chap. 16. 실습과제

학번	321848101	이름	하승원

[실습과제 1]

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

Source code :

```
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
from tqdm import tqdm
import matplotlib.pyplot as plt

K.clear_session()

# Load data
img = image.load_img('apple.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


# Discriminator model

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()


# Generator model

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())

generator.add(Conv2D(filters=128,       kernel_size=5,       padding='same',
```

```
strides=1))
generator.add(BatchNormalization(momentum=0.9))
generator.add(Activation('relu'))
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()


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


# Combined model (GAN)
model = Sequential()
model.add(generator)
model.add(discriminator)
model.compile(optimizer=RMSprop(learning_rate=0.0004),
loss='binary_crossentropy', metrics=['accuracy'])


# Training functions
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]
```

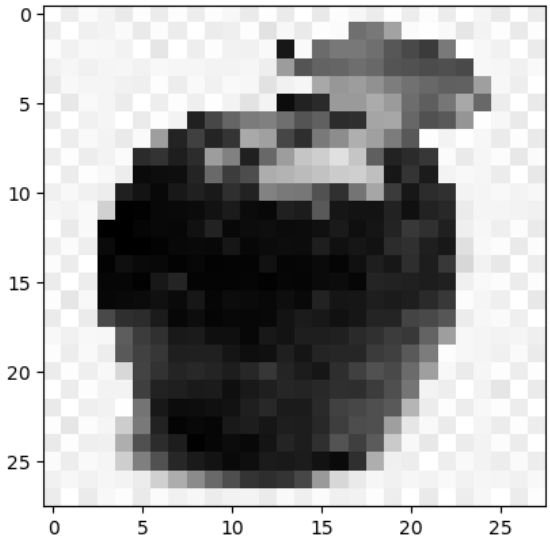
```
    discriminator.fit(true_imgs, valid, verbose=0)
    noise = np.random.normal(0, 1, (batch_size, 100))
    gen_imgs = generator.predict(noise)
    discriminator.fit(gen_imgs, fake, verbose=0)

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)

# Training loop
for epoch in tqdm(range(300)):
    train_discriminator(Xtrain, 64)
    train_generator(64)

# Display results
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))
gen_result = generator.predict(random_noise)
gen_img = array_to_img(gen_result[0])
plt.imshow(gen_img, cmap='gray')
plt.show()
```

실행화면 캡쳐:

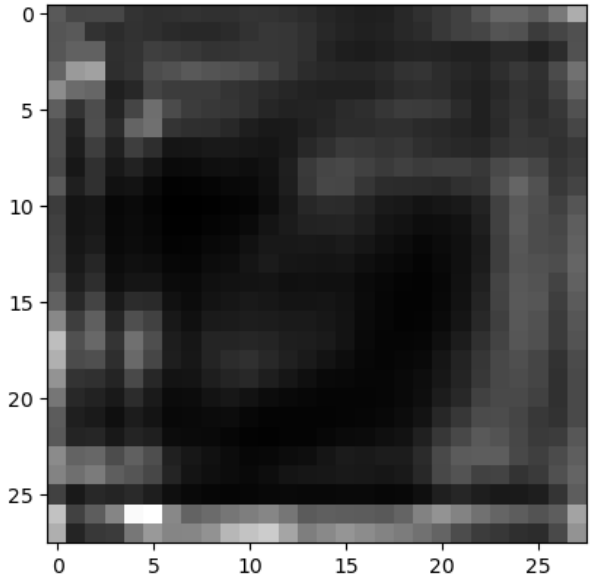
(결과가 긴 경우는 결과의 시작 부분중 일부만 캡쳐)

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_1 (Dense)             (None, 3136)              316736

 batch_normalization (Batch  (None, 3136)              12544
 Normalization)

 activation (Activation)     (None, 3136)              0

 reshape (Reshape)           (None, 7, 7, 64)          0

 up_sampling2d (UpSampling2  (None, 14, 14, 64)        0
 D)

 conv2d_3 (Conv2D)           (None, 14, 14, 128)       204928

 batch_normalization_1 (Bat  (None, 14, 14, 128)       512
 chNormalization)

 activation_1 (Activation)   (None, 14, 14, 128)       0

 up_sampling2d_1 (UpSamplin  (None, 28, 28, 128)       0
 g2D)

 conv2d_4 (Conv2D)           (None, 28, 28, 64)        204864

 batch_normalization_2 (Bat  (None, 28, 28, 64)        256
 chNormalization)
...
Total params: 844161 (3.22 MB)
Trainable params: 837377 (3.19 MB)
Non-trainable params: 6784 (26.50 KB)
```

-진짜



-가짜



[실습과제 2]

2. 주어진 코드는 28x28 이미지 학습에 맞도록 숫자들이 지정되어 있다. 32x32 이미지에 대해서도 코드가 작동하도록 코드를 수정하여 테스트하시오

Source code :

```
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
```

```
from tqdm import tqdm

import matplotlib.pyplot as plt


K.clear_session()


# Load data

img = image.load_img('apple.png', target_size=(32, 32))

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


# Discriminator model

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()


# Generator model
```

```
gen_dense_size = (8, 8, 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())

generator.add(Conv2D(filters=128,       kernel_size=5,       padding='same',
strides=1))

generator.add(BatchNormalization(momentum=0.9))

generator.add(Activation('relu'))

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()


# Compile discriminator

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


# Combined model (GAN)

model = Sequential()

model.add(generator)

model.add(discriminator)

model.compile(optimizer=RMSprop(learning_rate=0.0004),
```

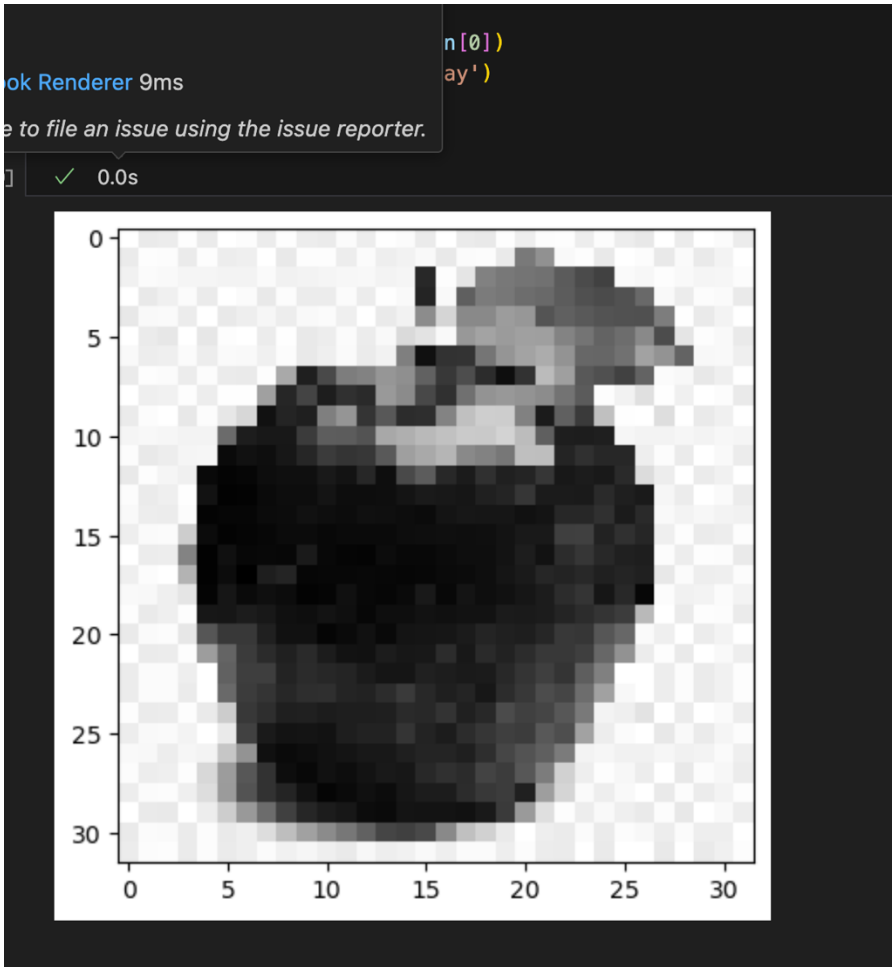
```
loss='binary_crossentropy', metrics=['accuracy'])


# Training functions
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)

# Training loop
for epoch in tqdm(range(300)):
    train_discriminator(Xtrain, 64)
    train_generator(64)

# Display results
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))
```

```
gen_result = generator.predict(random_noise)

gen_img = array_to_img(gen_result[0])

plt.imshow(gen_img, cmap='gray')

plt.show()
```

실행화면 캡쳐:

(결과가 긴 경우는 결과의 시작 부분중 일부만 캡쳐)

```
2/2 [==============================] - 1s 413ms/step - loss: 3.4062 - accuracy: 0.0312
 81%|████████  | 243/300 [04:23<01:04,  1.13s/it]
2/2 [==============================] - 0s 140ms/step
2/2 [==============================] - 1s 356ms/step - loss: 1.6335 - accuracy: 0.4531
 81%|████████  | 244/300 [04:24<01:02,  1.12s/it]
2/2 [==============================] - 0s 102ms/step
2/2 [==============================] - 1s 374ms/step - loss: 3.6169 - accuracy: 0.0000e+0
 82%|████████  | 245/300 [04:25<01:00,  1.10s/it]
2/2 [==============================] - 0s 104ms/step
2/2 [==============================] - 1s 369ms/step - loss: 1.3728 - accuracy: 0.4688
 82%|████████  | 246/300 [04:26<00:58,  1.09s/it]
2/2 [==============================] - 0s 109ms/step
2/2 [==============================] - 1s 384ms/step - loss: 2.9902 - accuracy: 0.0000e+0
 82%|████████  | 247/300 [04:27<00:57,  1.09s/it]
2/2 [==============================] - 0s 104ms/step
2/2 [==============================] - 1s 381ms/step - loss: 2.0141 - accuracy: 0.0625
 83%|████████  | 248/300 [04:28<00:57,  1.10s/it]
```


n[0])
ay')
ok Renderer 9ms
e to file an issue using the issue reporter.
0.0s
0
5
10
15
20
25
30
0
5
10
15
20
25
30