ASV-Subtools 声纹识别实战手册 V1.0

洪青阳
李琳
赵淼
周健峰
陆昊
李铮
江涛
童福川
廖德欣

厦门大学智能语音实验室
厦门天聪智能软件有限公司

首次发布 2022 年 11 月

目 录

第 1 章 声纹基础理论 4
1.1 应用任务 5
1.1.1 说话人确认 5
1.1.2 说话人辨认 6
1.1.3 说话人日志 6
1.1.4 反欺骗攻击 6
1.2 发展历程 7
1.3 系统框架 8
1.4 关键技术 10
1.5 评价指标 11
1.5.1 等错误率（EER） 11
1.5.2 minDCF 12
1.6 应用挑战 12
1.7 声纹数据库 13
1.8 开源工具 13
第 2 章 声纹核心算法 15
2.1 网络架构 15
2.1.1 TDNN 15
2.1.2 E-TDNN 16
2.1.3 F-TDNN 17
2.1.4 ResNet 19
2.1.5 ECAPA-TDNN 21
2.1.6 Conformer 22
2.2 损失函数 23
2.2.1 CE Loss 23
2.2.2 Margin Loss 23
2.3 后端分类器 24
2.3.1 LDA 24
2.3.2 Cosine 25
2.3.3 PLDA 26
2.4 学习机制 28
2.4.1 注意力学习 28
2.4.2 多任务学习 28
2.4.3 迁移学习 29
第 3 章 Subtools 工具介绍 30
3.1 工程结构 31
3.2 训练框架 35
第 4 章 环境配置 38
4.1 Kaldi 安装 38
4.2 Subtools 安装 39
4.3 环境配置 39

第 5 章 入门使用—Voxceleb1 训练和测试 43
5.1 数据介绍 43
5.2 数据下载 43
5.3 数据处理 44
5.4 特征提取 45
5.5 数据加噪 46
5.6 模型训练 47
5.7 模型测试 49
第 6 章 进阶研究—多任务/迁移学习 52
6.1 多任务学习 52
6.1.1 文本相关数据准备 53
6.1.2 文本相关基线模型 53
6.1.3 多任务模型训练 54
6.2 迁移学习 55
6.2.1 迁移模型训练 55
第 7 章 工程部署—Runtime 实现 58
7.1 JIT 导出模型 58
7.2 Runtime 实现 60
7.3 SDK 封装 62
参考文献 67

第 1 章 声纹基础理论

随着智能设备的日渐丰富，人机语音交互的场景越来越多，对基于生物特征的身份验证需求也与日俱增。人体生物特征包括指纹、人脸、声纹、掌纹、虹膜、视网膜等，其中声纹（Voiceprint）用来表征语音中反映说话人生理和行为信息，每个人的声纹具有唯一性、独特性。声纹识别，也称作说话人识别，是一种通过声音判别说话人身份的技术，具有采集方便、非接触式等独特的优势。声纹识别与语音识别是不同的任务，虽然都是从语音信号提取信息，但语音识别关注的是说话内容，声纹识别则要判断是谁说的。

近几年来，声纹识别技术的发展迅速，在工业界也有越来越多的应用，产品如图 1.1 所示，其中智能手机在语音唤醒时结合声纹识别，确保是机主本人的操作，而智能音箱和智能电视则对家庭成员进行识别，提供个性化服务。

图 1.1 采用声纹识别的智能设备

“声纹”(Voiceprint)是指人的语音中所蕴含的、能表征说话人身份的生理和行为特征，每个人的声纹具有唯一性、独特性，因此与指纹、人脸、虹膜等生物特征一样可用于进行身份识别。与指纹、人脸、虹膜等生物特征相比，声纹具有非接触获取、采集成本低、便于远程认证的优点。

声纹识别技术涉及信号处理、概率统计、机器学习等领域知识，实际应用还包括公安司法、市场调查、军事国防等领域，虽然技术不断进步，但仍存在不少挑战，接下来简要介绍声纹识别应用任务、发展历程、系统框架、关键技术、评价指标、应用挑战、声纹数据库和开源工具。

1.1 应用任务

如图 1.2 所示，声纹识别的任务主要包括说话人确认（Speaker Verification）、说话人辨认（Speaker Identification），其中说话人确认是 1:1 比对、说话人辨认是 1:N 比对。声纹识别还延伸出其它应用任务，包括说话人日志（Speaker Diarizaiton）和反欺骗攻击（Anti-Spoofing）。

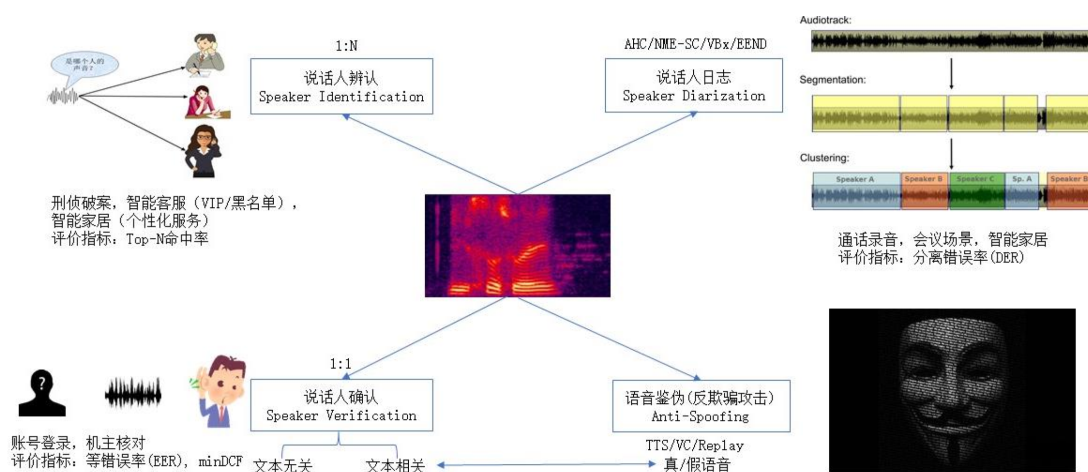


图 1.2 声纹识别应用任务

说话人确认主要用于账号登录、机主核对等应用，根据验证内容是否跟登记内容是否一致，可再分为文本无关和文本相关任务。说话人日志用来判断“谁在何时说话”，即进行说话人角色分离，在通话录音、会议场景、智能家居有迫切的应用。反欺骗攻击又称语音鉴伪，用来判断真假语音，防止 TTS 语音合成、VC 变声、录音冒充等欺骗攻击。

1.1.1 说话人确认

说话人确认是判断当前语音是否为某个已知的说话人对象模型的过程，可以认为是 1:1 的二元决策问题。在识别过程中将待测语音和给定说话人模型进行比对，若识别结果高于给定阈值，则认为是相同的说话人，反之则认为是不同说话人。根据识别内容与注册内容是否一致，可再分为文本无关和文本相关两大类型。说话人确认评价指标一般采用等错误率（EER），即错误接收率（FAR）和错误拒绝率（FRR）相等的点，其值越低越好。

1.1.2 说话人辨认

说话人辨认是判断当前语音是否属于数据库中已有的全部说话人对象模型，是 1:N 的多元决策问题。在辨认阶段，需要将待测语音与数据库中所有说话人模型进行对比，选定得分最高的说话人模型作为辨认结果。说话人辨认评价指标可采用 Top-N 命中率（其中 N 可为 1 或更大的整数），越高越好，但其性能会因为说话人模型的数量增多而下降。

1.1.3 说话人日志

说话人日志又称说话人分割聚类。在多人会话场景中，如通话录音、会议场景、智能家居等，一段录音存在多个说话人，更复杂的鸡尾酒会（Cocktail Party）场景，还存在更多的人声干扰。说话人日志从录音获取“谁在何时说话”的信息，把不同说话人分开，实现角色分离，既有利于整理录音，也对后续的语音识别、声纹识别等应用起着关键的作用。说话人日志评价指标主要采用分离错误率（DER），越低越好。

1.1.4 反欺骗攻击

反欺骗攻击又称语音鉴伪。通信技术和移动互联网给现代生活带来极大便利，用户可以随时随地互相沟通，远程交易也日益普及，但电信诈骗和金融诈骗也越来越多，严重影响老百姓的日常生活，急需监管机构或金融单位通过技术手段，检测欺骗攻击类型，并结合声纹识别系统，采取有效的反欺骗攻击（Anti-Spoofing）措施。欺骗攻击类型主要有三种：

1）变声（Voice Conversion，VC）：通过变声器等工具改变自己的声音，让用户区分不清，逃避监管机构追踪；

2）语音合成（Text-to-Speech, TTS）：采用先进的 TTS 技术，合成逼真语音，用于电话广告，骚扰用户；

3）录音冒充（Replay Attack）：采用高保真录音设备，进行录音回放，冒充目标人登录账号。

1.2 发展历程

自 20 世纪 30 年代起，就已经有学术界、工业界的研究者们对声纹识别进行研究。声纹（Voiceprint）的概念，在上个世纪 60 年代被提出，当时贝尔实验室通过对语音的语谱图进行研究，并区分出了不同的说话人，揭示了在现实场景下声纹识别技术的可行性。自此之后声纹识别技术经历了 60 多年的发展历程，如图 1.3 所示，主要包括特征工程、统计模型和深度学习三个阶段，每个阶段都有代表性的模型。

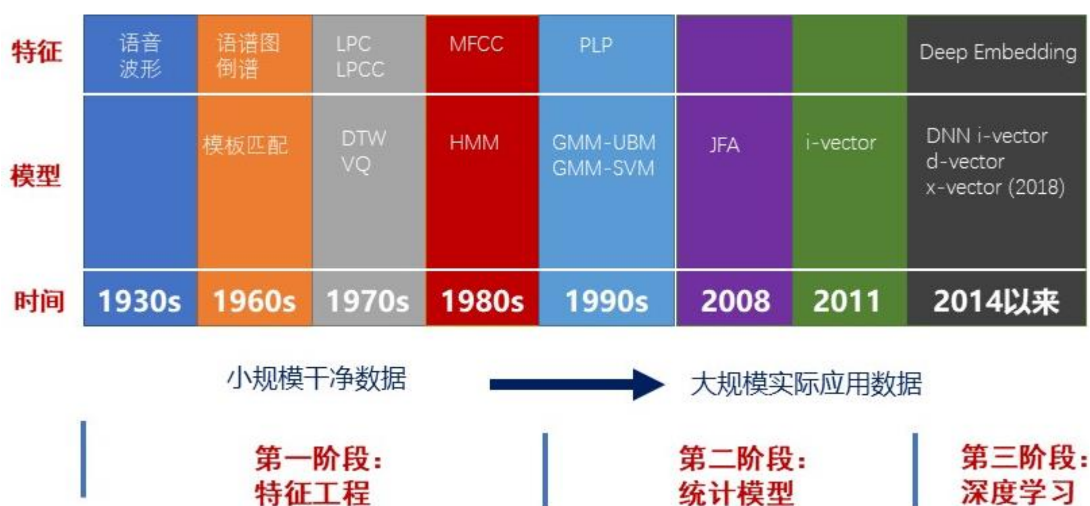


图 1.3 声纹识别发展历程

经典的声纹识别算法包括高斯混合模型-全局背景模型（GMM-UBM）[1]、联合因子分析（JFA）[3]和 i-vector[4]，这些算法各有优势，比如 i-vector 系统适用于长语音、跨信道任务的效果。但针对短语音，多组实验表明 i-vector 不如 GMM-UBM 系统，而 GMM-UBM 也只适合于安静、同信道的场合，无法应对复杂多变的应用场景。从 2014 年开始，深度学习开始成为声纹识别的研究热点，其主要思想是采用深度神经网络（Deep Neural Networks，DNN），从大量的说话人音频数据中，学习到说话人的本质特征并将其表征成说话人向量，用于后续说话人判别。如 DNN 可替代 GMM，得到更精细化建模的 DNN i-vector[7]。

2017 年，David Snyder 等人提出了 x-vector[8]框架。该框架首先将前端提取的特征送入到时延神经网络（Time-Delay Neural Networks, TDNN），再通过统计

层计算帧级别的特征向量的平均值和方差作为句子级别的特征向量，将可变长度的音频映射到固定维度特征向量。与因子分析方法的空间假设建模相比，x-vector 模型依靠 DNN 的学习能力，直接学习到能够区分说话人的向量表征（Embedding）。x-vector 框架的出现，标志着声纹识别领域正式进入 DNN 时代。

1.3 系统框架

声纹识别的系统框架如图 1.4 所示，包括注册和测试过程。语音提取声学特征（主要采用 FBank 和 MFCC）后，送入声纹模型，提取相应的说话人表征（i-vector 或 x-vector），然后再进行后端判别，一般采用余弦（Cosine）或概率线性区分分析（PLDA）[5]打分，必要的话还可做分数规整。

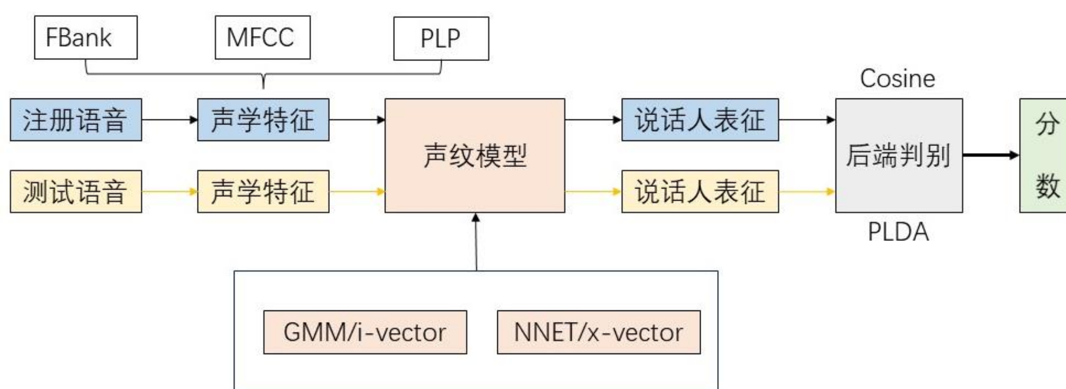


图 1.4 声纹识别系统框架

FBank 和 MFCC 特征提取过程如下图所示，具体包括预加重、分帧、加窗、Mel 滤波等步骤，其中 Mel 滤波器的设计模拟了人耳对不同频率的感知机制，在低频部分三角滤波器比较密集，而高频部分比较稀疏。FBank 直接采用对数功率输出，特征维度之间没有去相关，保留较多原始特征，比较适合神经网络的输入，而 MFCC 则进一步采用离散余弦变换（DCT）去相关和压缩维度，得到更紧凑的特征向量。

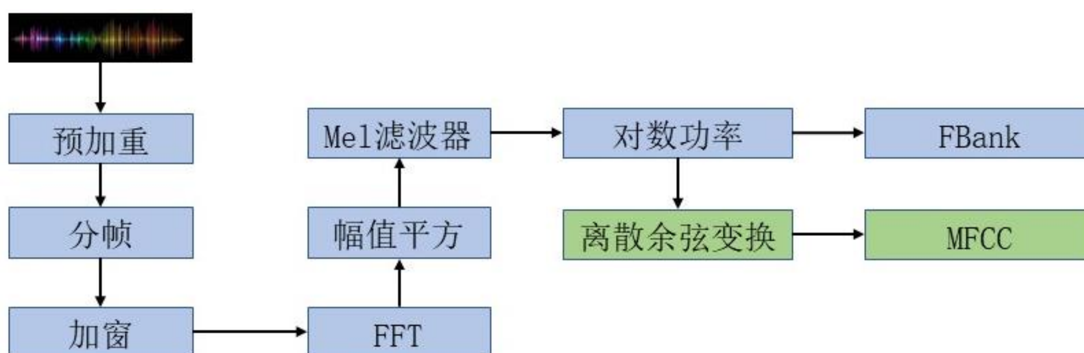


图 1.5 声学特征提取过程

声纹模型可采用基于 GMM 的 i-vector 或基于神经网络的 x-vector。i-vector 是基于单一空间的跨信道算法，该空间既包含了说话人空间的信息也包含了信道空间信息。对于给定的语音，高斯超向量表示如下：

$$M = m + Tw \tag{1-1}$$

其中，m是话者无关且信道无关的超向量，通常由 UBM（是一个 GMM）的均值向量拼接而成；T是一个低秩的矩阵；而w则是服从标准正态分布的随机向量，简称 i-vector。

i-vector 是生成式建模，泛化性较好，但对短语音（5 秒以内）性能不佳，一般建模语音要 40 秒以上，测试语音要 10 秒以上。

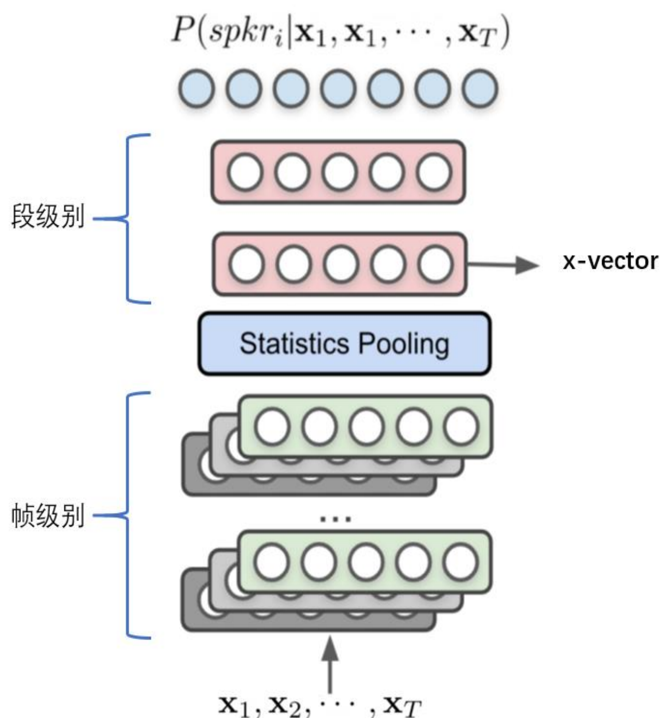


图 1.6 x-vector 模型

x-vector 的模型结构如图 1.6 所示，主要包含三部分：

- 帧级别：处理时序语音信息；
- Statistics Pooling：学习数据的全局统计信息；
- 段级别：进一步提取全局信息。

x-vector 网络输出节点对应说话人的 ID，是区分性建模，其优点是短语音性能较好，神经网络可以训练大数据，但泛化性较差，未见的说话人领域性能会下降很多。

1.4 关键技术

声纹识别以 x-vector 为主流框架，其关键技术涉及数据准备，损失函数，模型的架构和学习方法，以及后端分类器，如图 1.7 所示。通过数据加噪、模拟远场、Specaugment、Online 扩增，可有效提高声纹模型的鲁棒性。针对模型的网络结构、池化（pooling）方式、学习方法等多方面的改进，仍然是目前学术界的研究热点。由于采用 AM-Softmax、AAM-Softmax 等带 Margin 的损失函数，使得提取到的 x-vector 已具备较强的区分性，因此后端分类器已较多采用 Cosine 打分，其复杂度非常低，识别效率远超过 PLDA 打分。但针对跨领域的自适应，PLDA 更有优势。

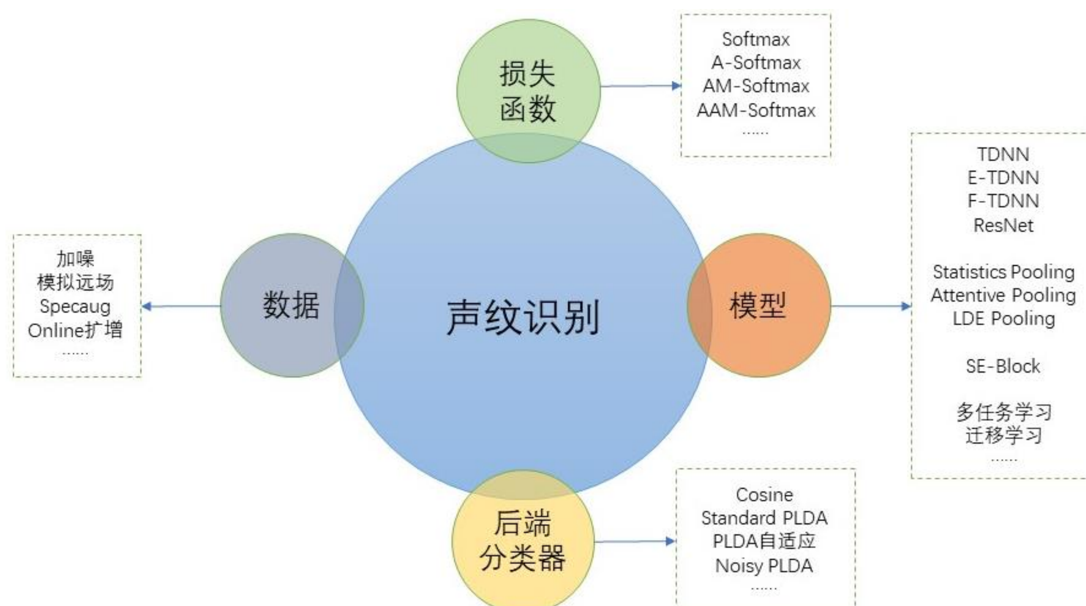


图 1.7 声纹识别关键技术

1.5 评价指标

1.5.1 等错误率（EER）

说话人确认有两个基本指标，分别是错误接受率（False Accepted Rate，FAR）和错误拒绝率（False Rejected Rate，FRR）：

$$FAR = \frac{Number\ of\ nontarget\ trials\ accepted}{Total\ nontarget\ trials} \tag{1-2}$$

$$FRR = \frac{Number\ of\ target\ trials\ rejected}{Total\ target\ trials} \tag{1-3}$$

当阈值的选定经过整个分数区间，以 FAR 和 FPR 分别为横纵坐标，即可得到检测错误均衡图（Detect Error Tradeoff，DET），如图 1.8 所示。

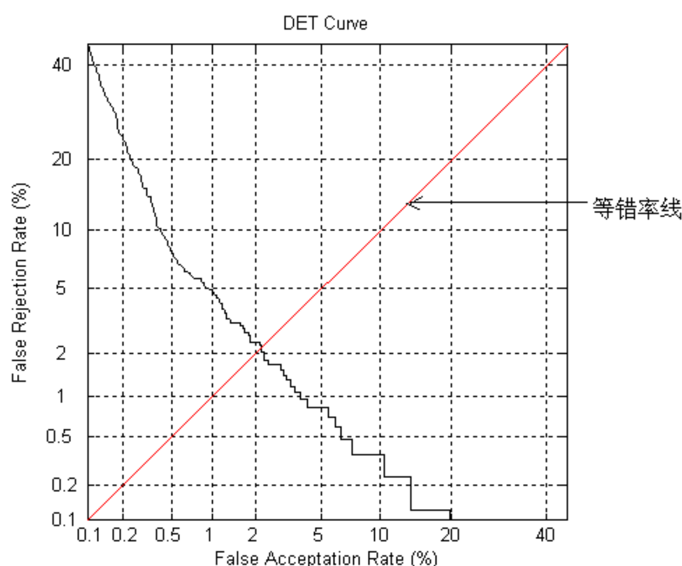


图 1.8 DET 曲线图

图中曲线基本能判定相应说话人识别系统的整体性能，但为了方便对比多个说话人识别系统的差异，消解阈值的影响，同时使两种错误率都尽可能的低，所以通常以等错误率（Equal Error Rate，EER）作为评价说话人系统综合性能的评价指标，即 FAR 与 FRR 相等时的错误率：

$$EER = FAR = FRR \tag{1-4}$$

而在不同的使用场景下，也可以自行调整 FAR 与 FRR 的关系。例如在银行、安保等安全性要求较高的场景下，需要尽可能低的错误接受率，则可以选择恒定

的 FRR 从而比较 FAR；而在智能家居的声纹解锁上更需要较高的解锁成功率，则可以选择恒定的 FAR 从而比较 FRR。这样做能够根据使用场景更加灵活的衡量声纹识别系统的性能，但也稍显麻烦。

1.5.2 minDCF

考虑到在一般测试集中，正负样本的数量差距极大，与实际使用场景可能不符，美国国家标准技术研究所（NIST）提出了一种模拟实际场景的指标，即检测代价函数（Detection Cost Function, DCF）：

$$DCF = C_{FR} \times P_{Target} \times FRR + C_{FA} \times P_{Nontarget} \times FAR \qquad (1\text{-}5)$$

其中C_{FR}和C_{FA}为相应的代价，P_{Target}和$P_{Nontarget}$ 分别为正负样本匹配的权重。通过调整这四个常数，起到模拟真实使用场景的作用，一般由评测方预先设定，最终取 DCF 最小值（minDCF）作为评价。

1.6 应用挑战

虽然声纹识别技术已取得很大发展，但等错误率（EER）普遍超过 1%（如 CNSRC2022 赛道 1 最好的成绩仍超过 4%），因此仍难以大规模应用，尤其在完全不受限的场景。如图 1.9 所示，声纹识别实际应用仍面临着噪声干扰、远场识别、短语音、快慢语速和跨信道等诸多挑战。

图 1.9 声纹应用挑战

究其原因，x-vector 网络是区分性模型，但训练人数往往有限（几千到几万人），难以穷尽所有说话人特征，对集外人泛化性不佳；另外声纹还需注册，而且条数有限，而测试数据则多样化，来自不同场景。

1.7 声纹数据库

NIST SRE—由 NIST 举办的说话人识别评测（Speaker Recognition Evaluation，SRE），始于 1996 年，是说话人识别领域目前最具代表性与知名度的评测任务。

Switchboard—对话式电话语音库，采样率 8kHz，包含来自美国各个地区 543 人的 2400 条通话录音。

Voxceleb—包含 Voxceleb1 和 Voxceleb2。VoxCeleb1 语音条数 153517 条，总时长约 350 个小时，说话人数 1251。VoxCeleb2 语音条数为 1092009，总时长约 2400 个小时，说话人数 5994。

CN-Celeb—该数据库包含了互联网公开可下载的 3000 位中国明星的声音数据，覆盖访谈、演讲、歌唱、影视、文娱等多种复杂场景。CN-Celeb 是目前已知最为复杂的说话人数据库，很适合验证声纹识别系统实际性能。

RSR2015—旨在为文本相关说话人识别领域提供包含不同语音时长和不同文本内容的开源数据集，总共包含超过 151 小时的来自 300 个说话人（143 女性，157 男性）的语音数据，文本内容均为英语。数据集由六台不同的移动设备录制，每个说话人随机选取其中的三台设备录制音频，存在跨信道的情况。

你好米雅—希尔贝壳（AISHELL）开源的中英文唤醒词语音数据库，可用于文本相关声纹识别实验，有 254 名发言人，录制过程在真实家居环境中，设置 7 个录音位，使用 6 个圆形 16 路 PDM 麦克风阵列录音板做远讲拾音（16kHz，16bit）、1 个高保真麦克风做近讲场音（44.1kHz，16bit）。

1.8 开源工具

声纹识别开源工具主要包括：

- Kaldi 是一个开源的语音识别工具箱，是基于 C++编写的，可以在 Windows 和 Unix 平台上编译，主要由 Daniel Povey 博士在维护。Kaldi 支持 TDNN、F-TDNN、ResNet 等声纹识别模型。
- SpeechBrain 是来自蒙特利尔大学等众多研究机构、Yoshua Bengio 参与的语音处理工具包，用户可以快速实现语音识别、声纹识别、语音增强、信号处理等等任务。
- ASV-Subtools 是厦门大学语音实验室于 2020 年 5 月推出的高效、易于开发扩展的声纹识别开源工具。ASV-Subtools 充分结合了 Kaldi 在语音信号和后端处理的高效性以及 PyTorch 开发和训练神经网络的便捷灵活性。除了集成 Kaldi 本身提供的脚本外，该工具还基于 Kaldi 封装了很多实用、高效的脚本，其中包括数据集处理、数据扩增、特征提取、静音消除、Kaldi 模型训练、x-vector 加速提取、后端打分和指标计算等。

ASV-Subtools 已被国内外不少研究机构采用，相对使用 Kaldi 工具，效果有较明显提升，其模块化设计也比 SpeechBrain 更友好。接下来我们将基于 ASV-Subtools 工具，重点介绍声纹识别核心算法和实战过程。

第 2 章 声纹核心算法

本章介绍基于 x-vector 的声纹识别核心算法，具体包括网络架构、损失函数、后端分类器和学习机制，其中后端分类器包括 Cosine、PLDA 以及可能采用的自适应方法，学习机制包括注意力学习、多任务学习、迁移学习等方法。

2. 1 网络架构

x-vector 网络包括帧级别特征提取模块、池化层和段级别特征提取模块，实现了从帧级别到段级别的学习，可以有效消除畸形帧带来的影响，使得到的说话人表征更加稳定，同时也让它在噪声环境下具备更强的鲁棒性。池化层一般采用统计池化层（Statistic Pooling）。统计池化层的作用就是在帧级别学习后，计算整句话的均值和标准差，拼接后作为段级别学习的输入。提取 x-vector 时只需要段级别第一层隐藏层的输出，因此在使用时可以将倒数两层剪裁掉，缩减模型大小。

2.1.1 TDNN

帧级别提取模块可以采用时延神经网络（Time Delay Neural Network, TDNN），如图 2.1 所示。借助卷积的操作在神经网络中实现了帧的拼接，通过参数设定，可以自由选择与相邻帧拼接或跳帧拼接，使得神经网络可以充分学习到上下文信息，而不只是在输入层堆叠上下文信息。

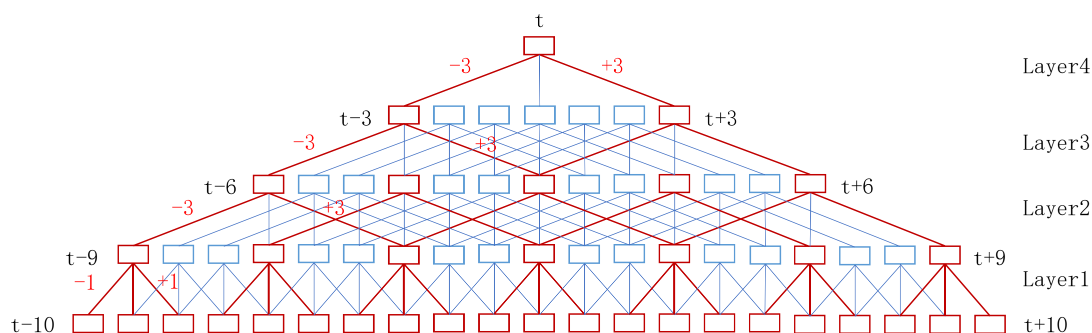


图 2.1 TDNN

TDNN 模型的优势在于，只需要很小的卷积核就能够通过拓展卷积获得较多的上下文信息，参数量较小，模型中的每一层网络都能汇聚当前帧的上下文信息，网络越深，信息范围越广。x-vector 框架下使用的标准 TDNN 模型参数见表 2-1，表中模型共汇聚了 15 帧信息。

表 2-1 标准 TDNN 模型结构与参数

层序	网络层	时序范围	维度
1	TDNN-ReLU	t-2:t+2	512
2	TDNN-ReLU	t-2,t,t+2	512
3	TDNN-ReLU	t-3,t,t+3	512
4	Dense-ReLU	t	512
5	Dense-ReLU	t	1500
	Pooling(mean+stddev)	Full-seq	3000
6	Dense-ReLU		512
7	Dense-ReLU		512
	Dense-Softmax		num_targets

2.1.2 E-TDNN

TDNN 模型还可进一步扩展，适当增加网络层数，从而包含更多上下文信息。扩展时延神经网络（Extended Time Delay Neural Networks, E-TDNN）模型参数见表 2-2。可以看到，E-TDNN 模型里的网络类型与常用的 TDNN 模型一样，没有增加其他类型神经网络，主要区别在于模型汇聚了 23 帧信息，上下文的信息更多。而在堆叠结构上也有一个特殊的设计，在每一个拼接了其他帧的网络层后都增添了一层全连接层，这样可以加深神经网络模型，但参数量不会有过大增长，在性能与模型参数量之间取得一定平衡。经实验验证，这样的模型结构较常用的 TDNN 模型有很大改善。

表 2-2 E-TDNN 模型结构与参数

层序	网络层	时序范围	维度

1	TDNN-ReLU	t-2:t+2	512
2	Dense-ReLU	t	512
3	TDNN-ReLU	t-2,t,t+2	512
4	Dense-ReLU	t	512
5	TDNN-ReLU	t-3,t,t+3	512
6	Dense-ReLU	t	512
7	TDNN-ReLU	t-4,t,t+4	512
8	Dense-ReLU	t	512
9	Dense-ReLU	t	512
10	Dense-ReLU	t	1500
	Pooling(mean+stddev)	Full-seq	3000
11	Dense-ReLU	-	512
12	Dense-ReLU	-	512
	Dense-Softmax	-	num_targets

2.1.3 F-TDNN

虽然 TDNN 模型较普通 DNN 模型，在参数量上已经大大缩减，但是在加深堆叠层数后，参数量还是显得过于庞大，而且堆叠层数带来的性能收益越来越微弱。在 TDNN 模型中，编码网络部分参数主要集中在 TDNN 权重矩阵中，想要继续减少参数，就要缩减矩阵参数。

常见缩减矩阵参数的方法是是奇异值分解（Singular Value Decomposition，SVD），将一个大的权重矩阵分解成两个小的权重矩阵来缩减参数量。Daniel Povey 在 2018 年提出分解时延神经网络（Factorized Time Delay Neural Networks, F-TDNN），模型结构如图 2.2。

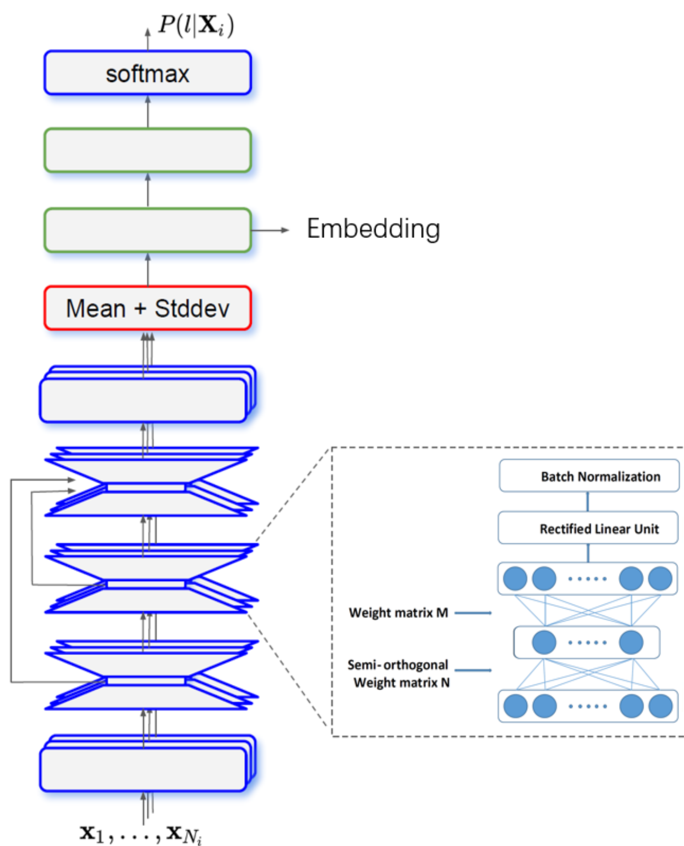


图 2.2 F-TDNN 模型结构图

F-TDNN 与 TDNN 相比，相同建模能力下参数量更少，而且由于使用了跳连接结构，所以可以堆叠得更深，学习到更抽象的说话人表征。F-TDNN 模型参数见表 2-3。

表 2-3 F-TDNN 模型结构与参数

层序	网络层	分解时序范围 1	分解时序范围 2	跳连接层	维度	内部维度
1	TDNN-ReLU	t-2:t+2	-		512	
2	F-TDNN-ReLU	t-2,t	t,t+2		1024	256
3	F-TDNN-ReLU	t	t		1024	256
4	F-TDNN-ReLU	t-3,t	t,t+3		1024	256
5	F-TDNN-ReLU	t	t		1024	256

6	F-TDNN-ReLU	t-3,t	t,t+3		1024	256
7	F-TDNN-ReLU	t-3,t	t,t+3	2,4	1024	256
8	F-TDNN-ReLU	t-3,t	t,t+3		1024	256
9	F-TDNN-ReLU	t	t	4,6,8	1024	256
10	Dense-ReLU	t	t		2048	-
	Pooling(mean+stddev)	Full-seq			4096	-
11	Dense-ReLU	-	-		512	-
12	Dense-ReLU	-	-		512	-
	Dense-Softmax	-	-		num_targets	-

2.1.4 ResNet

残差网络（Residual Networks，ResNet）将两层 CNN 组合在一起，使用捷径（Shortcut）将网络的输入加到输出中，构成一个残差模块。一般使用卷积核为 3 的 CNN，具体结构如图 2.3 中 Basic-Block 所示。也可以使用卷积核为 1 的 CNN 改变通道数，从而降低参数量，中间使用卷积核为 3 的 CNN，具体结构如图 2.3 中 Bottleneck-Block 所示，图中红线即为 Shortcut。在搭建网络模型时，直接堆叠残差模块即可。这样让深层网络可以从浅层网络中学习到特质，能够堆叠很深的网络模型，解决网络退化问题。

在语音信号处理中，声学特征 FBank 相较于 MFCC 没有经过离散余弦变换，所以不同维度上的信息具有一定相关性，与图像的特点吻合，适合使用残差网络。

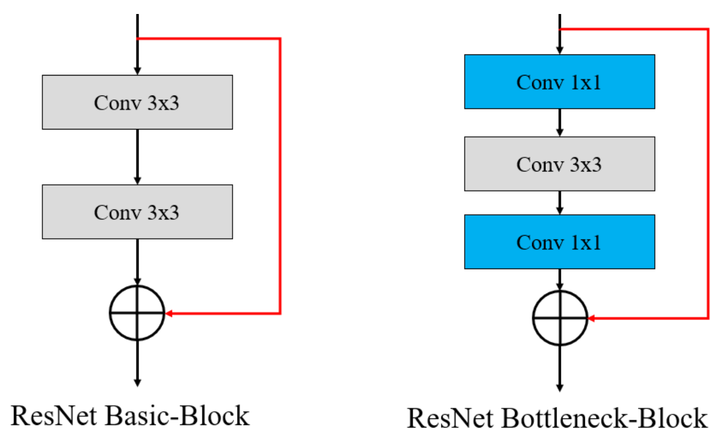


图 2.3 残差模块结构图

标准的 ResNet34 的结构相对于图像领域海量的数据来说比较友好，而说话人识别领域数据相对来说要少得多，将每个残差结构的维度进行缩小一半，并且在统计池化层之后的段级别处理层，只保留一层段级别层后接损失函数层。具体网络结构如表 2-4 所示。

表 2-4 ResNet34 模型结构与参数

网络层	结构	输出
Input	-	D*T*1
Conv2D-1	3×3, Stride 1	D*T*32
ResNet Block-1	$\begin{bmatrix} 3\times3,\ 32 \\ 3\times3,\ 32 \end{bmatrix}\times3$, Stride 1	D*T*32
ResNet Block-2	$\begin{bmatrix} 3\times3,\ 64 \\ 3\times3,\ 64 \end{bmatrix}\times4$, Stride 2	$\left\lceil\frac{D}{2}\right\rceil*\left\lceil\frac{T}{2}\right\rceil*64$
ResNet Block-3	$\begin{bmatrix} 3\times3,128 \\ 3\times3,128 \end{bmatrix}\times6$, Stride 2	$\left\lceil\frac{D}{4}\right\rceil*\left\lceil\frac{T}{4}\right\rceil*128$
ResNet Block-4	$\begin{bmatrix} 3\times3,256 \\ 3\times3,\ 256 \end{bmatrix}\times3$, Stride 2	$\left\lceil\frac{D}{8}\right\rceil*\left\lceil\frac{T}{8}\right\rceil*256$
Statistics Pooling	-	512
FC1	-	256
AM-Softmax	-	num_targets

ResNet 还可以将原本的残差模块替换为 Squeeze-and-Excitation Block（SE-Block），如图 2.4 所示，该模块主要的功能是对各个通道进行权重的分配，就像 Attention 一样，帮助网络学习到重要的特征信息。

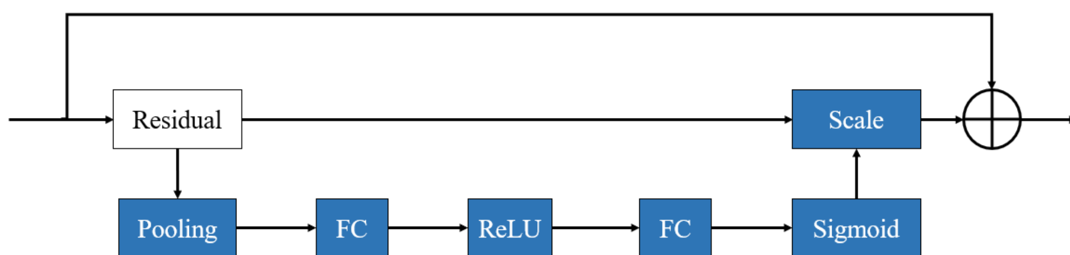


图 2.4 SE-Block

ResNet 加上 SE-Block 形成 ResNet-SE 网络，如表 2-5 所示。

表 2-5 ResNet34-SE 模型结构与参数

网络层	结构	输出
Input	-	D*T*1
Conv2D-1	3×3, Stride 1	D*T*32
ResNet Block-1	$\begin{bmatrix} 3\times 3, 32 \\ 3\times 3, 32 \\ SE-Block \end{bmatrix}$ ×3, Stride 1	D*T*32
ResNet Block-2	$\begin{bmatrix} 3\times 3, 64 \\ 3\times 3, 64 \\ SE-Block \end{bmatrix}$ ×, Stride 2	$\left\lceil \frac{D}{2} \right\rceil * \left\lceil \frac{T}{2} \right\rceil *64$
ResNet Block-3	$\begin{bmatrix} 3\times 3, 128 \\ 3\times 3, 128 \\ SE-Block \end{bmatrix}$ ×6, Stride 2	$\left\lceil \frac{D}{4} \right\rceil * \left\lceil \frac{T}{4} \right\rceil *128$
ResNet Block-4	$\begin{bmatrix} 3\times 3,256 \\ 3\times 3, 256 \\ SE-Block \end{bmatrix}$ ×3, Stride 2	$\left\lceil \frac{D}{8} \right\rceil * \left\lceil \frac{T}{8} \right\rceil *256$
Statistic Pooling	-	512
FC1	-	256
AM-Softmax	-	num_targets

2.1.5 ECAPA-TDNN

ECAPA-TDNN[10]基于 Res2Net 残差设计，采用跨层聚合，结合 SE-Block 通道增强，如图 2.5 所示。Res2Net 在粒度级别上表示了多尺度特征，并增加了每层的感受野。把输入 channels 均等分，比如 512 维切分成 8 个尺度，每个尺度

64 维分别进行一维扩张卷积，后面卷积的输入等于划分后的 channels 和前一卷积的输出之和。ECAPA 将 SE 接到模块末端。

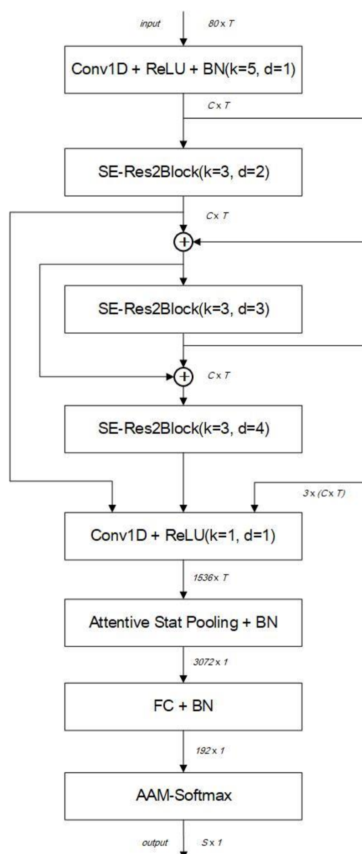


图 2.5 ECAPA-TDNN

2.1.6 Conformer

Conformer 是一种混合架构，它将注意力机制与卷积相结合，即通过 self-attention 学习全局交互，而 CNN 捕获局部信息。这种的卷积与自注意机制的混合设计可以很好地对长短信息建模，而且通常认为自注意力机制的网络模型容量大于卷积网络，因此更适合对大规模数据进行建模。

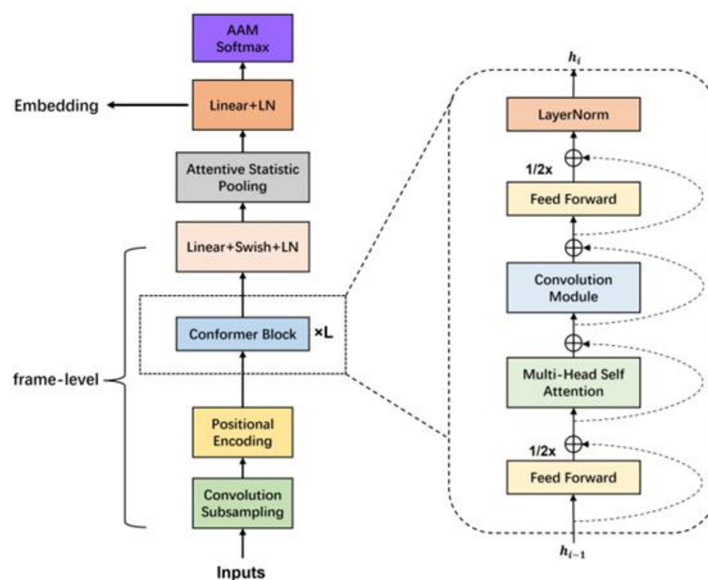


图 2.6 Conformer[11][10]模型结构图

2. 2 损失函数

2.2.1 CE Loss

以分类为目标的损失函数主要采用交叉熵损失（CE Loss）。结合 Softmax 激活函数，CE Loss 的形式如下：

$$\mathcal{L}_{CE}=-\frac{1}{n}\sum_{i=1}^{n}log\frac{e^{W_{y_i}^{T}f_i+b_{y_i}}}{\sum_{j=1}^{c}e^{W_j^{T}f_i+b_j}} \tag{2-1}$$

其中，n为 Batch 的大小，c为分类类别数（输出层结点），W为线性层权重，一共包含c个向量，分别对应到每个类别，b为线性层的偏置，而 f为上一层所输出的特征。

2.2.2 Margin Loss

为了统一类别间的区分方式以方便增加距离惩罚，首先将式（2-1）分子部分写为 $W_j^T\cdot f+b=||W_j||\cdot||f||\cdot\cos(\theta)+b$（其中$\theta$为特征和代表类别中心的权重向量之间的角度），然后令偏置$b=0$。

将 CE Loss 函数进行简单修改并总是通过向量长度规整使得 $||W||=1$ 后，即可获得基于角度的更改损失函数，如式（2-2）所示。

$$\mathcal{L}_{Modified}=-\frac{1}{n}\sum_{i=1}^{n}log\frac{e^{||f_i||\cos(\theta_{y_i,i})}}{\sum_{j=1}^{c}e^{||f_i||\cos(\theta_{j,i})}} \tag{2-2}$$

Margin Loss 通过对决策间距做出了不同的改进来增大类间间距，其具体形式包括 A-Softmax、AM-Softmax、AAM-Softmax 等损失函数。

AM-Softmax 损失函数通过对角度施加惩罚来改变决策边界并促使类别间产生空白区域，其形式如式（2-3）所示。

$$\mathcal{L}_{AM}=-\frac{1}{n}\sum_{i=1}^{n}log\frac{e^{s\cdot(\cos(\theta_{y_i,i})-m)}}{e^{s\cdot(\cos(\theta_{y_i,i})-m)}+\sum_{j=1,j\neq y_i}^{c}e^{s\cdot\cos(\theta_{j,i})}} \tag{2-3}$$

其中，AM-Softmax 损失函数的决策边界如式（2-4）所示：

$$\begin{cases}\cos(\theta_{y_i,i})-\cos(\theta_{y_j,i})=m, for\ class\ i\\ \cos(\theta_{y_j,j})-\cos(\theta_{y_i,j})=m\ , for\ class\ j\end{cases} \tag{2-4}$$

显然，对于不同类别来说，AM-Softmax 相对 Softmax 函数的决策边界相隔了距离m。

AAM-Softmax 损失函数的形式分别如式（2-5）所示：

$$\mathcal{L}_{AAM}=-\frac{1}{n}\sum_{i=1}^{n}log\frac{e^{s\cdot\cos(\theta_{y_i,i}+m)}}{e^{s\cdot\cos(\theta_{y_i,i}+m)}+\sum_{j=1,j\neq y_i}^{c}e^{s\cdot\cos(\theta_{j,i})}} \tag{2-5}$$

AM-Softmax 损失函数通过惩罚余弦来间接惩罚角度，而 AAM-Softmax 损失函数直接惩罚角度。

2.3 后端分类器

2.3.1 LDA

线性判别分析（Linear Discriminative Analysis, LDA）是传统机器学习中常用的降维方法，目标是降维原始数据，且使得降维后同类别数据尽可能的靠近，不同类别的数据尽可能远离。在说话人识别中，常使用 LDA 对说话人表征降维，使其对不同说话人有更好的区分度。LDA 作用示意图如图 2.6 所示。

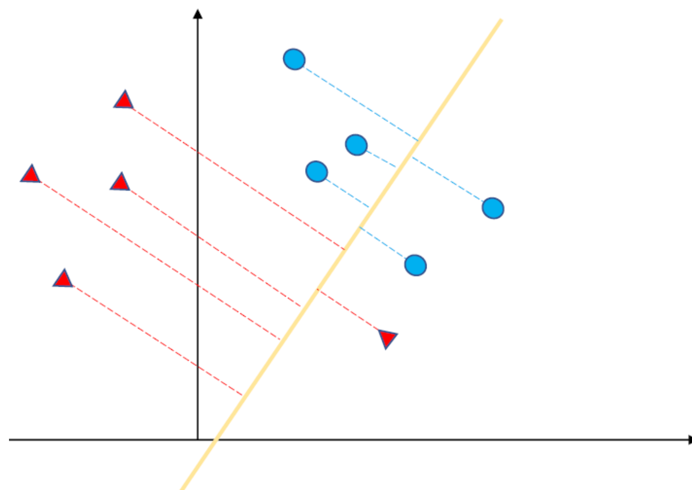


图 2.7 LDA 作用示意图

LDA 算法流程如下：

- 定义类内散布矩阵S_W

$$S_W=\sum_{c=1}^{C}\frac{1}{N_c}\sum_{x\in X_c}(x-\mu_c)(x-\mu_c)^T \quad (2\text{-}6)$$

- 定义类间散布矩阵S_B

$$S_B=\sum_{c=1}^{C}(\mu_c-\mu)(\mu_c-\mu)^T \quad (2\text{-}7)$$

- 设投影矩阵为w，求解

$$w^*=arg\max_{w}\left\{\frac{w^T S_B w}{w^T S_W w}\right\} \quad (2\text{-}8)$$

其中，C为样本数，x为样本，N_c为类内样本个数，μ_c为类内样本均值，μ为所有样本均值。计算矩阵$S_w^{-1}S_b$的特征值与特征向量，根据降维后的目标维度d，从小到大取前d个特征值对应的特征向量，组成投影矩阵，最后根据投影矩阵得到降维后的向量。

2.3.2 Cosine

余弦（Cosine）打分是声纹识别中最基础的打分方式。说话人表征大多是矢量的形式,一般认为如果两个矢量平行或重合，则相似度为 1；如果两个矢量垂直或者正交，则相似度为 0。余弦函数在 0 到 90 度区间内的值正好契合，所以可以用两个矢量夹角的余弦值来度量说话人表征间的相似度，余弦打分的公式如式（2-9），式中x_1和x_2分别表示注册语音和测试语音的说话人表征矢量。

$$s(x_1,x_2)=\frac{x_1\cdot x_2}{||x_1||\cdot||x_2||} \quad (2\text{-}9)$$

2.3.3 PLDA

概率线性判别分析（Probability Linear Discriminative Analysis, PLDA）是一种信道补偿算法，在声纹识别中也是一种主流打分方式。说话人表征的提取，通常完整的记录了说话人信息，其中也包含了部分信道信息和噪声信息，通过 PLDA 可以有效补偿信道差异，消除噪声干扰，从而进一步优化打分评价。训练数据中第i个说话人的第j句话的说话人表征可以用式（2-10）表示：

$$x_{ij} = \mu + Fh_i + Gw_{ij} + \epsilon_{ij} \tag{2-10}$$

其中，μ是全局均值向量，与说话人、信道、噪声都无关，是一个基底；F是说话人变换矩阵，h_i是它的隐变量，只与当前说话人i相关；G是信道变换矩阵，w_{ij}是它的隐变量，与说话人无关，只与当前句子j的信道信息相关；ϵ_{ij}是剩余变量，表征噪声等无关变量信息。

在 PLDA 训练时，会使用最大期望算法（Expectation Maximization，EM），通过使得隐变量h_i和w_{ij}期望最大化，迭代估算出μ、F和G的参数。

两个不同句子的说话人表征x_1和x_2，只可能存在两种情况：

H_0，两条语句来自同一个说话人，表达式可写为：

$$H_0 = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \mu \\ \mu \end{bmatrix} + \begin{bmatrix} F & G & 0 \\ F & 0 & G \end{bmatrix} \begin{bmatrix} h_{12} \\ w_1 \\ w_2 \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \end{bmatrix} \tag{2-11}$$

H_1，两条句子来自不同说话人，表达式可写为：

$$H_1 = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \mu \\ \mu \end{bmatrix} + \begin{bmatrix} F & G & 0 & 0 \\ 0 & 0 & F & G \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ w_1 \\ w_2 \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \end{bmatrix} \tag{2-12}$$

PLDA 模型将说话人表征投影到子空间F上，以此最大化表示说话人之间的差异，并排除信道子空间G的影响，最后用式(2-13)计算两个说话人表征的相似度。

$$s(x_1, x_2) = log\, p\,(x_1, x_2 \mid H_0) - log\, p\,(x_1, x_2 \mid H_1) \tag{2-13}$$

PLDA 的另一种方式是 Two-Covariance PLDA，假设样本为分阶段的采样生成过程。

首先，生成不同说话人$\boldsymbol{h}_i$的类别中心，其中，$\boldsymbol{\Phi}_b$为类间协方差矩阵

$$\boldsymbol{h}_i \sim \mathcal{N}(\boldsymbol{h}_i | \boldsymbol{\mu}, \boldsymbol{\Phi}_b) \tag{2-14}$$

基于给定的说话人中心，生成说话人表征样本，其中，$\boldsymbol{\Phi}_w$为类内协方差矩阵

$$\boldsymbol{x}_{ij}|\boldsymbol{h}_i \sim \mathcal{N}(\boldsymbol{x}_{ij}|\boldsymbol{h}_i, \boldsymbol{\Phi}_w) \tag{2-15}$$

Two-covariance PLDA 模型参数的估算过程也采用 EM 算法，具体如下。

1．初始化

根据训练集数据计算全局均值μ，随机生成Φ_b和Φ_w，令$B = \Phi_b^{-1}$和$W = \Phi_w^{-1}$。

2．计算统计量

按照每个说话人语音样本数M_i大小进行排序说话人样本；

计算统计量

$N = \sum_{i=1}^{K} M_i$，$f_i = \sum_{j=1}^{M_i} x_{ij}$，$S = \sum_{i=1}^{K}\sum_{j=1}^{M_i} x_{ij}x_{ij}^T$

3．迭代

3.1 E-step

设置 $T \leftarrow 0$，$R \leftarrow 0$，$\mathcal{Y} \leftarrow 0$ 。

当 i 从 1 增加到 K 时

如果$M_i \neq M_{i-1}$ ，计算$L_i = B + M_i W$；

否则$L_i \leftarrow L_{i-1}$

$\mathbb{E}[h_i] = L_i^{-1}\gamma$，$\mathbb{E}\left[h_i h_i^T\right] = L_i^{-1} + \mathbb{E}[h_i]\mathbb{E}[h_i]^T$，其中$\gamma = B\mu + Wf_i$

更新$T = \sum_{i=1}^{K} \mathbb{E}[h_i] f_i^T$，$R = \sum_{i=1}^{K} M_i \mathbb{E}\left[h_i h_i^T\right]$和$\mathcal{Y} = \sum_{i=1}^{K} M_i \mathbb{E}[h_i]$。

3.2 M-step

更新$\mu = \frac{1}{N}\mathcal{Y}$，　$\Phi_b = B^{-1} = \frac{1}{N}(R - 2\mu\mathcal{Y}^T) + \mu\mu^T$和 $\Phi_w = W^{-1} = \frac{1}{N}(S - 2T + R)$。

4．直到收敛，确定 PLDA 模型参数。

Two-covariance PLDA 的打分方式如下：

$$\begin{aligned} s(x_1, x_2) &= log\, p\,(\,x_1, x_2 \mid H_0\,) - log\, p\,(\,x_1, x_2 \mid H_1\,) \\ &= \log\mathcal{N}\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}; \begin{bmatrix} \boldsymbol{\mu} \\ \boldsymbol{\mu} \end{bmatrix}\begin{bmatrix} \boldsymbol{\Phi}_w + \boldsymbol{\Phi}_b & \boldsymbol{\Phi}_b \\ \boldsymbol{\Phi}_b & \boldsymbol{\Phi}_w + \boldsymbol{\Phi}_b \end{bmatrix}\right) - \log\mathcal{N}\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}; \begin{bmatrix} \boldsymbol{\mu} \\ \boldsymbol{\mu} \end{bmatrix}\begin{bmatrix} \boldsymbol{\Phi}_w + \boldsymbol{\Phi}_b & 0 \\ 0 & \boldsymbol{\Phi}_w + \boldsymbol{\Phi}_b \end{bmatrix}\right) \end{aligned} \tag{2-16}$$

PLDA 打分对数据分布较为敏感，当测试集与训练集存在显著差异时，性能会相对衰减，这也就是域不匹配（Domain Mismatch）。在打分过程中，对于域不匹配的任务可以通过调整 PLDA 参数，在新域的测试中稳定性能，即 Adapted Probability Linear Discriminative Analysis（APLDA）。APLDA 主要根据新域的数据求得全局均值，进行调整。

2.4 学习机制

2.4.1 注意力学习

注意力学习主要在池化层使用，原有的统计池化（Statistics Pooling）方式没有考虑每帧的重要性。如图 2.7 所示，注意力池化（Attention Pooling）针对每帧获取相应的注意力权重，然后结合到均值和方差的计算过程中。

多头（Multi-head）注意力机制则基于若干个池化层，做并行的注意力操作，各个池化层独立实施注意力机制，最后联合各池化层的输出。

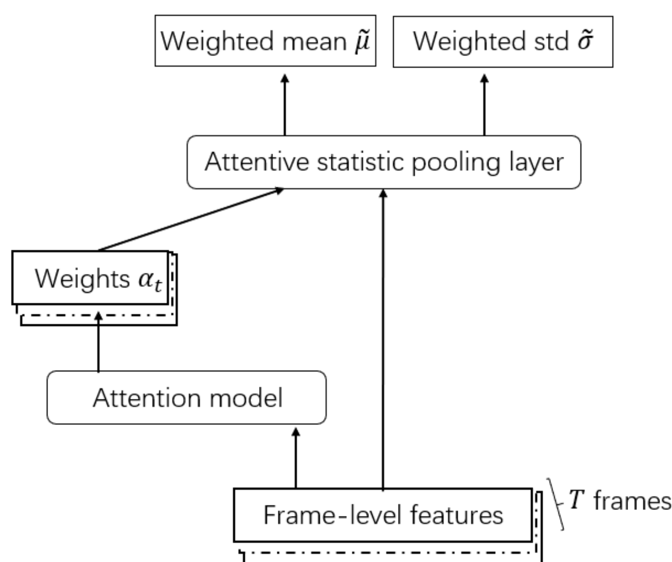


图 2.8 基于 Attention 机制的 Pooling 方式[9]

2.4.2 多任务学习

多任务学习（Multi-task Learning）是利用多个相关任务之间蕴含的共性信息对每个独立的任务起到辅助和性能提升的一种学习策略。常用的多任务学习框架在两个子任务中获取共性信息采用的是共享隐藏层的方式，通过共享隐藏层进行参数共享，从而提高所需任务的性能。采用共享层的形式一般应用于不同子任务具有相同的网络结构和训练数据的情况下，不同子任务的区别只是任务目的的不同。多任务学习常见的参数共享形式如图 2.8。

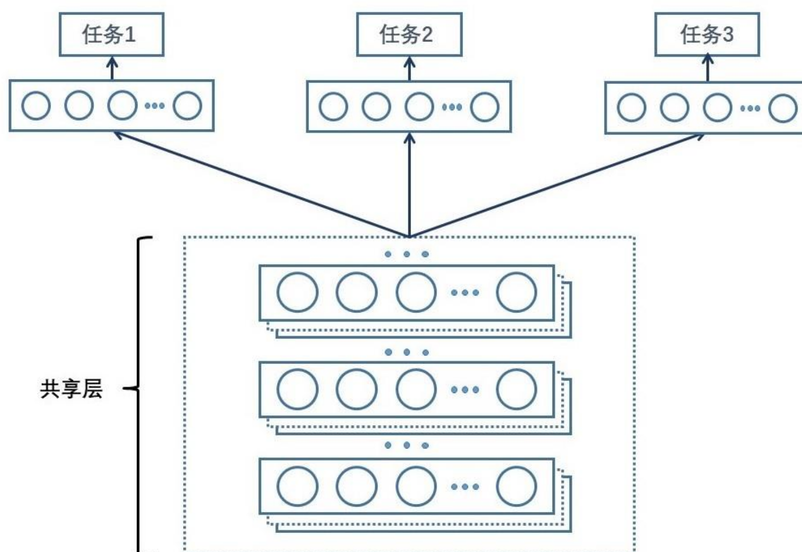


图 2.9 多任务学习（参数共享）

2.4.3 迁移学习

迁移学习的思想是从某个领域中学习到知识结构或模式，然后就这些知识经验通过迁移的方式，转移到新的目标领域，让目标领域的性能得到改进。例如，可以利用文本无关数据辅助文本相关任务的学习，具体结构如图 2.9 所示：

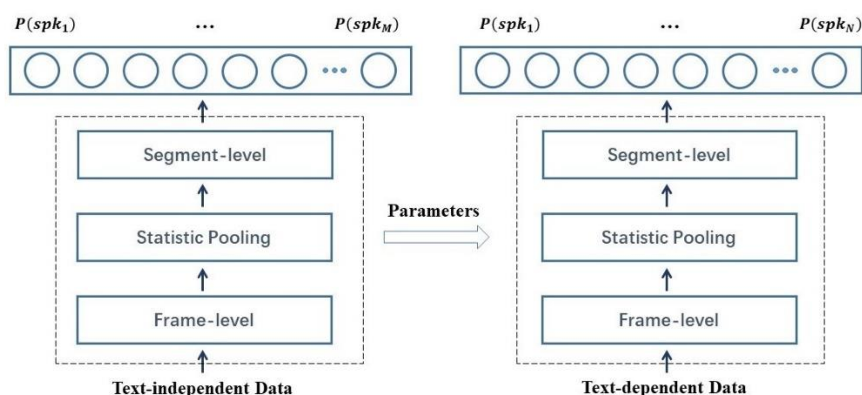


图 2.10 迁移学习（文本无关=>文本相关）

迁移学习的步骤一般如下：

Step 1：利用域外数据训练好网络；

Step 2：将预训网络参数作为域内数据集网络的初始参数；或者，冻结该预训网络的部分参数，使用域内数据训练更新少量网络参数；

Step 3：域内数据所使用的网络可以调整代价函数。

第 3 章 Subtools 工具介绍

为了满足日常研究需要，并能够在同样的条件下公平对比各种算法，厦门大学智能语音实验室（XMU Speech Lab）基于 Pytorch 自行设计并开发了一套声纹识别工具，命名为 ASV-Subtools，并于 2020 年 5 月在 GitHub 开源[1]。

ASV-Subtools 主要分为两部分，第一部分包含 Kaldi 本身提供的脚本和基于 Kaldi 封装的很多实用、高效的脚本，包括数据集处理、数据扩增、特征提取、静音消除、Kaldi 模型训练、x-vector 加速提取、后端打分和指标计算等脚本。第二部分包含基于 Pytorch 的神经网络训练高层框架和有关神经网络训练的流程脚本。它可用来替代 Kaldi 的模型训练，并兼容 Kaldi 其他脚本模块。该部分为 ASV-Subtools 的核心。

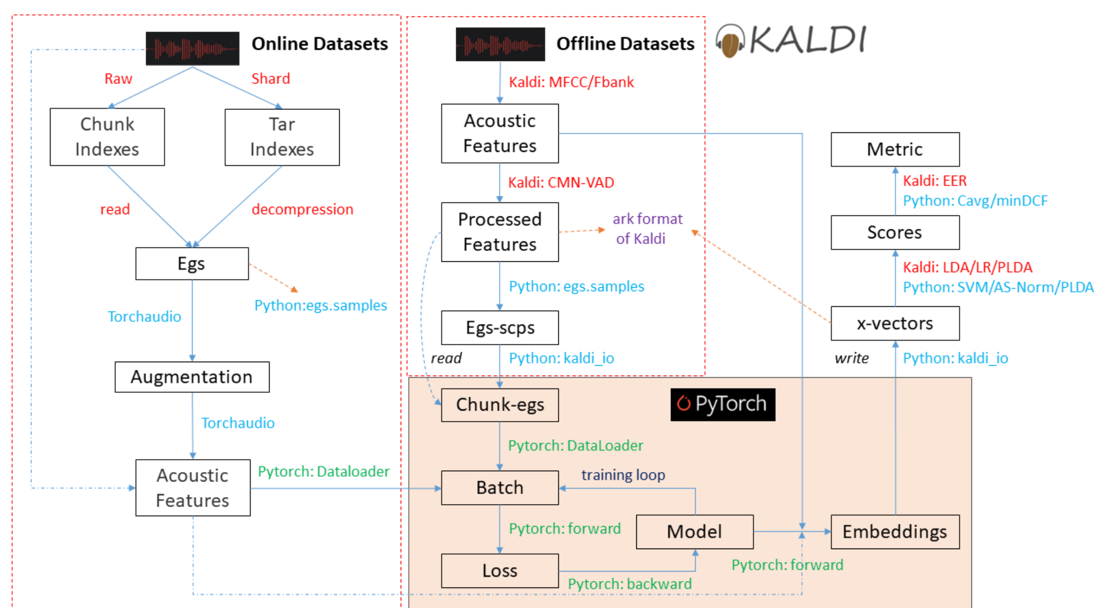


图 3.1 ASV-Subtools 框架

ASV-Subtools 主要的设计理念为在代码高度复用的同时保持模块分化和开发自由，这能够保证整体框架的简洁易懂，并方便与原生的 Pytorch 模型互相移植，进而达到能够快速复现、开发和加快研究效率的效果。目前为止，ASV-

[1] https://github.com/Snowdar/asv-subtools

Subtools 已开发了众多声纹识别中常用的算法，其中，使用一维卷积等价实现的标准 x-vector 网络，在同等训练集下训练同样的声纹识别模型，相对使用 Kaldi 工具，效果可整体提升 10%以上。

3.1 工程结构

本小节介绍 ASV-Subtools 的工程结构，并对 ASV-Subtools 中的特色优化进行简要介绍。

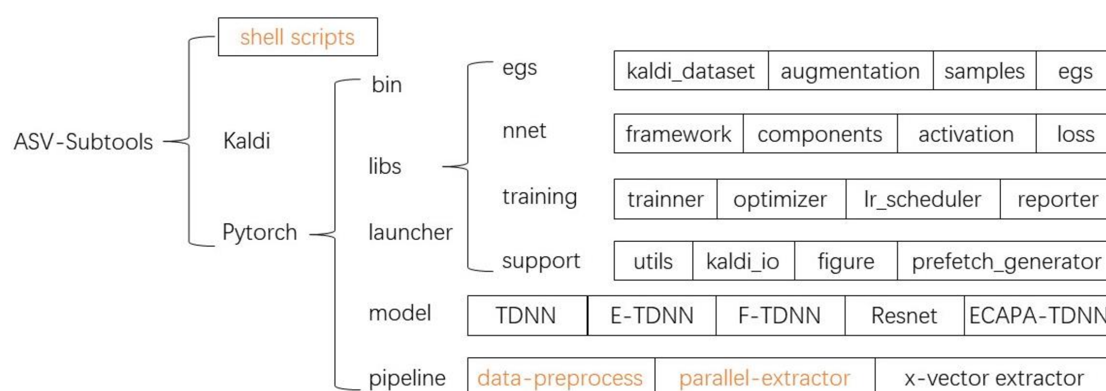


图 3.2 ASV-Subtools 组件

ASV-Subtools 的组件如图 3.2 所示，主要包含三部分。

首先是 shell 脚本部分。为了使得数据部分的操作更加方便，ASV-Subtools 额外封装了一系列数据集处理脚本作为 Kaldi 的工具集补充，如更方便的特征提取脚本，数据增强脚本，数据集过滤脚本，x-vector 数据过滤脚本，数据集分组脚本，打分脚本等，其中，过滤脚本可有效避免子集数据的重复操作，节省时间。而在数据集分组脚本中，为了平衡多进程处理时每个进程的处理速度，以尽可能保证整体速度最优，因此对数据集进行分组时不再使用 Kaldi 的按语音条数平均分配，而是按语音的帧数进行平均分配。同时，考虑到数据集的庞大，所有数据集处理脚本均进行了速度优化，如代码上的时间复杂度优化或使用多进程进行提速。

由于后端处理有很多可能的复杂组合，如可选的使用 LDA、去均值、白化和向量长度规整处理 x-vector，然后使用 SVM、LR、PLDA 以及 GMM 等分类器进行打分或直接使用 Cosine 进行打分，同时用于后端打分的训练集、注册集和测

试集之间也有较多组合，为了方便后端调试而避免每次重写代码，ASV-Subtools 中实现了一个打分集脚本。针对给定的打分顺序，该脚本通过图的深度遍历方法自动将整个打分过程连接起来，如图 3.3 所示，该脚本围绕注册集、测试集和打分三条线进行搜索，并自动将当前处理的输出送入到下一个处理中，并最终完成打分。

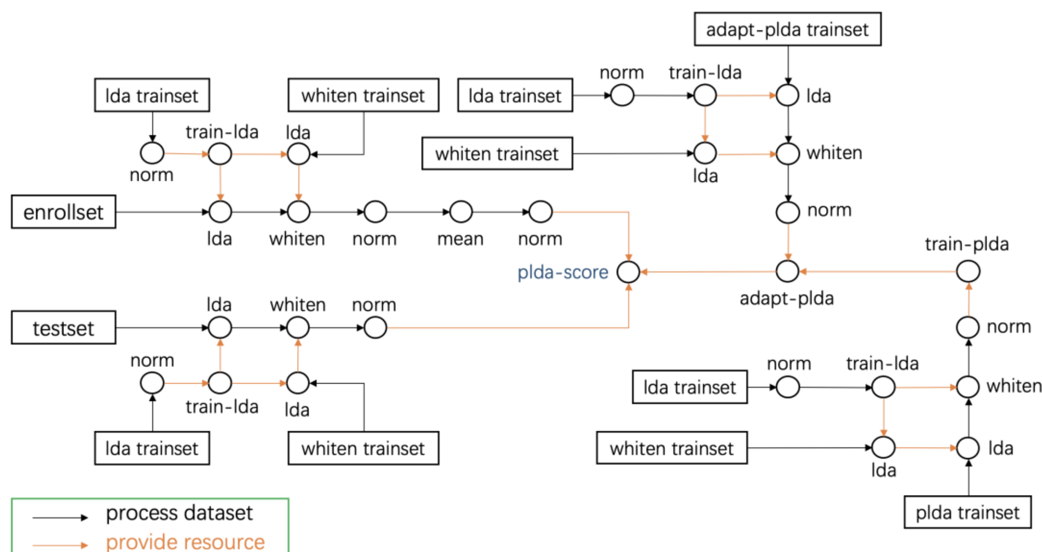


图 3.3 后端打分集脚本原理示意图

其次是 Kaldi 部分。该部分主要包含 Kaldi 的脚本集如 steps 和 utils，以及 Kaldi 的模型训练如 i-vector、x-vector 和多任务 x-vector 等训练脚本。同时，在该目录中，还包含快速编译 Kaldi C++命令的编译脚本，在 Linux 上基于 Kaldi 库进行 C++开发，通过该脚本无需额外搭建编译环境，即可快速实现编译并将可执行程序添加到 Kaldi 的环境中。

最后是 Pytorch 部分。该部分包含了整个训练框架，其中 bin 包含一些单独的工具脚本，如打印模型和计算模型参数等，libs 为核心代码库，model 为构建的模型示例，launcher 为运行整个训练程序的启动器示例，而 pipeline 则是用来在启动器中进行调用，以衔接 Kaldi 的数据处理和后端打分两个模块。由于该训练框架涉及的细节非常多，在后一小节进行详细介绍。

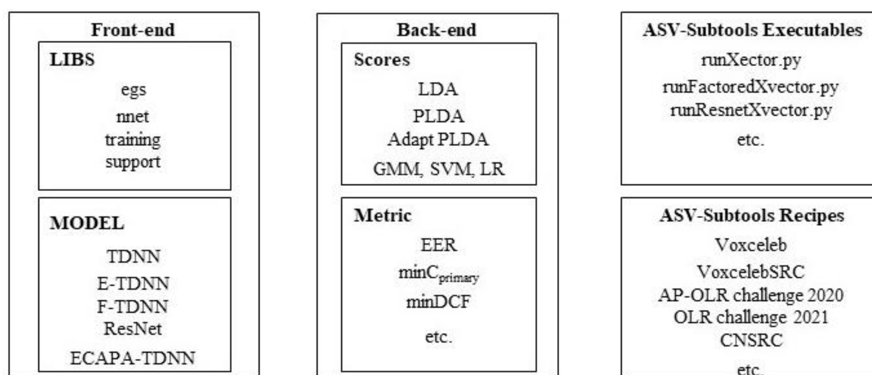


图 3.4 ASV-Subtools 工程模块

图 3.4 列出了 ASV-Subtools 工程模块，包括 Front-end、Back-end 部分，以及具体的运行代码。其中 Front-end 的 MODEL 包括 TDNN、E-TDNN、F-TDNN、Resnet 和 ECAPA-TDNN 等模型。ASV-Subtools 对每个模型均有对应的设计代码，如 TDNN 的结构设计如下：

```
# Nnet
self.aug_dropout = torch.nn.Dropout2d(p=aug_dropout) if aug_dropout > 0 else None
self.tdnn1 = ReluBatchNormTdnnLayer(inputs_dim,512,[-2,-1,0,1,2],nonlinearity=nonlinearity)
self.tdnn2 = ReluBatchNormTdnnLayer(512,512,[-2,0,2],nonlinearity=nonlinearity)
self.tdnn3 = ReluBatchNormTdnnLayer(512,512,[-3,0,3],nonlinearity=nonlinearity)
self.tdnn4 = ReluBatchNormTdnnLayer(512,512,nonlinearity=nonlinearity)
self.tdnn5 = ReluBatchNormTdnnLayer(512,1500,nonlinearity=nonlinearity)
self.stats = StatisticsPooling(1500, stddev=True)
self.tdnn6 = ReluBatchNormTdnnLayer(self.stats.get_output_dim(),512,nonlinearity=nonlinearity)
self.tdnn7 = ReluBatchNormTdnnLayer(512,512,nonlinearity=nonlinearity)
self.loss = SoftmaxLoss(512, num_targets)
```

运行 print(Xvector(23, 1211))，可以输出完整配置：

```
Xvector(
  (aug_dropout): Dropout2d(p=0.2, inplace=False)
  (tdnn1): ReluBatchNormTdnnLayer(
    (affine): TdnnAffine(23, 512, context=[-2, -1, 0, 1, 2], bias=True,
stride=1, pad=True, groups=1, norm_w=False, norm_f=False)
    (activation): ReLU(inplace=True)
    (batchnorm): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  )
```

```
  (tdnn2): ReluBatchNormTdnnLayer(
    (affine): TdnnAffine(512, 512, context=[-2, 0, 2], bias=True, stride=1,
pad=True, groups=1, norm_w=False, norm_f=False)
    (activation): ReLU(inplace=True)
    (batchnorm): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  )
  (tdnn3): ReluBatchNormTdnnLayer(
    (affine): TdnnAffine(512, 512, context=[-3, 0, 3], bias=True, stride=1,
pad=True, groups=1, norm_w=False, norm_f=False)
    (activation): ReLU(inplace=True)
    (batchnorm): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  )
  (tdnn4): ReluBatchNormTdnnLayer(
    (affine): TdnnAffine(512, 512, context=[0], bias=True, stride=1,
pad=True, groups=1, norm_w=False, norm_f=False)
    (activation): ReLU(inplace=True)
    (batchnorm): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  )
  (tdnn5): ReluBatchNormTdnnLayer(
    (affine): TdnnAffine(512, 1500, context=[0], bias=True, stride=1,
pad=True, groups=1, norm_w=False, norm_f=False)
    (activation): ReLU(inplace=True)
    (batchnorm): BatchNorm1d(1500, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  )
  (stats): StatisticsPooling(1500, 3000, stddev=True, unbiased=False,
eps=1e-10)
  (tdnn6): ReluBatchNormTdnnLayer(
    (affine): TdnnAffine(3000, 512, context=[0], bias=True, stride=1,
pad=True, groups=1, norm_w=False, norm_f=False)
    (activation): ReLU(inplace=True)
    (batchnorm): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  )
  (tdnn7): ReluBatchNormTdnnLayer(
```

```
    (affine): TdnnAffine(512, 512, context=[0], bias=True, stride=1,
pad=True, groups=1, norm_w=False, norm_f=False)
    (activation): ReLU(inplace=True)
    (batchnorm): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  )
  (loss): SoftmaxLoss(
    (affine): TdnnAffine(512, 1211, context=[0], bias=True, stride=1,
pad=True, groups=1, norm_w=False, norm_f=False)
    (loss_function): CrossEntropyLoss()
  )
)
```

3. 2 训练框架

本小节通过代码的上层、中层和底层这三层的关系来逐层介绍 ASV-Subtools 中的训练框架结构，同时详细介绍其中所包含的重要部分。

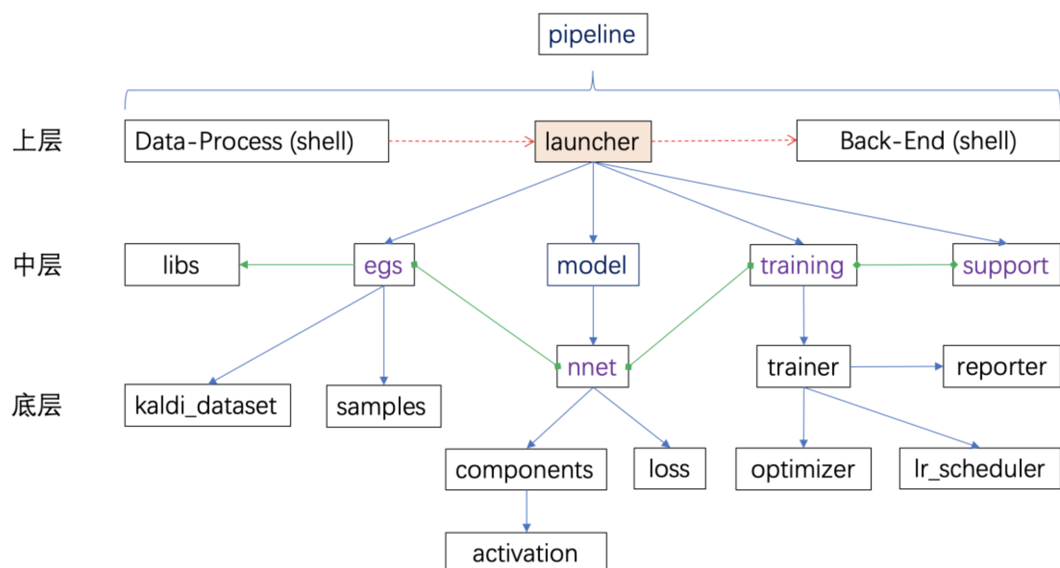


图 3.5 ASV-Subtools 框架结构

ASV-Subtools 框架结构如图 3.5 所示。首先，最上层为训练流程，包含数据预处理、训练启动器（Launcher）和后端三个部分。数据处理和后端使用 Kaldi 工具并基于 shell 脚本进行构建，前者包含 Kaldi 格式的数据映射目录准备、特征提

取和特征处理等部分，后者包含 x-vector 提取，后端打分和指标计算等部分。而训练启动器则主要用来衔接数据预处理和后端并进行模型的训练。

启动器先预处理数据（如 VAD）并对数据进行采样，然后进行模型的训练，最后调用后端的 x-vector 提取脚本来使用多进程提取 x-vector 以供后端进行后续操作。其中，启动器使用 Python 构建而不是 Shell，这是因为 Pytorch 提供了大量 Python 接口，因此可能会产生非常复杂的个性化的配置，如果使用 Shell 构建启动器并通过传参方式来调用训练代码，则存在参数封装的麻烦，从而丢失了使用上的灵活性。因此，基于 Python 构建的启动器既可保留训练模型时从头到尾执行完所有流程的简便，也保证了配置和使用 Pytorch 原生接口的灵活。

其次，框架中层主要是训练相关的高层组件，均包含在训练框架的核心库 libs 中，主要包含样本（egs）、模型（model）、训练（training）和额外支持工具（support），其中，模型可基于模型组件（nnet）进行配置，也可直接使用 Pytorch 提供的原生组件（torch.nn）进行构建，但在 ASV-Subtools 中，为了更好的模块分化（如模型定义和训练分开）并实现诸多功能（如 x-vector 提取），模型不再是 Pytorch 中的 torch.nn.module 的直接继承，而是使用 ASV-Subtools 中的虚拟对象 TopVirtualNnet 间接继承。由于模型是关键对象，训练模型仅仅是为模型的参数调整服务的，因此，在 ASV-Subtools 中，模型被设计为一个单独的 Python 文件对象，通过 Python 模块即时导入技术，模型不仅可以直接放在训练模块进行训练，还可以方便的用于 x-vector 提取，训练和提取两个单独模块之间仅通过模型产生连接，没有代码上的限制，这保证了在使用其他模块代码的时候可以高度复用而无需因为每次模型不同频繁修改。由于即时导入时，Python 创建对象需要对象的名称以及初始化参数，而这些都可能有各种各样的形式，为了更加灵活方便，ASV-Subtools 在即时导入技术中设计了两个概念：

（1）Model_blueprint：模型蓝图，即模型 Python 文件路径，在该文件中，可任意定义一个或多个模型，如定义一个名为 Xvector 模型对象。

（2）Creation：创建命令，基于 Model_blueprint 的定义，实例化对象时的代码字符串，如 Xvector(input_dim=23,speakers=1211)，其中 Creation 包含了一整句的创建命令，该命令实际可在 Python 环境中直接执行。考虑到作为参数的 Creation 其可能包含字符的复杂性，如空格，因此继承 TopVirtualNnet 的模型被实例化时，Creation 将会被自动收集和保存，并方便用于之后的特征提取。

最后，框架的底层为基于 Python 实现的各个基本对象，如对应到 Kaldi 映射目录的 kaldi_dataset，采样方法 samples，模型基本组件 components、activation 和 loss，训练有关的训练流程 trainer，训练进度显示 reporter，优化器以及学习率打包配置等。具体的，samples 主要提供两个主要采样方法，一是直接将训练集完全按等长进行切割，二是在切割样本的时候同时考虑说话人样本的平衡性。训练流程 trainer 目前支持单输入单输出的常规模型训练和多输入多输出的多任务（Multi-task）网络训练等，其中，对说话人和音素两个任务同时进行分类的多任务网络训练来说，ASV-Subtools 还额外提供相应脚本进行处理，先通过 Kaldi 语音模型强制对齐来获得音素标签，然后再组织好相关样本送入 Pytorch 进行训练。ASV-Subtools 同时也支持多 GPU 训练。神经网络组件 components 则主要包含构建模型隐藏层的组件，如基本的 TDNN、Statistics Pooling、LDE，自定义 Dropout，Attention 等网络层和 ReLU-BatchNorm-Layer 等套件层。而 Loss 则包含常用的 CE（Softmax）、AM-Softmax、AAM-Softmax 等损失函数。除此之外，support 中包含 GPU 自动选取支持模块、衔接 Kaldi 数据格式（如 ark）的 I/O 模块、加速训练时数据读取的支持模块、画图模块以及其他常用的函数包 utils 等模块。

总的来说，ASV-Subtools 通过各个模块之间的明确分工和联系来组合成整体框架，这不仅非常有利于实验效率的提高，而且非常方便上手和改造。同时，由于其数据处理主要基于 Pytorch、Numpy 和 Pandas，总依赖较少，因此也易于不同环境下的快速部署使用。

第 4 章 环境配置

4.1 Kaldi 安装

Subtools 包含 Kaldi 本身提供的脚本和基于 Kaldi 封装的很多实用、高效的脚本，因此安装 subtools 前需要进行 Kaldi 的安装。Kaldi 的安装采用源码编译安装的方式，可以参照 Kaldi 官方文档[2]进行安装，也可以参照以下步骤进行安装：

1、选取安装路径，本案例源路径为/work。将 Kaldi 源码从 github 上克隆至当前路径（/work 下），并进入 Kaldi 目录：

```
git clone https://github.com/kaldi-asr/kaldi.git
```

2、编译 tools

```
cd kaldi/tools
extras/check_dependencies.sh
make -j 4
```

3、编译 kaldi

```
cd ../src
./configure --shared
make depend -j 8
make -j 8
```

4、检查是否安装成功，输出显示如下即为安装成功：

```
cd ../egs/yesno/s5
sh run.sh
```

[2] http://www.kaldi-asr.org/doc/install.html

```
0.5342 -0.000422432
HCLGa is not stochastic
add-self-loops --self-loop-scale=0.1 --reorder=true exp/mono0a/final.mdl exp/mono0a/graph_tgpr/HCLGa.fst
steps/decode.sh --nj 1 --cmd utils/run.pl exp/mono0a/graph_tgpr data/test_yesno exp/mono0a/decode_test_yesno
decode.sh: feature type is delta
steps/diagnostic/analyze_lats.sh --cmd utils/run.pl exp/mono0a/graph_tgpr exp/mono0a/decode_test_yesno
steps/diagnostic/analyze_lats.sh: see stats in exp/mono0a/decode_test_yesno/log/analyze_alignments.log
Overall, lattice depth (10,50,90-percentile)=(1,1,2) and mean=1.2
steps/diagnostic/analyze_lats.sh: see stats in exp/mono0a/decode_test_yesno/log/analyze_lattice_depth_stats.log
local/score.sh --cmd utils/run.pl data/test_yesno exp/mono0a/graph_tgpr exp/mono0a/decode_test_yesno
local/score.sh: scoring with word insertion penalty=0.0,0.5,1.0
%WER 0.00 [ 0 / 232, 0 ins, 0 del, 0 sub ] exp/mono0a/decode_test_yesno/wer_10_0.0
```

https://blog.csdn.net/qq_40168949

图 4.1 Kaldi 样例输出

或者检查生成的文件是否存在

```
(base) [root@localhost s5]# ls /work/kaldi/src/bin/
acc-lda                   build-tree.o                    copy-post.cc                 logprob-to-post                 prons-to-wordali.cc
acc-lda.cc                build-tree-two-level            copy-post.o                  logprob-to-post.cc              prons-to-wordali.o
acc-lda.o                 build-tree-two-level.cc         copy-transition-model        logprob-to-post.o               scale-post
acc-tree-stats            build-tree-two-level.o          copy-transition-model.cc     Makefile                        scale-post.cc
acc-tree-stats.cc         cluster-phones                  copy-transition-model.o      make-h-transducer               scale-post.o
acc-tree-stats.o          cluster-phones.cc               copy-tree                    make-h-transducer.cc            show-alignments
add-self-loops            cluster-phones.o                copy-tree.cc                 make-h-transducer.o             show-alignments.cc
add-self-loops.cc         compare-int-vector              copy-tree.o                  make-ilabel-transducer          show-alignments.o
add-self-loops.o          compare-int-vector.cc           copy-vector                  make-ilabel-transducer.cc       show-transitions
align-compiled-mapped     compare-int-vector.o            copy-vector.cc               make-ilabel-transducer.o        show-transitions.cc
align-compiled-mapped.cc  compile-graph                   copy-vector.o                make-pdf-to-tid-transducer      show-transitions.o
align-compiled-mapped.o   compile-graph.cc                decode-faster                make-pdf-to-tid-transducer.cc   sum-lda-accs
align-equal               compile-graph.o                 decode-faster.cc             make-pdf-to-tid-transducer.o    sum-lda-accs.cc
align-equal.cc            compile-questions               decode-faster-mapped         matrix-dim                      sum-lda-accs.o
align-equal-compiled      compile-questions.cc            decode-faster-mapped.cc      matrix-dim.cc                   sum-matrices
align-equal-compiled.cc   compile-questions.o             decode-faster-mapped.o       matrix-dim.o                    sum-matrices.cc
align-equal-compiled.o    compile-train-graphs            decode-faster.o              matrix-sum                      sum-matrices.o
align-equal.o             compile-train-graphs.cc         draw-tree                    matrix-sum.cc                   sum-mllt-accs
align-mapped              compile-train-graphs-fsts       draw-tree.cc                 matrix-sum.o                    sum-mllt-accs.cc
align-mapped.cc           compile-train-graphs-fsts.cc    draw-tree.o                  matrix-sum-rows                 sum-mllt-accs.o
align-mapped.o            compile-train-graphs-fsts.o     est-lda                      matrix-sum-rows.cc              sum-post
align-text                compile-train-graphs.o          est-lda.cc                   matrix-sum-rows.o               sum-post.cc
align-text.cc             compute-gop                     est-lda.o                    phones-to-prons                 sum-post.o
align-text.o              compute-gop.cc                  est-mllt                     phones-to-prons.cc              sum-tree-stats
ali-to-pdf                compute-gop.o                   est-mllt.cc                  phones-to-prons.o               sum-tree-stats.cc
ali-to-pdf.cc             compute-wer                     est-mllt.o                   post-to-pdf-post                sum-tree-stats.o
ali-to-pdf.o              compute-wer-bootci              est-pca                      post-to-pdf-post.cc             transform-vec
ali-to-phones             compute-wer-bootci.cc           est-pca.cc                   post-to-pdf-post.o              transform-vec.cc
ali-to-phones.cc          compute-wer-bootci.o            est-pca.o                    post-to-phone-post              transform-vec.o
ali-to-phones.o           compute-wer.cc                  get-post-on-ali              post-to-phone-post.cc           tree-info
```

图 4.2 Kaldi 生成文件

4.2 Subtools 安装

1、构建工程目录，这里推荐构建四级目录，以 Kaldi 为第一级目录开始计算，在上节中已经将 Kaldi 安装至/work 下，因此四级目录结构为/work/kaldi/egs/xmuspeech/example。其中[xmuspeech]为自定义目录，[example]为工程目录。通过 shell 命令，创建工程目录。

```
mkdir -p /work/kaldi/egs/xmuspeech/example
```

2、从 github 下克隆 subtools。

```
cd /work/kaldi/egs/xmuspeech/example
git clone https://github.com/Snowdar/asv-subtools.git subtools
```

4.3 环境配置

为了方便系统环境的管理，解决多版本 python 并存、切换以及各种第三方

包安装问题，这里推荐使用 anaconda 配置环境，首先需要安装 anaconda3。

1、安装包可以直接在官网下载，但国内速度较慢可以换成清华镜像源[3]加快速度。在清华镜像站上找到最近日期的版本，选择一个对应自己系统版本的 Anaconda3 安装包，x86_64 表示兼容 32 位和 64 位系统，右键复制链接，在服务器上中使用 wget 下载，安装包会下载至当前目录下：

```
wget https://mirrors.tuna.tsinghua.edu.cn/anaconda/archive/Anaconda3-2021.11-Linux-x86_64.sh
```

mirrors.tuna.tsinghua.edu.cn/anaconda/archive/?C=M&O=D

Parent directory/	-	-
Anaconda3-2021.11-Windows-x86_64.exe	510.3 MiB	2021-11-18 02:14
Anaconda3-2021.11-Windows-x86.exe	404.1 MiB	2021-11-18 02:14
Anaconda3-2021.11-MacOSX-x86_64.sh	508.4 MiB	2021-11-18 02:14
Anaconda3-2021.11-MacOSX-x86_64.pkg	515.1 MiB	2021-11-18 02:14
Anaconda3-2021.11-Linux-x86_64.sh	580.5 MiB	2021-11-18 02:14
Anaconda3-2021.11-Linux-s390x.sh	241.7 MiB	2021-11-18 02:14
Anaconda3-2021.11-Linux-ppc64le.sh	254.9 MiB	2021-11-18 02:14
Anaconda3-2021.11-Linux-aarch64.sh	487.7 MiB	2021-11-18 02:14
Anaconda3-2021.05-Windows-x86_64.exe	477.2 MiB	2021-05-14 11:34
Anaconda3-2021.05-Windows-x86.exe	408.5 MiB	2021-05-14 11:34

图 4.3 anaconda 安装包

2、sh Anaconda3-2021.11-Linux-x86_64.sh，安装过程中会出现一堆 license 许可声明，一直回车向下，出现如下文字输入 yes：

```
Do you accept the license terms? [yes|no]
[no] >>>
Please answer 'yes' or 'no':
>>> yes
```

图 4.4 anaconda 安装过程提示

之后需要选择安装目录，若无需更改直接回车，默认安装路径为 /root/anaconda3。等待一会儿后有选项，问是否需要进行 conda 的初始化，建议输入选择 yes，会在/root/.bashrc 目录中自动添加环境变量，会使得开机自动启动 base 环境。看到如下提示则代表环境安装完成：

[3] https://mirrors.tuna.tsinghua.edu.cn/anaconda/archive/

```
Thank you for installing Anaconda3!

===========================================================================

Anaconda and JetBrains are working together to bring you Anaconda-powered
environments tightly integrated in the PyCharm IDE.

PyCharm for Anaconda is available at:
https://www.anaconda.com/pycharm
```

图 4.5 anaconda 安装完成

3、简单测试是否安装完成：

```
conda activate # 进入 conda 环境 出现(base)则说明安装成功
conda deactivate # 退出 conda 环境
```

安装完成 anaconda 之后，通过配置 subtools 使用的环境，避免与系统其他工程的环境造出冲突，这里以 python3.8 为例：

1、首先创建新环境。

```
conda create -n subtools_env python=3.8
```

构建过程中按提示输入 y，等待一会就可以完成基础环境构建，并通过命令激活环境。

```
#
# To activate this environment, use
#
#     $ conda activate subtools_env
#
# To deactivate an active environment, use
#
#     $ conda deactivate

(base) [root@localhost example]# conda activate subtools_env
(subtools_env) [root@localhost example]#
```

图 4.6 subtools_env 环境创建与激活

2、安装 Pytorch，这里默认安装最新版 Pytorch，经过等待即可完成安装。

```
pip install torch
```

3、安装依赖库，相关依赖库的列表在 subtools/requirements.txt 有列出，通过 pip 命令完成安装即可。

```
pip install -r subtools/requirements.txt
```

4、为 subtools 配置 Kaldi 路径。若 subtools 的路径按上述步骤在第四级目录，例如/work/kaldi/egs/xmuspeech/example/，则无需更改 path 路径，否则需要编辑 subtools/path.sh 配置 KALDI_ROOT，具体要求可按 subtools/path.sh 内容进行修改。

以上，就已完成 subtools 的环境安装。

第 5 章 入门使用—Voxceleb1 训练和测试

5.1 数据介绍

VoxCeleb[4]是一个英文数据集，包括两个子集 VoxCeleb1 和 Voxceleb2。该数据集的全部音频数据都是从视频网站 Youtube 上的采访视频上截取采集而来，音频内容来源于真实的完全自由的场景对话，如社会名人演讲、大型体育赛事讲解、真人访谈节目等等，属于真实场景下的文本无关数据集。其中 Voxceleb1 共有 1251 个说话人，共计约 15 万条语音，语音平均时长在 8 秒左右，时长超过 340 小时，覆盖了各种族、职业、口音和年龄，并且具有较好的性别平衡（55%男性，45%女性）。

VoxCeleb1 可以分为 dev 和 test 两个集，可以分别用于训练和测试。

The VoxCeleb1 Dataset

VoxCeleb1 contains over 100,000 utterances for 1,251 celebrities, extracted from videos uploaded to YouTube.

Verification split

	dev	test
# of speakers	1,211	40
# of videos	21,819	677
# of utterances	148,642	4,874

Identification split

	dev	test
# of speakers	1,251	1,251
# of videos	21,245	1,251
# of utterances	145,265	8,251

图 5.1 VoxCeleb1 数据划分

5.2 数据下载

数据下载可以使用两种方式，一种是直接在本地下载后上传至服务器，另一个是在服务器上利用 wget 命令进行下载。

[4] https://www.robots.ox.ac.uk/~vgg/data/voxceleb/index.html#about

5.3 数据处理

在数据下载完成之后需要准备数据映射文件，分别为 wav.scp、utt2spk、spk2utt，这三种文件的格式分别为：

```
wav.scp: utt-id utt-path
utt2spk: utt-id spk-id
spk2utt: spk-id utt-id
```

由于数据存放位置的不同，可以选择构建 python 脚本或利用 shell 命令进行以上三个文件的生成。以数据存放位置为/data1/voxceleb1/dev，结合 shell 命令为例，这里提供一种生成以上三个文件的范例：

1、创建数据文件夹，在第 4 章中构建工程的路径为/work/kaldi/egs/xmuspeech/example，通过 shell 命令在 example 下构建 data 目录用于存放数据映射文件。

```
mkdir -p data/voxceleb1_train
```

2、利用 find 命令获取该文件夹下所有 wav 文件的绝对路径并保存至 temp.lst 中。

```
find /data1/voxceleb1/dev -name *.wav > data/ voxceleb1_train/temp.lst
```

此时 temp.lst 文件里保存了所有 dev 的语音的绝对路径，例：/data1/voxceleb1/dev/id10001/1zclwhmdeo4/00001.wav

```
awk '{split($1,a,"/");{split(a[7],b,”.”)}print a[5]"-"a[6]"-"b[1],$1}'
data/ voxceleb1_train/temp.list > data/ voxceleb1_train/wav.scp
```

利用 awk 命令将 temp.lst 中的每一行进行切分并重新组合构建 wav.scp，输出格式为：

```
id0001-1zcIwhmdeo4-00001 /data1/voxceleb1_dev/id10001/1zcIwhmdeo4/00001.wav
```

3、同样利用 awk 命令生成 utt2spk 和 spk2utt 两个文件。

```
awk '{split($1,a,"/");{split(a[7],b, ".")};print a[5]"-"a[6]"-"b[1],a[5]}'
data/ voxceleb1_train/temp.lst > data/ voxceleb1_train/utt2spk
awk '{split($1,a,"/");{split(a[7],b,”.”)}print a[5], a[5]"-"a[6]"-"b[1]}'
data/ voxceleb1_train/temp.lst > data/ voxceleb1_train/spk2utt
```

格式分别为：

```
utt2spk: id0001-1zcIwhmdeo4-00001 id0001
spk2utt: id0001 id0001-1zcIwhmdeo4-00001
```

至此，就生成 Voxceleb1 训练集的三个映射文件。但是此时的三个文件的顺序并不是有序的，因此需要对文件夹进行过滤和筛选，利用命令完成：

```
sh subtools/kaldi/utils/fix_data_dir.sh data/ voxceleb1_train
```

以上命令针对不同的数据集，只需根据数据存放路径简单修改即可完成三个映射文件的生成。

特别地，对于 Voxceleb 数据集，subtools 集成了 perl 语言的脚本用于生成上述三个映射文件，脚本存放于 subtools/recipe/voxceleb/prepare 中，还是以 Voxceleb1 数据存放于/data1/voxceleb1 中为例，构建训练集只需执行命令：

```
subtools/recipe/voxceleb/prepare/make_voxceleb1_v2.pl /data1/voxceleb1/dev
dev data/voxceleb1_train
```

而对于测试集，同样执行一下命令，就能完成映射文件的生成以及测试列表 trials。

```
subtools/recipe/voxceleb/prepare/make_voxceleb1_v2.pl /data1/voxceleb1/test
test data/voxceleb1_test
```

5.4 特征提取

在完成数据映射文件之后，需要进行特征提取，这里的特征为离线 Kaldi 特征，提取特征的方式与 Kaldi 相同，subtools 提供了封装后的脚本用于特征提取。

进行特征提取时，首先需要建立配置特征的配置文件，在 subtools/conf 目录下提供了常见的一些特征及其参数配置，这里使用 23 维的 MFCC+pitch 作为模型的输入。需要注意的是，由于同一个数据集往往会提取不同特征进行实验，为了区分不同的特征同时方便后续训练脚本的调用，需要在 data 目录下为不同特征新建目录，这里通过执行命令：

```
subtools/newCopyData.sh mfcc_23_pitch "voxceleb1_train voxceleb1_test"
```

就能将 voxceleb1_train 和 voxceleb1_test 拷贝至 data/mfcc_23_pitch 下。因

此提取特征的命令为：

```
subtools/makeFeatures.sh --pitch true --pitch-config
subtools/conf/pitch.conf  data/mfcc_23_pitch/voxceleb1_train mfcc
subtools/conf/ sre-mfcc-23.conf
```

特征提取完成之后会在 data/mfcc_23_pitch/voxceleb1_train 下获得 feats.scp，里面的格式为 utt-id feats-ark-path，其中 feats-ark-path 为 Kaldi 特征存放的二进制文件的路径。

之后可对数据进行 VAD 计算，去除静音帧，通过命令完成计算并获得 vad.scp。

```
subtools/computeVad.sh data/mfcc_23_pitch/voxceleb1_train
subtools/conf/vad-5.0.conf
```

相同地，需要对测试数据进行特征提取与 VAD 计算。

5.5 数据加噪

数据扩增是深度学习中常用的技巧之一，主要是增加训练数据集，让训练集尽可能地多样化，学习到更多的信息，使得训练的模型具有更强的泛化能力。在语音领域，常见的数据扩增方法就是给原始数据增加噪声，在这里我们利用 Kaldi 工具进行数据扩充。具体的命令为：

```
sh subtools/augmentDataByNoise.sh data/mfcc_23_pitch/voxceleb_dev
```

加噪后的文件夹会存放于 data/mfcc_23_pitch/augment 下，如下图所示：

图 5.2 加噪数据集

其中后缀名为_babble、_music、_noise、_reverb 的文件夹为四种噪声扩充的文件夹，以及四倍扩充后的文件夹。

后续处理时，仅需要对所需的噪声文件夹进行加噪处理即可，在这里采用扩充四倍噪声的方案，因此运行

```
subtools/makeFeatures.s –pitch true --pitch-config subtools/conf/pitch.conf
data/mfcc_23_pitch/augment/voxceleb1_train_reverb_noise_music_babble mfcc
subtools/conf/sre-mfcc-23.conf
```

需注意的是，在这里并不需要重新计算 VAD 标签，由于噪声扩充之后噪声会影响 VAD 的计算，若重新计算加噪后的语音的 VAD 标签将无法去除原始静音部分，因此一般会利用未扩充数据计算出的 VAD 标签替换加噪后语音的 VAD 标签（不需要重复计算加噪语音的 VAD 标签）。

进一步的通过命令，即可合并原始数据集与噪声数据集。

```
subtools/kaldi/utils/combine_data.sh data/mfcc_23_pitch/voxceleb1_train_aug
data/mfcc_23_pitch/augment/voxceleb1_train_reverb_noise_music_babble/
data/mfcc_23_pitch/voxceleb1_train/
```

至此，模型训练前的数据集处理就已经完成，文件夹 data/mfcc_23_pitch/voceleb1_train_aug 即为最终的训练集。

这里需要指出的是，加噪与提取特征的顺序并不固定，这里采用的是先提取干净特征在加噪，最后合并最终训练集的顺序。如果先进行加噪，合并训练集最后统一提取特征，可以通过下述命令完成干净标签对加噪数据 VAD 标签的替换：

```
subtools/computeAugmentedVad.sh data/mfcc_23_pitch/voxceleb1_train_aug
data/mfcc_23_pitch/voxceleb1_train/utt2spk subtools/conf/vad-5.0.conf
```

5.6 模型训练

针对 Voxceleb1 的训练与测试，subtools 提供了专门的复现脚本，存放于 subtools/recipe/voxceleb 下，如下图所示。这里以简单的 TDNN-xvector 为例。

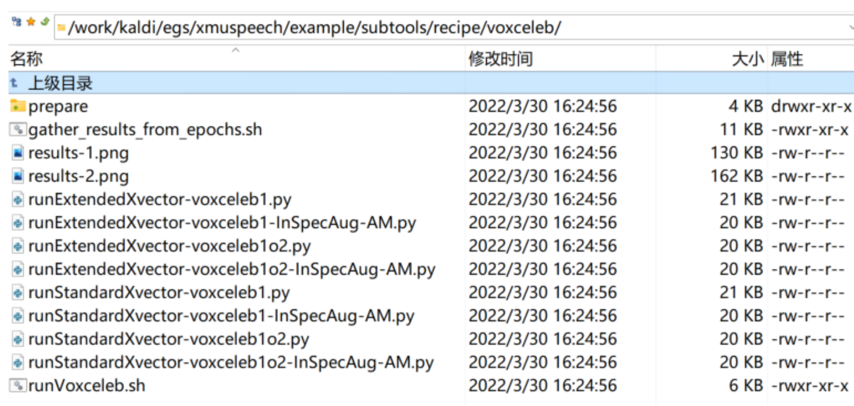


图 5.3 subtools 模型启动脚本列表

1、拷贝 runStandardXvector-voxceleb1.py 至当前目录下。

```
cp subtools/pytorch/recipe/runStandardXvector-voxceleb1.py ./
```

2、打开 runStandardXvector-voxceleb1.py 并修改脚本参数。

运行脚本可以分为五步，其中前三步为样本准备阶段，第四步为训练模型结构，第五步为提取 x-vector 阶段。

在启动器脚本中，可以看到前三步调用的脚本均为 subtools/pytorch/pipeline/preprocess_to_egs.sh，目的是分步执行 vad-cmn、删除不符合条件的语音、获取块样本（chunk egs），这几步与 Kaldi 训练模型前的样本准备一致。

```
#### Preprocess
if stage <= 2 and endstage >= 0:
    # Here only give limited options because it is not convenient.
    # Suggest to pre-execute this shell script to make it freedom and then continue to run this launcher.
    kaldi_common.execute_command("bash subtools/pytorch/pipeline/preprocess_to_egs.sh "
                                 "--stage {stage} --endstage {endstage} --valid-split-type
                                 {valid_split_type} "
                                 "--nj {nj} --cmn {cmn} --limit-utts {limit_utts} --min-chunk {chunk_size}
                                 --overlap {overlap} "
                                 "--sample-type {sample_type} --chunk-num {chunk_num} --scale {scale}
                                 --force-clear {force_clear} "
                                 "--valid-num-utts {valid_utts} --valid-chunk-num
                                 {valid_chunk_num_every_utt} --compress {compress} "
                                 "{traindata} {egs_dir}".format(stage=stage, endstage=endstage,
                                 valid_split_type=valid_split_type,
                                 nj=preprocess_nj, cmn=str(cmn).lower(), limit_utts=limit_utts, chunk_size=
                                 chunk_size, overlap=overlap,
                                 sample_type=sample_type, chunk_num=chunk_num, scale=scale, force_clear=str
                                 (force_clear).lower(),
                                 valid_utts=valid_utts, valid_chunk_num_every_utt=valid_chunk_num_every_utt
                                 , compress=str(compress).lower(),
                                 traindata=traindata, egs_dir=egs_dir))
```

图 5.4 启动器脚本 preprocess 代码

这一步的参数控制模块在启动器代码 88-104 行，相关参数的含义在代码中均有解释，可通过 subtools 源码进行查看，调用 preprocess_to_egs.sh 脚本需要

提供训练集路径以及存放 egs 路径，由于之前的数据准备过程时，相关文件夹的名称与样例相同，因此并不需要进行修改。

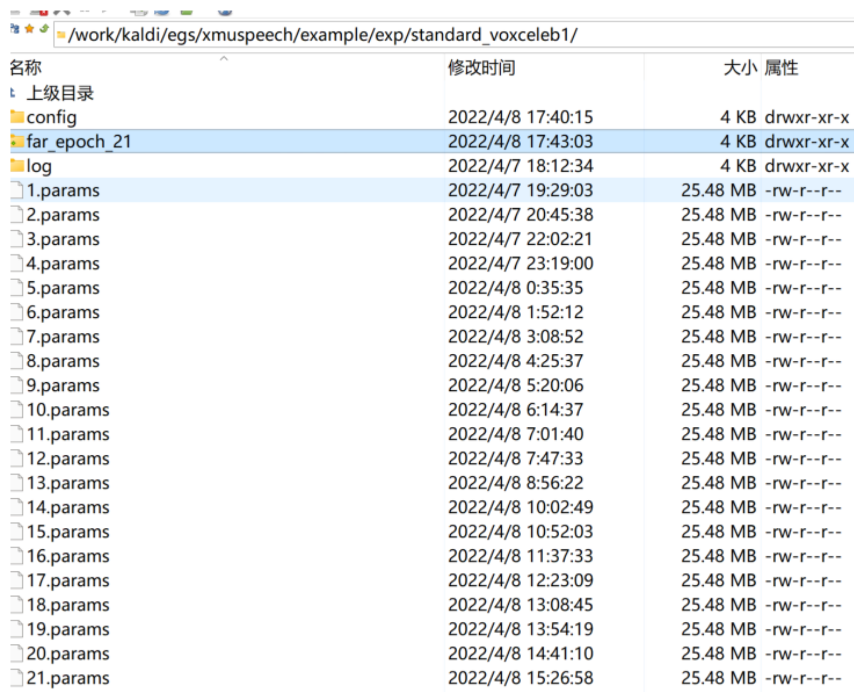
```
##--------------------------------------------------##
## Main params
traindata="data/mfcc_23_pitch/voxceleb1_train_aug"
egs_dir="exp/egs/mfcc_23_pitch_voxceleb1_train_aug" + "_" + sample_type

model_blueprint="subtools/pytorch/model/snowdar-xvector.py"
model_dir="exp/standard_voxceleb1"
##--------------------------------------------------##
```

图 5.5 启动器脚本数据集路径设置代码

因此可以直接运行脚本：

```
python3 runStandardXvector-voxceleb1.py -stage 0 -endstage 4
```

经过等待之后模型训练完成会存储在 model_dir 中，如下图所示：



/work/kaldi/egs/xmuspeech/example/exp/standard_voxceleb1/

名称	修改时间	大小	属性
上级目录			
config	2022/4/8 17:40:15	4 KB	drwxr-xr-x
far_epoch_21	2022/4/8 17:43:03	4 KB	drwxr-xr-x
log	2022/4/7 18:12:34	4 KB	drwxr-xr-x
1.params	2022/4/7 19:29:03	25.48 MB	-rw-r--r--
2.params	2022/4/7 20:45:38	25.48 MB	-rw-r--r--
3.params	2022/4/7 22:02:21	25.48 MB	-rw-r--r--
4.params	2022/4/7 23:19:00	25.48 MB	-rw-r--r--
5.params	2022/4/8 0:35:35	25.48 MB	-rw-r--r--
6.params	2022/4/8 1:52:12	25.48 MB	-rw-r--r--
7.params	2022/4/8 3:08:52	25.48 MB	-rw-r--r--
8.params	2022/4/8 4:25:37	25.48 MB	-rw-r--r--
9.params	2022/4/8 5:20:06	25.48 MB	-rw-r--r--
10.params	2022/4/8 6:14:37	25.48 MB	-rw-r--r--
11.params	2022/4/8 7:01:40	25.48 MB	-rw-r--r--
12.params	2022/4/8 7:47:33	25.48 MB	-rw-r--r--
13.params	2022/4/8 8:56:22	25.48 MB	-rw-r--r--
14.params	2022/4/8 10:02:49	25.48 MB	-rw-r--r--
15.params	2022/4/8 10:52:03	25.48 MB	-rw-r--r--
16.params	2022/4/8 11:37:33	25.48 MB	-rw-r--r--
17.params	2022/4/8 12:23:09	25.48 MB	-rw-r--r--
18.params	2022/4/8 13:08:45	25.48 MB	-rw-r--r--
19.params	2022/4/8 13:54:19	25.48 MB	-rw-r--r--
20.params	2022/4/8 14:41:10	25.48 MB	-rw-r--r--
21.params	2022/4/8 15:26:58	25.48 MB	-rw-r--r--

图 5.6 模型输出文件夹

5.7 模型测试

在 5.1 节中有介绍，Voxceleb1 数据总共有 1251 人，其实 dev（训练集）为 1211 人，之前已经用于模型训练，剩下的 test 即为测试集。在上一步训练时，runStandardXvector-voxceleb1.py 脚本已经集成 x-vector 的提取步骤，如下图所

示：

```
#### Extract xvector
if stage <= 4 <= endstage and utils.is_main_training():
    # There are some params for xvector extracting.
    data_root = "data" # It contains all dataset just like Kaldi recipe.
    prefix = "mfcc_23_pitch" # For to_extracted_data.

    to_extracted_positions = ["far", "near"] # Define this w.r.t extracted_embedding param of model_blueprint.
    to_extracted_data = ["voxceleb1_train_aug", "voxceleb1_test"] # All dataset should be in data_root/prefix.
    to_extracted_epochs = ["21"] # It is model's name, such as 10.params or final.params (suffix is w.r.t package).

    nj = 10
    force = False
    use_gpu = True
    gpu_id = ""
    sleep_time = 10
```

图 5.7 x-vector 提取代码

这里提取了训练集 voxceleb1_train_aug 以及测试集 voxceleb1_test 的 x-vector，提取的 epoch 为第 21 个。查看文件夹，可以看到这里存储的 x-vector 格式与 Kaldi 的方式相同，都是以 xvector.scp 以及相应的 ark 文件存储。

/work/kaldi/egs/xmuspeech/example/exp/standard_voxceleb1/far_epoch_21/voxceleb1_train_aug/

名称	修改时间	大小	属性
上级目录			
log	2022/4/8 15:29:05	4 KB	drwxr-xr-x
xvector.1.ark	2022/4/8 15:49:48	148.09 MB	-rw-r--r--
xvector.1.scp	2022/4/8 15:49:48	7.83 MB	-rw-r--r--
xvector.2.ark	2022/4/8 15:49:58	148.08 MB	-rw-r--r--
xvector.2.scp	2022/4/8 15:49:58	7.83 MB	-rw-r--r--
xvector.3.ark	2022/4/8 15:50:05	148.08 MB	-rw-r--r--
xvector.3.scp	2022/4/8 15:50:05	7.83 MB	-rw-r--r--
xvector.4.ark	2022/4/8 15:50:12	148.08 MB	-rw-r--r--
xvector.4.scp	2022/4/8 15:50:12	7.83 MB	-rw-r--r--
xvector.5.ark	2022/4/8 15:50:13	148.07 MB	-rw-r--r--
xvector.5.scp	2022/4/8 15:50:13	7.83 MB	-rw-r--r--
xvector.6.ark	2022/4/8 15:50:17	148.07 MB	-rw-r--r--
xvector.6.scp	2022/4/8 15:50:17	7.83 MB	-rw-r--r--
xvector.7.ark	2022/4/8 15:50:19	148.07 MB	-rw-r--r--
xvector.7.scp	2022/4/8 15:50:19	7.83 MB	-rw-r--r--
xvector.8.ark	2022/4/8 15:38:44	148.08 MB	-rw-r--r--
xvector.8.scp	2022/4/8 15:38:44	7.83 MB	-rw-r--r--
xvector.9.ark	2022/4/8 15:38:46	148.08 MB	-rw-r--r--
xvector.9.scp	2022/4/8 15:38:46	7.83 MB	-rw-r--r--
xvector.10.ark	2022/4/8 15:50:24	148.09 MB	-rw-r--r--
xvector.10.scp	2022/4/8 15:50:24	7.90 MB	-rw-r--r--
xvector.scp	2022/4/8 15:50:25	78.38 MB	-rw-r--r--

图 5.8 xvector 存储格式

x-vector 提取完成就可以进行测试，这里通过命令完成打分及结果计算：

```
subtools/recipe/voxceleb/gather_results_from_epochs.sh --vectordir
exp/standard_voxceleb1 --epochs "21" --score plda --score-norm false
```

这里采用 PLDA 打分的方式，脚本默认会先进行 LDA 将 x-vector 维度降至 256 维，后进行 PLDA 打分，具体结果如下：

```
compute-eer -
LOG (compute-eer[5.5]:main():compute-eer.cc:136) Equal error rate is 3.07529%, at threshold -3.8252
EER% 3.075

[ voxceleb1_test ]
3.075   exp/standard_voxceleb1//far_epoch_21/voxceleb1_test/score/plda_voxceleb1_enroll_voxceleb1_test_lda256_norm_voxceleb1_tr
in_aug.eer
```

图 5.9 voxceleb1 样例结果

至此，Voxceleb1 数据集的入门训练及测试就已经完成！

第 6 章 进阶研究—多任务/迁移学习

6.1 多任务学习

在 2.4 节中介绍了多任务学习的相关内容，利用多任务学习可以提高子任务的性能。而在说话人识别中，特别是文本相关说话人识别，由于其语料内容为固定文本，如果能利用好此部分信息，那么对文本相关说话人识别性能将大有帮助。音素多任务网络就是在常规说话人识别模型的基础上，在网络右侧引入音素标签的识别分支，即通过对相同训练数据提取出的音素标签进行识别，同时通过与说话人识别任务共享隐藏层的形式，起到提升说话人识别任务性能的一种网络。例如在 TDNN-xvector 的基础上，结合音素标签做多任务学习，以提高模型的性能，具体网络结构如图 6.1 所示。

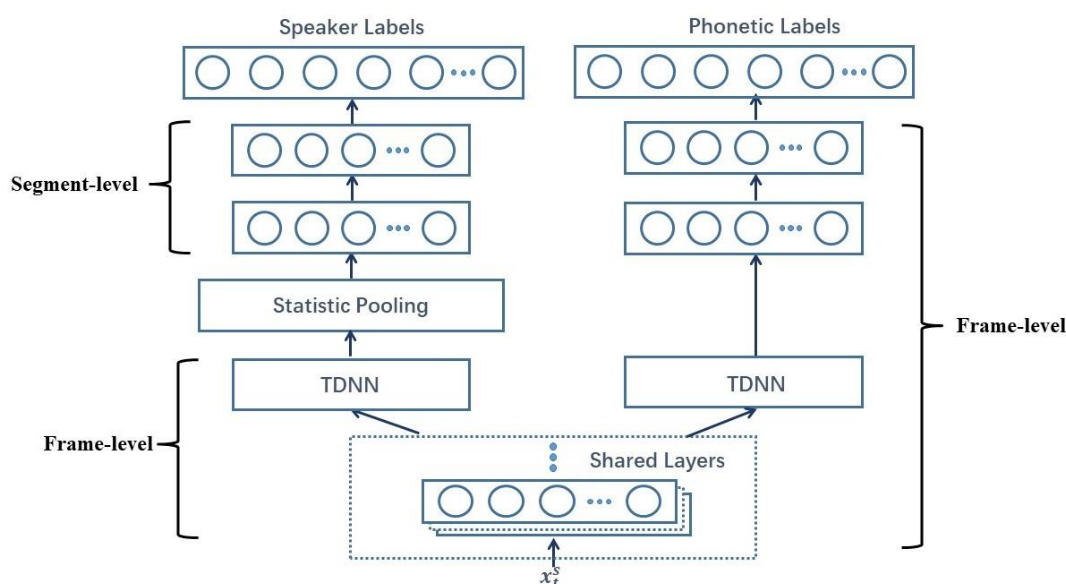


图 6.1 音素多任务网络结构

在图中，多任务参数共享部分为 TDNN x-vector 网络结构中帧级别的前四层，即 TDNN 1+TDNN 2+TDNN 3+TDNN 4。在说话人识别网络结构的基础上，经过共享层之后，在右侧引入音素标签识别任务的分支，其任务目的是识别每帧语音的音素标签。在获得说话人识别任务以及音素识别任务的损失值后经过一定

的权重计算，再通过反向传播对前面的共享层进行参数的更新，最终说话人识别任务可以在音素信息的辅助下获得性能的提升。

在音素多任务中，损失函数由两部分组成，分别是说话人分支的损失和音素分支的损失，具体如式(6.1)所示。

$$L_{total} = L_s + \beta L_p \tag{6.1}$$

其中 β 为经验控制因子，用于控制音素分支损失所含的比重。通过联合损失函数 L_{total} 对网络的参数进行更新，直至模型收敛。

6.1.1 文本相关数据准备

文本相关数据这里采用的是开源数据“你好米雅”[5]，该数据是希尔贝壳开源的中英文唤醒词语音数据库，有 254 名发言人，录制过程在真实家居环境中，设置 7 个录音位，使用 6 个圆形 16 路 PDM 麦克风阵列录音板做远讲拾音（16kHz，16bit）、1 个高保真麦克风做近讲场音（44.1kHz，16bit）。此数据库经过专业语音校对人员转写标注，并通过严格质量检验，字正确率 100%，可用于声纹识别、语音唤醒识别等研究使用。测试集为上述数据集的开发集，有 42 人，同样可以在官网下载获得。

在本实验中，仅用高保真语音作为训练与测试数据。其中训练集共有 254 人，每个人共有 80 条语音，经四倍噪声扩充后共有 101335（源数据部分说话人低于 80 条）。测试集共有 42 人，一条用于注册，其余用于测试。

6.1.2 文本相关基线模型

基线模型也采用 TDNN-xvector，与第五章 Voxceleb1 模型相同，因此这里拷贝 Voxceleb1 模型训练的启动器，并重命名为 runStanardXvector-miya.py，脚本内容除了修改数据集模型以及输出路径之外，特别地要修改以下几个地方：

1、chunk_size 设置为 100。由于该数据集均为短语音，因此 chunk_size 设为 100 仅将低于 1s 的语音剔除，若设置为 200 则整个数据集均被剔除了。

[5] http://www.aishelltech.com/wakeup_data

2、训练次数。由于该数据集过小，这里仅需要训练一个 epoch 即可，训练次数过多直接会导致过拟合。由于仅用于对比，这里不对学习率等参数进行调优。

模型训练完成后，需进行测试。测试集 trials 可通过 subtools/getTrials.sh 获得。后续打分测试集，则使用训练集训练 PLDA 用于打分，打分脚本为第三章描述的打分集脚本 subtools/scoreSets.sh，将里面的数据参数以及打分方式修改即可。最终该数据集基线系统的得分如下：

```
compute-eer -
LOG (compute-eer[5.5]:main():compute-eer.cc:136) Equal error rate is 2.95359%, at threshold 7.02112
EER% 2.954

[ miya_test ]
2.954   exp/standard_miya/far_epoch_1/miya_test/score/plda_miya_enroll_miya_test_submean_norm.eer
```

图 6.2 文本相关基线系统结果

6.1.3 多任务模型训练

多任务模型训练除了准备声纹分支的数据之外，还需额外准备音素分支的对齐标签数据。音素标签的获取需要通过 ASR 模型解码每条语音的文本信息获取，由于“你好米雅”数据文本内容简单，这里使用开源 ASR 模型获取对齐标签。ASR 模型可在 Kaldi 官网下载获得[6]。

在完成音素标签的获取之后，需要将生成的 ali.scp、ali.pdf.ark、num_jobs 和 phones 四个文件复制到对应训练集的目录下。

拷贝多任务启动脚本 subtool/pytorch/launcher/runMultiTaskXvector.py 至当前目录，修改以下几部分即可运行：

```
import libs.training.trainer_mt as trainer #第 22 行
endstage = min(5, args.endstage) #第 156 行
"extend":False #第 198 行
"warmR.T_max":1, #第 248 行
"warmR.T_mult":1, #第 249 行
epochs = 2 #第 255 行
traindata="data/mfcc_23_pitch/miya_train_aug" #第 265 行
egs_dir="exp/egs/miya_train_aug_mt" + "_" + sample_type #第 266 行
```

[6] http://kaldi-asr.org/models/2/0002_cvte_chain_model_v2.tar.gz

```
model_dir = "exp/trans_miya_train_aug_mt_pytorch_xvector_fix" #第 269 行
ali_dir="exp/ali/miya_train_aug"  # #第 271 行
```

之后便可以开始训练:

```
python3 runMultiTaskXvector.py -stage 0
```

训练完成之后进行测试，最终结果如下图所示：

```
compute-eer -
LOG (compute-eer[5.5]:main():compute-eer.cc:136) Equal error rate is 2.41109%, at threshold 10.4558
EER% 2.411

[ miya_test ]
2.411   exp/miya_train_aug_mt_pytorch_xvector_fix/far_epoch_2/miya_test/score/plda_miya_enroll_miya_test_submean_norm.eer
```

图 6.3 多任务模型结果

可以看到，多任务的加入提升了系统的性能，相对基线提升了约 18%。

6.2 迁移学习

在 2.4.3 节中有介绍简单迁移学习的内容，迁移学习对于小样本数据模型的性能可以起到较大的改进。一般来说，将文本无关模型参数迁移至文本相关模型的步骤可以分为以下三步：

1）首先需要确定进行迁移的模型结构以及特征，要求文本无关和文本相关数据的特征维度需一致。

2）确定需要迁移的网络层参数，可以选择只迁移部分网络层参数，也可以迁移全部网络层参数。一般来说，由于迁移前后的任务目标不一致，最后的输出层输出的节点数也会不相同，因此最后的输出层无法进行迁移。在上一步获取到的模型中将需要迁移的网络参数保存下来，然后利用这些参数对新训练的模型进行参数初始化。参数初始化时，进行迁移的参数会替代初始化的参数，没有迁移的参数会正常进行随机初始化操作。

3）完成模型的初始化之后，直接利用文本相关数据重新开始训练模型，直至模型拟合。

6.2.1 迁移模型训练

这里采用的文本相关数据与上节多任务学习的数据相同，也是“你好米雅”

的数据。第五章用 Voxceleb1 数据训练了 TDNN-xvector，这里直接利用该模型作为预训练模型初始化文本相关模型的参数。在 subtools 中，迁移学习的设置非常简单，只需两步：

1、在启动器脚本的 exist_model 中设置预训练模型的路径，并修改模型保存路径，如图 6.4 所示。这里的启动器，可以直接复制此前文本相关模型启动器脚本 runStandardXvector-miya.py 重命名为 runTranStandardXvector-vox2miya.py。

```
##-----------------------------------------------------##
## Other options
exist_model="exp/standard_voxceleb1/21.params"  # Use it in transfer learning.
##-----------------------------------------------------##
## Main params
traindata="data/mfcc_23_pitch/miya_train_aug"
egs_dir="exp/egs/mfcc_23_pitch_miya_train_aug" + "_" + sample_type

model_blueprint="subtools/pytorch/model/snowdar-xvector.py"
model_dir="exp/Trans_standard_vox2miya"
##-----------------------------------------------------##
```

图 6.4 迁移学习启动器脚本设置

2、修改 model_blueprint 脚本的 transfer_keys 参数以确定需要迁移的参数，如图 6.5 所示。这里我们除输出层之外均进行迁移，由于 snowdar-xvector.py 支持 E-TDNN 模型及 SE 模块，因此这里会写上这些层，这里写上对训练的模型不影响。

```
# An example to using transform-learning without initializing loss.affine parameters
self.transform_keys = ["tdnn1","tdnn2","tdnn3","tdnn4","tdnn5","stats","tdnn6","tdnn7",
                       "ex_tdnn1","ex_tdnn2","ex_tdnn3","ex_tdnn4","ex_tdnn5",
                       "se1","se2","se3","se4"]

if margin_loss and transfer_from == "softmax_loss":
    # For softmax_loss to am_softmax_loss
    self.rename_transform_keys = {"loss.affine.weight":"loss.weight"}
```

图 6.5 迁移参数设置

在模型训练时，采用迁移学习方案会显示以下代码：

```
#### Use exp/standard_voxceleb1/21.params as the initial model to start transform-training.
2022-04-08 06:00:28,038 [/work/kaldi/egs/xmuspeech/example/subtools/pytorch/libs/support/utils.py:65 - select_mod
0 ]
```

图 6.6 迁移学习运行提示

模型训练完成之后，进行测试。最终结果如下图所示：

```
compute-eer -
LOG (compute-eer[5.5]:main():compute-eer.cc:136) Equal error rate is 1.7179%, at threshold -1.35614
EER% 1.718

[ miya_test ]
1.718   exp/Trans_standard_vox2miya/far_epoch_1/miya_test/score/plda_miya_enroll_miya_test_submean_norm.eer
```

图 6.7 迁移学习结果

从结果上可以看到，采用迁移学习的方案，结果相对基线有约 42%的提升。

至此，迁移学习实战便已经完成。除了多任务和迁移学习外，subtools 还支持很多模型的训练，相关模型配置均存放于 subtools/pytorch/model 里面，供使用者研究、探索、完善。

第 7 章 工程部署—Runtime 实现

在前面的章节中，我们已经学会如何基于 ASV-Subtools 工具，训练 Pytorch 版本的声纹识别模型，这种模型可以用 python 代码进行读取测试，但在实际生产环境中，为提高运行效率，更多的是采用 C++进行调用，而且要能完整的从语音信号读取开始，直接运算得到输出结果，即特征提取、模型加载、x-vector 提取、后端打分全部串在一起，整合在一个可执行程序或软件开发包（SDK）里，方便工程部署。本章将具体介绍即时编译导出 TorchSript、Runtime 实现和 SDK 封装过程。

7.1 JIT 导出模型

C++调用 Pytorch 模型需要对模型进行序列化，首先转为 TorchScript（Pytorch 模型的一种中间表示），它可以被 Pytorch 的 TorchScript 编译器识别、编译并进行序列化，从而可以在高性能环境（例如 C++）中运行。因此，将 Pytorch 训练的模型导出并且不再依赖于复杂的 python 环境，是生产环境下模型部署的关键一环。

在 Pytorch 模型中，Module 是其基本组成单元，它由以下三部分组成：

(1) 一个构造器，在模块调用前对其做准备工作。

(2) 参数集以及一些子模块，它们通过构造器进行初始化，并用于之后的模块调用。

(3) forward 函数，模块调用后运行的代码，包含前向传播的逻辑。

在 Pytorch 的动态图框架中，计算图是运行与搭建同时进行的，具有灵活易调试的特点，并且易于 debug 和观察，但是牺牲了一部分性能的可优化性。TorchScript 保存的是模块运行时进行的操作的中间表示，在深度学习中被称作整个模型的静态计算图，这样算法的调参与算法的部署便可以解耦，从而达到分别优化的目的。其中，Pytorch 中的 torch.jit 模块可以将普通 Module 转化成 TorchScript module（TorchScript 的核心数据结构），TorchScript module 中主要保存着计算图、模型参数等信息，并可以被其他程序（如 C++）所调用。在 torch.jit

模块中提供两种模式对计算图进行捕捉：tracing 模式和 scripting 模式。正如其名，tracing 模式即追踪一个输入流所经过的路径并进行记录，因此需要提供一个输入样例，跟踪这个输入在模型 forward 过程中的操作可以得到整个计算图的结构，同时对一些无关的流程进行舍弃，在这个过程中无需对原 python 代码进行改动。可以看出，对一些逻辑简单的模型来说这种方式非常简洁，但是当模型存在复杂的流程控制语句时，比如在含有 if-else 逻辑分支的情况下，这种模式只能得到其中一条分支的路径，不能遍历所有逻辑分支来捕获它们的控制流，这种情况下就需要用另一种模式 scripting。在 scripting 模式下，编译器会整体分析所有代码，将 python 源代码“翻译”成 TorchScript，并在编译过程中进行一些优化，如此，可以得到整个模型的信息。但是这种模式有一个前提：Module 是用 TorchScript Language 编写。TorchScript Language 为 python 代码的一个子集，它通过 python3 的 typing 模块实现静态类型，除了一些特殊的特性以外它与原生 python 并无它异。因此可以用 TorchScript Language 提前定义完整静态图，从而使用方法与静态图深度学习框架如 TensorFlow 等靠拢。通过以上介绍我们可以知道：tracing 模式的优点是简单并可以直接使用，但是无法捕获模型中的控制流；而 scripting 模式可以得到模型的全部信息，但是需要对原代码进行修改使其符合 TorchScript Language 规范。最终，得到的 TorchScript 对象可以直接通过 Pytorch 进行序列化保存，保存文件中包含着模型的代码信息、参数、属性等，成为一个独立的部分并可以被其他程序加载。

ASV-Subtools 新框架设计开发了 JIT 导出模块，并针对一些主流声纹识别模型如 F-TDNN、ResNet 等进行修改以适应 TorchScript Language 语法，提供导出与测试脚本。由于在声纹识别中，主要是对注册语音与测试语音所提取的 embedding 进行相似度分析，故只需要编译网络中生成 embedding 的部分结构，我们将训练得到的 Pytorch 模型采用 scripting 模式编译成 TorchScript 对象，序列化后就可以在 C++程序中进行读取调用。

导出 TorchScript 后，为了兼容原本框架，采用 Kaldi 格式数据对提取的 embedding 进行存储。其中，xvector-compute 为 C++提取 embedding 的命令，在这里采用 libtorch 库以读取调用导出模型。libtorch 为 Pytorch 的 C++版本，其中接口设计与 Pytorch 大多一致，能够实现绝大部分 Pytorch 中的功能，因此调用 libtorch 的接口与使用 Pytorch 也非常类似，从而实现在 C++中对 tensor 进行操

作。

以上为 JIT 模块主要内容，整体流程如图 7.1 所示，此模块作为 python 与 C++ 之间的桥梁，将训练好的模型进行转化，从而进行后续部署。

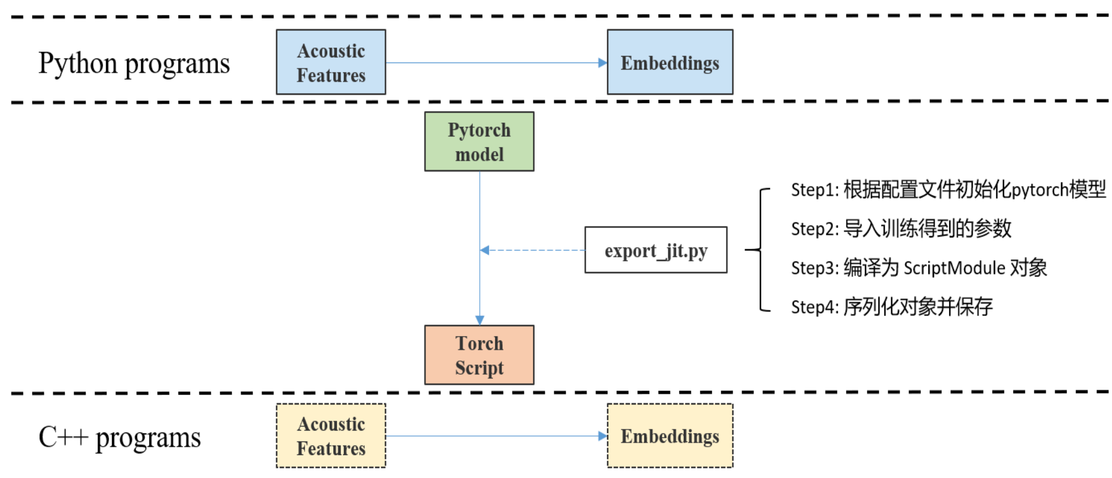


图 7.1 JIT 导出 TorchScript 示意图

7.2 Runtime 实现

Runtime 模块的工程结构如图 7.2 所示，主要由四部分组成。

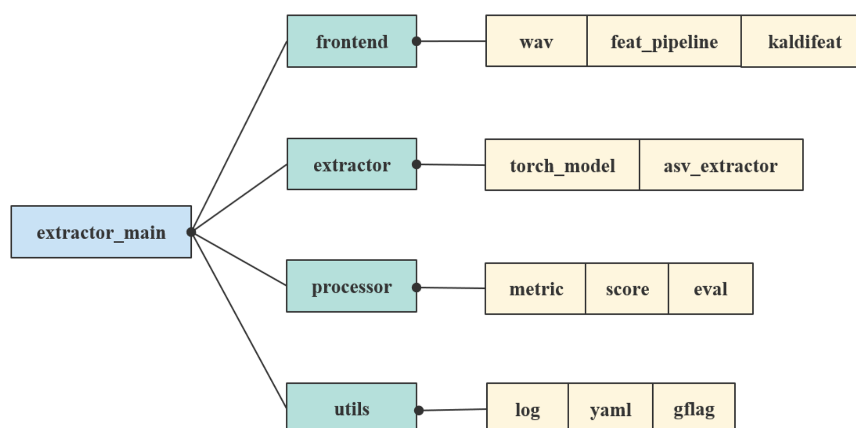


图 7.2 Runtime 模块工程结构图

首先，frontend 模块负责前端处理语音信号与特征提取功能。在特征提取方面，我们有如下几点需求：

（1）在语音领域中，Kaldi 形式特征应用非常广泛，并被很多其他的框架沿用，

同时 ASV-Subtools 在 python 端也是采用 Kaldi 特征，出于兼容性与通用性的考量，Runtime 模块需要适配 Kaldi 特征。

（2）后续模块需要调用 Pytorch 训练的模型，在 C++主要数据流为 Tensor 形式的数据类型，故采用基于 LibTorch 的 Tensor 能更好地与后续模块兼容。

（3）在实际应用场景中，需要配置一些适配的特征提取参数诸如采样率、高低频等，故 Runtime 需要可以灵活地配置特征参数。

Kaldifeat[16]是一个基于 LibTorch 编写的特征提取模块，它可以在 Pytorch 端兼容 Kaldi 形式特征，在诸如 k2[17]、Lhotse[18] 等优秀项目都有所采用。鉴于以上原因 frontend 模块特征提取模块集成 Kaldifeat 的 C++接口。Kaldifeat 特征提取如图 7.3 示，可完成几种主流的声学特征的提取。

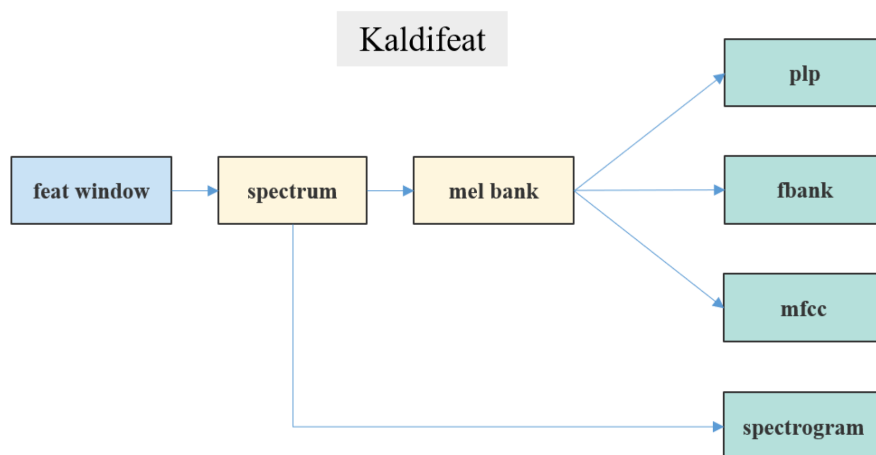


图 7.3 Kaldifeat 模块功能示意图

其次，extractor 模块涉及 embedding 的提取，是整个模块的核心部分，主要负责读取模型及模型推理。在 Pytorch 端训练好模型后，为了能被 C++读取，需要将其序列化为 TorchScript。在 asv_extractor 中涉及了基于能量的语音活动检测（VAD）、倒谱均值归一化（CMN）、embedding 提取等功能。

最后，processor 模块为后续打分模块，提取的 x-vector 通过 score 模块打分并与阈值对比进而判断是否为正例。最后，utils 模块为一些通用工具，比如日志，配置文件读取等。由于 python 训练端特征参数配置采用了 yaml 文件，故在这里我们了采用 yaml-cpp 库来适配 yaml 文件读取。gflags 是谷歌开源的一种命令行解析工具，它的特点是在工程中，flags 可以定义在分散的源文件中，即：在一个

源文件中定义了 flags，和这个源文件链接了的其他文件都可以使用它所定义的flags，增加了参数配置的灵活性。我们在工程中采用了 gflags 来进行命令行参数解析。

Runtime 模块调用流程如图 7.4 所示。

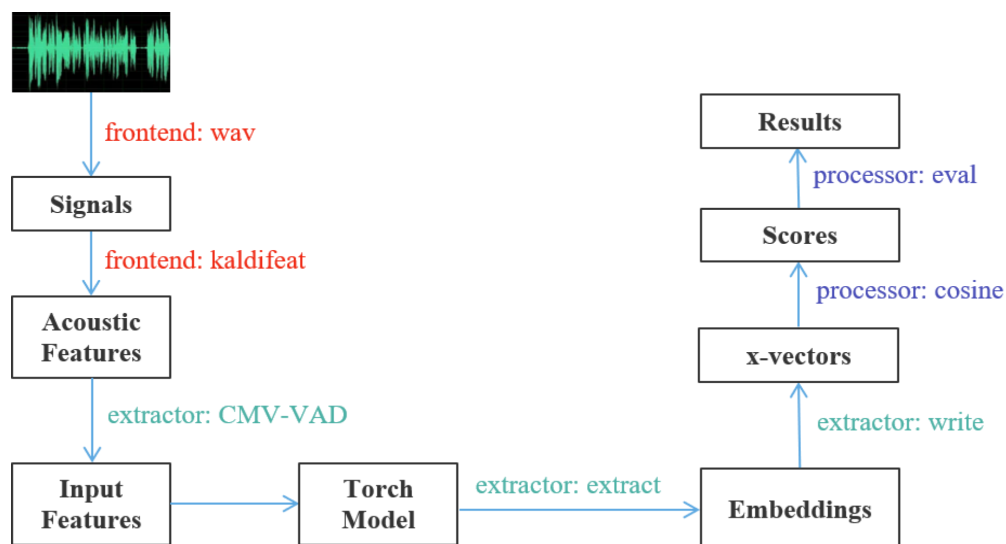


图 7.4 Runtime 模块调用流程示意图

原始语音首先经过 frontend 模块，语音转化成数字信号后，在计算声学特征之前转化为 tensor 形式的数据类型，随后利用 Kaldifeat 计算声学特征。extractor 模块首先读取之前准备好的模型并进行初始化，接收声学特征后进行 embedding 提取。此外，在此模块中也集成了常用的均值归一化（CMN）以及基于能量的语音活动检测（VAD）功能。最后，将提取的 x-vector 存储后进入后处理（processor）模块，后处理模块实现了余弦近似度打分，得到阈值后对样例进行判断。

在最后，我们提供了 CMake 文件来构建整个项目。CMake 可以产生可移植的 makefile 文件，简化构建编译过程，通过 CMake 文件我们可以一键构建整个 Runtime 模块。

7.3 SDK 封装

为了更方便调用 Runtime 模块，方便工程部署，我们可以进一步把核心代码封装成 SDK。

首先要定义好函数接口，假设引擎名为 ASV，我们定义说话人确认函数 ASV_VerifySpeaker 如下：

```
/**********************************************************************/
/*
                              确认说话人模型
                  @ handle          : 线路资源标识（必须是已打开）
                  @ buffer          : 语音缓冲
                  @ length          : 缓冲长度，即采样点个数
                  @ trial_model_file: 目标说话人模型保存的路径
                  @ res             : 返回确认结果
                  @ return          : 成功返回 SUCCESS
                                                                     */
/**********************************************************************/
return_ASV_Code ASV_VerifySpeaker(Handle handle, short* buffer, unsigned
long length, const char* trial_model_file, ASV_Result& res);
```

其中参数 handle 用来支持多线程调用，每个线程对应一个句柄号，用来区分不同解码过程，以防占用资源冲突。

函数返回结果是 return_ASV_Code 类型，包含运行中可能出现的各种情况，用枚举类型定义如下：

```
enum return_ASV_Code
{
        ASV_SUCCEEDED_OK=0,         //0:操作成功
        ASV_WORKINGDIR_NOT_FIND,    //1:工作目录不存在
        ASV_CONFIG_FILE_NOT_FOUND,//2:配置文件未找到
        ASV_MODEL_FILE_NOT_FOUND, //3:模型文件未找到
        ASV_LICENSE_ERROR,          //4:授权有误
        ASV_HANDLE_ERROR,           //5:句柄标识有误
        ASV_STATE_ERROR,            //6:句柄状态有误
        ASV_TOO_SHORT_BUFFER,       //7:语音太短
        ASV_EXTRACT_FEAT_ERROR,     //8:特征提取出错
        ASV_MODEL_LOAD_ERROR,       //9:模型加载出错
        ASV_MODEL_SAVE_ERROR,       //10:模型保存出错
        ASV_BUSY,                   //11:线路正忙
        ASV_IDLE,                   //12:线路空闲
        ASV_CLOSE,                  //13:线路关闭
        ASV_OTHER_ERROR,            //14:其他错误
};
```

其中 ASV_SUCCEEDED_OK 对应的值为 0，表示函数运行成功，可从输出参数获

取输出值。其它情况涉及工作目录访问不到、语音太短、线路忙等状态，根据 return_ASV_Code 返回值可帮助诊断引擎出现的问题。

引擎还要包含初始化和关闭、句柄打开和关闭等函数，定义如下：

```
/**************************************************************************/
/*
                                初始化引擎
                  @ working_dir     : 工作目录下有引擎工作所必需的文件
                  @ max_lines       : 引擎支持最大线数
                  @ return          : 成功初始化返回 SUCCESS
                                                                        */
/**************************************************************************/
return_ASV_Code ASV_Init(const char* config_file,int max_lines);

/**************************************************************************/
/*
                        关闭引擎：改变各线路状态为 CLOSE
                                                                         */
/**************************************************************************/
return_ASV_Code ASV_Release();

/**************************************************************************/
/*
                          打开线路：h 是线路资源标识
                                                                         */
/**************************************************************************/
return_ASV_Code ASV_Open(Handle &h);

/**************************************************************************/
/*
                          关闭线路：h 是线路资源标识
                                                                         */
/**************************************************************************/
return_ASV_Code ASV_Close(Handle &h);
```

ASV_Init 除了包含引擎文件的加载，同时也检查线路授权，分配能同时并发的路数，以支持多路调用。调用 ASV_Init 函数后，如果函数值返回 ASV_SUCCEEDED_OK，表示初始化工作成功。声纹识别服务如果要关闭，则要调用 ASV_Release 函数释放相关资源。由于涉及多线程调用，每个线程需要分配专门的句柄，因此还需 ASV_Open 和 ASV_Close 两个函数。外部程序调用 ASV_Open 分配到句柄后，执行完相关程序也要及时调用 ASV_Close 关闭释放句柄。

完成函数接口定义后，我们需要把引擎代码编译成动态库。Linux 环境不方便修改及调试代码，为便于操作，我们建议采用跨平台工具，开发环境可采用 CodeBlocks。工程配置保存完，把整个工程目录传到 Linux 环境，Linux 的编译需要 Makefile 配置文件。为提高效率，可采用 cbp2make 工具（可网上下载）把 asr.cbp 工程文件转化为 Makefile 文件。有了 Makefile 文件，即可在 Linux 环境进行 make 编译。

动态库生成后，我们即可进行调用。程序要调用时，先初始化引擎，然后分配句柄，再调用相关的识别函数，识别完关闭句柄。程序到最后还要关闭引擎，释放资源。以下是简单的调用流程。

```
//初始化引擎
return_ASV_Code asv_res = ASV_Init(config_file.c_str(),NUM_THREADS);
if(ASV_SUCCEEDED_OK == asv_res)
{
       cout<<"ASV init success!"<<endl;
}
else
{
       cout<<"return_ASV_Code="<<asv_res<<endl;
       cout<<"ASV init fails!"<<endl;
       return -1;
}

//打开句柄
Handle tsASV;
ASV_Open(tsASV);
ASV_Result& res;

//声纹识别
int asv_res =
ASV_VerifySpeaker(tsASV,pWavBuffer,length,model_full_file.c_str(),res);
{
      cout<<"Verify speaker "<<model_full_file.c_str()<<" success!"<<endl;
}
else
{
      cout<<"Verify speaker "<<model_full_file.c_str()<<" fails!"<<endl;
}
```

```
//关闭句柄
ASV_Close(tsASV);

//关闭引擎
ASV_Release();
```

参考文献

[1] D. Reynolds, T. Quatieri, R. Dunn, “Speaker Verification Using Adapted Gaussian Mixture Models,” Digital Signal Processing, 10(1-3), (2000)19-41.

[2] W. Campbell, “Support Vector Machines Using GMM Supervectors for Speaker Verification,” Signal Processing Letters, IEEE, v13, n5, p308-311, May 2006.

[3] P. Kenny, G. Boulianne, P. Ouellet, et al. “Joint Factor Analysis Versus Eigenchannels in Speaker Recognition,” IEEE Transactions on Audio Speech & Language Processing, 2007, 15(4):1435-1447.

[4] N. Dehak, P.J. Kenny, R. Dehak, et al. “Front-end Factor Analysis for Speaker Verification,” IEEE/ACM Transactions on Audio, Speech, and Language Processing, 2011, 19(4): 788-798.

[5] S.J.D. Prince and J.H. Elder, “Probabilistic Linear Discriminant Analysis for Inferences About Identity,” in Proc. IEEE ICCV, Rio de Janeiro, Brazil, Oct. 2007.

[6] D. Garcia-Romero and C.Y. Espy-Wilson, “Analysis of i-vector Length Normalization in Speaker Recognition Systems,” INTERSPEECH 2011.

[7] Y. Lei, N. Scheffer, L. Ferrer, M. McLaren, "A Novel Scheme for Speaker Recognition Using a Phonetically-aware Deep Neural Network", ICASSP 2014.

[8] D. Snyder, D. Garcia-Romero, G. Sell, D. Povey, S. Khudanpur, “X-vectors: Robust DNN Embeddings for Speaker Recognition,” ICASSP 2018.

[9] K. Okabe, T. Koshinaka, K. Shinoda, “Attentive Statistics Pooling for Deep Speaker Embedding,” INTERSPEECH 2018.

[10] B. Desplanques, J. Thienpondt, and K. Demuynck, “ECAPA-TDNN: Emphasized Channel Attention, Propagation and Aggregation in TDNN Based Speaker Verification,” INTERSPEECH 2020.

[11] D. Liao, T. Jiang, F. Wang, L. Lin, Q. Hong, “Towards A Unified Conformer Structure: from ASR to ASV Task,” arXiv preprint arXiv:2211.07201, 2022.

[12] 赵淼，基于 ASV-Subtools 的声纹识别系统设计与鲁棒性优化，厦门大学硕士学位论文，2020.

[13] 江涛，噪声场景下的文本相关说话人识别技术研究，厦门大学硕士学位论文，2021.

[14] 陆昊，基于深度学习的说话人识别与分割聚类研究，厦门大学硕士学位论文，2021.

[15] F.C. Tong, M. Zhao, J.F. Zhou, H. Lu, Z. Li, L. Li, Q.Y. Hong, “ASV-Subtools: Open Source Toolkit for Automatic Speaker Verification,” ICASSP 2021.

[16] https://github.com/csukuangfj/kaldifeat

[17] https://github.com/k2-fsa/k2

[18] P. Żelasko, D. Povey, J. Trmal, et al. “Lhotse: a speech data representation library for the modern deep learning ecosystem,” arXiv preprint arXiv:2110.12561, 2021.

[19] 廖德欣，基于 ASV-Subtools 的唤醒声纹系统，厦门大学硕士学位论文，2022.