

# 智能計算 HW3

資工碩一 M1354024 戴育琪

DCGAN

FID: 80.2799

# 1. 自定義數據集與數據加載

# 自定义数据集类

```
class AnimeDataset(Dataset):  
    def __init__(self, image_dir, transform=None):  
        self.image_dir = image_dir  
        self.transform = transform  
        self.image_files = [f for f in os.listdir(image_dir) if f.endswith('.jpg')]  
  
    def __len__(self):  
        return len(self.image_files)  
  
    def __getitem__(self, idx):  
        img_path = os.path.join(self.image_dir, self.image_files[idx])  
        image = Image.open(img_path).convert('RGB')  
        if self.transform:  
            image = self.transform(image)  
        return image
```

- 從指定資料夾中讀取圖片並應用預處理(調整大小、正規化等)

## 2. 生成器 (Generator)

```
class Generator(nn.Module):  
    def __init__(self):  
        super(Generator, self).__init__()  
        self.model = nn.Sequential(  
            nn.Linear(100, 128 * 8 * 8),  
            nn.ReLU(),  
            nn.Unflatten(1, (128, 8, 8)),  
            nn.ConvTranspose2d(128, 64, 4, 2, 1),  
            nn.ReLU(),  
            nn.ConvTranspose2d(64, 32, 4, 2, 1),  
            nn.ReLU(),  
            nn.ConvTranspose2d(32, 3, 4, 2, 1),  
            nn.Tanh()  
        )  
  
    def forward(self, x):  
        return self.model(x)
```

- 功能: 將隨機噪聲轉換為高維圖像。
- 模型結構:
  - 全連接層: 將噪聲映射為高維特徵。
  - 反卷積層 (ConvTranspose2d): 逐步放大圖像並生成 RGB 圖像。
  - 激活函數 (ReLU 和 Tanh): 提供非線性轉換, Tanh 將輸出範圍限制為  $[-1, 1]$ 。

### 3. 判別器 (Discriminator)

```
class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        self.model = nn.Sequential(
            nn.Conv2d(3, 64, 4, 2, 1),
            nn.LeakyReLU(0.2),
            nn.Conv2d(64, 128, 4, 2, 1),
            nn.LeakyReLU(0.2),
            nn.Conv2d(128, 128, 4, 2, 1),
            nn.LeakyReLU(0.2),
            nn.Flatten(),
            nn.Linear(128 * 8 * 8, 1),
            nn.Sigmoid()
        )

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

- 功能: 區分輸入圖像是真實的還是生成的。
- 模型結構:
  - 卷積層 (Conv2d): 提取圖像特徵, 逐步降低空間維度。
  - LeakyReLU: 引入非線性特徵並避免梯度消失。
  - 全連接層 + Sigmoid: 輸出一個真實性概率值。

## 4.1 訓練開始與進度條初始化

```
# 訓練循環
for epoch in range(num_epochs):
    progress_bar = tqdm(enumerate(dataloader), total=len(dataloader))
    for i, real_images in progress_bar:
        real_images = real_images.to(device)
        batch_size = real_images.size(0)
```

- 訓練過程會進行多個 epoch，每次迭代都會從數據加載器 (dataloader) 中讀取批次的圖像數據。
- tqdm 是用於顯示進度條的工具，提供直觀的進度與相關資訊，例如當前 epoch、損失值等。
- 輸入真實圖像：從 dataloader 加載的真實圖像傳入判別器，輸出一個向量，表示每張圖像的真實性分數。

## 4.2 訓練判別器:真實圖像損失

```
# 訓練判別器
optimizerDis.zero_grad()

# 真實數據
labels_real = torch.ones(batch_size, device=device)
output_real = discriminator(real_images).view(-1)
loss_real = criterion(output_real, labels_real)
loss_real.backward()
```

- 生成標籤: 為真實圖像生成標籤(labels\_real), 其值均為 1, 代表目標希望判別器輸出「真實」。
- 計算損失: 使用 BCELoss, 計算判別器對真實圖像的輸出與標籤之間的誤差。
- 更新梯度: 進行反向傳播 (loss\_real.backward()), 計算與儲存梯度以便稍後更新參數。

## 4.3 訓練判別器:生成圖像損失

```
noise = torch.randn(batch_size, nz, device=device)
fake_images = generator(noise)
labels_fake = torch.zeros(batch_size, device=device)
output_fake = discriminator(fake_images.detach()).view(-1)
loss_fake = criterion(output_fake, labels_fake)
loss_fake.backward()
optimizerDis.step()
```

- 生成假圖像:使用隨機噪聲作為輸入,經由生成器生成假圖像( fake\_images)。此處的噪聲是一個隨機張量,其形狀為 (batch\_size,nz),其中 nz=100 為噪聲向量的維度。
- 標籤生成:假圖像的標籤( labels\_fake)設為 0,目標是希望判別器將這些圖像判為假。
- 計算損失:判別器輸出與標籤進行對比,計算假圖像的損失( loss\_fake)。
- 停止生成器梯度更新:使用 fake\_images.detach(),確保假圖像梯度不影響生成器的參數。
- 更新判別器參數:使用兩部分損失(真實與假圖像損失)進行參數更新。



## 4.4 訓練生成器

```
optimizerGen.zero_grad()
output_fake = discriminator(fake_images).view(-1)
loss_gen = criterion(output_fake, labels_real) # 使用真實標籤
loss_gen.backward()
optimizerGen.step()
```

- 重新評估假圖像: 將生成的假圖像再傳入判別器, 希望它的輸出接近 1 (即假圖像能「騙過」判別器)。
- 標籤與目標: 生成器目標是生成能被判別器認為真實的圖像, 因此損失函數的標籤設為 1 (labels\_real)。
- 更新生成器: 計算生成器的損失後, 進行反向傳播, 更新生成器的參數。

## 5. 保存模型

# 保存模型

```
torch.save(generator.state_dict(), f'/home/yuchi/AI/DCGAN/model/generator_{epoch+1}.pth')  
torch.save(discriminator.state_dict(), f'/home/yuchi/AI/DCGAN/model/discriminator_{epoch+1}.pth')
```

- 每個 epoch 結束後保存生成器與判別器模型參數，便於後續測試與部署。

Epoch [15/20]: 100%	██████████	3973/3973	[12:58<00:00, 5.11it/s]
Epoch [16/20]: 100%	██████████	3973/3973	[12:14<00:00, 5.41it/s]
Epoch [17/20]: 100%	██████████	3973/3973	[10:31<00:00, 6.29it/s]
Epoch [18/20]: 100%	██████████	3973/3973	[09:00<00:00, 7.35it/s]
Epoch [19/20]: 100%	██████████	3973/3973	[09:05<00:00, 7.28it/s]
Epoch [20/20]: 100%	██████████	3973/3973	[09:06<00:00, 7.27it/s]

```
≡ generator_15.pth  
≡ generator_16.pth  
≡ generator_17.pth  
≡ generator_18.pth  
≡ generator_19.pth  
≡ generator_20.pth
```

## 6. 生成結果

```
generator = Generator()  
# 載入模型  
generator.load_state_dict(torch.load(f"/home/yuchi/AI/DCGAN/model/generator_20.pth"))  
generator.eval()
```

- 載入模型架構: `Generator()` 初始化生成器的網路結構
- 載入已訓練參數: 使用 `torch.load` 載入訓練完成後儲存的模型參數檔案( `generator_20.pth`)
- 設定為評估模式: `generator.eval()` 將模型切換到評估模式, 避免影響模型行為

## 7. 生成圖像

```
for i in range(500):  
    # Generate a batch of noise vectors  
    noise = torch.randn(1, 100)  
    # Generate an image  
    with torch.no_grad():  
        fake_image = generator(noise)  
  
    # Scale the image back to [0, 1] range for saving  
    fake_image = (fake_image + 1) / 2  
  
    # Save image  
    save_path = os.path.join(output_dir, f"{i+1}.jpg")  
    save_image(fake_image, save_path)
```

- 生成隨機噪聲: 使用 `torch.randn` 生成一個隨機噪聲向量, 代表一組數值分布為標準正態分布的噪聲。
- 使用生成器生成圖像  
: `generator(noise)` 將噪聲向量轉換為假圖像。
- 調整像素值範圍: 使用 `(fake_image + 1) / 2` 將像素值重新映射到 `[0, 1]`, 以便儲存成標準圖像格式。
- 儲存圖像: 使用 `save_image` 將圖像存入目錄

## 8. 使用 FID 指標進行品質評估

```
!python -m pytorch_fid /home/yuchi/AI/anim /home/yuchi/AI/DCGAN/Result --batch-size 16
```

```
100%|████████████████████████████████████████████████████████████████████████████████| 63565/63565 [39:31<00:00, 26.80it/s]
```

```
100%|████████████████████████████████████████████████████████████████████████████████| 500/500 [00:23<00:00, 20.88it/s]
```

```
FID: 80.27990383204934
```



199.jpg



200.jpg



201.jpg



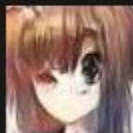
202.jpg



203.jpg



204.jpg



205.jpg



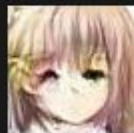
206.jpg



207.jpg



208.jpg



209.jpg



210.jpg

ACGAN

FID: 1176832

# 1. 自定義數據集與數據加載

# 自定义数据集类

```
class AnimeDataset(Dataset):  
    def __init__(self, image_dir, transform=None):  
        self.image_dir = image_dir  
        self.transform = transform  
        self.image_files = [f for f in os.listdir(image_dir) if f.endswith('.jpg')]  
  
    def __len__(self):  
        return len(self.image_files)  
  
    def __getitem__(self, idx):  
        img_path = os.path.join(self.image_dir, self.image_files[idx])  
        image = Image.open(img_path).convert('RGB')  
        if self.transform:  
            image = self.transform(image)  
        return image
```

- 從指定資料夾中讀取圖片並應用預處理(調整大小、正規化等)

## 2. 權重初始化函數

```
def weights_init_normal(m):
    classname = m.__class__.__name__
    if classname.find("Conv") != -1:
        torch.nn.init.normal_(m.weight.data, 0.0, 0.02)
    elif classname.find("BatchNorm2d") != -1:
        torch.nn.init.normal_(m.weight.data, 1.0, 0.02)
        torch.nn.init.constant_(m.bias.data, 0.0)
```

- 初始化卷積層 (Conv) 和批量歸一化層 (BatchNorm2d) 的權重



### 3. 生成器架構

```
class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()
        self.init_size = img_size // 4
        self.fc = 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, 0.8),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Upsample(scale_factor=2),
            nn.Conv2d(128, 64, 3, stride=1, padding=1),
            nn.BatchNorm2d(64, 0.8),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(64, channels, 3, stride=1, padding=1),
            nn.Tanh(),
        )

    def forward(self, noise):
        out = self.fc(noise)
        out = out.view(out.shape[0], 128, self.init_size, self.init_size)
        img = self.conv_blocks(out)
        return img
```

- 全連接層 (fc): 噪聲向量經過線性變換, 轉化為適合卷積層輸入的形狀
- 卷積塊 (conv\_blocks): 使用了多層上採樣 (Upsample)、卷積 (Conv2d)、批量歸一化 (BatchNorm2d) 和激活函數 (LeakyReLU)。
- 輸出層: 使用 Tanh 將像素值限制在  $[-1, 1]$  範圍內。

## 4. 判別器架構

```
class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        def block(in_filters, out_filters, bn=True):
            layers = [nn.Conv2d(in_filters, out_filters, 3, 2, 1), nn.LeakyReLU(0.2, inplace=True)]
            if bn:
                layers.append(nn.BatchNorm2d(out_filters, 0.8))
            return layers

        self.model = nn.Sequential(
            *block(channels, 16, bn=False),
            *block(16, 32),
            *block(32, 64),
            *block(64, 128),
        )
        ds_size = img_size // 2 ** 4
        self.fc = nn.Sequential(nn.Linear(128 * ds_size ** 2, 1), nn.Sigmoid())

    def forward(self, img):
        out = self.model(img)
        out = out.view(out.shape[0], -1)
        validity = self.fc(out)
        return validity
```

- 多層卷積 (block): 特徵數量逐層增加 (16 → 32 → 64 → 128), 並逐步下採樣。
- 全連接層: 最後的卷積輸出展平後通過線性層, 並使用 Sigmoid 獲得有效性分數。

## 5.1 訓練開始與進度條初始化

```
for epoch in range(n_epochs):  
    with tqdm(dataloader, desc=f"Epoch {epoch + 1}/{n_epochs}", unit="batch") as progress_bar:  
        for i, imgs in enumerate(progress_bar):
```

- 訓練過程會進行多個 epoch，每次迭代都會從數據加載器 (dataloader) 中讀取批次的圖像數據。
- tqdm 是用於顯示進度條的工具，提供直觀的進度與相關資訊，例如當前 epoch、損失值等。
- 輸入真實圖像：從 dataloader 加載的真實圖像傳入判別器，輸出一個向量，表示每張圖像的真實性分數。

## 5.2 建立標籤

```
batch_size = imgs.size(0)
valid = Variable(torch.ones(batch_size, 1).cuda() if cuda else torch.ones(batch_size, 1))
fake = Variable(torch.zeros(batch_size, 1).cuda() if cuda else torch.zeros(batch_size, 1))
real_imgs = Variable(imgs.cuda() if cuda else imgs)
```

- `batch_size = imgs.size(0)`:從數據集中加載的圖像批次數量。
- `valid = Variable(torch.ones(batch_size, 1))`:創建一個大小為 `(batch_size, 1)` 的張量, 全部為 1。用作真實圖像的標籤, 對應判別器輸出的目標值。
- `fake = Variable(torch.zeros(batch_size, 1))`:創建一個大小為 `(batch_size, 1)` 的張量, 全部為 0。用作生成圖像的標籤, 對應判別器輸出的目標值。
- `real_imgs = Variable(imgs)`:將載入的真實圖像批次轉為可訓練的張量, 方便之後傳入模型。

## 5.3 訓練生成器

```
# Train Generator
optimizer_G.zero_grad()
z = Variable(torch.randn(batch_size, latent_dim).cuda() if cuda else torch.randn(batch_size, latent_dim))
gen_imgs = generator(z)
g_loss = adversarial_loss(discriminator(gen_imgs), valid)
g_loss.backward()
optimizer_G.step()
```

- 將隨機噪聲輸入生成器，生成假圖像。
- 計算假圖像被判別器判斷為真實的損失。
- 更新生成器權重。
- 目的：提升生成器欺騙判別器的能力。

## 5.4 訓練判別器





```
# Train Discriminator
optimizer_D.zero_grad()
real_loss = adversarial_loss(discriminator(real_imgs), valid)
fake_loss = adversarial_loss(discriminator(gen_imgs.detach()), fake)
d_loss = (real_loss + fake_loss) / 2
d_loss.backward()
optimizer_D.step()
```

- 分別計算真實圖像與假圖像的損失。
- 更新判別器權重，使其更準確地區分真實與假圖像。
- 目的：提升判別器的判別能力。

## 6. 保存模型

```
torch.save(generator.state_dict(), f'/home/yuchi/AI/ACGAN/model/generator_{epoch+1}.pth')
torch.save(discriminator.state_dict(), f'/home/yuchi/AI/ACGAN/model/discriminator_{epoch+1}.pth')
```

- 每個 epoch 結束後保存生成器與判別器模型參數，便於後續測試與部署。

Epoch [21/30]: 100%			3973/3973	[54:37<00:00, 1.21batch/s]
Epoch [22/30]: 100%			3973/3973	[46:10<00:00, 1.43batch/s]
Epoch [23/30]: 100%			3973/3973	[49:54<00:00, 1.33batch/s]
Epoch [24/30]: 100%			3973/3973	[40:12<00:00, 1.65batch/s]
Epoch [25/30]: 100%			3973/3973	[1:14:58<00:00, 1.13s/batch]

```
≡ generator_21.pth
≡ generator_22.pth
≡ generator_23.pth
≡ generator_24.pth
≡ generator_25.pth
```

## 7. 生成結果

```
# 加載訓練好的生成器模型
generator = Generator()
generator.load_state_dict(torch.load("/home/yuchi/AI/ACGAN/model/generator_20.pth")) # 替換為最後保存的生成器模型檔案
generator.eval() # 設定為推理模式
```

- 載入模型架構: `Generator()` 初始化生成器的網路結構
- 載入已訓練參數: 使用 `torch.load` 載入訓練完成後儲存的模型參數檔案
- 設定為評估模式: `generator.eval()` 將模型切換到評估模式, 避免影響模型行為



## 8.1 禁用梯度計算, 初始化計數器

```
with torch.no_grad(): # 禁用梯度計算以提高效率
    img_counter = 1 # 從 1 開始命名圖片
```

- `torch.no_grad()` 禁用梯度計算, 避免不必要的計算開銷, 適合推理階段。
- 初始化計數器 `img_counter`, 用於生成圖片的編號。

## 8.2 批次生成與儲存圖片

```
for batch in range(total_batches):
    # 隨機生成潛在向量 z
    z = torch.randn(batch_size, latent_dim, device=device)

    # 使用生成器生成圖片
    gen_imgs = generator(z)

    # 儲存圖片
    for i in range(batch_size):
        save_path = os.path.join(f"/home/yuchi/AI/ACGAN/Result/{img_counter}.jpg")
        save_image(gen_imgs[i], save_path, normalize=True)
        img_counter += 1
```

- 生成潛在向量 (z): 生成潛在向量。
- 生成圖片: 將潛在向量輸入生成器, 生成對應的假圖像。
- 儲存圖片
- 模擬真實批次生成過程, 確保生成的圖片整齊保存, 便於後續質量檢驗。

## 8.3 處理非整數倍的剩餘圖片

```
# 處理剩餘的圖片 (如果 num_images 不是 batch_size 的整數倍)
remaining = num_images % batch_size
if remaining > 0:
    z = torch.randn(remaining, latent_dim, device=device)
    gen_imgs = generator(z)
    for i in range(remaining):
        save_path = os.path.join(f"/home/yuchi/AI/ACGAN/Result/{img_counter}.jpg")
        save_image(gen_imgs[i], save_path, normalize=True)
        img_counter += 1
```

- 處理總圖片數目 num\_images 不是批次大小 batch\_size 的整數倍時，對剩餘的圖片進行單獨處理。
- 生成並儲存剩餘圖片。
- 確保生成的圖片總數符合預期 (num\_images)。

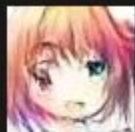
## 9. 使用 FID 指標進行品質評估

```
!python -m pytorch_fid /home/yuchi/AI/anim /home/yuchi/AI/ACGAN/Result --batch-size 16
```

[illegible]

```
100%|███████████| 32/32 [00:17<00:00, 1.80it/s]
```

FID: 117.68326672888227



7.jpg



8.jpg



9.jpg



10.jpg



11.jpg



12.jpg



13.jpg



14.jpg



15.jpg



16.jpg



17.jpg



18.jpg

WGAN-GP  
FID:

# 1. 自定義數據集與數據加載

# 自定义数据集类

```
class AnimeDataset(Dataset):  
    def __init__(self, image_dir, transform=None):  
        self.image_dir = image_dir  
        self.transform = transform  
        self.image_files = [f for f in os.listdir(image_dir) if f.endswith('.jpg')]  
  
    def __len__(self):  
        return len(self.image_files)  
  
    def __getitem__(self, idx):  
        img_path = os.path.join(self.image_dir, self.image_files[idx])  
        image = Image.open(img_path).convert('RGB')  
        if self.transform:  
            image = self.transform(image)  
        return image
```

- 從指定資料夾中讀取圖片並應用預處理(調整大小、正規化等)

## 2. 生成器架構

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

        def block(in_feat, out_feat, normalize=True):
            layers = [nn.Linear(in_feat, out_feat)]
            if normalize:
                layers.append(nn.BatchNorm1d(out_feat, 0.8))
            layers.append(nn.LeakyReLU(0.2, inplace=True))
            return layers

        self.model = nn.Sequential(
            *block(latent_dim, 128, normalize=False),
            *block(128, 256),
            *block(256, 512),
            *block(512, 1024),
            nn.Linear(1024, int(np.prod(img_shape))),
            nn.Tanh()
        )

    def forward(self, z):
        img = self.model(z)
        img = img.view(img.shape[0], *img_shape)
        return img
```

- 接受隨機潛在空間向量  $z$  作為輸入，通過多層全連接層進行特徵轉換，最終生成符合指定形狀的影像。
- Tanh 激活函數將生成的像素值標準化到範圍  $[-1,1]$ ，方便後續處理。
- 目的：基於隨機噪聲生成多樣化且逼真的影像。

### 3. 判別器

# 定义判别器

```
class Discriminator(nn.Module):  
    def __init__(self):  
        super(Discriminator, self).__init__()  
  
        self.model = nn.Sequential(  
            nn.Linear(int(np.prod(img_shape)), 512),  
            nn.LeakyReLU(0.2, inplace=True),  
            nn.Linear(512, 256),  
            nn.LeakyReLU(0.2, inplace=True),  
            nn.Linear(256, 1),  
        )  
  
    def forward(self, img):  
        img_flat = img.view(img.shape[0], -1)  
        validity = self.model(img_flat)  
        return validity
```

- 將輸入影像展平成向量，通過幾層全連接層進行特徵學習，最終輸出一個標量表示真實度分數。
- 用於區分輸入影像是真實樣本還是生成樣本。
- 目的：提供對抗信號，幫助生成器提升生成影像的真實感。



## 4. 梯度懲罰

# 梯度懲罰函數

```
def compute_gradient_penalty(D, real_samples, fake_samples):
    alpha = torch.rand(real_samples.size(0), 1, 1, 1, device=device)
    interpolates = (alpha * real_samples + (1 - alpha) * fake_samples).requires_grad_(True)
    d_interpolates = D(interpolates)
    fake = torch.ones(real_samples.shape[0], 1, device=device, requires_grad=False)
    gradients = autograd.grad(
        outputs=d_interpolates,
        inputs=interpolates,
        grad_outputs=fake,
        create_graph=True,
        retain_graph=True,
        only_inputs=True,
    )[0]
    gradients = gradients.view(gradients.size(0), -1)
    gradient_penalty = ((gradients.norm(2, dim=1) - 1) ** 2).mean()
    return gradient_penalty
```

- 計算真實影像與生成影像插值的梯度，並對梯度範數偏離 1 的情況進行懲罰。
- 目的: 解決 WGAN 訓練過程中梯度消失或爆炸的問題。

## 5.1 訓練開始與進度條初始化

```
# 开始训练
for epoch in range(n_epochs):
    progress_bar = tqdm(enumerate(dataloader), total=len(dataloader), desc=f"Epoch {epoch+1}/{n_epochs}")
    for i, imgs in progress_bar:
        real_imgs = imgs.to(device)
        batch_size = real_imgs.size(0)
```

- 訓練過程會進行多個 epoch，每次迭代都會從數據加載器 (dataloader) 中讀取批次的圖像數據。
- tqdm 是用於顯示進度條的工具，提供直觀的進度與相關資訊，例如當前 epoch、損失值等。
- 輸入真實圖像：從 dataloader 加載的真實圖像傳入判別器，輸出一個向量，表示每張圖像的真實性分數。

## 5.2 訓練判別器

# 訓練判別器

```
optimizer_D.zero_grad()
z = torch.randn(batch_size, latent_dim, device=device)
fake_imgs = generator(z)
real_validity = discriminator(real_imgs)
fake_validity = discriminator(fake_imgs)
gradient_penalty = compute_gradient_penalty(discriminator, real_imgs.data, fake_imgs.data)
d_loss = -torch.mean(real_validity) + torch.mean(fake_validity) + lambda_gp * gradient_penalty
d_loss.backward()
optimizer_D.step()
```

- 真實樣本得分 (real\_validity): 將真實影像輸入判別器, 計算得分。
- 生成樣本得分 (fake\_validity): 將生成影像輸入判別器, 計算得分。
- 梯度懲罰 (gradient\_penalty): 計算插值影像的梯度偏離 1 的懲罰。
- 判別器損失 (d\_loss): 包括 WGAN 損失和梯度懲罰。
- 目的: 提高判別器辨別真實影像與生成影像的能力。

## 5.3 訓練生成器

```
# 训练生成器
if i % 5 == 0:
    optimizer_G.zero_grad()
    fake_imgs = generator(z)
    fake_validity = discriminator(fake_imgs)
    g_loss = -torch.mean(fake_validity)
    g_loss.backward()
    optimizer_G.step()
```

- 每 5 次判别器更新後，訓練一次生成器。
- 生成器損失 (g\_loss): 最大化判别器對生成影像的得分。
- 讓生成器生成的影像更逼真，難以被判别器區分。

## 6. 保存模型

# 每轮训练结束后保存生成器模型

```
torch.save(generator.state_dict(), os.path.join(f"/home/yuchi/AI/WGAN-GP/model/generator_epoch_{epoch+1}.pth"))
```

- 每個 epoch 結束後保存生成器與判別器模型參數，便於後續測試與部署。

Epoch [20/40]: 100%	██████████		1987/1987	[13:53<00:00,	2.39it/s]
Epoch [21/40]: 100%	██████████		1987/1987	[17:47<00:00,	1.86it/s]
Epoch [22/40]: 100%	██████████		1987/1987	[14:51<00:00,	2.23it/s]
Epoch [23/40]: 100%	██████████		1987/1987	[06:18<00:00,	5.25it/s]
Epoch [24/40]: 100%	██████████		1987/1987	[02:40<00:00,	12.34it/s]
Epoch [25/40]: 100%	██████████		1987/1987	[02:30<00:00,	13.17it/s]

```
generator_epoch_20.pth
generator_epoch_21.pth
generator_epoch_22.pth
generator_epoch_23.pth
generator_epoch_24.pth
generator_epoch_25.pth
```

## 7. 生成結果

```
# 加載生成器模型
generator = Generator()
model_path = "/home/yuchi/AI/WGAN-GP/model/generator_epoch_40.pth" # 替換成實際的模型路徑
generator.load_state_dict(torch.load(model_path))
generator.eval()
```

- 載入模型架構: `Generator()` 初始化生成器的網路結構
- 載入已訓練參數: 使用 `torch.load` 載入訓練完成後儲存的模型參數檔案
- 設定為評估模式: `generator.eval()` 將模型切換到評估模式, 避免影響模型行為

## 8. 逐批生成與儲存影像

```
while generated_count < total_images:
    # 確定本批次生成圖片數量
    current_batch_size = min(batch_size, total_images - generated_count)
    z = torch.randn(current_batch_size, latent_dim, device=device) # 在正確設備上生成隨機向量
    gen_imgs = generator(z) # 利用生成器生成圖片

    # 保存圖片
    for i in range(current_batch_size):
        img_index = generated_count + i + 1
        save_path = os.path.join(f"/home/yuchi/AI/WGAN-GP/Result/{img_index}.jpg")
        save_image(gen_imgs[i], save_path, normalize=True)

    generated_count += current_batch_size
```

- 循環生成500張影像



## 9. 使用 FID 指標進行品質評估

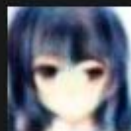
```
!python -m pytorch_fid /home/yuchi/AI/anim /home/yuchi/AI/WGAN-GP/Result --batch-size 32
```

[illegible][illegible]

FID: 183.4882343679278



25.jpg



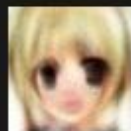
26.jpg



27.jpg



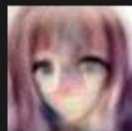
28.jpg



29.jpg



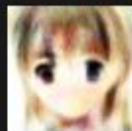
30.jpg



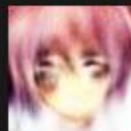
31.jpg



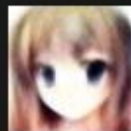
32.jpg



33.jpg



34.jpg



35.jpg



36.jpg



心得

# 心得

這次實驗的整體過程其實挺有挑戰的，主要因為時間真的很緊湊，模型的訓練尤其耗時，尤其是 WGAN-GP，因為需要計算梯度懲罰，導致每一輪的訓練時間都比一般的 GAN 長很多。這對在時間有限的情況下是一個很大的挑戰。

雖然過程有點壓力，但從中學到了很多。不只是對 GAN 的理解變深了，像 Wasserstein 距離的概念、梯度懲罰的作用，這些之前只是在理論上知道，現在通過實際實驗有了更多體會。

經過這次的實驗，我也更認識到效率的重要性。事前的計畫和資源分配真的很重要，如果能有更充分的準備，像時間規劃或訓練資料處理，都可以讓整體流程更順暢。