基于MindSpore的DCGAN生成漫画头像

实验手册



华为技术有限公司

|  |
| --- |
| 版权所有 © 华为技术有限公司 2024。 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](#_Toc176535173)

[1.1 实验目的 2](#_Toc176535174)

[1.2 实验清单 2](#_Toc176535175)

[1.3 开发平台介绍 2](#_Toc176535176)

[1.4 开发环境搭建 3](#_Toc176535177)

[2 DCGAN生成漫画头像 4](#_Toc176535178)

[2.1 DCGAN网络介绍背景知识 4](#_Toc176535179)

[2.1.1 GAN基础原理 4](#_Toc176535180)

[2.1.2 DCGAN 5](#_Toc176535181)

[2.2 数据下载与处理 5](#_Toc176535182)

[2.3 创建网络 8](#_Toc176535183)

[2.4 损失函数和优化器 10](#_Toc176535184)

[2.5 模型训练 11](#_Toc176535185)

[2.6 结果展示 13](#_Toc176535186)

[2.7 实验总结 15](#_Toc176535187)

# 实验介绍

生成式对抗网络是一种深度学习模型，是近年来复杂分布上无监督学习最具前景的方法之一。本实验通过MindSpore框架搭建DCGAN网络，并使用动漫头像数据集来训练此生成式对抗网络，接着使用该网络生成动漫头像图片。

## 实验目的

本案例使用MindSpore框架进行深度学习网络DCGAN搭建，利用动漫头像数据集进行模型训练、图像生成。通过本实验可以了解到深度学习任务开发的具体流程，包括：数据处理、创建网络、连接网络和损失函数 损失函数和优化器和模型训练及图像生成；了解经典神经网络DCGAN的结构特点和创新设计，以及深度学习网络搭建的重要概念。

## 实验清单

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 实验 | 简述 | 难度 | 软件环境 | 开发环境 |
| 基于MindSpore的DCGAN生成漫画头像 | 本实验使用MindSpore框架DCGAN网络，利用动漫头像数据集进行模型训练和图像生成 | 中级 | MindSpore2.2 | ModelArts |

## 开发平台介绍

昇腾训练处理器是华为自研的一款基于达芬奇架构的AI处理器，具有超高算力的和极高的能效比，最高可达320TFLOPS（FP16）的浮点计算能力。昇腾训练处理器的片上系统（SoC）还集成了多个CPU、DVPP和任务调度器，因而具有自我管理能力，可以充分发挥其高算力的优势。强大的矩阵、向量运行能力，在进行海量数据运算的神经网络训练场景有巨大的优势。

昇思MindSpore（官方网站：<https://www.mindspore.cn/>）是一种适用于端边云场景的新型开源深度学习训练/推理框架。 MindSpore提供了友好的设计和高效的执行，旨在提升数据科学家和算法工程师的开发体验，并为Ascend AI处理器提供原生支持，以及软硬件协同优化。同时，MindSpore作为全球AI开源社区，致力于进一步开发和丰富AI软硬件应用生态。

ModelArts（官方网站：<https://console.huaweicloud.com/modelarts/>）是面向AI开发者的一站式开发平台，提供海量数据预处理及半自动化标注、大规模分布式训练、自动化模型生成及模型按需部署能力，帮助用户快速创建和部署AI应用，管理全周期AI工作流。

## 开发环境搭建

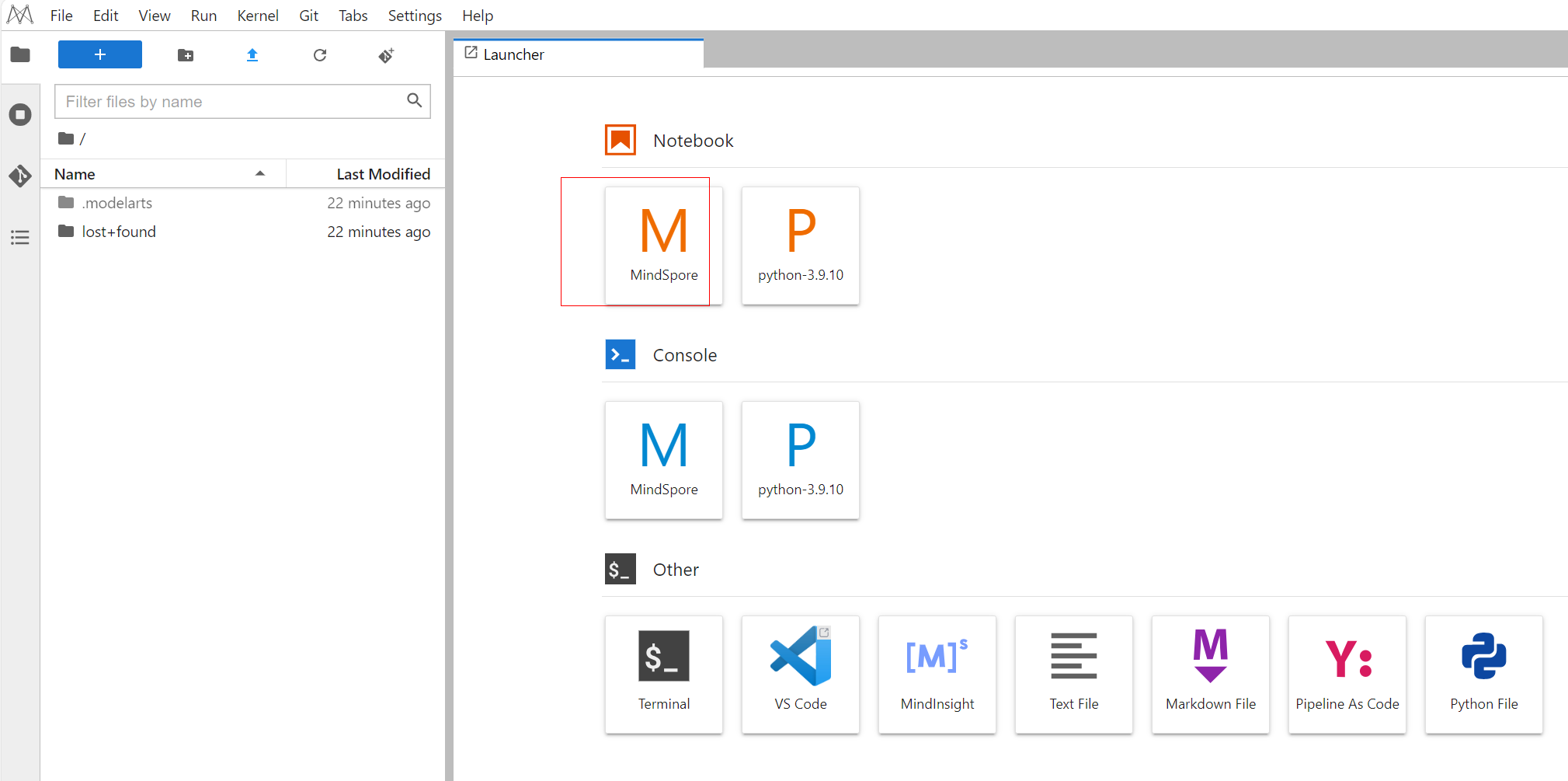
进入ModelArts开发环境

参考下方《华为云ModelArts环境搭建手册》，完成开发环境搭建。



打开Notebook

打开Notebook控制台后，新建或打开ipynb文件，选择MindSpore环境作为Kernel，即可开始编辑实验代码。



创建ipynb文件

\*注意：Notebook环境内上传、创建和编辑的文件均在/home/ma-user/work目录下。

# 基于MindSpore的DCGAN生成漫画头像

## DCGAN网络介绍背景知识

### GAN基础原理

最初，GAN由Ian Goodfellow于2014年发明，并在论文[Generative Adversarial Nets](https://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf" \t "https://mindspore.test.osinfra.cn/tutorials/application/zh-CN/master/cv/_blank)中首次进行了描述，GAN由两个不同的模型组成：**生成器**和**判别器**：

生成器的任务是生成看起来像训练图像的“假”图像；

判别器需要判断从生成器输出的图像是真实的训练图像还是虚假的图像。

在训练过程中，生成器会不断尝试通过生成更好的假图像来骗过判别器，而判别器在这过程中也会逐步提升判别能力。这种博弈的平衡点是，当生成器生成的假图像和训练数据图像的分布完全一致时，判别器拥有50%的真假判断置信度。

下面，我们首先定义一些在整个过程中需要用到的符号：

|  |  |
| --- | --- |
| 参数 | 参数说明 |
|  | 代表图像数据。 |
|  | 判别器网络，给出图像判定为真实图像的概率。 |

判别器参数表

由于我们在判别过程中需要处理图像，因此要为提供CHW格式且大小为3x64x64的图像。当来自训练数据时，数值应该趋近于1，而当来自生成器时，数值应该趋近于0。因此也可以被认为是传统的二分类器。

接下来我们来定义生成器的表示方法：

|  |  |
| --- | --- |
| 参数 | 参数说明 |
|  | 标准正态分布中提取出的隐向量。 |
|  | 表示将隐向量映射到数据空间的生成器函数。 |

生成器参数表

函数的目标是将一个随机高斯噪声通过一个生成网络生成一个和真实数据分布差不多的数据分布，其中θ是网络参数，我们希望找到θ使得和尽可能的接近。

是生成器G生成的假图像被判定为真实图像的概率。

如[Goodfellow 的论文](https://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf)中所述，D和G在进行一场博弈，D想要最大程度的正确分类真图像与假图像，也就是参数logD()；而G试图欺骗D来最小化假图像被识别到的概率，也就是参数log(1−D(G()))。GAN的损失函数为：

从理论上讲，此博弈游戏的平衡点是=，此时判别器会随机猜测输入是真图像还是假图像。然而，GAN的收敛可行性仍在研究当中，在实际场景中模型并不会被训练到这一步。

### DCGAN

DCGAN（深度卷积对抗生成网络，Deep Convolutional Generative Adversarial Networks）是GAN的直接扩展。不同之处在于，DCGAN会分别在判别器和生成器中使用卷积和卷积转置层。

它最早由Radford等人在论文[Unsupervised Representation Learning With Deep Convolutional Generative Adversarial Networks](https://arxiv.org/pdf/1511.06434.pdf" \t "https://mindspore.test.osinfra.cn/tutorials/application/zh-CN/master/cv/_blank)中进行描述。判别器由分层的卷积层、BatchNorm层和LeakyReLU激活层组成。输入是3x64x64的图像，输出是该图像为真图像的概率。生成器则是由转置卷积层、BatchNorm层和ReLU激活层组成。输入是标准正态分布中提取出的隐向量，输出是3x64x64的RGB图像。

本教程将使用70,171张动漫头像数据集来训练一个生成对抗网络，接着使用该网络生成动漫头像图片。

## 数据下载与处理

下载实验所需模块

开始实验之前，需要首先更新环境中的依赖包，在Notebook中输入以下命令并运行。

# 安装过程可能会有异常提示，不影响实验操作

!pip install download

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

数据下载

在本教程中，我们使用的动漫头像数据集共有70,171张动漫头像图片，图片大小均为96\*96。首先我们将数据集下载到指定目录下并解压。

示例代码如下：

from download import download

url = "https://download.mindspore.cn/dataset/Faces/faces.zip"

path = download(url, "./faces", kind="zip", replace=True)

下载后的数据集目录结构如下：

faces/

├── faces

   ├── 0.jpg

   ├── 1.jpg

   ├── 2.jpg

   ├── 3.jpg

   ├── 4.jpg

   ├── …

├── 70169.jpg

   └── 70170.jpg

数据处理相关的一些输入定义：

|  |  |
| --- | --- |
| 参数 | 参数说明 |
| batch\_size | 训练中使用的批量大小，DCGAN论文使用的批量大小为128； |
| image\_size | 训练图像的大小，此实现默认为`64x64`，如果需要其他尺寸，则必须同时更改`D`和`G`的结构； |
| nc | 输入图像中的彩色通道数，因为此次是彩色图像所以设为3； |
| nz | 隐向量的长度； |
| ngf | 设置通过生成器的特征图的深度； |
| ndf | 设置通过判别器传播的特征图的深度； |
| num\_epochs | 要运行的训练周期数，训练更长的时间可能会导致更好的结果，但也会花费更长的时间； |
| lr | 训练的学习率； |
| beta1 | Adam优化器的`beta1`超参数。如DCGAN论文所述，该数字应为0.5； |

模型参数说明

batch\_size = 128 # 批量大小

image\_size = 64 # 训练图像空间大小

nc = 3 # 图像彩色通道数

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

定义`create\_dataset\_imagenet`函数对数据进行处理和增强操作。

import numpy as np

import mindspore.dataset as ds

import mindspore.dataset.vision as vision

def create\_dataset\_imagenet(dataset\_path):

"""数据加载"""

dataset = ds.ImageFolderDataset(dataset\_path,

num\_parallel\_workers=4,

shuffle=True,

decode=True)

# 数据增强操作

transforms = [

vision.Resize(image\_size),

vision.CenterCrop(image\_size),

vision.HWC2CHW(),

lambda x: ((x / 255).astype("float32"))

]

# 数据映射操作

dataset = dataset.project('image')

dataset = dataset.map(transforms, 'image')

# 批量操作

dataset = dataset.batch(batch\_size)

return dataset

dataset = create\_dataset\_imagenet('./faces')

通过create\_dict\_iterator函数将数据转换成字典迭代器，然后使用`matplotlib`模块可视化部分训练数据。

import matplotlib.pyplot as plt

def plot\_data(data):

# 可视化部分训练数据

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

for i, image in enumerate(data[0][:30], 1):

plt.subplot(3, 10, i)

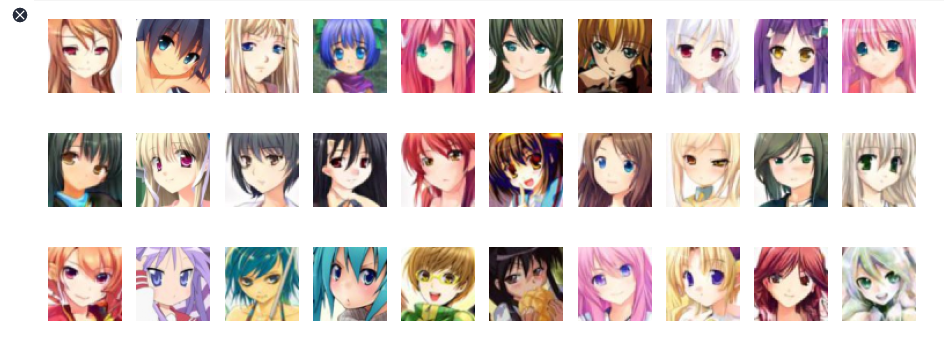
plt.axis("off")

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

plt.show()

sample\_data = next(dataset.create\_tuple\_iterator(output\_numpy=True))

plot\_data(sample\_data)



训练数据图

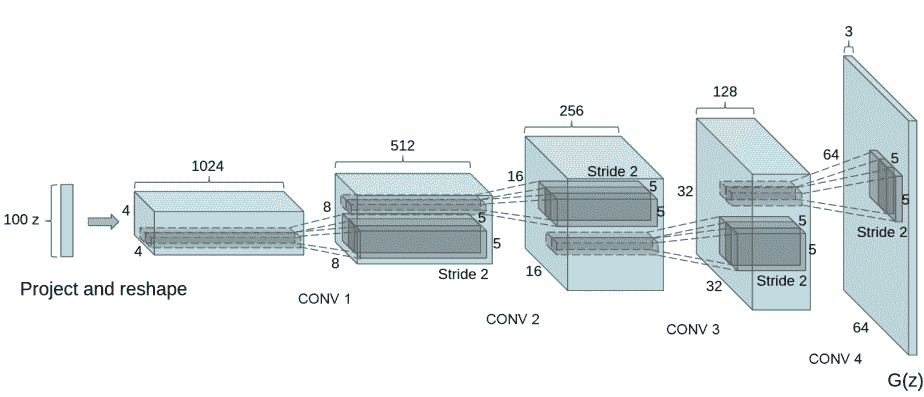
## 创建网络

当处理完数据后，就可以来进行网络的搭建了。按照DCGAN论文中的描述，所有模型权重均应从mean为0，sigma为0.02的正态分布中随机初始化。

生成器

生成器G的功能是将隐向量z映射到数据空间。由于数据是图像，这一过程也会创建与真实图像大小相同的 RGB 图像。在实践场景中，该功能是通过一系列Conv2dTranspose转置卷积层来完成的，每个层都与BatchNorm2d层和ReLu激活层配对，输出数据会经过tanh函数，使其返回[-1,1]的数据范围内。

DCGAN论文生成图像如下所示：



我们通过输入部分中设置的nz、ngf和nc来影响代码中的生成器结构。nz是隐向量z的长度，ngf与通过生成器传播的特征图的大小有关，nc是输出图像中的通道数。

以下是生成器的代码实现：

import mindspore as ms

from mindspore import nn, ops

from mindspore.common.initializer import Normal

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

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

class Generator(nn.Cell):

"""DCGAN网络生成器"""

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

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

self.generator = nn.SequentialCell(

nn.Conv2dTranspose(nz, ngf \* 8, 4, 1, 'valid', weight\_init=weight\_init),

nn.BatchNorm2d(ngf \* 8, gamma\_init=gamma\_init),

nn.ReLU(),

nn.Conv2dTranspose(ngf \* 8, ngf \* 4, 4, 2, 'pad', 1, weight\_init=weight\_init),

nn.BatchNorm2d(ngf \* 4, gamma\_init=gamma\_init),

nn.ReLU(),

nn.Conv2dTranspose(ngf \* 4, ngf \* 2, 4, 2, 'pad', 1, weight\_init=weight\_init),

nn.BatchNorm2d(ngf \* 2, gamma\_init=gamma\_init),

nn.ReLU(),

nn.Conv2dTranspose(ngf \* 2, ngf, 4, 2, 'pad', 1, weight\_init=weight\_init),

nn.BatchNorm2d(ngf, gamma\_init=gamma\_init),

nn.ReLU(),

nn.Conv2dTranspose(ngf, nc, 4, 2, 'pad', 1, weight\_init=weight\_init),

nn.Tanh()

)

def construct(self, x):

return self.generator(x)

generator = Generator()

判别器

如前所述，判别器D是一个二分类网络模型，输出判定该图像为真实图的概率。通过一系列的Conv2d、BatchNorm2d和LeakyReLU层对其进行处理，最后通过Sigmoid激活函数得到最终概率。

DCGAN论文提到，使用卷积而不是通过池化来进行下采样是一个好方法，因为它可以让网络学习自己的池化特征。

判别器的代码实现如下：

class Discriminator(nn.Cell):

"""DCGAN网络判别器"""

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

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

self.discriminator = nn.SequentialCell(

nn.Conv2d(nc, ndf, 4, 2, 'pad', 1, weight\_init=weight\_init),

nn.LeakyReLU(0.2),

nn.Conv2d(ndf, ndf \* 2, 4, 2, 'pad', 1, weight\_init=weight\_init),

nn.BatchNorm2d(ngf \* 2, gamma\_init=gamma\_init),

nn.LeakyReLU(0.2),

nn.Conv2d(ndf \* 2, ndf \* 4, 4, 2, 'pad', 1, weight\_init=weight\_init),

nn.BatchNorm2d(ngf \* 4, gamma\_init=gamma\_init),

nn.LeakyReLU(0.2),

nn.Conv2d(ndf \* 4, ndf \* 8, 4, 2, 'pad', 1, weight\_init=weight\_init),

nn.BatchNorm2d(ngf \* 8, gamma\_init=gamma\_init),

nn.LeakyReLU(0.2),

nn.Conv2d(ndf \* 8, 1, 4, 1, 'valid', weight\_init=weight\_init),

)

self.adv\_layer = nn.Sigmoid()

def construct(self, x):

out = self.discriminator(x)

out = out.reshape(out.shape[0], -1)

return self.adv\_layer(out)

discriminator = Discriminator()

## 损失函数和优化器

当定义了D和G后，接下来将使用MindSpore中定义的二进制交叉熵损失函数[BCELoss](https://www.mindspore.cn/docs/api/zh-CN/master/api_python/nn/mindspore.nn.BCELoss.html" \t "https://mindspore.test.osinfra.cn/tutorials/application/zh-CN/master/cv/_self) ，为D和G加上损失函数和优化器。

# 定义损失函数

adversarial\_loss = nn.BCELoss(reduction='mean')

这里设置了两个单独的优化器，一个用于D，另一个用于G。这两个都是lr = 0.0002和beta1 = 0.5的Adam优化器。

# 为生成器和判别器设置优化器

optimizer\_D = nn.Adam(discriminator.trainable\_params(), learning\_rate=lr, beta1=beta1)

optimizer\_G = nn.Adam(generator.trainable\_params(), learning\_rate=lr, beta1=beta1)

optimizer\_G.update\_parameters\_name('optim\_g.')

optimizer\_D.update\_parameters\_name('optim\_d.')

## 模型训练

训练分为两个主要部分：训练判别器和训练生成器。

训练判别器

训练判别器的目的是最大程度地提高判别图像真伪的概率。按照Goodfellow的方法，是希望通过提高其随机梯度来更新判别器，所以我们要最大化的值。

训练生成器

如DCGAN论文所述，我们希望通过最小化来训练生成器，以产生更好的虚假图像。

在这两个部分中，分别获取训练过程中的损失，并在每个周期结束时进行统计，将fixed\_noise批量推送到生成器中，以直观地跟踪G的训练进度。

下面实现模型训练正向逻辑：

def generator\_forward(real\_imgs, valid):

# 将噪声采样为发生器的输入

z = ops.standard\_normal((real\_imgs.shape[0], nz, 1, 1))

# 生成一批图像

gen\_imgs = generator(z)

# 损失衡量发生器绕过判别器的能力

g\_loss = adversarial\_loss(discriminator(gen\_imgs), valid)

return g\_loss, gen\_imgs

def discriminator\_forward(real\_imgs, gen\_imgs, valid, fake):

# 衡量鉴别器从生成的样本中对真实样本进行分类的能力

real\_loss = adversarial\_loss(discriminator(real\_imgs), valid)

fake\_loss = adversarial\_loss(discriminator(gen\_imgs), fake)

d\_loss = (real\_loss + fake\_loss) / 2

return d\_loss

grad\_generator\_fn = ms.value\_and\_grad(generator\_forward, None,

optimizer\_G.parameters,

has\_aux=True)

grad\_discriminator\_fn = ms.value\_and\_grad(discriminator\_forward, None,

optimizer\_D.parameters)

@ms.jit

def train\_step(imgs):

valid = ops.ones((imgs.shape[0], 1), mindspore.float32)

fake = ops.zeros((imgs.shape[0], 1), mindspore.float32)

(g\_loss, gen\_imgs), g\_grads = grad\_generator\_fn(imgs, valid)

optimizer\_G(g\_grads)

d\_loss, d\_grads = grad\_discriminator\_fn(imgs, gen\_imgs, valid, fake)

optimizer\_D(d\_grads)

return g\_loss, d\_loss, gen\_imgs

循环训练网络，每经过50次迭代，就收集生成器和判别器的损失，以便于后面绘制训练过程中损失函数的图像。

import mindspore

G\_losses = []

D\_losses = []

image\_list = []

total = dataset.get\_dataset\_size()

for epoch in range(num\_epochs):

generator.set\_train()

discriminator.set\_train()

# 为每轮训练读入数据

for i, (imgs, ) in enumerate(dataset.create\_tuple\_iterator()):

g\_loss, d\_loss, gen\_imgs = train\_step(imgs)

if i % 100 == 0 or i == total - 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, total, d\_loss.asnumpy(), g\_loss.asnumpy()))

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

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

# 每个epoch结束后，使用生成器生成一组图片

generator.set\_train(False)

fixed\_noise = ops.standard\_normal((batch\_size, nz, 1, 1))

img = generator(fixed\_noise)

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

# 保存网络模型参数为ckpt文件

mindspore.save\_checkpoint(generator, "./generator.ckpt")

mindspore.save\_checkpoint(discriminator, "./discriminator.ckpt")

完整的10个epoch模型训练结果，如下图：

[ 1/10][ 1/549] Loss\_D: 0.8013 Loss\_G: 0.5065

[ 1/10][101/549] Loss\_D: 0.1116 Loss\_G:13.0030

[ 1/10][201/549] Loss\_D: 0.1037 Loss\_G: 2.5631

...

[ 1/10][401/549] Loss\_D: 0.6240 Loss\_G: 0.5548

[ 1/10][501/549] Loss\_D: 0.3345 Loss\_G: 1.6001

[ 1/10][549/549] Loss\_D: 0.4250 Loss\_G: 1.1978

...

[10/10][501/549] Loss\_D: 0.2898 Loss\_G: 1.5352

[10/10][549/549] Loss\_D: 0.2120 Loss\_G: 3.1816

## 结果展示

运行下面代码，描绘‘D’和‘G’损失与训练迭代的关系图：

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

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

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

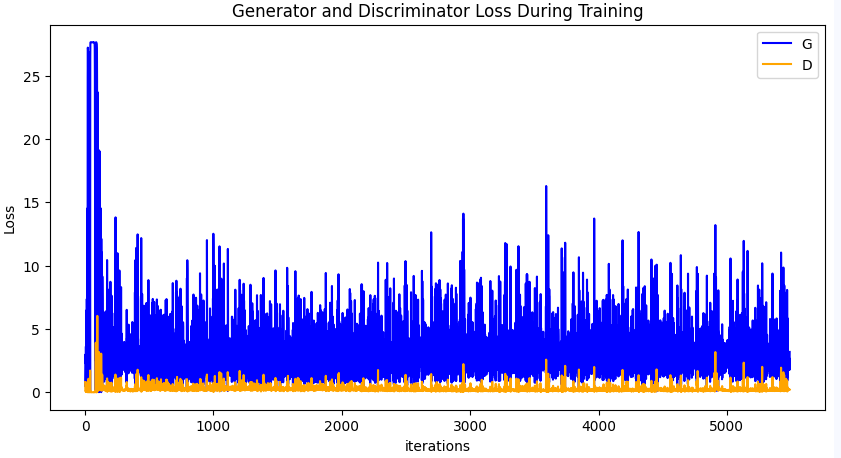
plt.plot(D\_losses, label="D", color='orange')

plt.xlabel("iterations")

plt.ylabel("Loss")

plt.legend()

plt.show()



可视化训练过程中通过隐向量`fixed\_noise`生成的图像。

import matplotlib.pyplot as plt

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)



从上面的图像可以看出，随着训练次数的增多，图像质量也越来越好。如果增大训练周期数，当num\_epochs达到50以上时，生成的动漫头像图片与数据集中的较为相似，下面我们通过加载生成器网络模型参数文件来生成图像，代码如下：

# 从文件中获取模型参数并加载到网络中

mindspore.load\_checkpoint("./generator.ckpt", generator)

fixed\_noise = ops.standard\_normal((batch\_size, nz, 1, 1))

img64 = generator(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()



## 实验总结

本章实验在华为云ModelArts平台上使用MindSpore完整地实现了一个基于DCGAN的生成式对抗网络模型，主要是利用动漫头像数据集进行模型训练和图像生成，通过实验使学员了解生成式对抗网络的算法模型，数据处理，模型训练等知识内容。