

리뷰 - Cycle GAN

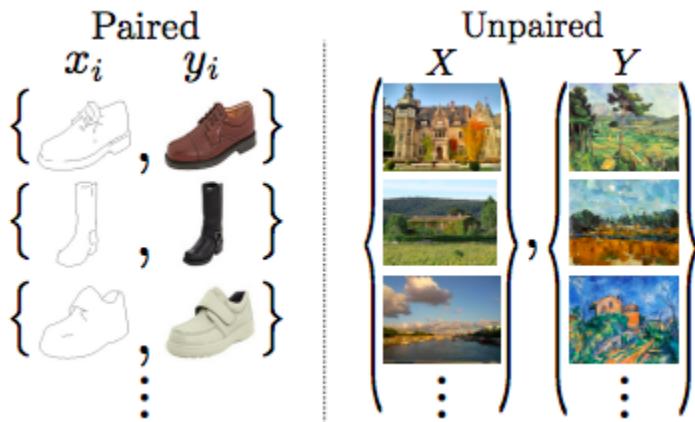
개요

원제 : Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks

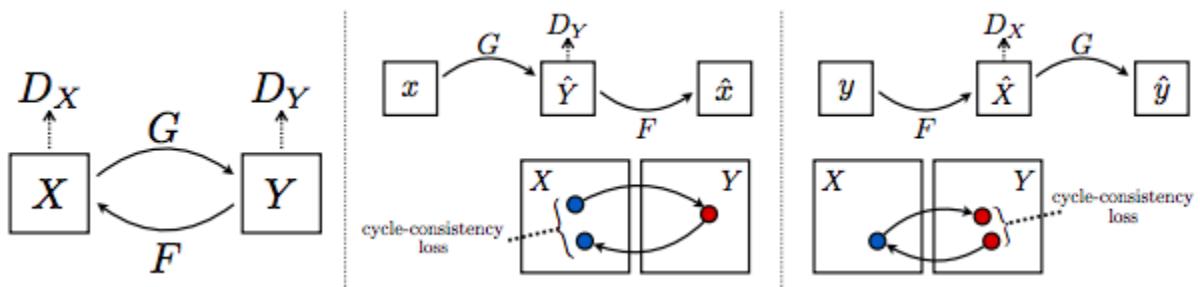
두 도메인간의 이미지 변형(image translation) 방법.

paired data가 없는 데이터를 사용.

기존 많은 방법들은 paired data를 사용했는데, Cycle GAN으로 그런 paired가 없어도 가능함.

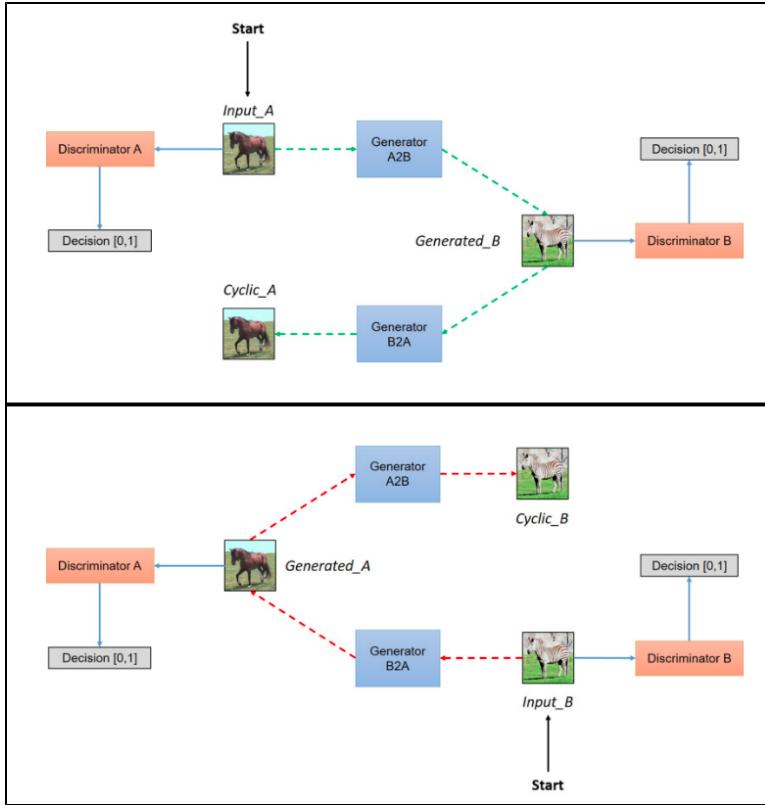


방법



$G(X)$ 가 D_Y 를 통과하고 $F(G(X))$ 가 원본 X 와 일치하도록 학습.

$F(Y)$ 가 D_X 를 통과하고 $G(F(Y))$ 가 원본 Y 와 일치하도록 학습.



로스 평선

X, Y : 2개의 도메인

2개의 generator

- $G : X \rightarrow Y$
- $F : Y \rightarrow X$

2개의 discriminator

- D_X
- D_Y

3개의 항목으로 구성

$$\begin{aligned} \mathcal{L}(G, F, D_X, D_Y) = & \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) \\ & + \mathcal{L}_{\text{GAN}}(F, D_X, Y, X) \\ & + \lambda \mathcal{L}_{\text{cyc}}(G, F), \end{aligned}$$

$$\begin{aligned} \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) = & \mathbb{E}_{y \sim p_{\text{data}}(y)} [\log D_Y(y)] \\ & + \mathbb{E}_{x \sim p_{\text{data}}(x)} [\log(1 - D_Y(G(x)))] \end{aligned}$$

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

실제 구현에서는 \log 를 사용하지 않은 다음을 사용.

G :

$$\mathbb{E}_{x \sim p_{\text{data}}(x)} [(D(G(x)) - 1)^2]$$

D :

$$\mathbb{E}_{y \sim p_{\text{data}}(y)} [(D(y) - 1)^2] +$$

$$\mathbb{E}_{x \sim p_{\text{data}}(x)} [D(G(x))^2]$$

▼ loss code

Discriminator

```

D_A_loss_1 = tf.reduce_mean(tf.squared_difference(dec_A, 1))
D_B_loss_1 = tf.reduce_mean(tf.squared_difference(dec_B, 1))

D_A_loss_2 = tf.reduce_mean(tf.square(dec_gen_A))
D_B_loss_2 = tf.reduce_mean(tf.square(dec_gen_B))

D_A_loss = (D_A_loss_1 + D_A_loss_2)/2
D_B_loss = (D_B_loss_1 + D_B_loss_2)/2

```

Generator

```

g_loss_B_1 = tf.reduce_mean(tf.squared_difference(dec_gen_A, 1))
g_loss_A_1 = tf.reduce_mean(tf.squared_difference(dec_gen_A, 1))

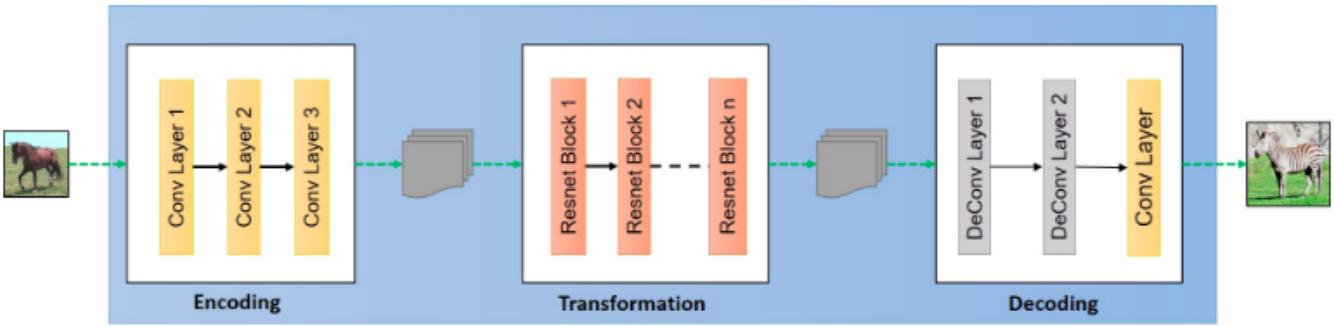
cyc_loss = tf.reduce_mean(tf.abs(input_A-cyc_A)) +
tf.reduce_mean(tf.abs(input_B-cyc_B))

g_loss_A = g_loss_A_1 + 10*cyc_loss
g_loss_B = g_loss_B_1 + 10*cyc_loss

```

세부 구조

Generator



▼ Click here to expand...

```
def build_generator(input_gen):
    o_c1 = general_conv2d(input_gen, num_features=ngf, window_width=7,
    window_height=7, stride_width=1, stride_height=1)
    o_c2 = general_conv2d(o_c1, num_features=ngf*2, window_width=3,
    window_height=3, stride_width=2, stride_height=2)
    o_enc_A = general_conv2d(o_c2, num_features=ngf*4, window_width=3,
    window_height=3, stride_width=2, stride_height=2)

    # Transformation
    o_r1 = build_resnet_block(o_enc_A, num_features=64*4)
    o_r2 = build_resnet_block(o_r1, num_features=64*4)
    o_r3 = build_resnet_block(o_r2, num_features=64*4)
    o_r4 = build_resnet_block(o_r3, num_features=64*4)
    o_r5 = build_resnet_block(o_r4, num_features=64*4)
    o_enc_B = build_resnet_block(o_r5, num_features=64*4)

    #Decoding
    o_d1 = general_deconv2d(o_enc_B, num_features=ngf*2 window_width=3,
    window_height=3, stride_width=2, stride_height=2)
    o_d2 = general_deconv2d(o_d1, num_features=ngf, window_width=3,
    window_height=3, stride_width=2, stride_height=2)
    gen_B = general_conv2d(o_d2, num_features=3, window_width=7,
    window_height=7, stride_width=1, stride_height=1)

    return gen_B
```

▼ def general_conv2d()

```

def general_conv2d(inputconv, o_d=64, f_h=7, f_w=7, s_h=1, s_w=1):
    with tf.variable_scope(name):
        conv = tf.contrib.layers.conv2d(inputconv, num_features,
[window_width, window_height], [stride_width, stride_height],
padding, activation_fn=None,
weights_initializer=tf.truncated_normal_initializer(stddev=stddev),
biases_initializer=tf.constant_initializer(0.0))

```

▼ def resnet_blocks()

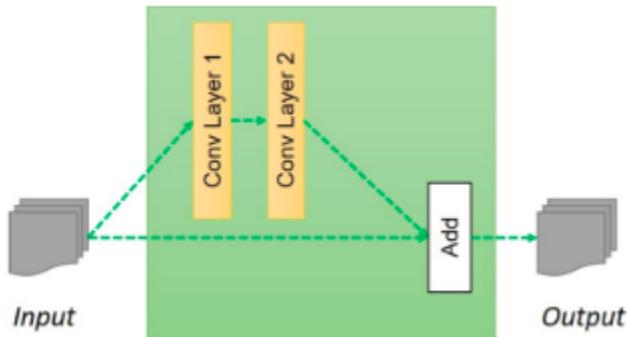
```

def resnet_blocks(input_res, num_features):

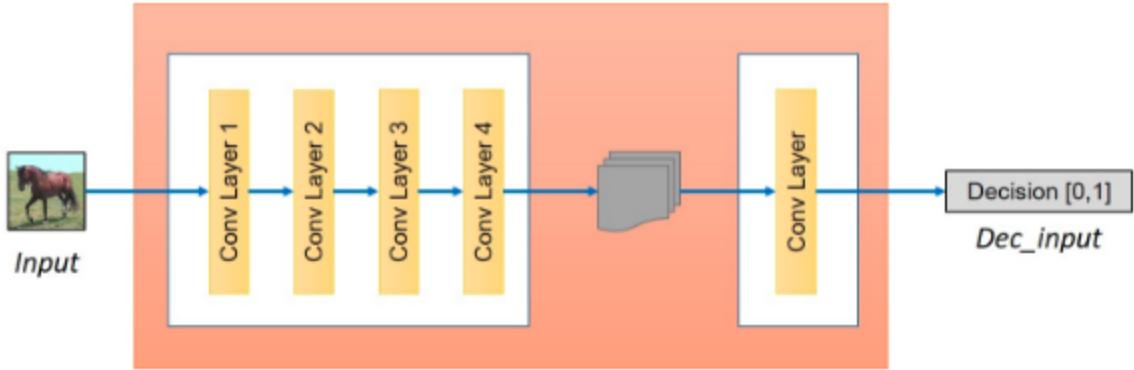
    out_res_1 = general_conv2d(input_res, num_features,
                               window_width=3,
                               window_heighth=3,
                               stride_width=1,
                               stride_heighth=1)
    out_res_2 = general_conv2d(out_res_1, num_features,
                               window_width=3,
                               window_heighth=3,
                               stride_width=1,
                               stride_heighth=1)
    return (out_res_2 + input_res)

```

Resnet Block



Discriminator



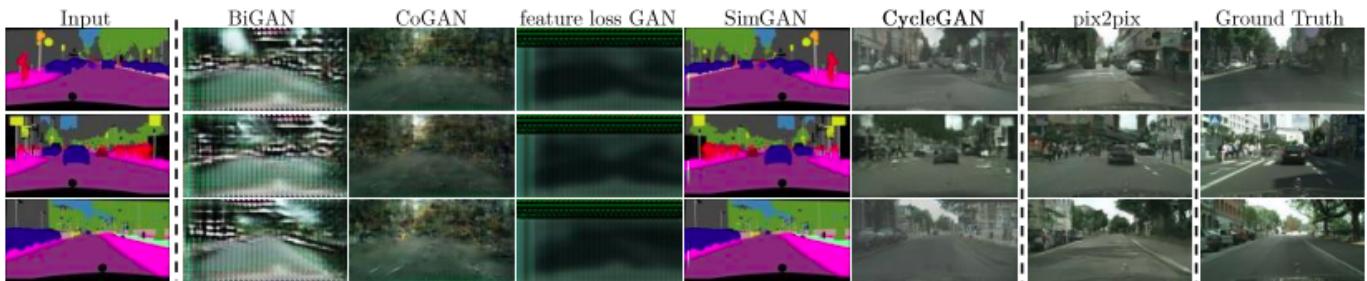
```
v def build_discriminator()
```

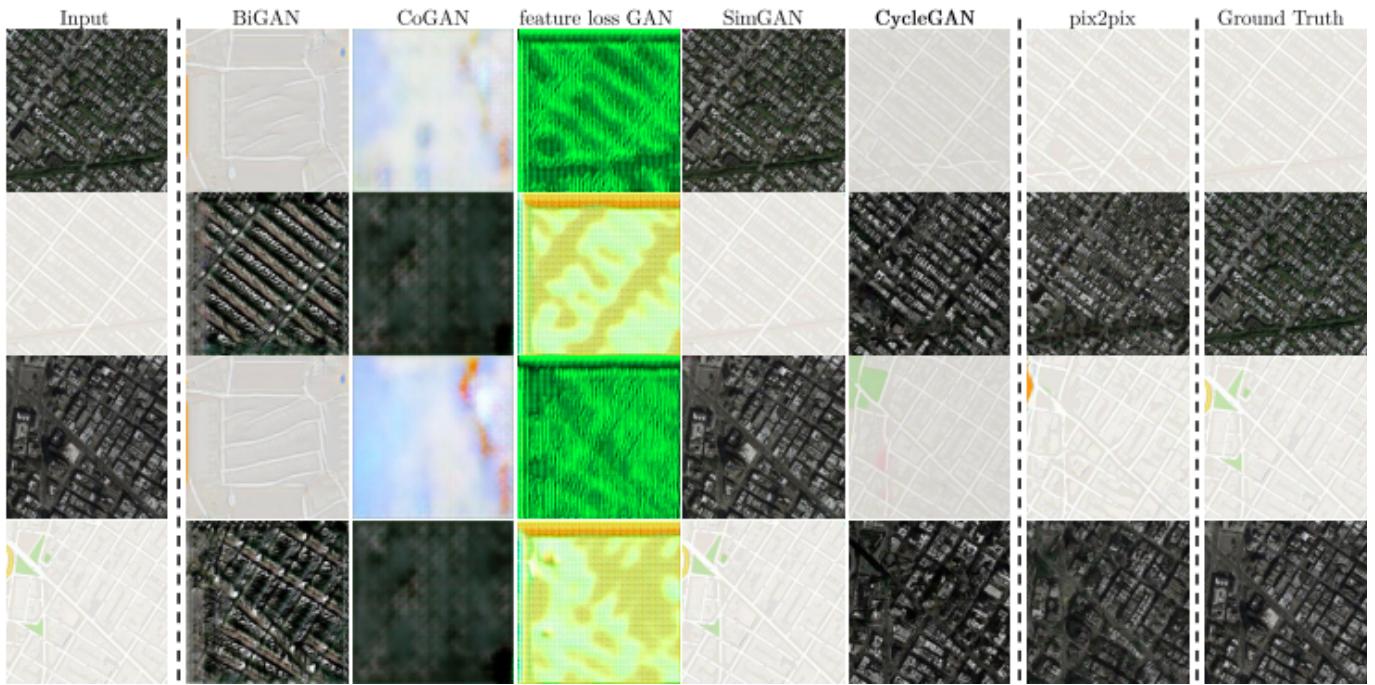
```

o_c1 = general_conv2d(input_disc, ndf, f, f, 2, 2)
o_c2 = general_conv2d(o_c1, ndf*2, f, f, 2, 2)
o_enc_A = general_conv2d(o_c2, ndf*4, f, f, 2, 2)
o_c4 = general_conv2d(o_enc_A, ndf*8, f, f, 2, 2)
decision = general_conv2d(o_c4, 1, f, f, 1, 1, 0.02)

```

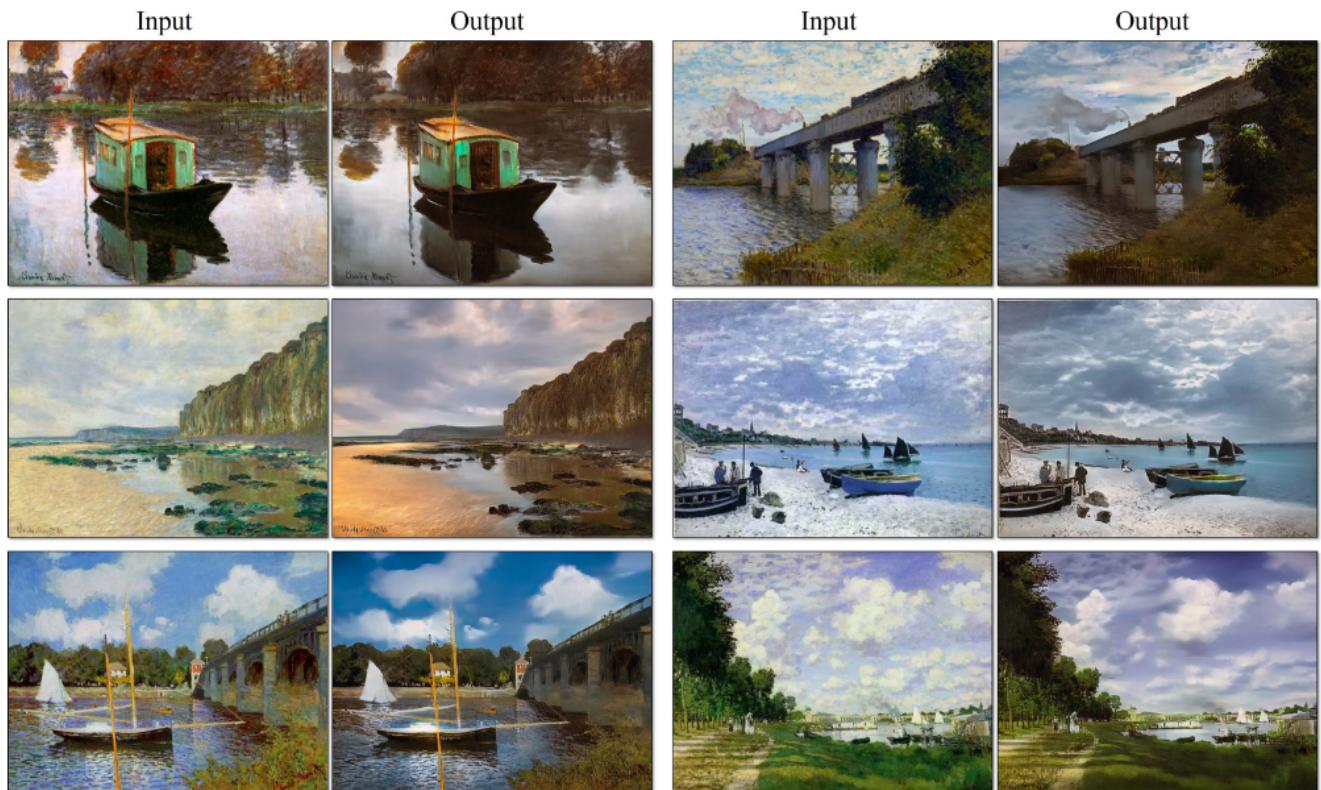
실험 결과



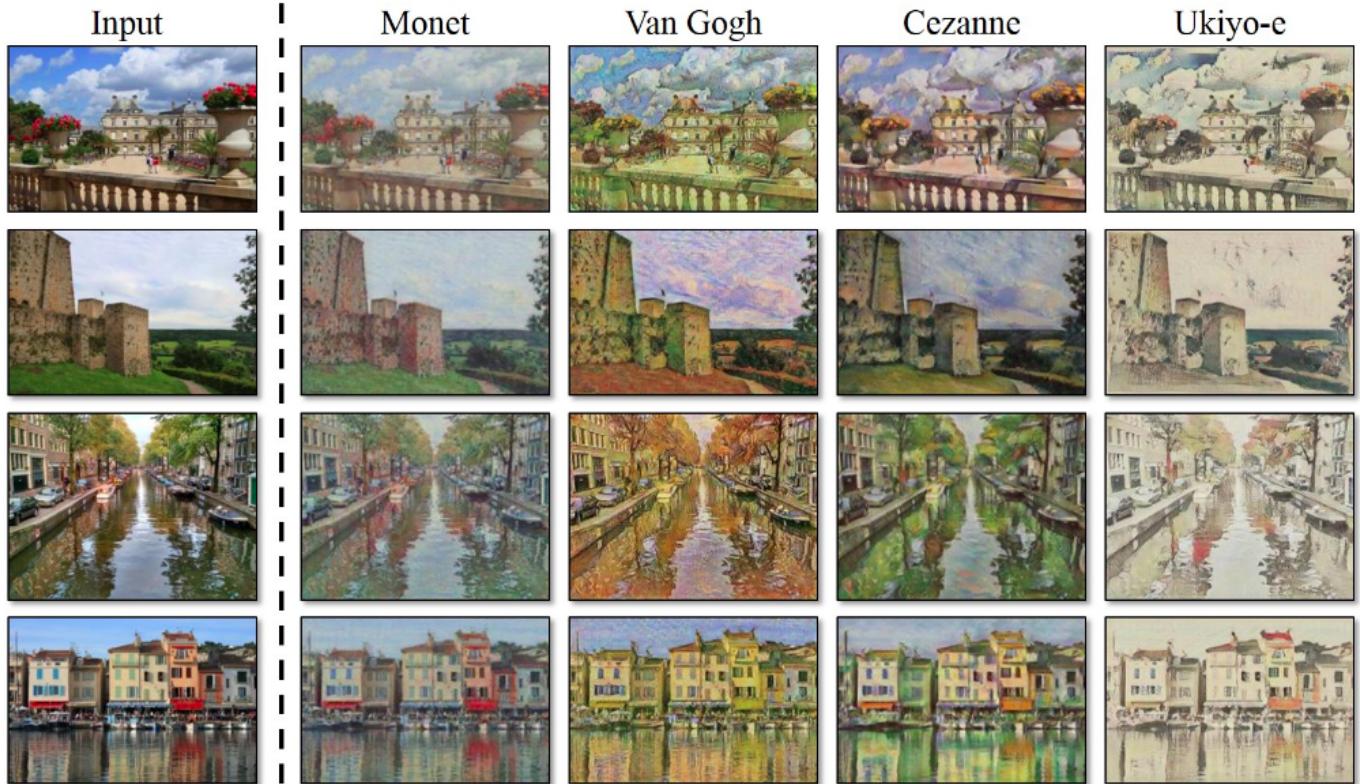


Application

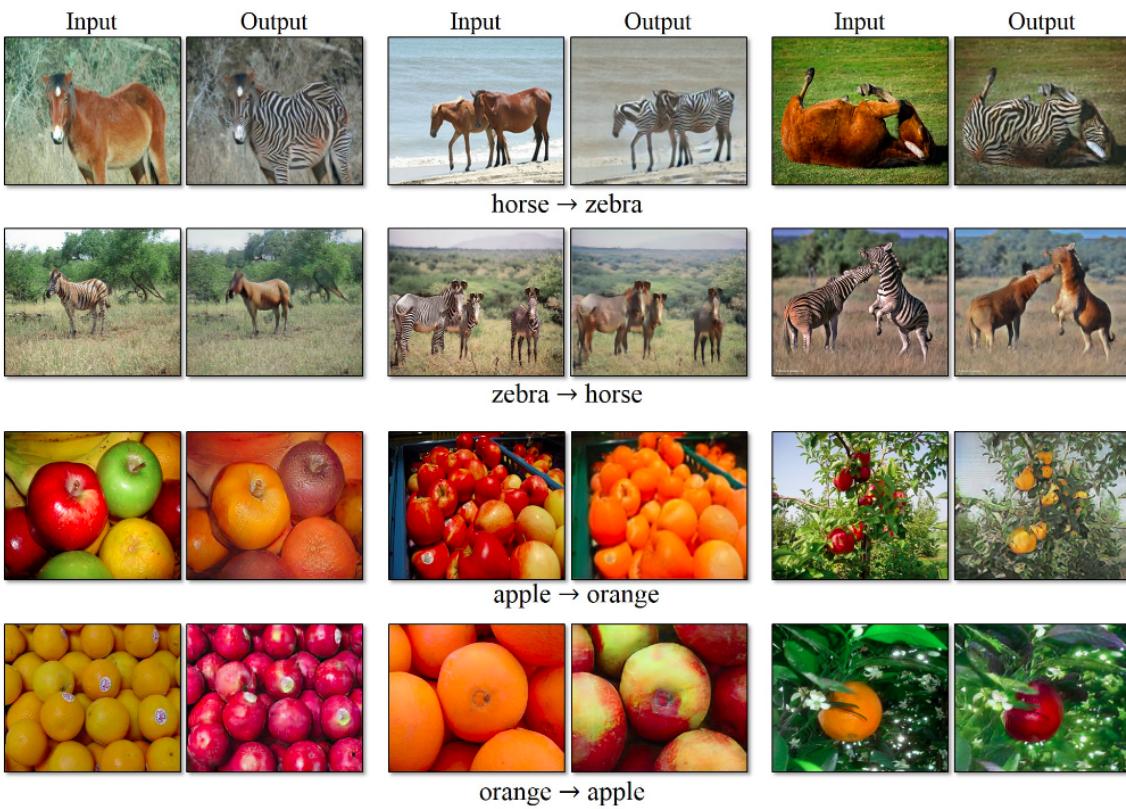
Monet Painting → Photo



Syle Transfer



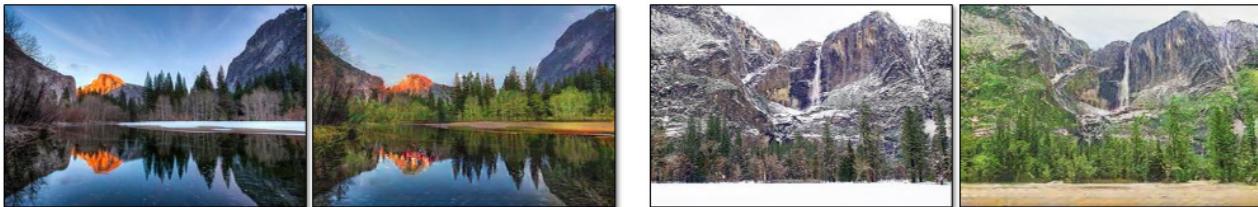
Object Transfiguration



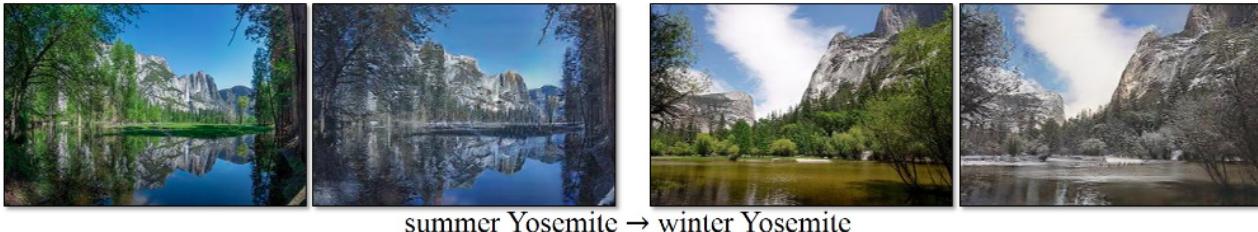
GTA → city scape



Season Transfer

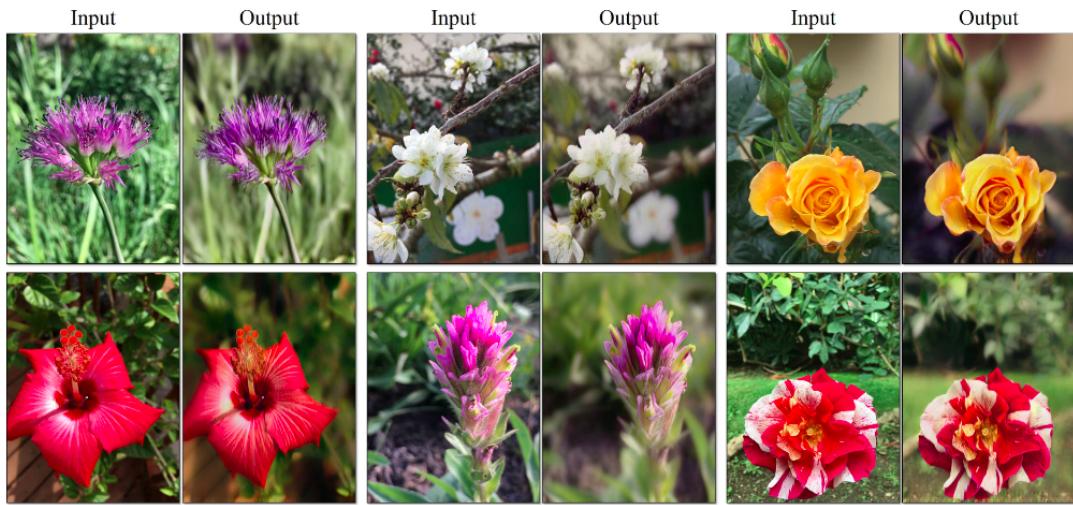


winter Yosemite → summer Yosemite



summer Yosemite → winter Yosemite

Photo Enhancement



Reference

- paper : <https://arxiv.org/pdf/1703.10593.pdf>
- homepage : <https://juyanz.github.io/CycleGAN/>
 - 코드로 설명 : <https://hardikbansal.github.io/CycleGANBlog/>