华为昇腾-昇思MindSpore《计算机视觉》精品课程

HCIA-计算机视觉

精品课程

实验指导手册

版本:1.0



华为技术有限公司

|  |
| --- |
| 版权所有 © 华为技术有限公司 2021。 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 2](#_Toc112175156)

[1.1 实验案例简介 2](#_Toc112175157)

[1.2 实验目的 3](#_Toc112175158)

[1.3 实验总体设计 3](#_Toc112175159)

[2 实验过程 5](#_Toc112175160)

[2.1 实验环境配置 5](#_Toc112175161)

[2.1.1 Anaconda安装 5](#_Toc112175162)

[2.1.2 Mindspore安装 6](#_Toc112175163)

[2.1.3 Jupyter notebook配置 6](#_Toc112175164)

[2.2 数据加载及可视化 7](#_Toc112175165)

[2.3 模型建立 10](#_Toc112175166)

[2.4模型训练 14](#_Toc112175167)

[2.4.1 损失函数与优化器创建 14](#_Toc112175168)

[2.4.2 模型训练 16](#_Toc112175169)

[2.4.3 训练结果 17](#_Toc112175170)

[2.5实验总结 19](#_Toc112175171)

[2.6 实验任务与参考解答 20](#_Toc112175172)

[2.6.1生成器模型实验任务 20](#_Toc112175173)

[2.6.2 判别器模型实验任务 20](#_Toc112175174)

# 实验介绍

## 实验案例简介

GAN，又叫生成对抗网络，是一种非监督的学习。该网络中有一个生成器(Generators)和判别器(Discriminators)，而训练过程就是这两个网络不断博弈对抗的过程。生成器不断生成假图希望通过判别器的识别，而判别器则将图片划分为真实图像和生成图像，其结构如图1-1所示。

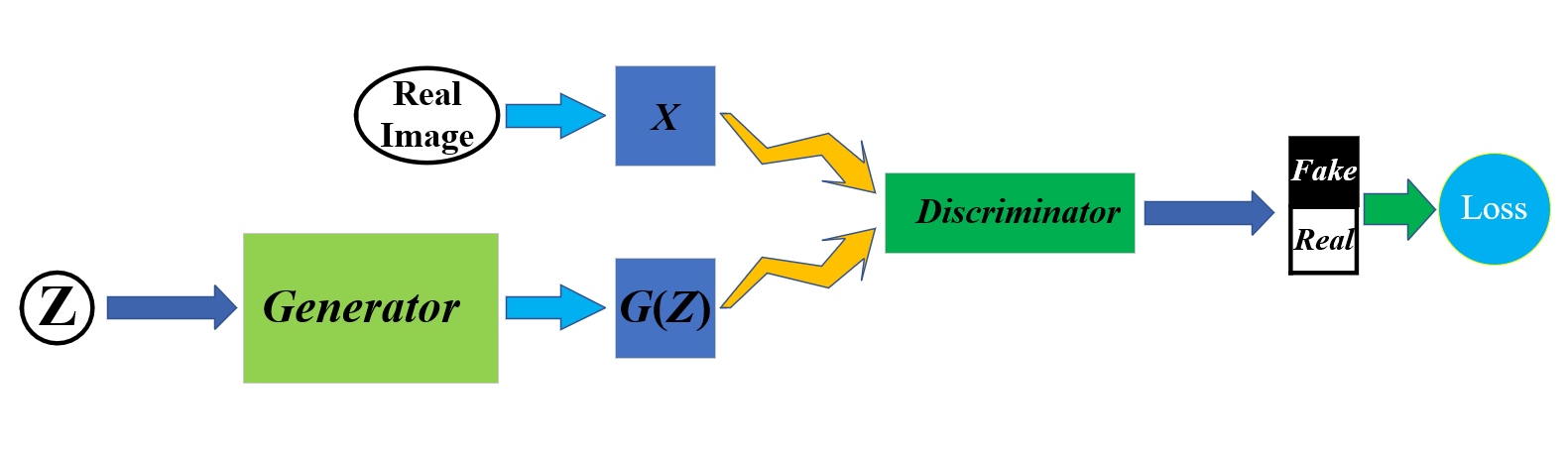


图1-1 GAN模型结构示意图

在该模型中，生成器如图1-2所示，其输入是一串噪音，输出是一张生成的假图，而生成器致力于让判别器无法识别出这张假图是生成的图还是真实图像。在训练过程中，不断用判别器的分数做反馈使生成器效果越来越好。

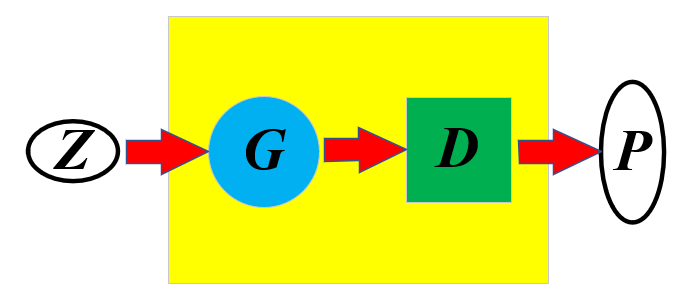


图1-2 生成器G示意图

判别器的输入是一张图片，如图1-3所示，输出则是图片的分数，分数越高说明此时生成的图像越接近真实图像。判别器致力于识别图片是真图还是假图，在训练过程中不断投喂假图，输出一个分数再与真实图像的标签进行比较。实际上也是一个二分类的过程。

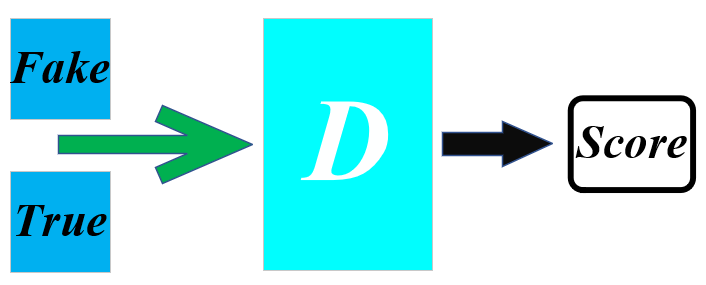


图1-3 判别器D示意图

* 1. 实验目的

本次实验的目的是通过GAN进行卡通头像生成，训练判别器和生成器，使二者收敛，达到平衡，从而使生成的卡通动漫头像更加接近真实图像。另外，本次实验希望读者在理解GAN算法原理的同时，也了解实际的图像生成操作，增强代码能力。通过学习，希望大家掌握如下知识：

了解MindSpore的基本语法；

了解生成器和判别器的模型建立方法；

了解网络模型的训练过程。

* 1. 实验总体设计

实验设计如图1-2所示。



图1-2 实验总体设计图

# 实验过程

## 实验环境配置

### 2.1.1 Anaconda安装

Anaconda 是一个用于科学计算的 python 发行版，支持 Linux, Mac, Windows 系统，提供了包管理与环境管理的功能。Anaconda是一个开源的包、环境管理器，其包含了conda、Python等180多个科学包及其依赖项，可以用于在同一个机器上安装不同版本的软件包及其依赖，并能够在不同的环境之间切换。

Anaconda有如下优点：

(1) 安装Anaconda后不再需要安装python，且python版本会在下载时进行选择。

(2) 开发时，有时需要不同的框架进行支撑，只需要对Anaconda添加虚拟环境，就可以在不同环境中完成开发，而不需要顾及兼容性等问题，可以为特殊项目配置相应的环境，方便管理。

Anaconda安装步骤如下：

步骤1 登录Anaconda官网(https://www.Anaconda.com)，如图2-1所示，点击Download即可下载；

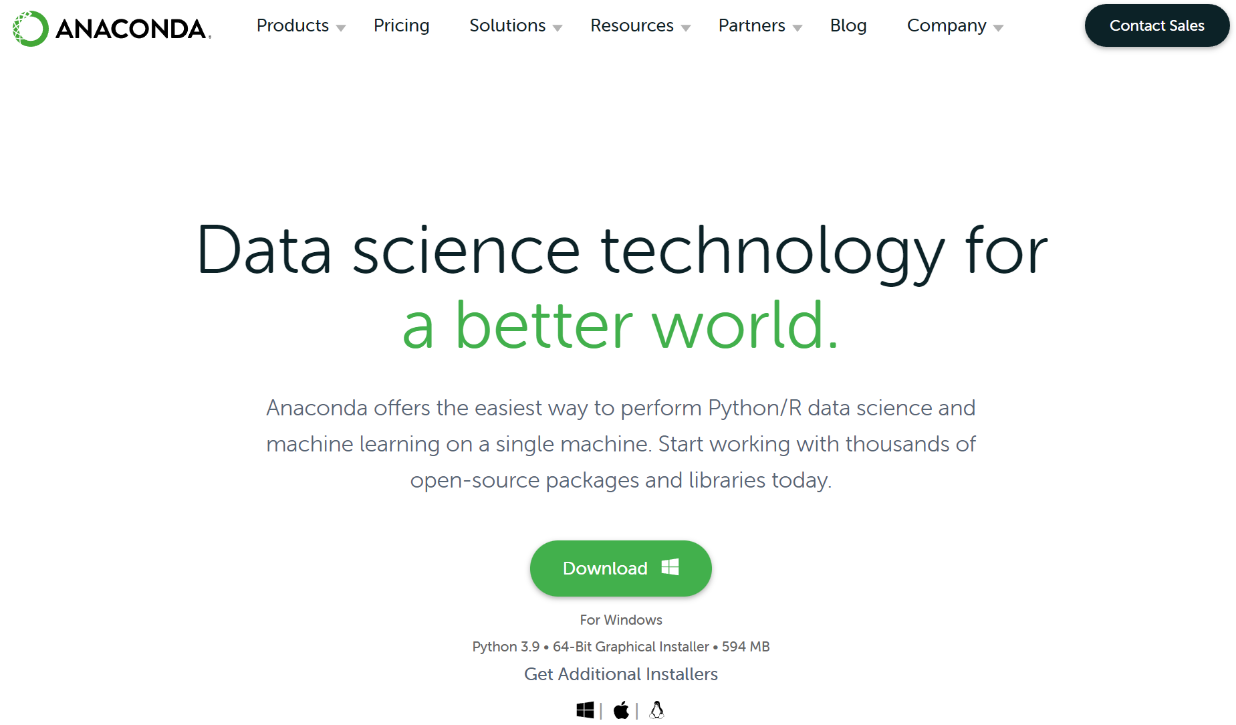


图2-1 Anaconda官网

步骤2 双击下载好的Anaconda3-x.x.x-Windows-x86\_64.exe文件，点击Next，随后点击I Agree进行安装；

步骤3 选择软件安装地址；

步骤4 配置环境变量。

### 2.1.2 Mindspore安装

MindSpore是华为公司推出的新一代深度学习框架，是源于全产业的最佳实践，最佳匹配昇腾处理器算力，支持终端、边缘、云全场景灵活部署，开创全新的AI编程范式，降低AI开发门槛。为了助力开发者与产业更加从容地应对这一系统级挑战，新一代AI框架MindSpore具有编程简单、端云协同、调试轻松、性能卓越、开源开放等特点，降低了AI开发门槛。

启动Anaconda Prompt，输入如下命令：

#创建python环境

conda create -n mindspore\_py37 -c conda-forge python=3.7.5

#激活环境

activate mindspore\_py37

#安装MindSpore

conda install mindspore-cpu -c mindspore -c conda-forge

#测试MindSpore是否安装成功

python -c "import mindspore;mindspore.run\_check()"

### 2.1.3 Jupyter notebook配置

Jupyter Notebook是基于网页的用于交互计算的应用程序。其可被应用于全过程计算：开发、文档编写、运行代码和展示结果。Jupyter Notebook是以网页的形式打开，可以在网页页面中直接编写代码和运行代码，代码的运行结果也会直接在代码块下显示。如在编程过程中需要编写说明文档，可在同一个页面中直接编写，便于作及时的说明和解释。

启动Anaconda Prompt，输入如下命令：

#激活创建的环境

conda activate ms1.7

#安装jupyter notebook

pip install jupyter

#生成ipykernel的配置文件

python -m ipykernel install --name ms1.7

#安装扩展包

pip install jupyter\_contrib\_nbextensions

#安装javascript和css文件

jupyter contrib nbextension install --user

#在指定地址下启动jupyter notebook

jupyter notebook --notebook-dir （相应地址）

输入命令启动jupyter notebook，界面如下：

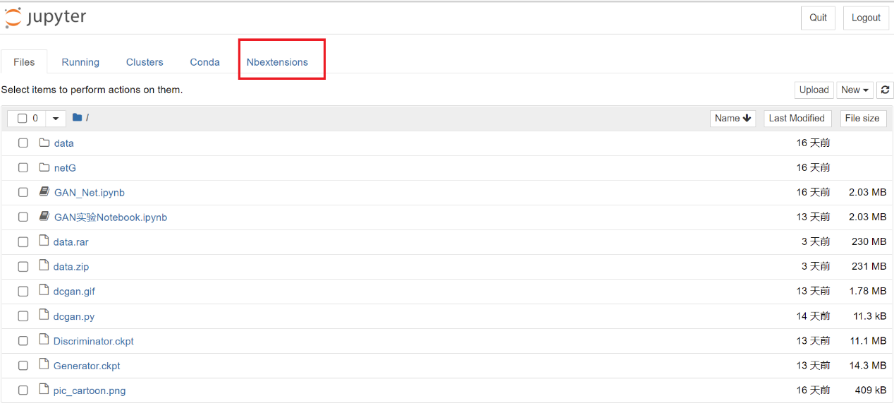


图2-2 jupyter notebook

## 数据加载及可视化

在本次实验中，将使用动漫头像数据集，该数据集包含21551张各式各样的卡通动漫头像，如图2-3所示。每张图像大小为64×64。为了使程序运行流畅，本次实验在当前路径下创建一个data目录，并在此目录下创建一个名为train的子目录。

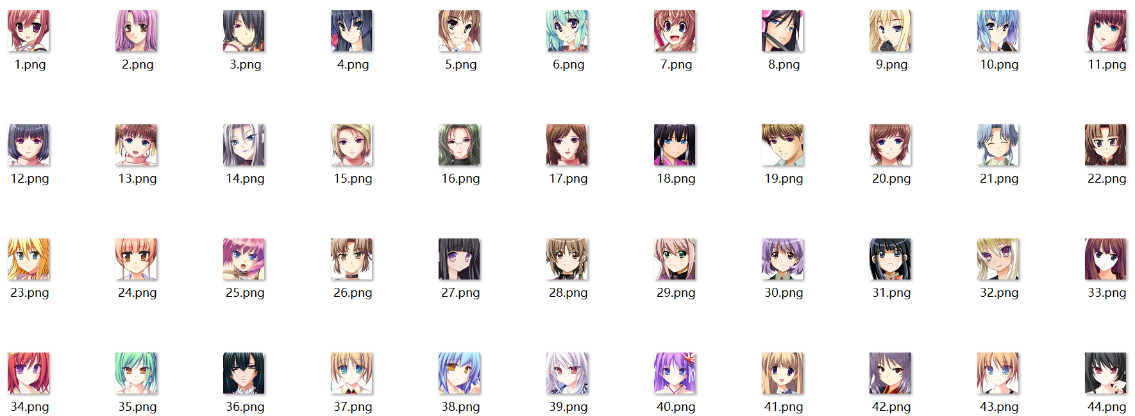


图2-3 数据集

使用GRAPH的模式运行实验，并对相关参数进行定义

from mindspore import set\_context, GRAPH\_MODE

# 选择执行模式为图模式；指定训练使用的平台为"GPU"，如需使用昇腾硬件可将其替换为"Ascend"

set\_context(mode=GRAPH\_MODE, device\_target="CPU")

data\_root = "./data" # 数据集根目录

workers = 4 # 载入数据线程数

batch\_size = 128 # 批量大小

image\_size = 64 # 训练图像空间大小，所有图像都将调整为该大小

nc = 3 # 图像彩色通道数，对于彩色图像为3

nz = 100 # 隐向量的长度

ngf = 64 # 特征图在生成器中的大小

ndf = 64 # 特征图在判别器中的大小

num\_epochs = 10 # 训练周期数

lr = 0.0002 # 学习率

beta1 = 0.5 # Adam优化器的beta1超参数

自定义函数create\_dataset\_imagenet对数据进行处理和增强。

import mindspore.dataset as ds

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

def create\_dataset\_imagenet(dataset\_path, num\_parallel\_workers=None):

"""数据加载"""

data\_set=ds.ImageFolderDataset(

dataset\_path,

num\_parallel\_workers=num\_parallel\_workers,

shuffle=True,decode=True

)

# 数据增强操作

transform\_img = [

vision.Resize(image\_size),

vision.CenterCrop(image\_size),

vision.HWC2CHW(),

lamada x:((x/255).astype("float32"),

np.random.normal(size=(nz,1,1)).astype("float32"))

]

# 数据映射操作

data\_set=data\_set.map(

input\_columns="image",

num\_parrallel\_workers=num\_parallel\_workers,

operations=transform\_img,

output\_columns=["image","latent\_code"],

column\_order=["image", "latent\_code"]

)

# 批量操作

data\_set = data\_set.batch(batch\_size)

return data\_set

# 获取处理后的数据集

data = create\_dataset\_imagenet(data\_root, num\_parallel\_workers=workers)

# 获取数据集大小

size = data.get\_dataset\_size()

print(size)

对数据集图像进行可视化操作

mport matplotlib.pyplot as plt

data\_iter = next(data.create\_dict\_iterator(output\_numpy=True))

# 可视化部分训练数据

plt.figure(figsize=(10, 3), dpi=140)

for i, image in enumerate(data\_iter['image'][:30], 1):

plt.subplot(3, 10, i)

plt.axis("off")

plt.imshow(image.transpose(1, 2, 0))

plt.show()

输出：



图2-4 卡通数据集可视化

## 模型建立

处理数据后，需搭建GAN的网络模型，本节分别从权重初始化出发，搭建生成器、判别器和损失函数模型。

生成器模型创建：

from mindspore import nn

from mindspore.common.initializer import Normal

def conv\_t(in\_channels,out\_channels,kernel\_size,stride=1,padding=0,

pad\_mode="pad"):

"""定义转置卷积层"""

weight\_init = Normal(mean=0, sigma=0.02)

return nn.Conv2dTranspose(

in\_channels,

out\_channels,

kernel\_size=kernel\_size,

stride=stride,

padding=padding,

weight\_init=weight\_init,

has\_bias=False,

pad\_mode=pad\_mode

)

def bn(num\_features):

"""定义BatchNorm2d层"""

gamma\_init = Normal(mean=1, sigma=0.02)

return nn.BatchNorm2d(

num\_features=num\_features,

gamma\_init=gamma\_init

)

class Generator(nn.Cell):

"""DCGAN网络生成器"""

def \_\_init\_\_(self):

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

self.generator = nn.SequentialCell()

self.generator.append(conv\_t(nz, ngf \* 8, 4, 1, 0))

self.generator.append(bn(ngf \* 8))

self.generator.append(nn.ReLU())

self.generator.append(conv\_t(ngf \* 8, ngf \* 4, 4, 2, 1))

self.generator.append(bn(ngf \* 4))

self.generator.append(nn.ReLU())

self.generator.append(conv\_t(ngf \* 4, ngf \* 2, 4, 2, 1))

self.generator.append(bn(ngf \* 2))

self.generator.append(nn.ReLU())

self.generator.append(conv\_t(ngf \* 2, ngf, 4, 2, 1))

self.generator.append(bn(ngf))

self.generator.append(nn.ReLU())

self.generator.append(conv\_t(ngf, nc, 4, 2, 1))

self.generator.append(nn.Tanh())

def construct(self, x):

return self.generator(x)

# 实例化生成器

netG = Generator()

print(netG)

输出：

Generator<

(generator): SequentialCell<

(0):Conv2dTranspose<input\_channels=100,output\_channels=512, kernel\_size=(4, 4), stride=(1, 1), pad\_mode=pad, padding=0, dilation=(1, 1), group=1,has\_bias=False,weight\_init=<mindspore.common.initializer.Normal object at 0x0000023FE916CE48>, bias\_init=zeros>

(1):BatchNorm2d<num\_features=512,eps=1e-05, momentum=0.09999999999999998,gamma=Parameter(name=1.gamma,shape=(512,),dtype=Float32,requires\_grad=True),beta=Parameter(name=1.beta,shape=(512,), dtype=Float32,requires\_grad=True),moving\_mean=Parameter(name=1.moving\_mean,shape=(512,),dtype=Float32, requires\_grad=False), moving\_variance=Parameter (name=1.moving\_variance, shape=(512,), dtype=Float32, requires\_grad=False)>

(2): ReLU<>

(3):Conv2dTranspose<input\_channels=512,output\_channels=256, kernel\_size=(4,4),stride=(2,2),pad\_mode=pad,padding=1,dilation=(1,1),group=1,has\_bias=False,weight\_init=<mindspore.common.initializer.

Normal object at 0x0000023FEAB8BC08>, bias\_init=zeros>

(4):BatchNorm2d<num\_features=256,eps=1e-05, momentum=0.09999999999999998,gamma=Parameter(name=4.gamma,shape=(256,),dtype=Float32,requires\_grad=True),beta=Parameter(name=4.beta,shape=(256,), dtype=Float32,requires\_grad=True),moving\_mean=Parameter(name=4.moving\_mean,shape=(256,),dtype=Float32, requires\_grad=False), moving\_variance=Parameter (name=4.moving\_variance, shape=(256,), dtype=Float32, requires\_grad=False)>

(5): ReLU<>

(6):Conv2dTranspose<input\_channels=256,output\_channels=128, kernel\_size=(4, 4),stride=(2, 2),pad\_mode=pad,padding=1, dilation=(1, 1), group=1, has\_bias=False,weight\_init=<mindspore.common.initializer.

Normal object at 0x0000023FEA172848>, bias\_init=zeros>

(7):BatchNorm2d<num\_features=128,eps=1e-05, momentum=0.09999999999999998,gamma=Parameter(name=7.gamma,shape=(128,),dtype=Float32,requires\_grad=True),beta=Parameter(name=7.beta,shape=(128,), dtype=Float32,requires\_grad=True),moving\_mean=Parameter(name=7.moving\_mean,shape=(128,),dtype=Float32, requires\_grad=False), moving\_variance=Parameter (name=7.moving\_variance, shape=(128,), dtype=Float32, requires\_grad=False)>

(8): ReLU<>

(9):Conv2dTranspose<input\_channels=128,output\_channels=64,

kernel\_size=(4,4),stride=(2,2), pad\_mode=pad, padding=1, dilation=(1, 1), group=1, has\_bias=False,weight\_init=<mindspore.common.initializer.

Normal object at 0x0000023FEA1990C8>, bias\_init=zeros>

(10):BatchNorm2d<num\_features=64,eps=1e-05, momentum=0.09999999999999998,gamma=Parameter(name=10.gamma,shape=(64,),dtype=Float32,requires\_grad=True),beta=Parameter(name=10.beta,shape=(64,), dtype=Float32,requires\_grad=True),moving\_mean=Parameter(name=10.moving\_mean,shape=(64,), dtype=Float32, requires\_grad=False), moving\_variance=Parameter (name=10.moving\_variance, shape=(64,), dtype=Float32, requires\_grad=False)>

(11): ReLU<>

(12):Conv2dTranspose<input\_channels=64,output\_channels=3, kernel\_size=(4,4),stride=(2,2), pad\_mode=pad, padding=1, dilation=(1, 1), group=1, has\_bias=False,weight\_init=<mindspore.common.initializer.

Normal object at 0x0000023FEA1C5088>, bias\_init=zeros>

(13): Tanh<>

>

>

判别器模型建立

from mindspore import nn

from mindspore.common.initializer import Normal

def conv(in\_channels,out\_channels,kernel\_size,stride=1, padding=0,

pad\_mode="pad"):

"""定义卷积层"""

weight\_init = Normal(mean=0, sigma=0.02)

return nn.Conv2d(

in\_channels,

out\_channels,

kernel\_size=kernel\_size,

stride=stride,

padding=padding,

weight\_init=weight\_init,

has\_bias=False,

pad\_mode=pad\_mode

)

class Discriminator(nn.Cell):

"""DCGAN网络判别器"""

def \_\_init\_\_(self):

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

self.discriminator = nn.SequentialCell()

self.discriminator.append(conv(nc, ndf, 4, 2, 1))

self.discriminator.append(nn.LeakyReLU(0.2))

self.discriminator.append(conv(ndf, ndf \* 2, 4, 2, 1))

self.discriminator.append(bn(ndf \* 2))

self.discriminator.append(nn.LeakyReLU(0.2))

self.discriminator.append(conv(ndf \* 2, ndf \* 4, 4, 2, 1))

self.discriminator.append(bn(ndf \* 4))

self.discriminator.append(nn.LeakyReLU(0.2))

self.discriminator.append(conv(ndf \* 4, ndf \* 8, 4, 2, 1))

self.discriminator.append(bn(ndf \* 8))

self.discriminator.append(nn.LeakyReLU(0.2))

self.discriminator.append(conv(ndf \* 8, 1, 4, 1))

self.discriminator.append(nn.Sigmoid())

def construct(self, x):

return self.discriminator(x)

# 实例化判别器

netD = Discriminator()

print(netD)

输出：

Discriminator<

(discriminator): SequentialCell<

(0):Conv2d<input\_channels=3,output\_channels=64,kernel\_size=(4,4),

stride=(2,2), pad\_mode=pad, padding=1, dilation=(1,1), group=1, has\_bias=False, weight\_init=<mindspore.common.initializer.

Normal object at 0x0000023FEAB7BA08>, bias\_init=zeros, format=NCHW>

(1): LeakyReLU<>

(2):Conv2d<input\_channels=64,output\_channels=128,kernel\_size=(4,4),

stride=(2,2), pad\_mode=pad, padding=1, dilation=(1,1), group=1, has\_bias=False, weight\_init=<mindspore.common.initializer.

Normal object at 0x0000023FEAB7BA88>, bias\_init=zeros, format=NCHW>

(3):BatchNorm2d<num\_features=128,eps=1e-05,

momentum=0.09999999999999998,gamma=Parameter(name=3.gamma,shape=(128,),dtype=Float32,requires\_grad=True),beta=Parameter(name=3.beta,shape=(128,), dtype=Float32,requires\_grad=True),moving\_mean=Parameter(name=3.moving\_meanshape=(128,), dtype=Float32, requires\_grad=False), moving\_variance=Parameter (name=3.moving\_variance, shape=(128,), dtype=Float32, requires\_grad=False)>

(4): LeakyReLU<>

(5):Conv2d<input\_channels=128,output\_channels=256,kernel\_size=(4,4), stride=(2,2), pad\_mode=pad, padding=1, dilation=(1,1), group=1, has\_bias=False, weight\_init=<mindspore.common.initializer.

Normal object at 0x0000023FE7451A88>, bias\_init=zeros, format=NCHW>

(6):BatchNorm2d<num\_features=256,eps=1e-05, momentum=0.09999999999999998,gamma=Parameter(name=6.gamma,shape=(256,),dtype=Float32,requires\_grad=True),beta=Parameter(name=6.beta,shape=(256,), dtype=Float32,requires\_grad=True),moving\_mean=Parameter(name=6.moving\_mean,shape=(256,),dtype=Float32, requires\_grad=False), moving\_variance=Parameter (name=6.moving\_variance, shape=(256,), dtype=Float32, requires\_grad=False)>

(7): LeakyReLU<>

(8):Conv2d<input\_channels=256,output\_channels=512,kernel\_size=(4,4),

stride=(2,2), pad\_mode=pad, padding=1, dilation=(1,1), group=1, has\_bias=False, weight\_init=<mindspore.common.initializer.

Normal object at 0x0000023FEA1AC788>, bias\_init=zeros, format=NCHW>

(9):BatchNorm2d<num\_features=512,eps=1e-05, momentum=0.09999999999999998,gamma=Parameter(name=9.gamma,shape=(512,),dtype=Float32,requires\_grad=True),beta=Parameter(name=9.beta,shape=(512,), dtype=Float32,requires\_grad=True),moving\_mean=Parameter(name=9.moving\_mean,shape=(512,),dtype=Float32, requires\_grad=False), moving\_variance=Parameter (name=9.moving\_variance, shape=(512,), dtype=Float32, requires\_grad=False)>

(10): LeakyReLU<>

(11):Conv2d<input\_channels=512,output\_channels=1,kernel\_size=(4,4), stride=(1,1), pad\_mode=pad, padding=0, dilation=(1,1), group=1, has\_bias=False, weight\_init=<mindspore.common.initializer.

Normal object at 0x0000023FEA1FFAC8>, bias\_init=zeros, format=NCHW>

(12): Sigmoid<>

>

>

## 2.4模型训练

### 2.4.1 损失函数与优化器创建

Mindspore将损失函数和优化器都封装到了Cell中，由于GAN结构比较特殊，导致它和一般的分类网络不同，因此我们需要自定义WithLossCell类，起到连接网络和Loss的作用。

from mindspore import Tensor

from mindspore import dtype as mstype

class WithLossCellG(nn.Cell):

"""连接生成器和损失"""

def \_\_init\_\_(self, netD, netG, loss\_fn):

super(WithLossCellG, self).\_\_init\_\_(auto\_prefix=True)

self.netD = netD

self.netG = netG

self.loss\_fn = loss\_fn

def construct(self, latent\_code):

"""构建生成器损失计算结构"""

fake\_data = self.netG(latent\_code)

out = self.netD(fake\_data)

label\_real = ops.OnesLike()(out)

loss = self.loss\_fn(out, label\_real)

return loss

class WithLossCellD(nn.Cell):

"""连接判别器和损失"""

def \_\_init\_\_(self, netD, netG, loss\_fn):

super(WithLossCellD, self).\_\_init\_\_(auto\_prefix=True)

self.netD = netD

self.netG = netG

self.loss\_fn = loss\_fn

def construct(self, real\_data, latent\_code):

"""构建判别器损失计算结构"""

out\_real = self.netD(real\_data)

label\_real = ops.OnesLike()(out\_real)

loss\_real = self.loss\_fn(out\_real, label\_real)

fake\_data = self.netG(latent\_code)

fake\_data = ops.stop\_gradient(fake\_data)

out\_fake = self.netD(fake\_data)

label\_fake = ops.ZerosLike()(out\_fake)

loss\_fake = self.loss\_fn(out\_fake, label\_fake)

return loss\_real + loss\_fake

# 定义损失函数

criterion = nn.BCELoss(reduction='mean')

# 创建一批隐向量用来观察G

np.random.seed(1)

fixed\_noise = Tensor(np.random.randn(64, nz, 1, 1), dtype=mstype.float32)

# 为生成器和判别器设置优化器

optimizerD = nn.Adam(netD.trainable\_params(), learning\_rate=lr, beta1=beta1)

optimizerG = nn.Adam(netG.trainable\_params(), learning\_rate=lr, beta1=beta1)

### 2.4.2 模型训练

进行网络定义：

from mindspore import ops

from mindspore import save\_checkpoint

class DCGAN(nn.Cell):

"""定义DCGAN网络"""

def \_\_init\_\_(self, myTrainOneStepCellForD, myTrainOneStepCellForG):

super(DCGAN, self).\_\_init\_\_(auto\_prefix=True)

self.myTrainOneStepCellForD = myTrainOneStepCellForD

self.myTrainOneStepCellForG = myTrainOneStepCellForG

def construct(self, real\_data, latent\_code):

output\_D = self.myTrainOneStepCellForD(real\_data, latent\_code).view(-1)

netD\_loss = output\_D.mean()

output\_G = self.myTrainOneStepCellForG(latent\_code).view(-1)

netG\_loss = output\_G.mean()

return netD\_loss, netG\_loss

# 实例化WithLossCell

netD\_with\_criterion = WithLossCellD(netD, netG, criterion)

netG\_with\_criterion = WithLossCellG(netD, netG, criterion)

# 实例化TrainOneStepCell

myTrainOneStepCellForD=nn.TrainOneStepCell(netD\_with\_criterion, optimizerD)

myTrainOneStepCellForG=nn.TrainOneStepCell(netG\_with\_criterion, optimizerG)

# 实例化DCGAN网络

dcgan = DCGAN(myTrainOneStepCellForD, myTrainOneStepCellForG)

dcgan.set\_train()

# 创建迭代器

data\_loader=data.create\_dict\_iterator(output\_numpy=True,num\_epochs=num\_epochs)

G\_losses = []

D\_losses = []

image\_list = []

# 开始循环训练

print("Starting Training Loop...")

for epoch in range(num\_epochs):

# 为每轮训练读入数据

for i, d in enumerate(data\_loader):

real\_data = Tensor(d['image'])

latent\_code = Tensor(d["latent\_code"])

netD\_loss, netG\_loss = dcgan(real\_data, latent\_code)

if i % 50 == 0 or i == size - 1:

# 输出训练记录

print('[%2d/%d][%3d/%d] Loss\_D:%7.4f Loss\_G:%7.4f' % (

epoch+1,num\_epochs,i+1,size,netD\_loss.asnumpy(),netG\_loss.asnumpy()))

D\_losses.append(netD\_loss.asnumpy())

G\_losses.append(netG\_loss.asnumpy())

# 每个epoch结束后，使用生成器生成一组图片

img = netG(fixed\_noise)

image\_list.append(img.transpose(0, 2, 3, 1).asnumpy())

# 保存网络模型参数为ckpt文件

save\_checkpoint(netG, "Generator.ckpt")

save\_checkpoint(netD, "Discriminator.ckpt")

### 2.4.3 训练结果

运行代码，描述生成器和判别器的损失训练迭代关系图

import matplotlib.pyplot as plt

plt.figure(figsize=(10, 5))

plt.title("Generator and Discriminator Loss During Training")

plt.plot(G\_losses, label="G", color='blue')

plt.plot(D\_losses, label="D", color='orange')

plt.xlabel("iterations")

plt.ylabel("Loss")

plt.legend()

plt.show()

输出：

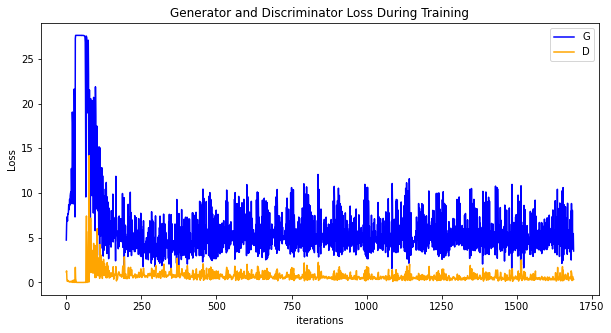


图 2-5 生成器和判别器训练关系图

import matplotlib.animation as animation

def showGif(image\_list):

show\_list = []

fig = plt.figure(figsize=(8, 3), dpi=120)

for epoch in range(len(image\_list)):

images = []

for i in range(3):

row = np.concatenate((image\_list[epoch][i \* 8:(i + 1) \* 8]), axis=1)

images.append(row)

img = np.clip(np.concatenate((images[:]), axis=0), 0, 1)

plt.axis("off")

show\_list.append([plt.imshow(img)])

ani = animation.ArtistAnimation(fig, show\_list, interval=1000, repeat\_delay=1000, blit=True)

ani.save('./dcgan.gif', writer='pillow', fps=1)

showGif(image\_list)

输出：



图2-6 图像训练

生成卡通头像

from mindspore import load\_checkpoint

from mindvision import dataset

dl\_path = "./netG"

dl\_url = "https://download.mindspore.cn/vision/classification/Generator.ckpt"

dl = dataset.DownLoad() # 下载Generator.ckpt文件

dl.download\_url(url=dl\_url, path=dl\_path)

# 从文件中获取模型参数并加载到网络中

param\_dict = load\_checkpoint("./netG/Generator.ckpt", netG)

img64 = netG(fixed\_noise).transpose(0, 2, 3, 1).asnumpy()

fig = plt.figure(figsize=(8, 3), dpi=120)

images = []

for i in range(3):

images.append(np.concatenate((img64[i \* 8:(i + 1) \* 8]), axis=1))

img = np.clip(np.concatenate((images[:]), axis=0), 0, 1)

plt.axis("off")

plt.imshow(img)

plt.show()

输出：



图2-7 卡通头像生成

## 2.5实验总结

本实验搭建了GAN网络训练模型并进行训练，最终生成卡通图像。

## 2.6 实验任务与参考解答

### 2.6.1生成器模型实验任务

#### 2.6.1.1 实验任务

利用图像反卷积构建生成器神经网络，网络包括5个反卷积层，4个Relu()激活函数。最后以1个Tanh()激活函数作为输出，最终生成144维图像。

#### 2.6.1.2 参考解答

from mindspore import nn

class Generator(nn.Cell):

def \_\_init\_\_(self):

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

self.generator = nn.SequentialCell()

self.generator.append(conv\_t(3, 6 4, 1, 0))

self.generator.append(nn.ReLU())

self.generator.append(conv\_t(6, 18 4, 1, 0))

self.generator.append(nn.ReLU())

self.generator.append(conv\_t(18, 36, 4, 2, 1))

self.generator.append(nn.ReLU())

self.generator.append(conv\_t(36, 72, 4, 2, 1))

self.generator.append(nn.ReLU())

self.generator.append(conv\_t(72, 144, 4, 2, 1))

self.generator.append(nn.Tanh())

def construct(self, x):

return self.generator(x)

### 2.6.2 判别器模型实验任务

#### 2.6.2.1实验任务

利用图像卷积构建判别器神经网络，网络包括5个卷积层，4个LeakRelu()激活函数。最后以1个SIgmoid()激活函数作为输出，得到1维判别概率。

#### 2.6.2.2 参考答案

from mindspore import nn

class Discriminator(nn.Cell):

def \_\_init\_\_(self):

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

self.discriminator = nn.SequentialCell()

self.discriminator.append(conv(144, 72, 4, 2, 1))

self.discriminator.append(nn.LeakyReLU(0.2))

self.discriminator.append(conv(72, 36, 4, 2, 1))

self.discriminator.append(nn.LeakyReLU(0.2))

self.discriminator.append(conv(36, 18, 4, 2, 1))

self.discriminator.append(nn.LeakyReLU(0.2))

self.discriminator.append(conv(18, 3 4, 2, 1))

self.discriminator.append(nn.LeakyReLU(0.2))

self.discriminator.append(conv(3, 1, 4, 1))

self.discriminator.append(nn.Sigmoid())

def construct(self, x):

return self.discriminator(x)