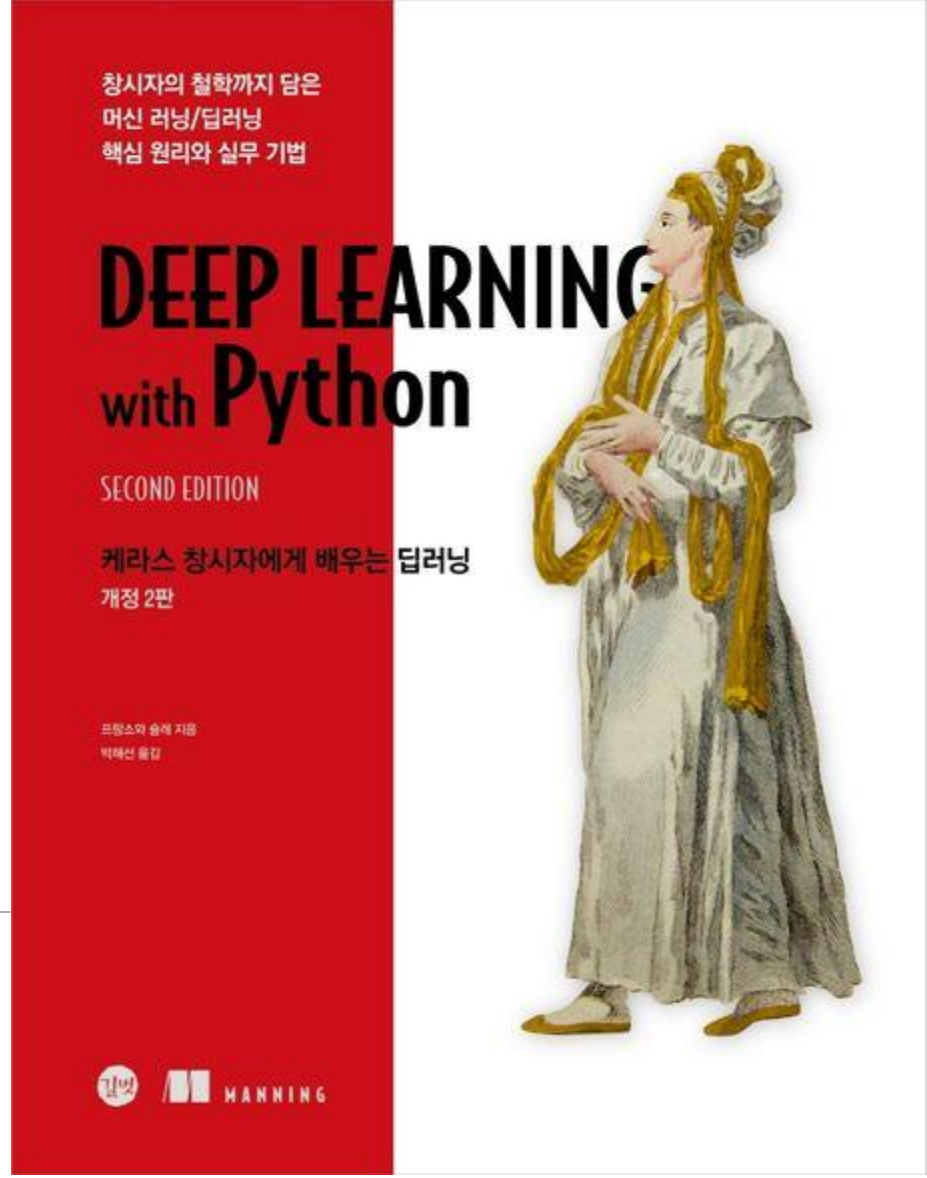
딥러닝

대구가톨릭대학교 AI빅데이터공학과

이 승 민



12章 用于生成模型的深度学习

12.1 文本生成

12.2 DeepDream（深度梦境）

12.3 神经风格迁移（Neural Style Transfer）

12.4 利用变分自编码器（VAE）进行图像生成

12.5 生成对抗网络（GAN）简介

12.6 小结（总结）

12.4 使用变分自编码器进行图像生成

12.4 使用变分自编码器进行图像生成

使用变分自编码器进行图像生成

- 当今在创造性 AI 中最受欢迎、最成功的应用之一就是图像生成。
- 模型学习潜在视觉空间（latent visual space），并在这个空间中进行采样，从实际照片之间进行插值，从而生成全新的图像。
- 例如虚拟人物、虚拟空间、虚拟猫和狗的图像等。
- 本章将同时介绍该领域的两大核心方法：变分自编码器（VAE）与生成式对抗网络（GAN），并深入讲解它们的具体实现。
- 这里介绍的技术并不仅限于图像。
- 利用 GAN 和 VAE 也可以构建声音、音乐或文本的潜在空间。
- 在实际应用中，当使用照片进行处理时，往往能得到最有趣的结果。

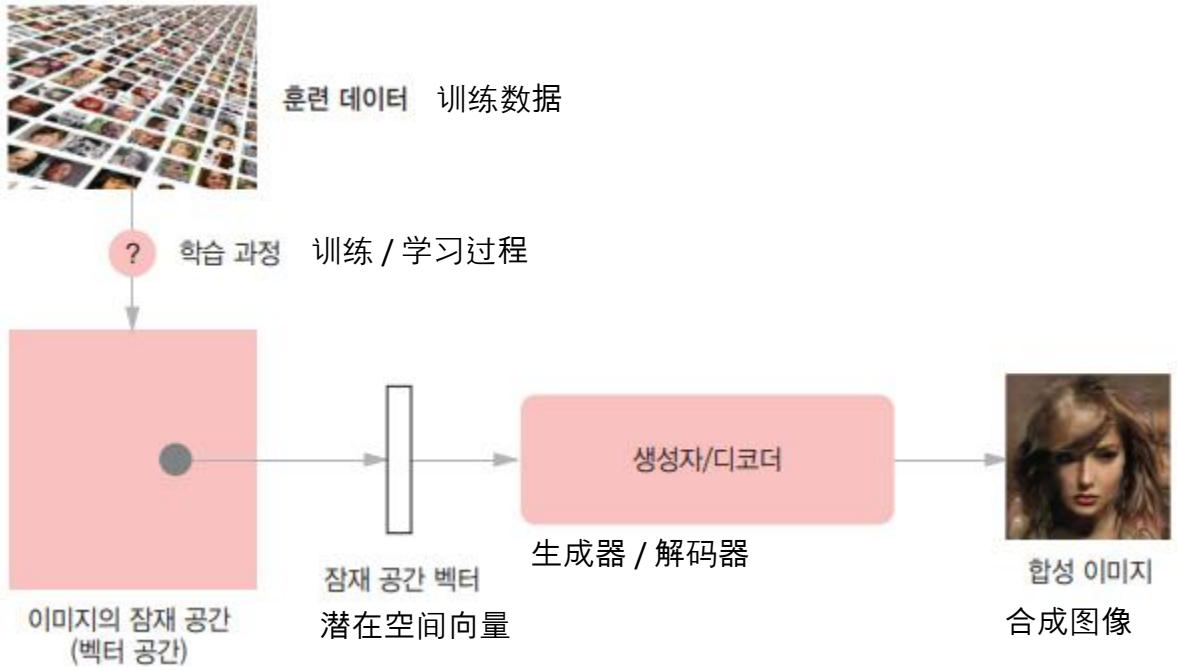
12.4 使用变分自编码器进行图像生成

使用变分自编码器进行图像生成

- 图像生成的核心思想是：构建一个低维的潜在空间表示，使其中的每个点都能映射为逼真的图像（和深度学习中其它内容一样，这也是一个向量空间）。
- 在这个潜在空间中取一个点作为输入，并输出图像（像素网格）的模块，在 GAN 中称为 生成器（generator），在 VAE 中称为 解码器（decoder）。
- 当这个潜在空间被学习完成后，我们就能从其中采样任意一个点。
- 接着将该点映射回图像空间，就能生成原训练集中从未出现过的新图像（参见图 12–13）。
- 这些新生成的图像位于训练样本之间的空间中。

12.4 使用变分自编码器进行图像生成

▼ 图 12-13 学习图像的潜在向量空间并利用该空间对新图像进行采样



图像的潜在空间（向量空间）

12.4 使用变分自编码器进行图像生成

在图像的潜在空间中进行采样

- GAN 和 VAE 是学习图像潜在空间表示的两种策略，并且各自具有不同的特点。
- VAE 擅长学习结构化的潜在空间（见图 12–14）。
- 在这个空间中，某些方向会编码来自数据中具有意义的变化方向。
- GAN 能够生成非常逼真的图像。
- 然而由 GAN 生成的潜在空间可能并不具备结构性或连续性。

12.4 使用变分自编码器进行图像生成

▼ 图 12-14 使用 VAE 生成的汤姆·怀特（Tom White）的人脸连续空间

12.4 使用变分自编码器进行图像生成

用于图像变换的概念向量

- 在第 11 章处理中我们已经学习过词嵌入，并获得了关于**概念向量（concept vector）**的想法。
- 这里的思想与之前完全相同。
- 当给定潜在空间或嵌入空间时，在这个空间中某些方向可以被视为编码原始数据中有趣变化的坐标轴。

12.4 使用变分自编码器进行图像生成

用于图像变换的概念向量

- 例如，在人脸图像的潜在空间中，可能存在一个“微笑向量”。
- 如果潜在空间中的点 z 表示某张脸的嵌入表示，那么点 z + s 就表示同一张脸但带着笑容的嵌入表示。
- 如果能够找到这样的向量，就可以将图像投影到潜在空间中，并沿着这个有意义的方向移动，再解码回图像空间，从而得到变化后的图像。
- 基本上，图像空间中那些可以独立变化的维度都可以视为概念向量。
- 对于人脸来说，可以找到例如戴眼镜 ↔ 不戴眼镜、男脸 ↔ 女脸等这样的向量。
- 图 12-15 展示的是新西兰维多利亚大学设计学院的 Tom White 找到的概念向量中的“微笑向量”的例子。
- 使用了在名人脸数据集（CelebA 数据集）上训练的 VAE。

12.4 使用变分自编码器进行图像生成

▼ 图 12-15 微笑向量

12.4 使用变分自编码器进行图像生成

变分自编码器（VAE）

- 变分自编码器是由 Kingma 与 Welling 于 2013 年 12 月，以及 Rezende、Mohamed 与 Wierstra 于 2014 年 1 月几乎同时提出的一类生成模型，它非常适合利用概念向量进行图像变换。
- 自编码器是一种将输入编码到低维潜在空间，再通过解码器还原的网络结构。
- 变分自编码器是将深度学习与贝叶斯推断（Bayesian inference）的思想结合而成的最新版本自编码器。

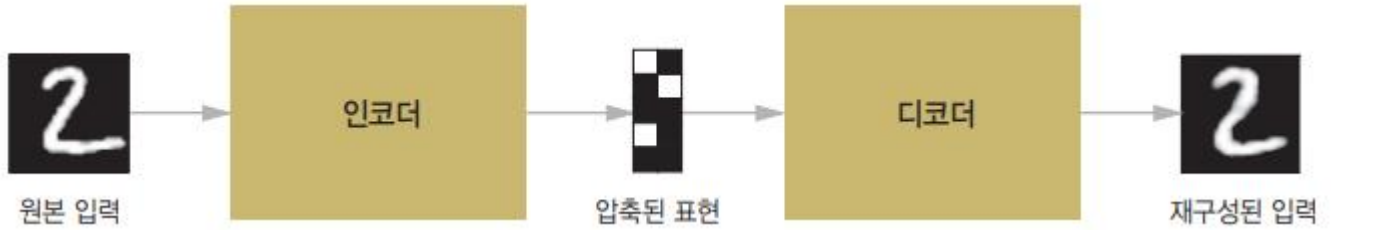
12.4 使用变分自编码器进行图像生成

变分自编码器（VAE）

- 传统的自编码器将输入图像送入编码器模块，将其映射到潜在向量空间。
- 然后再使用解码器模块，将其还原成与原始图像相同维度的输出（图 12-16）。
- 自编码器使用与输入图像相同的图像作为目标数据进行训练。
- 换句话说，自编码器学习的是如何重构原始输入。
- 如果对编码（coding，指编码器的输出）施加某些限制，那么自编码器就可以学习到更有趣或更无趣的潜在空间表示。
- 通常会对编码施加约束，使其低维且稀疏（即包含大量 0）。
- 在这种情况下，编码器会努力将输入数据的信息压缩到更少的比特中。

12.4 使用变分自编码器进行图像生成

▼ 图 12-16 自编码器：将输入 x 映射为压缩后的表示，并对其解码以恢复出 x'



12.4 使用变分自编码器进行图像生成

变分自编码器（VAE）

- 实际上，传统的自编码器并不能生成特别有用或结构化良好的潜在空间。
- 它的压缩能力也并不出色。
- 因此，传统自编码器逐渐在潮流中被淘汰。
- VAE 在自编码器的基础上加入了统计方法，使其能够学习连续且结构化的潜在空间。
- 最终，VAE 演变成了用于图像生成的强大工具。

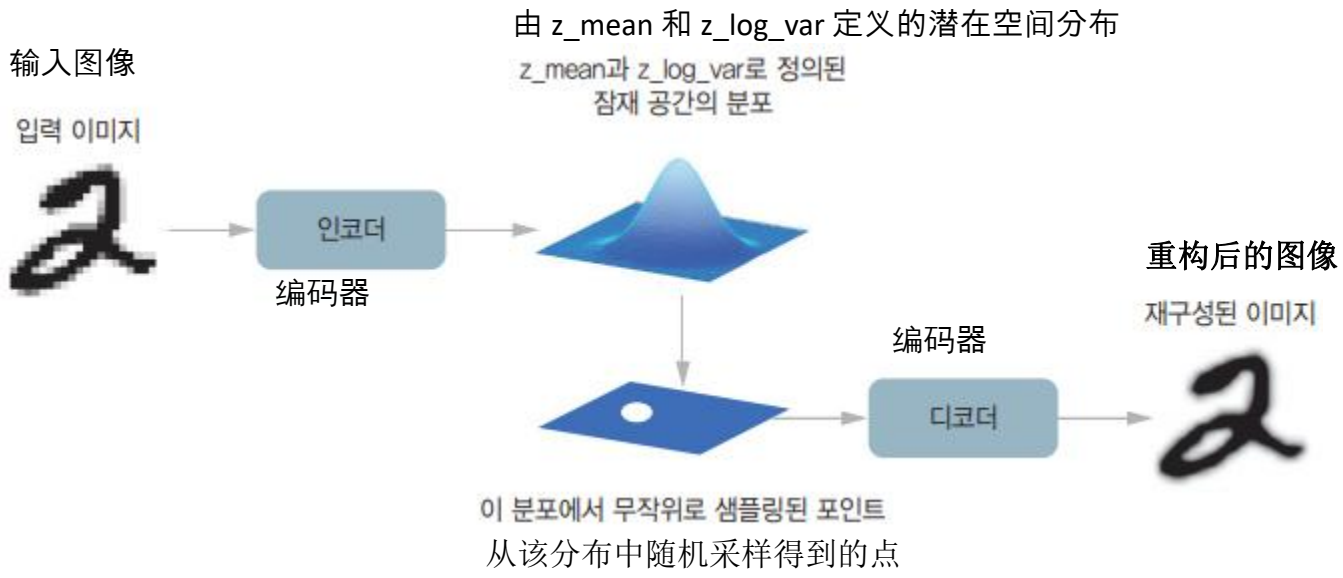
12.4 使用变分自编码器进行图像生成

变分自编码器（VAE）

- 与其将输入图像压缩成潜在空间中的固定编码，VAE 会将图像映射成某个统计分布的参数。
- 这意味着在编码和解码过程中会引入随机性，因为假设输入图像是通过统计过程生成的。
- VAE 使用均值和方差参数从该分布中随机采样一个样本。
- 然后对该样本进行解码并恢复为原始输入（图 12-17）。
- 这种随机过程提高了稳定性，并能让潜在空间的任意位置都编码有意义的表示。
- 换句话说，从潜在空间中采样得到的任何点都可以被解码为有效输出。

12.4 使用变分自编码器进行图像生成

▼ 图 12-17 VAE 将图像映射为两个向量 z_mean 和 z_log_var。
这两个向量在潜在空间中定义概率分布，并用于采样潜在空间中的点以进行解码。



12.4 使用变分自编码器进行图像生成

变分自编码器（VAE）

- 从技术角度来看，VAE 的工作过程如下：
  - 1. 编码器模块将输入样本 **input_img** 转换为潜在空间的两个参数 **z_mean** 和 **z_log_var**。
  - 2. 假设输入图像由某个统计过程生成，则从潜在空间的正态分布中随机采样点 **z**： z = z_mean + exp(0.5 * z_log_var) *epsilon
    - 其中 **epsilon** 是一个取值较小的随机张量。
  - 3解码器模块将该潜在空间中的点映射回原始输入图像并进行重构。

12.4 使用变分自编码器进行图像生成

变分自编码器（VAE）

- 因为 epsilon 是随机生成的，所以靠近输入图像编码位置（z_mean）的潜在点会被解码为与 input_img 相似的图像。
- 这使潜在空间成为一个连续且有意义的空间。
- 在潜在空间中彼此接近的两个点，将被解码为非常相似的图像。
- 这种潜在空间的低维连续性，使得潜在空间中所有方向都能够编码数据的有意义变化。
- 因此，潜在空间变得高度结构化，并且非常适用于用概念向量来处理。

12.4 使用变分自编码器进行图像生成

变分自编码器（VAE）

- VAE 的参数通过两个损失函数进行训练。
- 一个是使解码后的样本尽可能与原始输入一致的重构损失（reconstruction loss），
- 另一个是为了良好地形成潜在空间并减少对训练数据过拟合的正则化损失（regularization loss）。
- 概括来说如下：

```
z_mean, z_log_var = encoder(input_img) -------- 입력을 평균과 분산 파라미터로 인코딩합니다.
z = z_mean + exp(0.5 * z_log_var) * epsilon ---------- 무작위로 선택한 작은 epsilon 값을 사용
                                                       하여 잠재 공간의 포인트를 뽑습니다.
reconstructed_img = decoder(z) -------- z를 이미지로 디코딩합니다.
model = Model(input_img, reconstructed_img)-------- 입력 이미지와 재구성 이미지를 매핑한
                                                    오토인코더 모델 객체를 만듭니다.
```

```
z_mean, z_log_var = encoder(input_img)
# 将输入编码为均值和方差参数。

z = z_mean + exp(0.5 * z_log_var) * epsilon
# 使用随机选取的 epsilon 值，从分布中采样潜在空间的点。

reconstructed_img = decoder(z)
# 将 z 解码成图像。

model = Model(input_img, reconstructed_img)
# 创建一个将输入图像映射到重构图像的自编码器模型对象
```

12.4 使用变分自编码器进行图像生成

变分自编码器（VAE）

- 接下来就可以使用重构损失和正则化损失来训练模型。
- 正则化损失通常采用将编码器输出的分布拉向以 0 为中心的标准正态分布的形式，即 KL 散度（Kullback–Leibler divergence）。
- 这样的正则化为编码器所建模的潜在空间结构提供了合理的假设。

12.4 使用变分自编码器进行图像生成

用 Keras 实现 VAE

- 我们来实现一个可以生成 MNIST 数字的 VAE。
- 该模型由三部分构成：
  - 编码器网络：将真实图像转换为潜在空间中的均值和方差；
  - 采样层：接收这些均值和方差，在潜在空间中随机采样点；
  - 解码器网络：将潜在空间中的点再转换为图像。

12.4 使用变分自编码器进行图像生成

用 Keras 实现 VAE

- 代码 12-24 展示了一个将图像映射为潜在空间概率分布参数的编码器网络。
- 这是一个将输入图像 x 映射成两个向量 z_mean 和 z_log_var 的简单卷积网络。
- 有一个重要点：我们使用步幅（stride）来对特征图进行下采样，而不是使用最大池化。
- 在第 9 章的图像分割示例中，我们最后也是这样做的。
- 对于需要考虑信息位置（information location）的模型来说，步幅通常优于最大池化——
- 也就是说，图像中物体的位置很重要。
- 此外，我们必须创建一种能够用于重构有效图像的编码方式。

12.4 使用变分自编码器进行图像生成

用 Keras 实现 VAE

코드 12-24 VAE 인코더 네트워크

```
from tensorflow import keras
from tensorflow.keras import layers

latent_dim = 2 -------- 잠재 공간의 차원: 2D 평면

encoder_inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(
    32, 3, activation="relu", strides=2, padding="same")(encoder_inputs)
x = layers.Conv2D(64, 3, activation="relu", strides=2, padding="same")(x)
x = layers.Flatten()(x)
x = layers.Dense(16, activation="relu")(x)
z_mean = layers.Dense(latent_dim, name="z_mean")(x)
z_log_var = layers.Dense(latent_dim, name="z_log_var")(x)
encoder = keras.Model(encoder_inputs, [z_mean, z_log_var], name="encoder")
```

입력 이미지는 결국 2개의 파라미터로 인코딩됩니다.

12.4 使用变分自编码器进行图像生成

用 Keras 实现 VAE

- summary() 导出结果如下

```
>>> encoder.summary()
Model: "encoder"
```

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 28, 28, 1)]	0	
conv2d (Conv2D)	(None, 14, 14, 32)	320	input_1[0][0]
conv2d_1 (Conv2D)	(None, 7, 7, 64)	18496	conv2d[0][0]
flatten (Flatten)	(None, 3136)	0	conv2d_1[0][0]

12.4 使用变分自编码器进行图像生成

用 Keras 实现 VAE

```
dense (Dense)                   (None, 16)           50192       flatten[0][0]
__________________________________________________________________________________
z_mean (Dense)                  (None, 2)            34          dense[0][0]
__________________________________________________________________________________
z_log_var (Dense)               (None, 2)            34          dense[0][0]
==================================================================================
Total params: 69,076
Trainable params: 69,076
Non-trainable params: 0
__________________________________________________________________________________
```

12.4 使用变分自编码器进行图像生成

用 Keras 实现 VAE

- 下面是使用 z_mean 和 z_log_var 来生成潜在空间点 z 的代码。
- 假设这两个参数是生成 input_img 的统计分布的参数。

코드 12-25 잠재 공간 샘플링 층

```
import tensorflow as tf

class Sampler(layers.Layer):
    def call(self, z_mean, z_log_var):
        batch_size = tf.shape(z_mean)[0]
        z_size = tf.shape(z_mean)[1]
        epsilon = tf.random.normal(shape=(batch_size, z_size))
        return z_mean + tf.exp(0.5 * z_log_var) * epsilon
```

정규 분포를 따르는 랜덤한 벡터의 배치를 만듭니다.

VAE 샘플링 공식을 적용합니다.

12.4 使用变分自编码器进行图像生成

用 Keras 实现 VAE

- 代码 12-26 实现了解码器。
- 它将向量 z 转换为图像维度，并通过若干卷积层生成最终的图像输出。
- 该输出图像的维度与原始 input_img 相同。

코드 12-26 잠재 공간 포인트를 이미지로 매핑하는 VAE 디코더 네트워크

```
latent_inputs = keras.Input(shape=(latent_dim,)) ····· z를 입력으로 사용합니다.
x = layers.Dense(7 * 7 * 64, activation="relu")(latent_inputs) ····· 인코더에 있는 Flatten 층의 출력 크기와 동일하게 지정합니다.
x = layers.Reshape((7, 7, 64))(x) ····· 인코더의 Flatten 층 작업을 되돌립니다.
x = layers.Conv2DTranspose(64, 3, activation="relu", strides=2, padding="same")(x)
x = layers.Conv2DTranspose(32, 3, activation="relu", strides=2, padding="same")(x)
decoder_outputs = layers.Conv2D(1, 3, activation="sigmoid", padding="same")(x)
decoder = keras.Model(latent_inputs, decoder_outputs, name="decoder")
```

인코더의 Conv2D 층 작업을 되돌립니다.

출력은 결국 (28, 28, 1) 크기가 됩니다.

```
latent_inputs = keras.Input(shape=(latent_dim,))
# 使用 z 作为输入。

x = layers.Dense(7 * 7 * 64, activation="relu")(latent_inputs)
# 输出大小与编码器中的 Flatten 层相同。

x = layers.Reshape((7, 7, 64))(x)
# 还原编码器中 Flatten 层之前的形状。

x = layers.Conv2DTranspose(64, 3, activation="relu", strides=2, padding="same")(x)
x = layers.Conv2DTranspose(32, 3, activation="relu", strides=2, padding="same")(x)

decoder_outputs = layers.Conv2D(1, 3, activation="sigmoid", padding="same")(x)
# 输出的最终尺寸是 (28, 28, 1)。

decoder = keras.Model(latent_inputs, decoder_outputs, name="decoder")
```

12.4 使用变分自编码器进行图像生成

用 Keras 实现 VAE

- summary() 导出结果如下

```
>>> decoder.summary()
Model: "decoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 2)]               0
_________________________________________________________________
dense_1 (Dense)              (None, 3136)              9408
_________________________________________________________________
reshape (Reshape)            (None, 7, 7, 64)          0
_________________________________________________________________
```

12.4 使用变分自编码器进行图像生成

用 Keras 实现 VAE

```
conv2d_transpose (Conv2DTran (None, 14, 14, 64)        36928
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 28, 28, 32)        18464
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 28, 28, 1)         289
=================================================================
Total params: 65,089
Trainable params: 65,089
Non-trainable params: 0
_________________________________________________________________
```

12.4 使用变分自编码器进行图像生成

用 Keras 实现 VAE

- 现在来构建 VAE 模型。
- 这个模型是我们第一次实现的不执行传统监督学习的模型（自编码器是一种把输入本身作为目标的自监督学习 self-supervised learning的例子）。
- 当不是一般的监督学习任务时，通常会像第 7 章所学的那样：继承 Model 类，并实现自定义的 train_step() 方法，把训练逻辑写在里面。

코드 12-27 사용자 정의 train_step() 메서드를 사용하는 VAE 모델

```
class VAE(keras.Model):
    def __init__(self, encoder, decoder, **kwargs):
        super().__init__(**kwargs)
        self.encoder = encoder
        self.decoder = decoder
        self.sampler = Sampler()
        self.total_loss_tracker = keras.metrics.Mean(name="total_loss")
        self.reconstruction_loss_tracker = keras.metrics.Mean(
            name="reconstruction_loss")
        self.kl_loss_tracker = keras.metrics.Mean(name="kl_loss")
```

에포크마다 손실 평균을 추적합니다.

12.4 使用变分自编码器进行图像生成

用 Keras 实现 VAE

각 에포크가 완료된 후 (또는 fit( )과 evaluate( ) 호출 사이에) 모델
이 손실을 재설정할 수 있도록 metrics 속성에 손실을 나열합니다.

```
@property
def metrics(self):
    return [self.total_loss_tracker,
            self.reconstruction_loss_tracker,
            self.kl_loss_tracker]

def train_step(self, data):
    with tf.GradientTape() as tape:
        z_mean, z_log_var = self.encoder(data)
        z = self.sampler(z_mean, z_log_var)
        reconstruction = decoder(z)
        reconstruction_loss = tf.reduce_mean(
            tf.reduce_sum(
```

공간 차원(축 1과 축 2)에 대해 재구성 손실을
더하고 배치 차원에 대해 평균을 계산합니다.

```
@property
def metrics(self):
  # 在每个 epoch 完成后（或 fit() 和 evaluate() 调用之间），
  # 将这些损失暴露给 metrics 属性以便能够重新初始化它们。
  return [
    self.total_loss_tracker,
    self.reconstruction_loss_tracker,
    self.kl_loss_tracker,
  ]


def train_step(self, data):
  with tf.GradientTape() as tape:
    z_mean, z_log_var = self.encoder(data)
    z = self.sampler(z_mean, z_log_var)
    reconstruction = decoder(z)

    reconstruction_loss = tf.reduce_mean(
      # 对空间维度（第 1 和 2 轴）上的重构损失求和，
      # 然后在 batch 维度上求平均。
      tf.reduce_sum(
        ...
      )
    )
```

12.4 使用变分自编码器进行图像生成

用 Keras 实现 VAE

```
                keras.losses.binary_crossentropy(data, reconstruction),
                axis=(1, 2)
            )
        )
        kl_loss = -0.5 * (1 + z_log_var - tf.square(z_mean) -
          tf.exp(z_log_var))
        total_loss = reconstruction_loss + tf.reduce_mean(kl_loss)
    grads = tape.gradient(total_loss, self.trainable_weights)
    self.optimizer.apply_gradients(zip(grads, self.trainable_weights))
    self.total_loss_tracker.update_state(total_loss)
    self.reconstruction_loss_tracker.update_state(reconstruction_loss)
    self.kl_loss_tracker.update_state(kl_loss)
    return {
        "total_loss": self.total_loss_tracker.result(),
        "reconstruction_loss": self.reconstruction_loss_tracker.result(),
        "kl_loss": self.kl_loss_tracker.result(),
    }
```

규제 항(쿨백-라이블러 발산)을 더합니다.

12.4 使用变分自编码器进行图像生成

用 Keras 实现 VAE

- 现在已经创建好了模型对象，可以在 MNIST 数字数据上进行训练了。
- 由于在自定义层中已经自行管理了损失，因此在 compile() 方法里不再额外指定损失函数（loss=None）。
- 这也意味着在训练时不需要传入目标数据（正如代码 12-28 所示，调用 fit() 方法时只传入 mnist_digits）。

코드 12-28 VAE 훈련

```
import numpy as np

(x_train, _), (x_test, _) = keras.datasets.mnist.load_data()
mnist_digits = np.concatenate([x_train, x_test], axis=0)
mnist_digits = np.expand_dims(mnist_digits, -1).astype("float32") / 255

vae = VAE(encoder, decoder)
vae.compile(optimizer=keras.optimizers.Adam(), run_eagerly=True)
vae.fit(mnist_digits, epochs=30, batch_size=128)
```

훈련에 전체 MNIST 숫자를 사용하므로 훈련 샘플과 테스트 샘플을 합칩니다.

손실은 이미 train_step( )에서 처리하기 때문에 compile( ) 메서드에 손실 매개변수를 지정하지 않습니다.

train_step( )은 타깃을 기대하지 않으므로 fit( ) 메서드에 타깃을 전달하지 않습니다.

```
import numpy as np

(x_train, _), (x_test, _) = keras.datasets.mnist.load_data()
# 由于训练时使用整个 MNIST 数字，因此合并训练集与测试集。

mnist_digits = np.concatenate([x_train, x_test], axis=0)
mnist_digits = np.expand_dims(mnist_digits, -1).astype("float32") / 255

vae = VAE(encoder, decoder)

# 由于损失已经在 train_step() 中处理，
# 因此在 compile() 中不指定损失函数参数。
vae.compile(optimizer=keras.optimizers.Adam(), run_eagerly=True)

# train_step() 不需要 targets，所以不给 fit() 传入 targets。
vae.fit(mnist_digits, epochs=30, batch_size=128)
```

12.4 使用变分自编码器进行图像生成

用 Keras 实现 VAE

- 模型训练完成之后，可以使用 decoder 网络把任意的潜在空间向量转换成图像。

코드 12-29 2D 잠재 공간에서 이미지 그리드를 샘플링하기

```
import matplotlib.pyplot as plt

n = 30 -------- 30×30 크기의 숫자 그리드를 출력합니다(총 900개의 숫자).
digit_size = 28
figure = np.zeros((digit_size * n, digit_size * n))

grid_x = np.linspace(-1, 1, n)
grid_y = np.linspace(-1, 1, n)[::-1]      2D 그리드에서 선형적으로
                                          포인트를 샘플링합니다.

for i, yi in enumerate(grid_y):           그리드 위치에 대해 반복합니다.
    for j, xi in enumerate(grid_x):
        z_sample = np.array([[xi, yi]])
        x_decoded = vae.decoder.predict(z_sample)
        digit = x_decoded[0].reshape(digit_size, digit_size)
                                          각 위치에서 숫자를 샘플링하여
                                          그림에 추가합니다.
```

12.4 使用变分自编码器进行图像生成

用 Keras 实现 VAE

```
        figure[
            i * digit_size : (i + 1) * digit_size,
            j * digit_size : (j + 1) * digit_size,
        ] = digit

plt.figure(figsize=(15, 15))
start_range = digit_size // 2
end_range = n * digit_size + start_range
pixel_range = np.arange(start_range, end_range, digit_size)
sample_range_x = np.round(grid_x, 1)
sample_range_y = np.round(grid_y, 1)
plt.xticks(pixel_range, sample_range_x)
plt.yticks(pixel_range, sample_range_y)
plt.xlabel("z[0]")
plt.ylabel("z[1]")
plt.axis("off")
plt.imshow(figure, cmap="Greys_r")
```

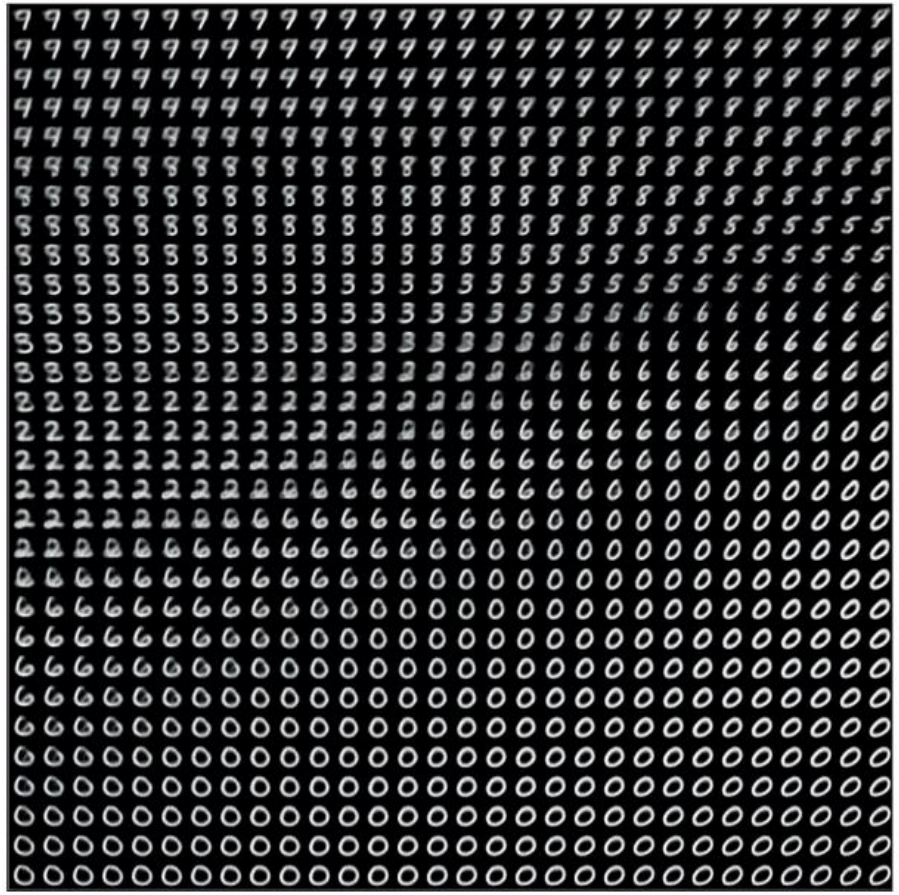
12.4 使用变分自编码器进行图像生成

用 Keras 实现 VAE

- 从采样得到的数字网格（图 12-18）可以看到，不同数字类别之间呈现完全连续的分布
- 沿着潜在空间中的某一条路径，一个数字会自然地变化成另一个数字
- 这个空间中的特定方向具有某种含义
- 例如“朝向 5 的方向”、“朝向 1 的方向”等等

12.4 使用变分自编码器进行图像生成

▼ 图 12-18 从潜在空间解码得到的数字网格



12.4 使用变分自编码器进行图像生成

总结

- 通过深度学习，可以学习包含图像数据统计信息的潜在空间，并用它来生成图像
  - 在潜在空间中采样并解码，可以生成从未见过的新图像
  - 主要方法包括 VAE 和 GAN

- VAE 能够创建非常结构化且连续的潜在空间表示
  - 这样一来，潜在空间中的各种图像变换都能很好完成：
  - 将一张脸变成另一张脸
  - 将皱眉的脸变成微笑的脸
  - 基于潜在空间插值的动画变换也很适合
  - 初始图像可以连续、平滑地变成另一个图像

- GAN 可以生成逼真的单张图像但它 无法构建结构化、连续的潜在空间。

12.5 生成式对抗神经网络介绍

12.5 生成式对抗神经网络介绍

生成式对抗神经网络简介

- 2014 年 Goodfellow 等人提出的生成式对抗网络（GAN）是一种与 VAE 不同的方法来学习图像的潜在空间
- GAN 会强制生成的图像在统计上几乎与真实图像无法区分，从而生成非常逼真的合成图像
- 直观理解 GAN 的方法是把它想象成一个“制作假毕加索画作的伪造者”
- 一开始伪造者的作品非常拙劣
- 把真正的毕加索作品和伪造品混在一起给画商看
- 画商会评判哪些画是真的、哪些像毕加索，并对伪造者进行反馈
- 伪造者回到自己的工作室准备新的伪作
- 随着时间推移，伪造者模仿毕加索风格的能力越来越强
- 画商区分伪作的能力也越来越专业
- 最终，伪造者能够绘制出极为精巧、逼真的“毕加索假画”

12.5 生成式对抗神经网络介绍

生成式对抗神经网络介绍

- GAN 就是由“伪造者网络”和“专家网络”组成的系统
- 这两个网络通过互相对抗来进行训练
- GAN 包含两个网络：
  - **生成器网络（generator network）**
  - 输入一个随机向量（潜在空间中的随机点）将它解码成合成图像
  - 判别器网络（ discriminator 或 adversary network ）
    - 输入图像（真实图像或生成的图像）判断它来自训练数据还是生成器生成的

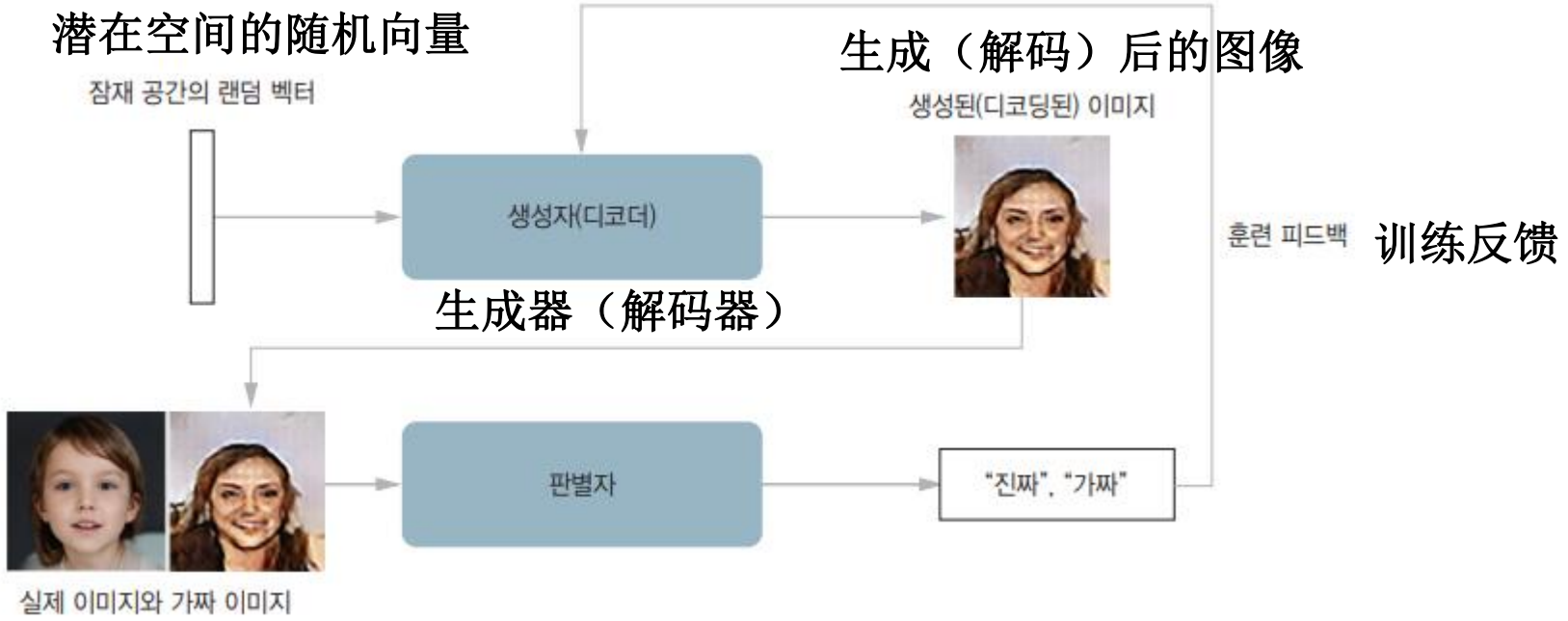
12.5 生成式对抗神经网络介绍

生成式对抗神经网络介绍

- 生成器网络的目标是欺骗判别器网络
- 随着训练不断进行，生成的图像会越来越逼真
- 生成器会产生几乎和真实图像无法区分的人工图像，使判别器无法判断两者的差异（如图 12-19）
- 同时，为了应对生成器越来越强，判别器会不断提高判断标准，努力区分真假图像
- 训练结束后，生成器可以将输入空间中的任意点转换为逼真的图像
- 与 VAE 不同，这种潜在空间 不保证具有有意义的结构
- 尤其是，这个潜在空间 并不是连续的

12.5 生成式对抗神经网络介绍

▼ 图 12-19 生成器将随机的潜在空间向量转换为图像，而判别器区分真实图像和生成图像。生成器通过训练来欺骗判别器。



真实图片和假图片 → 判别器 → “真假”

12.5 生成式对抗神经网络介绍

生成式对抗神经网络介绍

- 令人惊讶的是，GAN 是一个最小值并不固定的优化系统
- 与本书介绍的其他训练设置都不同
- 通常梯度下降是在固定的损失空间中往下走山坡
- 但在 GAN 中，每一步往下走的过程中，整个空间都会发生改变
- 优化过程不是寻找最小值，而是寻找两股力量之间的平衡点的动态系统
- 因此 GAN 以训练困难而闻名
- 为了构建 GAN，需要对模型结构和训练参数进行非常谨慎和大量的调整

12.5 生成式对抗神经网络介绍

▼ 图 12-20 从潜在空间采样得到的图像。使用 StyleGAN2 模型在 https://thispersondoesnotexist.com 上生成的图像（图片提供：该网站开发者 Phillip Wang，所使用的模型参考以下论文：https://arxiv.org/abs/1912.04958)

12.5 生成式对抗神经网络介绍

GAN 的实现方法

- 由于 GAN 是一项高级技术，像图 12-20 中那种由 StyleGAN2 这类结构生成的图像，其技术细节已经超出了本书的讨论范围。
- 在这个例子中，我们要实现的模型是 深度卷积 GAN（DCGAN）。
- 这是最基本的一种 GAN：生成器和判别器都是深度卷积网络。
- 使用由 20 万张名人脸组成的 CelebA（Large-scale CelebFaces Attributes）数据集（https://mmlab.ie.cuhk.edu.hk/projects/CelebA.html）来训练 GAN 模型。
- 为了提高训练速度，我们将图像大小改为 64×64。
- 模型将学习如何生成 64×64 尺寸的人脸图像。

12.5 生成式对抗神经网络介绍

GAN 的实现方法

- GAN 的结构如下：
  - 1. 生成器（generator）网络：
  - 将形状为 (latent_dim,) 的向量映射成形状为 (64, 64, 3) 的图片。
  - 2. 判别器（discriminator）网络：
  - 将形状为 (64, 64, 3) 的图片映射为一个二分类值，用来估计这张图片是真还是假。
  - 3. 构建一个把生成器和判别器连接起来的 gan 网络：
    - gan(x) = discriminator(generator(x))
    - gan 网络将潜在空间中的向量映射为判别器的输出评价。
    - 换句话说，判别器用来判断生成器解码出来的图片看起来是否逼真
  - 4. 使用带有“真 / 假”标签的真实图像和生成图像来训练判别器，与训练普通的图像分类模型完全相同。
  - 5若要训练生成器，需要使用 gan 模型的损失对生成器的权重进行反向传播。
    - 这意味着在每一步迭代中，都会调整生成器，使它生成的图片更容易被判别器分类为“真”的图像”。
    - 换句话说：
    - 训练生成器的目的，就是不断欺骗判别器。

12.5 生成式对抗神经网络介绍

训练方法

- 训练和调优 GAN 的过程是出了名的困难。
- 有一些实用技巧是必须了解的。
- 像深度学习的大多数方法一样，这更像是一门炼金术而不是科学。
- 这些技巧并不是基于理论的原则，而是通过经验发现的。
- 它们是在对实际现象进行直观理解之后得到验证的。
- 虽然并不是必须在所有问题中都使用这些技巧，但经验表明它们通常效果很好。

12.5 生成式对抗神经网络介绍

训练方法

- 以下是本节中用于实现 GAN 生成器和判别器的一些技巧
- 下面的列表不是所有的 GAN 技巧。
- 在 GAN 文献中可以找到更多方法。
  - 像在 VAE 的解码器中那样，在判别器中使用 stride 代替 pooling 来进行特征图下采样
  - 使用正态分布（高斯分布）而不是均匀分布在潜在空间中采样点
  - 随机性可以让模型更健壮
    - GAN 的训练会形成动态平衡，因此非常容易陷入某种不良状态。
    - 在训练时加入随机性有助于避免这种情况。
    - 例如，在判别器的标签中加入随机噪声。

12.5 生成式对抗神经网络介绍

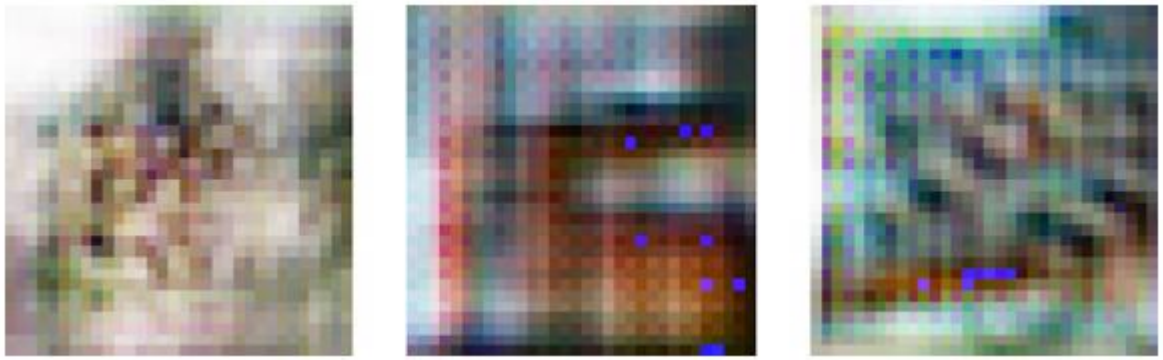
训练方法

- **稀疏梯度会妨碍 GAN 的训练**
  - 在深度学习中，梯度稀疏有时是件好事，但在 GAN 中却不是。
  - 会让梯度变稀疏的两件事：
  - 最大池化操作 ReLU 激活函数
  - 使用步幅卷积（strided convolution）进行下采样，而不要使用最大池化
  - 用 LeakyReLU 代替 ReLU
  - LeakyReLU 与 ReLU 类似，但允许少量负值激活，因此能减轻梯度稀疏问题

- 生成器有时无法均匀处理像素空间，因此在生成图像中会出现棋盘格（checkerboard）伪影（图 12-21）
- 为解决这个问题：
  - 在生成器和判别器中使用 Conv2DTranspose 或 Conv2D 时，选择卷积核大小可以被步幅整除。

12.5 生成式对抗神经网络介绍

▼ 图 12-21由于步幅（stride）和卷积核大小不匹配，无法在像素空间中进行均匀处理而产生的棋盘格纹理。这是 GAN 中发现的众多经验技巧之一。



12.5 生成式对抗神经网络介绍

准备 CelebA 数据集

- 可以从 CelebA 网站 (http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html) 手动下载该数据集。
- 如果使用 Google Colab，可以运行以下命令从 Google Drive 下载并解压数据。

코드 12-30 CelebA 데이터 내려받기

```
!mkdir celeba_gan -------- 작업 디렉터리를 만듭니다.
!gdown 1up5bN8LCE2vHigVY-Z9yY2_aKRW5jN_9 -O celeba_gan/data.zip --- gdown(코랩이 아닌 경우 설치가 필요합니다) 명령을 사용해서 압축된 데이터를 내려받습니다.
!unzip -qq celeba_gan/data.zip -d celeba_gan -------- 압축을 풉니다.
```

```
!mkdir celeba_gan          # 创建工作目录
!gdown 1up5bN8LCE2vHigVY-Z9yY2_aKRW5jN_9 -O celeba_gan/data.zip
                    # 使用 gdown（如果不是在 Colab，需要先安装 gdown）
                    # 下载压缩后的数据

!unzip -qq celeba_gan/data.zip -d celeba_gan
                    # 解压数据
```

12.5 生成式对抗神经网络介绍

准备 CelebA 数据集

- 如果目录中已经有解压后的图片，可以使用 image_dataset_from_directory 函数来创建数据集
- 因为不需要标签，只要图片即可，所以需要设置 label_mode=None

코드 12-31 이미지 디렉터리에 데이터셋을 만든다

```
from tensorflow import keras

dataset = keras.utils.image_dataset_from_directory(
    "celeba_gan",
    label_mode=None,        -------- 레이블 없이 이미지만 반환합니다.
    image_size=(64, 64),
    batch_size=32,          이미지를 64×64 크기로 변환할 때 가로세로 비율을 유지하면서
    smart_resize=True)      자르고 크기를 바꿉니다. 얼굴의 비율이 왜곡되면 안 됩니다!
```

```
from tensorflow import keras

dataset = keras.utils.image_dataset_from_directory(
  "celeba_gan",
  label_mode=None,      # 只返回图片，不返回标签
  image_size=(64, 64),  # 将图像转换成 64×64，同时保持宽高比例并裁切调整大小
  batch_size=32,
  smart_resize=True     # 保持图像比例裁切再缩放。脸的比例不能变形！
)
```

12.5 生成式对抗神经网络介绍

准备 CelebA 数据集

- 最后，将图像像素值调整到 [0-1] 区间

코드 12-32 픽셀 값 범위 바꾸기

```
dataset = dataset.map(lambda x: x / 255.)
```

- 可以使用代码 12-33 输出一些样本图像

코드 12-33 첫 번째 이미지 출력하기

```
import matplotlib.pyplot as plt

for x in dataset:
    plt.axis("off")
    plt.imshow((x.numpy() * 255).astype("int32")[0])
    break
```

12.5 生成式对抗神经网络介绍

判别器

- 首先，我们需要构建一个 discriminator（判别器）模型，它接收候选图像（真实图像或生成图像）作为输入，并将其分类为两类之一：（“生成的图像”或“来自训练集的图像”）。
- 在 GAN 中常见的问题之一是：生成器可能停止在只产生噪声般的图像。
- 在判别器中加入 Dropout 可以作为解决方法之一，因此我们在这里使用 Dropout 来改善这种情况。

12.5 生成式对抗神经网络介绍

判别器

코드 12-34 GAN 판별자 네트워크

```
from tensorflow.keras import layers

discriminator = keras.Sequential(
    [
        keras.Input(shape=(64, 64, 3)),
        layers.Conv2D(64, kernel_size=4, strides=2, padding="same"),
        layers.LeakyReLU(alpha=0.2),
        layers.Conv2D(128, kernel_size=4, strides=2, padding="same"),
        layers.LeakyReLU(alpha=0.2),
        layers.Conv2D(128, kernel_size=4, strides=2, padding="same"),
        layers.LeakyReLU(alpha=0.2),
        layers.Flatten(),
        layers.Dropout(0.2), -------- 드롭아웃 층: 중요합니다!
        layers.Dense(1, activation="sigmoid"),
    ],
    name="discriminator",
)
```

12.5 生成式对抗神经网络介绍

判别器

- 下面是判别器模型的 summary() 输出结果

```
>>> discriminator.summary()
Model: "discriminator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 64)        3136
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 32, 32, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 16, 16, 128)       131200
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 16, 16, 128)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 8, 8, 128)         262272
_________________________________________________________________
```

12.5 生成式对抗神经网络介绍

判别器

leaky_re_lu_2 (LeakyReLU)	(None, 8, 8, 128)	0
flatten (Flatten)	(None, 8192)	0
dropout (Dropout)	(None, 8192)	0
dense (Dense)	(None, 1)	8193

Total params: 404,801
Trainable params: 404,801
Non-trainable params: 0

12.5 生成式对抗神经网络介绍

生成器

- 接下来，我们将构建一个 generator（生成器）模型，它会把（训练过程中从潜在空间随机采样的）向量转换成候选图像

코드 12-35 GAN 생성자 네트워크

```
latent_dim = 128 ····· 잠재 공간은 128차원 벡터로 구성됩니다.
```

latent_dim = 128 # 潜在空间由 128 维向量构成

```
generator = keras.Sequential(
    [
        keras.Input(shape=(latent_dim,)),
        layers.Dense(8 * 8 * 128), ····· 판별자에 있는 Flatten 층의 출력 크기와 동일하게 지정합니다.
        layers.Reshape((8, 8, 128)), ····· 판별자의 Flatten 층 작업을 되돌립니다.
        layers.Conv2DTranspose(128, kernel_size=4, strides=2, padding="same"),
        layers.LeakyReLU(alpha=0.2),
        layers.Conv2DTranspose(256, kernel_size=4, strides=2, padding="same"),
        layers.LeakyReLU(alpha=0.2),
        layers.Conv2DTranspose(512, kernel_size=4, strides=2, padding="same"),
        layers.LeakyReLU(alpha=0.2),
```

판별자의 Conv2D 층 작업을 되돌립니다.

LeakyReLU 활성화 함수를 사용합니다.

```
generator = keras.Sequential(
[
  keras.Input(shape=(latent_dim,)),
  layers.Dense(8 * 8 * 128),    # 指定与判别器中 Flatten 层输出相同的尺寸
  layers.Reshape((8, 8, 128)),  # 恢复为判别器 Flatten 前的形状

  layers.Conv2DTranspose(128, kernel_size=4, strides=2, padding="same"),
  layers.LeakyReLU(alpha=0.2),

  layers.Conv2DTranspose(256, kernel_size=4, strides=2, padding="same"),
  layers.LeakyReLU(alpha=0.2),

  layers.Conv2DTranspose(512, kernel_size=4, strides=2, padding="same"),
  layers.LeakyReLU(alpha=0.2),
```

12.5 生成式对抗神经网络介绍

生成器

```
        layers.Conv2D(3, kernel_size=5, padding="same", activation="sigmoid"),
    ],
    name="generator",
)
```

출력 크기는 (28, 28, 1)이 됩니다.

12.5 生成式对抗神经网络介绍

生成器

- 下面是生成器模型的 summary() 输出结果

```
>>> generator.summary()
Model: "generator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 8192)              1056768
_________________________________________________________________
reshape (Reshape)            (None, 8, 8, 128)         0
_________________________________________________________________
conv2d_transpose (Conv2DTran (None, 16, 16, 128)       262272
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 16, 16, 128)       0
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 32, 32, 256)       524544
```

12.5 生成式对抗神经网络介绍

生成器

```
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 32, 32, 256)       0
_________________________________________________________________
conv2d_transpose_2 (Conv2DTr (None, 64, 64, 512)       2097664
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 64, 64, 512)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 64, 64, 3)         38403
=================================================================
Total params: 3,979,651
Trainable params: 3,979,651
Non-trainable params: 0
_________________________________________________________________
```

12.5 生成式对抗神经网络介绍

对抗网络

- 最后，将生成器和判别器连接起来构成一个 GAN 模型。
- 训练时，这个模型会让生成器不断提高欺骗判别器的能力。
- 这个模型将来自潜在空间的点映射为对 “真” 或 “假” 的分类结果。
- 在训练 GAN 时，使用的目标标签始终是 “真图片”。
- 训练 GAN 的目标，是让判别器在看到假图片时仍然预测为真，从而更新生成器的权重。

12.5 生成式对抗神经网络介绍

对抗网络

- 让我们总结一下训练循环的内容每一次训练循环中执行以下步骤：
  - 1. 从潜在空间中随机采样一个点（随机噪声）
  - 2. 使用这个随机噪声通过生成器生成一张图像
  - 3. 将生成的图像与真实图像混合
  - 4. 使用包含真假图像的混合数据以及对应标签来训练判别器
    - **标签为：“真”（真实图像）或“假”（生成图像）**
  - 5. 再次从潜在空间中随机采样一个新的点
  - 6. 使用这个随机向量来训练生成器
    - 这次所有标签都设为“真”
    - 目标是让判别器把生成的图像错误地预测为“真”
    - 最终的结果是：生成器被训练成可以欺骗判别器。

12.5 生成式对抗神经网络介绍

对抗网络

- 那就实际来实现一下吧。
- 和 VAE 示例一样，我们会对 Model 类进行子类化，并使用自定义的 train_step() 方法。
- 因为要使用两个优化器，所以会重写 compile() 方法，使其可以接收这两个优化器。

코드 12-36 GAN 모델

```
import tensorflow as tf

class GAN(keras.Model):
    def __init__(self, discriminator, generator, latent_dim):
        super().__init__()
        self.discriminator = discriminator
        self.generator = generator
        self.latent_dim = latent_dim
```

12.5 生成式对抗神经网络介绍

对抗网络

```
    self.d_loss_metric = keras.metrics.Mean(name="d_loss")
    self.g_loss_metric = keras.metrics.Mean(name="g_loss")
def compile(self, d_optimizer, g_optimizer, loss_fn):
    super(GAN, self).compile()
    self.d_optimizer = d_optimizer
    self.g_optimizer = g_optimizer
    self.loss_fn = loss_fn

@property
def metrics(self):
    return [self.d_loss_metric, self.g_loss_metric]

def train_step(self, real_images):
    batch_size = tf.shape(real_images)[0]
    random_latent_vectors = tf.random.normal(
        shape=(batch_size, self.latent_dim))
    generated_images = self.generator(random_latent_vectors)
```

훈련 에포크마다 2개의 손실을 추적하기 위한 속성을 설정합니다.

- 为了在每个训练 epoch 中跟踪两种损失而设置度量属性。
- 从潜在空间中随机采样点。
- 将这些随机采样的点解码成“假”图像。

잠재 공간에서 랜덤한 포인트를 샘플링합니다.

랜덤한 포인트를 가짜 이미지로 디코딩합니다.

12.5 生成式对抗神经网络介绍

对抗网络

```
combined_images = tf.concat([generated_images, real_images], axis=0)
labels = tf.concat(
    [tf.ones((batch_size, 1)), tf.zeros((batch_size, 1))],
    axis=0
)
labels += 0.05 * tf.random.uniform(tf.shape(labels))
with tf.GradientTape() as tape:
    predictions = self.discriminator(combined_images)
    d_loss = self.loss_fn(labels, predictions)
grads = tape.gradient(d_loss, self.discriminator.trainable_weights)
self.d_optimizer.apply_gradients(
    zip(grads, self.discriminator.trainable_weights)
)
```

진짜 이미지와 합칩니다.

진짜 이미지와 가짜 이미지의 레이블을 합칩니다.

레이블에 랜덤한 잡음을 추가합니다: 중요한 트릭입니다!

판별자를 훈련합니다.

- 将生成的假图像和真实图像在 batch 维度(axis=0)上拼接
- `labels` 也对应拼接：
  - 真图像 → label=1
  - 假图像 → label=0

给标签加入一点随机扰动，防止 discriminator 过早把标签记死。
这是训练 GAN 时一个非常经典的 trick，用来增强训练稳定性。

12.5 生成式对抗神经网络介绍

对抗网络

```
random_latent_vectors = tf.random.normal(
    shape=(batch_size, self.latent_dim))
misleading_labels = tf.zeros((batch_size, 1))

with tf.GradientTape() as tape:
    predictions = self.discriminator(
        self.generator(random_latent_vectors))
    g_loss = self.loss_fn(misleading_labels, predictions)
grads = tape.gradient(g_loss, self.generator.trainable_weights)

self.g_optimizer.apply_gradients(
    zip(grads, self.generator.trainable_weights))
self.d_loss_metric.update_state(d_loss)
self.g_loss_metric.update_state(g_loss)
return {"d_loss": self.d_loss_metric.result(),
        "g_loss": self.g_loss_metric.result()}
```

잠재 공간에서 랜덤한 포인트를 샘플링합니다.

모두 '진짜 이미지'라고 말하는 레이블을 만듭니다(사실 거짓말입니다!).

생성자를 훈련합니다.

12.5 生成式对抗神经网络介绍

对抗网络

- 在开始训练之前，先做一个用于监控结果的回调函数。
- 每个 epoch 结束时，用生成器生成若干张假图像并保存下来。

코드 12-37 훈련 과정 동안에 이미지를 생성하기 위한 콜백

```
class GANMonitor(keras.callbacks.Callback):
    def __init__(self, num_img=3, latent_dim=128):
        self.num_img = num_img
        self.latent_dim = latent_dim

    def on_epoch_end(self, epoch, logs=None):
        random_latent_vectors = tf.random.normal(
            shape=(self.num_img, self.latent_dim))
        generated_images = self.model.generator(random_latent_vectors)
        generated_images *= 255
        generated_images.numpy()
        for i in range(self.num_img):
            img = keras.utils.array_to_img(generated_images[i])
            img.save(f"generated_img_{epoch:03d}_{i}.png")
```

12.5 生成式对抗神经网络介绍

对抗网络

- 现在能开始训练了

코드 12-38 GAN 모델 컴파일하고 훈련하기

```
epochs = 100 -------- 20번째 에포크부터 흥미로운 결과를 얻기 시작할 것입니다.

gan = GAN(discriminator=discriminator, generator=generator,
          latent_dim=latent_dim)
gan.compile(
    d_optimizer=keras.optimizers.Adam(learning_rate=0.0001),
    g_optimizer=keras.optimizers.Adam(learning_rate=0.0001),
    loss_fn=keras.losses.BinaryCrossentropy(),
)

gan.fit(
    dataset, epochs=epochs,
    callbacks=[GANMonitor(num_img=10, latent_dim=latent_dim)]
)
```

12.5 生成式对抗神经网络介绍

对抗网络

- 训练时，你可能会看到对抗损失（GAN 的 전체 loss）急剧增大。
- 相反，判别器的损失会趋近于 0。
- 结果就是判别器的能力压倒了生成器。
- 出现这种情况时，可以尝试 降低判别器的学习率，并且 提高判别器中的 dropout 比例。

▼ 图 12-22：第 30 个 epoch 后生成的图像

12.5 生成式对抗神经网络介绍

总结

- GAN 是由生成器网络和判别器网络相互连接组成的
  - 判别器被训练来区分生成器输出和真实数据集中的真实图像
  - 生成器被训练来欺骗判别器
  - 令人惊讶的是：生成器从来没有直接见过训练集中的真实图像
  - 与数据相关的知识全部来自判别器

- GAN 很难训练
  - 训练 GAN 不是在固定的损失空间中进行的简单梯度下降，而是一个动态博弈过程
  - 要正确训练 GAN 通常需要大量经验技巧与调参

- GAN 可以生成非常逼真的图像
  - 与 VAE 不同，GAN 学到的潜在空间不一定具有连续且规则的结构
  - 使用潜在向量进行变换等任务时，不一定适合某些特定应用场景

12.6 总结

12.6总结

总结

- **Sequence-to-Sequence 模型**
  - 能够一次一步地生成序列数据
  - 不仅能用于文本生成，也能用于逐个音符生成音乐，或在任何时序数据生成任务中应用

- **DeepDream**
  - **通过在输入空间执行梯度上升，最大化卷积网络某些层的激活来工作**

- **风格迁移算法（Style Transfer）**
  - 通过梯度下降将内容图像与风格图像结合
  - 生成同时具有内容图像的高级特征和风格图像局部纹理特征的图像

- **VAE 与 GAN**
  - 这类模型可以学习图像的潜在空间，并从该空间采样来生成全新的图像
  - 还能利用 概念向量（concept vector） 来对图像进行变形或编辑