武汉大学国家网络安全学院实验报告

课程名称	内容安全实验	实验日期	2024-3-25
实验名称	实验 2：音频处理与分析		
姓名	学号	专业	班级
邓鹏	2021302181152	网络空间安全	6 班

目录

一、实验目的及实验内容

（本次实验的具体内容；必要的原理分析）

1.1 实验目的

1. 提取任意一段不小于 30s 的 wav 音频的声谱图特征，并详细解释实验原理
2. 复现任一深度学习模型，在 UrbanSound8K 数据集上完成音频分类任务

1.2 实验内容

1.2.1 声谱图特征相关原理

• 声谱图(Mel-frequency spectrogram)

声谱图是一种表示音频信号频率内容随时间变化的图形，它通过将音频信号分解为一系列时间段内的频率成分来构建。声谱图中的每一点表示特定时间和特定频率下的信号强度（或能量）。在音频分析、语音识别、音乐信息检索等领域中，声谱图是一个非常重要的工具，因为它提供了音频信号的丰富信息，包括音高、音色和节奏等。

声谱图的生成基于以下步骤：

1. 分帧：首先，将连续的音频信号分割成重叠的短时间帧。这是因为音频信号是非平稳的，其频率成分随时间变化。通过将音频分成短时间的帧，我们可以假设每一帧内的音频信号是平稳的，从而可以分析其频率成分。

2. 窗函数：为了减少帧两端的不连续性对频率分析的影响，每个帧通常会通过一个窗函数进行加权。常用的窗函数包括汉宁窗、汉明窗等。窗函数可以减少帧边缘的信号能量，从而减少傅里叶变换时产生的频谱泄露。

3. 快速傅里叶变换（FFT）：对每个窗口化的帧应用快速傅里叶变换（FFT），将其从时间域转换到频率域。FFT 揭示了该时间段内音频信号的频率成分及其强度。

4. 构建声谱图：将所有帧的 FFT 结果按照时间顺序排列，形成二维矩阵。这个矩阵的横轴表示时间，纵轴表示频率，而每个点的亮度或颜色表示特定时间和频率下的能量或幅度。

5. 颜色映射：为了更好地可视化声谱图，通常会将能量或幅度映射到颜色或灰度值上。亮度或颜色的强度表示该频率成分的强度或能量大小。

简而言之，声谱图通过短时傅里叶变换（STFT）计算得到，显示了音频信号随时间变化的频率成分。Mel 频率声谱图是一种基于 Mel 刻度的声谱图，它更贴近人类的听觉感知，使得频率分辨率在低频时更高。

• MFCCs (Mel-frequency cepstral coefficients)

MFCCs 是声音信号处理中常用的一种特征表示方法，特别是在语音识别和音乐信息检索等领域。它们的主要目的是捕获音频信号的短时功率谱的形状，方式是通过模拟人耳的听觉特性。

MFCCs 的计算过程可以分为以下几个步骤：

1. 预加重和帧分割：首先，对原始的音频信号进行预加重处理，以增强高频部分。之后，将信号分割成小的固定长度的帧，通常帧与帧之间会有一定的重叠，以减少边界效应。

2. 汉明窗和快速傅立叶变换（FFT）：对每一帧的信号应用汉明窗（或其他窗函数）以减少帧边缘的不连续性。然后，对窗函数处理后的帧信号执行快速傅立叶变换（FFT），以获取其频谱。

3. 梅尔滤波器组处理：使用一组梅尔尺度的滤波器处理 FFT 的结果。梅尔尺度是一种基于人耳听觉特性的频率尺度，它使得特征更加符合人类的听觉感知。这些滤波器通常是三角形的，并覆盖整个可听频率范围。每个滤波器的输出是该滤波器覆盖范围内频谱能量的总和。

4. 对数能量：取每个滤波器输出的对数能量。这是因为人耳对声音的感知是对数的，这样可以使特征更加接近人类的听觉感知。

5. 离散余弦变换（DCT）：最后，对每一帧的滤波器组输出的对数能量进行离散余弦变换（DCT），以得到梅尔频率倒谱系数。DCT 的作用是将滤波器组的对数能量信号从频域转换到倒谱域。这有助于降低特征之间的相关性，并且通常只选择 DCT 系数的一个子集作为最终的特征，因为高阶系数代表的是快速变化的部分，通常不包含对区分不同声音有用的信息。

简而言之，MFCCs 是通过对信号的 Mel 频率声谱图进行离散余弦变换（DCT）得到的系数，反映了音频信号的频谱包络。

• 频谱质心 (Spectral Centroid)

频谱质心是描述声音色彩的一个特征，它代表了一个声音频谱的重心或“平均”频率。在音乐和音频处理领域，频谱质心用于识别乐器声音的明亮度，辨识音乐风格，以及音乐信息检索等多种应用中。一个声音的频谱质心越高，声音听起来越“明亮”；反之，则听起来更“暗”或更“浑”。

频谱质心的计算基于以下步骤：

1. 傅里叶变换：首先对音频信号进行短时傅里叶变换（STFT），将音频信号从时域转换到频域，得到频域中每个频率分量的幅度和相位信息。

2. 计算加权平均频率：计算所有频率分量的加权平均值，加权系数为该频率分量的幅度。这个过程就是找到频率分量幅度作为权重时的平均频率位置。

3. 归一化：有时候，为了方便比较不同信号之间的频谱质心，可能会对频谱质心的值进行归一化处理。

简而言之，计算信号频谱的“重心”，反映了音频信号的“亮度”或音色的感知属性。

• 频谱平坦度 (Spectral Flatness)

频谱平坦度是衡量一个信号频谱分布均匀程度的度量。它用于描述一个信号的频谱是接近噪声还是包含很多尖锐的峰值。在音频处理和音乐信息检索中，频谱平坦度常被用来区分音乐和噪声，识别不同类型的声音和音乐风格，以及音频编码优化等应用。

频谱质心的计算基于以下步骤：

1.频谱提取：首先，需要对音频信号执行快速傅里叶变换（FFT）以获得其频谱表示。这一步展示了信号在不同频率上的能量分布。

2.计算公式：频谱平坦度通过比较频谱中所有频率成分的幅度的几何平均数和算术平均数来计算。其数学表达式为：

$$S = \frac{10}{N} \log_{10} \left(\frac{\prod_{i=1}^{N} P(f_i)}{\frac{1}{N} \sum_{i=1}^{N} P(f_i)} \right)$$

其中，S 是频谱平坦度，N 是频率成分的总数，$P(f_i)$ 是第 i 个频率成分的功率谱密度。

简而言之，衡量声谱的平坦程度，值越大表示声谱越平坦，趋近于白噪声。

• 过零率 (Zero Crossing Rate)

过零率是信号处理中的一个概念，尤其在音频分析和语音处理领域中得到了广泛应用。它指的是信号在单位时间内通过零点的次数，换句话说，就是信号波形正负极性改变的次数。过零率是一个表征信号频率特性的简单而有效的参数，对于分析音频特征、识别语音和乐器声音、音乐风格识别等有着重要的应用价值。

计算过零率的基本步骤如下：

1. 信号分帧：首先，将连续的音频信号分割成短时间帧。每帧包含了固定数量的样本点，通常是 25ms 到 30ms 长度。这是因为音频信号的特性在这么短的时间内可以认为是稳定的。

2. 确定过零点：对于每帧信号，检查相邻样本点的符号（正或负）。如果相邻样本点之间的符号不同，则认为在这两个样本点之间发生了一次过零。

3. 计算过零率：对于每一帧，计算总的过零次数。然后，可以将这个值标准化，例如，通过除以帧的长度（秒），得到每秒的过零次数，从而提供一个时间独立的度量。

简而言之，计算单位时间内信号正负变化的次数，反映了音频信号的频率特性。

• 声谱能量 (Spectral Bandwidth)

声谱能量是音频信号处理中用来描述信号频谱分布宽度的一个特征。在音乐和语音分析领域，这个特征被用来量化频谱中能量的分散程度。简单来说，声谱

能量衡量了主要能量集中的频带宽度，能够反映信号的频谱特性和声音的色彩。

计算声谱能量的基本步骤如下：

1. 傅里叶变换：首先，使用快速傅里叶变换（FFT）将音频信号从时域转换到频域。这一步骤得到的频谱包含了信号中各个频率成分的幅度信息。

2. 计算频谱质心：频谱质心是信号能量分布的“中心”频率。计算方法是通过加权平均所有频率成分，其中每个成分的权重是其幅度。频谱质心为计算声谱能量提供了一个参考点。

3. 计算声谱能量：声谱能量是量化频谱中能量分布宽度的度量，通常定义为频谱中各点到频谱质心的距离的加权平均值。其计算公式可以表示为：

$$SBW = \sqrt{\frac{\sum (f_i - f_c)^2 \cdot a_i}{\sum a_i}}$$

其中，SBW 是声谱能量，fi 是第 i 个频率成分，fc 是频谱质心，ai 是 fi 的幅度。

简而言之，计算声谱质心周围能量的分布宽度，反映了声音的“宽度”或“丰满度”。

• 声谱对比度 (Spectral Contrast)

声谱对比度是音频信号处理中用于描述信号频谱中不同频带之间能量对比度的特征。它基于这样一个观察：音频信号（包括音乐和语音）的频谱通常在某些频带内展示出较高的能量，而在其他频带内则能量较低。声谱对比度通过量化这些频带之间的能量差异，提供了一种捕捉信号内在结构的方法，有助于音乐和语音的分析、识别和分类。

声谱对比度的计算涉及以下几个步骤：

1. 频谱分析：首先，对音频信号进行快速傅里叶变换（FFT），得到其频谱表示。这一步揭示了信号在不同频率上的能量分布。

2. 频带划分：将整个频谱划分成若干个频带。这些频带可以是等宽的，也可以根据听觉感知的特性（如梅尔刻度或巴克刻度）进行非等宽划分。

3. 计算每个频带的能量：在每个频带内，计算频率成分的能量。这可以通过直接计算该频带内所有频点的幅度平方和来完成。

4. 提取顶部和底部的能量：在每个频带内，识别出能量最高的顶部区域和能量最低的底部区域。顶部区域代表了该频带内主要的能量集中，而底部区域则反映了背景噪声或能量较低的成分。

5. 计算对比度：对每个频带，通过比较其顶部和底部区域的能量平均值，计算出声谱对比度。具体来说，可以通过取顶部能量平均值与底部能量平均值的比值或差值来量化。

简而言之，计算声谱峰值与谷值之间的对比度，描述了声谱中不同频段的动态范围。

• 声谱滚降点 (Spectral Rolloff)

声谱滚降点是描述音频信号频谱形状的一个特征，特别是用来量化频谱高端的边界。它是一个阈值，表示在该频率点之下的频率成分累积了信号总能量的某个百分比（通常是 85%至 95%）。声谱滚降点用于音乐信息检索、音乐风格分类、乐器识别、以及语音处理等领域，因为它能够反映出音频信号频谱的形状和能量分布特性。

声谱滚降点的计算涉及以下几个步骤：

1. 频谱提取：对音频信号进行快速傅里叶变换（FFT）以获取其频谱表示，揭示了不同频率成分的幅度信息。

2. 计算累积能量：计算频谱中每个频率点的能量，并计算从最低频率到每个点的累积能量。这一步的目的是确定频谱中能量的分布情况。

3. 确定滚降点：设定一个阈值，比如总能量的 85%。然后，从最低频率开始累加能量，直到达到这个阈值。对应的频率即为声谱滚降点。这意味着在该频率以下的区域累积了总能量的指定百分比，而剩余的能量分布在该频率以上。

简而言之，计算声谱中一定百分比（如 85%）的能量累积在该点以下的频率，反映了声谱形状。

1.2.2 CNN 完成音频分类任务

1. 模型结构

（1）输入层：模型的输入维度被设置为(16, 8, 1)，这意味着每个输入样本是一个 16x8 的矩阵，且为单通道（例如，灰度图像或单通道的频谱图）。

（2）卷积层：模型包含两个卷积层，每个卷积层后面跟着一个最大池化层。卷积层使用 Conv2D，第一个卷积层有 64 个过滤器，第二个有 128 个过滤器，都使用 tanh 激活函数。卷积层的作用是提取输入数据的特征，过滤器数量决定了能够提取的特征数量。

（3）池化层：MaxPool2D 池化层用于降低特征图的空间维度，通过提取每个小区域内的最大值来实现。这有助于减少计算量并提取更加显著的特征。

（4）Dropout 层：在两个卷积层之后使用了一个 Dropout 层，比率为 0.1。Dropout 层用于减少过拟合，通过在训练过程中随机丢弃一部分神经元的输出来增加模型的泛化能力。

（5）Flatten 层：将多维的输出展平为一维，以便在全连接层（Dense 层）中使用。

（6）全连接层：首先是一个含有 1024 个神经元的 Dense 层，使用 tanh 激活函数，其次是输出层，输出层包含 10 个神经元（对应 10 个类别），使用 softmax 激活函数进行多分类。

2. 训练和评估

（1）编译模型：使用 adam 优化器和 categorical_crossentropy 损失函数进行编译，这适合多分类问题。

（2）训练模型：模型使用提供的训练数据进行训练，训练周期（epochs）设置为 90，批量大小（batch size）为 50，同时使用了验证集（X_test, Y_test）来评

估模型的性能。
（3）性能评估：通过在测试集上运行模型，并打印出准确率和损失值来评估模型的性能。

二、实验环境

（本次实验所使用的开发环境、依赖包等的情况）

1. Windows10+Pycharm+numpy+librosa+matplotlib
2. Windows10+Anaconda+Tensorflow+pandas+numpy+matplotlib+seaborn+sklearn+librosa+skimage
3. 数据集：UrbanSound8K，下载链接 UrbanSound8K (kaggle.com)

三、实验步骤及结果分析

（详细描述实验步骤，并根据具体实验，记录、整理相应的数据表格等，对实验结果进行分析）

3.1 wav 音频的声谱图特征

1. 加载音频文件

```
# 加载音频文件
audio_path = 'D:/Content_Secu/task2/xm2512.wav'
y, sr = librosa.load(audio_path)
```

2. 计算声谱图

```
# 计算声谱图
S = librosa.feature.melspectrogram(y=y, sr=sr, n_mels=128, fmax=8000)
S_dB = librosa.power_to_db(S, ref=np.max)
```

3. 计算 MFCCs

```
# 计算MFCCs
mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=13)
```

4. 计算频谱质心

```
# 计算频谱质心
spectral_centroids = librosa.feature.spectral_centroid(y=y, sr=sr)
```

5. 计算频谱平坦度

```
# 计算频谱平坦度
spectral_flatness = librosa.feature.spectral_flatness(y=y)
```

6. 计算过零率

```
# 计算过零率
zero_crossing_rate = librosa.feature.zero_crossing_rate(y)
```

7. 计算声谱能量

```
# 计算声谱能量
spectral_bandwidth = librosa.feature.spectral_bandwidth(y=y, sr=sr)
```

8. 计算声谱对比度

```
# 计算声谱对比度
spectral_contrast = librosa.feature.spectral_contrast(y=y, sr=sr)
```

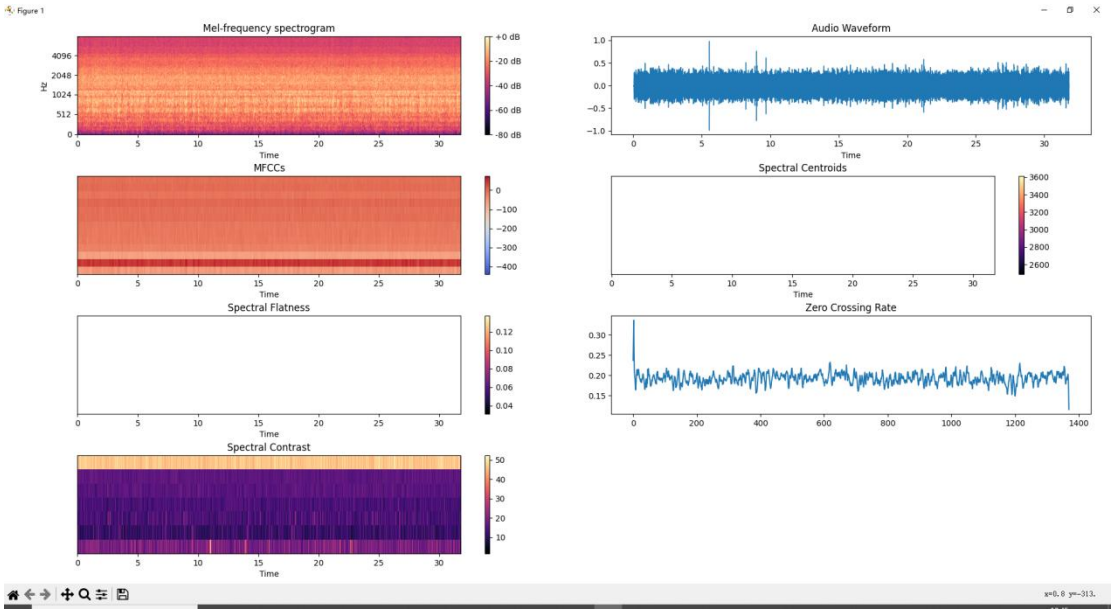
9. 计算声谱降滚点

```
# 计算声谱滚降点
spectral_rolloff = librosa.feature.spectral_rolloff(y=y, sr=sr)
```

10. 绘图

```
# 绘图展示特征
plt.figure(figsize=(12, 11))

# 绘制声谱图
plt.subplot( *args: 4, 2, 1)
librosa.display.specshow(S_dB, sr=sr, x_axis='time', y_axis='mel', fmax=8000)
plt.colorbar(format='%+2.0f dB')
plt.title('Mel-frequency spectrogram')

# 绘制波形
plt.subplot( *args: 4, 2, 2)
librosa.display.waveshow(y, sr=sr)
plt.title('Audio Waveform')

# 绘制MFCCs
plt.subplot( *args: 4, 2, 3)
librosa.display.specshow(mfccs, sr=sr, x_axis='time')
plt.colorbar()
plt.title('MFCCs')

# 绘制频谱质心
plt.subplot( *args: 4, 2, 4)
librosa.display.specshow(spectral_centroids, sr=sr, x_axis='time')
plt.colorbar()
plt.title('Spectral Centroids')

# 绘制频谱平坦度
plt.subplot( *args: 4, 2, 5)
librosa.display.specshow(spectral_flatness, x_axis='time')
plt.colorbar()
plt.title('Spectral Flatness')

# 绘制过零率
plt.subplot( *args: 4, 2, 6)
plt.plot(zero_crossing_rate[0])
plt.title('Zero Crossing Rate')

# 绘制声谱对比度
plt.subplot( *args: 4, 2, 7)
librosa.display.specshow(spectral_contrast, sr=sr, x_axis='time')
plt.colorbar()
plt.title('Spectral Contrast')

plt.tight_layout()
plt.savefig('1.png')
plt.show()
```

11. 效果展示



12. 图谱分析

（1）梅尔频率谱图（Mel-frequency spectrogram）：位于左上角，显示了频率（垂直轴）随时间（水平轴）变化的能量分布。这个图采用了梅尔刻度，是一种基于人类听觉感知的频率尺度。谱图中亮色代表高能量区域，暗色则表示低能量区域，通常亮色的部分对应于声音中的主要频率成分。

（2）音频波形（Audio Waveform）：位于右上角，直观地表示了声波的振幅随时间变化的情况。在波形图中，可以看到声波的振幅变化，峰值可能表示音频中的突出事件或声音。

（3）MFCCs：位于左下角的第二行第一个图，显示了 MFCC 随时间的变化。MFCC 是音频分析中常用的特征，用于捕获音频信号的短时功率谱的形状，并广泛应用于语音识别和音乐信息检索等领域。

（4）频谱质心（Spectral Centroids）：位于右下角第二行第一个图，表示信号频谱中心的变化。频谱质心提供了声音“亮度”或“暗度”的量度，高值通常表示亮度较高的声音。

（5）过零率（Zero Crossing Rate）：位于右下角第二行第二个图，显示了音频信号中波形过零点的频率。过零率高的地方表明音频信号变化较快，可能对应于噪音或高频震动。

（6）频谱平坦度（Spectral Flatness）：位于左下角第三行第一个图，虽然数据看起来不是很清晰，但频谱平坦度通常用来衡量信号的噪声程度，值接近于 1 的平坦谱表示信号类似于白噪声。

（7）声谱对比度（Spectral Contrast）：位于左下角第三行第二个图，展示了不同频带之间的能量对比度。声谱对比度可以揭示音频信号中不同频率带之间的能量差异，反映信号的动态范围和纹理。

3.2 CNN 在 UrbanSound8K 数据集上完成音频分类任务

1. 导入库并读取 csv 文件

In [1]:
```
import pandas as pd
import numpy as np

pd.plotting.register_matplotlib_converters()
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.model_selection import GridSearchCV

from sklearn.preprocessing import MinMaxScaler
```

In [2]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, Flatten, Dense, MaxPool2D, Dropout
from tensorflow.keras.utils import to_categorical

from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
```

In [3]:
```
import os
import librosa
import librosa.display
import glob
import skimage
```

In [4]:
```
# 读取数据
df = pd.read_csv("D:/Content_Secu/task2/input/UrbanSound8K.csv")
df.head()
```

Out[4]:

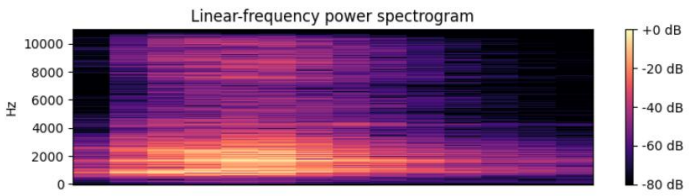
	slice_file_name	fsID	start	end	salience	fold	classID	class
0	100032-3-0-0.wav	100032	0.0	0.317551	1	5	3	dog_bark
1	100263-2-0-117.wav	100263	58.5	62.500000	1	5	2	children_playing
2	100263-2-0-121.wav	100263	60.5	64.500000	1	5	2	children_playing
3	100263-2-0-126.wav	100263	63.0	67.000000	1	5	2	children_playing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

2. 使用 librosa 库加载音频文件并展示其波形和频谱图

In [5]:
```
# 读取两个音频文件，用于后续处理
dat1, sampling_rate1 = librosa.load('D:/Content_Secu/task2/input/fold5/100032-3-0-0.wav')
dat2, sampling_rate2 = librosa.load('D:/Content_Secu/task2/input/fold5/100263-2-0-117.wav')
```

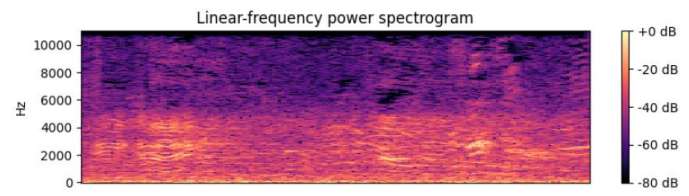
In [6]:
```
# 生成两个音频文件的线性频率功率谱图
plt.figure(figsize=(20, 10))
D = librosa.amplitude_to_db(np.abs(librosa.stft(dat1)), ref=np.max)
plt.subplot(4, 2, 1)
librosa.display.specshow(D, y_axis='linear')
plt.colorbar(format='%+2.0f dB')
plt.title('Linear-frequency power spectrogram')
```

Out[6]: Text(0.5, 1.0, 'Linear-frequency power spectrogram')

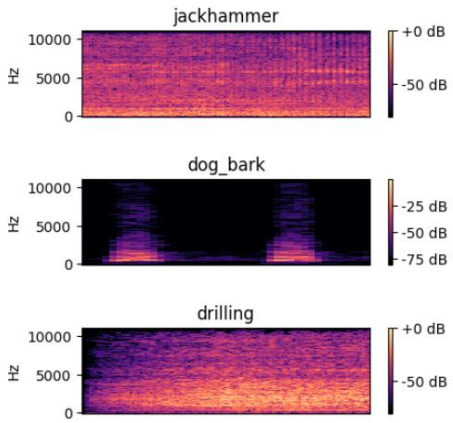


In [7]:
```
plt.figure(figsize=(20, 10))
D = librosa.amplitude_to_db(np.abs(librosa.stft(dat2)), ref=np.max)
plt.subplot(4, 2, 1)
librosa.display.specshow(D, y_axis='linear')
plt.colorbar(format='%+2.0f dB')
plt.title('Linear-frequency power spectrogram')
```

Out[7]: Text(0.5, 1.0, 'Linear-frequency power spectrogram')



In [8]:
```
# 使用随机样本来观察波形差异
arr = np.array(df["slice_file_name"])
fold = np.array(df["fold"])
cla = np.array(df["class"])

# 循环处理随机选择的音频文件，展示它们的谱图和类别
for i in range(192, 197, 2):
    path = 'D:/Content_Secu/task2/input/fold' + str(fold[i]) + '/' + arr[i]
    data, sampling_rate = librosa.load(path)
    plt.figure(figsize=(10, 5))
    D = librosa.amplitude_to_db(np.abs(librosa.stft(data)), ref=np.max)
    plt.subplot(4, 2, 1)
    librosa.display.specshow(D, y_axis='linear')
    plt.colorbar(format='%+2.0f dB')
    plt.title(cla[i])
```



3. 特征提取

In [9]:
```
# 读取一个音频文件，并获取它的Mel频谱
dat1, sampling_rate1 = librosa.load('D:/Content_Secu/task2/input/fold5/100032-3-0-0.wav')
arr = librosa.feature.melspectrogram(y=dat1, sr=sampling_rate1)
arr.shape
```

Out[9]: (128, 14)

In [10]:
```
# 初始化特征和标签列表
feature = []
label = []

# 定义解析函数，用于从DataFrame中提取音频特征和标签
def parser(df):
    features = []
    labels = []
    for i in range(len(df)):
        file_name = f'D:/Content_Secu/task2/input/fold{df["fold"].iloc[i]}/{df["slice_file_name"].iloc[i]}'
        X, sample_rate = librosa.load(file_name, res_type='kaiser_fast')
        mels = np.mean(librosa.feature.melspectrogram(y=X, sr=sample_rate).T, axis=0)
        features.append(mels)
        labels.append(df["classID"].iloc[i])

    features = np.array(features, dtype=np.float32)
    labels = np.array(labels, dtype=np.int32)
    return features, labels
```

In [11]:
```
# 调用解析函数处理数据集，并获取特征和标签
features, labels = parser(df)
```

```
C:\Users\35083\.conda\envs\tensorflow\lib\site-packages\librosa\core\spectrum.py:257: UserWarning: n_fft=2048 is too large for input si
gnal of length=1323
  warnings.warn(
C:\Users\35083\.conda\envs\tensorflow\lib\site-packages\librosa\core\spectrum.py:257: UserWarning: n_fft=2048 is too large for input si
gnal of length=1103
  warnings.warn(
C:\Users\35083\.conda\envs\tensorflow\lib\site-packages\librosa\core\spectrum.py:257: UserWarning: n_fft=2048 is too large for input si
gnal of length=1523
  warnings.warn(
```

In [12]:
```
# 将标签转换为one-hot编码，用于多分类
labels = to_categorical(labels)
```

In [13]:
```
# 打印最终的特征和标签数组的形状
print(features.shape)
print(labels.shape)
```

```
(8732, 128)
(8732, 10)
```

4. 数据预处理

In [14]:
```
# 将数据集划分为训练集和测试集，测试集比例为20%
X_train, X_test, Y_train, Y_test = train_test_split(features, labels, test_size=0.2, random_state=42)
```

In [15]:
```
# 计算训练集和测试集的样本数量
n_samples_train = X_train.shape[0]  # 训练集样本数
n_samples_test = X_test.shape[0]  # 测试集样本数

# 重塑X_train和X_test以适配模型的输入形状
# 保证总的元素个数不变，这里128等于16*8，所以将每个样本重塑为(16, 8, 1)形状
X_train = X_train.reshape(n_samples_train, 16, 8, 1)
X_test = X_test.reshape(n_samples_test, 16, 8, 1)
```

5. 模型构建

In [16]:
```
# 设置模型输入的维度
input_dim = (16, 8, 1)
```

In [17]:
```
# 构建Sequential模型
model = Sequential()
```

In [18]:
```
# 向模型中添加卷积层、池化层和dropout层
model.add(Conv2D(64, (3, 3), padding = "same", activation = "tanh", input_shape = input_dim))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3), padding = "same", activation = "tanh"))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.1))
# 展平特征图以传入全连接层
model.add(Flatten())
# 添加全连接层，最后是输出层，用softmax激活函数进行分类
model.add(Dense(1024, activation = "tanh"))
model.add(Dense(10, activation = "softmax"))
```

In [19]:
```
# 编译模型，设置优化器、损失函数和评估指标
model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

6. 模型训练

In [20]:
```
# 训练模型，设置训练周期、批量大小和验证数据
model.fit(X_train, Y_train, epochs = 90, batch_size = 50, validation_data = (X_test, Y_test))
```

```
Epoch 82/90
140/140 [==============================] - 6s 40ms/step - loss: 0.0634 - accuracy: 0.9778 - val_loss: 0.8860 - val_accuracy: 0.8592
Epoch 83/90
140/140 [==============================] - 6s 40ms/step - loss: 0.0895 - accuracy: 0.9709 - val_loss: 0.9648 - val_accuracy: 0.8523
Epoch 84/90
140/140 [==============================] - 6s 40ms/step - loss: 0.0852 - accuracy: 0.9686 - val_loss: 0.8901 - val_accuracy: 0.8638
Epoch 85/90
140/140 [==============================] - 6s 40ms/step - loss: 0.0632 - accuracy: 0.9781 - val_loss: 0.9122 - val_accuracy: 0.8655
Epoch 86/90
140/140 [==============================] - 6s 40ms/step - loss: 0.0504 - accuracy: 0.9830 - val_loss: 0.8454 - val_accuracy: 0.8695
Epoch 87/90
140/140 [==============================] - 6s 40ms/step - loss: 0.0596 - accuracy: 0.9802 - val_loss: 0.8798 - val_accuracy: 0.8626
Epoch 88/90
140/140 [==============================] - 6s 40ms/step - loss: 0.0433 - accuracy: 0.9851 - val_loss: 0.9145 - val_accuracy: 0.8758
Epoch 89/90
140/140 [==============================] - 5s 39ms/step - loss: 0.0471 - accuracy: 0.9824 - val_loss: 0.9030 - val_accuracy: 0.8666
Epoch 90/90
140/140 [==============================] - 6s 39ms/step - loss: 0.0466 - accuracy: 0.9837 - val_loss: 0.9106 - val_accuracy: 0.8718
```

Out[20]:
```
<keras.callbacks.History at 0x23936da94f0>
```

In [21]:
```
# 打印模型概览
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 16, 8, 64)         640

 max_pooling2d (MaxPooling2D  (None, 8, 4, 64)         0
 )

 conv2d_1 (Conv2D)           (None, 8, 4, 128)         73856

 max_pooling2d_1 (MaxPooling  (None, 4, 2, 128)        0
 2D)

 dropout (Dropout)           (None, 4, 2, 128)         0

 flatten (Flatten)           (None, 1024)              0

 dense (Dense)               (None, 1024)              1049600

 dense_1 (Dense)             (None, 10)                10250

=================================================================
Total params: 1,134,346
Trainable params: 1,134,346
Non-trainable params: 0
_________________________________________________________________
```

7. 模型评估

```
In [22]: # 使用测试集进行预测
predictions = model.predict(X_test)
# 评估模型性能，输出损失值和准确率
score = model.evaluate(X_test, Y_test)
print(score)

55/55 [==============================] - 0s 3ms/step
55/55 [==============================] - 0s 3ms/step - loss: 0.9106 - accuracy: 0.8718
[0.9105537533760071, 0.8717802166938782]
```

8. 结果保存

```
In [23]: # 将预测结果转换为标签
preds = np.argmax(predictions, axis = 1)

In [24]: # 将预测结果保存为pandas DataFrame，并写入CSV文件
result = pd.DataFrame(preds)
result.to_csv("UrbanSound8kResults.csv")
```

可见，效果不错。

四、个人小结

（描述实验心得，可提出实验的改进意见）

在实验中，首先对声谱图特征相关的原理进行了深入的探讨。声谱图通过短时傅里叶变换（STFT）反映了音频信号随时间变化的频率成分。通过分帧、窗函数、FFT 以及颜色映射等步骤生成了声谱图。接着，深入理解了 MFCCs）的计算过程，这是通过模拟人耳听觉特性，从频域到倒谱域的转换，用于捕捉音频信号的短时功率谱形状。实验还分析了频谱质心、平坦度、过零率、能量、对比度以及滚降点等多种声谱特征，这些特征从不同角度描述了音频信号的属性。

实验过程中，使用 librosa 库对 UrbanSound8K 数据集中的音频文件进行了加载和特征提取。计算了各种声谱图特征，并将其可视化。在对数据进行预处理后，构建了一个基于卷积神经网络的分类模型。模型包含了多层卷积、池化、全连接层，以及用于减少过拟合的 Dropout 层。通过设定适当的训练轮数和批量大小，使用训练数据对模型进行了训练，并在验证集上进行了性能评估。训练完成后，模型在测试集上的表现被详细分析，准确率和损失值作为评估指标。

五、教师评语及评分

教师签名：

年　　月　　日