南開大學
Nankai University

南开大学

计算机学院和密码与网络空间安全学院

《深度学习及应用》课程作业

Lab04：生成对抗网络

姓名：陆皓喆

学号：2211044

专业：信息安全

指导教师：李重仪

2025 年 6 月 18 日

目录

1 实验目的

1. 掌握 GAN 原理；
2. 学会使用 PyTorch 搭建 GAN 网络来训练 FashionMNIST 数据集。

2 实验要求

1. 老师提供的原始版本 GAN 网络结构（也可以自由调整网络）在 FashionMNIST 上的训练 loss 曲线，生成器和判别器的模型结构（print(G)、print(D)）；
2. 自定义一组随机数，生成 8 张图；
3. 针对自定义的 100 个随机数，自由挑选 5 个随机数，查看调整每个随机数时，生成图像的变化（每个随机数调整 3 次，共生成 15x8 张图），总结调整每个随机数时，生成图像发生的变化；
4. 解释不同随机数调整对生成结果的影响（重点部分）；
5. 格式不限；
6. **加分项：用卷积实现生成器和判别器**。

3 实验原理

3.1 任务概述

本次实验，我们需要完成图像生成的任务，使用 FashionMNIST 数据集进行训练。然后，我们需要使用原始版本的 GAN 网络结构或者自己的网络来进行训练，然后通过调整随机数来生成不同的图像，观察生成图像发生的变化。我们还可以使用卷积来实现我们的生成器和判别器。

3.2 数据集选择

本次实验，我们选取了 FashionMNIST 数据集来进行训练与验证。该数据集共含 **60000 张训练图像**与 **10000 张测试图像**，每张都是 28×28 像素的灰度服饰照片，类别标签覆盖 10 种常见衣物，如图3.1所示。

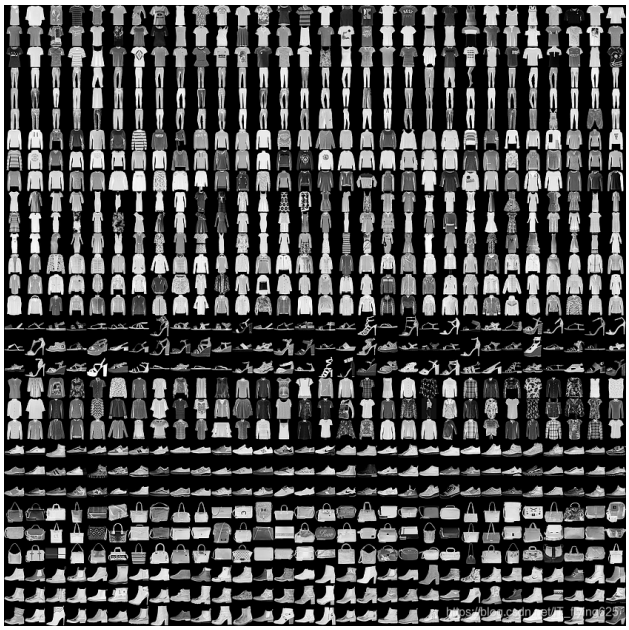

图 3.1: FashionMNIST 数据集

4 实验环境

本次实验继续使用 A100 的四卡 linux 环境的服务器上进行实验。我们还是建立我们的虚拟环境，通过：

```
python -m venv GAN
source GAN/bin/activate
pip install -r requirements.txt
```

这样就可以完成我们的本次实验的环境配置了！

5 实验过程

5.1 原始 GAN 网络的实现

我们按照老师所提供的网络结构进行代码的编写，最终得到的网络结构如下所示：

```
Discriminator(
  (model): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=784, out_features=512, bias=True)
    (2): LeakyReLU(negative_slope=0.2, inplace=True)
    (3): Linear(in_features=512, out_features=256, bias=True)
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Linear(in_features=256, out_features=1, bias=True)
    (6): Sigmoid()
  )
```

```
)
Generator(
  (model): Sequential(
    (0): Linear(in_features=100, out_features=256, bias=True)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): BatchNorm1d(256, eps=0.8, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (3): Linear(in_features=256, out_features=512, bias=True)
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): BatchNorm1d(512, eps=0.8, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (6): Linear(in_features=512, out_features=784, bias=True)
    (7): Tanh()
  )
)
```

该结构取自论文 [1]，我们主要分为两个部分来进行说明——生成器和判别器。

首先是判别器。判别器的结构是一个是全连接神经网络，输入层接收形状为 (1, 28, 28) 的图像，将图像展平为一维向量 (784)。隐藏层使用两层神经元 LeakyReLU 来完成压缩操作，输出层将 256 维的空间映射到 1，用 Sigmoid 来进行激活。输出的范围为 0-1，表示输入是真实图像的概率。

然后是生成器。生成器也是一个全连接的神经网络，接收 100 维的随机噪声向量，然后按照前面所说的进行反向的还原的一个操作。输出层最后重塑为 (1, 28, 28) 的图像形状。

总结来说，生成器的维度变化过程为 100 → 256 → 512 → 784；判别器的变化过程为 784 → 512 → 256 → 1。

5.2 原始 GAN 网络训练分析

我们对原始的 GAN 网络进行了训练，展示一下我们的训练结果，如图5.2和图5.3所示。

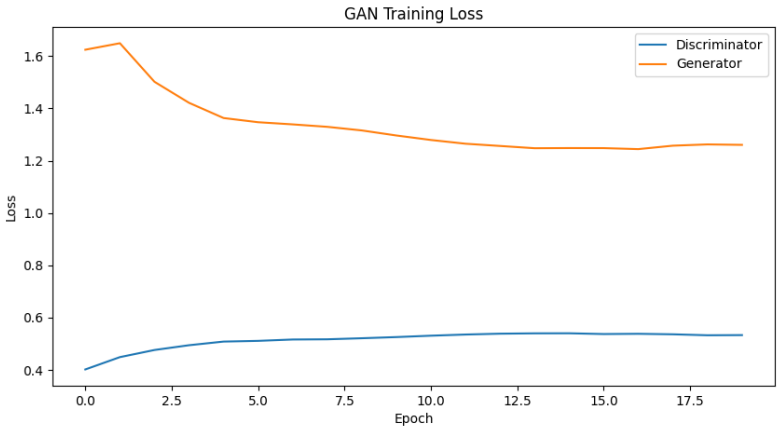

图 5.2: gan_loss_curve

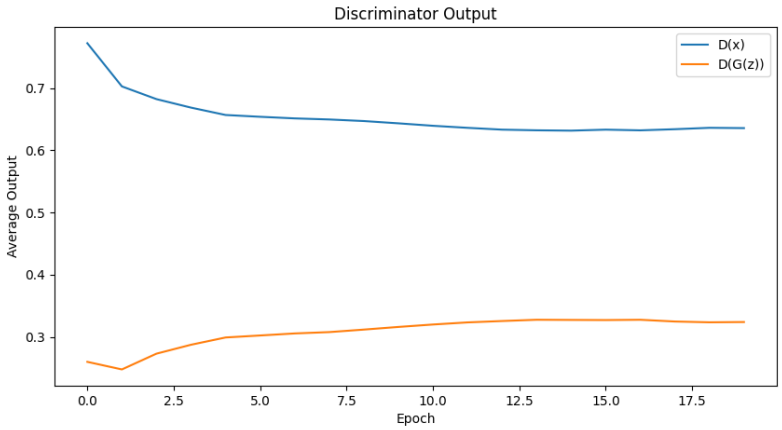

图 5.3: gan_discriminator_output

下面我们简单对我们的训练损失结果进行分析。

- **生成器（Generator）**
  - **初始阶段（Epoch 0 - 2 左右）**：损失值较高且有短暂上升，可能是生成器刚开始学习，生成的样本质量差，被判别器轻易识别，导致损失快速变化。
  - **中期阶段（Epoch 2 - 12 左右）**：损失呈明显下降趋势，说明生成器在不断优化，生成的样本逐渐“骗过”判别器，训练效果提升。
  - **后期阶段（Epoch 12 之后）**：损失趋于平稳，波动小，意味着生成器已较稳定，生成样本质量进入平台期，难再大幅提升。
- **判别器（Discriminator）**
  - **初始阶段（Epoch 0 - 2 左右）**：损失从较低值快速上升，因生成器初期生成样本差，判别器容易区分真假，随着生成器优化，判别难度增加，损失上升。
  - **中期及后期（Epoch 2 之后）**：损失缓慢上升后趋于平稳，反映判别器逐渐适应生成器生成的样本，二者在训练中达到动态博弈平衡，判别器难以更精准区分真假，损失不再大幅变化。

展示一下我们训练后得到的图像。我们一共训练了 20 个轮次，由于篇幅原因，我们只展示第一轮（如图5.4）和最后一轮（如图5.5）的训练结果。

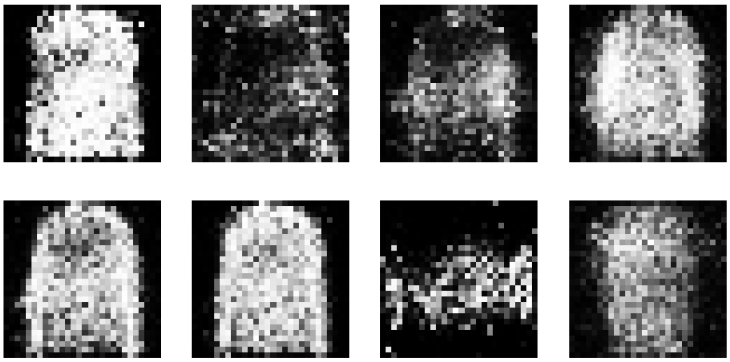

图 5.4: 原始 GAN 的第一轮训练结果

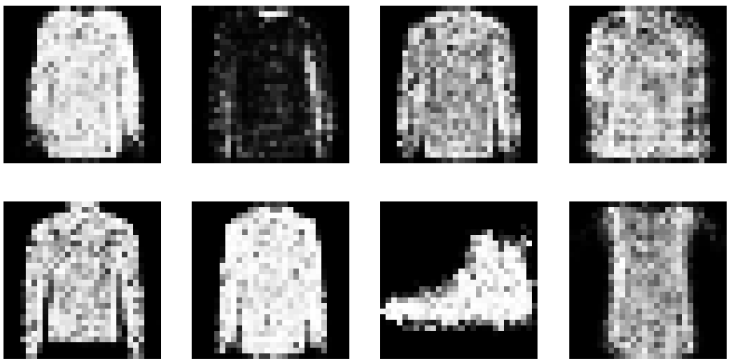

图 5.5: 原始 GAN 的最后一轮训练结果

可以看出，我们的图像由第一轮的十分模糊，到了最后一轮的十分清晰。图像由随机噪声逐步演化出可辨识的轮廓与纹理，说明生成网络成功学到了判别器隐含的真实数据分布信息。

5.3 卷积实现 CNN-GAN 网络

接下来我们实现加分项，也就是利用卷积神经网络来实现 GAN 网络。我们设计了一个具有三层卷积层和一层输出层的神经网络，结构大体如下所示：

```
Discriminator(
  (model): Sequential(
    (0): Conv2d(1, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Conv2d(128, 256, kernel_size=(4, 4), stride=(1, 1))
```

```
    (6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.2, inplace=True)
    (8): Conv2d(256, 1, kernel_size=(4, 4), stride=(1, 1))
    (9): Sigmoid()
  )
)

Generator(
  (linear): Sequential(
    (0): Linear(in_features=100, out_features=6272, bias=True)
  )
  (model): Sequential(
    (0): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (1): Upsample(scale_factor=2.0, mode='nearest')
    (2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): BatchNorm2d(128, eps=0.8, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Upsample(scale_factor=2.0, mode='nearest')
    (6): Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): BatchNorm2d(64, eps=0.8, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (8): LeakyReLU(negative_slope=0.2, inplace=True)
    (9): Conv2d(64, 1, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (10): Tanh()
  )
)
```

我们来进行简单的分析。

- **判别器（Discriminator）**
  - 采用卷积神经网络结构，输入层接收形状为 $(1, 28, 28)$ 的图像
  - 通过一系列卷积操作逐步提取特征并降低空间维度：
    - 第一层：使用 64 个 4×4 的卷积核，输出维度为 $(64, 14, 14)$
    - 第二层：使用 128 个卷积核，输出维度为 $(128, 7, 7)$
    - 第三层：使用 256 个卷积核，输出维度为 $(256, 4, 4)$
    - 输出层：通过卷积和 Sigmoid 激活函数，输出范围在 $[0, 1]$ 之间的概率值，表示输入是真实图像的可能性
- **生成器（Generator）**

- 设计采用上采样和卷积的组合来逐步还原图像：
  - 输入层：接收 100 维随机噪声向量 $\mathbf{z} \in \mathbb{R}^{100}$
  - 线性变换：将噪声向量扩展为 $128 \times 7 \times 7$ 的特征张量
  - 第一次上采样：将特征图尺寸扩大到 $(128, 14, 14)$
  - 第二次上采样：进一步扩大到 $(128, 28, 28)$
  - 输出层：通过卷积层和 Tanh 激活函数，生成最终的 $(1, 28, 28)$ 图像

整个网络充分利用了卷积操作的特点，能够更好地保持图像的空间特征，相比全连接网络更适合图像生成任务。每一层都采用了 BatchNorm 和 LeakyReLU 等技术来保证训练的稳定性和效果。

5.4 CNN-GAN 网络训练分析

我们来对我们设计的卷积 GAN 进行测试。展示一下我们的训练结果，如图5.6和图5.7所示。

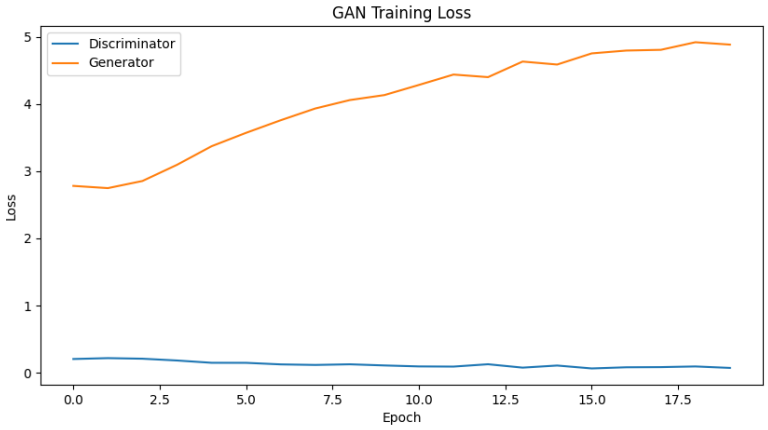

图 5.6: cnn_gan_loss_curve

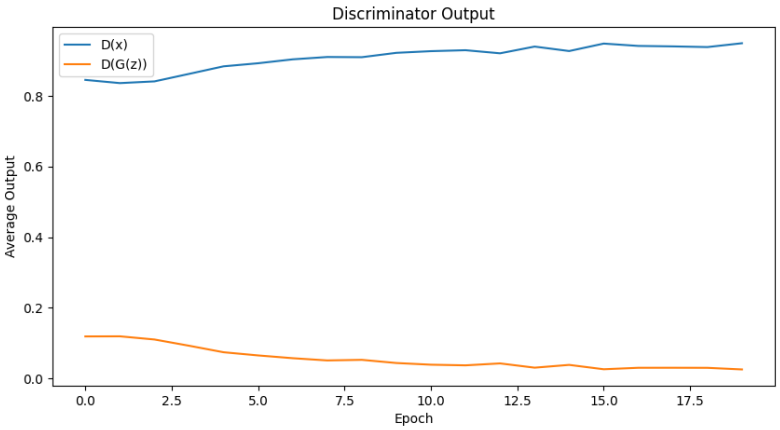

图 5.7: cnn_gan_discriminator_output

我们也对损失值做一个简单的分析。

1. **判别器：**损失值始终维持在极低水平（接近 0 ），且训练过程中波动极小。这表明判别器对真假样本的区分能力极强，能稳定、精准地识别输入样本的真伪，在整个训练周期里，判别器的判别逻辑未出现明显波动，一直保持高效区分状态。

2. **生成器：**损失值呈现持续上升态势。说明随着训练推进，生成器生成的样本越来越难“骗过”判别器，判别器对生成样本的识别准确率不断提高，生成器面临的优化压力持续增大，生成样本与真实样本的差距虽在训练中被判别器放大，但也反映出生成器尚未达到较好的生成效果，仍需优化。

展示一下我们训练后得到的图像。我们一共训练了 20 个轮次，由于篇幅原因，我们只展示第一轮（如图5.8）和最后一轮（如图5.9）的训练结果。

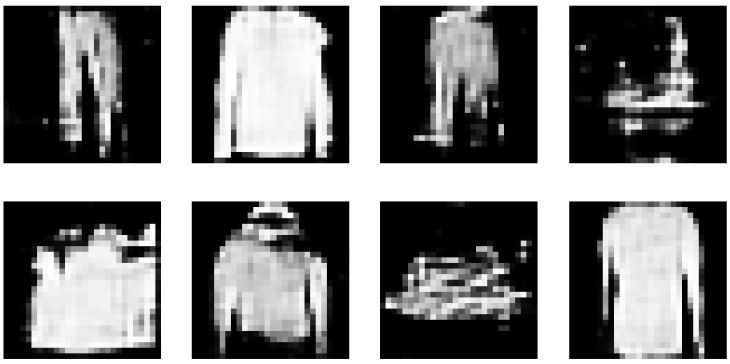

图 5.8: 卷积 GAN 的第一轮训练结果

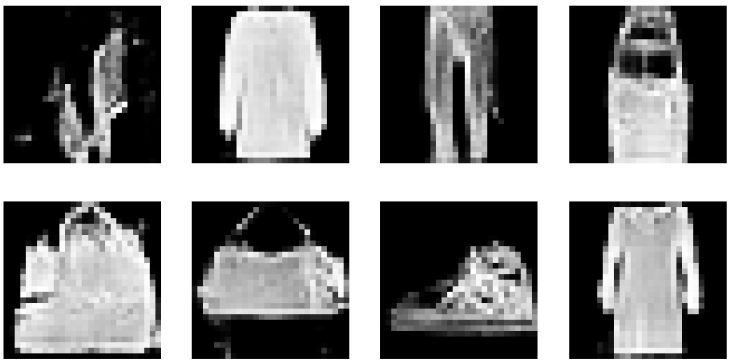

图 5.9: 卷积 GAN 的最后一轮结果

可以看出，我们的图像也是从模糊到了清晰，说明我们的生成网络也成功学到了判别器隐含的真实数据分布。

5.5 随机数调整对于图像的影响

下面我们研究一下对应的随机数的调整对于我们整个图像的影响有多大。本次实验中我们有一个参数 latent_dim，维度为 100，可以取 0-99 的任何值。我们的任务就是调整随机数的值，来观察不同的随机数对于我们图像的影响情况。

本人编写了一个程序来完成这一个调参实验，具体代码实现详见我的代码文件，此处就不再赘述。通过改变维度值和同一维度下的参数调整值，我们分别完成了维度为 10、20、37、53、67，调整幅度为 2、4、6、8、10 的实验，共生成了 25 个比较结果，每个比较结果中含有负向调整、不调整和正向调整的三张图像，每张图像中含有八个样例。下由于篇幅原因，此处仅仅展示两个层面的比较。**第一个层面是固定维度，变化调整幅度，观察带来的影响；第二个层面是固定调整幅度，变化维度，观察带来的影响**。

首先是固定维度。我们挑选维度为 10 的情况进行实验，分别选择调整幅度为 2、4、6、8、10 来进行实验，实验结果如下所示。

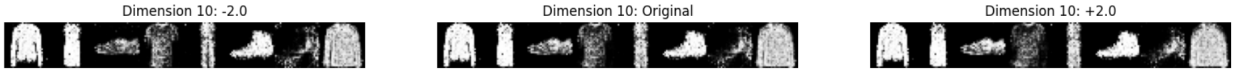

图 5.10: 维度为 10，幅度为 2

图 5.11: 维度为 10，幅度为 4

图 5.12: 维度为 10，幅度为 6

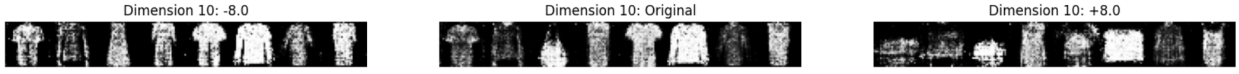

图 5.13: 维度为 10，幅度为 8

图 5.14: 维度为 10，幅度为 10

从上面这些图像中，我们可以发现，随着幅度的增大，我们的图像的变化情况是非常大的。

- **低幅度（**±2、±4 **）**：调整后样本（左右列）与原始样本（中间列）差异相对小，轮廓、主体特征保留度高。比如幅度 ±2 时，左右样本能看出和原始样本的“继承性”，说明小幅度调整对生成样本的“篡改”温和，特征变化可控。

- **中高幅度 (**±6、±8、±10 **)**：调整后样本与原始样本差异显著增大，部分样本轮廓、类别特征（如衣物样式）开始扭曲、模糊。幅度越大（如 ±10 ），样本失真越明显，甚至难以识别原始类别，反映高幅度调整会突破生成样本的特征约束，导致特征混乱。

然后我们再来进行固定幅度不变、调整维度的实验。我们固定幅度为 2，分别选取幅度为 2、4、6、8、10 来进行实验。对应的结果如下所示。

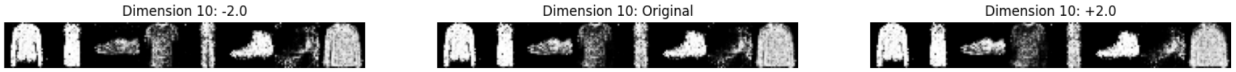

图 5.15: 维度为 10，幅度为 2

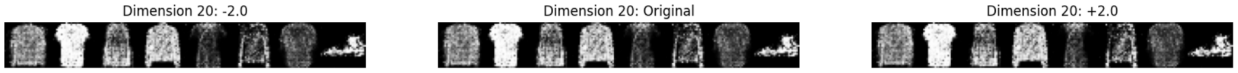

图 5.16: 维度为 20，幅度为 2

图 5.17: 维度为 37，幅度为 2

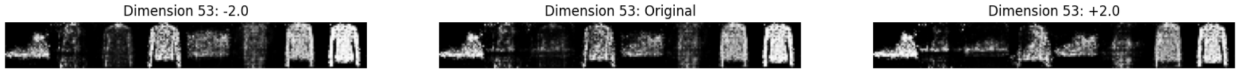

图 5.18: 维度为 53，幅度为 2

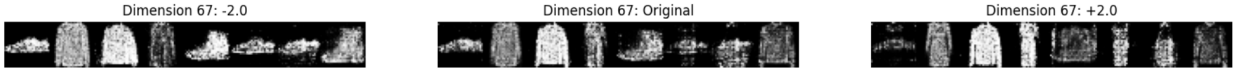

图 5.19: 维度为 67，幅度为 2

我们发现，每个维度进行调整后，样本变化呈现维度专属模式：

- **维度 10：**调整后样本与原始样本差异小，轮廓、主体特征（衣物形态）保留度高，说明该维度对应细粒度、局部特征（如衣物纹理、小细节）。
- **维度 20：**样本变化集中在类别内差异（如衣物排列、局部轮廓变形），但整体仍可识别，对应特征偏向中等粒度的结构调整（如衣物形状微调）。
- **维度 37：**样本出现类别特征模糊（部分衣物样式难识别），但仍有原始样本的“影子”，说明该维度调控较复杂的特征组合（如纹理 + 形状的混合变化）。
- **维度 53：**样本失真更明显，原始特征（如衣物数量、排列）被大幅改变，对应高维度特征交互（多个基础特征的联动调整）。
- **维度 67：**样本严重扭曲，原始类别几乎无法识别，反映该维度涉及全局特征重构（生成器对样本整体结构的“重写”）。

由于篇幅关系，我们不再展示关于卷积 GAN 的修改参数实验。

5.6 不同随机数对生成结果的影响

随机数中的向量可以认为是在图像特征空间中的一个点，而向量中的每一个维度按理来说都会包含一些判别性的信息在其中。如果我们调整了某一个维度的值，可能就是调整了某一判别性特征的值，导致图像在生成的过程中朝着某一个类别靠近。

我们也可以将这个问题反过来理解，判别器就是根据输入的图像，最终得到一个特征向量，然后将这个特征向量通过分类器进行分类，因此特征向量中的某些维度就是显式的表明了一些类别信息。而生成器则可以反过来认为输入的就是这个具有判别性的特征向量，根据其中每一维数据所指示的类型判别信息，生成对应类别的图片。

简言之，随机向量是隐含类别特征的 **“预编码指令”**，判别器负责显式定义 **“指令维度与类别特征的对应关系”**，生成器则按该关系**逆向**生成图像——本质是**判别-生成**在特征空间的双向博弈与协同，让随机数维度调整成为可控的类别特征编辑工具。

6 总结与体会

通过本次实验，我掌握了简单的 GAN 网络的编写，并实现了卷积 GAN 网络。

通过原始 GAN 与卷积 GAN 的对比，我发现原始 GAN 通过全连接网络实现生成与判别对抗，损失曲线呈现典型的动态平衡趋势，生成样本从噪声逐步演化出清晰轮廓；而卷积 GAN 利用 CNN 的空间特征提取能力，虽判别器性能更强（损失接近 0），但生成器损失持续上升，暴露了训练失衡问题，验证了卷积操作对特征提取的高效性与对抗博弈平衡的挑战性。

另外，我们通过固定维度调整实验，观察到不同维度对特征的专属调控作用。生成器在训练中自发将噪声维度映射到解耦的语义轴，为定向图像编辑提供了可能。

GAN 的魅力在于其“创造性”——通过噪声与对抗学习，从无到有生成接近真实的样本。本次实验也让我更加深入地了解到了该网络的一些基本特征，希望在后续的学习之中，还可以学习更多深度学习相关的知识。

7 文件目录结构

本次实验的代码文件目录结构如下所示：

```
codes ..........................................项目总目录
├── .gitignore ..................................git 忽略文件
├── experiment_latent.py ..........................潜在空间分析脚本
├── main.py ..................................主程序入口
├── requirements.txt ..........................依赖库清单
├── utils.py ..................................工具函数集合
├── results ..................................实验结果目录
│   ├── latent_analysis ..........................潜在空间分析结果
│   ├── gan ..................................GAN 模型结果
│   │   ├── discriminator.pth ..........................判别器模型权重
│   │   ├── discriminator_output.png ..........................判别器输出可视化
│   │   ├── generator.pth ..........................生成器模型权重
│   │   ├── loss_curve.png ..........................训练损失曲线
│   │   └── images ..........................生成图像序列
│   └── cnn_gan ..................................CNN-GAN 模型结果
│       ├── discriminator.pth ..........................判别器模型权重
│       ├── discriminator_output.png ..........................判别器输出可视化
│       ├── generator.pth ..........................生成器模型权重
│       ├── loss_curve.png ..........................训练损失曲线
│       └── images ..........................生成图像序列
└── model ..................................模型定义模块
    ├── CNN_GAN.py ..........................卷积生成对抗网络模型
    └── GAN.py ..................................生成对抗网络模型
```

本次实验的有关代码和文件，都已经上传至我的个人 github 中。您可以通过访问**此链接**来查阅我的代码文件。

附录 A GAN 网络代码实现

```python
import torch.nn as nn
from math import prod

class Discriminator(nn.Module):

    def __init__(self, input_shape=(1, 28, 28)):

        super(Discriminator, self).__init__()
        self.model = nn.Sequential(
            nn.Flatten(),
            nn.Linear(prod(input_shape), 512),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(512, 256),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(256, 1),
            nn.Sigmoid()
        )

    def forward(self, x):

        return self.model(x)

class Generator(nn.Module):

    def __init__(self, latent_dim=100, output_shape=(1, 28, 28)):

        super(Generator, self).__init__()
        self.latent_dim = latent_dim
        self.output_shape = output_shape
        self.model = nn.Sequential(
            nn.Linear(latent_dim, 256),
            nn.LeakyReLU(0.2, inplace=True),
            nn.BatchNorm1d(256, 0.8),
            nn.Linear(256, 512),
            nn.LeakyReLU(0.2, inplace=True),
            nn.BatchNorm1d(512, 0.8),
            nn.Linear(512, prod(output_shape)),
            nn.Tanh()
        )

```

```
    def forward(self, x):

        out = self.model(x)
        return out.view(out.size(0), *self.output_shape)

    def get_latent_dim(self):
        return self.latent_dim
```

附录 B CNN GAN 网络代码实现

```
import torch.nn as nn

class Discriminator(nn.Module):

    def __init__(self, input_shape=(1, 28, 28)):

        super(Discriminator, self).__init__()
        self.model = nn.Sequential(
            nn.Conv2d(input_shape[0], 64, kernel_size=4, stride=2, padding=1),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(64, 128, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(128, 256, kernel_size=4, stride=1, padding=0),
            nn.BatchNorm2d(256),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(256, 1, kernel_size=4, stride=1, padding=0),
            nn.Sigmoid()
        )

    def forward(self, x):

        return self.model(x).view(x.size(0), -1)

class Generator(nn.Module):

    def __init__(self, latent_dim=100, output_shape=(1, 28, 28)):

        super(Generator, self).__init__()
        self.latent_dim = latent_dim
        self.output_shape = output_shape
```

```
        self.init_size = output_shape[1] // 4
        self.linear = nn.Sequential(nn.Linear(latent_dim, 128 * self.init_size *
        ↪  self.init_size))

        self.model = nn.Sequential(
            nn.BatchNorm2d(128),
            nn.Upsample(scale_factor=2),
            nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(128, 0.8),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Upsample(scale_factor=2),
            nn.Conv2d(128, 64, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(64, 0.8),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(64, output_shape[0], kernel_size=3, stride=1, padding=1),
            nn.Tanh()
        )

    def forward(self, x):

        out = self.linear(x)
        out = out.view(out.size(0), 128, self.init_size, self.init_size)
        return self.model(out)

    def get_latent_dim(self):
        return self.latent_dim
```

参考文献

[1] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.