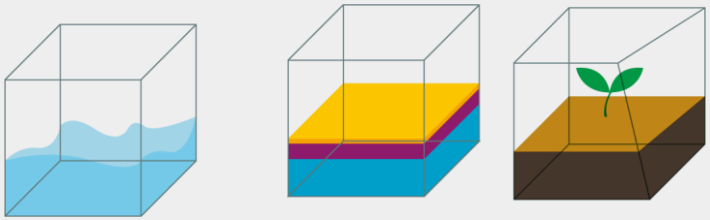
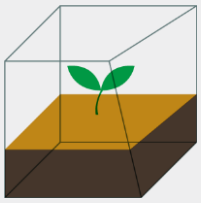
CycleGAN

Generator Discriminator



2021.11.10

김 민 준



CONTENTS

전처리

CHAPTER 1

```python
from PIL import Image

def to_rgb(image):
    rgb_image = Image.new("RGB", image.size)
    rgb_image.paste(image)
    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.*

전처리

CHAPTER 1

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

    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:
            image_B = Image.open(self.files_B[index % len(self.files_B)])
        if image_A.mode != "RGB":
            image_A = to_rgb(image_A)
        if image_B.mode != "RGB":
            image_B = to_rgb(image_B)

        item_A = self.transform(image_A)
        item_B = self.transform(image_B)
        return {"A": item_A, "B": item_B}

    def __len__(self):
        return max(len(self.files_A), len(self.files_B))
```

전처리

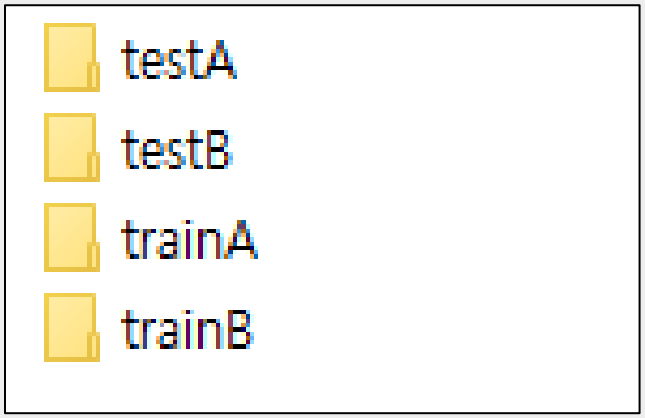
CHAPTER 1

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

    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:
            image_B = Image.open(self.files_B[index % len(self.files_B)])
        if image_A.mode != "RGB":
            image_A = to_rgb(image_A)
        if image_B.mode != "RGB":
            image_B = to_rgb(image_B)

        item_A = self.transform(image_A)
        item_B = self.transform(image_B)
        return {"A": item_A, "B": item_B}

    def __len__(self):
        return max(len(self.files_A), len(self.files_B))
```

전처리

CHAPTER 1

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



* : 임의 길이의 모든 문자열

→ 폴더 내의 모든 항목

전처리

CHAPTER 1

```python
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:
            image_B = Image.open(self.files_B[index % len(self.files_B)])

        # Convert grayscale images to rgb
        if image_A.mode != "RGB":
            image_A = to_rgb(image_A)
        if image_B.mode != "RGB":
            image_B = to_rgb(image_B)

        item_A = self.transform(image_A)
        item_B = self.transform(image_B)
        return {"A": item_A, "B": item_B}

    def __len__(self):
        return max(len(self.files_A), len(self.files_B))
```

전처리

CHAPTER 1

```python
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:
            image_B = Image.open(self.files_B[index % len(self.files_B)])

        # Convert grayscale images to rgb
        if image_A.mode != "RGB":
            image_A = to_rgb(image_A)
        if image_B.mode != "RGB":
            image_B = to_rgb(image_B)

        item_A = self.transform(image_A)
        item_B = self.transform(image_B)
        return {"A": item_A, "B": item_B}

    def __len__(self):
        return max(len(self.files_A), len(self.files_B))
```

Unaligned 변수를 통해 학습할 쌍을 무작위로 고를지, 고정시킬지 정함

전처리

CHAPTER 1

```python
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:
            image_B = Image.open(self.files_B[index % len(self.files_B)])

        # Convert grayscale images to rgb
        if image_A.mode != "RGB":
            image_A = to_rgb(image_A)
        if image_B.mode != "RGB":
            image_B = to_rgb(image_B)

        item_A = self.transform(image_A)
        item_B = self.transform(image_B)
        return {"A": item_A, "B": item_B}

    def __len__(self):
        return max(len(self.files_A), len(self.files_B))
```

RGB가 아니면 RGB로 변환

PIL image를 pytorch tensor로 변환

Generator 구현 – initialize weight

CHAPTER 2

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

Layer의 종류에 따라 다른 가중치 초기화

Generator 구현 – Residual Block

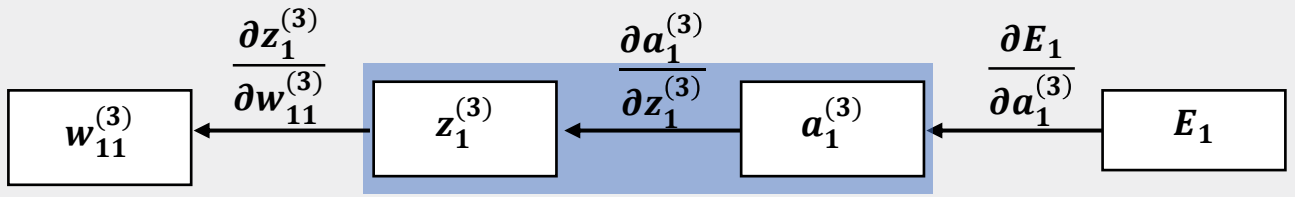
CHAPTER 2

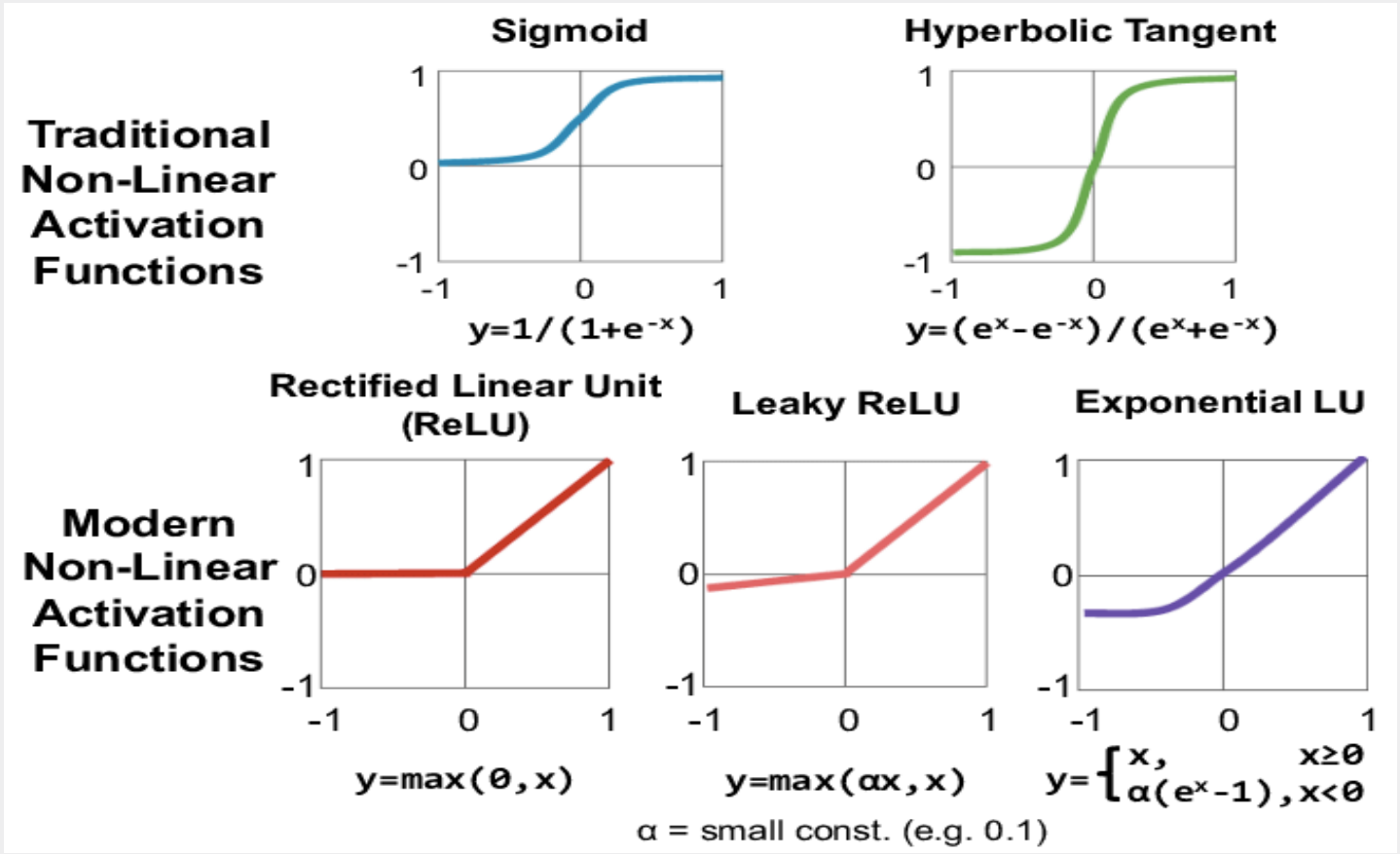
```python
class ResidualBlock(nn.Module):
    def __init__(self, in_features):
        super(ResidualBlock, self).__init__()

        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),
            nn.Conv2d(in_features, in_features, 3),
            nn.InstanceNorm2d(in_features),
        )

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

이전 Layer와 현재 Layer의 출력값을 더해서 Forward
→ Gradient vanishing 해결

Gradient vanishing?

Back Propagation





Generator 구현

CHAPTER 2

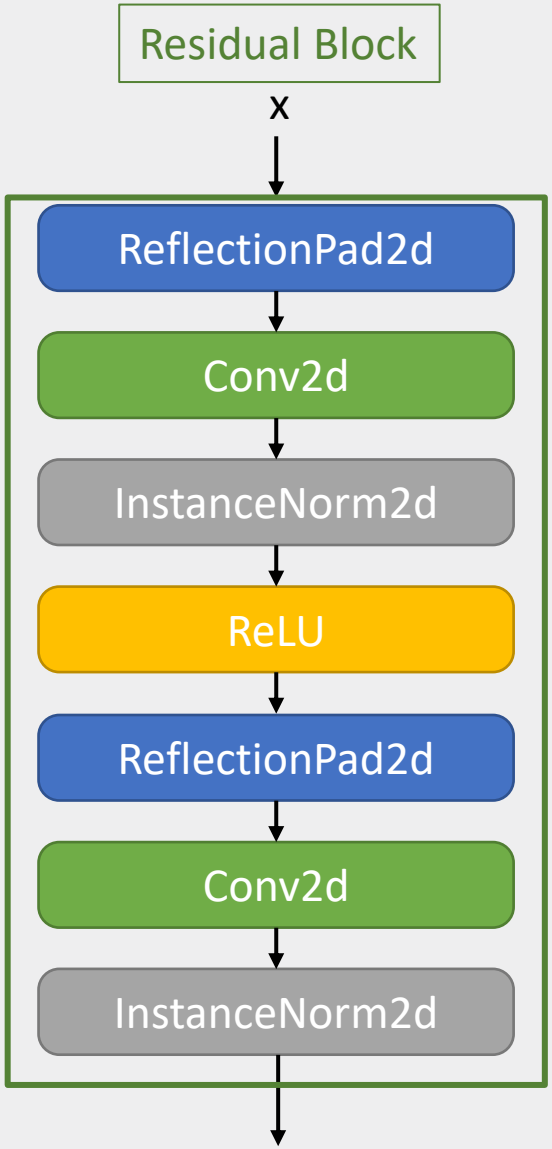
```python
class ResidualBlock(nn.Module):
    def __init__(self, in_features):
        super(ResidualBlock, self).__init__()

        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),
            nn.Conv2d(in_features, in_features, 3),
            nn.InstanceNorm2d(in_features),
        )

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

가장 가까운 픽셀 값을 복사 후 padding에 사용
→ Zero padding보다 더욱 자연스러운 이미지 생성

Generator 구현

CHAPTER 2

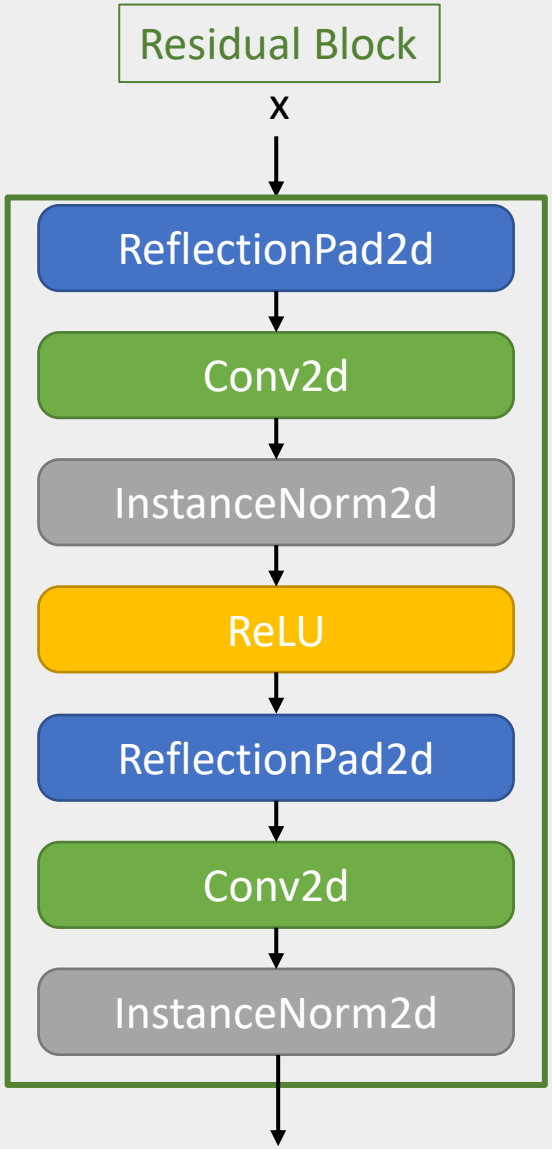
```python
class ResidualBlock(nn.Module):
    def __init__(self, in_features):
        super(ResidualBlock, self).__init__()

        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),
            nn.Conv2d(in_features, in_features, 3),
            nn.InstanceNorm2d(in_features),
        )

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



Generator 구현

CHAPTER 2

```python
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),
            nn.Conv2d(in_features, in_features, 3),
            nn.InstanceNorm2d(in_features),
        )

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



이미지에 특화된 정규화 방법으로 개별 이미지를 정규화

Generator 구현

CHAPTER 2

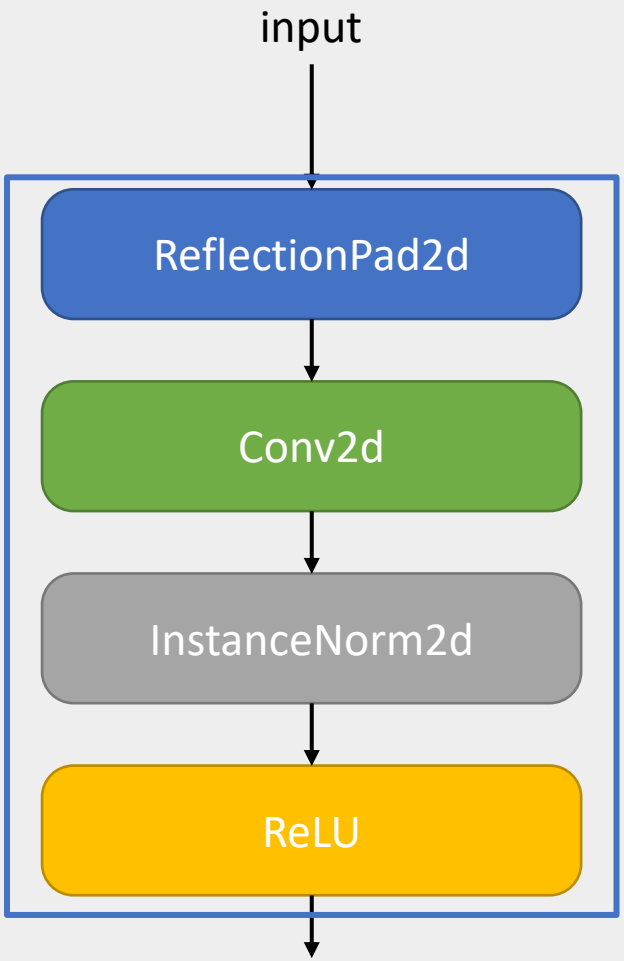
```python
class GeneratorResNet(nn.Module):
    def __init__(self, input_shape, num_residual_blocks):
        super(GeneratorResNet, self).__init__()
        channels = input_shape[0]
        # Initial convolution block
        out_features = 64
        model = [
            nn.ReflectionPad2d(channels),
            nn.Conv2d(channels, out_features, 7),
            nn.InstanceNorm2d(out_features),
            nn.ReLU(inplace=True),
        ]
        in_features = out_features
        # Downsampling
        for _ in range(2):
            out_features *= 2
            model += [
                nn.Conv2d(in_features, out_features, 3, stride=2, padding=1),
                nn.InstanceNorm2d(out_features),
                nn.ReLU(inplace=True),
            ]
            in_features = out_features
        # Residual blocks
        for _ in range(num_residual_blocks):
            model += [ResidualBlock(out_features)]
        # Upsampling
        for _ in range(2):
            out_features //= 2
            model += [
                nn.Upsample(scale_factor=2),
                nn.Conv2d(in_features, out_features, 3, stride=1, padding=1),
                nn.InstanceNorm2d(out_features),
                nn.ReLU(inplace=True),
            ]
            in_features = out_features
        # Output layer
        model += [nn.ReflectionPad2d(channels), nn.Conv2d(out_features, channels, 7), nn.Tanh()]
        self.model = nn.Sequential(*model)
    def forward(self, x):
        return self.model(x)
```

Generator 구현

CHAPTER 2

```python
class GeneratorResNet(nn.Module):
    def __init__(self, input_shape, num_residual_blocks):
        super(GeneratorResNet, self).__init__()

        channels = input_shape[0]

        # Initial convolution block
        out_features = 64
        model = [
            nn.ReflectionPad2d(channels),
            nn.Conv2d(channels, out_features, 7),
            nn.InstanceNorm2d(out_features),
            nn.ReLU(inplace=True),
        ]
        in_features = out_features
```

초기 convolution block 선언



Generator 구현

CHAPTER 2

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

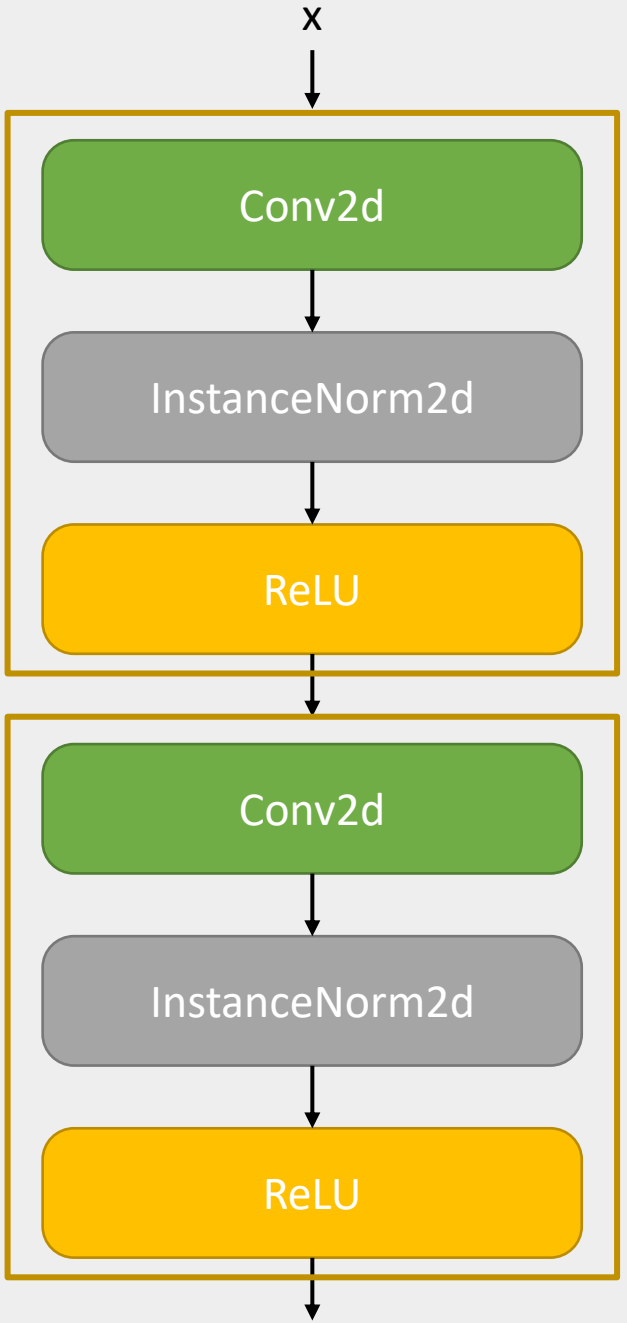
Down sampling 2회, stride = 2 → 크기 $\frac{1}{4}$ 배

Downsampling은 input image의 특징을 추출

Generator 구현

CHAPTER 2

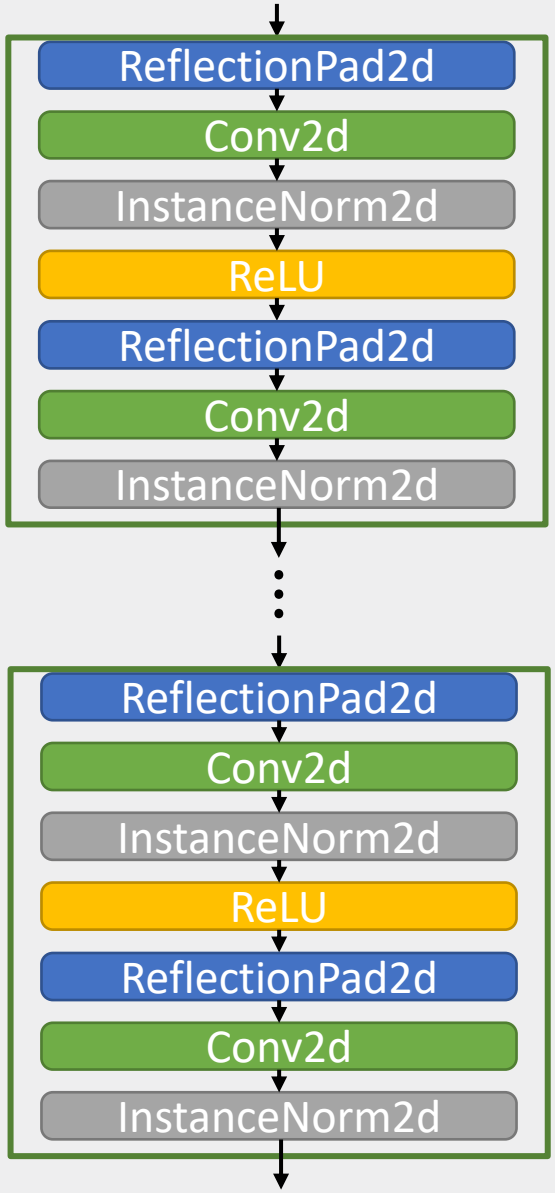
```
16          # Downsampling
17          for _ in range(2):
18              out_features *= 2
19              model += [
20                  nn.Conv2d(in_features, out_features, 3, stride=2, padding=1),
21                  nn.InstanceNorm2d(out_features),
22                  nn.ReLU(inplace=True),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



Generator 구현

CHAPTER 2

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



Generator 구현

CHAPTER 2

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

Up Sampling 2회 → 크기 4배

Upsampling을 통해 이미지의 스타일을 바꿔(translation)주는 용도로 사용

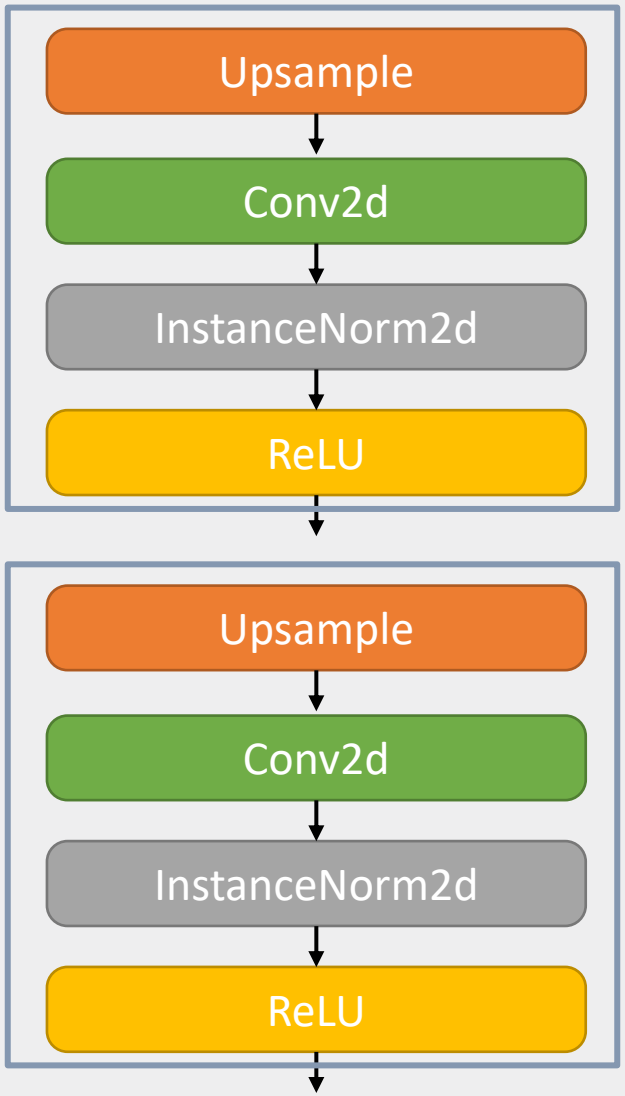
Generator 구현

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

```
>>> input = torch.arange(1, 5, dtype=torch.float32).view(1, 1, 2, 2)
>>> input
tensor([[[[ 1.,  2.],
          [ 3.,  4.]]]])

>>> m = nn.Upsample(scale_factor=2, mode='nearest')
>>> m(input)
tensor([[[[ 1.,  1.,  2.,  2.],
          [ 1.,  1.,  2.,  2.],
          [ 3.,  3.,  4.,  4.],
          [ 3.,  3.,  4.,  4.]]]])
```

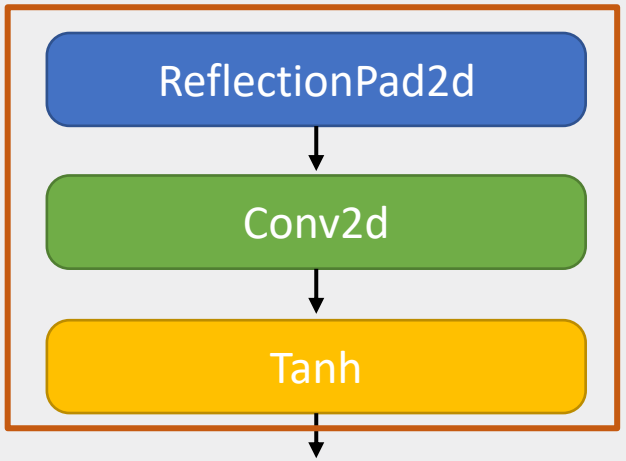
Generator 구현

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



Generator 구현

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



Generator 구현

Discriminator 구현

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

Discriminator 구현

CHAPTER 3

```python
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):
            """Returns downsampling layers of each discriminator block"""
            layers = [nn.Conv2d(in_filters, out_filters, 4, stride=2, padding=1)]
            if normalize:
                layers.append(nn.InstanceNorm2d(out_filters))
            layers.append(nn.LeakyReLU(0.2, inplace=True))
            return layers

        self.model = nn.Sequential(
            *discriminator_block(channels, 64, normalize=False),
            *discriminator_block(64, 128),
            *discriminator_block(128, 256),
            *discriminator_block(256, 512),
            nn.ZeroPad2d((1, 0, 1, 0)),
            nn.Conv2d(512, 1, 4, padding=1)
        )

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

Discriminator 구현

CHAPTER 3

```python
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):
            """Returns downsampling layers of each discriminator block"""
            layers = [nn.Conv2d(in_filters, out_filters, 4, stride=2, padding=1)]
            if normalize:
                layers.append(nn.InstanceNorm2d(out_filters))
            layers.append(nn.LeakyReLU(0.2, inplace=True))
            return layers
```

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

Discriminator 구현

CHAPTER 3

```python
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):
            """Returns downsampling layers of each discriminator block"""
            layers = [nn.Conv2d(in_filters, out_filters, 4, stride=2, padding=1)]
            if normalize:
                layers.append(nn.InstanceNorm2d(out_filters))
            layers.append(nn.LeakyReLU(0.2, inplace=True))
            return layers
```

Down sampling을 통해 출력 image의 크기를 줄임

Discriminator 구현

CHAPTER 3

```
17          self.model = nn.Sequential(
18              *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)
```

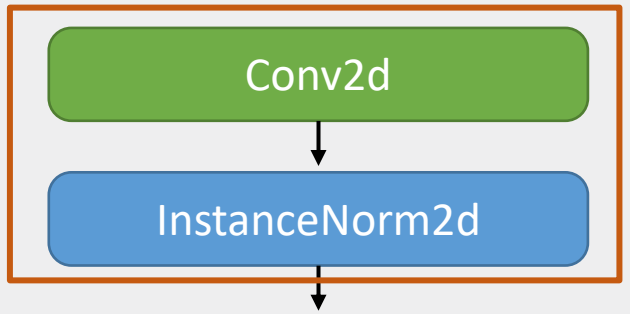
Discriminator_block 4번 통과 → size $\frac{1}{16}$ 배

Discriminator 구현

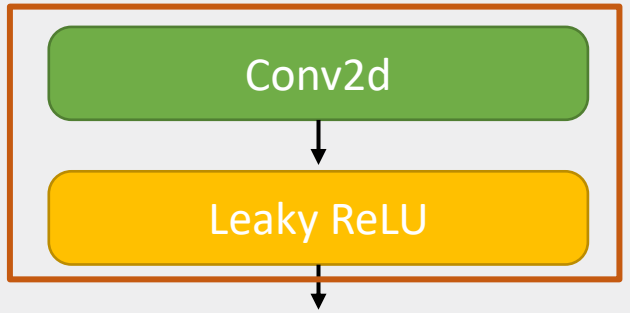
CHAPTER 3

```python
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):
            """Returns downsampling layers of each discriminator block"""
            layers = [nn.Conv2d(in_filters, out_filters, 4, stride=2, padding=1)]
            if normalize:
                layers.append(nn.InstanceNorm2d(out_filters))
            layers.append(nn.LeakyReLU(0.2, inplace=True))
            return layers

        self.model = nn.Sequential(
            *discriminator_block(channels, 64, normalize=False),
            *discriminator_block(64, 128),
            *discriminator_block(128, 256),
            *discriminator_block(256, 512),
            nn.ZeroPad2d((1, 0, 1, 0)),
            nn.Conv2d(512, 1, 4, padding=1)
        )

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

Normalize = True



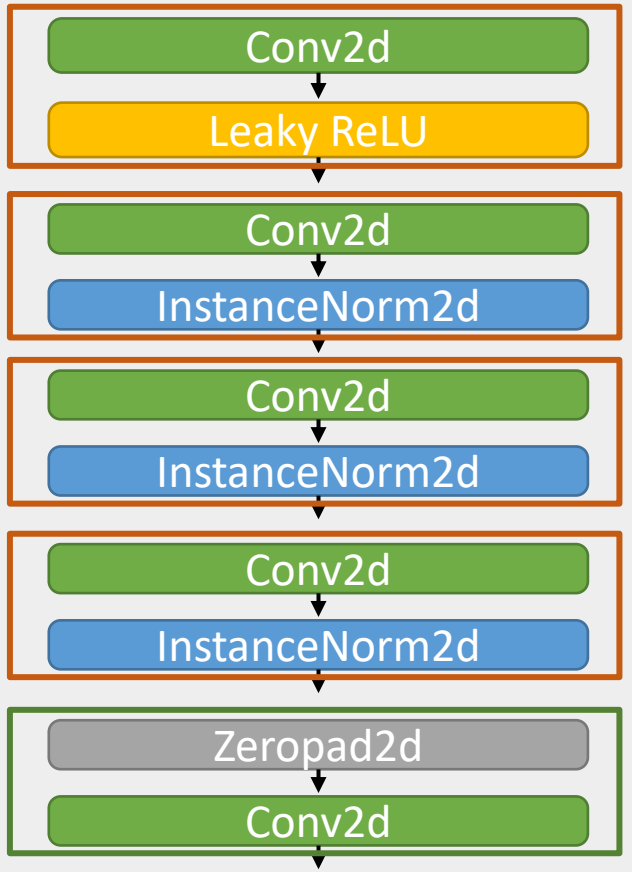
Normalize = false



Discriminator 구현

CHAPTER 3

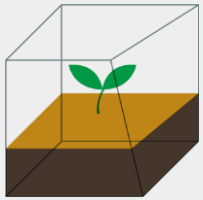
```python
class Discriminator(nn.Module):
    def __init__(self, input_shape):
        super(Discriminator, self).__init__()

        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):
            """Returns downsampling layers of each discriminator block"""
            layers = [nn.Conv2d(in_filters, out_filters, 4, stride=2, padding=1)]
            if normalize:
                layers.append(nn.InstanceNorm2d(out_filters))
            layers.append(nn.LeakyReLU(0.2, inplace=True))
            return layers

        self.model = nn.Sequential(
            *discriminator_block(channels, 64, normalize=False),
            *discriminator_block(64, 128),
            *discriminator_block(128, 256),
            *discriminator_block(256, 512),
            nn.ZeroPad2d((1, 0, 1, 0)),
            nn.Conv2d(512, 1, 4, padding=1)
        )

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



Q&A

2021.11.10

김 민 준



THANK YOU.

2021.11.10

김 민 준