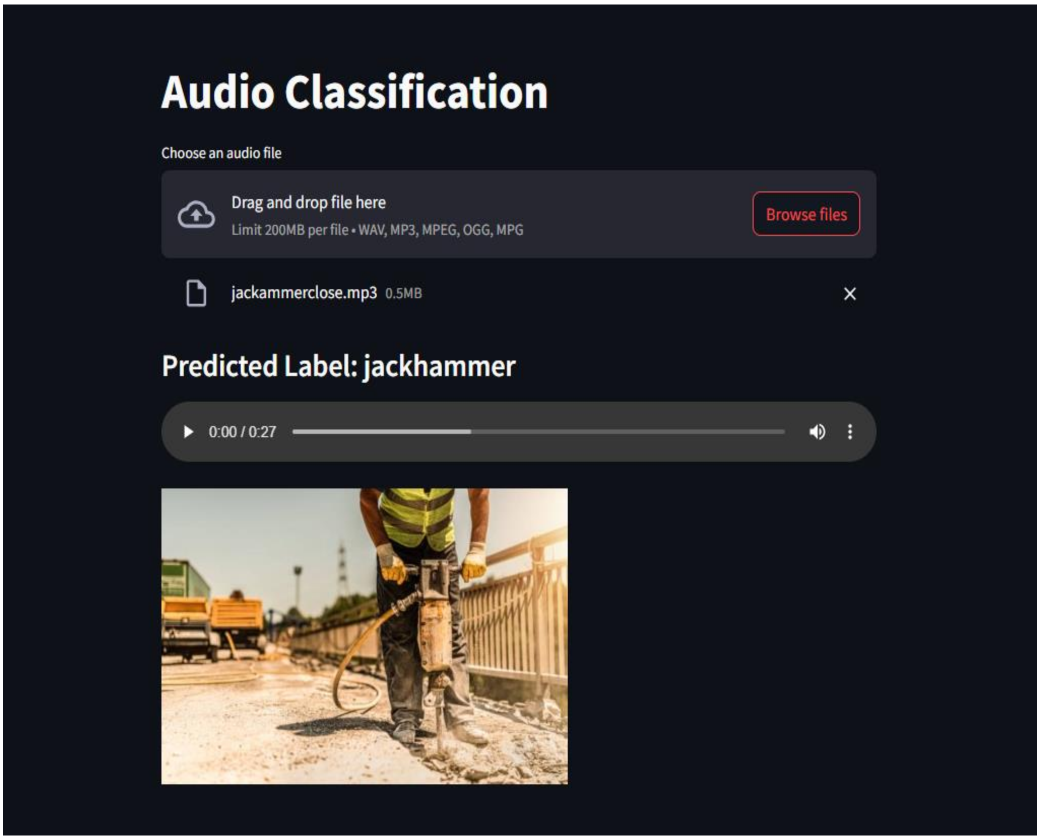
```
                  precision    recall  f1-score   support

 air_conditioner       0.96      0.98      0.97       182
        car_horn       0.95      0.85      0.90        82
children_playing       0.87      0.85      0.86       213
        dog_bark       0.88      0.81      0.84       192
        drilling       0.89      0.95      0.92       204
   engine_idling       0.94      0.98      0.96       208
        gun_shot       0.88      0.84      0.86        80
      jackhammer       0.96      0.99      0.97       203
        laughter       0.98      0.98      0.98       177
           siren       0.95      0.94      0.95       193
    street_music       0.82      0.82      0.82       192

        accuracy                           0.92      1926
       macro avg       0.92      0.91      0.91      1926
    weighted avg       0.91      0.92      0.91      1926
```

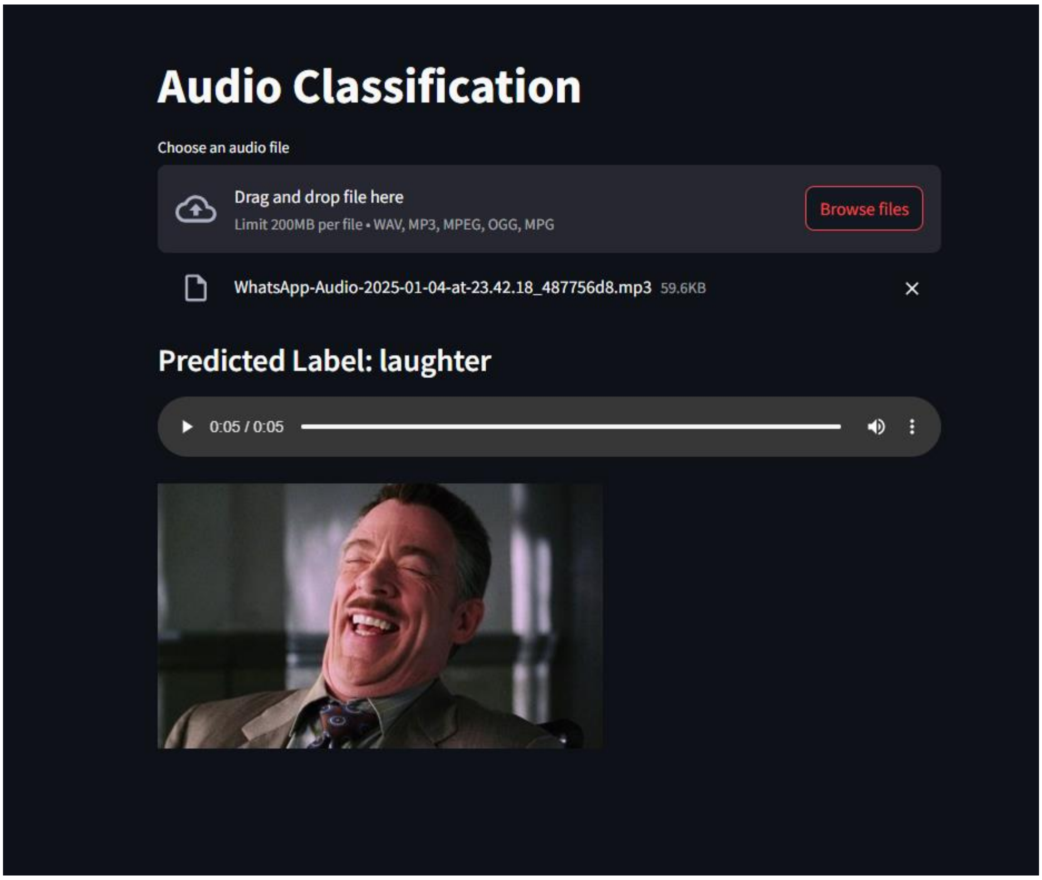
In [111…

```
import pickle
with open('final_audio_classes.pkl', 'wb') as file:
    pickle.dump(labelencoder, file)
```

0 = air_conditioner 1 = car_horn 2 = children_playing 3 = dog_bark 4 = drilling 5 = engine_idling 6 = gun_shot 7 = jackhammer 8 = laughter 9 = siren 10 = street_music


Audio Classification
Choose an audio file
Drag and drop file here
Limit 200MB per file • WAV, MP3, MPEG, OGG, MPG
Browse files
siren_gun_p.wav 0.7MB
Predicted Label: siren
0:00 / 0:04


Audio Classification
Choose an audio file
Drag and drop file here
Limit 200MB per file • WAV, MP3, MPEG, OGG, MPG
Browse files
jackammerclose.mp3 0.5MB
Predicted Label: jackhammer
0:00 / 0:27


Audio Classification
Choose an audio file
Drag and drop file here
Limit 200MB per file • WAV, MP3, MPEG, OGG, MPG
Browse files
WhatsApp-Audio-2025-01-04-at-23.42.18_487756d8.mp3 59.6KB
Predicted Label: laughter
0:05 / 0:05