实验六　基于生成对抗网络(GAN)的医学影像模态转换

【实验目的】

1. 理解 GAN 基本结构与原理：包括生成器（Generator）和判别器（Discriminator）的协同工作机制。

2. 掌握图像到图像转换任务（Image-to-Image Translation）：实现 T1 加权图像向 T2 加权图像的转换。

3. 学会加载与预处理医学影像数据：使用 PyTorch 和 torchvision 处理灰度医学图像配对数据集。

4. 了解医学图像生成模型的评估方式：通过可视化结果对比、损失变化分析等方式进行定性评价。

【实验器材】

软件环境：

- 操作系统：Windows/Linux/MacOS（推荐配备 GPU）
- Python 版本：3.6 及以上
- PyTorch 框架：1.10 及以上
- 第三方库：torchvision、matplotlib、PIL (Pillow)

硬件需求：

- 推荐使用具有 NVIDIA GPU 的计算机（支持 CUDA 加速）
- 显存 ≥ 4GB（可适配 CPU 版本运行）

数据准备：

- 实验数据来自 IXI Dataset 的 T1 和 T2 加权图像（已处理为 256×256 大小的灰度图 .png 格式）
- 训练集路径：dataset/ixi2000/trainA（T1）、trainB（T2）
- 测试集路径：dataset/ixi2000/testA（T1）、testB（T2）

【实验原理】

本实验基于生成对抗网络（GAN），实现对医学图像的模态转换任务，即将磁共振成像（MRI）中的 T1 加权图像自动转换为 T2 加权图像。此类任务属于医学图像的跨模态合成，对于缺失模态补全、图像增强、辅助诊断等具有重要应用价值。

一、医学影像基础

T1 加权图像及其特点

T1 加权图像是磁共振成像（MRI）中通过调节脉冲序列参数（短重复时间 TR 和短回波时间 TE）生成的图像。其特点在于能够清晰显示解剖结构：脂肪组织呈现高信号（明亮），

水（如脑脊液）呈现低信号（黑暗）。这种对比使 T1 加权图像特别适合观察器官的形态结构，例如区分脑灰质与白质、显示肌肉与脂肪的边界。临床上常用于评估肿瘤、出血等病变的解剖位置和大小。

T2 加权图像及其特点

T2 加权图像通过长 TR 和长 TE 的参数设置，突出横向弛豫时间的差异。其特点是液体（如水肿、脑脊液）呈现高信号（明亮），而脂肪信号较低。这种特性使 T2 加权图像对病理变化（如炎症、水肿、肿瘤内坏死）极为敏感，能清晰显示组织中的水分分布异常。例如，在脑部扫描中，T2 加权图像可有效检测中风后的缺血区域或多发性硬化病变。

特性	T1加权图像	T2加权图像
物理原理	反映纵向弛豫时间（T1）	反映横向弛豫时间（T2）
组织对比	脂肪呈高信号，水呈低信号	水呈高信号，脂肪呈中等信号
典型应用	解剖结构显示（如脑灰质/白质区分）	病理检测（如水肿、肿瘤）

T1 向 T2 加权图像转换的意义

临床需求：某些患者因扫描时间限制或禁忌症（如体内金属植入物）无法完成 T2 序列扫描，通过 T1 生成 T2 图像可补充诊断信息。

数据增强：配对的多模态医学影像数据稀缺，生成模型可扩充数据集，辅助训练其他 AI 模型（如病灶分割）。

研究价值：验证生成模型能否捕捉到 T2 特有的病理特征（如液体信号变化），推动跨模态医学图像合成技术的发展。

成本与效率：减少重复扫描的需求，降低医疗成本，提升诊断流程效率。

二、GAN 基础原理

生成对抗网络（Generative Adversarial Network，简称 GAN）是一种创新的深度学习框架，其核心思想源自博弈论中的对抗竞争。该网络由两个相互对抗的神经网络组成：生成器（Generator）和判别器（Discriminator），二者通过持续的对抗训练共同进步，最终实现高质量数据的生成能力。

生成器的主要任务是接收随机噪声向量作为输入，通过复杂的神经网络结构将其转化为与真实数据相似的样本。这个网络通常采用转置卷积或全连接层的架构，其目标是学习真实数据的潜在分布特征，生成足以迷惑判别器的逼真样本。随着训练的进行，生成器会不断提升其生成能力，使输出样本越来越接近真实数据分布。

判别器则扮演着"鉴定专家"的角色，它的结构往往采用卷积神经网络或全连接网络。判别器接收来自真实数据集或生成器的样本作为输入，输出一个 0 到 1 之间的概率值，表示该样本来自真实数据分布的可能性。判别器的目标是尽可能准确地区分真实样本和生成样本，

为生成器提供明确的改进方向。

GAN 的训练过程本质上是一个极小极大博弈问题。从数学角度看，这个对抗过程可以表述为一个价值函数的优化问题：生成器试图最小化判别器的鉴别能力，而判别器则试图最大化其区分真假样本的准确率。这种对抗关系形成了 GAN 训练的核心动力，促使两个网络在竞争中不断进步。其核心训练目标是一个最小最大优化问题，目标函数如下：

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}}[\log D(x)] + \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z)))]$$

在理论层面上，当生成器产生的数据分布与真实数据分布完全一致时，系统达到纳什均衡点。此时判别器将无法区分真假样本，对任何输入都只能给出 0.5 的概率判断。这种均衡状态代表了 GAN 训练的理想目标，但在实际应用中往往难以完全实现。

GAN 的训练呈现出独特的动态特性。训练初期，生成器产生的样本质量较低，判别器可以轻松识别。随着训练的推进，生成器的能力逐步提升，判别器的判断难度也随之增加。这种动态平衡需要精心调整训练策略，通常采用交替更新的方式：先固定生成器训练判别器，再固定判别器训练生成器。

从理论基础上看，GAN 的优化过程与 JS 散度（Jensen-Shannon Divergence）密切相关。原始 GAN 框架实际上是在最小化真实分布与生成分布之间的 JS 散度。对于固定的生成器，存在一个理论上的最优判别器，其输出值反映了两个分布密度的比值关系。

尽管 GAN 具有强大的生成能力，原始框架仍存在一些固有挑战。模式崩溃（Mode Collapse）是常见问题之一，表现为生成器倾向于产生有限种类的样本，缺乏多样性。训练不稳定性也是重要挑战，可能导致梯度消失或训练震荡等问题。针对这些问题，研究者提出了诸多改进方案，如 Wasserstein GAN（WGAN）通过使用 Wasserstein 距离替代 JS 散度来提升训练稳定性。

三、条件 GAN（cGAN）与图像到图像转换

本实验采用的不是传统 GAN，而是其变体：条件生成对抗网络（Conditional GAN, cGAN）。与基本 GAN 的随机噪声输入不同，cGAN 的生成器接收某种条件输入，并生成与其对应的目标图像。

在本实验中，cGAN 的输入条件是 T1 加权 MRI 图像，输出目标是对应的 T2 加权图像，任务属于图像到图像的转换任务（Image-to-Image Translation）。与传统图像风格转换或超分辨率重建类似，这里本质上是从一类医学图像中学习对应关系，生成另一类结构相似但模态不同的图像。

四、网络结构设计

1. 生成器（G）

本实验使用一个简单的 Encoder-Decoder 结构（对称结构）作为生成器。其流程为：

- 下采样阶段（Encoder）：通过两个卷积层逐步提取图像特征并压缩空间维度；

- 上采样阶段（Decoder）：通过反卷积（转置卷积）逐步还原图像尺寸；
- 激活函数采用 ReLU 和 Tanh，输出范围标准化到 [-1, 1]，便于图像归一化。

2. 判别器（D）

判别器采用 PatchGAN 结构的简化版本，输入为 T1 图像和对应 T2 图像的拼接，输出是该对图像是否为真实配对的概率。

- 两者沿通道维拼接后输入网络；
- 网络采用多层卷积提取局部特征；
- 最后输出一个 1×1 的 Sigmoid 激活结果，作为真假判别。

五、损失函数设计

GAN 中的损失函数对模型效果起到决定性作用。本实验结合使用以下两种损失函数：

对抗损失（GAN Loss）：用于鼓励生成器生成更真实的图像，使得判别器无法区分真伪。

$$\mathcal{L}_{\text{GAN}} = \mathbb{E}[\log D(x, y)] + \mathbb{E}[\log(1 - D(x, G(x)))]$$

L1 像素级损失（L1 Loss）：用于确保生成图像与真实图像在像素层面尽可能接近，保持解剖结构一致性。相比于 MSE，L1 损失更适合图像生成任务，能更好保留边缘细节。

$$\mathcal{L}_{L1} = \|G(x) - y\|_1$$

最终的生成器损失为加权总和：

$$\mathcal{L}_G = \mathcal{L}_{\text{GAN}} + \lambda \mathcal{L}_{L1}$$

其中 λ 是权重系数，用于平衡图像真实性和像素准确度。

六、训练与推理流程

整个模型的训练流程如下：

1. **生成器先训练**：生成对应的 T2 图像；
2. **判别器随后训练**：分别判断真实 T2 和生成 T2 是否可信；
3. **交替优化**：G 和 D 不断博弈，提升各自性能；
4. **推理阶段**：固定生成器，将测试集中的 T1 图像输入，生成对应的 T2 图像，用于可视化与评估。

最终，通过对生成图、真实图及输入图的对比，可以评估模型是否学习到了模态转换中的结构与纹理特征。

【补充内容：生成对抗网络核心组件详解】

一、生成器（Generator）结构详解

1. 结构概览：

生成器是一个对称的 **Encoder-Decoder 结构**，用于将输入的 T1 图像转换为风格类似 T2

的图像。其结构如下：

- **输入**：尺寸为(1, 256, 256)的单通道 T1 图像；
- **下采样阶段（Encoder）**：
  - ✓ Conv2d(1 → 64)：提取基础边缘特征；
  - ✓ Conv2d(64 → 128)：继续压缩图像尺寸，提取更深层语义；
  - ✓ 激活函数使用 LeakyReLU，能保留更多负向特征；
- **上采样阶段（Decoder）**：
  - ✓ ConvTranspose2d(128 → 64)：恢复图像尺寸；
  - ✓ ConvTranspose2d(64 → 1)：输出恢复后的图像，使用 Tanh 进行归一化；
- **输出**：尺寸仍为(1, 256, 256)的生成 T2 图像。

2. 设计目的：

- 下采样负责提取特征，上采样用于恢复图像形态；
- 对称结构有利于保持解剖结构一致性；
- 最后输出通过 Tanh 归一化，使输出图像像素范围为[-1, 1]，配合前处理标准化保持一致。

3. 医学图像适用性：

- Encoder-Decoder 结构简单直观，适合小规模医学图像任务；
- 能够保证图像细节在一定程度上的还原，适合结构清晰的模态转换场景（如 T1→T2）；
- 若后期图像细节不清晰，可考虑升级为 U-Net 或 ResNet 结构。

二、判别器（Discriminator）结构详解

1. 结构概览：

判别器是一个小型的**二分类卷积网络**，接收一对图像（输入 T1 图和目标 T2 图），判断它们是否为“真实配对”或“伪造配对”。其结构如下：

- 输入：将 T1 图和 T2 图在通道维度拼接后，形成 (2, 256, 256)；
- 网络结构：
  - ✓ Conv2d(2 → 64) → LeakyReLU；
  - ✓ Conv2d(64 → 128) → BatchNorm → LeakyReLU；
  - ✓ Conv2d(128 → 1) → Sigmoid 输出概率图。

2. 判别方式说明：

- 本实验判别器不是输出一个全局真/假标志，而是输出一个 PatchGAN 风格的概率图（即局部判别结果）；
- 每个局部 Patch 区域是否真实由 Sigmoid 输出判断。

3. 设计目的：

- PatchGAN 结构可增强局部纹理判断能力，避免判别器只关注全局对齐；
- 保持网络简单，减少判别器过强导致生成器无法收敛的风险。

【实验内容】

本实验任务是基于 T1 加权图像生成对应的 T2 加权图像，实现**医学图像的模态转换**。整体分为数据加载、模型构建、训练过程和结果展示四个阶段。以下是详细步骤。

Step 1：构建医学图像配对数据集

在 GAN 中，我们需要成对的图像输入和目标图像（T1→T2），因此需先构建配对数据加载器。

编程任务

请完成如下 PairedMedicalDataset 类的实现（已给出完整代码，理解其构造方式）：

```
from torch.utils.data import Dataset
from torchvision import transforms
from PIL import Image
import os

class PairedMedicalDataset(Dataset):
    def __init__(self, t1_dir, t2_dir):
        self.t1_paths = sorted([os.path.join(t1_dir, f) for f in os.listdir(t1_dir) if f.endswith('.png')])
        self.t2_paths = sorted([os.path.join(t2_dir, f) for f in os.listdir(t2_dir) if f.endswith('.png')])

        self.transform = transforms.Compose([
            transforms.Resize((256, 256)),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.5], std=[0.5])
        ])

    def __len__(self):
        return min(len(self.t1_paths), len(self.t2_paths))

    def __getitem__(self, idx):
        t1_img = Image.open(self.t1_paths[idx]).convert('L')
        t2_img = Image.open(self.t2_paths[idx]).convert('L')
        return self.transform(t1_img), self.transform(t2_img)

```

运行结果

确保路径 dataset/ixi2000/trainA 和 trainB 下存在 .png 格式的医学图像。使用以下代码进行数据加载：

```
from torch.utils.data import DataLoader

dataset = PairedMedicalDataset("dataset/ixi2000/trainA", "dataset/ixi2000/trainB")
dataloader = DataLoader(dataset, batch_size=1, shuffle=True)
```

Step 2：定义生成器与判别器结构

编程任务

构建生成器（G）和判别器（D）的神经网络结构，参考下列代码：

```
import torch.nn as nn

class Generator(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.Conv2d(1, 64, 4, 2, 1), nn.LeakyReLU(0.2),
            nn.Conv2d(64, 128, 4, 2, 1), nn.BatchNorm2d(128), nn.LeakyReLU(0.2),
            nn.ConvTranspose2d(128, 64, 4, 2, 1), nn.BatchNorm2d(64), nn.ReLU(),
            nn.ConvTranspose2d(64, 1, 4, 2, 1), nn.Tanh()
        )

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

```
class Discriminator(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.Conv2d(2, 64, 4, 2, 1), nn.LeakyReLU(0.2),
            nn.Conv2d(64, 128, 4, 2, 1), nn.BatchNorm2d(128), nn.LeakyReLU(0.2),
            nn.Conv2d(128, 1, 4, 1, 1), nn.Sigmoid()
        )

    def forward(self, x1, x2):
        x = torch.cat([x1, x2], dim=1)
        return self.model(x)
```

运行提示

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
G = Generator().to(device)
D = Discriminator().to(device)
```

Step 3：定义损失函数与优化器

编程任务

使用 PyTorch 提供的损失函数构建 GAN 损失和 L1 损失：

```
import torch.nn.functional as F

criterion_GAN = nn.BCELoss()
criterion_L1 = nn.L1Loss()
optimizer_G = torch.optim.Adam(G.parameters(), lr=0.0002, betas=(0.5, 0.999))
optimizer_D = torch.optim.Adam(D.parameters(), lr=0.0002, betas=(0.5, 0.999))
```

Step 4：训练生成对抗网络

编程任务

实现训练循环：

```
epochs = 50
for epoch in range(epochs):
    for i, (t1, t2) in enumerate(dataloader):
        t1, t2 = t1.to(device), t2.to(device)

        # --- Train Generator ---
        fake_t2 = G(t1)
        pred_fake = D(t1, fake_t2)
        valid = torch.ones_like(pred_fake)
        loss_G = criterion_GAN(pred_fake, valid) + 100 * criterion_L1(fake_t2, t2)

        optimizer_G.zero_grad()
        loss_G.backward()
        optimizer_G.step()

        # --- Train Discriminator ---
        pred_real = D(t1, t2)
        loss_real = criterion_GAN(pred_real, torch.ones_like(pred_real))

        pred_fake = D(t1, fake_t2.detach())
        loss_fake = criterion_GAN(pred_fake, torch.zeros_like(pred_fake))

        loss_D = 0.5 * (loss_real + loss_fake)

        optimizer_D.zero_grad()
        loss_D.backward()
        optimizer_D.step()

    print(f"Epoch {epoch+1}/{epochs} | Loss_G: {loss_G.item():.4f} | Loss_D: {loss_D.item():.4f}")
```

训练效果预期

随着训练迭代，生成器的 L1 损失应该逐步下降，说明生成的 T2 图像更接近真实图像。判别器的损失也应保持在一个稳定区间。

【加分挑战内容（拓展任务）】

以下内容为实验挑战部分，供完成基础训练后的同学进一步探索：

Step 5：模型推理与结果可视化（不提供完整源码）

- 调用训练好的生成器 G，输入 T1 图像，输出预测的 T2 图像；
- 使用 matplotlib 显示三张图：输入的 T1 图像、生成的 T2 图像、真实的 T2 图像。

实验提示

- 使用 G.eval() 进入推理模式；
- 使用 torch.no_grad() 禁用梯度计算；
- 使用 plt.subplot() 构建多图显示。