CycleGAN 구현

이다해

데이터 셋: ImageNet (horse2zebra)

Train Data: 2,401, Val Data: 260

Domain A (e.g., Horses)

Domain B (e.g., Zebras)

1. Custom Dataset

```python
class CycleGANDataset(Dataset):
    def __init__(self, root_dir_A, root_dir_B, transform=None): #전체 데이에 관한거 데이터 최상위 root

        self.root_dir_A = root_dir_A
        self.root_dir_B = root_dir_B
        self.transform = transform

        image_extensions = ('.jpg', '.jpeg', '.png', '.gif', '.bmp', '.tiff')

        self.files_A = [os.path.join(root_dir_A, f) for f in os.listdir(root_dir_A)
                        if os.path.isfile(os.path.join(root_dir_A, f)) and f.lower().endswith(image_extensions)]

        self.files_B = [os.path.join(root_dir_B, f) for f in os.listdir(root_dir_B)
                        if os.path.isfile(os.path.join(root_dir_B, f)) and f.lower().endswith(image_extensions)]

        self.length = max(len(self.files_A), len(self.files_B))


    def __getitem__(self, index): #하나의 데이터 호출마다 실행

        path_A = self.files_A[index % len(self.files_A)]
        img_A = Image.open(path_A)

        path_B = self.files_B[random.randint(0, len(self.files_B) - 1)]
        img_B = Image.open(path_B)

        if img_A.mode != "RGB":
            img_A = to_rgb(img_A)
        if img_B.mode != "RGB":
            img_B = to_rgb(img_B)

        if self.transform is not None:
            img_A = self.transform(img_A)
            img_B = self.transform(img_B)

        return {"A": img_A, "B": img_B}

    def __len__(self):
        return self.length
```

Custom Dataset은 기존의 코드들과 같게 구성, Unpair 쌍이므로 쌍을 맞춰 입력할 필요가 없어 random으로 배정

2. Model

2-1. Generator

	구조	Output shape
C7S1-64	7×7 Convolution + InstanceNorm + ReLU, Stride=1	256×256×64
d128	3×3 Convolution + InstanceNorm + ReLU, Stride=2	128×128×128
d256	3×3 Convolution + InstanceNorm + ReLU, Stride=2	64×64×256
R256 x 9	Residual Block	64×64×256
u128	Fractional-strided Conv + InstanceNorm + ReLU, Stride=2	128×128×128
u64	Fractional-strided Conv + InstanceNorm + ReLU, Stride=2	256×256×64
C7S1-3	Fractional-strided Conv + InstanceNorm + ReLU, Stride=2	256×256×3

1) C7S1-64
   - 입력 이미지에서 특징 추출

```
class Generator(nn.Module):
    def __init__(self, nch_in, nch_out):
        super(Generator, self).__init__()

        model = [
        #C7S1-64
        nn.ReflectionPad2d(3),
        nn.Conv2d(3, 64 ,7 , stride=1, padding=1),
        nn.InstanceNorm2d(64),
        nn.ReLU(),
```

2) d128, d256
   - 해상도를 절반으로 줄이면서 채널을 늘림

```
#d128
nn.Conv2d(64, 128 ,3 , stride=2, padding=1),
nn.InstanceNorm2d(128),
nn.ReLU(),

#d256
nn.Conv2d(128, 256 ,3 , stride=2, padding=1),
nn.InstanceNorm2d(256),
nn.ReLU()
]
```

3) R256 x 9 / Residual Block (Skip Connection)

- 깊은 특성 학습, 정보 유지
- Residual Block은 두개의 Conv 층으로 이루어져 있으며 여러 층을 거친 출력과 입력을 더함
- 논문에서 9개의 Residual Block으로 구성

```
class Residual_Block(nn.Module):
    def __init__(self, channels):
        super(Residual_Block, self).__init__()

        self.block = nn.Sequential(
            nn.ReflectionPad2d(1),
            nn.Conv2d(channels, channels, kernel_size=3),
            nn.InstanceNorm2d(channels),
            nn.ReLU(inplace=True),

            nn.ReflectionPad2d(1),
            nn.Conv2d(channels, channels, kernel_size=3),
            nn.InstanceNorm2d(channels)
        )

    def forward(self, x):
        return x + self.block(x)  # Skip connection
```

```
for i in range(9):
    model += [Residual_Block(256)]
```

4) u128, u64, C7S1-3

- Downsampling 해상도 늘리면서 채널 수 줄임

```
model += [
#u128
nn.ConvTranspose2d(256,128,3,stride=2,padding=1,output_padding=1),
nn.InstanceNorm2d(128),
nn.ReLU(),

#u64
nn.ConvTranspose2d(128,64,3,stride=2,padding=1,output_padding=1),
nn.InstanceNorm2d(64),
nn.ReLU(),
```

```
nn.Conv2d(64, 3 ,7 , stride=1,padding=1),
nn.InstanceNorm2d(3),
nn.Tanh()
]
```

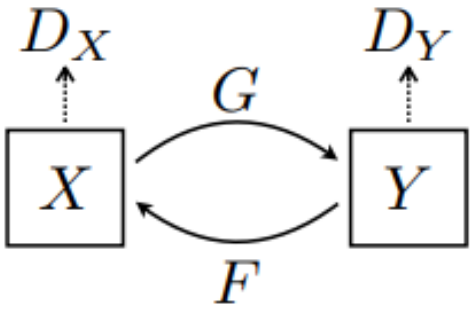
2-2. Discriminator

PatchGAN으로 이루어져 있음, 이미지를 Patch단위로 나누어 이미지 판별

	구조	Output shape
C64	4×4 Convolution + LeakyReLU (slope=0.2), Kernel=3, Stride=1	128×128×6
C128	4×4 Convolution + InstanceNorm + LeakyReLU (slope=0.2), Stride=1	64×64×128
C256	4×4 Convolution + InstanceNorm + LeakyReLU (slope=0.2), Stride=1	32×32×256
C512	4×4 Convolution + InstanceNorm + LeakyReLU (slope=0.2), Stride=1	31×31×512
Conv1	Kernel=4, stride=1, padding=1	30×30×1

```python
class Discriminator(nn.Module):
    def __init__(self, nch_in):
        super(Discriminator, self).__init__()

        model = [

            #C64
            nn.Conv2d(3, 64 , 4 , stride=2),
            nn.LeakyReLU(0.2),


            #C128
            nn.Conv2d(64, 128 , 4 , stride=2),
            nn.InstanceNorm2d(128),
            nn.LeakyReLU(0.2),

            #C256
            nn.Conv2d(128, 256 , 4 , stride=2),
            nn.InstanceNorm2d(256),
            nn.LeakyReLU(0.2),

            #C512
            nn.Conv2d(256, 512 , 4 , stride=2),
            nn.InstanceNorm2d(512),
            nn.LeakyReLU(0.2),
            nn.Conv2d(512, 1, 4, stride=1)

        ]

        self.model = nn.Sequential(*model) #하나로 묶는 역할


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

3-1. Generator Loss



CycleGAN 의 목표는 Unpaired Data 에서 스타일 전이가 잘 되도록 하는 것이고. 이를 위해 Cycle Consistency Loss 와 GAN Loss 로 구성됨. X 에서 Y 로, Y 에서 X 로 모두 잘 학습되게 해야 하니까 Generator 는 2 개, Discriminator 는 2 개로 구성됨. 코드도 이에 맞게 작성되고, X 에서 Y 로 변환하는 Generator 는 G, Discriminator 는 D_X, Y 에서 X 로 변환하는 Generator 는 F, Discriminator 는 D_Y 로 설정됨.

```
fake_B = G(image_A)
fake_A = F(image_B)

pred_fake_B = D_X(fake_B)
pred_fake_A = D_Y(fake_A)

loss_GAN_AB = criterion_GAN(pred_fake_B,  torch.ones_like(pred_fake_B))
loss_GAN_BA = criterion_GAN(pred_fake_A, torch.ones_like(pred_fake_A))
loss_GAN = (loss_GAN_AB + loss_GAN_BA) / 2
```

Generator 는 생성한 가짜 이미지가 진짜처럼 판단되도록 학습하는 것이 목적

이를 위해 pred_fake_B 와 torch.ones_like(pred_fake_B)를 사용하여 Loss 를 최소화함.

그러나 생성된 이미지가 스타일 전이만 이루어지고 원본과 차이가 있을 경우, Discriminator 는 이를 진짜라고 판단하여 학습이 종료될 수 있음. 이로 인해 원본과 다르게 변형된 이미지가 생성되는 문제가 발생할 수 있어 이를 해결하기 위해 저자는 Consistency Loss 를 추가하였습니다. 즉, 생성된 이미지를 다시 원래 도메인으로 변환해도 동일한 이미지가 나와야 하며, L1 norm 을 사용하여 원본 이미지와의 차이를 최소화함.

$$\mathcal{L}_{\text{cyc}}(G,F) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y\sim p_{\text{data}}(y)}[\|G(F(y)) - y\|_1].$$

```
recov_A = F(fake_B)
recov_B = G(fake_A)

loss_cyc_A = consistency(recov_A, image_A)
loss_cyc_B = consistency(recov_B, image_B)
loss_cycle = (loss_cyc_A + loss_cyc_B) / 2
```

3-2. Discriminator Loss

Discriminator의 목적은 생성된 가짜 이미지를 가짜로 판단하고, 진짜 이미지를 진짜로 판단하는 것입니다. 이를 위해 다음과 같은 방식으로 Loss를 계산

loss_D_X = (criterion_GAN(pred_real_B, torch.ones_like(pred_fake_B)) +

criterion_GAN(pred_fake_B, torch.zeros_like(pred_fake_B))) * 0.5

진짜 이미지는 1 과 비교하여 Loss 를 최소화하고, 가짜 이미지는 0 과 비교하여 Loss 를 최소화합니다. 이를 통해 각각의 이미지를 별도로 학습하게 됨.

```
optimizer_D_Y.zero_grad()
pred_real_A = D_Y(image_A)
pred_fake_A = D_Y(fake_A.detach())
loss_D_Y = (criterion_GAN(pred_real_A, torch.ones_like(pred_fake_A)) +
            criterion_GAN(pred_fake_A, torch.zeros_like(pred_fake_A))) * 0.5

D_Y_epoch_loss += loss_D_Y.item()
num_batches += 1

loss_D_Y.backward()
optimizer_D_Y.step()
```

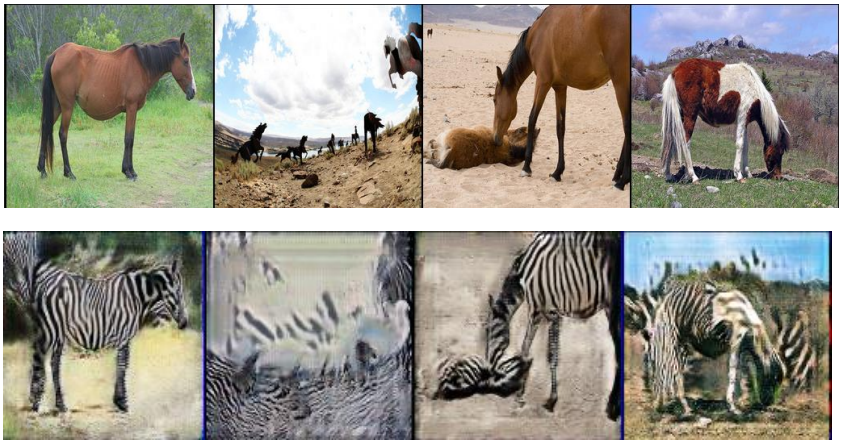
```
optimizer_D_X.zero_grad()
pred_real_B = D_X(image_B)
pred_fake_B = D_X(fake_B.detach())
loss_D_X = (criterion_GAN(pred_real_B, torch.ones_like(pred_fake_B)) +
            criterion_GAN(pred_fake_B, torch.zeros_like(pred_fake_B))) * 0.5

D_X_epoch_loss += loss_D_X.item()
num_batches += 1


loss_D_X.backward()
optimizer_D_X.step()
```

3-3. 학습 시 설정

Optimizer: Adam, lr: 0.0002 (100 epoch 이후에 lr 감소), epoch: 200

4. 실험 결과

말 -> 얼룩말



말 -> 얼룩말

물체가 작거나 배경까지 스타일 전이가 되는 경우가 있음. 또한 '말'에서 '얼룩말'로 변환하는 경우는 잘 진행되지만, 반대로 '얼룩말'에서 '말'로 변환하는 경우 성능이 떨어지는 이미지가 많음. 이는 '말'의 경우 데이터셋이 백마 등으로 다양해 스타일 전이가 완벽하게 이루어지지 않은 것으로 보임.