

HW1-P2 (F23)

2024年8月10日 星期六 17:36

本项目以F23版为主，S24版仅仅是数据集MFCC的特征由28维变为27维。作业讲义的内容几乎完全一致。

本作业最终精度：85% (F23 kaggle HW1P2最高87%)

本项目实践的代码和经验：

1. R-drop
2. 优化器训练中途切换：Adam to SGD
3. 各种mask预处理
4. 各种消融实验

[1 项目目标](#)

[2 语音处理领域简介](#)

[2.1 音色](#)

[2.2 音调](#)

[2.3 音强](#)

[2.3.1 关于振幅的单位](#)

[2.3.2 关于分贝](#)

[2.4 音长](#)

[2.4.1 关于采样率](#)

[2.4.2 问题：为什么录音设备能够在极短时间内完全记录声音的音调、音色、音强？](#)

[2.5 语音信号的时域图、频谱图](#)

[2.5 梅尔频率倒谱系数 \(MFCC\)](#)

[2.5.1 基本概念](#)

[2.5.2 计算过程](#)

[2.5.3 应用](#)

[2.5.4 MFCC图、及其一阶微分、二阶微分图](#)

[2.6 音素](#)

[3 数据集介绍](#)

[4 数据处理介绍](#)

[5 消融实验](#)

[5.1 模型结构、参数实验](#)

[5.2 其他实验](#)

[5.3 全数据集实验](#)

[6 附GPT回答：](#)

[6.1 关于前后填充帧数的选择](#)

[6.2 关于mask方法的建议](#)

1 项目目标

任务：自定义一个纯MLP模型，实现对音素数据的分类，项目得分根据所有人的精度排名来定。

项目难点：数据集的理解以及训练集的制作和预处理。难点主要自定义dataset类。

音素共40类，音素暂时简单理解为音标发音，如下：

```
PHONEMES = [
    '[SIL]', 'AA', 'AE', 'AH', 'AO', 'AW', 'AY',
    'B', 'CH', 'D', 'DH', 'EH', 'ER', 'EY',
    'F', 'G', 'HH', 'IH', 'IY', 'JH', 'K',
    'L', 'M', 'N', 'NG', 'OW', 'OY', 'P',
    'R', 'S', 'SH', 'T', 'TH', 'UH', 'UW',
    'V', 'W', 'Y', 'Z', 'ZH', '[SOS]', '[EOS]']
```

- [SOS]和[EOS]：分别是标签文件中的起始和结束符，不算做音素类型。
- [SIL]：静默音素，即代表此语音片段是静默无声状态。

数据形态和数量详解见下文第3章。

2 语音处理领域简介

本章节自己查资料、做实验补充。课程讲义没有。

2.1 音色

又称为音质，是一种声音区别于另一种声音的基本特性。与人声带的振动频率、发音器官的送气方式和声道的形状、尺寸密切相关（由不同频率成分的组合和强度决定）。

2.2 音调

声音的高低，取决于声波的频率（由采样捕捉到的频率变化决定）。

下面是一些常见动物能够听到的声音频率范围：

1. **人类**：20-20000赫兹（20kHz）。年龄和听力健康状况会影响这个范围。
2. **狗**：40-60000赫兹。狗能听到比人类高得多的频率，这就是为什么狗可以听到一些对人类来说是超声波的哨声。
3. **猫**：48-85000赫兹。猫的听觉比狗还要敏感，特别是在高频范围内。
4. **蝙蝠**：1000-120000赫兹。蝙蝠依赖超声波进行回声定位。

2.3 音强

声音的强弱，它由声波的振动幅度所决定。

2.3.1 关于振幅的单位

在数字音频处理中，音频信号的振幅值通常没有特定的物理单位，因为它们表示的是声音波形的数字化样本。这些振幅值是原始模拟声音信号的量化表示，其大小取决于声音录制或处理时使用的比特深度（bit depth）。

- **比特深度**：这是每个音频样本的位数。常见的比特深度包括16位、24位或32位。比特深度越高，音频的动态范围越大，能够更精确地表示声音的细微差别。
- **归一化的值**：在许多音频处理软件和库（如 `librosa`）中，振幅值常常被归一化。例如，对于16位音频，原始的振幅值可能在 -32768 到 32767 之间，但在处理时，这些值会被归一化到 -1.0 到 1.0 之间。这意味着，不论实际的比特深度是多少，振幅值都会被调整到这个标准范围内，使得处理更加一致。
- **物理意义**：尽管数字振幅值本身没有直接的物理单位，但它们与实际声音的压力波形有对应关系。较大的振幅值对应于声波的较高压力水平，通常我们会感知为更响的声音。

2.3.2 关于分贝

分贝（dB）是一个对数单位，用于描述幅度的相对水平。幅度转换为分贝单位的过程，实质上是一种将振幅值转换为更符合人耳感知的形式的过程。

人类的听觉感知对声音的强度是对数响应的，这意味着当声音的物理能量增加时，我们感知到的声音响度增加的速度要慢得多。分贝单位正是基于这种对数感知来量化声音响度的。具体来说，声音的响度（分贝值）计算公式通常如下：

$$\text{分贝} = 20 \times \log_{10} \left(\frac{\text{信号幅度}}{\text{参考幅度}} \right)$$

其中，参考幅度通常是一个标准压力值（通常用于声音），或者是信号幅度中的最小值或平均值。

在分贝尺度上，即使是小的变化也可以更容易地观察到，而在线性尺度上这些变化可能会不那么明显。

2.4 音长

声音的长短，取决于发音持续时间的长短。

2.4.1 关于采样率

- 采样是将连续的模拟音频信号转换为离散的数字信号的过程。
- 采样率（每秒的样本数），表示每秒钟从连续的音频信号中提取多少个样本。
- 例如，一个常见的采样率是44100 Hz，意味着每秒钟有44100个样本。
- 采样率越高，音频的质量通常越好。

2.4.2 问题：为什么录音设备能够在极短时间内完全记录声音的音调、音色、音强？

在极短的音频采样过程中，计算机能够完全记录声音的音调、音色、音强等属性，主要是因为现代数字音频技术采用了足够高的采样率和比特深度来准确捕捉声音的各种特性。这是通过两个关键的数字音频处理技术实现的：采样（Sampling）和量化（Quantization）。

1. 采样（Sampling）：

- 采样是将连续的模拟音频信号转换为离散的数字信号的过程。这是通过在规定的时间间隔内测量音频信号的振幅来完成的。
- 采样率（Sampling Rate），即每秒采样次数，是这个过程的关键参数。根据奈奎斯特定理（Nyquist Theorem），为了准确捕捉到所有可听频率的音频信号，采样率应该至少是声音信号最高频率的两倍。对于人类听觉，这意味着采样率至少需要为40kHz。实际上，大多数音频系统使用44.1kHz或更高的采样率，足以捕捉到20kHz以下的所有听觉频率。

2. 量化（Quantization）：

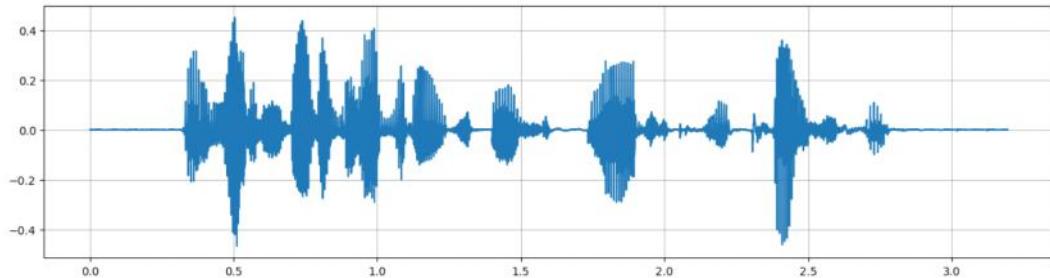
- 量化是将采样得到的连续振幅值转换为离散值的过程。这是通过比特深度（Bit Depth）来控制的，比特深度越高，能够表示的振幅级别就越多，从而能更精确地表示声音的动态范围（从最安静到最响亮的声音）。
- 例如，16位音频可以提供65,536 (2^{16}) 个不同的振幅级别，而24位音频可以提供超过1600万 (2^{24}) 个级别。

通过这两种技术，计算机可以在极短的时间内捕捉到声音的细微变化，包括音调（由采样捕捉到的频率变化决定）、音色（由不同频率成分的组合和强度决定）和音强（由振幅的大小决定）。因此，即使是极短的音频采样，只要采样率和比特深度足够高，计算机也能够完整地记录下声音

的所有这些属性。

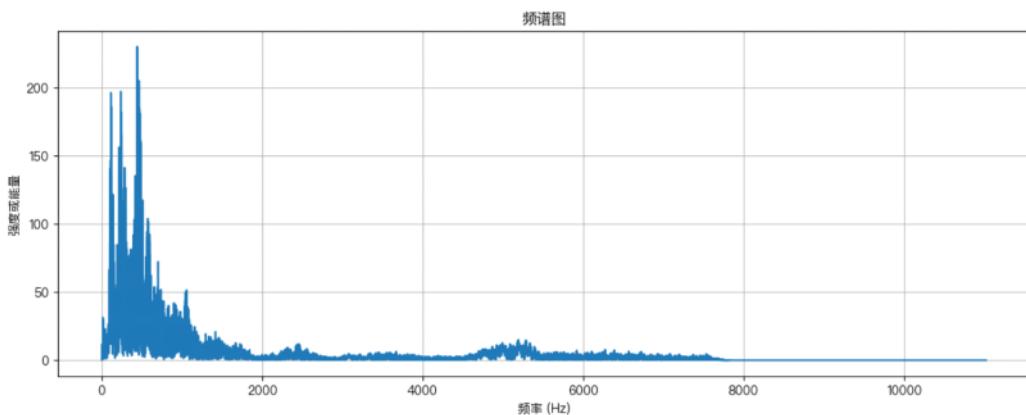
2.5 语音信号的时域图、频谱图

语音 “And what sort of evidence is logically possible” 的时域图如下：



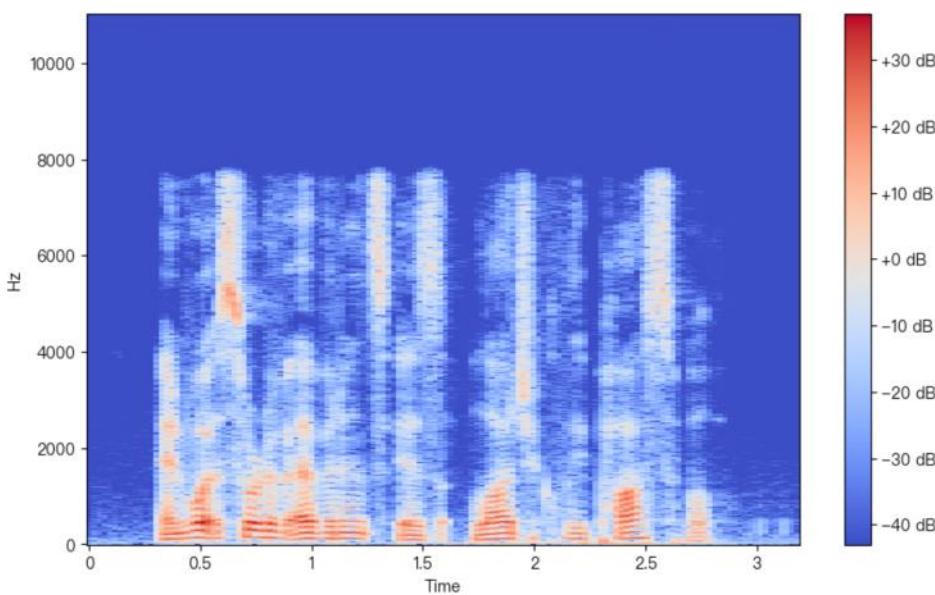
- **横轴**：表示时间（秒）
- **纵轴**：表示声音信号的振幅（已归一化）
- **采样率**：22050 Hz，意味着每秒钟有22050个样本。（绘制图时一般不会完全显示所有点）

频谱图（Spectrum Plot）如下：



- **横轴 (x轴)**：横轴是频率 (Hz)。显示了音频信号频率成分的范围。
- **纵轴 (y轴)**：纵轴是能量或强度（通常用分贝表示）。它主要用于展示音频信号中不同频率成分的能量大小。
- 此图展示的是信号在整个录音期间或特定时间窗口内的频率分布和能量强度。

频谱图（Spectrogram）：



- **横轴 (x轴)**：表示时间。
- **纵轴 (y轴)**：表示频率。
- 显示了音频信号中频率随时间的变化情况。每个格子中的颜色强度表示在特定时间和频率下的信号幅度（分贝表示）。

2.5 梅尔频率倒谱系数 (MFCC)

MFCC (Mel Frequency Cepstral Coefficients, 梅尔频率倒谱系数) 是一种在语音和音频处理中广泛使用的特征表示方法。它们特别适用于语音识别和相关任务，**因为它们能够有效地捕捉到语音信号的关键特性**。

2.5.1 基本概念

1. 梅尔尺度 (Mel Scale) :

- 梅尔尺度是基于人耳对不同频率声音敏感度的感知特性而设计的。它是一种非线性的频率尺度，使得MFCC能够更贴近人类的听觉感知。
- 低频下，梅尔尺度与线性频率尺度类似，但在高频时，梅尔尺度增长得更缓慢。

• 倒谱 (Cepstrum) :

- 倒谱是一种信号处理技术，通过对信号的频谱进行逆傅里叶变换来得到的。它有效地分离了信号中的周期性（谐波）和非周期性（如噪声）成分。

2.5.2 计算过程

MFCC的计算过程通常包括以下步骤：

1. 预加重:

- 对原始音频信号应用高通滤波，增强高频成分，模拟人耳对高频声音的敏感度。

2. 分帧和加窗:

- 将音频信号分割成小片段（帧），每帧一般在20-40毫秒之间。
- 对每帧应用窗函数（如汉明窗），减少帧边界的不连续性。

3. 快速傅里叶变换 (FFT) :

- 对每帧信号进行FFT，将其从时域转换到频域。

4. 梅尔滤波器组:

- 应用一组梅尔滤波器（通常是三角形滤波器组）到FFT结果上。这些滤波器覆盖了整个可听频率范围。
- 通过梅尔滤波器组提取每帧的能量分布。

5. 对数能量：

- 对每个滤波器输出的能量取对数。这样做是为了模拟人耳对声音响度的感知（对数响应）。

6. 离散余弦变换 (DCT) :

- 对梅尔滤波器组的对数能量输出应用DCT，从而得到最终的MFCC。
- DCT有助于去除特征向量中的相关性，并产生更紧凑的特征表示。

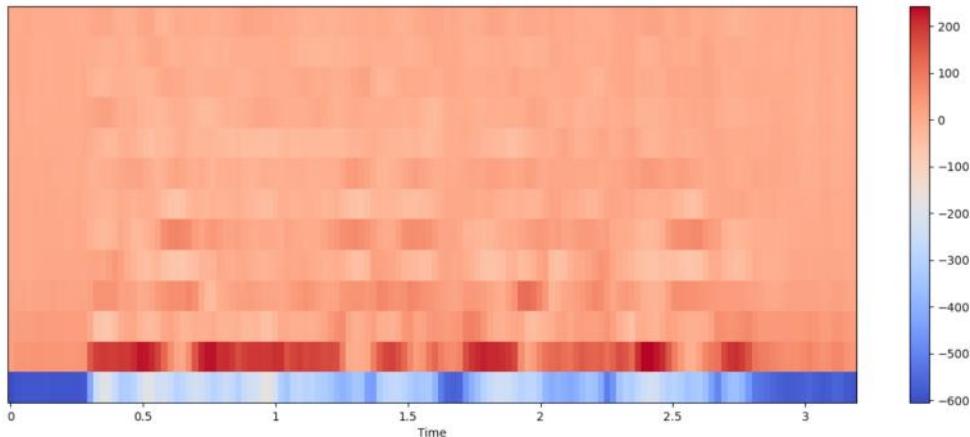
2.5.3 应用

- **语音识别**: MFCC在自动语音识别系统中是一种标准的特征，因为它能够有效地表示语音的特性。
- **说话者识别**: MFCC也被用于识别不同的说话者，因为它可以捕捉个体的发音特征。
- **音频分类**: 在音乐信息检索和情感分析等领域，MFCC也是一种重要的特征。

MFCC之所以有效，是因为它结合了人类的听觉感知特性和先进的信号处理技术，能够从原始音频信号中提取出有用的信息。

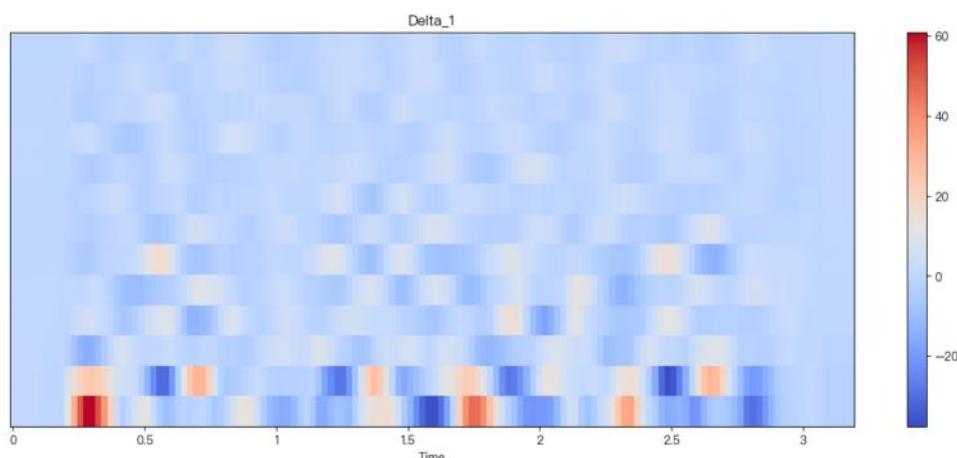
2.5.4 MFCC图、及其一阶微分、二阶微分图

“And what sort of evidence is logically possible” 音频的MFCC图如下（设定13个特征）：

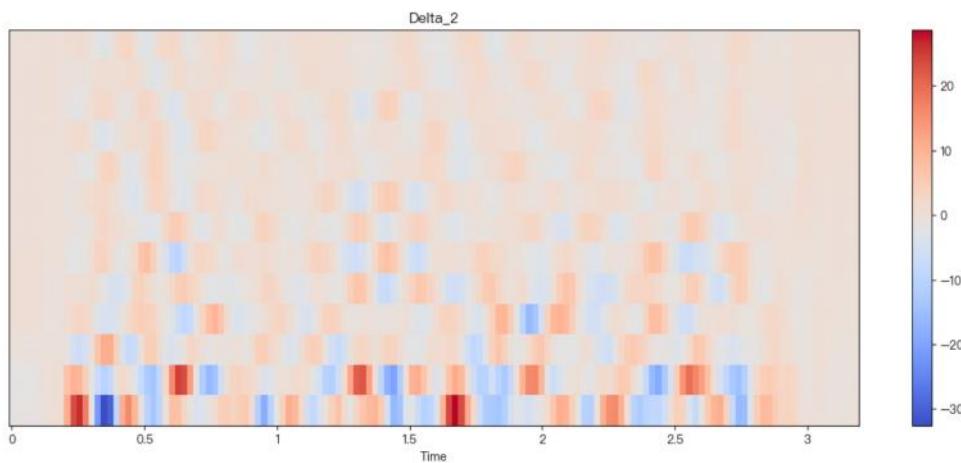


- **横轴**: 时间轴。它表示音频的时间进程。由于MFCC是按音频帧计算的，因此横轴显示了整个音频信号的持续时间。**每一列代表一帧**。本例MFCC划分为138个音频帧（**每帧持续时间一般是20-40毫秒**）。
- **纵轴**: MFCC特征（或系数）。纵轴显示了计算出的13个MFCC特征，这些特征是从低到高排序的。每一行代表一个特定的MFCC特征。这些特性包括声音的总体能量（通常由第一个MFCC特征表示）、声音的频谱形状、声音的纹理等。不同的MFCC特征捕获了从低到高的不同频谱特性。较低序号的特征通常与音频信号的一般特性（如音调）相关，而较高序号的特征则与更细微的特性（如特定发音的细节）相关。
- **颜色**: 颜色的强度表示每个MFCC特征在每个时间点的大小或值，这提供了关于音频信号在那一瞬间的详细声学特性的信息。

MFCC的一阶微分（MFCC特征随时间的变化率）：



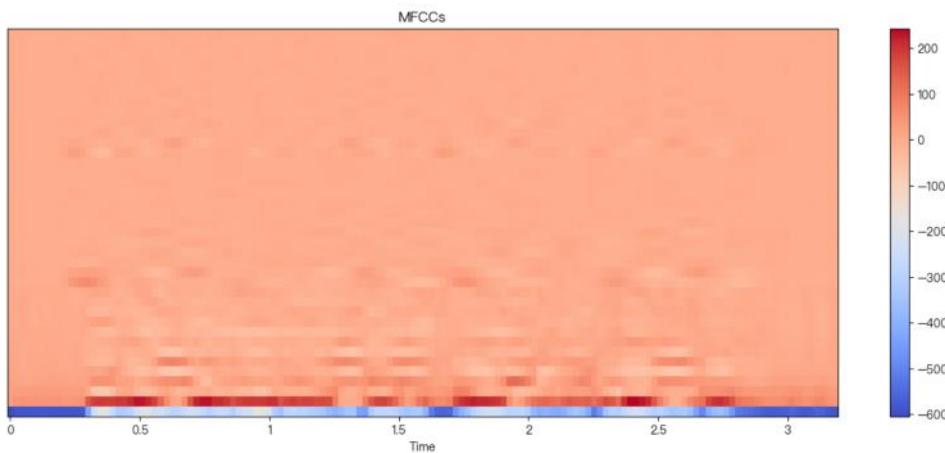
MFCC的二阶微分（MFCC特征动态变化的加速度）：



结合MFCC、一阶和二阶特征：

将原始的MFCC特征、一阶微分（Delta_1）和二阶微分（Delta_2）合并（直接向量拼接，形象的说就是将3张图垂直方向拼接在一起）成一个综合特征集。

- 这种结合方式是常见的做法，因为它不仅包括了音频信号的静态特征（MFCC），还包括了动态特征（Delta和Delta-Delta）。
- 这样的特征集提供了更全面的信息，对于训练更复杂、更准确的语音识别模型非常有帮助。



上图就是3个特征图垂直拼接图，每一层就是原来单个特征图。注意：分贝强度刻度尺自适应调整了。

2.6 音素

音素：人说话的声音是由若干单个的音组成的，即使是一个很短的字、词也是由一定的读音组成的。英语把组成一个读音的最小单位叫音素。英语音素，分为元音20个，辅音28个。

英语音素分类表(48个)								
元音 (20个)								
单元音	/i:/	/ɛ:/	/ɔ:/	/u:/	/ə:/			
	/ɪ/	/ə/	/ɑ/	/ʊ/	/ʌ/	/e/	/æ/	
双元音	/eɪ/	/aɪ/	/ɔɪ/	/əʊ/	/aʊ/	/ɪə/	/ʊə/	
辅音 (28个)								
清辅音	/p/	/t/	/k/	/f/	/s/	/θ/	/ʃ/	/tʃ/
	/tʃr/	/ts/	/h/					
浊辅音	/b/	/d/	/g/	/v/	/z/	/ð/	/ʒ/	/dʒ/
	/dr/	/dʒz/						
	/m/	/n/	/ɳ/	/l/	/ɳ/	/w/	/j/	

音素和音标的区别：

- 音素是从音质角度划分的最小的语音单位，从发音特征上可分为两类，即元音（也叫母音）音素和辅音（也叫子音）音素。
- 音标是记录音素的符号，是音素的标写符号。
- 它的制定原则是：一个音素只用一个音标表示，而一个音标并不只表示一个音素（双元音就是由2个音素组成的，相对于单元音来说。由2个音素构成的音标我们称之为双元音）。

注意：不同的**英语变体**（如美式英语、英式英语、澳大利亚英语等）、**不同的方言**有不同的音素集。在语音识别系统的实际应用中，有可能会去除一些很不常用的音素，并加入一些**呼吸声、咳嗽声、噪声、啧啧声**等其他对语音处理有帮助的声音事件标签。

关于音素发音的持续时间：与语速、强调和情感状态、语言风格相关。一般在**50~200毫秒**左右，甚者更长。

3 数据集介绍

训练集、验证集、test数量：

```
data
└── train-clean-100
    ├── mfcc (28539, )
    └── transcript (28539, )
└── dev-clean
    ├── mfcc (2703, )
    └── transcript (2703, )
└── test-clean
    └── mfcc (2620, )
```

提供了语音的**MFCC数据**（每个数据用一个.npy文件存储），及其对应的**转录文件**（即音素标签，也用一个跟对应MFCC数据同名的.npy文件存储）。

- .npy格式：**NumPy库用来存储数组数据的一种文件格式。
- MFCC数据：**每个MFCC数据文件源自一段语音数据（好像都是一句话的长度）。MFCC中每25ms的语音信号浓缩为一帧，提炼为28个特征（即28个浮点数表示）。注意：单个样本文件中，MFCC数据帧和帧之间的步长是10ms，即意味着帧之间有15毫秒的语音重叠（25毫秒的帧长减去10毫秒的步长），这种重叠可以使得语音信号中的连续信息在帧与帧之间平滑过渡，减少信息的丢失。
 - 物理意义说明：**比如“19-198-0000.npy”文件，它本身是由一句音频（假设是“are you ok”）数据转成MFCC数据得来的。该数据的维度为**(192, 28)**，即该MFCC有192个帧，每帧提炼浓缩出了25ms音频时长中的28个特征。这个文件原始语音信息的时长能计算出来： $25\text{毫秒} + (192 - 1) * 10\text{毫秒} = 1.935\text{秒}$ 。即说话人在说“are you ok”时，大概花了1.9秒，注意数据采集时，前后还有短暂的音频静默时刻。
 - 下图是原始音频数据（时域图）转成MFCC数据的示意图，“19-198-0000.npy”文件类似于最下方那个，项目里MFCC特征是28而非13。

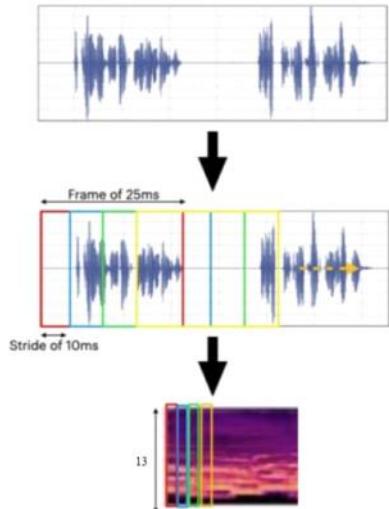


Figure 2: Feature extraction

- **转录文件**: 即MFCC数据对应的音素标签文件，本质就是存了个list数组，里面存储了MFCC文件每帧对应的音素标签。
 - 比如“19-198-0000.npy”MFCC数据对应的转录文件，维度为(194,)，而这个MFCC维度为(192, 28)，之所以多了2帧，是因为转录文件数据中首尾分别加了 [SOS] 和 [EOS]，分别代表“Start of Speech”和“End of Speech”，表示语音片段的开始和结束。完整的标签内容如下：[SIL] 表示“silence”（静默帧，即本帧内没有在说话）

4 数据处理介绍

1. **抽取全部数据集**: 训练集总共28000多个文件，每个文件的维度为[MFCC帧数, 28]，将所有文件全部提取出来放在一个数组中。
 2. **前后填充帧**: 由于单个音素的**完整发音**持续时间一般在50~200毫秒，而MFCC单帧采集的音频时间是25毫秒。所以，单帧MFCC数据并不能用于音素分类，需要在单帧的前后位置填充邻近的MFCC数据一同输入网络进行音素分类。本模型默认前后各填充20帧（可选超参数），即总共41帧（约1秒时长）连续的MFCC数据，用于预测这单帧MFCC属于什么音素类别。
 3. **输入MLP网络的样本形态**: 单个样本的输入维度形如[41, 28]，41是连续的41帧音频MFCC数据（注意帧和帧之间其实有10ms信息重叠），28是MFCC的特征维度。
 4. 作业手册建议尝试用mask预处理，尝试了时间轴上mask和MFCC特征维度mask，均无助于提升精度，具体见消融实验。

5 消融实验

HW1P2作业讲义中最基本的建议：

Hyperparameters	Values
Number of Layers	2-8
Activations	ReLU, LeakyReLU, softplus, tanh, sigmoid
Batch Size	64, 128, 256, 512, 1024, 2048
Architecture	Cylinder, Pyramid, Inverse-Pyramid, Diamond
Dropout	0-0.5, Dropout in alternate layers
LR Scheduler	Fixed, StepLR, ReduceLROnPlateau, Exponential, CosineAnnealing
Weight Initialization	Gaussian, Xavier, Kaiming(Normal and Uniform), Random, Uniform
Context	0-50
Batch-Norm	Before or After Activation, Every layer or Alternate Layer or No Layer
Optimizer	Vanilla SGD, Nesterov's momentum, RMSProp, Adam
Regularization	Weight Decay
LR	0.001, you can experiment with this
Normalization	You can try Cepstral Normalization

Table 1: Hyperparameter Tuning

讲义中特别提到的建议：

- 权重初始化的尝试。
- R-Drop (Regularized Dropout)
- Mask：对帧的mask，以及对MFCC每帧中28个特征的mask。

5.1 模型结构、参数实验

基线参数：

- 数据集：只选了1/10数据集测试。
- context = 20 (前后各填充20帧。注意，相邻两帧，有15ms重叠的语音信息，所以实际上前后各填充20帧的数据，实际源音频时长并不是 $41 \times 0.025 = 1$ 秒，而是 $\approx 41 \times 0.010 = 0.4$ 秒)
- MLP结构：[1, 2, 4, 8, 4, 2, 1] * 512 + 输出层（总参数量22M），**线性层→BN→激励函数→dropout**。
 - 参数初始化方法：
- 激励函数：默认GELU

基线	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
dropout=0.2		✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓
BN			✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
测试：GELU激活 → BN (放激励函数后面)				✓										

测试: Tanh激活 → BN					✓										
测试: BN → Tanh激活						✓									
测试使用随机正态分布 权重初始化 (框架默认用Kaiming 均匀初始化)							✓								
测试删除dropout								✓							
测试dropout 放 BN 前面									✓						
测试使用LeakyReLU (本模型默认用 GELU)										✓					
测试增加2层: [1, 2, 4, 8, 8, 4, 2, 1] * 512 (56M参数)											✓				
测试减少1层: [1, 2, 4, 4, 2, 1] * 512 (10M参数)												✓			
测试context =30													✓		
测试context =40														✓	
epoch=1精度	63.6	58.2	67.5	68.4	53.9	60.9	67.1	69.2	65.1	66.9	66.7	68.3	67.3	66.2	
epoch=5精度	72.5	69.4	75.3	75.9	63.1	70.9	75.1	75.37	69.97	74.3	74.6	75.5	75.5	75.1	
epoch=20精度	72.1	75.0	79.7	79.5	70.1	76.5	79.7	75.44	72.1	78.9	79.7	79.4	80.07	80.08	

一些结论:

- 应用dropout刚开始收敛慢点，但最终对精度提升很好。
- 测试: BN → Tanh激活:** 全连接层出来的值，上下限非常大，进入Tanh时，几乎很少会落在(-2, 2)区间，这就导致从tanh出来的值几乎都是-1或1，没什么区别，再进入BN层也没什么用。而反过来，如果先进BN归一化一下，值基本在0到1之间，再进Tanh，就能差异化了。
- 测试dropout 放 BN 前面:** BN层可以和dropout一起用，但BN层必须在dropout之前！具体见：
<https://zhuanlan.zhihu.com/p/61725100>
- 测试增加2层网络:** 小数据集：参数量不必过大。

5.2 其他实验

基线：

- [1, 2, 4, 8, 4, 2, 1] * 512 + 输出层 (总参数量22M)
- 线性层→BN→GELU激励函数→dropout 0.2
- context = 30
- 优化器: Adam (初始学习率1e-3)
- 学习率调度器: CosineAnnealingLR (optimizer, T_max = 20, eta_min = 1e-4)

下文备注：

- SWATS：如果验证集损失在连续 3 个epoch没有改善，则将优化器Adam切换为SGD。

2. Rdrop: <https://lonepatient.top/2021/07/20/R-Drop-Regularized-Dropout-for-Neural-Networks>

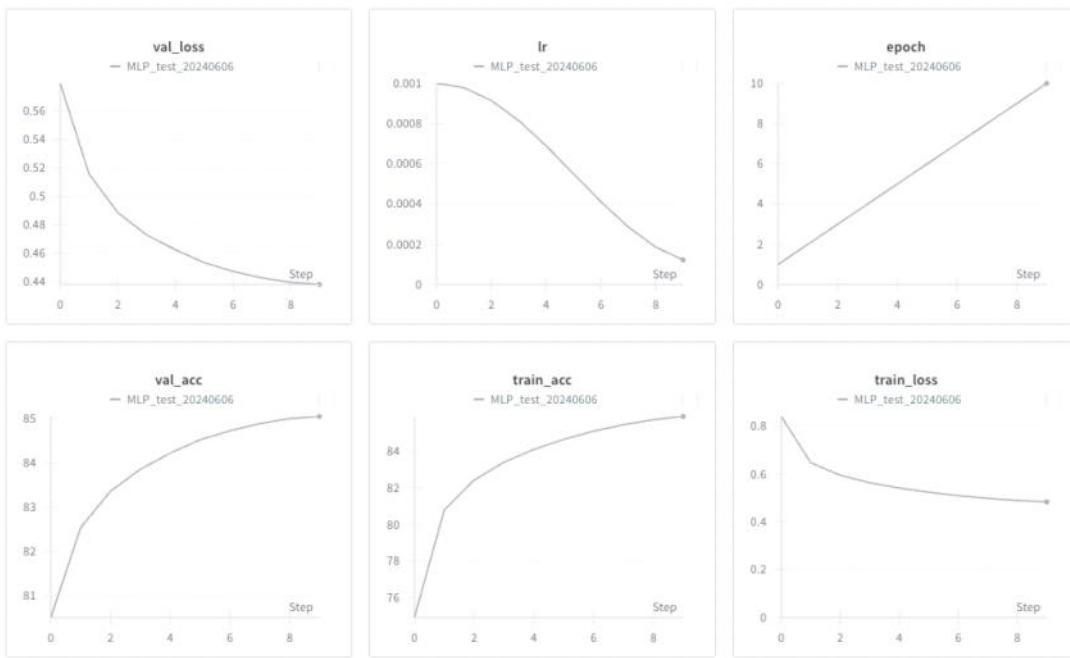
	epoch 1	epoch 5	epoch 20	结论
基线	67.3	75.5	80.07	
基线 + 数据标准化 (MFCC数据减均值除以方差) (train和val必须一起加!)	67.7	76.1	80.30	复习课-lab2中提到建议音频识别任务中使用。
基线 + 测试dropout掩码：所有特征点，独立5%几率置为0.	64.9	73.7	78.5	
基线 + 测试时间掩码：10%的随机位置帧置为0. (train和val都加mask)	63.3	57.6		第10个epoch=42.4。val集不能加mask!
基线 + 测试时间掩码：10%的随机位置帧置为0. (仅train)	64.8	71.79	71.78	
基线 + 测试频率掩码：10%的随机行的特征置为0. (仅train)	66.2	73.1	74.9	
基线 + 测试时间+频率掩码：各5%。 (仅train)	65.1	72.4	73.8	
基线 + 测试SGD优化器 (momentum=0.9)	37.5	51.6	58.5	
基线 + 测试batchsize = 32768 (提高4倍)	61.1	72.7	78.97	
基线 + 测试batchsize = 2048 (缩小4倍)	68.4	76.2	80.33	batchsize 太大和太低似乎都不好
基线 + 测试batchsize = 2048 + 数据标准化	69.2	76.8	80.62	
基线 + 测试batchsize = 2048 + 数据标准化 + SWATS (第17epoch损失值不下降，学习率1.8e-4，第18epoch开始，将Adam切换为SGD优化器，学习率重置为1e-3)	69.2	76.8	80.52	此类任务和模型，SWATS收敛好像不大，或者训练轮次太少，SGD的学习率还没来得及降低。
基线 + 测试batchsize = 2048 + 数据标准化 + Rdrop ($\alpha=0.5$) (原论文 $\alpha=1\sim10$ ；本项目第一个step，预测loss=3.7左右，kl_loss=0.13左右，最终loss=3.7+ $\alpha*0.13$)	69.7	77.6	81.32 loss刚停滞	
基线 + 测试batchsize = 2048 + 数据标准化 + Rdrop ($\alpha=0$) (目的是检查两次前向传播统计一次loss这种训练方式对精度影响精度多大，因为本质上20个训练epoch却遍历了40次数据集)	70.2	77.7	80.84	
基线 + 测试batchsize = 2048 + 数据标准化 + Rdrop ($\alpha=5$)	63.1	71.7	76.55	
基线 + 测试batchsize = 2048 + 数据标准化 + Rdrop ($\alpha=2$)	67.5	75.6	79.99	
基线 + 测试batchsize = 2048 + 数据标准化 + Rdrop ($\alpha=1$)	68.9	76.94	81.08	

5.3 全数据集实验

1. [1, 2, 4, 8, 4, 2, 1] * 512 + 输出层 (总参数量22M)
2. 模型结构：线性层→BN→GELU激励函数→dropout 0.2

3. context = 30
4. 优化器: Adam (初始学习率1e-3)
5. 学习率调度器: CosineAnnealingLR (optimizer, T_max = 10, eta_min = 1e-4)
6. Rdrop=0.5
7. 训练epoch: 10

Layer	Kernel Shape	Output Shape	Params	Mult-Adds
0_model.Linear_0	[1708, 512]	[2048, 512]	875.008k	874.496k
1_model.BatchNormId_1	[512]	[2048, 512]	1.024k	512.0
2_model.GELU_2	-	[2048, 512]	-	-
3_model.Dropout_3	-	[2048, 512]	-	-
4_model.Linear_4	[512, 1024]	[2048, 1024]	525.312k	524.288k
5_model.BatchNormId_5	[1024]	[2048, 1024]	2.048k	1.024k
6_model.GELU_6	-	[2048, 1024]	-	-
7_model.Dropout_7	-	[2048, 1024]	-	-
8_model.Linear_8	[1024, 2048]	[2048, 2048]	2.0992M	2.097152M
9_model.BatchNormId_9	[2048]	[2048, 2048]	4.096k	2.048k
10_model.GELU_10	-	[2048, 2048]	-	-
11_model.Dropout_11	-	[2048, 2048]	-	-
12_model.Linear_12	[2048, 4096]	[2048, 4096]	8.392704M	8.388608M
13_model.BatchNormId_13	[4096]	[2048, 4096]	8.192k	4.096k
14_model.GELU_14	-	[2048, 4096]	-	-
15_model.Dropout_15	-	[2048, 4096]	-	-
16_model.Linear_16	[4096, 2048]	[2048, 2048]	8.390656M	8.388608M
17_model.BatchNormId_17	[2048]	[2048, 2048]	4.096k	2.048k
18_model.GELU_18	-	[2048, 2048]	-	-
19_model.Dropout_19	-	[2048, 2048]	-	-
20_model.Linear_20	[2048, 1024]	[2048, 1024]	2.098176M	2.097152M
21_model.BatchNormId_21	[1024]	[2048, 1024]	2.048k	1.024k
22_model.GELU_22	-	[2048, 1024]	-	-
23_model.Dropout_23	-	[2048, 1024]	-	-
24_model.Linear_24	[1024, 512]	[2048, 512]	524.8k	524.288k
25_model.BatchNormId_25	[512]	[2048, 512]	1.024k	512.0
26_model.GELU_26	-	[2048, 512]	-	-
27_model.Dropout_27	-	[2048, 512]	-	-
28_model.Linear_28	[512, 40]	[2048, 40]	20.52k	20.48k
<hr/>				
Totals				
Total params	22.948904M			
Trainable params	22.948904M			
Non-trainable params	0.0			
Mult-Adds	22.926336M			
<hr/>				
Epoch 1/10				
Train Acc:	74.9363%	Train Loss:	0.8411	Learning Rate: 0.0010000
Val Acc:	80.5127%	Val Loss:	0.5797	
Validation accuracy improved from 0.0000% to 80.5127%. Saving model...				
Epoch 2/10				
Train Acc:	80.7835%	Train Loss:	0.6472	Learning Rate: 0.0009780
Val Acc:	82.5387%	Val Loss:	0.5158	
Validation accuracy improved from 80.5127% to 82.5387%. Saving model...				
Epoch 3/10				
Train Acc:	82.4145%	Train Loss:	0.5945	Learning Rate: 0.0009141
Val Acc:	83.3644%	Val Loss:	0.4886	
Validation accuracy improved from 82.5387% to 83.3644%. Saving model...				
Epoch 4/10				
Train Acc:	83.3937%	Train Loss:	0.5635	Learning Rate: 0.0008145
Val Acc:	83.8563%	Val Loss:	0.4730	
Validation accuracy improved from 83.3644% to 83.8563%. Saving model...				
Epoch 5/10				
Train Acc:	84.0883%	Train Loss:	0.5412	Learning Rate: 0.0006891
Val Acc:	84.2209%	Val Loss:	0.4625	
Validation accuracy improved from 83.8563% to 84.2209%. Saving model...				
Epoch 6/10				
Train Acc:	84.6325%	Train Loss:	0.5237	Learning Rate: 0.0005500
Val Acc:	84.5230%	Val Loss:	0.4534	
Validation accuracy improved from 84.2209% to 84.5230%. Saving model...				
Epoch 7/10				
Train Acc:	85.0812%	Train Loss:	0.5094	Learning Rate: 0.0004109
Val Acc:	84.7261%	Val Loss:	0.4473	
Validation accuracy improved from 84.5230% to 84.7261%. Saving model...				
Epoch 8/10				
Train Acc:	85.4319%	Train Loss:	0.4980	Learning Rate: 0.0002855
Val Acc:	84.8882%	Val Loss:	0.4427	
Validation accuracy improved from 84.7261% to 84.8882%. Saving model...				
Epoch 9/10				
Train Acc:	85.7036%	Train Loss:	0.4894	Learning Rate: 0.0001859
Val Acc:	85.0011%	Val Loss:	0.4395	
Validation accuracy improved from 84.8882% to 85.0011%. Saving model...				
Epoch 10/10				
Train Acc:	85.8917%	Train Loss:	0.4835	Learning Rate: 0.0001220
Val Acc:	85.0528%	Val Loss:	0.4380	
Validation accuracy improved from 85.0011% to 85.0528%. Saving model...				



6 附GPT回答：

6.1 关于前后填充帧数的选择

- 对于音素识别任务，考虑到一帧MFCC表示的是25毫秒的音频数据，选择合适的前后填充帧数（上下文窗口）是关键，这可以帮助模型更好地理解音素与其周围环境的关系。选择多少邻近帧作为填充，主要取决于以下因素：

1. 音素的平均持续时间

如前所述，英语中音素的持续时间差异较大，从大约50毫秒到200毫秒不等。为了确保上下文包含足够的信息来识别每个音素，需要考虑足够长的时间窗口来覆盖至少几个音素的平均持续时间。

2. 模型的处理能力

更大的上下文窗口意味着模型需要处理更多的数据，这会增加计算负担。因此，必须在计算效率和上下文信息的充分性之间找到平衡。

3. 实验和实践中的指导

通常，通过实验来确定最佳的上下文窗口大小。开始可以使用较小的窗口，然后逐渐增加，观察模型性能的变化。

建议的填充帧数

- **基于音素平均持续时间：**由于大部分音素的发音时间在100-200毫秒之间，至少需要覆盖这个长度的时间。考虑到每帧为25毫秒，大约4-8帧可以覆盖一个较长音素的平均持续时间。
- **考虑到上下文的影响：**为了确保能够捕获足够的前后信息，可以考虑在每侧增加约10-20帧作为上下文。这样的设置可以使每个分析窗口覆盖大约 $25\text{ms} \times (1 + 2 \times 20) = 1025\text{ms}$ 。

结论

对于您的设置，考虑到MFCC的每帧为25毫秒，推荐您尝试前后各填充10至20帧，这将为每个音素提供足够的上下文来进行有效的音素识别。具体数字可以根据您模型的性能和计算资源进行调整。通常，最佳的填充帧数需要通过交叉验证和模型性能测试来确定。

6.2 关于mask方法的建议

- ⑤ 关于音频处理特别是使用 MFCC 特征进行音素识别时，如何实施特征维度的掩蔽（masking）是一个值得考虑的问题。选择最佳的掩蔽策略取决于希望模型学到什么样的信息以及数据的特性。下面我将解释两种方法的优劣，并给出推荐。

1. 全行特征掩蔽 (Channel Masking)

这种方法涉及选择一定数量的频率特征 (MFCC 系数)，然后在所有时间帧中将这些特征完全掩蔽 (置零)。

优点:

- **强化模型学习:** 通过全行掩蔽，模型被迫学习如何不依赖被掩蔽的特征，增强了模型对于特征丢失的鲁棒性。
- **简单易实施:** 这种方法易于编程实现，且易于复制和验证实验结果。

缺点:

- **可能过度扰乱数据:** 掩蔽整个特征可能导致重要信息的丢失，尤其是如果掩蔽的特征对于分类任务至关重要。
- **减少模型灵活性:** 这可能限制了模型利用所有可能的信息来做决策的能力。

2. Dropout式特征掩蔽 (Random Dropout)

这种方法涉及在所有帧上对每个特征进行随机掩蔽，相似于神经网络中的dropout 技术。

优点:

- **提高泛化能力:** 随机掩蔽有助于防止模型对特定的特征过拟合，提升模型在未见过的数据上的表现。
- **模拟现实世界干扰:** 在真实世界的应用中，信号可能会受到各种随机噪声的影响，随机掩蔽模拟了这种情况。

缺点:

- **随机性导致的不稳定性:** 每次训练的结果可能因为掩蔽的随机性而有所不同，导致模型表现出波动。
- **可能掩蔽过多或过少:** 难以控制掩蔽的总量，可能在某些训练批次中掩蔽太多或太少的信息。

推荐策略

对于音素识别任务，建议采用 **Dropout式特征掩蔽**。这种方法更能模拟真实世界中的不确定性，并且通常更适合处理复杂的音频信号。Dropout式掩蔽也可以帮助模型更好地泛化到新的、未见过的数据，尤其是在音素间的差异可能较小的情况下。