基于 MindSpore 的 MelGAN 网络实现语音生成实验

一、实验目的

- 掌握 MelGAN 网络模型结构。
- 掌握 MindSpore 模型训练的流程。

二、实验清单

实验	简述	开发环境
MelGAN网络语音生成实验	本实验用LJ Speech数据集，实现MindSpore在语音生成应用	ModelArts：MindSpore2.1，Python3.7，Ascend 910 +ARM

三、开发平台搭建

请参考《华为云 ModelArts 环境搭建手册》完成云上环境搭建。

华为云ModelArts环境搭建手册.doc›

特别注意：本实验中 ModelArts 环境搭建----创建 Notebook 时的配置规格，需特别注意存储配置选择：

云硬盘 EVS（80GB）

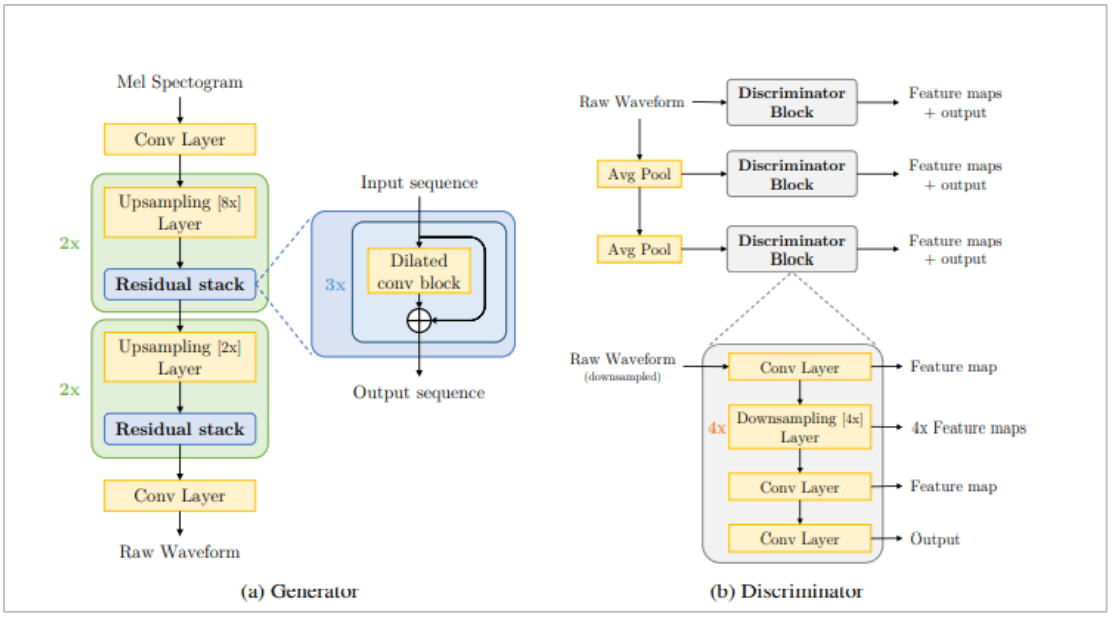
如图所示：

Notebook 创建配置时需特别调节存储规格

四、模型介绍

MelGAN 是一种 GAN 网络，可将音频 Mel 谱特征转化为高质量的音频。该网络不需要任何硬件优化技巧，就可以实现快速的音频合成。对比于相同功能的 Wavenet 网络，速度提高 1000 倍以上。

MelGAN 是基于 GAN 实现的，由生成器和判别器组成，整体结构如下图：



MelGAN 网络结构图

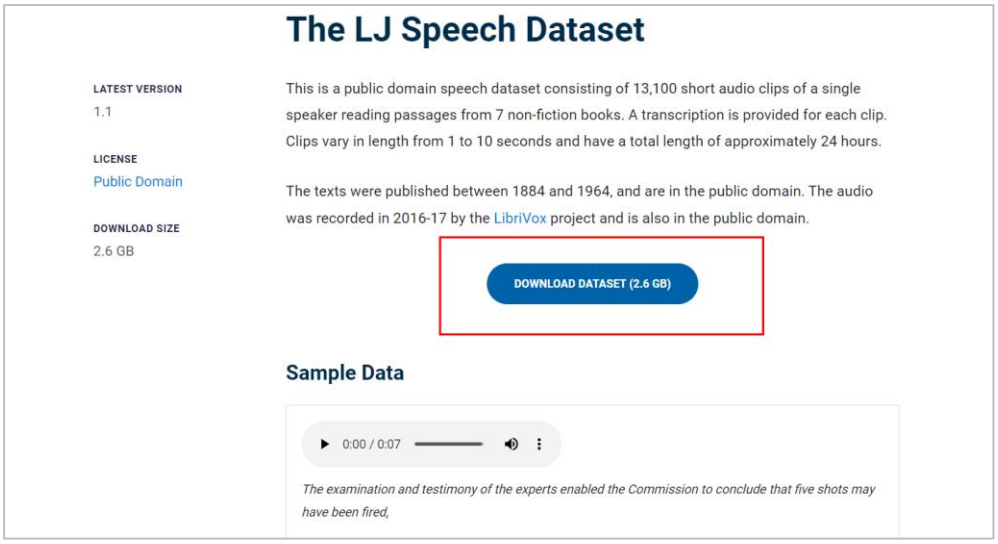
论文: Kundan Kumar, Rithesh Kumar, Thibault de Boissiere, Lucas Gestin, Wei Zhen Teoh, Jose Sotelo, Alexandre de Brebisson, Yoshua Bengio, Aaron Courville. MelGAN: Generative Adversarial Networks for Conditional Waveform Synthesis.

MelGAN 模型是非自回归全卷积模型。它的参数比同类模型少得多，并且对于看不见的说话人也有很好的效果。它的生成器由 4 个上采样层和 4 个残差堆栈组成，而判别器是多尺度架构。本实验网络模型和文章设计的结构不一样的是，我们修改了判别器中部分卷积核的大小，同时我们使用一维卷积代替了判别器中的 avpool。

五、LJSpeech 数据集

Dataset size：2.6GB，包含 13,100 条只有一个说话人的短语音。语音的内容来自 7 本纪实书籍。

数据格式：每条语音文件都是单声道、16-bit 以及采样率为 22050。



数据集下载界面

本实验需要把 LJ Speech 数据集原 wav 格式处理成 Mel 谱；下载链接提供的数据集已经完成格式转换，并且被划分成训练集、验证集、测试集，请直接使用。

关于数据如何处理成 Mel 谱，可以参考：

https://github.com/seungwonpark/MelGAN/blob/master/preprocess.py；

```
%env no_proxy='a.test.com,127.0.0.1,2.2.2.2'
!wget https://ascend-professional-construction-dataset.obs.cn-north-4.myhuaweicloud.com:443/ASR/melgan/LJSpeech-1.1.tar.gz
!tar -xvzf LJSpeech-1.1.tar.gz
```

六、实验过程

1. 环境准备

请参考“开发平台环境搭建”章节，先完成 ModelArts 环境搭建，本章节均基于 ModelArts-Jupyter Notebook 完成。

2. 模型训练流程

下载项目代码：使用 git 从 modelzoo 下载训练脚本的源码。

```
!git clone -b r2.1 https://gitee.com/mindspore/models.git
```

本实验 melgan 项目代码位于 models/official/audio/MELGAN。

项目文件结构（本实验所用代码，均以加粗显示）

```
├── MELGAN
  ├── README.md              // melgan 说明
  ├── README_CN.md           // melgan 中文说明
  ├── ascend310_infer        // 实现 310 推理源代码
    ├── scripts
```

```
  ├──run_standalone_train_ascend       // 启动 Ascend 单机训练
  ├──run_distribute_train_ascend.sh    // 启动 Ascend 分布式训练（8 卡）
  ├──run_eval_ascend.sh                // 启动评估
  ├──run_infer_310.sh                  // 启动 310 评估
├── src
  ├──dataset.py       // 创建数据集
  ├──model.py         // 生成器和判别器网络结构
  ├──loss.py          // 计算损失函数
  ├── model_utils
    ├──config.py                // 参数配置
    ├──device_adapter.py        // 设备配置
    ├──local_adapter.py         // 本地设备配置
    ├──moxing_adapter.py        // modelarts 设备配置
├── train.py          // 训练网络脚本
├── eval.py           // 评估网络脚本
├── config.yaml       // 参数配置项
  ├── export.py         // 将 checkpoint 文件导出到 air/mindir
```

下载预训练模型文件

```
!wget https://ascend-professional-construction-dataset.obs.cn-north-4.myhuaweicloud.com/ASR/melgan/basemodel.ckpt
```

在开发环境下载该模型，用于微调训练，可缩短训练时间，获得较好的推理结果。

3. 修改配置文件参数

参数文件为 models/official/audio/MELGAN/config.yaml 可以同时配置训练参数和评估参数，可根据实际开发环境进行修改。

```
enable_modelarts: False # 是否使用 modelarts
network: "melgan"  #网络模型
# Url for modelarts
data_url: ""  #modelarts 数据地址
train_url: "" #train 数据地址
checkpoint_url: ""
# Path for local
run_distribute: 0
enable_profiling: False
data_path: "/cache/data"
output_path: "/cache/train"
load_path: "/cache/checkpoint_path/"
device_target: "Ascend" #芯片类型
checkpoint_path: "./checkpoint/"
checkpoint_file_path: ""
# ==============================================================================
```

```
# Training options
device_id: 1        #芯片 id
'pre_trained': 'Flase'   # 是否基于预训练模型训练
'checkpoint_path':  './melgan_20-215_176000.ckpt'   # 预训练模型路径
'lr_g': 0.0001        # 生成器初始学习率
'lr_d': 0.0001        # 判别器初始学习率
'batch_size': 4       # 训练批次大小（使用单卡训练时可适当增大为 32）
'epoch_size': 5000      # 总训练 epoch 数
'momentum': 0.9       # 权重衰减值
'leaky_alpha': 0.2      # leaky relu 参数
'train_length': 32      # 训练时输入序列的帧数(最大值:240)
'beta1':0.9           # 第一矩估计的指数衰减率
'beta2':0.999         # 第二矩估计的指数衰减率
'weight_decay':0.0      # 权重衰减值（L2 惩罚）
'hop_size': 256       # Mel 谱中一帧的长度
'mel_num': 80         # Mel 谱中通道数
'filter_length': 1024    # n 点短时傅里叶变换
'win_length': 1024      # 窗函数长度
'segment_length': 16000   # 计算 Mel 谱时的最大长度
'sample': 22050       # 训练音频采样率
'data_path':'/home/datadisko/voice/melgan/data/'   # 训练数据绝对路径
'save_steps': 4000      # 保存点间隔.
'save_checkpoint_name': 'melgan'  # 保存模型的名字.
'save_checkpoint_path': './saved_model'  # 保存模型的绝对路径
'eval_data_path': '/home/datadisko/voice/melgan/val_data/'  # 验证集绝对路径
'eval_model_path': './melgan_20-215_176000.ckpt'# 验证模型路径
'output_path': 'output/'  # 验证结果保存路径
'eval_length': 240      # 验证时输入序列的帧数 (最大值:240)
```

本实验在 modelarts 上采用昇腾 910 单卡芯片，模型训练时，修改如下参数：

- device_id： 0 # **设备 id，原值为 1，本实验是单卡训练，故改为 0**
- pre_trained： True# 是否基于预训练模型训练，本实验，选 True
- checkpoint_path: **'/home/ma-user/work/basemodel.ckpt'** # 预训练模型路径(模型来源于 OBS 桶)
- **batch_size： 32 #训练批次大小**
- **epoch_size： 20 #（大概用时 20min）**
- data_path：**'/home/ma-user/work/LJSpeech-1.1/train'** #数据集绝对路径
- save_steps： **1000 #**（运行多少 steps 保存一个模型文件，可以自己设置）

本实验在 modelarts 上模型评估时，需要修改如下参数：

- eval_data_path: '/home/ma-user/work/LJSpeech-1.1/val/mels' # 验证集绝对路径
- eval_model_path: '/home/ma-user/work/models/official/audio/MELGAN/saved_model/melgan-18_5000.ckpt' # 验证模型绝对路径

备注：pre_trained: False or True；设置为 False，从头训练，epoch_size :100（大概用时 1.5h，但经验表明 300-500 方可有清晰语音生成；设置为：True，做微调训练，需要下载预训练模型 basemodel.ckpt,epoch_size 设置为 20（大概用时 20Min,生成的语音比较清晰）;。本实验采用微调训练，一方面可以缩短训练时间，另外一方面是获得较好的语音生成文件。

4. 模型训练

MindSpore 模型训练需调用如下脚本：

- /src/dataset.py： 对数据集进行处理
- /src/model.py ： 生成器和判别器网络结构
- /src/loss.py ： 计算损失函数
- /src/ model_utils/config.py： 参数配置
- /src/ model_utils/moing_adapter.py :modelarts 设备配置
- train.py： 训练网络脚本

train.py 文件内容如下，可以根据实际开发情况进行修改。

- mindspord.nn 当中主要会包括网络可能涉及到的各类网络层，诸如卷积层、池化层、全连接层，也包括损失函数，激活函数等。mindspore.train.callback 下面会涉及到各类回调函数，如 checkpoint,lossMonitor 等，主要用于在每个 epoch 训练完的时候自动执行。
- mindspore.common 包中会有诸如 type 形态转变、权重初始化等的常规工具。
- mindspore.dataset 数据集的载入与处理，也可以自定义数据集。
- mindspore.Tensor 提供了 mindspore 网络可用的张量，context 用于设定 mindspore 的运行环境与运行设备，Model 用来承载网络结构，并能够调用优化器，损失函数，评价指标。

```
"""MelGAN train"""
import time
import os
#导入 mindspore 相关包
import mindspore.nn as nn
from mindspore.common import set_seed
import mindspore.common.dtype as mstype
from mindspore.common.tensor import Tensor
from mindspore.context  import ParallelMode
from mindspore.communication.management import init, get_rank, get_group_size
import mindspore.dataset as de
import mindspore.context as context
from mindspore.train.loss_scale_manager import DynamicLossScaleManager
```

```
from mindspore.train.callback import RunContext, ModelCheckpoint, CheckpointConfig,
_InternalCallbackParam
from mindspore.train.serialization import load_checkpoint, load_param_into_net

from src.model_utils.config import config as cfg
from src.model import MultiDiscriminator, Generator
from src.trainonestep import TrainOneStepCellGEN, TrainOneStepCellDIS
from src.loss import MelganLoss_G, MelganLoss_D
from src.dataset import Generator1D
from src.sampler import DistributedSampler
from src.model_utils.moxing_adapter import moxing_wrapper

set_seed(1)
#定义生成器
class BuildGenNetwork(nn.Cell):
    """build generator"""
    def __init__(self, network, criterion):
        super(BuildGenNetwork, self).__init__(auto_prefix=False)
        self.network = network
        self.criterion = criterion
    def construct(self, data):
        fake_wav = self.network(data)
        return fake_wav
#定义判别器
class BuildDisNetwork(nn.Cell):
    """build discriminator"""
    def __init__(self, network, criterion):
        super(BuildDisNetwork, self).__init__(auto_prefix=False)
        self.network = network
        self.criterion = criterion

    def construct(self, fake_wav, wav):
        y1 = self.network(fake_wav)
        y2 = self.network(wav)
        loss = self.criterion(y1, y2)
        return loss

@moxing_wrapper()
def train():
    """main train process"""
    # init distributed
    if cfg.run_distribute:
        device_id = int(os.getenv('DEVICE_ID', '0'))
        context.set_context(mode=context.GRAPH_MODE, device_target="Ascend", device_id=device_id)
        init()
        cfg.rank = get_rank()
        cfg.group_size = get_group_size()
        context.reset_auto_parallel_context()
```

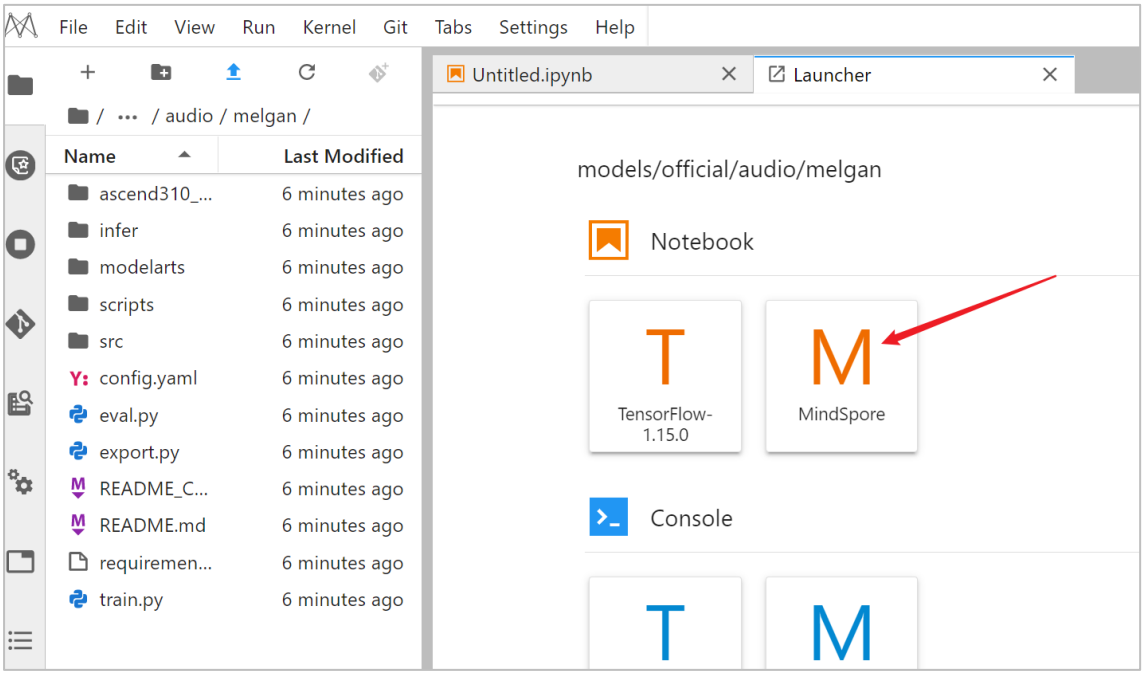
```
    context.set_auto_parallel_context(parallel_mode=ParallelMode.DATA_PARALLEL, gradients_mean=True,
device_num=8,
                                      parameter_broadcast=True)
  else:
    cfg.rank = 0
    cfg.group_size = 1
    context.set_context(mode=context.GRAPH_MODE, device_target="Ascend", device_id=cfg.device_id)
     # get network and init
     #网络初始化
  net_D = MultiDiscriminator()
  net_G = Generator(alpha=cfg.leaky_alpha)

  criterion_G = MelganLoss_G()
  criterion_D = MelganLoss_D()

  gen_network_train = BuildGenNetwork(net_G, criterion_G)
  gen_network_train.set_train()
  dis_network_train_1 = BuildDisNetwork(net_D, criterion_G)
  dis_network_train_1.set_train()
  dis_network_train_2 = BuildDisNetwork(net_D, criterion_D)
  dis_network_train_2.set_train()
  scale_manager = DynamicLossScaleManager(init_loss_scale=2 ** 10, scale_factor=2, scale_window=2000)

     # optimizer
     #Adam 优化器
  opt_G = nn.Adam(params=net_G.trainable_params(), learning_rate=cfg.lr_g, beta1=cfg.beta1,
beta2=cfg.beta2,
            weight_decay=cfg.weight_decay)
  opt_D = nn.Adam(params=net_D.trainable_params(), learning_rate=cfg.lr_d, beta1=cfg.beta1,
beta2=cfg.beta2,
            weight_decay=cfg.weight_decay)
  if cfg.pre_trained:
    param_dict = load_checkpoint(cfg.checkpoint_path)
    load_param_into_net(net_G, param_dict)
    load_param_into_net(net_D, param_dict)

  gen_network_train_wrap = TrainOneStepCellGEN(gen_network_train, opt_G, dis_network_train_1,
criterion_G)
  dis_network_train_wrap = TrainOneStepCellDIS(gen_network_train, dis_network_train_2, opt_D,
criterion_D)

  # dataloader
  Wavmeldataset = Generator1D(cfg.data_path, cfg.train_length, cfg.hop_size)
  distributed_sampler = DistributedSampler(len(Wavmeldataset), cfg.group_size, cfg.rank, shuffle=True)
  dataset = de.GeneratorDataset(Wavmeldataset, ["data", "wav", "datad", "wavd"],
sampler=distributed_sampler)
  dataset = dataset.batch(cfg.batch_size, drop_remainder=True)

  # checkpoint save
  config_ck = CheckpointConfig(save_checkpoint_steps=cfg.save_steps, keep_checkpoint_max=100000)
```

```
    ckpt_cb = ModelCheckpoint(prefix=cfg.save_checkpoint_name, directory=cfg.save_checkpoint_path,
config=config_ck)
    cb_params = _InternalCallbackParam()
    cb_params.train_network = gen_network_train_wrap
    cb_params.epoch_num = cfg.epoch_size
    run_context = RunContext(cb_params)
    ckpt_cb.begin(run_context)

    i = 1
    print(cfg.epoch_size)
    epoch_t = time.perf_counter()

    # epoch loop
    for epoch in range(cfg.epoch_size):
      cb_params.cur_epoch_num = epoch + 1
      for data, wav, datad, wavd in dataset.create_tuple_iterator():
        scaling_sens = Tensor(scale_manager.get_loss_scale(), dtype=mstype.float32)
        start = time.perf_counter()
        data = (data + 5.0) / 5.0
        datad = (datad + 5.0) / 5.0

        _, loss_G, cond_g = gen_network_train_wrap(Tensor(wav, mstype.float32), Tensor(data, mstype.float32),
                                   scaling_sens)

        _, loss_D, cond_d = dis_network_train_wrap(Tensor(datad, mstype.float32), Tensor(wavd,
mstype.float32),
                                   scaling_sens)
        if cond_g:
          scale_manager.update_loss_scale(cond_g)
        else:
          scale_manager.update_loss_scale(False)
        if cond_d:
          scale_manager.update_loss_scale(cond_d)
        else:
          scale_manager.update_loss_scale(False)
        duration = time.perf_counter() - start

        print('{}epoch {}iter loss_G={} loss_D={} {:.2f}s/it'.format(epoch+1, i, loss_G.asnumpy(), loss_D.asnumpy(),
                                          duration))

        i = i + 1
        if cfg.rank == 0:
          cb_params.cur_step_num = i
          cb_params.batch_num = i
          ckpt_cb.step_end(run_context)

    duration = time.perf_counter() - epoch_t
    print('finish in {:.2f}mins'.format(duration / 60))

if __name__ == "__main__":
```

```
train()
```

运行脚本，进行模型训练。

在 models/official/audio/MELGAN 下新建 notebook 运行 python 脚本。




```
!python train.py
```

训练日志：其中 loss_G 是生成器损失值，loss_D 是判别器损失值

```
1epoch 1iter loss_G=184.332275390625 loss_D=11.07717514038086 216.07s/it
1epoch 2iter loss_G=177.47218322753906 loss_D=10.63742733001709 0.19s/it
1epoch 3iter loss_G=170.89083862304688 loss_D=7.081489562988281 0.18s/it
1epoch 4iter loss_G=178.0736541748047 loss_D=6.877992153167725 0.18s/it
1epoch 5iter loss_G=183.238647609375 loss_D=5.559493541717529 0.18s/it
1epoch 6iter loss_G=203.33578491210938 loss_D=9.262943267822266 0.18s/it
1epoch 7iter loss_G=197.37368774414062 loss_D=9.5786981582641 6 0.18s/it
.....
20epoch 5717iter loss_G=168.9973907470703 loss_D=10.959349632263184 0.18s/it
20epoch 5718iter loss_G=164.40310668945312 loss_D=11.20811653137207 0.18s/it
20epoch 5719iter loss_G=170.856201171875 loss_D=8.815239906311035 0.18s/it
20epoch 5720iter loss_G=179.6353302001953 loss_D=10.675108909606934 0.18s/it
finish in 21.35mins
```

5. 模型评估

MindSpore 模型评估需要调用如下脚本：

- /src/ model_utils/config.py： 参数配置
- /src/ model_utils/moing_adapter.py :modelarts 设备配置
- eval.py

eval.py 文件内容如下，可根据实际情况进行修改。

- Model 是 MindSpore 提供的模型训练高阶 API，可以进行模型训练、评估和推理。
- model.eval 接口进行模型验证
- mindspore 模块的 load_checkpoint 和 load_param_into_net 从本地加载模型与参数，传入验证数据集后即可进行模型推理，验证数据集的处理方式与训练数据集相同。
- model.predict 为推理接口

```
"""MelGAN eval"""
import os
import numpy as np
from scipy.io.wavfile import write

from mindspore import Model
from mindspore.train.serialization import load_checkpoint, load_param_into_net
from mindspore.common.tensor import Tensor
import mindspore.context as context

from src.model import Generator
from src.model_utils.config import config as cfg

context.set_context(mode=context.GRAPH_MODE, device_target="Ascend")


if __name__ == '__main__':
    context.set_context(device_id=cfg.device_id)
    if not os.path.exists(cfg.output_path):
        os.mkdir(cfg.output_path)

    net_G = Generator(alpha=cfg.leaky_alpha)
    net_G.set_train(False)

    # load checkpoint
    print('ooooooooooooooooo',cfg.eval_model_path)
    param_dict = load_checkpoint(cfg.eval_model_path)
    load_param_into_net(net_G, param_dict)
    print('load model done !')

    model = Model(net_G)

    # get list
    mel_path = cfg.eval_data_path
    data_list = os.listdir(mel_path)

    for data_name in data_list:

        melpath = os.path.join(mel_path, data_name)
```

```
# data preprocessing
meldata = np.load(melpath)
meldata = (meldata + 5.0) / 5.0
pad_node = 0
if meldata.shape[1] < cfg.eval_length:
  pad_node = cfg.eval_length - meldata.shape[1]
  meldata = np.pad(meldata, ((0, 0), (0, pad_node)), mode='constant', constant_values=0.0)
meldata_s = meldata[np.newaxis, :, 0:cfg.eval_length]

# first frame
#第一帧
wav_data = np.array([])
output = model.predict(Tensor(meldata_s)).asnumpy().ravel()
wav_data = np.concatenate((wav_data, output))

# initialization parameters
#初始化参数
repeat_frame = cfg.eval_length // 8
i = cfg.eval_length - repeat_frame
length = cfg.eval_length
num_weights = i
interval = (cfg.hop_size*repeat_frame) // num_weights
weights = np.linspace(0.0, 1.0, num_weights)

while i < meldata.shape[1]:
  # data preprocessing
  meldata_s = meldata[:, i:i+length]
  if meldata_s.shape[1] != cfg.eval_length:
    pad_node = cfg.hop_size * (cfg.eval_length-meldata_s.shape[1])
    meldata_s = np.pad(meldata_s, ((0, 0), (0, cfg.eval_length-meldata_s.shape[1])), mode='edge')
  meldata_s = meldata_s[np.newaxis, :, :]

  # i-th frame
  # 第 i 帧
  output = model.predict(Tensor(meldata_s)).asnumpy().ravel()
  print('output{}={}'.format(i, output))
  lenwav = cfg.hop_size*repeat_frame
  lenout = 0

  # overlap
  for j in range(num_weights-1):
    wav_data[-lenwav:-lenwav+interval] = weights[-j-1] * wav_data[-lenwav:-lenwav+interval] +\
                  weights[j] * output[lenout:lenout+interval]
    lenwav = lenwav - interval
    lenout = lenout + interval
  wav_data[-lenwav:] = weights[-num_weights] * wav_data[-lenwav:] +\
          weights[num_weights-1] * output[lenout:lenout+lenwav]
  wav_data = np.concatenate((wav_data, output[cfg.hop_size*repeat_frame:]))
  i = i + length - repeat_frame
```

运行脚本，进行模型评估

同时在 output 文件夹下生成语音 wav，可以下载到本地试听。

模型评估示例：

```
python eval.py
```

本实验运行脚本：

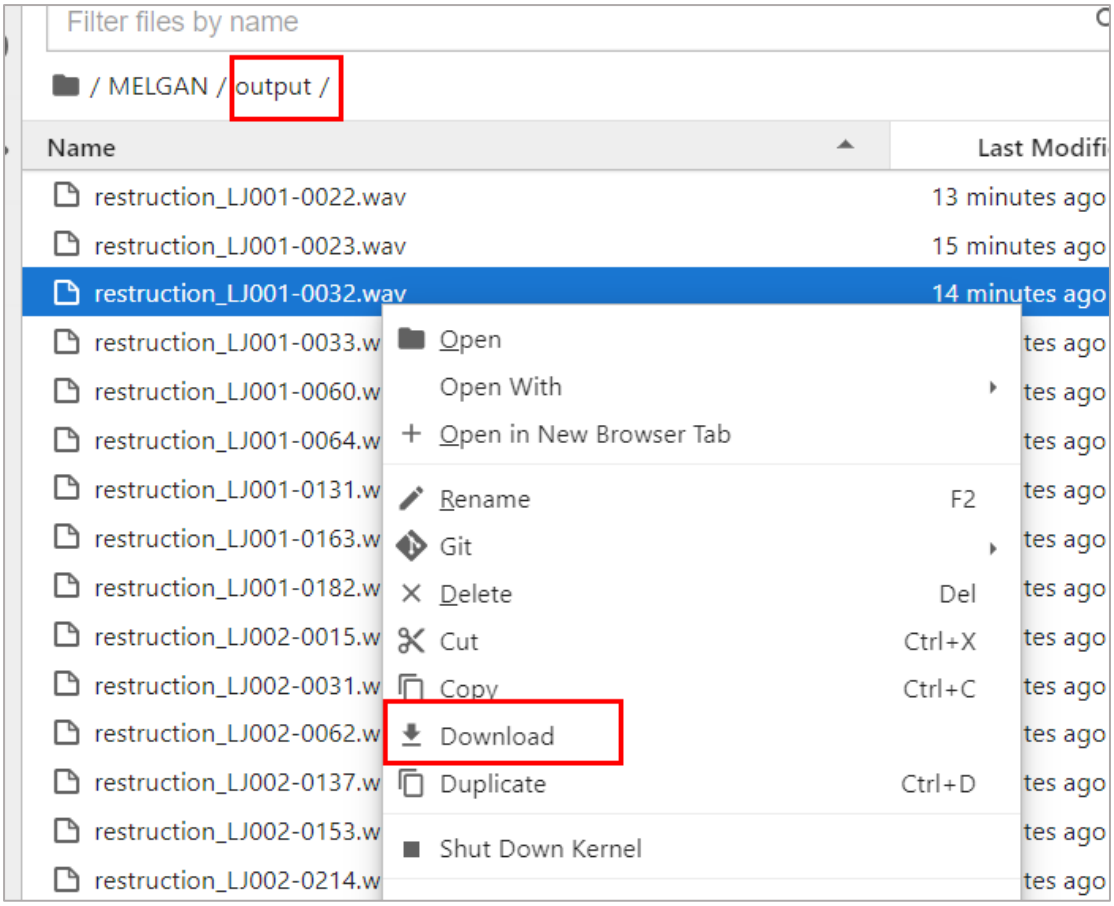
```
!python eval.py
```

推理日志：

每个语音文件是按 210 帧切分，生成一段或多段 output；每个 output 打印 210 帧的推理结果，该值衡量模型是否收敛，若出现所有结果均为 0 则代表模型无法收敛，需调整训练参数重新训练。

```
output210=[-0.02639623 -0.03571026 -0.03692935 ...  0.10518043  0.05373079
  0.0176632 ]
output420=[0.00062764 0.01323923 0.0118585  ... 0.00106345 0.00083652 0.00103483]
output630=[-0.001335   -0.00159535 -0.00198062 ...  0.00025333 -0.00050769
 -0.00073751]
output840=[-0.02435283 -0.03622842 -0.03036028 ...  0.00025333 -0.00050769
 -0.00073751]
LJ001-0014.npy done!
output210=[-0.10944302 -0.16244727 -0.17087346 ...  0.00095759  0.00055852
  0.00058189]
output420=[-0.01928935 -0.01543766 -0.01637586 ... -0.0890776  -0.05048352
 -0.03388143]
output630=[-4.6165935e-03  8.9520048e-03  4.1304426e-03 ...  4.5082255e-05
 -1.3378858e-04 -2.2456860e-04]
LJ001-0021.npy done!
output210=[ 0.01204159  0.0156038   0.00863925 ... -0.03257061  0.01874622
 -0.00465092]
output420=[ 1.8241144e-05 -2.8908544e-04 -6.5803928e-05 ... -2.8446650e-03
 -1.0446389e-03 -2.0817176e-03]
LJ001-0022.npy done!
```

结果保存到 output 目录，可下载试听输出 wav 结果。


Filter files by name
/ MELGAN / output /
Name
Last Modifi
restruction_LJ001-0022.wav
13 minutes ago
restruction_LJ001-0023.wav
15 minutes ago
restruction_LJ001-0032.wav
14 minutes ago
restruction_LJ001-0033.w
restruction_LJ001-0060.w
restruction_LJ001-0064.w
restruction_LJ001-0131.w
restruction_LJ001-0163.w
restruction_LJ001-0182.w
restruction_LJ002-0015.w
restruction_LJ002-0031.w
restruction_LJ002-0062.w
restruction_LJ002-0137.w
restruction_LJ002-0153.w
restruction_LJ002-0214.w
tes ago
Open
Open With
Open in New Browser Tab
Rename
F2
Git
Delete
Del
Cut
Ctrl+X
Copy
Ctrl+C
Download
Duplicate
Ctrl+D
Shut Down Kernel