华为“智能基座”系列课程

《深度学习》

CycleGAN生成实验

版本：1.1（武汉理工大学更新开发）



华为技术有限公司

|  |
| --- |
| **版权所有 © 华为技术有限公司 2022。 保留一切权利。**  非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。  **商标声明**  C:\Users\jwx341670\Desktop\华为标志 Huawei Logo 2018\竖版标志Vertical Version\PNG\HW_POS_RBG_Vertical-150ppi.png 和其他华为商标均为华为技术有限公司的商标。  本文档提及的其他所有商标或注册商标，由各自的所有人拥有。  **注意**  您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。  由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。 |

|  |  |
| --- | --- |
| 华为技术有限公司 | |
| 地址： | 深圳市龙岗区坂田华为总部办公楼 邮编：518129 |
| 网址： | http://[e](http://e.huawei.com/).huawei.com |

目录

[1 实验介绍 5](#_Toc165142200)

[1.1 背景知识 5](#_Toc165142201)

[1.1.1 生成对抗网络GAN介绍 5](#_Toc165142207)

[2 基于Mindspore的CycleGAN生成网络实验 7](#_Toc165142208)

[2.1 实验介绍 7](#_Toc165142209)

[2.2 实验预备知识 7](#_Toc165142210)

[2.3 实验环境要求 8](#_Toc165142211)

[2.4 实验总体设计 8](#_Toc165142212)

[2.5 实验操作 8](#_Toc165142213)

[2.6 代码详解 9](#_Toc165142214)

[2.6.1 src/utils/args.py文件（模型参数设置） 9](#_Toc165142215)

[2.6.2 src/dataset/cyclegan\_dataset.py文件（数据加载） 11](#_Toc165142216)

[2.6.3 src/model/networks.py文件（定义基础卷积模块） 13](#_Toc165142217)

[2.6.4 src/model/resnet.py文件(resnet生成器搭建) 16](#_Toc165142218)

[2.6.5 src/model/cycle\_gan.py（判别器与模型搭建） 18](#_Toc165142224)

[2.6.6 src/model/losses.py（loss计算） 22](#_Toc165142225)

[2.6.7 train.py（训练模型） 24](#_Toc165142226)

[2.6.8 训练 26](#_Toc165142227)

[2.6.9 eval.py（推理模型） 26](#_Toc165142228)

[2.6.10 调整训练参数 29](#_Toc165142229)

[2.7 实验总结 29](#_Toc165142230)

# 实验介绍

## 背景知识



### 生成对抗网络GAN介绍

图像生成一直是图形学、计算机视觉研究领域具有挑战性的任务之一。近几年，生成对抗网络（Generative Adversarial Networks，GAN）在众多生成任务中展现了其强大的数据拟合能力。

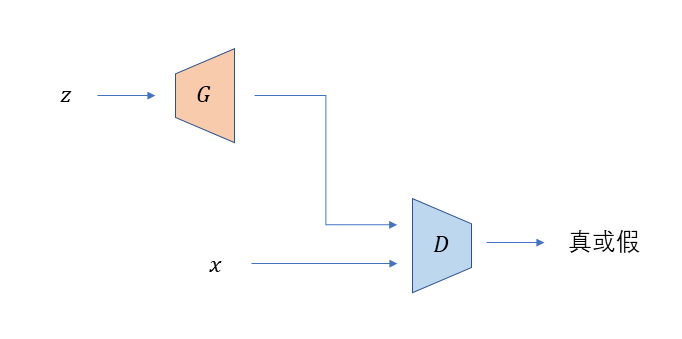


图1-2 生成对抗网络示意图

如图1-2所示，在生成对抗网络中，同时训练两个模型，一个生成模型和一个判别模型。生成模型用来捕获数据分布，判别模型用来估计样本来自训练数据而不是生成模型的概率。训练过程中，判别模型的目标是将“真”样本和生成模型生成的“假”样本区别开来。而生成模型的目标是生成判别模型无法区分真假的样本，使判别模型分类犯错的概率最大化。在和的不断对抗迭代训练过程中，模型最终达到一个动态平衡，捕获了真实数据分布，对真假样本的分类均等于0.5。生成对抗网络的损失函数如下：

|  |  |
| --- | --- |
|  | （1-1） |

其中，表示真实的训练数据，表示原始的噪声数据，表示噪声经过生成模型获得的假样本。训练过程中，最小化上述损失函数训练，即期望尽可能等于0，尽可能等于1。与之相反，最大化上述损失函数训练，即期望尽可能等于0。在对抗训练框架下，生成模型和判别模型相当于进行一个极大极小的双人博弈。

图1-3给出了训练过程中，判别模型和生成模型数据分布的变化情况，其中蓝色虚线表示判别模型的数据分布，黑色虚线表示真实的数据分布，绿色实线表示生成模型捕获的数据分布。图1-3底部的水平线表示噪声的采样空间，向上的箭头表示从噪声到真实数据的映射。图（a）表示判别模型和生成模型的初始状态，此刻生成模型捕获的假数据分布与真实样本分布存在明显差异，判别模型对真假样本的分类也不准确；图（b）表示训练判别模型之后的状态，判别模型可以显著地区分假样本和真样本；图（c）表示训练生成模型之后的状态，生成模型尽可能拟合真实样本的数据分布，导致判别模型的分类准确率下降；图（d）表示多次训练后，达到了稳定状态，捕获了真实数据分布，对真假样本的分类均等于0.5。

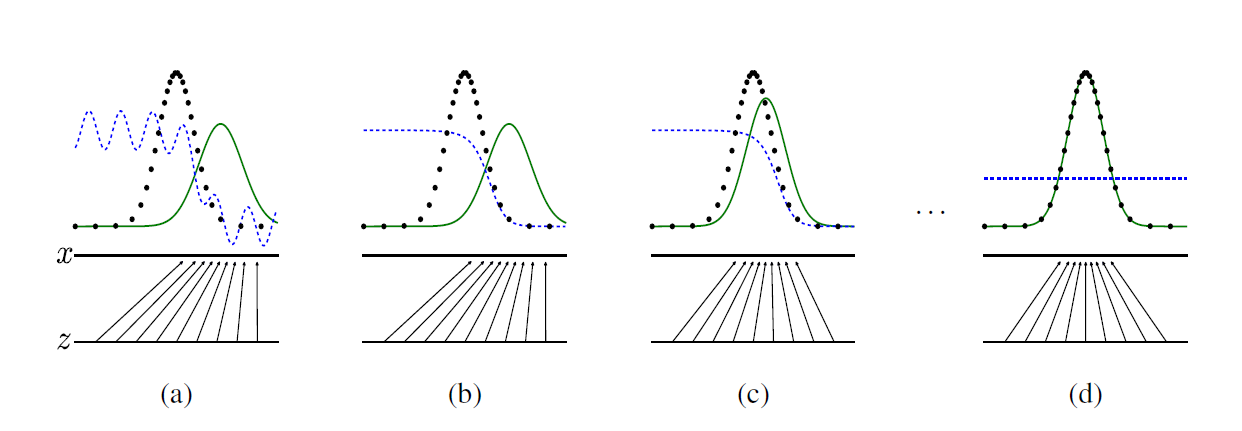


图1-3 训练过程判别模型和生成模型数据分布图

# 基于Mindspore的CycleGAN生成网络实验

## 实验介绍

生成对抗网络（GAN）是生成模型的一种神经网络架构，而本实验在此基础上，将搭建经典的CycleGAN网络架构，完成不同域图像之间的生成转换。训练数据包括两个图像数据集合A（苹果图像集）和B（橘子图像集），本实验的任务是训练CycleGAN网络，将苹果图像转化生成橘子，同时也可以将橘子转化生成苹果，可以看做一个基于MindSpore的生成任务。

|  |  |
| --- | --- |
|  |  |
| 实验总体设计示意图 | |

实验数据：

从下面地址下载训练测试代码

代码链接：

<http://59.69.101.2/CycleGAN.zip>

## 实验预备知识

有图像预处理的基础。

有相应Python语言的编程基础。

有生成对抗网络GAN的理论基础。

## 实验环境要求

ModelArts平台：Mindspore-python3.7-aarch64

## 实验总体设计

实验总体设计示意图

实验模型：

这里我们将学习一种经典且应用广泛的生成对抗网络（GAN），与其他生成模型不同，生成对抗网络被表述为一个极小极大游戏，判别器努力区分生成图像和真实图像，而生成器尽力生成逼真的图像来欺骗判别器，在这样对抗的训练中，最终生成尽可能逼真的图像。

本实验主要介绍使用MindSpore在Ascend环境下训练并测试CycleGAN模型。

本实验将搭建两个生成器(,)和两个判别器(,)，其中生成器的任务是将属于X域的图像转化生成属于B域的图像，生成器的任务是将属于Y域的图像转化生成属于X域的图像，判别器用来鉴别图像是否属于X域，判别器用来判别图像是否属于Y域。同时为了保持图像在转化中保持内容的一致性，CycleGAN提出了循环一直损失，约束属于X域的图像经过转换到Y域后，再经过转换到X域后，应用原始图像保持一致。数学公式描述为。模型训练完成后，结果如图2-1所示。

## 实验操作

进入ModelArts开发环境

参考文末附录，创建ModelArts上的开发环境Notebook并进入。

在Notebook中下载数据集和代码

由于原始文件比较大，因此需要在Notebook环境中下载数据集apple2orange.zip（以苹果橘子数据集为例）

# 下载代码与数据集

%env no\_proxy='a.test.com,127.0.0.1,2.2.2.2'

!wget http://59.69.101.2/CycleGAN.zip

解压数据和代码文件

!unzip CycleGAN.zip

## 代码详解

本节将详细介绍实验的设计与实现。

本实验主要介绍使用MindSpore在Ascend环境下训练并测试CycleGAN模型。

该实验主要步骤包括：

模型参数设置

加载数据集，进行数据处理。

定义基础卷积模块。

resnet生成器搭建。

判别器与模型搭建

loss计算

训练模型

推理模型，得到预期结果

### src/utils/args.py文件（模型参数设置）

import argparse

import ast

from mindspore import context

from mindspore.context import ParallelMode

from mindspore.communication.management import init, get\_rank, get\_group\_size

parser = argparse.ArgumentParser(description='Cycle GAN')

# basic parameters

# 省略了部分参数设置

parser.add\_argument('--platform', type=str, default='Ascend', help='only support GPU and Ascend')#硬件场景

parser.add\_argument('--device\_id', type=int, default=0, help='device id, default is 0.')

parser.add\_argument('--image\_size', type=int, default=256, help='input image\_size, default is 256.')# 图像大小

parser.add\_argument('--batch\_size', type=int, default=1, help='batch\_size, default is 1.') # batchsize大小

parser.add\_argument('--pool\_size', type=int, default=50, \

help='the size of image buffer that stores previously generated images') # 图像池设置

parser.add\_argument('--beta1', type=float, default=0.5, help='Adam beta1, default is 0.5.')

parser.add\_argument('--lr', type=float, default=0.0002, help='learning rate, default is 0.0002.') # 学习率设置

parser.add\_argument('--lr\_policy', type=str, default='linear', choices=('linear', 'constant'), \

help='learning rate policy, default is linear') #学习率策略选择

parser.add\_argument('--max\_epoch', type=int, default=200, help='epoch size for training, default is 200.')#

parser.add\_argument('--n\_epochs', type=int, default=100, \

help='number of epochs with the initial learning rate, default is 100')

# model parameters

parser.add\_argument('--in\_planes', type=int, default=3, help='input channels, default is 3.') #输入通道数

parser.add\_argument('--ngf', type=int, default=64)#生成器通道过滤器个数，显存不够可调小为16、32

parser.add\_argument('--gl\_num', type=int, default=9)#生成器残差块个数，显存不够可任意调小

parser.add\_argument('--ndf', type=int, default=64) #判别器通道过滤器个数，显存不够可调小为16、32

parser.add\_argument('--dl\_num', type=int, default=3) #判别器残差块个数，显存不够可任意调小

parser.add\_argument('--norm\_mode', type=str, default='batch', choices=('batch', 'instance'), \

help='norm mode, default is batch.')# 归一层选择

parser.add\_argument('--lambda\_A', type=float, default=10.0, \

help='weight for cycle loss (A -> B -> A), default is 10.')# loss权重设置

parser.add\_argument('--lambda\_B', type=float, default=10.0, \

help='weight for cycle loss (B -> A -> B), default is 10.') # loss权重设置

parser.add\_argument('--lambda\_idt', type=float, default=0.5) # loss权重设置

parser.add\_argument('--gan\_mode', type=str, default='lsgan', choices=('lsgan', 'vanilla'), \

help='the type of GAN loss, default is lsgan.') #对抗loss选择

# additional parameters

parser.add\_argument('--dataroot', default='./data/horse2zebra/', \

help='path of images (should have subfolders trainA, trainB, testA, testB).')#数据集路径

parser.add\_argument('--data\_dir', default='testA', choices=('testA', 'testB'), \

help='the translation direction of CycleGAN.')

parser.add\_argument('--outputs\_dir', type=str, default='./outputs', \

help='models are saved here, default is ./outputs.')

parser.add\_argument('--load\_ckpt', type=ast.literal\_eval, default=False, \

help='whether load pretrained ckpt')

parser.add\_argument('--save\_checkpoint\_epochs', type=int, default=10, \

help='Save checkpoint epochs, default is 10.')

parser.add\_argument('--print\_iter', type=int, default=100, help='log print iter, default is 100.')

args = parser.parse\_args()})

### src/dataset/cyclegan\_dataset.py文件（数据加载）

# 引入所需模块

import os

import random

import multiprocessing

import numpy as np

from PIL import Image

import mindspore.dataset as de

import mindspore.dataset.vision.c\_transforms as C

# 获取目录里所有图像的路径

def make\_dataset(dir\_path, max\_dataset\_size=float("inf")):

"""Return image list in dir."""

images = []

assert os.path.isdir(dir\_path), '%s is not a valid directory' % dir\_path

for root, \_, fnames in sorted(os.walk(dir\_path)):

for fname in fnames:

if is\_image\_file(fname):

path = os.path.join(root, fname)

images.append(path)

return images[:min(max\_dataset\_size, len(images))]

# 自定义数据集

class UnalignedDataset:

"""

This dataset class can load unaligned/unpaired datasets.

It requires two directories to host training images from domain A '/path/to/data/trainA'

and from domain B '/path/to/data/trainB' respectively.

You can train the model with the dataset flag '--dataroot /path/to/data'.

Similarly, you need to prepare two directories:

'/path/to/data/testA' and '/path/to/data/testB' during test time.

Returns:

Two domain image path list.

"""

def \_\_init\_\_(self, dataroot, phase, max\_dataset\_size=float("inf"), use\_random=True):

self.dir\_A = os.path.join(dataroot, phase + 'A') # 获取数据集A的根目录

self.dir\_B = os.path.join(dataroot, phase + 'B') # 获取数据集B的根目录

self.A\_paths = sorted(make\_dataset(self.dir\_A, max\_dataset\_size)) # load images from '/path/to/data/trainA'

self.B\_paths = sorted(make\_dataset(self.dir\_B, max\_dataset\_size)) # load images from '/path/to/data/trainB'

self.A\_size = len(self.A\_paths) # get the size of dataset A

self.B\_size = len(self.B\_paths) # get the size of dataset B

self.use\_random = use\_random

def \_\_getitem\_\_(self, index):# 获取数据

"""Return a data point and its metadata information.

Parameters:

index (int) -- a random integer for data indexing

Returns a dictionary that contains A, B, A\_paths and B\_paths

A (tensor) -- an image in the input domain

B (tensor) -- its corresponding image in the target domain

A\_paths (str) -- image paths

B\_paths (str) -- image paths

"""

index\_B = index % self.B\_size

if index % max(self.A\_size, self.B\_size) == 0 and self.use\_random:#随机获取B路径中的数据

random.shuffle(self.A\_paths)

index\_B = random.randint(0, self.B\_size - 1)

A\_path = self.A\_paths[index % self.A\_size]

B\_path = self.B\_paths[index\_B]

A\_img = np.array(Image.open(A\_path).convert('RGB'))

B\_img = np.array(Image.open(B\_path).convert('RGB'))

return A\_img, B\_img

def \_\_len\_\_(self):

"""Return the total number of images in the dataset.

"""

return max(self.A\_size, self.B\_size)

# 创建用于供模型读取的数据集

def create\_dataset(args):

"""

Create dataset

This dataset class can load images for train or test.

Args:

dataroot (str): Images root directory.

Returns:

RGB Image list.

"""

dataroot = args.dataroot

phase = args.phase

batch\_size = args.batch\_size

device\_num = args.device\_num

rank = args.rank

shuffle = args.use\_random

max\_dataset\_size = args.max\_dataset\_size

cores = multiprocessing.cpu\_count()

num\_parallel\_workers = min(8, int(cores / device\_num))

image\_size = args.image\_size

mean = [0.5 \* 255] \* 3

std = [0.5 \* 255] \* 3

if phase == "train":#训练时的数据读取

dataset = UnalignedDataset(dataroot, phase, max\_dataset\_size=max\_dataset\_size, use\_random=args.use\_random)

distributed\_sampler = DistributedSampler(len(dataset), device\_num, rank, shuffle=shuffle)

ds = de.GeneratorDataset(dataset, column\_names=["image\_A", "image\_B"],

sampler=distributed\_sampler, num\_parallel\_workers=num\_parallel\_workers)#生成MSP模型可用的数据集

if args.use\_random:

trans = [

C.RandomResizedCrop(image\_size, scale=(0.5, 1.0), ratio=(0.75, 1.333)),

C.RandomHorizontalFlip(prob=0.5),

C.Normalize(mean=mean, std=std),

C.HWC2CHW()

]#数据集的增强操作

else:

trans = [

C.Resize((image\_size, image\_size)),

C.Normalize(mean=mean, std=std),

C.HWC2CHW()

]

ds = ds.map(operations=trans, input\_columns=["image\_A"], num\_parallel\_workers=num\_parallel\_workers)

ds = ds.map(operations=trans, input\_columns=["image\_B"], num\_parallel\_workers=num\_parallel\_workers)

ds = ds.batch(batch\_size, drop\_remainder=True)

else:#测试时的数据读取

datadir = os.path.join(dataroot, args.data\_dir)

dataset = ImageFolderDataset(datadir, max\_dataset\_size=max\_dataset\_size)

ds = de.GeneratorDataset(dataset, column\_names=["image", "image\_name"],

num\_parallel\_workers=num\_parallel\_workers)

trans = [

C.Resize((image\_size, image\_size)),

C.Normalize(mean=mean, std=std),

C.HWC2CHW()

]

ds = ds.map(operations=trans, input\_columns=["image"], num\_parallel\_workers=num\_parallel\_workers)

ds = ds.batch(1, drop\_remainder=True)

args.dataset\_size = len(dataset)

return ds

### src/model/networks.py文件（定义基础卷积模块）

import mindspore.nn as nn

from mindspore.common import initializer as init

# 基础卷积模块 [Conv + Norm + ReLU] 卷积层+归一化层+激活层，一般用来下采样

class ConvNormReLU(nn.Cell):

"""

Convolution fused with BatchNorm/InstanceNorm and ReLU/LackyReLU block definition.

Args:

in\_planes (int): Input channel.

out\_planes (int): Output channel.

kernel\_size (int): Input kernel size. Default: 4.

stride (int): Stride size for the first convolutional layer. Default: 2.

alpha (float): Slope of LackyReLU. Default: 0.2.

norm\_mode (str): Specifies norm method. The optional values are "batch", "instance".

pad\_mode (str): Specifies padding mode. The optional values are "CONSTANT", "REFLECT", "SYMMETRIC".

Default: "CONSTANT".

use\_relu (bool): Use relu or not. Default: True.

padding (int): Pad size, if it is None, it will calculate by kernel\_size. Default: None.

Returns:

Tensor, output tensor.

"""

def \_\_init\_\_(self,

in\_planes,

out\_planes,

kernel\_size=4,

stride=2,

alpha=0.2,

norm\_mode='batch',

pad\_mode='CONSTANT',

use\_relu=True,

padding=None):

super(ConvNormReLU, self).\_\_init\_\_()

norm = nn.BatchNorm2d(out\_planes)#归一化层

if norm\_mode == 'instance':

# Use BatchNorm2d with batchsize=1, affine=False, training=True instead of InstanceNorm2d

norm = nn.BatchNorm2d(out\_planes, affine=False)#当batchsize=1时BatchNorm退化为InstanceNorm

has\_bias = (norm\_mode == 'instance')

if padding is None:

padding = (kernel\_size - 1) // 2

if pad\_mode == 'CONSTANT':

conv = nn.Conv2d(in\_planes, out\_planes, kernel\_size, stride, pad\_mode='pad',

has\_bias=has\_bias, padding=padding)#卷积层

layers = [conv, norm]# 卷积层+归一化层

else:

paddings = ((0, 0), (0, 0), (padding, padding), (padding, padding))

pad = nn.Pad(paddings=paddings, mode=pad\_mode)

conv = nn.Conv2d(in\_planes, out\_planes, kernel\_size, stride, pad\_mode='pad', has\_bias=has\_bias)

layers = [pad, conv, norm] ]#先pad， 卷积层+归一化层

if use\_relu:

relu = nn.ReLU()# 激活层

if alpha > 0:

relu = nn.LeakyReLU(alpha) #选择LeakyReLU激活层

layers.append(relu)# 加上激活层

self.features = nn.SequentialCell(layers)

def construct(self, x):

output = self.features(x)

return output

# 基础反卷积模块 [ConvT + Norm + ReLU] 反卷积层+归一化层+激活层 ，一般用来上采样

class ConvTransposeNormReLU(nn.Cell):

"""

ConvTranspose2d fused with BatchNorm/InstanceNorm and ReLU/LackyReLU block definition.

Args:

in\_planes (int): Input channel.

out\_planes (int): Output channel.

kernel\_size (int): Input kernel size. Default: 4.

stride (int): Stride size for the first convolutional layer. Default: 2.

alpha (float): Slope of LackyReLU. Default: 0.2.

norm\_mode (str): Specifies norm method. The optional values are "batch", "instance".

pad\_mode (str): Specifies padding mode. The optional values are "CONSTANT", "REFLECT", "SYMMETRIC".

Default: "CONSTANT".

use\_relu (bool): use relu or not. Default: True.

padding (int): pad size, if it is None, it will calculate by kernel\_size. Default: None.

Returns:

Tensor, output tensor.

"""

def \_\_init\_\_(self,

in\_planes,

out\_planes,

kernel\_size=4,

stride=2,

alpha=0.2,

norm\_mode='batch',

pad\_mode='CONSTANT',

use\_relu=True,

padding=None):

super(ConvTransposeNormReLU, self).\_\_init\_\_()

conv = nn.Conv2dTranspose(in\_planes, out\_planes, kernel\_size, stride=stride, pad\_mode='same')

norm = nn.BatchNorm2d(out\_planes)

if norm\_mode == 'instance':

# Use BatchNorm2d with batchsize=1, affine=False, training=True instead of InstanceNorm2d

norm = nn.BatchNorm2d(out\_planes, affine=False)

has\_bias = (norm\_mode == 'instance')

if padding is None:

padding = (kernel\_size - 1) // 2

if pad\_mode == 'CONSTANT':

conv = nn.Conv2dTranspose(in\_planes, out\_planes, kernel\_size, stride, pad\_mode='same', has\_bias=has\_bias)

layers = [conv, norm]#反卷积层+归一化层

else:

paddings = ((0, 0), (0, 0), (padding, padding), (padding, padding))

pad = nn.Pad(paddings=paddings, mode=pad\_mode)

conv = nn.Conv2dTranspose(in\_planes, out\_planes, kernel\_size, stride, pad\_mode='pad', has\_bias=has\_bias)

layers = [pad, conv, norm]

if use\_relu:

relu = nn.ReLU()

if alpha > 0:

relu = nn.LeakyReLU(alpha)

layers.append(relu)

self.features = nn.SequentialCell(layers)

def construct(self, x):

output = self.features(x)

return output

### src/model/resnet.py文件(resnet生成器搭建)

"""ResNet Generator."""

import mindspore.nn as nn

import mindspore.ops as ops

from .networks import ConvNormReLU, ConvTransposeNormReLU

# 基础的残差块

class ResidualBlock(nn.Cell):

"""

A resnet block is a conv block with skip connections

We construct a conv block with build\_conv\_block function,

and implement skip connections in <forward> function..

Args:

dim (int): Input and output channel.

norm\_mode (str): Specifies norm method. The optional values are "batch", "instance".

dropout (bool): Use dropout or not. Default: False.

pad\_mode (str): Specifies padding mode. The optional values are "CONSTANT", "REFLECT", "SYMMETRIC".

Default: "CONSTANT".

Returns:

Tensor, output tensor.

"""

def \_\_init\_\_(self, dim, norm\_mode='batch', dropout=False, pad\_mode="CONSTANT"):

super(ResidualBlock, self).\_\_init\_\_()

self.conv1 = ConvNormReLU(dim, dim, 3, 1, 0, norm\_mode, pad\_mode)#卷积层+归一化层+激活层

self.conv2 = ConvNormReLU(dim, dim, 3, 1, 0, norm\_mode, pad\_mode, use\_relu=False)#卷积层+归一化层，激活层设置为False（残差块一般第二个卷积块不设置激活层）

self.dropout = dropout

if dropout:

self.dropout = nn.Dropout(0.5)

def construct(self, x):

out = self.conv1(x)

if self.dropout:

out = self.dropout(out)

out = self.conv2(out)

return x + out

class ResNetGenerator(nn.Cell):

"""

ResNet Generator of GAN.

Args:

in\_planes (int): Input channel.

ngf (int): Output channel.

n\_layers (int): The number of ConvNormReLU blocks.

alpha (float): LeakyRelu slope. Default: 0.2.

norm\_mode (str): Specifies norm method. The optional values are "batch", "instance".

dropout (bool): Use dropout or not. Default: False.

pad\_mode (str): Specifies padding mode. The optional values are "CONSTANT", "REFLECT", "SYMMETRIC".

Default: "CONSTANT".

Returns:

Tensor, output tensor.

"""

def \_\_init\_\_(self, in\_planes=3, ngf=64, n\_layers=9, alpha=0.2, norm\_mode='batch', dropout=False,

pad\_mode="CONSTANT"):

super(ResNetGenerator, self).\_\_init\_\_()

self.conv\_in = ConvNormReLU(in\_planes, ngf, 7, 1, alpha, norm\_mode, pad\_mode=pad\_mode)

self.down\_1 = ConvNormReLU(ngf, ngf \* 2, 3, 2, alpha, norm\_mode)#stride=2，下采样2倍

self.down\_2 = ConvNormReLU(ngf \* 2, ngf \* 4, 3, 2, alpha, norm\_mode) #stride=2，下采样2倍

layers = [ResidualBlock(ngf \* 4, norm\_mode, dropout=dropout, pad\_mode=pad\_mode)] \* n\_layers

self.residuals = nn.SequentialCell(layers)

self.up\_2 = ConvTransposeNormReLU(ngf \* 4, ngf \* 2, 3, 2, alpha, norm\_mode) #stride=2，上采样2倍

self.up\_1 = ConvTransposeNormReLU(ngf \* 2, ngf, 3, 2, alpha, norm\_mode) #stride=2，上采样2倍

if pad\_mode == "CONSTANT":

self.conv\_out = nn.Conv2d(ngf, 3, kernel\_size=7, stride=1, pad\_mode='pad', padding=3)

else:

pad = nn.Pad(paddings=((0, 0), (0, 0), (3, 3), (3, 3)), mode=pad\_mode)

conv = nn.Conv2d(ngf, 3, kernel\_size=7, stride=1, pad\_mode='pad')

self.conv\_out = nn.SequentialCell([pad, conv])

self.activate = ops.Tanh()

def construct(self, x):

x = self.conv\_in(x) #图像x，经过卷积块

x = self.down\_1(x)# 特征x，空间下采样2倍

x = self.down\_2(x)# 特征x，空间下采样2倍

x = self.residuals(x) # 特征x，经过残差块

x = self.up\_2(x) # 特征x，空间上采样2倍

x = self.up\_1(x) # 特征x，空间上采样2倍

output = self.conv\_out(x)# 最后一个卷积块，一般只有一个卷积层，没有归一化层

return self.activate(output) #激活函数选择为Tanh，目的是为了将数值归一化到[-1,1]



### src/model/cycle\_gan.py（判别器与模型搭建）

import mindspore as ms

import mindspore.nn as nn

from mindspore import context

from mindspore.context import ParallelMode

from mindspore.parallel.\_auto\_parallel\_context import auto\_parallel\_context

from mindspore.communication.management import get\_group\_size

import mindspore.ops as ops

from .networks import ConvNormReLU, init\_weights

from .resnet import ResNetGenerator

from .depth\_resnet import DepthResNetGenerator

from .unet import UnetGenerator

# 判别器搭建，就是卷积块的堆叠

class Discriminator(nn.Cell):

"""

Discriminator of GAN.

Args:

in\_planes (int): Input channel.

ndf (int): Output channel.

n\_layers (int): The number of ConvNormReLU blocks.

alpha (float): LeakyRelu slope. Default: 0.2.

norm\_mode (str): Specifies norm method. The optional values are "batch", "instance".

Returns:

Tensor, output tensor.

Examples:

>>> Discriminator(3, 64, 3)

"""

def \_\_init\_\_(self, in\_planes=3, ndf=64, n\_layers=3, alpha=0.2, norm\_mode='batch'):

super(Discriminator, self).\_\_init\_\_()

kernel\_size = 4

layers = [

nn.Conv2d(in\_planes, ndf, kernel\_size, 2, pad\_mode='pad', padding=1),

nn.LeakyReLU(alpha)

]

nf\_mult = ndf

for i in range(1, n\_layers):#堆叠卷积块

nf\_mult\_prev = nf\_mult

nf\_mult = min(2 \*\* i, 8) \* ndf

layers.append(ConvNormReLU(nf\_mult\_prev, nf\_mult, kernel\_size, 2, alpha, norm\_mode, padding=1))

nf\_mult\_prev = nf\_mult

nf\_mult = min(2 \*\* n\_layers, 8) \* ndf

layers.append(ConvNormReLU(nf\_mult\_prev, nf\_mult, kernel\_size, 1, alpha, norm\_mode, padding=1))

layers.append(nn.Conv2d(nf\_mult, 1, kernel\_size, 1, pad\_mode='pad', padding=1))

self.features = nn.SequentialCell(layers)

def construct(self, x):

output = self.features(x)

#生成过程

class Generator(nn.Cell):

"""

Generator of CycleGAN, return fake\_A, fake\_B, rec\_A, rec\_B, identity\_A and identity\_B.

Args:

G\_A (Cell): The generator network of domain A to domain B.

G\_B (Cell): The generator network of domain B to domain A.

use\_identity (bool): Use identity loss or not. Default: True.

Returns:

Tensors, fake\_A, fake\_B, rec\_A, rec\_B, identity\_A and identity\_B.

Examples:

>>> Generator(G\_A, G\_B)

"""

def \_\_init\_\_(self, G\_A, G\_B, use\_identity=True):

super(Generator, self).\_\_init\_\_()

self.G\_A = G\_A # The generator network of domain A to domain B

self.G\_B = G\_B # The generator network of domain B to domain A.

self.ones = ops.OnesLike()

self.use\_identity = use\_identity

def construct(self, img\_A, img\_B):

"""If use\_identity, identity loss will be used."""

fake\_A = self.G\_B(img\_B)# G\_B将img\_B（属于B域） 生成转化为fake\_A（属于A域）

fake\_B = self.G\_A(img\_A) # G\_A将img\_A（属于A域） 生成转化为fake\_B（属于B域）

rec\_A = self.G\_B(fake\_B) # G\_B将fake\_B再次转化到A域

rec\_B = self.G\_A(fake\_A) # G\_A将fake\_A再次转化到B域

if self.use\_identity:#是否使用一致性

identity\_A = self.G\_B(img\_A)# G\_B将imgA再次转化到A域，因为imgA本身就属于A域，所以应该保持不变，即identity\_A= img\_A

identity\_B = self.G\_A(img\_B)

else:

identity\_A = self.ones(img\_A)

identity\_B = self.ones(img\_B)

return fake\_A, fake\_B, rec\_A, rec\_B, identity\_A, identity\_B

#训练生成器G

class TrainOneStepG(nn.Cell):

"""

Encapsulation class of Cycle GAN generator network training.

Append an optimizer to the training network after that the construct

function can be called to create the backward graph.

Args:

G (Cell): Generator with loss Cell. Note that loss function should have been added.

generator (Cell): Generator of CycleGAN.

optimizer (Optimizer): Optimizer for updating the weights.

sens (Number): The adjust parameter. Default: 1.0.

"""

def \_\_init\_\_(self, G, generator, optimizer, sens=1.0):

super(TrainOneStepG, self).\_\_init\_\_(auto\_prefix=False)

self.optimizer = optimizer

self.G = G

self.G.set\_grad()

self.G.set\_train()

self.G.D\_A.set\_grad(False)#只更新训练生成器，不更新训练判别器

self.G.D\_A.set\_train(False)

self.G.D\_B.set\_grad(False)

self.G.D\_B.set\_train(False)

self.grad = ops.GradOperation(get\_by\_list=True, sens\_param=True)

self.sens = sens

self.weights = ms.ParameterTuple(generator.trainable\_params())

self.net = WithLossCell(G)

self.reducer\_flag = False

self.grad\_reducer = None

self.parallel\_mode = context.get\_auto\_parallel\_context("parallel\_mode")

if self.parallel\_mode in [ParallelMode.DATA\_PARALLEL, ParallelMode.HYBRID\_PARALLEL]:

self.reducer\_flag = True

if self.reducer\_flag:

mean = context.get\_auto\_parallel\_context("gradients\_mean")

if auto\_parallel\_context().get\_device\_num\_is\_set():

degree = context.get\_auto\_parallel\_context("device\_num")

else:

degree = get\_group\_size()

self.grad\_reducer = nn.DistributedGradReducer(optimizer.parameters, mean, degree)

def construct(self, img\_A, img\_B):

weights = self.weights

fake\_A, fake\_B, lg, lga, lgb, lca, lcb, lia, lib = self.G(img\_A, img\_B)#生成对应的图像，并返回相应的生成loss

sens = ops.Fill()(ops.DType()(lg), ops.Shape()(lg), self.sens)

grads\_g = self.grad(self.net, weights)(img\_A, img\_B, sens)# 根据loss，进行反向传播，更新参数

if self.reducer\_flag:

# apply grad reducer on grads

grads\_g = self.grad\_reducer(grads\_g)

return fake\_A, fake\_B, ops.depend(lg, self.optimizer(grads\_g)), lga, lgb, lca, lcb, lia, lib

class TrainOneStepD(nn.Cell):

"""

Encapsulation class of Cycle GAN discriminator network training.

Append an optimizer to the training network after that the construct

function can be called to create the backward graph.

Args:

G (Cell): Generator with loss Cell. Note that loss function should have been added.

optimizer (Optimizer): Optimizer for updating the weights.

sens (Number): The adjust parameter. Default: 1.0.

"""

def \_\_init\_\_(self, D, optimizer, sens=1.0):

super(TrainOneStepD, self).\_\_init\_\_(auto\_prefix=False)

self.optimizer = optimizer

self.D = D

self.D.set\_grad()

self.D.set\_train()

self.grad = ops.GradOperation(get\_by\_list=True, sens\_param=True)

self.sens = sens

self.weights = ms.ParameterTuple(D.trainable\_params())

self.reducer\_flag = False

self.grad\_reducer = None

self.parallel\_mode = context.get\_auto\_parallel\_context("parallel\_mode")

if self.parallel\_mode in [ParallelMode.DATA\_PARALLEL, ParallelMode.HYBRID\_PARALLEL]:

self.reducer\_flag = True

if self.reducer\_flag:

mean = context.get\_auto\_parallel\_context("gradients\_mean")

if auto\_parallel\_context().get\_device\_num\_is\_set():

degree = context.get\_auto\_parallel\_context("device\_num")

else:

degree = get\_group\_size()

self.grad\_reducer = nn.DistributedGradReducer(optimizer.parameters, mean, degree)

def construct(self, img\_A, img\_B, fake\_A, fake\_B):

weights = self.weights

ld = self.D(img\_A, img\_B, fake\_A, fake\_B)#返回判别器loss

sens\_d = ops.Fill()(ops.DType()(ld), ops.Shape()(ld), self.sens)

grads\_d = self.grad(self.D, weights)(img\_A, img\_B, fake\_A, fake\_B, sens\_d) # 根据loss，进行反向传播，更新参数

if self.reducer\_flag:

# apply grad reducer on grads

grads\_d = self.grad\_reducer(grads\_d)

return ops.depend(ld, self.optimizer(grads\_d))

### src/model/losses.py（loss计算）

class GANLoss(nn.Cell):

"""

The GANLoss class abstracts away the need to create the target label tensor

that has the same size as the input.

Args:

mode (str): The type of GAN objective. It currently supports 'vanilla', 'lsgan'. Default: 'lsgan'.

reduction (str): Specifies the reduction to be applied to the output.

Its value must be one of 'none', 'mean', 'sum'. Default: 'none'.

Parameters:

gan\_mode (str) - - the type of GAN objective. It currently supports vanilla, lsgan, and wgangp.

target\_real\_label (bool) - - label for a real image

target\_fake\_label (bool) - - label of a fake image

Note: Do not use sigmoid as the last layer of Discriminator.

LSGAN needs no sigmoid. vanilla GANs will handle it with BCEWithLogitsLoss.

Outputs:

Tensor or Scalar, if `reduction` is 'none', then output is a tensor and has the same shape as `inputs`.

Otherwise, the output is a scalar.

"""

def \_\_init\_\_(self, mode="lsgan", reduction='mean'):

super(GANLoss, self).\_\_init\_\_()

self.loss = None

self.ones = ops.OnesLike()

if mode == "lsgan":

self.loss = nn.MSELoss(reduction)#lsgan采用mse损失

elif mode == "vanilla":

self.loss = BCEWithLogits(reduction)#原始gan采用交叉熵损失

else:

raise NotImplementedError(f'GANLoss {mode} not recognized, we support lsgan and vanilla.')

def construct(self, predict, target):

target = ops.cast(target, ops.dtype(predict))

target = self.ones(predict) \* target

loss = self.loss(predict, target)

return loss

class GeneratorLoss(nn.Cell):

"""

Cycle GAN generator loss.

Args:

args (class): Option class.

generator (Cell): Generator of CycleGAN.

D\_A (Cell): The discriminator network of domain A to domain B.

D\_B (Cell): The discriminator network of domain B to domain A.

Outputs:

Tuple Tensor, the losses of generator.

"""

def \_\_init\_\_(self, args, generator, D\_A, D\_B):

super(GeneratorLoss, self).\_\_init\_\_()

self.lambda\_A = args.lambda\_A

self.lambda\_B = args.lambda\_B

self.lambda\_idt = args.lambda\_idt

self.use\_identity = args.lambda\_idt > 0

self.dis\_loss = GANLoss(args.gan\_mode)

self.rec\_loss = nn.L1Loss("mean")

self.generator = generator

self.D\_A = D\_A

self.D\_B = D\_B

self.true = Tensor(True, mstype.bool\_)

def construct(self, img\_A, img\_B):

"""If use\_identity, identity loss will be used."""

# fake\_A是img\_B转换到A域的图像，rec\_B是img\_B->fake\_A->recB这就是cylegan名字来源

fake\_A, fake\_B, rec\_A, rec\_B, identity\_A, identity\_B = self.generator(img\_A, img\_B)

loss\_G\_A = self.dis\_loss(self.D\_B(fake\_B), self.true)# 生成器让判别器尽可能判别fake\_B为B域

loss\_G\_B = self.dis\_loss(self.D\_A(fake\_A), self.true)

loss\_C\_A = self.rec\_loss(rec\_A, img\_A) \* self.lambda\_A

loss\_C\_B = self.rec\_loss(rec\_B, img\_B) \* self.lambda\_B#img\_B->fake\_A->recB，生成一拳回来后，应保持原来内容

if self.use\_identity:

loss\_idt\_A = self.rec\_loss(identity\_A, img\_A) \* self.lambda\_A \* self.lambda\_idt

loss\_idt\_B = self.rec\_loss(identity\_B, img\_B) \* self.lambda\_B \* self.lambda\_idt

else:

loss\_idt\_A = 0

loss\_idt\_B = 0

loss\_G = loss\_G\_A + loss\_G\_B + loss\_C\_A + loss\_C\_B + loss\_idt\_A + loss\_idt\_B

return (fake\_A, fake\_B, loss\_G, loss\_G\_A, loss\_G\_B, loss\_C\_A, loss\_C\_B, loss\_idt\_A, loss\_idt\_B)

class DiscriminatorLoss(nn.Cell):

"""

Cycle GAN discriminator loss.

Args:

args (class): option class.

D\_A (Cell): The discriminator network of domain A to domain B.

D\_B (Cell): The discriminator network of domain B to domain A.

real (tensor array) -- real images

fake (tensor array) -- images generated by a generator

Outputs:

Tuple Tensor, the loss of discriminator.

"""

def \_\_init\_\_(self, args, D\_A, D\_B):

super(DiscriminatorLoss, self).\_\_init\_\_()

self.D\_A = D\_A

self.D\_B = D\_B

self.false = Tensor(False, mstype.bool\_)

self.true = Tensor(True, mstype.bool\_)

self.dis\_loss = GANLoss(args.gan\_mode)

self.rec\_loss = nn.L1Loss("mean")

def construct(self, img\_A, img\_B, fake\_A, fake\_B):

D\_fake\_A = self.D\_A(fake\_A)

D\_img\_A = self.D\_A(img\_A)

D\_fake\_B = self.D\_B(fake\_B)

D\_img\_B = self.D\_B(img\_B)

loss\_D\_A = self.dis\_loss(D\_fake\_A, self.false) + self.dis\_loss(D\_img\_A, self.true)#判别器努力将真实图像D\_img\_A判别为真，将生成图像D\_fake\_A判别为假，这与生成器目标形成了极大极小游戏，从而进行对抗训练

loss\_D\_B = self.dis\_loss(D\_fake\_B, self.false) + self.dis\_loss(D\_img\_B, self.true)

loss\_D = (loss\_D\_A + loss\_D\_B) \* 0.5

return loss\_D

### train.py（训练模型）

import mindspore as ms

import mindspore.nn as nn

from src.utils.args import get\_args

from src.utils.reporter import Reporter

from src.utils.tools import get\_lr, ImagePool, load\_ckpt

from src.dataset.cyclegan\_dataset import create\_dataset

from src.models.losses import DiscriminatorLoss, GeneratorLoss

from src.models.cycle\_gan import get\_generator, get\_discriminator, Generator, TrainOneStepG, TrainOneStepD

ms.set\_seed(1)

def train():

"""Train function."""

args = get\_args("train")

if args.need\_profiler:

from mindspore.profiler.profiling import Profiler

profiler = Profiler(output\_path=args.outputs\_dir, is\_detail=True, is\_show\_op\_path=True)

ds = create\_dataset(args)

G\_A = get\_generator(args)#获取生成器G\_A

G\_B = get\_generator(args) #获取生成器G\_B

D\_A = get\_discriminator(args) #获取判别器D\_A

D\_B = get\_discriminator(args) #获取判别器D\_A

if args.load\_ckpt:

load\_ckpt(args, G\_A, G\_B, D\_A, D\_B)

imgae\_pool\_A = ImagePool(args.pool\_size)#这是个图像池，用来存储生成的图像，更新D的时候从里边取

imgae\_pool\_B = ImagePool(args.pool\_size)

generator = Generator(G\_A, G\_B, args.lambda\_idt > 0)# 生成过程

loss\_D = DiscriminatorLoss(args, D\_A, D\_B)

loss\_G = GeneratorLoss(args, generator, D\_A, D\_B)

optimizer\_G = nn.Adam(generator.trainable\_params(), get\_lr(args), beta1=args.beta1)#设置优化器

optimizer\_D = nn.Adam(loss\_D.trainable\_params(), get\_lr(args), beta1=args.beta1)

net\_G = TrainOneStepG(loss\_G, generator, optimizer\_G)#还有参数更新过程的生成器

net\_D = TrainOneStepD(loss\_D, optimizer\_D) #还有参数更新过程的判别器

data\_loader = ds.create\_dict\_iterator()

if args.rank == 0:

reporter = Reporter(args)

reporter.info('==========start training===============')

for \_ in range(args.max\_epoch):

if args.rank == 0:

reporter.epoch\_start()

for data in data\_loader:

img\_A = data["image\_A"]# 加载图像

img\_B = data["image\_B"]# 加载图像

res\_G = net\_G(img\_A, img\_B) # 更新生成器并返回结果

fake\_A = res\_G[0]

fake\_B = res\_G[1]

res\_D = net\_D(img\_A, img\_B, imgae\_pool\_A.query(fake\_A), imgae\_pool\_B.query(fake\_B))#更新判别器

if args.rank == 0:

reporter.step\_end(res\_G, res\_D)

reporter.visualizer(img\_A, img\_B, fake\_A, fake\_B)

if args.rank == 0:

reporter.epoch\_end(net\_G)

if args.need\_profiler:

profiler.analyse()

break

if args.rank == 0:

reporter.info('==========end training===============')

if \_\_name\_\_ == "\_\_main\_\_":

train()

def convert\_to\_mindrecord(embed\_size, aclimdb\_path, preprocess\_path, glove\_path):

"""

将IMDB数据集转化为mindrecord数据格式

"""

parser = ImdbParser(aclimdb\_path, glove\_path, embed\_size)

parser.parse()

if not os.path.exists(preprocess\_path):

# 如果preprocess文件夹不存在，则新建文件夹

print(f"preprocess path {preprocess\_path} is not exist")

os.makedirs(preprocess\_path)

# 训练集

train\_features, train\_labels, train\_weight\_np = parser.get\_datas('train')

\_convert\_to\_mindrecord(preprocess\_path, train\_features, train\_labels, train\_weight\_np)

#

test\_features, test\_labels, \_ = parser.get\_datas('test')

\_convert\_to\_mindrecord(preprocess\_path, test\_features, test\_labels, training=False)

### 训练

使用终端进入CycleGAN目录，进行训练

cd CycleGAN

python train.py

### eval.py（推理模型）

import os

from mindspore import Tensor

from src.models.cycle\_gan import get\_generator

from src.utils.args import get\_args

from src.dataset.cyclegan\_dataset import create\_dataset

from src.utils.reporter import Reporter

from src.utils.tools import save\_image, load\_ckpt

def predict():

"""Predict function."""

args = get\_args("predict")

G\_A = get\_generator(args)#加载生成器，无需加载判别器

G\_B = get\_generator(args)

G\_A.set\_train(True)

G\_B.set\_train(True)

load\_ckpt(args, G\_A, G\_B)#加载模型参数

imgs\_out = os.path.join(args.outputs\_dir, "predict")

if not os.path.exists(imgs\_out):

os.makedirs(imgs\_out)

if not os.path.exists(os.path.join(imgs\_out, "fake\_A")):

os.makedirs(os.path.join(imgs\_out, "fake\_A"))

if not os.path.exists(os.path.join(imgs\_out, "fake\_B")):

os.makedirs(os.path.join(imgs\_out, "fake\_B"))

args.data\_dir = 'testA'

ds = create\_dataset(args)# 创建数据集

reporter = Reporter(args)

reporter.start\_predict("A to B")

for data in ds.create\_dict\_iterator(output\_numpy=True):

img\_A = Tensor(data["image"])

path\_A = str(data["image\_name"][0], encoding="utf-8")

path\_B = path\_A[0:-4] + "\_fake\_B.jpg"

fake\_B = G\_A(img\_A)#img\_A->fake\_B

save\_image(fake\_B, os.path.join(imgs\_out, "fake\_B", path\_B))#保存fake\_B

save\_image(img\_A, os.path.join(imgs\_out, "fake\_B", path\_A))

reporter.info('save fake\_B at %s', os.path.join(imgs\_out, "fake\_B", path\_A))

reporter.end\_predict()

args.data\_dir = 'testB'

ds = create\_dataset(args)

reporter.dataset\_size = args.dataset\_size

reporter.start\_predict("B to A")

for data in ds.create\_dict\_iterator(output\_numpy=True):

img\_B = Tensor(data["image"])

path\_B = str(data["image\_name"][0], encoding="utf-8")

path\_A = path\_B[0:-4] + "\_fake\_A.jpg"

fake\_A = G\_B(img\_B) #img\_B->fake\_A

save\_image(fake\_A, os.path.join(imgs\_out, "fake\_A", path\_A)) #保存fake\_A

save\_image(img\_B, os.path.join(imgs\_out, "fake\_A", path\_B))

reporter.info('save fake\_A at %s', os.path.join(imgs\_out, "fake\_A", path\_B))

reporter.end\_predict()

if \_\_name\_\_ == "\_\_main\_\_":

predict()

在CycleGAN目录，使用上述代码进行推理

python eval.py

预期结果



苹果与橘子相互转换预期结果图



马与斑马相互转换预期结果图

### 调整训练参数

调整训练参数，提升模型效果，例如增加训练迭代次数等，将迭代次数增大到200。

## 实验总结

本章提供了一个基于MindSpore框架的图像生成转换实验。通过本次体验全面了解了如何使用MindSpore进行图像生成以及域转化问题，理解了GAN的工作原理以及训练过程，具备了简单生成对抗模型GAN构建的基础编程能力。