8. 대립생성 네트워크 (Generative Adversarial Network)

실감응용 인공지능

고병철

GAN 소개

AI Course

- **대립생성 네트워크(GAN)**
  - 2014년 이안 굿펠로우(Ian Goodfellow)가 NeurIPS에서 발표
  - Generative Adversarial Network : 서로 대립하며 생성하는 네트워크

GAN을 이용해 유명인 사진을 바탕으로 생성된 가짜 인물 이미지, NVIDIA 20108

GAN 소개

AI Course

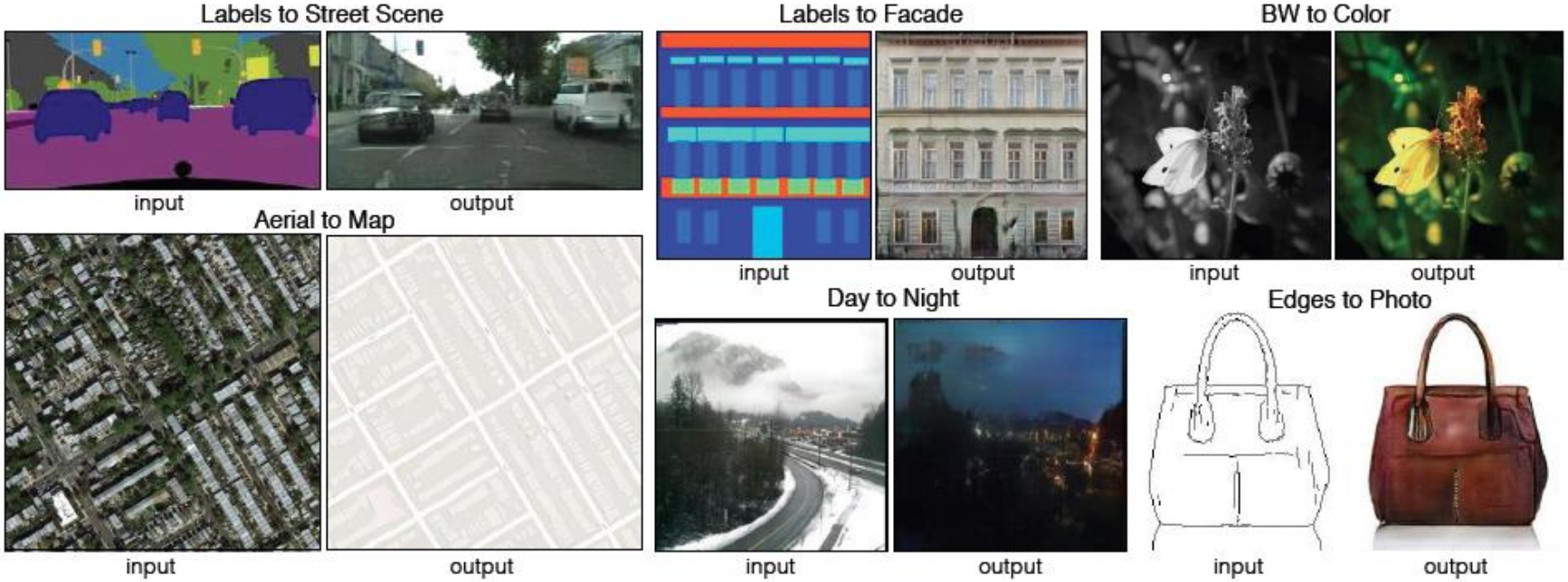
- **GAN을 통해 생성된 결과물**
  - 2017년 8월 미국 워싱턴대학교 연구진은 버락 오바마 전 미국 대통령의 가짜 영상을 GAN을 이용하여 생성

Teaser – Synthesizing Obama: Learning Lip Sync from Audio
< 출처: 유튜브, https://youtu.be/MVBe6_o4cMI >

GAN 소개

AI Course

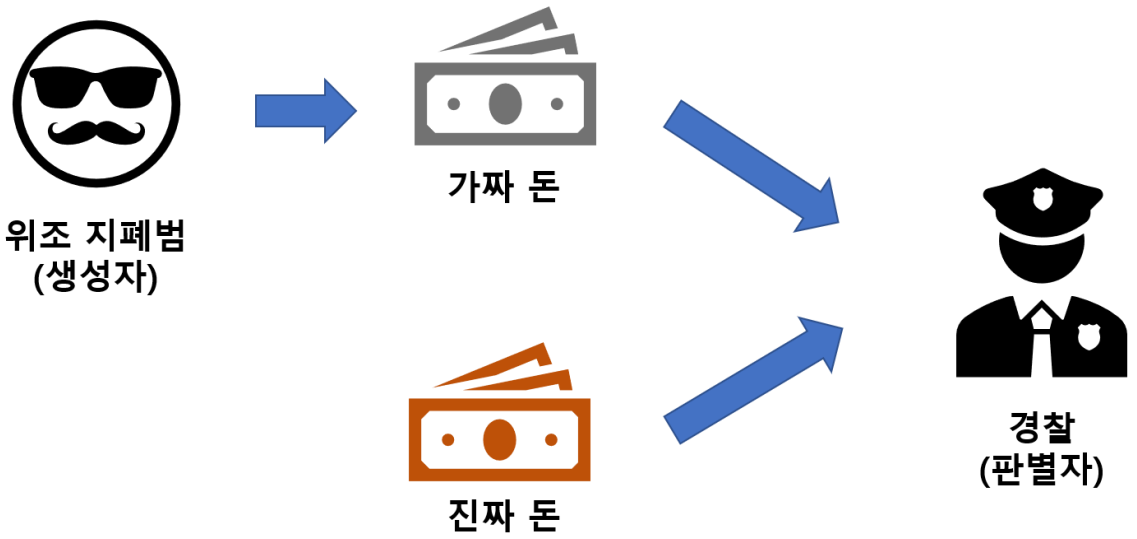
- **GAN을 통해 생성된 결과물**
  - pix2pixHD



GAN 이해

AI Course

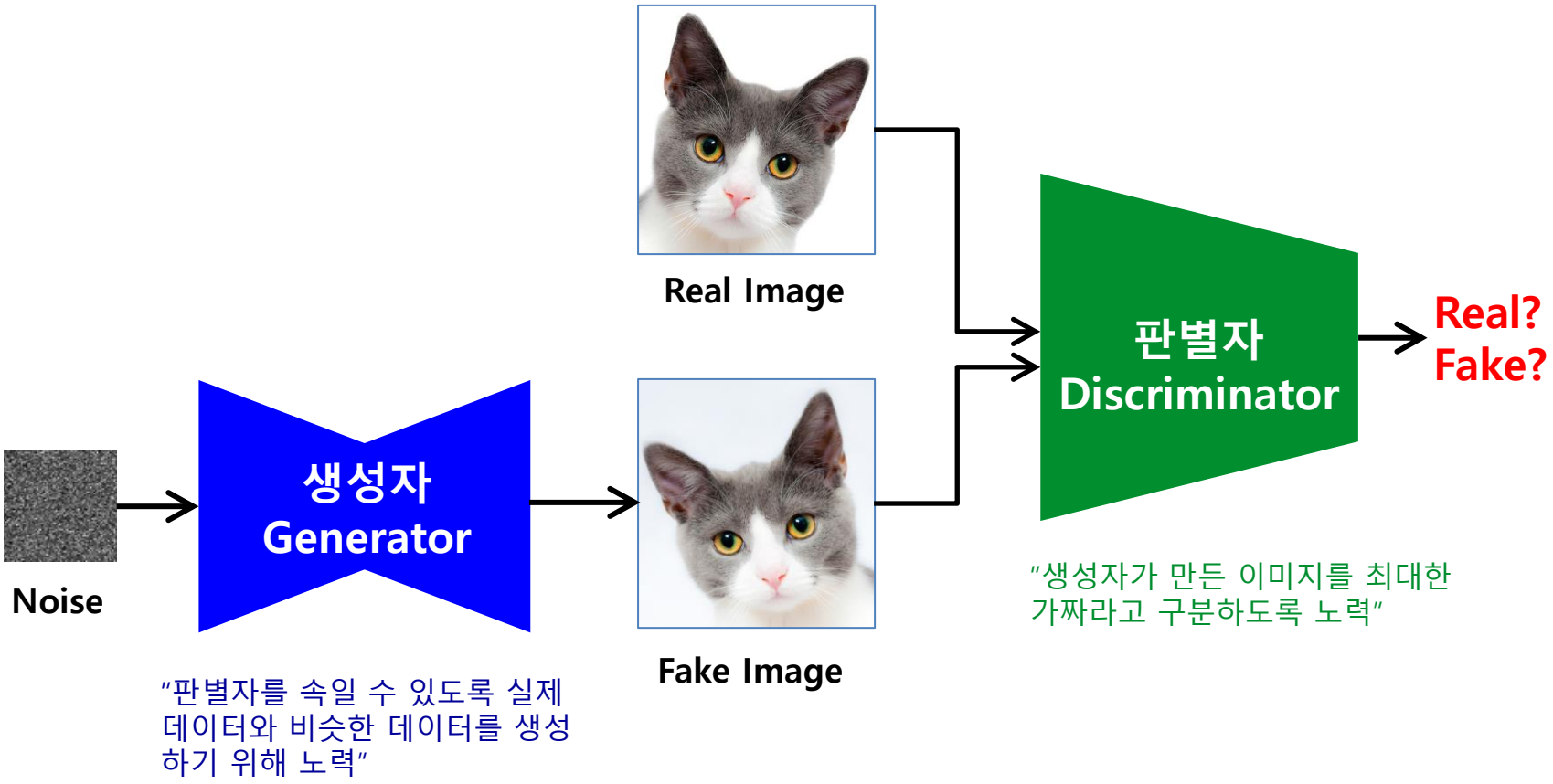
- **GAN 구성**
  - 구성 : 생성자(Generator) + 판별자(Discriminator)
  - 서로 대립하는 두 신경망을 경쟁시켜가며 점점 더 개선된 결과물을 생성하도록 학습
  - 생성자 (위조지폐범) vs 판별자 (경찰)
    - 위조지폐범 목표 : 가짜 돈을 만들되 경찰을 속일 수 있는 진짜같은 가짜 돈을 만드는 것
    - 경찰 목표 : 위조 지폐범이 만들어낸 진짜 같은 돈을 판별하는 것



GAN 이해

AI Course

- GAN의 적대적 학습(Adversarial Training)

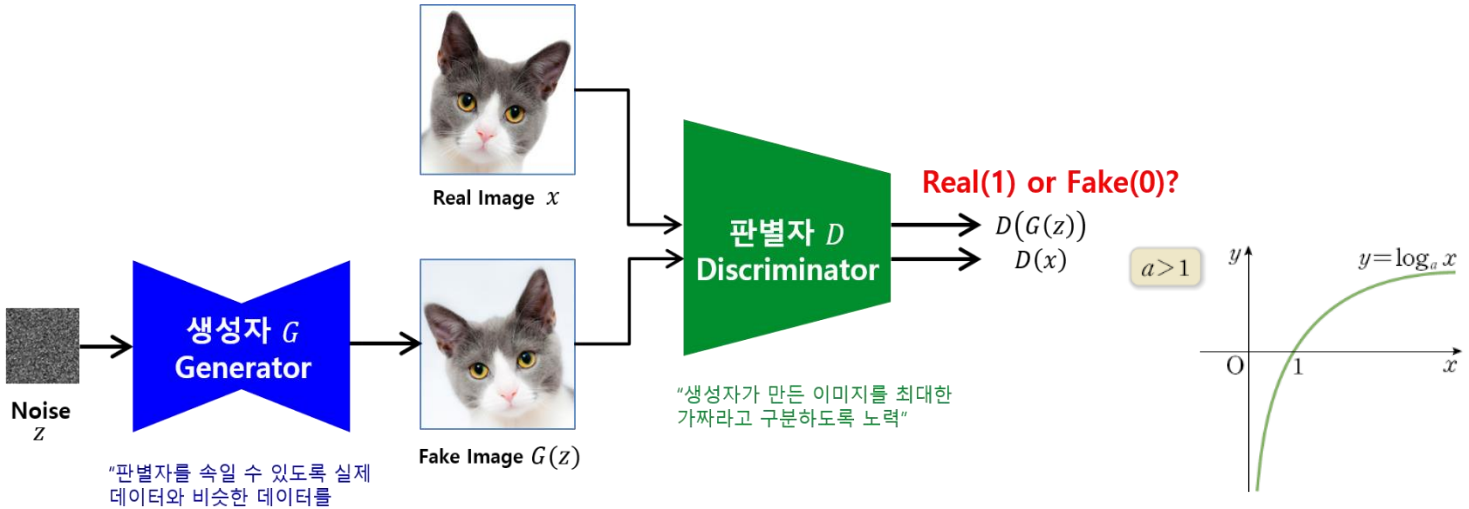


→ 생성자와 판별자의 경쟁을 통해 결과적으로 생성자는 실제 이미지와 상당히 비슷한 이미지를 생성할 수 있게 된다.

GAN 이해

AI Course

- GAN의 적대적 학습(Adversarial Training)



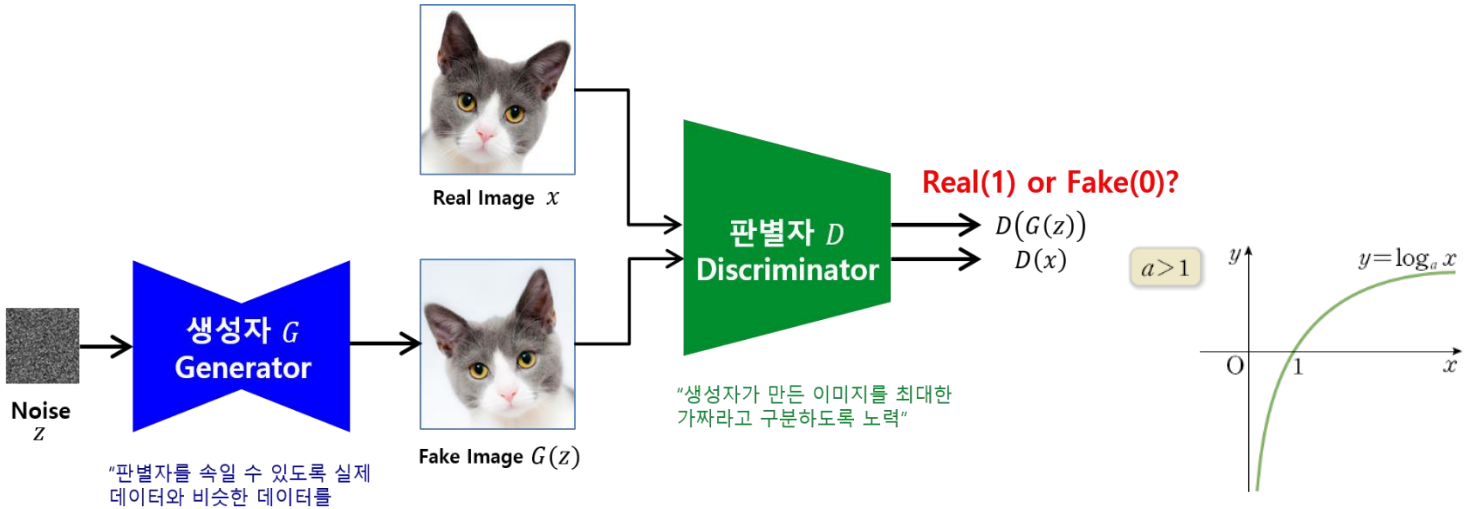
$$\min_G \max_D V(D,G) = \underset{①}{\mathbb{E}_{x\sim p_{data}(x)}[logD(x)]} + \underset{②}{\mathbb{E}_{z\sim p_x(z)}[\log(1 - D(G(z)))]}$$

- **G(생성자)는 V(D, G)가 최소가 되도록 학습 수행**
  - 생성자가 생성한 이미지를 판별자가 진짜라고 판단하도록 학습 수행 → ②번 항만을 고려
  - 생성자는 D(G(z))가 1(Real)에 가까운 값이 나오기를 원함
  - Best 결과 : log(1-D(G(z))) = log(1-1) = log0 = -∞

GAN 이해

AI Course

- GAN의 적대적 학습(Adversarial Training)



$$\min_G \max_D V(D,G) = \underset{①}{\mathbb{E}_{x\sim p_{data}(x)}[logD(x)]} + \underset{②}{\mathbb{E}_{z\sim p_x(z)}[\log(1 - D(G(z)))]}$$

- **D(판별자)는 V(D, G)가 최대가 되도록 학습 수행**
  - 판별자는 진짜이미지는 진짜(1)로 가짜이미지는 가짜(0)로 판단하도록 학습 수행 → ①, ②번 항 고려
  - 판별자는 D(y)는 1(Real)에 가까운 값이, D(G(z))는 0(Fake)에 가까운 값이 나오기를 원함
- **Best 결과 : ①항 log(D(y))=log1=0**
  ②항 log(1-D(G(z))=log(1-0)=log1=0

GAN 구현

AI Course

- **GAN을 이용하여 MNIST 손글씨 무작위 생성**

① 라이브러리 모듈 import 및 MNIST 데이터 로드

- 기존 라이브러리에서 새로운 Input, Model 함수가 추가되었음

. 텐서플로우 케라스에서 모델을 정의하는 방법중의 하나인 Functional API 관련 함수들임

```
from tensorflow.keras.datasets import mnist
from tensorflow.keras.layers import Input, Dense, Dropout
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.optimizers import Adam
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
```

GAN 구현

AI Course

② 생성자 모델 정의

- 입력층, 은닉층2, 출력층 구성
- Input : 128차원 random noise vector
- Output : 784(28x28)차원

```
n_output = 784   # 28×28 이미지 크기
n_noise = 128    # 랜덤 노이즈 벡터 크기

# 생성자 모델 정의
# 128차원의 랜덤 노이즈 벡터를 입력받아서 784차원의 벡터(28×28)이미지를 출력
generator = Sequential()
generator.add(Dense(units=256, input_dim=n_noise, activation='relu'))
generator.add(Dense(units=512, activation='relu'))
# tanh 활성화 함수 사용하여 -1~1사이로 픽셀값 통일
generator.add(Dense(units=n_output, activation='tanh'))
```

출력은 영상 크기와 동일한 784차원

훈련시 Discriminator에 입력되는 영상도 -1~1로 정규화 시켜 입력

GAN 구현

AI Course

③ 판별자 모델 정의

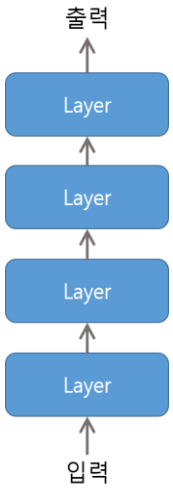
- 입력층, 은닉층1, 출력층 구성
- Input : 784차원 이미지 (-1 ~ +1)로 정규화
- Output : 0~1 사이 확률값 (0: Fake, 1: Real)

```
# 판별자 모델 정의
# 이미지(28×28=784)를 입력받아, 진짜인지 가짜인지 판단
discriminator = Sequential()
discriminator.add(Dense(256, input_dim=n_output, activation='relu'))
Discriminator.add(Dropout(0.3))
# sigmoid 활성화 함수 사용하여 0~1 사이 출력값 가짐
discriminator.add(Dense(1, activation='sigmoid'))
```

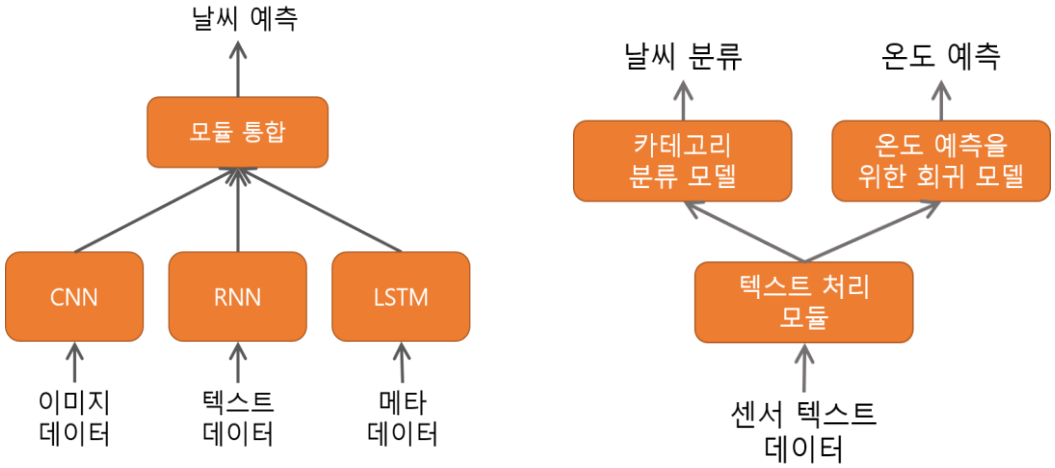
GAN 구현

AI Course

- 텐서플로우 케라스의 모델 디자인 방법
  - **Sequential API**
    - 레이어를 쌓아 구성하는 방법으로 직관적이므로 사용하기 쉬움
    - 하지만, 레이어를 공유하는 구조나 다중 입력 및 출력을 하지 못하며, 복잡한 네트워크를 구성할 때 한계를 가짐
  - **Functional API**
    - Sequential API보다 유연하게 모델을 구성할 수 있음
    - 레이어 공유 및 다중 입출력 사용한 모델 구성 가능
    - 일종의 함수(function)로 각 레이어를 정의하며, 각 함수를 조합하기 위한 연산자들을 제공, 이를 이용하여 모델을 구성함



Sequential API 구조



Sequential API의 구현 불가능 모델 예시: (왼) 다중 입력 모델, (오) 다중 출력 모델

GAN 구현

AI Course

- **Sequential API vs. Functional API 구현 코드 비교**
  - 네트워크 구성
    - ✓ Input: 128 dim. Vector, Output: 1, activation func-sigmoid
    - ✓ Hidden layer 1: 256 node, activation func-ReLU
    - ✓ Hidden layer 2: 512 node, activation func-ReLU

```
# Sequential API를 이용한 모델 구현
# tensorflow.keras.models.Sequential 초기 모델 설정해서 add() 메소드 사용해서 레이어 추가
s_model = tf.keras.models.Sequential()
s_model.add(tf.keras.layers.Dense(units=256 ,input_shape=(128, ), activation='relu'))
s_model.add(tf.keras.layers.Dense(units=512, activation='relu'))
s_model.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
```

```
# Functional API를 이용한 모델 구현
# tensorflow.keras.models.Model의 inputs() & outputs() 메소드 설정을 이용해 모델 정의
inputs = tf.keras.Input(shape=(128, ))
x = tf.keras.layers.Dense(units=256, activation='relu')(inputs)
x = tf.keras.layers.Dense(units=512, activation='relu')(x)
outputs = tf.keras.layers.Dense(units=1, activation='sigmoid')(x)
model = tf.keras.models.Model(inputs=inputs, outputs=outputs)
```

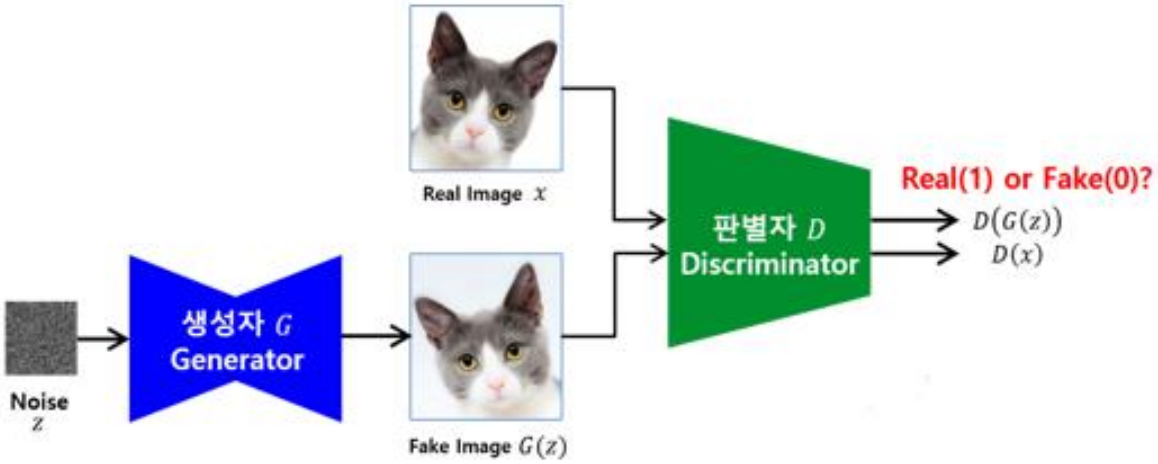
GAN 구현

AI Course

④ 생성자와 판변자 모델을 통합한 GAN 모델 구현

- Functional API를 이용한 모델 정의
- GAN 모델 구성
  - (random noise) → 생성자 → (784 dim, 28x28 image) → 판별자 → (1 dim, 진짜or가짜)
  - Input : 128 random noise
  - Output : discriminator's output

```
# 생성자와 판별자 모델을 통합한 GAN 모델 구현
g_input = Input(shape=(n_noise,))
g_output = discriminator(generator(g_input))
gan = Model(g_input, g_output)
```



GAN 구현

AI Course

⑤ 최적화 함수 및 손실 함수 설정

- GAN에서 학습시켜야 할 신경망 : 판별자 & 생성자

- **판별자 모델**
  - 학습목표: 판별자 신경망이 진짜 및 가짜 이미지를 올바르게 분류하는 방향으로 학습 수행
  - 손실 함수 : 판별자 모델 학습을 위해 진짜(1) 및 가짜(0)에 대해 분류가 가능한 binary_crossentropy를 설정
  - 최적화 함수 : Adam
  - Discriminator에 대해 컴파일 수행

```
# 판별자 모델 학습을 위한 최적화 함수 및 손실 함수 설정
discriminator.trainable = True
adam = tf.optimizers.Adam(lr=0.0002, beta_1=0.5)
discriminator.compile(loss='binary_crossentropy', optimizer=adam)
```

GAN 구현

AI Course

⑤ 최적화 함수 및 손실 함수 설정

– 생성자 모델

- 학습목표 : 생성자가 생성한 가짜 이미지를 판별자가 진짜라고 판단하는 방향으로 생성자 신경망을 학습
  - 생성자는 단독으로 학습을 수행하는 것이 아니라 판별자 모델이 함께 학습에 사용되어야함. 따라서, 앞서 정의한 GAN 모델을 이용하여 생성자 모델을 학습함
  - 생성자 학습을 위해서 GAN모델을 사용하므로 학습(가중치 업데이트)과정에서 GAN모델의 판별자의 학습은 수행되지 않고 오직 생성자만이 학습을 수행해야 함
    - » 이를위해 discriminator.trainable = False로 설정
- 손실 함수 : GAN 모델의 최종적인 출력값은 판별자의 진짜(1) 및 가짜(0)에 대한 분류 값이므로 binary_crossentropy를 사용
- 최적화 함수 : Adam
- Discriminator에 대해 컴파일 수행

```
# GAN 모델 학습을 위한 최적화 함수 및 손실 함수 설정
# GAN 모델에 대한 학습은 오직 생성자에 대한 가중치 업데이트가 수행되어야 하므로 판별자 모델의 학습 여부를 False로 설정함
discriminator.trainable = False
gan.compile(loss='binary_crossentropy', optimizer=adam)
```

GAN 구현

AI Course

⑥ 학습 수행

- 에폭, 미니배치, 결과 저장 간격 등 설정
- MNIST 데이터 로드 : 판별자 모델 학습에서 진짜 이미지로 사용하므로 훈련 이미지만 로드 및 reshape
- 클래스 레이블 필요 없음. 이미지 픽셀값 [-1~1]로 정규화

```
# 학습 관련 변수 설정
n_epoch = 100           # 에폭 수
batch_size = 128        # 미니 배치 크기
Saving_interval = 10    # 결과 이미지 저장을 위한 간격

# MNIST 데이터 불러오기 및 정규화
(train_images, _), (_, _) = mnist.load_data()
n_image = len(train_images) # 훈련 이미지 개수 추출 : 60,000개
train_images = train_images.reshape((n_image, 784)) # 28x28

# 이미지 픽셀값을 [–1~1]로 정규화
train_images = (train_images - 127.5) / 127.5
```

GAN 구현

AI Course

⑥ 학습 수행

- 배치 데이터 생성
  - Tensorflow.data.Dataset.from_tensor_slices() 함수 :배치크기만큼 데이터 배치를 만들고 섞어줌

```
train_dataset = tf.data.Dataset.from_tensor_slices(train_images).
shuffle(n_image).batch(batch_size)
```

- 에폭 수 만큼 반복하면서 모델 학습
  1) 모델 오차 계산 및 가중치 갱신
  2) 중간 과정에서 생성되는 이미지 확인

```
for i in range(n_epoch):
    print('Epoch Num : {}/{}'.format((i + 1), n_epoch))
    ##############################
    # 1) 모델 오차 계산 및 가중치 갱신
    ##############################
    ##############################
    # 2) 중간 과정 생성 이미지 확인
    ##############################
```

GAN 구현

AI Course

⑥ 학습 수행

1) 모델 오차 계산 및 가중치 갱신

- **train_on_batch(x, y)** : x-input data, y-target label로 현재 입력한 x, y에 대하여 가중치 업데이트를 수행함

```
# 1) 미니-배치별 모델 가중치 갱신 및 오차 계산
for image_batch in train_dataset:     # 미니-배치별 가중치 갱신
    # 현재 배치 크기만큼 진짜 및 가짜 레이블 배열 생성
    n_bt_imgs = len(image_batch)              # 현재 반복에서의 배치 이미지 개수 추출
    true_labels = np.ones((n_bt_imgs, 1))  # 진짜 레이블(1) 생성
    fake_labels = np.zeros((n_bt_imgs, 1)) # 가짜 레이블(0) 생성

    # 현재 배치에 대한 판별자 모델의 1회 가중치 갱신 및 오차 계산
    # 진짜 이미지들에 대한 판별자 모델 가중치 갱신 및 오차 계산
    d_loss_real = discriminator.train_on_batch(image_batch, true_labels)
    # 가짜 이미지들에 대한 판별자 모델 가중치 갱신 및 오차 계산
    noise = np.random.normal(0, 1, (n_bt_imgs, n_noise))
    gen_imgs = generator.predict(noise)
    d_loss_fake = discriminator.train_on_batch(gen_imgs, fake_labels)
     # 판별자 모델의 최종 오차 : d_loss_real과 d_loss_fake의 평균값으로 사용
    d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)
    # GAN 모델에 대한 학습은 오직 생성자에 대한 가중치 업데이트가 수행
    # 현재 배치에 대한 GAN 모델 1회 가중치 갱신 및 오차 계산=>Generator만 학습
    g_loss = gan.train_on_batch(noise, true_labels)
```

GAN 구현

AI Course

⑥ 학습 수행

2) 중간 과정에서의 생성 이미지 확인

- 학습된 생성자로 25개의 이미지 생성시키고 이미지 출력 및 저장

```
 # 2) 중간 과정 생성 이미지 확인
if (i + 1) % saving_interval == 0 or i == 0:
    print('Epoch:%d' % (i + 1), 'd_loss:%.4f' % d_loss, ' g_loss:%.4f' % g_loss)

    # 학습된 생성자 모델을 이용하여 25개의 이미지 생성
    noise = np.random.normal(0, 1, (25, n_noise)) # 25개의 노이즈 벡터 생성
    gen_imgs = generator.predict(noise) # 생성자를 이용한 25개의 이미지 생성

    # 이미지 픽셀값 0~1로 조정
    gen_imgs = 0.5 * gen_imgs + 0.5

    # 생성된 이미지 출력 및 저장
    fig, axs = plt.subplots(5, 5) # 5×5 subplot에 25개 이미지 출력
    count = 0
    for j in range(5):
        for k in range(5):
            axs[j, k].imshow(np.reshape(gen_imgs[count], (28, 28)))
            axs[j, k].axis('off')
            count += 1
     plt.show()
     fig.savefig("gan_results/g_mnist_%d.png" % (i + 1)) # 반드시 경로 존재
```

GAN 구현

AI Course

- 전체 코드

```
from tensorflow.keras.datasets import mnist
from tensorflow.keras.layers import Input, Dense,Dropout
from tensorflow.keras.models import Sequential, Model
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

n_output = 784    # 28×28 이미지 크기
n_noise = 128     # 랜덤 노이즈 벡터 크기

# 생성자 모델 정의, 128차원의 랜덤 노이즈 벡터를 입력받아서 784차원의 벡터(28×28) 이미지를 출력
generator = Sequential()
generator.add(Dense(units=256 , input_dim=n_noise, activation='relu'))
generator.add(Dense(units=512, activation='relu'))
generator.add(Dense(units=n_output, activation='tanh'))

# 판별자 모델 정의
discriminator = Sequential()
discriminator.add(Dense(256, input_dim=n_output, activation='relu'))
discriminator.add(Dropout(0.3))
discriminator.add(Dense(1, activation='sigmoid'))
```

GAN 구현

AI Course

```
# 생성자와 판별자 모델을 통합한 GAN 모델 구현
g_input = Input(shape=(n_noise,))
g_output = discriminator(generator(g_input))
gan = Model(g_input, g_output)

# 판별자 모델 학습을 위한 최적화 함수 및 손실 함수 설정
discriminator.trainable = True
adam = tf.optimizers.Adam(lr=0.0002, beta_1=0.5)
discriminator.compile(loss='binary_crossentropy', optimizer=adam)

# GAN 모델 학습을 위한 최적화 함수 및 손실 함수 설정
# GAN 모델에 대한 학습은 오직 생성자에 대한 가중치 업데이트가 수행되어야 하므로 판별자 모델의 학습 여부를 False로 설정함
discriminator.trainable = False
gan.compile(loss='binary_crossentropy', optimizer=adam)

# GAN 모델 구조 출력
gan.summary()
```

GAN 구현

AI Course

```
# 모델 학습
n_epoch = 100
batch_size = 128
saving_interval = 10

# MNIST 데이터 불러오기
# 모델 학습을 위해서는 훈련 이미지만 사용할 것이기 때문에 train_images만 로드
(train_images, _), (_, _) = mnist.load_data()

buffer_size = len(train_images)  # 60000
train_images = train_images.reshape((buffer_size, n_output)) #28x28


# 이미지 픽셀값을 [-1~1]로 정규화
train_images = (train_images - 127.5) / 127.5

train_dataset = tf.data.Dataset.from_tensor_slices(train_images).shuf
fle(buffer_size).batch(batch_size)
```

GAN 구현

AI Course

```
for i in range(n_epoch):
    print('Epoch Num : {}/{}'.format((i + 1), n_epoch))

    # 1) 미니-배치별 모델 가중치 갱신 및 오차 계산
    for image_batch in train_dataset:    # 미니-배치별 가중갱신
        # 현재 배치 크기만큼 진짜 및 가짜 레이블 배열 생성
        n_bt_imgs = len(image_batch)
        true_labels = np.ones((n_bt_imgs, 1))  # 진짜 레이블(1) 생성
        fake_labels = np.zeros((n_bt_imgs, 1)) # 가짜 레이블(0) 생성

        # 현재 배치에 대한 판별자 모델의 가중치 1회 갱신 및 오차 갱신
        # 진짜 이미지에 대한 판별자 모델 가중치 갱신 및 오차 계산
        d_loss_real = discriminator.train_on_batch(image_batch, true_l
abels)
        # 가짜 이미지에 대한 판별자 모델 가중치 갱신 및 오차 계산
        noise = np.random.normal(0, 1, (n_bt_imgs, n_noise))
        gen_imgs = generator.predict(noise)
        d_loss_fake = discriminator.train_on_batch(gen_imgs, fake_labe
ls)
        d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

        # 현재 배치에 대한 GAN 모델 가중치 1회 갱신 및 오차 계산
        g_loss = gan.train_on_batch(noise, true_labels)
```

GAN 구현

AI Course

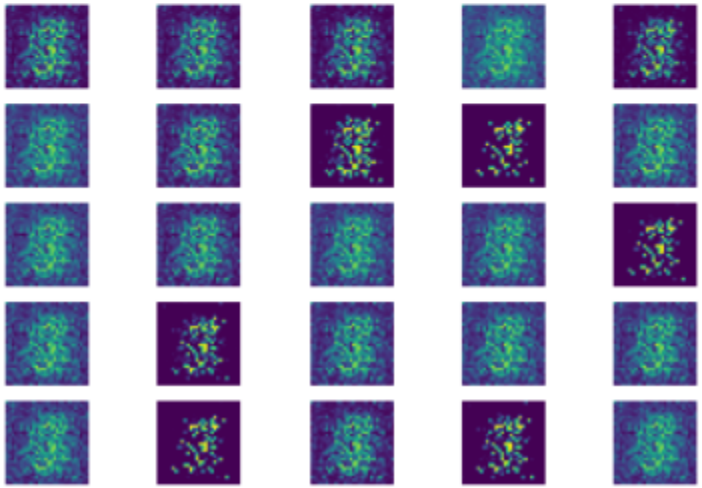
```
# 2) 중간 과정 생성되는 이미지 확인
    if (i +1) % saving_interval == 0 or i == 1:
        print('Epoch:%d' % (i + 1),
              'd_loss:%.4f' % d_loss, ' g_loss:%.4f' % g_loss)

        # 학습된 생성자 모델을 이용하여 25개의 이미지 생성
        noise = np.random.normal(0, 1, (25, n_noise))   # 25개의 노이즈 벡터 생성
        gen_imgs = generator.predict(noise)   # 생성자를 이용한 25개의 이미지 생성

        # 이미지 픽셀값 0~1로 조정
        gen_imgs = 0.5 * gen_imgs + 0.5

        # 생성된 이미지 출력 및 저장
        fig, axs = plt.subplots(5, 5)
        count = 0
        for j in range(5):
            for k in range(5):
                axs[j, k].imshow(np.reshape(gen_imgs[count], (28, 28)))
                axs[j, k].axis('off')
                count += 1
        plt.show()
        fig.savefig("gan_results/g_mnist_%d.png" % (i + 1)) #경로가 지정되
어 있지 않다면 이 부분을 막고 실행
```

GAN 구현

AI Course

- 실행 결과

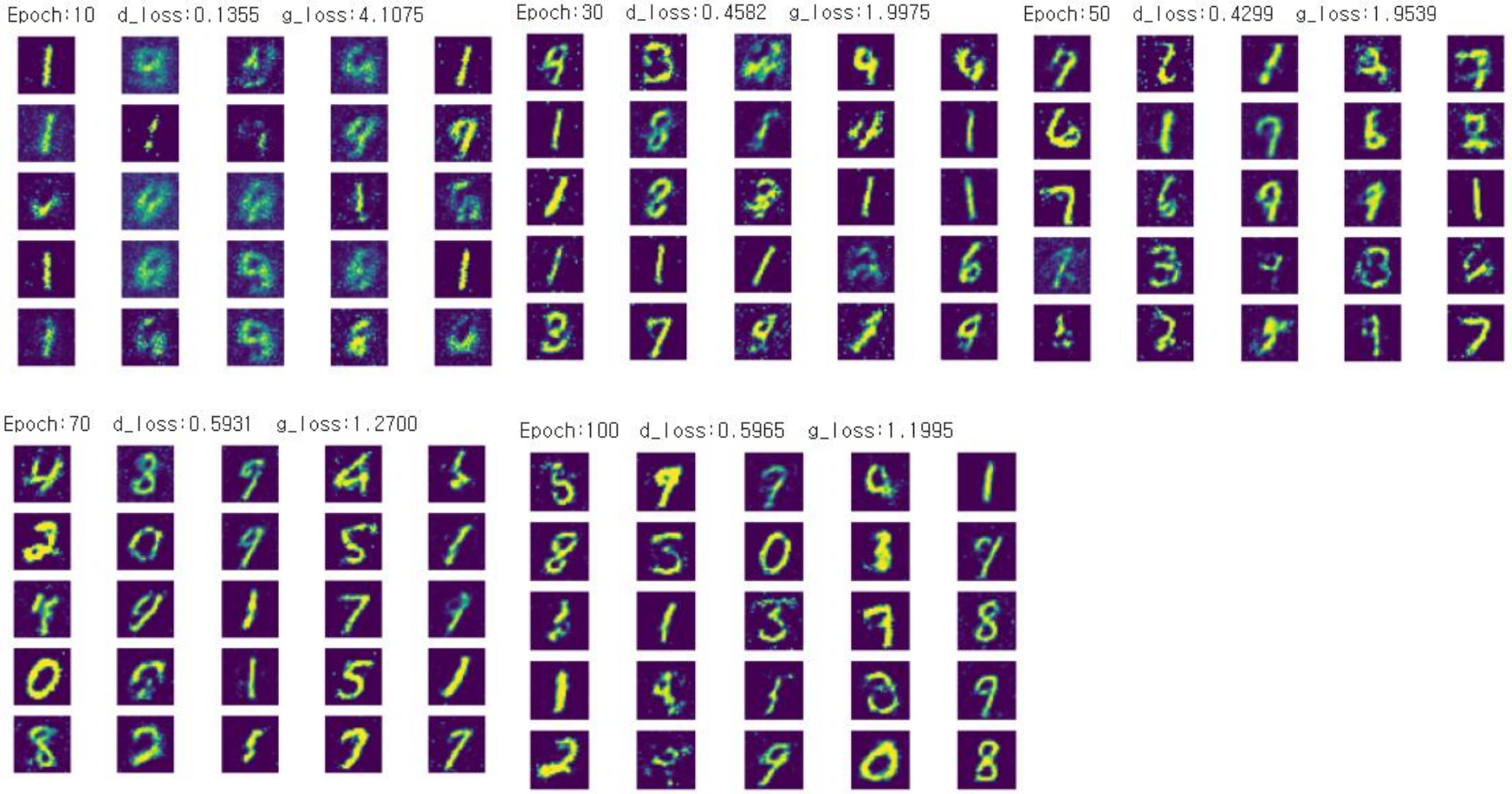
```
Model: "functional_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         [(None, 128)]             0
_________________________________________________________________
sequential_6 (Sequential)    (None, 784)               566800
_________________________________________________________________
sequential_7 (Sequential)    (None, 1)                 201217
=================================================================
Total params: 768,017
Trainable params: 566,800
Non-trainable params: 201,217
_________________________________________________________________
Epoch Num : 0/100
Epoch:0  d_loss:0.0270  g_loss:8.6927
```



GAN 구현

AI Course

- Epoch 수에 따른 생성된 이미지 결과



GAN 구현

AI Course

- 1. GAN은 이안 굿펠로우(Ian Goodfellow)가 2014년 NeurIPS에서 발표한 논문에서 제안된 딥러닝 네트워크로 ‘Generative Adversarial Network’의 약자로 생성자(Generator)와 판별자(Discriminator)라는 두 신경망이 서로 경쟁하면서 더 개선된 결과물을 생성하도록 학습하게 된다.

- 2. GAN은 진짜에 가까운 가짜 데이터를 생성하기 위해서 생성자(Generator)는 판별자(Discriminator)를 속일 수 있는 실제 데이터와 비슷한 데이터를 생성할 수 있는 능력을 가질 수 있는 방향으로 신경망을 개선해 나갈 것이며, 판별자는 생성자가 만든 이미지를 최대한 가짜라고 구분하는 능력을 가질 수 있도록 학습한다.

GAN 구현

AI Course

- 3. GAN은 생성자 신경망과 판별자 신경망을 학습시켜야 하며, 판별자는 가짜 및 진짜 데이터를 가 지고 학습을 수행하면 된다. 하지만 생성자의 학습은 생성자가 생성한 가짜 이미지를 판별자가 진짜 이미지라고 판단하는 방향으로 생성자 신경망은 학습되어야 하므로 생성자와 판별자 모델 을 통합한 모델을 구성하고, 이를 학습에 사용하여야 한다. 이때, 주의할 점은 생성자와 학습자가 통합된 GAN 모델은 생성자를 학습시키기 위한 용도로 사용하므로 **GAN 모델을 학습시킬 때는 생성자와 관련된 가중치만이 업데이트되어야** 하고, 판별자의 가중치는 업데이트가 되어서는 안 된다.

- 4. 텐서플로우 케라스의 모델을 디자인하는 방법은 Sequential API와 Functional API를 기반으로 하는 방법이 존재한다..

GAN 구현

AI Course

- 5. Sequential API 방법은 레이어를 쌓아 구성하는 방법으로 직관적이므로 사용하기 쉽다는 특징을 가진다. 하지만 이 방법은 레이어를 공유하는 구조나 다중 입력 및 출력을 사용하지 못하며 복잡한 네트워크를 구성할 때 한계가 존재한다.

- 6. Functional API는 Sequential API보다 더 유연하게 모델을 구성할 수 있으며 Sequential API가 가졌던 공유 레이어 및 다중 입출력을 사용하여 모델을 처리할 수 있다.

연습문제

AI Course

- 1. 연습 8-1에서 생성자와 판별자 모델에 대하여 은닉층을 증가시켜 보고 그에 따른 성능을 비교해보자.

- 2. 연습 8-1에서 생성자에 입력되는 랜덤 노이즈 벡터 크기를 변경해 보고 그에 따른 결과를 확인해보자.

- 3. 6장 연습문제에서 제시되었던 Fashion_MNIST 데이터셋을 사용하여 GAN을 학습시켜보고 생성되는 이미지를 관찰해보자.

- 4. https://github.com/nniceko/AI-Book의 upgrade GAN을 구현하고 테스트해보자. 연습 8-1과의 차이점을 분석해 보자.