CycleGAN

Generator Discriminator







2021.11.10

김민준



CONTENTS

CHAPTER 1

CHAPTER 2

- 전처리

- Generator

CHAPTER 3

- Discriminator

전처리

CHAPTER 1

```
1 from PIL import Image
2
3 def to_rgb(image):
4    rgb_image = Image.new("RGB", image.size)
5    rgb_image.paste(image)
6    return rgb_image
```

Syntax:

PIL.Image.new(mode, size)

PIL.Image.new(mode, size, color)

Parameters:

mode: The mode to use for the new image. (It could be RGB, RGBA)

size: A 2-tuple containing (width, height) in pixels.

color: What color to use for the image. Default is black. If given, this should be a single integer or floating poin

t value for single-band modes, and a tuple for multi-band modes.

Return Value: An Image object.

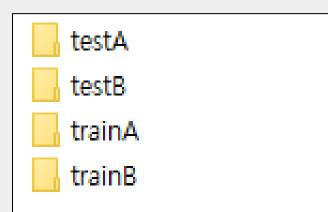
```
1 class ImageDataset(Dataset):
 2
       def __init__(self, root, transforms_=None, unaligned=False, mode="train"):
           self.transform = transforms.Compose(transforms_)
 3
          self.unaligned = unaligned
 4
          if mode=="train":
 5
 6
               self.files_A = sorted(glob.glob(os.path.join(root, "trainA") + "/*.*"))
 7
               self.files_B = sorted(glob.glob(os.path.join(root, "trainB") + "/*.*"))
 8
           else:
 9
               self.files A = sorted(glob.glob(os.path.join(root, "testA") + "/*.*"))
10
               self.files_B = sorted(glob.glob(os.path.join(root, "testB") + "/*.*"))
11
12
       def __getitem__(self, index):
13
           image_A = Image.open(self.files_A[index % len(self.files_A)])
14
           if self.unaligned:
               image_B = Image.open(self.files_B[random.randint(0, len(self.files_B) - 1)])
15
          else:
               image B = Image.open(self.files_B[index % len(self.files_B)])
16
17
           if image_A.mode != "RGB":
               image A = to rgb(image A)
18
           if image B.mode != "RGB":
19
               image_B = to_rgb(image_B)
20
21
22
           item A = self.transform(image A)
           item B = self.transform(image B)
23
24
           return {"A": item A, "B": item B}
25
       def __len__(self):
26
27
           return max(len(self.files_A), len(self.files_B))
```

```
1 class ImageDataset(Dataset):
       def __init__(self, root, transforms =None, unaligned=False, mode="train"):
 2
           self.transform = transforms.Compose(transforms )
 3
          self.unaligned = unaligned
 4
 5
           if mode=="train":
 6
               self.files A = sorted(glob.glob(os.path.join(root, "trainA") + "/*.*"))
 7
               self.files B = sorted(glob.glob(os.path.join(root, "trainB") + "/*.*"))
 8
           else:
 9
               self.files A = sorted(glob.glob(os.path.join(root, "testA") + "/*.*"))
10
               self.files_B = sorted(glob.glob(os.path.join(root, "testB") + "/*.*"))
11
12
       def getitem (self, index):
13
           image_A = Image.open(self.files_A[index % len(self.files_A)])
           if self.unaligned:
14
               image_B = Image.open(self.files_B[random.randint(0, len(self.files_B) - 1)])
15
          else:
               image_B = Image.open(self.files_B[index % len(self.files_B)])
16
17
           if image_A.mode != "RGB":
               image A = to rgb(image A)
18
           if image B.mode != "RGB":
19
               image_B = to_rgb(image_B)
20
21
22
           item A = self.transform(image A)
           item B = self.transform(image B)
23
           return {"A": item A, "B": item B}
24
25
       def __len__(self):
26
27
           return max(len(self.files A), len(self.files B))
```

전처리

CHAPTER 1

```
1 class ImageDataset(Dataset):
       def __init__(self, root, transforms_=None, unaligned=False, mode="train"):
           self.transform = transforms.Compose(transforms_)
 3
 4
           self.unaligned = unaligned
           if mode=="train":
               self.files_A = sorted(glob.glob(os.path.join(root, "trainA") + "/*.*"))
 6
               self.files_B = sorted(glob.glob(os.path.join(root, "trainB") + "/*.*"))
          else:
 8
               self.files_A = sorted(glob.glob(os.path.join(root, "testA") + "/*.*"))
               self.files_B = sorted(glob.glob(os.path.join(root, "testB") + "/*.*"))
10
```





*: 임의 길이의 모든 문자열

→ 폴더 내의 모든 항목

```
1 class ImageDataset(Dataset):
       def __getitem__(self, index):.
           image_A = Image.open(self.files_A[index % len(self.files_A)])
 3
           if self.unaligned:
 4
               image_B = Image.open(self.files_B[random.randint(0, len(self.files_B) - 1)])
           else:
 6
               image_B = Image.open(self.files_B[index % len(self.files_B)])
 7
 8
          # Convert grayscale images to rgb
 9
           if image_A.mode != "RGB":
10
11
               image_A = to_rgb(image_A)
           if image_B.mode != "RGB":
12
13
               image_B = to_rgb(image_B)
14
15
           item_A = self.transform(image_A)
16
           item_B = self.transform(image_B)
17
           return {"A": item_A, "B": item_B}
18
       def __len__(self):
19
20
           return max(len(self.files_A), len(self.files_B))
```

전처리

CHAPTER 1

```
1 class ImageDataset(Dataset):
       def __getitem__(self, index):.
           image_A = Image.open(self.files_A[index % len(self.files_A)])
 3
           if self.unaligned:
               image_B = Image.open(self.files_B[random.randint(0, len(self.files_B) - 1)])
           else:
 6
               image_B = Image.open(self.files_B[index % len(self.files_B)])
 9
          # Convert grayscale images to rgb
          if image_A.mode != "RGB":
10
11
               image_A = to_rgb(image_A)
           if image_B.mode != "RGB":
12
               image_B = to_rgb(image_B)
13
14
15
          item_A = self.transform(image_A)
           item_B = self.transform(image_B)
16
17
           return {"A": item_A, "B": item_B}
18
       def __len__(self):
19
20
           return max(len(self.files_A), len(self.files_B))
```

Unaligned 변수를 통해 학습할 쌍을 무작위로 고를지, 고정시킬지 정함

전처리

```
1 class ImageDataset(Dataset):
      def __getitem__(self, index):.
          image_A = Image.open(self.files_A[index % len(self.files_A)])
          if self.unaligned:
              image_B = Image.open(self.files_B[random.randint(0, len(self.files_B) - 1)])
          else:
 6
              image_B = Image.open(self.files_B[index % len(self.files_B)])
 8
          # Convert grayscale images to rgb
 9
          if image_A.mode != "RGB":
10
                                                  RGB가 아니면 RGB로 변환
              image_A = to_rgb(image_A)
11
          if image_B.mode != "RGB":
12
              image_B = to_rgb(image_B)
13
14
15
          item_A = self.transform(image_A)
                                                  PIL image를 pytorch tensor로 변환
          item_B = self.transform(image_B)
16
          return {"A": item_A, "B": item_B}
17
18
      def __len__(self):
19
20
          return max(len(self.files_A), len(self.files_B))
```

Generator 구현 - initialize weight

```
1 def weights_init_normal(m):
2    classname = m.__class__.__name__
3    if classname.find("Conv") != -1:
4         torch.nn.init.normal_(m.weight.data, 0.0, 0.02)
5         if hasattr(m, "bias") and m.bias is not None:
6             torch.nn.init.constant_(m.bias.data, 0.0)
7    elif classname.find("BatchNorm2d") != -1:
8         torch.nn.init.normal_(m.weight.data, 1.0, 0.02)
9         torch.nn.init.constant_(m.bias.data, 0.0)
```

Layer의 종류에 따라 다른 가중치 초기화

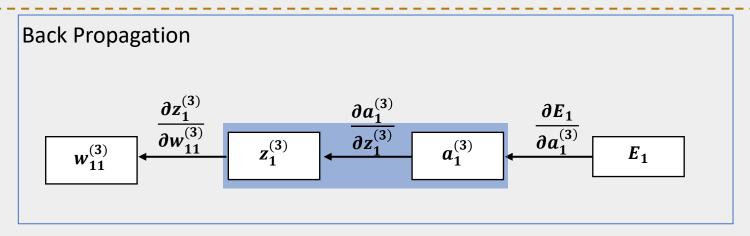
Generator 구현 – Residual Block

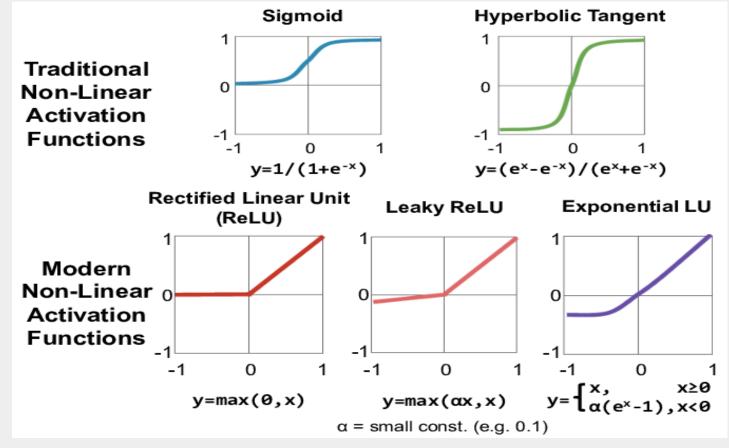
CHAPTER 2

```
1 class ResidualBlock(nn.Module):
       def __init__(self, in_features):
 3
           super(ResidualBlock, self).__init__()
 4
           self.block = nn.Sequential(
 5
               nn.ReflectionPad2d(1),
 6
               nn.Conv2d(in_features, in_features, 3),
               nn.InstanceNorm2d(in_features),
 8
               nn.ReLU(inplace=True),
 9
               nn.ReflectionPad2d(1),
10
               nn.Conv2d(in_features, in_features, 3),
11
               nn.InstanceNorm2d(in_features),
12
13
14
      def forward(self, x):
15
16
           return x + self.block(x)
```

이전 Layer와 현재 Layer의 출력값을 더해서 Forward > Gradient vanishing 해결

Gradient vanishing?





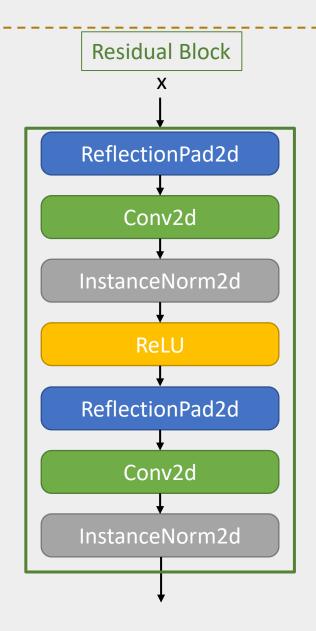
CHAPTER 2

```
1 class ResidualBlock(nn.Module):
      def init (self, in features):
 3
           super(ResidualBlock, self). init ()
           self.block = nn.Sequential(
 6
               nn.ReflectionPad2d(1),
               nn.Conv2d(in features, in features, 3),
               nn.InstanceNorm2d(in_features),
 8
               nn.ReLU(inplace=True),
               nn.ReflectionPad2d(1),
10
               nn.Conv2d(in_features, in_features, 3),
11
12
               nn.InstanceNorm2d(in_features),
13
14
      def forward(self, x):
15
16
           return x + self.block(x)
```

가장 가까운 픽셀 값을 복사 후 padding에 사용

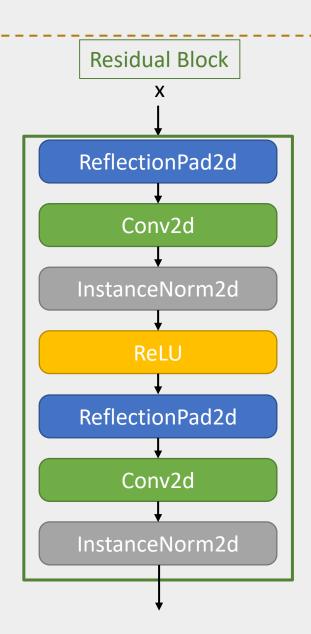
→ Zero padding보다 더욱 자연스러운 이미지 생성

```
1 class ResidualBlock(nn.Module):
      def __init__(self, in_features):
           super(ResidualBlock, self).__init__()
          self.block = nn.Sequential(
               nn.ReflectionPad2d(1),
               nn.Conv2d(in_features, in_features, 3),
               nn.InstanceNorm2d(in_features),
               nn.ReLU(inplace=True),
               nn.ReflectionPad2d(1),
10
               nn.Conv2d(in_features, in_features, 3),
11
12
               nn.InstanceNorm2d(in_features),
13
14
      def forward(self, x):
15
16
          return x + self.block(x)
```



CHAPTER 2

```
1 class ResidualBlock(nn.Module):
      def __init__(self, in_features):
           super(ResidualBlock, self).__init__()
           self.block = nn.Sequential(
               nn.ReflectionPad2d(1),
               nn.Conv2d(in_features, in_features, 3),
               nn.InstanceNorm2d(in_features),
 9
               nn.ReLU(inplace=True),
10
               nn.ReflectionPad2d(1),
               nn.Conv2d(in_features, in_features, 3),
11
12
               nn.InstanceNorm2d(in_features),
13
14
      def forward(self, x):
15
16
           return x + self.block(x)
```

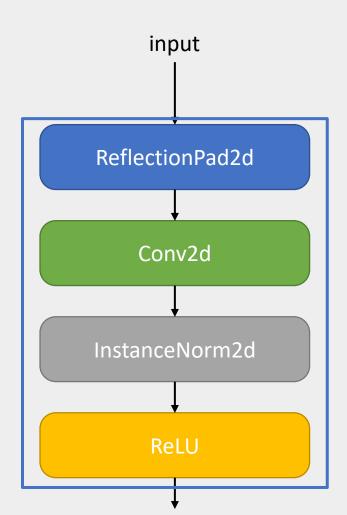


이미지에 특화된 정규화 방법으로 개별 이미지를 정규화

```
. . .
 1 class GeneratorResNet(nn.Module):
       def __init__(self, input_shape, num_residual_blocks):
 3
           super(GeneratorResNet, self).__init__()
 4
           channels = input shape[0]
 5
           # Initial convolution block
 6
          out_features = 64
          model = [
 8
               nn.ReflectionPad2d(channels),
 9
               nn.Conv2d(channels, out_features, 7),
10
               nn.InstanceNorm2d(out_features),
11
               nn.ReLU(inplace=True),
12
13
           in_features = out_features
14
           # Downsampling
           for _ in range(2):
15
16
               out_features *= 2
17
               model += [
18
                  nn.Conv2d(in_features, out_features, 3, stride=2, padding=1),
19
                  nn.InstanceNorm2d(out_features),
20
                  nn.ReLU(inplace=True),
21
22
               in_features = out_features
23
           # Residual blocks
24
           for _ in range(num_residual_blocks):
25
               model += [ResidualBlock(out_features)]
26
           # Upsampling
27
           for _ in range(2):
28
               out_features //= 2
29
               model += [
30
                   nn.Upsample(scale_factor=2),
31
                  nn.Conv2d(in_features, out_features, 3, stride=1, padding=1),
32
                  nn.InstanceNorm2d(out_features),
33
                  nn.ReLU(inplace=True),
34
35
               in_features = out_features
36
          # Output layer
37
          model += [nn.ReflectionPad2d(channels), nn.Conv2d(out features, channels, 7), nn.Tanh()]
38
           self.model = nn.Sequential(*model)
39
       def forward(self, x):
40
           return self.model(x)
```

CHAPTER 2

```
1 class GeneratorResNet(nn.Module):
      def __init__(self, input_shape, num_residual_blocks):
           super(GeneratorResNet, self).__init__()
 3
           channels = input_shape[0]
 6
           # Initial convolution block
           out_features = 64
8
9
           model = [
10
               nn.ReflectionPad2d(channels),
               nn.Conv2d(channels, out_features, 7),
11
12
               nn.InstanceNorm2d(out_features),
13
               nn.ReLU(inplace=True),
14
           in_features = out_features
15
```



초기 convolution block 선언

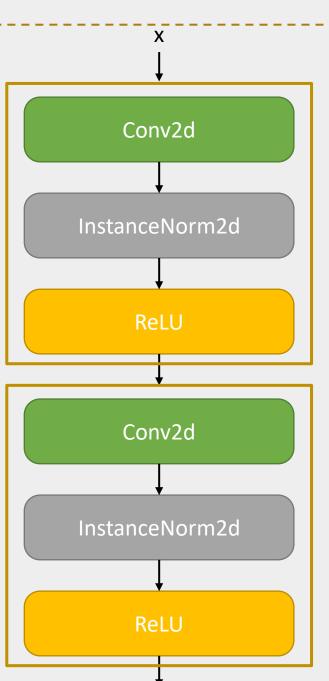
CHAPTER 2

```
# Downsampling
16
17
           for _ in range(2):
               out_features *= 2
18
               model += [
19
                   nn.Conv2d(in_features, out_features, 3, stride=2, padding=1),
20
21
                   nn.InstanceNorm2d(out_features),
22
                   nn.ReLU(inplace=True),
23
24
               in features = out features
25
26
          # Residual blocks
          for _ in range(num_residual_blocks):
27
               model += [ResidualBlock(out features)]
28
```

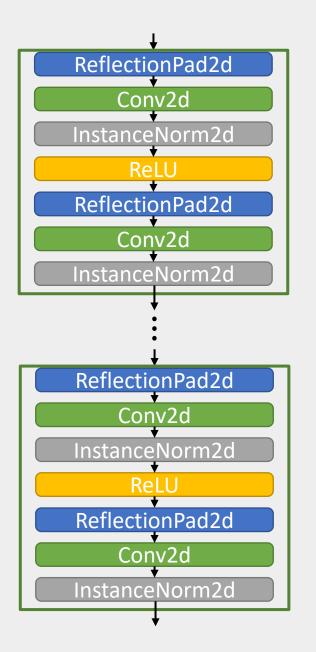
Down sampling 2회, stride = 2
$$\rightarrow$$
 크기 $\frac{1}{4}$ 배

Downsampling은 input image의 특징을 추출

```
16
           # Downsampling
           for _ in range(2):
17
               out_features *= 2
18
               model += [
19
                   nn.Conv2d(in_features, out_features, 3, stride=2, padding=1),
20
                   nn.InstanceNorm2d(out_features),
21
                   nn.ReLU(inplace=True),
22
23
24
               in_features = out_features
25
26
           # Residual blocks
           for _ in range(num_residual_blocks):
27
               model += [ResidualBlock(out_features)]
28
```



```
# Downsampling
16
           for _ in range(2):
17
               out_features *= 2
18
               model += [
19
20
                   nn.Conv2d(in_features, out_features, 3, stride=2, padding=1),
                   nn.InstanceNorm2d(out features),
21
22
                   nn.ReLU(inplace=True),
23
24
               in_features = out_features
25
26
           # Residual blocks
           for _ in range(num_residual_blocks):
27
               model += [ResidualBlock(out features)]
28
```



CHAPTER 2

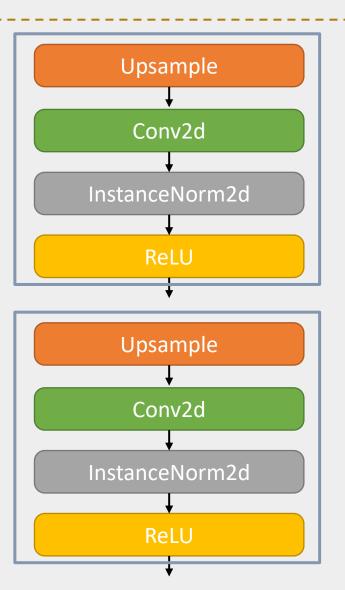
```
29
          # Upsampling
          for _ in range(2):
30
               out_features //= 2
31
               model += [
32
33
                   nn.Upsample(scale_factor=2),
                   nn.Conv2d(in_features, out_features, 3, stride=1, padding=1),
34
                   nn.InstanceNorm2d(out_features),
35
                   nn.ReLU(inplace=True),
36
37
               in_features = out_features
38
39
          # Output layer
40
          model += [nn.ReflectionPad2d(channels), nn.Conv2d(out_features, channels, 7), nn.Tanh()]
41
42
           self.model = nn.Sequential(*model)
43
44
45
       def forward(self, x):
           return self.model(x)
46
```

Up Sampling 2회 → 크기 4배

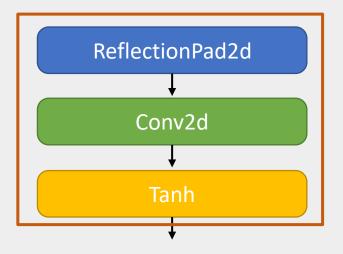
Upsampling을 통해 이미지의 스타일을 바꿔(translation)주는 용도로 사용

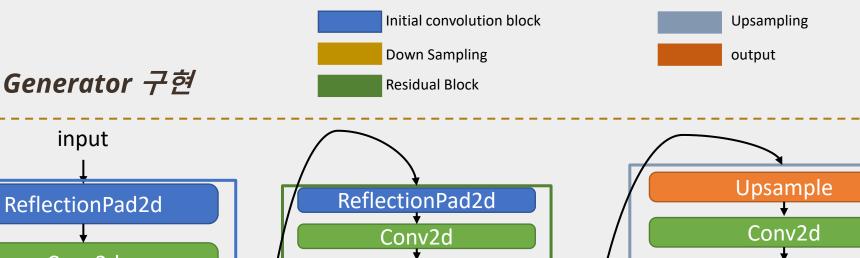
```
# Upsampling
29
          for _ in range(2):
30
31
              out_features //= 2
              model += [
32
33
                  nn.Upsample(scale_factor=2),
34
                  nn.Conv2d(in_features, out_features, 3, stride=1, padding=1),
                  nn.InstanceNorm2d(out_features),
35
                  nn.ReLU(inplace=True),
36
37
              in_features = out_features
38
39
          # Output layer
40
          model += [nn.ReflectionPad2d(channels), nn.Conv2d(out_features, channels, 7), nn.Tanh()]
41
42
           self.model = nn.Sequential(*model)
43
44
      def forward(self, x):
45
          return self.model(x)
46
```

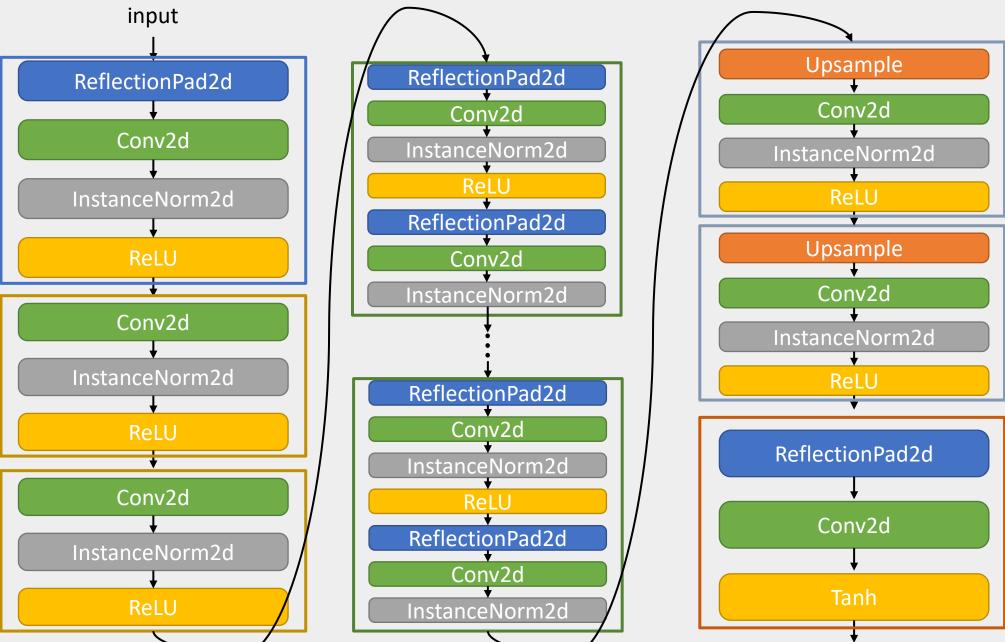
```
# Upsampling
29
          for _ in range(2):
30
              out_features //= 2
31
              model += [
32
                  nn.Upsample(scale_factor=2),
33
34
                  nn.Conv2d(in_features, out_features, 3, stride=1, padding=1),
                  nn.InstanceNorm2d(out_features),
35
                  nn.ReLU(inplace=True),
36
37
              in_features = out_features
38
39
          # Output layer
40
          model += [nn.ReflectionPad2d(channels), nn.Conv2d(out_features, channels, 7), nn.Tanh()]
41
42
          self.model = nn.Sequential(*model)
43
44
      def forward(self, x):
45
           return self.model(x)
46
```



```
# Upsampling
29
          for _ in range(2):
30
              out_features //= 2
31
              model += [
32
                  nn.Upsample(scale_factor=2),
33
                  nn.Conv2d(in_features, out_features, 3, stride=1, padding=1),
34
                  nn.InstanceNorm2d(out_features),
35
                  nn.ReLU(inplace=True),
36
37
              in_features = out_features
38
39
          # Output layer
40
          model += [nn.ReflectionPad2d(channels), nn.Conv2d(out_features, channels, 7), nn.Tanh()]
41
42
          self.model = nn.Sequential(*model)
43
      def forward(self, x):
45
           return self.model(x)
46
```







CHAPTER 3

Discriminator : 실제 이미지인지 (Generator에 의해 생성된)가짜 이미지인지 판별

CycleGAN의 Discriminator : PatchGAN의 Discriminator 기반
→ 이미지 패치 영역에 대하여 생성된 이미지가 진짜인지 가짜인지 판별

PatchGAN의 Discriminator 특징

- → 패치 영역을 따로 판단하여 원하는 스타일의 변환을 학습
- → 전체 이미지가 아닌 작은 이미지 패치에 대하여 연산을 수행, Parameter 수 감소

```
1 class Discriminator(nn.Module):
       def __init__(self, input_shape):
 2
 3
           super(Discriminator, self).__init__()
 4
           channels, height, width = input shape
 5
 6
           # Calculate output shape of image discriminator (PatchGAN)
 7
           self.output_shape = (1, height // 2 ** 4, width // 2 ** 4)
 8
 9
           def discriminator_block(in_filters, out_filters, normalize=True):
10
               """Returns downsampling layers of each discriminator block"""
11
               layers = [nn.Conv2d(in filters, out filters, 4, stride=2, padding=1)]
12
               if normalize:
13
14
                   layers.append(nn.InstanceNorm2d(out filters))
15
               layers.append(nn.LeakyReLU(0.2, inplace=True))
16
               return layers
17
18
           self.model = nn.Sequential(
19
               *discriminator_block(channels, 64, normalize=False),
               *discriminator_block(64, 128),
20
21
               *discriminator_block(128, 256),
               *discriminator_block(256, 512),
22
               nn.ZeroPad2d((1, 0, 1, 0)),
23
               nn.Conv2d(512, 1, 4, padding=1)
24
25
           )
26
27
       def forward(self, img):
           return self.model(img)
28
```

CHAPTER 3

```
1 class Discriminator(nn.Module):
       def __init__(self, input_shape):
           super(Discriminator, self).__init__()
 3
 4
 5
           channels, height, width = input_shape
 6
           # Calculate output shape of image discriminator (PatchGAN)
 8
           self.output_shape = (1, height // 2 ** 4, width // 2 ** 4)
 9
10
           def discriminator_block(in_filters, out_filters, normalize=True):
               """Returns downsampling layers of each discriminator block"""
11
12
               layers = [nn.Conv2d(in_filters, out_filters, 4, stride=2, padding=1)]
13
               if normalize:
                   layers.append(nn.InstanceNorm2d(out_filters))
14
               layers.append(nn.LeakyReLU(0.2, inplace=True))
15
16
               return layers
```

PatchGAN의 Discriminator output : 입력 이미지의 1/16 size의 이진화된 feature map

CHAPTER 3

```
1 class Discriminator(nn.Module):
       def __init__(self, input_shape):
           super(Discriminator, self).__init__()
 3
 4
           channels, height, width = input_shape
 5
 6
           # Calculate output shape of image discriminator (PatchGAN)
           self.output shape = (1, height // 2 ** 4, width // 2 ** 4)
 8
10
           def discriminator_block(in_filters, out_filters, normalize=True):
               """Returns downsampling layers of each discriminator block"""
12
               layers = [nn.Conv2d(in_filters, out_filters, 4, stride=2, padding=1)]
               if normalize:
13
14
                   layers.append(nn.InstanceNorm2d(out_filters))
               layers.append(nn.LeakyReLU(0.2, inplace=True))
15
16
               return layers
```

Down sampling을 통해 출력 image의 크기를 줄임

CHAPTER 3

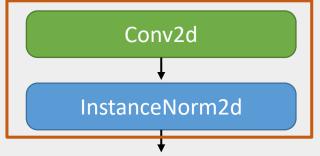
```
self.model = nn.Sequential(
17
               *discriminator_block(channels, 64, normalize=False),
18
               *discriminator_block(64, 128),
19
               *discriminator_block(128, 256),
20
               *discriminator_block(256, 512),
21
               nn.ZeroPad2d((1, 0, 1, 0)),
22
               nn.Conv2d(512, 1, 4, padding=1)
23
24
25
26
       def forward(self, img):
27
           return self.model(img)
```

Discriminator_block 4번 통과 \rightarrow size $\frac{1}{16}$ 배

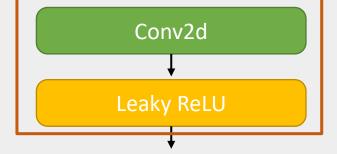
CHAPTER 3

```
1 class Discriminator(nn.Module):
       def __init__(self, input_shape):
           super(Discriminator, self).__init__()
           channels, height, width = input shape
           # Calculate output shape of image discriminator (PatchGAN)
           self.output_shape = (1, height // 2 ** 4, width // 2 ** 4)
 8
 9
           def discriminator_block(in_filters, out_filters, normalize=True):
10
               """Returns downsampling layers of each discriminator block"""
11
               layers = [nn.Conv2d(in filters, out filters, 4, stride=2, padding=1)]
12
13
               if normalize:
                   layers.append(nn.InstanceNorm2d(out_filters))
14
15
               layers.append(nn.LeakyReLU(0.2, inplace=True))
               return layers
16
17
18
           self.model = nn.Sequential(
               *discriminator_block(channels, 64, normalize=False),
19
               *discriminator_block(64, 128),
20
               *discriminator_block(128, 256),
21
               *discriminator_block(256, 512),
22
               nn.ZeroPad2d((1, 0, 1, 0)),
23
               nn.Conv2d(512, 1, 4, padding=1)
24
25
26
       def forward(self, img):
27
           return self.model(img)
28
```

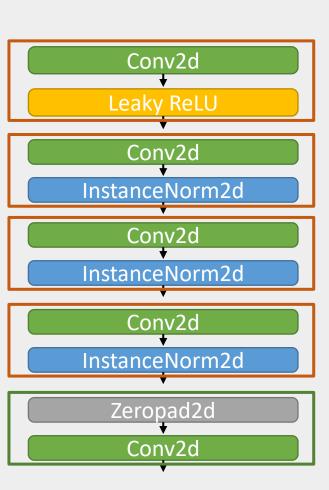
Normalize = True



Normalize = false



```
1 class Discriminator(nn.Module):
       def __init__(self, input_shape):
           super(Discriminator, self).__init__()
           channels, height, width = input shape
          # Calculate output shape of image discriminator (PatchGAN)
           self.output_shape = (1, height // 2 ** 4, width // 2 ** 4)
           def discriminator_block(in_filters, out_filters, normalize=True):
10
               """Returns downsampling layers of each discriminator block"""
11
               layers = [nn.Conv2d(in filters, out filters, 4, stride=2, padding=1)]
12
13
               if normalize:
                   layers.append(nn.InstanceNorm2d(out filters))
14
15
               layers.append(nn.LeakyReLU(0.2, inplace=True))
               return layers
16
17
18
           self.model = nn.Sequential(
               *discriminator_block(channels, 64, normalize=False),
19
               *discriminator block(64, 128),
20
               *discriminator_block(128, 256),
21
               *discriminator_block(256, 512),
22
               nn.ZeroPad2d((1, 0, 1, 0)),
23
               nn.Conv2d(512, 1, 4, padding=1)
24
25
26
       def forward(self, img):
27
           return self.model(img)
28
```



Q&A

2021.11.10

김 민 준



THANK YOU.

2021.11.10

김 민 준