딥러닝/클라우드

Chapter 15

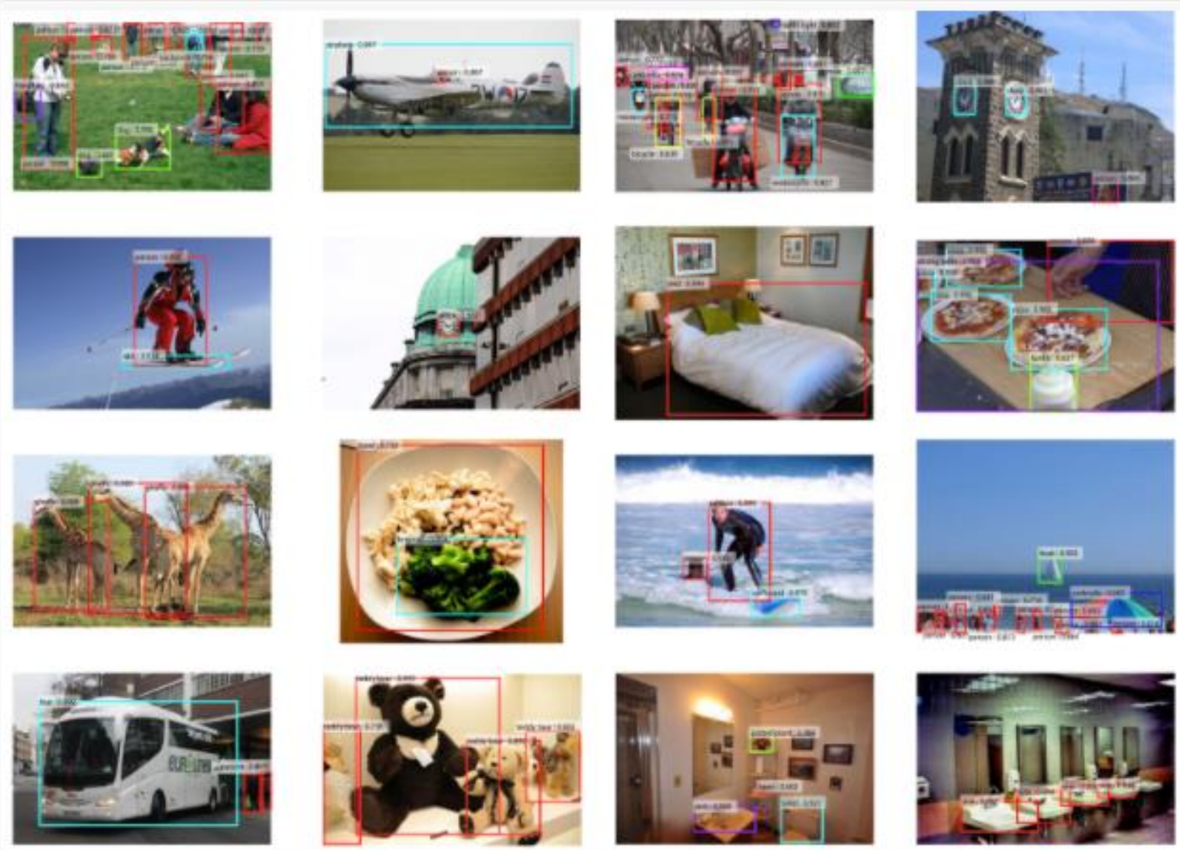
Object Detection, GAN

오 세 종

DANKOOK UNIVERSITY

Contents

1. Object Detection

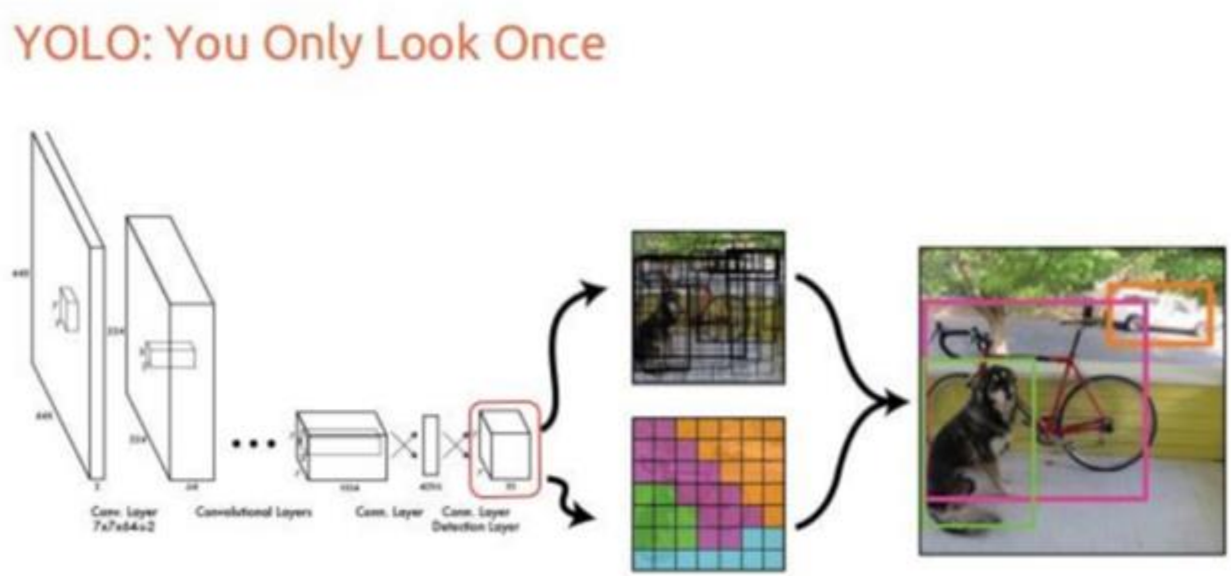


Example of Object Detection With Faster R-CNN on the MS COCO Dataset

- 학습시 이미지데이터 + object 좌표(xywh) with 레이블 필요
- https://blog.roboflow.com/how-to-train-yolov8-on-a-custom-dataset/
- 예측결과: object 좌표(xywh) with 레이블

1. Object Detection

- YOLO model
  - 조셉 레드몬에 의해 2015년 소개
  - real time object detection 추구
  - 9000 개 이상의 object 탐지



https://www.datascienceprophet.com/object-detection-with-yolo-exploring-the-latest-deep-learning-techniques/

1. Object Detection

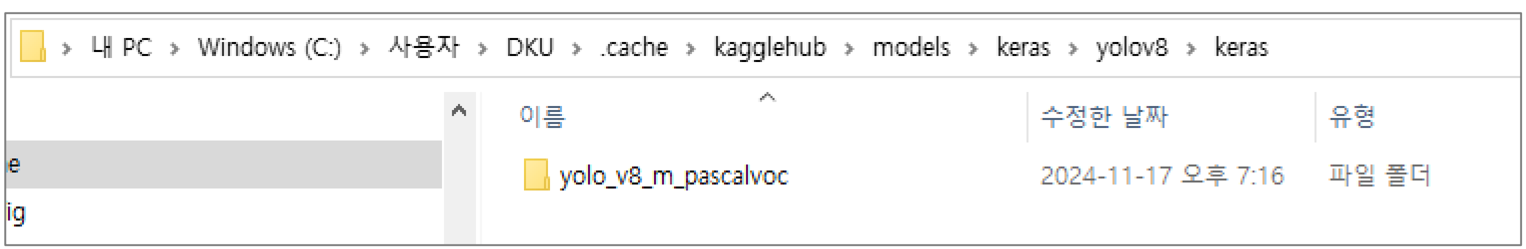
- Example: Keras-cv + YOLO model
  - requirement

```
pip install keras-cv
pip install opencv-python
```

1. Object Detection

15_Object dection example.py

```
import os

os.environ["KERAS_BACKEND"] = "tensorflow"

import keras
import keras_cv
import numpy as np

from keras_cv import visualization
import matplotlib.pyplot as plt

# Load a pretrained model
pretrained_model = keras_cv.models.YOLOV8Detector.from_preset(
    "yolo_v8_m_pascalvoc", bounding_box_format="xywh"
)
```



1. Object Detection

```
# Load an image from a URL or local path
filepath = "C:/Users/DKU/Downloads/gCNcJJI.jpeg"
image = keras.utils.load_img(filepath)
image = np.array(image)

visualization.plot_image_gallery(
    np.array([image]),
    value_range=(0, 255),
    rows=1,
    cols=1,
    scale=5,
)
plt.show()
```

1. Object Detection

```
# resize image
inference_resizing = keras_cv.layers.Resizing(
    640, 640, pad_to_aspect_ratio=True,
    bounding_box_format="xywh"
)

image_batch = inference_resizing([image])

class_ids = [
    "Aeroplane",  "Bicycle", "Bird",  "Boat", "Bottle",
    "Bus",  "Car", "Cat", "Chair", "Cow", "Dining Table",
    "Dog", "Horse", "Motorbike", "Person", "Potted Plant",
    "Sheep", "Sofa", "Train", "Tvmonitor", "Total", ]

class_mapping = dict(zip(range(len(class_ids)), class_ids))
```

```
>>> class_mapping
{0: 'Aeroplane', 1: 'Bicycle', 2: 'Bird', 3: 'Boat', 4: 'Bottle', 5: 'Bus', 6: 'Car', 7: 'Cat', 8: 'Chair', 9: 'Cow',
 10: 'Dining Table', 11: 'Dog', 12: 'Horse', 13: 'Motorbike', 14: 'Person', 15: 'Potted Plant', 16: 'Sheep', 17: 'Sof
a', 18: 'Train', 19: 'Tvmonitor', 20: 'Total'}
```

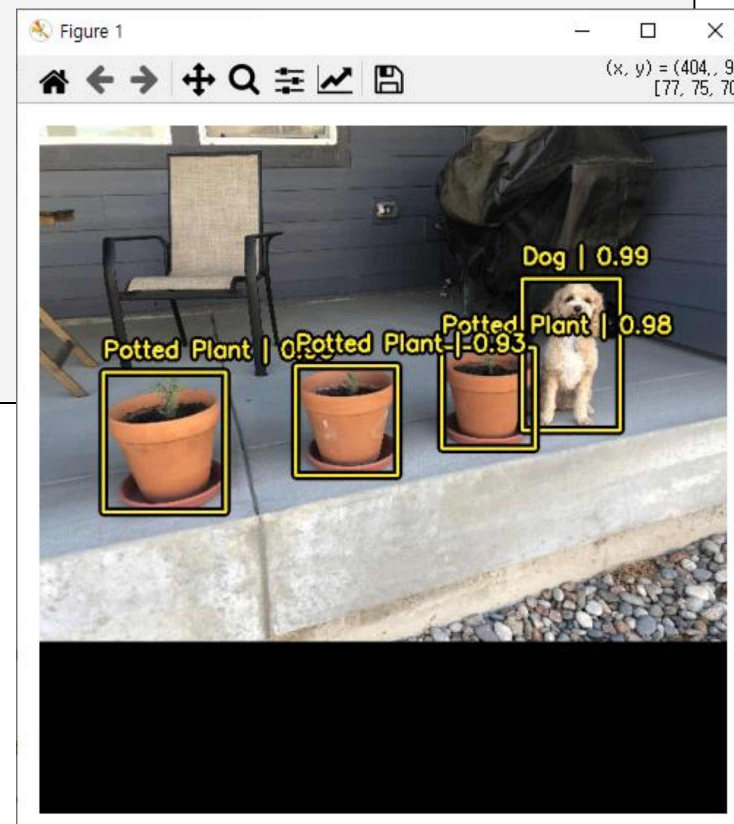
1. Object Detection

```python
y_pred = pretrained_model.predict(image_batch)
# y_pred is a bounding box Tensor:
# {"boxes": ..., "confidence": ..., "classes": ...,}
print(y_pred['classes'][0][:4])
print(y_pred['confidence'][0][:4])
print(y_pred['boxes'][0][:4])
```

```
>>> y_pred
{'boxes': array([[[449.5106  , 142.71707 ,  90.91397 , 141.12839 ],
        [374.74982 , 206.06046 ,  86.41052 ,  94.26883 ],
        [ 59.679283, 229.16298 , 114.07429 , 130.60637 ],
        [238.03307 , 223.13684 ,  95.846085, 102.32648 ],
        [ -1.      ,  -1.      ,  -1.      ,  -1.      ],
        [ -1.      ,  -1.      ,  -1.      ,  -1.      ]

        [ -1.      ,  -1.      ,  -1.      ,  -1.      ],
        [ -1.      ,  -1.      ,  -1.      ,  -1.      ]]], dtype=float32), 'confidence': array([[ 0.99133146,  0.975
2819 ,  0.9600598 ,  0.9280971 , -1.        ,
        -1.        , -1.        , -1.        , -1.        , -1.        ,

 dtype=float32), 'classes': array([[11, 15, 15, 15, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
  -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
```

1. Object Detection

```
visualization.plot_bounding_box_gallery(
    image_batch,
    value_range=(0, 255),
    rows=1,
    cols=1,
    y_pred=y_pred,
    scale=5,
    font_scale=0.7,
    bounding_box_format="xywh",
    class_mapping=class_mapping,
)
plt.show()
```

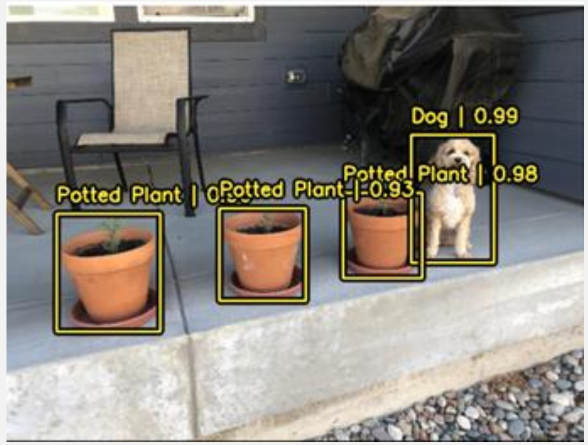


[Exercise]

- 이미지를 load하면 object detection 결과를 보여주는 앱을 개발한다

Object Detection Test

Load image



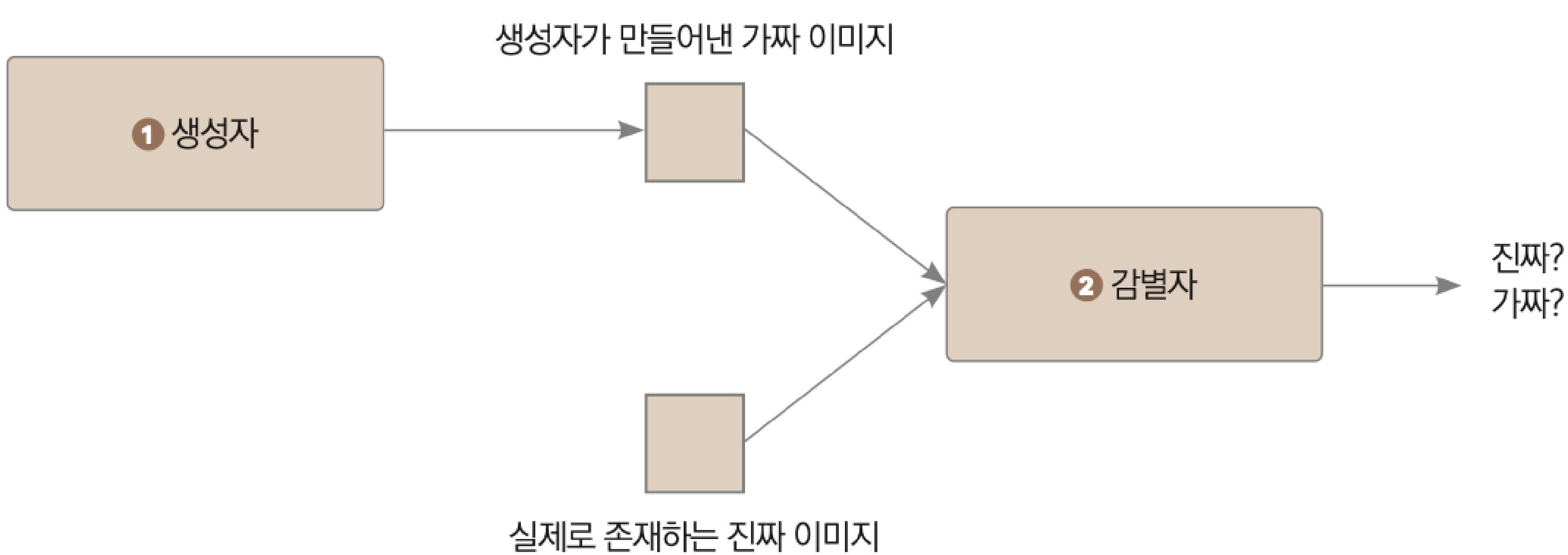
Object list

----

1. Dog (0.99)
2. Potted Plant (0.96)
3. Potted Plant (0.95)
4. Potted Plant (0.93)

2. GAN

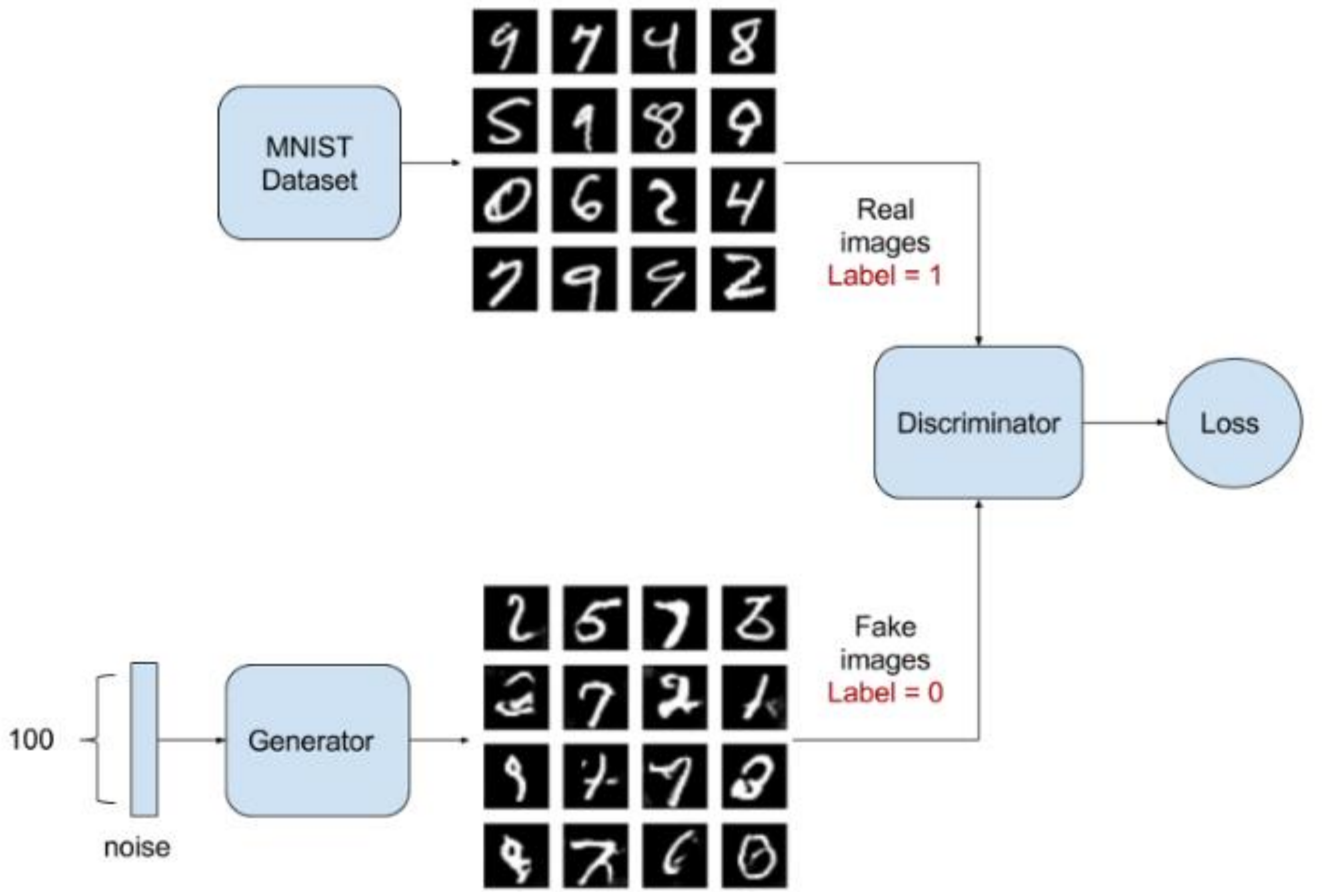
- GAN (Generative Adversarial Networks)
  - 진짜와 너무나도 똑같아서 구별이 가지 않는 가짜 그림이 있다고 하면, 그 그림은 진짜일까, 가짜일까
  - GAN은 이런 생각에서 출발한 모델
  - ❶ 가짜 이미지를 만들어내도록 학습되는 생성자와 ❷ 가짜와 진짜 이미지를 구별하는 감별자를 경쟁시켜 학습하면, 생성자가 점점 진짜와 같은 이미지를 만들게 되는 원리



https://goldenrabbit.co.kr/2023/07/31/%ED%8C%8C%EC%9D%B4%ED%86%A0%EC%B9%98-%EB%94%A5%EB%9F%AC%EB%8B%9D-gan%EC%9C%BC%EB%A1%9C-%EC%82%AC%EB%9E%8C-%EC%96%BC%EA%B5%B4-%EB%A7%8C%EB%93%A4%EA%B8%B0/

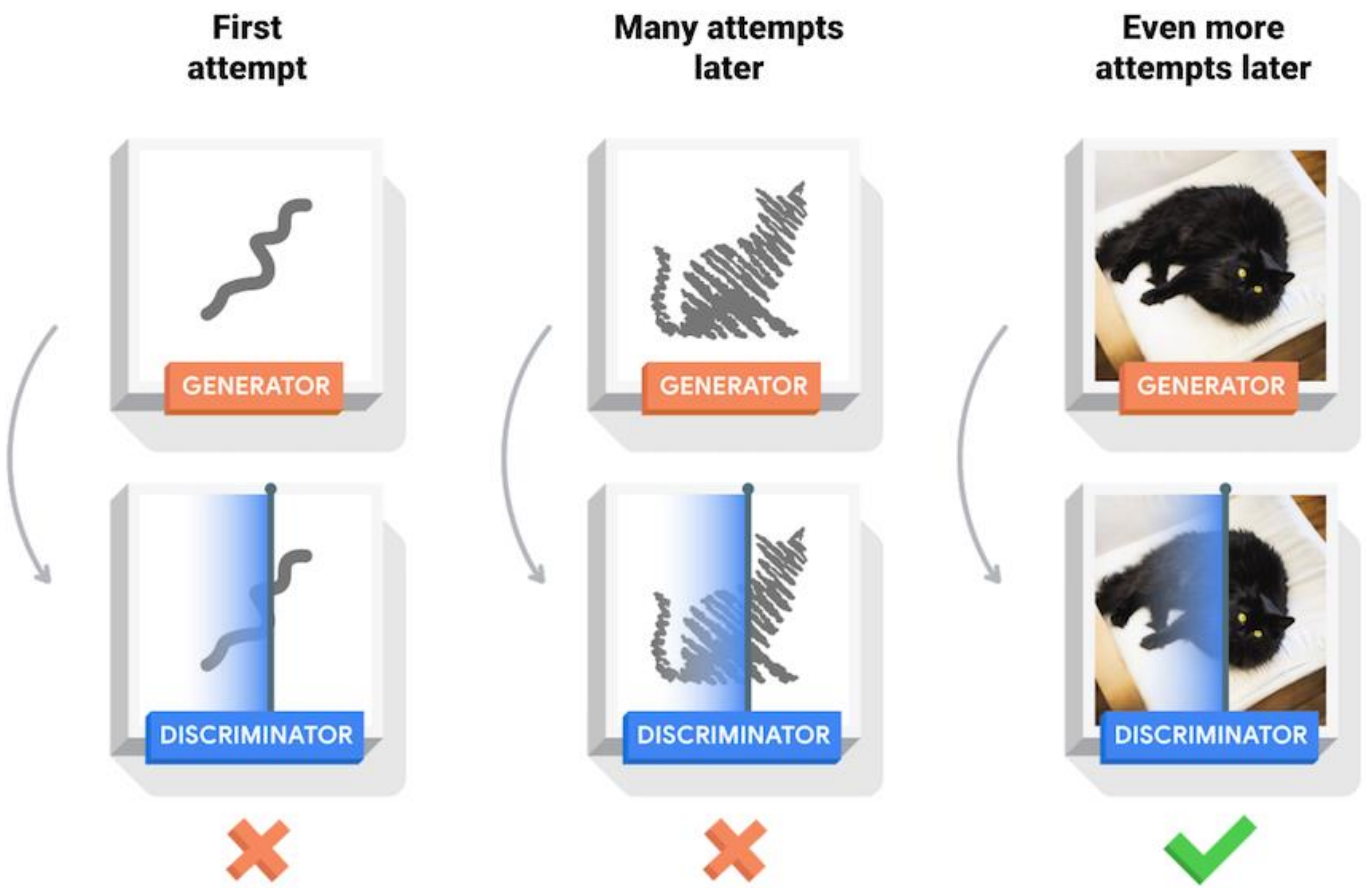
2. GAN

- Gan concept



https://towardsdatascience.com/gan-by-example-using-keras-on-tensorflow-backend-1a6d515a60d0

2. GAN



https://www.tensorflow.org/tutorials/generative/dcgan?hl=ko

GAN의 장단점

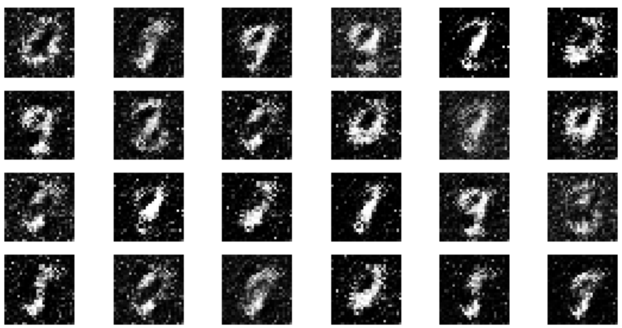
▼ GAN 장단점

장점	단점
• 일반적인 학습 방법에 비해 자연스러운 이미지를 생성할 수 있습니다. • 정답과 단순히 비교하는 것이 아니라, 감별자를 이용해 학습하므로 좋은 성능을 낼 수 있습니다.	• 학습이 올바로 이루어지기 어렵고 필요한 데이터 수가 많아야 합니다. • 사용하는 특징의 개수에 매우 민감하기 때문에 좋은 성능을 내는 특징의 개수를 정하기 어렵습니다.

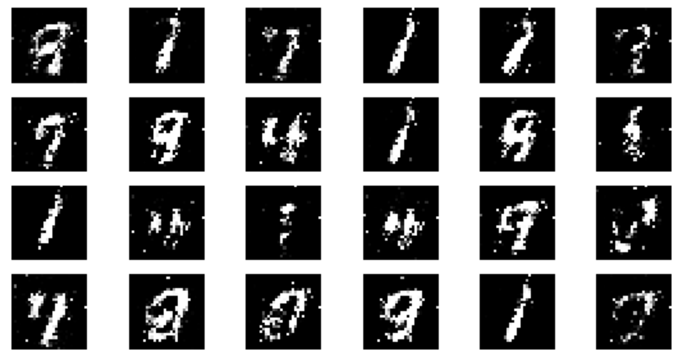
▼ GAN 장단점

- 이미지 생성, 화질 올리기 등
- 인코더 디코더에서 디코더를 학습할 때 사용하기 좋습니다.

Real images

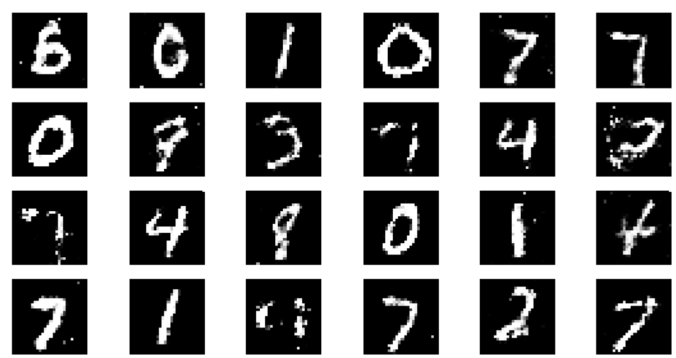


epoch 1



epoch 50

Fake images

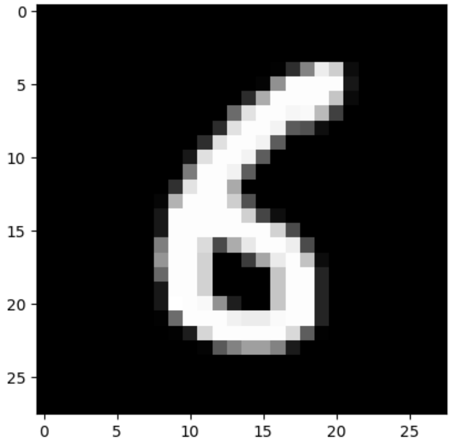


epoch 50

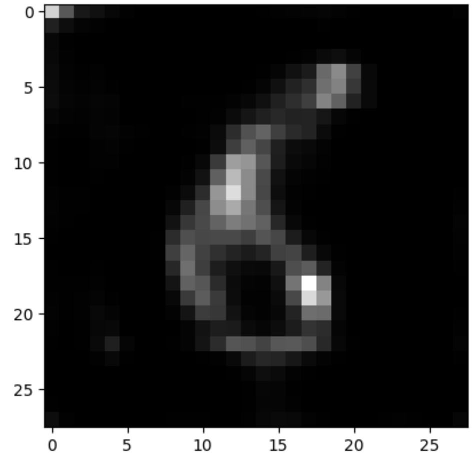
https://dacon.io/codeshare/4231

2. GAN

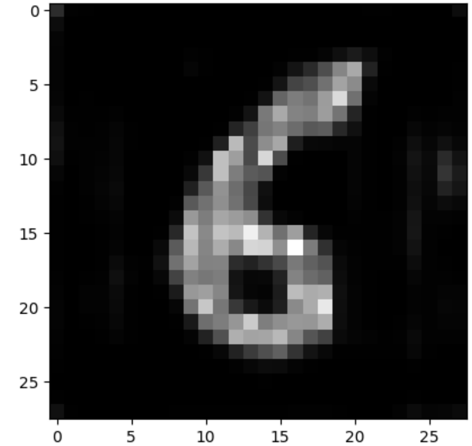
- 존재하지 않는 숫자의 이미지를 생성해 보자



실제 이미지



GAN으로 생성한 가짜 이미지 (epoch=300)



GAN으로 생성한 가짜 이미지 (epoch=500)

2. GAN

- Note
  - Keras 3.6 버전에서는 GAN code 실행이 안되니 3.4.1 버전으로 downgrade 할 것

```
pip install keras==3.4.1
pip install tensorflow==2.16.2
```

2. GAN

- Import modules

15_keras_GAN_example.py

```
import numpy as np
import keras.backend as K
from keras.models import Sequential
from keras.layers import Conv2D, Activation, Dropout, Flatten,
     Dense, BatchNormalization, Reshape, UpSampling2D, LeakyReLU
from keras.optimizers import RMSprop
from keras.preprocessing import image
from keras.preprocessing.image import array_to_img
from skimage import color

import warnings ; warnings.filterwarnings('ignore')

import matplotlib.pyplot as plt
from tqdm import tqdm                                   # progress bar
```

2. GAN

- Prepare data

```
K.clear_session()

# load data
img = image.load_img('d:/data/6_img.png', target_size=(28,28))
img = color.rgb2gray(img)
img_array_train = image.img_to_array(img)
img_array_train = np.expand_dims(img_array_train, axis=0)

Xtrain = img_array_train
Xtrain = Xtrain / 255

img_shape = (img_array_train.shape[1], img_array_train.shape[2],
             img_array_train.shape[3])    # row, col, channel
```

```
>>> img_shape
(28, 28, 1)
```

2. GAN

- 판별자(discriminator) 만들기

```
discriminator = Sequential()
discriminator.add(Conv2D(64, kernel_size=(5, 5),
                         input_shape=img_shape,
                         strides=2, padding='same',
                         activation=LeakyReLU(alpha=0.2)))
discriminator.add(Dropout(rate=0.4))
discriminator.add(Conv2D(64, kernel_size=(5, 5),
                         strides=2, padding='same',
                         activation=LeakyReLU(alpha=0.2)))
discriminator.add(Dropout(rate=0.4))
discriminator.add(Conv2D(128, kernel_size=(5, 5),
                         strides=2, padding='same',
                         activation=LeakyReLU(alpha=0.2)))
discriminator.add(Dropout(rate=0.4))
discriminator.add(Flatten())
discriminator.add(Dense(units=1, activation='sigmoid' ))

discriminator.summary()
```

2. GAN

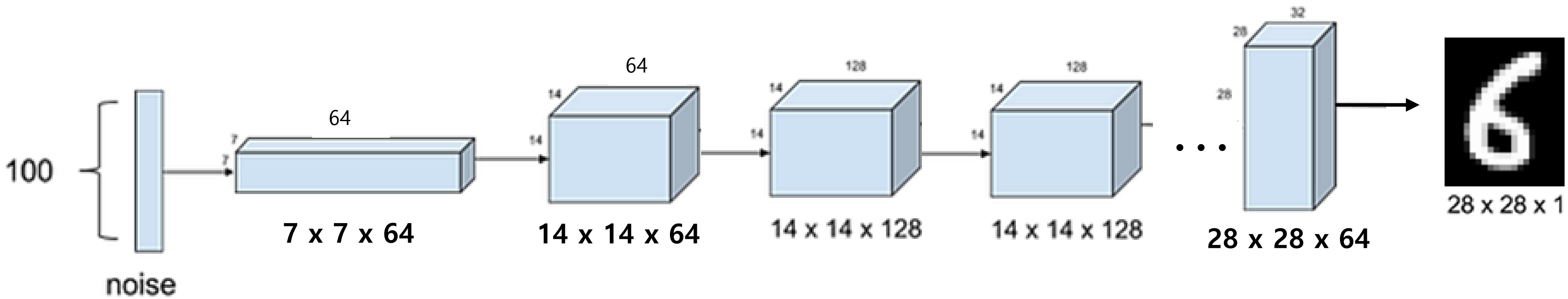
```
>>> discriminator.summary()
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 14, 14, 64)	1,664
dropout (Dropout) conv2d_1 (Conv2D)	(None, 14, 14, 64) (None, 7, 7, 64)	0 102,464
dropout_1 (Dropout)	(None, 7, 7, 64)	0
conv2d_2 (Conv2D)	(None, 4, 4, 128)	204,928
dropout_2 (Dropout)	(None, 4, 4, 128)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 1)	2,049

```
Total params: 311,105 (1.19 MB)
Trainable params: 311,105 (1.19 MB)
Non-trainable params: 0 (0.00 B)
```

2. GAN

- 생성자 구조



[BatchNormalization]

- 딥러닝 모델에서 학습을 안정화하는 데 사용되는 Keras 레이어
- 그래디언트 소실 문제를 완화, 모델의 가중치 초기화에 덜 민감, 상대적으로 큰 학습률 사용으로 학습 속도가 빨라짐, 정규화를 통해 과적합(overfitting) 문제를 방지

2. GAN

- 생성자 만들기

```
gen_dense_size=(7, 7, 64)

generator = Sequential()
generator.add(Dense(units=np.prod(gen_dense_size),
                    input_shape=(100,)))
generator.add(BatchNormalization())
generator.add(Activation('relu'))
generator.add(Reshape(gen_dense_size))

generator.add(UpSampling2D())        # 이미지 사이즈를 2배로
generator.add(Conv2D(filters = 128, kernel_size=5,
                     padding='same', strides=1))
generator.add(BatchNormalization(momentum=0.9))
generator.add(Activation('relu'))
```

2. GAN

```
generator.add(UpSampling2D())
generator.add(Conv2D(filters = 64, kernel_size=5,
                     padding='same', strides=1))
generator.add(BatchNormalization(momentum=0.9))
generator.add(Activation('relu'))

generator.add(Conv2D(filters = 64, kernel_size=5,
                     padding='same', strides=1))
generator.add(BatchNormalization(momentum=0.9))
generator.add(Activation('relu'))

generator.add(Conv2D(filters = 1, kernel_size=5,
                     padding='same', strides=1))
generator.add(Activation('sigmoid'))

generator.summary()
```

```
>>> generator.summary()
Model: "sequential_1"
```

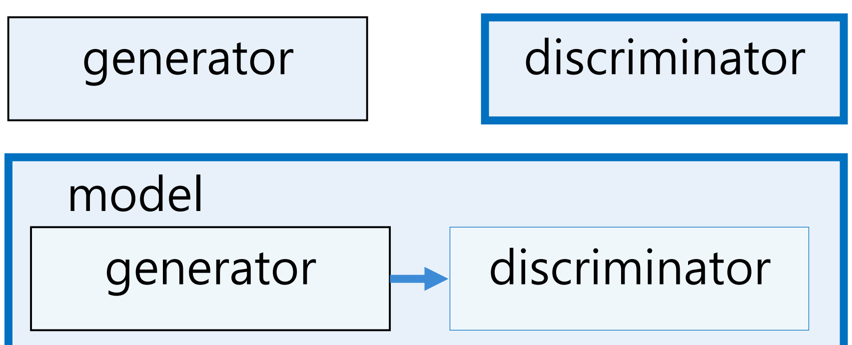
Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 3136)	316,736
batch_normalization (BatchNormalization)	(None, 3136)	12,544
activation (Activation)	(None, 3136)	0
reshape (Reshape)	(None, 7, 7, 64)	0
up_sampling2d (UpSampling2D)	(None, 14, 14, 64)	0
conv2d_3 (Conv2D)	(None, 14, 14, 128)	204,928
batch_normalization_1 (BatchNormalization)	(None, 14, 14, 128)	512
activation_1 (Activation)	(None, 14, 14, 128)	0
up_sampling2d_1 (UpSampling2D)	(None, 28, 28, 128)	0
conv2d_4 (Conv2D)	(None, 28, 28, 64)	204,864
batch_normalization_2 (BatchNormalization)	(None, 28, 28, 64)	256
activation_2 (Activation)	(None, 28, 28, 64)	0
conv2d_5 (Conv2D)	(None, 28, 28, 64)	102,464
batch_normalization_3 (BatchNormalization)	(None, 28, 28, 64)	256
activation_3 (Activation)	(None, 28, 28, 64)	0
conv2d_6 (Conv2D)	(None, 28, 28, 1)	1,601
activation_4 (Activation)	(None, 28, 28, 1)	0

```
Total params: 844,161 (3.22 MB)
Trainable params: 837,377 (3.19 MB)
Non-trainable params: 6,784 (26.50 KB)
```

2. GAN

- Model compile

```
discriminator.compile(optimizer=RMSprop(learning_rate=0.0008),
                      loss='binary_crossentropy',
                      metrics=['accuracy’])

model = Sequential()
model.add(generator)
model.add(discriminator)

model.compile(optimizer=RMSprop(learning_rate=0.0004),
              loss='binary_crossentropy',
              metrics=['accuracy'])
```



2. GAN

- Fitting

```
def train_discriminator(x_train, batch_size):
    valid = np.ones((batch_size, 1))
    fake = np.zeros((batch_size, 1))

    idx = np.random.randint(0, len(x_train), batch_size)
    true_imgs = x_train[idx]
    discriminator.fit(true_imgs, valid, verbose=0)

    noise = np.random.normal(0, 1, (batch_size, 100))
    gen_imgs = generator.predict(noise)

    discriminator.fit(gen_imgs, fake, verbose=0)

def train_generator(batch_size):
    valid = np.ones((batch_size, 1))
    noise = np.random.normal(0, 1, (batch_size, 100))
    model.fit(noise, valid, verbose=1)
```

☆ N개의 input image중 batch_size 개수만큼 선택

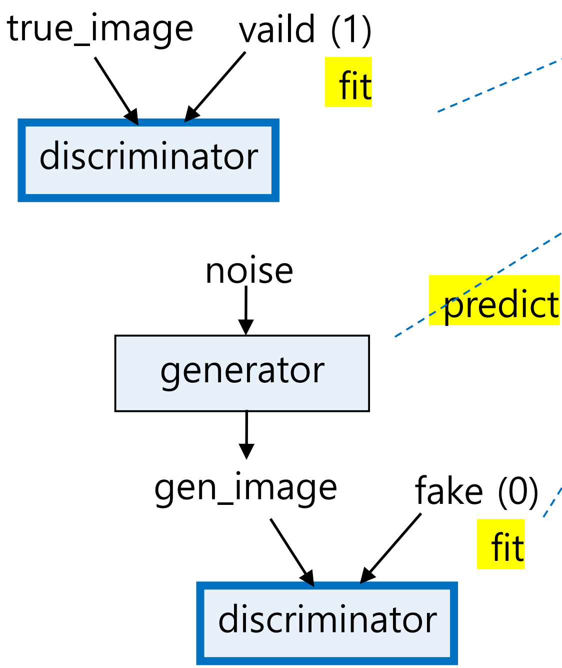
2. GAN

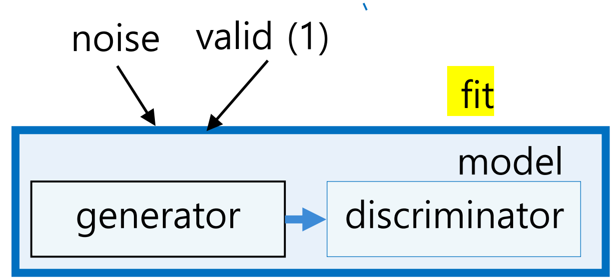
```python
for epoch in tqdm(range(500)):            # Try 2000
    train_discriminator(Xtrain, 64)
    train_generator(64)
```

```
2/2 [==============================] - 0s 195ms/step - loss: 1.5415 - accuracy: 0.3281
2/2 [==============================] - 0s 57ms/step██████████████████████████████  | 293/300 [04:09<00:05,  1.19it/s]
2/2 [==============================] - 0s 197ms/step - loss: 1.5356 - accuracy: 0.0625
2/2 [==============================] - 0s 56ms/step██████████████████████████████  | 294/300 [04:10<00:04,  1.21it/s]
2/2 [==============================] - 0s 272ms/step - loss: 2.5275 - accuracy: 0.1406
2/2 [==============================] - 0s 59ms/step██████████████████████████████  | 295/300 [04:11<00:04,  1.19it/s]
2/2 [==============================] - 1s 293ms/step - loss: 1.9715 - accuracy: 0.0781
2/2 [==============================] - 0s 67ms/step██████████████████████████████  | 296/300 [04:12<00:03,  1.10it/s]
2/2 [==============================] - 0s 190ms/step - loss: 1.7042 - accuracy: 0.3750
2/2 [==============================] - 0s 64ms/step██████████████████████████████▌ | 297/300 [04:13<00:02,  1.10it/s]
2/2 [==============================] - 0s 224ms/step - loss: 1.8096 - accuracy: 0.1719
2/2 [==============================] - 0s 56ms/step██████████████████████████████▊ | 298/300 [04:14<00:01,  1.13it/s]
2/2 [==============================] - 0s 196ms/step - loss: 1.5157 - accuracy: 0.3125
2/2 [==============================] - 0s 57ms/step███████████████████████████████| 299/300 [04:14<00:00,  1.16it/s]
2/2 [==============================] - 0s 184ms/step - loss: 1.7288 - accuracy: 0.1406
100%|████████████████████████████████████████████████████████| 300/300 [04:15<00:00,  1.17it/s]
```

```python
for epoch in tqdm(range(300)):
    train_discriminator(Xtrain, 64)
    train_generator(64)
```

```python
def train_discriminator(x_train, batch_size):
    valid = np.ones((batch_size, 1))
    fake = np.zeros((batch_size, 1))

    idx = np.random.randint(0, len(Xtrain), batch_size)
    true_imgs = Xtrain[idx]
    discriminator.fit(true_imgs, valid, verbose=0)

    noise = np.random.normal(0, 1, (batch_size, 100))
    gen_imgs = generator.predict(noise)

    discriminator.fit(gen_imgs, fake, verbose=0)

def train_generator(batch_size):
    valid = np.ones((batch_size, 1))
    noise = np.random.normal(0, 1, (batch_size, 100))
    model.fit(noise, valid, verbose=1)
```

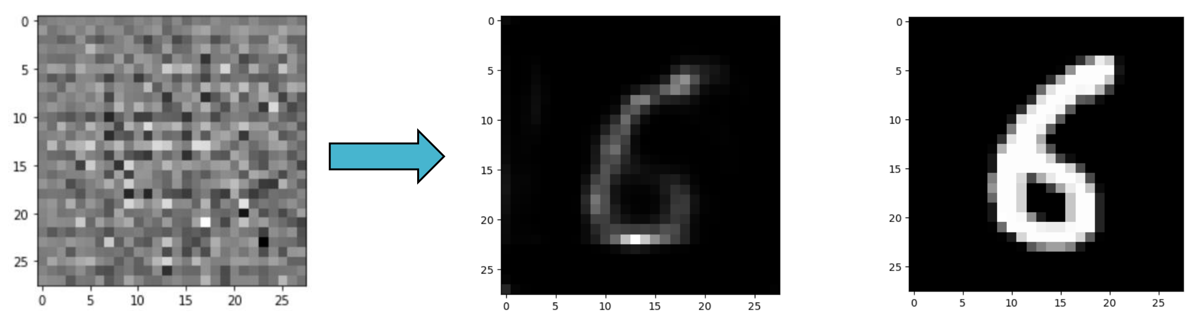




generator 를 훈련

- Test

```
original=array_to_img(Xtrain[0])
plt.imshow(original, cmap='gray')
plt.show()

np.random.seed(123)
random_noise=np.random.normal(0, 1, (1, 100)) # source of fake image
gen_result = generator.predict(random_noise)
gen_img = array_to_img(gen_result[0])
plt.imshow(gen_img, cmap='gray')
plt.show()
```



Random noise image

[실습]

- 1. 주어진 코드를 이용하여 진짜 이미지와 비슷한 가짜 이미지를 생성해 보자. (진짜 이미지는 각자 선택)
  - 진짜 이미지와 가짜이미지를 보인다

- Note. 주어진 코드는 28x28 이미지 학습에 맞도록 숫자들이 지정되어 있다. 선택한 이미지가 28x28 가 아니어도 코드가 작동하도록 코드를 수정하여 테스트 하시오

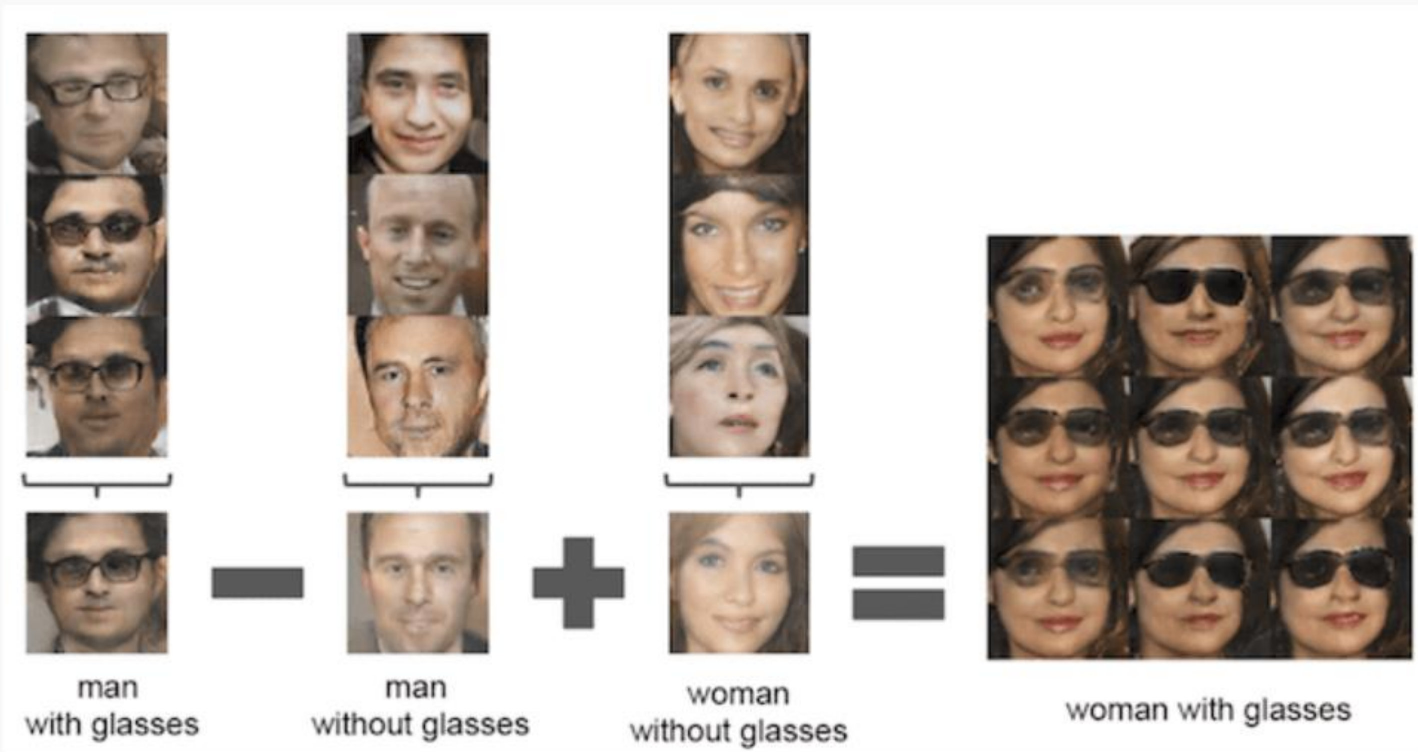
```
# load data
img = image.load_img('d:/data/6_img.png', target_size=(28,28))
```

여기를 바꾸어도 작동하도록

3. Applications of GAN

- https://machinelearningmastery.com/impressive-applications-of-generative-adversarial-networks/

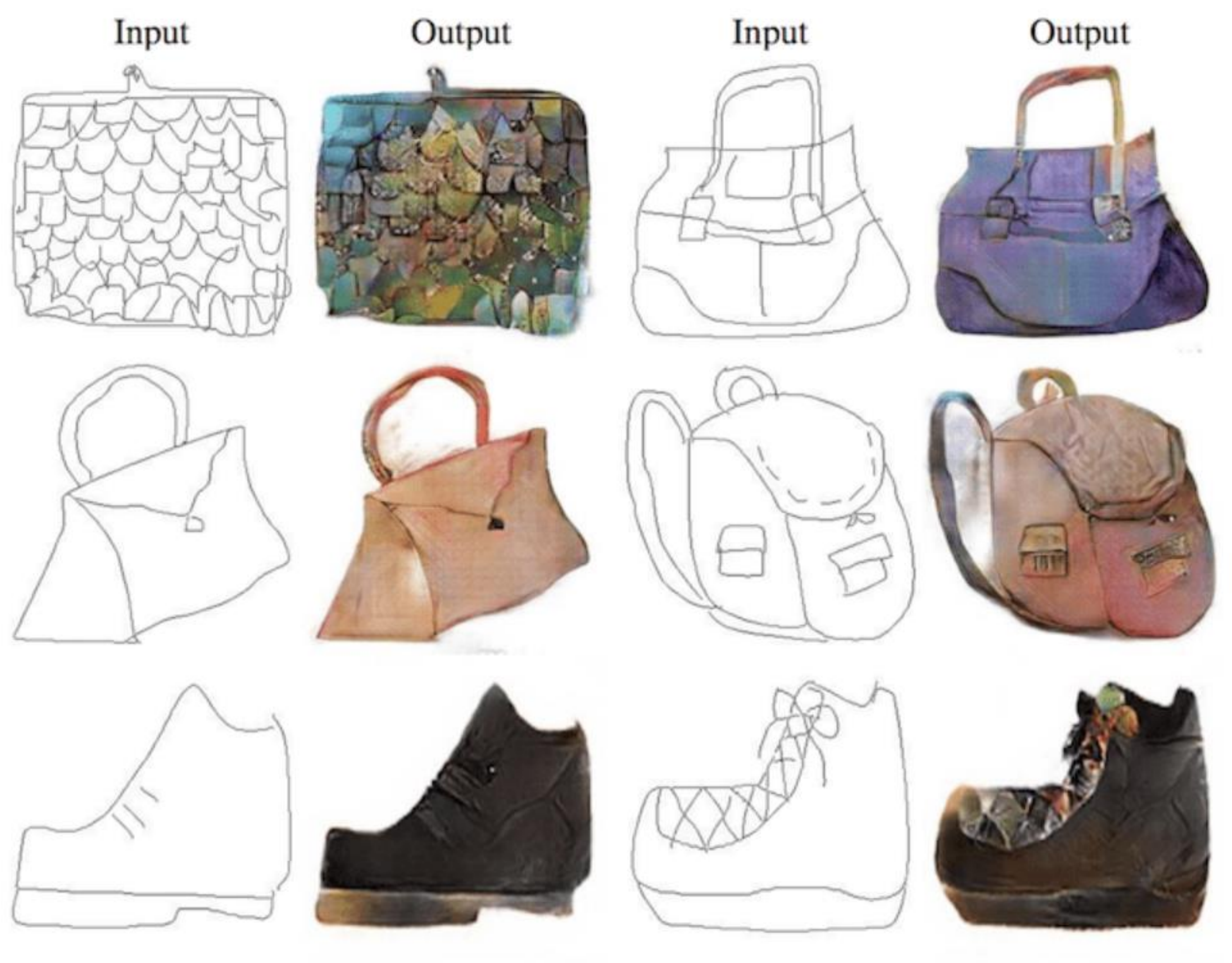
18 Impressive Applications of Generative Adversarial Networks (GANs)



Example of Vector Arithmetic for GAN-Generated Faces.Taken from Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, 2015.

3. Applications of GAN

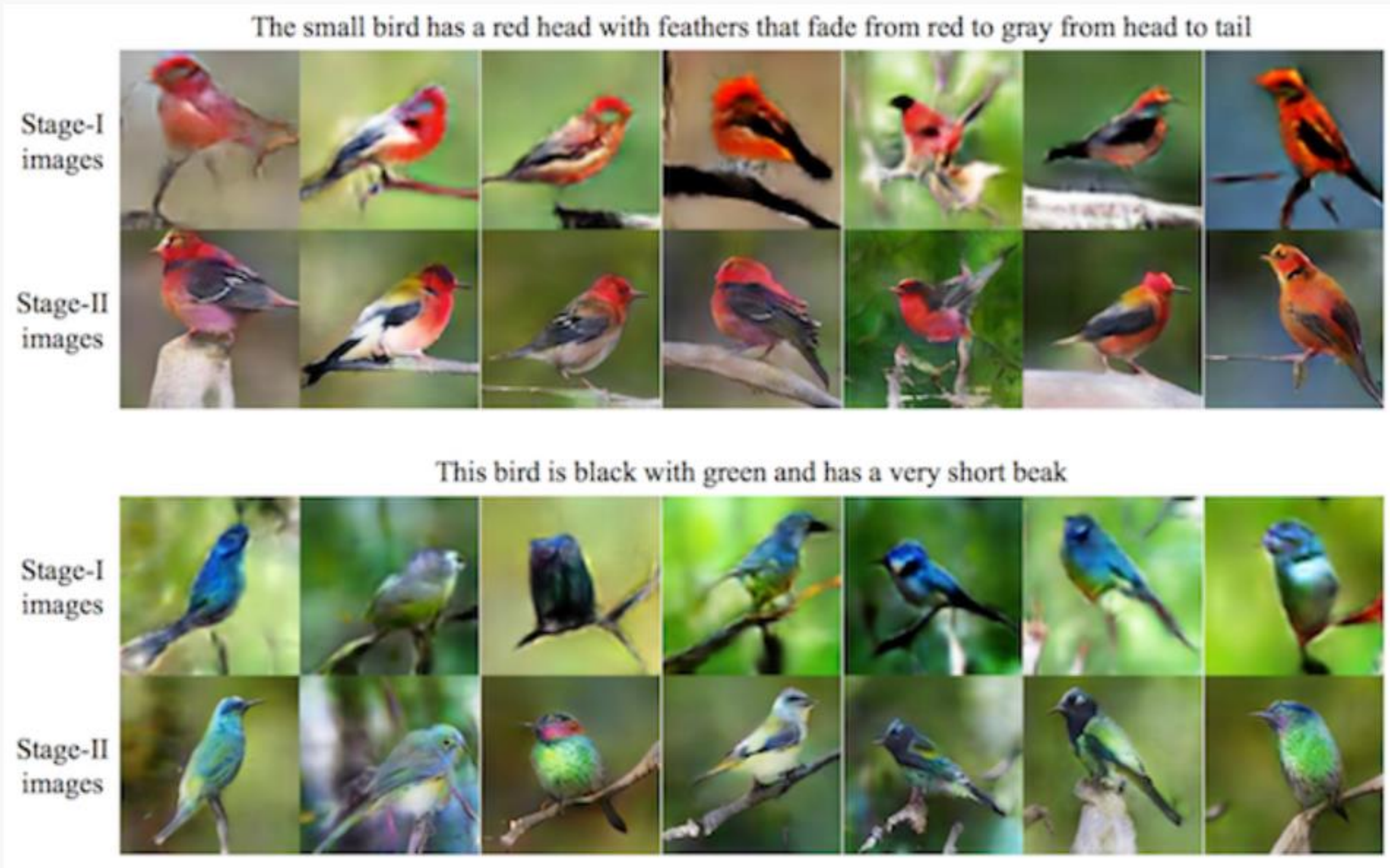
- Sketches to Color Photographs



Example of Sketches to Color Photographs With pix2pix.Taken from Image-to-Image Translation with Conditional Adversarial Networks, 2016.

3. Applications of GAN

- Text to image translation



Example of Textual Descriptions and GAN-Generated Photographs of BirdsTaken from StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks, 2016.