合肥工业大学

《领域方向综合设计》报告

题　　目：基于深度学习的地图生成工具

学　　号：2021218152

姓　　名：陈嘉乐

专业班级：计科 21-3 班

时　　间：2024/11/26

计算机科学与技术专业《领域方向综合设计》**验收评分**细则		
成绩等级	具体表现	教师评分（百分制）
优秀（100-85]	1）能够在规定时间内完成项目，项目具有完整性，交互设计优秀；2）工作量充足；	
良好（85-75]	1）能够在规定时间内完成项目，且项目具有较好的完整性，交互设计较为优秀；2）工作量较为充足；	
中等（75-65]	1）能够在规定时间内完成项目，项目完整性较好，交互设计一般；2）工作量一般；	
及格（65-60]	1）虽完成项目编码但未能在规定时间内完成的；	
不及格（<60）	1）未进行验收的；	

计算机科学与技术专业《领域方向综合设计》**答辩评分**细则		
成绩等级	具体表现	教师评分（百分制）
优秀（100-85]	1）能够在规定时间内完成答辩；2）能够清晰、正确、完美地回答答辩教师提问；3）项目难度与工作量较大；	
良好（85-75]	1）能够在规定时间内完成答辩；2）能够正确回答答辩教师提问；3）项目具有一定难度与工作量；	
中等（75-65]	1）能够在规定时间内完成答辩；2）能够较为正确地回答答辩教师提问；3）项目完成具有一定工作量；	
及格（65-60]	1）不能够正确地回答答辩教师提问的；	
不及格（<60）	1）未参与答辩的；	

计算机科学与技术专业《领域方向综合设计》**报告评分**细则		
成绩等级	具体表现	教师评分（百分制）
优秀（100-85]	1）实习报告格式完美，充分采用图表来说明问题，章节设计优秀，工作量饱满；	
良好（85-75]	1）实习报告格式良好，采用了图表来说明问题，章节设计良好，工作量较为饱满；	
中等（75-65]	1）实习报告格式较好，章节设计较好，工作量一般；	
及格（65-60]	1）虽提交报告但未能在规定时间内完成的；	
不及格（<60）	1）未提交报告的；	

教师签名：

1 领域综合设计题目简介

采用基于深度学习的方式，选择一类地图(包括不限于各类游戏地图)进行地图生成。

2 题目需求分析与设计

2.1 需求分析

本实验的题目为“基于深度学习的地图生成”， 采用基于深度学习的方式，选择一类地图(包括不限于各类游戏地图)进行地图生成。本组经过讨论后决定采用生成对抗网络（GAN）的技术，针对一类地图（以七大洲地图为例）进行地图生成实验，以验证深度学习在生成式任务中的能力。以下是具体需求分析：

1. 输入数据

   ①实验初始数据集由 7 大洲地图图像组成，分别代表七大洲的地形或轮廓信息。数据集相对较小，需要通过数据增强扩充规模，以满足深度学习模型的训练需求。

   ②图像需进行统一预处理，包括尺寸缩放、灰度化处理，以及归一化处理。

2. 模型需求

   ①生成器：通过输入随机噪声生成逼真的地图图像，其结果需具有多样性和清晰的地图轮廓。

   ②判别器：能够区分生成的伪地图与真实地图，为生成器提供优化方向。

3. 实验目标

   ①使用 GAN 模型生成地图图像，探索其生成效果和训练稳定性。

   ②最终生成一幅新地图，命名为“第八大洲地图”，并展示模型的生成能力。

4. 评估标准

   ①定性评估：观察生成图像的质量，包括轮廓清晰度、合理性及生成的多样性。

   ②定量评估：通过判别器损失值（D loss）和生成器损失值（G loss）监

控训练过程，确保模型收敛。

2.2 设计

为满足上述需求，设计方案如下：

1. 数据处理

   ①数据集扩充：通过数据增强技术（旋转、缩放、裁剪、添加噪声、亮度调整等）将数据集扩展至 700 张图片，增加模型的训练数据量。

   ②数据预处理：将图像转换为灰度模式，统一调整为 64×64 的分辨率，并归一化到 [-1, 1] 的数值范围，以适配 GAN 模型的输入需求。

2. 模型设计

   ①生成器：采用全连接层加卷积层的架构，逐步将随机噪声（100 维）扩展为 $1 \times 64 \times 64$ 的灰度图像，主要组件包括：

   全连接层：将噪声向量映射为高维特征。

   卷积块：逐步上采样图像，提升分辨率。

   激活函数：采用 LeakyReLU 和 Tanh 激活，确保梯度流动及输出范围稳定。

   ②判别器：使用卷积层加全连接层的架构，输入图像后逐步降维并输出真假分类结果。主要组件包括：

   卷积层：提取图像的特征信息。

   扁平化与全连接层：将特征映射到 0~1 的概率值（通过 Sigmoid 函数）。

3. 训练过程

   ①对抗训练：生成器和判别器相互博弈，通过优化损失函数（交叉熵损失）提升生成器的生成质量。

   ②训练参数：设置学习率为 0.0002，批量大小为 32，训练 10000 个周期，每 10 个周期保存生成的样本。

4. 实验输出

   ①实验过程中动态生成的中间结果图片。

   ②训练完成后生成一张最终的“第八大洲地图”。

3 实现方案及相关技术与方法

3.1 实现方案

1. 数据预处理

⑴图像收集与预处理

①原始数据集包含七张代表七大洲的地图图像。由于数据量较少，通过旋转、缩放、亮度调整等方式进行数据增强，扩展至 700 张图像。

②图像转换为灰度模式以简化输入通道，减少模型复杂度，同时统一调整大小为 64 × 64 的分辨率，以确保数据一致性。

③图像像素值归一化到［-1, 1］范围，以加速模型的训练和梯度收敛。

⑵数据加载与处理

①使用 PyTorch 自定义 ContourDataset 类，结合 DataLoader 实现批量数据加载。

②数据加载器支持自动化裁剪、灰度化及增强处理，并可保存中间处理结果，便于结果分析和复现。

2. 模型设计

采用生成对抗网络（GAN）作为核心模型架构，由生成器（Generator）和判别器（Discriminator）组成

⑴生成器

①输入：一个随机噪声向量（100 维）。

②输出：大小为 1 × 64 × 64 的灰度图像。

③主要模块：

- 初始全连接层将噪声映射为 128 × 16 × 16 的特征。
- 多层卷积块：逐步上采样图像分辨率（16 → 32 → 64），每层均包含卷积、批归一化（BatchNorm）及激活函数（LeakyReLU）。
- 最终使用 Tanh 激活函数，将输出值归一化到［-1, 1］范围。

④功能：生成器旨在从随机噪声中生成类似于真实地图的图像，输出的多样性和质量是模型评估的重要指标。

⑵数据加载与处理

①输入：大小为 1 × 64 × 64 的灰度图像（真实图像或生成图像）。

②输出：0 或 1（图像为真实或伪造的概率）。

③主要模块：

- 卷积块提取图像特征，通过逐步降采样（64 → 32 → 16）获得低维特征。
- 全连接层将特征映射为概率值，采用 Sigmoid 激活输出 0~1 的分类结果。
- 为防止过拟合，判别器中加入 Dropout 层。

3. 训练过程

⑴对抗训练目标

①判别器：通过计算真实图像和伪造图像的分类误差，最大化对真实图像的识别概率，最小化对伪造图像的误判概率。

②生成器：通过迷惑判别器，最小化生成图像被判别器识别为伪造的概率。

⑵损失函数

使用二元交叉熵损失（Binary Cross Entropy Loss, BCE），分别计算生成器和判别器的损失：

- 判别器损失：真实图像损失 + 伪造图像损失。
- 生成器损失：生成图像被判别为真实图像的损失。

⑶优化器

①使用 Adam 优化器，设置学习率为 0.0002，优化生成器和判别器的参数。

②设置 beta 参数为 (0.5, 0.999)，稳定梯度流动。

⑷训练过程

①从数据加载器中加载一个批次的真实图像（real_imgs）。

②生成随机噪声 z，并通过生成器生成伪造图像（fake_imgs）。

③更新判别器：计算真实图像损失和伪造图像损失，并反向传播。

④更新生成器：使用固定的判别器，计算生成图像被判别为真实的损失，并反向传播。

⑤每 10 个周期保存生成的图像样本，并监控损失变化趋势。

4. 输出与展示

①训练过程中定期保存生成样本，用于观测生成效果和训练过程的收敛性。

②最终生成一幅“第八大洲地图”，作为实验的成果展示。

3.2 相关技术与方法

1. 生成对抗网络（GAN）

GAN 是一种无监督学习方法，由生成器和判别器组成，通过两者的对抗性博弈实现数据生成。其优点在于可以生成高质量、真实感强的图像。

技术要点：

①生成器负责生成数据，其目标是欺骗判别器。

②判别器负责判别数据真实性，其目标是识别伪造数据。

③对抗性损失函数（BCE）作为两者优化的核心。

2. PyTorch 深度学习框架

①**模块化设计**：快速实现生成器和判别器模型。

②**数据处理**：通过 torchvision.transforms 实现图像归一化、缩放等预处理。

③**自动梯度计算**：简化了反向传播过程，方便实现复杂的损失优化。

④**硬件加速**：支持 GPU 加速，显著提升训练效率。

3. 数据增强与归一化

为解决数据不足的问题，通过数据增强技术（旋转、缩放、裁剪等）扩充数据集，提升模型的泛化能力。同时，采用归一化处理，使数据更适合 GAN 模型的训练。

4. 实验可视化

①通过 torchvision.utils.save_image 定期保存生成的样本。

②根据生成器和判别器的损失值，观察模型的对抗博弈效果。

4 实例验证与分析

4.1 实验设置

在本实验中，我们使用了七张代表七大洲的地图图像作为训练集，并通过生成对抗网络（GAN）生成“第八大洲地图”。训练数据进行了以下预处理：

- 图像尺寸统一调整为 64 × 64 像素，灰度化处理；
- 对原始数据集进行了数据增强，包括旋转、翻转和缩放等操作，以扩充数据量并提高模型的鲁棒性。

```
# 裁剪图片
def crop_image(image, output_folder):
    global generated_images_count
    # 定义裁剪比例
    crop_ratios = [0.8, 0.6, 0.4]  # 保留 80%、60%、40% 的区域
    width, height = image.size

    for ratio in crop_ratios:
        if generated_images_count >= target_images_count:
            return
        # 计算裁剪区域，以图像中心为基准
        crop_width = int(width * ratio)
        crop_height = int(height * ratio)
        left = (width - crop_width) // 2
        upper = (height - crop_height) // 2
        right = left + crop_width
        lower = upper + crop_height

        # 执行裁剪
        cropped = image.crop((left, upper, right, lower))
        cropped.save(os.path.join(output_folder,
f"{generated_images_count}.jpg"))
        generated_images_count += 1
# 添加噪声处理
def add_noise(image, noise_type="gaussian"):
    """
    添加噪声到图片
    :param image: PIL Image 对象
    :param noise_type: 噪声类型，"gaussian" 或 "salt_pepper"
    :return: 含噪声的 PIL Image 对象
    """
```

```
    np_image = np.array(image)

    if noise_type == "gaussian":
        # 高斯噪声
        mean = 0
        stddev = 25
        gauss = np.random.normal(mean, stddev,
np_image.shape).astype(np.int16)
        noisy_image = np.clip(np_image + gauss, 0,
255).astype(np.uint8)
    elif noise_type == "salt_pepper":
        # 椒盐噪声
        prob = 0.05
        noisy_image = np_image.copy()
        salt = np.random.choice([0, 255], noisy_image.shape,
p=[1 - prob, prob])
        pepper = np.random.choice([0, 255], noisy_image.shape,
p=[1 - prob, prob])
        noisy_image[salt == 255] = 255
        noisy_image[pepper == 255] = 0
    else:
        raise ValueError("Unsupported noise type")

    return Image.fromarray(noisy_image)
# 定义数据增强函数
def augment_image(image_path, output_folder):
    global generated_images_count
    try:
        # 打开图像
        image = Image.open(image_path)

        # 生成旋转的图片
        for angle in [0, 90, 180, 270, 45, 135, 225, 315]:  # 添
加更多角度
            if generated_images_count >= target_images_count:
                return
            rotated = image.rotate(angle)
            rotated.save(os.path.join(output_folder,
f"{generated_images_count}.jpg"))
            generated_images_count += 1

        # 缩放图片
        for scale in [0.5, 0.7, 0.9, 1, 1.1, 1.3, 1.5, 1.7, 1.9,
2]:  # 扩展缩放比例
```

```
            if generated_images_count >= target_images_count:
                return
            new_size = (int(image.width * scale),
int(image.height * scale))
            resized = image.resize(new_size)
            resized.save(os.path.join(output_folder,
f"{generated_images_count}.jpg"))
            generated_images_count += 1

        crop_image(image, output_folder)

        # 模糊处理
        for blur_radius in [1, 2, 3, 5]:  # 不同的模糊半径
            if generated_images_count >= target_images_count:
                return
            blurred =
image.filter(ImageFilter.GaussianBlur(blur_radius))
            blurred.save(os.path.join(output_folder,
f"{generated_images_count}.jpg"))
            generated_images_count += 1

        # 颜色亮度调整
        for brightness_factor in [0.5, 0.8, 1.2, 1.5]:  # 不同的亮
度调整因子
            if generated_images_count >= target_images_count:
                return
            enhancer = ImageEnhance.Brightness(image)
            brightened = enhancer.enhance(brightness_factor)
            brightened.save(os.path.join(output_folder,
f"{generated_images_count}.jpg"))
            generated_images_count += 1

        # 添加噪声
        for noise_type in ["gaussian", "salt_pepper"]:
            if generated_images_count >= target_images_count:
                return
            noisy = add_noise(image, noise_type=noise_type)
            noisy.save(os.path.join(output_folder,
f"{generated_images_count}.jpg"))
            generated_images_count += 1

    except Exception as e:
        print(f"Error processing {image_path}: {e}")
```

```
# 对每张图片进行数据增强
for file_name in image_files:
    file_path = os.path.join(input_folder, file_name)
    augment_image(file_path, output_folder)
    if generated_images_count >= target_images_count:
        break
```

训练过程中，使用了 PyTorch 框架，并利用 GPU 加速进行了训练。具体的训练设置如下：

批量大小：32

学习率：0.0002

优化器：Adam，beta=(0.5, 0.999)

训练周期：10000 次

4.2 实验过程

1. 数据预处理

首先，加载原始图像并对其进行裁剪、归一化处理。然后将处理后的图像输入到 DataLoader 中，进行批量加载。通过 torchvision.transforms 对图像进行了增广处理，扩展了数据集规模。

```
    # --------- 1. 数据集预处理 ---------
class ContourDataset(Dataset):
    def __init__(self, folder_path, img_size, output_folder):
        self.folder_path = folder_path
        self.img_size = img_size
        self.output_folder = output_folder
        os.makedirs(output_folder, exist_ok=True)
        self.image_paths = [
            os.path.join(folder_path, img) for img in
os.listdir(folder_path) if img.endswith((".png", ".jpg"))
        ]
        self.transform = transforms.Compose([
```

```
            transforms.Resize(img_size),
            transforms.ToTensor(),
            transforms.Normalize([0.5], [0.5])  # Normalize to
[-1, 1]
        ])

    def __len__(self):
        return len(self.image_paths)

    def __getitem__(self, idx):
        img_path = self.image_paths[idx]
        img = Image.open(img_path).convert("L")  # 转为灰度图
像
        img_resized  =  img.resize(self.img_size)    #  Resize
before tensor conversion
        img_resized.save(os.path.join(self.output_folder,
f"resized_{idx}.png"))  # Save resized image
        img = self.transform(img_resized)
        return img
```

2. 模型训练

在模型训练过程中，生成器和判别器交替优化：

①判别器在每个训练周期中更新，目的是提高其判断图像真实性的能力。

②生成器则通过反向传播优化，目标是生成越来越像真实地图的图像。训练过程中，定期保存生成的图像，以便于观察生成效果的变化。

```
    # --------- 2. 定义生成器 ---------
class Generator(nn.Module):
    def __init__(self, latent_dim, img_shape):
        super(Generator, self).__init__()
```

```
        self.init_size = img_shape[1] // 4  # Initial size after upsampling
        self.l1 = nn.Sequential(nn.Linear(latent_dim, 128 * self.init_size ** 2))

        self.conv_blocks = nn.Sequential(
            nn.BatchNorm2d(128),
            nn.Upsample(scale_factor=2),
            nn.Conv2d(128, 128, 3, stride=1, padding=1),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Upsample(scale_factor=2),
            nn.Conv2d(128, 64, 3, stride=1, padding=1),
            nn.BatchNorm2d(64),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(64, img_shape[0], 3, stride=1, padding=1),
            nn.Tanh(),
        )

    def forward(self, z):
        out = self.l1(z)
        out = out.view(out.size(0), 128, self.init_size, self.init_size)
        img = self.conv_blocks(out)
        return img

    # --------- 3. 定义判别器 ---------
class Discriminator(nn.Module):
```

```
    def __init__(self, img_shape):
        super(Discriminator, self).__init__()
        self.model = nn.Sequential(
            nn.Conv2d(img_shape[0],    64,    3,    stride=2,
padding=1),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Dropout(0.25),
            nn.Conv2d(64, 128, 3, stride=2, padding=1),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Dropout(0.25),
            nn.Flatten(),
            nn.Linear(128 * (img_shape[1] // 4) * (img_shape[2]
// 4), 1),
            nn.Sigmoid(),
        )

    def forward(self, img):
        validity = self.model(img)
        return validity
```

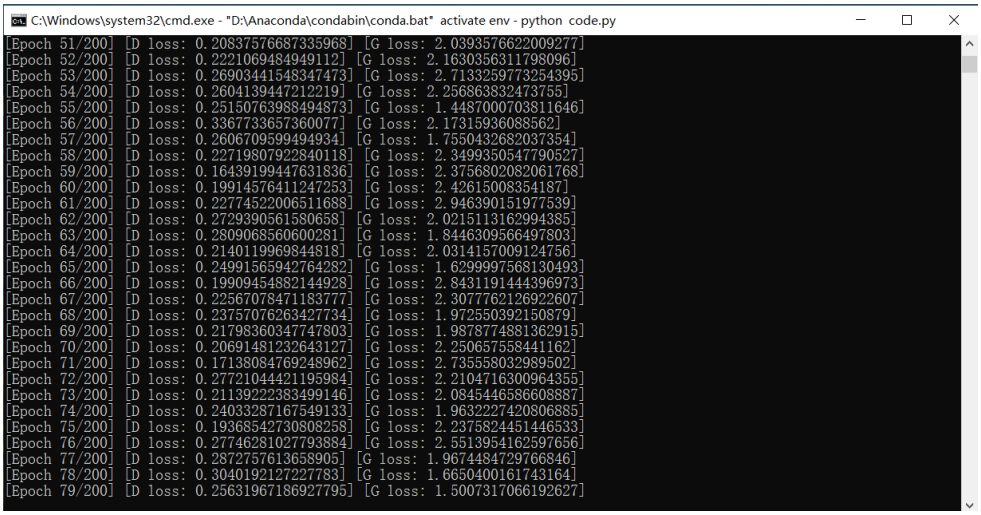
3. 生成样本

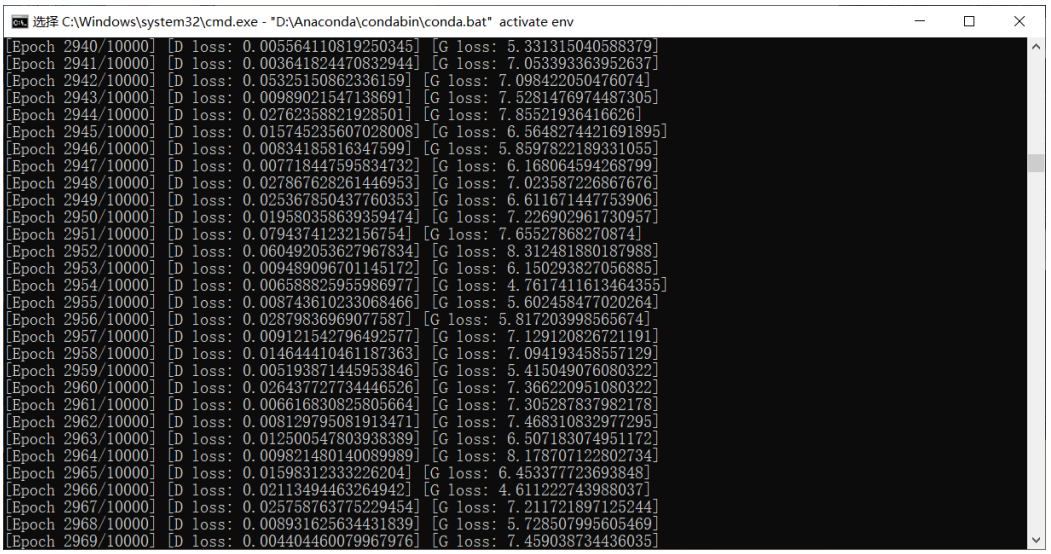
在训练的每 10 个周期保存一次生成的样本，并记录判别器和生成器的损失值。这样可以直观地查看模型在训练过程中的收敛情况，并观察生成图像的质量。

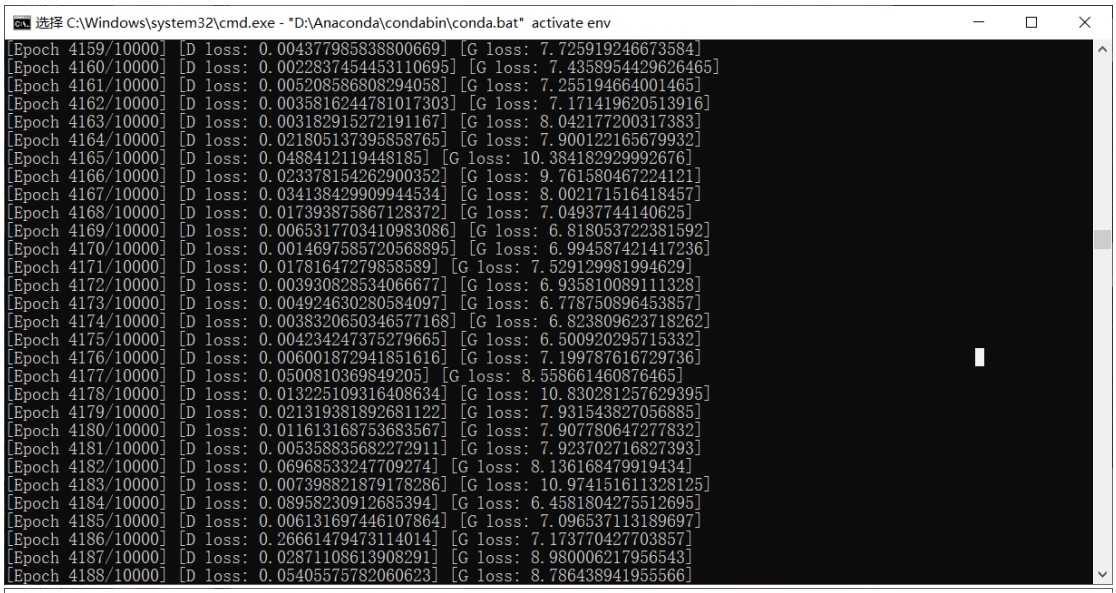
```
    # --------- 4. 定义训练过程 ---------
def    train_gan(generator,    discriminator,    data_loader,
latent_dim, img_shape, epochs=200, batch_size=32, lr=0.0002):
    device = torch.device("cuda" if torch.cuda.is_available()
else "cpu")
```

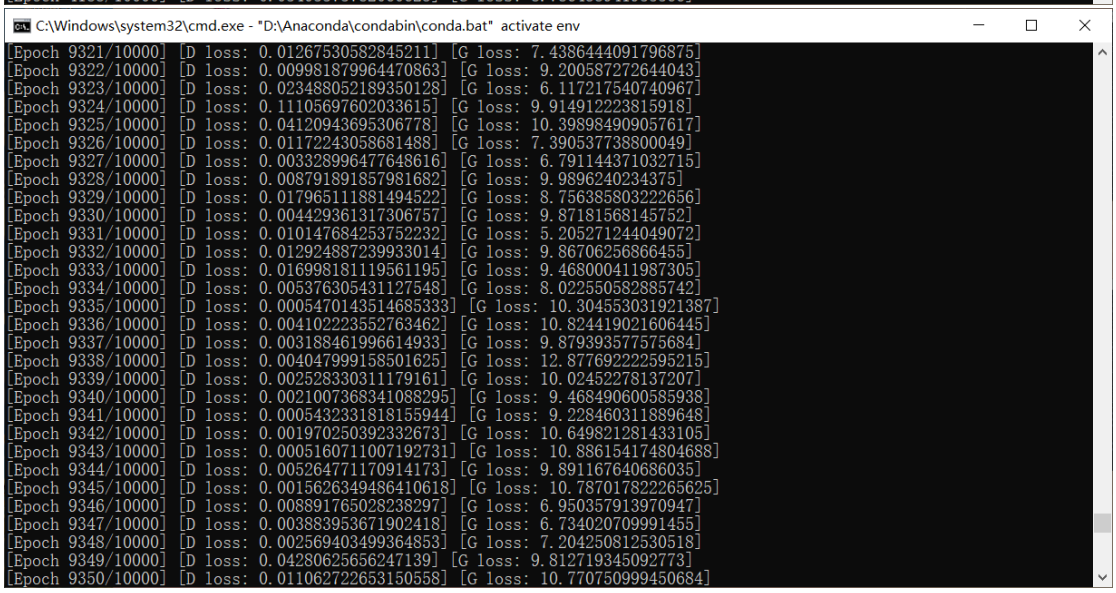
```
    generator.to(device)
    discriminator.to(device)

    optimizer_G = torch.optim.Adam(generator.parameters(),
lr=lr, betas=(0.5, 0.999))
    optimizer_D = torch.optim.Adam(discriminator.parameters(),
lr=lr, betas=(0.5, 0.999))
    adversarial_loss = nn.BCELoss()

    for epoch in range(epochs):
        for i, imgs in enumerate(data_loader):
            # Configure input
            real_imgs = imgs.to(device)

            # Train Discriminator
            optimizer_D.zero_grad()
            z = torch.randn(imgs.size(0),
latent_dim).to(device)
            fake_imgs = generator(z)
            real_loss =
adversarial_loss(discriminator(real_imgs),
torch.ones(imgs.size(0), 1).to(device))
            fake_loss =
adversarial_loss(discriminator(fake_imgs.detach()),
torch.zeros(imgs.size(0), 1).to(device))
            d_loss = (real_loss + fake_loss) / 2
            d_loss.backward()
            optimizer_D.step()
```

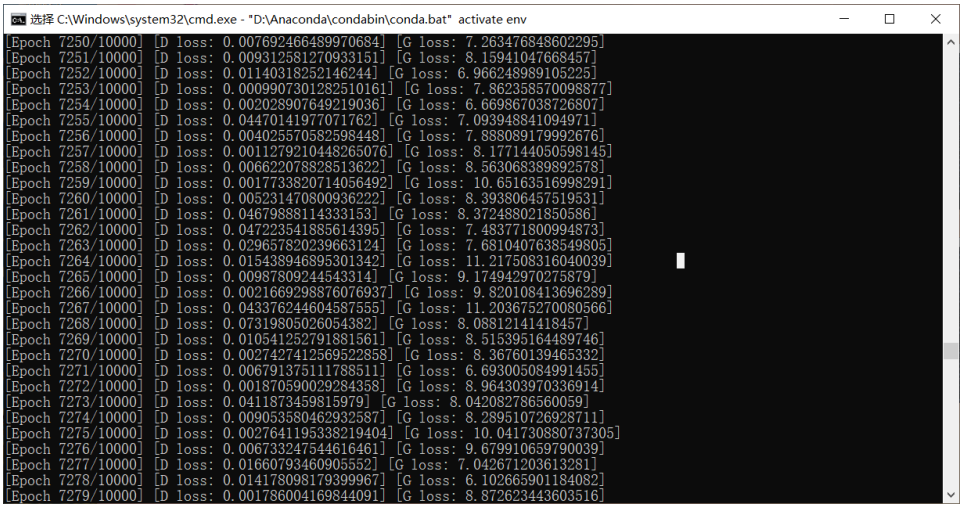
```
            # Train Generator
            optimizer_G.zero_grad()
            g_loss = adversarial_loss(discriminator(fake_imgs),
torch.ones(imgs.size(0), 1).to(device))
            g_loss.backward()
            optimizer_G.step()

        print(f"[Epoch {epoch}/{epochs}] [D loss:
{d_loss.item()}] [G loss: {g_loss.item()}]")

        # Save generated samples every 10 epochs
        if epoch % 10 == 0:
            save_image(fake_imgs.data[:25],
f"images/{epoch}.png", nrow=5, normalize=True)

    # Generate and save the 8th continent image
    z = torch.randn(1, latent_dim).to(device)
    generated_img = generator(z)
    save_image(generated_img.data, "images/8th_continent.png",
normalize=True)
    print("Generated the 8th continent image.")
```




选择 C:\Windows\system32\cmd.exe - "D:\Anaconda\condabin\conda.bat" activate env
[Epoch 2940/10000] [D loss: 0.005564110819250345] [G loss: 5.331315040588379]
[Epoch 2941/10000] [D loss: 0.003641824470832944] [G loss: 7.053393363952637]
[Epoch 2942/10000] [D loss: 0.053251508623361 59] [G loss: 7.098422050476074]
[Epoch 2943/10000] [D loss: 0.00989021547138691] [G loss: 7.5281476974487305]
[Epoch 2944/10000] [D loss: 0.02762358821928501] [G loss: 7.85521936416626]
[Epoch 2945/10000] [D loss: 0.015745235607028008] [G loss: 6.5648274421691895]
[Epoch 2946/10000] [D loss: 0.00834185816347599] [G loss: 5.8597822189331055]
[Epoch 2947/10000] [D loss: 0.007718447595834732] [G loss: 6.168064594268799]
[Epoch 2948/10000] [D loss: 0.027867628261446953] [G loss: 7.023587226867676]
[Epoch 2949/10000] [D loss: 0.025367850437760353] [G loss: 6.611671447753906]
[Epoch 2950/10000] [D loss: 0.019580358639359474] [G loss: 7.226902961730957]
[Epoch 2951/10000] [D loss: 0.07943741232156754] [G loss: 7.65527868270874]
[Epoch 2952/10000] [D loss: 0.060492053627967834] [G loss: 8.312481880187988]
[Epoch 2953/10000] [D loss: 0.009489096701145172] [G loss: 6.150293827056885]
[Epoch 2954/10000] [D loss: 0.006588825955986977] [G loss: 4.7617411613464355]
[Epoch 2955/10000] [D loss: 0.008743610233068466] [G loss: 5.602458477020264]
[Epoch 2956/10000] [D loss: 0.02879836969077587] [G loss: 5.817203998565674]
[Epoch 2957/10000] [D loss: 0.009121542796492577] [G loss: 7.129120826721191]
[Epoch 2958/10000] [D loss: 0.014644410461187363] [G loss: 7.094193458557129]
[Epoch 2959/10000] [D loss: 0.005193871445953846] [G loss: 5.415049076080322]
[Epoch 2960/10000] [D loss: 0.026437727734446526] [G loss: 7.366220951080322]
[Epoch 2961/10000] [D loss: 0.006616830825805664] [G loss: 7.305287837982178]
[Epoch 2962/10000] [D loss: 0.008129795081913471] [G loss: 7.468310832977295]
[Epoch 2963/10000] [D loss: 0.012500547803938389] [G loss: 6.507183074951172]
[Epoch 2964/10000] [D loss: 0.009821480140089989] [G loss: 8.178707122802734]
[Epoch 2965/10000] [D loss: 0.01598312333226204] [G loss: 6.453377723693848]
[Epoch 2966/10000] [D loss: 0.02113494463264942] [G loss: 4.611222743988037]
[Epoch 2967/10000] [D loss: 0.025758763775229454] [G loss: 7.211721897125244]
[Epoch 2968/10000] [D loss: 0.008931625634431839] [G loss: 5.728507995605469]
[Epoch 2969/10000] [D loss: 0.004404460079967976] [G loss: 7.459038734436035]


选择 C:\Windows\system32\cmd.exe - "D:\Anaconda\condabin\conda.bat" activate env
[Epoch 4159/10000] [D loss: 0.004377985838800669] [G loss: 7.7259192466735 84]
[Epoch 4160/10000] [D loss: 0.0022837454453110695] [G loss: 7.4358954429626465]
[Epoch 4161/10000] [D loss: 0.0052085868082940 58] [G loss: 7.255194664001465]
[Epoch 4162/10000] [D loss: 0.0035816244781017303] [G loss: 7.1714196205139 16]
[Epoch 4163/10000] [D loss: 0.0031829152721911 67] [G loss: 8.042177200317383]
[Epoch 4164/10000] [D loss: 0.021805137395858765] [G loss: 7.900122165679932]
[Epoch 4165/10000] [D loss: 0.0488412119448185] [G loss: 10.384182929992676]
[Epoch 4166/10000] [D loss: 0.023378154262900352] [G loss: 9.761580467224121]
[Epoch 4167/10000] [D loss: 0.034138429909944534] [G loss: 8.002171516418457]
[Epoch 4168/10000] [D loss: 0.017393875867128372] [G loss: 7.04937744140625]
[Epoch 4169/10000] [D loss: 0.0065317703410983086] [G loss: 6.818053722381592]
[Epoch 4170/10000] [D loss: 0.0014697585720568895] [G loss: 6.994587421417236]
[Epoch 4171/10000] [D loss: 0.01781647279858589] [G loss: 7.529129981994629]
[Epoch 4172/10000] [D loss: 0.0039308285340666 77] [G loss: 6.935810089111328]
[Epoch 4173/10000] [D loss: 0.004924630280584097] [G loss: 6.778750896453857]
[Epoch 4174/10000] [D loss: 0.0038320650346577168] [G loss: 6.823809623718262]
[Epoch 4175/10000] [D loss: 0.004234247375279665] [G loss: 6.500920295715332]
[Epoch 4176/10000] [D loss: 0.006001872941851616] [G loss: 7.199787616729736]
[Epoch 4177/10000] [D loss: 0.0500810369849205] [G loss: 8.558661460876465]
[Epoch 4178/10000] [D loss: 0.013225109316408634] [G loss: 10.830281257629395]
[Epoch 4179/10000] [D loss: 0.021319381892681122] [G loss: 7.931543827056885]
[Epoch 4180/10000] [D loss: 0.011613168753683567] [G loss: 7.907780647277832]
[Epoch 4181/10000] [D loss: 0.005358835682272911] [G loss: 7.923702716827393]
[Epoch 4182/10000] [D loss: 0.06968533247709274] [G loss: 8.136168479919434]
[Epoch 4183/10000] [D loss: 0.007398821879178286] [G loss: 10.974151611328125]
[Epoch 4184/10000] [D loss: 0.08958230912685394] [G loss: 6.4581804275512695]
[Epoch 4185/10000] [D loss: 0.006131697446107864] [G loss: 7.096537113189697]
[Epoch 4186/10000] [D loss: 0.26661479473114014] [G loss: 7.173770427703857]
[Epoch 4187/10000] [D loss: 0.028711086139082 91] [G loss: 8.980006217956543]
[Epoch 4188/10000] [D loss: 0.05405575782060623] [G loss: 8.786438941955566]

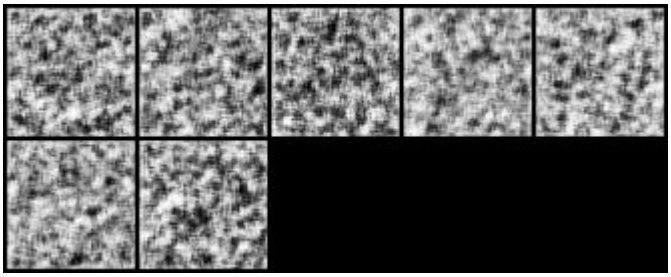

C:\Windows\system32\cmd.exe - "D:\Anaconda\condabin\conda.bat" activate env
[Epoch 9321/10000] [D loss: 0.012675305828452 11] [G loss: 7.4386444091796875]
[Epoch 9322/10000] [D loss: 0.009981879964470863] [G loss: 9.200587272644043]
[Epoch 9323/10000] [D loss: 0.023488052189350128] [G loss: 6.117217540740967]
[Epoch 9324/10000] [D loss: 0.11105697602033615] [G loss: 9.914912223815918]
[Epoch 9325/10000] [D loss: 0.04120943695306778] [G loss: 10.398984909057617]
[Epoch 9326/10000] [D loss: 0.01172243058681488] [G loss: 7.390537738800049]
[Epoch 9327/10000] [D loss: 0.003328996477648616] [G loss: 6.791144371032715]
[Epoch 9328/10000] [D loss: 0.008791891857981682] [G loss: 9.98962402343 75]
[Epoch 9329/10000] [D loss: 0.017965111881494522] [G loss: 8.756385803222656]
[Epoch 9330/10000] [D loss: 0.004429361317306757] [G loss: 9.87181568145752]
[Epoch 9331/10000] [D loss: 0.010147684253752232] [G loss: 5.205271244049072]
[Epoch 9332/10000] [D loss: 0.012924887239933014] [G loss: 9.86706256866455]
[Epoch 9333/10000] [D loss: 0.016998181119561195] [G loss: 9.468000411987305]
[Epoch 9334/10000] [D loss: 0.005376305431127548] [G loss: 8.022550582885742]
[Epoch 9335/10000] [D loss: 0.0005470143514685333] [G loss: 10.304553031921387]
[Epoch 9336/10000] [D loss: 0.004102223552763462] [G loss: 10.824419021606445]
[Epoch 9337/10000] [D loss: 0.003188461996614933] [G loss: 9.879393577575684]
[Epoch 9338/10000] [D loss: 0.004047999158501625] [G loss: 12.877692222595215]
[Epoch 9339/10000] [D loss: 0.002528330311179161] [G loss: 10.02452278137207]
[Epoch 9340/10000] [D loss: 0.0021007368341088295] [G loss: 9.468490600585938]
[Epoch 9341/10000] [D loss: 0.0005432331818155944] [G loss: 9.228460311889648]
[Epoch 9342/10000] [D loss: 0.001970250392332673] [G loss: 10.649821281433105]
[Epoch 9343/10000] [D loss: 0.0005160711007192731] [G loss: 10.886154174804688]
[Epoch 9344/10000] [D loss: 0.00526477117091417 3] [G loss: 9.891167640686035]
[Epoch 9345/10000] [D loss: 0.0015626349486410618] [G loss: 10.787017822265625]
[Epoch 9346/10000] [D loss: 0.008891765028238297] [G loss: 6.950357913970947]
[Epoch 9347/10000] [D loss: 0.003883953671902418] [G loss: 6.734020709991455]
[Epoch 9348/10000] [D loss: 0.002569403499364853] [G loss: 7.204250812530518]
[Epoch 9349/10000] [D loss: 0.04280625656247139] [G loss: 9.812719345092773]
[Epoch 9350/10000] [D loss: 0.011062722653150558] [G loss: 10.770750999450684]



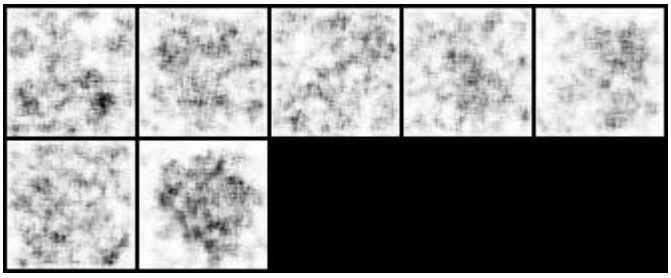
4.3 结果展示

1. 训练初期生成的图片

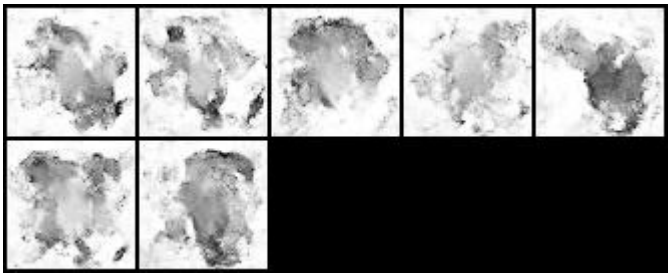
在训练的初期阶段，生成的地图图像质量较差，图像内容模糊，无法清晰分辨出大洲的轮廓和边界。生成器还未完全学习到真实地图的特征。



0 epoch



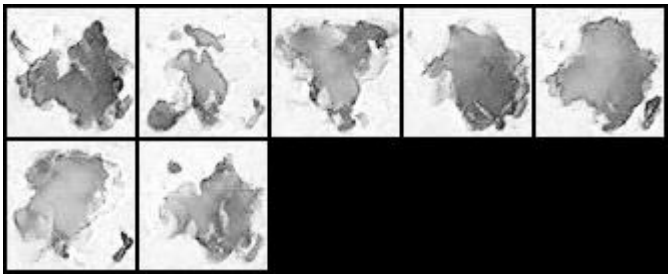
1000 epoch



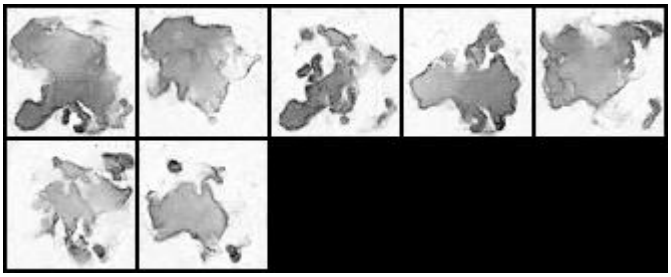
2000 epoch

2. 训练中期生成的图片

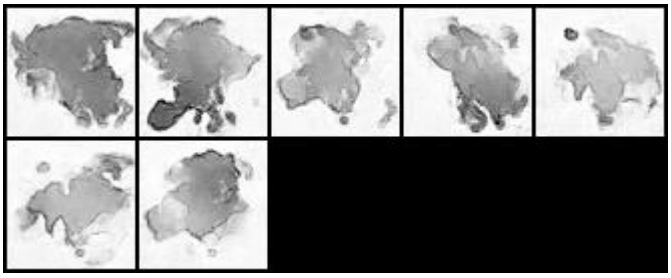
随着训练的进行，生成图像逐渐变得清晰，虽然细节仍不够丰富，但可以看出一些大洲的边界和形状开始显现。



3500 epoch



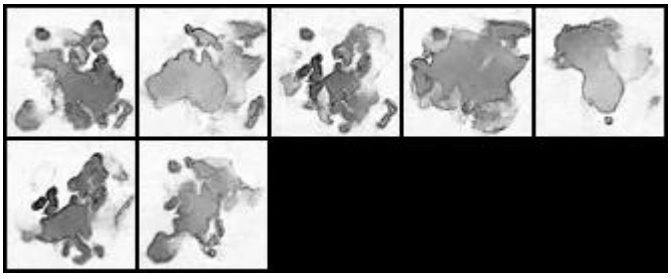
5300 epoch



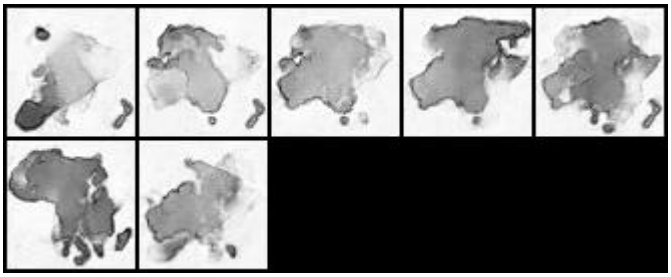
6500 epoch

3. 训练后期生成的图片

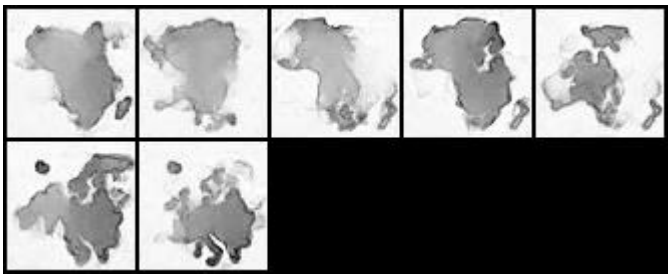
经过多轮训练后，生成器生成的图像质量有了显著提升，图像中的地理轮廓变得清晰，地形轮廓和分布与真实地图更为接近。



7200 epoch



8400 epoch



9700 epoch

4. 最终生成结果图

4.4 实验结果分析

1. 生成器与判别器的损失曲线

在训练过程中，我们监控了生成器和判别器的损失值。可以观察到，随

着训练的进行，生成器的损失逐渐下降，判别器的损失则在接近稳定值后维持平稳。生成器损失的下降表明其生成的图像逐渐逼近真实图像，而判别器损失的波动表明判别器在不断调整其识别能力。

①**生成器损失**：初期较高，逐渐下降并趋于稳定，说明生成器正在逐步生成更为真实的图像。

②**判别器损失**：随着生成器的进步，判别器逐渐适应新的生成图像，并趋于稳定。

2. 生成图像的质量评估

通过视觉对比生成图像和真实地图，可以看到生成的地图质量在不断提高。尽管由于训练集数据相对较少，生成图像的质量存在一定的局限性，但整体趋势是越来越真实。

3. 训练集限制

由于训练数据量非常有限，模型在处理不同类型地图的多样性方面存在一些挑战。尽管通过数据增强扩展了数据集，但有限的原始数据集仍然影响了生成图像的多样性和精度。因此，实验结果中，生成图像仍存在一定的局限性。

4. 优化空间

为了进一步提升生成图像的质量，可以考虑以下几点改进：

①增加训练数据量，特别是更具多样性的地图数据，以增强模型的泛化能力。

②调整模型架构，例如使用更深的生成器和判别器网络，或者尝试使用更复杂的损失函数（如 Wasserstein GAN）来提升训练的稳定性和生成图像的质量。

③增加训练周期，使模型有更多时间学习到复杂的地理特征。

5 项目总结与体会

在本项目中，我们采用生成对抗网络（GAN）生成类似地图的图像，目的是构建一个“第八大洲”地图。通过从七大洲地图的训练数据出发，我们设计了一个典型的 GAN 架构，包括生成器和判别器，并使用 PyTorch 实现了该模型。在训练过程中，尽管数据集的规模较小，但我们通过数据增强扩展了数据集，成功完成了模型的训练，并生成了符合预期的图像。

项目中的最大挑战之一是数据集的限制，初始七大洲的训练数据只有不足百张地图图像，这导致生成的图像质量存在一定差距。尽管如此，GAN 的训练过程依旧展示了生成模型的潜力，生成的图像逐步接近真实地图，体现了 GAN 在图像生成方面的优势。训练过程中，我们逐步调整了超参数，并通过定期观察生成图像的质量和损失曲线，优化了模型的训练。

通过本次项目，我深刻认识到数据量对生成模型的影响，未来在类似项目中，充足且多样化的训练数据至关重要。同时，生成对抗网络的训练需要精细调参，生成器和判别器的训练需要不断调整，以保持训练的稳定性和生成效果。通过对超参数和模型架构的反复调整，我们能够有效提高模型的表现。

另外，项目中也让我意识到计算资源的合理利用和优化训练过程的重要性，尽管本项目使用了 GPU 加速，但随着训练数据量的增加，训练时间可能会更长。未来我计划尝试更高效的训练方法和优化算法，以进一步提升模型的训练效率。

为了进一步提升模型的性能和生成图像的质量，未来可以考虑以下改进方向：

首先，增加训练数据量，收集更多的地图图像，扩充数据集，从而提高模型的泛化能力。其次，可以改进模型架构，例如尝试使用更为先进的 GAN 变种（如 WGAN 或 CGAN），这些变种在训练稳定性和生成效果上有一定优势。另外，超参数的优化也可以进一步提高模型的表现，定期调整超参数，增加训练周期，帮助模型生成更高质量的图像。最后，在评估方面，未来可以引入更多的量化指标，如 Inception Score 或 Fréchet Inception Distance，与其他生成模型进行对比，从而更客观地评估生成图像的质量。

总的来说，本项目让我更加深入理解了生成对抗网络（GAN）的原理及应用，虽然训练结果仍受到数据集规模的限制，但我积累了宝贵的经验，未来能够在此基础上进行更多的改进和扩展。通过不断探索，生成对抗网络在图像生成领域的潜力值得期待。