```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Reshape
from keras.layers.core import Activation
from keras.layers.normalization import BatchNormalization
from keras.layers.convolutional import UpSampling2D
from keras.layers.convolutional import Convolution2D, MaxPooling2D
from keras.layers.core import Flatten
from keras.optimizers import SGD
from keras.datasets import mnist
import keras.backend as K
import numpy as np
from PIL import Image
import argparse
import math
import matplotlib.pyplot as plt
import os

image_dim_order = K.image_dim_ordering()


def generator_model():
    # bulid the generator model, it is a model made up of UpSample and Convolution
    model = Sequential()

    model.add(Dense(input_dim=100, output_dim=1024))        # 100개의 입력 벡터 설정
    model.add(Activation('tanh'))

    model.add(Dense(128*7*7))
    model.add(BatchNormalization())
    model.add(Activation('tanh'))

    if image_dim_order == 'th':
        model.add(Reshape((128, 7, 7), input_shape=(128*7*7,)))
    else:
        model.add(Reshape((7, 7, 128), input_shape=(128*7*7,)))

    model.add(UpSampling2D(size=(2, 2)))                    # 업샘플링 팩터값 rowxcol: ➲
      2x2
    model.add(Convolution2D(64, 5, 5, border_mode='same'))  # 컨벌루션 오퍼레이터, 커널➲
      크기: 5x5 필터 수:64
    model.add(Activation('tanh'))

    model.add(UpSampling2D(size=(2, 2)))
    model.add(Convolution2D(1, 5, 5, border_mode='same'))
    model.add(Activation('tanh'))
    return model


def discriminator_model():
    # build the discriminator model, it is one common convolutional neural network
    model = Sequential()
    # different backend has different image dim order, so we need to judge first.
```

```
    if image_dim_order == 'th':
        input_shape = (1,28,28)
    else:
        input_shape = (28,28,1)

    model.add(Convolution2D(
                        64, 5, 5,
                        border_mode='same',
                        input_shape=input_shape))
    model.add(Activation('tanh'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Convolution2D(128, 5, 5))
    model.add(Activation('tanh'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Flatten())
    model.add(Dense(1024))
    model.add(Activation('tanh'))
    model.add(Dense(1))
    model.add(Activation('sigmoid'))
    return model


def generator_containing_discriminator(generator, discriminator):
    model = Sequential()
    model.add(generator)
    # when train the generator, the discriminator model cannot be trained
    discriminator.trainable = False     # Discriminator는 학습 안되게 설정
    model.add(discriminator)
    return model


def combine_images(generated_images):
    # combine one batch of images to one big image
    num = generated_images.shape[0]
    width = int(math.sqrt(num))
    height = int(math.ceil(float(num)/width))

    if image_dim_order == 'th':
        shape = generated_images.shape[2:]
    else:
        shape = generated_images.shape[1:3]

    image = np.zeros((height*shape[0], width*shape[1]),
                     dtype=generated_images.dtype)
    for index, img in enumerate(generated_images):
        i = int(index/width)
        j = index % width
        if image_dim_order == 'th':
            image[i*shape[0]:(i+1)*shape[0], j*shape[1]:(j+1)*shape[1]] = ₩
            img[0, :, :]
        else:
            image[i*shape[0]:(i+1)*shape[0], j*shape[1]:(j+1)*shape[1]] = ₩
            img[:, :, 0]
```

```
    return image


def train(BATCH_SIZE, epoch_num):
    (X_train, y_train), (X_test, y_test) = mnist.load_data()    # mnist 숫자 데이터 로딩
    X_train = (X_train.astype(np.float32) - 127.5)/127.5        # 입력값 -1 ~ +1 범위로 정규화

    if image_dim_order == 'th':                                 # 백엔드 엔진이 Theano (th)인 경
        X_train = X_train.reshape((X_train.shape[0], 1) + X_train.shape[1:])
    else:                                                       # 백엔드엔진이 Tensorflow (tf)인 경우
        X_train = X_train.reshape((X_train.shape[0], ) + X_train.shape[1:] + (1,))

    discriminator = discriminator_model()                       # Discriminator 모델 생성
    generator = generator_model()                               # Generator 모델 생성

    discriminator_on_generator = ₩
        generator_containing_discriminator(generator, discriminator)    # Discriminator 를 포함하는 Generator 모델 생성

    d_optim = SGD(lr=0.0005, momentum=0.9, nesterov=True)       # 옵티마이저 설정
    g_optim = SGD(lr=0.0005, momentum=0.9, nesterov=True)

    generator.compile(loss='binary_crossentropy', optimizer="SGD")  # Generator 컴파일
    discriminator_on_generator.compile(                         # Discriminator on Generator 컴파일
        loss='binary_crossentropy', optimizer=g_optim)

    discriminator.trainable = True
    discriminator.compile(loss='binary_crossentropy', optimizer=d_optim)    # Discriminator 컴파일
    noise = np.zeros((BATCH_SIZE, 100))

    false_negative_list = []
    false_positive_list = []
    total_accuracy_list = []

    for epoch in range(epoch_num):
        print('~'*70)
        print("Epoch is", epoch)
        print("Number of batches", int(X_train.shape[0]/BATCH_SIZE))

        batches_num = int(X_train.shape[0]/BATCH_SIZE)          # 468 = 60,000 (학습데이터 수) / 128 (배치사이즈)


        for index in range(batches_num):

            for i in range(BATCH_SIZE):                         # Batch 갯수만큼 -1 ~ 1
```

```
            사이값의, 크기 100인 Noise 난수 배열 발생
            noise[i, :] = np.random.uniform(-1, 1, 100)

        # note that when train the generator, the discriminator is not trainalbe
        discriminator.trainable = False                    # Discriminator는 학습 ➲
          안되게 설정

                                                           # 잡음(noise배열)      ➲
                  batch_size 갯수 전체를 차례로 사용하여 Generator 학습
                                                           # train_on_batch()는   ➲
                  keras에서 제공하는 배치 학습 함수
                                                           # Generator 먼저 학습  ➲
                  시킴
        g_loss = discriminator_on_generator.train_on_batch(
            noise, [1] * BATCH_SIZE)                       # target 값을 1로 줘서 ➲
              가짜이지만 진짜라고 전제하고 loss 계산, Generator 가중치만 학습

        discriminator.trainable = True                     # Discriminator 학습 가➲
          능하게 변경
                                                           # Generator loss 값 출 ➲
                  력
        print("epoch %d/%d batch %d/%d g_loss : %f" % (epoch+1, epoch_num,index,   ➲
          batches_num, g_loss))

        # leverage the generative images and natural images to train the           ➲
          discriminator
        for i in range(BATCH_SIZE):
            noise[i, :] = np.random.uniform(-1, 1, 100)

        image_batch = X_train[index*BATCH_SIZE:(index+1)*BATCH_SIZE]

        generated_images = generator.predict(noise, verbose=0)  # Discriminator 학 ➲
          습시키기 위하여, Generator로 Batch Size 수만큼 이미지 생성

        # save the generative images on the process of training regularly
        if index % 80 == 0:
            image = combine_images(generated_images)
            image = image*127.5+127.5                      # 출력용으로 0~255 값으➲
              로 변환
            Image.fromarray(image.astype(np.uint8)).save('images/'+
                str(epoch)+"_"+str(index)+".png")

        # create the trainset of the discriminator          # 원 학습데이터 이미지 ➲
          와 생성된 이미지를 결합, 학습용 데이터셋 X 생성
        X = np.concatenate((image_batch, generated_images))
                                                           # 원 학습데이터는      ➲
                  target 값을 1로, 생성된 이미지 데이터의 target은 0으로 설정
        y = [1] * BATCH_SIZE + [0] * BATCH_SIZE

        d_loss = discriminator.train_on_batch(X, y)        # 생성된 이미지와 학습 ➲
          데이터를 이용, Discriminator 학습
        print("epoch %d/%d batch %d/%d d_loss : %f" % (epoch+1, epoch_num, index,  ➲
          batches_num, d_loss))
```

```
                                                    # 매 20줄마다 Generator
                    와 Discriminator의 성능(accuracy) 평가
            if  index%20 == 0:
                # to test the generator  and discriminator's ability
                y_pred = discriminator.predict(X)
                false_positive = float(np.sum(y_pred[BATCH_SIZE:]>0.5))/BATCH_SIZE
                true_negative = float(np.sum(y_pred[BATCH_SIZE:]<0.5))/BATCH_SIZE
                true_positive = float(np.sum(y_pred[:BATCH_SIZE]>0.5))/BATCH_SIZE
                false_negative = float(np.sum(y_pred[:BATCH_SIZE]<0.5))/BATCH_SIZE

                total_accuracy = (true_positive + true_negative) /2

                print('*'*10)
                print("the ratio that generator deceive discrimator successly is :%f"%
                  false_positive)
                print("the ratio that discrimator fail to discrimate true/natural
                  mnist :%f"%false_negative)
                print("the totoal accuracy of the discriminator is :%f"%total_accuracy)

                false_negative_list.append(false_negative)
                false_positive_list.append(false_positive)
                total_accuracy_list.append(total_accuracy)

            if index % 20 == 0:
                generator.save_weights('generator', True)
                discriminator.save_weights('discriminator', True)

    # when training process is finished, plt the accuracy of generator and
      discriminator
    plt.figure()
    plt.title("the log of the training process")
    plt.plot(range(len(false_negative_list)), false_negative_list,'g-',label='D failed
      to discrimate natural image')
    plt.plot(range(len(false_positive_list)), false_positive_list,'r-', label='G
      deceive D successly')
    plt.plot(range(len(total_accuracy_list)), total_accuracy_list,'b-',label='total
      accuracy of D')
    plt.legend()
    plt.show()
    plt.savefig('log.png')

    # 이미지 생성 함수
    # Batch_Size 갯수 만큼 이미지 생성하여 generated_image.png에 저장
def generate(BATCH_SIZE, nice=False):

    generator = generator_model()
    generator.compile(loss='binary_crossentropy', optimizer="SGD")
    generator.load_weights('generator')

    # if nice is true, choose the 5% best generative images for storage, else the
      original generative images
    if nice:
```

```
        discriminator = discriminator_model()
        discriminator.compile(loss='binary_crossentropy', optimizer="SGD")
        discriminator.load_weights('discriminator')

        noise = np.zeros((BATCH_SIZE*20, 100))

        for i in range(BATCH_SIZE*20):
            noise[i, :] = np.random.uniform(-1, 1, 100)
        generated_images = generator.predict(noise, verbose=1)  # 20배 만큼 생성

        d_pret = discriminator.predict(generated_images, verbose=1)
        index = np.arange(0, BATCH_SIZE*20)
        index.resize((BATCH_SIZE*20, 1))
        pre_with_index = list(np.append(d_pret, index, axis=1))

        pre_with_index.sort(key=lambda x: x[0], reverse=True)   # 정렬

        if image_dim_order == 'th':
            nice_images = np.zeros((BATCH_SIZE, 1) +
                                   (generated_images.shape[2:]), dtype=np.float32)
        else:
            nice_images = np.zeros((BATCH_SIZE, ) +
                                   (generated_images.shape[1:3]) + (1,), dtype=np.float32)

        for i in range(int(BATCH_SIZE)):                        # 배치크기만큼 (즉 20배 ➲
          중 5%만) 선정
            idx = int(pre_with_index[i][1])
            if image_dim_order == 'th':
                nice_images[i, 0, :, :] = generated_images[idx, 0, :, :]
            else:
                nice_images[i, :, :, 0] = generated_images[idx, :, :, 0]
        image = combine_images(nice_images)
    else:
        noise = np.zeros((BATCH_SIZE, 100))
        for i in range(BATCH_SIZE):
            noise[i, :] = np.random.uniform(-1, 1, 100)
        generated_images = generator.predict(noise, verbose=1)
        image = combine_images(generated_images)
    image = image*127.5+127.5
    Image.fromarray(image.astype(np.uint8)).save(
        "generated_image.png")


def get_args():
    parser = argparse.ArgumentParser()
    parser.add_argument("--mode", type=str)
    parser.add_argument("--batch_size", type=int, default=128)
    parser.add_argument("--nice", dest="nice", action="store_true")
    parser.add_argument("--epoch_num",type=int,default=100)
    parser.set_defaults(nice=False)
    args = parser.parse_args()
    return args
```

```
if __name__ == "__main__":
    # path to save generative images
    if not os.path.exists('images'):
        os.mkdir('images')
    args = get_args()
    if args.mode == "train":
        print ('totol epochs of the train:'+str(args.epoch_num))
        train(BATCH_SIZE=args.batch_size , epoch_num=args.epoch_num)
    elif args.mode == "generate":
        generate(BATCH_SIZE=args.batch_size, nice=args.nice)
```