Sketch 2 Photo

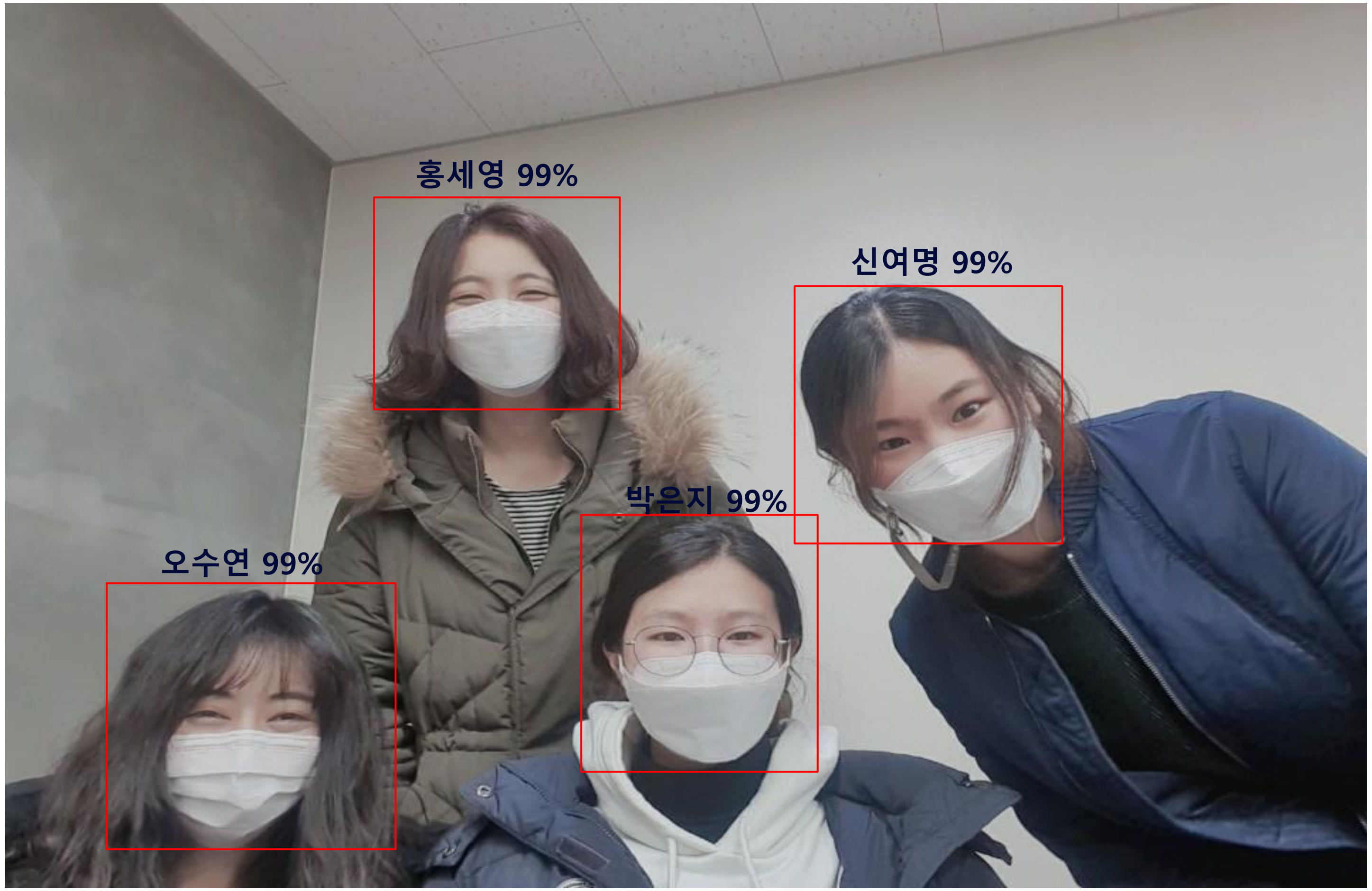
팀: GAN 때문이야

팀장 신여명

홍세영, 박은지, 오수연

GAN
때문이야

팀원 소개



목차

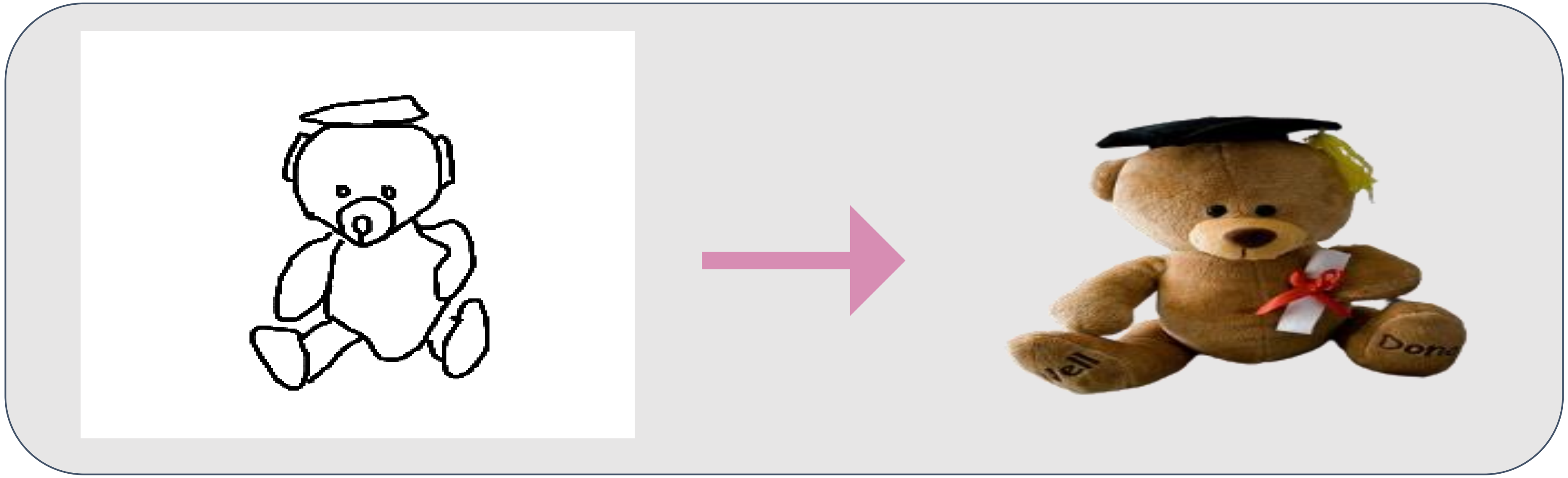
아이디어

구현기술

결과 및 시연

아이디어

- GAN를 이용한 이미지 - 이미지 번역
- 모델이 유저가 그린 스케치를 보고 실제 사진처럼 변환한 이미지를 출력할 수 있을까?



개발환경

HTML

JS

matplotlib

Flask
web development,
one drop at a time

glob

python™

NumPy

GitHub

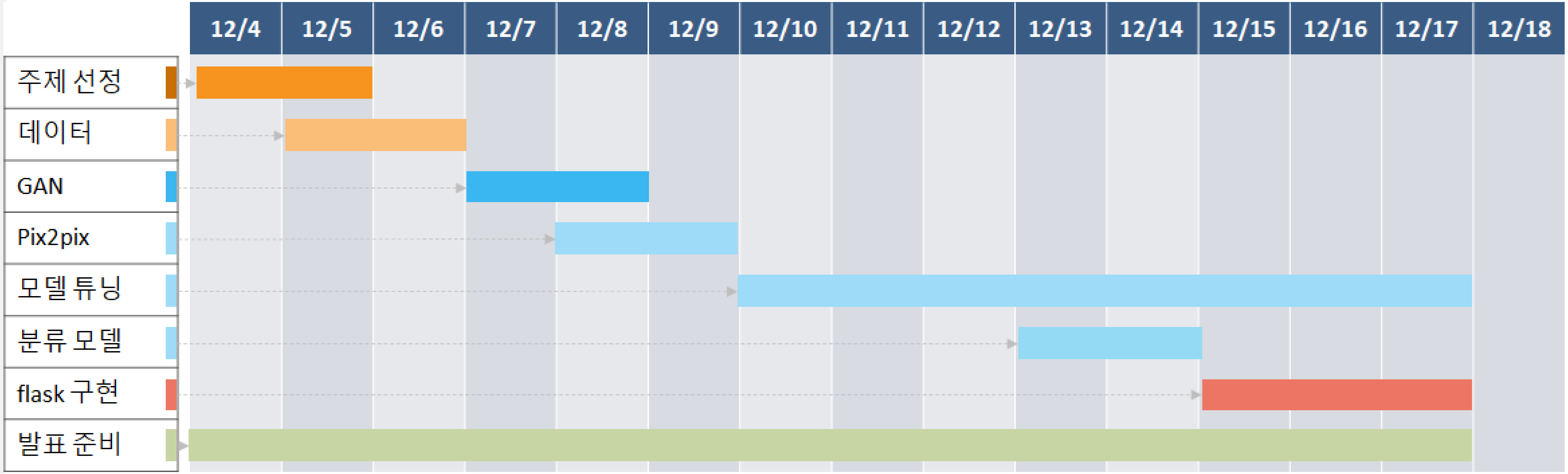
scikit learn

K Keras

OpenCV

TensorFlow

프로젝트 일정


12/4
12/5
12/6
12/7
12/8
12/9
12/10
12/11
12/12
12/13
12/14
12/15
12/16
12/17
12/18
주제 선정
데이터
GAN
Pix2pix
모델 튜닝
분류 모델
flask 구현
발표 준비

참조

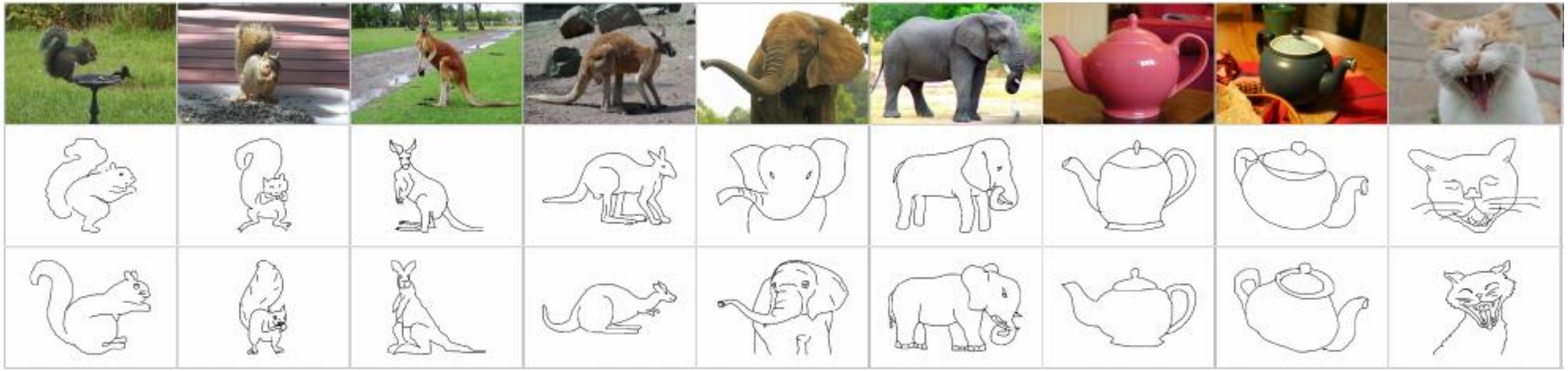
관련 논문

- Sketch to Image Translation using GANs
  by Lisa Fan
- Image-to-Image Translation with Conditional Adversarial Networks *by Berkeley AI Research (BAIR) Laboratory*

Tensorflow 공식 문서의 Pix2Pix

데이터 : Georgia Tech의 sketchy 예제
 - 스케치/사진 데이터 쌍

데이터 수집



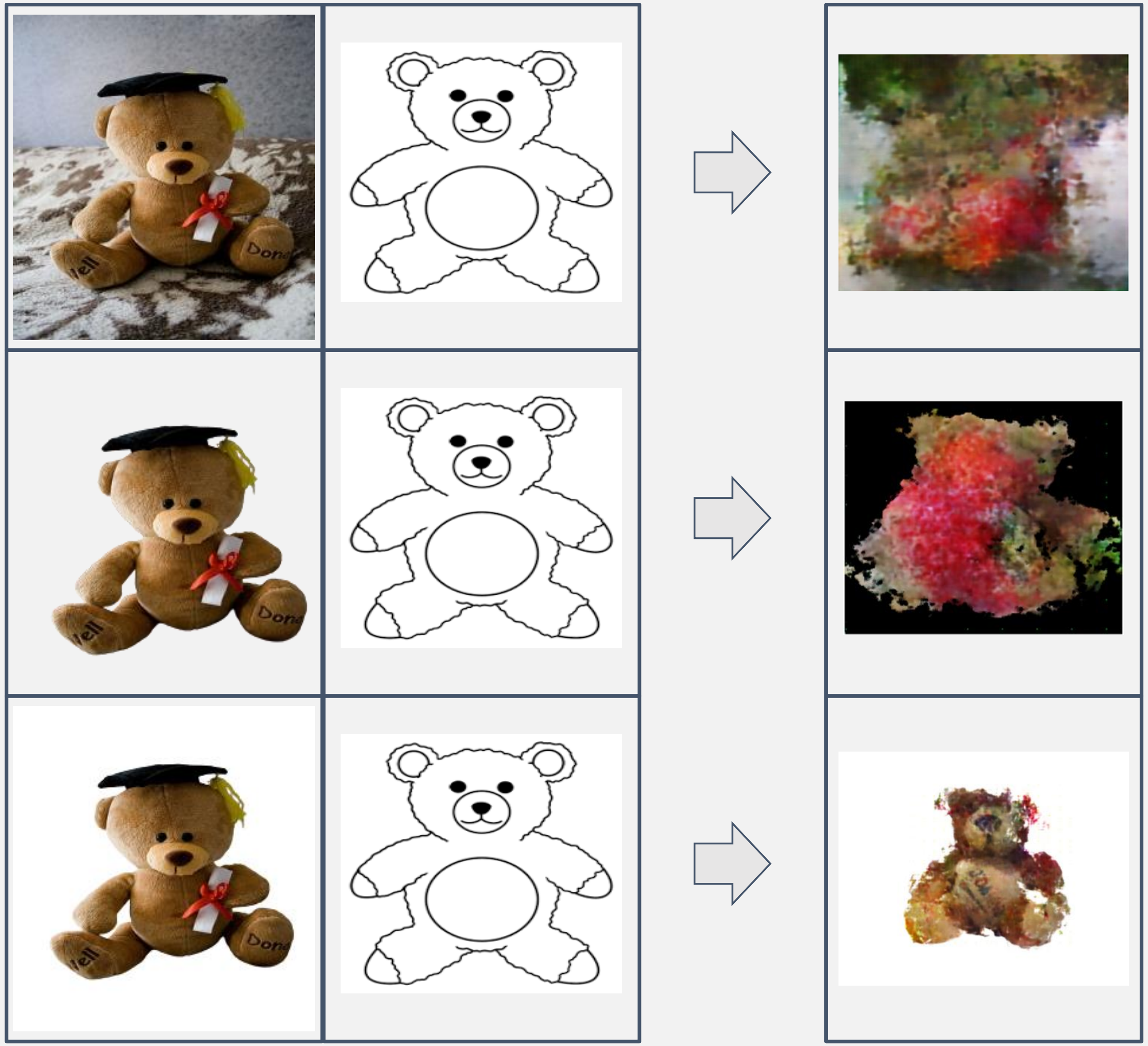
- Georgia Tech의 스케치-사진 페어(쌍) 데이터
- 125개의 카테고리
- 12,500개의 사진
- 75,471개의 사진의 스케치

Image segmentation

default
배경 O

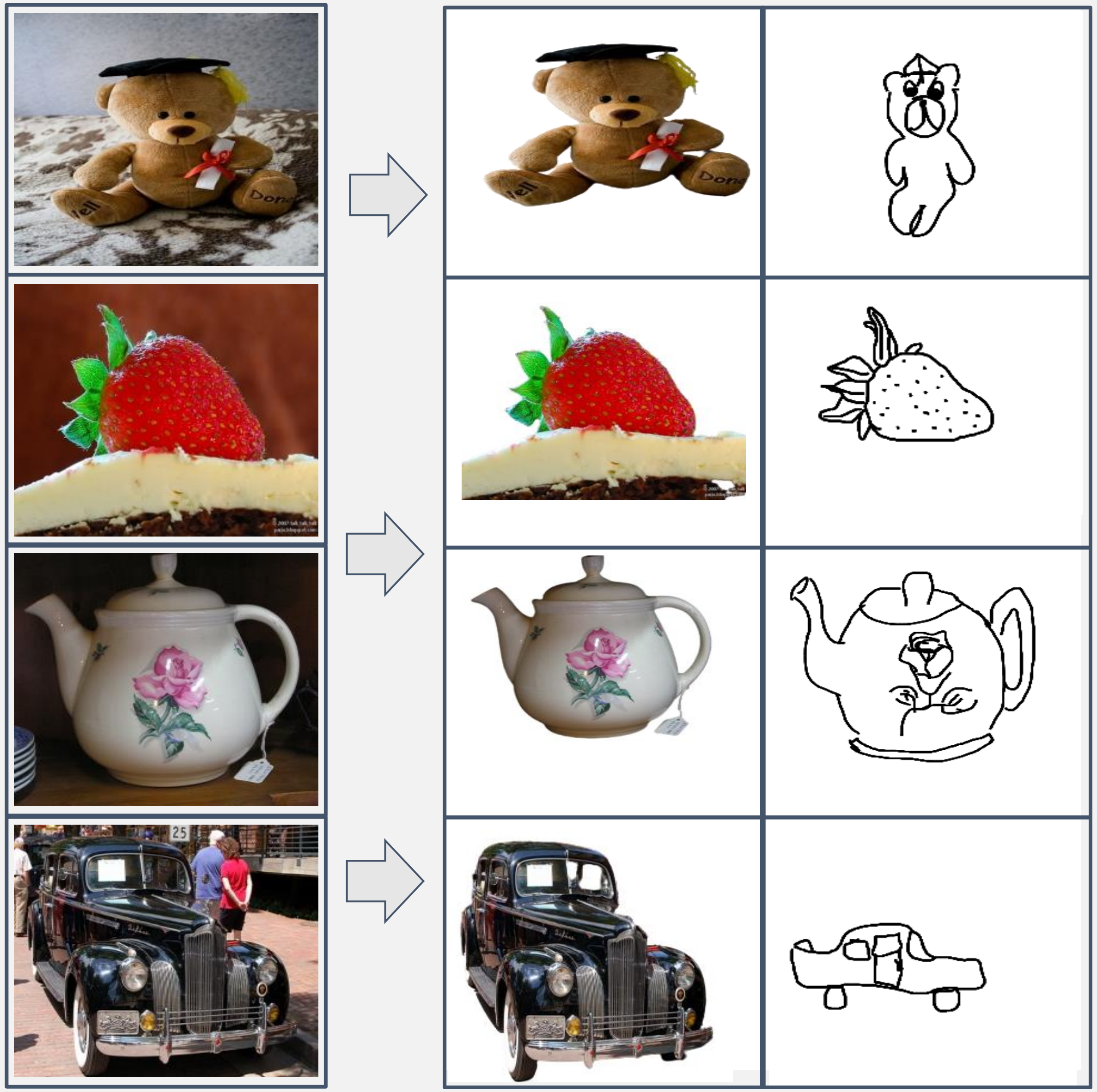
배경 X

흰 배경



- 이미지 내에 있는 객체들을 의미있는 단위로 분할하는 과정
- 입력 사진의 전체를 학습하기 때문에 생성기가 우리가 학습하고 싶은 객체에 집중하는 것이 아니라 배경도 같이 학습(모방)하는 결과를 보임
- 입력 사진을 주요 객체만 포함하도록 자르면 높은 품질의 출력 이미지가 생성되지 않을까 추측

훈련 데이터 셋



이미지 - 스케치 페어(쌍) 예시

- 4개의 카테고리 선정
  - 곰인형, 딸기, 찻주전자,자동차
- 최종 training set
  - 스케치 : 9148장 (256*256*3)
  - 사진 : 9148장 (256*256*3)

구현기술

- Pix2Pix
  - Image-to-Image Translation with Conditional Adversarial Networks (Pix2Pix), Phillip Isola 등, 2017
  - Conditional GAN에 기반을 둔 image-to-image translation model
  - **기본적으로 DCGAN의 모델을 사용**

Generative Adversarial Networks
(GAN, 생성적 적대 신경망)

↓

Deep **Convolutional** Generative Adversarial Networks
(DCGAN)

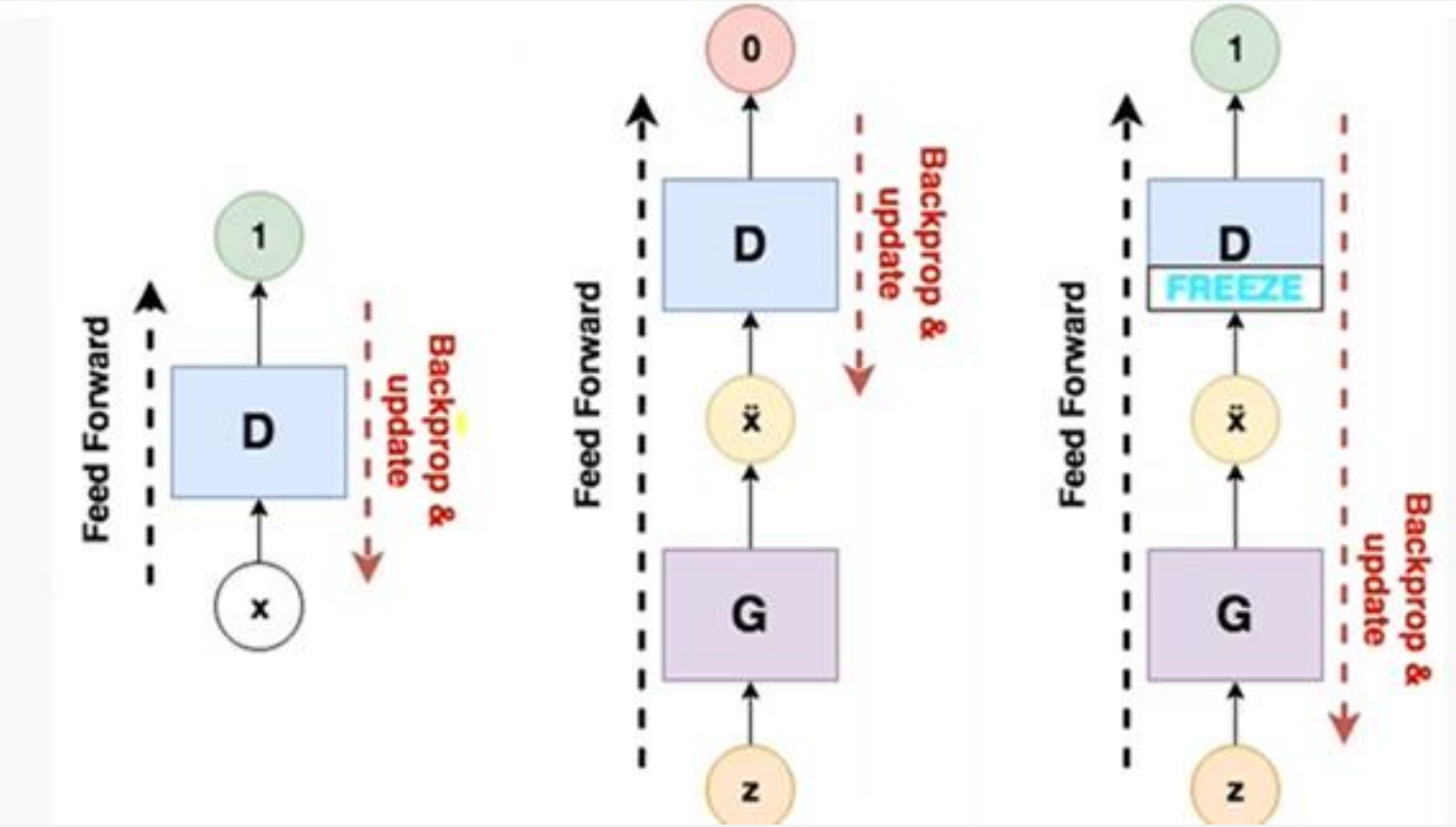
↓

Conditional Generative Adversarial Nets
(cGAN)

↓

Image-to-Image Translation with **Conditional** Adversarial Networks
(Pix2Pix)

구현기술



Generative Adversarial Networks
(GAN, 생성적 적대 신경망)

⇩

Deep **Convolutional** Generative Adversarial Networks
(DCGAN)

⇩

Conditional Generative Adversarial Nets
(cGAN)

⇩

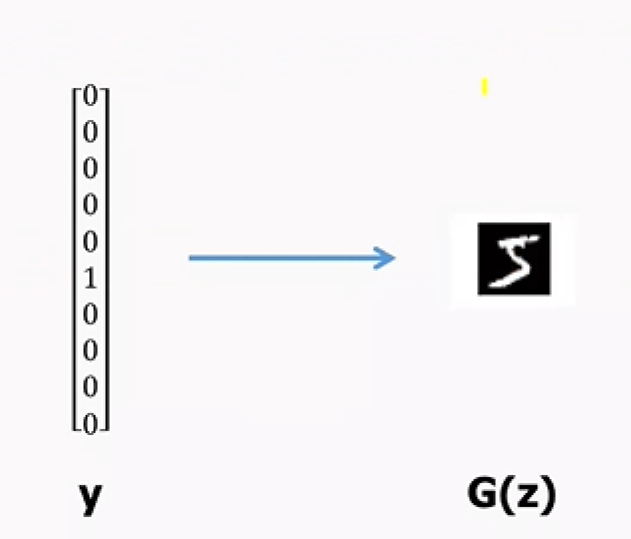
Image-to-Image Translation with **Conditional** Adversarial Networks
(Pix2Pix)

DCGAN : 기존의 GAN에 CNN을 적용하여 네트워크 구조를 발전시켜 최초로 고화질 영상을 생성

구현기술

In an (Unconditioned) Generative Model,

there is *no control* on *modes of the data* being generated



Generative Adversarial Networks
(GAN, 생성적 적대 신경망)

⇩

Deep **Convolutional** Generative Adversarial Networks
(DCGAN)

⇩

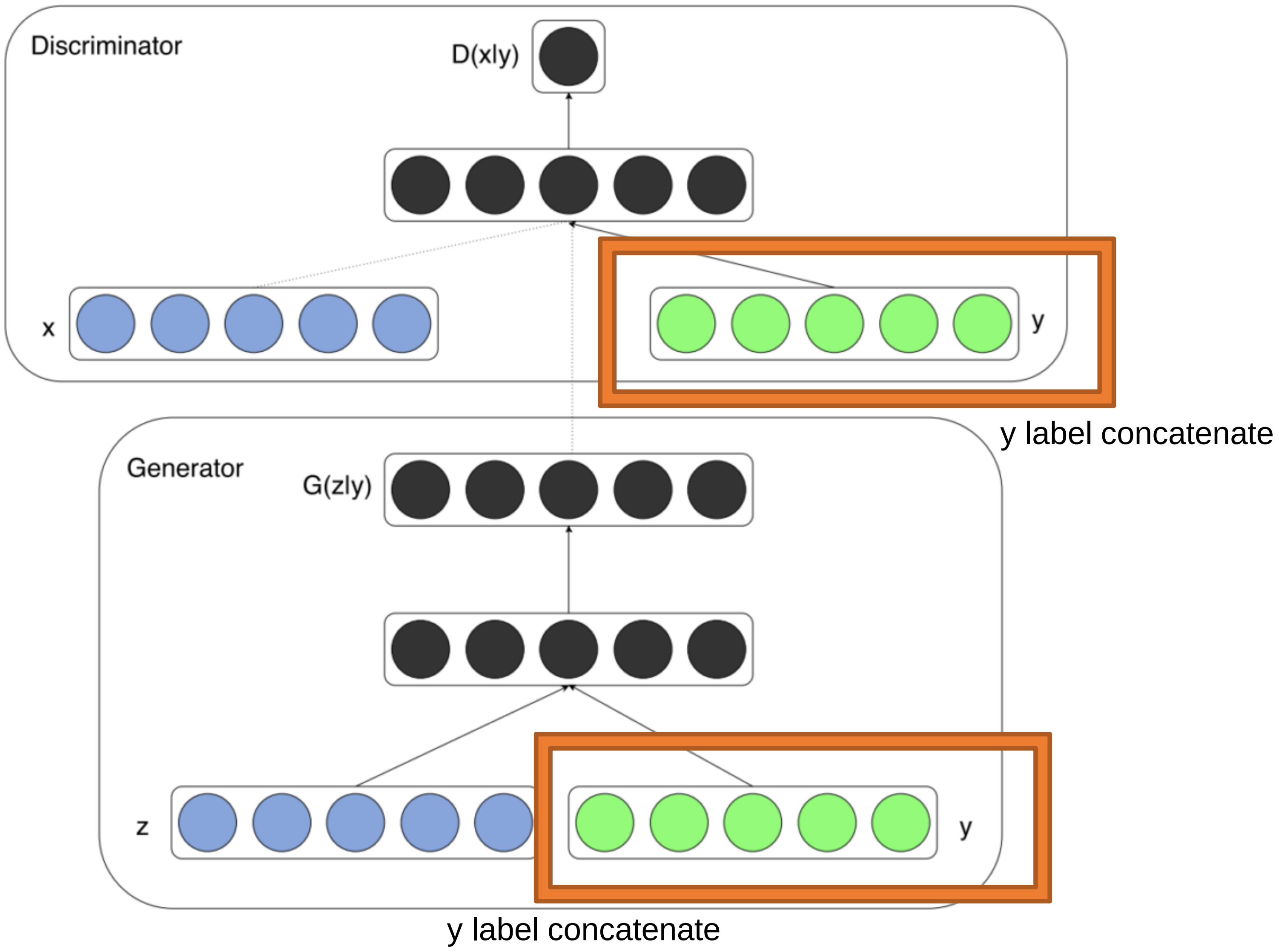
Conditional Generative Adversarial Nets
(cGAN)

⇩

Image-to-Image Translation with **Conditional** Adversarial Networks
(Pix2Pix)

cGAN : 클래스 라벨이나 문장 특징으로 조건을 달아서 해당 이미지를 생성

구현기술



Generative Adversarial Networks
(GAN, 생성적 적대 신경망)

⇩

Deep **Convolutional** Generative Adversarial Networks
(DCGAN)

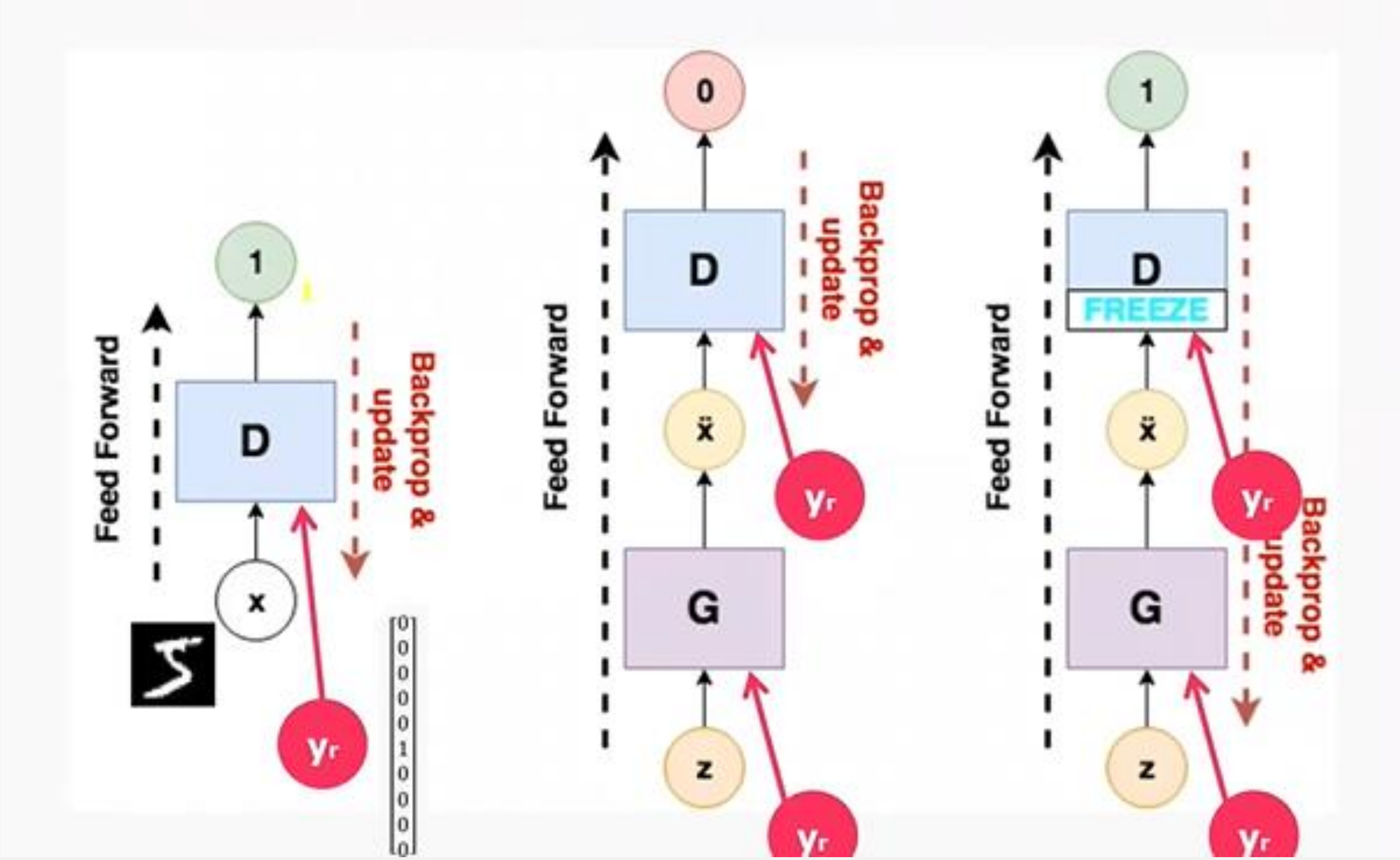
⇩

Conditional Generative Adversarial Nets
(cGAN)

⇩

Image-to-Image Translation with **Conditional** Adversarial Networks
(Pix2Pix)

구현기술



Both the Generator and Discriminator
are conditioned on some extra information **y**

where **y** : **label** or "**data from other modalities**"
→ Pix2Pix 로 발전

Generative Adversarial Networks
(GAN, 생성적 적대 신경망)

⇩

Deep **Convolutional** Generative Adversarial Networks
(DCGAN)

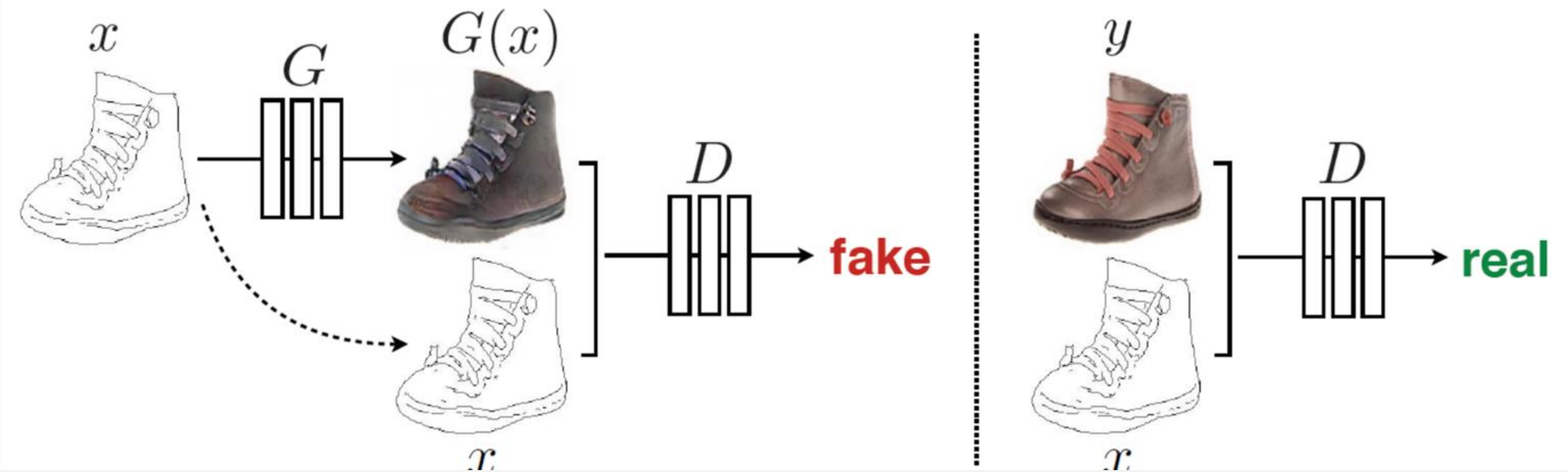
⇩

Conditional Generative Adversarial Nets
(cGAN)

⇩

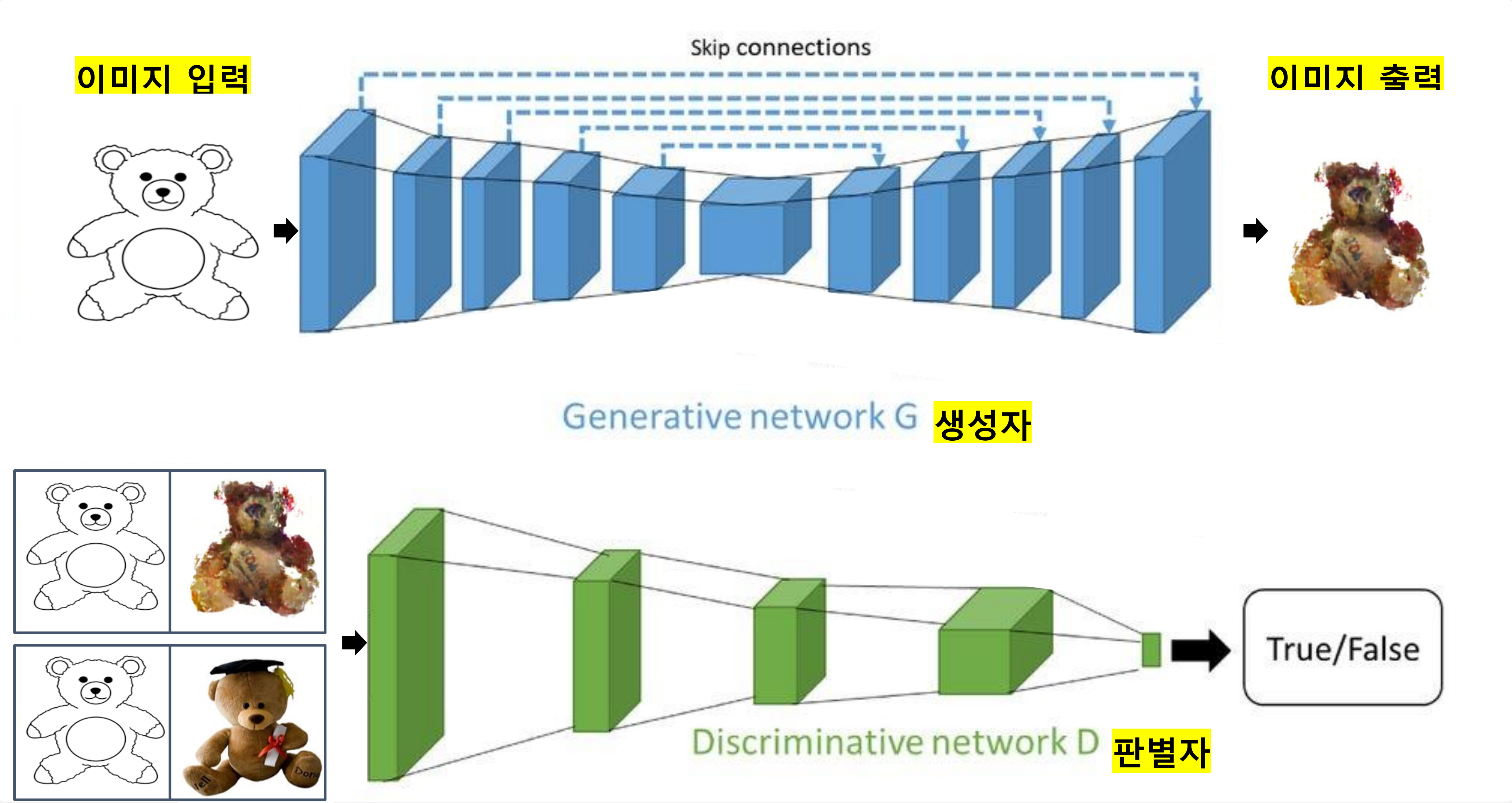
Image-to-Image Translation with **Conditional** Adversarial Networks
(Pix2Pix)

Pix2Pix 네트워크 구조

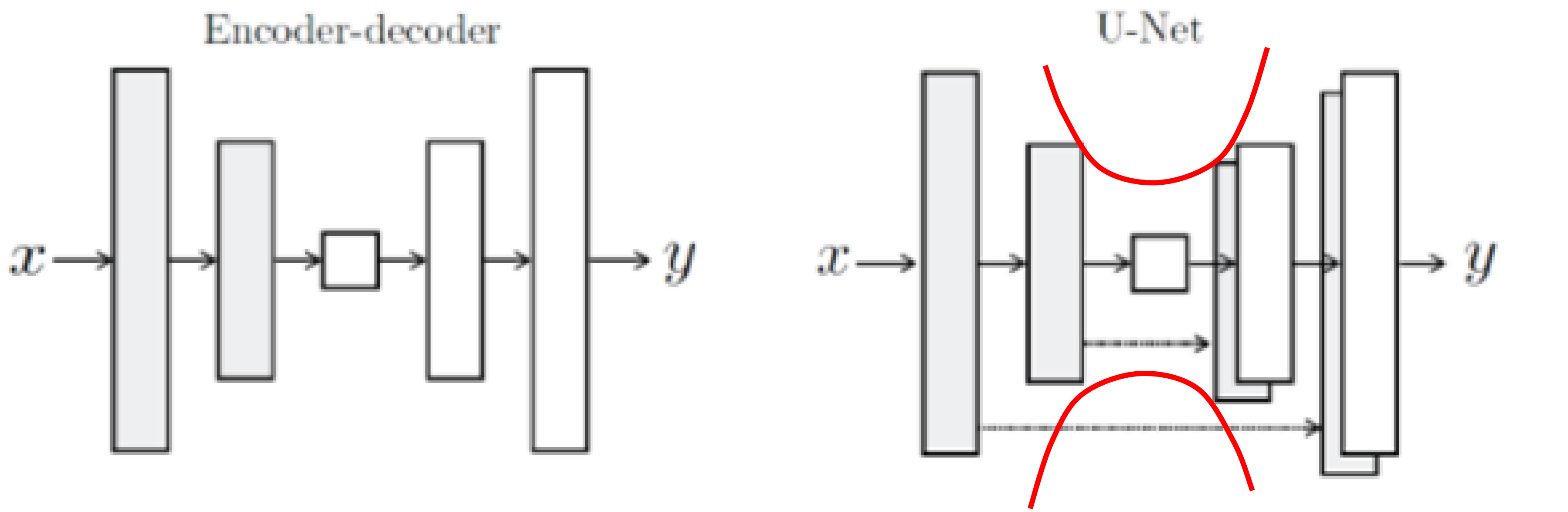


결국 cGAN에서 Discriminator는
(data x, condition data y) pair가
(data G(z), condition data y) pair와
matching이 되는지를 판단

Pix2Pix 네트워크 구조

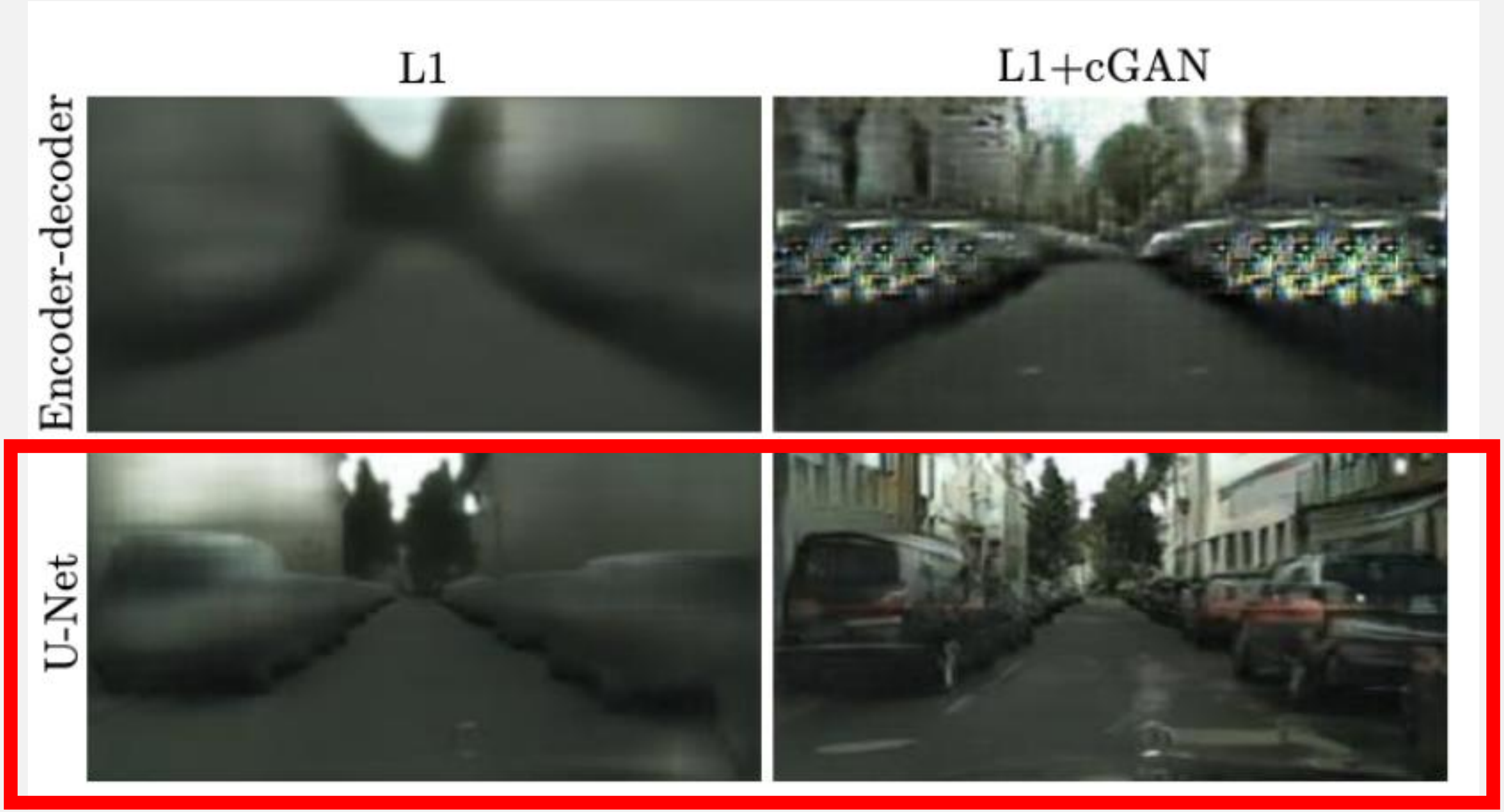


생성자 (Generator)

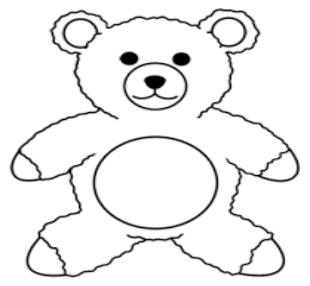


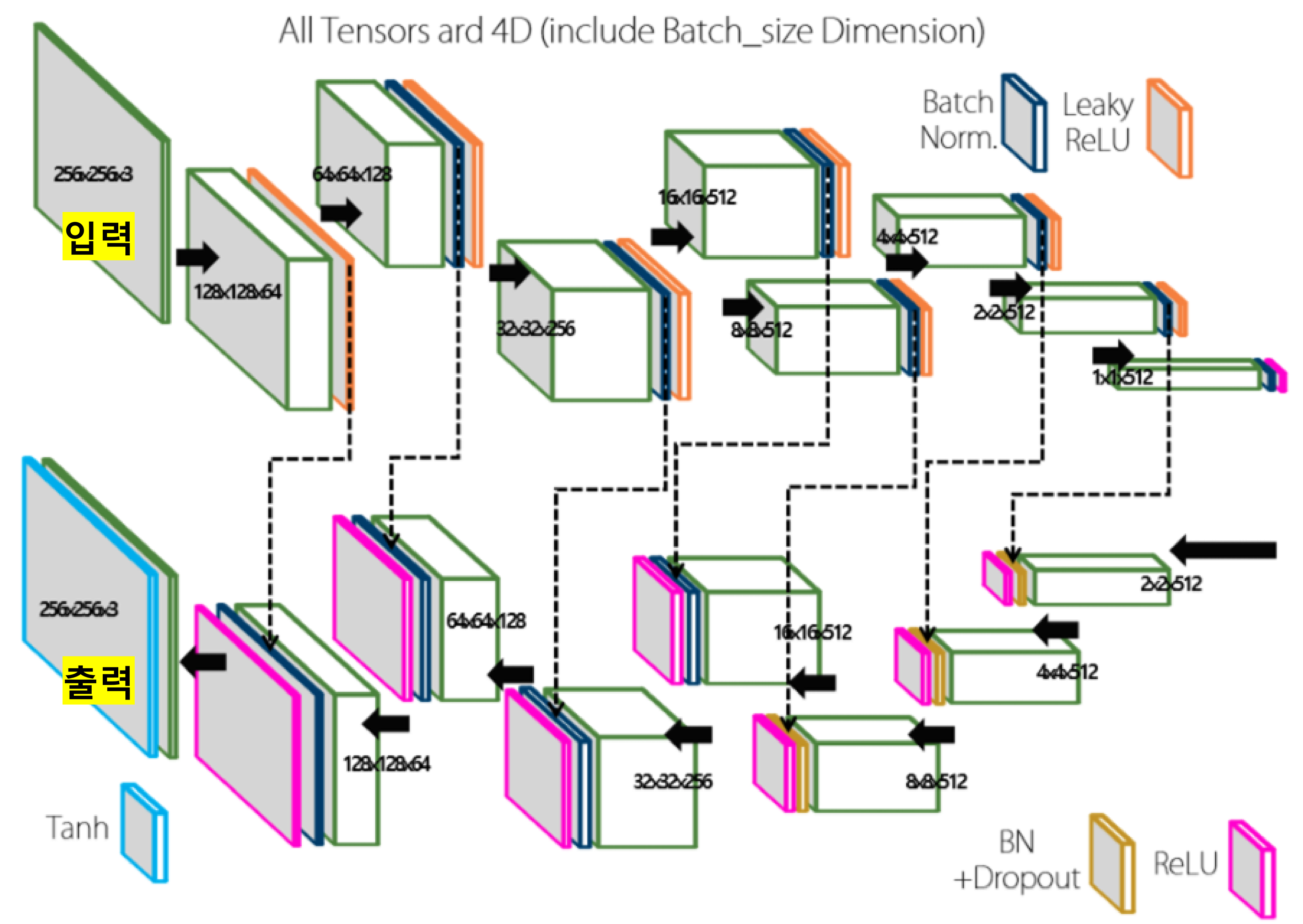
- input과 output이 전부 이미지
  - 사이즈가 줄어들었다가 다시 커지는 구조 (Encoder - Decoder)
    - 중앙의 Feature Dimension이 input보다 작기 때문에 정보의 손실이 발생
- U - Net 구조 (skip-connection 추가된 Encoder - Decoder 형태의 네트워크)
  - Decoder의 Layer들에게 정보가 손실 되기 전 Encoder Layer의 Feature들을 제공하여 참고하게 함
  - i 번째 레이어와 n−i 번째 레이어를 연결 **(concatenate)**
  - encoder -> decoder 직접 정보를 넘기기 때문에 훨씬 선명한 결과

U-Net



U - Net (skip-connection)



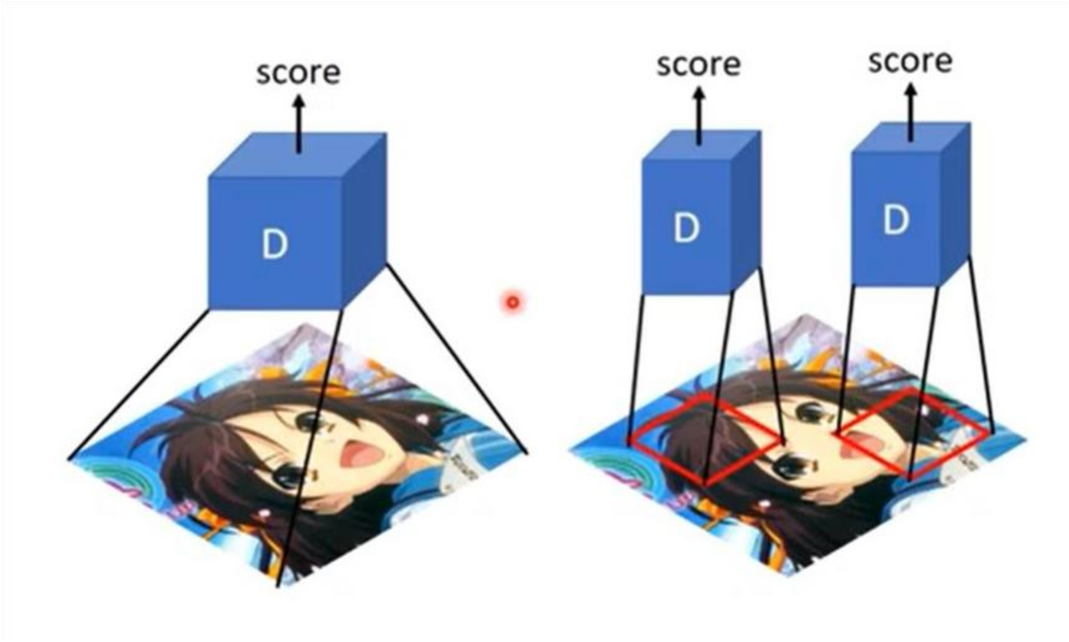


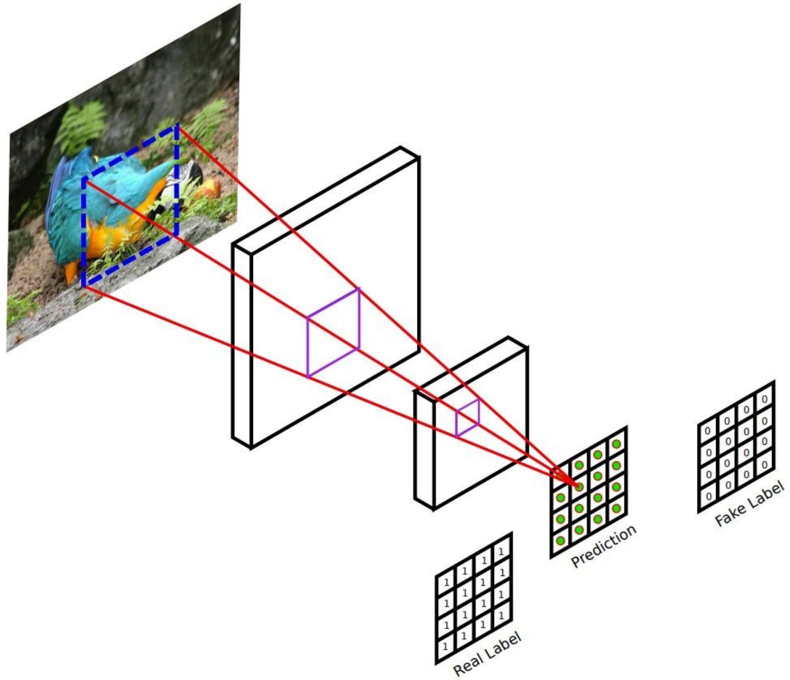
U - Net (skip-connection)

Layer (type)	Output Shape	Param #	Connected to
input_3 (InputLayer)	[(None, 256, 256, 3)	0	
conv2d_6 (Conv2D)	(None, 128, 128, 64)	3136	input_3[0][0]
leaky_re_lu_5 (LeakyReLU)	(None, 128, 128, 64)	0	conv2d_6[0][0]
conv2d_7 (Conv2D)	(None, 64, 64, 128)	131200	leaky_re_lu_5[0][0]
batch_normalization_4 (BatchNor	(None, 64, 64, 128)	512	conv2d_7[0][0]
leaky_re_lu_6 (LeakyReLU)	(None, 64, 64, 128)	0	batch_normalization_4[0][0]
conv2d_8 (Conv2D)	(None, 32, 32, 256)	524544	leaky_re_lu_6[0][0]
. . .			
leaky_re_lu_10 (LeakyReLU)	(None, 4, 4, 512)	0	batch_normalization_8[0][0]
conv2d_12 (Conv2D)	(None, 2, 2, 512)	4194816	leaky_re_lu_10[0][0]
batch_normalization_9 (BatchNor	(None, 2, 2, 512)	2048	conv2d_12[0][0]
leaky_re_lu_11 (LeakyReLU)	(None, 2, 2, 512)	0	batch_normalization_9[0][0]
conv2d_13 (Conv2D)	(None, 1, 1, 512)	4194816	leaky_re_lu_11[0][0]
activation_1 (Activation)	(None, 1, 1, 512)	0	conv2d_13[0][0]
conv2d_transpose (Conv2DTranspo	(None, 2, 2, 512)	4194816	activation_1[0][0]
. . .			
activation_6 (Activation)	(None, 32, 32, 512)	0	concatenate_5[0][0]
conv2d_transpose_5 (Conv2DTrans	(None, 64, 64, 128)	1048704	activation_6[0][0]
batch_normalization_15 (BatchNo	(None, 64, 64, 128)	512	conv2d_transpose_5[0][0]
concatenate_6 (Concatenate)	(None, 64, 64, 256)	0	batch_normalization_15[0][0] leaky_re_lu_6[0][0]
activation_7 (Activation)	(None, 64, 64, 256)	0	concatenate_6[0][0]
conv2d_transpose_6 (Conv2DTrans	(None, 128, 128, 64)	262208	activation_7[0][0]
batch_normalization_16 (BatchNo	(None, 128, 128, 64)	256	conv2d_transpose_6[0][0]
concatenate_7 (Concatenate)	(None, 128, 128, 128	0	batch_normalization_16[0][0] leaky_re_lu_5[0][0]
activation_8 (Activation)	(None, 128, 128, 128	0	concatenate_7[0][0]
conv2d_transpose_7 (Conv2DTrans	(None, 256, 256, 3)	6147	activation_8[0][0]
activation_9 (Activation)	(None, 256, 256, 3)	0	conv2d_transpose_7[0][0]

판별자 (Discriminator)

- patchGAN의 구조를 사용
- 기존의 DCGAN과는 다르게 이미지의 작은 patch에 대해서 판단하여 각 patch 별로의 real/fake 여부를 판단
- 이미지 전체를 연산하는 것보다 연산의 수가 적고 빠름
- generator가 각각의 이미지 patch 조각들의 진위여부를 속이기 위해서 학습
  - output image가 더 high resolution (고해상도) 을 가지게 됨



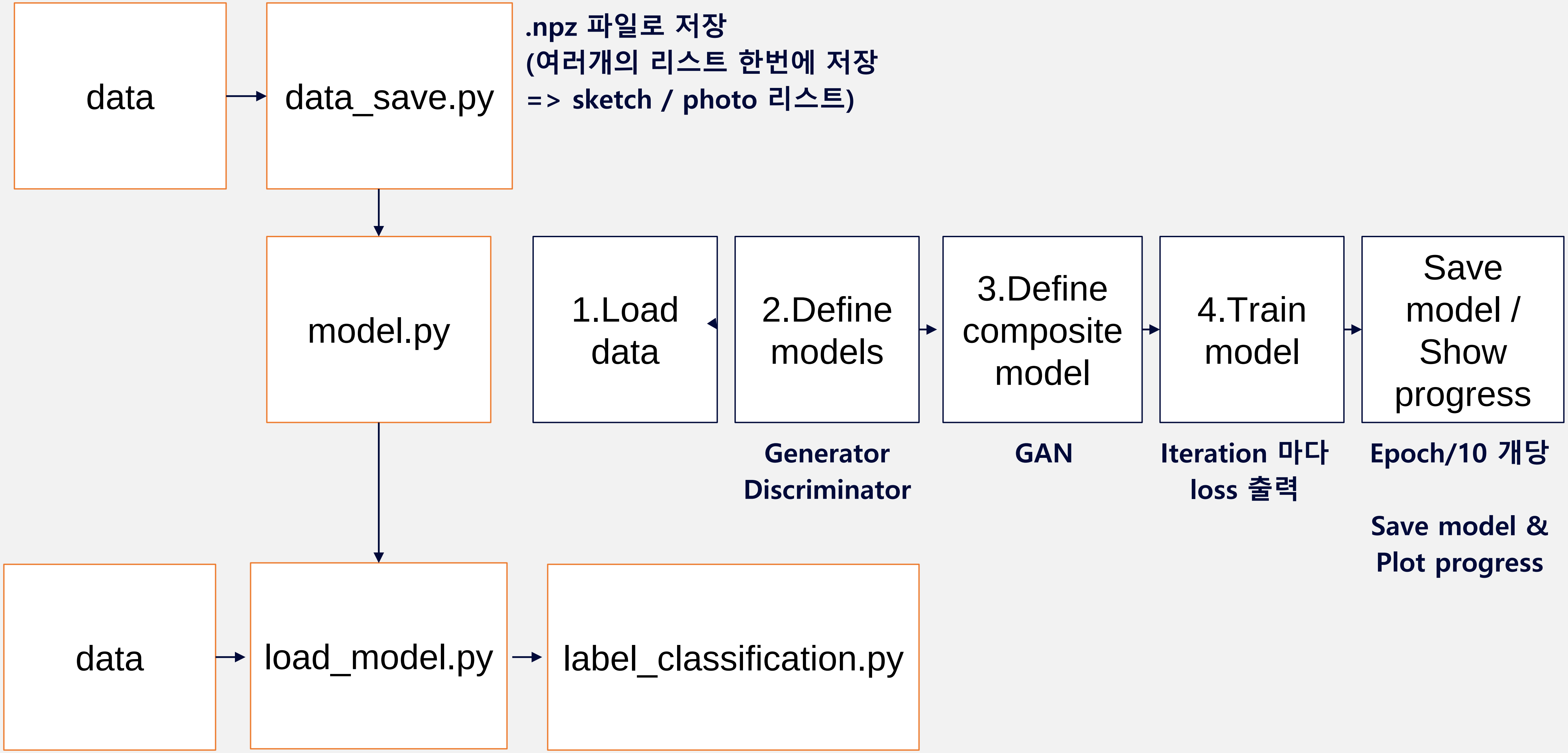


patchGAN

Discriminator receptive field	Per-pixel acc.	Per-class acc.	Class IOU
1×1	0.44	0.14	0.10
16×16	0.62	0.20	**0.16**
70×70	**0.63**	**0.21**	**0.16**
256×256	0.47	0.18	0.13

- 70x70의 patch size로 나누어 진위여부를 판단한 경우,
- 기존의 1x1의 pixelGAN / 286x286의 imageGAN 구조보다 더 선명한 이미지를 만들어낸다

Code Summary



생성자

● DCGAN 연구진들이 다양한 실험을 통해 알아낸 최적의 결과를 내는 Generator 네트워크 구조

1. Max pooling layer를 없애고 strided convolution을 통해 feature map의 크기를 조절한다.
2. Batch Normalization을 적용한다.
3. Fully connected hidden layer를 제거한다.
4. Generator의 마지막 활성함수로 Tanh를 사용하고, 나머지는 ReLU를 사용한다.
5. Discriminator의 활성함수로 LeakyReLU를 사용한다.

Architecture guidelines for stable Deep Convolutional GANs

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

안정적인 GAN 트레이닝을 위한 팁

생성자

```python
# 생성자 모델
def define_generator(image_shape=(256,256,3)):
    init = RandomNormal(stddev=0.02)
    in_image = Input(shape=image_shape)
    # encoder model
    e1 = define_encoder_block(in_image, 64, batchnorm=False)
    e2 = define_encoder_block(e1, 128)
    e3 = define_encoder_block(e2, 256)
    e4 = define_encoder_block(e3, 512)
    e5 = define_encoder_block(e4, 512)
    e6 = define_encoder_block(e5, 512)
    e7 = define_encoder_block(e6, 512)
    # 병목현상방지
    b = Conv2D(512, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(e7)
    b = Activation('relu')(b)
    # decoder model
    d1 = decoder_block(b, e7, 512)
    d2 = decoder_block(d1, e6, 512)
    d3 = decoder_block(d2, e5, 512)
    d4 = decoder_block(d3, e4, 512, dropout=False)
    d5 = decoder_block(d4, e3, 256, dropout=False)
    d6 = decoder_block(d5, e2, 128, dropout=False)
    d7 = decoder_block(d6, e1, 64, dropout=False)
    # output
    g = Conv2DTranspose(3, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(d7)
    out_image = Activation('tanh')(g)
    # define model
    model = Model(in_image, out_image)
    return model
```

생성자

```python
# 생성자 모델
def define_generator(image_shape=(256,256,3)):
    init = RandomNormal(stddev=0.02)
    in_image = Input(shape=image_shape)
    # encoder model
    e1 = define_encoder_block(in_image, 64, batchnorm=False)
    e2 = define_encoder_block(e1, 128)
    e3 = define_encoder_block(e2, 256)
    e4 = define_encoder_block(e3, 512)
    e5 = define_encoder_block(e4, 512)
    e6 = define_encoder_block(e5, 512)
    e7 = define_encoder_block(e6, 512)
    # 병목현상방지
    b = Conv2D(512, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(e7)
    b = Activation('relu')(b)
    # decoder model
    d1 = decoder_block(b, e7, 512)
    d2 = decoder_block(d1, e6, 512)
    d3 = decoder_block(d2, e5, 512)
    d4 = decoder_block(d3, e4, 512, dropout=False)
    d5 = decoder_block(d4, e3, 256, dropout=False)
    d6 = decoder_block(d5, e2, 128, dropout=False)
    d7 = decoder_block(d6, e1, 64, dropout=False)
    # output
    g = Conv2DTranspose(3, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(d7)
    out_image = Activation('tanh')(g)
    # define model
    model = Model(in_image, out_image)
    return model
```

```python
# Encoder (downsampling)
def encoder_layer(layer_in, n_filters, batchnorm=True):

    # 가중치 초기화
    init = RandomNormal(stddev=0.02)

    # layer 추가
    g = Conv2D(n_filters, (4,4), strides=(2,2), padding='same',
               kernel_initializer=init)(layer_in)

    # BatchNormalization이 True일 경우에만 추가
    if batchnorm:
        g = BatchNormalization()(g, training=True)

    g = LeakyReLU(alpha=0.2)(g)

    return g
```

```python
# Decoder (upsampling)
def decoder_layer(layer_in, skip_in, n_filters, dropout=True):

    # 가중치 초기화
    init = RandomNormal(stddev=0.02)

    # layer 추가
    g = Conv2DTranspose(n_filters, (4,4), strides=(2,2), padding='same',
                        kernel_initializer=init)(layer_in)
    g = BatchNormalization()(g, training=True)

    # Dropout이 True일 경우에만 추가
    if dropout:
        g = Dropout(0.5)(g, training=True)

    # Encoder layer에서 Activation 거치기 전의 output 복사 & merge
    g = Concatenate()([g, skip_in])

    g = Activation('relu')(g)

    return g
```

생성자

```python
# 생성자 모델
def define_generator(image_shape=(256,256,
    init = RandomNormal(stddev=0.02)
    in_image = Input(shape=image_shape)
    # encoder model
    e1 = define_encoder_block(in_image, 64, batchnorm=False)
    e2 = define_encoder_block(e1, 128)
    e3 = define_encoder_block(e2, 256)
    e4 = define_encoder_block(e3, 512)
    e5 = define_encoder_block(e4, 512)
    e6 = define_encoder_block(e5, 512)
    e7 = define_encoder_block(e6, 512)
    # 병목현상방지
    b = Conv2D(512, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(e7)
    b = Activation('relu')(b)
    # decoder model
    d1 = decoder_block(b, e7, 512)
    d2 = decoder_block(d1, e6, 512)
    d3 = decoder_block(d2, e5, 512)
    d4 = decoder_block(d3, e4, 512, dropout=False)
    d5 = decoder_block(d4, e3, 256, dropout=False)
    d6 = decoder_block(d5, e2, 128, dropout=False)
    d7 = decoder_block(d6, e1, 64, dropout=False)
    # output
    g = Conv2DTranspose(3, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(d7)
    out_image = Activation('tanh')(g)
    # define model
    model = Model(in_image, out_image)
    return model
```

input data를 normalize 해주는 것처럼
은닉층에서도 학습이 잘되게
입력값을 normalize해 입력 분포를 조정

학습 편의성이 개선되어 수렴 속도가 빨라지고,
Local Optima를 피할 가능성이 높아진다
=> 학습과정 안정화

Generator의 Output Layer와 Discriminator의 Input
Layer에는 BN을 적용X => 실제 이미지와는 값의 범
위가 다름 (정확한 값으로 학습)

Dropout은 학습시 노이즈가 되어
고정의 이미지만 생성하는 것을 방
지하는 역할을 함

```python
# Encoder (downsampling)
def encoder_layer(layer_in, n_filters, batchnorm=True):

    # 가중치 초기화
    init = RandomNormal(stddev=0.02)

    # layer 추가
    g = Conv2D(n_filters, (4,4), strides=(2,2), padding='same',
               kernel_initializer=init)(layer_in)

    # BatchNormalization이 True일 경우에만 추가
    if batchnorm:
        g = BatchNormalization()(g, training=True)

    g = LeakyReLU(alpha=0.2)(g)

    return g
```

```python
# Decoder (upsampling)
def decoder_layer(layer_in, skip_in, n_filters, dropout=True):

    # 가중치 초기화
    init = RandomNormal(stddev=0.02)

    # layer 추가
    g = Conv2DTranspose(n_filters, (4,4), strides=(2,2), padding='same',
                        kernel_initializer=init)(layer_in)
    g = BatchNormalization()(g, training=True)

    # Dropout이 True일 경우에만 추가
    if dropout:
        g = Dropout(0.5)(g, training=True)

    # Encoder layer에서 Activation 거치기 전의 output 복사 & merge
    g = Concatenate()([g, skip_in])

    g = Activation('relu')(g)

    return g
```

생성자

```python
# 생성자 모델
def define_generator(image_shape=(256,256,3)):
    init = RandomNormal(stddev=0.02)
    in_image = Input(shape=image_shape)
    # encoder model
    e1 = define_encoder_block(in_image, 64, batchnorm=False)
    e2 = define_encoder_block(e1, 128)
    e3 = define_encoder_block(e2, 256)
    e4 = define_encoder_block(e3, 512)
    e5 = define_encoder_block(e4, 512)
    e6 = define_encoder_block(e5, 512)
    e7 = define_encoder_block(e6, 512)
    # 병목현상방지
    b = Conv2D(512, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(e7)
    b = Activation('relu')(b)
    # decoder model
    d1 = decoder_block(b, e7, 512)
    d2 = decoder_block(d1, e6, 512)
    d3 = decoder_block(d2, e5, 512)
    d4 = decoder_block(d3, e4, 512, dropout=False)
    d5 = decoder_block(d4, e3, 256, dropout=False)
    d6 = decoder_block(d5, e2, 128, dropout=False)
    d7 = decoder_block(d6, e1, 64, dropout=False)
    # output
    g = Conv2DTranspose(3, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(d7)
    out_image = Activation('tanh')(g)
    # define model
    model = Model(in_image, out_image)
    return model
```

ReLU는 generate 부분에서만 권장되고

나머지에서는 ReLU의 변형체인 Leaky ReLU가 권장

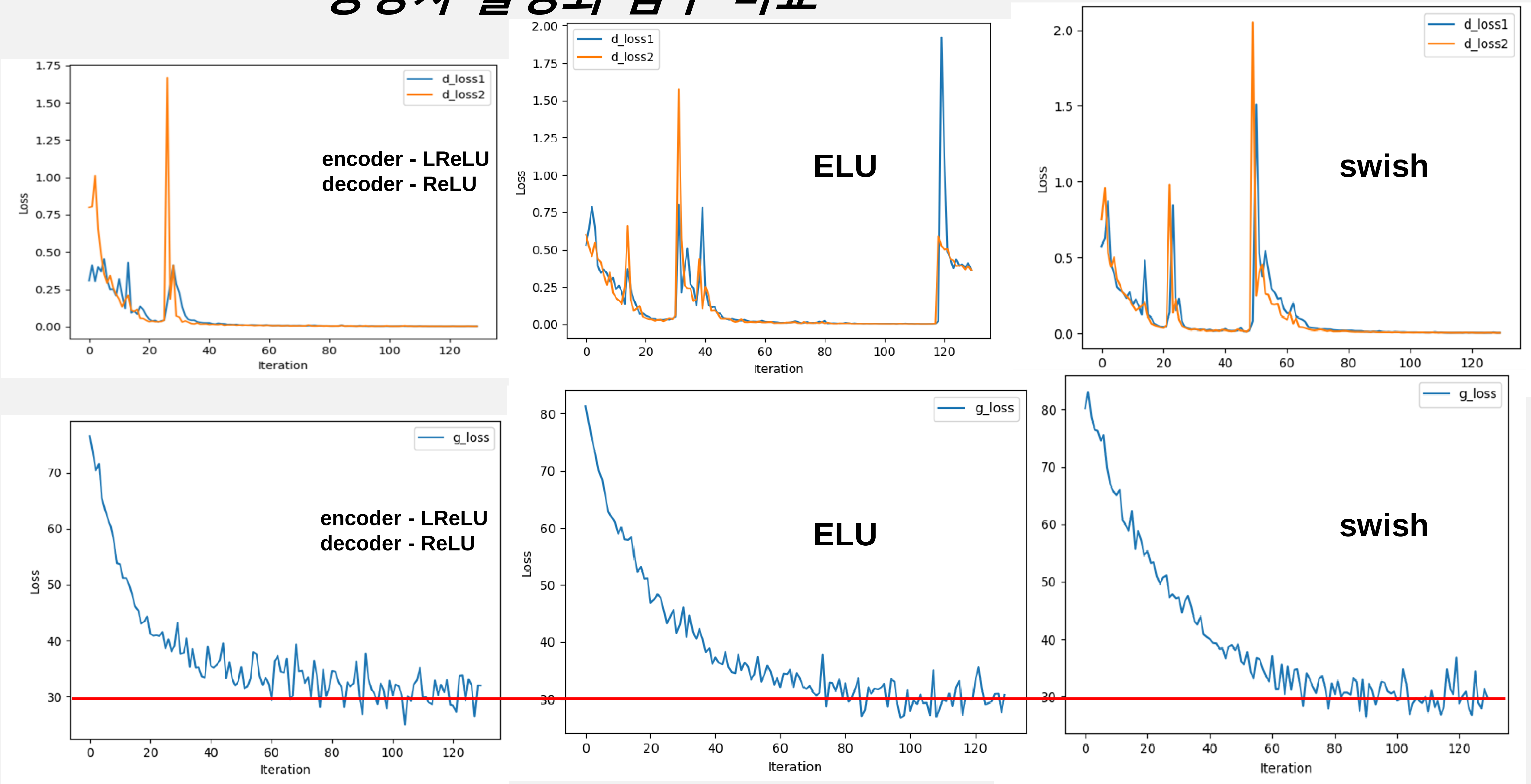
깊은 CNN에서

gradient vanishing 문제를 해결

```python
# Encoder (downsampling)
def encoder_layer(layer_in, n_filters, batchnorm=True):

    # 가중치 초기화
    init = RandomNormal(stddev=0.02)

    # layer 추가
    g = Conv2D(n_filters, (4,4), strides=(2,2), padding='same',
               kernel_initializer=init)(layer_in)

    # BatchNormalization이 True일 경우에만 추가
    if batchnorm:
        g = BatchNormalization()(g, training=True)

    g = LeakyReLU(alpha=0.2)(g)

    return g
```

```python
# Decoder (upsampling)
def decoder_layer(layer_in, skip_in, n_filters, dropout=True):

    # 가중치 초기화
    init = RandomNormal(stddev=0.02)

    # layer 추가
    g = Conv2DTranspose(n_filters, (4,4), strides=(2,2), padding='same',
                        kernel_initializer=init)(layer_in)
    g = BatchNormalization()(g, training=True)

    # Dropout이 True일 경우에만 추가
    if dropout:
        g = Dropout(0.5)(g, training=True)

    # Encoder layer에서 Activation 거치기 전의 output 복사 & merge
    g = Concatenate()([g, skip_in])

    g = Activation('relu')(g)

    return g
```

생성자 활성화 함수 비교


d_loss1
d_loss2
encoder - LReLU
decoder - ReLU
ELU
swish
g_loss
Loss
Iteration

생성자

```python
# 생성자 모델
def define_generator(image_shape=(256,256,3)):
    init = RandomNormal(stddev=0.02)
    in_image = Input(shape=image_shape)
    # encoder model
    e1 = define_encoder_block(in_image, 64, batchnorm=False)
    e2 = define_encoder_block(e1, 128)
    e3 = define_encoder_block(e2, 256)
    e4 = define_encoder_block(e3, 512)
    e5 = define_encoder_block(e4, 512)
    e6 = define_encoder_block(e5, 512)
    e7 = define_encoder_block(e6, 512)
    # 병목현상방지
    b = Conv2D(512, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(e7)
    b = Activation('relu')(b)
    # decoder model

     g = Reshape((256, 256, 32))(d7)

     g = UpSampling2D()(g)

     g = Conv2D(3, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(g)

    # output
    g = Conv2DTranspose(3, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(d7)
    out_image = Activation('tanh')(g)
    # define model
    model = Model(in_image, out_image)
    return model
```

```
           1, 2
Input = (3, 4)

           1, 1, 2, 2
Output = (1, 1, 2, 2)
           3, 3, 4, 4
           3, 3, 4, 4
```

판별자

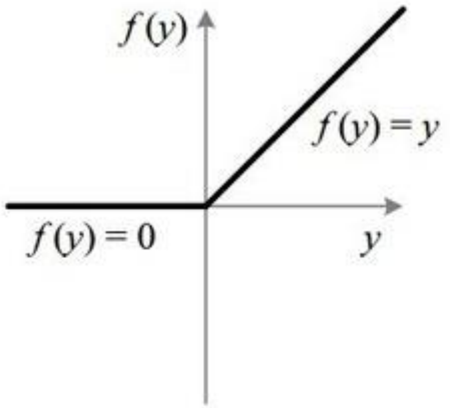
```
# 판별자 모델
def define_discriminator(image_shape):
    init = RandomNormal(stddev=0.02)
    in_src_image = Input(shape=image_shape)
    in_target_image = Input(shape=image_shape)

    merged = Concatenate()([in_src_image, in_target_image])

    d = Conv2D(64, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(merged)
    d = LeakyReLU(alpha=0.2)(d)
    d = Conv2D(128, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(d)
    d = BatchNormalization()(d)
    d = LeakyReLU(alpha=0.2)(d)
    d = Conv2D(256, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(d)
    d = BatchNormalization()(d)
    d = LeakyReLU(alpha=0.2)(d)
    d = Conv2D(512, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(d)
    d = BatchNormalization()(d)
    d = LeakyReLU(alpha=0.2)(d)
    d = Conv2D(512, (4,4), padding='same', kernel_initializer=init)(d)
    d = BatchNormalization()(d)
    d = LeakyReLU(alpha=0.2)(d)
    d = Conv2D(1, (4,4), padding='same', kernel_initializer=init)(d)
    patch_out = Activation('sigmoid')(d)
    model = Model([in_src_image, in_target_image], patch_out)

    # compile model
    opt = Adam(lr=0.0002, beta_1=0.5)
    model.compile(loss='binary_crossentropy', optimizer=opt, loss_weights=[0.5])
    return model
```

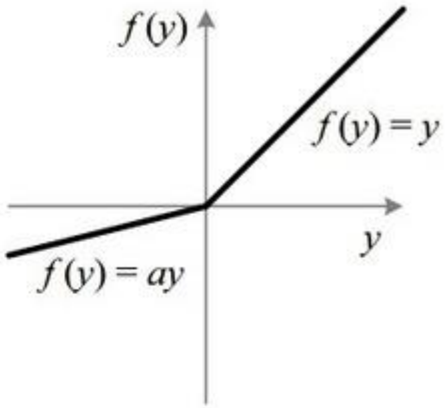
모든 weight는 0을 중심으로 표준편차 0.02의
normal distribution으로 초기화

alpha값 0.2로 통일

LeakyReLU(alpha)

- alpha : 새는(leaky) 정도, 즉 x<0 일때 함수의 기울기
- 일반적으로 alpha = 0.01 ~ 0.1 - 0.2
- a = 0.2 (huge leak) 가 a = 0.01 (small leak) 보다 나은 성능을 낸다고 알려져 있음
  - Empirical Evaluation of Rectified Activations in Convolutional Network. CoRR, 2015
  - ReLU, Leaky ReLU (or parametric ReLU,PReLU), Randomized ReLU (RReLU)을 실험적으로 비교한 논문





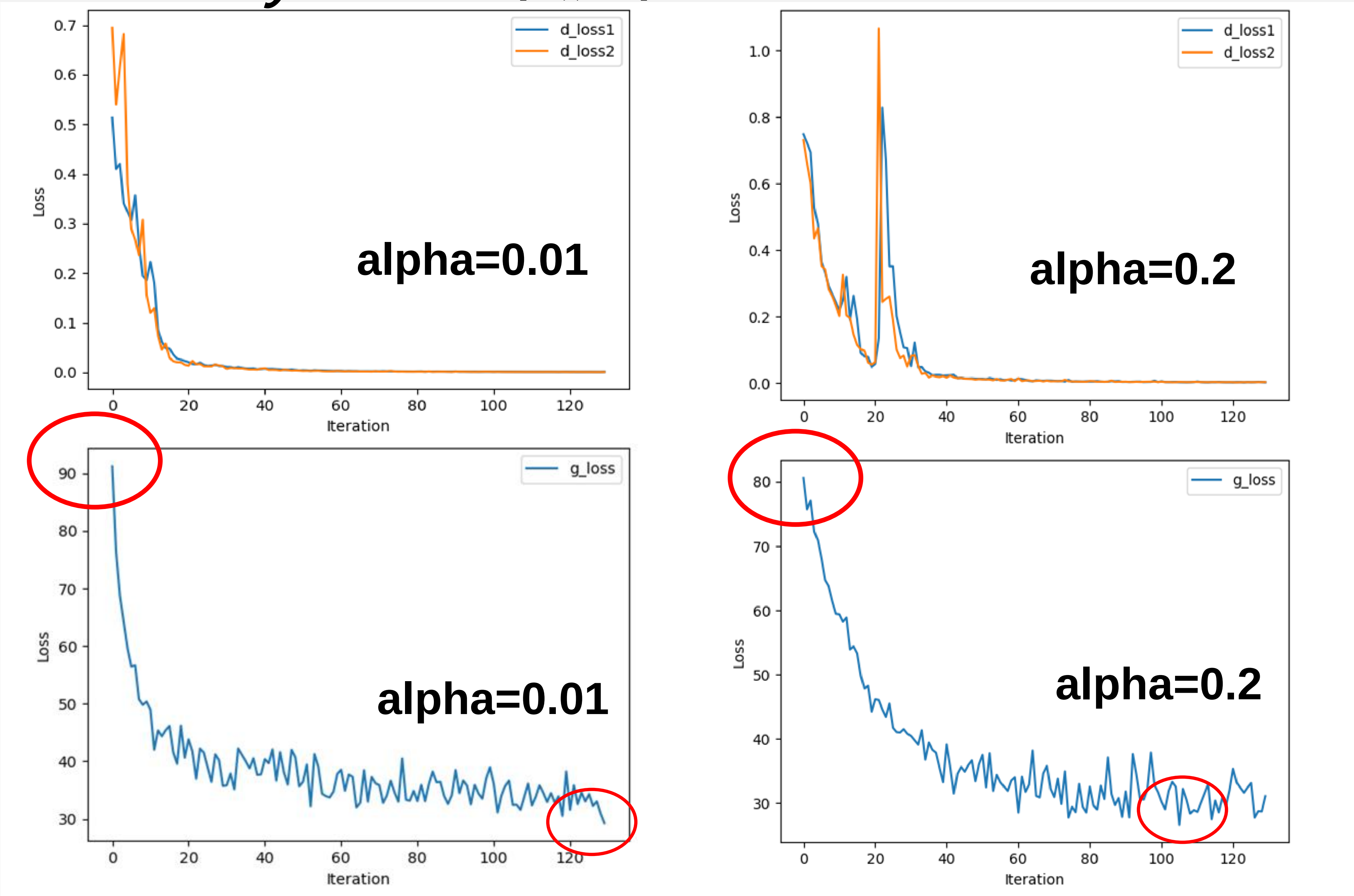
Activation	Training Error	Test Error
ReLU	0.00318	0.1245
Leaky ReLU, $a = 100$	0.0031	0.1266
Leaky ReLU, $a = 5.5$	0.00362	**0.1120**
PReLU	0.00178	0.1179
RReLU ($y_{ji} = x_{ji}/\frac{l+u}{2}$)	0.00550	**0.1119**

Table 3. Error rate of CIFAR-10 Network in Network with different activation function

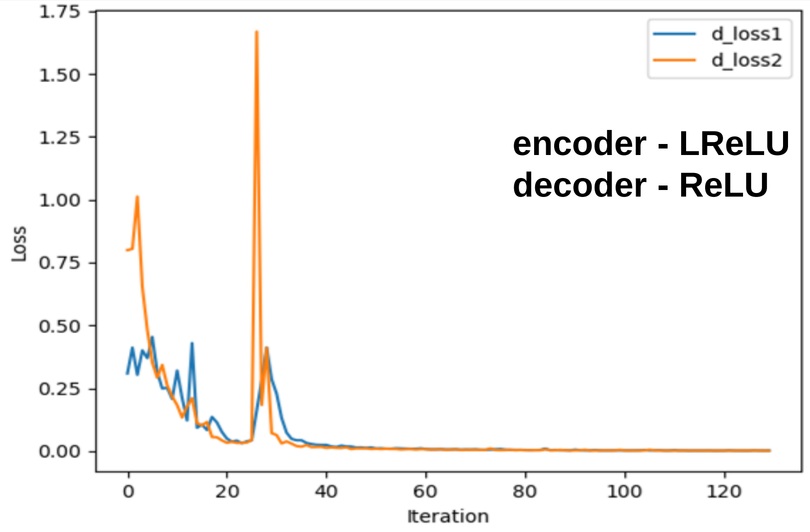
Activation	Training Error	Test Error
ReLU	0.1356	0.429
Leaky ReLU, $a = 100$	0.11552	0.4205
Leaky ReLU, $a = 5.5$	0.08536	**0.4042**
PReLU	0.0633	0.4163
RReLU ($y_{ji} = x_{ji}/\frac{l+u}{2}$)	0.1141	**0.4025**

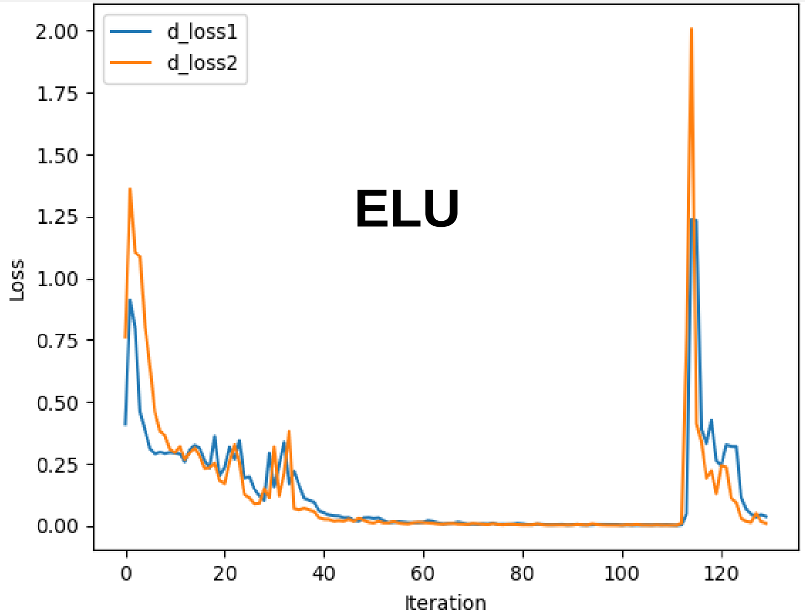
Table 4. Error rate of CIFAR-100 Network in Network with different activation function

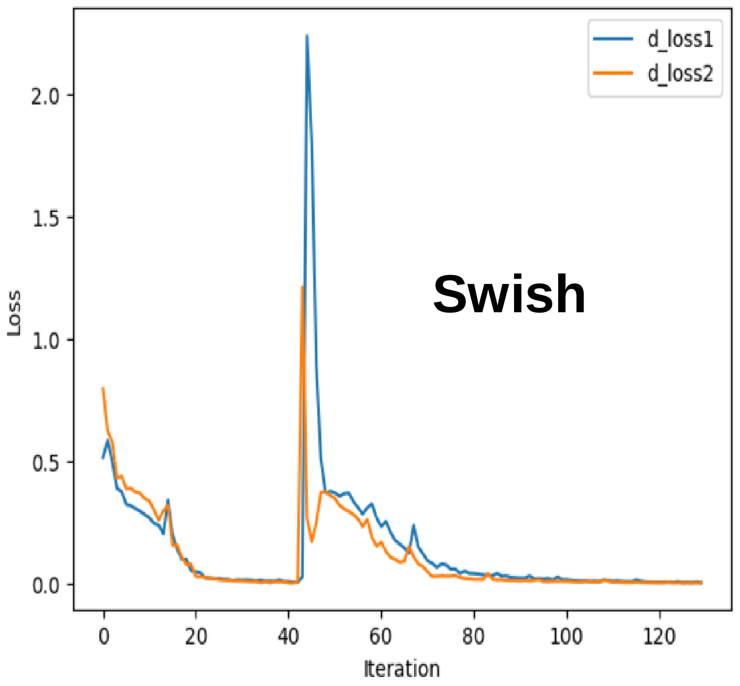
LeakyReLU 알파값 비교

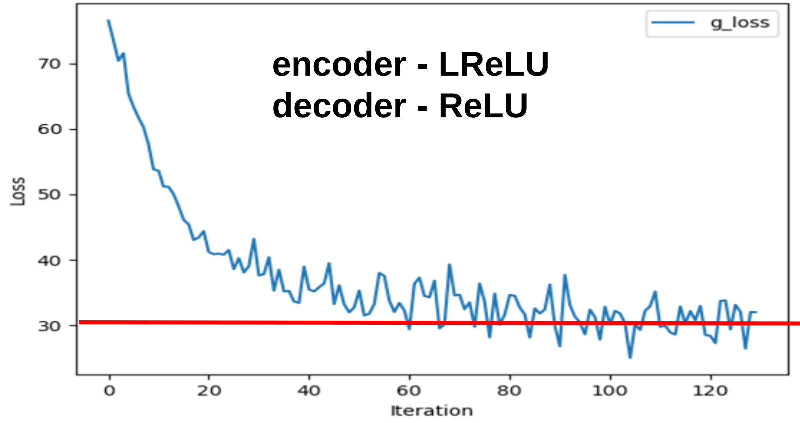


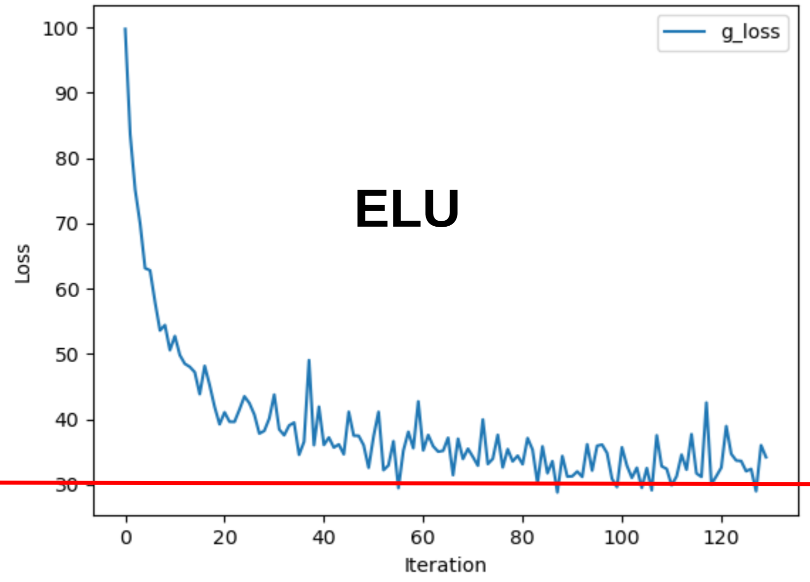
판별자 활성화 함수 비교

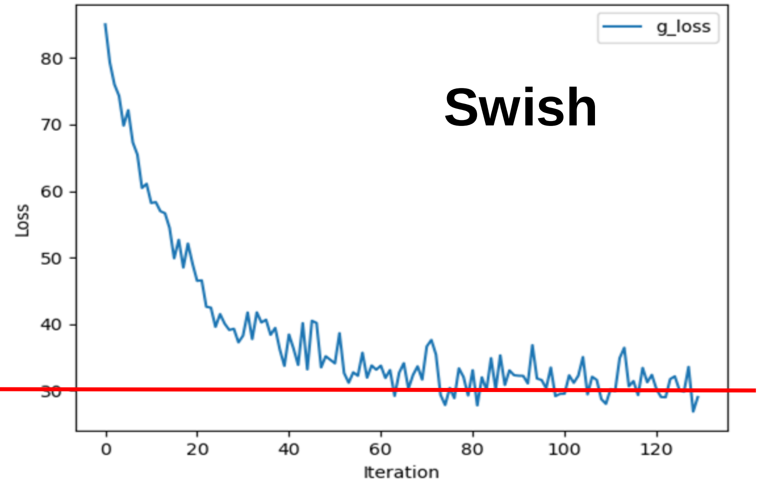

d_loss1
d_loss2
encoder - LReLU
decoder - ReLU
Loss
Iteration


d_loss1
d_loss2
ELU
Loss
Iteration

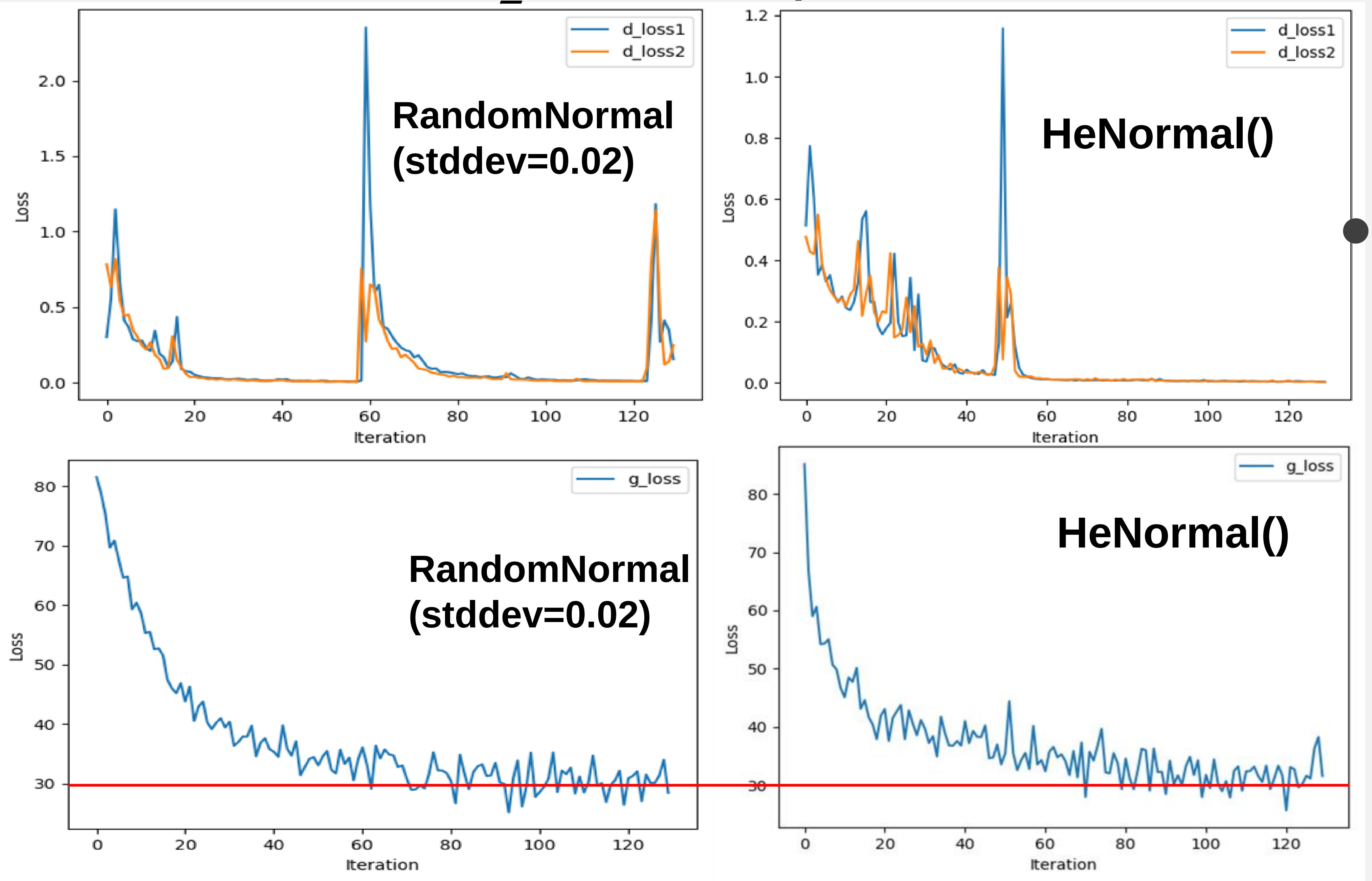

d_loss1
d_loss2
Swish
Loss
Iteration


g_loss
encoder - LReLU
decoder - ReLU
Loss
Iteration


g_loss
ELU
Loss
Iteration


g_loss
Swish
Loss
Iteration

kernel_initializer 비교



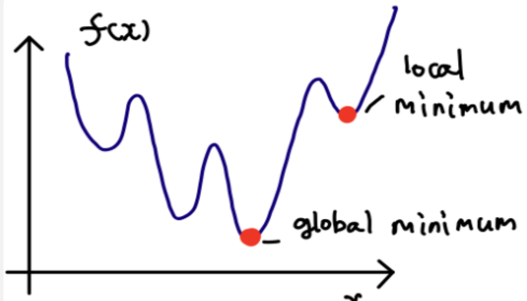
- 논문에서 쓴 표준편차를 0.02로 하는 정규분포 형태로 가중치를 초기와 했을 때, gan loss가 더 낮음

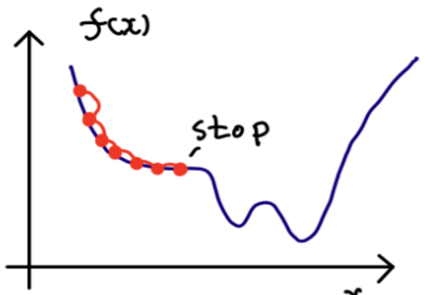
판별자 최적화 함수

```
# compile model
opt = Adam(lr=0.0002, beta_1=0.5)
model.compile(loss='binary_crossentropy', optimizer=opt, loss_weights=[0.5])
return model
```

- generator에 비해 discriminator의 변경을 늦추기 위해 일반적인 효과의 절반(0.5)을 갖도록 가중치 사용

- beta: momentum
  경사 하강법 + 관성
  계산된 기울기에 한 시점 전의 기울기 값을 일정한 비율만큼 반영
- 언덕에서 공이 내려올 때 중간의 작은 웅덩이에 빠지더라도 관성의 힘으로 넘어서는 효과





GAN

```
# GAN 모델
def define_gan(g_model, d_model, image_shape):
    d_model.trainable = False #훈련 동결
    in_src = Input(shape=image_shape) #input:photo
    gen_out = g_model(in_src) #생성자가 생성한 이미지
    dis_out = d_model([in_src, gen_out]) #판별자가 판별한 결과
    model = Model(in_src, [dis_out, gen_out])
    opt = Adam(lr=0.0002, beta_1=0.5)
    # d_loss 1: binary_crossentropy // d_loss2 : mae
    # d_loss 1 : 진짜 사진을 '진짜'라고 판별
    # d_loss 2: 가짜 사진을 '가짜'라고 판별
    model.compile(loss=['binary_crossentropy', 'mae'], optimizer=opt, loss_weights=[1,100])
    return model
```

- gan의 손실을 최소화하는 방향으로 모델 학습
- discriminator가 모델의 학습 방향에 너무 큰 영향을 안 주도록 1로 설정

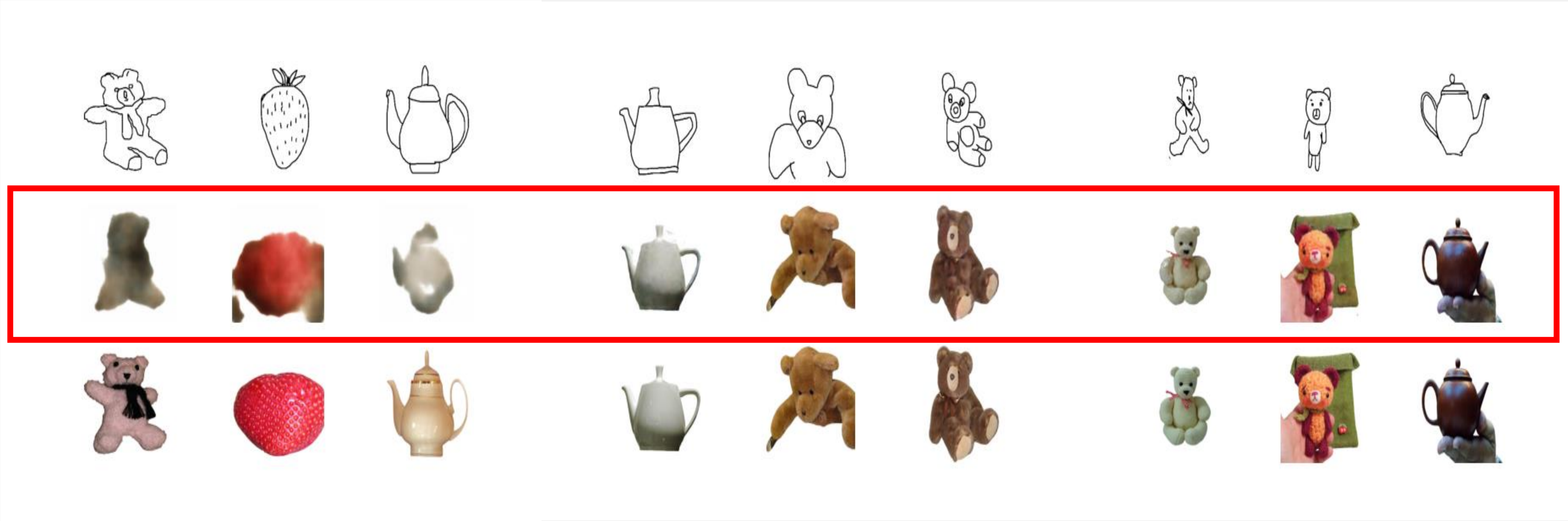
```
g_loss, _, _ = gan_model.train_on_batch(X_realA, [y_real, X_realB])
```

sum(binary_crossentropy * 1, mae * 100), binary_crossentropy, mae

훈련

```python
# 훈련
def train(d_model, g_model, gan_model, dataset, n_epochs=150, n_batch=16):
    # 판별자의 output shape
    n_patch = d_model.output_shape[1]

    trainA, trainB = dataset

    # epoch마다의 배치 숫자 계산
    bat_per_epo = int(len(trainA) / n_batch)

    # 전체 epoch동안의 train data 분할 횟수(batch 횟수)
    n_steps = bat_per_epo * n_epochs

    for i in range(n_steps):
        # batch 한 번 만큼의 sketch, photo
        [X_realA, X_realB], y_real = generate_real_samples(dataset, n_batch, n_patch)

        # batch 한 번 만큼의 생성된 가짜 사진
        X_fakeB, y_fake = generate_fake_samples(g_model, X_realA, n_patch)

        # 진짜 사진을 '진짜'라고 판별
        d_loss1 = d_model.train_on_batch([X_realA, X_realB], y_real)

        # 가짜 사진을 '가짜'라고 판별
        d_loss2 = d_model.train_on_batch([X_realA, X_fakeB], y_fake)

        # GAN
        g_loss, _, _ = gan_model.train_on_batch(X_realA, [y_real, X_realB])
        print('>%d, d1[%.3f] d2[%.3f] g[%.3f]' % (i+1, d_loss1, d_loss2, g_loss))

        # 진행 상황 확인
        if (i+1) % (bat_per_epo * 10) == 0:
            summarize_performance(i, g_model, dataset)
```

훈련 결과



Epoch 10　　　Epoch 100　　　Epoch 200

웹 프로토타입

Sketch to Photo

파일선택

이미지
업로드

변환 중

변환 완료
테디베어일 확률은 100% 입니다.

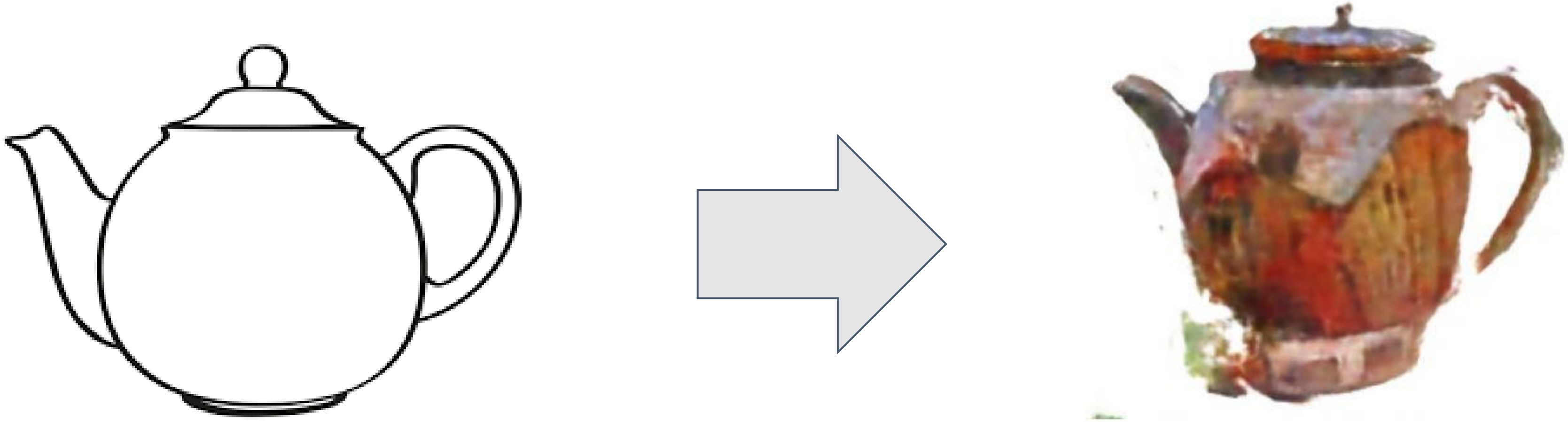
label classification

```
# 모델
vgg16=VGG16(weights="imagenet", include_top=False, input_shape=(256, 256, 3))
vgg16.trainable=False

model = Sequential()
model.add(vgg16)
model.add(Flatten())
model.add(Dense(256))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dense(4, activation="softmax"))


# 컴파일
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['acc'])
```

Flask : visualization

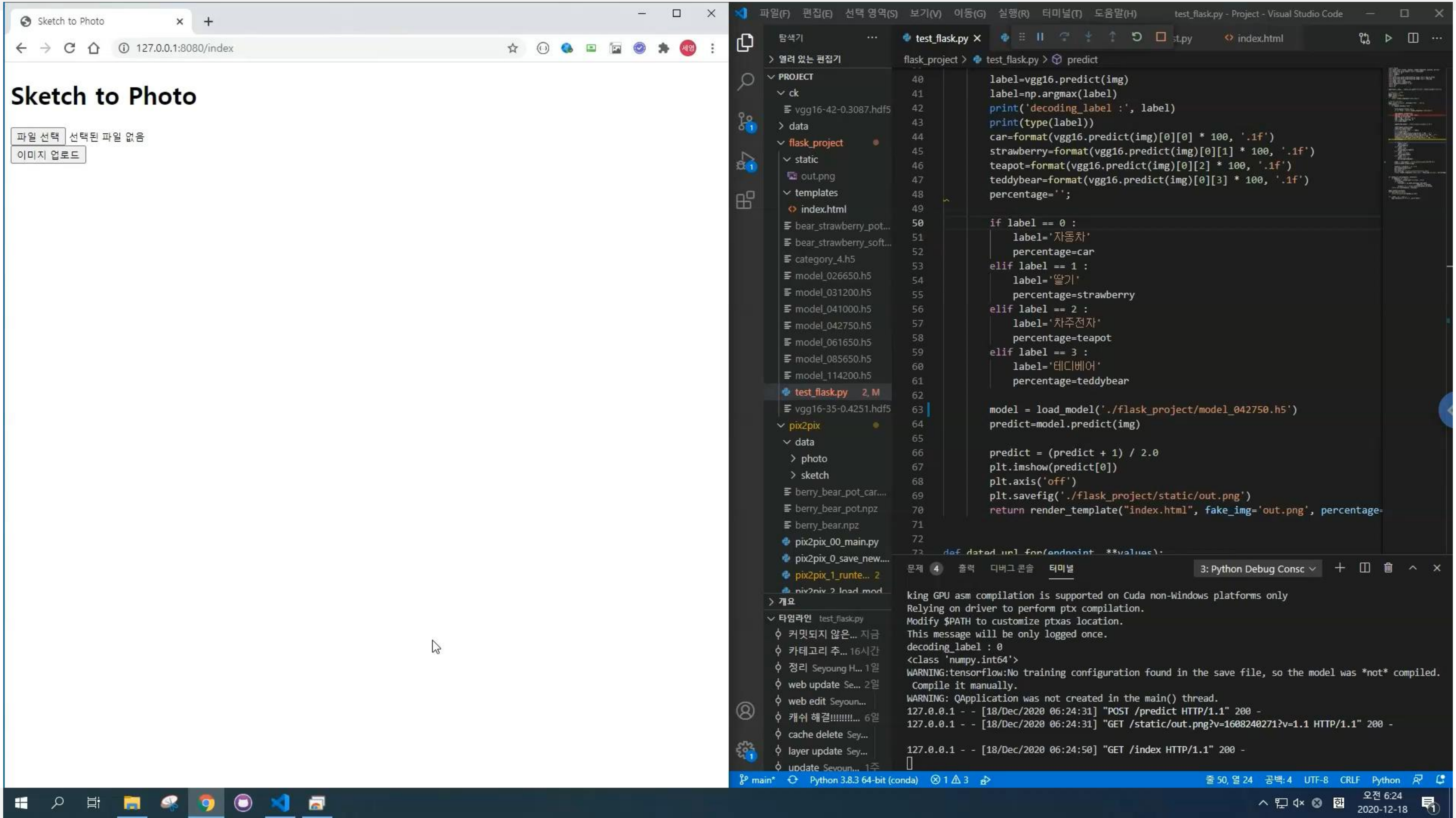
```
vgg16=load_model('./flask_project/category_4.h5')

label=vgg16.predict(img)
label=np.argmax(label)

car=format(vgg16.predict(img)[0][0] * 100, '.1f')
strawberry=format(vgg16.predict(img)[0][1] * 100, '.1f')
teapot=format(vgg16.predict(img)[0][2] * 100, '.1f')
teddybear=format(vgg16.predict(img)[0][3] * 100, '.1f')
percentage='';

if label == 0 :
    label='자동차'
    percentage=car
elif label == 1 :
    label='딸기'
    percentage=strawberry
elif label == 2 :
    label='차주전자'
    percentage=teapot
elif label == 3 :
    label='테디베어'
    percentage=teddybear

model = load_model('./flask_project/model_061650.h5')
predict=model.predict(img)

predict = (predict + 1) / 2.0
plt.imshow(predict[0])
plt.axis('off')
plt.savefig('./flask_project/static/out.png')
return render_template("index.html", fake_img='out.png', percentage=percentage, label=label)
```

출력 결과



차주전자일 확률이 98.0 % 입니다.

시연 영상



Sketch to Photo
127.0.0.1:8080/index
Sketch to Photo
파일 선택 선택된 파일 없음
이미지 업로드
test_flask.py - Project - Visual Studio Code
label=vgg16.predict(img)
label=np.argmax(label)
print('decoding_label :', label)
print(type(label))
car=format(vgg16.predict(img)[0][0] * 100, '.1f')
strawberry=format(vgg16.predict(img)[0][1] * 100, '.1f')
teapot=format(vgg16.predict(img)[0][2] * 100, '.1f')
teddybear=format(vgg16.predict(img)[0][3] * 100, '.1f')
percentage='';
if label == 0 :
label='자동차'
percentage=car
elif label == 1 :
label='딸기'
percentage=strawberry
elif label == 2 :
label='차주전자'
percentage=teapot
elif label == 3 :
label='테디베어'
percentage=teddybear
model = load_model('./flask_project/model_042750.h5')
predict=model.predict(img)
predict = (predict + 1) / 2.0
plt.imshow(predict[0])
plt.axis('off')
plt.savefig('./flask_project/static/out.png')
return render_template("index.html", fake_img='out.png', percentage=
decoding_label : 0
<class 'numpy.int64'>
WARNING:tensorflow:No training configuration found in the save file, so the model was *not* compiled. Compile it manually.
127.0.0.1 - - [18/Dec/2020 06:24:31] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [18/Dec/2020 06:24:31] "GET /static/out.png?v=1608240271?v=1.1 HTTP/1.1" 200 -
127.0.0.1 - - [18/Dec/2020 06:24:50] "GET /index HTTP/1.1" 200 -
오전 6:24
2020-12-18

프로젝트 링크

- GitHub : https://github.com/ym0179/sketch2image


질문 없죠
?

THANK YOU!