IMVFX HW2-1 DCGAN Practice

1. Please provide a brief introduction about your experiments, including details such as setting of hyperparameter, data augmentation techniques used, network structure, etc.

(1) Hyperparameter

```
# Root directory for dataset
dataroot = "anime_face_dataset"
# Number of workers for dataloader
workers = 8
# Batch size during training
batch_size = 64
# Spatial size of training images. All images will be resized to this size using a transformer.
image_size = 64
# Number of channels in the training images. For color images this is 3
nc = 3
# Size of z latent vector (i.e. size of generator input)
nz = 100
# Size of feature maps in generator
ngf = 64
# Size of feature maps in discriminator
ndf = 64
# Number of training epochs
num_epochs = 200
# Learning rate for optimizers
lr = 0.0002
# Beta1 hyperparam for Adam optimizers
beta1 = 0.5
# Number of GPUs available. Use 0 for CPU mode.
ngpu = 1
# Decide which device we want to run on
device = torch.device("cuda:0" if (torch.cuda.is_available() and ngpu > 0) else "cpu")

# log(img) and checkpoints directory
log_dir = os.path.join(workspace_dir, 'logs')
ckpt_dir = os.path.join(workspace_dir, 'checkpoints')
os.makedirs(log_dir, exist_ok=True)
os.makedirs(ckpt_dir, exist_ok=True)
```

上圖是我的超參數設定，除了 worker 改成 8、batch_size 改成 64、num_epochs 改成 200 之外，其他參數都跟助教給的 sample code[1]一樣。

(2) Data augmentation

```
def get_dataset(dataroot):
    dataset = dset.ImageFolder(root=dataroot,
                               transform=transforms.Compose([
                                   #transforms.Resize(image_size),
                                   #transforms.CenterCrop(image_size),
                                   transforms.RandomHorizontalFlip(),
                                   transforms.ColorJitter(brightness=0.1, contrast=0.2, saturation=0.2, hue=0.05),
                                   transforms.RandomRotation(degrees=5),
                                   transforms.ToTensor(),
                                   transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
                               ]))
    return dataset
```

在 augmentation 的部分，我用了 RandomHorizontalFlip、ColorJitter、

Normalize，讓原本的 dataset 的圖片更多樣性，模型適應力更強性能更好，因為原本 dataset 的圖片就是 64*64，所以 Resize、CenterCrop 就沒用了。

(3) network structure

```
class Generator(nn.Module):
  """
  Input shape: (N, in_dim, 1, 1)
  Output shape: (N, nc, image_size, image_size)

  In our sample code, input/output shape are:
    Input shape: (N, 100, 1, 1)
    Output shape: (N, 3, 64, 64)
  """

  def __init__(self, ngpu):
    super(Generator, self).__init__()
    self.ngpu = ngpu
    self.main = 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
    )
```

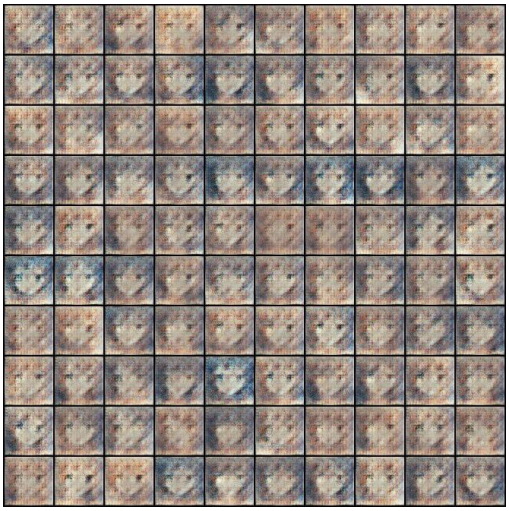
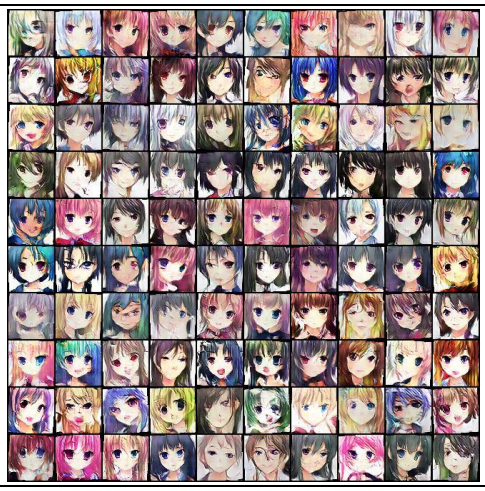
```
  def forward(self, input):
    return self.main(input)
```

```python
class Discriminator(nn.Module):
    """
    Input shape: (N, nc, image_size, image_size)
    Output shape: (N, )

    In our sample code, input/output are:
        Input shape: (N, 3, 64, 64)
        Output shape: (N, )
    """
    def __init__(self, ngpu):
        super(Discriminator, self).__init__()
        self.ngpu = ngpu
        self.main = 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.Sigmoid()
        )

    def forward(self, input):
        return self.main(input)
```

上面兩張圖是我 DCGAN 的 Generator 跟 Discriminator 的架構，跟助教給的 sample code[1]是完全一樣的，Generator 是用 ConvTranspose2d 再用 BatchNorm2d，activation function 前面幾層用 ReLU，最後一層用 Tanh，Discriminator 的部分則是用 Conv2d 再用 BatchNorm2d，activation function 前面幾層用 LeakyReLU，最後一層用 Sigmoid。

2. Place the generated image series from various epochs during the training process here and provide a discussion of your observations.

Epoch=1	Epoch=20

	
Epoch=40	Epoch=60
	
Epoch=80	Epoch=100
	
Epoch=120	Epoch=140

	
Epoch=160	Epoch=180
	
Epoch=200	
	

上面的表格是每隔 20 個 epoch 生成的圖，從上面的圖中可以看到，在 epoch=20 的時候人物臉蛋大部分已經成形了，但是雜訊還是很多，到 epoch=40 的時候雜訊就降低很多了，在 epoch=40 到 epoch=200 之間生成的圖片看起來都是微調圖片亮度、髮色、瞳色、眼鏡大小，讓圖片能夠更逼真，不過在訓練過程中，也有發現像 epoch=120 時，瞳色全部變成綠色這種意外狀況。

3. Please explain how the z vector influence your images here.

上圖是 Interpolation 的結果，從左到右代表 z1 占比逐漸變小 z2 占比逐漸變大，從上圖的第一列可以看出變化過程還算平滑，最左邊的部分，也就是全部是 z1 的時候，是粉色頭髮嘴巴開開的微笑，瞳孔青藍色，到中間，z1、z2 接近各佔一半的時候，可以看到髮色逐漸變紫，嘴巴也接近閉合，瞳孔也變得有點紫色，到最右邊，全部是 z2 的時候，可以看出髮色已經變成深藍色了，嘴巴也閉合著微笑，眼睛也變成紫色跟紅色了。

Reference

[1] Sample Code :IMVFX_HW2_DCGAN.ipynb - Colaboratory (google.com)

[2] https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html