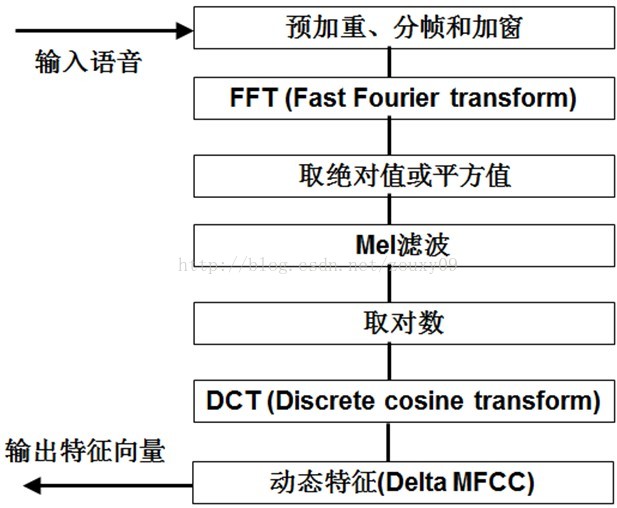
1. 声音特征提取

1.1 常用声音特征

1.1.1 MFCC

MFCC（Mel-frequency cepstral coefficients）:梅尔频率倒谱系数。梅尔频率是基于人耳听觉特性提出来的， 它与Hz频率成非线性对应关系。梅尔频率倒谱系数(MFCC)则是利用它们之间的这种关系，计算得到的Hz频谱特征。主要用于语音数据特征提取和降低运算维度。对fbank做离散余弦变换（DCT）即可获得mfcc特征。

MFCC特征提取流程如下：



1.1.2 Fbank

Fbank（FilterBank）：人耳对声音频谱的响应是非线性的，Fbank就是一种前端处理算法，以类似于人耳的方式对音频进行处理，可以提高语音识别的性能。获得语音信号的fbank特征的一般步骤是：**预加重、分帧、加窗、短时傅里叶变换（STFT）、mel滤波等**。

1.2 声音特征提取

1.2.1 声音特征提取流程

语音参数提取特征的步骤：预增强->分帧->加窗->添加噪声->FFT->Mel滤波->对数运算->DCT。其中 FFT 和 DCT 是快速傅里叶变换和离散余弦变换。

1.2.1 传统方式

1.2.2 基于深度学习方式

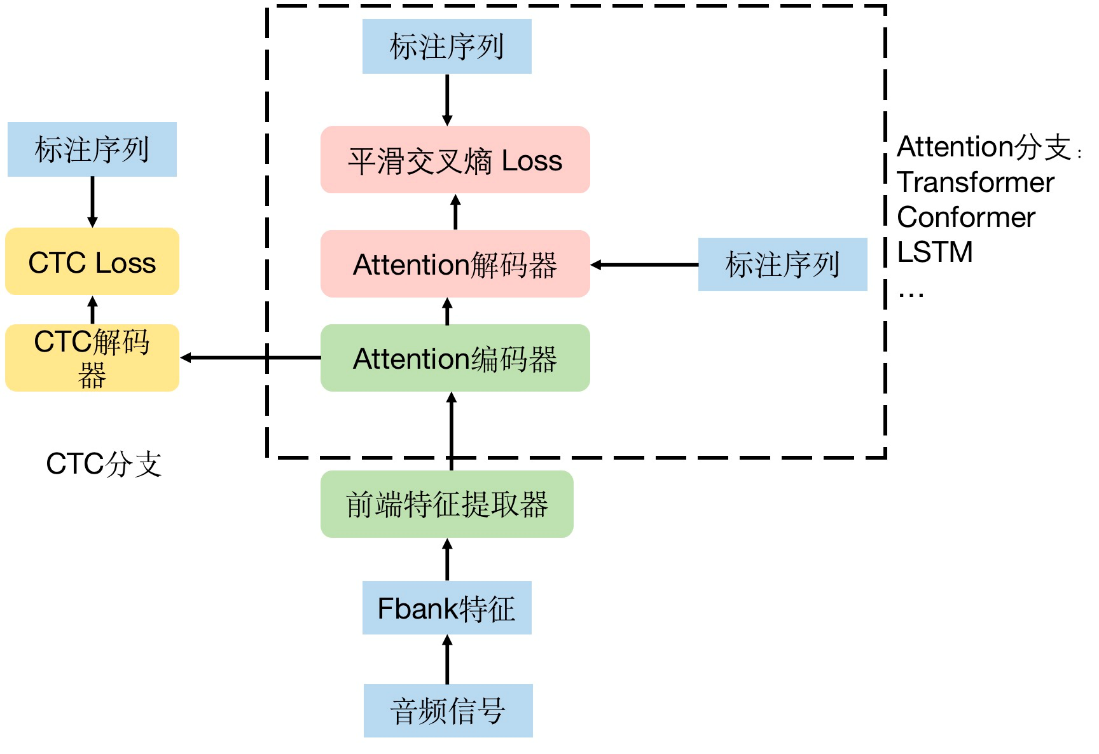
基于深度学习的声音特征提取

2. 基于深度学习的ASR

ASR模型可分为

2.1 端到端的ASR策略

端到端语音识别是区别传统语音识别的一种框架，并逐渐成为一种趋势。**传统语音识别一般分为声学模型与语言模型**，声学模型负责将音频序列转化为音素序列，常见的音素比如汉语拼音、英文音标等，语言模型则负责将这些音素序列转化成文字序列。声学模型和语言模型在训练时并不需要耦合，可以独立训练，传统语音模型的劣势在于需要有发音字典，需要有音素的标注。**端到端模型**就是克服了这一点，**直接将音频序列转化为文字序列**。

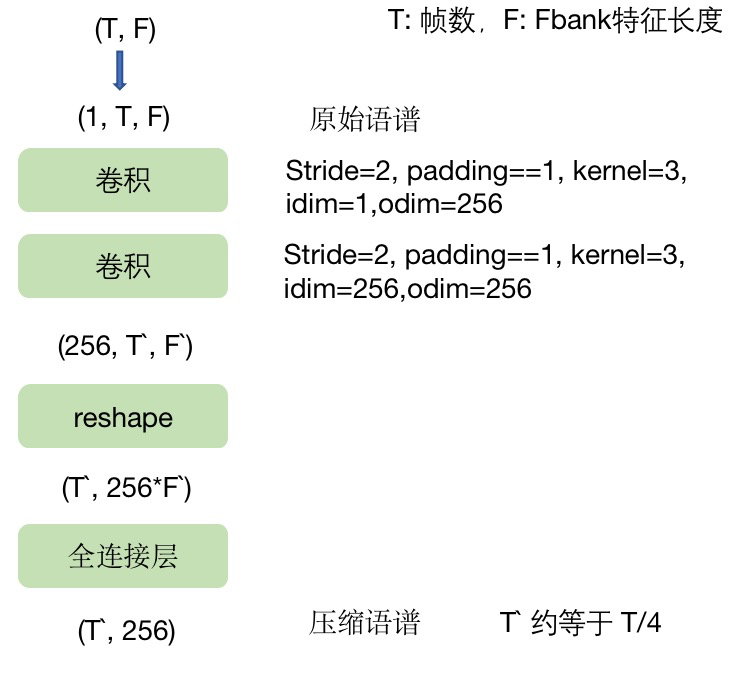


一般的端到端语音模型结构中，大致可以分成三个部分，分别是前端特征提取器、CTC分支和Attention分支：

1. 前端特征提取器：一般是若干层卷积，其目的是在时间维度上降采样，从而减少计算开销。
2. Attention分支：包含attention机制的一系列模型，比如Transformer、Comformer，甚至之前的LSTM，可以任意选择。通常用平滑交叉熵损失约束。
3. CTC分支：其结构上通常是一个全连接层，将encoder的特征转化为每个字的概率，用CTC损失约束。

如果要实现语音识别的基本功能，这三个结构并不都是必须的，比如前端特征提取器，但是它可以减少计算开销，很多框架和开源代码中都会有这一步。**其次CTC分支和Attention实际上有其中一条就可以实现语音识别的基本功能**。但是有实验表明这两条支路的两种损失一起训练模型可以得到更好的效果。需要注意的是，**尽管训练时两条支路都会用到，但是推理时一般只会用到其中一条**支路。对于Attention分支，主要基于Transformer、LSTM这类模型进行实现。

2.1.1 前端特征提取器



前端特征提取器常见的操作是对语谱特征Fbank做两次stride=2的卷积，同时将通道数扩展到256，再进行reshape和全连接层映射，将特征长度统一为256。这也是后面Attention分支里所采用的特征长度。经过这么一通计算，时间轴的长度约为原来的1/4，这样可以降低后面attention的计算开销。

2.1.2 CTC分支

CTC分支本身可以理解为是一个全连接层，它以Attention编码器的输出特征作为输入，通过一个全连接层将256维的特征映射成字典大小的向量，再进行softmax归一化，得到每个字的预测概率。

但是这里的预测存在一个问题，假设提取Fbank特征时的步长是10ms，那么1秒的音频，其语谱特征就大约100帧，经过前端特征提取器进行4倍降采样后，大约是25帧。因为attention编码器的输入和输出维度保持一致，都是25\*256，这意味着1s的音频，CTC可能会解码出25个字，这显然是不合理的。

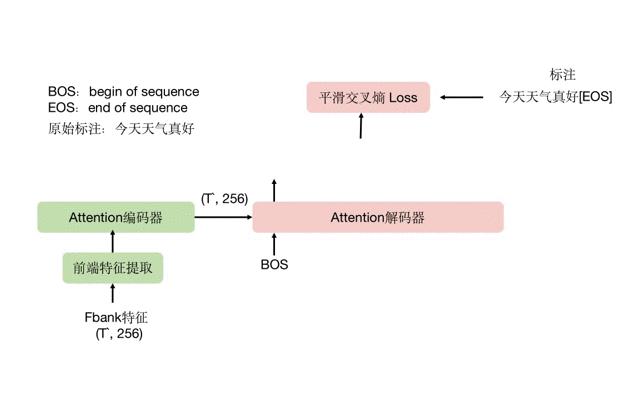
2.1.3 CTC预测结果处理

在带CTC分支的语音识别框架中，一般会在字典中加入一个特殊的字符——空，下面用Ԑ来表示空字符。人在说话的时候，每句话之间是有空隙的，并且每个人说话的语速不同，每个字之间有不同的空隙，引入空字符Ԑ之后，实现了模型对这类空隙的建模。这时候联想回前面说的1s似乎要输出25字，问题就能得到一定程度的解决，大部分时候是可以输出空字符的，并不需要输出有意义的字。这时候再来思考一个问题，我们说话1s说不了几个字，但是1s对应25帧，那么每个字的发音都会占据很多帧，可能会连续几帧都识别成一个字，那怎么办？这就引入CTC分支解码时的规整方法（有些地方也称之映射）：

**（1）删除所有Ԑ字符**

**（2）合并连续出现的相同字符**

2.1.4 Attension分支



这里以Transformer类的模型为例，解码器以文本作为输入，同时也会接收编码器输出的特征，进行cross-attention操作。先简单说下推理时，attention解码的方式，一开始我们给解码器送入一个BOS起始符号，希望解码器结合编码器送过来的音频编码特征，预测出这段音频的第一个字“今”，模型成功预测出“今”字，将它重新放回解码器的输入，希望解码器预测出下一个字“天”，一直这样循环下去，直到最终预测出句子终止符号EOS为止。这套过程也称为自回归解码，是attention解码的一大特点，它的好处是能更好地结合已经预测的上文文本，去推测下文信息，句子的通顺程度和字错率通常都更好。​

但是如果是训练，则可能出现一种情况，如果解码器输出的字是错的，比如“天气”的“气”预测成了“器”，那么在预测下一个字时，放到解码器的输入层是“器”还是“气”呢？尤其是模型刚开始训练时，可以预见几乎每个字都是错，以错字为已知信息放到解码器输入，岂不是错上加错？这就涉及两种不同的训练思路：

* 用“气”。也就是甭管解码器预测的上一个字是什么，只从标注中拿到正确的那个字放到下一个时刻的输入。这种方法叫做Teacher-Forcing。
* 用“器”。就是模型认为上一个时刻是什么字，就把这个字作为下一个时刻的解码器输入，这种方法叫做Schedule-Sampling。

这两种方法各有优势，Teacher-Forcing的优势有两点：

* 减少了误差累计，模型训练起来更容易收敛
* ​如果Attention分支采用的是Transformer这类允许序列并行计算的模型，由于在训练时不关心上一时刻输出什么，所以完全可以一次性将一整句话送入解码器，即[BOS，今，天，天，气，真，好]，而对应的标注应该是[今，天，天，气，真，好，EOS]。这就可以实现并行计算，而避免了一个字一个字往外崩的串行计算了。​

如果你对Transformer不是很熟，可能会有疑问，一整句话作为输入，那么在预测第i个字时，岂不是已经可以看到第i+1个字是什么呢？甚至i后的所有字都可以看到。其实是在transformer在进行attention操作时，会通过mask盖住后面的信息，这样就实现了训练与推理时的一致性。

但是Teacher-Forcing也有一个弊端，就是他没有模拟实际推理时会出现的上文出现了错字，下文要如何预测的问题，因此训练和推理仍然存在不一致的问题。

至于Schedule-Sampling，很明显它和推理是一致的，它可以很好地模拟推理时出现错字的情况，因此具有更好的鲁棒性。但是其缺点也是很明显的，就是更难训练，收敛更慢，并且由于只能串行计算所以效率不高。

有人也提出了一种折衷的方法，以结合两者的优点，就是在训练时以一定概率选择要用“气”还是“器”，即用真实标注还是解码器预测的上一时刻输出。这个概率要怎么定呢？基本的原则就是在训练初期，以更高的概率选择真实标注，等到模型具备一定的预测能力，再以更高的概率去选择解码器上一时刻的输出，即使该输出是错的。具体可采用的衰减策略比如线性衰减、指数衰减等。

3.声音信号处理手段

3.1 3A算法

3A 音频处理技术，是**声学回声消除（AEC）、背景噪声抑制（ANS）、自动增益控制（AGC）**三种音频算法的合称。一般对语音信号的处理大都需要经过三个步骤。C++程序中可以使用**speexdsp**调用3A算法来对声音信号进行处理。

4. 声音信号传输

4.1 编解码

**(1) 编码**

音频编码的基本原理是利用声学特性和听觉心理学原理对音频信号进行压缩。具体过程包括以下几个步骤：

* 采样：将连续的模拟音频信号转换为离散的数字信号，即进行采样操作。
* 量化：将采样后的信号进行量化，将连续的信号值映射为离散的取值，以减小数据量。
* 压缩编码：对量化后的信号进行编码，将信号表示为更紧凑的形式。编码方法包括预测编码、变换编码和熵编码等。
* 打包：将编码后的数据组织为数据包，便于传输和存储。

**(2) 解码**

音频解码的过程则是编码过程的逆过程，包括解包、解码、反量化和重构等步骤。解码后的信号经过数字模拟转换，即可恢复为模拟音频信号，下面是音频解码的主要步骤和原理：

* 解包（Demultiplexing）：音频解码器首先需要对编码后的音频数据进行解包。编码过程中，音频数据可能被分割成多个包或帧，并与其他相关的信息（如采样率、比特率、声道数等）一起打包在一起。解包阶段的任务是从数据流中提取出音频数据和相关信息。
* 解码（Decoding）：解码是音频解码的核心步骤。在这一步骤中，音频编码器使用特定的解码算法将压缩的音频数据转换回原始的数字音频信号。解码算法根据编码过程中使用的压缩算法和编码参数，对压缩后的数据进行逆操作，还原出原始的频域或时域表示。
* 反量化（Dequantization）：在编码过程中，音频信号的频域或时域表示经过了量化操作，即将连续的信号值转换为离散的数值。在解码阶段，需要对量化后的数据进行反量化操作，将离散的数值恢复为连续的近似值。反量化过程使用与编码过程相反的量化表或算法，将离散的数据重新映射到原始的数值范围内。
* 重构（Synthesis）：重构是将解码和反量化后的数据转换为模拟音频信号的过程。根据音频编码的特性和算法，重构阶段可能涉及滤波、插值和合成等操作。这些操作的目标是将解码后的数据转换为与原始音频尽可能接近的模拟信号。
* 数字模拟转换（Digital-to-Analog Conversion）：解码和重构后的音频数据仍然是数字形式的，需要进行数字模拟转换（DAC）以将其转换为模拟音频信号。DAC将离散的数字样本转换为连续的模拟信号，以便在扬声器或耳机等音频播放设备上播放。

通过以上步骤，音频解码器能够逆向操作，从压缩的音频数据中还原出高质量的模拟音频信号。解码的过程需要根据所采用的音频编码标准和算法进行相应的解析和计算，以确保解码后的音频质量和准确性。

4.2 常用编解码算法

**(1) MP3**

这种压缩方式的全称叫MPEG Audio Layer3,MP3是利用MPEG Audio Layer 3的技术，将音乐以1:10甚至1:12 的压缩率，压缩成容量较小的file，换句话说，能够在音质丢失很小的情况下把文件压缩到更小的程度。>而且还非常好的保持了原来的音质。 \* 压缩率：10~12倍 \* 优点：压缩比高，适合用于互联网上的传播 \* 缺点： MP3 在 128KBitrate 及以下时，会出现明显的高频丢失

**(2) AAC高级音频编码**

Advanced Audio Coding。一种专为声音数据设计的文件压缩格式，与MP3不同，它采用了全新的算法进行编码，更加高效，具有更高的“性价比”。利用AAC格式，可使人感觉声音质量没有明显降低aac标志的前提下，更加小巧。AAC属于有损压缩的格式，与时下流行的APE、FLAC等无损格式相比音质存在“本质上”的差距。加之，传输速度更快的USB3.0和16G以上大容量MP3正在加速普及，也使得AAC头上“小巧”的光环不复存在了。

优点：支持多种音频声道组合，提供优质的音质。

**(3) WMA**

WMA的全称是Windows Media Audio，是微软力推的一种音频格式。WMA格式是以减少数据流量但保持音质的方法来达到更高的压缩率目的，其压缩率一般可以达到1:18，生成的文件大小只有相应MP3文件的一半。

压缩率：10~12倍

缺点：在高比率的渲染能力低下，同音源的一个320KBPS的MP3与比较192KBPS的WMA相比，音质和渲染力很容易分别出是前者较优。因为：

当 Bitrate 小于 128K 时， WMA 最为出色且编码后得到的音频文件很小。

当 Bitrate 大于 128K 时， WMA 音质损失过大。

优点：WMA还可以通过DRM（Digital Rights Management）方案加入防止拷贝，或者加入限制播放时间和播放次数，甚至是播放机器的限制，可有力地防止盗版。

**(4) WAV**

WAV是录音时用的标准的windows文件格式，文件的扩展名为“.wav”，WAVE文件作为最经典的Windows多媒体音频格式，应用非常广泛。声道有单声道和立体声之分，采样频率一般有11kHz、22kHz和44kHz三种。 WAVE文件所占容量=（采样频率×采样位数×声道）×时间/8（1字节=8bit）。

**(5) OGG**

OGG格式的全称应该是OGG Vobis。它是一种新的音频压缩格式，类似于MP3等现有的音乐格式。但有一点不同的是，它是完全免费、开放和没有专利限制的。OGG Vobis有一个很出众的特点，就是支持多声道， OGG Vobis在压缩技术上比MP3好，而且它的多声道，免费，开源这些特点，使它很有可能成为一个流行的趋势，这也正是一些MP3播放器对其支持的原因

可以对所有的声道进行编码，而不是MP3只能编码2个声道。多声道音乐的兴起，给音乐欣赏带来了革命性的变化，尤其在欣赏交响时，会带来更多临场感。这场革命性的变化是MP3无法适应的。在以后的播放技术>不断提高以后，而且人们对音质要求不断提高，Ogg的优势将更加明显。

优点：完全免费。开放没有专利限制。支持多声道

**(6) APE**

APE的本质，其实它是一种无损压缩音频格式。庞大的WAV音频文件可以通过Monkey”s Audio这个软件进行“瘦身”压缩为APE。有时候它被用做网络音频文件传输，因为被压缩后的APE文件容量要比WAV源文件小一半多，可以节约传输所用的时间。更重要的是，通过Monkey”s Audio解压缩还原以后得到的WAV文件可以做到与压缩前的源文件完全一致,.。所以APE被誉为“无损音频压缩格式”

4.2.1 Opus

**(1) OPUS编码特性**

OPUS被设计为低延迟的音频编解码器，适用于实时通信等对延迟要求较高的应用。OPUS的延迟主要由帧长和编码算法决定，较短的帧长可以减少延迟，但会导致较高的比特率。在VOIP模式下，OPUS可以实现非常低的端到端延迟，通常在20毫秒到60毫秒之间。

* 容错性：OPUS具有一定的容错性，可以在网络丢包或丢失部分数据的情况下仍能提供较好的音频质量。OPUS使用了纠错编码和前向纠错技术，通过重采样、插值和隐藏丢失数据等方法来恢复丢失的音频数据。
* 平台支持：OPUS是一种跨平台的音频编解码格式，支持在多种操作系统和硬件平台上使用。它具有广泛的应用支持，包括桌面应用、移动应用、浏览器等，可以在不同设备和平台上进行音频编解码和传输。
* 音频质量：OPUS以其出色的音频质量而闻名。它具有广泛的音频频带宽范围（从超低频到高频），能够适应不同类型的音频内容，并提供高保真度的音频重现。
* 低延迟：OPUS被设计为低延迟的音频编解码器，适用于实时通信应用。它可以在保持较低延迟的同时提供高质量的音频传输。
* 高效的压缩：OPUS具有出色的压缩性能，可以在较低的比特率下提供高质量的音频。它采用了多种音频编码技术和自适应比特率控制，以提供最佳的音频编解码效率。
* 支持多通道音频：OPUS支持多通道音频编解码，包括立体声、5.1声道和7.1声道等。这使得OPUS适用于多通道音频应用，如音频录制、音频处理和游戏音频。
* 不同应用场景的配置参数：OPUS提供了一系列的配置参数，允许用户根据应用需求进行定制。这些参数包括帧大小、比特率、预测算法和复杂度等。通过调整这些参数，可以在音频质量、延迟和压缩率之间进行平衡。

总体而言，OPUS是一种灵活、高效和适用于多种应用场景的音频编解码格式。OPUS的编解码过程可以根据实际需求进行参数配置，以平衡音频质量、延迟和压缩率，它能够提供低延迟、高音质的音频传输和存储解决方案，广泛用于语音通信、音频会议、流媒体、语音识别和语音交互等领域。

1.3.2 OPUS为何能抗弱网

Opus编码之所以能够在弱网环境下表现出良好的抗丢包能力，主要有以下几个原因：

鲁棒性设计：Opus编码器在设计时就考虑到了网络的不稳定性。它内置了多种机制来处理网络中的丢包问题，如数据包丢失隐藏（PLC）技术。当网络中的数据包丢失时，Opus能够利用已接收的数据包来预测和重建丢失的数据，从而保持音频的连续性和可听性。

可变比特率（VBR）和帧大小：Opus支持可变比特率和帧大小，这意味着它可以根据网络条件实时调整编码参数。在弱网环境下，Opus可以降低比特率并减小帧大小，以减少数据包的大小和数量，从而降低网络拥塞和丢包的风险。

高效的编码算法：Opus采用了先进的编码算法，能够在保证音频质量的同时，尽可能减小编码后的数据量。这使得在相同的网络带宽下，Opus能够传输更多的音频数据，从而提高了音频的流畅性和清晰度。

灵活的带宽分配：Opus编码器能够动态地调整音频带宽，以适应网络带宽的变化。在弱网环境下，Opus可以减少音频带宽的使用，以确保音频数据的稳定传输。

支持前向纠错（FEC）：虽然Opus本身并不直接支持前向纠错，但它可以与其他支持FEC的传输协议（如RTP/RTCP）结合使用。FEC能够在传输过程中为数据包添加冗余信息，以便在接收端检测到丢包时能够恢复部分丢失的数据。这种结合使用的方式可以进一步提高Opus在弱网环境下的抗丢包能力。

综上所述，Opus编码之所以能够抗弱网，主要是因为其具备鲁棒性设计、可变比特率和帧大小、高效的编码算法、灵活的带宽分配以及支持前向纠错等特性。这些特性使得Opus能够在弱网环境下保持音频的连续性和可听性，为用户提供更好的音频体验。

1.3.2 Opus使用

可以基于libopus开源库实现原始音频数据的Opus编解码操作

4.3 抗弱网手段  
传输层：jitterBuffer（重组）、RTCP、自动重传

编码层：opus编解码：前向纠错（FEC）、丢包隐藏（PLC）、流量控制

参考文献

1. 一文入门端到端语音识别：<https://zhuanlan.zhihu.com/p/536647952>
2. 语音识别中的MFCC 与 Fbank特征的基本原理和python实现: <https://zhuanlan.zhihu.com/p/363153781>
3. 语音识别 FBank 和 MFCC 特征：https://murphypei.github.io/blog/2021/10/asr-fbank-mfcc