

武汉大学国家网络安全学院

课程实验(设计)报告

课程名称：_____内容安全_____

实验名称：_____基于 LCNN 模型的语音检测技术_____

专业(班)：_____网络安全专业_____

指导教师：_____石小川 张典_____

学号：_____2020302181081_____

姓名：_____陈曦_____

2023 年 5 月 30 日

目 录

一. 项目描述	3
二. 项目原理	9
三. 项目关键过程、数据及其分析	14
四. 项目结果	35
五. 实验心得.....	37

一. 项目描述

1. 课题背景与项目简介

语音伪造技术作为当下新兴的网络恶意攻击技术引发了全世界信息安全界的关注,语音承载着人类语言和说话人身份信息,通过语音伪造技术可以精确模仿目标说话人的声音以达到欺骗人或机器听觉的目的。目前,深度伪造(Deepfake)正在对全球的政治经济及社会稳定带来极大的威胁,其中语音伪造是 Deepfake 实现舆论操控的核心技术之一。语音伪造技术一旦被不法分子利用,如应用于舆论误导或攻击身份识别系统等不良用途,将会对全球的民生、经济、政治和社会带来严重的威胁。

为了有效防御恶意滥用的伪造语音对人类和机器造成的上述危害,近年来针对伪造语音的检测技术在同步发展,伪造语音检测技术也应运而生。2013 年,INTERSPEECH 会议举办了第一届关于 ASV 伪造和防御的特别会议(special session),诞生了两年一届的 ASVspoof 系列挑战赛,该挑战赛通过颁布公开数据集并组织竞争性评估来促进伪造语音检测技术的发展。

本实验便是针对伪造语音样本检测的技术尝试。我们参照了综述《语音伪造及检测技术研究综述 ——任延珍 1,2 刘晨雨 2 刘武洋 2 王丽娜 1,2》的指导,选用了综述中推荐的一种处理技术:前端采用 LFCC 特征提取+后端基于 LCNN 模型的训练分类的组合方法,试图利用 ASVspoof 数据集训练出一个高准确率的伪造音频的鉴别模型。

2. 课题主要研究方法

课题的研究方法可概述为:采用 LFCC 特征提取+LCNN 模型的训练分类

LFCC: 对数频率倒谱系数,是一种的语音特征表示方法,可以有效地描述语音信号的频谱特征。2018 年, TODISCO 等人[60]在 ASVspoof 2017 v2 数据集上比较了使用 GMM 后

内容安全大实验 - 基于 LCNN 模型的语音检测

端时，MFCC 特征和 LFCC 特征的性能，结果如下图所示。实验表明， LFCC 特征的效果普遍好于 MFCC 特征。所以我们在实验中直接选取了效果普遍较好的 LFCC 特征。

(下图源自上面提到的综述)

特征	描述	性能比较 (EER)	应用
对数功率谱 (LPS)	描述语音信号功率随频率的变化关系。	-	[62]、[63]、[64]、[65]、[66]、[67]
STFT 语谱图	通过短时傅里叶变换直接提取,描述语音中各频率信号能量的时变关系,是声音的时频表示。	-	[68]、[69]、[70]、[71]、[72]、[73]、[74]
梅尔频率倒谱系数 (MFCC)	结合人耳听觉感知特性的特征。使用梅尔尺度三角滤波器组,低频区域滤波器间隔更密集,高频区域更稀疏,因此更关注低频区域。	13.81%	[68]、[75]、[76]、[77]
逆梅尔频率倒谱系数 (IMFCC)	类似于 MFCC,但滤波器组遵循逆的梅尔标度,即更关注高频区域。	-	[76]、[77]
线性频率倒谱系数 (LFCC)	类似于 MFCC,使用三角滤波器组,但滤波器间隔相等,即对全频带同等关注。	10.43%	[69]、[78]、[79]、[80]、[72]、[81]、[82]、[67]、[83]
线性预测倒谱系数 (LPCC)	从信号的全极点建模中得出,将线性预测系数 (LPC) 转换为 LPCC。	-	[84]
线性滤波器组系数 (LFB)	短时傅里叶变换的简化,计算复杂度更低。	-	[72]、[85]、[86]

LCNN:轻量卷积神经网络,通常具有较浅的网络深度和较窄的网络宽度。相比传统的 CNN 模型,它们通常具有更少的卷积层和更少的通道数,以减少模型的复杂性和计算开销。轻量卷积神经网络(LCNN)由 Wu 等人于 2018 年提出,最早用于面部识别领域,目前已被证明可有效地运用于伪造语音检测领域,ASVspoof 2017 挑战赛的 best system 和 ASVspoof 2019 挑战赛 LA 场景的 best system 都基于 LCNN 网络。

(下图源自上文的综述)

文献	年份	伪造语音检测系统设计				实验结果		
		网络输入	网络结构	损失函数	融合	子系统数量	t-DCF	EER(%)
[62]	2019	LPS、CQCC	SENet-34、SENet-50、均值方差 ResNet、扩张 ResNet、AFN	BCE、MCE	贪婪融合	5	LA 0.155 PA 0.016	LA 6.70 PA 0.59
[68]	2019	MFCC、CQCC、对数 STFT	ResNet	CE	分数加权平均	3	LA 0.153 PA 0.069	LA 6.02 PA 2.78
[69]	2020	语谱图、LFCC、CQT 谱图	Res2Net、SE-Res2Net	BCE	分数平均	3	LA 0.074 PA 0.012	LA 2.50 PA 0.46
[79]	2020	LFCC	ResNet-18	OC-softmax	-	-	LA 0.059	LA 2.19
[85]	2020	LFB	ResNet-18	LMCL	-	-	LA 0.052	LA 1.81
[80]	2020	LFCC、MGDCC、乘积谱(prod-spec)	ResNet-18	BCE	分数加权平均	3	LA 0.189 PA 0.061	LA 9.87 PA 1.75
[65]	2020	基于 CQT 的 LPS	CNN、LCNN	-	-	-	LA 0.102	LA 4.07
[81]	2020	LFCC	GMM、CNN	CE	-	-	LA 0.093 PA 0.195	LA 3.79 PA 7.98
[67]	2020	LPS、CQCC、LFCC	TCN、LCNN	MSE、CE	-	-	-	LA 5.31
[72]	2021	LFCC、LFB、语谱图	LCNN	P2SGrad	-	-	LA 0.089	LA 3.99
[86]	2021	LFB	ResNet-18、GAT	BCE	SVM 融合	4	LA 0.048	LA 1.68
[83]	2021	LFCC	LCNN、ResNet、ResNet-OC	OC-softmax	-	-	-	LA 3.92
[59]	2021	s-vector、x-vector	ResNet18-L-FM	LCML	逻辑回归	2	-	LA E1 0.64
[78]	2021	LFCC	LCNN	PSA	-	-	-	LA→PA 9.25 / PA→LA 10.45
[110]	2021	LFCC、STFT	胶囊网络	边缘损失	分数平均	2	LA 0.033 PA 0.051	LA 1.07 PA 2.05
[97]	2021	CQT 谱图	ResMax	-	-	-	LA 0.06 PA 0.009	LA 2.19 PA 0.37
[93]	2021	Global M	CNN+GRU	CE	加权投票	2	LA 0.074	LA 4.03

从上面两图的研究方法可得到：采用 LFCC 特征提取+LCNN 模型的训练分类是一种可行的,轻量级的且准确率较高的语音鉴别的组合方法,所以我们由此确定了我们项目的研究方法。

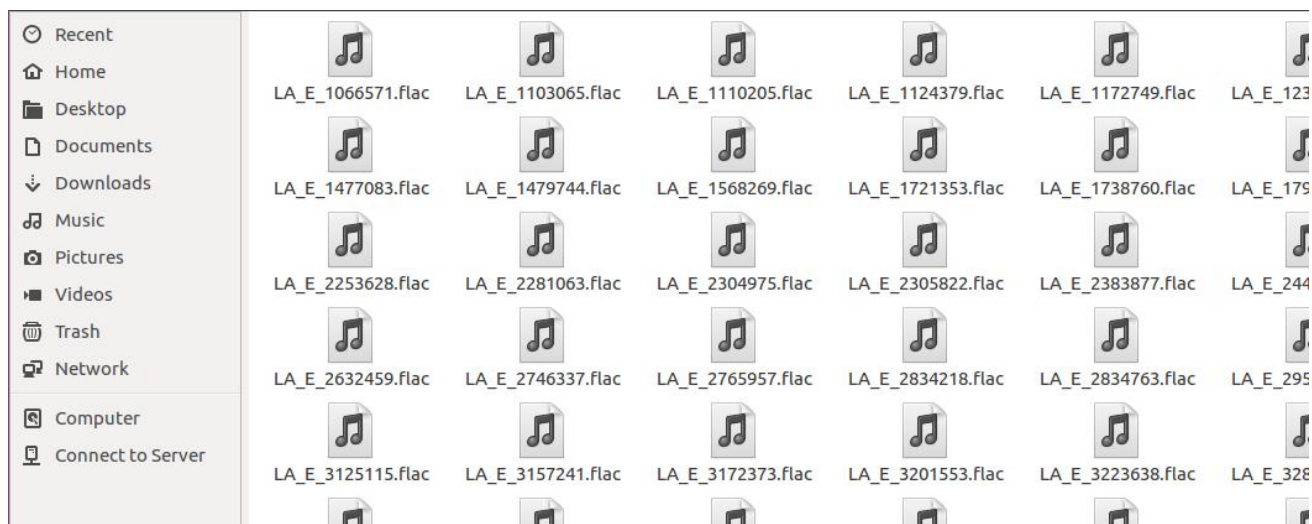
3. 数据集介绍

ASVspoof 2019 数据集有 LA 和 PA 两个子集, LA 子集中包含真实语音和合成/转换语音, PA 子集中包含真实语音和重放语音。同时, ASVspoof 2019 数据集在一个模拟且可控的声学环境中创建, 以控制重放语音中的非受控因素。

ASVspoof 2019 的 LA 数据集中的语音数据全部是基于 VCTK 语料库得到的。VCTK 语料库采集了包含 46 名男性和 61 名女性在内的 107 名说话人的真实语音, 所有的真实语音均采用了相同的录音配置, 且没有信道和背景噪声的干扰。而 ASVspoof 2019 LA 数据集中的欺骗语音是由真实语音使用不同的语音合成和语音转换技术得到的。训练集和开发集中的欺骗语音均来源于 6 种相同的语音合成和语音转换技术, 这 6 种技术作为已知的攻击类型, 可以用来对合成语音检测系统进行训练和调整。而测试集中欺骗语音的生成方法包含了 2 种上述的已知攻击和 11 种与 6 种已知攻击不同的语音合成和语音转换技术, 这 11 种技术作为系统面临的未知攻击类型。

本项目的数据集源自 ASVspoof 2019 的 LA 数据集, 因个人电脑计算能力的限制所以仅提取了源数据集的一小部分命名为 “toy_example” 作为实验的数据集。

内容安全大实验 - 基于 LCNN 模型的语音检测

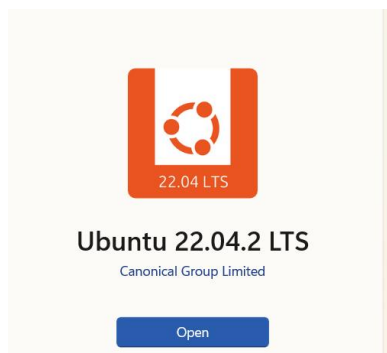


数据集的组成为测试集 150 条+验证集 150 条的音频文件,其中混杂了真实的音频文件和人工合成的伪造音频文件。可以作为实验的数据集。

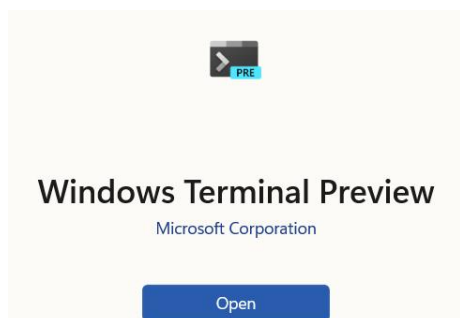
4. 实验环境

该实验使用 Linux 系统以及 GPU。故在 Windows 本机上安装 Ubuntu 子系统。

可以使用微软商店中的软件包。这里下载的是 Ubuntu 22.04 版本。



为了方便使用与视图,这里下载了 Windows Terminal Preview。



内容安全大实验 - 基于 LCNN 模型的语音检测

配置好 Windows 上的 Linux 子系统。

```
Windows PowerShell  chenxi@LAPTOP-CI081861: ~  +  v
适用于 Linux 的 Windows 子系统现已在 Microsoft Store 可用！
你可以通过运行 “wsl.exe --update” 或通过访问 https://aka.ms/wslstorepage
从 Microsoft Store 安装 WSL 将可以更快地获取最新的 WSL 更新。
有关详细信息，请访问 https://aka.ms/wslstoreinfo
>

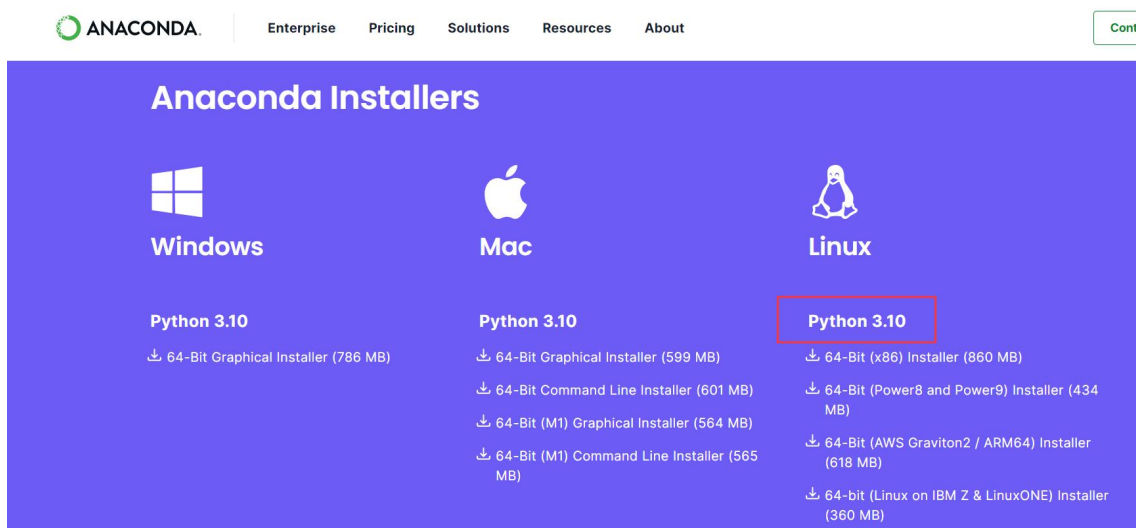
Welcome to Ubuntu 22.04.2 LTS (GNU/Linux 5.10.16.3-microsoft-standard-WSL2 x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

This message is shown once a day. To disable it please create the
/home/chenxi/.hushlogin file.

(base) chenxi@LAPTOP-CI081861:~$
```

下载 Linux 版的 conda。



The image shows the Anaconda website's 'Installers' page. It has a blue header with the Anaconda logo and navigation links: Enterprise, Pricing, Solutions, Resources, About, and a green 'Contact' button. Below the header, there are three main sections for Windows, Mac, and Linux. Each section lists various installers for Python 3.10. The Linux section is highlighted with a red box around the 'Python 3.10' heading. The Linux installers include: 64-Bit (x86) Installer (860 MB), 64-Bit (Power8 and Power9) Installer (434 MB), 64-Bit (AWS Graviton2 / ARM64) Installer (618 MB), and 64-bit (Linux on IBM Z & LinuxONE) Installer (360 MB).

配置 conda 环境。

```
(base) chenxi@LAPTOP-CI081861:~$ conda
usage: conda [-h] [-V] command ...

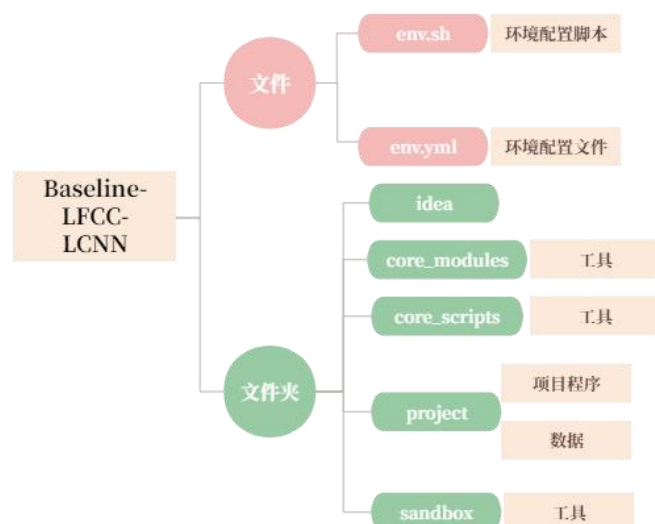
conda is a tool for managing and deploying applications, environments and packages.

Options:
positional arguments:
  command
  clean                Remove unused packages and caches.
  compare              Compare packages between conda environments.
  config               Modify configuration values in .condarc. This is modeled after the git config command.
                       Writes to the user .condarc file (/home/chenxi/.condarc) by default. Use the --show-
                       sources flag to display all identified configuration locations on your computer.
```

项目运行环境配置完成。

5. 项目结构

项目名称: Baseline-LFCC-LCNN



env.sh 以及 env.yml 用来环境配置，包含 Python 项目所依赖的工具包。

core_modules, core_scripts, sandbox 包含了实现模型以及实现数据处理所使用的工具函数，还有一些数学处理函数和信息反馈输出函数。

project 是核心的项目代码，/project/DATA 中包含数据集，以及解压和处理后的数据集。

/project/Baseline_LA 中包含项目运行主要代码，包括配置与运行。

二. 项目原理

1. LFCC 特征提取

LFCC 是一种音频特征提取方法，与 MFCC 类似，在此处详细介绍其原理：

在 LFCC 中，首先对音频信号进行预加重处理，以增强高频部分的能量。然后，将预加重后的信号分帧，将每一帧划分为重叠的时间窗口。对于每个窗口，进行快速傅里叶变换 (FFT)，将时域信号转换为频域表示。

接下来，应用一组滤波器来对频谱进行滤波。这些滤波器通常采用三角形滤波器组成的滤波器组，用于模拟人耳对不同频率的敏感程度。滤波器组的中心频率以及带宽可以根据人耳的听觉特性进行设计。

在滤波之后，对每个滤波器输出的能量值取对数。这是因为人耳的感知是非线性的，对于音频信号的较小变化更敏感。取对数可以近似人耳的感知特性。

最后，对取对数后的能量值应用离散余弦变换 (DCT)，将频域能量转换为倒谱系数。通常只保留较低频的倒谱系数，因为这些系数包含了关于音频信号的主要信息。这些倒谱系数称为 LFCC。

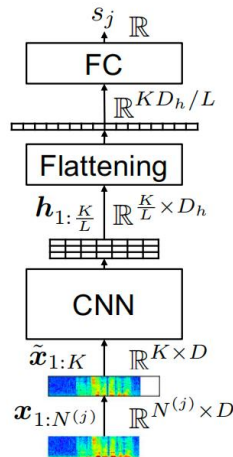
通过提取 LFCC 特征，可以捕捉音频信号的时域和频域特征，从而用于音频处理任务，如语音识别、说话人识别、情感分析等。LFCC 特征具有较好的鲁棒性和区分度，广泛应用于语音和音频领域。

MFCC 更贴近人耳的听觉特性，常用于语音识别任务；而 LFCC 在保留音频细节信息的同时，具备一定的鲁棒性，因此在音频识别的任务中选用 LFCC 特征。

在前端将音频的特征值提取后将这些特征提供给神经网络后端进行进一步的处理和分类。

由于输入的语音数据长度并不确定，因此在神经网络后段的选择上有三种选择：

(1) 假设输入大小固定的政策

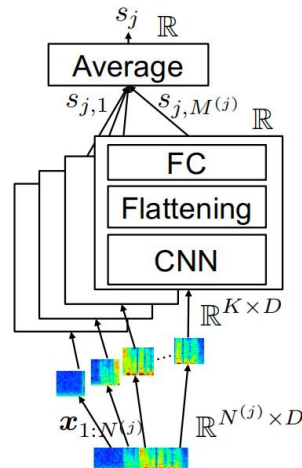


将音频填充或修剪，使其成为一个大小固定的矩阵，并使用 CNN 将其转换为 $h_{1:K/L}$ ，其中 L 由卷积的步长确定。然后，将 $h_{1:K/L}$ 展平，并将其转换为得分。

这种策略在 LCNN、ResNet 或其他基于 CNN 的分类模型中被使用。分类模型可能会对较短的样本进行填充，使用随机噪声或复制帧的方式[。对于长度大于 K 的样本，模型可能会选择从输入序列中随机选择一个长度为 K 的子序列，其中起始位置 n 可以是随机整数，或者固定为 1。

这种策略假设输入具有固定的大小，通过填充或修剪输入序列，然后使用 CNN 对其进行转换和处理，最终得到单个得分用于分类或其他任务。

(2) 将音频切分为小块的策略

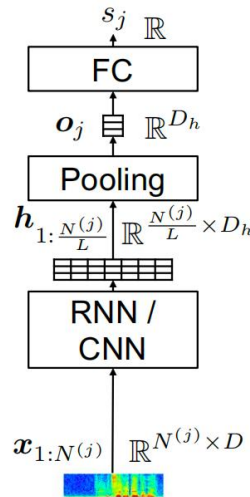


内容安全大实验 - 基于 LCNN 模型的语音检测

第二种方法通常将音频长度设置得较小，并将整段音频分帧为固定大小的块。然后，使用 CNN 将每个块转换为得分，并平均计算最终得分。有些分类模型在分帧之前可能还会对音频进行填充，使其成为一个长序列。

使用固定大小的输入会产生一些副作用：修剪会丢弃音频中的信息，而填充会引入伪影。虽然分块保留了所有的信息，但块之间是独立地由神经网络进行评分。

对于长度不同的语音输入，另一种替代策略是转换音频特征的第一步由常见的神经网络支持。



图中中 $L > 1$ 表示使用大步幅的卷积神经网络（CNN），或者 $L = 1$ 表示使用递归神经网络（RNN）。隐藏层进一步在语音级别上进行池化得到一个向量。

池化权重可以是均匀的，也可以通过注意力机制进行计算。这种策略已经在一些 RNN 和基于 CNN 的分类模型中使用过。

2. LCNN 模型

轻量卷积神经网络（Lightweight Convolutional Neural Network）是一种设计用于在资源受限的环境下实现高效推理的神经网络结构。它专注于在保持模型尺寸较小和计算复杂度较低的同时，尽可能保持良好的性能。

轻量卷积神经网络的设计原则主要包括以下几个方面：

网络结构简化：通过减少模型中的层数和参数数量，轻量卷积神经网络可以降低模型的复杂度。常见的做法包括使用更浅的网络结构、减少卷积层和全连接层的数量，以及使用较小的卷积核和池化核。

模型压缩和量化：轻量卷积神经网络可以通过模型压缩和参数量化技术来减少模型的存储空间和计算量。例如，可以使用剪枝（pruning）方法删除不必要的连接或参数，或者将模型参数量化为较低精度的表示。

网络结构优化：通过网络结构的优化和设计选择，轻量卷积神经网络可以在保持较小尺寸的同时提供较好的性能。

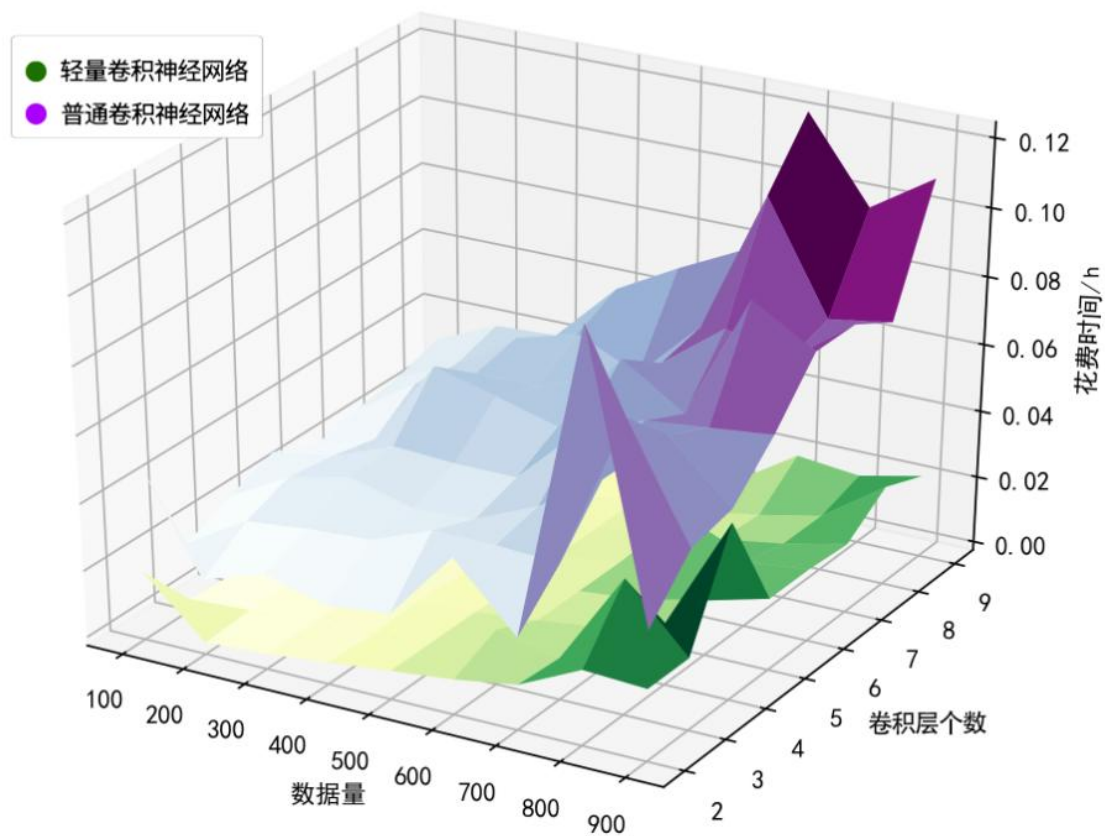
轻量卷积神经网络在嵌入式设备、移动设备和边缘计算等资源受限的环境中具有重要的应用价值。它们可以在保持较低的计算资源消耗的同时，实现高效的音频处理、图像识别、物体检测等任务。

本模型 LCNN 可在减少计算成本和存储空间的情况下处理带有大量噪声标签的大规模数据，其核心在于每个卷积层后引入了最大特征图激活函数。对比 ReLU 激活函数，最大特征图激活后得到的特征图更紧密，同时可实现特征选择和特征降维。实验表明，最大特征图激活丢弃音频特征中的噪音影响(环境噪音、信号失真等)，保留核心信息，增强下层网络特征学习能力。

除使用最大特征图的激活函数之外同时简化网络结构，采用更浅的网络结构、减少卷积层和全连接层的数量来轻量化模型。

内容安全大实验 - 基于 LCNN 模型的语音检测

关于轻量级卷积和普通卷积，这里做了算法分析并用图片输出。可以看出轻量级卷积效率更高，时间复杂度更低。

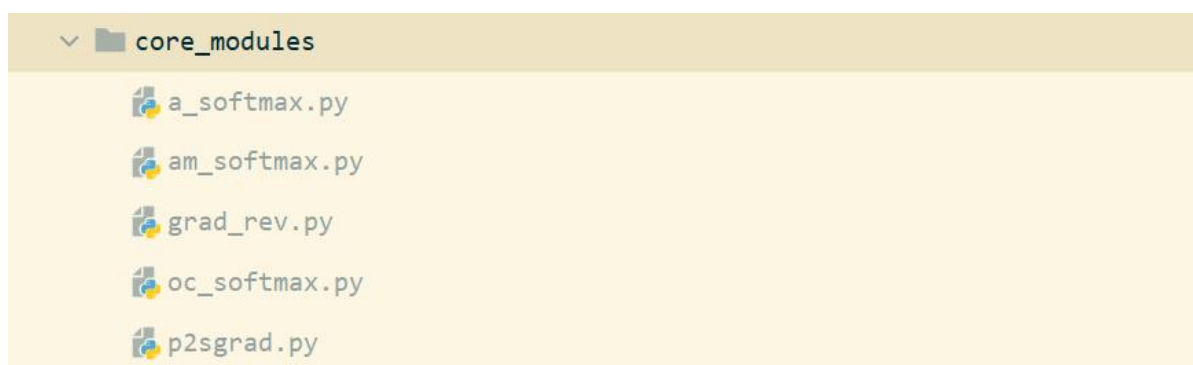


三. 项目关键过程、数据及其分析

1. 项目工具

(1) core_modules

core_modules 主要实现了损失函数以及分类输出。



a_softmax.py: 实现一个 Angular Softmax 损失，由两个类组成。输出层为 Angular 的 softmax 层生成激活。

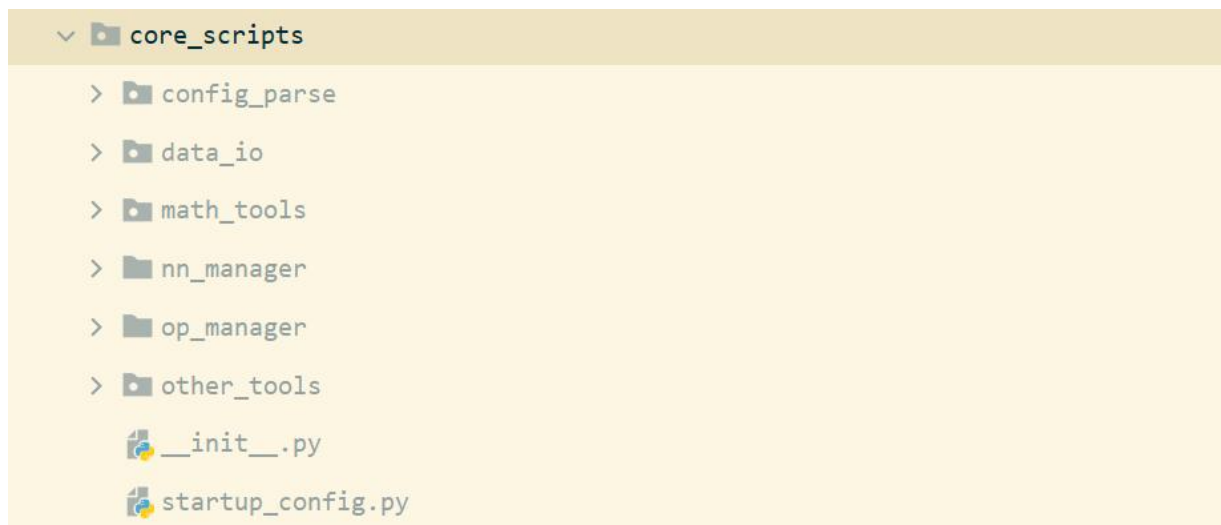
am_softmax.py: 实现了 AM-Softmax 的 Pytorch 模块,包含两个类。激活 AM-Softmax 层，并定义 AM-Softmax 层的损失函数。

grad_rev.py: 一个 pytorch 模块，实现了梯度反转层。自动求导函数计算梯度反转层的前向传播和反向传播。

oc_softmax.py: pytorch 模块，用于实现一类学习中的 OCAngle 层和 OCSoftmaxWithLoss 损失函数。

p2sgrad.py: pytorch 模块，用于实现 P2SGrad 激活层和 P2SGradLoss 损失函数。计算特征向量与类向量之间的余弦相似度，使用优化的梯度更新策略，使得优化过程更加稳定和准确。

(2) core_scripts



config_parse: 编写了配置函数，在程序出错的时候报错，并提示错误原因。

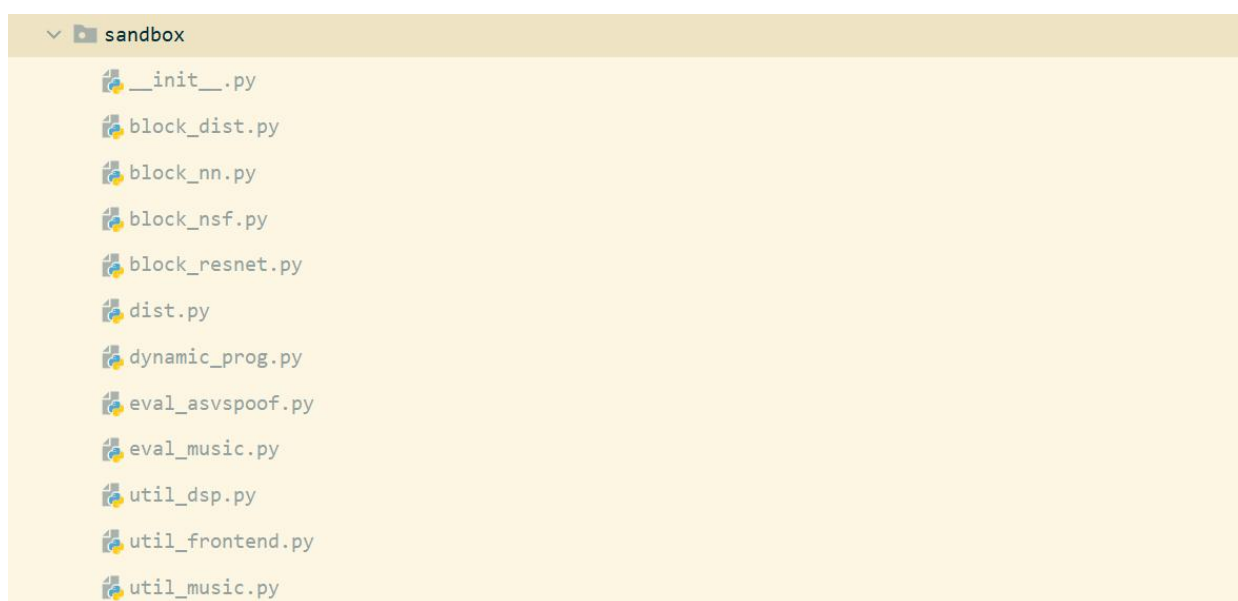
data_io: 使文本规范化的很多功能，如将文本字符转换成代码索引，基于大括号修饰文本等。

Math_tools: 变换输入列表等数学操作函数。

nn_manager、op_manager: 一些基础管理函数。

other_tools: debug，处理字符串等其它函数。

(3) sandbox



内容安全大实验 - 基于 LCNN 模型的语音检测

block_dist.py: 实现分类分布的输出层。

block_nn.py: 神经网络的常用模块。

block_nsf.py: 为 NSF 定义的主要模块。

block_resnet.py: ResNet 模型的主要实现, 包含一些预激活。

dist.py: 概率分布的效用。

dynamic_prog.py: 实现了一些动态规划函数。

eval_asvspoof.py: 用于评估的函数- asvspoof 和相关的二进制分类任务。

eval_music.py: 具有一些音频评估功能有关的函数。

util_dsp.py: 一些信号处理工具。

util_frontend.py: 有一些特征提取的工具。

Util_music.py: 音频实用程序的应用工具。

2. 数据预处理

(1) 加载模型

importlib 模块动态地导入两个模块, 分别是 args.module_config 和 args.module_model

```
1. prj_conf = importlib.import_module(args.module_config)
2. prj_model = importlib.import_module(args.module_model)
```

import_module()函数的返回值是一个表示导入的模块的对象, 可以使用这个对象来访问模块中定义的函数、类、变量等。

(3) 初始化--准备数据的输入输出

```
1. nii_startup.set_random_seed(args.seed, args)
2. use_cuda = not args.no_cuda and torch.cuda.is_available()
3. device = torch.device("cuda" if use_cuda else "cpu")
```

(4) 定义参数

```
1. if not args.inference:
```


内容安全大实验 - 基于 LCNN 模型的语音检测

```
2.         params = {'batch_size': args.batch_size,
3.                   'shuffle': args.shuffle,
4.                   'num_workers': args.num_workers,
5.                   'sampler': args.sampler}
```

(5) 读取文件夹里的文件

1. `trn_lst = nii_list_tool.read_list_from_text(prj_conf.trn_list)`
调用的函数如下 (`core_scripts/other_tools/list_tools.py`) :

```
1. def read_list_from_text(filename, f_chop=True):
2.
3.     data = []
4.     with open(filename, 'r') as file_ptr:
5.         for line in file_ptr:
6.             line = nii_str_tool.string_chop(line) if f_chop else line
7.             data.append(line)
8.     return data
```

如果 `InStr` 字符串的长度大于等于 2，并且最后两个字符的 ASCII 码分别为 13 和 10（即 `"\r\n"`），则返回一个截断后的字符串，该字符串是从 `InStr` 中去掉最后两个字符（即 `"\r\n"`）后的结果。

如果 `InStr` 字符串的长度大于等于 1，并且最后一个字符的 ASCII 码为 10（即 `"\n"`），则返回一个截断后的字符串，该字符串是从 `InStr` 中去掉最后一个字符（即 `"\n"`）后的结果。

如果以上两个条件都不满足，则返回 `InStr` 字符串本身。

(6) 预处理

【定义参数】

```
1.     trn_set = nii_dset.NIIDDataSetLoader(
2.         prj_conf.trn_set_name, \
3.         trn_lst,
4.         prj_conf.input_dirs, \
5.         prj_conf.input_exts, \
6.         prj_conf.input_dims, \
7.         prj_conf.input_reso, \
8.         prj_conf.input_norm, \
9.         prj_conf.output_dirs, \
```

内容安全大实验 - 基于 LCNN 模型的语音检测

```
10.         prj_conf.output_exts, \
11.         prj_conf.output_dims, \
12.         prj_conf.output_reso, \
13.         prj_conf.output_norm, \
14.         './',
15.         params = params,
16.         truncate_seq = prj_conf.truncate_seq,
17.         min_seq_len = prj_conf.minimum_len,
18.         save_mean_std = True,
19.         wav_samp_rate = prj_conf.wav_samp_rate,
20.         global_arg = args)
```

【调用 PyTorch.torch.utils.data.Dataset 工具包】

```
1. class NIIDataSet(torch.utils.data.Dataset):
```

先把原始数据转变成 `torch.utils.data.Dataset` 类，随后再把得到的 `torch.utils.data.Dataset` 类当作一个参数传递给 `torch.utils.data.DataLoader` 类，得到一个数据加载器，这个数据加载器每次可以返回一个 Batch 的数据供模型训练使用。

在 pytorch 中，提供了一种十分方便的数据读取机制，即使用 `torch.utils.data.Dataset` 与 `Dataloader` 组合得到数据迭代器。在每次训练时，利用这个迭代器输出每一个 batch 数据，并能在输出时对数据进行相应的预处理或数据增广操作。

【自定义数据类和数据读取方式】

构造读取文件函数

```
1. def __init__( )
```

接受一个表示文件名列表的参数 `filenames` 和一个可选的数据变换函数 `transform`，用于对数据进行预处理。初始化一个数据集对象。它接受多个参数，包括数据集名称、数据文件列表、输入和输出特征的维度、分辨率、路径、扩展名、是否标准化等信息，还有是否截断序列、最小序列长度、是否保存均值和标准差、音频采样率等。

构造长度函数

```
1. def __len__(self):
```

内容安全大实验 - 基于 LCNN 模型的语音检测

```
2.         return len(self.m_seq_info)
```

返回当前数据集中样本的数量。它直接返回了 `m_seq_info` 列表的长度，该列表存储了所有样本的信息，方便后期遍历。

构造索引函数

```
1.         def __getitem__(self, idx):
2.
3.             try:
4.                 tmp_seq_info = self.m_seq_info[idx]
5.             except IndexError:
6.                 nii_warn.f_die("Sample %d is not in seq_info" % (idx))
```

接受一个整数参数 `idx`，表示要获取的样本的索引。它首先通过索引获取 `m_seq_info` 列表中对应的样本信息，并将其存储在 `tmp_seq_info` 中。

波形静音处理函数

```
1.         def f_post_data_process(self, in_data, out_data, seq_info, idx):
```

构造检查样本长度的函数，排除异常样本

```
1.         def f_valid_len(self, t_1, t_2, min_length):
2.
3.             if max(t_1, t_2) > min_length:
4.                 if (np.abs(t_1 - t_2) * 1.0 / t_1) > 0.1:
5.                     return False
6.             return True
```

如果 `t_1` 或 `t_2` 大于 `min_length`，则说明这个样本的序列长度很长，可能需要进行截断或其他处理。如果两个时间步长的差异太大，则说明这个样本可能存在异常，需要进行进一步的处理。在这个方法中，如果两个时间步长的差异大于 `t_1` 的 10%，则会返回 `False`，否则返回 `True`。

检查特定文件的输入和输出特征数据长度

```
1.         def f_check_specific_data(self, file_name):
2.             """ check the data length of a specific file
3.             """
4.             tmp_dirs = self.m_input_dirs.copy()
5.             tmp_exts = self.m_input_exts.copy()
```

内容安全大实验 - 基于 LCNN 模型的语音检测

```
6.         tmp_dims = self.m_input_dims.copy()
7.         tmp_reso = self.m_input_reso.copy()
8.         tmp_dirs.extend(self.m_output_dirs)
9.         tmp_exts.extend(self.m_output_exts)
10.        tmp_dims.extend(self.m_output_dims)
11.        tmp_reso.extend(self.m_output_reso)
12.        # loop over each input/output feature type
13.        for t_dir, t_ext, t_dim, t_res in \
14.            zip(tmp_dirs, tmp_exts, tmp_dims, tmp_reso):
15.
16.            file_path = nii_strTk.f_realpath(t_dir, file_name, t_ext)
17.            if not nii_ioTk.file_exist(file_path):
18.                nii_warn.f_die("%s not found" % (file_path))
19.            else:
20.                t_len = self.f_length_data(file_path) // t_dim
21.                print("%s, length %d, dim %d, reso: %d" % \
22.                    (file_path, t_len, t_dim, t_res))
23.        return
```

将输入和输出特征的文件夹、扩展名、维度和分辨率存储在临时变量中，并将它们合并在一起。

然后，遍历每种输入/输出特征类型，并使用 `nii_strTk.f_realpath()` 方法获取文件的完整路径。如果文件不存在，则会输出错误信息并退出。否则，代码将调用 `f_length_data()` 方法获取文件的数据长度，并将其除以特征的维度，以获取序列长度。最后，代码会输出文件的完整路径、序列长度、特征维度和分辨率信息。

记录数据文件的长度

```
1.     def f_log_data_len(self, file_name, t_len, t_reso):
```

如果数据长度小于等于 1 或数据分辨率小于 0，则说明这个数据不需要考虑序列长度。否则，代码会计算序列长度，并将其存储在 `m_data_length` 字典中。然后，代码会检查 `file_name` 是否已经存在于 `m_data_length` 字典中，如果存在，则只保留最短的序列长度。最后，代码会调用 `f_adjust_len()` 方法对序列长度进行调整，以确保序列长度符合要求。

归一化函数

```
1.     def f_init_mean_std(self, ms_input_path, ms_output_path):
```

内容安全大实验 - 基于 LCNN 模型的语音检测

```
2.
3.         self.m_input_mean = np.zeros([self.m_input_all_dim])
4.         self.m_input_std = np.ones([self.m_input_all_dim])
5.         self.m_output_mean = np.zeros([self.m_output_all_dim])
6.         self.m_output_std = np.ones([self.m_output_all_dim])
7.
8.         flag = True
```

初始化输入和输出特征的均值和标准差。在初始化时，代码会将所有特征的均值设置为 0，将所有特征的标准差设置为 1。如果需要从文件中加载均值和标准差，则会尝试加载它们。

如果加载成功，则将加载的均值和标准差存储在相应的属性中。如果加载失败，则需要重新计算均值和标准差。

计算文件长度函数

```
1.     def f_sum_data_length(self):
2.
3.         return sum([x.seq_length() for x in self.m_seq_info])
```

检查数据集中的数据长度是否已经被存储在文件中并加载数据长度。因为在训练模型时，我们需要知道每个文件的长度，以便将它们划分成适当的批次。

打印数据集的相关信息的函数

```
1.     def f_print_info(self):
2.         """
3.         """
4.         mes = "Dataset {:}:".format(self.m_set_name)
5.         mes += "\n Time steps: {:d} ".format(self.m_data_total_length)
6.         if self.m_truncate_seq is not None:
7.             mes += "\n Truncate length: {:d}".format(self.m_truncate_seq)
8.         mes += "\n Data sequence num: {:d}".format(len(self.m_seq_info))
9.         tmp_min_len = min([x.seq_length() for x in self.m_seq_info])
10.        tmp_max_len = max([x.seq_length() for x in self.m_seq_info])
11.        mes += "\n Maximum sequence length: {:d}".format(tmp_max_len)
12.        mes += "\n Minimum sequence length: {:d}".format(tmp_min_len)
```

内容安全大实验 - 基于 LCNN 模型的语音检测

包括数据集名称、时间步数、数据序列数量、输入和输出特征的相关信息（包括存储目录、扩展名、维度、分辨率和归一化方式）等，以便查看数据集的配置和参数设置是否正确

计算数据集的统计信息函数

```
1.         def f_calculate_stats(self, flag_cal_data_len, flag_cal_mean_std)
2.         :
3.             tmp_dirs = self.m_input_dirs.copy()
4.             tmp_exts = self.m_input_exts.copy()
5.             tmp_dims = self.m_input_dims.copy()
6.             tmp_reso = self.m_input_reso.copy()
7.             tmp_norm = self.m_input_norm.copy()
8.             tmp_dirs.extend(self.m_output_dirs)
9.             tmp_exts.extend(self.m_output_exts)
10.            tmp_dims.extend(self.m_output_dims)
11.            tmp_reso.extend(self.m_output_reso)
12.            tmp_norm.extend(self.m_output_norm)
```

包括每个文件的长度、输入和输出特征的均值和标准差等将模型的输出数据保存到文件中。在生成预测结果时，我们需要调用这个方法保存输出数据。

返回输入输出特征的总维度的函数

```
1.         def f_input_dim(self):
2.             return self.m_input_all_dim
3.
4.         def f_output_dim(self):
5.             return self.m_output_all_dim
```

idx 调整函数

```
1.         def f_adjust_idx(self, data_tuple, idx_shift):
2.
3.             for idx in np.arange(len(data_tuple[-1])):
4.                 data_tuple[-1][idx] += idx_shift
5.             return data_tuple
```

当合并多个数据集时，应该调整__getitem__中的 idx。

3. 模型实现

首先自定义最大特征图模块，即 MaxFeatureMap2D 函数：

在 sandbox.block_nn 中定义：

```
1. class MaxFeatureMap2D(torch.nn.Module):
2.
3.     def __init__(self, max_dim = 1):
4.         super().__init__()
5.         self.max_dim = max_dim
6.
7.     def forward(self, inputs):
8.         # suppose inputs (batchsize, channel, length, dim)
9.
10.        shape = list(inputs.size())
11.
12.        if self.max_dim >= len(shape):
13.            print("MaxFeatureMap: maximize on %d dim" % (self.ma
14.                x_dim))
15.            print("But input has %d dimensions" % (len(shape)))
16.            sys.exit(1)
17.            if shape[self.max_dim] // 2 * 2 != shape[self.max_dim]:
18.                print("MaxFeatureMap: maximize on %d dim" % (self.ma
19.                    x_dim))
20.                print("But this dimension has an odd number of data"
21.                    )
22.                sys.exit(1)
23.                shape[self.max_dim] = shape[self.max_dim]//2
24.                shape.insert(self.max_dim, 2)
25.
26.                # view to (batchsize, 2, channel//2, ...)
27.                # maximize on the 2nd dim
28.                m, i = inputs.view(*shape).max(self.max_dim)
29.                return m
```

用于在 2D 维度上进行最大特征图操作。

功能是将输入的特征图在指定的维度上进行最大特征图操作。默认情况下，最大特征图

操作是在通道 (channel) 维度上进行的, 即对每个通道的特征图进行最大池化 (maxout)。

具体操作如下:

初始化方法 (init) 接受一个 max_dim 参数, 用于指定进行最大特征图操作的维度, 默认为 1 (通道维度)。

前向传播方法 (forward) 接受一个输入张量 inputs, 并按照指定的维度进行最大特征图操作。

在前向传播中, 首先获取输入张量的形状 shape, 并进行一些错误检查, 例如检查指定的最大特征图操作维度是否超出了输入张量的维度, 以及指定维度的大小是否为偶数。

然后, 根据指定的维度进行特征图形状的修改, 将指定维度的大小缩小为原来的一半, 并在该维度前插入一个大小为 2 的维度。

最后, 使用 view 方法将输入张量重塑为新的形状, 即(batchsize, 2, channel//2, ...), 其中 2 表示最大特征图操作所得到的两个最大值, 然后在指定的维度上进行最大池化操作, 得到最大值张量 m。

返回最大值张量 m 作为最终的输出。

模块的实现中包含了 debug 代码, 以确保输入张量的维度和大小符合预期。在实际使用时, 可以根据需要修改或删除这些错误处理部分的代码。

此处仅分析后端神经网络细节:

```
1. torch.nn.Conv2d(1, 64, [5, 5], 1, padding=[2, 2]),
2. nii_nn.MaxFeatureMap2D(),
3. torch.nn.MaxPool2d([2, 2], [2, 2]),
4.
5. torch.nn.Conv2d(32, 64, [1, 1], 1, padding=[0, 0]),
6. nii_nn.MaxFeatureMap2D(),
7. torch.nn.BatchNorm2d(32, affine=False),
8. torch.nn.Conv2d(32, 96, [3, 3], 1, padding=[1, 1]),
9. nii_nn.MaxFeatureMap2D(),
```


内容安全大实验 - 基于 LCNN 模型的语音检测

```
10.
11. torch.nn.MaxPool2d([2, 2], [2, 2]),
12. torch.nn.BatchNorm2d(48, affine=False),
13.
14. torch.nn.Conv2d(48, 96, [1, 1], 1, padding=[0, 0]),
15. nii_nn.MaxFeatureMap2D(),
16. torch.nn.BatchNorm2d(48, affine=False),
17. torch.nn.Conv2d(48, 128, [3, 3], 1, padding=[1, 1]),
18. nii_nn.MaxFeatureMap2D(),
19.
20. torch.nn.MaxPool2d([2, 2], [2, 2]),
21. torch.nn.Conv2d(64, 128, [1, 1], 1, padding=[0, 0]),
22. nii_nn.MaxFeatureMap2D(),
23. torch.nn.BatchNorm2d(64, affine=False),
24. torch.nn.Conv2d(64, 64, [3, 3], 1, padding=[1, 1]),
25. nii_nn.MaxFeatureMap2D(),
26. torch.nn.BatchNorm2d(32, affine=False),
27. torch.nn.Conv2d(32, 64, [1, 1], 1, padding=[0, 0]),
28. nii_nn.MaxFeatureMap2D(),
29. torch.nn.BatchNorm2d(32, affine=False),
30. torch.nn.Conv2d(32, 64, [3, 3], 1, padding=[1, 1]),
31. nii_nn.MaxFeatureMap2D(),
32. torch.nn.MaxPool2d([2, 2], [2, 2]),
33.
34. torch.nn.Dropout(0.7)
```

最大特征图层即为 MaxFeatureMap2D 自定义模块

输入层：接受 1 个输入通道（假设是灰度图像），无特定大小要求。

卷积层 1：使用 64 个大小为 5x5 的卷积核进行卷积操作，步幅为 1，填充为 2。输出特征图的通道数为 64。

最大特征图层 1：对卷积层 1 的输出特征图进行一种自定义的最大特征图操作。

最大池化层 1：使用 2x2 的池化窗口进行最大池化操作，步幅为 2。

卷积层 2：使用 64 个大小为 1x1 的卷积核进行卷积操作，步幅为 1，填充为 0。输出特

征图的通道数为 64。

最大特征图层 2: 对卷积层 2 的输出特征图进行一种自定义的最大特征图操作。

批归一化层 1: 对卷积层 2 的输出特征图进行批归一化操作, 其中 `affine=False` 表示不使用可学习的参数。

卷积层 3: 使用 96 个大小为 3×3 的卷积核进行卷积操作, 步幅为 1, 填充为 1。输出特征图的通道数为 96。

最大特征图层 3: 对卷积层 3 的输出特征图进行一种自定义的最大特征图操作。

最大池化层 2: 使用 2×2 的池化窗口进行最大池化操作, 步幅为 2。

批归一化层 2: 对卷积层 3 的输出特征图进行批归一化操作, 其中 `affine=False` 表示不使用可学习的参数。

卷积层 4: 使用 96 个大小为 1×1 的卷积核进行卷积操作, 步幅为 1, 填充为 0。输出特征图的通道数为 96。

最大特征图层 4: 对卷积层 4 的输出特征图进行一种自定义的最大特征图操作。

批归一化层 3: 对卷积层 4 的输出特征图进行批归一化操作, 其中 `affine=False` 表示不使用可学习的参数。

卷积层 5: 使用 128 个大小为 3×3 的卷积核进行卷积操作, 步幅为 1, 填充为 1。输出特征图的通道数为 128。

最大特征图层 5: 对卷积层 5 的输出特征图进行一种自定义的最大特征图操作。

最大池化层 3: 使用 2×2 的池化窗口进行最大池化操作, 步幅为 2。

卷积层 6: 使用 64 个大小为 1×1 的卷积核进行卷积操作, 步幅为 1, 填充为 0。输出特征图的通道数为 64。

最大特征图层 6: 对卷积层 6 的输出特征图进行一种自定义的最大特征图操作。

内容安全大实验 - 基于 LCNN 模型的语音检测

批归一化层 4: 对卷积层 6 的输出特征图进行批归一化操作, 其中 `affine=False` 表示不使用可学习的参数。

卷积层 7: 使用 64 个大小为 3×3 的卷积核进行卷积操作, 步幅为 1, 填充为 1。输出特征图的通道数为 64。

最大特征图层 7: 对卷积层 7 的输出特征图进行一种自定义的最大特征图操作。

批归一化层 5: 对卷积层 7 的输出特征图进行批归一化操作, 其中 `affine=False` 表示不使用可学习的参数。

卷积层 8: 使用 64 个大小为 1×1 的卷积核进行卷积操作, 步幅为 1, 填充为 0。输出特征图的通道数为 64。

最大特征图层 8: 对卷积层 8 的输出特征图进行一种自定义的最大特征图操作。

批归一化层 6: 对卷积层 8 的输出特征图进行批归一化操作, 其中 `affine=False` 表示不使用可学习的参数。

卷积层 9: 使用 64 个大小为 3×3 的卷积核进行卷积操作, 步幅为 1, 填充为 1。输出特征图的通道数为 64。

最大特征图层 9: 对卷积层 9 的输出特征图进行自定义的最大特征图操作。

最大池化层 4: 使用 2×2 的池化窗口进行最大池化操作, 步幅为 2。

Dropout 层: 以 0.7 的概率随机丢弃部分神经元, 用于防止过拟合。

本模型 LCNN 可在减少计算成本和存储空间的情况下处理带有大量噪声标签的大规模数据, 其核心在于每个卷积层后引入了最大特征图激活函数。对比 ReLU 激活函数, 最大特征图激活后得到的特征图更紧密, 同时可实现特征选择和特征降维。实验表明, 最大特征图激活丢弃音频特征中的噪音影响(环境噪音、信号失真等), 保留核心信息, 增强下层网络特征学习能力。

4. 项目主程序

Main.py 是用来作为训练和推理过程的默认包装器。它需要准备 `config.py` 和 `model.py` 文件。

在函数的开头，它初始化了一些参数，然后加载了模块配置和模型模块。接下来，它根据参数设置随机种子，并确定是否使用 CUDA。

```
1.      # arguments initialization
2.      args = nii_arg_parse.f_args_parsed()
3.
4.      #
5.      nii_warn.f_print_w_date("Start program", level='h')
6.      nii_warn.f_print("Load module: %s" % (args.module_config))
7.      nii_warn.f_print("Load module: %s" % (args.module_model))
8.      prj_conf = importlib.import_module(args.module_config)
9.      prj_model = importlib.import_module(args.module_model)
10.
11.     # initialization
12.     nii_startup.set_random_seed(args.seed, args)
13.     use_cuda = not args.no_cuda and torch.cuda.is_available()
14.     device = torch.device("cuda" if use_cuda else "cpu")
```

如果不是进行推理，它会准备数据输入输出，并加载文件列表和创建数据加载器。

```
15.     # prepare data io
16.     if not args.inference:
17.         params = {'batch_size': args.batch_size,
18.                   'shuffle': args.shuffle,
19.                   'num_workers': args.num_workers,
20.                   'sampler': args.sampler}
21.
22.     # Load file list and create data loader
23.     trn_lst = nii_list_tool.read_list_from_text(prj_conf.trn_list)
24.     trn_set = nii_dset.NIIDDataSetLoader(
25.         prj_conf.trn_set_name, \
26.         (部分参数因为篇幅的原因省略)
27.         params = params,
28.         truncate_seq = prj_conf.truncate_seq,
29.         min_seq_len = prj_conf.minimum_len,
```

内容安全大实验 - 基于 LCNN 模型的语音检测

```
30.         save_mean_std = True,
31.         wav_samp_rate = prj_conf.wav_samp_rate,
32.         global_arg = args)
33.
34.     if prj_conf.val_list is not None:
35.         val_lst = nii_list_tool.read_list_from_text(prj_conf.val
    _list)
36.         val_set = nii_dset.NIIDDataSetLoader(
37.             prj_conf.val_set_name,
38.             val_lst,
39.             prj_conf.input_dirs, \
40.             (部分参数因为篇幅的原因省略)
41.             params = params,
42.             truncate_seq= prj_conf.truncate_seq,
43.             min_seq_len = prj_conf.minimum_len,
44.             save_mean_std = False,
45.             wav_samp_rate = prj_conf.wav_samp_rate,
46.             global_arg = args)
47.     else:
48.         val_set = None
```

然后，它初始化模型和损失函数，并初始化优化器。

如果需要，它会恢复训练。然后开始训练。

```
1.         # initialize the model and loss function
2.         model = prj_model.Model(trn_set.get_in_dim(), \
3.                                 trn_set.get_out_dim(), \
4.                                 args, prj_conf, trn_set.get_data_mean_std())
5.         loss_wrapper = prj_model.Loss(args)
6.
7.         # initialize the optimizer
8.         optimizer_wrapper = nii_op_wrapper.OptimizerWrapper(model, args)
9.
10.        # if necessary, resume training
11.        if args.trained_model == "":
12.            checkpoint = None
13.        else:
14.            checkpoint = torch.load(args.trained_model)
15.
16.        # start training
17.        nii_nn_wrapper.f_train_wrapper(args, model,
```

内容安全大实验 - 基于 LCNN 模型的语音检测

```
18.                                     loss_wrapper, device,
19.                                     optimizer_wrapper,
20.                                     trn_set, val_set, checkpoint)
21.         # done for training
22.
23.     else:
24.
25.         # for inference
26.
27.         # default, no truncating, no shuffling
28.         params = {'batch_size': args.batch_size,
29.                  'shuffle': False,
30.                  'num_workers': args.num_workers}
31.
32.         if type(prj_conf.test_list) is list:
33.             t_lst = prj_conf.test_list
34.         else:
35.             t_lst = nii_list_tool.read_list_from_text(prj_conf.test_
36. list)
37.         test_set = nii_dset.NIIDDataSetLoader(
38.             prj_conf.test_set_name, \
39.             (由于篇幅原因整理省略部分参数)
40.             params = params,
41.             truncate_seq= None,
42.             min_seq_len = None,
43.             save_mean_std = False,
44.             wav_samp_rate = prj_conf.wav_samp_rate,
45.             global_arg = args)
```

如果进行推理，则默认不截断、不洗牌。它会加载测试列表并创建测试集。然后初始化模型。如果没有指定训练过的模型，则默认加载保存的模型。最后进行推理并输出数据。

```
1.
2.     # initialize model
3.     model = prj_model.Model(test_set.get_in_dim(), \
4.                             test_set.get_out_dim(), \
5.                             args, prj_conf)
6.     if args.trained_model == "":
7.         print("No model is loaded by ---trained-model for inference")
8.         print("By default, load %s%s" % (args.save_trained_name,
```

内容安全大实验 - 基于 LCNN 模型的语音检测

```
9.                                     args.save_model_ext))
10.             checkpoint = torch.load("%s%s" % (args.save_trained_name
11.                                     args.save_model_ext))
12.         else:
13.             checkpoint = torch.load(args.trained_model)
14.
15.         # do inference and output data
16.         nii_nn_wrapper.f_inference_wrapper(args, model, device, \
17.                                             test_set, checkpoint)
```

主程序的构造便是如上图所示。

5. 脚本编写

(1) /project/00_download.sh

下载语音验证系统的数据集。为了运行方便，仅下载了一个原有数据集的子数据集，成为“玩具数据集”。主要代码如下。

```
1.  echo "=== Downloading toy dataset ==="
2.  SRC=https://www.dropbox.com/sh/gf3zp00qvdp3row/AABc-QK2BvzEPj-s8nBwC
   kMna/temp/asvspoof2021/toy_example.tar.gz
3.  cd DATA
4.  wget -q ${SRC}
```

(2) /project/01_wrapper_eval.sh

在“玩具数据集”上使用模型进行评估。运行 02_eval_alternative.sh 脚本并传递了三个参数。

```
1.  cd DATA
2.  tar -xzf toy_example.tar.gz
3.  cd ..
4.  cd baseline_LA
5.  TRAINED_MODEL
6.  bash 02_eval_alternative.sh $PWD/../../DATA/toy_example/eval toy_eval _
   _pretrained/trained_network.pt
```

内容安全大实验 - 基于 LCNN 模型的语音检测

(3) /project/02_toy_example.sh

安装 pytorch 环境，解压数据集。

```
1. RED='\033[0;32m'
2. NC='\033[0m'
3.
4. echo -e "\n${RED}=====
   ===${NC}"
5. echo -e "${RED}Step1. install conda environment${NC}"
6. bash conda.sh
7.
8. echo -e "\n${RED}=====
   ===${NC}"
9. echo -e "${RED}Step2. untar toy data set${NC}"
10. cd DATA
11. tar -xzf toy_example.tar.gz
12. cd ..
13. echo -e "\n${RED}=====
   ===${NC}"
14. echo -e "${RED}Step3. run evaluation process (using pre-trained mode
   l)${NC}"
15. cd baseline_LA
16. bash 01_eval.sh
17. echo -e "\n${RED}=====
   ===${NC}"
18. echo -e "${RED}Step4. run training process (start with pre-trained m
   odel)${NC}"
19. bash 00_train.sh
```

(4) /project/conda.sh

创建一个 conda 环境并从 env.yml 导入配置信息。

```
1. conda env create -f ../env.yml
```

(5) /project/baseline_LA/00_train.sh

训练模型生成训练日志。

```
1. log_train_name=log_train
2. log_err_name=log_err
3. pretrained_model=__pretrained/trained_network.pt
4.
5. echo -e "Training"
```


内容安全大实验 - 基于 LCNN 模型的语音检测

```
6.  echo -e "Please monitor the log trainig: $PWD/${log_train_name}.txt\n"
7.  source $PWD/../../env.sh
8.  python main.py --model-forward-with-file-name \
9.      --num-workers 3 --epochs 100 \
10.     --no-best-epochs 50 --batch-size 64 \
11.     --sampler block_shuffle_by_length \
12.     --lr-decay-factor 0.5 --lr-scheduler-type 1 \
13.     --trained-model ${pretrained_model} \
14.     --ignore-training-history-in-trained-model \
15.     --lr 0.0003 --seed 1000 > ${log_train_name}.txt 2>${log_err_name}.txt
16. echo -e "Training process finished"
17. echo -e "Trainig log has been written to $PWD/${log_train_name}.txt"
```

(6) /project/baseline_LA/01_eval.sh

运行模型评估并生成评估日志和分数文件。

```
1.  log_name=log_eval
2.  trained_model=__pretrained/trained_network.pt
3.
4.  echo -e "Run evaluation"
5.  source $PWD/../../env.sh
6.  python main.py --inference --model-forward-with-file-name \
7.      --trained-model ${trained_model}> ${log_name}.txt 2>${log_name}_err.txt
8.  cat ${log_name}.txt | grep "Output," | awk '{print $2" "$4}' | sed 's:,:,:g' > ${log_name}_score.txt
9.  echo -e "Process log has been written to $PWD/${log_name}.txt"
10. echo -e "Score has been written to $PWD/${log_name}_score.txt"
```

(7) /project/baseline_LA/02_eval_alternative.sh

对一个音频数据集进行评估并生成评估日志和分数文件。

```
1.  if [ "$#" -ne 3 ]; then
2.      echo -e "Invalid input arguments. Please check doc of the script, and use:"
3.      echo -e "bash 02_eval_alternative.sh <wav_dir> <name_eval> <trained_model>"
4.      exit
```

内容安全大实验 - 基于 LCNN 模型的语音检测

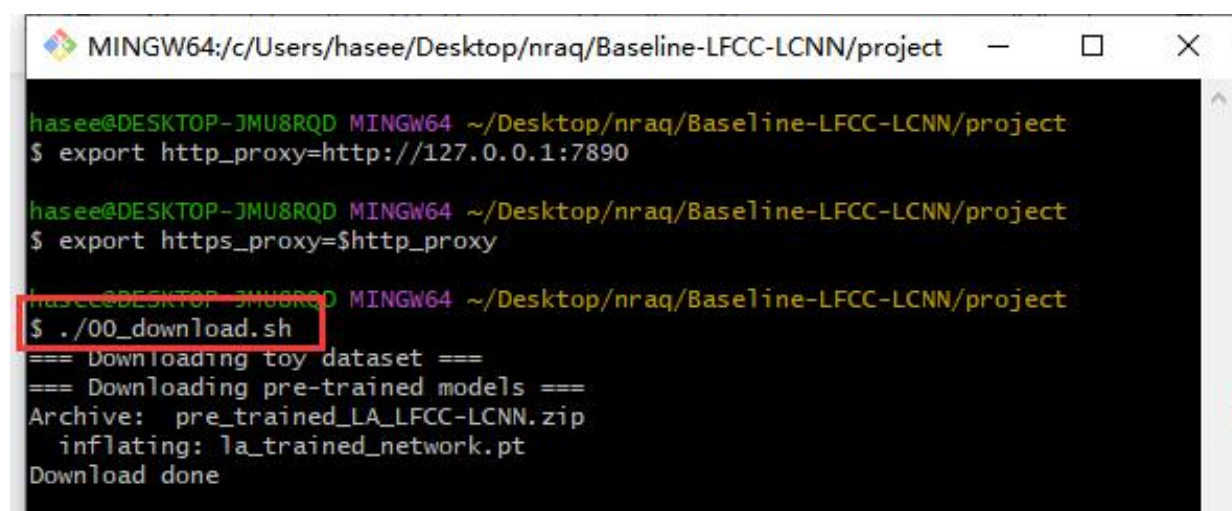
```
5.  fi
6.
7.  # path to the directory of waveforms
8.  eval_wav_dir=$1
9.  echo ${eval_wav_dir}
10. # name of the evaluation set (any string)
11. eval_set_name=$2
12. echo ${eval_set_name}
13. # path to the trained model
14. trained_model=$3
15. echo ${trained_model}
16.
17. echo -e "Run evaluation"
18.
19. # step1. load python environment
20. source $PWD/../../env.sh
21.
22. # step2. prepare test.lst
23. ls ${eval_wav_dir} | xargs -I{} basename {} .wav | xargs -I{} basena
    me {} .flac > ${eval_set_name}.lst
24.
25. # step3. export for config_auto.py
26. export TEMP_DATA_NAME=${eval_set_name}
27. export TEMP_DATA_DIR=${eval_wav_dir}
28.
29. log_name=log_eval_${eval_set_name}
30. python main.py \
31.     --inference \
32.     --model-forward-with-file-name \
33.     --trained-model ${trained_model} \
34.     --module-config config_auto > ${log_name}.txt 2>&1
35.
36. cat ${log_name}.txt | grep "Output," | awk '{print $2" "$4}' | sed '
    s,:,::g' > ${log_name}_score.txt
37.
38. echo -e "Process log has been written to $PWD/${log_name}.txt"
39. echo -e "Score has been written to $PWD/${log_name}_score.txt"
40.
41. rm ${eval_set_name}.lst
42. rm ${eval_set_name}_utt_length.dic
```

四. 项目结果

1. 数据集成功下载

实验中使用 clash 代理链接 bash 解决数据集的获取困难问题，代理细节不多做赘述。

在 bash 命令行中运行编写的数据集下载脚本：



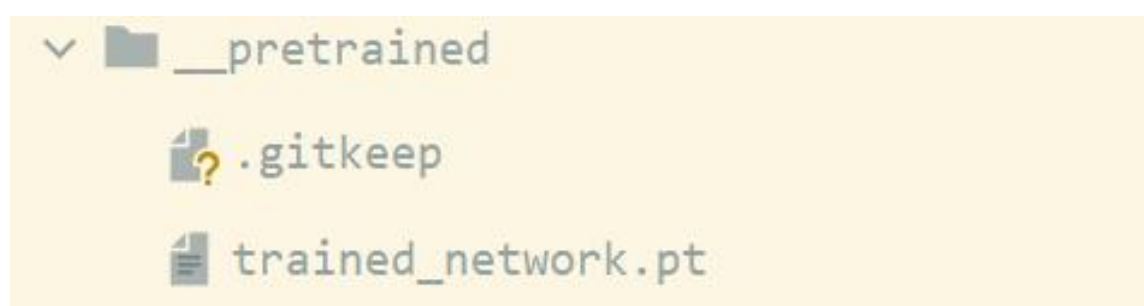
```
MINGW64:/c:/Users/hasee/Desktop/nraq/Baseline-LFCC-LCNN/project
hasee@DESKTOP-JMU8RQD MINGW64 ~/Desktop/nraq/Baseline-LFCC-LCNN/project
$ export http_proxy=http://127.0.0.1:7890

hasee@DESKTOP-JMU8RQD MINGW64 ~/Desktop/nraq/Baseline-LFCC-LCNN/project
$ export https_proxy=$http_proxy

hasee@DESKTOP-JMU8RQD MINGW64 ~/Desktop/nraq/Baseline-LFCC-LCNN/project
$ ./00_download.sh
=== Downloading toy dataset ===
=== Downloading pre-trained models ===
Archive:  pre_trained_LA_LFCC-LCNN.zip
  inflating: la_trained_network.pt
Download done
```

2. 训练集生成

运行脚本生成训练集。



内容安全大实验 - 基于 LCNN 模型的语音检测

3. 测试集结果生成

环境配置脚本运行，配置好项目运行环境。

```
(base) chenxi@LAPTOP-CI081861:/mnt/d/chenxi/files2/Baseline-LFCC-LCNN-withdata/Baseline-LFCC-LCNN$ conda env create -f env.yml
Collecting package metadata (repodata.json): done
Solving environment: done
```

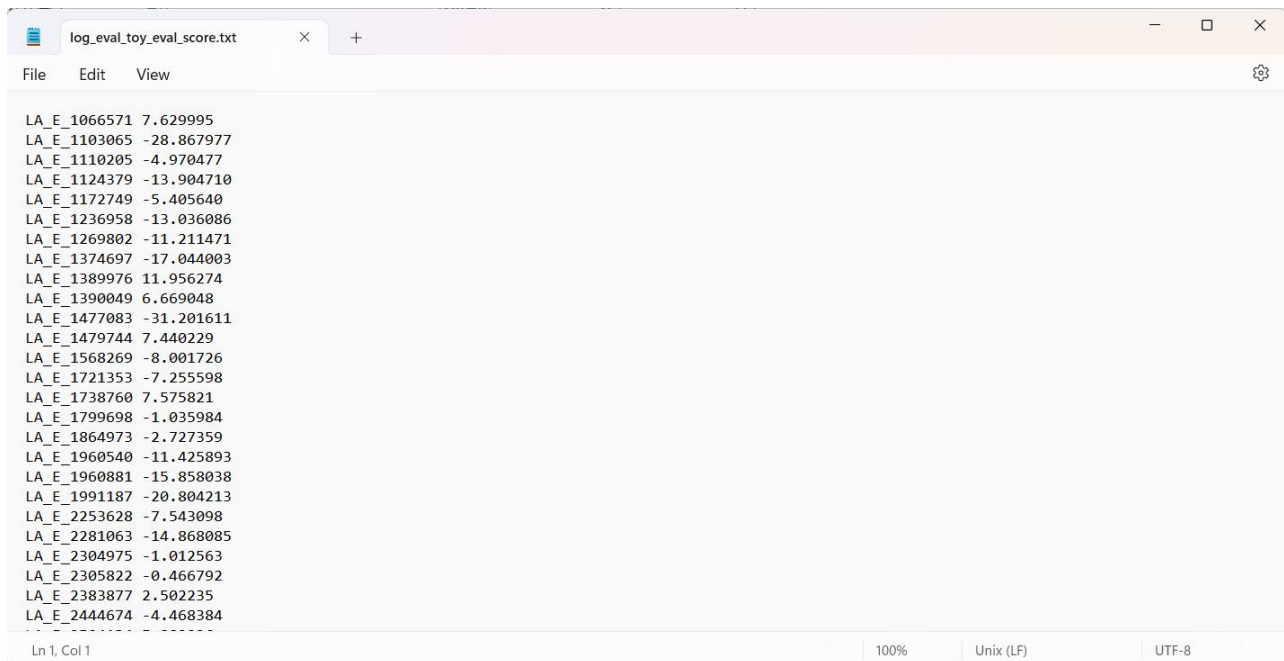
激活 pytorch 环境。

```
(base) chenxi@LAPTOP-CI081861:/mnt/d/chenxi/files2/Baseline-LFCC-LCNN-withdata/Baseline-LFCC-LCNN$ conda activate pytorch-asvspoof2022
(pytorch-asvspoof2022) chenxi@LAPTOP-CI081861:/mnt/d/chenxi/files2/Baseline-LFCC-LCNN-withdata/Baseline-LFCC-LCNN$
```

进入 project，运行脚本 01_wrapper_eval.sh，可以看到运行项目成功，项目分数结果保存在 log_eval_toy_eval_score.txt 中

```
(pytorch-asvspoof2022) chenxi@LAPTOP-CI081861:/mnt/d/chenxi/files2/Baseline-LFCC-LCNN-withdata/Baseline-LFCC-LCNN/project$ cd project/
(pytorch-asvspoof2022) chenxi@LAPTOP-CI081861:/mnt/d/chenxi/files2/Baseline-LFCC-LCNN-withdata/Baseline-LFCC-LCNN/project$ ./01_wrapper_eval.sh
Run evaluation
Process log has been written to /mnt/d/chenxi/files2/Baseline-LFCC-LCNN-withdata/Baseline-LFCC-LCNN/project/baseline_LA/log_eval_toy_eval.txt
Score has been written to /mnt/d/chenxi/files2/Baseline-LFCC-LCNN-withdata/Baseline-LFCC-LCNN/project/baseline_LA/log_eval_toy_eval_score.txt
(pytorch-asvspoof2022) chenxi@LAPTOP-CI081861:/mnt/d/chenxi/files2/Baseline-LFCC-LCNN-withdata/Baseline-LFCC-LCNN/project$
```

打开 txt 文件，分数为正且数值越大，语音真实度越高；相反，分数为负且绝对值越大，语音伪造的可能性越大。



```
LA_E_1066571 7.629995
LA_E_1103065 -28.867977
LA_E_1110205 -4.970477
LA_E_1124379 -13.904710
LA_E_1172749 -5.405640
LA_E_1236958 -13.036086
LA_E_1269802 -11.211471
LA_E_1374697 -17.044003
LA_E_1389976 11.956274
LA_E_1390049 6.669048
LA_E_1477083 -31.201611
LA_E_1479744 7.440229
LA_E_1568269 -8.001726
LA_E_1721353 -7.255598
LA_E_1738760 7.575821
LA_E_1799698 -1.035984
LA_E_1864973 -2.727359
LA_E_1960540 -11.425893
LA_E_1960881 -15.858038
LA_E_1991187 -20.804213
LA_E_2253628 -7.543098
LA_E_2281063 -14.868085
LA_E_2304975 -1.012563
LA_E_2305822 -0.466792
LA_E_2383877 2.502235
LA_E_2444674 -4.468384
```

五. 实验心得

陈曦:

个人贡献:

1. 组织项目结构骨架, 策划项目文件加以及文件布局
2. 收集并编写部分工具函数, 完备神经网络结构
3. 编写全部运行脚本, env 工具配置脚本文件
4. 使用 Ubuntu 子系统, conda 以及 GPU 环境训练出训练集
5. 对测试集调试运行, 得出实验评价效果

实验心得:

在本次实验中, 我学习了脚本的实现原理和编写, 直接执行脚本文件能够让复杂程序的执行更加方便快捷。其次, 本次实验的代码需要在 Linux 环境下运行, 所以我学会了在 Windows 系统上搭建 Linux 子系统, 这是一个繁琐的过程, 花费了一定的时间与精力; 其中也碰到了许多问题, 通过查找资料和与同学讨论, 最终都一步步的解决了。通过比较 LCNN 模型和 CNN 模型对语音伪造检测的运行结果, 更直观的看到了 LCNN 模型的优势与便利, 提高了语音伪造检测的准确率。

除此之外, 我收集了一些神经网络以及其实现 softmax 输出的函数, 并自己编写了一些数学函数和补充函数, 使项目得以运行。在这个经历中, 我理解了模型的运作原理, 对深度神经网络的认识又更深了一层, 提高了代码编写能力。

统筹规划了项目的组织结构, 加强了我的组织能力和安排能力。期末作业不是学习的终点, 期待之后对内容安全的继续学习。