from keras.datasets import mnist

import numpy as np

from keras.layers import Activation, Dense

from keras.layers import Concatenate

(X\_train, \_), (\_, \_) = mnist.load\_data()

X\_train = (X\_train.astype(np.float32) - 127.5) / 127.5

X\_train = np.expand\_dims(X\_train, axis=3)

def build\_generator(noise\_shape=(100,)):

          input = input(noise\_shape)

x = Dense(128 \* 7 \* 7, activation="relu")(input)

x = Reshape((7, 7, 128))(x)

x = BatchNormalization(momentum=0.8)(x)

x = UpSampling2D()(x)

x = Conv2D(128, kernel\_size=3, padding="same")(x)

x = Activation("relu")(x)

x = BatchNormalization(momentum=0.8)(x)

x = UpSampling2D()(x)

x = Conv2D(64, kernel\_size=3, padding="same")(x)

x = Activation("relu")(x)

x = BatchNormalization(momentum=0.8)(x)

x = Conv2D(1, kernel\_size=3, padding="same")(x)

out = Activation("tanh")(x)

model = Model(input, out)

print("-- Generator -- ")

model.summary()

return model

def build\_discriminator(img\_shape):

        input = input(img\_shape)

x =Conv2D(32, kernel\_size=3, strides=2, padding="same")(input)

x = LeakyReLU(alpha=0.2)(x)

x = Dropout(0.25)(x)

x = Conv2D(64, kernel\_size=3, strides=2, padding="same")(x)

x = ZeroPadding2D(padding=((0, 1), (0, 1)))(x)

x = (LeakyReLU(alpha=0.2))(x)

x = Dropout(0.25)(x)

x = BatchNormalization(momentum=0.8)(x)

x = Conv2D(128, kernel\_size=3, strides=2, padding="same")(x)

x = LeakyReLU(alpha=0.2)(x)

x = Dropout(0.25)(x)

x = BatchNormalization(momentum=0.8)(x)

x = Conv2D(256, kernel\_size=3, strides=1, padding="same")(x)

x = LeakyReLU(alpha=0.2)(x)

x = Dropout(0.25)(x)

x = Flatten()(x)

out = Dense(1, activation='sigmoid')(x)

model = Model(input, out)

print("-- Discriminator -- ")

model.summary()

return model

discriminator = build\_discriminator(img\_shape=(28, 28, 1))

generator = build\_generator()

z = input(shape=(100,))

img = generator(z)

discriminator.trainable = False

real = discriminator(img)

combined = Model(z, real)

gen\_optimizer = Adam(lr=0.0002, beta\_1=0.5)

disc\_optimizer = Adam(lr=0.0002, beta\_1=0.5)

discriminator.compile(loss='binary\_crossentropy',

optimizer=disc\_optimizer,

metrics=['accuracy'])

generator.compile(loss='binary\_crossentropy', optimizer=gen\_optimizer)

combined.compile(loss='binary\_crossentropy', optimizer=gen\_optimizer)

num\_examples = X\_train.shape[0]

num\_batches = int(num\_examples / float(batch\_size))

half\_batch = int(batch\_size / 2)

for epoch in range(epochs + 1):

    for batch in range(num\_batches):

# noise images for the batch

      noise = np.random.normal(0, 1, (half\_batch, 100))

fake\_images = generator.predict(noise)

fake\_labels = np.zeros((half\_batch, 1))

# real images for batch

idx = np.random.randint(0, X\_train.shape[0], half\_batch)

real\_images = X\_train[idx]

real\_labels = np.ones((half\_batch, 1))

# Train the discriminator (real classified as ones and generated as zeros

d\_loss\_real = discriminator.train\_on\_batch(real\_images,real\_labels)

d\_loss\_fake = discriminator.train\_on\_batch(fake\_images,fake\_labels)

d\_loss = 0.5 \* np.add(d\_loss\_real, d\_loss\_fake)

noise = np.random.normal(0, 1, (batch\_size, 100))

# Train the generator

g\_loss = combined.train\_on\_batch(noise, np.ones((batch\_size, 1)))

# Plot the progress

print("epoch %d Batch %d/%d [D loss: %f, acc.: %.2f%%] [G loss: %f]%"(epoch,batch, num\_batches, d\_loss[0], 100 \* d\_loss[1], g\_loss))

if batch % 50 == 0:

       save\_imgs(generator, epoch, batch)

       noise = np.random.normal(0, 1, (half\_batch, 100))

       fake\_images = generator.predict(noise)

       fake\_labels = np.zeros((half\_batch, 1))

       idx = np.random.randint(0, X\_train.shape[0], half\_batch)

       real\_images = X\_train[idx]

       real\_labels = np.ones((half\_batch, 1))

       # Train the discriminator (real classified as ones and generated as zeros)

       d\_loss\_real = discriminator.train\_on\_batch(real\_images, real\_labels)

       d\_loss\_fake = discriminator.train\_on\_batch(fake\_images, fake\_labels)

       d\_loss = 0.5 \* np.add(d\_loss\_real, d\_loss\_fake)

       noise = np.random.normal(0, 1, (batch\_size, 100))

       g\_loss = combined.train\_on\_batch(noise, np.ones((batch\_size, 1)))

       print("Epoch %d Batch %d/%d [D loss: %f, acc.: %.2f%%] [G loss: %f]" %

      (epoch,batch, num\_batches, d\_loss[0], 100 \* d\_loss[1], g\_loss))

       if batch % 50 == 0:

          save\_imgs(generator, epoch, batch)

          def save\_imgs(generator, epoch, batch):

           r, c = 5, 5

           noise = np.random.normal(0, 1, (r \* c, 100))

           gen\_imgs = generator.predict(noise)

           gen\_imgs = 0.5 \* gen\_imgs + 0.5

           fig, axs = plt.subplots(r, c)

           cnt = 0

for i in range(r):

        for j in range(c):

            axs[i, j].imshow(gen\_imgs[cnt, :, :, 0], cmap='gray')

            axs[i, j].axis('off')

            cnt += 1

        fig.savefig("images/mnist\_%d\_%d.png" % (epoch, batch))

        plt.close()

        plt.show()