1 深度卷積 GAN - DCGAN

1.1 介紹

在上幾節，你會發現我們已經正確的開始創作了，我們先秀一下之前 GAN 的結果

圖: 使用原始 GAN 產生的數字

你會發現我們有許多雜點在四周圍，為什麼呢？我們的 GAN 是這樣的

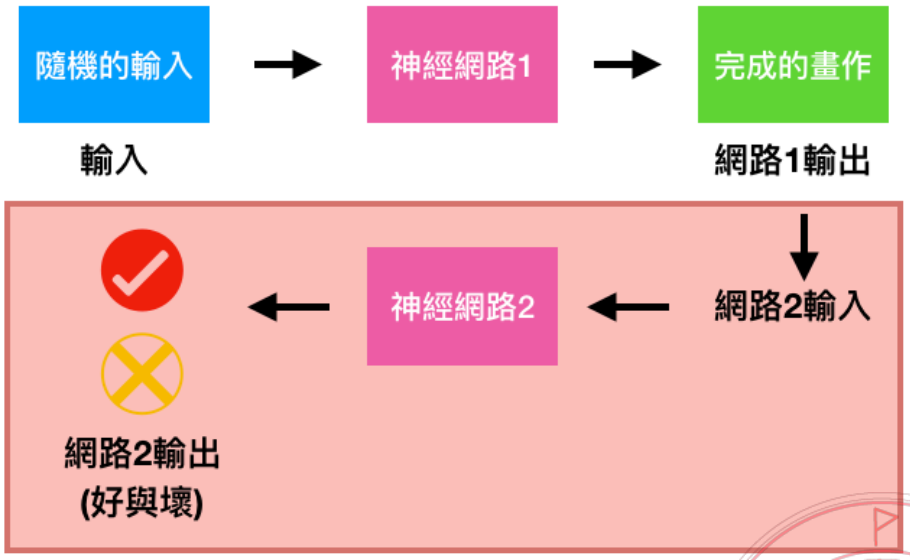


圖: 原始 GAN 的整個創作架構

那我們的神經網路選用的是 MLP(多層感知器)！在深度學習的基礎裡面，我們曾經學過，單純 MLP 對於複雜的圖像就會開始力不從心，為什麼呢？因為使用 MLP 和我們人類平常的感知其實是有點差距的，MLP 是把所有的『像素』攤開，用全部『像素』做出一個判斷！

那你會發現人類不是這麼判斷事物的，你的眼睛會『聚焦』在你這次的目標上，而忽略其他『無關緊要的像素』，這就是我們所謂的『特徵抓取』，而我們也學過，我們有模仿這個『特徵抓取』的網路，也就是所謂的 CNN(卷積網路)！之前我們的卷積網路就是原圖 - CNN(眼睛) - MLP(大腦) - 判斷這樣一個完整流程

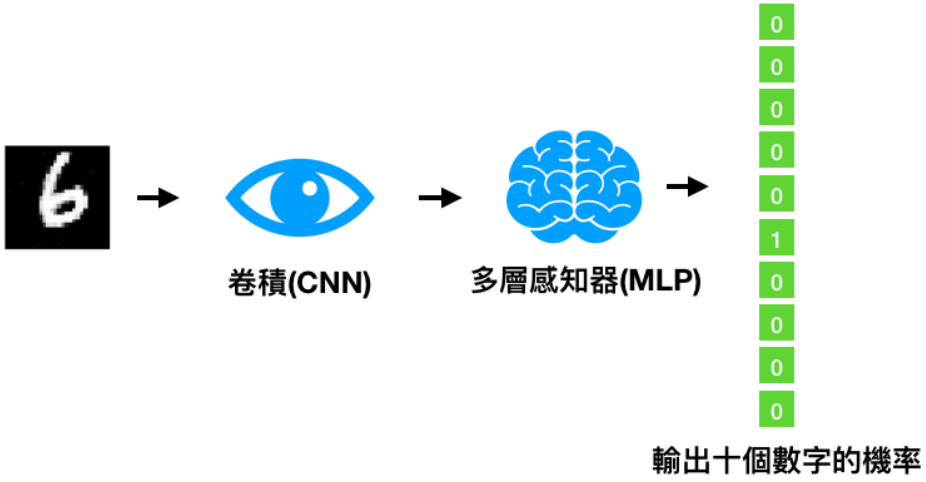


圖: 傳統卷積網路

回到我們原本的問題，你想想我們的判斷器會因為那一點點小點就判斷不正確嗎？很顯然不會，這也導出一個問題，單純使用 MLP 的 GAN 像是一個『死背書的孩子』，他不理解創作的『神』(重要特徵)，而是硬記『形』(全部像素)，那怎麼辦呢？很簡單！就把抓取『神』(卷積網路) 加入其中即可！

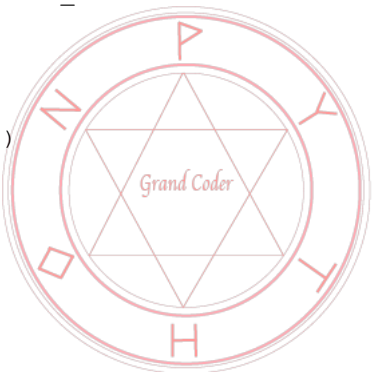
1.2 ✓ Step1. 準備資料集

一樣來試試看 MNIST 資料集

```
[程式]: # 我們會使用到一些內建的資料庫, MAC 需要加入以下兩行, 才不會把對方的 ssl 憑證視為無效
        import ssl
        ssl._create_default_https_context = ssl._create_unverified_context

[程式]: from keras.datasets import mnist
        # 回傳值: ((訓練特徵, 訓練目標), (測試特徵, 測試目標))
        (x_train, y_train),(x_test, y_test) = mnist.load_data()
```

老樣子的看看 shape



```
[程式]: x_train.shape

[輸出]: (60000, 28, 28)
```

1.3 ✓ Step2. 建立創作家

1.4 反卷積/轉置卷積

還記得之前我們的創作家是使用一個反向的 MLP，那現在在卷積的世界我們又該如何反向呢? 其實也是一樣，最早的卷積長得像這樣!

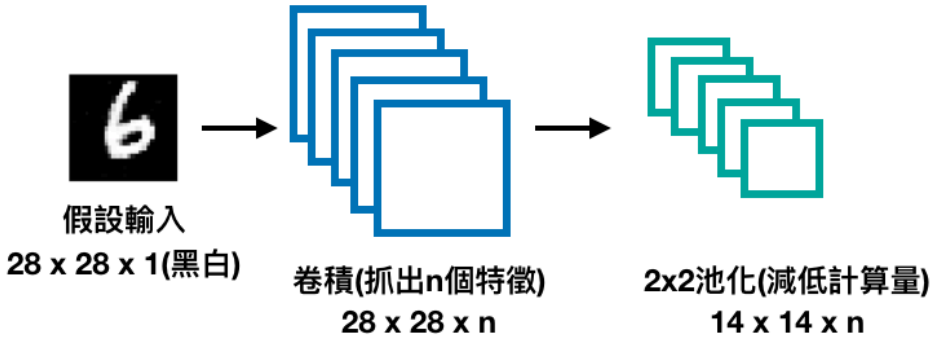


圖: 之前我們習慣的標準版卷積

反向的卷積我們通常稱:

1. 反卷積: 其實這個反會容易讓人誤解，會誤以為是『逆』的意思，誤以為兩個乘起來會等於『單位矩陣』(你可以想像成矩陣版的 1)
2. 轉置卷積: 這個稱呼會比較好一點，因為『轉置矩陣』就比較有像維度反轉的感覺，e.g. 二維向量 - (2, 3) 矩陣 - 三維向量，而三維向量 - (3, 2) 矩陣 - 二維向量，如果要避免誤導，說轉置矩陣倒是會比較好一點

不過撇除名稱這種小小問題，後續兩個稱呼大家應該都知道我們在講同一個東西，我們通常會這樣來實作一個轉置卷積

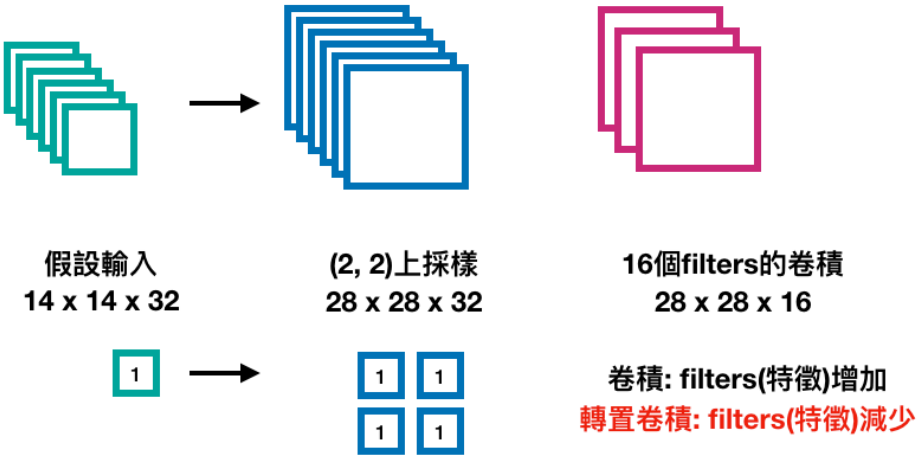
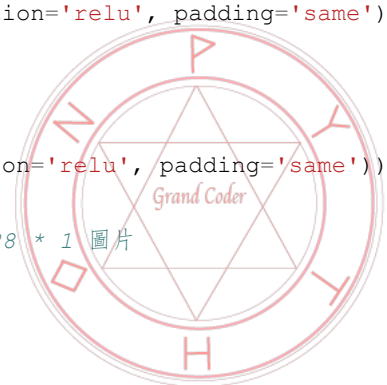


圖: 轉置卷積

你可以把 (2, 2) 的上採樣 (UpSampling) 當成池化的反向，接著再做一次 filters 數目減少的卷積，你會發現剛好是跟我們平常卷積相反的過程

1. 卷積: 特徵數目增加的卷積 - 池化讓長寬減少
2. 轉置 (反) 卷積: 上採樣讓長寬增加 - 特徵數目減少的卷積

```
[程式]: from keras.layers import Input
        from keras.models import Model, Sequential
        from keras.layers.core import Reshape, Dense, Dropout, Flatten
        from keras.layers import BatchNormalization, UpSampling2D, Conv2D
        random_dim = 100
        generator = Sequential()
        # 先讓 100 隨機亂數可以變成 7 * 7 * 128
        # 為何是 7 * 7 呢?
        # 因為 7 *7 -> (第一次轉置) 14 * 14 -> (第二次轉置) 28 * 28
        # 128 則是使用類似 VGG 的概念, 選擇 128 開始
        generator.add(Dense(7 * 7 * 128, input_dim=random_dim, activation='relu'))
        # 轉換成三維
        generator.add(Reshape((7, 7, 128)))
        # 上採樣, 長寬變兩倍
        generator.add(UpSampling2D(size=(2, 2)))
        # (4, 4) 卷積窗的卷積, 之所以做 (4, 4) 是為了跟 discriminator 配合, 我們等 discriminator 再談
        generator.add(Conv2D(128, kernel_size=(4, 4), activation='relu', padding='same'))
        # 卷積層間我喜歡使用 BN 來 normalize
        generator.add(BatchNormalization())
        generator.add(UpSampling2D(size=(2, 2)))
        generator.add(Conv2D(64, kernel_size=(4, 4), activation='relu', padding='same'))
        generator.add(BatchNormalization())
        # 最後讓 filter 數目回到 1, 因為是灰階圖片, 最後輸出 28 * 28 * 1 圖片
```



```
    # 一樣使用 tanh(-1 - 1) 作為激活
    generator.add(Conv2D(1, kernel_size=(4, 4), activation='tanh', padding='same'))
    generator.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 6272)              633472
_________________________________________________________________
reshape_2 (Reshape)          (None, 7, 7, 128)         0
_________________________________________________________________
up_sampling2d_3 (UpSampling2 (None, 14, 14, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 14, 14, 128)       262272
_________________________________________________________________
batch_normalization_3 (Batch (None, 14, 14, 128)       512
_________________________________________________________________
up_sampling2d_4 (UpSampling2 (None, 28, 28, 128)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 28, 28, 64)        131136
_________________________________________________________________
batch_normalization_4 (Batch (None, 28, 28, 64)        256
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 28, 28, 1)         1025
=================================================================
Total params: 1,028,673
Trainable params: 1,028,289
Non-trainable params: 384
_________________________________________________________________
```

1.5 ✓ Step3. 建立鑑賞家

1.5.1 步長 2 卷積

以前我們習慣的是步長 1 的卷積窗，也就是每次卷積窗移動一格，再加上最外圍的 padding，我們可以保持長寬不變，再加上 2 × 2 的池化，讓長寬縮小，減低計算量，但大家後來想了一想，何不就直接一次走兩格就好呢!! 一次走兩格的話不就相當於長寬縮小一半嗎? 因此，你可以這樣記得

步長 2 卷積 = 卷積 + 池化

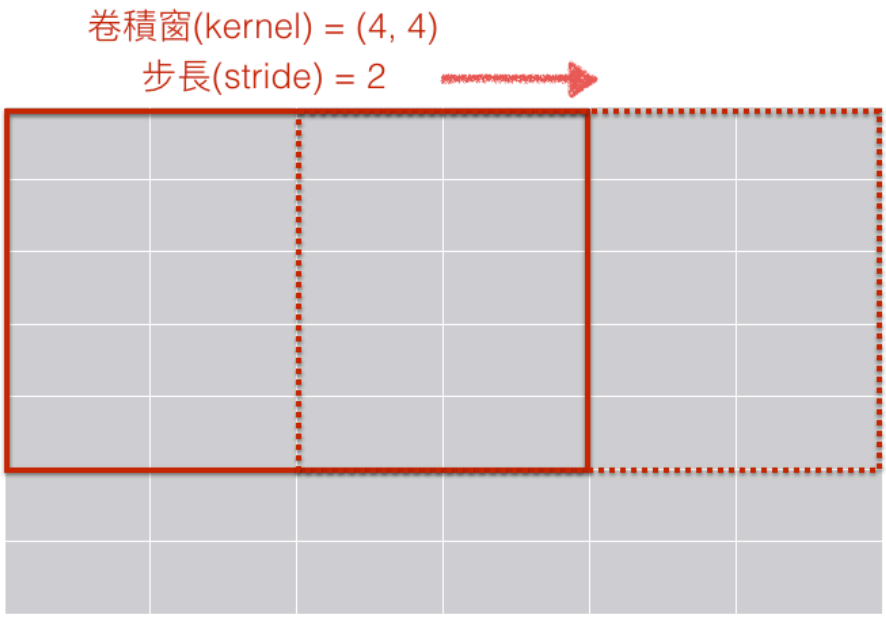
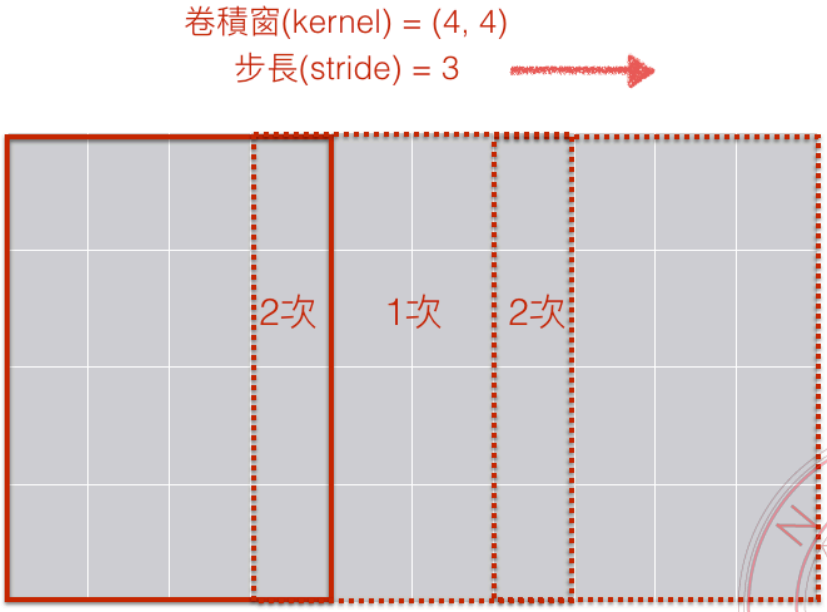


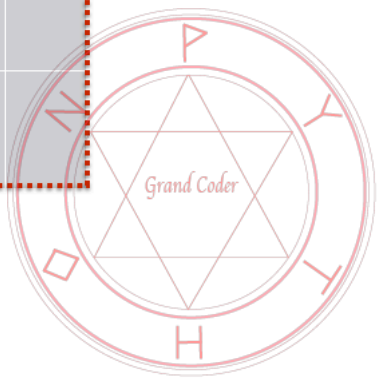
圖: 步長 2 卷積

不過會發現

大家通常會選用 (4, 4) 卷積窗和步長 2，而不是我們之前常用的 (3, 3) 卷積窗

因為如果你的步長不能整除卷積窗的話，會有貢獻不均衡的問題
我們把不均衡的例子放在下面





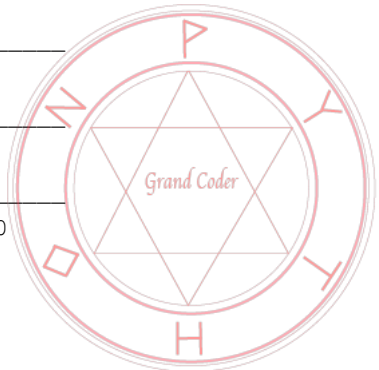
你會發現我們的步長 3 沒辦法整除卷積窗的大小 4

這樣就會有人只貢獻一次，很容易讓我們的特徵抓取變成一個不均衡的抓取！導致產生的圖像方格化 (一行強一行弱，因為貢獻次數不均)，所以請最好讓

卷積窗 / 步長 = 整數

```
[程式]: discriminator = Sequential()
        # 步長 2 卷積
        discriminator.add(Conv2D(32, kernel_size=4,
                                 strides=2,
                                 input_shape=(28, 28, 1),
                                 padding="same",
                                 activation='relu'))
        # 我一樣會在兩層卷積中加入 BN
        discriminator.add(BatchNormalization())
        discriminator.add(Conv2D(64, kernel_size=4,
                                 strides=2,
                                 padding="same",
                                 activation='relu'))
        discriminator.add(BatchNormalization())
        discriminator.add(Conv2D(128, kernel_size=4,
                                 strides=2,
                                 padding="same",
                                 activation='relu'))
        discriminator.add(BatchNormalization())
        # 開始全連接層 (MLP)
        discriminator.add(Flatten())
        discriminator.add(Dense(256, activation='relu'))
        discriminator.add(Dropout(0.25))
        discriminator.add(Dense(1, activation='sigmoid'))
        # 因為鑑賞家是必須單獨訓練的, 所以記得 compile
        discriminator.compile(loss='binary_crossentropy', optimizer="adam")
        discriminator.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 14, 14, 32)        544
_________________________________________________________________
batch_normalization_3 (Batch (None, 14, 14, 32)        128
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 7, 7, 64)          32832
_________________________________________________________________
batch_normalization_4 (Batch (None, 7, 7, 64)          256
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 4, 4, 128)         131200
```



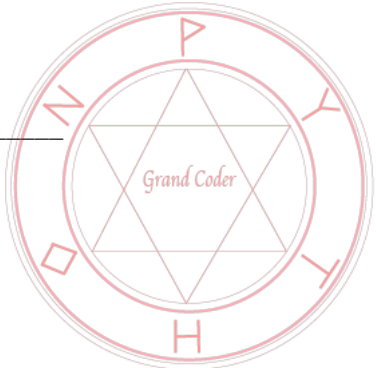
```
_________________________________________________________________
batch_normalization_5 (Batch (None, 4, 4, 128)         512
_________________________________________________________________
flatten_1 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_2 (Dense)              (None, 256)               524544
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 257
=================================================================
Total params: 690,273
Trainable params: 689,825
Non-trainable params: 448
_________________________________________________________________
```

1.6 ✓ Step4. 組合網路

這裡跟我們之前一樣，先將鑑賞家設定成固定，在開始組合網路

```
[程式]: from keras.layers import Input
        discriminator.trainable = False
        gan_input = Input(shape=(random_dim,))
        x = generator(gan_input)
        gan_output = discriminator(x)
        gan = Model(inputs=gan_input, outputs=gan_output)
        gan.compile(loss='binary_crossentropy', optimizer="adam")
        gan.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 100)               0
_________________________________________________________________
sequential_1 (Sequential)    (None, 28, 28, 1)         1028673
_________________________________________________________________
sequential_2 (Sequential)    (None, 1)                 690273
=================================================================
Total params: 1,718,946
Trainable params: 1,028,289
Non-trainable params: 690,657
_________________________________________________________________
```



1.7 ✓ Step5. 開始訓練

開始訓練

[程式]:
```
import numpy as np
from keras.utils import np_utils
# reshape 讓他從 32 * 32 變成 784 * 1 的一維陣列
# 讓我們標準化到-1~1 區間
x_train_shaped = (x_train - 127.5)/127.5
# 由於我們設定輸入是 (28, 28, 1) 所以把 (60000, 28, 28) -> (60000, 28, 28, 1)
x_train_shaped = np.expand_dims(x_train_shaped, axis=3)
print('原本維度:', x_train.shape)
print('Expand 後維度:', x_train_shaped.shape)
```

```
原本維度: (60000, 28, 28)
Expand 後維度: (60000, 28, 28, 1)
```

[程式]:
```
batch_size = 200
epoch_count = 10
for epoch in range(0, epoch_count):
    for batch_count in range(0, 300):
        idx = np.random.randint(0, x_train.shape[0], batch_size)
        imgs = x_train_shaped[idx]

        valid = np.ones((batch_size, 1))
        fake = np.zeros((batch_size, 1))
        # 步驟 0: 讓創作家製造出 fake image
        noise = np.random.normal(0, 1, (batch_size, random_dim))
        gen_imgs = generator.predict(noise)

        discriminator.trainable = True
        # 步驟 1: 讓鑑賞家鑑賞對的 image
        d_loss_real = discriminator.train_on_batch(imgs, valid)
        # 步驟 2: 讓鑑賞家鑑賞錯的 image
        d_loss_fake = discriminator.train_on_batch(gen_imgs, fake)
        d_loss = (d_loss_real + d_loss_fake) / 2

        discriminator.trainable = False
        noise = np.random.normal(0, 1, (batch_size, random_dim))
        # 步驟 3: 訓練創作家的創作能力
        g_loss = gan.train_on_batch(noise, valid)
    if (epoch + 1) % 10 == 0:
        dash = "-" * 15
        print(dash, "epoch", epoch + 1, dash)
        print("Discriminator loss:", d_loss)
        print("Generator loss:", g_loss)
```

```
--------------- epoch 10 ---------------
Discriminator loss: 0.030969567596912384
Generator loss: 3.6941195
```

上面只是我最後 10 次的 loss，其實大概訓練了數個 epochs 了，你可以看到我們下面的成果，雖然並不一定比 MLP 出來的數字好看，但你會發現，跟之前 MLP 不同的點是，電腦似乎是在學習每一個數字的特徵，而不是表象了，也很少出現外面的雜點了！這才是我們要的，讓機器真的學會數字的『神』而不是只是硬記其『形』!

[程式]:
```
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
examples = 100
noise = np.random.normal(0, 1, (examples, random_dim))
gen_imgs = generator.predict(noise)

# Rescale images 0 - 1
gen_imgs = 0.5 * gen_imgs + 0.5
gen_imgs = gen_imgs.reshape(examples, 28, 28)
plt.figure(figsize = (14, 14))

w = 10
h = int(examples / w) + 1
for i in range(0, examples):
    plt.subplot(h, w, i + 1)
    plt.axis('off')
    plt.imshow(gen_imgs[i], cmap='gray')
```