

第四次实验报告

课程名称	内容安全实验				
学生姓名	陈曦	学号	2020302181081	指导老师	张典
专业	网络安全	班级	2020 级 3 班	实验时间	2023.5.4

一、实验描述

1. 提取任意一段不小于 10s 的 wav 音频的声谱图特征和 MFCC 特征
2. 调用公开语音识别 API，完成语音识别任务
3. 使用深度学习方法，基于 UrbanSound8K 数据集完成音频分类任务

二、实验原理

1. 提取 wav 的音频特征

WAV 失忆中以 RIFF 为基础的无压缩音频编码格式。声谱图是一个信号的频谱随时间变化的直观表示。提取特征就是把音频信号中具有辨识性的成分提取出来，然后把其它的乱七八糟的信息丢弃，例如背景噪声等。

声谱图的主要特征如下。本次实验的重点在“声谱图特征”和“梅尔频率倒谱系数”。

音频特征	特征解释
过零率	每帧信号通过零点的次数
频谱质心	在一定频率范围通过能量加权平均的频率,用于分析音色

音频特征	特征解释
声谱衰减	对声音信号形状的一种衡量,表示低于总频谱能量的指定百分比的频率
色度频率	音乐音频的表示
声谱图特征	表示的是不同频率的信号的振幅随着时间的变化关系
梅尔频率倒谱系数	自动语音识别的主要特征类型

2. 语音识别 API

语音识别 API 是为语言识别提供技术和服务。本次实验使用 Python 调用百度语音识别包 `baidu-api`，完成一段自制语音转文本。创建音频转写任务，查询音频转写任务结果。

3. 使用 Urbansound8K 数据集和深度学习算法完成音频分类任务

[Urbansound8K 数据集]

是目前应用较为广泛的用于自动城市环境分类研究的公共数据集，包含十个分类：空调声，车鸣笛声，儿童玩耍声，狗叫声，钻孔声，引擎空转声，枪声，手提钻声，警笛声，街道音乐声。每个录音长度为 4s，被放在 10 个 fold 文件夹中。

本次实验实现这 10 种语音的分类。

[深度学习算法]

音频分类任务是一个模式识别任务，包括特征选择和分类。特征选择如声谱图和 MFCC，分类有线性分类器和神经网络模型。

三、实验步骤

1. 提取任意一段不小于 10s 的 wav 音频的声谱图特征和 MFCC 特征

[安装 librosa 包]

使用命令 `pip install librosa`，安装包。

```
(base) D:\chenxi\code\neian4>pip install librosa
WARNING: Ignoring invalid distribution -rotobuf (d:\chenxi\try\ide\anaconda\lib\site-packages)
Collecting librosa
  Downloading librosa-0.10.0.post2-py3-none-any.whl (253 kB)
    ━━━━━━━━━━━━━━━━━━━ 253.0/253.0 kB 517.6 kB/s eta 0:00:00
```

[编程实现]

(1) 提取声谱图特征

导入需要的包。`plt` 是画图所用包，`librosa.core` 是计算 stft 所用包，`librosa.display` 为画声谱图所用包，`wavfile` 是用来获取 wav 文件采样率的包。

```
1. import matplotlib.pyplot as plt
2. import librosa.core as lc
3. import numpy as np
4. import librosa.display
5. from scipy.io import wavfile
```

接下来使用 `path` 导入 wav 文件 `test.wav`，并使用 `wavfile` 读取文件的采样率。

设置 FFT 长度为 1024。FFT 是离散傅立叶变换的快速算法，可以将一个信号变换到频域。有些信号在时域上是很难看出什么特征的，但是如果变换到频域之后，就很容易看出特征了。

使用 `librosa.load` 提取 wav 文件。

指定要生成图像的宽高。

```
1. path = "test.wav"
2. fs, y_ = wavfile.read(path)
3. n_fft = 1024
4. y, sr = librosa.load(path, sr=fs)
5. plt.figure(figsize=(17, 8))
```

获取宽带声谱图。宽带语谱图的时宽窄，那么在时间上就“分得开”，即能将语音在时间上重复的部分“看得很清楚”，即表现为“竖线”。“竖”就体现出了时间分辨率高。时间分辨率越高，谱图上的竖线看得越清楚。

先进行短时傅里叶变换，并获取幅度。再将幅度转换为 db 单位。最后画出声谱图。

```
1. mag = np.abs(lc.stft(y, n_fft=n_fft, hop_length=10, win_length=40, window='hamming'))
2. D = librosa.amplitude_to_db(mag, ref=np.max)
3. librosa.display.specshow(D, sr=fs, hop_length=10, x_axis='s', y_axis='linear')
```

使用 `matplotlib` 工具画出声谱图。

```
1. plt.colorbar(format='%+2.0f dB')
2. plt.title('broadband spectrogram', fontsize = 14)
3. plt.tick_params(axis = 'both', which = 'major', labelsize = 13)
4. plt.xlabel('Time', fontsize = 14)
5. plt.ylabel('Hz', fontsize = 14)
6. plt.show()
```

接下来获取窄带声谱图。与宽带相比，设置了 `mag1_log`，改变了 `hop_length` 的设置为 100。

```
1. mag1 = np.abs(lc.stft(y, n_fft=n_fft, hop_length=100, win_length=400, window='hamming'))
2. mag1_log = 20*np.log(mag1)
3. D1 = librosa.amplitude_to_db(mag1, ref=np.max)
4. librosa.display.specshow(D1, sr=fs, hop_length=100, x_axis='s', y_axis='linear')
```

将窄带声谱图展示。

```

1. plt.colorbar(format='%+2.0f dB')
2. plt.title('narrow spectrogram', fontsize=14)
3. plt.tick_params(axis='both', which='major', labelsize=13)
4. plt.xlabel('Time', fontsize=14)
5. plt.ylabel('Hz', fontsize=14)
6. plt.show()

```

(2) 提取 MFCC 特征

导入工具包。scipy.io.wavfile 可以读写 wav 文件。

```

1. import numpy as np
2. import scipy.io.wavfile
3. import matplotlib.pyplot as plt

```

读入 wav 文件 test.wav 采样率和信号。再将音频的前 3.5 秒存储在 signal 中。最后生成与音频对应的时间序列。

```

1. sample_rate, signal = scipy.io.wavfile.read("test.wav")
2. signal = signal[0: int(3.5*sample_rate)]
3. t = np.linspace(0, 3.5, num=len(signal))

```

这段代码的作用是将音频信号分成多个帧，并对每个帧进行窗函数处理，为后续的语音信号处理做准备。设置了多个参数。

```

1. pre_emphasis = 0.97
2. emphasized_signal = np.append(signal[0], signal[1:] - pre_emphasis*signal[:-1])
3. frame_size = 0.025
4. frame_stride = 0.01
5. frame_length, frame_step = frame_size*sample_rate, frame_stride*sample_rate
6. signal_length = len(emphasized_signal)
7. frame_length = int(round(frame_length))
8. num_frames = int(np.ceil(float(np.abs(signal_length - frame_length)) / frame_step))
9. pad_signal_length = num_frames * frame_step + frame_length
10. z = np.zeros(int(pad_signal_length - signal_length))
11. pad_signal = np.append(emphasized_signal, z)
12. indices = np.tile(np.arange(0, frame_length), (num_frames, 1)) + np.tile(np.arange(0, num_frames*frame_step, frame_step), (frame_length, 1)).T

```

```

13. frames = pad_signal[indices.astype(np.int32, copy=False)]
14. ham = np.hamming(frame_length)

```

展示图片。

```

1. plt.plot(ham)
2. plt.show()

```

对每个帧进行短时傅里叶变换处理，得到对应的频谱信息。

```

1. frames *= ham
2. NFFT = 512
3. mag_frames = np.absolute(np.fft.rfft(frames, NFFT))
4. pow_frames = ((1.0 / NFFT) * ((mag_frames) ** 2)) #(348, 257)

```

计算梅尔滤波器组的中心频率，用于将功率谱转换为梅尔频率谱。

```

1. nfilt = 40
2. low_freq_mel = 0
3. high_freq_mel = (2595 * np.log10(1 + (sample_rate/2) / 700))
4. mel_points = np.linspace(low_freq_mel, high_freq_mel, nfilt + 2)
5. hz_points = (700 * (10**(mel_points / 2595) - 1))

```

计算梅尔滤波器组的加权系数，用于将功率谱转换为梅尔频率谱。

```

1. bin = np.floor((NFFT + 1)*hz_points / sample_rate)
2. fbank = np.zeros((nfilt, int(np.floor(NFFT / 2 + 1))))
3. for m in range(1, nfilt + 1):
4.     f_m_minus = int(bin[m-1])
5.     f_m = int(bin[m])
6.     f_m_plus = int(bin[m+1])
7.
8.     for k in range(f_m_minus, f_m):
9.         fbank[m-1, k] = (k-bin[m-1]) / (bin[m]-bin[m - 1])
10.
11.     for k in range(f_m, f_m_plus):
12.         fbank[m-1, k] = (bin[m + 1] - k) / (bin[m + 1] - bin[m])

```

`filter_banks` 数组的形状为 `(num_frames, nfilt)`，其中每个元素表示对应帧的梅尔频率谱值，可以用于后续的特征提取和分析。

```

1. filter_banks = np.dot(pow_frames, fbank.T)
2. filter_banks = np.where(filter_banks == 0, np.finfo(float).eps, filter_banks)

```

```
3. filter_banks = 20 * np.log10(filter_banks)
```

使用 matplotlib 库将梅尔频率谱可视化展示出来。

```
1. plt.figure(figsize=(15,6))
2. plt.imshow(np.flipud(filter_banks.T), cmap=plt.cm.jet, as
   pect=0.2, extent=[0,filter_banks.shape[1],0,filter_banks.s
   hape[0]])
3. plt.axis("on")
4. # plt.savefig('./test2.png')
5. plt.tick_params(axis = 'both', which = 'major', labelsize
   = 13)
6. plt.show()
```

2. 调用公开语音识别 API，完成语音识别任务

[申请应用]

在百度智能云注册并登陆账号。



您已经成功通过个人认证

若您未消费过的新用户，可了解[新人福利](#)，超低价领取云上资源

认证信息

真实姓名： *曦

在应用中找到语音识别，并创建应用，获得 API Key 和 Secret Key 获取 Access_token 来调用服务。

语音技术

概览

应用列表

监控报表

技术文档

API在线调试

SDK下载

离线词&语义设置

离线合成SDK管理

私有部署服务管理

应用列表

1 使用说明

1. 您可通过应用的 API Key 和 Secret Key 获取 Access_token 来调用百度 AI 服务；为了您的财产和服务安全，请妥善保管和定期更新 Secret Key。
2. 快速搜索 AI 全部应用详情，可使用自助工具。[AI 应用详情查询>](#)

创建应用

序号	应用名称	AppID	API Key	Secret Key
1	voice	33279664	2totk... 展开 复制	S7nhH... 展开 复制

可以复制 API Key 和 Secret Key 来调度应用。

[编程实现]

导入需要的包。urllib 库用于操作网页，并对网页的内容进行抓取处理。在这里

使用 urllib 包的三个模块：

request 模块用于打开和读取 URL；

error 模块包含 request 模块抛出的异常；

parse 模块用于解析 URL。

ast 库是一个用于处理代码抽象语法树（AST）的库。

```
1. import json
2. import base64
3. import time
4. import os
5. from urllib.request import urlopen
6. from urllib.request import Request
7. from urllib.error import URLError
8. from urllib.parse import urlencode
9. import ast
10. timer = time.perf_counter
11. import requests
```


定义与百度语音识别 API 的相关变量。定义 Key 和 Secert Key，并复制值到变量中；设置 token，stt，asr 的 URL，分别用于获取访问令牌、请求将语音转化为文本、请求语音识别 API。DEV_PID 制定了要识别语音的语言设备为英语。

```
1. API_KEY = '2totkLuyZieGGc1zxBtyCb1q'
2. SECRET_KEY = 'S7nhHZIlc4NIagZnmFct0rsLS9B0yYju' # 自行补充
3. TOKEN_URL = 'http://aip.baidubce.com/oauth/2.0/token' # 接口功能URL
4. STT_URL = 'https://aip.baidubce.com/rpc/2.0/aasr/v1/create' # 长文本的转写请求url
5. RATE = 16000
6. DEV_PID = 1737 # 1737 英文, 1537 中文
7. ASR_URL = 'http://vop.baidu.com/server_api'
8. SCOPE = 'audio_voice_assistant_get'
9. CUID = '123456PYTHON'
10. Format = ['pcm', 'wav', 'amr']
```

自定义了名为 DemoError 的异常类。

```
1. class DemoError(Exception):
2.     pass
```

对格式进行转换，如果是 mp3 格式，会被转换成 wav 格式。

```
1. def converta(d=None):
2.     origin_path = d
3.     # new_path = d + f[: -4] + ".mp3"
4.     x = 3
5.     if (d[-5] == '.'):
6.         x = 4
7.     if (d[-4] == '.'):
8.         x = 3
9.     new_path = d[: -x] + "wav"
10.    os.system("ffmpeg -i " + origin_path + " " + new_path
11.    )
11.    return new_path
```

获取访问令牌并返回该令牌。如果获取失败则引发上述自定义异常 DmoError。

```
1. def fetch_token():
2.     params = {'grant_type': 'client_credentials',
3.               'client_id': API_KEY,
4.               'client_secret': SECRET_KEY}
5.     post_data = urlencode(params).encode('utf-8')
```

```

6.         req = Request(TOKEN_URL, post_data)
7.         try:
8.             f = urlopen(req)
9.             result_str = f.read()
10.        except URLError as err:
11.            print('token http response http code : ' + str(er
r.code))
12.            result_str = err.read().decode()
13.            result = json.loads(result_str)
14.            if ('access_token' in result.keys() and 'scope' in re
sult.keys()):
15.                if SCOPE and (not SCOPE in result['scope'].split(
' ')): # SCOPE = False 忽略检查
16.                    raise DemoError('scope is not correct')
17.                print('SUCCESS WITH TOKEN: %s  EXPIRES IN SECONDS:
%s' % (result['access_token'], result['expires_in']))
18.                return result['access_token'] # access token
19.            else:
20.                raise DemoError('MAYBE API_KEY or SECRET_KEY not
correct: access_token or scope not found in token response
')

```

获取发送格式。将音频文件转换为可用于发送请求的格式。并创建一个包含请求参数的字典。包含如语音类型，采样率，用户唯标识等。

```

1.  def tok(file):
2.      if file[-3:] not in Format:
3.          readfile = converta(file)
4.      else :
5.          readfile = file
6.      with open(readfile, "rb") as speech_file:
7.          speech_data = speech_file.read()
8.          leng = len(speech_data)
9.          if (leng == 0):
10.             raise DemoError('file %s length read 0 bytes' % r
eadfile)
11.          speech = str(base64.b64encode(speech_data), "utf-8")
12.          params = {
13.              'dev_pid': DEV_PID, # 语音类
              型 DEV_PID = 1737 表示英文, 1537 中文
14.              #"lm_id" : LM_ID,      #测试自训练平台开启此项
15.              'format': readfile[-3:], # 格式, 支持
              pcm/wav/amr/m4a
16.              'rate': RATE, # 采样率

```

```

17.         'token': token, # 权限 token
18.         'cuid': CUID, # 用户唯一标识
19.         'channel': 1, # 声道
20.         'speech': speech, # 语音数据 b64 解析
21.         'len': leng # 长度
22.     }
23.     post_data = json.dumps(params, sort_keys=False)
24.     return post_data

```

向百度语音识别 API 发出 POST 请求，并返回识别结果。

```

1. def requ(post_data):
2.     req = Request(ASR_URL, post_data.encode('utf-8'))
3.     req.add_header('Content-Type', 'application/json')
4.     try:
5.         begin = timer()
6.         f = urlopen(req)
7.         # print(f)
8.         result_str = f.read()
9.         # print ("Request time cost %f" % (timer() - begin))
10.    except URLError as err:
11.        print('asr http response http code : ' + str(err.code))
12.        result_str = err.read()
13.        # print(result_str)
14.        result_str = str(result_str, 'utf-8')
15.        result = ast.literal_eval(result_str)
16.        # print(result) # 转换为字典
17.        # print(result['result']) # 文字部分

```

查询百度语音识别 API 异步转写任务的结果，并打印是否成功。

```

1. def sttquery(stt_task_list):
2.     print(stt_task_list)
3.     for task_id in stt_task_list:
4.         url = 'https://aip.baidubce.com/rpc/2.0/aasr/v1/query' # 查询音频任务转写结果请求地址
5.         body = {
6.             "task_ids": [task_id],
7.         }
8.         token = {"access_token": fetch_token()}
9.         headers = {'content-type': "application/json"}
10.        response = requests.post(url, params=token, data=json.dumps(body), headers=headers)

```

```
11.         print(json.dumps(response.json(), ensure_ascii=False))
```

提交异步撰写任务。并返回该任务的 ID。

```
1.  def sttbody(audio):
2.      with open(audio, "rb") as speech_file:
3.          speech_data = speech_file.read()
4.          s = str(base64.b64encode(speech_data), "utf-8")
5.          body = {
6.              "speech_url": s,
7.              "format": audio[-3:],
8.              "pid":
9.              1737, # 中文1537 英文1737
10.             "rate": 16000
11.         }
12.         token = {"access_token": fetch_token()}
13.         headers = {'content-type': "application/json"}
14.         respons = requests.post(STT_URL, params=token, data=json.dumps(body), headers=headers)
15.         results = ast.literal_eval(respons.text)
16.         print(results, results["task_id"])
17.         # stt_task_list.append(results["task_id"])
18.         # print(stt_task_list)
```

批量处理 test 文件夹下的音频文件，并将文本转换结果写入 result.txt 中。

```
1.  def grouptts(d):
2.      audios = os.listdir(d)
3.      res = open(d + "result.txt", "w", encoding="utf-8")
4.      for a in audios:
5.          if a[-3:] == 'txt' :
6.              continue
7.          patha = d + a
8.          post_req = tok(patha)
9.          text = requ(post_req)
10.         print(a, text)
11.         res.write(a + "\t" + text + "\n")
12.     res.close()
```

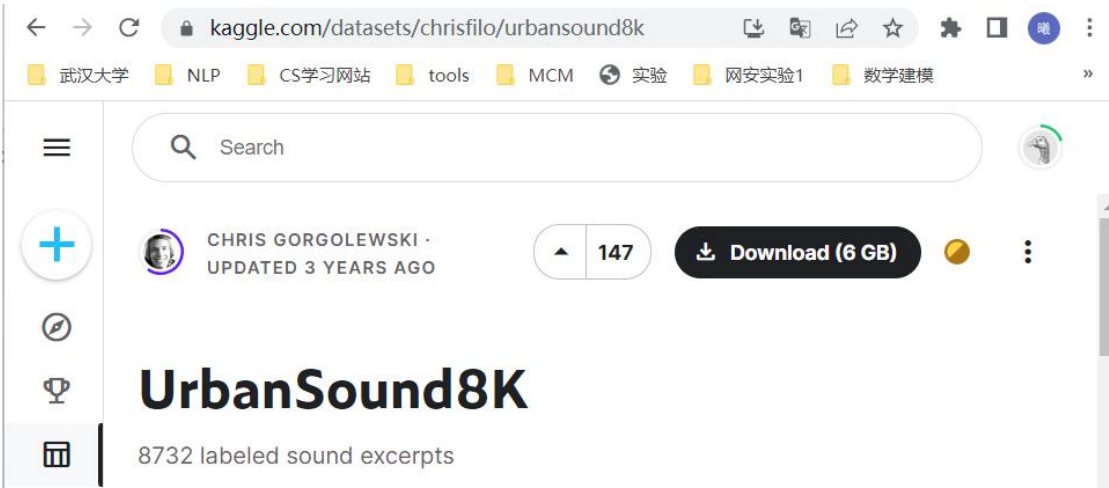
命令执行入口，调用 fetch_token 函数，并设置音频文件夹为 test。

```
1.  if __name__ == "__main__":
2.      token = fetch_token()
3.      dir = "./test/"
4.      grouptts(dir)
```

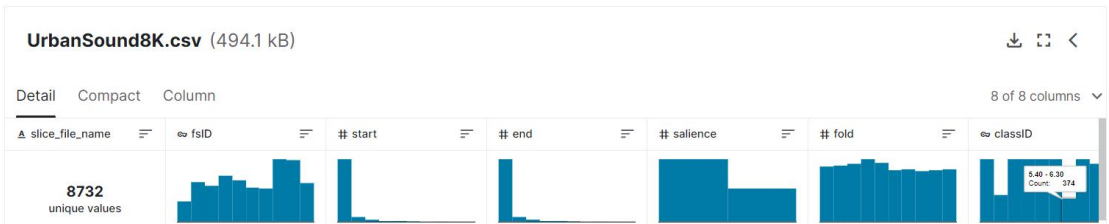
3. 使用深度学习方法，基于 UrbanSound8K 数据集完成音频分类任务

[下载数据集]

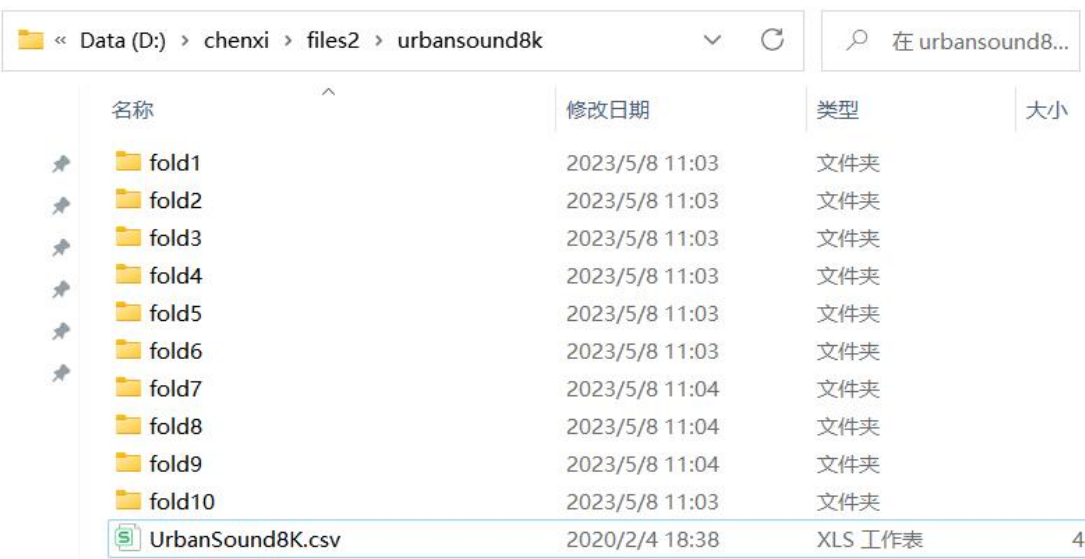
在 Kaggle 网站上下载 UrbanSound8K 数据集。



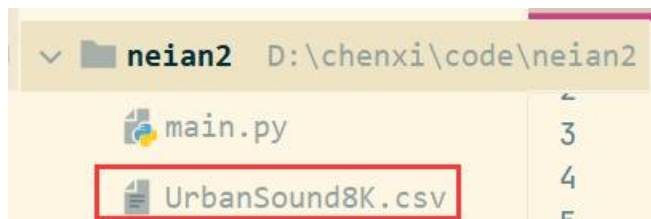
数据集结构如下。



下载完成后解压到目标文件夹。



将 UrbanSound8K.csv 文件放到项目文件夹中。



[编程实现——tensorflow]

(1) 查看数据集

在 main.py 文件中，查看数据集前 5 行数据。

```
1. import pandas as pd
2.
3. data = pd.read_csv('UrbanSound8K.csv')
4. content = data.head()
5. print(content)
```

查看结果。

```
D:\chenxi\try\ide\anaconda\python.exe D:/chenxi/code/neian2/main.py

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

[5 rows x 8 columns]
```

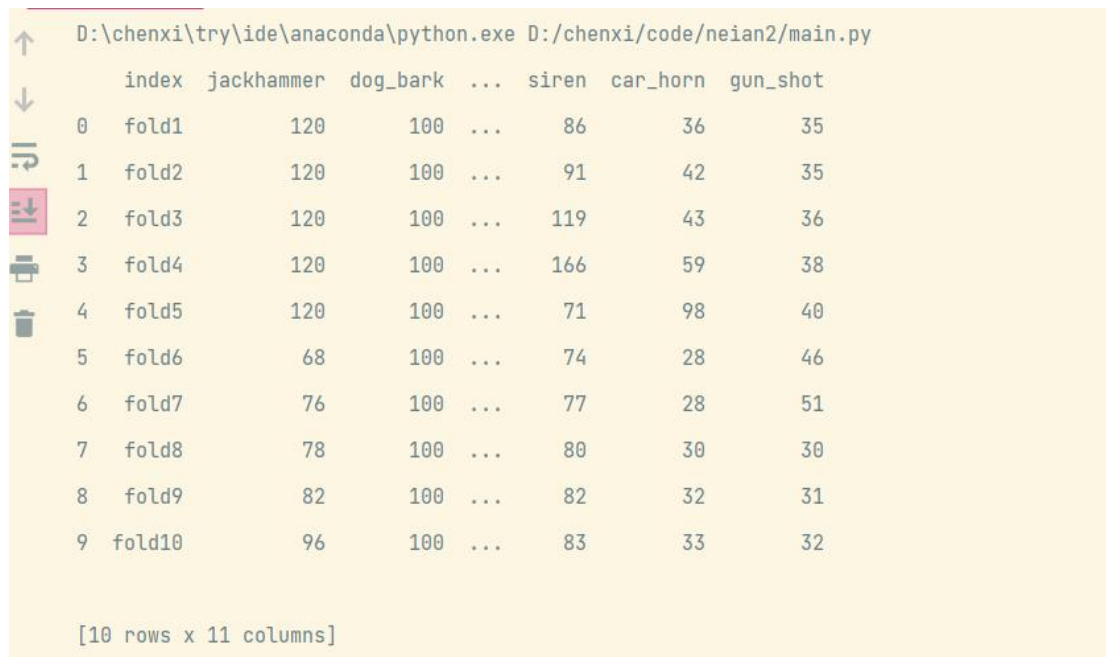
进程已结束,退出代码0

查看各个文件夹中的声音分布情况。

```
1. appended = []
2. for i in range(1, 11):
3.     appended.append(data[data.fold == i]['class'].value_counts())
4.
5. class_distribution = pd.DataFrame(appended)
6. class_distribution = class_distribution.reset_index()
7. class_distribution['index'] = ["fold" + str(x) for x in range(1, 11)]
```

8. `print(class_distribution)`

查看运行结果。



	index	jackhammer	dog_bark	...	siren	car_horn	gun_shot
0	fold1	120	100	...	86	36	35
1	fold2	120	100	...	91	42	35
2	fold3	120	100	...	119	43	36
3	fold4	120	100	...	166	59	38
4	fold5	120	100	...	71	98	40
5	fold6	68	100	...	74	28	46
6	fold7	76	100	...	77	28	51
7	fold8	78	100	...	80	30	30
8	fold9	82	100	...	82	32	31
9	fold10	96	100	...	83	33	32

[10 rows x 11 columns]

(2) 音频文件可视化

可视化 wav 文件。在程序输入入口输入需要查看图谱的 wav 音频文件，就可以使用 `path_class` 函数在本机上找到 UrbanSound8K 数据集所在的位置并找到该文件，再使用 `wav_plotter` 函数就可以绘制出图谱，并在终端打印音频特征。

```
1. # 读取wav 文件函数
2. def path_class(filename):
3.     excerpt = data[data['slice_file_name'] == filename]
4.     path_name = os.path.join('D:/chenxi/files2/urbansound
      8k', 'fold'+str(excerpt.fold.values[0]), filename)
5.     return path_name, excerpt['class'].values[0]
6. # 绘图wav 函数
7. def wav_plotter(full_path, class_label):
8.     rate, wav_sample = wavfile.read(full_path)
9.     wave_file = open(full_path, "rb")
10.    riff_fmt = wave_file.read(36)
11.    bit_depth_string = riff_fmt[-2:]
12.    bit_depth = struct.unpack("H",bit_depth_string)[0]
13.    print('sampling rate: ',rate,'Hz')
14.    print('bit depth: ',bit_depth)
```

```

15.     print('number of channels: ',wav_sample.shape[1])
16.     print('duration: ',wav_sample.shape[0]/rate,' second'
    )
17.     print('number of samples: ',len(wav_sample))
18.     print('class: ',class_label)
19.     plt.figure(figsize=(12, 4))
20.     plt.plot(wav_sample)
21.     print("SUCCESS")
22.     plt.show()
23.     return ipd.Audio(full_path)
24. if __name__ == "__main__":
25.     fullpath, label = path_class('100263-2-0-117.wav')
26.     wav_plotter(fullpath, label)

```

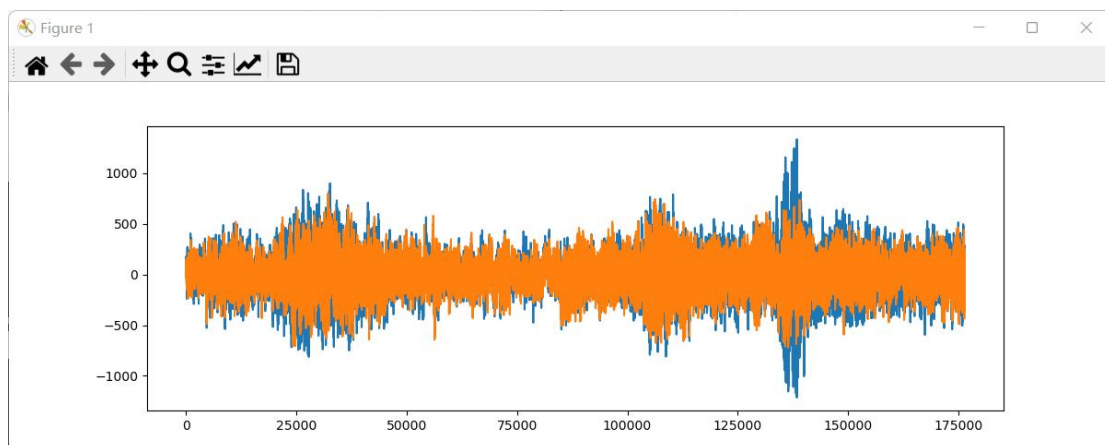
函数运行终端结果，可以查看该音频特征。

```

D:\chenxi\try\ide\anaconda\python.exe D:/chenxi/code/neian2/main.py
sampling rate: 44100 Hz
bit depth: 16
number of channels: 2
duration: 4.0 second
number of samples: 176400
class: children_playing
SUCCESS

```

函数运行结果，可以可视化该音频文件。



(3) 训练 tensorflow 模型

导入工具包。Tensorflow 用于模型训练。


```

1. import os
2. import random
3. import librosa
4. import pandas as pd
5. from tqdm import tqdm
6. import tensorflow as tf
7. import numpy as np
8. class_dim = 10
9. EPOCHS = 100
10. BATCH_SIZE = 32
11. init_model = None

```

创建训练集用于生成 tensorflow。

```

1. def get_urbansound8k_list(path, urbansound8k_cvs_path):
2.     data_list = []
3.     data = pd.read_csv(urbansound8k_cvs_path)
4.     # 过滤掉长度少于 3 秒的音频
5.     valid_data = data[['slice_file_name', 'fold', 'classID', 'class']][data['end'] - data['start'] >= 3]
6.     valid_data['path'] = 'fold' + valid_data['fold'].astype('str') + '/' + valid_data['slice_file_name'].astype('str')
7.     for row in valid_data.itertuples():
8.         data_list.append([row.path, row.classID])
9.
10.    f_train = open(os.path.join(path, 'train_list.txt'), 'w')
11.    f_test = open(os.path.join(path, 'test_list.txt'), 'w')
12.
13.    for i, data in enumerate(data_list):
14.        sound_path = os.path.join('UrbanSound8K/', data[0])
15.        if i % 100 == 0:
16.            f_test.write('%s\t%d\n' % (sound_path, data[1]))
17.        else:
18.            f_train.write('%s\t%d\n' % (sound_path, data[1]))
19.
20.    f_test.close()
21.    f_train.close()

```

生成 TensorflowRecord 文件。

获取浮点数组。

```
1. def _float_feature(value):
2.     if not isinstance(value, list):
3.         value = [value]
4.     return tf.train.Feature(float_list=tf.train.FloatList
        (value=value))
```

获取整型数据。

```
1. def _int64_feature(value):
2.     if not isinstance(value, list):
3.         value = [value]
4.     return tf.train.Feature(int64_list=tf.train.Int64List
        (value=value))
```

把数据添加到 TensorflowRecord 中。

```
1. def data_example(data, label):
2.     feature = {
3.         'data': _float_feature(data),
4.         'label': _int64_feature(label),
5.     }
6.     return tf.train.Example(features=tf.train.Features(fe
        ature=feature))
```

开始创建 TensorflowRecord 数据。

```
1. def create_data_tfrecord(data_list_path, save_path):
2.     with open(data_list_path, 'r') as f:
3.         data = f.readlines()
4.         with tf.io.TFRecordWriter(save_path) as writer:
5.             for d in tqdm(data):
6.                 try:
7.                     path, label = d.replace('\n', '').split('
                        \t')
8.                     wav, sr = librosa.load(path, sr=16000)
9.                     intervals = librosa.effects.split(wav, to
                        p_db=20)
10.                    wav_output = []
11.                    wav_len = int(16000 * 2.04)
12.                    for sliced in intervals:
13.                        wav_output.extend(wav[sliced[0]:slice
                        d[1]])
14.                    for i in range(5):
```

```

15.             if len(wav_output) > wav_len:
16.                 l = len(wav_output) - wav_len
17.                 r = random.randint(0, l)
18.                 wav_output = wav_output[r:wav_len
+ r]
19.             else:
20.                 wav_output.extend(np.zeros(shape=
[wav_len - len(wav_output)], dtype=np.float32))
21.                 wav_output = np.array(wav_output)
22.                 # 转成梅尔频谱
23.                 ps = librosa.feature.melspectrogram(y
=wav_output, sr=sr, hop_length=256).reshape(-1).tolist()
24.                 if len(ps) != 128 * 128: continue
25.                 tf_example = data_example(ps, int(label))
26.                 writer.write(tf_example.SerializeToString())
27.                 if len(wav_output) <= wav_len:
28.                     break
29.             except Exception as e:
30.                 print(e)

```

读取 TensorflowRecord 文件数据。

```

1.  def _parse_data_function(example):
2.
3.      data_feature_description = {
4.          'data': tf.io.FixedLenFeature([16384], tf.float32)
5.      ,
6.          'label': tf.io.FixedLenFeature([], tf.int64),
7.      }
8.      return tf.io.parse_single_example(example, data_feature_description)
9.
10. def train_reader_tfrecord(data_path, num_epochs, batch_size):
11.     raw_dataset = tf.data.TFRecordDataset(data_path)
12.     train_dataset = raw_dataset.map(_parse_data_function)
13.     train_dataset = train_dataset.shuffle(buffer_size=1000) \
14.         .repeat(count=num_epochs) \
15.         .batch(batch_size=batch_size) \
16.         .prefetch(buffer_size=tf.data.experimental.AUTOTUNE)

```

```

17.     return train_dataset
18.
19.
20. def test_reader_tfrecord(data_path, batch_size):
21.     raw_dataset = tf.data.TFRecordDataset(data_path)
22.     test_dataset = raw_dataset.map(_parse_data_function)
23.     test_dataset = test_dataset.batch(batch_size=batch_size)
24.     return test_dataset

```

程序运行入口。运行并计算准确率损失值等，最后保存模型。

```

1.  if __name__ == '__main__':
2.
3.     '''生成数据列表'''
4.     get_urbansound8k_list('UrbanSound8K/', 'UrbanSound8K.csv')
5.
6.     '''生成 tfrecord'''
7.     create_data_tfrecord('UrbanSound8K/train_list.txt', 'UrbanSound8K/train.tfrecord')
8.     create_data_tfrecord('UrbanSound8K/test_list.txt', 'UrbanSound8K/test.tfrecord')
9.
10.    '''训练'''
11.    model = tf.keras.models.Sequential([
12.        tf.keras.applications.ResNet50V2(include_top=False, weights=None, input_shape=(128, None, 1)),
13.        tf.keras.layers.ActivityRegularization(l2=0.5),
14.        tf.keras.layers.Dropout(rate=0.5),
15.        tf.keras.layers.GlobalMaxPooling2D(),
16.        tf.keras.layers.Dense(units=class_dim, activation=tf.nn.softmax)
17.    ])
18.
19.    model.summary()
20.
21.    # 定义优化方法
22.    optimizer = tf.keras.optimizers.Adam(learning_rate=1e-3)
23.
24.    train_dataset = train_reader_tfrecord('UrbanSound8K/train.tfrecord', EPOCHS, batch_size=BATCH_SIZE)
25.    test_dataset = test_reader_tfrecord('UrbanSound8K/test.tfrecord', batch_size=BATCH_SIZE)

```

```

26.
27.     if init_model:
28.         model.load_weights(init_model)
29.
30.     for batch_id, data in enumerate(train_dataset):
31.
32.         sounds = data['data'].numpy().reshape((-1, 128, 1
33.         28, 1))
34.         labels = data['label']
35.         # 执行训练
36.         with tf.GradientTape() as tape:
37.             predictions = model(sounds)
38.             # 获取损失值
39.             train_loss = tf.keras.losses.sparse_categorical_crossentropy(labels, predictions)
40.             train_loss = tf.reduce_mean(train_loss)
41.             # 获取准确率
42.             train_accuracy = tf.keras.metrics.sparse_categorical_accuracy(labels, predictions)
43.             train_accuracy = np.sum(train_accuracy.numpy()) / len(train_accuracy.numpy())
44.
45.             # 更新梯度
46.             gradients = tape.gradient(train_loss, model.trainable_variables)
47.             optimizer.apply_gradients(zip(gradients, model.trainable_variables))
48.
49.             if batch_id % 20 == 0:
50.                 print("Batch %d, Loss %f, Accuracy %f" % (batch_id, train_loss.numpy(), train_accuracy))
51.
52.                 if batch_id % 200 == 0 and batch_id != 0:
53.                     test_losses = list()
54.                     test accuracies = list()
55.                     for d in test_dataset:
56.
57.                         test_sounds = d['data'].numpy().reshape((-1, 128, 128, 1))
58.                         test_labels = d['label']
59.
60.                         test_result = model(test_sounds)
61.                         # 获取损失值

```

```

61.         test_loss = tf.keras.losses.sparse_categorical_crossentropy(test_labels, test_result)
62.         test_loss = tf.reduce_mean(test_loss)
63.         test_losses.append(test_loss)
64.         # 获取准确率
65.         test_accuracy = tf.keras.metrics.sparse_categorical_accuracy(test_labels, test_result)
66.         test_accuracy = np.sum(test_accuracy.numpy()) / len(test_accuracy.numpy())
67.         test accuracies.append(test_accuracy)
68.
69.         print('=====')
70.         print("Test, Loss %f, Accuracy %f" % (
71.             sum(test_losses) / len(test_losses), sum(
72.                 test accuracies) / len(test accuracies)))
73.         print('=====')
74.         # 保存模型
75.         model.save(filepath='files/resnet50.h5')
76.         model.save_weights(filepath='files/model_weights.h5')

```

(4) 测试 tensorflow 模型

导入需要的包。

```

1. import librosa
2. import numpy as np
3. import tensorflow as tf

```

读入训练模型。

```

1. model = tf.keras.models.load_model('files/resnet50.h5')

```

读取音频数据。

```

1. def load_data(data_path):
2.     wav, sr = librosa.load(data_path, sr=16000)
3.     intervals = librosa.effects.split(wav, top_db=20)
4.     wav_output = []
5.     for sliced in intervals:
6.         wav_output.extend(wav[sliced[0]:sliced[1]])
7.     assert len(wav_output) >= 8000, "有效音频小于 0.5s"

```

```

8.     wav_output = np.array(wav_output)
9.     ps = librosa.feature.melspectrogram(y=wav_output, sr=
sr, hop_length=256).astype(np.float32)
10.    ps = ps[np.newaxis, ..., np.newaxis]
11.    return ps
12.
13.
14. def infer(audio_path):
15.     data = load_data(audio_path)
16.     result = model.predict(data)
17.     lab = tf.argmax(result, 1)
18.     return lab

```

输入要预测的音频文件。可以查看到预测结果标签。

```

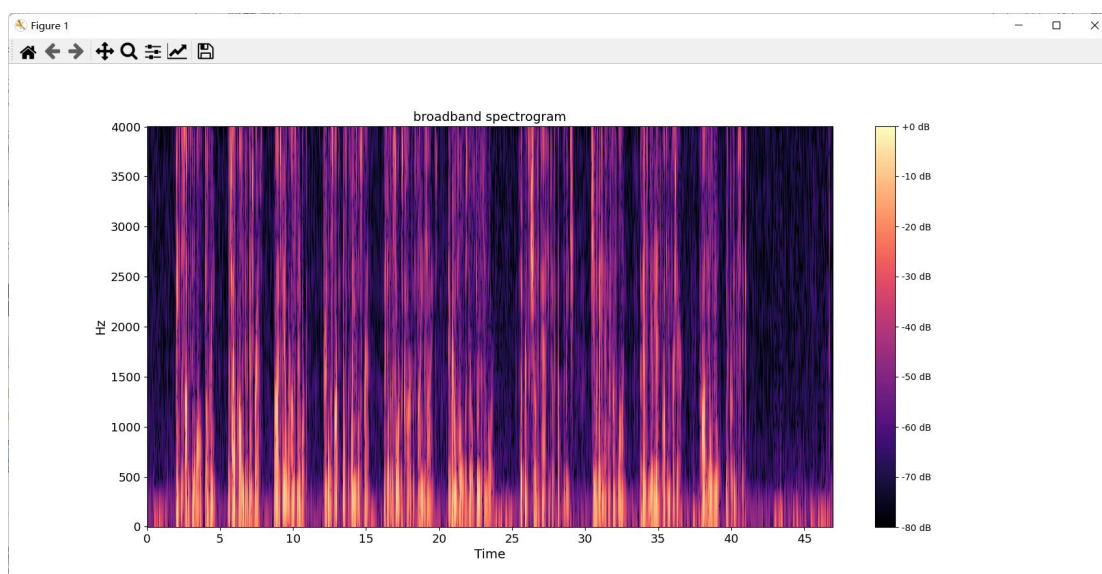
1.  if __name__ == '__main__':
2.     path = 'UrbanSound8K/fold1/7383-3-0-0.wav'
3.     label = infer(path)
4.     print('音频: %s 的预测结果标签为: %d' % (path, label))

```

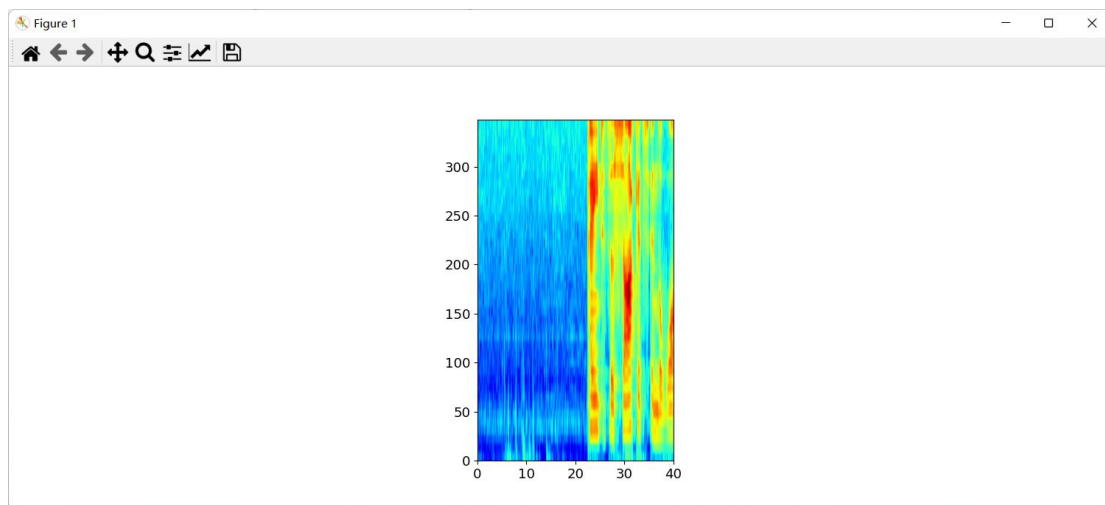
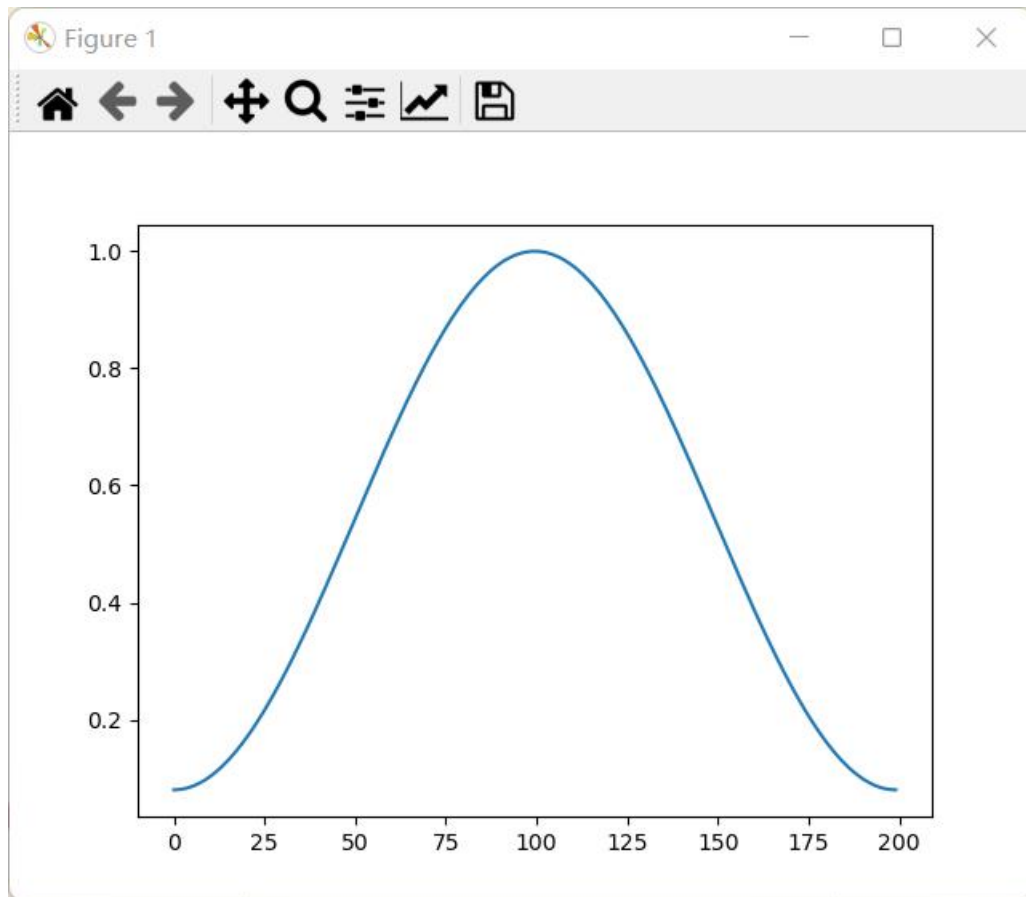
四、实验结果

1. 提取任意一段不小于 10s 的 wav 音频的声谱图特征和 MFCC 特征

提取声谱图如下所示：



提取 MFCC 特征如下所示：



2. 调用公开语音识别 API，完成语音识别任务

语音识别结果如下所示：


```
main1.py x main1.2.py x result.txt x main2.py x
1 83-3054-0000.wav a woman's life saved the cause of her falling into the sea
2 83-3054-0001.wav I omitted several very material parts in my father's journey across the English channel to Holland which that they
3 83-3054-0002.wav I will now faithfully give you in his own words as I heard him relate them to his friends several times on my arrival says my
4 83-3054-0003.wav carrying their heads at the extremity of their tails I crossed continued he one prodigious range of rocks equal in height to
5 83-3054-0004.wav are said to be upwards of one hundred fathoms below the surface of the sea on the sides of which there was a great variety of
6 83-3054-0005.wav fruit shook off the branches of the tree it grows upon by the motion of the water as those in our gardens are by that of the
7 83-3054-0006.wav the periwinkle is a kind of shrub it grows at the foot of the oyster tree and twines round it as the ivy does the oak
8 83-3054-0007.wav as she sank she fell upon her side and forced a very large lobster tree out of its place it was in the spring when the lobster
9 83-3054-0008.wav and many of them being separated by the violence of the shock they fell upon a Crabtree which was growing below them they have
10 83-3054-0009.wav I endeavoured to bring one with me but it was too cumbersome and my salt water Pegasus seemed much displeased at every attempt
11 83-3054-0010.wav besides I was then though galloping over a mountain of rocks that lay about midway the passage at least five hundred fathom b
12 83-3054-0011.wav therefore I had to no inclination to prolong the time add to this my situation was in other respects very unpleasant I am at
13 83-3054-0012.wav not only able but really wished to devour us now as my rose and auntie was blind I had to these hungry gentleman's attempts t
14 83-3054-0013.wav as we drew near the Dutch shore and the body of water over our heads did not exceed twenty fathoms I thought I saw a human fi
15 83-3054-0014.wav when I came close I perceived her hand move I took it in to mine and brought her on shore as a corpse an apothecary who had j
16 83-3054-0015.wav of London treated her properly and she recovered was the rib of a man who commanded a vessel belonging to hell that's lit he
17 83-3054-0016.wav followed him in an open boat as soon as she had got on the quarterdeck she flew at her husband and attempted to strike him wi
18 83-3054-0017.wav and let her make the impression of her fingers upon the waves rather than his face he was not much out in his ideas of the co
19 83-3054-0018.wav and it was my unfortunate lot to lay the foundation for bringing this happy pair together again I can easily conceive what e
20 83-3054-0019.wav he found this gentle creature waiting his arrival and learned the means by which she came into the world again
21
```

3. 使用深度学习方法，基于 UrbanSound8K 数据集完成音频分类任务

分类结果如下所示：

```
D:\chenxi\try\ide\anaconda\python.exe D:\chenxi\code\test\runme.py
1/1 [=====] - 1s 1s/step
音频：UrbanSound8K/fold1/7383-3-0-0.wav 的预测结果标签为：3

进程已结束,退出代码0
```

五. 实验心得

本次实验我学习了 wav 音频文件格式，并了解音频文件各个特征的概念，并实现了声谱图的输出；学习了 MFCC 的原理和方法，实现了 MFCC 特征的输出。

除此之外，申请了百度云应用的权限，使用它们对十个音频进行了识别。

最后使用 Tensorflow 算法对语音进行了分类。

本次实验对内容安全的知识有了更深的了解，并初步学习和实现了有关音频处理的相关技术。