福州大学

《机器学习》
科研实训报告

题　　目：基于深度学习和 VAE 结构的歌声转换算法研究

姓名（学号）：高孙炜（102101141）（组长）

姓名（学号）：林中天（102101135）

指 导 教 师：朱丹红

目录

一、实验概述.....-3-

二、实验数据集.....-3-

三、已有工作.....-3-

四、数据预处理.....-4-

4.1 格式转换.....-4-

4.2 特征提取.....-5-

五、模型设计.....-7-

5.1 模型架构.....-7-

5.2 先验编码器.....-9-

5.3 后验编码器.....-10-

5.4 解码器.....-12-

5.5 FLOW.....-13-

5.6 定义模型损失.....-13-

六、实验结果.....-15-

七、模型评估和实验总结.....-17-

7.1 模型评估.....-17-

7.2 实验总结.....-17-

八、参考文献.....-18-

九、附录.....-18-

一、实验概述

Singing Voice Conversion (SVC) 是音频处理领域的一个活跃研究方向，涉及到将一位歌手的歌声风格转换为另一位歌手的声音特征。近年来，SVC 技术受到多个领域的广泛关注。它不仅可以用于创作音乐，还可以在声音替代、个性化内容创造和多语种媒体制作中发挥重要作用，在音乐制作、语音合成等领域得到了广泛应用。随着人工智能技术的快速发展，SVC 系统在声音质量和转换效果方面有了显著提升，但依然存在一些技术挑战，例如音质保真度、实时性以及多说话人转换等问题。

本实验尝试探索一种基于深度学习和 VAE（变分自编码）结构的 SVC 技术。主要实验步骤如下：

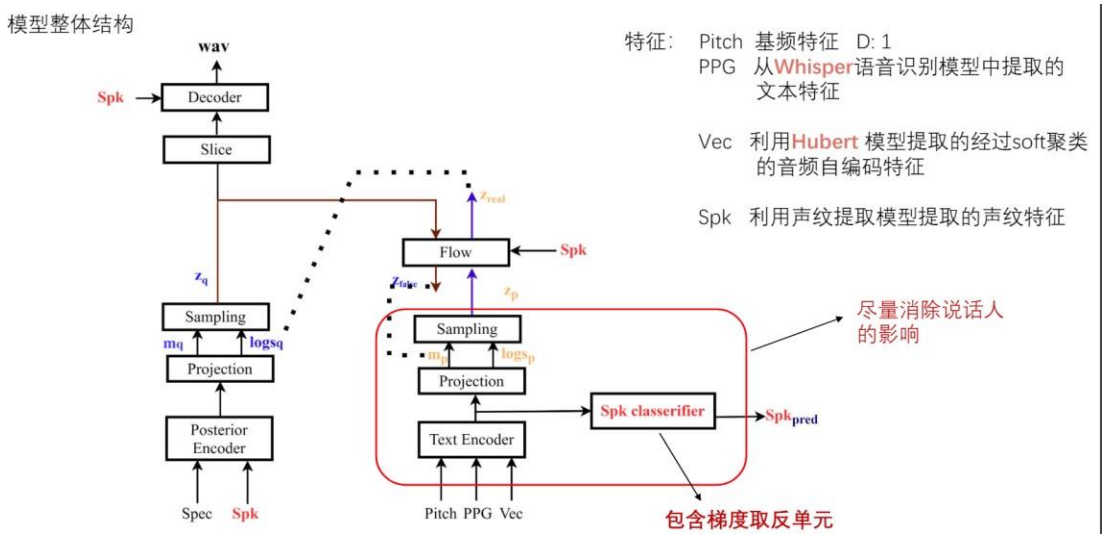
数据准备：收集并处理大规模的音频数据集，包括多种不同风格的音乐和语音样本。

模型训练：基于 PyTorch 框架，利用预训练的 wide_resnet50 作为编码器，对模型进行训练。训练过程中应用了多种数据增强技术，以提升模型的泛化能力。

模型评估：使用客观指标和主观评测相结合的方法，对转换后的音频进行质量评估，指标包括音质保真度、转换准确性和自然度等。

结果分析：通过对比实验，分析不同模型架构和训练方法对转换效果的影响，提炼出最佳实践方案。

具体流程图如下：



二、实验数据集

AISHELL-3，一个大规模、高保真的多语者普通话语音语料库。它可以用于训练多语者文本到语音(TTS)系统。该语料库包含大约 85 小时的情感中立录音，由218 名母语为中文普通话的说话者说出，总计88035 个话语。他们的辅助属性，如性别、年龄组和地方口音，都被明确标记并提供在语料库中。因此，与录音一起提供了汉字级别和拼音级别的转录。字词和声调转录准确率达到 98%以上，通过专业语音标注和严格的声调和语调质量检查。

三、已有工作

DiffSVC：

简介：DiffSVC 是一个基于扩散概率模型的歌声转换系统。它采用去噪扩散建模技术，旨在提高转换后歌声的质量。

技术特点：DiffSVC 利用基于深度 FSMN 的 ASR 声学模型来提取 PPG（音素后验图）作为内容特征，并结合了对数基频特征（Log-F0）和响度特征。其主要架构包括扩散解码器和辅助条件输入，以实现高质量的歌声转换 (ar5iv)

FastSVC：

简介：FastSVC 通过特征线性调制（FiLM）模块进行快速跨领域歌声转换，融合了语言特征、正弦激励信号和响度特征。

技术特点：采用了基于 MelGAN 的多尺度鉴别器架构，并结合了多尺度 STFT 损失来优化生成器。FastSVC 能够处理多说话人/歌手的歌声转换，支持跨语言的歌声转换，使用 WORLD 声码器提取基频（F0）值 (ar5iv)。

UCD-SVC：

简介：UCD-SVC 是一个基准系统，用于与其他 SVC 系统进行性能比较。该系统主要针对单说话人和多说话人歌声转换任务。

技术特点：UCD-SVC 系统训练了超过 60 万步，使用 ADAM 优化器，并结合了最小平方 GAN 设置和多尺度 STFT 损失。它在任何到一个（A2O）和任何到多个（A2M）的跨领域和跨语言 SVC 性能测试中表现良好 (ar5iv)。

四、数据预处理

4.1 格式转换

对音频文件进行重新采样，分别处理为采样率 16K 和 32K.统一采样率（标准化采样率）。有几个原因：

数据标准化和预处理：机器学习模型通常需要输入数据具有一致的格式和特征，包括采样率。通过将所有音频文件重采样到统一的采样率（如 16kHz 或 32kHz），可以确保输入数据的一致性，避免因采样率不同而引入的数据不一致性问题。

计算效率： 较低的采样率（如 16kHz）可以减少音频数据的大小和复杂性，从而提高处理速度和计算效率。在某些情况下，降低采样率可以在保持数据信息基本完整的同时，减少模型训练和推断的计算成本。

模型需求：某些机器学习模型对输入数据的采样率有特定的要求或偏好。例如，部分语音识别或音频分类模型可能设计为在特定采样率下达到最佳性能。因此，将音频文件重采样到这些标准采样率可以更好地适配模型的输入要求。

```python
import os
import librosa
import argparse
import numpy as np
from tqdm import tqdm
from concurrent.futures import ThreadPoolExecutor, as_completed
from scipy.io import wavfile


def resample_wave(wav_in, wav_out, sample_rate):
    wav, _ = librosa.load(wav_in, sr=sample_rate)
    wav = wav / np.abs(wav).max() * 0.6
    wav = wav / max(0.01, np.max(np.abs(wav))) * 32767 * 0.6
    wavfile.write(wav_out, sample_rate, wav.astype(np.int16))


def process_file(file, wavPath, spks, outPath, sr):
    if file.endswith(".wav"):
        file = file[:-4]
        resample_wave(f"{wavPath}/{spks}/{file}.wav", f"{outPath}/{spks}/{file}.wav", sr)


def process_files_with_thread_pool(wavPath, spks, outPath, sr, thread_num=None):
    files = [f for f in os.listdir(f"./{wavPath}/{spks}") if f.endswith(".wav")]

    with ThreadPoolExecutor(max_workers=thread_num) as executor:
        futures = {executor.submit(process_file, file, wavPath, spks, outPath, sr): file for file in files}

        for future in tqdm(as_completed(futures), total=len(futures), desc=f'Processing {sr} {spks}'):
            future.result()
```

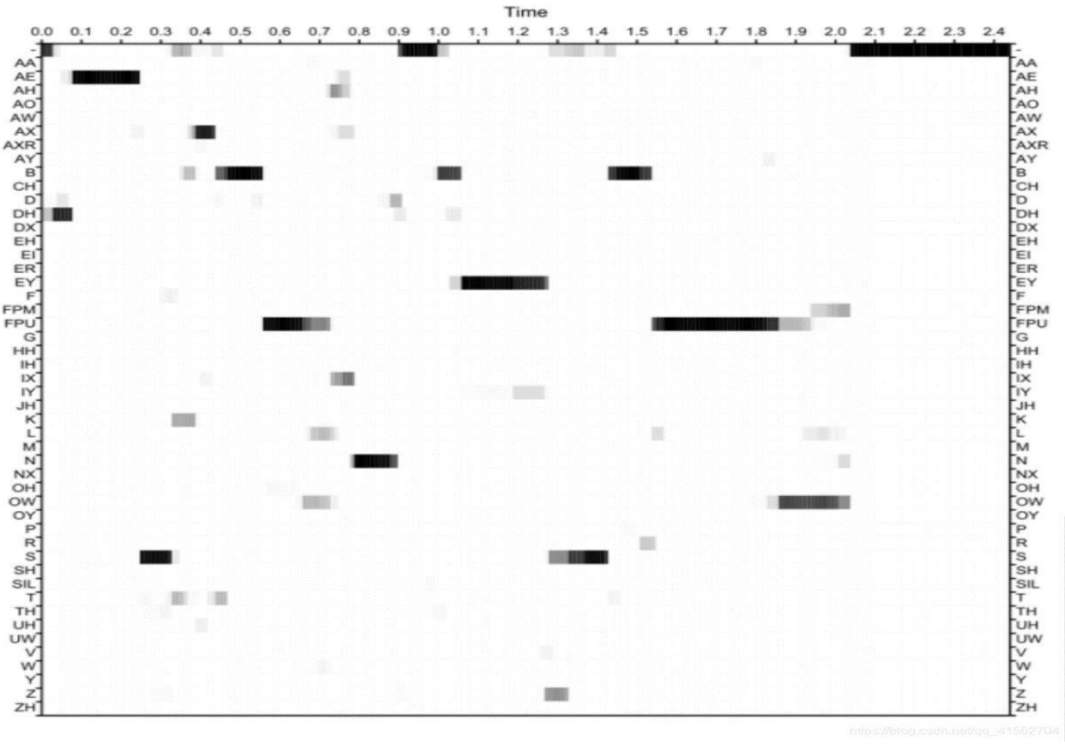
4.2 特征提取

PPG（Photoplethysmogram）特征：

PPG 的全称是 phonetic posteriorgrams，即语音后验概率，PPG 是一个时间对类别的矩阵，其表示对于一个话语的每个特定时间帧，每个语音类别的后验概率。单个音素的后验概率作为时间的函数称为后验轨迹。

一般来讲是从目标说话者的语音中，使用与说话者无关的自动语音识别(SI-ASR)系统来提取 PPG。提取到的 PPG 用作映射不同的说话者之间的关系。PPG 包括与时间范围和语音类别范围相对应的值集合，该语音类别对应于音素状态。

以 PPG 特征图为例，横坐标表示时间，纵坐标表示音素类别，每个坐标表示在给定时间点出现这个音素的后验概率大小，颜色越深，概率越大。



本实验基于 Whispher-Large-V2 语音识别模型提取 PPG 特征。

Pitch 特征：

pitch 跟声音的基频 fundamental frequency（F0）有关，反应的是音高的信息，即声调。计算 F0 也被称为‘‘pitch detection algorithms（PDA）。

```python
def compute_f0(filename, save, device):
    audio, sr = librosa.load(filename, sr=16000)
    assert sr == 16000
    # Load audio
    audio = torch.tensor(np.copy(audio))[None]
    audio = audio + torch.randn_like(audio) * 0.001
    # Here we'll use a 10 millisecond hop length
    hop_length = 160
    # Provide a sensible frequency range for your domain (upper limit is 2006 Hz)
    # This would be a reasonable range for speech
    fmin = 50
    fmax = 1000
    # Select a model capacity--one of "tiny" or "full"
    model = "full"
    # Pick a batch size that doesn't cause memory errors on your gpu
    batch_size = 512
    # Compute pitch using first gpu
    pitch, periodicity = crepe.predict(
        audio,
        sr,
        hop_length,
        fmin,
        fmax,
        model,
        batch_size=batch_size,
        device=device,
        return_periodicity=True,
    )
    # CREPE was not trained on silent audio. some error on silent need filter.pitPath
    periodicity = crepe.filter.median(periodicity, 7)
    pitch = crepe.filter.mean(pitch, 5)
    pitch[periodicity < 0.5] = 0
    pitch = pitch.squeeze(0)
    np.save(save, pitch, allow_pickle=False)
```

Vec(音频自编码)特征：

自编码器可以通过训练学习音频的低维表示，这些表示通常被称为音频的嵌入向量（Embedding Vector）或特征向量。

本实验基于 Hubert 提取经过 soft 聚类的音频自编码特征。

Spk（声纹特征）：

声纹识别是一种通过分析声音的生物特征来识别和验证个体身份的技术。它类似于指纹识别或视网膜扫描等生物识别技术，但是基于声音的特征。在声纹识别中，主要的声学特征包括声音的频谱特性、共振峰、语音节奏等，这些特征能够辨别出不同说话人的声音特征。

为什么要使用这些特征在 SVC 中?

复杂声音建模： 歌声转换涉及复杂的声音特征和个性化的声音特征捕捉，这些特征如 PPG 和 VVC 可以提供关于声音的丰富信息，帮助模型更好地理解和转换声音。

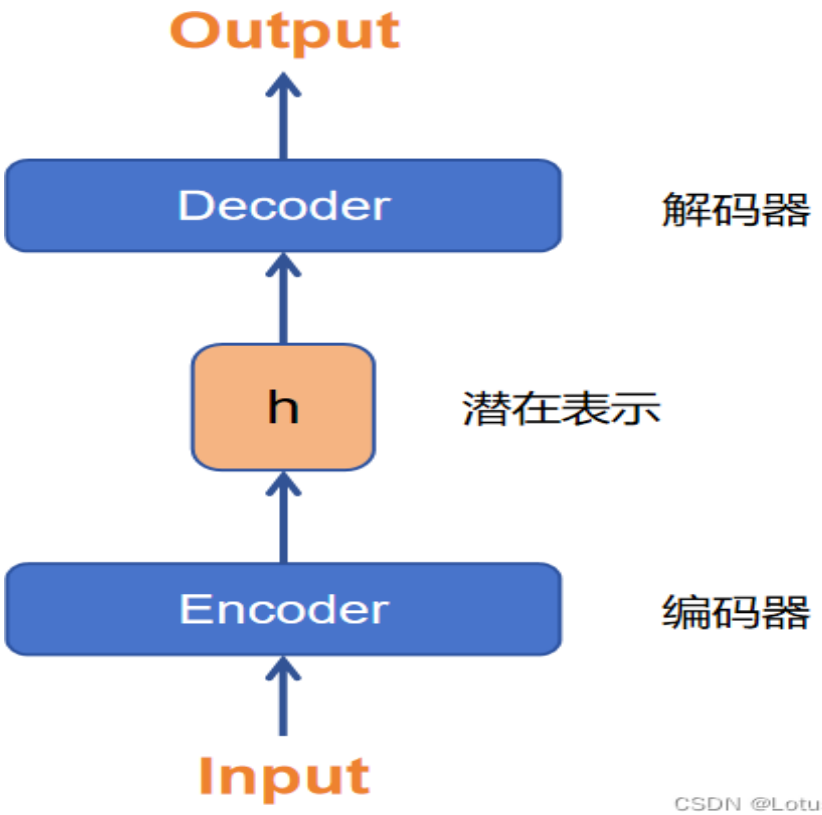
说话人保持和语音合成： 说话人特征和 Vec 特征可以用来保持原始声音中的个性化特征，或者用来实现不同说话人之间的声音转换，从而实现声音的个性化合成或转换。

五、模型设计

5.1 模型架构

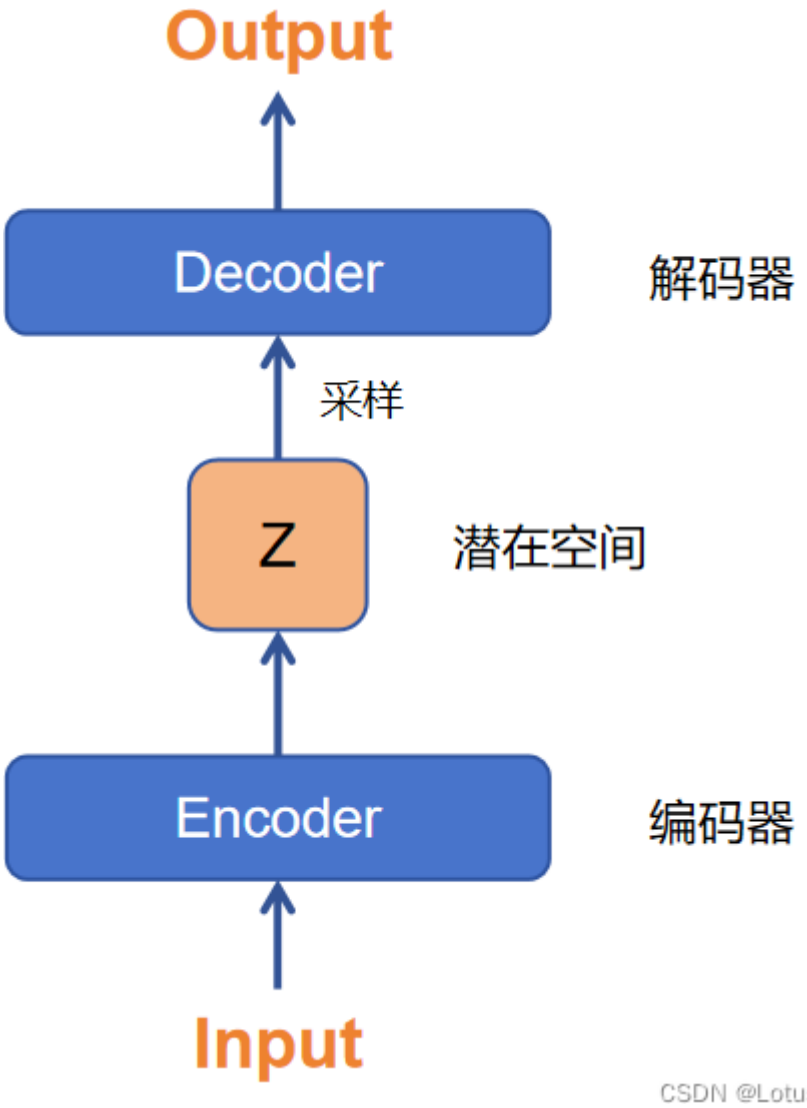
本组主要是采用从声音-声音的过程，主体使用 VAE（变分自编码）结构，AE（自编码器）是一种无监督学习的神经网络结构，它试图学习输入数据的紧凑表

示（编码），然后通过解码器将该表示还原为输入数据。自编码器包括一个编码器网络和一个解码器网络。



变分自编码器（VAE）和自编码器差别在哪里呢？最大的改变就是潜在表示。在自编码器中，潜在表示是一个固定值，而 VAE 中，潜在表示则是一个不确定变量，或者换一个说法，是一个概率分布。

在 VAE 中，编码器不再只学习提取输入数据的编码信息，而是去学习获取输入数据的概率分布。在原始论文中，我们设定这一概率分布为正态分布，模型也就变成了这样：



与此同时，中间量我们不再叫潜在变量，而是称为潜在空间（也称潜在分布，隐变量等），并使用 Z 来表示，用来凸显其是一个变化的量。
解码器要做的工作就是如何从这样的概率分布中采样并还原成最终的输出。

5.2 先验编码器

先验编码器的输入不是简单的文本，而是先前处理好的 3 种音频特征，分别是 PPG，音频中的文本特征。Vec，音频自编码特征，应用 softvc。Pitch，基频特征。

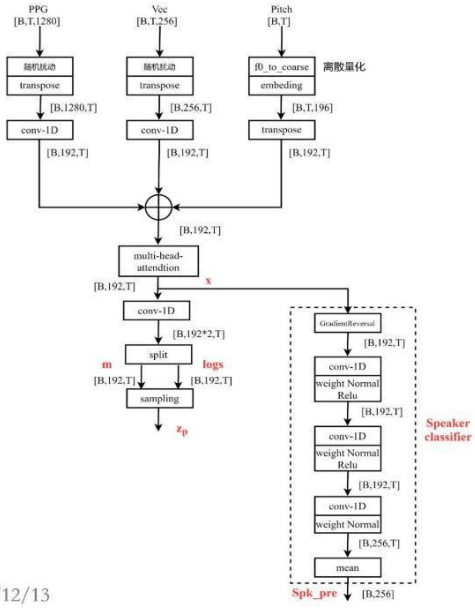
PPG 和 VEC 在预处理的时候，都加入随机扰动，模拟音频的随机性，读一段文字，文字是不会变的，但是多次读，音频是会有差异的。

然后把 3 种特征都处理成 size-192，求和作为输入。

Speaker classifier 说话人的预测器，让预测器预测出的 spk 和真实 spk 差别越大越好，所以这里加了梯度取反，给梯度加上负号，这样本来下降的，反而上升了。梯度取反部分代码如下：

```python
class GradientReversalFunction(Function):
    @staticmethod
    def forward(ctx, x, lambda_):
        ctx.lambda_ = lambda_
        return x.clone()

    @staticmethod
    def backward(ctx, grads):
        lambda_ = ctx.lambda_
        lambda_ = grads.new_tensor(lambda_)
        dx = -lambda_ * grads
        return dx, None


class GradientReversal(torch.nn.Module):
    ''' Gradient Reversal Layer
            Y. Ganin, V. Lempitsky,
            "Unsupervised Domain Adaptation by Backpropagation",
            in ICML, 2015.
        Forward pass is the identity function
        In the backward pass, upstream gradients are multiplied by -lambda (i.e. gradient are reversed)
    '''

    def __init__(self, lambda_reversal=1):
        super(GradientReversal, self).__init__()
        self.lambda_ = lambda_reversal

    def forward(self, x):
        return GradientReversalFunction.apply(x, self.lambda_)
```

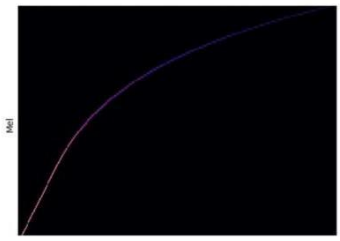
先验编码



随机扰动:

ppg = ppg + torch.randn_like(ppg) * 1
vec = vec + torch.randn_like(vec) * 2

f0->mel ->量化[0-255]　　mel =1127*ln(1+f/700)

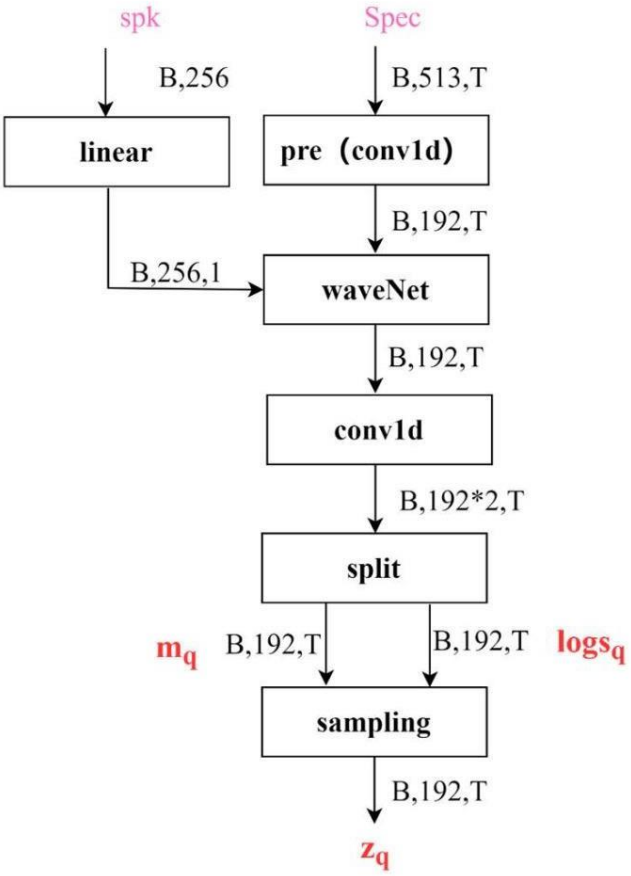


2023/12/13　　107

5.3 后验编码器

传入的说话人信息不是 id，是声纹特征。

Posterior Encoder
后验编码



```python
class PosteriorEncoder(nn.Module):
    def __init__(
        self,
        in_channels,
        out_channels,
        hidden_channels,
        kernel_size,
        dilation_rate,
        n_layers,
        gin_channels=0,
    ):
        super().__init__()
        self.out_channels = out_channels
        self.pre = nn.Conv1d(in_channels, hidden_channels, 1)
        self.enc = modules.WN(
            hidden_channels,
            kernel_size,
            dilation_rate,
            n_layers,
            gin_channels=gin_channels,
        )
        self.proj = nn.Conv1d(hidden_channels, out_channels * 2, 1)

    def forward(self, x, x_lengths, g=None):
        x_mask = torch.unsqueeze(commons.sequence_mask(x_lengths, x.size(2)), 1).to(
            x.dtype
        )
        x = self.pre(x) * x_mask
        x = self.enc(x, x_mask, g=g)
        stats = self.proj(x) * x_mask
        m, logs = torch.split(stats, self.out_channels, dim=1)
        z = (m + torch.randn_like(m) * torch.exp(logs)) * x_mask
        return z, m, logs, x_mask

    def remove_weight_norm(self):
        self.enc.remove_weight_norm()
```

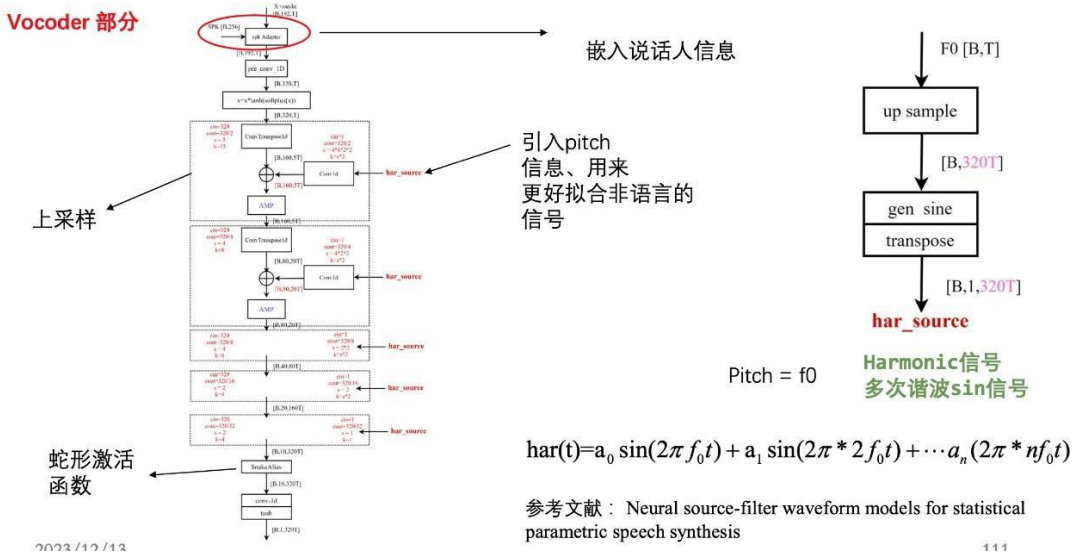
5.4 解码器

a. 加入说话人的声纹信息

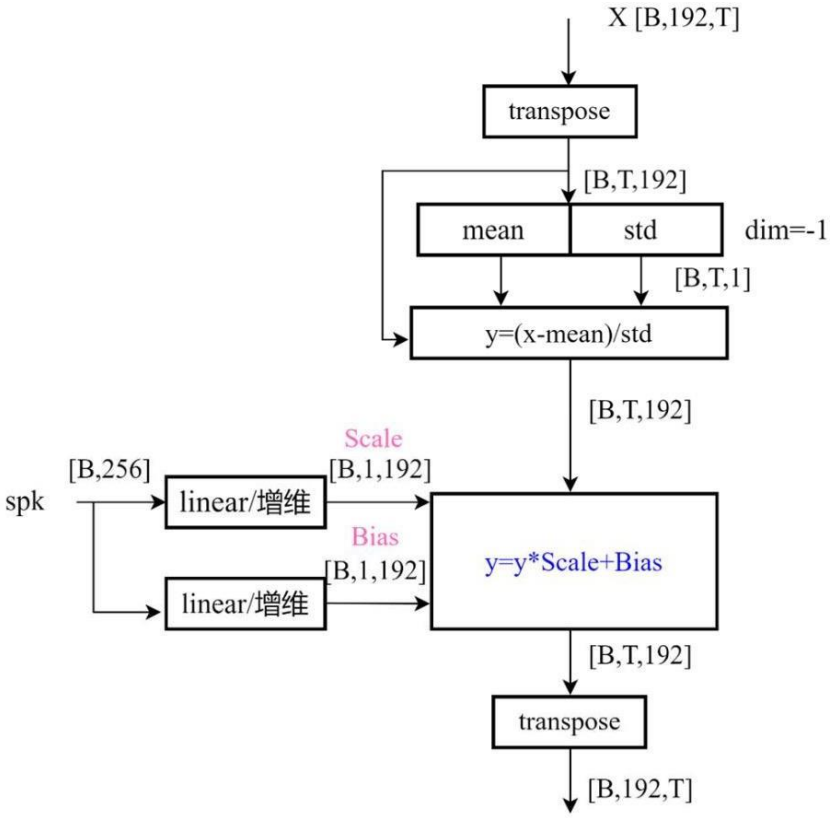
b. 上采样的每一步，加入基频 pitch 特征，用于更好的拟合非语音的声音

c. 蛇形激活函数，取代 relu，更好的适应音频的周期属性

基频 f0，经过上采样，copy*320，再加上多次谐波信号，形成 har_source

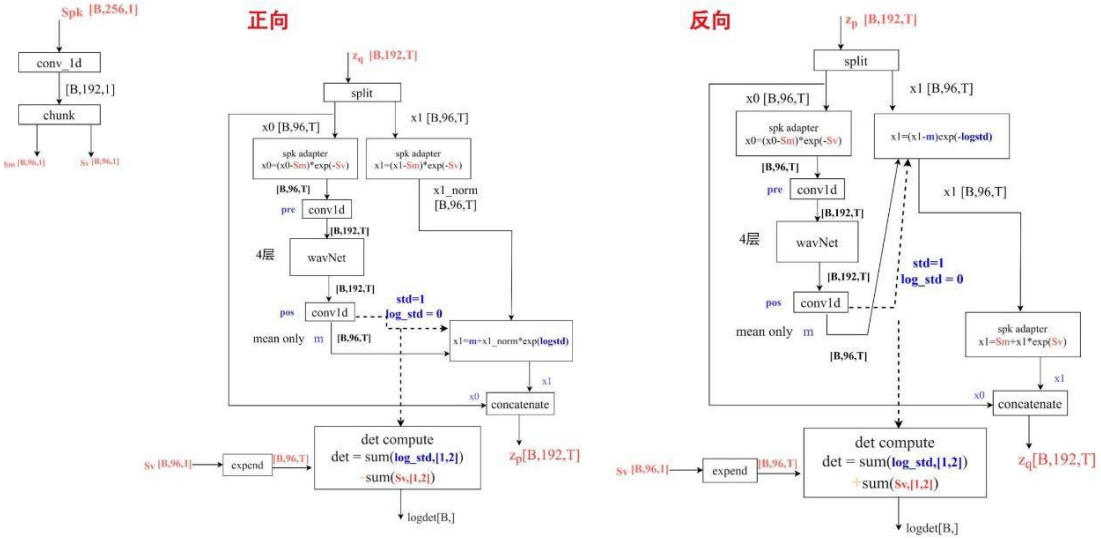


对于输入音频，在特征维度进行正则化，layernorm。对于 size-192 的特征维，求均值和方差，然后正则化 spk 声纹，线性变换成 scale 和 bias，用于叠加输入音频，达到加入声纹的目的。



5.5 Flow

输入的时候需要增加 skp 声纹信息 adapter,从 zq 到 zp，需要正则化掉 skp 的声纹信息，因为 p 分布本身是类似文本特征的分布，不包含 spk 声纹,从 zp 到 zq，需要添加目标声纹信息，用于 decode.



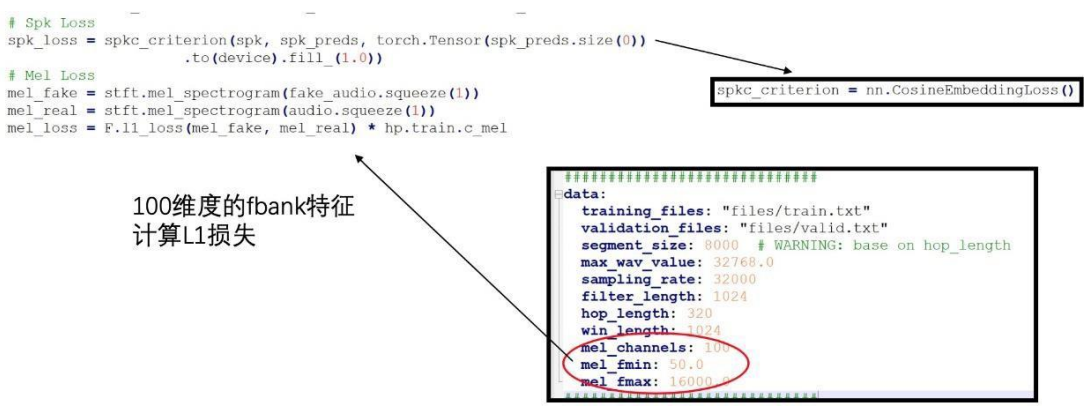
5.6 定义模型损失

D_loss 和G_loss，来自于对抗生成网络（GAN）：

```
# Loss
loss_g = score_loss + feat_loss + mel_loss + stft_loss + loss_kl_f + loss_kl_r * 0.5 + spk_loss * 2
loss_g.backward()
```

其中 score_loss 和 feat_loss，是 GAN 训练生成器的时候 ，把生成的数据进行判别得到的最终 score，以及中间层每层结果和 real 的 loss。

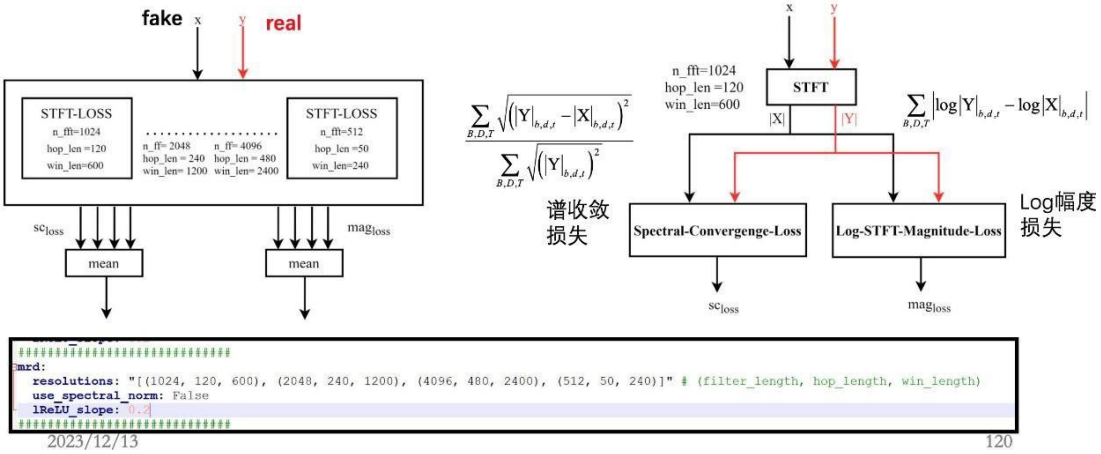
Spk_loss:使预测的 spk 与真实 spk 不相似。



mel_loss 和 stft_loss，是重构损失，mel_loss 就是把输入和生成的音频进行 mel 滤波后比较的 loss,stft_loss，在不同的尺度的 stft 变换下的，损失计算，更加全面 Score_loss(GAN)生成音频输入鉴别器，输出为 1

```
# Multi-Resolution STFT Loss
sc_loss, mag_loss = stft_criterion(fake_audio.squeeze(1), audio.squeeze(1))
stft_loss = (sc_loss + mag_loss) * hp.train.c_stft
```

多尺度STFT损失



2023/12/13 120

Feature_loss:生成音频和真实音频在鉴别器每层的输出都相似
KL_loss：分布 p 和分布 q 接近，分别计算 KL(p||q)和 KL(q||p)

```
# Kl Loss
loss_kl_f = kl_loss(z_f, logs_q, m_p, logs_p, logdet_f, z_mask) * hp.train.c_kl
loss_kl_r = kl_loss(z_r, logs_p, m_q, logs_q, logdet_r, z_mask) * hp.train.c_kl
```

```python
def kl_loss(z_p, logs_q, m_p, logs_p, total_logdet, z_mask):
    """
    z_p, logs_q: [b, h, t_t]
    m_p, logs_p: [b, h, t_t]
    total_logdet: [b] - total_logdet summed over each batch
    """
    z_p = z_p.float()
    logs_q = logs_q.float()
    m_p = m_p.float()
    logs_p = logs_p.float()
    z_mask = z_mask.float()

    kl = logs_p - logs_q - 0.5
    kl += 0.5 * ((z_p - m_p) ** 2) * torch.exp(-2.0 * logs_p)
    kl = torch.sum(kl * z_mask)
    # add total_logdet (Negative LL)
    kl -= torch.sum(total_logdet)
    l = kl / torch.sum(z_mask)
    return l
```

$$L_{kl}(q\|p) = E\left(\log q_\phi(z_q|x_{spec}) - \log p_\theta(z_q|x_{ppg,pitch,vec})\right)$$

$$z_q \sim N(z; m_q, \log s_q)$$

$$p_\theta(z_q|x_{ppg,pitch,vec}) = N(f_\theta(z_q); m_p, \log s_p)\left|\det\frac{\partial f_\theta(z_q)}{\partial z_q}\right|$$

$$\mathrm{E}\left[\log q_\phi(z_q|x_{spec})\right] = \int_{-\infty}^{+\infty} N(z_q; m_q, s_q)\log N(z_q; m_q, s_q)dz$$

$$= -\frac{1}{2} - \frac{1}{2}\log(2\pi) - \log s_q$$

$$\log N(z; m_p, s_p) = \log\frac{1}{\sqrt{2\pi}s_p}e^{-\frac{(z_p-m_p)^2}{2s_p^2}} = -\frac{1}{2}\log 2\pi - \log s_p - \frac{(z_p-m_p)^2}{2s_p^2}$$

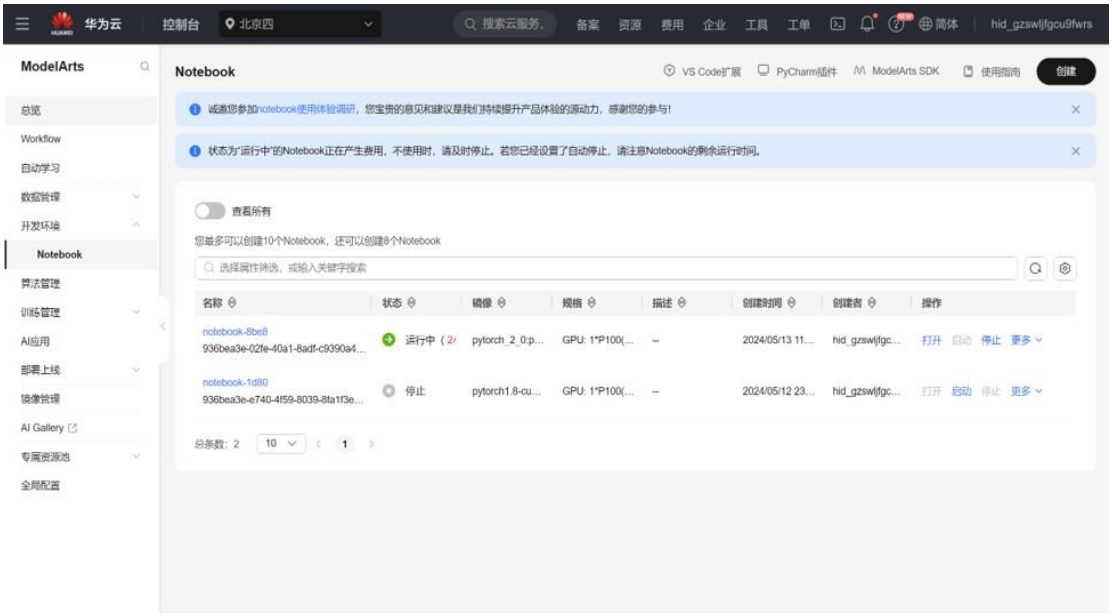
$$E\left(\log p_\theta(z_q|x_{ppg,pitch,vec})\right) = mean\left(\log\left(N(f_\theta(z_q); m_p, \log s_p)\right) + \log\left|\det\frac{\partial f_\theta(z_q)}{\partial z_q}\right|\right)$$

$$L_{kl}(q\|p) = mean\left(-\frac{1}{2} - \log s_q + \log s_p + \frac{(z_p-m_p)^2}{2s_p^2} - \log\left|\det\frac{\partial f_\theta(z_q)}{\partial z_q}\right|\right)$$

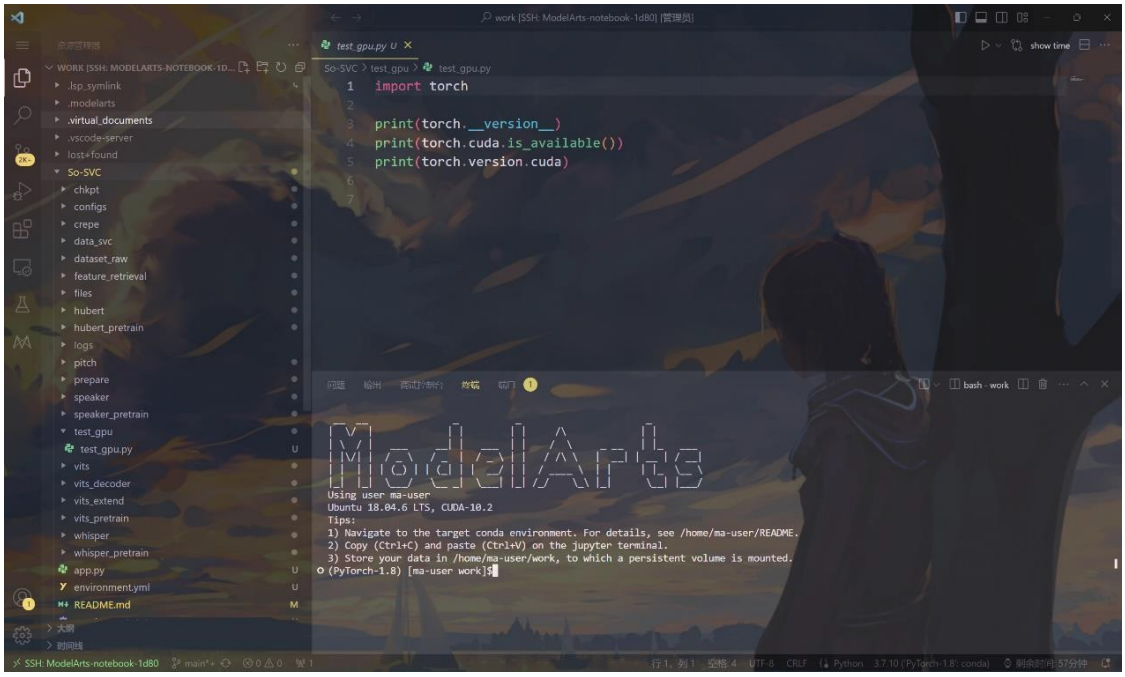
2023/12/13 124

六、实验结果

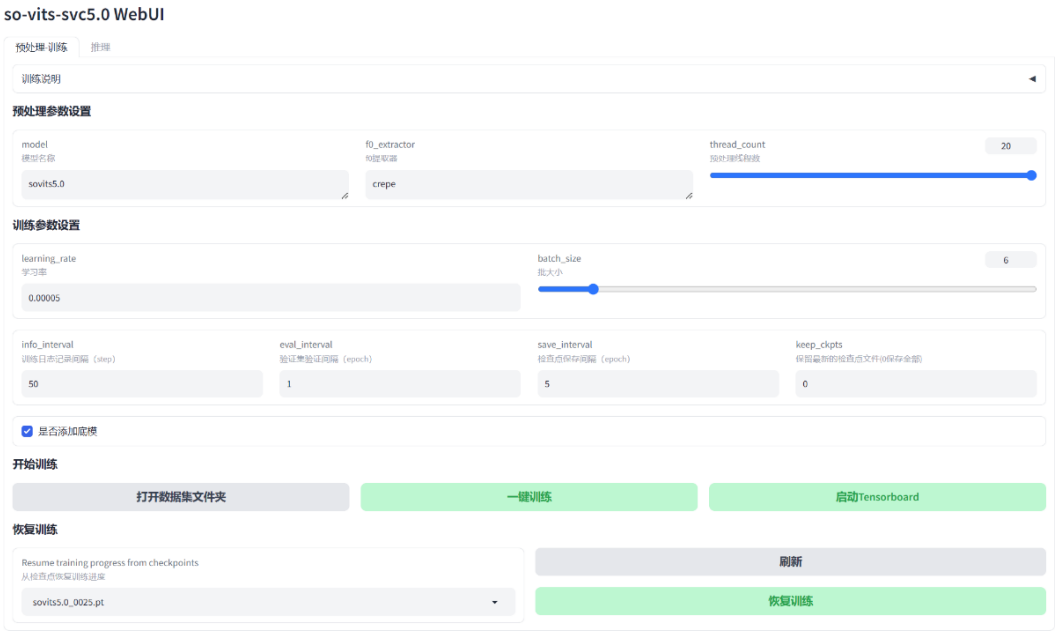
在华为云平台使用 notebook 平台（可以租用带 GPU 版本）：



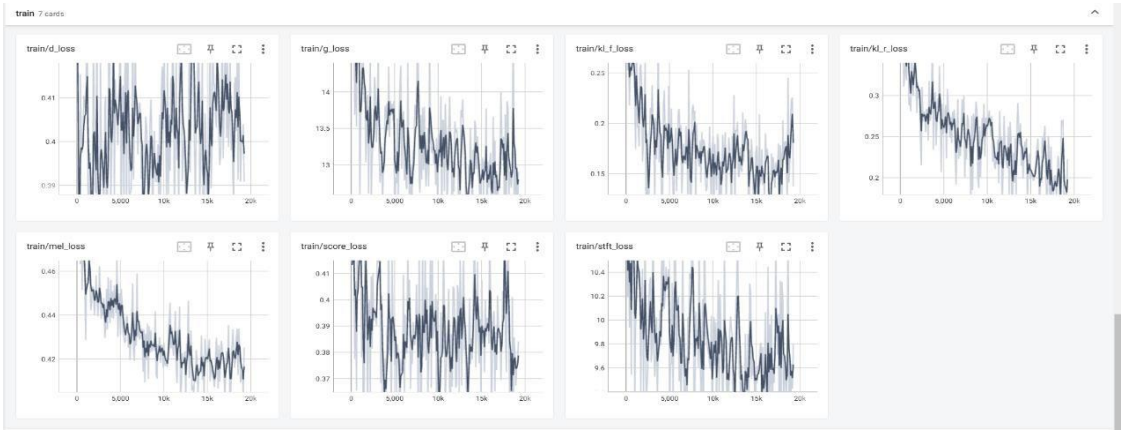
检查是否能够使用 GPU 功能：



可视化训练页面：



使用 tensorboard 实现训练日志可视化：



七、模型评估与实验总结

7.1 模型评估

对损失图像的分析:

1. train/d_loss（判别器损失）:

判别器损失的波动较大，但总体上有下降趋势。波动可能是因为训练过程中样本的多样性和对抗训练的固有性质。

2. train/g_loss（生成器损失）:

生成器损失也呈现下降趋势，但波动较大。这表明生成器在逐渐学习到更好的生成策略，但还需要更多的稳定训练。

3. train/k/l_f_loss 和 train/k/l_r_loss（Kullback-Leibler 散度损失）:

这两个损失总体上也在下降，说明模型的分布正在逐渐逼近目标分布。但同样存在较大的波动，需要更多的稳定训练。

4. train/mel_loss:

这个损失函数显示了音频特征的误差，下降趋势明显，表明模型在音频特征方面的学习效果较好。

5. train/score_loss:

评分损失波动较大，但总体趋势向下，表明模型的评分逐渐提高。

6. train/srft_loss:

同样有明显下降趋势，但波动大，表明模型还需要更多训练来提高稳定性。

总体而言，虽然各个损失函数的波动较大，但大多数损失函数都呈现下降趋势，表明模型在逐渐学习和优化。这是一个好的迹象，表明训练在朝着正确的方向前进。然而，波动较大可能表明需要更多的训练步骤或者调整学习率等超参数来提高训练的稳定性和收敛速度。

7.2 实验总结

项目优点：

1. 声音的自然度：

通过结合损失函数 Mel_loss 和 Stft_loss，模型能够在音频的波形和频谱特性上更好地逼近真实音频。Mel_loss 通过比较输入和生成的音频的 Mel 频谱，

有助于保持音频的音色和音质。而 Stft_loss 则进一步确保了在频域上的一致性，提高了音频的时间-频率分辨率。

2. 高灵活性：

svc 技术使用简单的损失函数或生成对抗网络（GAN）来重建声学特征，捕捉源歌手和目标歌手的声音特征，可以通过微调来适应特定的应用场景。

3. 良好的用户体验:

歌声转换技术采用了端到端的非并行结构，使用生成对抗网络（GAN）技术和迁移学习。这意味着整个转换过程可以从输入直接得到输出。

4. 应用的广泛性：

SVC 技术不仅可以应用于音乐制作，还可以用于 SVC 技术的应用范围可能会扩展到更多的媒体类型，如电影配音、虚拟角色配音、音乐制作等领域。它的灵活性和广泛的应用前景使其成为语音处理领域的一个热点。

项目缺点：

1. 算力平台成本高昂，导致训练迭代轮数有限、同时数据集规模较小，导致模型损失图像显示上波动较大、尚未稳定
2. 目前的使用需要用户提供转换目标音色文件或者编码捏造音色，用户门槛较高，最终目标应该让用户可以直接以录音方式实现声音转换

八、参考文献

[1]索罗格.[AI 孙燕姿]歌声转换技术原理浅析[EB/OL].https://zhuanlan.zhihu.com/p/631685001?utm_id=0,2020-09-24.

[2] Yiquan Zhou,Meng Chen,Yi Lei,Jihua Zhu,Weifeng Zhao .VITS-based Singing Voice Conversion System with DSPGAN post-processing for SVCC 2023[EB/OL].https://doi.org/10.48550/arXiv.2310.05118,2023-10-8.

九、附录

格式转换（重采样）：

```
def resample_wave(wav_in, wav_out, sample_rate):
    wav, _ = librosa.load(wav_in, sr=sample_rate)
    wav = wav / np.abs(wav).max() * 0.6
    wav = wav / max(0.01, np.max(np.abs(wav))) * 32767 * 0.6
    wavfile.write(wav_out, sample_rate, wav.astype(np.int16))


def process_file(file, wavPath, spks, outPath, sr):
    if file.endswith(".wav"):
        file = file[:-4]
        resample_wave(f"{wavPath}/{spks}/{file}.wav", f"
{outPath}/{spks}/{file}.wav", sr)


def process_files_with_thread_pool(wavPath, spks, outPath, sr, thread_num=None):
    files = [f for f in os.listdir(f"./{wavPath}/{spks}") if f.endswith(".wav")]

    with ThreadPoolExecutor(max_workers=thread_num) as executor:
        futures = {executor.submit(process_file, file, wavPath, spks, outPath,
sr): file for file in files}

        for future in tqdm(as_completed(futures), total=len(futures),
desc=f'Processing {sr} {spks}'):
            future.result()
```

pitch提取计算：

```
def compute_f0(path, save):
    x, sr = librosa.load(path, sr=16000)
    assert sr == 16000
    f0, t =
        pyworld.dio( x.astype(np.
        double), fs=sr,
        f0_ceil=900,
        frame_period=1000 * 160 / sr,
    )
    f0 = pyworld.stonemask(x.astype(np.double), f0, t, fs=16000)
    for index, pitch in enumerate(f0):
        f0[index] = round(pitch, 1)
    np.save(save, f0, allow_pickle=False)


def process_file(file, wavPath, spks, pitPath):
    if file.endswith(".wav"):
        file = file[:-4]
        compute_f0(f"{wavPath}/{spks}/{file}.wav", f"
{pitPath}/{spks}/{file}.pit")


def process_files_with_process_pool(wavPath, spks, pitPath, process_num=None):
    files = [f for f in os.listdir(f"./{wavPath}/{spks}") if f.endswith(".wav")]

    with ProcessPoolExecutor(max_workers=process_num) as executor:
```

```
        futures = {executor.submit(process_file, file, wavPath, spks, pitPath):
file for file in files}

        for future in tqdm(as_completed(futures), total=len(futures),
desc=f'Processing f0 {spks}'):
            future.result()
```

梯度反转代码：

```
class GradientReversalFunction(Function):
    @staticmethod
    def forward(ctx, x, lambda_):
        ctx.lambda_ = lambda_
        return x.clone()

    @staticmethod
    def backward(ctx, grads):
        lambda_ = ctx.lambda_
        lambda_ = grads.new_tensor(lambda_)
        dx = -lambda_ * grads
        return dx, None


class GradientReversal(torch.nn.Module):
    ''' Gradient Reversal Layer
            Y. Ganin, V. Lempitsky,
            "Unsupervised Domain Adaptation by Backpropagation",
            in ICML, 2015.
        Forward pass is the identity function
        In the backward pass, upstream gradients are multiplied by -lambda (i.e.
gradient are reversed)
    '''

    def __init__(self, lambda_reversal=1):
        super(GradientReversal, self).__init__()
        self.lambda_ = lambda_reversal

    def forward(self, x):
        return  GradientReversalFunction.apply(x,  self.lambda_)
```

说话人鉴别器：

```
class SpeakerClassifier(nn.Module):

    def __init__(self, embed_dim, spk_dim):
        super(SpeakerClassifier, self).__init__()
        self.classifier = nn.Sequential(
            GradientReversal(lambda_reversal=1),
            weight_norm(nn.Conv1d(embed_dim,  embed_dim,  kernel_size=5,
padding=2)),
            nn.ReLU(),
            weight_norm(nn.Conv1d(embed_dim, embed_dim, kernel_size=5,
padding=2)),
            nn.ReLU(),
```

```
            weight_norm(nn.Conv1d(embed_dim, spk_dim, kernel_size=5, padding=2))
        )

    def forward(self, x):
        ''' Forward function of Speaker Classifier:
            x = (B, embed_dim, len)
        '''
        # pass through classifier
        outputs = self.classifier(x) # (B, nb_speakers)
        outputs = torch.mean(outputs, dim=-1)
        return outputs
```

后验编码器:

```
class PosteriorEncoder(nn.Module):
    def __init__(
        self,
        in_channels,
        out_channels,
        hidden_channels,
        kernel_size,
        dilation_rate,
        n_layers,
        gin_channels=0,
    ):
        super().__init__()
        self.out_channels = out_channels
        self.pre = nn.Conv1d(in_channels, hidden_channels, 1)
        self.enc = modules.WN(
            hidden_channels,
            kernel_size,
            dilation_rate,
            n_layers,
            gin_channels=gin_channels,
        )
        self.proj = nn.Conv1d(hidden_channels, out_channels * 2, 1)

    def forward(self, x, x_lengths, g=None):
        x_mask = torch.unsqueeze(commons.sequence_mask(x_lengths, x.size(2)),
1).to(
            x.dtype
        )
        x = self.pre(x) * x_mask
        x = self.enc(x, x_mask, g=g)
        stats = self.proj(x) * x_mask
        m, logs = torch.split(stats, self.out_channels, dim=1)
        z = (m + torch.randn_like(m) * torch.exp(logs)) * x_mask
        return z, m, logs, x_mask

    def remove_weight_norm(self):
        self.enc.remove_weight_norm()
```

STFT_LOSS:

```
def stft(x, fft_size, hop_size, win_length, window):
    """Perform STFT and convert to magnitude spectrogram.
    Args:
        x (Tensor): Input signal tensor (B, T).
        fft_size (int): FFT size.
        hop_size (int): Hop size.
        win_length (int): Window length.
        window (str): Window function type.
    Returns:
        Tensor: Magnitude spectrogram (B, #frames, fft_size // 2 + 1).
    """
    x_stft = torch.stft(x, fft_size, hop_size, win_length, window,
return_complex=False)
    real = x_stft[..., 0]
    imag = x_stft[..., 1]

    # NOTE(kan-bayashi): clamp is needed to avoid nan or inf
    return torch.sqrt(torch.clamp(real ** 2 + imag ** 2, min=1e-7)).transpose(2,
1)


class SpectralConvergengeLoss(torch.nn.Module):
    """Spectral convergence loss module."""

    def __init__(self):
        """Initilize spectral convergence loss module."""
        super(SpectralConvergengeLoss, self).__init__()

    def forward(self, x_mag, y_mag):
        """Calculate forward propagation.
        Args:
            x_mag (Tensor): Magnitude spectrogram of predicted signal (B,
#frames, #freq_bins).
            y_mag (Tensor): Magnitude spectrogram of groundtruth signal (B,
#frames, #freq_bins).
        Returns:
            Tensor: Spectral convergence loss value.
        """
        return torch.norm(y_mag - x_mag, p="fro") / torch.norm(y_mag, p="fro")


class LogSTFTMagnitudeLoss(torch.nn.Module):
    """Log STFT magnitude loss module."""

    def __init__(self):
        """Initilize los STFT magnitude loss module."""
        super(LogSTFTMagnitudeLoss, self).__init__()

    def forward(self, x_mag, y_mag):
        """Calculate forward propagation.
        Args:
            x_mag (Tensor): Magnitude spectrogram of predicted signal (B,
#frames, #freq_bins).
            y_mag (Tensor): Magnitude spectrogram of groundtruth signal (B,
```

```
#frames, #freq_bins).
```

```
        Returns:
            Tensor: Log STFT magnitude loss value.
        """
        return  F.l1_loss(torch.log(y_mag),  torch.log(x_mag))


class STFTLoss(torch.nn.Module):
    """STFT loss module."""

    def __init__(self, device, fft_size=1024, shift_size=120, win_length=600,
window="hann_window"):
        """Initialize STFT loss module."""
        super(STFTLoss, self).__init__()
        self.fft_size = fft_size
        self.shift_size = shift_size
        self.win_length = win_length
        self.window = getattr(torch, window)(win_length).to(device)
        self.spectral_convergenge_loss = SpectralConvergengeLoss()
        self.log_stft_magnitude_loss = LogSTFTMagnitudeLoss()

    def forward(self, x, y):
        """Calculate forward propagation.
        Args:
            x (Tensor): Predicted signal (B, T).
            y (Tensor): Groundtruth signal (B, T).
        Returns:
            Tensor: Spectral convergence loss value.
            Tensor: Log STFT magnitude loss value.
        """
        x_mag = stft(x, self.fft_size, self.shift_size, self.win_length,
self.window)
        y_mag = stft(y, self.fft_size, self.shift_size, self.win_length,
self.window)
        sc_loss = self.spectral_convergenge_loss(x_mag, y_mag)
        mag_loss = self.log_stft_magnitude_loss(x_mag, y_mag)

        return sc_loss, mag_loss


class MultiResolutionSTFTLoss(torch.nn.Module):
    """Multi resolution STFT loss module."""

    def __init__(self,
                 device,
                 resolutions,
                 window="hann_window"):
        """Initialize Multi resolution STFT loss module.
        Args:
            resolutions (list): List of (FFT size, hop size, window length).
            window (str): Window function type.
        """
        super(MultiResolutionSTFTLoss, self).__init__()
        self.stft_losses = torch.nn.ModuleList()
        for fs, ss, wl in resolutions:
            self.stft_losses += [STFTLoss(device, fs, ss, wl, window)]
```

```
    def forward(self, x, y):
        """Calculate forward propagation.
        Args:
            x (Tensor): Predicted signal (B, T).
            y (Tensor): Groundtruth signal (B, T).
        Returns:
            Tensor: Multi resolution spectral convergence loss value.
            Tensor: Multi resolution log STFT magnitude loss value.
        """
        sc_loss = 0.0
        mag_loss = 0.0
        for f in self.stft_losses:
            sc_l, mag_l = f(x, y)
            sc_loss += sc_l
            mag_loss += mag_l

        sc_loss /= len(self.stft_losses)
        mag_loss /= len(self.stft_losses)

        return sc_loss, mag_loss
```

trainer:

```
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('-c', '--config', type=str, required=True,
                        help="yaml file for configuration")
    parser.add_argument('-p', '--checkpoint_path', type=str, default=None,
                        help="path of checkpoint pt file to resume training")
    parser.add_argument('-n', '--name', type=str, required=True,
                        help="name of the model for logging, saving checkpoint")
    args = parser.parse_args()

    hp = OmegaConf.load(args.config)
    with open(args.config, 'r') as f:
        hp_str = ''.join(f.readlines())

    assert hp.data.hop_length == 320, \
        'hp.data.hop_length must be equal to 320, got %d' % hp.data.hop_length

    args.num_gpus = 0
    torch.manual_seed(hp.train.seed)
    if torch.cuda.is_available():
        torch.cuda.manual_seed(hp.train.seed)
        args.num_gpus = torch.cuda.device_count()
        print('Batch size per GPU :', hp.train.batch_size)

        if args.num_gpus > 1:
            mp.spawn(train, nprocs=args.num_gpus,
                     args=(args, args.checkpoint_path, hp, hp_str,))
        else:
            train(0, args, args.checkpoint_path, hp, hp_str)
    else:
        print('No GPU find!')
```