Machine Learning 2020 Spring

Homework #11

b06902059 資工三 謝宜儒

1. **(2.5%) 訓練一個 model。**

   a. **(1%) 請描述你使用的 model（可以是 baseline model）。包含 generator 和 discriminator 的 model architecture、loss function、optimizer 參數、以及訓練 step 數（或是 epoch 數）。**

   在此使用的是 sample code 中的 baseline model 。

   - Generator :

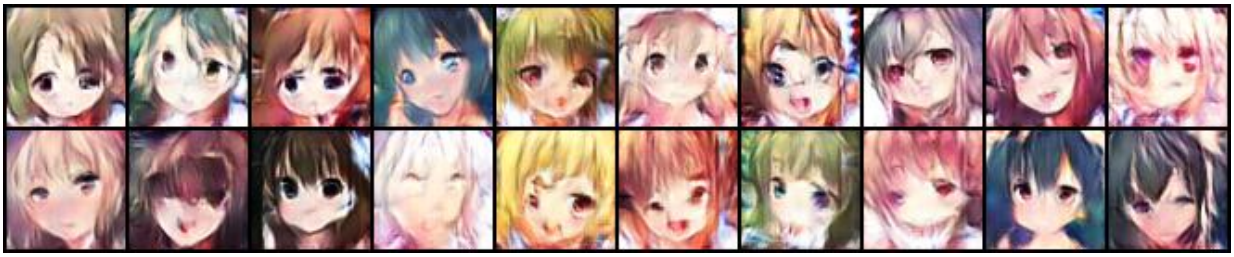
```
class Generator(nn.Module):
    def __init__(self, in_dim, dim = 64):
        super(Generator, self).__init__()
        def dconv_bn_relu(in_dim, out_dim):
            return nn.Sequential(
                nn.ConvTranspose2d(in_dim, out_dim, 5, 2,
                                   padding = 2, output_padding = 1,
                                   bias = False),
                nn.BatchNorm2d(out_dim),
                nn.ReLU())

        self.l1 = nn.Sequential(
            nn.Linear(in_dim, dim * 8 * 4 * 4, bias = False),
            nn.BatchNorm1d(dim * 8 * 4 * 4),
            nn.ReLU())

        self.l2_5 = nn.Sequential(
            dconv_bn_relu(dim * 8, dim * 4),
            dconv_bn_relu(dim * 4, dim * 2),
            dconv_bn_relu(dim * 2, dim),
            nn.ConvTranspose2d(dim, 3, 5, 2, padding = 2, output_padding
= 1),
            nn.Tanh())

        self.apply(weights_init)
    def forward(self, x):
        y = self.l1(x)
        y = y.view(y.size(0), -1, 4, 4)
        y = self.l2_5(y)
        return y
```

- Discriminator :

```
class Discriminator(nn.Module):
    def __init__(self, in_dim, dim=64):
        super(Discriminator, self).__init__()
        def conv_bn_lrelu(in_dim, out_dim):
            return nn.Sequential(
                nn.Conv2d(in_dim, out_dim, 5, 2, 2),
                nn.BatchNorm2d(out_dim),
                nn.LeakyReLU(0.2))

        self.ls = nn.Sequential(
            nn.Conv2d(in_dim, dim, 5, 2, 2), nn.LeakyReLU(0.2),
            conv_bn_lrelu(dim, dim * 2),
            conv_bn_lrelu(dim * 2, dim * 4),
            conv_bn_lrelu(dim * 4, dim * 8),
            nn.Conv2d(dim * 8, 1, 4),
            nn.Sigmoid())

        self.apply(weights_init)

    def forward(self, x):
        y = self.ls(x)
        y = y.view(-1)
        return y
```

- Loss Function : Binary Cross Entropy Loss (`nn.BCELoss()`)
- Optimizer :
  - Generator : Adam (lr = 1e-4, betas = (0.5, 0.999))
  - Discriminator : Adam (lr = 1e-4, betas = (0.5, 0.999))
- Number of Epochs : 10

b. **(1.5%) 請畫出至少 16 張 model 生成的圖片。**



2. **(3.5%) 請選擇下列其中一種 model： WGAN, WGAN-GP, LSGAN, SNGAN（不要和 1. 使用的 model 一樣，至少 architecture 或是 loss function 要不同）**

   a. (1%) 同 1.a ，請描述你選擇的 model，包含 generator 和 discriminator 的 model architecture、loss function、optimizer 參數、及訓練 step 數（或是 epoch 數）。

   我選擇的 model 是 SNGAN，主要更改的部分是將 discriminator 的各個 module 前加上 `torch.nn.utils.spectral_norm`。

   - Generator :

```
class Generator(nn.Module):
    def __init__(self, in_dim, dim = 64):
        super(Generator, self).__init__()
        def dconv_bn_relu(in_dim, out_dim):
            return nn.Sequential(
                nn.ConvTranspose2d(in_dim, out_dim, 5, 2,
                                   padding = 2, output_padding = 1,
                                   bias = False),
                nn.BatchNorm2d(out_dim),
                nn.ReLU())

        self.l1 = nn.Sequential(
            nn.Linear(in_dim, dim * 8 * 4 * 4, bias = False),
            nn.BatchNorm1d(dim * 8 * 4 * 4),
            nn.ReLU())

        self.l2_5 = nn.Sequential(
            dconv_bn_relu(dim * 8, dim * 4),
            dconv_bn_relu(dim * 4, dim * 2),
            dconv_bn_relu(dim * 2, dim),
            nn.ConvTranspose2d(dim, 3, 5, 2, padding = 2, output_padding
= 1),
            nn.Tanh())

        self.apply(weights_init)

    def forward(self, x):
        y = self.l1(x)
        y = y.view(y.size(0), -1, 4, 4)
        y = self.l2_5(y)
        return y
```

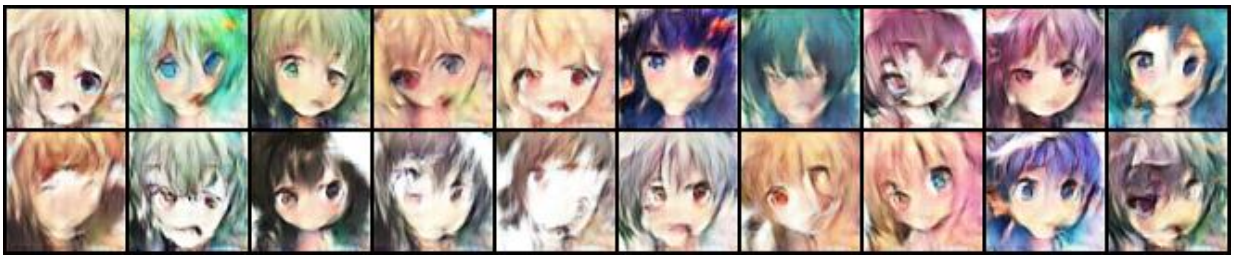
   - Discriminator :

```
class SNGANDiscriminator(nn.Module):
    def __init__(self, in_dim, dim = 64):
        super(SNGANDiscriminator, self).__init__()
        def conv_bn_lrelu(in_dim, out_dim):
            return nn.Sequential(
                spectral_norm(nn.Conv2d(in_dim, out_dim, 5, 2, 2)),
                nn.BatchNorm2d(out_dim),
                nn.LeakyReLU(0.2))
```

```
        self.ls = nn.Sequential(
            spectral_norm(nn.Conv2d(in_dim, dim, 5, 2, 2)),
            nn.LeakyReLU(0.2),
            conv_bn_lrelu(dim, dim * 2),
            conv_bn_lrelu(dim * 2, dim * 4),
            conv_bn_lrelu(dim * 4, dim * 8),
            spectral_norm(nn.Conv2d(dim * 8, 1, 4)),
            nn.Sigmoid())

        self.apply(weights_init)

    def forward(self, x):
        y = self.ls(x)
        y = y.view(-1)
        return y
```

- Loss Function : Binary Cross Entropy Loss (`nn.BCELoss()`)
- Optimizer :
  - Generator : Adam (lr = 1e-4, betas = (0.5, 0.999))
  - Discriminator : Adam (lr = 1e-4, betas = (0.5, 0.999))
- Number of Epochs : 10

b. **(1.5%) 和 1.b 一樣，就你選擇的 model，畫出至少 16 張 model 生成的圖片。**



c. **(1%) 請簡單探討你在 1. 使用的 model 和 2. 使用的 model，他們分別有何性質，描述你觀察到的異同。**

在 1. 使用的 model 為一般的 GAN，在 2. 使用的 model 是 SNGAN。

不同於一般的 GAN， SNGAN 使用了 spectral normalization 來做 weight normalization，此方法可以透過控制 Lipschitz constant 來限縮每一層的輸出，如此可以使得 discriminator 在 training 時可以更穩定。比較兩個 model 生成的圖片來比較，我認為生成的品質並沒有太大的差異，同樣都會有五官位移、混在一起的情形發生，也許要多 train 一些 epoch 才能看出差異。另外可以發現 1. 生成的圖片中許多髮色都是粉紅色、紅色，相較之下 2. 生成的圖片髮色則多為綠色、藍色、黃色，說明兩個 model 對 random sample 的 dimensions 有不同的解讀（ 兩個 model 是使用同樣的 Z sample 來生成圖片）。

Reference:

1. https://arxiv.org/pdf/1802.05957.pdf
2. https://xiaosean.github.io/deep%20learning/computer%20vision/2018-10-22-SNGAN/

3. **(4%) 請訓練一個會導致 mode collapse 的 model。**

   a. **(1%) 同 1.a ，請描述你選擇的 model，包含 generator 和 discriminator 的 model architecture、loss function、optimizer 參數、及訓練 step 數（或是 epoch 數）。**

   大部分都和 baseline model 一樣，唯一改動的地方是增加 epoch 的數量 (10 → 40)。

   - Generator :

```
class Generator(nn.Module):
    def __init__(self, in_dim, dim = 64):
        super(Generator, self).__init__()
        def dconv_bn_relu(in_dim, out_dim):
            return nn.Sequential(
                nn.ConvTranspose2d(in_dim, out_dim, 5, 2,
                                   padding = 2, output_padding = 1,
                                   bias = False),
                nn.BatchNorm2d(out_dim),
                nn.ReLU())

        self.l1 = nn.Sequential(
            nn.Linear(in_dim, dim * 8 * 4 * 4, bias = False),
            nn.BatchNorm1d(dim * 8 * 4 * 4),
            nn.ReLU())

        self.l2_5 = nn.Sequential(
            dconv_bn_relu(dim * 8, dim * 4),
            dconv_bn_relu(dim * 4, dim * 2),
            dconv_bn_relu(dim * 2, dim),
            nn.ConvTranspose2d(dim, 3, 5, 2, padding = 2, output_padding
= 1),
            nn.Tanh())

        self.apply(weights_init)

    def forward(self, x):
        y = self.l1(x)
        y = y.view(y.size(0), -1, 4, 4)
        y = self.l2_5(y)
        return y
```

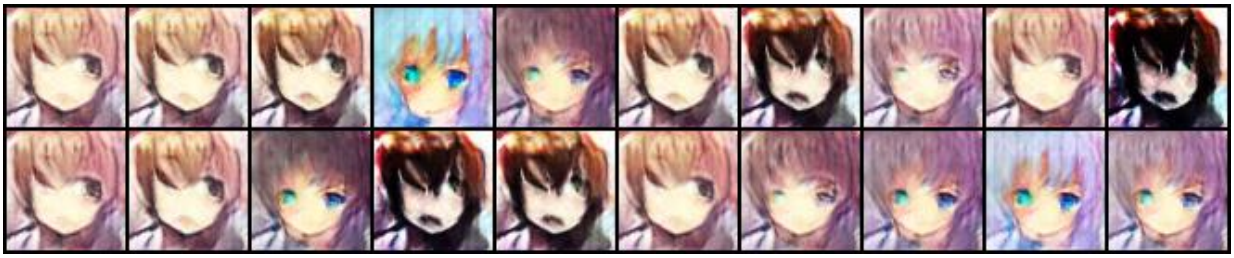
   - Discriminator :

```
class Discriminator(nn.Module):
    def __init__(self, in_dim, dim=64):
        super(Discriminator, self).__init__()
        def conv_bn_lrelu(in_dim, out_dim):
            return nn.Sequential(
                nn.Conv2d(in_dim, out_dim, 5, 2, 2),
                nn.BatchNorm2d(out_dim),
                nn.LeakyReLU(0.2))

        self.ls = nn.Sequential(
```

```
            nn.Conv2d(in_dim, dim, 5, 2, 2), nn.LeakyReLU(0.2),
            conv_bn_lrelu(dim, dim * 2),
            conv_bn_lrelu(dim * 2, dim * 4),
            conv_bn_lrelu(dim * 4, dim * 8),
            nn.Conv2d(dim * 8, 1, 4),
            nn.Sigmoid())

        self.apply(weights_init)

    def forward(self, x):
        y = self.ls(x)
        y = y.view(-1)
        return y
```

- Loss Function : Binary Cross Entropy Loss (`nn.BCELoss()`)
- Optimizer :
  - Generator : Adam (lr = 1e-4, betas = (0.5, 0.999))
  - Discriminator : Adam (lr = 1e-4, betas = (0.5, 0.999))
- Number of Epochs : **40**

b. **(1.5%) 請畫出至少16張 model 生成且具有 mode collapse 現象的圖片。**



可以看出每張圖片都是同樣的臉型、角度，不同的只有髮色與眼睛的顏色，尤其第 1、2 張圖片幾乎看不出差異，的確有 mode collapse 的現象。

c. **(1.5%) 在不改變 optimizer 和訓練 step 數的情況下，請嘗試使用一些方法來減緩 mode collapse。說明你嘗試了哪些方法，請至少舉出一種成功改善的方法，若有其它失敗的方法也可以記錄下來。**

首先我嘗試使用 SNGAN 的 model architectue，但並沒有觀察到 mode collapse 的減緩。接著，我再進一步在 SNGAN 的 model architecture 加入 dropout layer，使得最後 discriminator 的架構變為：

```
class SNGANDiscriminator(nn.Module):
    def __init__(self, in_dim, dim = 64):
        super(SNGANDiscriminator, self).__init__()
        def conv_bn_lrelu(in_dim, out_dim):
            return nn.Sequential(
                spectral_norm(nn.Conv2d(in_dim, out_dim, 5, 2, 2)),
                nn.BatchNorm2d(out_dim),
                nn.LeakyReLU(0.2))

        self.ls = nn.Sequential(
            spectral_norm(nn.Conv2d(in_dim, dim, 5, 2, 2)),
            nn.LeakyReLU(0.2),
            conv_bn_lrelu(dim, dim * 2),
            nn.Dropout(0.5),
```

```
            conv_bn_lrelu(dim * 2, dim * 4),
            nn.Dropout(0.5),
            conv_bn_lrelu(dim * 4, dim * 8),
            nn.Dropout(0.5),
            spectral_norm(nn.Conv2d(dim * 8, 1, 4)),
            nn.Sigmoid())

        self.apply(weights_init)

    def forward(self, x):
        y = self.ls(x)
        y = y.view(-1)
        return y
```

在這個架構下，產生的圖片如下圖，成功改善了 mode collapse 的現象，雖然還是有蠻多重複的情形，但生成的圖片已經不全都是同樣的角度和臉型。不過這樣的做法似乎也讓生成的圖片品質有所下降，圖片變得很模糊，也出現了類似黑影的部分，所以我認為仍有蠻大的改進空間。