14장 생성 모델

학습 목표

- 오토인코더를 이해한다.
- 오토인코더를 이용한 노이즈 제거를 알아본다.
- GAN 생성 모델을 이해한다..

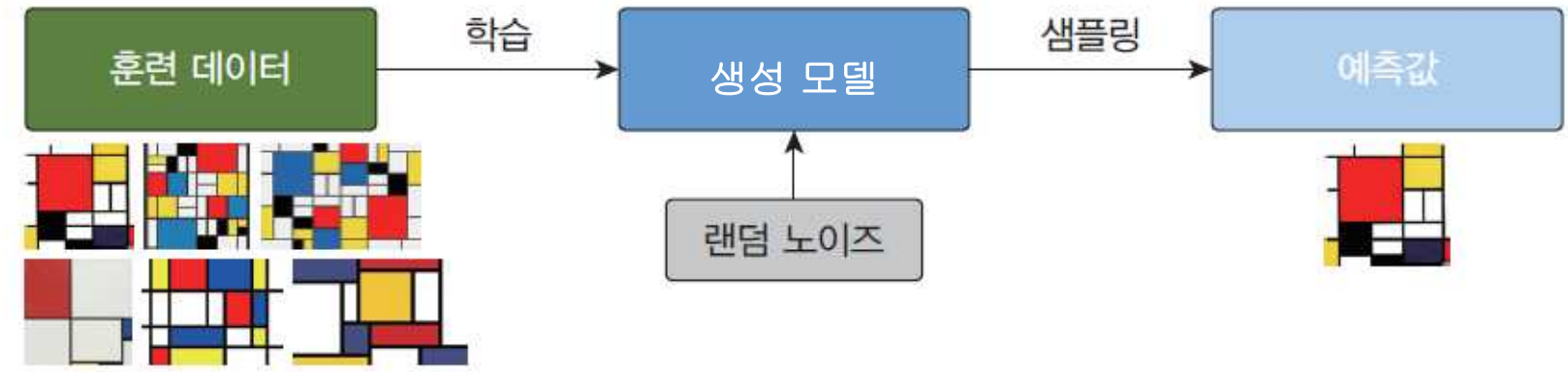
생성 모델이란?

- 생성 모델(generative model)은 훈련 데이터의 규칙성 또는 패턴을 자동으로 발견하고 학습하여, 훈련 데이터의 확률 분포와 유사하지만, 새로운 샘플을 생성하는 신경망이다.
- 생성 모델은 훈련 데이터들의 잠재 공간 표현을 학습할 수 있으며, 학습이 종료된 후에, 잠재 공간에서 랜덤으로 하나의 좌표가 입력되면 거기에 대응되는 출력을 만들어 낼 수 있다.

그림 14-1 GAN으로 생성된 사실적인 사진

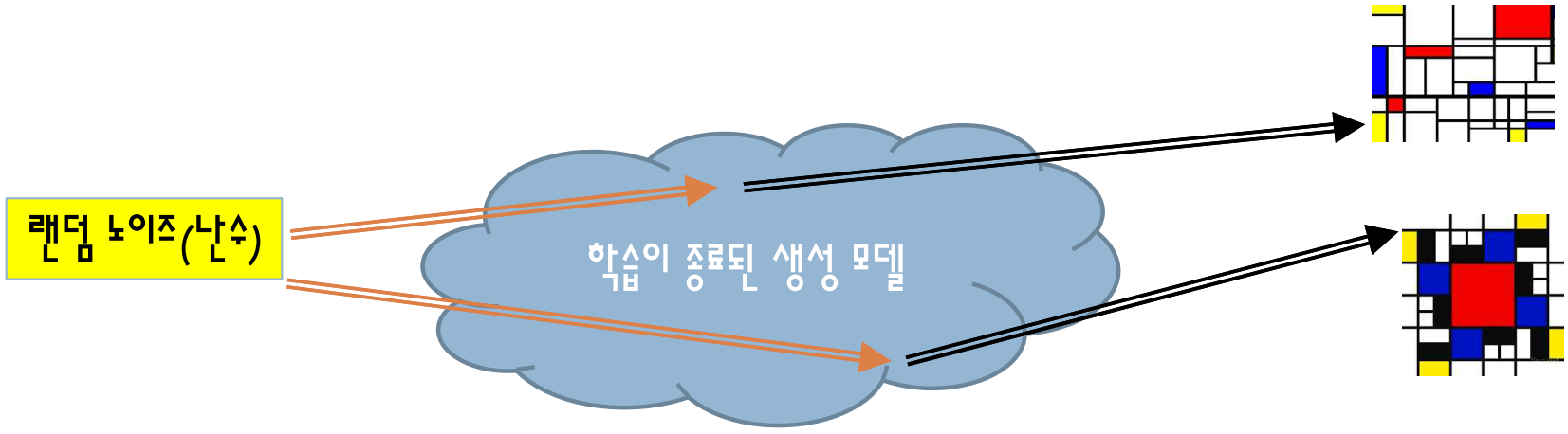
생성 모델은 훈련 데이터를 생성하는 규칙을 파악한다

- 만약 생성 모델이 사실적인 몬드리안 스타일 그림을 생성할 수 있다면, 생성 모델은 몬드리안 스타일을 지배하는 어떤 일반적인 규칙을 학습하였다고 봐야 한다.



생성 모델은 결정적이기보다는 확률적이어야 한다.

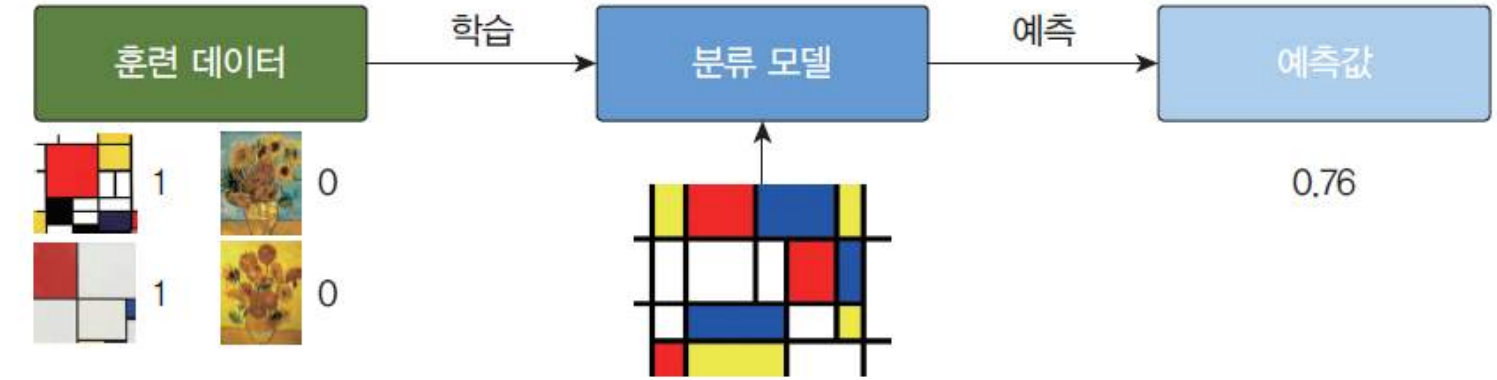
- 예를 들어서 우리는 모든 샘플 이미지에 대하여 각 픽셀 위치에서 픽셀값의 평균을 계산하면, 샘플 이미지와 유사한 출력을 생성할 수 있다. 하지만 이와 같은 고정된 계산일 경우, 생성 모델이 매번 동일한 출력을 생성하기 때문에 전혀 흥미롭지 않을 것이다.
- 따라서 생성 모델은 출력에 영향을 미치는 확률적인 무작위 요소를 포함해야 한다. 생성 모델에서는 학습이 종료된 후에 입력되는 랜덤 노이즈가 이 역할을 한다.

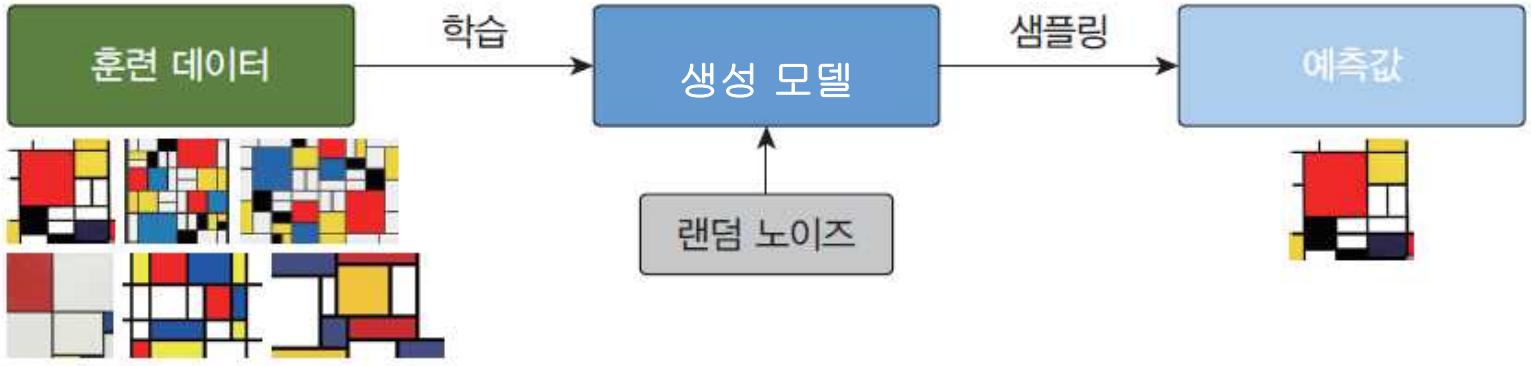


그림이 저장된 확률 분포로 생각할 수 있다.

분류 모델과 생성 모델의 차이

- 분류 모델(discriminative modeling) 은 데이터 x와 레이블 y가 주어지면 분류 모델은 x가 어떤 부류에 속하는지를 판단한다.

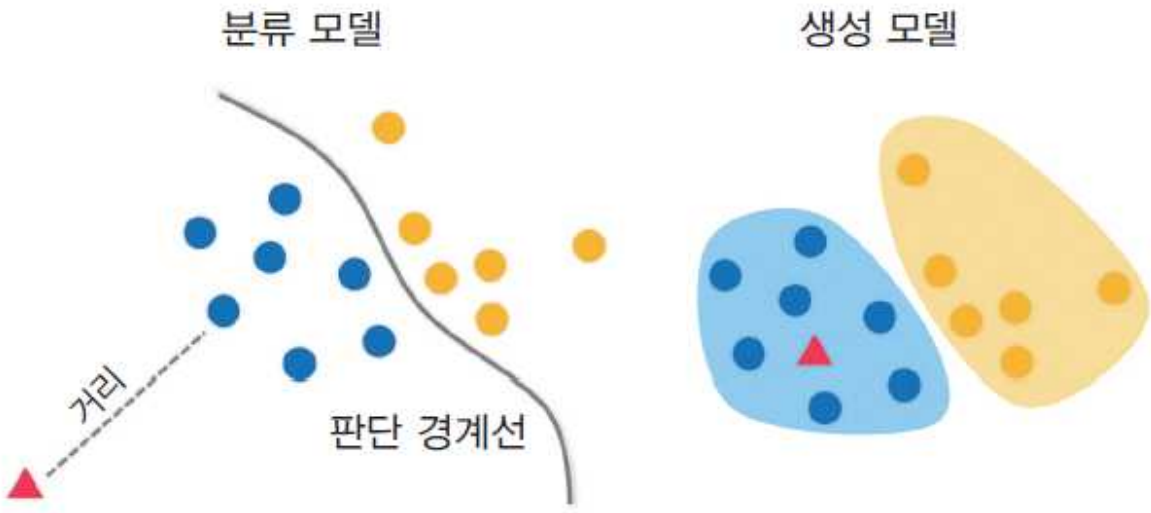




분류 모델과 생성 모델의 차이

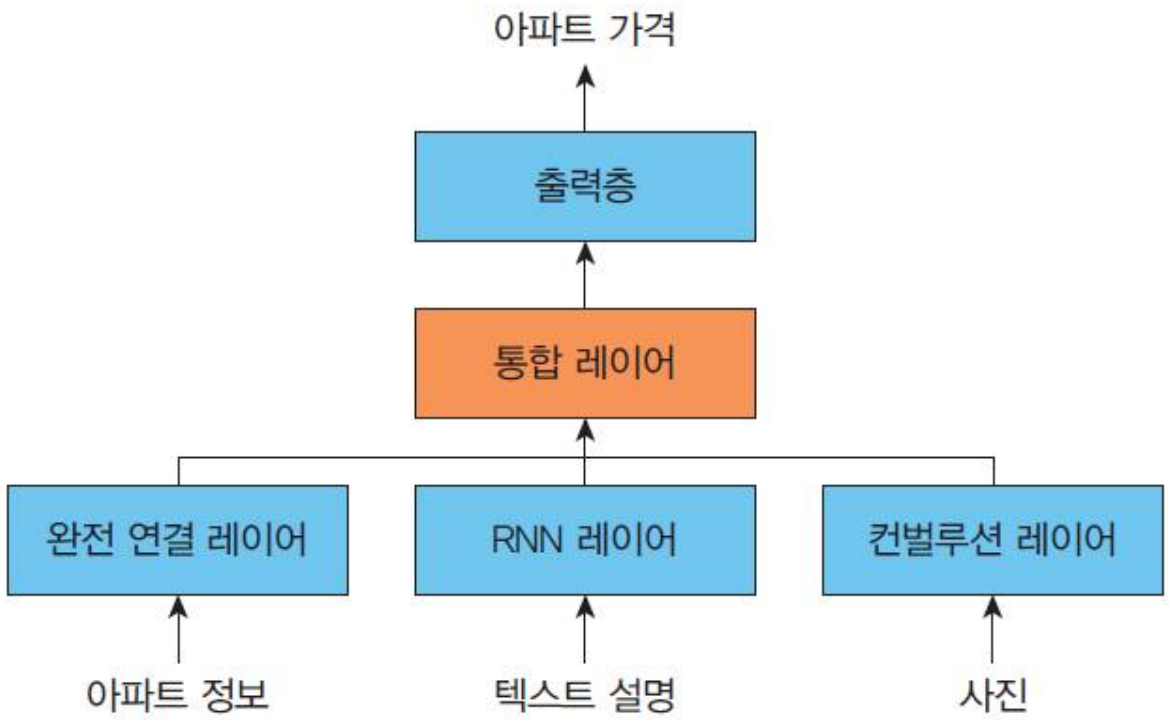
- 분류 모델은 조건부 확률인 p(y|x)를 알아내는 것이다. 즉 샘플 x가 주어진 상태에서 레이블 y의 확률을 추정한다.

- 생성 모델은 입력 데이터의 확률 분포 p(x)를 알려고 노력한다.

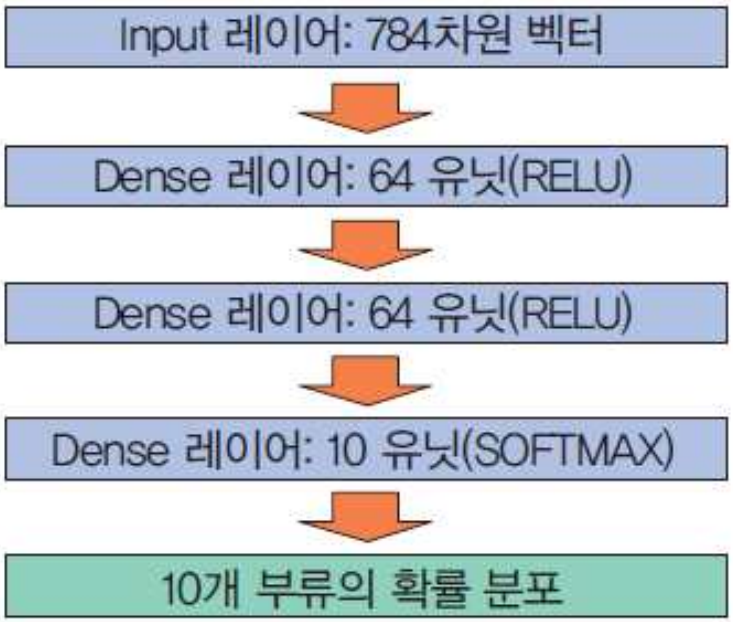


케라스의 함수형 API

- 함수형 API를 사용하게 되면, 우리가 원하는 방식으로 객체들을 연결하여 사용할 수 있다. 이번 장에서는 은닉층의 출력을 알아야 한다. 이런 경우에 함수형 API를 사용하는 것이 좋다.



예제: *MNIST* 숫자 이미지를 처리하는 신경망



함수형 *API*로 작성해보자.

예제: MNIST 숫자 이미지를 처리하는 신경망

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(784,))                          # (1)
dense = layers.Dense(64, activation="relu")                 # (2)
x = dense(inputs)                                           # (3)
x = layers.Dense(64, activation="relu")(x)                  # (4)
outputs = layers.Dense(10)(x)                               # (5)

model = keras.Model(inputs=inputs, outputs=outputs)
```

tmp = layers.Dense(64, activation="relu")
x = tmp(x)

함수형 API의 함수

```
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

x_train = x_train.reshape(60000, 784).astype("float32") / 255
x_test = x_test.reshape(10000, 784).astype("float32") / 255

model.compile(
    loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    optimizer=keras.optimizers.RMSprop(),
    metrics=["accuracy"],
)

history = model.fit(x_train, y_train, batch_size=64, epochs=2, validation_split=0.2)
test_scores = model.evaluate(x_test, y_test, verbose=2)
```

기본형 오토인코더

- 오토인코더(auto encoder)는 입력과 동일한 출력을 만드는 것을 목적으로 하는 신경망이다.
- 오토인코더는 특징 학습, 차원 축소, 표현 학습 등에 많이 사용된다. 우리는 차원 축소(dimensionality reduction) 문제를 중점적으로 다루도록 하자.

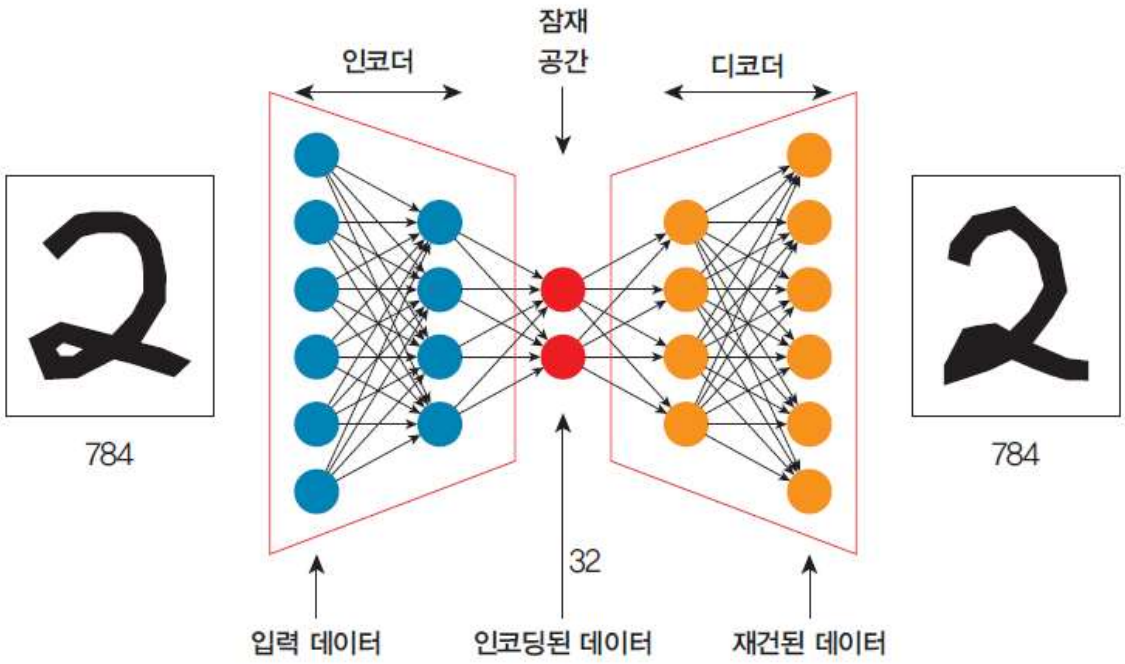
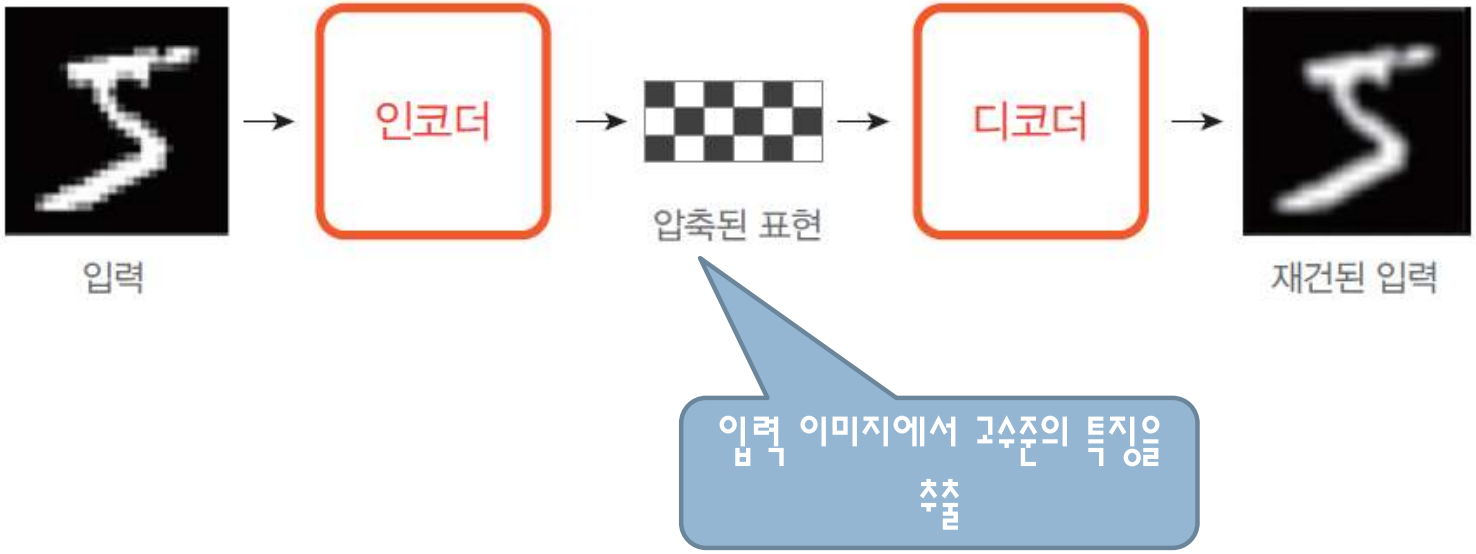


그림 14-4 오토인코더의 구조

오토인코더의 요소

- 인코더(encoder): 입력을 잠재 표현으로 인코딩한다.
- 디코더(decoder): 잠재 표현을 풀어서 입력을 복원한다.
- 손실 함수: 입력 이미지와 출력 이미지의 MSE를 사용한다.



예제: 필기체 숫자를 압축하는 오토인코더

```
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf

encoding_dim = 32                          # 32 픽셀로 압출

# 함수형 API로 신경망 구성
input_img = tf.keras.layers.Input(shape=(784,))
encoded = tf.keras.layers.Dense(encoding_dim, activation='relu')(input_img)
decoded = tf.keras.layers.Dense(784, activation='sigmoid')(encoded)
autoencoder = tf.keras.models.Model(input_img, decoded)

autoencoder.compile(optimizer='adam', loss=tf.keras.losses.MeanSquaredError())
```

손실 함수로 MSE 사용, 픽셀 간의 차이를 계산한다.

예제: 필기체 숫자를 압축하는 오토인코더

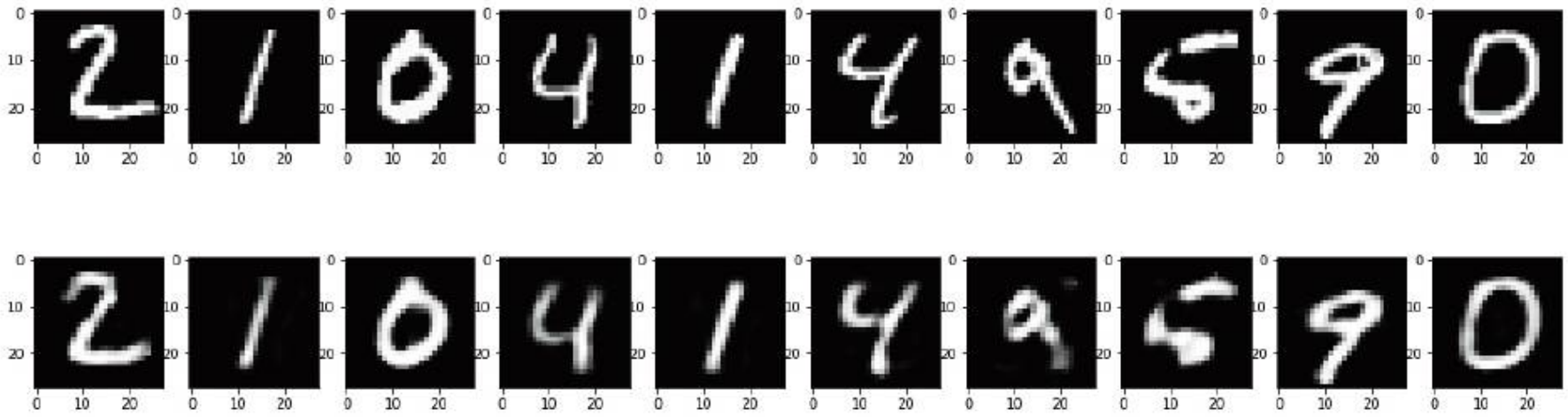
```
mnist = tf.keras.datasets.mnist
(x_train, _), (x_test, _) = mnist.load_data()

x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))

autoencoder.fit(x_train, x_train,
                epochs=50,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))
```

1차원으로 평탄화

예제: 필기체 숫자를 압축하는 오토인코더

```
decoded_imgs = autoencoder.predict(x_test)

n = 10
plt.figure(figsize=(20, 6))
for i in range(1, n + 1):
    ax = plt.subplot(2, n, i)
    plt.imshow(x_test[i].reshape(28, 28), cmap='gray')

    ax = plt.subplot(2, n, i + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28), cmap='gray')
plt.show()
```

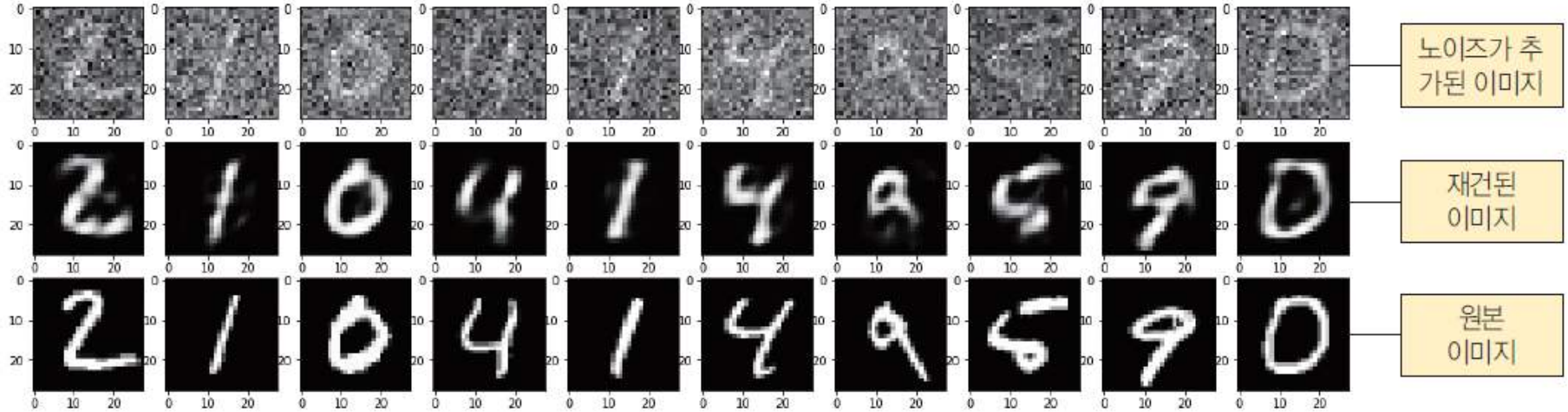
실행 결과



노이즈 제거 오토인코더

- 오토인코더는 노이즈(noise)가 있는 이미지에서 노이즈를 제거하는 용도로도 사용할 수 있다.

노이즈 제거 오토인코더



예제: 잡음이 들어간 필기체 숫자를 복원하는 오토인코더

```
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf

encoding_dim = 32
input_img = tf.keras.layers.Input(shape=(784,))
encoded = tf.keras.layers.Dense(encoding_dim, activation='relu')(input_img)
decoded = tf.keras.layers.Dense(784, activation='sigmoid')(encoded)
autoencoder = tf.keras.models.Model(input_img, decoded)
```

함수형 API를 이용하여 신경망을 구축

예제: 잡음이 들어간 필기체 숫자를 복원하는 오토인코더

```
mnist = tf.keras.datasets.mnist

(x_train, _), (x_test, _) = mnist.load_data()
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))
```

MNIST 데이터 처리

예제: 잡음이 들어간 필기체 숫자를 복원하는 오토인코더

```
noise_factor = 0.55

original_train = x_train
original_test = x_test
noise_train = np.random.normal(0, 1, original_train.shape)
noise_test = np.random.normal(0, 1, original_test.shape)
noisy_train = original_train + noise_factor * noise_train
noisy_test = original_test + noise_factor * noise_test
```

넘파이 연산을 이용하여 영상에 노이즈를 추가

예제: 잡음이 들어간 필기체 숫자를 복원하는 오토인코더

```
autoencoder.compile(optimizer='adam', loss='mse')

autoencoder.fit(noisy_train, original_train,
                epochs=50,
                batch_size=256,
                shuffle=True,
                validation_data=(noisy_test, original_test))

denoised_images = autoencoder.predict(noisy_test)
```

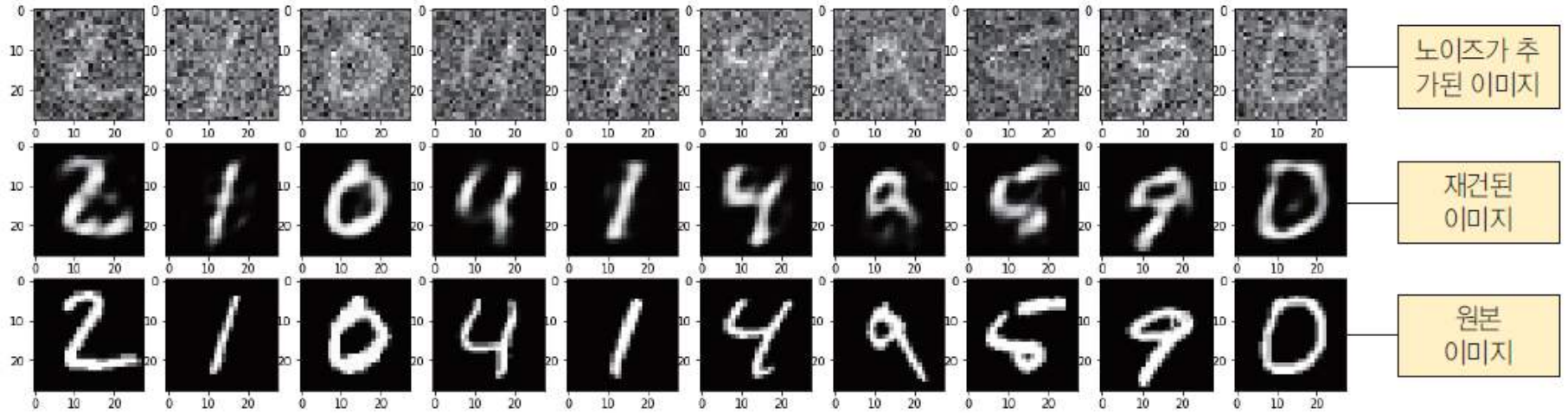
학습 및 예측

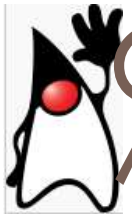
예제: 잡음이 들어간 필기체 숫자를 복원하는 오토인코더

```
n = 10
plt.figure(figsize=(20, 6))
for i in range(1, n + 1):
    ax = plt.subplot(3, n, i)
    plt.imshow(noisy_test[i].reshape(28, 28), cmap='gray')
    plt.gray()

    ax = plt.subplot(3, n, i + n)
    plt.imshow(denoised_images[i].reshape(28, 28), cmap='gray')
    plt.gray()

    ax = plt.subplot(3, n, i + 2*n)
    plt.imshow(original_test[i].reshape(28, 28), cmap='gray')
    plt.gray()
plt.show()
```

학습 및 예측

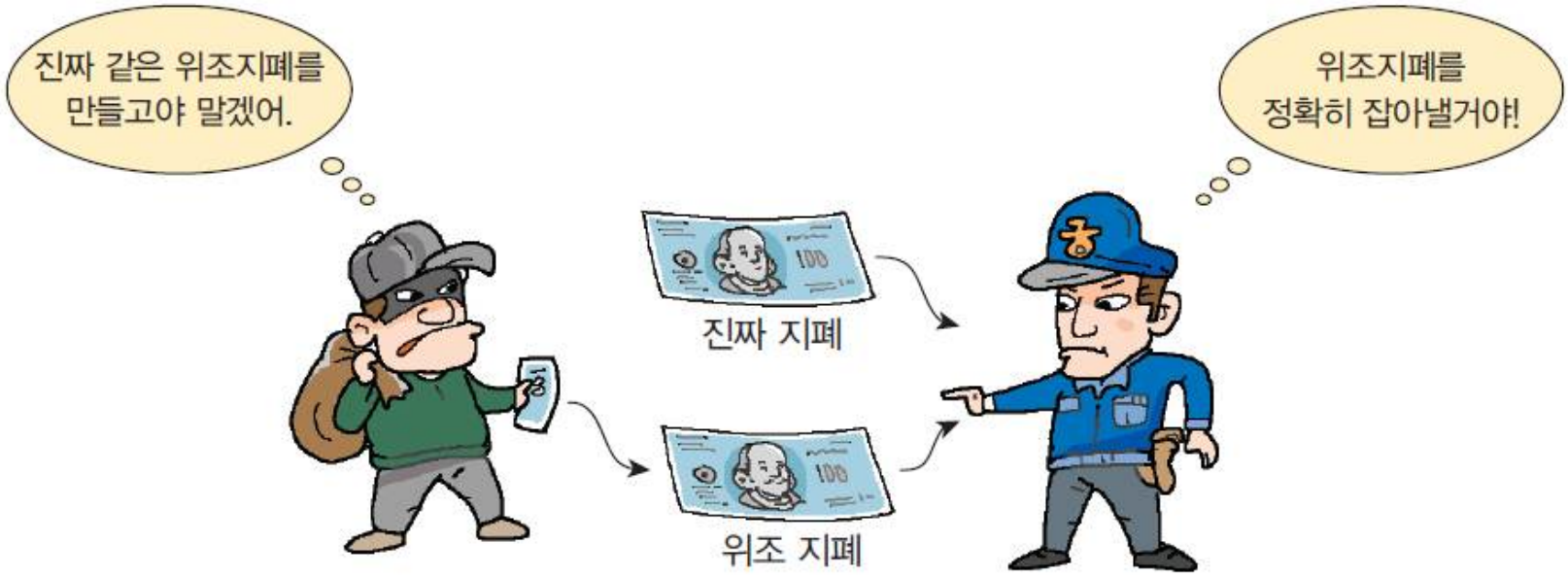
실행 결과





GAN(Generative adversarial network, 생성적 적대 신경망)

- Goodfellow(2014) 등이 설계한 신경망 모델이다.
- 이 모델에서는 생성자 신경망과 판별자 신경망이 서로 적대적으로 경쟁하면서, 훈련을 통하여 자신의 작업을 점점 정교하게 수행한다.

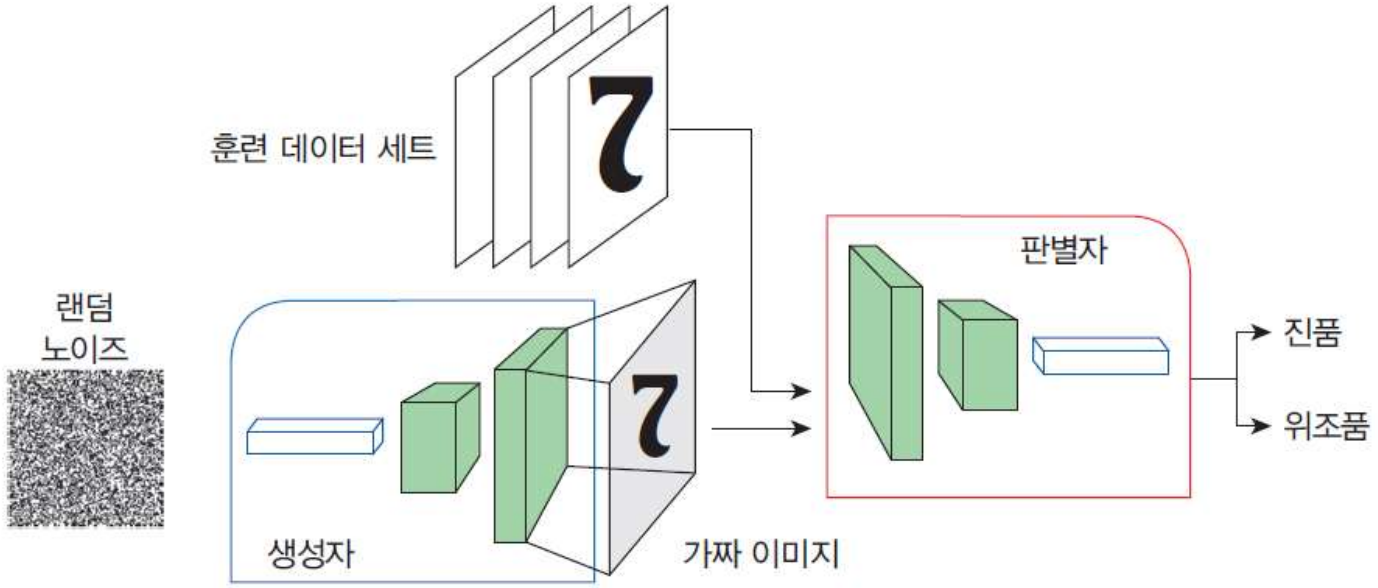


GAN을 이용한 결과물

출처: ICLR 2018 "PROGRESSIVE GROWING OF GANS FOR IMPROVED QUALITY, STABILITY, AND VARIATION", 1024 × 1024 images generated using the CELEBA-HQ dataset

GAN의 구조

- 생성자(generator): 가짜 데이터를 생성하는 것을 학습한다. 생성된 데이터는 판별자를 위한 학습 예제가 된다.
- 판별자(discriminator): 생성자의 가짜 데이터를 진짜 데이터와 구분하는 방법을 학습한다. 판별자는 생성자가 유사하지 않은 데이터를 생성하면 불이익을 준다.



GAN 훈련 과정

- 판별자 훈련과 생성자 훈련이 번갈아 가며 수행된다.

- GAN 훈련이 시작될 때 생성자는 아직 무엇을 만들어야 하는지 전혀 알지 못한다. 따라서 입력으로 임의의 노이즈가 공급되며, 생성자는 출력으로 임의의 노이즈 이미지를 생성한다.

- 이러한 저품질 가짜 이미지는 진짜 이미지와 극명하게 대조되므로, 판별자는 처음에 진짜와 가짜를 판별하는 데 전혀 문제가 없다. 그러나 생성자가 훈련되면서 진짜 이미지의 일부 구조를 복제하는 방법을 점차적으로 학습한다.

사진을 생성하는 GAN

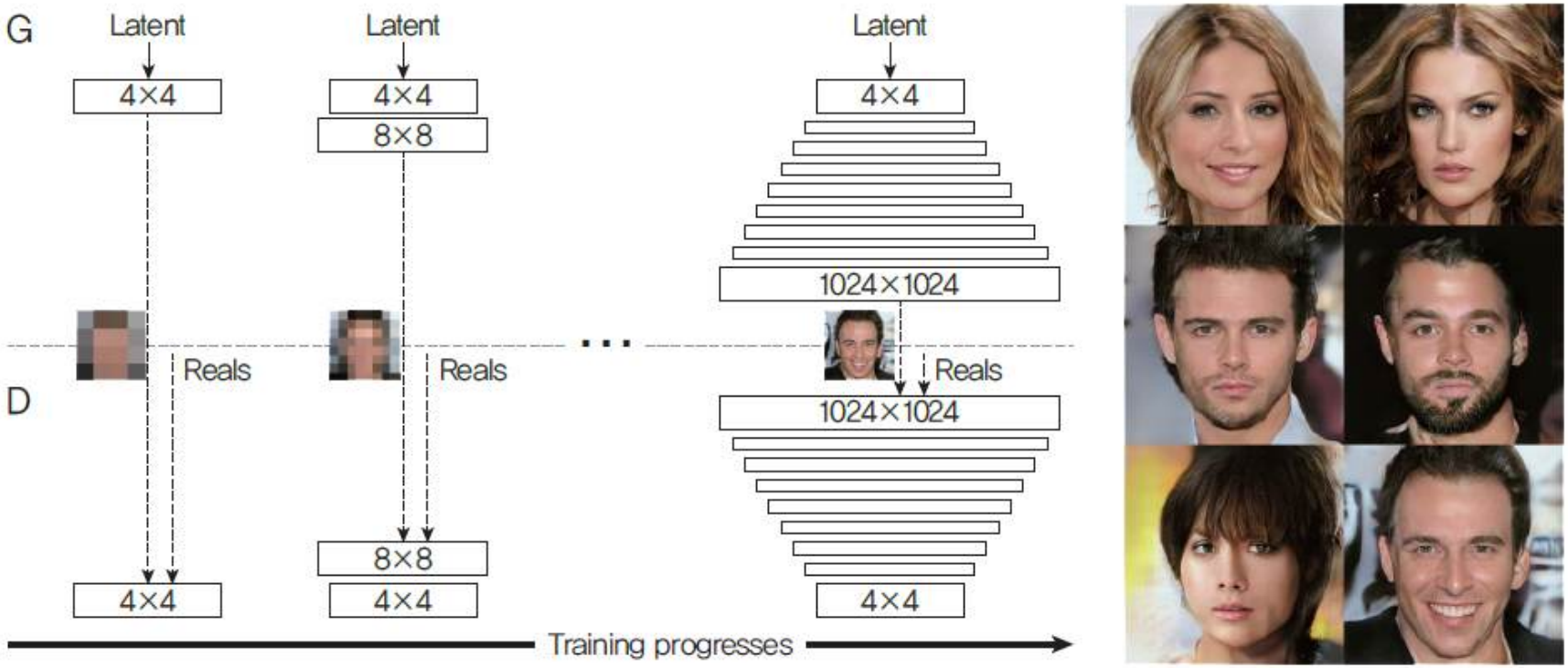


그림 14-6 사진을 생성하는 GAN

(출처: ICLR 2018 "PROGRESSIVE GROWING OF GANS FOR IMPROVED QUALITY, STABILITY, AND VARIATION", 1024 × 1024 images generated using the CELEBA-HQ dataset)

판별자 훈련

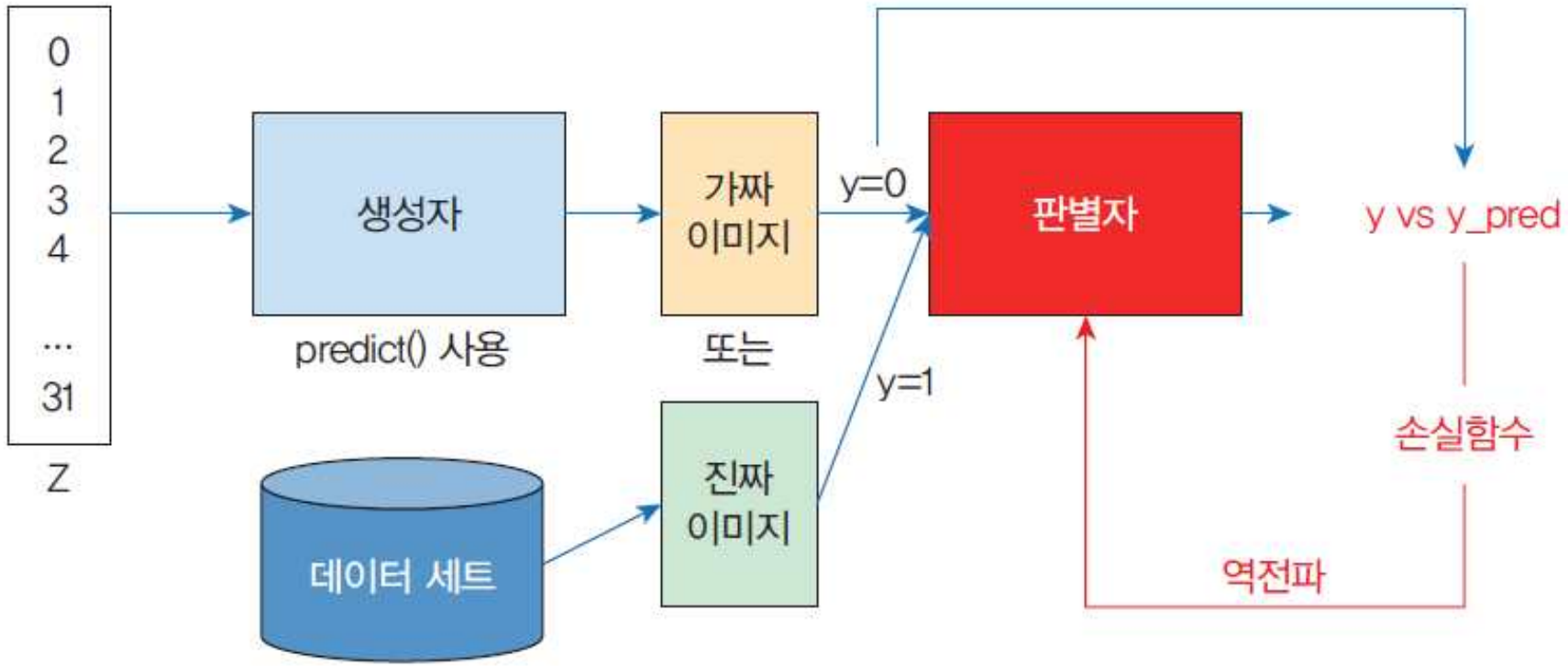


그림 14-7 판별자 훈련 개요

생성자 훈련

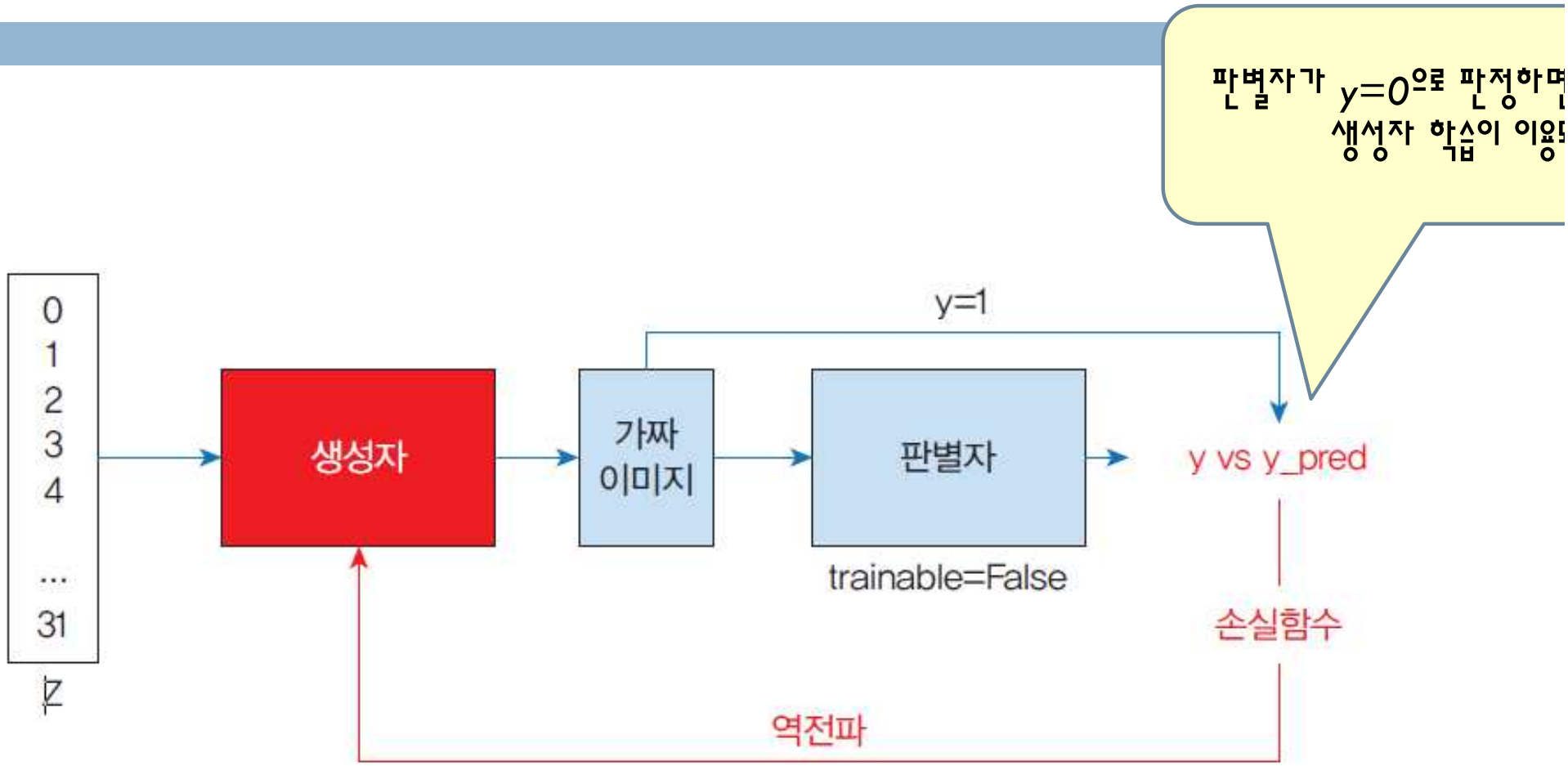
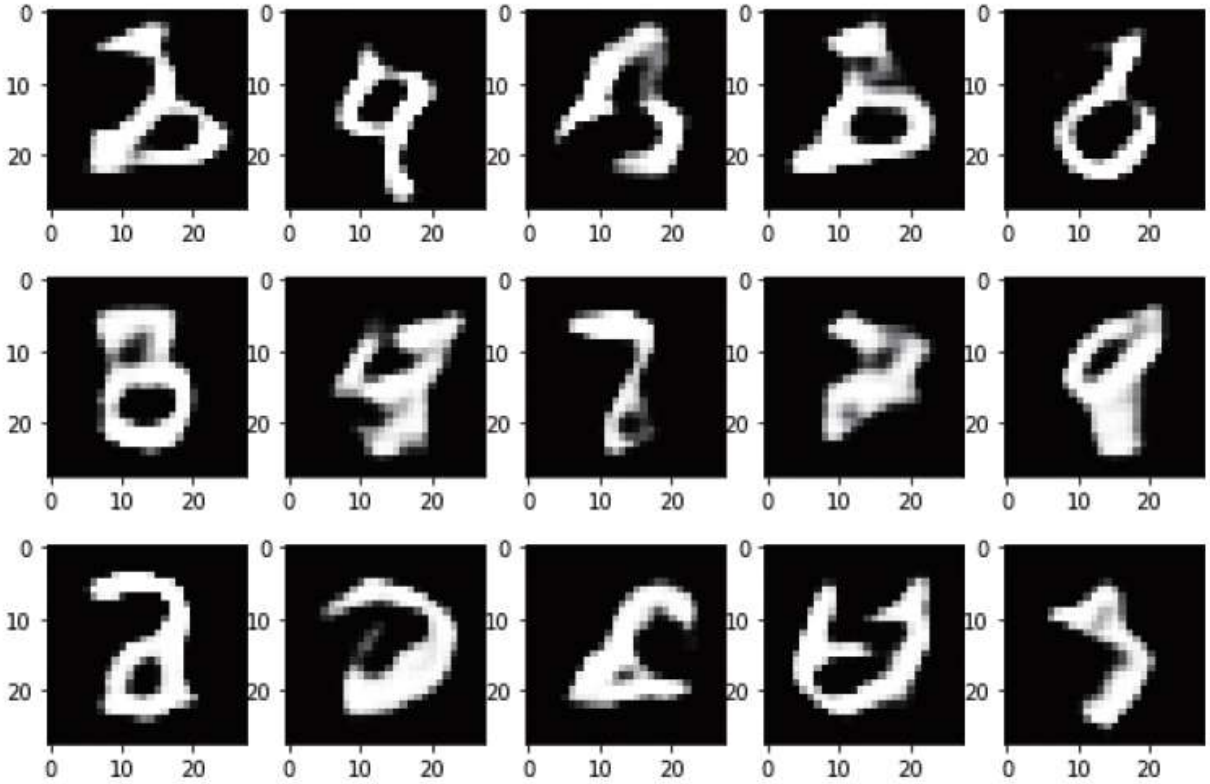


그림 14-8 생성자 훈련 개요

예제: GAN으로 숫자 이미지 생성

- 가장 전형적인 예제는 MNIST 필기체 숫자 이미지를 가지고 가상적인 숫자 이미지를 생성해보는 것이다. -> 실제 이미지 생성은 학습 시간이 너무 오래 걸린다.



필요한 모듈을 포함시킨다.

```
import numpy as np
import tensorflow as tf
from matplotlib import pyplot as plt

# 학습 데이터와 테스트 데이터 분리
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

# 이미지를 [0, 1] 범위로 스케일링
x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255
```

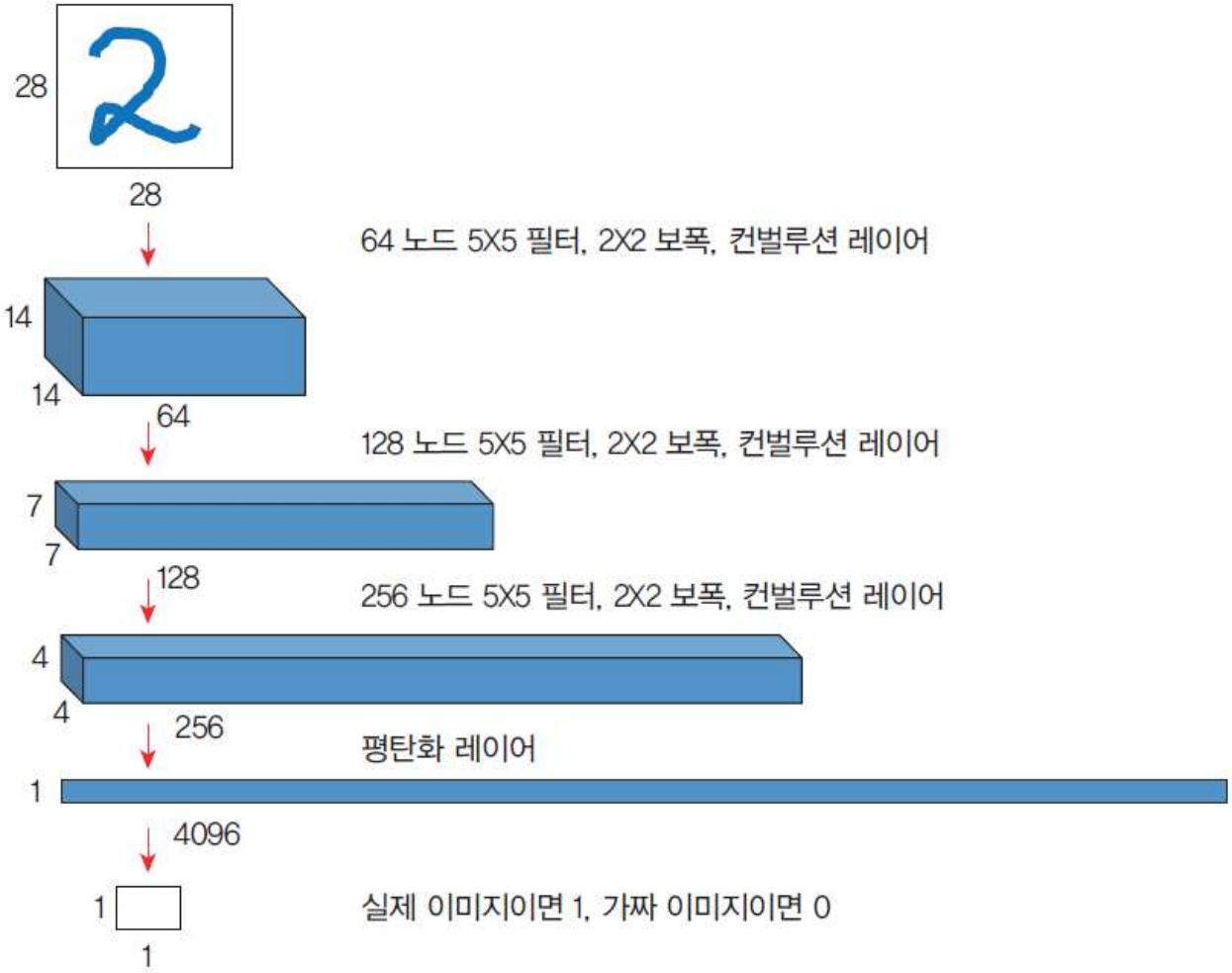
입력 데이터 평탄화

```
BATCH_SIZE=128
EPOCHS=2000
Z_DIMENSIONS=32
data = np.reshape(x_train, (x_train.shape[0], 28, 28, 1))
```

판별자 신경망 구축하기

```
def make_discriminator():
    model = tf.keras.Sequential()
    model.add(tf.keras.layers.Conv2D(64, (5, 5), strides=(2, 2), padding='same',
                                     activation='relu', input_shape=[28, 28, 1]))
    model.add(tf.keras.layers.Dropout(0.4))

    model.add(tf.keras.layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same',
                                     activation='relu'))
    model.add(tf.keras.layers.Dropout(0.4))

    model.add(tf.keras.layers.Conv2D(256, (5, 5), strides=(2, 2), padding='same',
                                     activation='relu'))
    model.add(tf.keras.layers.Dropout(0.4))
    model.add(tf.keras.layers.Flatten())
    model.add(tf.keras.layers.Dense(1, activation='sigmoid'))

    return model
```

판별자


28
28
64 노드 5X5 필터, 2X2 보폭, 컨벌루션 레이어
14
14
64
128 노드 5X5 필터, 2X2 보폭, 컨벌루션 레이어
7
7
128
256 노드 5X5 필터, 2X2 보폭, 컨벌루션 레이어
4
4
256
평탄화 레이어
1
4096
1
실제 이미지이면 1, 가짜 이미지이면 0
1

판별자 신경망 구축하기

```
discriminator = make_discriminator()

discriminator.compile(loss='binary_crossentropy',
                      optimizer=tf.keras.optimizers.Adam(lr=0.0004),
                      metrics=['accuracy'])
```

생성자 신경망 구축하기

- 생성자는 일반적인 컨볼루션 레이어의 반대 기능을 수행한다. 일반적인 컨벌루션 레이어는 입력 이미지로부터 특징을 추출하여서 활성화 맵을 출력하지만, GAN 생성자는 활성화 맵을 가져와서 이미지를 구성한다

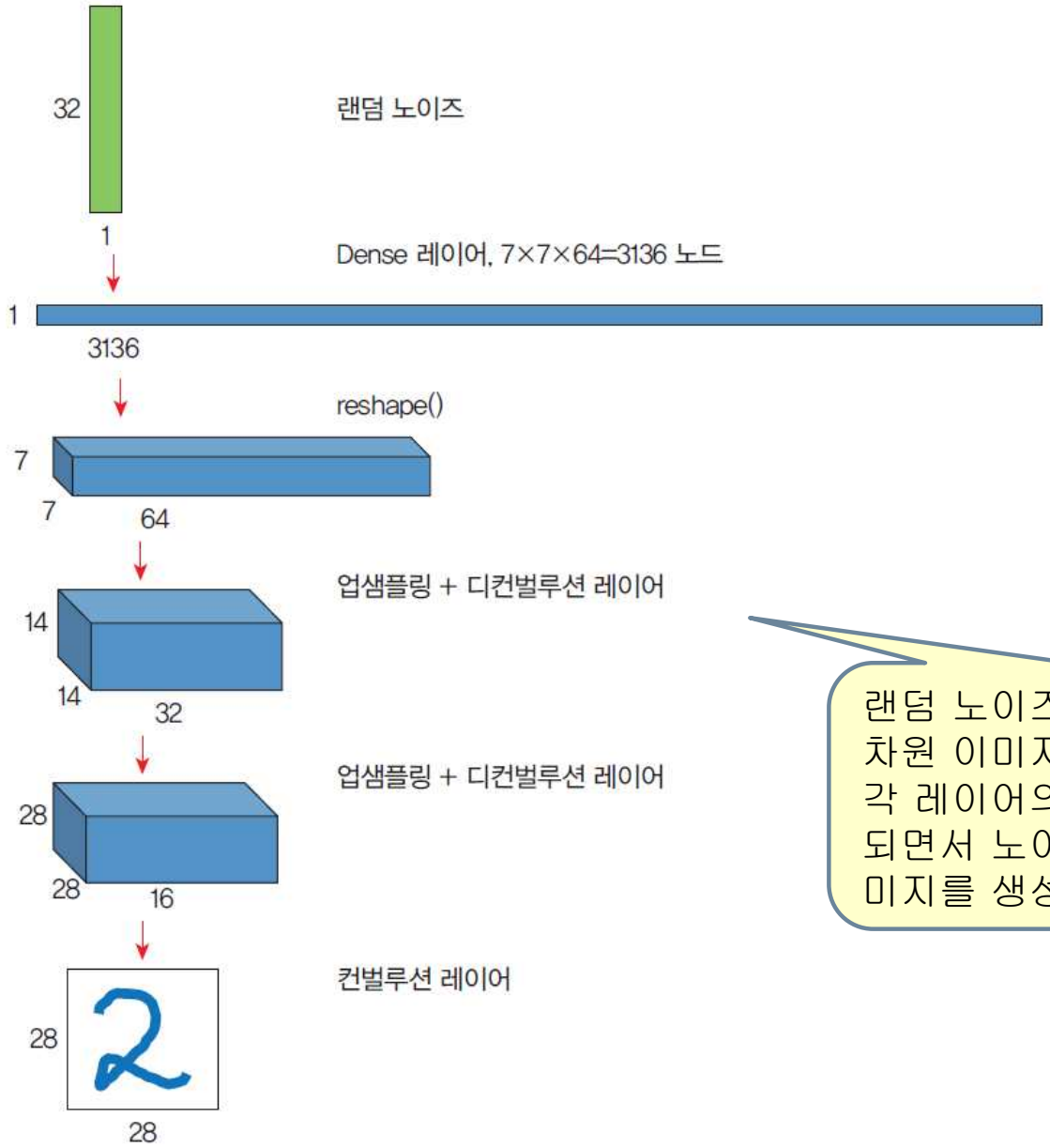
생성자 신경망 구축하기

```
def make_generator():
    model = tf.keras.Sequential()

    model.add(tf.keras.layers.Dense(7*7*64, input_shape=(Z_DIMENSIONS,)))
    model.add(tf.keras.layers.BatchNormalization(momentum=0.9))
    model.add(tf.keras.layers.LeakyReLU())
    model.add(tf.keras.layers.Reshape((7, 7, 64)))
    model.add(tf.keras.layers.Dropout(0.4))

    model.add(tf.keras.layers.UpSampling2D())
    model.add(tf.keras.layers.Conv2DTranspose(32,
                        kernel_size=5, padding='same',
                        activation=None,))
    model.add(tf.keras.layers.BatchNormalization(momentum=0.9))
    model.add(tf.keras.layers.LeakyReLU())
```

생성자 신경망 구축하기

```
model.add(tf.keras.layers.UpSampling2D())
model.add(tf.keras.layers.Conv2DTranspose(16,
                    kernel_size=5, padding='same',
                    activation=None,))
model.add(tf.keras.layers.BatchNormalization(momentum=0.9))
model.add(tf.keras.layers.LeakyReLU())

model.add(tf.keras.layers.Conv2D(1, kernel_size=5, padding='same',
            activation='sigmoid'))
return model
```

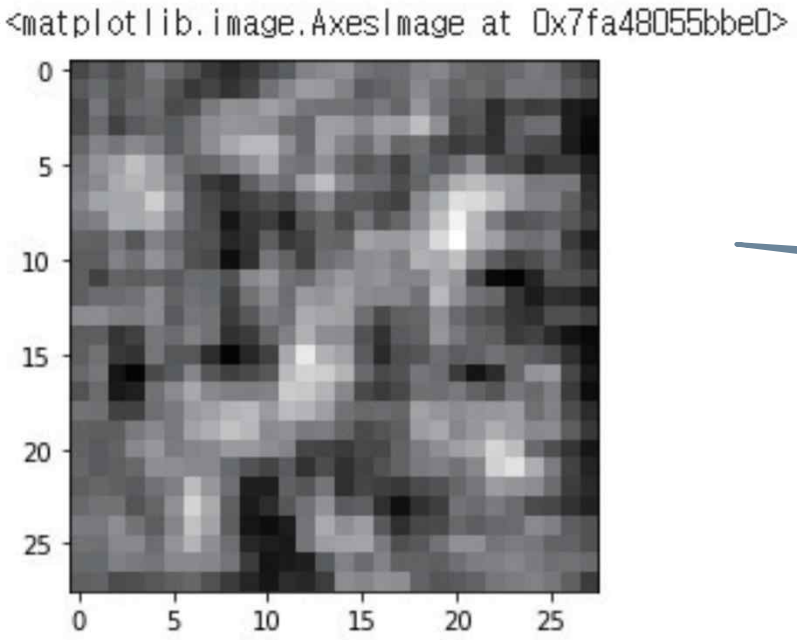
생성자 신경망 구축하기



랜덤 노이즈가 입력되어서 2차원 이미지로 업샘플링된다. 각 레이어의 가중치들이 학습되면서 노이즈에 해당하는 이미지를 생성할 수 있다.

생성자 생성 및 테스트

```
generator = make_generator()

noise = tf.random.normal([1, Z_DIMENSIONS])
generated_image = generator(noise, training=False)
plt.imshow(generated_image[0, :, :, 0], cmap='gray')
```



지금은 완전한 노이즈만 출력된다.

생성적 적대 신경망 구축

```
z = tf.keras.layers.Input(shape=(Z_DIMENSIONS,))
fake_image = generator(z)
discriminator.trainable = False
prediction = discriminator(fake_image)
gan_model = tf.keras.models.Model(z, prediction)

gan_model.compile(loss='binary_crossentropy',
                  optimizer=tf.keras.optimizers.Adam(lr=0.0004),
                  metrics=['accuracy'])
```

GAN의 학습

```
def train_gan():
    for i in range(EPOCHS):
        real_images = np.reshape(
            data[np.random.choice(data.shape[0],
                                  BATCH_SIZE,
                                  replace=False)], (BATCH_SIZE,28,28,1))
        fake_images = generator.predict(
          np.random.uniform(-1.0, 1.0,
                            size=[BATCH_SIZE, Z_DIMENSIONS]))

        x = np.concatenate((real_images,fake_images))

        y = np.ones([2*BATCH_SIZE,1])
        y[BATCH_SIZE:,:] = 0

        discriminator.train_on_batch(x, y)
```

진짜 이미지와 가짜 이미지를 붙인다.

정답 레이블을 생성한다.

판별자를 훈련한다.

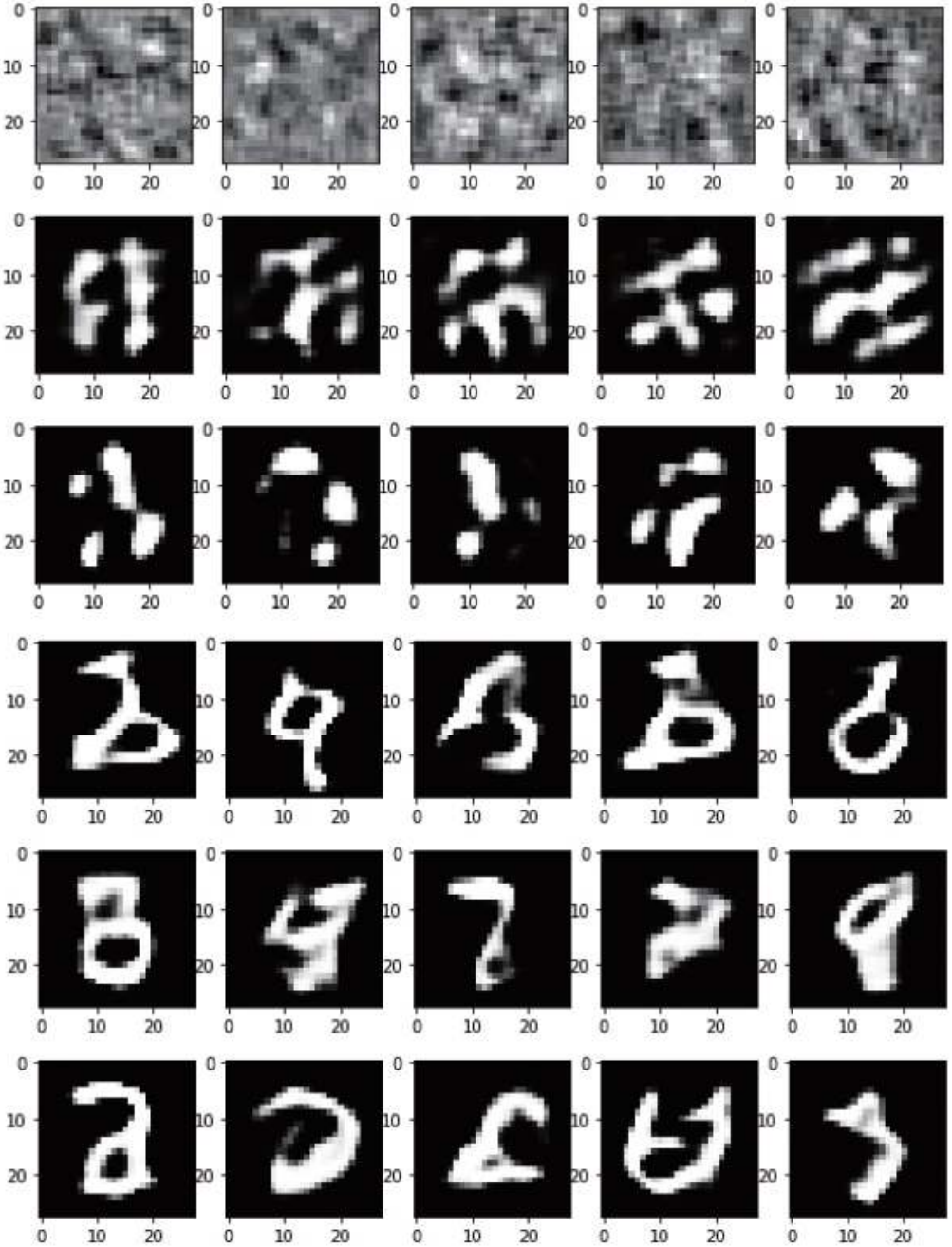
GAN의 학습

```
        noise = np.random.uniform(-1.0, 1.0, size=[BATCH_SIZE, Z_DIMENSIONS])
        y = np.ones([BATCH_SIZE,1])

        gan_model.train_on_batch(noise, y)

        if i%100 == 0:
            noise = np.random.uniform(-1.0, 1.0,
                                      size=[5, Z_DIMENSIONS])
            generated_image = generator.predict(noise)
            plt.figure(figsize=(10,10))
            for i in range(generated_image.shape[0]):
                plt.subplot(1, 5, i+1)
                plt.imshow(generated_image[i, :, :, 0],
                           cmap='gray')
            plt.show()
train_gan()
```

노이즈를 입력하여서 생성자를 훈련한다.

중간 결과를 출력한다.

실행 결과



Summary

- 생성 모델(generative model)은 훈련 데이터의 규칙성 또는 패턴을 자동으로 발견하고 학습하여, 훈련 데이터의 확률 분포와 유사하지만, 새로운 샘플을 생성할 수 있는 신경망이다.
- 오토인코더(auto encoder)는 입력과 동일한 출력을 만드는 것을 목적으로 하는 신경망이다. 오토인코더는 특징 학습, 차원 축소, 표현 학습 등에 많이 사용된다.
- GAN(Generative adversarial network, 생성적 적대 신경망)은 Goodfellow(2014) 등이 설계한 신경망 모델이다. 이 모델에서는 생성자 신경망과 판별자 신경망이 서로 적대적으로 경쟁하면서, 훈련을 통하여 자신의 작업을 점점 정교하게 수행한다.

Q & A