# 武汉大学国家网络安全学院
# 本科生实验报告

## 《内容安全》第六次课堂作业

专 业 名 称 ：网络空间安全

课 程 名 称 ：内容安全

学 生 学 号 ：2022302181129

学 生 姓 名 ：王佳瑶

二o二五年四月

目录

# 1.实验内容和目的

(1) 认真观看 PPT 内容
(2) 观看 B 站李宏毅老师的关于 GAN 的课程视频
(3) 采用 DCGAN 或 WGAN 或 WGAN-GP 生成 1000 张不同的二次元漫画脸图像
(4) 完成 PDF 格式的实验报告，生成的漫画脸图像，源代码和程序运行截屏视频用 ZIP 格式打包发送

# 2.实验环境

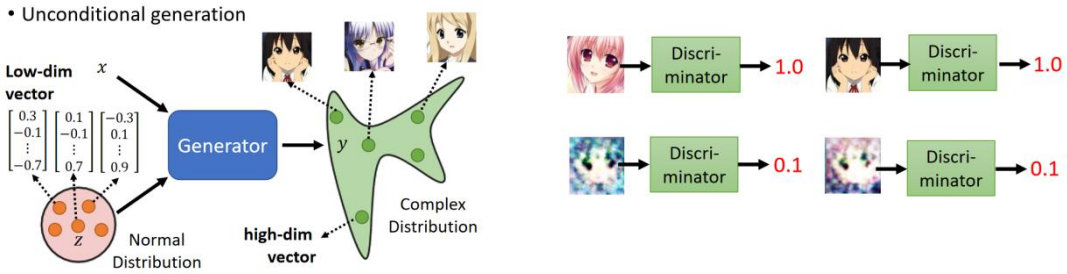
（1）硬件环境：Windows 11
（2）软件环境：Pycharm + Google Lab

# 3.任务一 学习关于 GAN 的课程视频

## 3.1 GAN 的原理和架构

GAN 由生成器（Generator）和判别器（Discriminator）组成。**生成器用于生成数据**（如图片、文章），输入一个随机向量，输出一个随机的“长向量”；**判别器用于判别数据的真实程度**，输入一个数据，输出一个标量（0-1），值越大表示越真实。

例如下图展示了生成器和判别器的核心工作，




生成对抗网络的核心思想是**通过两个网络之间的对抗来训练：生成器生成接近真实数据的样本，而判别器则努力将生成样本与真实样本区分开来。**这种对抗促使生成器不断提升生成质量，从而达到拟合真实数据分布的目的。

### 3.1.1 训练过程

**（1）基础 GAN 训练过程：**

① 准备真实图片数据集，

② 随机初始化生成器和判别器。

③ 先训练判别器（用生成器生成的图片和真实图片让其分辨真假并更新参数，会告诉 Discriminator 真实的图片长什么样子），

④ 再训练生成器（用生成器生成的图片让判别器分辨真假并更新参数，注意，本次只会让 Discriminator 看生成的图片）

⑤ 重复迭代步骤 3、4 直至满意。

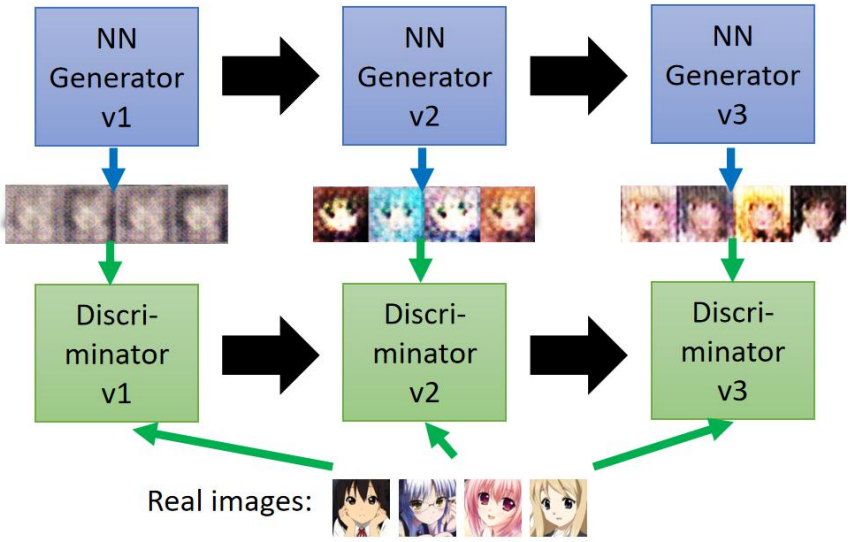
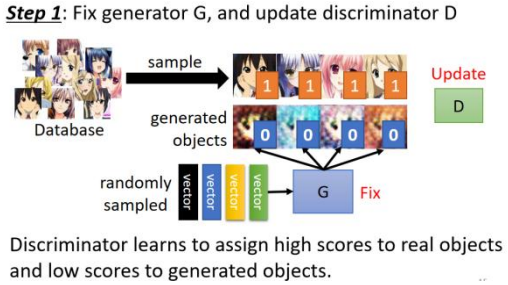
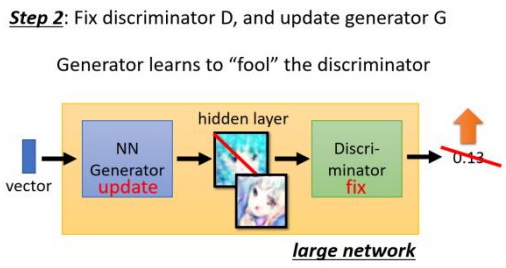



图 1 GAN 训练过程示意图



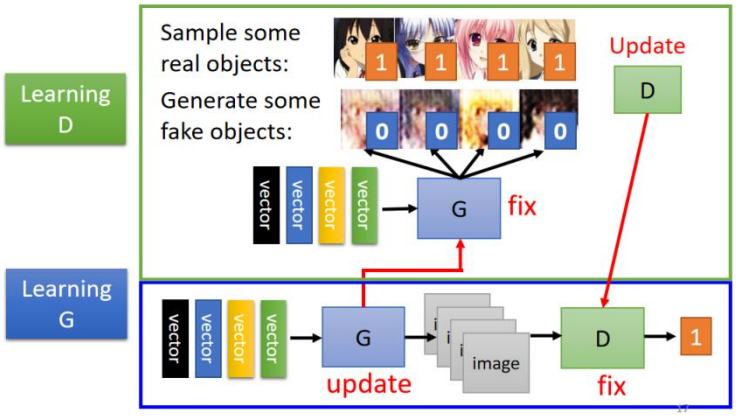



图 2 GAN 训练过程详解

## 3.2 条件生成对抗网络（CGAN）

**核心思想：**在输入时除随机噪音外，**额外给 GAN 一个类别**来指定生成图片类别，训练判别器时也考虑类别。

**训练过程：**

- 训练判别器：
  ① 从数据集中取样本（含图片和对应类别）、从分布中取噪音向量
  ② 生成器根据噪音向量和类别生成数据
  ③ 再从数据集中取样本（不含对应类别）
  ④ 计算对真实样本及对应类别、生成样本及对应类别、真实样本对应错误类别的判别结果和损失函数并更新参数
- 训练生成器：
  ① 从分布中取噪音向量、从数据集中取类别
  ② 生成器生成数据送判别器判别
  ③ 计算损失函数并更新生成器参数

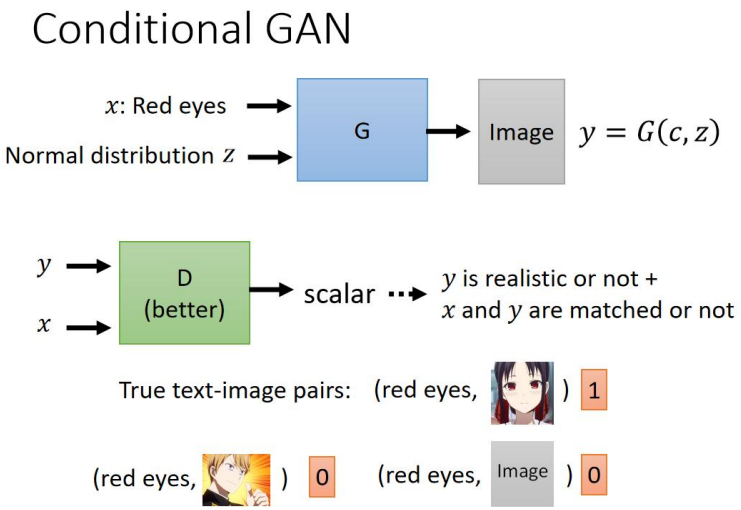



图 3 条件生成网络的核心优化

## 3.3 DCGAN、WGAN 和 WGAN-GP

### 3.3.1 DCGAN 模型

**DCGAN**，是一种**将卷积神经网络（CNN）结构引入到 GAN 框架中**的改进模型。相较于传统 GAN，DCGAN 通过使用卷积层与转置卷积层代替全连接层，增强图像生成的表现力和稳定性。

**设计要点如下：**

1、将卷积网络中的池化层用相应步长的卷积层代替；

2、在生成模型和判别模型中都使用了批归一化层；

3、去掉了网络中的全连接层；

4、在生成模型中采用 Re LU 激活函数；

5、在判别模型中采用 Leaky Re LU 激活函数。

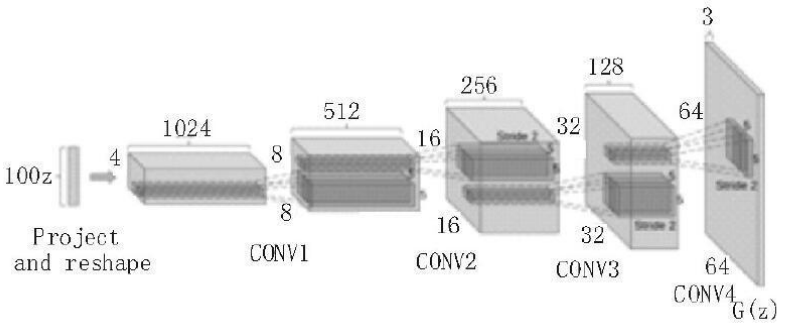



图 4DCGAN 生成器结构

深度卷积生成网络相对于一般的生成对抗网络而言具有更强大的生成能力，同时训练起来更加稳定、容易，而且生成样本更加多样化。但缺点是生成图像分辨率比较低（64*64）。

### 3.3.2 WGAN 和 WGAN-GP 模型

（1） **WGAN**，主要是从损失函数的角度对 GAN 进行了改进，通过**引入 Wasserstein 距离作为衡量真实分布与生成分布之间差异的指标**，有效缓解了传统 GAN 中的梯度消失问题，显著提升了模型的收敛性与训练稳定性。

同时也解决了模式崩溃问题（collapse mode）*（生成器倾向于生成一些有把握但相似的图片，而不敢轻易地尝试去生成没把握的新图片，从而缺乏多样性的情况）*，使得生成器生成结果更加多样。

**设计要点如下：**

1、判别器（在 WGAN 中通常称为 Critic）不使用 Sigmoid 输出，而是直接输出一个实值分数

2、训练时采用权重剪切以保证 Lipschitz 连续性

（2） **WGAN-GP**，是 **WGAN 的改进版本，**通过在判别器损失函数中添加梯度惩罚项，进一步解决了权重剪切带来的训练不稳定问题，使模型训练更加稳健，尤其适用于生成高分辨率图像。

**设计要点如下：**

1、使用 Critic 输出实值分数，

2、取消权重剪切，加入梯度惩罚项来稳定训练

$$L = \underbrace{\mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}\left[D(\tilde{\boldsymbol{x}})\right] - \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathbb{P}_r}\left[D(\boldsymbol{x})\right]}_{\text{Original critic loss}} + \underbrace{\lambda \mathop{\mathbb{E}}_{\hat{\boldsymbol{x}}\sim\mathbb{P}_{\hat{\boldsymbol{x}}}}\left[(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\|_2 - 1)^2\right]}_{\text{Our gradient penalty}}.$$

图 5 WGAN-GP 的梯度惩罚项

## 3.4 参考资料

视频资料：[李宏毅对抗生成网络(GAN)国语教程(2018)_哔哩哔哩_bilibili]

参考资料：

[1]【学习笔记-李宏毅】GAN（生成对抗网络）全系列（一）_生成对抗网络 李宏毅 ppt-CSDN 博客

[2]GAN 的发展系列一（CGAN、DCGAN、WGAN、WGAN-GP、LSGAN、BEGAN）-腾讯云开发者社区-腾讯云

# 4.任务二 采用 DCGAN 生成 1000 张不同的二次元漫画脸图像

## 4.1 数据集和环境准备

### 4.1.1 数据集和参考

本次训练使用数据集为动漫人脸图像数据集 Anime Face Dataset，数据集的图像统一尺寸为 64x64 分辨率。

下载链接：

https://openi.pcl.ac.cn/develop-productivity/01/raw/branch/master/anime-faces.zip

可以直接在 kaggle 中使用：Anime Faces

本次实验参考开源项目：toobie-2b/AnimeGAN： 实现 DCGAN 以生成动漫角色面孔。

### 4.1.2 运行环境

Python 版本：Python ==3.8

深度学习框架：PyTorch == 1.13

依赖库：torch, torchvision,numpy, PIL, matplotlib,tqdm

## 4.2 DCGAN 的网络架构和原理

构建的 DCGAN 主要由两个网络组成，是无条件生成：

- 生成器（Generator）：将随机噪声向量映射为逼真的图像。
- 判别器（Discriminator）：判断一张图像是真实的（来自数据集）还是伪造的（由生成器生成）。

### 4.2.1 Generator 网络结构分析

使用 DCGAN 上采样结构定义 Generator 模型：

**（1）基础结构如下，**

- nn.ConvTranspose2d(...)　# 反卷积：将低维向量逐步放大为图像
- nn.BatchNorm2d(...)　# 批归一化：稳定训练
- nn.ReLU()　# 激活函数：非线性增强

首先输入一个 100 维的随机向量，表示噪声。

接着逐步通过反卷积层放大为图像大小，通道数逐渐减小（1024→512→256→128→3），空间尺寸逐渐增大。

最后一层使用 Tanh()：将输出归一化到 [-1, 1]，与图像预处理对应。

```
class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()

        self.model = nn.Sequential(
            nn.ConvTranspose2d(100, 1024, kernel_size=4, stride=1, padding=0),
            nn.BatchNorm2d(1024),
            nn.ReLU(),
            nn.ConvTranspose2d(1024, 512, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(512),
            nn.ReLU(),
            nn.ConvTranspose2d(512, 256, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(),
            nn.ConvTranspose2d(256, 128, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.ConvTranspose2d(128, 3, kernel_size=4, stride=2, padding=1),
            nn.Tanh(),
        )

    def forward(self, x):
        return self.model(x)
```

**（2）Generator 的损失函数，**

定义一个经过对数变换的生成器损失形式，让生成器生成的图像更容易骗过

判别器。

```
def modified_g_loss(fake_output, eps=1e-6):
    loss = (fake_output + eps).log().mean()
    return loss.neg()
```

### 4.2.2 Discriminator 网络结构分析

Discriminator 使用了典型的 DCGAN 判别结构，

**（1）基础结构如下，**

- nn.Conv2d(...)　　# 正常卷积：提取特征
- nn.BatchNorm2d(...)　# 批归一化
- nn.LeakyReLU(0.2)　　# 激活函数

（ReLU 适用于生成器，LeakyReLU 适用于判别器）

首先接收大小为 (3, 64, 64)的图像，

接着逐层卷积降维，通道数不断增加（64→128→256→512→1），空间尺寸逐渐减小。

最后一层输出一个 1×1×1 的值，经过 Sigmoid 输出概率（0~1），判断图像的真伪。

```
class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()

        self.model = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(64),
            nn.LeakyReLU(0.2),
            nn.Conv2d(64, 128, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2),
            nn.Conv2d(128, 256, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(256),
            nn.LeakyReLU(0.2),
            nn.Conv2d(256, 512, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(512),
            nn.LeakyReLU(0.2),
            nn.Conv2d(512, 1, kernel_size=4, stride=2, padding=0),
            nn.Sigmoid()
        )

    def forward(self, x):
        return self.model(x)
```

## 4.3DCGAN 模型训练

### 4.3.1 权重初始化方法

DCGAN 模型权重初始化如下，

- 卷积层：mean=0, std=0.02 的正态分布
- BN 层的权重：mean=1, std=0.02，偏置为 0

```
def apply_weights(model):
    classname = model.__class__.__name__
    if classname.find('Conv') != -1:
        nn.init.normal_(model.weight.data, 0.0, 0.02)
    elif classname.find('BatchNorm') != -1:
        nn.init.normal_(model.weight.data, 1.0, 0.02)
        nn.init.constant_(model.bias.data, 0)
```

### 4.3.2 损失函数与优化器

**（1）损失函数，**

判别器损失使用二分类交叉熵损失 BCELoss；
生成器使用一个自定义损失函数，在上一节讲解过原理。

**（2）优化器，**

优化器均使用 Adam，

```
criterion = nn.BCELoss()
optimizerD = optim.Adam(netD.parameters(), lr=0.002, betas=(0.5, 0.999))
optimizerG = optim.Adam(netG.parameters(), lr=0.002, betas=(0.5, 0.999))
```

### 4.3.3 训练过程与标签平滑

DCGAN 训练过程每轮包含以下两个阶段：

**（1）训练判别器 ，**

① 首先计算真实图像损失
② 接着生成伪造图像，并计算假图像损失

③ 更新判别器参数

同时进行以下处理，提高训练效果：

- 对图像进行标签平滑 Label Smoothing，防止判别器过度自信。
- 假图像使用.detach()，断开与生成器的计算图连接，防止判别器梯度错误传播到生成器

```
for idx, (images, _) in enumerate(dataloader):
    images = images.to(device)
    optimizerD.zero_grad()
    output = netD(images).reshape(-1)

    smooth_real = round(random.uniform(0.7, 1.12), 2)
    labels = (smooth_real*torch.ones(images.shape[0])).to(device)
    lossD_real = criterion(output, labels)

    fake = netG(torch.randn(images.shape[0], 100, 1, 1).to(device))
    output = netD(fake.detach()).reshape(-1)

    smooth_fake = round(random.uniform(0.0, 0.3), 2)
    labels = (smooth_fake*torch.ones(images.shape[0])).to(device)
    lossD_fake = criterion(output, labels)

    lossD = lossD_real + lossD_fake
    lossD.backward()
    optimizerD.step()
```

**（2）训练生成器，**

④ 伪造图像送入判别器，重新评估

⑤ 使用自定义的对数损失更新生成器参数

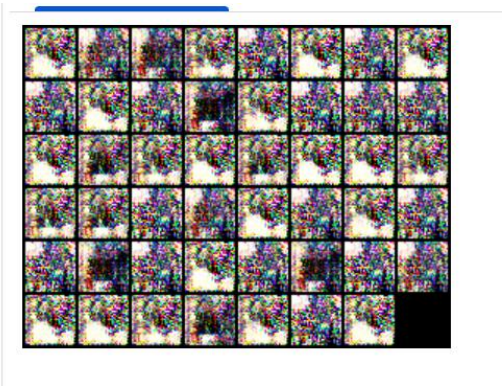
```
optimizerG.zero_grad()
output = netD(fake).reshape(-1)
lossG = modified_g_loss(output)
lossG.backward()
optimizerG.step()
```

## 4.3 实验结果分析

### （1）训练效果说明

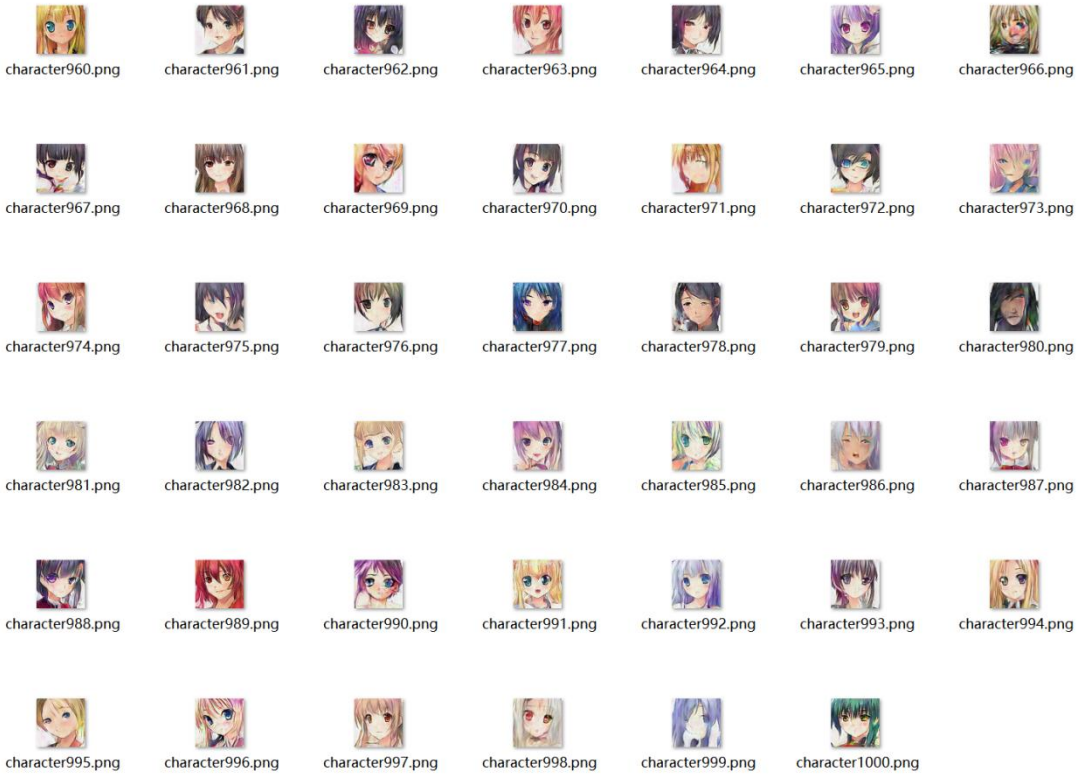
为了还原训练效果，撰写 ipynb 文件（保留更简洁的模型架构），使用数据集 Anime Face Dataset 上传到 google lab 进行了数十个 epochs 训练，效果很一般，

模型还不能完全学习到图片内容。



## （2）生成 1000 张图片效果

生成 1000 张动漫头像，只需要使用到生成器模型 Generator.pth 即可，目前使用的模型版本经过了 100 个 epochs，生成效果如下：




图片的**分辨率为 64x64**，有些模糊，但整体看来生成效果还可以，图片之间**存在区分度和多样性**，部分图片拼接痕迹明显，但**有部分图片很自然**。




## 5.代码与文件说明

代码运行详见 readme.md；附录包括在 google lab 上的训练视频，以及使用下载的模型参数生成 1000 张动漫图像的视频。文件夹/images/内含有生成的动

漫头像。

## 6.实验思考与反思

在本次理论课程学习和动漫人脸生成实践中，我学习了对抗生成网络、并理解了当中的两个重要结构生成器和判别器，在实战模型训练中我体会到了该模型的巧妙和挑战性。

理论层面，通过观看视频和查阅资料，我学习了 GAN 模型的基本原理和基础模型的训练过程，同时进一步探究了 CGAN、DCGAN、WGAN 和 WGAN - GP 等多种对抗网络模型的原理与结构。

实践任务《生成动漫人脸》中，我我选用 DCGAN 卷积对抗架构生成 1000 张动漫人脸。在学习该网络结构过程中，我明晰了反卷积上采样、批归一化等关键技术对生成质量的影响。

初期我自行搭建 DCGAN 网络模型，上传到云盘训练。最初生成图像仅为噪点，经过数十轮训练后逐渐出现轮廓，但细节仍显模糊，暴露出简单模型对数据特征学习不足的问题，同时也存在训练不稳定、生成图像模式单一的问题。接下来通过调整学习率、添加标签平滑和设计合适的损失函数来进行优化。最终，利用 100 个批次得到的模型中的生成器，来生成动漫头像最终使，输出效果还不错。但生成器的特征表达能力仍有不足，部分图像存在拼接感。

本次实验让我认识到 GAN 训练中平衡生成与判别能力的重要性，以及超参数调优的敏感性。未来还可以考虑从以下方面优化模型，一是尝试引入 WGAN-GP 梯度惩罚来提升训练稳定性，二是可以尝试增加注意力机制增强局部特征生成。