浙江大学软件学院

软件服务业分营

智能语音计算方向夏令营报告

选题：	真伪语音鉴别
姓名：	于阔溪
本科学校：	北京邮电大学
目标方向：	音乐信息检索
分营导师：	杨莹春

目录

1. 实验报告

1.1 任务描述与背景调研

声纹识别是一种判断说话人的方式，但目前的语音合成技术可以一定程度上伪造音频，生成有类似声纹的伪造语音。目前的声纹识别系统可能将这类伪造的语音与目标人真实发出的自然语音混淆，容易将它们认为是目标人发出的声音，从而产生认证错误。这从一定程度上影响了声纹识别系统的可靠性，限制了其在实际场景中的应用。本任务计划实现的目标为鉴别自然语音与伪造的合成语音。

在语音合成领域，较成熟的合成方式有基于物理模型的合成方法、基于时域拼接的合成方法、和基于参数模型的方法等几大类[1].基于物理模型的歌声合成方法从人的嗓音机理出发进行物理建模，但难以精确设定参数模拟人类复杂结构的发声器官，导致音色失真。而基于时域拼接的方式较常使用基频同步叠加（Pitch-Synchronous OverLap-Add, PSOLA)类的算法通过对语料库音频进行基频同步分析后进行变速、变调及拼接[2]，但输出质量强烈依赖于基频周期标志点判定，同时无法独立的对基频以及共振峰分别进行处理。基于参数模型的方法则利用机器学习进行参数特征提取后使用隐马尔可夫（HMM）等模型进行合成，相比而言可以具有更高的音色模仿程度，但自然度不及拼接模型。

对以上三种模型生成的伪造语音,学界采用了许多方式来进行鉴别。对于 HMM 生成模型，郭武等人分析了使用 HMM 参数合成法的伪造音频与自然音频在实倒谱域上的区别,发现了倒谱域上的高阶分量上存在差别[3][4]，说明 HMM 合成语音只在一定横渡上还原了频谱包络，而损失了包络细微部分的变化信息。同时，因为人耳对相位信息不敏感，合成器通常会丢弃相位信息[5]，De Leon[6][7]的研究中根据这一特点采用了对相位偏移 RPS[8]的特征进行分辨。对于时域拼接合成的语音，传统方法研究较少，同时由于拼接音频来源为用户自身，所以基于频谱和相位的特征基本没有鉴别能力，目前有可观效果的是 De Leon 从基音模式 PP [9]图像提出的特征[10]，这个特征可以反应语音基频的动态变化以达到区别伪造语音的目的，但所需计算时间较长。还有何朝霞[11]等学者亦从录音时携带的微弱电网信号计算 Duffing 共振特征来判断录音是否经过处理或伪造。

1.2 实验方案设计

考虑到：

①本次提供的数据集不包含拼接法合成的音频；

②本次实验提供的数据集相对干净，没有噪声；

本次实验拟提取梅尔频率倒谱系数 MFCCs（Mel-Frequency Cepstral Coefficients）特征进行处理。不包含拼接法代表频域特征或倒谱域特征可以区分真伪音频；而目标数据噪声较少又避开了 MFCCs 对噪声适应能力弱[12]的缺点。而后将处理后的特征输入神经网络进行训练后使用二分类模型判断输出。整体流程如 图 1-1 所示。

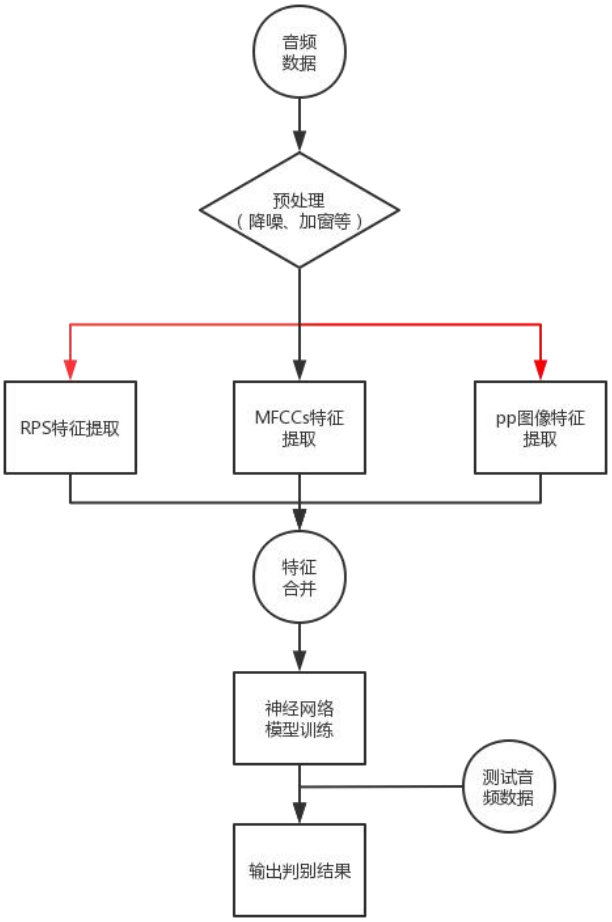

图 1-1 系统工作流程图

不同于常用的 12 维 MFCCs + 2 * 12 维 delta 特征 + 3 * 能量特征共 39 维特征，本次实验直接采用了 40 维 MFCCs 特征保留足够多的高维倒谱域信息，将原来用 delta 特征表示的时序变化情况用神经网络卷积层进行自学习。图中的 RPS 特征与 pp 图像特征为暂未实现的工作内容。

1.3 实现过程

1.3.1 基础方法介绍

1.3.1.1 梅尔频率倒谱系数 MFCCs（Mel-Frequency Cepstral Coefficients）

音频处理集中经常对音频信号进行傅里叶变换转换到频域后进行分析。对原始音频信号的离散 Fourier 变换公式如下所示。

$$X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x(n)e^{-j\omega n} \tag{1}$$

但频域信息仍然不够直观。Fletcher[13]的响度研究表明，人耳对于声音强度的感受度呈指数增长，并且对各个频段的声音感受度也不尽相同，所以处理前首先对频谱取对数，再对对数频谱进行傅里叶变换进入倒谱域。人耳较为敏感的频谱包络相当于频谱图中的低频成

分，而细节则相当于频谱图中的高频成分，所以在倒谱域中加入合适的低通滤波器即可分离包络 h[k]与细节 e[k]。具体实现如以下公式：

$$|X[k]|=|H[k]||E[k]| \tag{2}$$

两边取对数得,

$$log||X[k]|| = log||H[k]|| + log||E[k]|| \tag{3}$$

同时进行 IFFT 得

$$x[k] = h[k] + e[k] \tag{4}$$

而梅尔滤波器则是由根据人类听觉特点设置的一些滤波器组，这些滤波器密集覆盖了低频区域而较少的覆盖在了高频区域，符合人耳听觉在低频区感受更明显的特点[14]。学术界一般处理前我们需要先将频率按如下公式转换为梅尔频率：

$$mel(f) = 2595 * \log_{10}(1 + f/700) \tag{5}$$

将原始音频频率转化为 Mel 频率后，按 Mel 频率等间隔设置若干滤波器组，求取每组的第 2-41 个系数作为特征进行训练。

1.3.1.2 卷积神经网络 CNN（Convolutional Neural Network）

根据以往的研究，CNN 具有很强的适应性，逐渐成为图像和语音领域的主要研究工具[15][16]。尽管语音是在不同时间尺度范围内具有复杂相关性的时变信号，但仍然可以尝试使用基于短时傅里叶变换后的每一帧的 MFCCs 进行时序堆叠后作为 CNN 的输入，这个类频谱图的特征输入包含说话人的身份信息，是一个二维信号。同时，CNN 可以提供在时间和空间上的平移不变性，因此我们可以在不破坏时间序列的情况下获得频谱空间中的声纹特征。

CNN 的卷积层包含多个卷积核（Feature Map）[46][17]。卷积核本质上是一个权重矩阵，输入通过卷积核进行局部滤波。 CNN 层可以有效地从谱图中提取结构特征，并通过权重共享降低模型的复杂度。

1.3.1.3 门控循环单元 GRU（Convolutional Neural Network）

在 GRU 单元中，LSTM 单元中的两个门合并为一个门。

在这个架构中，重置门决定了如何将新输入与之前存储的信息结合起来，而更新门决定了之前存储的内容有多少起作用。在没有输出门的情况下，可以说 GRU 是信息传递和组合的不同实现。 与 LSTM 相比，GRU 网络在结构上做了一些简化，如图 **1-2** 所示。他们之间的函数关系如下：

$$r_t = \sigma(W_{xr}^T \cdot x_t + W_{yr}^T \cdot y_{t-1} + b_r) \tag{5}$$

$$z_t = \sigma(W_{xz}^T \cdot x_t + W_{yz}^T \cdot y_{t-1} + b_z) \tag{6}$$

$$\tilde{y}_t = \tanh(W_{x\tilde{y}}^T \cdot x_t + W_{y\tilde{y}}^T \cdot (r_t \otimes y_{t-1}) + \mathrm{b}_{\tilde{y}}) \tag{7}$$

$$y_t = z_t \otimes y_{t-1} + (1 - z_t) \otimes \tilde{y}_t \tag{8}$$

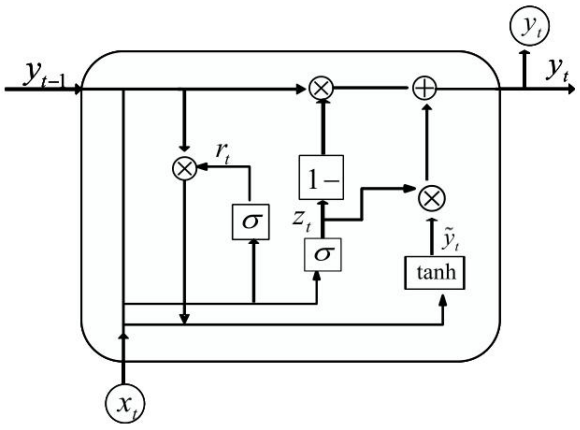

图 1-2 门控循环单元架构

1.3.2 数据选择与预处理

本次实验采用了给定数据集进行训练与调试。将给定数据集进行降噪、加窗后，计算出每帧的短时傅里叶变换（STFT），可视化显示如图 **1-3**。 每帧可以计算出 40 维的 MFCCs 特征。此后提取出每条音频 199 帧的 MFCCs 特征，可以构造出一个 199*40 的矩阵输入神经网络。

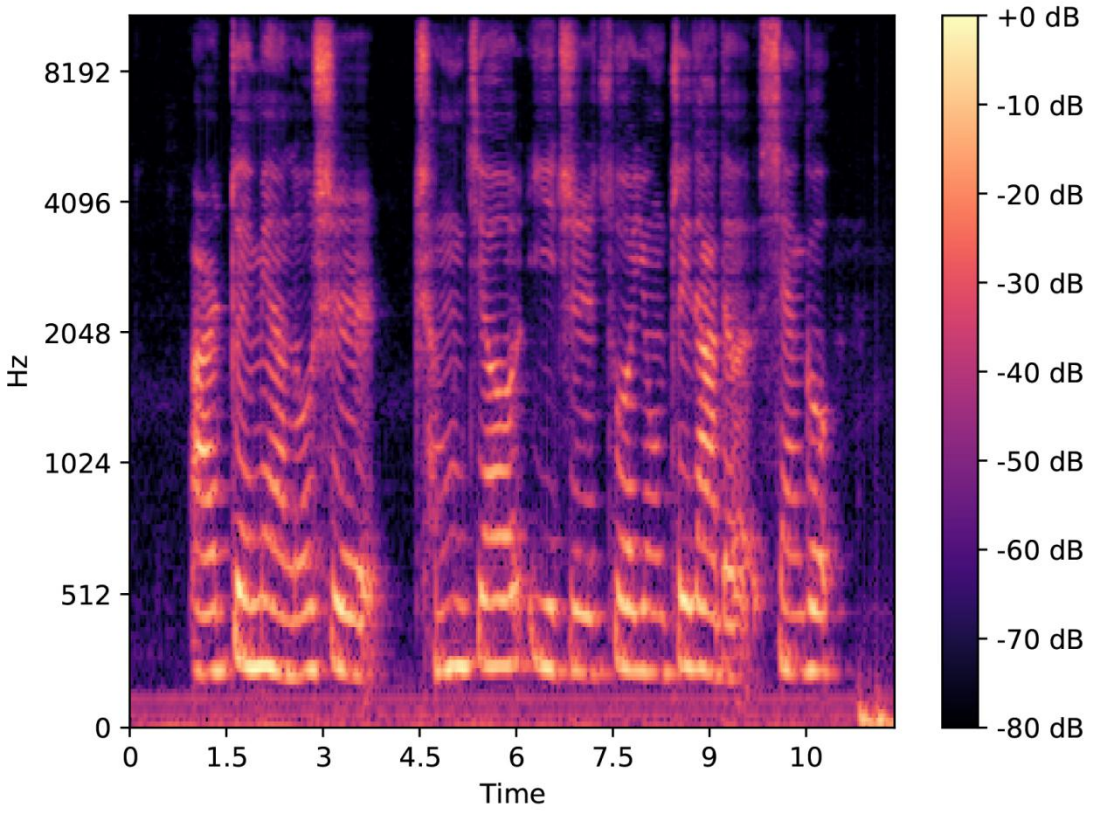

图 1-3 声音的短时傅里叶变换图

1.3.3 神经网络结构的搭建

本次实验使用了(batch_size,cols,filters,channel)的格式进行输入。batch_size 为一个批量的数量，cols 为每个样本使用的帧数，filters 为 mel 滤波器的组数，channel 为通道数。输入的数据将依次进行卷积、池化、三层 GRU 单元、时序全连接、全连接和 softmax 层进行最终分类。

在进行卷积池化后、进行时序全连接后都加入 0.3 的 Dropout，在每两层 GRU 之间同样加入 0.2 的 Dropout 以防止过拟合。同时，在卷积池化时加入了批归一化[18]，在 GRU 层之间加入了层归一化[19]以防止数据绝对值过大而对 sigmoid 函数响应不敏感问题，同时可以加速模型运算。

本次实验所采用的神经网络如下表 **1** 所示。

层名	结构	步长	参数量
Input	(32) * 199 * 40 * 1	-	0
2D Conv	(32) * 100 * 20 * 64	2 * 2	1664
Average Pooling	(32) * 50 * 10 * 64	2 * 2	0
Reshape	(32) * 50 * 640	-	0
GRU_1	(32) * 50 * 1024	-	6297600
GRU_2	(32) * 50 * 1024	-	6297600
GRU_3	(32) * 50 * 1024	-	6297600
TimeDistributed	(32) * 50 * 1024	-	1049600
Dense_1	(32) * 256	-	13107456
Softmax	(32) * 2	-	514

表 1 实验中使用的神经网络结构

1.4 实验结果

采用教师提供的训练集 2500 例、验证集 4000 例进行学习。训练中的损失函数与精确度图像如下图 **1-5** 所示。训练集在 50 轮、验证集在 80 轮左右就已经接近收敛。

在教师提供的测试数据表现中，本次实验的准确性达到了 86.9%。

1.5 问题与改进思路

虽然本模型达到了可观的效果，但仍有一些不足之处有待进一步研究。比如，模型收敛过快，但最终在测试集上的表现确并不高。考虑到模型已经添加了正则化与 Dropout 来防止

过拟合，则可能的问题还是出在分析的特征过于单一导致泛化性不强。噪声的扰动会十分影

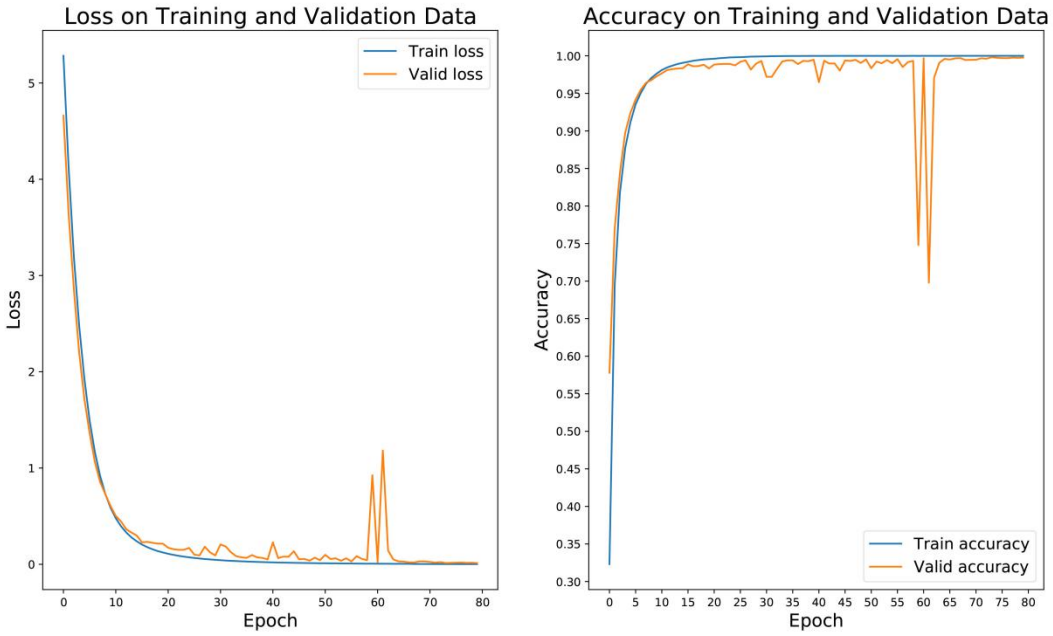

图 1-5 使用 1.3.3 中网络模型的损失和准确率。左图描述了损失与训练轮次的关系，右图描述了准确率与训练轮次的关系。

响 MFCCs 特征的效果。将来的研究中会探索 RPS 特征、pp 图像特征共同参与后对伪造音频判断的处理效果是否有更大的帮助。

1.6 参考文献

[1] 李伟,李子晋,邵曦.音频音乐与计算机的交融--音频音乐技术[M].上海：复旦大学出版社，2019：368-369.

[2]Moulines E,Charpentier F.Pitch-synchronous waveform processing techniques for text-to-speech synthesis using diphones[J]. Speech Communication, 1990, 9(5-6):453-467.

[3] 陈联武, 郭武, 戴礼荣. 声纹识别中合成语音的鲁棒性 [J]. 模式识别与人工智能, 2011, 24(6): 734-747.

[4] Chen L W, Guo W, Dai L R. Speaker verification against synthetic speech [C] International Symposium on Chinese Spoken Language Processing, 2010: 309-312.

[5] Quatieri T F. Discrete-Time speech signal processing: principles and practice [M]. Prentice-Hall, 2002.

[6] De Leon P L, Pucher M, Yamagishi J, et al. Evaluation of speaker verification security and detection of hmm-based synthetic speech [J]. IEEE Transactions on Audio Speech & Language Processing, 2012, 20(8):2280-2290.

[7] Leon P L D, Hernaez I, Saratxaga I, et al. Detection of synthetic speech for the problem of imposture[C]. IEEE International Conference on Acoustics, Speech and Signal Processing, 2011:4844-4847.

[8] Saratxaga I, Hernaez I, Erro D, et al. Simple representation of signal phase for harmonic speech models [J]. Electronics Letters, 2009(45): 381 – 383.

[9] Fujisaki H, Hirose K, Seto S, A method for pitch extraction of speech with reduced errors due to

analysis frame positioning [R]. IEICE Technical Report, 1989.
[10] De Leon P L, Stewart B, Yamagishi J. Synthetic speech discrimination using pitch pattern statistics derived from image analysis [C]. Interspeech, 2012.
[11] He Z , Kai L . Detection of speech authenticity based on duffing resonance[J]. Microcomputer & Its Applications, 2016.
[12] 宫晓梅, 王怀阳. 噪声环境下 MFCC 特征提取[J]. 微计算机信息, 2007(22):247-249.
[13] Loudness, Its Definition, Measurement and Calculation[J]. Bell System Technical Journal, 1933, 12(4):377-430.
[14] Ankur, M.; Divya, K.; Agarwal, R.K. Speaker recognition for Hindi speech signal using MFCC-GMM approach. Procedia Comput.
Sci. 2018, 125, 880–887.
[15] Manabe, K.; Asami, Y.; Yamada, T.; Sugimori, H. Improvement in the Convolutional Neural Network for Computed Tomography
Images. Appl. Sci. 2021, 11 1505.
[16] Lin, Y.-Y.; Zheng, W.-Z.; Chu, W.C.; Han, J.-Y.; Hung, Y.-H.; Ho, G.-M.; Chang, C.-Y.; Lai, Y.-H. A Speech Command Control-Based
Recognition System for Dysarthric Patients Based on Deep Learning Technology. Appl. Sci. 2021, 11 2477.
[17] Naderi, N.; Nasersharif, B. Ultiresolution convolutional neural network for robust speech recognition. In Proceed-
ings of the 2017 Iranian Conference on Electrical Engineering (ICEE), Tehran, Iran, 2–4 May 2017; pp. 1459–1464,
doi:10.1109/IranianCEE.2017.7985272.
[18] Ioffe, S.; Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In
Proceedings of the 32nd International Conference on Machine Learning, Lille, France, 7–9 July 2015; pp. 448–456.
[19] Ba, J.L.; Kiros, J.R.; Hinton, G.E. Layer normalization. arXiv 2016, arXiv:1607.06450.

2. 源码及解析

2.1 源码工程结构

源码工程一共由五个文件组成：

1. Feature_extraction.py 进行特征提取
2. Generate.py 输入音频，生成结构化数据供神经网络训练。
3. Model.py 存放神经网络模型
4. Predict.py 用训练好的模型进行预测
5. Train.py 进行模型训练

五个文件均包含在主目录下。

2.2 源码参考

无

2.3 源码解析

2.3.1 feature_extraction.py

此文件实现了特征提取。利用 librosa 哭的 mfcc 函数提取了目标音频的 mfcc（包含预加重、分帧和加窗、FFT、取绝对值、mel 滤波、取对数、离散余弦变换六步），此后对向量

进行归一化。对每帧重复此操作，并选出每个样本的前 199 帧（不足则补 0）叠加成 199*40 的特征矩阵返回。

```python
import librosa
import numpy as np
import librosa.display


class FeatureExtraction:
    def __init__(self, path=r"D:\Users\11201\Downloads\data_aishell\data_aishell\wav\train\S0002\BAC009S0002W0123.wav",
                 sr=16000, n_mfcc=40):
        self.y, self.sr = librosa.load(path, sr=sr)

        self.n_mfcc = n_mfcc
        self.mfccs = librosa.feature.mfcc(y=self.y, sr=self.sr, n_mfcc=self.n_mfcc)
        self.normalized_mfccs = (self.mfccs - np.repeat(np.mean(self.mfccs, axis=1).reshape(self.n_mfcc, 1),
                                                        self.mfccs.shape[1], axis=1)) / \
                                np.repeat(np.std(self.mfccs, axis=1).reshape(self.n_mfcc, 1), self.mfccs.shape[1], axis=1)
        self.normalized_mfccs = self.normalized_mfccs.T

    def vector_199_40(self):
        if self.normalized_mfccs.shape[0] >= 199:
            return self.normalized_mfccs[:199, :]
        return np.pad(self.normalized_mfccs, ((0, 199 - self.normalized_mfccs.shape[0]), (0, 0)), 'constant',
                      constant_values=(0, 0))


if __name__ == "__main__":
    print(FeatureExtraction().vector_199_40().shape)
```

图 2-1　feature_extraction.py 代码实现

2.3.2 generate.py

该文件继承 tf.keras.utils.Sequence 实现了一个对于时间序列的 generator。

实现了__len__方法返回总批次数，实现__getitem__（随机）返回一个 batch_size 的样本数据与对应标签。其中的“样本”是经前一步特征提取后的特征矩阵。

```python
import tensorflow
import numpy as np
from feature_extraction import FeatureExtraction


class DataGenerator(tensorflow.keras.utils.Sequence):
    def __init__(self, files, labels, batch_size=32, shuffle=True, random_state=42):
        'Initialization'
        self.files = files
        self.labels = labels
        self.batch_size = batch_size
        self.shuffle = shuffle
        self.random_state = random_state
        self.on_epoch_end()

    def __len__(self):
        return int(np.floor(len(self.files) / self.batch_size))

    def __getitem__(self, index):
        # Generate indexes of the batch
        indexes = self.indexes[index * self.batch_size:(index + 1) * self.batch_size]

        files_batch = [self.files[k] for k in indexes]
        y = np.array([self.labels[k] for k in indexes])

        # Generate data
        x = self.__data_generation(files_batch)

        return x, y
```

图 2-2　generate.py 代码实现 1

```
    def on_epoch_end(self):
        'Updates indexes after each epoch'
        self.indexes = np.arange(len(self.files))
        if self.shuffle:
            np.random.seed(self.random_state)
            np.random.shuffle(self.indexes)

    def __data_generation(self, files):

        features = []
        for audio_file in files:
            feature = FeatureExtraction(path=audio_file).vector_199_40()
            features.append(feature)
        features = np.stack(features, axis=0)
        features = features.reshape((features.shape[0], features.shape[1], features.shape[2], 1))

        return features

```

图 2-3　generate.py 代码实现 2

2.3.3 model.py

该文件按 1.3 中的结构自定义了一个神经网络。

```
import tensorflow as tf

num_classes = 2


def camp_model():
    inputs = tf.keras.layers.Input(shape=(199, 40, 1))
    x = tf.keras.layers.Conv2D(filters=64, kernel_size=(5, 5), strides=2, padding='same')(inputs)
    x = tf.keras.layers.Dropout(0.3)(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.AveragePooling2D(pool_size=2, strides=2)(x)
    x = tf.keras.layers.BatchNormalization()(x)
    num0utput_P = 64 * x.get_shape()[2]
    x = tf.keras.layers.Reshape((-1, num0utput_P))(x)
    # x = tf.keras.layers.Masking()(inputs)
    x = tf.keras.layers.GRU(1024, return_sequences=True)(x)
    x = tf.keras.layers.Dropout(0.2)(x)
    x = tf.keras.layers.LayerNormalization()(x)
    x = tf.keras.layers.GRU(1024, return_sequences=True)(x)
    x = tf.keras.layers.Dropout(0.2)(x)
    x = tf.keras.layers.LayerNormalization()(x)
    x = tf.keras.layers.GRU(1024, return_sequences=True)(x)
    x = tf.keras.layers.Dropout(0.2)(x)
    x = tf.keras.layers.LayerNormalization()(x)
    x = tf.keras.layers.TimeDistributed(tf.keras.layers.Dense(1024, activation='relu'))(x)
    # x = tf.keras.layers.Lambda(lambda x: x[:, :, 0])(x)
    # x = tf.keras.layers.Dense(units=512, activation='relu')(x)
    x = tf.keras.layers.Flatten()(x)
    x = tf.keras.layers.Dense(units=256, activation='relu')(x)
    x = tf.keras.layers.Dense(units=num_classes, activation='softmax')(x)

    model = tf.keras.models.Model(inputs, x)
    return model
```

图 2-4　model.py 代码实现

2.3.4 predict.py

该文件读入了训练好的模型参数，并读入了提供的测试文本提取其中的音频文件地址，将对应的音频文件转化为特征矩阵后送入网络进行预测并将结果输出至文本文档。

```python
def predictModel(model, output_model_file):
    # load the weights of model
    model.load_weights(output_model_file)
    features = []
    test_dir = r'C:\Users\11201\Downloads\zju_deepfake\eval1\flac'
    # test_dir = r'D:\Users\11201\Downloads\data_aishell\data_aishell\wav\train\S0002'
    res = []
    # with open("result.txt", "w") as f:
    num = 0
    fname = []
    for root, path, files in os.walk(test_dir):
        for file in files:
            num += 1
            audio_path = os.path.join(root, file)
            # feature = FeatureExtraction(path=audio_path).vector_199_40().reshape(199, 40, 1)
            # # print(feature.shape)
            # pre = model.predict(feature)
            # if pre[0][0] > pre[0][1]:
            #     res.append((audio_path.split("\\")[-1].split(".")[0], "spoof"))
            #     print(audio_path, ' is classified as Spoof.')
            # else:
            #     res.append((audio_path.split("\\")[-1].split(".")[0], "bonafide"))
            #     print(audio_path, ' is classified as Bonafide.')
            fname.append(audio_path.split("\\")[-1].split(".")[0])
            feature = FeatureExtraction(path=audio_path).vector_199_40().reshape(199, 40, 1)
            features.append(feature)
    features = np.stack(features, axis=0)
    pre = model.predict(features)
    pre = np.argmax(pre, axis=1)
    spoof = 0
    with open("result0716.txt", "w") as f:
        for i in range(len(pre)):
            f.writelines(fname[i] + " " + ("spoof" + "\n" if pre[i] == 0 else "bonafide" + "\n"))
            spoof += pre[i] == 0
    print(spoof)
```

图 2-5　predict.py 代码实现

2.3.5 train.py

该文件用于对神经网络进行训练，主要分四个部分：训练、绘图、获得标签和保存进度。

trainModel 函数如下所示，输入 2.2 中定义的 generator 和 2.3 中定义的模型结构，设置好回调参数即开始训练。

```python
def trainModel(model, train_generator, valid_generator, callbacks):
    history = model.fit(
        train_generator,
        epochs=epochs,
        validation_data=valid_generator,
        callbacks=callbacks,
        # shuffle=True
    )
    return history
```

plot_learning_curves 函数如下所示，其就是一个简单的绘图函数以可视化 loss 和 acc 的变化趋势。

```python
def plot_learning_curves(history, label, epochs, min_value, max_value):
    data = {}
    data[label] = history.history[label]
    data['val_' + label] = history.history['val_' + label]
    pd.DataFrame(data).plot(figsize=(8, 5))
```

```
    plt.grid(True)
    plt.axis([0, epochs, min_value, max_value])
    plt.show()
    return history
```

get_files_and_labels 函数如下所示，用于从给定的文本文件中获得音频地址和对应的标签。

```
def get_files_and_labels(txt_path, mode):
    X = []
    y = []
    with open(os.path.join(train_dir, txt_path)) as f:
        num = 0
        for line in f.readlines():
            num += 1
            if num % 500 == 0:
                print(num)
            fname, tag = line.split()
            tag = np.array([1, 0]) if tag == "spoof" else np.array([0, 1])

            X.append(os.path.join(train_dir, mode, "flac", "%s.flac" % fname))
            y.append(tag)
        y = np.vstack(y)
        # print(X, y.shape, X[0], y[0])
        return X, y
```

主函数如下所示。当存在预训练参数时直接读取，否则进行选择，1 为训练，2 为预测。选择训练则通过 trainModel 函数进行训练，选择预测则通过 predict.py 中的函数进行预测。

```
if __name__ == '__main__':
    print('Building model...')

    # Build model
    model = model.camp_model()
    model.compile(loss='categorical_crossentropy',
                  optimizer='adam', metrics=['accuracy'])
    model.summary()

    # set path of saving model
    logdir = os.path.join('graph_def_and_weights')
    if not os.path.exists(logdir):
        os.mkdir(logdir)

    output_model_file = os.path.join(logdir, "zjuCamp_weights.h5")

    print('Start training ...')
    # start training
```

```
mode = input('Select mode: 1.Train 2.Predict\nInput number: ')
if mode == '1':

    files, labels = get_files_and_labels("train.txt", "train")
    # print(files, labels)
    train_generator = DataGenerator(files, labels)
    files, labels = get_files_and_labels("dev.txt", "dev")
    val_generator = DataGenerator(files, labels)

    for i in range(len(train_generator)):
        x, y = train_generator[i]
        # print(x, y)
        print('%s => %s' % (x.shape, y.shape))
    callbacks = [
        keras.callbacks.TensorBoard(logdir),
        keras.callbacks.ModelCheckpoint(output_model_file,
                                        save_best_only=True,
                                        save_weights_only=True),
        keras.callbacks.EarlyStopping(patience=5, min_delta=1e-3)
    ]
    # model.load_weights(output_model_file)
    history = trainModel(model, train_generator, val_generator, callbacks=callbacks)
    plot_learning_curves(history, 'accuracy', epochs, 0, 1)
    plot_learning_curves(history, 'loss', epochs, 0, 5)
elif mode == '2':
    # Only run this mode if you have already finished training your model and saved it.
    predictModel(model, output_model_file)
else:
    print('Please input the correct number.')

print('Finish! Exit.')
```