AI : 리뷰 - Cycle GAN

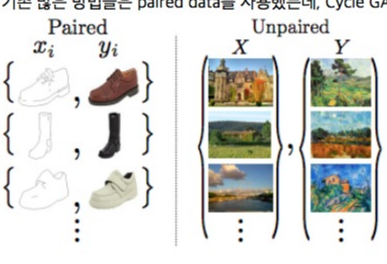
Created by Unknown User (8a7f808563eba3750163edfa6cb90018) on Jun 09, 2018

개요

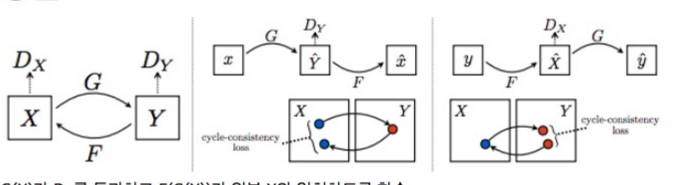
원제 : Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks

두 도메인간의 이미지 변형(image translation) 방법.
paired data가 없는 데이터를 사용.

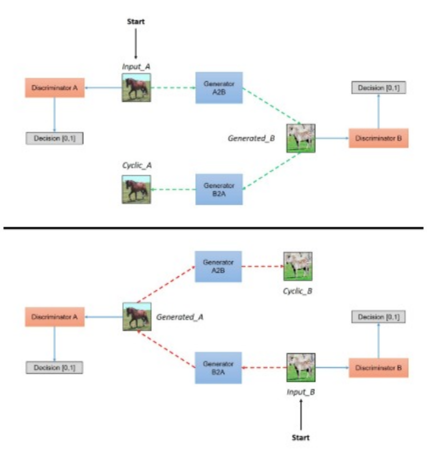
기존 많은 방법들은 paired data를 사용했는데, Cycle GAN으로 그런 paired가 없어도 가능함.



방법



G(X)가 Dy를 통과하고 F(G(X))가 원본 X와 일치하도록 학습.
F(Y)가 Dx를 통과하고 G(F(Y))가 원본 Y와 일치하도록 학습.



로스 펑션

X, Y : 2개의 도메인
2개의 generator

- G : X → Y
- F : Y → X

2개의 discriminator

- D_X
- D_Y

3개의 항목으로 구성

$$\mathcal{L}(G,F,D_X,D_Y) = \mathcal{L}_{\mathrm{GAN}}(G,D_Y,X,Y) + \mathcal{L}_{\mathrm{GAN}}(F,D_X,Y,X) + \lambda\mathcal{L}_{\mathrm{cyc}}(G,F),$$

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

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

실제 구현에서는 log를 사용하지 않은 다음을 사용.
G : $\mathbb{E}_{x\sim p_{\mathrm{data}}(x)}[(D(G(x))-1)^2]$
D : $\mathbb{E}_{y\sim p_{\mathrm{data}}(y)}[(D(y)-1)^2] + \mathbb{E}_{x\sim p_{\mathrm{data}}(x)}[D(G(x))^2]$

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

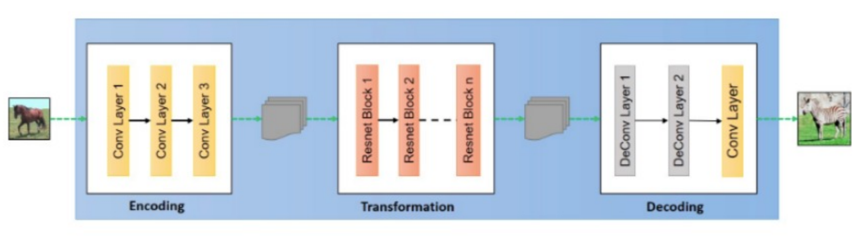
```
g_loss_B_1 = tf.reduce_mean(tf.squared_difference(dec_gen_A,1))
g_loss_A_1 = tf.reduce_mean(tf.squared_difference(dec_gen_A,1))


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()

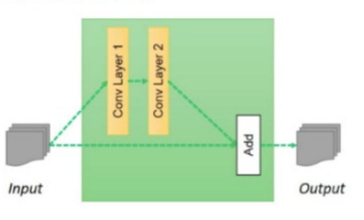
```
def general_conv2d(inputconv, o_d=64, f_h=7, f_w=7, s_h=1, s_w=1):
    with tf.variable_scope(name):
        conv = tf.contrib.layers.conv2d(inputconv, num_features, [window_width, window_height], [stride_width, stride_height],
                                        padding, activation_fn=None,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

def resnet_blocks()

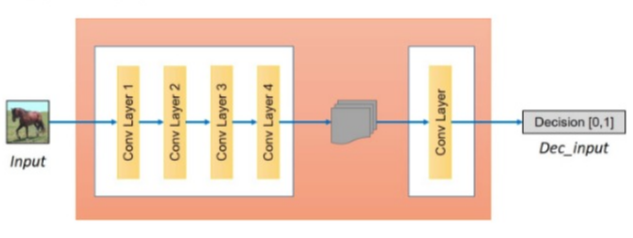
```
def resnet_blocks(input_res, num_features):

    out_res_1 = general_conv2d(input_res, num_features,
                               window_width=3,
                               window_heigth=3,
                               stride_width=1,
                               stride_heigth=1)
    out_res_2 = general_conv2d(out_res_1, num_features,
                               window_width=3,
                               window_heigth=3,
                               stride_width=1,
                               stride_heigth=1)
    return (out_res_2 + input_res)
```

Resnet Block



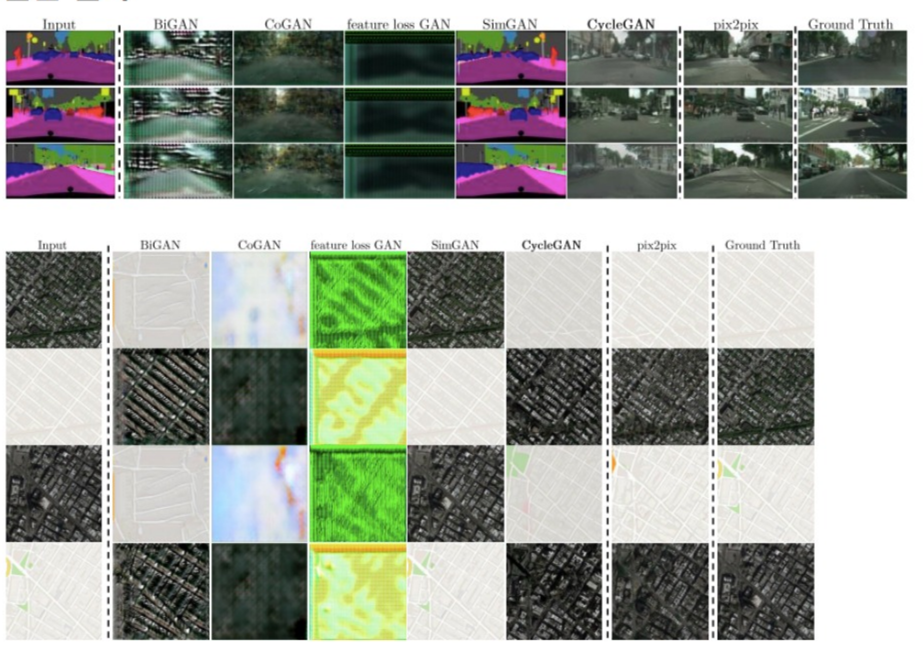
Discriminator



def build_descriminator()

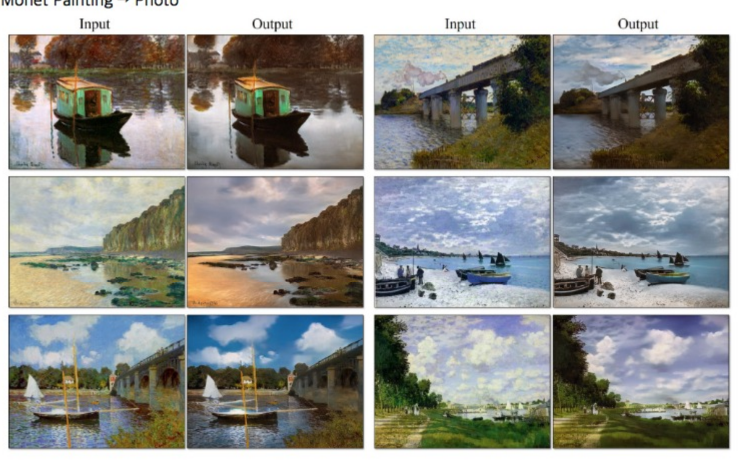
```
o_c1 = general_conv2d(input_disc, ndf, f, f, 2, 2)
o_c2 = general_conv2d(o_c1, ndf*2, f, f, 2, 2)
o_enc_A = general_conv2d(o_c2, ndf*4, f, f, 2, 2)
o_c4 = general_conv2d(o_enc_A, ndf*8, f, f, 2, 2)
decision = general_conv2d(o_c4, 1, f, f, 1, 1, 0.02)
```

실험 결과

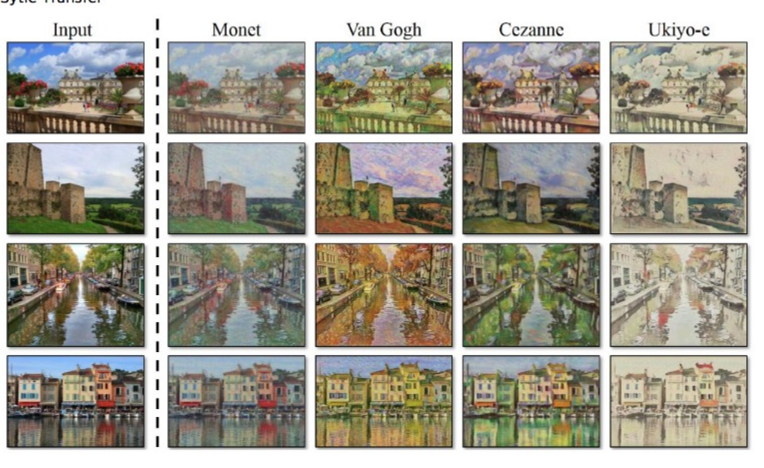


Application

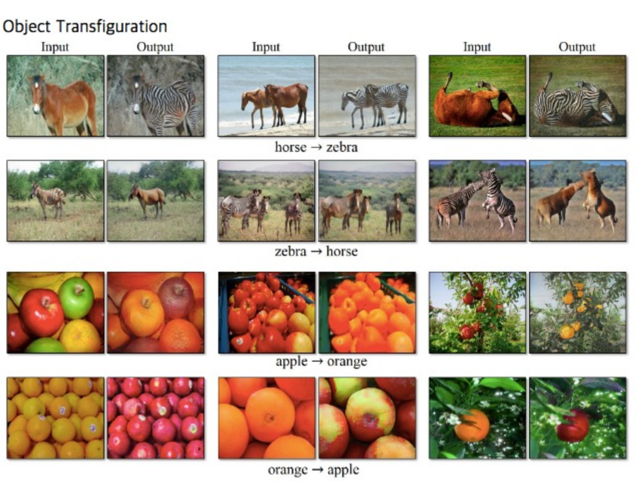
Monet Painting → Photo



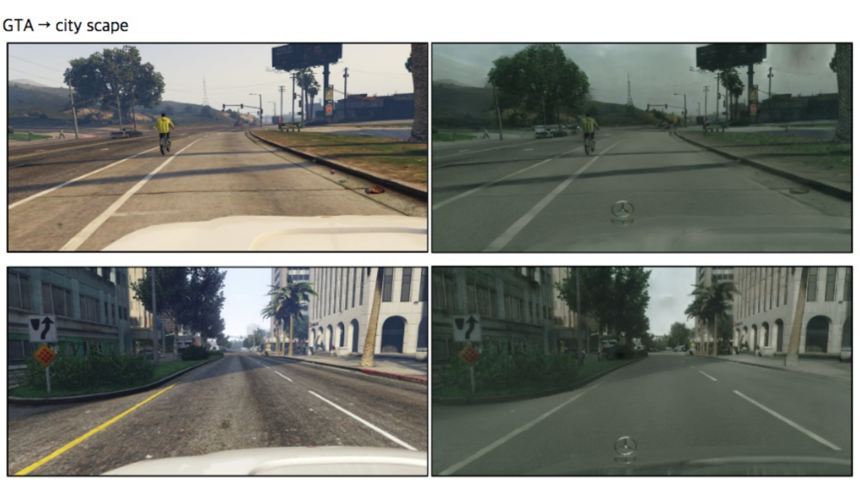
Sytle Transfer



Object Transfiguration



GTA → city scape



Season Transfer

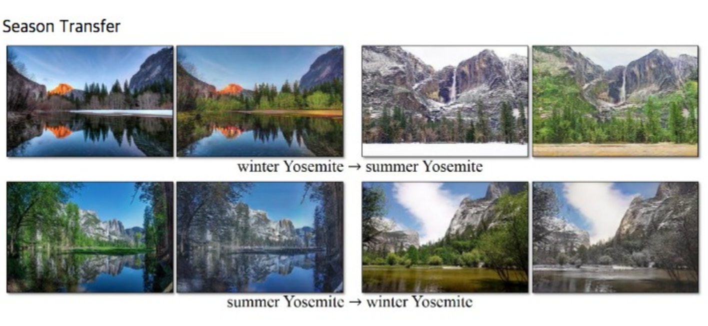


Photo Enhancement

Reference

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

Attachments:

- image2018-6-8_14-14-53.png (image/png)
- image2018-6-8_14-15-17.png (image/png)
- image2018-6-8_14-18-51.png (image/png)
- image2018-6-8_14-19-48.png (image/png)
- image2018-6-8_14-20-21.png (image/png)
- image2018-6-8_14-23-1.png (image/png)
- image2018-6-8_14-23-46.png (image/png)
- image2018-6-8_14-24-9.png (image/png)
- image2018-6-8_14-25-51.png (image/png)
- image2018-6-8_14-27-0.png (image/png)
- image2018-6-8_14-27-57.png (image/png)
- image2018-6-8_14-28-41.png (image/png)
- image2018-6-8_14-54-9.png (image/png)
- image2018-6-8_14-54-42.png (image/png)
- image2018-6-8_14-54-59.png (image/png)
- image2018-6-8_14-56-38.png (image/png)
- image2018-6-8_14-57-26.png (image/png)
- image2018-6-8_15-0-11.png (image/png)
- image2018-6-8_15-0-54.png (image/png)
- image2018-6-8_15-2-43.png (image/png)
- image2018-6-8_15-3-28.png (image/png)
- image2018-6-8_15-3-46.png (image/png)
- image2018-6-8_15-19-15.png (image/png)