第 三 次 实 验 报 告

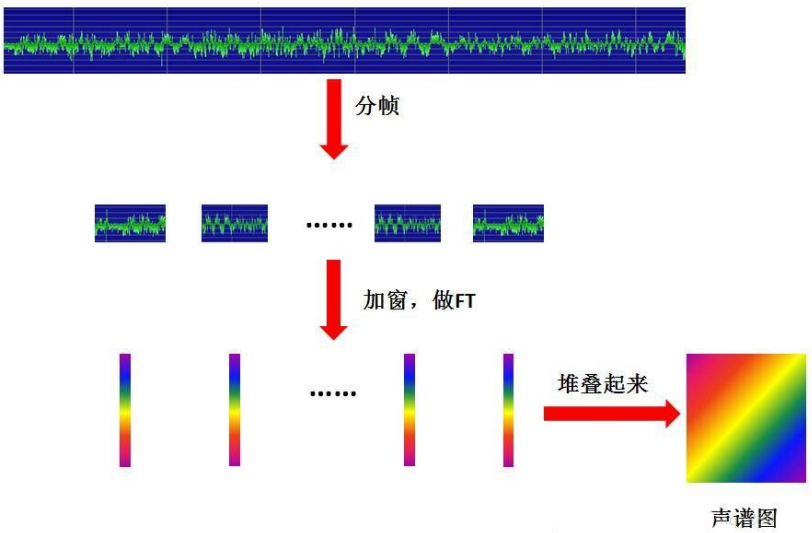
课程名称	内容安全实验				
学生姓名		学号		指导老师	**熊翘楚**
专业	**网络空间安全**	班级		实验时间	2022. 04. 07

一、实验内容

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

声音信号是一维信号，直观上只能看到时域信息，而看不到频域信息。声谱图是声音或其他信号的频率随时间变化时的频谱的一种直观表示。通过傅里叶变换将时域信号转换到频域上，但这样会失去时域信息，无法得到频率随时间分布的变化。短时傅里叶变化(STFT)是为了解决这个问题而发明出来的方法，短时傅里叶变换把一段长信号分帧、加窗，再对每一帧做快速傅里叶变换(FFT)，最后把每一帧的结果沿另一个维度堆叠起来，得到类似于一幅图的二维信号形式。



其中，关于分帧，为了便于处理，相邻的帧之间需要有一定的重合，通常以25ms 为 1 帧，帧移为 10ms，因此 1s 的信号会有 10 帧。然后可以使用离散傅里叶变换(DFT)把每一帧信号变换到时域，公式如下所示。其中 si(n)是第 i 帧的时域信号，Si(k)表示的是第 i 帧的第 k 个复系数，h(n)是一个 n 点的窗函数，k 是 DFT 的长度。

$$S_i(k) = \sum_{n=1}^{N} s_i(n)h(n)e^{-j2\pi kn/N} 1 \le k \le N$$

功率谱可以用如下公式计算得到，Pi(k)是第 i 帧的功率谱。得到的随时间变化的功率谱即为声谱图。

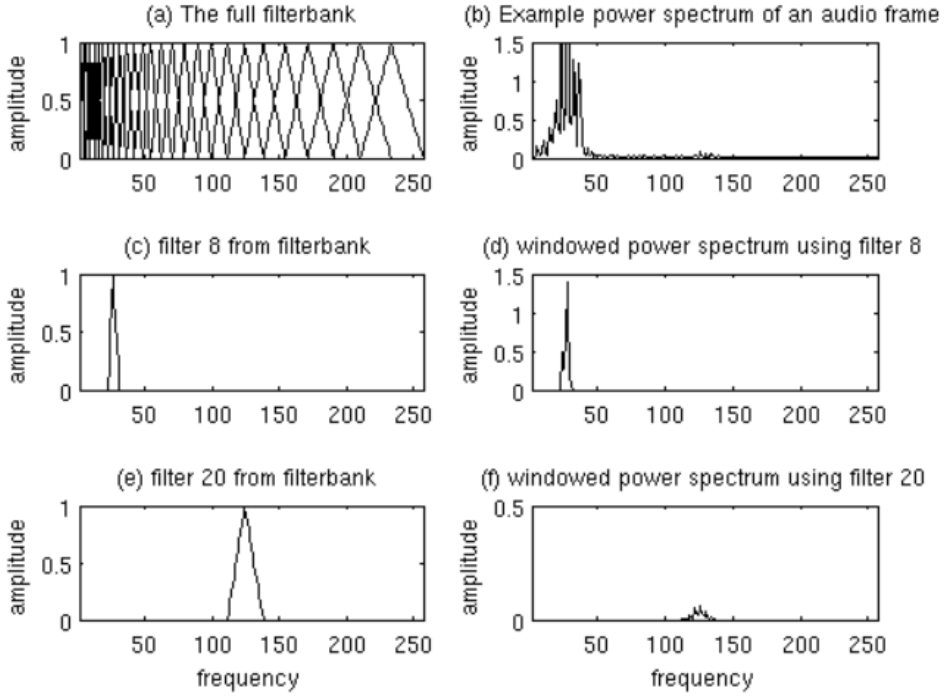
$$P_i(k) = \frac{1}{N}|S_i(k)|^2$$

梅尔倒谱系数(MFCC)特征提取主要包含以下四个步骤：对语音信号进行分帧处理；计算得到功率谱估计，即声谱图；对功率谱用 Mel 滤波器组进行滤波，得到 Mel 频谱；在 Mel 频谱上面进行倒谱分析，从而得到 MFCC 特征。

梅尔尺度是建立从人类的听觉感知的频率到声音实际频率的直接映射。人耳对频率的感知并不是线性的，比如音调频率在数值上上升一倍，人耳只能够感觉到频率提高一些而不能感觉到上升一倍，因此使用梅尔频率就能更好匹配人类的听觉感知结果。从频率到梅尔频率的转换公式如下所示。

$$M(f) = 1125ln(1 + f/700)$$

为了模拟人耳对声音的感知，梅尔滤波器组被发明了出来。一组一般有 26 个三角滤波器组，它会对上一步得到的周期图的功率谱估计进行滤波。滤波过程如下图所示，图(a)是 26 个滤波器，图(b)是滤波后的信号，图(c)是其中的 8 个滤波器，它只让其中某一频率范围的信号通过，图(d)是通过它的信号的能量，图(e)是第 20 个滤波器，图(f)是通过它的信号的能量。



在梅尔频谱上做倒谱分析，即取对数，做离散余弦变换(DCT)，就得到了梅尔倒谱。对上述滤波操作得到的 26 个点的信号进行 DCT，得到 26 个倒谱系数，最后保留 2-13 这 12 个数字，这 12 个数字就被称为 MFCC 特征。其中做 DCT 的原因是不同的 Mel 滤波器是有交集的，因此它们是相关的，使用 DCT 可以去掉这些相关性。而关于只取 2-13 个数字作为特征的原因是后面的能量表示的是变化很快的高频信号，前面的能量表示的是变换很快的低频信号，在实践中发现识别效果相对不好，而中间的 2-13 个数字更具有区分性，能够作为特征使用。

倒谱具体计算过程可以分为以下六步，假设上面的频率谱 X(k)，时域信号为 x(n)那么满足：X(k)=DFT(x(n))；考虑将频域 X(k)拆分为两部分的乘积：X(k)=H(k)E(k)；假设两部分对应的时域信号分别是 h(n)和 e(n)，那么满足：x(n)=h(n)*e(n)；对频域两边取 log：log(X(k))=log(H(k))+log(E(k))；然后进行反傅里叶变换：IDFT(log(X(k)))=IDFT(log(H(k)))+IDFT(log(E(k)))；得到的时域信号如下：x(n)=h(n)+e(n) 。此时获得的时域信号 x(n) 即为倒谱。

卷积神经网络(CNN)是一类包含卷积计算且具有深度结构的前馈神经网络，其隐含层内的卷积核参数共享和层间连接的稀疏性使得卷积神经网络能够以较

小的计算量对格点化特征，适合用于像素和音频的学习，因此广泛应用于计算机视觉和音频处理等领域。本实验中的 CNN 使用的网络结构由 Dense 层，激活函数和 dropout 层组成。dropout 层在训练过程中会以一定概率将神经网络中的单元丢弃掉，从而缓解过拟合的情况发生，提高了模型的稳定性和鲁棒性。

本次实验主要使用到了三种评价指标，precision、recall 和 f1-score。

在以下内容中，TP 表示预测为正的正样本，TN 表示预测为负的负样本、FP 表示预测为正的负样本、FN 表示预测为负的正样本。

(1) precision

$$Precision = \frac{TP}{TP + FP}$$

Precision，即精确率，表示预测结果为正的样本中正样本的比例。当产生 FP 的代价很高，即需要尽可能避免将负样本预测为正样本的情况时，应当注重提高 precision，而不是下文提到的 recall。

(2) recall

$$Recall = \frac{TP}{TP + FN}$$

Recall，即召回率，表示所有正样本中被成功预测为正样本的比例。当产生 FN 的成本很高，即需要尽可能避免将正样本预测为负样本的情况时，应当注重提高 recall，而不是上文提到的 precision。

(3) f1-score

$$F1\text{-}score = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

F1-score 是 Precision 和 Recall 的综合考量，同时兼顾了精确率和召回率两个衡量指标，与上述两个衡量指标相同，f1-score 应当尽量处于较高值。

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

本实验的代码分为两个函数，一个函数用于提取音频的声谱图特征，另一个函数用于提取音频的 MFCC 特征。

先看提取音频的声谱图特征函数 power_spectrogram，首先读取音频文件。然后将读取到的数据使用 librosa 库的 stft 函数做短时傅里叶变换，该函数返回一个复数矩阵。再使用 numpy 库的 abs 函数提取出复数的实部，即提取频率的振幅。最后使用 librosa 库的 amplitude_to_db 函数将幅度频谱转换为 dB 标度频谱，也就是对输入的数据取对数，从而完成声谱图特征的提取。

```
#读取音频文件
y, sr = librosa.load('./export.wav')
#准备绘图
plt.figure()

#做短时傅里叶变换，并提取频率的振幅，将其转换为dB标度频谱
D = librosa.amplitude_to_db(np.abs(librosa.stft(y)), ref=np.max)
```

然后使用 librosa.display 库的 specshow 函数将提取到的特征分别绘制线性频谱图和对数频谱图。需要注意的是 librosa.display 模块默认并不包含在 librosa 库中，因此既需要 import librosa，同时也需要 import librosa.display，才能够调用 specshow 函数。

```
#绘制线性频谱图
librosa.display.specshow(D, y_axis='linear', x_axis='time')
plt.colorbar(format='%+2.0f dB')
plt.title('Linear-frequency power spectrogram')
plt.show()

#绘制对数频谱图
librosa.display.specshow(D, y_axis='log', x_axis='time')
plt.colorbar(format='%+2.0f dB')
plt.title('Log-frequency power spectrogram')
plt.show()
```

再看提取音频的 MFCC 特征函数 mel_spectrogram，首先读取音频文件，然后使用 librosa 库的 melspectrogram 函数计算 Mel 频谱，并将 Mel 频谱绘制出来。

```
#读取音频文件
y, sr = librosa.load('./export.wav')
#准备绘图
plt.figure(figsize=(10, 4))

#计算得到Mel频谱
S = librosa.feature.melspectrogram(y=y, sr=sr)
#绘制Mel频谱
librosa.display.specshow(librosa.power_to_db(S, ref=np.max), y_axis='mel', x_axis='time')
plt.colorbar(format='%+2.0f dB')
plt.title('Mel spectrogram')
plt.tight_layout()
plt.show()
```

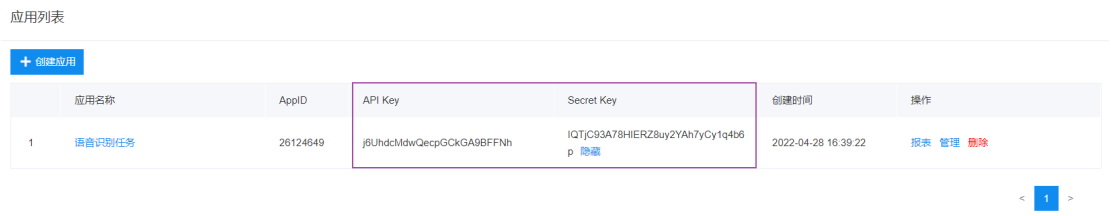
接下来使用 librosa 库的 mfcc 函数提取 MFCC 系数，得到该音频的 MFCC 特征，再绘制 Mel 倒谱。除此之外，可以对 MFCC 特征进行特征缩放操作，再将缩放后的 Mel 倒谱绘制出来。

```
#提取MFCC系数
mfccs = librosa.feature.mfcc(y, sr)
#绘制Mel倒谱
librosa.display.specshow(mfccs, sr=sr, x_axis='time')
plt.show()

#MFCC特征缩放
mfccs = sklearn.preprocessing.scale(mfccs, axis=1)
#绘制缩放后的Mel倒谱
librosa.display.specshow(mfccs, sr=sr, x_axis='time')
plt.show()
```

2. 调用公开语音识别 API，完成语音识别任务

首先在百度智能云平台上创建带有音频转写功能的应用，然后记录下应用的 API Key 和 Secret Key。



然后调用百度语音 API，使用百度智能云给出的 demo 代码，需要修改的地方有以下几处。首先写入 API Key 和 Secret Key。

```
#填写百度控制台中相关开通了“音频文件转写”接口的应用的的API_KEY及SECRET_KEY
API_KEY = 'j6UhdcMdwQecpGCkGA9BFFNh'
SECRET_KEY = 'IQTjC93A78HIERZ8uy2YAh7yCy1q4b6p'
```

修改待进行语音识别的音频文件 url 地址，此处我使用自己的图床存放了 mp3 格式的音频文件。

```
#待进行语音识别的音频文件url地址，需要可公开访问，建议使用百度云对象存储（https://cloud.baidu.com/product/bos.html）
speech_url_list = [
    "https://gitee.com/tsuyaki/image/raw/master/image/export_ofoct.com.mp3",
    ]
```

然后将得到的 task_id 放入转写任务 id 列表中，task_id 是通过创建音频转写任务时获取到的，每个音频任务的值。除了手动写入之外，我也设置了自动将百度语音 API 返回的转写任务 id 放入请求之中，不过音频转写需要一定的时间，因此音频任务创建成功后立刻查询结果可能得到的是 Running 状态，即音频正在转写中。

```
def query_result(task_id_list=None):
    """  发送查询结果请求 """

    #转写任务id列表，task_id是通过创建音频转写任务时获取到的，每个音频任务对应的值
    if not task_id_list:
        task_id_list = [
            "626b8cfc72dec6800729ea0e",
            ]
```

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

首先定义对数据进行预处理的函数，将 UrbanSound8K 数据集中的音频文件转换为 MFCC 特征，之后训练时可以直接使用提取到的 MFCC 特征数据进行训练，可以减小存储数据集的空间，并且减少数据集处理时间。将每个音频文件对应的 MFCC 特征和 label 标签到 dataset.npy 文件中，作为预处理后的数据集。

```python
# 数据预处理，将音频文件数据转换为mfcc特征数据保存
def data_preprocessing():
    #读取数据列表
    data = pd.read_csv('D:/Downloads/UrbanSound8K/metadata/UrbanSound8K.csv')

    # 读取wav文件函数
    def path_class(data, filename):
        excerpt = data[data['slice_file_name'] == filename]
        path_name = os.path.join('D:/Downloads/UrbanSound8K/audio', 'fold'+str(excerpt.fold.values[0]), filename)
        return path_name, excerpt['class'].values[0]

    # 读取wav声音文件，并提取mfcc特征，以及label标签，将其保存
    dataset = []
    for i in range(data.shape[0]):

        fullpath, class_id = path_class(data, data.slice_file_name[i])
        try:
            X, sample_rate = librosa.load(fullpath, res_type='kaiser_fast')
            mfccs = np.mean(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=40).T,axis=0)
        except Exception:
            print("Error encountered while parsing file: ", data.slice_file_name[i])
            mfccs,class_id = None, None

        dataset.append((mfccs, class_id))

    # 保存预处理后的数据集
    np.save("dataset",dataset,allow_pickle=True)
```

然后定义计算评价指标的函数，用于模型训练后计算和输出相关的评价指标数据。

```python
# 定义评价指标
def acc(y_test,prediction):
    ### PRINTING ACCURACY OF PREDICTION
    ### RECALL
    ### PRECISION
    ### CLASIFICATION REPORT
    ### CONFUSION MATRIX
    cm = confusion_matrix(y_test, prediction)
    recall = np.diag(cm) / np.sum(cm, axis = 1)
    precision = np.diag(cm) / np.sum(cm, axis = 0)

    print ('Recall:', recall)
    print ('Precision:', precision)
    print ('\n clasification report:\n', classification_report(y_test,prediction))
    print ('\n confussion matrix:\n',confusion_matrix(y_test, prediction))

    ax = sns.heatmap(confusion_matrix(y_test, prediction),linewidths= 0.5,cmap="YlGnBu")
```

以上两个都是函数，接下来进入主程序代码。首先判断 dataset.npy 文件是否存在，如果不存在则调用 data_preprocessing 函数创建再导入数据，如果存在则直接导入数据。

```
# 判断预处理后的数据集是否存在
if not os.path.exists('dataset.npy'):
    data_preprocessing()

# 导入数据
data = pd.DataFrame(np.load("dataset.npy",allow_pickle= True))
data.columns = ['feature', 'label']
```

然后将数据集分为训练数据和标签集，并使用 sklearn.model_selection 模块中的 train_test_split 函数随机划分训练集和验证集，再对标签进行独热码处理，从而准备好训练数据、训练标签、验证数据、验证标签四个数据集。

```
X = np.array(data.feature.tolist())
y = np.array(data.label.tolist())

# 数据分割
from sklearn.model_selection import train_test_split
X,val_x,y,val_y = train_test_split(X,y)

# 对标签进行one-hot处理
lb = LabelEncoder()
from keras.utils import np_utils
y = np_utils.to_categorical(lb.fit_transform(y))
val_y = np_utils.to_categorical(lb.fit_transform(val_y))

# 准备标签集
num_labels = y.shape[1]
nets = 5
```

然后定义模型结构，此处定义了五个 CNN 模型，然后训练模型，并打印出每个 CNN 模型的 epoch、训练集准确率和验证集准确率这三个数据。

```
model = [0] * nets

# 定义模型结构
for net in range(nets):
    model[net] = Sequential()


    model[net].add(Dense(512, input_shape=(40,)))
    model[net].add(Activation('relu'))
    model[net].add(Dropout(0.45))


    model[net].add(Dense(256))
    model[net].add(Activation('relu'))
    model[net].add(Dropout(0.45))


    model[net].add(Dense(num_labels))
    model[net].add(Activation('softmax'))



    model[net].compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='RMSprop')

# 训练网络
history = [0] * nets
epochs = 132
for j in range(nets):
    X_train2, X_val2, Y_train2, Y_val2 = X,val_x, y, val_y
    history[j] = model[j].fit(X,Y_train2, batch_size=256,
        epochs = epochs,
        validation_data = (X_val2,Y_val2),  verbose=0)
    print("CNN {0:d}: Epochs={1:d}, Train accuracy={2:.5f}, Validation accuracy={3:.5f}".format(
        j+1,epochs,max(history[j].history['accuracy']),max(history[j].history['val_accuracy']) ))
```

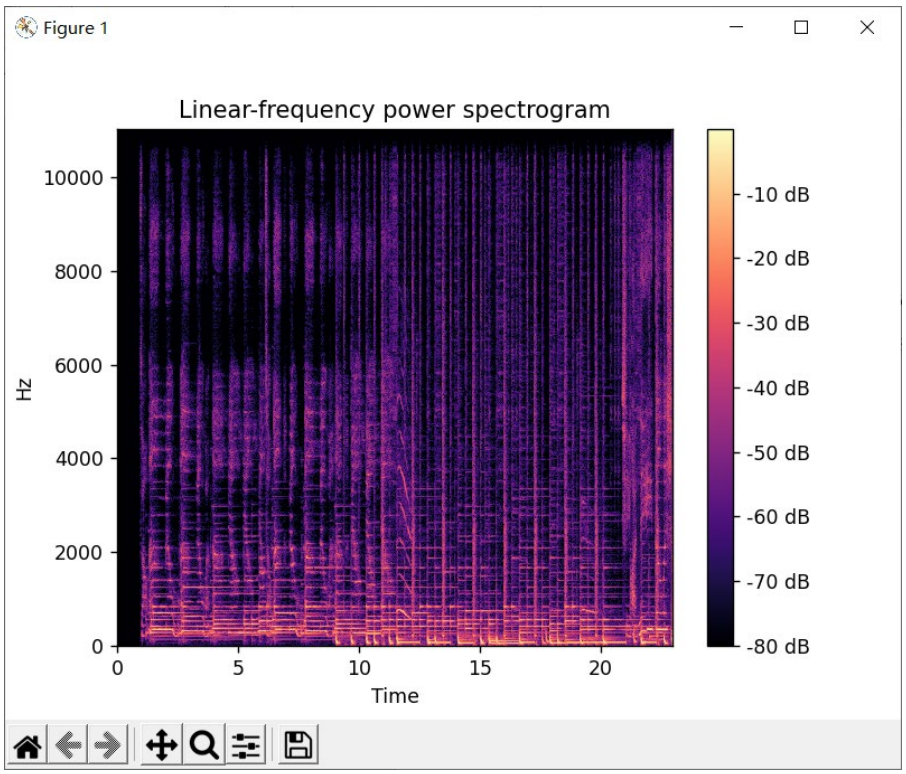
最后计算并查看评价指标和混淆矩阵。

```
# 查看评价指标以及混淆矩阵
results = np.zeros( (val_x.shape[0],10) )
for j in range(nets):
    results = results  + model[j].predict(val_x)
results = np.argmax(results,axis = 1)
val_y_n = np.argmax(val_y,axis =1)
acc(val_y_n,results)
```

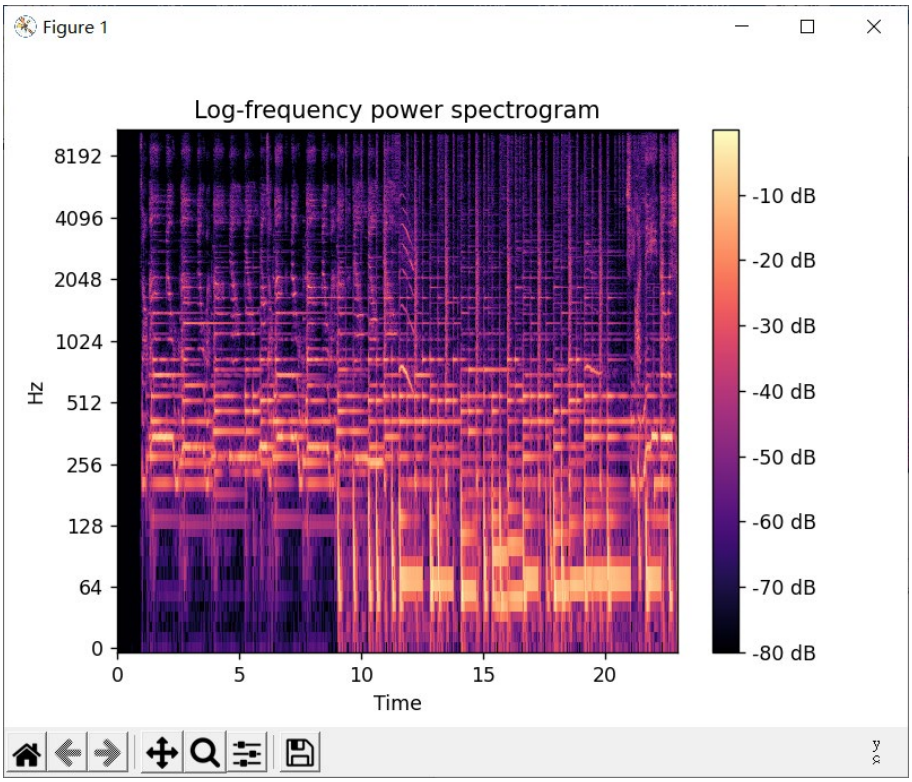
四、实验结果

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

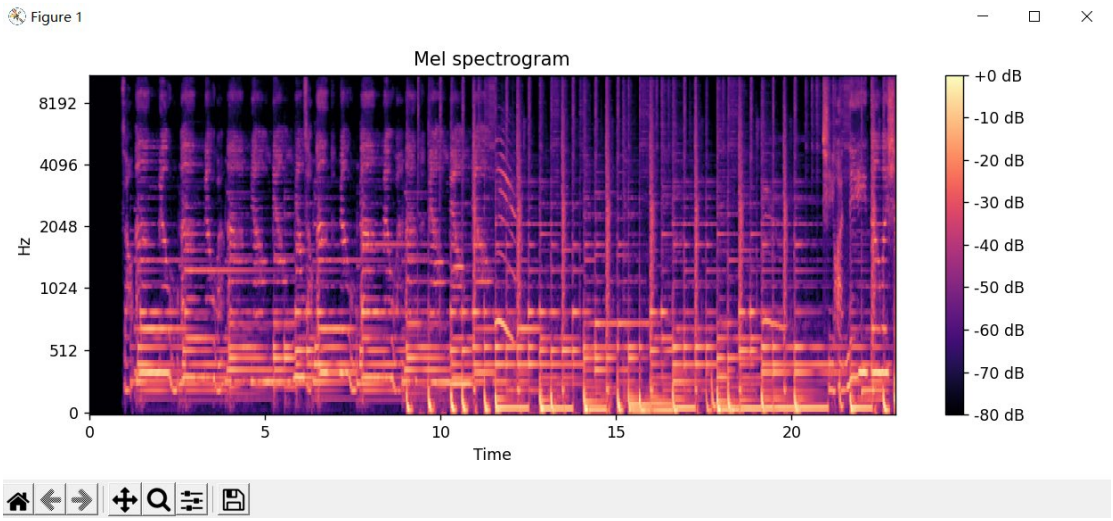
将提取到的声谱图特征绘制为线性频谱图。



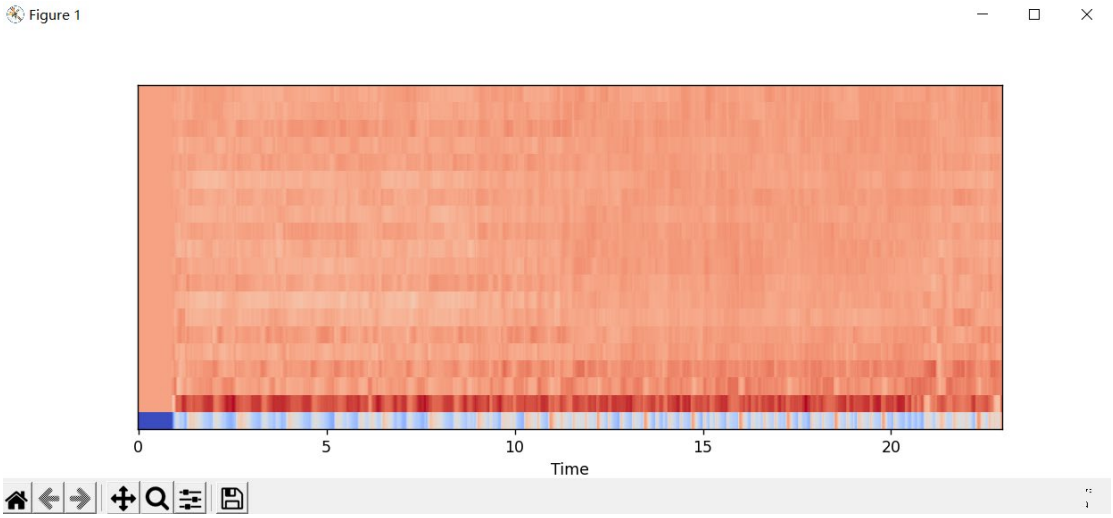
将提取到的声谱图特征绘制为对数频谱图。



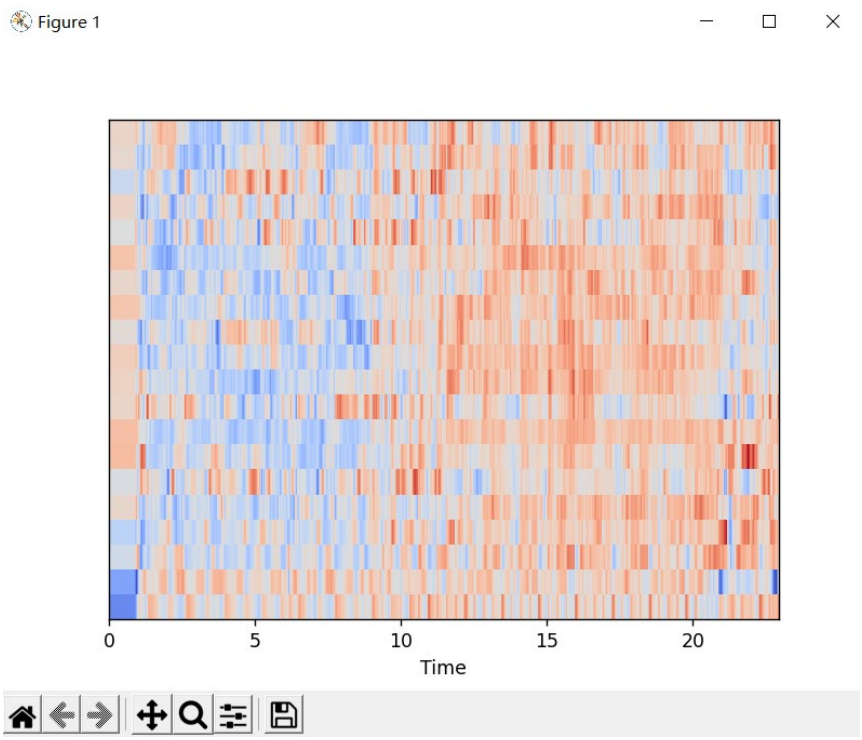
得到的 Mel 频谱图。



得到的 Mel 倒谱图以及 MFCC 特征。



得到的经过缩放后的 Mel 倒谱图。



2. 调用公开语音识别 API，完成语音识别任务

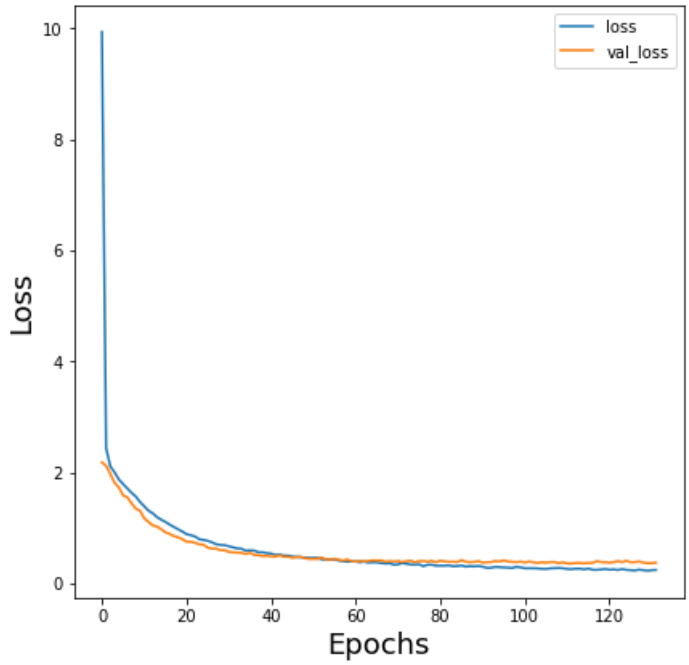
创建音频转写任务，获取 task_id 结果，再根据 task_id 查询音频转写任务结果。

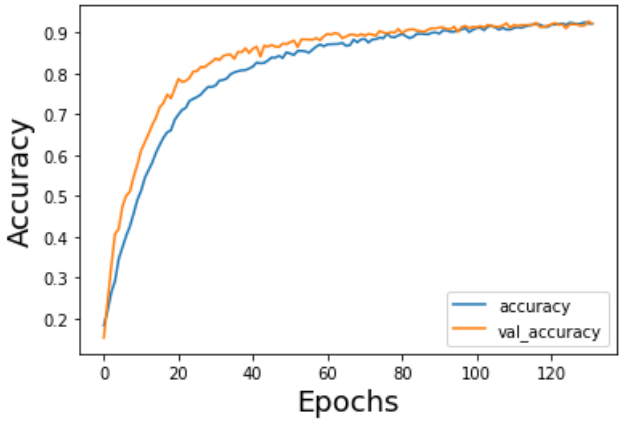
3. 使用深度学习方法，基于 UrbanSound8K 数据集完成音频分类任务

训练过程中每个 CNN 模型的 epoch、训练集准确率、验证集准确率。

```
CNN 1: Epochs=132, Train accuracy=0.96060, Validation accuracy=0.94182
CNN 2: Epochs=132, Train accuracy=0.92564, Validation accuracy=0.92808
CNN 3: Epochs=132, Train accuracy=0.92487, Validation accuracy=0.92396
CNN 4: Epochs=132, Train accuracy=0.92457, Validation accuracy=0.92945
CNN 5: Epochs=132, Train accuracy=0.92426, Validation accuracy=0.92716
```

训练过程中的 epochs-loss 折线图和 epochs-accuracy 折线图。





训练过程中的召回率和准确率等评价指标的值。

```
Recall: [0.98785425 0.9245283  0.93801653 0.87826087 0.91153846 0.96577947
 0.86813187 0.98168498 0.95132743 0.87755102]
Precision: [0.96825397 0.97029703 0.85338346 0.86695279 0.96341463 0.96946565
 0.95180723 0.95035461 0.94713656 0.93073593]
```

clasification report:

	precision	recall	f1-score	support
0	0.97	0.99	0.98	247
1	0.97	0.92	0.95	106
2	0.85	0.94	0.89	242
3	0.87	0.88	0.87	230
4	0.96	0.91	0.94	260
5	0.97	0.97	0.97	263
6	0.95	0.87	0.91	91
7	0.95	0.98	0.97	273
8	0.95	0.95	0.95	226
9	0.93	0.88	0.90	245
accuracy			0.93	2183
macro avg	0.94	0.93	0.93	2183
weighted avg	0.94	0.93	0.93	2183

混淆矩阵。

```
confussion matrix:
[[244   0   0   1   0   0   0   1   0   1]
 [  0  98   0   4   0   0   0   2   0   2]
 [  1   0 227   5   0   3   0   1   2   3]
 [  3   0   7 202   3   1   2   0   8   4]
 [  0   0   3   6 237   1   1   8   0   4]
 [  0   0   7   1   0 254   0   0   0   1]
 [  0   0   3   7   1   0  79   0   1   0]
 [  0   0   0   0   4   0   0 268   0   1]
 [  0   1   6   3   0   1   0   0 215   0]
 [  4   2  13   4   1   2   1   2   1 215]]
```