HW8

612415013 蕭宥羽

1. How to execute codes.
   - Original

定義了生成對抗網絡（GAN）中的生成器（Generator）和判別器（Discriminator）的架構

生成器（G）：生成器負責將隨機噪聲轉換成假圖片

✓ nn.ConvTranspose2d：轉置卷積層，用於上採樣。每一層將特徵圖放大，並將通道數減少。

✓ nn.BatchNorm2d：批量歸一化層，標準化輸出以加速訓練並穩定網絡。

✓ nn.ReLU：ReLU 激活函數，使輸出非線性。

✓ nn.Tanh：Tanh 激活函數，將輸出範圍限制在 [-1, 1]，適合圖片生成。

這些層依次將潛在向量（隨機噪聲）轉換成 64x64 的三通道（彩色）圖片。

```
G = nn.Sequential(
    # input is Z, going into a convolution
    nn.ConvTranspose2d( nz, ngf * 8, 4, 1, 0, bias=False),
    nn.BatchNorm2d(ngf * 8),
    nn.ReLU(True),
    # state size. (ngf*8) x 4 x 4
    nn.ConvTranspose2d(ngf * 8, ngf * 4, 4, 2, 1, bias=False),
    nn.BatchNorm2d(ngf * 4),
    nn.ReLU(True),
    # state size. (ngf*4) x 8 x 8
    nn.ConvTranspose2d( ngf * 4, ngf * 2, 4, 2, 1, bias=False),
    nn.BatchNorm2d(ngf * 2),
    nn.ReLU(True),
    # state size. (ngf*2) x 16 x 16
    nn.ConvTranspose2d( ngf * 2, ngf, 4, 2, 1, bias=False),
    nn.BatchNorm2d(ngf),
    nn.ReLU(True),
    # state size. (ngf) x 32 x 32
    nn.ConvTranspose2d( ngf, nc, 4, 2, 1, bias=False),
    nn.Tanh()
    # state size. (nc) x 64 x 64
)
```

判別器（D）：判別器負責將輸入的圖片（真實或生成的）判斷為真實還是偽造

- ✓ nn.Conv2d：卷積層，用於下採樣。每一層將特徵圖的大小減小，並將通道數增加。
- ✓ nn.BatchNorm2d：批量歸一化層，標準化輸出以加速訓練並穩定網絡。
- ✓ nn.LeakyReLU：Leaky ReLU 激活函數，使輸出非線性並允許部分負值通過。
- ✓ nn.Flatten：將多維輸出展平為一維。
- ✓ nn.Sigmoid：Sigmoid 激活函數，將輸出範圍限制在 [0, 1]，表示真實或偽造的概率。

這些層依次將輸入的 64x64 的三通道圖片轉換為一個標量，表示圖片的真實性。

```
D = nn.Sequential(
    # input is (nc) x 64 x 64
    nn.Conv2d(nc, ndf, 4, 2, 1, bias=False),
    nn.LeakyReLU(0.2, inplace=True),
    # state size. (ndf) x 32 x 32
    nn.Conv2d(ndf, ndf * 2, 4, 2, 1, bias=False),
    nn.BatchNorm2d(ndf * 2),
    nn.LeakyReLU(0.2, inplace=True),
    # state size. (ndf*2) x 16 x 16
    nn.Conv2d(ndf * 2, ndf * 4, 4, 2, 1, bias=False),
    nn.BatchNorm2d(ndf * 4),
    nn.LeakyReLU(0.2, inplace=True),
    # state size. (ndf*4) x 8 x 8
    nn.Conv2d(ndf * 4, ndf * 8, 4, 2, 1, bias=False),
    nn.BatchNorm2d(ndf * 8),
    nn.LeakyReLU(0.2, inplace=True),
    # state size. (ndf*8) x 4 x 4
    nn.Conv2d(ndf * 8, 1, 4, 1, 0, bias=False),
    nn.Flatten(),
    nn.Sigmoid()
)
```

權重初始化函數

- ✓ 卷積層和轉置卷積層：權重使用均值為 0、標準差為 0.02 的正態分布初始化。
- ✓ 批量歸一化層：權重使用均值為 1、標準差為 0.02 的正態分布初始化，偏置設為 0。

```
def weight_init(m):
    if isinstance(m, nn.Conv2d) or isinstance(m, nn.ConvTranspose2d):
        nn.init.normal_(m.weight.data, 0.0, 0.02)
    if isinstance(m, nn.BatchNorm2d):
        nn.init.normal_(m.weight.data, 1.0, 0.02)
        nn.init.constant_(m.bias.data, 0)
```

原始架構

生成器 (Generator)

```
Sequential(
  (0): ConvTranspose2d(100, 512, kernel_size=(4, 4), stride=(1, 1), bias=False)
  (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): ReLU(inplace=True)
  (3): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (5): ReLU(inplace=True)
  (6): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (8): ReLU(inplace=True)
  (9): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  (10): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (11): ReLU(inplace=True)
  (12): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  (13): Tanh()
)
```

判別器 (Discriminator)

```
Sequential(
  (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  (1): LeakyReLU(negative_slope=0.2, inplace=True)
  (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  (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=(2, 2), padding=(1, 1), bias=False)
  (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, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  (9): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (10): LeakyReLU(negative_slope=0.2, inplace=True)
  (11): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)
  (12): Flatten(start_dim=1, end_dim=-1)
  (13): Sigmoid()
)
```

改動後

```
G = nn.Sequential(
    # input is Z, going into a convolution
    nn.ConvTranspose2d(nz, ngf * 8, 4, 1, 0, bias=False),
    nn.BatchNorm2d(ngf * 8),
    nn.ReLU(True),
    # state size. (ngf*8) x 4 x 4
    # New block added here
    nn.ConvTranspose2d(ngf * 8, ngf * 8, 3, 1, 1, bias=False),
    nn.BatchNorm2d(ngf * 8),
    nn.ReLU(True),
    # state size. (ngf*8) x 4 x 4

    nn.ConvTranspose2d(ngf * 8, ngf * 4, 4, 2, 1, bias=False),
    nn.BatchNorm2d(ngf * 4),
    nn.ReLU(True),
    # state size. (ngf*4) x 8 x 8
    nn.ConvTranspose2d(ngf * 4, ngf * 2, 4, 2, 1, bias=False),
    nn.BatchNorm2d(ngf * 2),
    nn.ReLU(True),
    # state size. (ngf*2) x 16 x 16
    nn.ConvTranspose2d(ngf * 2, ngf, 4, 2, 1, bias=False),
```

```
    nn.BatchNorm2d(ngf),
    nn.ReLU(True),
    # state size. (ngf) x 32 x 32
    nn.ConvTranspose2d(ngf, nc, 4, 2, 1, bias=False),
    nn.Tanh()
    # state size. (nc) x 64 x 64
)

D = nn.Sequential(
    # input is (nc) x 64 x 64
    nn.Conv2d(nc, ndf, 4, 2, 1, bias=False),
    nn.LeakyReLU(0.2, inplace=True),
    # state size. (ndf) x 32 x 32

    # New block added here
    nn.Conv2d(ndf, ndf * 2, 3, 1, 1, bias=False),
    nn.BatchNorm2d(ndf * 2),
    nn.LeakyReLU(0.2, inplace=True),
    # state size. (ndf*2) x 32 x 32

    nn.Conv2d(ndf * 2, ndf * 4, 4, 2, 1, bias=False),
    nn.BatchNorm2d(ndf * 4),
    nn.LeakyReLU(0.2, inplace=True),
    # state size. (ndf*4) x 16 x 16
    nn.Conv2d(ndf * 4, ndf * 8, 4, 2, 1, bias=False),
    nn.BatchNorm2d(ndf * 8),
    nn.LeakyReLU(0.2, inplace=True),
    # state size. (ndf*8) x 8 x 8
    nn.Conv2d(ndf * 8, ndf * 8, 4, 2, 1, bias=False),
    nn.BatchNorm2d(ndf * 8),
    nn.LeakyReLU(0.2, inplace=True),
    # state size. (ndf*8) x 4 x 4
    nn.Conv2d(ndf * 8, 1, 4, 1, 0, bias=False),
    nn.Flatten(),
    nn.Sigmoid()
A)
```

改動說明

在這個改版中，我們對生成器和判別器的架構進行了如下改動：

生成器 (Generator)

1. 在第一個區塊之後新增了一個上採樣區塊，以增加生成器的複雜度。這個區塊包括一個 ConvTranspose2d 層、一個 BatchNorm2d 層和一個 ReLU 激活函數。

```
Sequential(
  (0): ConvTranspose2d(100, 512, kernel_size=(4, 4), stride=(1, 1), bias=False)
  (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): ReLU(inplace=True)
  (3): ConvTranspose2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (5): ReLU(inplace=True)
  (6): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  (7): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (8): ReLU(inplace=True)
  (9): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  (10): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (11): ReLU(inplace=True)
  (12): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  (13): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (14): ReLU(inplace=True)
  (15): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  (16): Tanh()
)
```

判別器 (Discriminator)

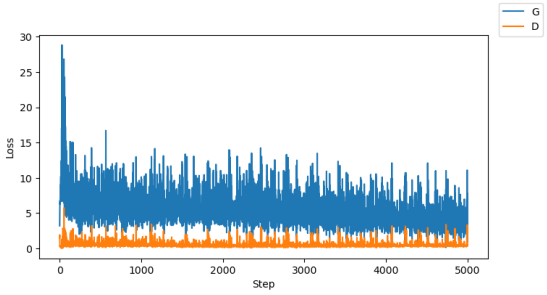
1. 在第一個區塊之後新增了一個下採樣區塊，以增加判別器的深度和容量。這個區塊包括一個 Conv2d 層、一個 BatchNorm2d 層和一個 LeakyReLU 激活函數。

```
Sequential(
  (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  (1): LeakyReLU(negative_slope=0.2, inplace=True)
  (2): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (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=(2, 2), padding=(1, 1), bias=False)
  (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, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  (9): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (10): LeakyReLU(negative_slope=0.2, inplace=True)
  (11): Conv2d(512, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  (12): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (13): LeakyReLU(negative_slope=0.2, inplace=True)
  (14): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)
  (15): Flatten(start_dim=1, end_dim=-1)
  (16): Sigmoid()
)
```

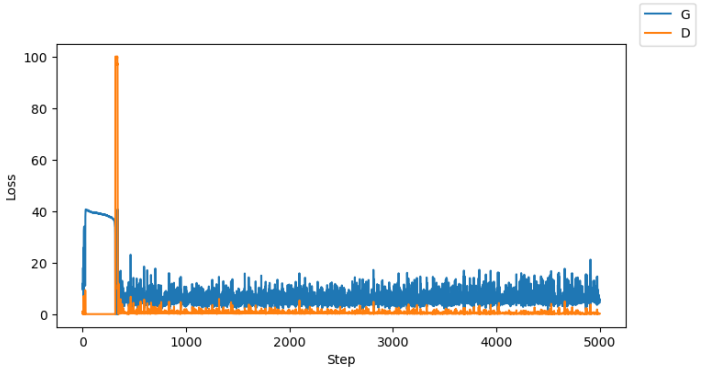
通過新增這些區塊，我們希望提高網絡生成和判別更複雜特徵的能力。

2. Experimental results

- Original



- 改動後



3. Conclusion

I. Loss function:

- Original

藍色曲線（G）代表生成器的損失。可以看到，損失值一開始非常高，隨著訓練的進行，有一定的波動，但總體呈現下降趨勢，並且在後期趨於穩定。

橙色曲線（D）代表判別器的損失。判別器的損失在整個訓練過程中也有波動，但相比生成器，波動較小，且相對穩定。

- 改動後

✓ 初期損失變化：

判別器損失 (橙色曲線 D)：在訓練初期，判別器的損失迅速增高，達到接近 100 的高峰。這表示在訓練初期，判別器很難區分生成器生成的假樣本和真樣本。

生成器損失 (藍色曲線 G)：生成器的損失在訓練初期也較高，但隨著訓練步驟的增加，損失迅速下降。

✓ 中期損失穩定：

判別器損失：在達到高峰後，判別器的損失迅速下降並趨於穩定，並保持在一個較低的水平。

生成器損失：生成器的損失在初期下降後，也趨於穩定，並在後續訓練中維持在一定範圍內波動。

後期損失趨於穩定：

✓ 判別器損失：在初期的劇烈變化之後，判別器的損失在整個訓練過程中保持穩定且較低的值。

✓ 生成器損失：生成器的損失在後期也保持穩定，並在一定範圍內波動，但波動幅度較小。

II. 圖片比較

- Original

- 改動後

由其中的一些圖片可以看出，用原本的架構作訓練後最終得到的圖片蠻多張圖兩隻眼睛會不一樣，感覺像是拼貼上去的，但是改動後有些許的改善。

但是比較模糊程度，可以看出原本的比較清楚，可能原因如下:

- ✓ 架構修改引入不合理的層數或參數配置，導致生成器和判別器之間的對抗不平衡。
- ✓ 學習率或其他超參數設置不當，影響了模型的訓練效果。
- ✓ 訓練時間不足，模型尚未完全收斂

但我認為差距其實沒有到特別明顯，如果想要有更大的差異，我們可能要對模型架構做更大的改動。

4. Discussion

在過去的經驗中，訓練 DCGAN 是一個極具挑戰性的過程。根據過往的經驗以及目前的實驗結果，單純調整模型架構往往不會帶來顯著的改進。即使對生成器和判別器進行了一些調整，模型的性能提升也非常有限，這讓人感到挫折。特別是在生成對抗網絡的訓練中，從生成器和判別器的損失曲線中難以直觀地看出模型是否有改善。

在做作業中遇到資源上的困難，DCGAN 的訓練不僅需要大量的時間和資源，還經常面臨模型崩潰的情況。生成的圖像質量很難用客觀的標準去衡量，通常結果看起來都不理想，甚至可以說是崩壞的。這使得評估模型改進效果變得更加困難。想要客觀探斷出結果的好壞，需要更多的研究和實驗來驗證其效果。經過這次的作業以及在訓練 DCGAN 的過程中，我深刻體會到找到合適的模型和技術來改進圖像生成質量是一個艱難且持續的探索過程。

在這次作業中，我學到了許多有關生成對抗網絡（GAN）尤其是 DCGAN 的知識。經過多次實驗，我深刻體會到僅僅通過調整模型架構難以顯著提升生成圖片的質量。這次經驗讓我認識到，為了獲得更好的結果，更先進的模型。這些學習和挑戰為我未來的研究奠定了寶貴的基礎。