생성 모델과 시각 지능

Generative Model and Visual Intelligence

10 주차 | GAN 발전 5

상명대학교 컴퓨터과학과

민 경 하

상명대학교
SANGMYUNG UNIVERSITY

학습목차

1. CycleGAN의 개념
2. CycleGAN의 구조
3. CycleGAN의 구성 요소

1. CycleGAN의 개념

1. CycleGAN의 개념

CycleGAN

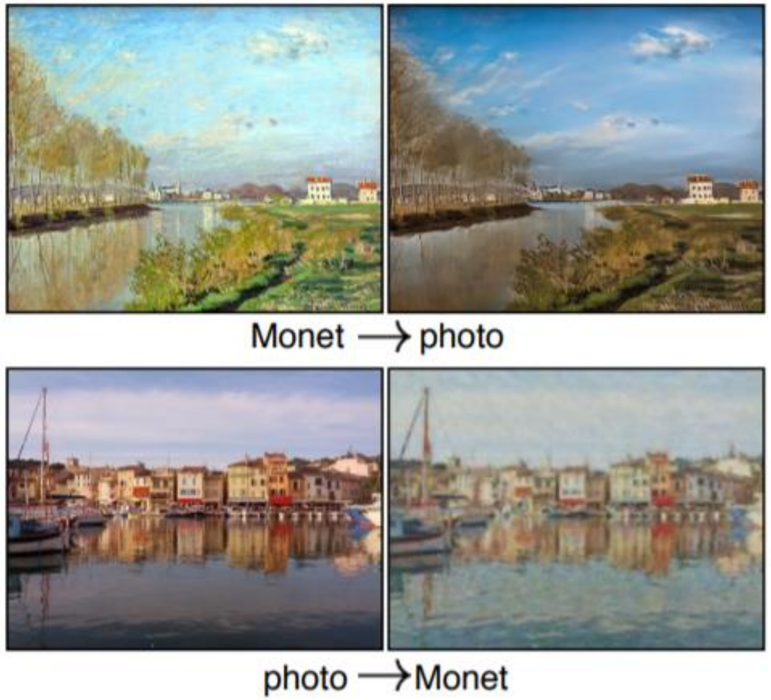
- 두 Image domain 사이의 스타일 변환
- Unpaired image domain 사이의 변환

pix2pix → CycleGAN → gauGAN

출처: Zhu, J. Y., Park, T., Isola, P., and Efros, A., Unpaired image-to-image translation using cycle-consistent adversarial networks, *Proc. of CVPR 2017*, pp. 2223-2232, 2017.

CycleGAN

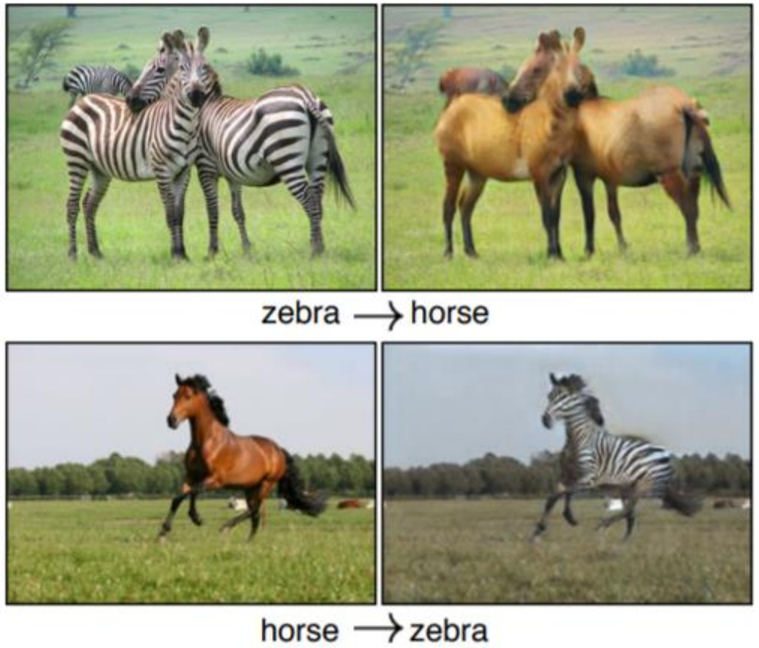
- Unpaired image domain 사이의 스타일 변환



Monet ↔ Photo

CycleGAN

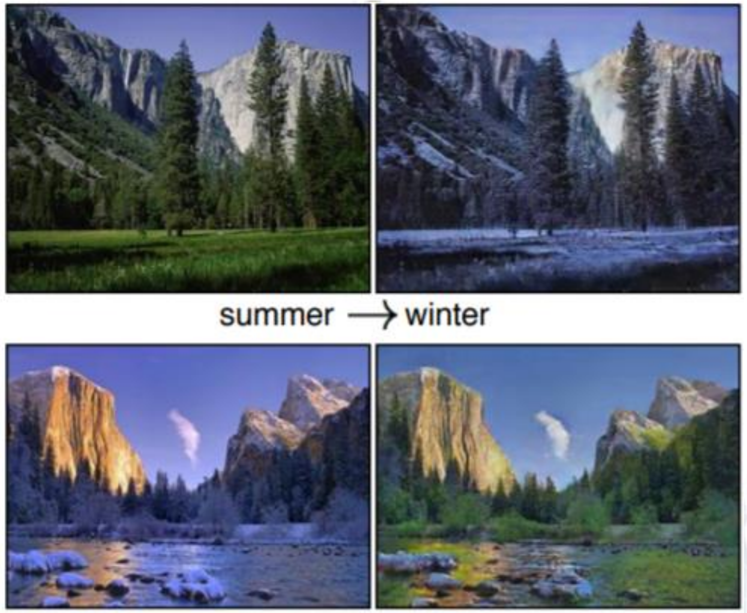
- Unpaired image domain 사이의 스타일 변환



얼룩말(Zebra) ↔ 말(Horse)

CycleGAN

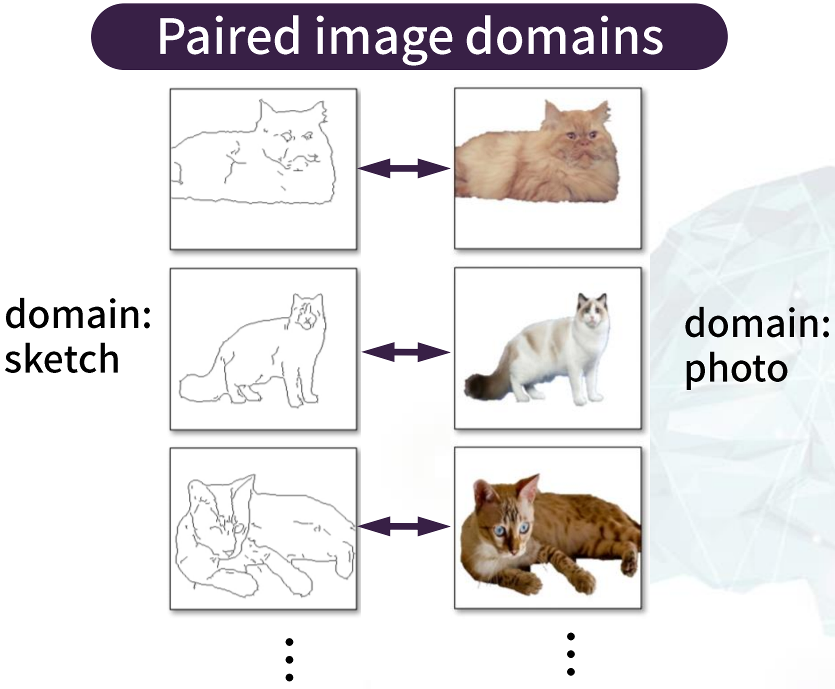
- Unpaired image domain 사이의 스타일 변환

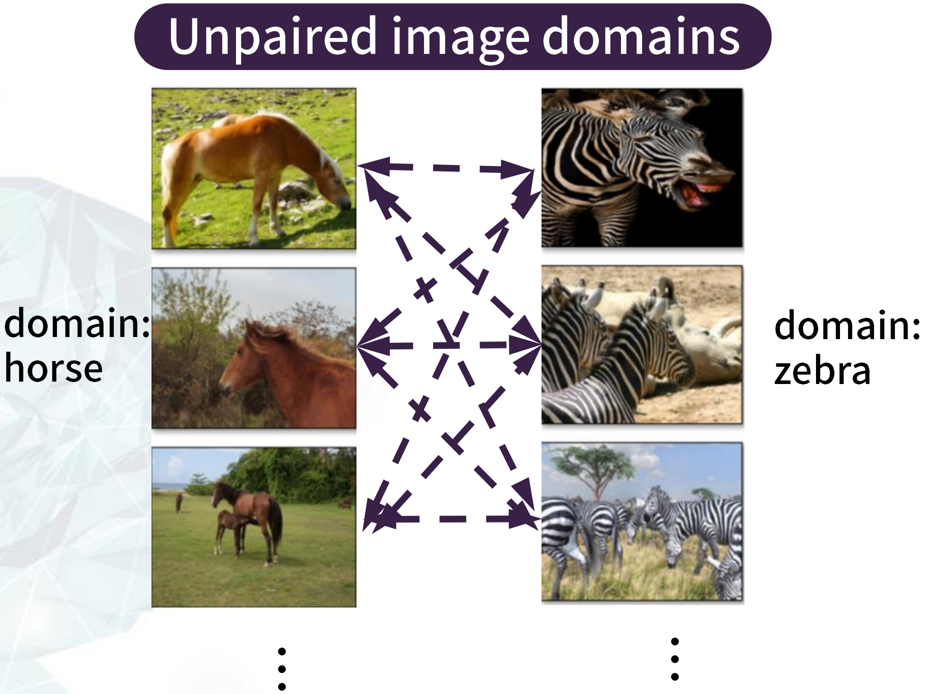


Summer ↔ Winter

CycleGAN의 배경

- Paired image-to-image translation (pix2pix)의 한계
  - 두 Domain의 Image들이 서로 대응 관계를 유지해야 함

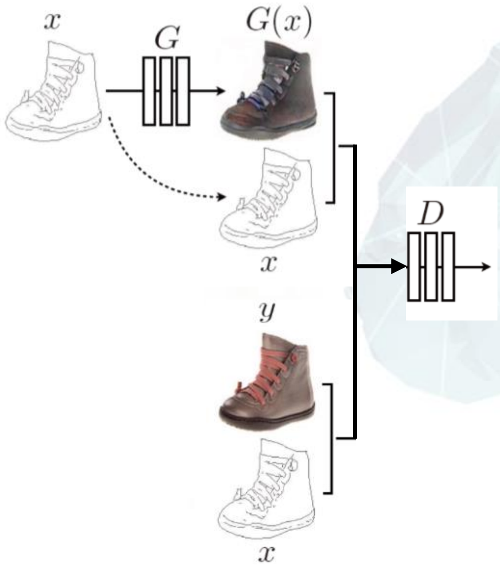




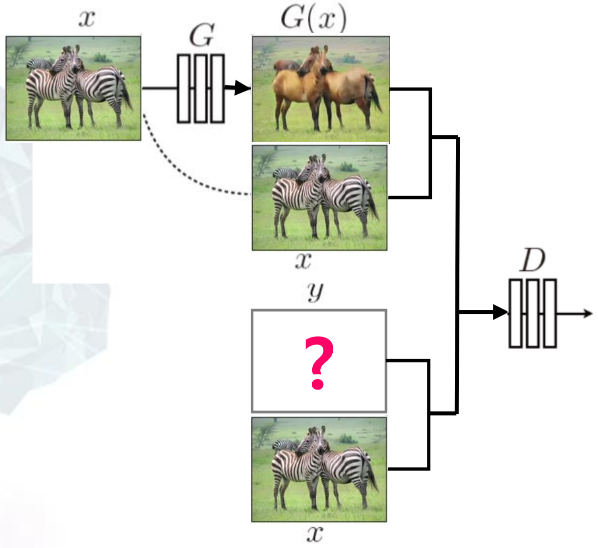
CycleGAN의 배경

- Paired image-to-image translation의 한계
  - Paired image domains의 경우에는 변환 후에 비교할 수 있는 GT가 있음
  - Unpaired image domains의 경우에는 비교할 수 있는 GT가 없음

Paired image domains

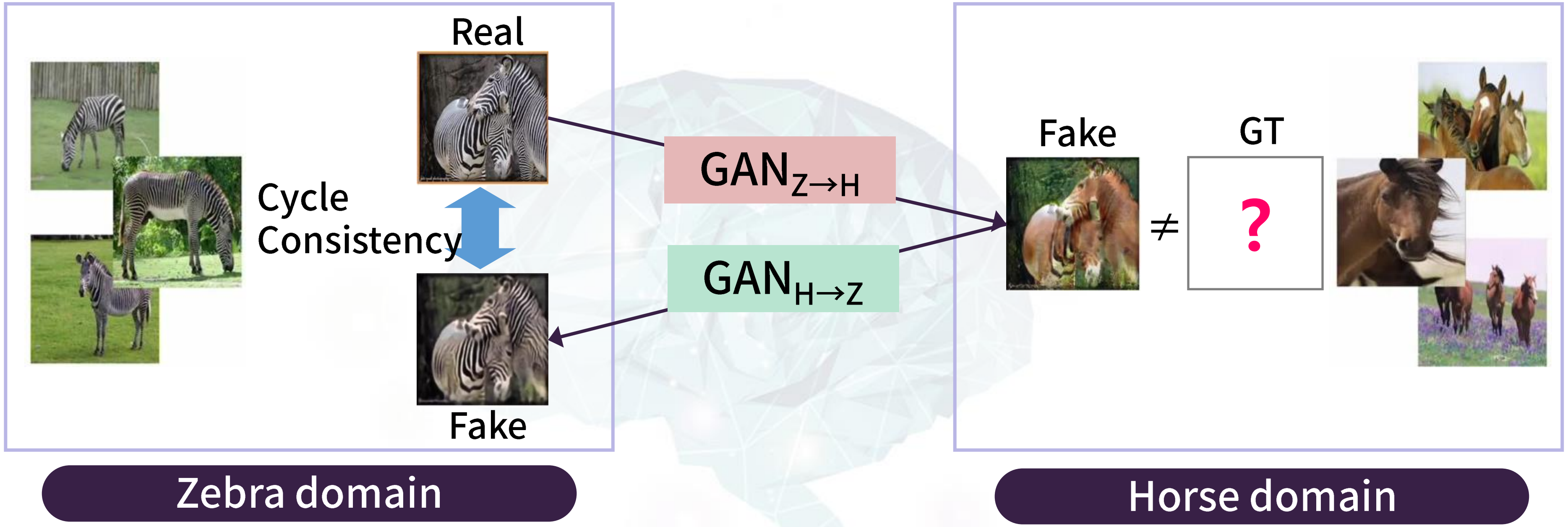


Unpaired image domains



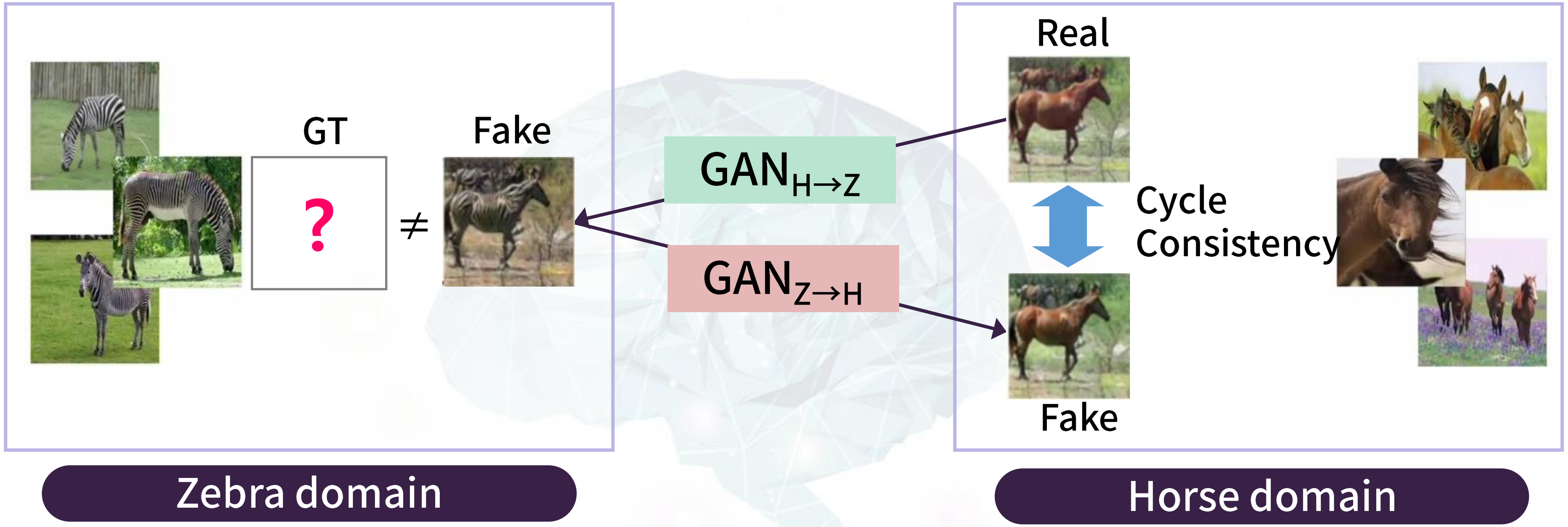
CycleGAN의 배경

- Paired image-to-image translation의 한계 극복
  - *Key idea 1*: 2 개의 GAN을 배치 ($GAN_{Z\rightarrow H}$와 $GAN_{H\rightarrow Z}$: 같은 구조, 다른 Training)
  - *Key idea 2*: Fake를 생성해서 비교 (Cycle consistency)



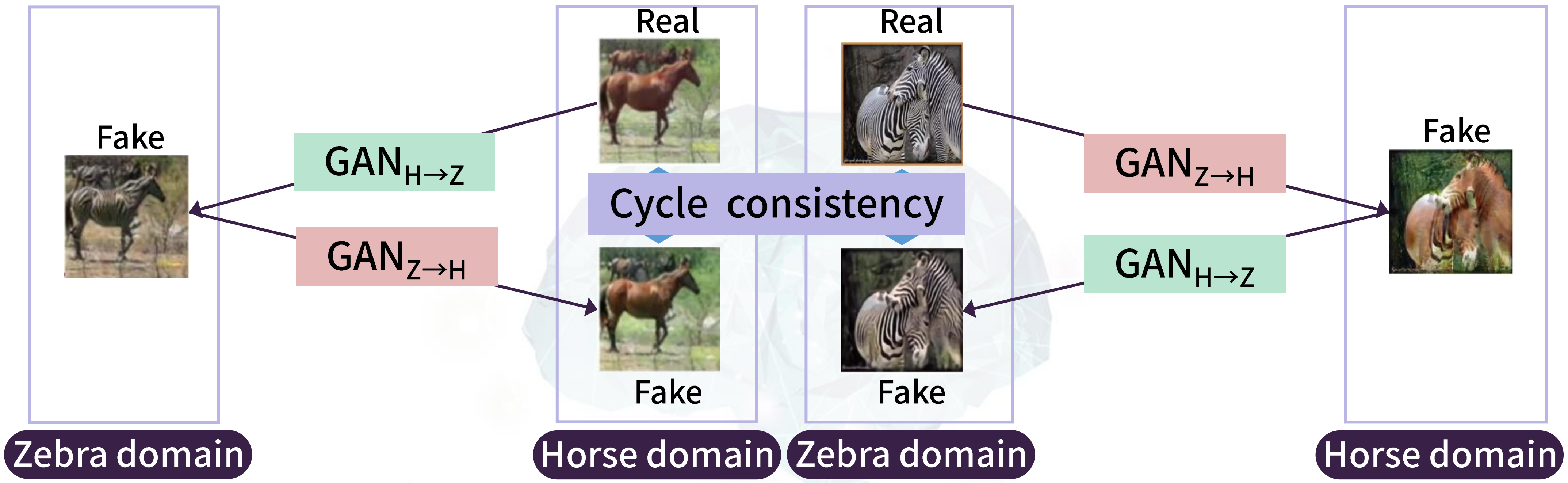
CycleGAN의 배경

- Paired image-to-image translation의 한계 극복
  - *Key idea 1*: 2 개의 GAN을 배치 ($GAN_{Z \rightarrow H}$와 $GAN_{H \rightarrow Z}$)
  - *Key idea 2*: GT를 생성해서 비교 (Cycle consistency)



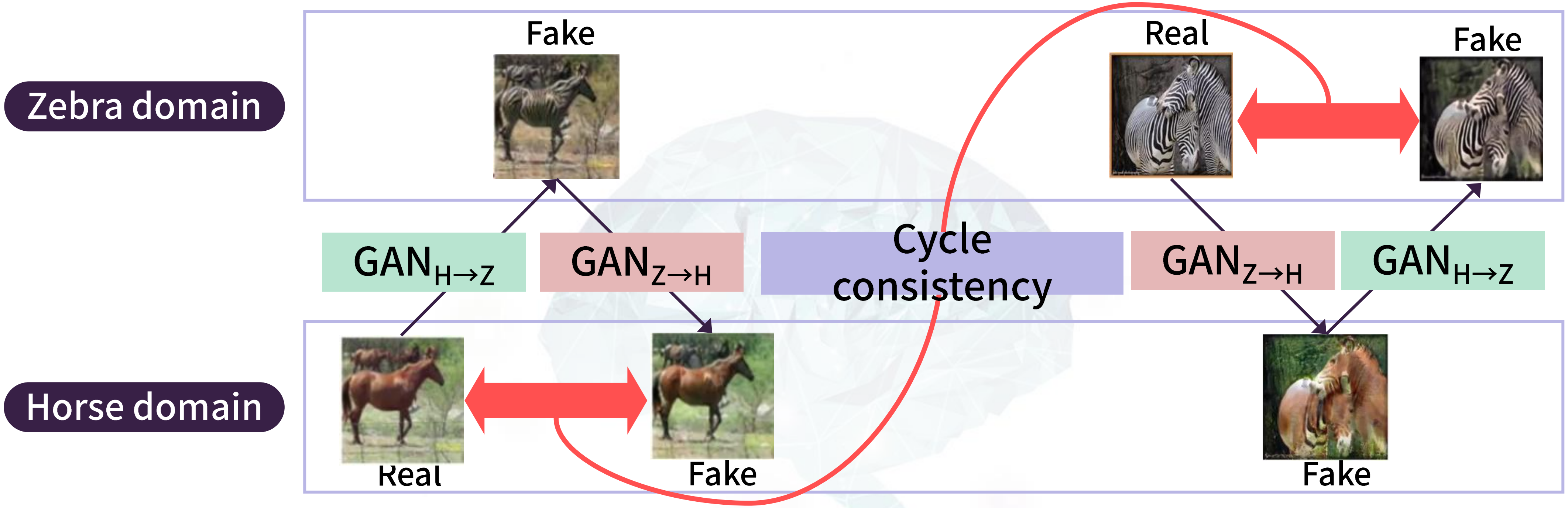
CycleGAN의 배경

- Paired image-to-image translation의 한계 극복
  - 2 개의 GAN ($GAN_{H \to Z}$와 $GAN_{Z \to H}$)을 이용한 2개의 Cycle consistency 유지



CycleGAN의 배경

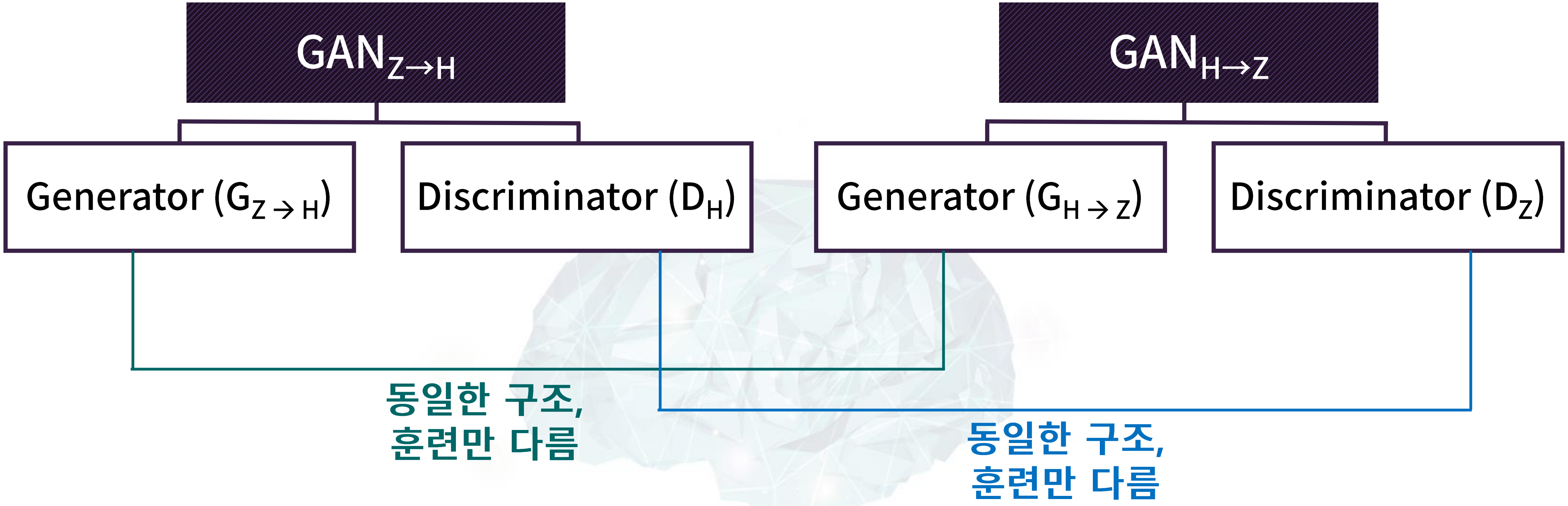
- Paired image-to-image translation의 한계 극복
  - 2 개의 GAN ($GAN_{H \rightarrow Z}$와 $GAN_{Z \rightarrow H}$)을 이용한 2개의 Cycle consistency 유지



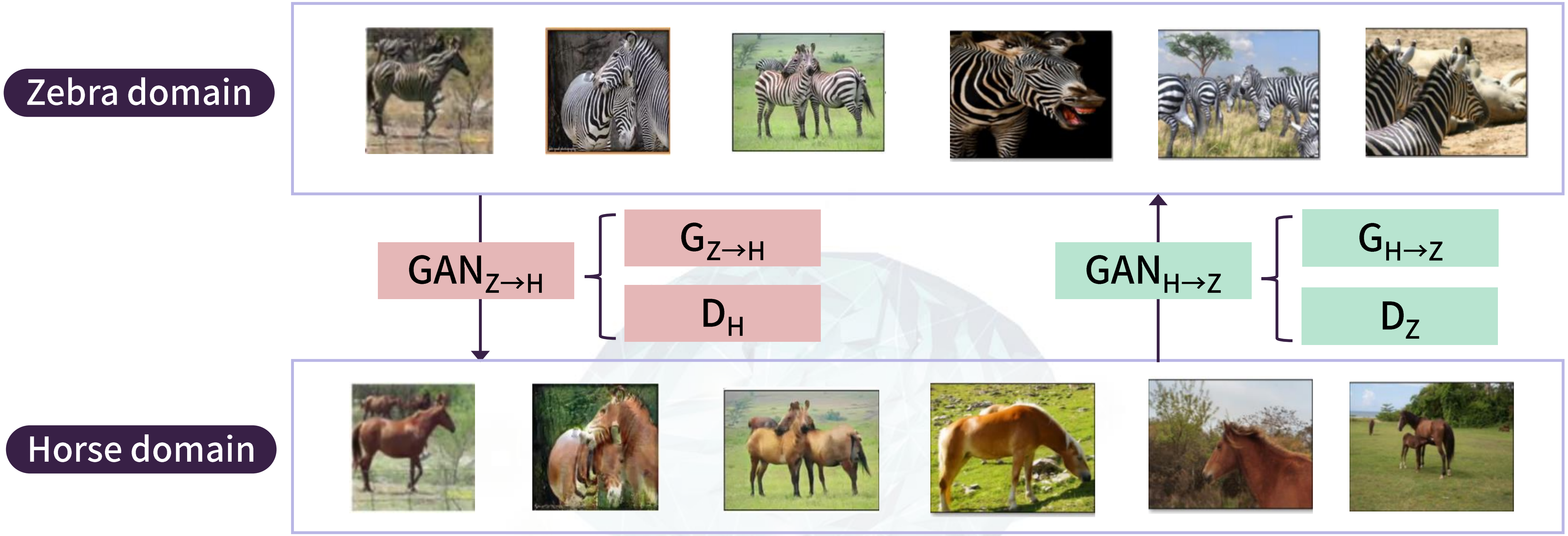
2. CycleGAN의 구조

CycleGAN의 구조

- CycleGAN의 구조는 2개의 GAN으로 구성됨

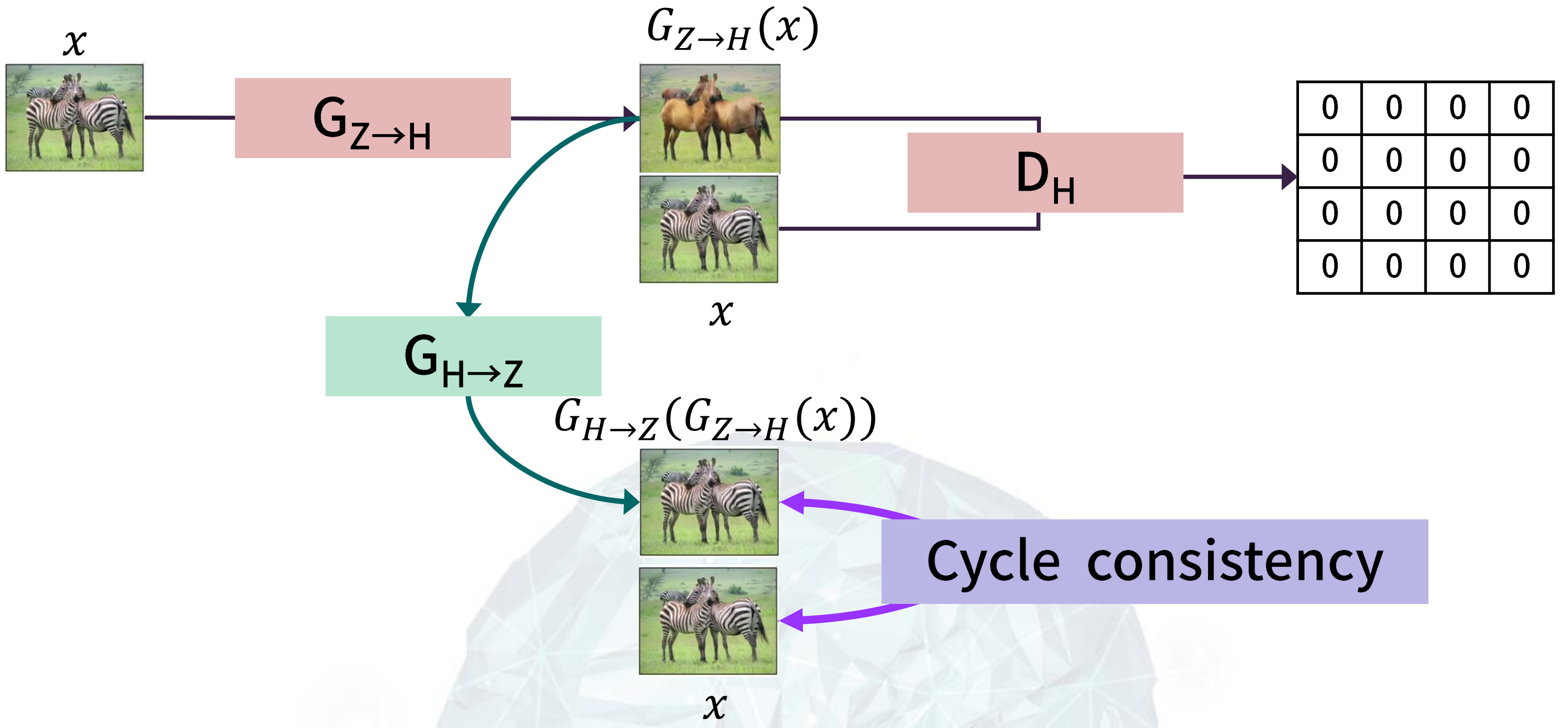


$GAN_{Z\rightarrow H}$와 $GAN_{H\rightarrow Z}$의 관계



2. CycleGAN의 구조

생성구조

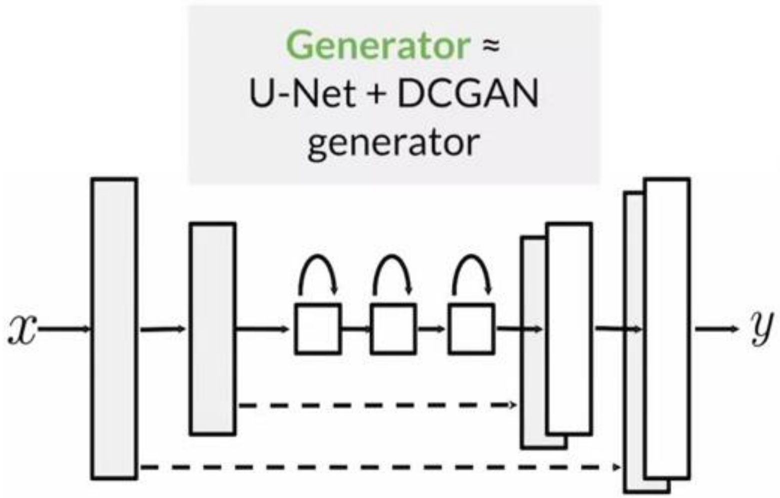


① 입력 (x)에서 Fake ($G_{Z\to H}$(x)) 생성
② 입력 (x)과 Fake ($G_{Z\to H}$(x))에 대한 Discrimination
③ Fake horse로부터 Reconstructed zebra ($G_{H\to Z}$($G_{Z\to H}$(x))) 생성
④ 입력 Zebra와 Reconstructed zebra 간의 Cycle consistency

3. CycleGAN의 구성 요소

Generator

- pix2pix의 Generator와 동일함
  - Generator: Unet + DCGAN + skip-connection



FeatureMap block

Contractor block

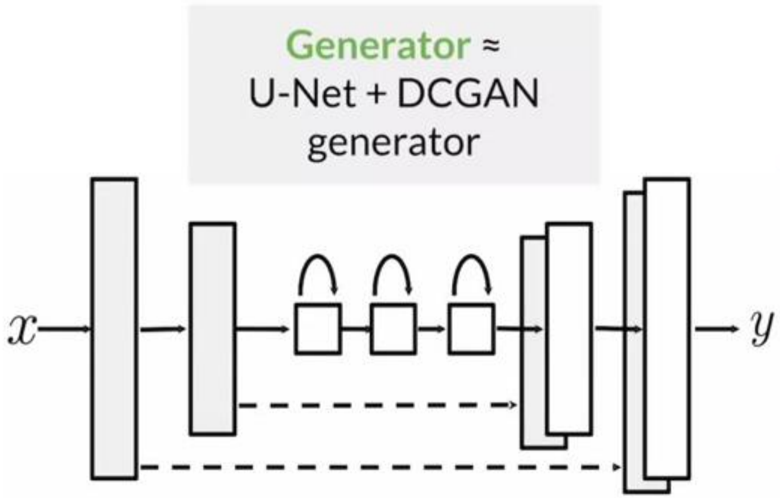
Residual block

Expander block

FeatureMap block

Generator

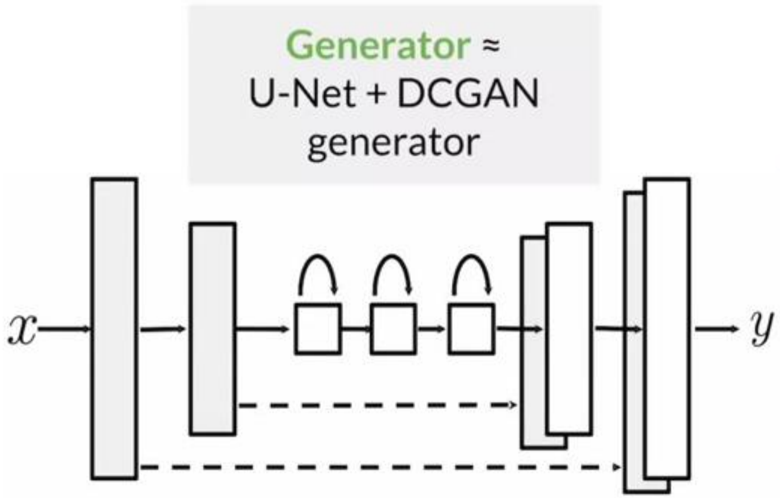
- pix2pix의 Generator와 동일함
  - Generator: Unet + DCGAN + skip-connection



FeatureMap block	upfeature module
Contractor block	contractor module contractor module
Residual block	residual module x 9
Expander block	expander module expander module
FeatureMap block	downfeature module

Generator

- pix2pix의 Generator와 동일함
  - Generator: Unet + DCGAN + skip-connection



		1x3x256x256
FeatureMap block	upfeature module	
		1x64x256x256
Contractor block	contractor module	
		1x128x128x128
	contractor module	
		1x256x64x64
Residual block	residual module x 9	
		1x256x64x64
Expander block	expander module	
		1x128x128x128
	expander module	
		1x64x256x256
FeatureMap block	downfeature module	
		1x3x256x256

Generator

- Featuremap block
  - 1번의 Convolution 연산을 통해서 영상의 해상도는 유지하면서 Channel을 변화시킴
  - Upfeature module: 3 channels → 64 channels
  - Downfeature module: 64 channels → 3 channels

```
class FeatureMapBlock(nn.Module):
    def __init__(self, input_channels, output_channels):
        super(FeatureMapBlock, self).__init__()
        self.conv = nn.Conv2d(input_channels, output_channels, kernel_size=7, padding=3,
                              padding_mode='reflect')

    def forward(self, x):
        x = self.conv(x)
        return x
```

Generator

Contractor block

- 이전 영상의 해상도를 ½로 줄이면서 Channel의 수를 2배로 증가

```
class ContractingBlock(nn.Module):
    def __init__(self, input_channels, use_bn=True, activation='relu'):
        super(ContractingBlock, self).__init__()
        self.conv1 = nn.Conv2d(


        self.activation = nn.ReLU() if activation == 'relu' else nn.LeakyReLU(0.2)
        if use_bn:
            self.instancenorm = nn.InstanceNorm2d(input_channels * 2)
        self.use_bn = use_bn

    def forward(self, x):
        x = self.conv1(x)
        if self.use_bn:
            x = self.instancenorm(x)
        x = self.activation(x)
        return x
```

Generator

Residual block

- 이전 영상의 크기를 유지하면서 conv 연산을 수행하고,
  그 결과 영상과 입력 영상을 Concatenation

```
class ResidualBlock(nn.Module):
    def __init__(self, input_channels):
        super(ResidualBlock, self).__init__()
        self.conv1 = nn.Conv2d(

        self.conv2 = nn.Conv2d(

        self.instancenorm = nn.InstanceNorm2d(input_channels)
        self.activation = nn.ReLU()

    def forward(self, x):
        original_x = x.clone()
        x = self.conv1(x)
        x = self.instancenorm(x)
        x = self.activation(x)
        x = self.conv2(x)
        x = self.instancenorm(x)
        return original_x + x
```

Generator

Expander block

- 이전 영상의 해상도를 2배로 늘이면서 Channel의 수를 ½ 배로 감소

```
class ExpandingBlock(nn.Module):
    def __init__(self, input_channels, use_bn=True):
        super(ExpandingBlock, self).__init__()
        self.conv1 = nn.ConvTranspose2d(

        if use_bn:
            self.instancenorm = nn.InstanceNorm2d(input_channels // 2)
        self.use_bn = use_bn
        self.activation = nn.ReLU()

    def forward(self, x):
        x = self.conv1(x)
        if self.use_bn:
            x = self.instancenorm(x)
        x = self.activation(x)
        return x
```

Generator

Generator

```
class Generator(nn.Module):
    def __init__(self, input_channels, output_channels, hidden_channels=64):
        super(Generator, self).__init__()
        self.upfeature = FeatureMapBlock(input_channels, hidden_channels)
        self.contract1 = ContractingBlock(hidden_channels)
        self.contract2 = ContractingBlock(hidden_channels * 2)
        res_mult = 4
        self.res0 = ResidualBlock(hidden_channels * res_mult)
        self.res1 = ResidualBlock(hidden_channels * res_mult)
        self.res2 = ResidualBlock(hidden_channels * res_mult)
        self.res3 = ResidualBlock(hidden_channels * res_mult)
        self.res4 = ResidualBlock(hidden_channels * res_mult)
        self.res5 = ResidualBlock(hidden_channels * res_mult)
        self.res6 = ResidualBlock(hidden_channels * res_mult)
        self.res7 = ResidualBlock(hidden_channels * res_mult)
        self.res8 = ResidualBlock(hidden_channels * res_mult)
        self.expand2 = ExpandingBlock(hidden_channels * 4)
        self.expand3 = ExpandingBlock(hidden_channels * 2)
        self.downfeature = FeatureMapBlock(hidden_channels, output_channels)
        self.tanh = torch.nn.Tanh()
```

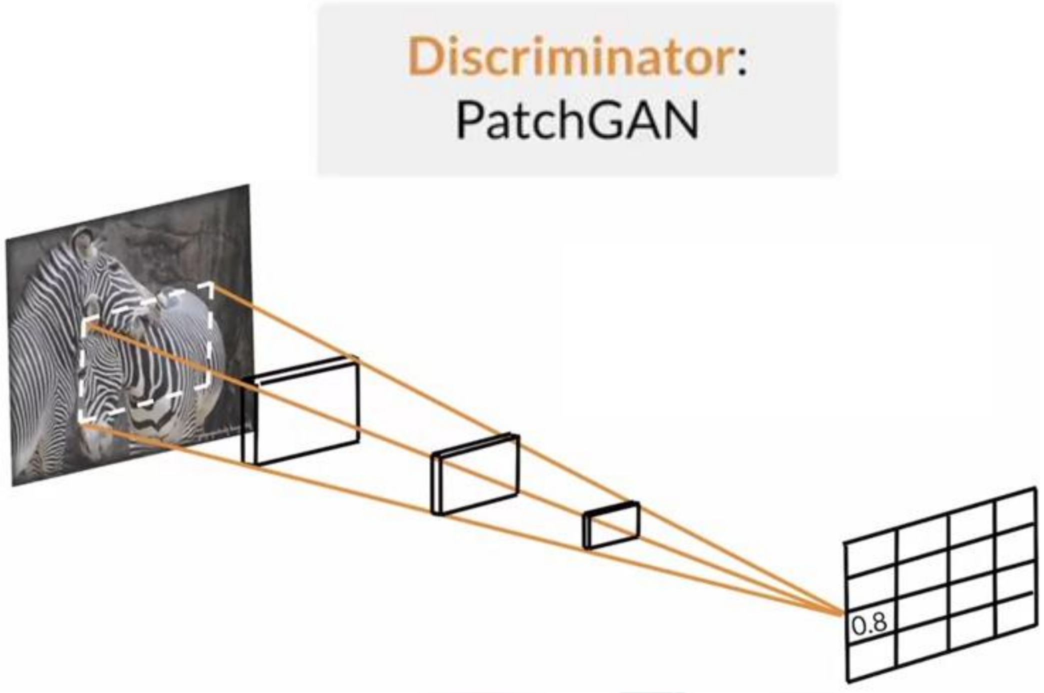
Generator

Generator

```
def forward(self, x):
        x0 =
        x1 =
        x2 =
        x3 =
        x4 =
        x5 =
        x6 =
        x7 =
        x8 =
        x9 =
        x10 =
        x11 =
        x12 =
        x13 =
        xn =

        return self.tanh(xn)
```

Discriminator

- pix2pix의 Discriminator와 동일함
  - Discriminator: PatchGAN



Discriminator

- pix2pix의 Discriminator와 동일함
  - 입력 영상(3x256x256)을 (1x8x8) 크기의 patch로 만들어서 판별함
  - 구조

1x3x256x256

upfeature module

1x64x256x256

contractor module

1x128x128x128

contractor module

1x256x64x64

contractor module

1x512x32x32

final layer

1x1x32x32

Discriminator

- pix2pix의 Discriminator와 동일함

```
class Discriminator(nn.Module):
    def __init__(self, input_channels, hidden_channels=64):
        super(Discriminator, self).__init__()
        self.upfeature = FeatureMapBlock(input_channels, hidden_channels)
        self.contract1 = ContractingBlock(hidden_channels, use_bn=False, kernel_size=4,
activation='lrelu')
        self.contract2 = ContractingBlock(hidden_channels * 2, kernel_size=4, activation='lrelu')
        self.contract3 = ContractingBlock(hidden_channels * 4, kernel_size=4, activation='lrelu')
        self.final = nn.Conv2d(hidden_channels * 8, 1, kernel_size=1)

    def forward(self, x):
        x0 = 
        x1 = 
        x2 = 
        x3 = 
        xn = 

        return xn
```

loss 함수

Adversarial loss (GAN loss)

$GAN_{Z \to H}$에 대한 loss

- Generator $G_{Z \to H}$와 Discriminator D_H를 이용해서 측정

$$\mathcal{L}_{GAN}(G_{Z \to H}, D_H, Z, H) = \mathbb{E}_{y \sim H}[\log D_H(y)] + \mathbb{E}_{x \sim Z}[\log(1 - D_H(G_{Z \to H}(x)))]$$

$GAN_{H \to Z}$에 대한 loss

- Generator $G_{H \to Z}$와 Discriminator D_Z를 이용해서 측정

$$\mathcal{L}_{GAN}(G_{H \to Z}, D_Z, Z, H) = \mathbb{E}_{x \sim Z}[\log D_Z(x)] + \mathbb{E}_{y \sim H}[\log(1 - D_Z(G_{H \to Z}(y)))]$$

loss 함수

- BCE 기반의 loss 함수는 훈련이 힘듦
  - gradient vanishing + mode collapsing
- WGAN과 같은 Least square를 이용한 loss 함수의 정의가 필요

Discriminator loss

$$\mathbb{E}_{y\sim H}[(D_H(y)-1)^2]+\mathbb{E}_{x\sim Z}\left[\left(D_H\left(G_{Z\rightarrow H}(x)\right)-0\right)^2\right]$$

Generator loss

$$\mathbb{E}_{x\sim Z}\left[\left(D_H\left(G_{Z\rightarrow H}(x)\right)-1\right)^2\right]$$

loss 함수

Discriminator loss

$$\mathbb{E}_{y\sim H}[(D_H(y)-1)^2]+\mathbb{E}_{x\sim Z}\left[\left(D_H\left(G_{Z\to H}(x)\right)-0\right)^2\right]$$

```
def get_disc_loss(real_X, fake_X, disc_X, adv_criterion):
    disc_fake_X_hat =
    disc_fake_X_loss =
    disc_real_X_hat =
    disc_real_X_loss =
    disc_loss =

    return disc_loss
```

loss 함수

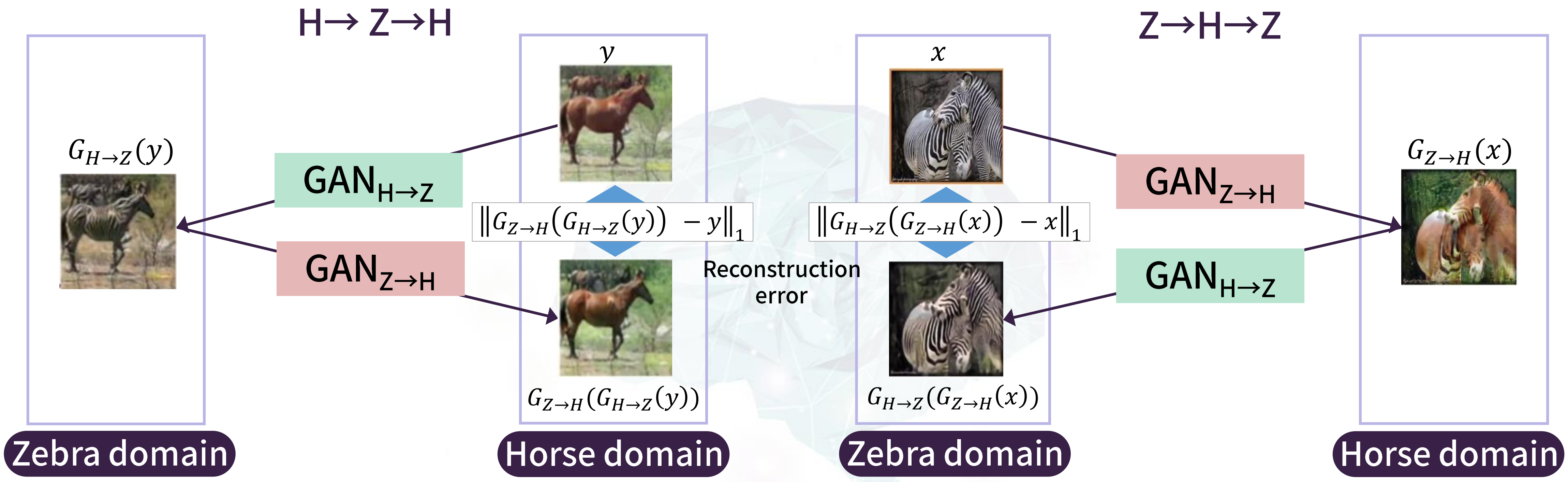
Generator loss (Generator adversarial loss)

$$\mathbb{E}_{x\sim Z}\left[\left(D_H\left(G_{Z\to H}(x)\right)-1\right)^2\right]$$

```
def get_gen_adversarial_loss(real_X, disc_Y, gen_XY, adv_criterion):
    fake_Y =
    disc_fake_Y_hat =
    adversarial_loss =

    return adversarial_loss, fake_Y
```

loss 함수

- cycle consistency loss
  - H→ Z→H에 대한 loss와 Z→H→Z에 대한 loss를 고려



loss 함수

- cycle consistency loss
  - H→ Z→H에 대한 loss와 Z→H→Z에 대한 loss를 고려

$$\mathcal{L}_{cyc}(G_{H\to Z}, G_{Z\to H}) = \mathbb{E}_{x\sim Z}\left[\left\|G_{H\to Z}\left(G_{Z\to H}(x)\right) - x\right\|_1\right] + \mathbb{E}_{y\sim H}\left[\left\|G_{Z\to H}\left(G_{H\to Z}(y)\right) - y\right\|_1\right]$$

loss 함수

- Cycle consistency loss (H→Z)

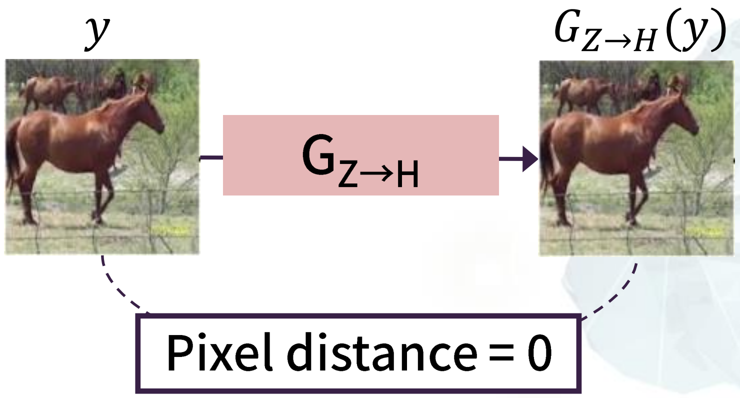
$$\mathbb{E}_{y \sim H}\left[\left\| G_{Z \to H}\left(G_{H \to Z}(y)\right) - y \right\|_1\right]$$

```
def get_cycle_consistency_loss(real_X, fake_Y, gen_YX, cycle_criterion):
    cycle_X =
    cycle_loss =

    return cycle_loss, cycle_X
```

loss 함수

Identity loss

- $G_{Z \to H}$를 Zebra가 아닌 Horse에 적용하면 어떻게 될까?
  - » y와 $G_{Z \to H}(y)$의 Pixel distance는 0이 되어야 함



$$\|G_{Z \to H}(y) - y\|_2 = 0$$

loss 함수

Identity loss

- $G_{Z\to H}$를 Zebra가 아닌 Horse에 적용하면 어떻게 될까?
  - » y와 $G_{Z\to H}(y)$의 Pixel distance는 0이 되어야 함
  - » 실제로 $G_{Z\to H}(y)$는 y와 유사한 구조를 갖지만 색의 차이가 발생하는 경우가 많음

$\|G_{Z\to H}(y) - y\|_2 > 0$

$\|G_{H\to Z}(x) - x\|_2 > 0$

loss 함수

- Identity loss는 다음과 같이 정의함

$$\mathcal{L}_{id}(G_{H\to Z}, G_{Z\to H}) = \mathbb{E}_{x\sim Z}[\|G_{H\to Z}(x) - x\|_2] + \mathbb{E}_{y\sim H}[\|G_{Z\to H}(y) - y\|_2]$$

loss 함수

- Identity loss

$$\mathbb{E}_{x\sim Z}[\|G_{H\to Z}(x) - x\|_2]$$

```
def get_identity_loss(real_X, gen_YX, identity_criterion):
    identity_X = 
    identity_loss = 

    return identity_loss, identity_X
```

loss 함수

- CycleGAN의 loss 함수는 다음의 항으로 구성됨

Adversarial loss ($\mathcal{L}_{GAN}$)	Cycle consistency loss ($\mathcal{L}_{cyc}$)	Identity loss ($\mathcal{L}\_id$)

$$\mathcal{L}(G_{H\to Z}, G_{Z\to H}, D_H, D_Z) = \mathcal{L}_{GAN}(G_{Z\to H}, D_H, Z, H) + \mathcal{L}_{GAN}(G_{Z\to H}, D_H, Z, H)$$

$$+\lambda_1 \mathcal{L}_{cyc}(G_{H\to Z}, G_{Z\to H}) + \lambda_2 \mathcal{L}_{id}(G_{H\to Z}, G_{Z\to H})$$

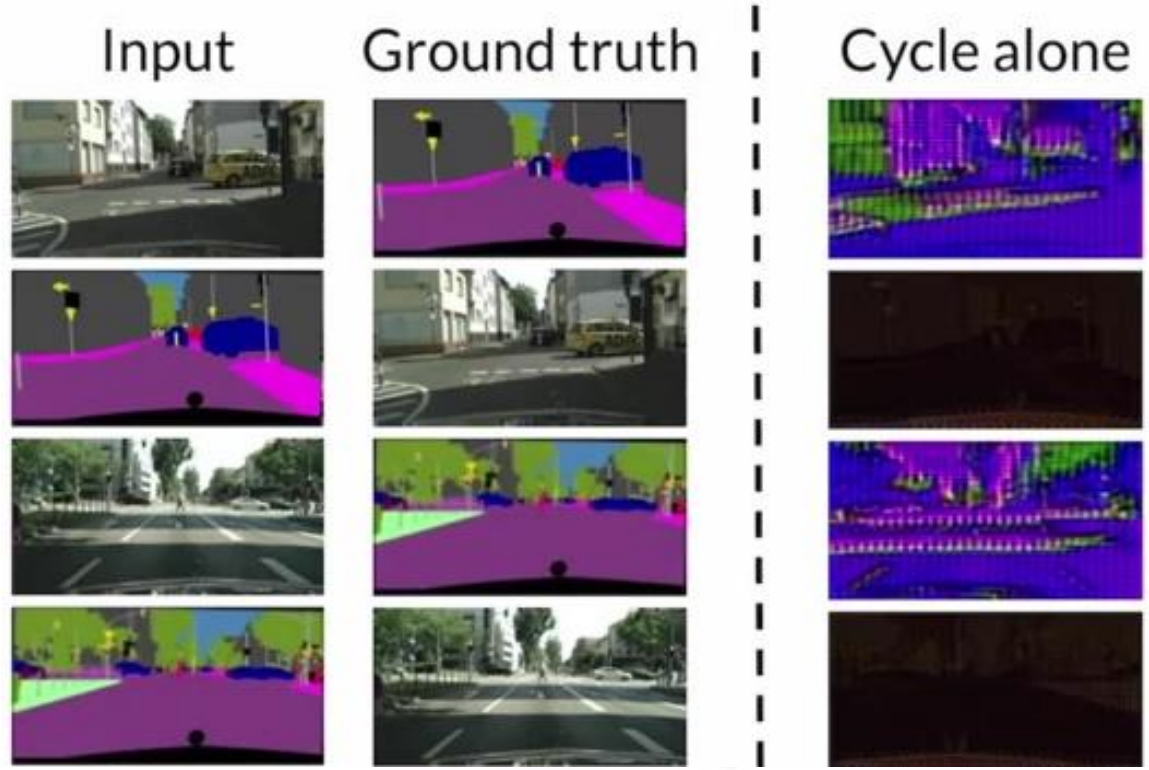
loss 함수

CycleGAN의 loss 함수

```
def get_gen_loss(real_A, real_B, gen_AB, gen_BA, disc_A, disc_B, adv_criterion, identity_criterion,
cycle_criterion, lambda_identity=0.1, lambda_cycle=10):
    # Adversarial Loss
    adv_loss_BA, fake_A =
    adv_loss_AB, fake_B =
    gen_adversarial_loss =

    # Identity Loss
    identity_loss_A, identity_A =
    identity_loss_B, identity_B =
    gen_identity_loss =

    # Cycle-consistency Loss
    cycle_loss_BA, cycle_A =
    cycle_loss_AB, cycle_B =
    gen_cycle_loss =

    # Total loss
    gen_loss =

    return gen_loss, fake_A, fake_B
```

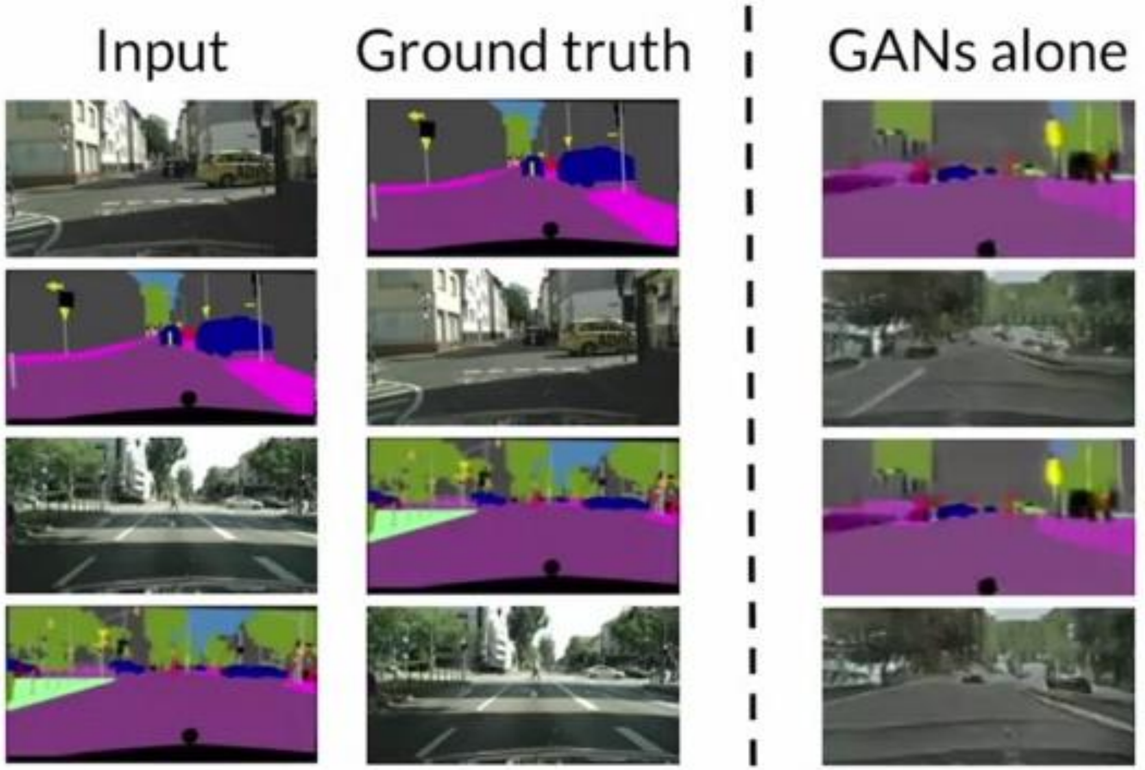
loss 함수

- Ablation study
  - Cycle loss만 고려하고 GAN loss를 고려하지 않으면 결과가 안 좋아짐



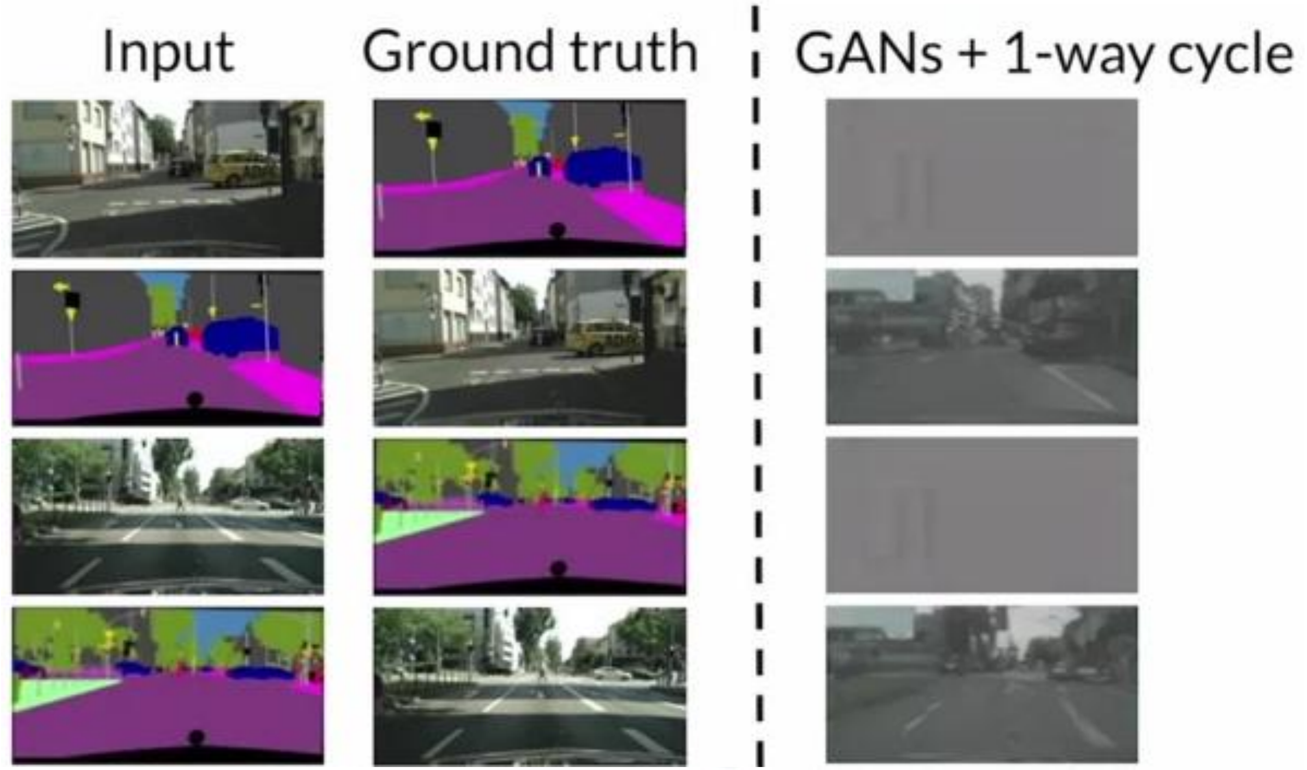
loss 함수

- Ablation study
  - GAN loss만 고려하고 Cycle loss를 고려하지 않으면 Mode collapse



loss 함수

- Ablation study
  - 한 방향의 Cycle loss만 고려해도 Mode collapse



loss 함수

- Ablation study
  - GAN loss와 Cycle loss를 함께 고려해야 좋은 결과가 나옴

loss 함수

- Ablation study
  - Identity loss는 원본의 색을 유지하도록 함