南開大學
Nankai University

计算机学院

计算机图形学实验报告

基于计图（jittor）框架实现 Conditional GAN 模型

姓名：杨至尧

学号：2213904

专业：计算机科学与技术

2024 年 12 月 7 日

目录

1 引言

条件生成对抗网络（cGAN）是一种扩展的生成对抗网络（GAN），其主要特点是通过引入条件信息来生成特定类型的样本。cGAN 的架构包括两个主要组件：生成器和判别器。生成器的输入不仅包括随机噪声向量，还包括条件信息，例如类别标签或其他描述性信息。这样，生成器能够根据给定的条件生成相应的样本，例如生成特定类别的图像。判别器的任务是判断输入样本的真实性，同时也接收条件信息，以便更好地评估样本是否符合条件。训练过程中，生成器和判别器通过对抗训练相互竞争，生成器试图生成能够欺骗判别器的样本，而判别器则努力正确区分真实样本和生成样本。cGAN 的损失函数设计使得生成器的目标是最大化判别器对生成样本的误判，而判别器则最小化对真实样本和生成样本的判断错误。cGAN 在许多应用场景中表现出色，如图像生成、图像到图像的转换以及文本生成等。

2 实验方法

2.1 数据集

模型在 MNIST 数据集上进行了实验，MNIST 数据集一共有 7 万张图片，其中 6 万张是训练集，1 万张是测试集。每张图片是 28×28 的 0–9 的手写数字图片组成。每个图片是黑底白字的形式，黑底用 0 表示，白字用 0-1 之间的浮点数表示，越接近 1，颜色越白。

图片的标签以一维数组的 one-hot 编码形式给出，例如 [0 , 0 , 0 , 0 , 0 , 1 , 0 , 0 , 0 , 0] 这个标签就代表数字 5。

在本次实验中，使用 jittor.dataset.mnist 库导入 MNIST 模型，并利用 jittor.transform 库进行数据预处理，相关代码如下：

数据集导入和数据预处理

```
transform = transform.Compose([
    transform.Resize(opt.img_size),
    transform.Gray(),
    transform.ImageNormalize(mean=[0.5], std=[0.5]),
])
dataloader = MNIST(train=True,
    transform=transform).set_attrs(batch_size=opt.batch_size, shuffle=True)
```

可以看到，在预处理中，首先统一图像的尺寸，再将输入图像转换为灰度图像，并且对图像进行归一化处理，使其像素值的均值为 0.5，标准差为 0.5。

2.2 模型

生成器的主要任务是根据输入的随机噪声和条件标签生成图像。其架构如图2.1所示：

嵌入层：使用 nn.Embedding 将类别标签转换为嵌入向量。这个嵌入层的输出与随机噪声向量连接，形成生成器的输入。

全连接层：生成器包含多个全连接层，每个层后面通常跟随一个批归一化层和 Leaky ReLU 激活函数。具体结构为：第一个全连接层将输入映射到 128 维。第二个全连接层将 128 维映射到 256 维。第三个全连接层将 256 维映射到 512 维。第四个全连接层将 512 维映射到 1024 维。

输出层：最后一个全连接层将 1024 维的输出映射到图像的总特征数（即图像的像素数量），并使用 Tanh 激活函数将输出值限制在 [-1, 1] 的范围内，以适应生成图像的标准化。生成的图像特征向量被调整为目标图像的形状，以便后续处理。

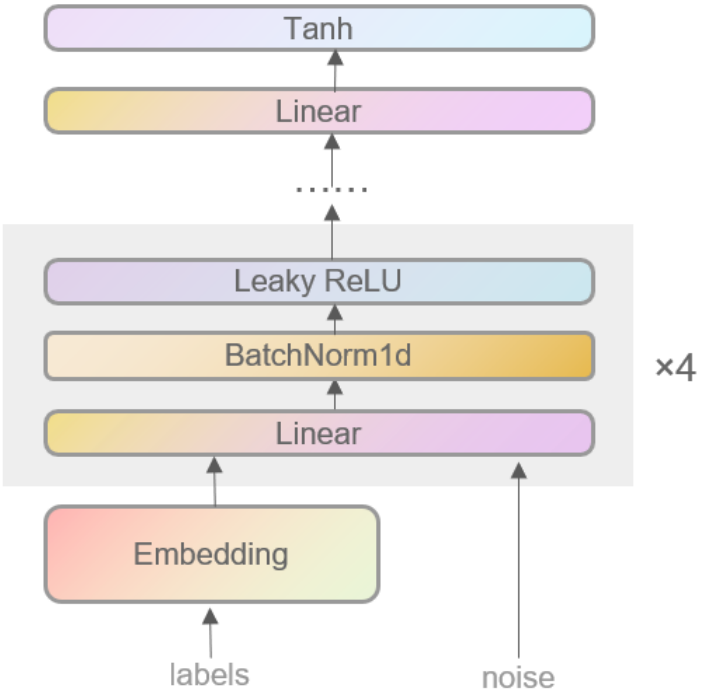

图 2.1: 生成器架构图

判别器的主要任务是判断输入图像的真实性，即判断图像是来自真实数据集还是由生成器生成的。其架构如图2.2所示：

嵌入层：同样使用 nn.Embedding 将类别标签转换为嵌入向量，以便与图像特征结合。输入层：判别器的输入是一个连接了展平的图像特征和类别嵌入的向量。图像特征通过 img.view((img.shape[0], (-1))) 展平，形成一个一维向量。

全连接层：判别器也包含多个全连接层，结构为：第一个全连接层将输入映射到 512 维，并使用 Leaky ReLU 激活函数。第二个全连接层将 512 维映射到 512 维，并使用 Dropout 层和 Leaky ReLU 激活函数以减少过拟合。第三个全连接层再次将 512 维映射到 512 维，并使用 Dropout 层和 Leaky ReLU 激活函数。

输出层：最后一个全连接层将 512 维的输出映射到一个标量值，表示输入图像的真实性评分。这个输出值可以是任意实数，用于与真实标签（1）和生成标签（0）进行比较。

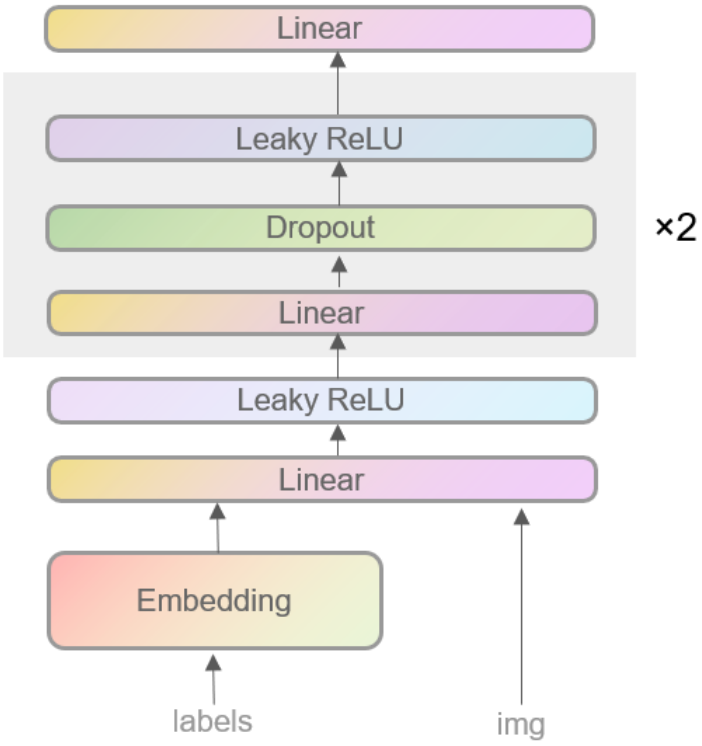

图 2.2: 判别器架构图

2.3 训练过程

在模型训练过程中，首先通过遍历数据加载器获取真实图像和对应的标签。对于每个训练轮次，生成器首先生成一批图像，输入包括随机噪声和随机选择的标签。生成的图像随后被传递给判别器，以评估其真实性，并计算生成器的损失，目标是使生成的图像尽可能“真实”。接着，判别器被训练以区分真实图像和生成图像，分别计算真实图像和生成图像的损失，并取其平均值作为判别器的总损失。通过这种对抗训练，生成器和判别器相互竞争，逐步提高生成图像的质量和判别器的判断能力。

在本次实验中，使用均方误差（MSE）作为损失函数并使用 Adam 优化器来优化模型的参数：

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

其中：n 是样本的总数量，y_i 是第 i 个样本的真实值，$\hat{y}_i$ 是第 i 个样本的预测值。

模型训练过程的具体代码如下所示（省略了计算 FID 分数和保存模型的部分代码）：

模型训练

```
for epoch in range(opt.n_epochs):
    for i, (imgs, labels) in enumerate(dataloader):

        batch_size = imgs.shape[0]

        # Adversarial ground truths
        valid = jt.ones([batch_size, 1]).float32().stop_grad()
        fake = jt.zeros([batch_size, 1]).float32().stop_grad()
```

```
9
10          # Configure input
11          real_imgs = jt.array(imgs)
12          labels = jt.array(labels)
13
14          # ---------------------
15          #  Train Generator
16          # ---------------------
17
18          # Sample noise and labels as generator input
19          z = jt.array(np.random.normal(0, 1, (batch_size, opt.latent_dim))).float32()
20          gen_labels = jt.array(np.random.randint(0, opt.n_classes,
                batch_size)).float32()
21
22          # Generate a batch of images
23          gen_imgs = generator(z, gen_labels)
24          # Loss measures generator's ability to fool the discriminator
25          validity = discriminator(gen_imgs, gen_labels)
26          g_loss = adversarial_loss(validity, valid)
27          g_loss.sync()
28          optimizer_G.step(g_loss)
29
30          # ---------------------
31          #  Train Discriminator
32          # ---------------------
33
34          # Loss for real images
35          validity_real = discriminator(real_imgs, labels)
36          d_real_loss = adversarial_loss(validity_real, valid)
37
38          # Loss for fake images
39          validity_fake = discriminator(gen_imgs.stop_grad(), gen_labels)
40          d_fake_loss = adversarial_loss(validity_fake, fake)
41
42          # Total discriminator loss
43          d_loss = (d_real_loss + d_fake_loss) / 2
44          d_loss.sync()
45          optimizer_D.step(d_loss)
46          if i % 50 == 0:
47              print(
48                  "[Epoch %d/%d] [Batch %d/%d] [D loss: %f] [G loss: %f]"
49                  % (epoch, opt.n_epochs, i, len(dataloader), d_loss.data, g_loss.data)
50              )
51
52          batches_done = epoch * len(dataloader) + i
53          if batches_done % opt.sample_interval == 0:
54              sample_image(n_row=10, batches_done=batches_done)
```

3 实验结果

3.1 训练过程中损失函数的变化

在训练过程中，每个 epoch 对生成器和判别器的损失进行一次采集，在训练完成后绘制损失函数变化折线图，结果如图3.3所示：

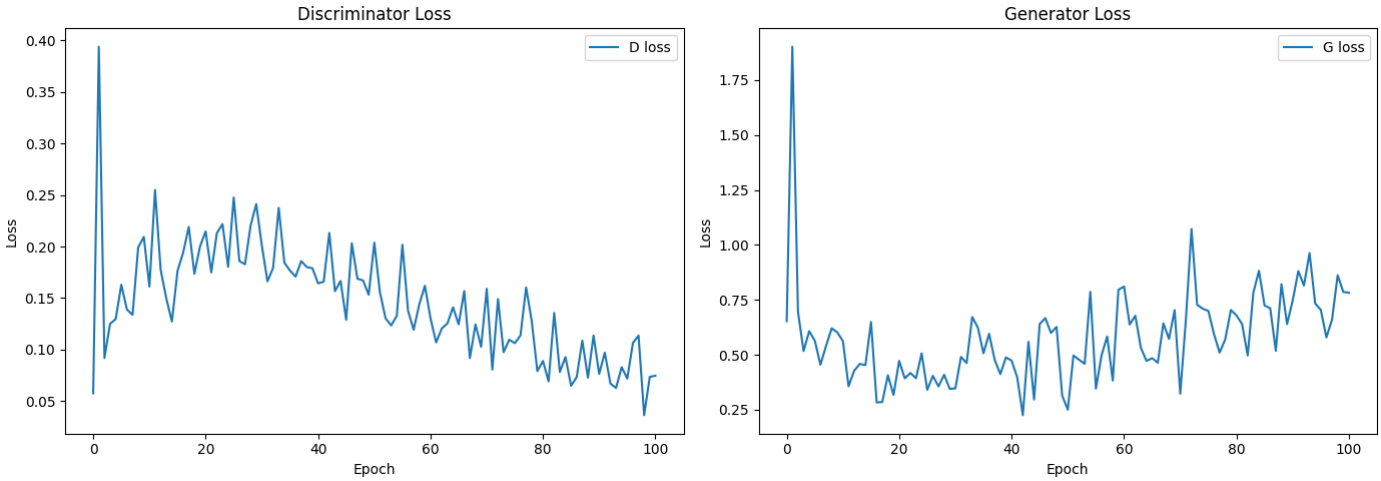

图 3.3: 损失函数变化折线图

可以看到：在前 40 个 epoch 中，判别器 loss 在 0.2 附近上下波动，生成器 loss 在 0.5 附近上下波动。在后 60 个 epoch 中，判别器 loss 持续波动，但大体呈现下降趋势，而生成器 loss 大体呈现上升趋势。这表明在训练初期，生成器和判别器之间的竞争相对平衡，两者都在逐步提高自己的性能，生成器在尝试生成更逼真的图像，而判别器在尝试更准确地识别这些图像。而在后 60 个 epoch 中，判别器变得更强大，判别器的损失下降，意味着它在识别真实图像和生成图像方面变得更准确。这表明判别器在训练过程中逐渐学习到了更有效的特征，能够更好地区分真实图像和生成图像。而生成器在对抗中逐渐处于下风，但生成器也可能在持续提高生成图像的逼真度，只是由于生成器的提升更快，从而无法在损失函数图像中体现出来。

为了验证生成器是否在继续提高，我还在训练过程中进行了 FID 分数的计算。

3.2 训练过程中 FID 分数的变化

FID 是一种用于评估生成模型和真实数据分布之间差异的指标。FID 是通过计算两个分布之间的 Fréchet 距离来衡量生成模型和真实数据分布之间的差异。在计算 FID 时，首先从真实数据分布和生成模型中分别抽取一组样本，然后使用预训练的 Inception 网络从这些样本中提取特征向量。接下来，计算两个分布的均值和协方差矩阵，并计算它们之间的 Fréchet 距离，得到 FID 值。FID 值越小，表示生成模型生成的图像越接近于真实数据分布。FID 的计算公式为：

$$\mathrm{FID}(x, g) = \|\mu_x - \mu_g\| + \mathrm{Tr}\left(\Sigma_x + \Sigma_g - 2\sqrt{\Sigma_x \Sigma_g}\right)$$

其中：μ_x 和 Σ_x 分别是真实图像集合在 Inception Net-V3 输出的 2048 维特征向量集合的均值和协方差矩阵，μ_g 和 Σ_g 分别是生成图像集合在 Inception Net-V3 输出的 2048 维特征向量集合的均值和协方差矩阵。

对于 Inception Net-V3 模型，jittor 提供了预训练好的模型，使用 jittor.models.inception 包调用 Inception Net-V3 模型用于图像特征的提取。

在训练过程中，每 10 个 epoch 对生成器生成的图像进行一次和真实图像的对比，每次对比随机从测试集中抽取 100 个图像与生成器生成的 100 个图像进行对比，在训练完成后绘制 FID 分数变化折线图，结果如图3.4所示：

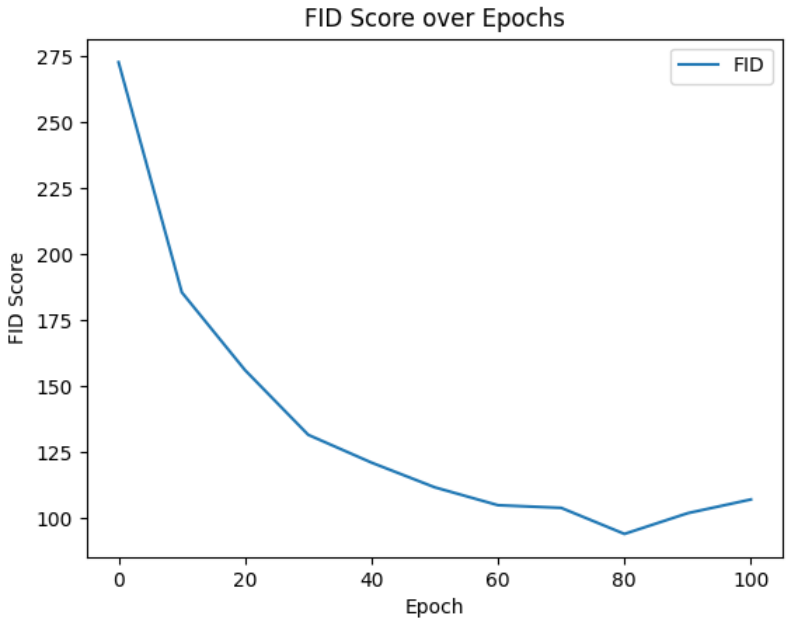

图 3.4: FID 分数变化折线图

可以看到：随着模型的训练，FID 分数逐渐变小，最终在达到 100 左右，这说明生成图片分布与真实图片分布之间较为接近，生成器的生成效果较好。

3.3 训练过程中生成的图片变化

在训练过程中，每经过 1000 个批次，保存一张生成的图像，用于展示训练过程中生成器的效果。结果如图3.5，5.7所示：

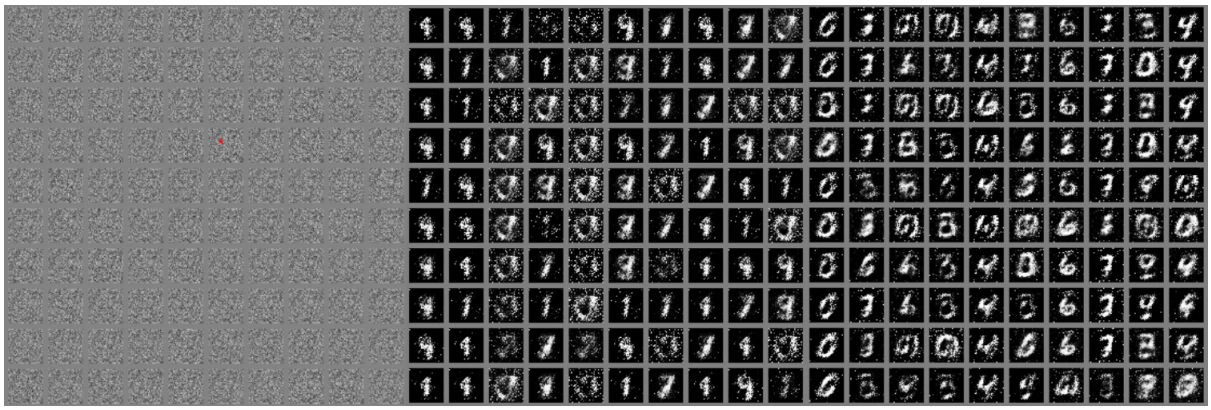

图 3.5: 0,1000,2000 批次生成效果图

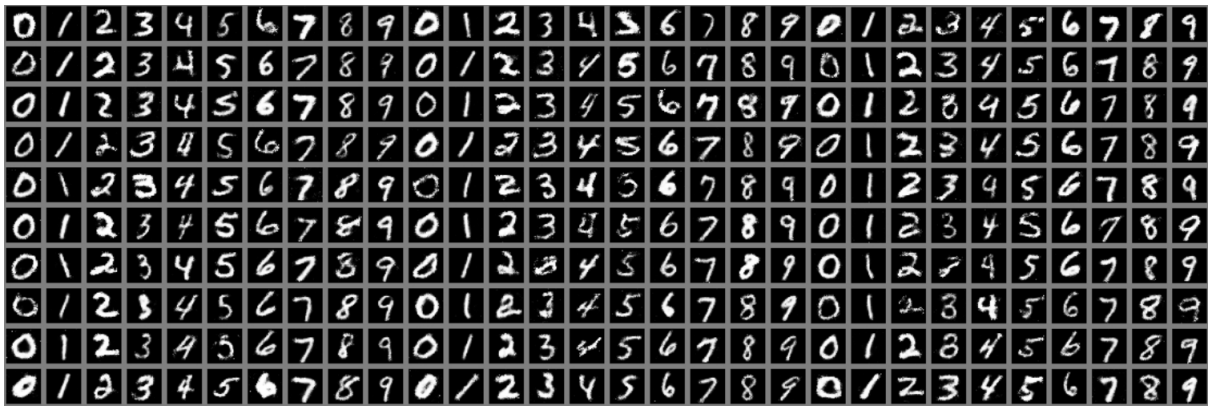

图 3.6: 92000，93000，94000 批次生成效果图

可以看到：还未进行训练时，生成的图像布满噪点，看不出数字的形状，随着训练批次数的增多，图像的噪点逐渐减少，数字的形状逐渐清晰，到最后生成的图像已经具有很高的辨识度。

4 总结和思考

在本次实验中，我实现并训练了一个条件生成对抗网络（cGAN），我使用了标准的 MNIST 数据集，生成手写数字图像。在训练过程中，生成器和判别器通过对抗训练相互竞争，逐步提高了生成图像的质量。实验结果表明，随着训练的进行，生成的图像质量逐渐提高，判别器的准确性也得到了提升。通过计算 (FID) 分数，我能够量化生成图像与真实图像之间的相似性，进一步验证了模型的有效性。

但在训练的后期，出现了判别器和生成器不平衡竞争的问题，生成器的损失上升，意味着生成器生成的图像越来越难以欺骗判别器。经过查阅资料，我发现可以通过简化判别器网络结构，降低参数量，或者减小判别器的学习率等操作来解决这个问题，但由于使用 Ubuntu 虚拟机进行模型训练速度过慢，来不及进行网络结构和学习率等方面的调整。

5 Gitlink 项目开源地址与主页截图

Gitlink 项目开源地址为：https://gitlink.org.cn/iverson_yao/cgan_jittor

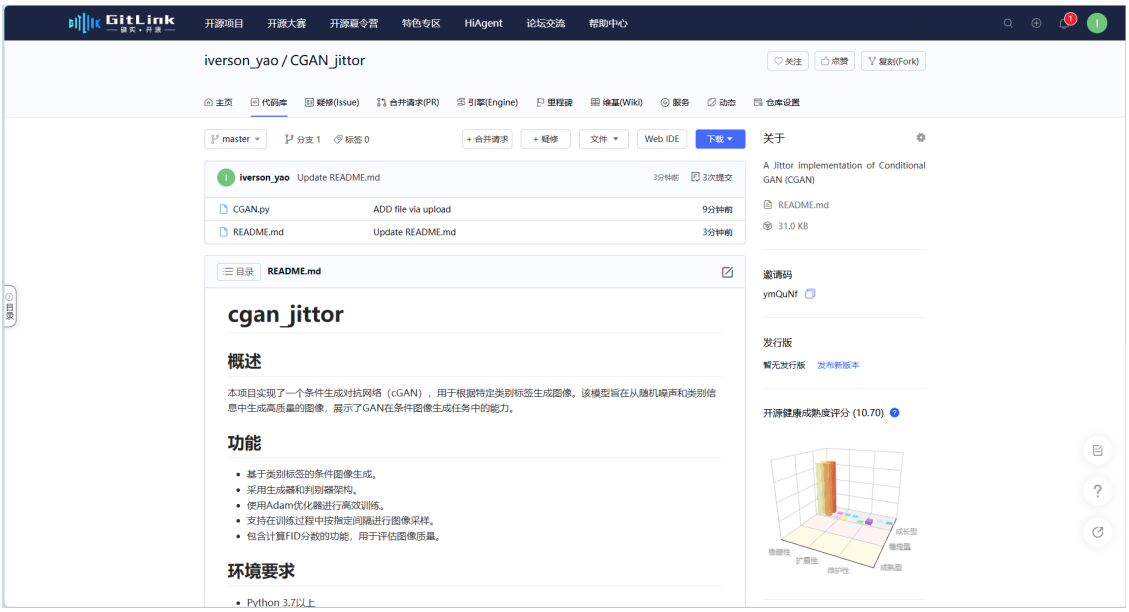
GitLink
开源项目
开源大赛
开源夏令营
特色专区
HiAgent
论坛交流
帮助中心
iverson_yao / CGAN_jittor
关注
点赞
复刻(Fork)
主页
代码库
疑修(Issue)
合并请求(PR)
引擎(Engine)
里程碑
维基(Wiki)
服务
动态
仓库设置
master
分支 1
标签 0
+ 合并请求
+ 疑修
文件
Web IDE
下载
iverson_yao Update README.md
3分钟前
3次提交
CGAN.py
ADD file via upload
9分钟前
README.md
Update README.md
3分钟前
目录
README.md
cgan_jittor
概述
本项目实现了一个条件生成对抗网络（cGAN），用于根据特定类别标签生成图像。该模型旨在从随机噪声和类别信息中生成高质量的图像，展示了GAN在条件图像生成任务中的能力。
功能
基于类别标签的条件图像生成。
采用生成器和判别器架构。
使用Adam优化器进行高效训练。
支持在训练过程中按指定间隔进行图像采样。
包含计算FID分数的功能，用于评估图像质量。
环境要求
Python 3.7以上
关于
A Jittor implementation of Conditional GAN (CGAN)
README.md
31.0 KB
邀请码
ymQuNf
发行版
暂无发行版
发布新版本
开源健康成熟度评分 (10.70)

图 5.7: 主页截图