深度卷積 GAN
(Deep Convolutional GAN, DCGAN)

Chapter 4

Outline

4.1 卷積神經網路（CNN)

4.2 DCGAN 簡史

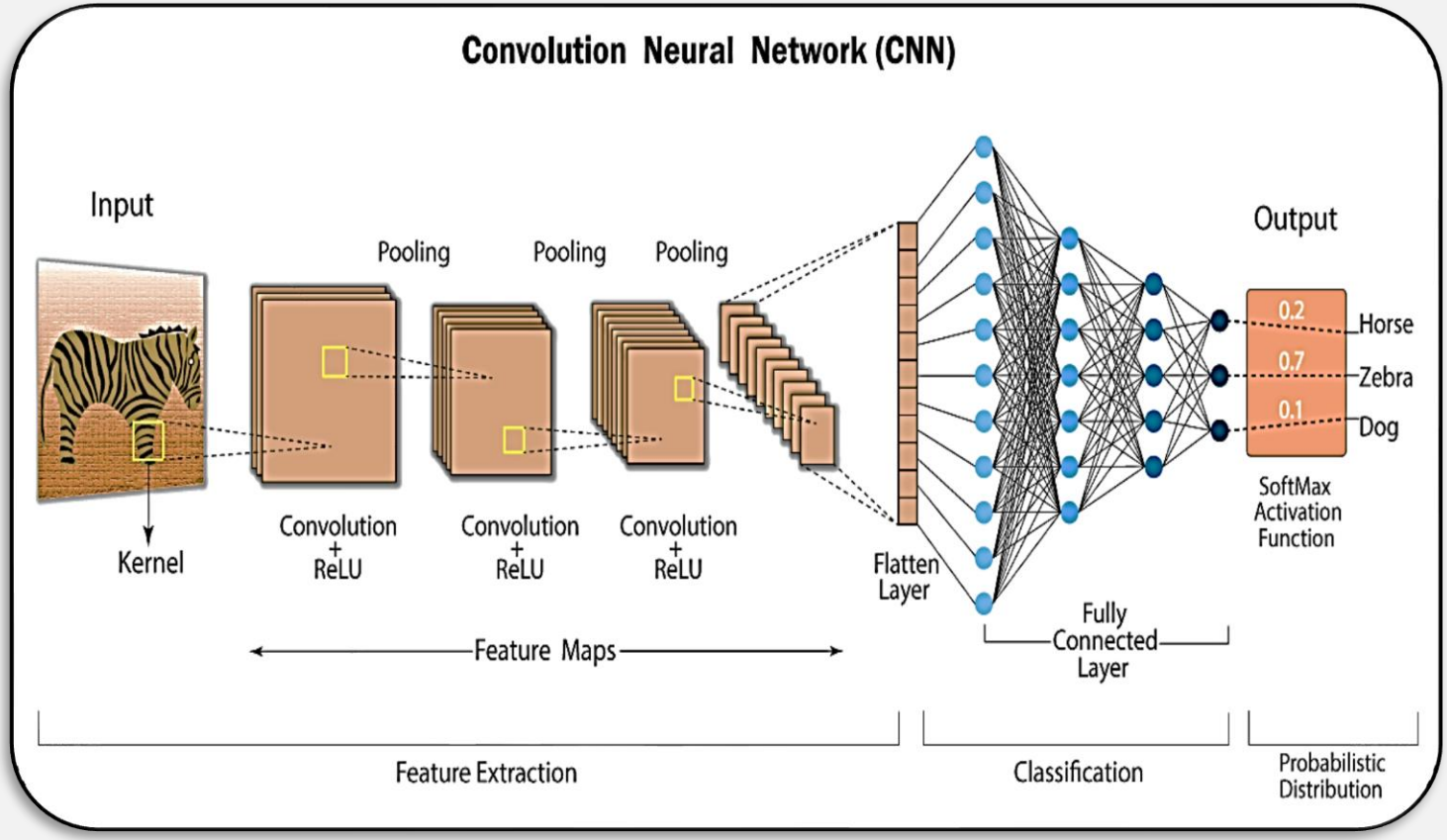
4.3 批次正規化（Batch normalization)

4.4 實例：用 DCGAN 生成手寫數字

4.5 重點整理

4.1 卷積神經網路（CNN）

卷積神經網絡 (Convolutional Neural Network, CNN)



結 構

特徵擷取

→ 捲積層 (Convolution)

→ 池化層 (Pooling)

分類

→ 全連接層 (Fully Connected Layer)

應 用

- 影像分類
- 語音辨識
- 文本分析

4.1 卷積神經網路（CNN）

卷積層 Convolution Layer

目的

提取圖像特徵 ⇨ 縮短運算時間

方法

卷積核：二維矩陣

由許多不同的kernel在輸入圖片進行卷積運算
以得到激活值，構成 feature map

利用filter在輸入圖片上滑動
並持續進行矩陣內積

特色

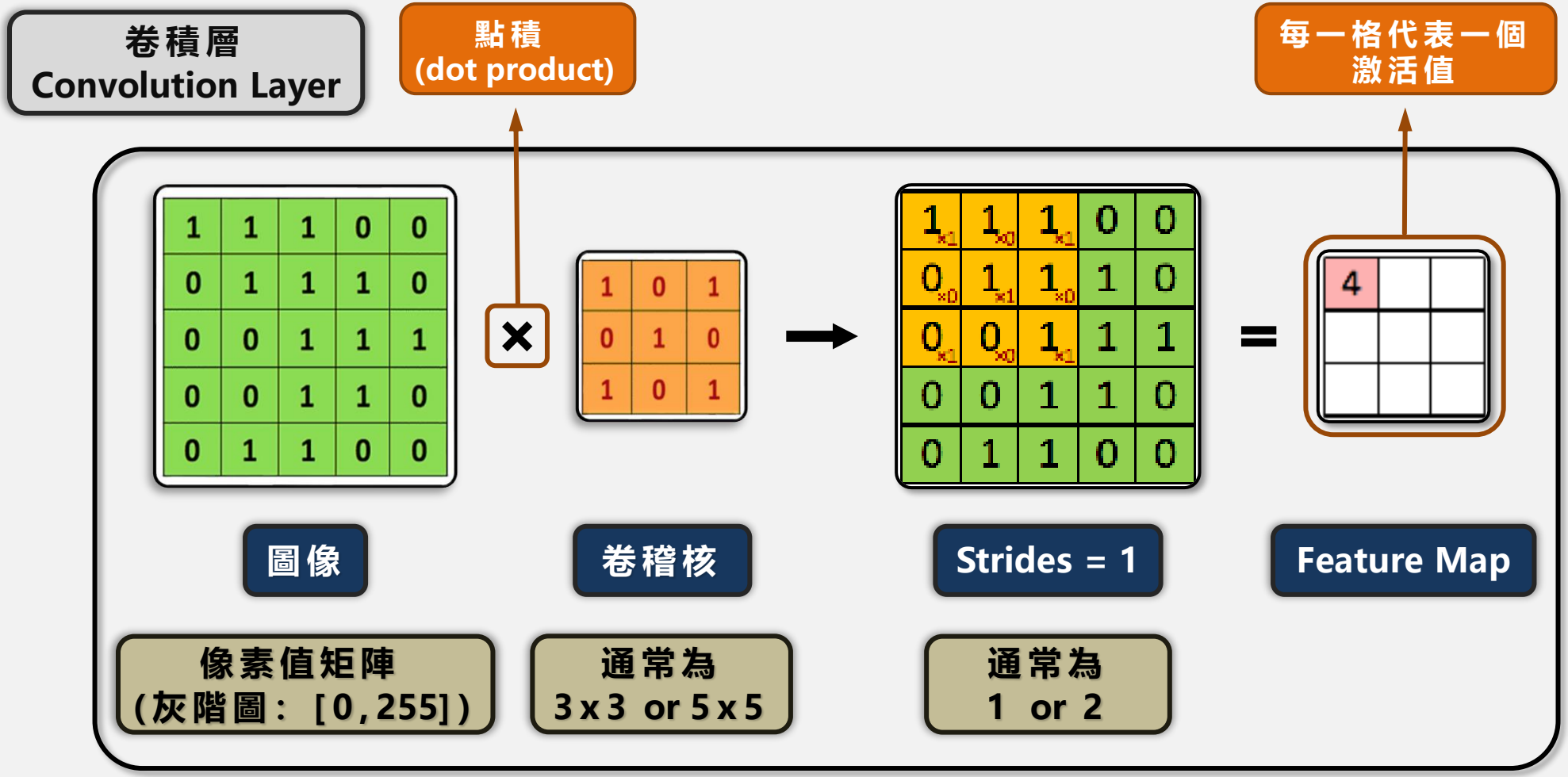
- 保留圖片空間結構，並從中提取特徵
- 利用權重共享減少參數
- 藉由步長(Strides)、填充(Padding)
  控制圖像長寬

滑動filter

filter (濾鏡)

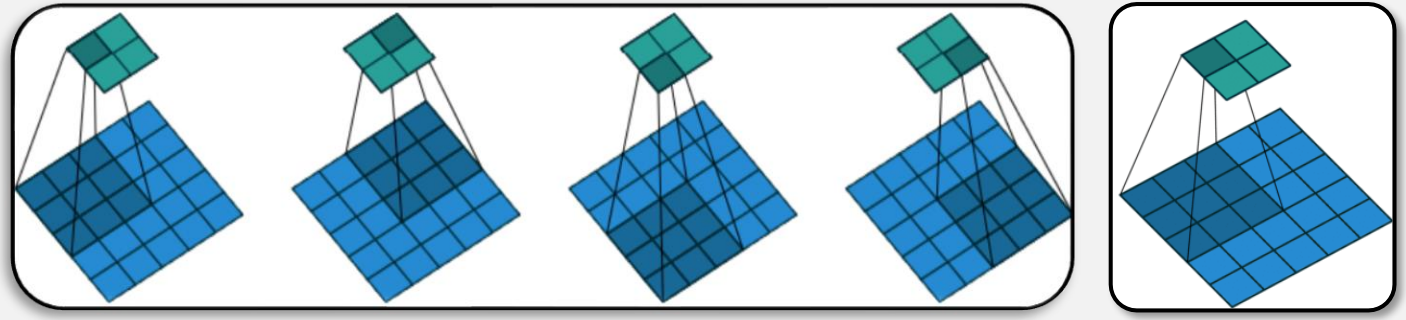
- 特徵篩選器，類似權重
- 多個 kernal 組成的三維矩陣
  (多出的一維是channel)

4.1 卷積神經網路（CNN）

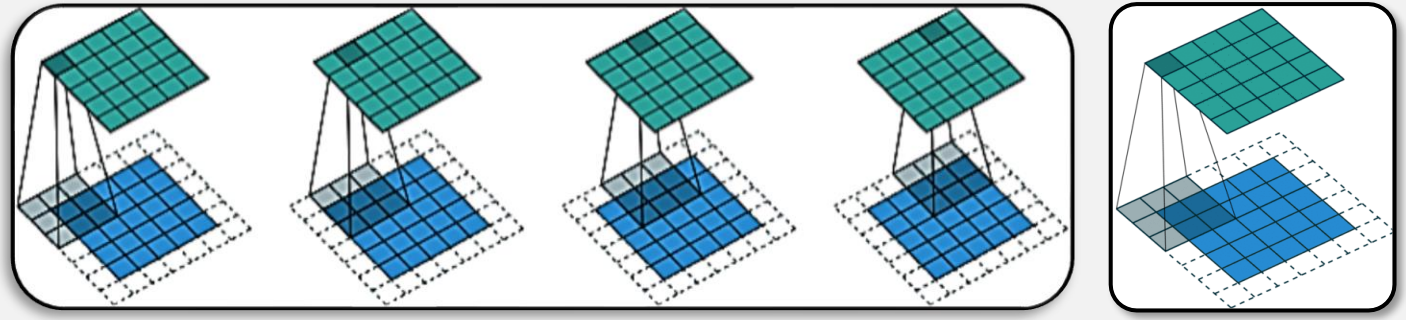


4.1 卷積神經網路（CNN）

卷積層
Convolution Layer



i = 5, k = 3, s = 2, p = 0



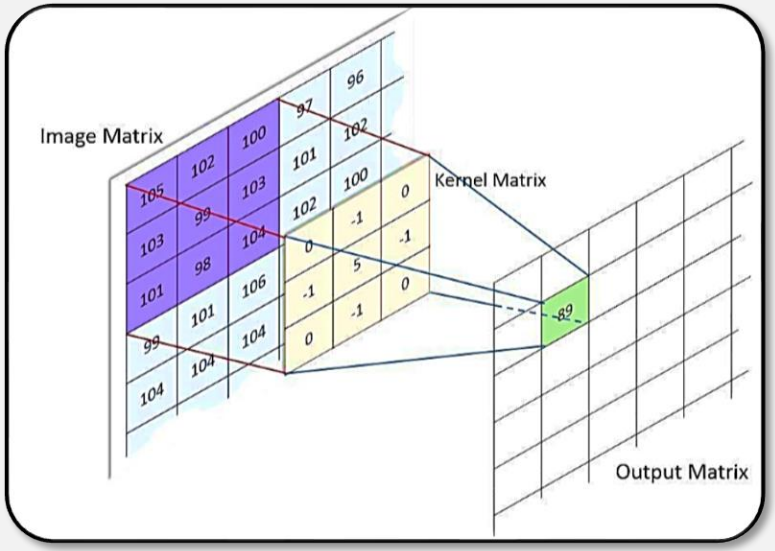
i = 5, k = 3, s = 1, p = 1

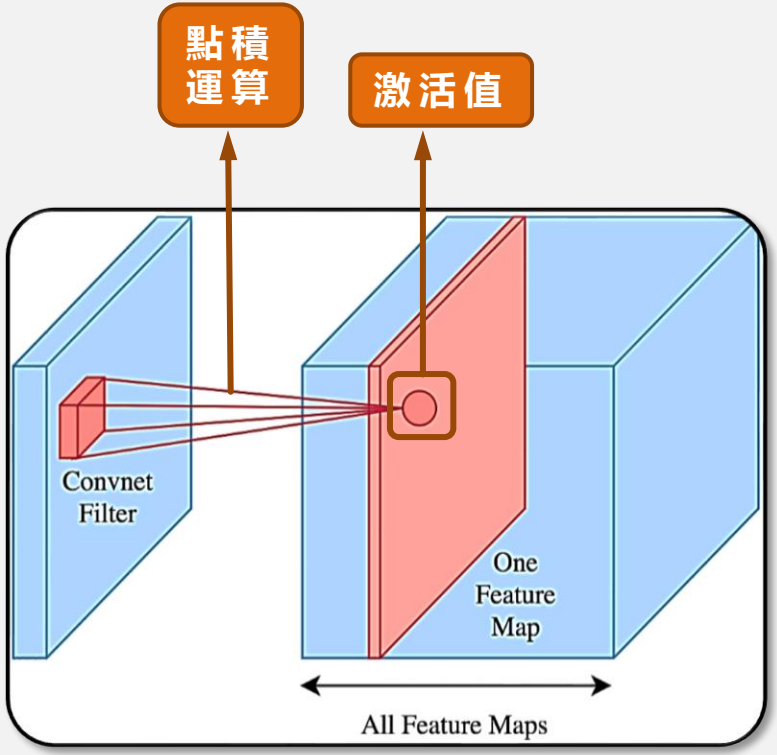
填充(Padding)

1. = valid
   → 不進行任何處理
   → feature map 依 filter、stride 縮小
2. = same
   → 把超出邊界的值補零
   → feature map 大小不變

4.1 卷積神經網路（CNN）

卷積層
Convolution Layer





濾鏡每走一步產生一個值 » 走完全部得到一張特徵圖 » 輸出：所有特徵圖相疊

輸出資料的深度 = 所有特徵圖的層數 = 濾鏡數

4.1 卷積神經網路（CNN）

池化層
Pooling Layer

目 的

- 壓縮圖片
- 幫助 CNN 判斷圖片中是否包含某項特徵

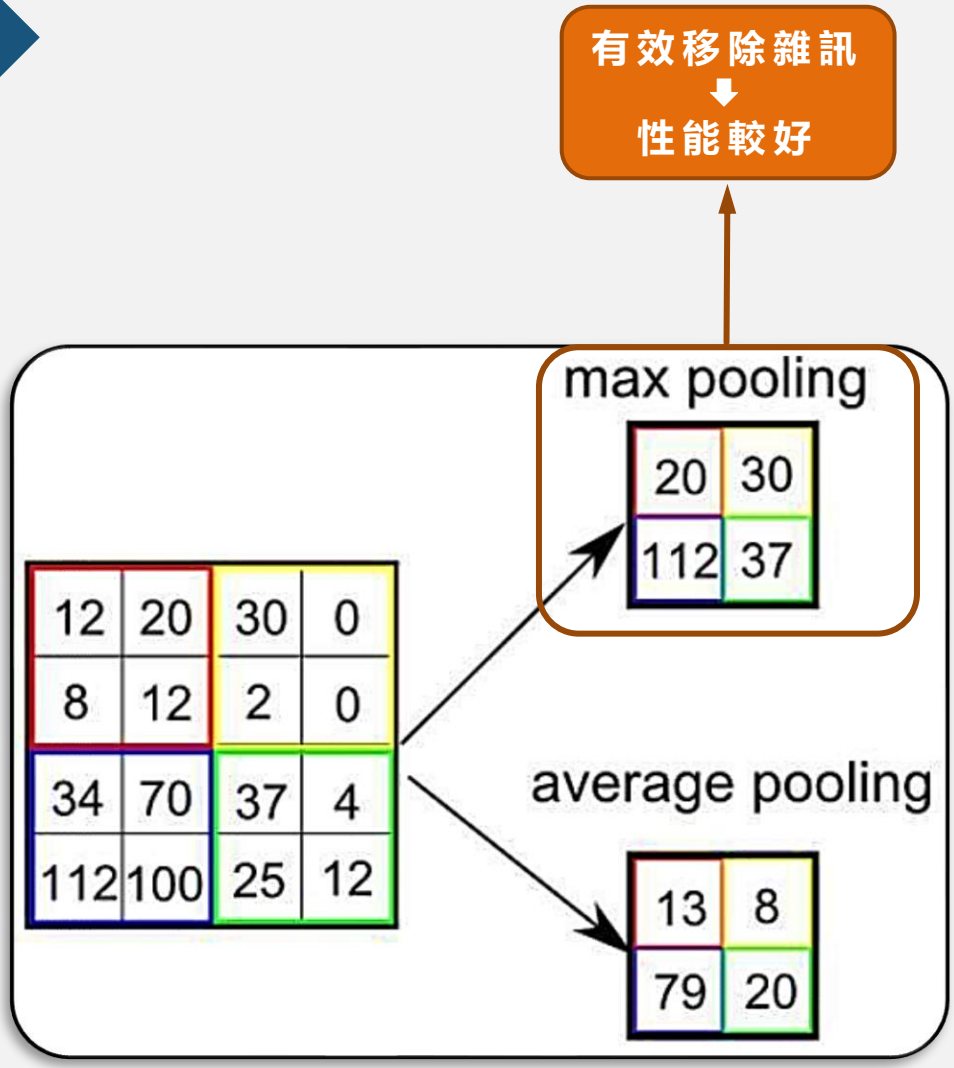
方 法

滑動視窗

- 最大池化法 (Max Pooling)
- 平均池化法 (Mean/Average Pooling)

特 色

- 保留重要資訊 ⇨ 特徵不變性
- 提高運算速度



4.1 卷積神經網路（CNN）

全連接層
Fully Connected Layer

目 的

- 使用卷積、池化後的特徵訓練數據集
- 如同分類器，將輸入圖片進行**分類**

方 法

- 將卷積、池化後的特徵**展平**(flatten)作為輸入
- 加入激活函數，使 NN 轉換成**非線性**

特 色

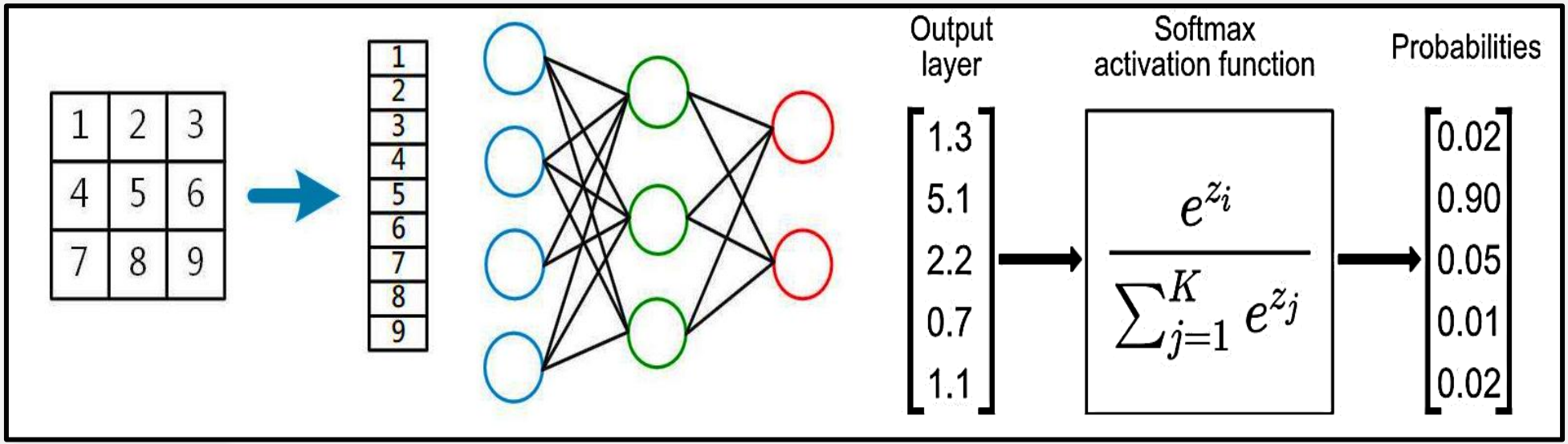
使用 **softmax** 作為輸出層的激活函數

輸出值範圍[0,1]

將資料轉換成一維
使神經網路可接收
到全部資料

4.1 卷積神經網路（CNN）

全連接層
Fully Connected Layer

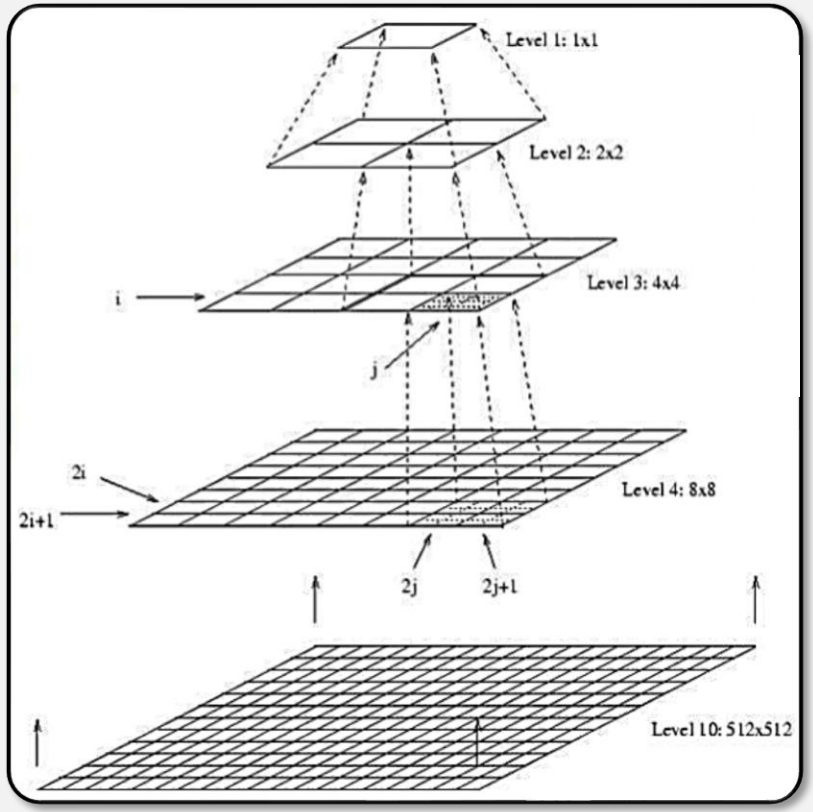


Pooled Feature Map » **Flatten** » **FCL** » **以 Softmax 輸出**

分子：原輸出機率
分母：所有輸出機率加總

4.2 DCGAN 簡史

4.2 DCGAN 簡史



LAPGAN
(Laplacian Pyramid GAN)

目 的

生成由粗糙到精緻的圖像，從而生成高辨別性的圖像

方 法

1. 下採樣：高斯金字塔
2. 上採樣：拉普拉斯金字塔 (Laplacian Pyramid)

特 色

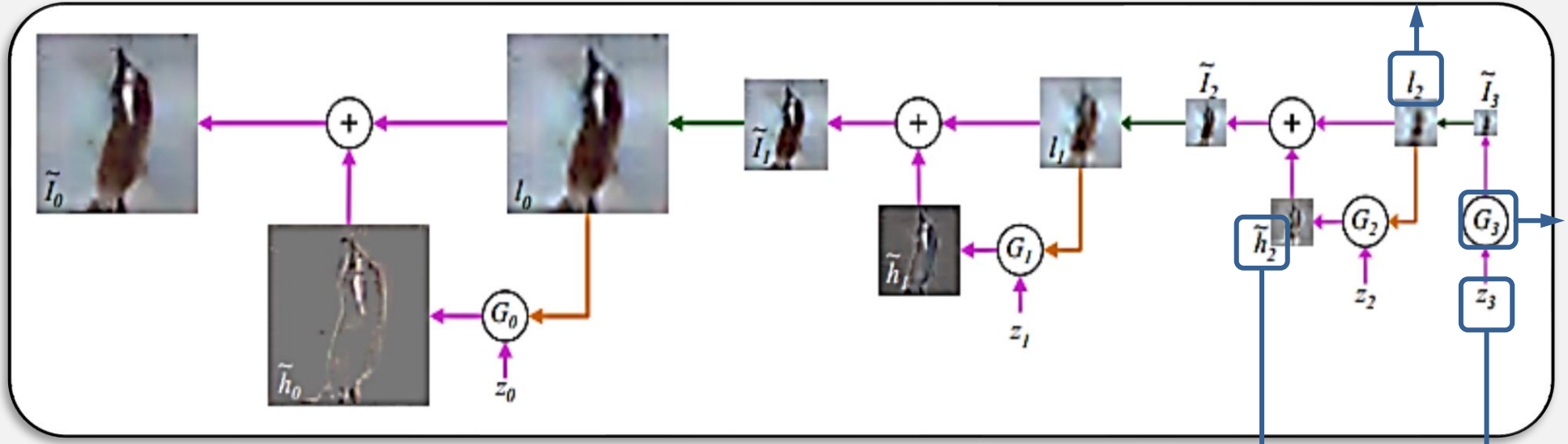
減少了每一次GAN 需學習的內容
⇨ 進而提升 GAN 的學習能力

缺 點

程序繁瑣、計算耗時

4.2 DCGAN 簡史

LAPGAN 模型採樣程序



Note

1. 由右到左
2. 綠色箭頭(→) 代表 “上採樣”

4.2 DCGAN 簡史

深度卷積對抗式生成網路
(Deep Convolutional GAN, DCGAN)

目 的

生成真偽難辨的圖像

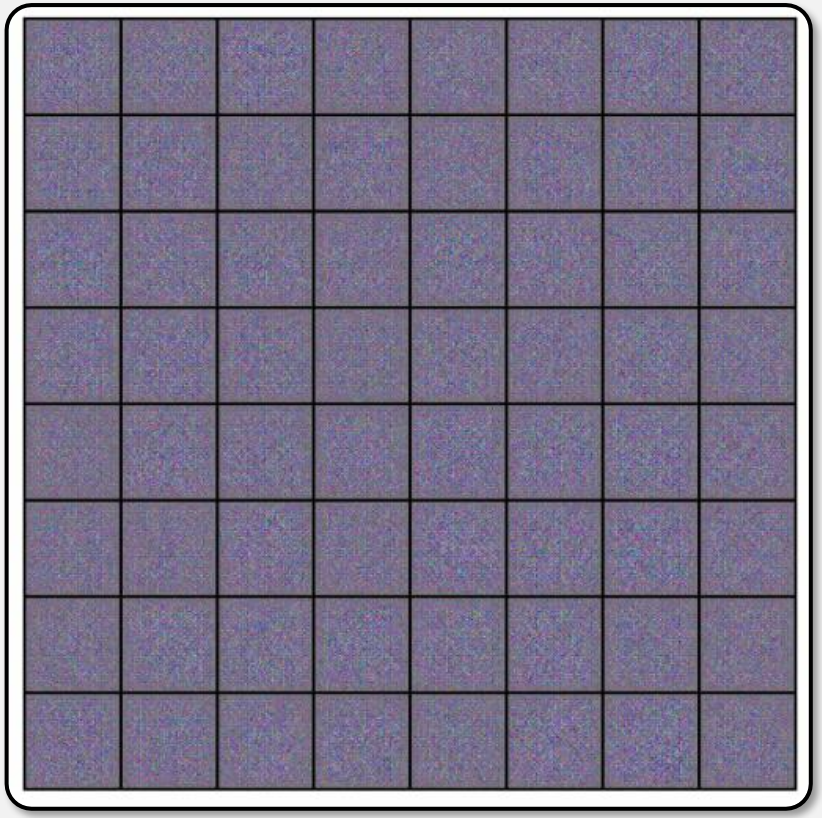
方 法

將轉置卷積(Transposed convolution) 生成的圖像放入 GAN 模型中執行，並重複訓練過程

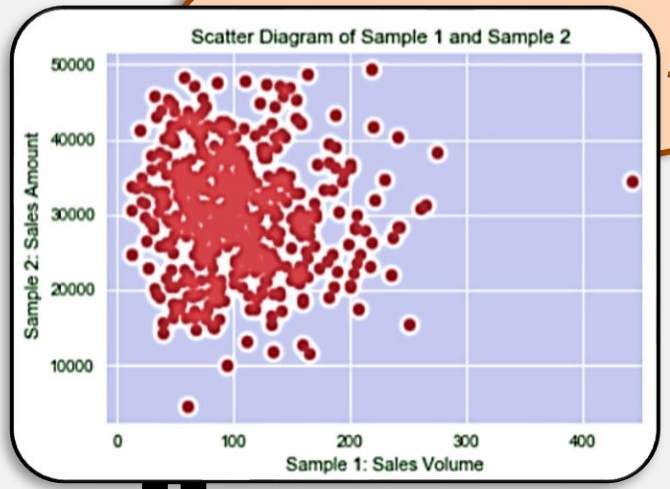
DCGAN 與 CNN 的差異

1. 生成器網路(G)：使用**轉置卷積**進行上取樣
   鑑別器網路(D)：加入stride的卷積代替池化
2. 使用**批次正規化**(Batch Normalization)

DCGAN 生成人臉訓練過程



4.3 批次正規化

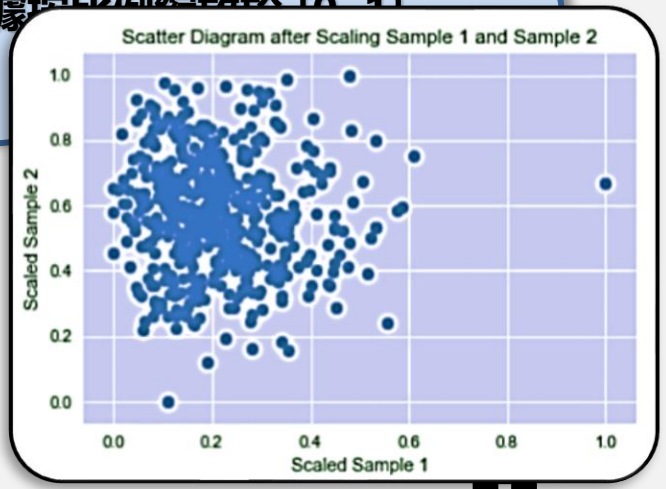


Sample 1 → 銷售量

Sample 2 → 銷售額

兩組資料同時轉換成純量
縮放於 [0,1] 區間中

正規化是將原始資料的數
據按比例縮放於 [0, 1]



相同基準 ⇨ 進行比較分析

4.3 批次正規化

正規化 (Normalization)

目的

避免訓練因某特徵劇烈變化而失去穩定性

方法

將原始資料的數據按比例縮放於原點附近

公式：$\hat{x} = \frac{\text{每筆資料}\,(x) - \text{平均值}\,(\mu)}{\text{標準差}\,(\sigma)}$

優點

不同特徵數值縮放成相同比例即可進行合理比較

批次正規化 (Batch Normalization)

目的

訓練期間各層輸入值資料分佈狀況不斷發生變化

- 減緩梯度消失 ⇨ 穩定訓練過程
- 解決協變量偏移(covariate shift)的問題

方法

每次只針對當前批次

將資料以批次為單位做正規化處理變異數

公式：$\hat{x} = \frac{\text{當批資料}\,(x) - \text{平均值}\,(\mu)}{\sqrt{\text{變異數}(\sigma^2) + \text{正值常數}\,(\varepsilon)}}$ $(\blacktriangle\ \sqrt{\sigma^2} = \sigma)$

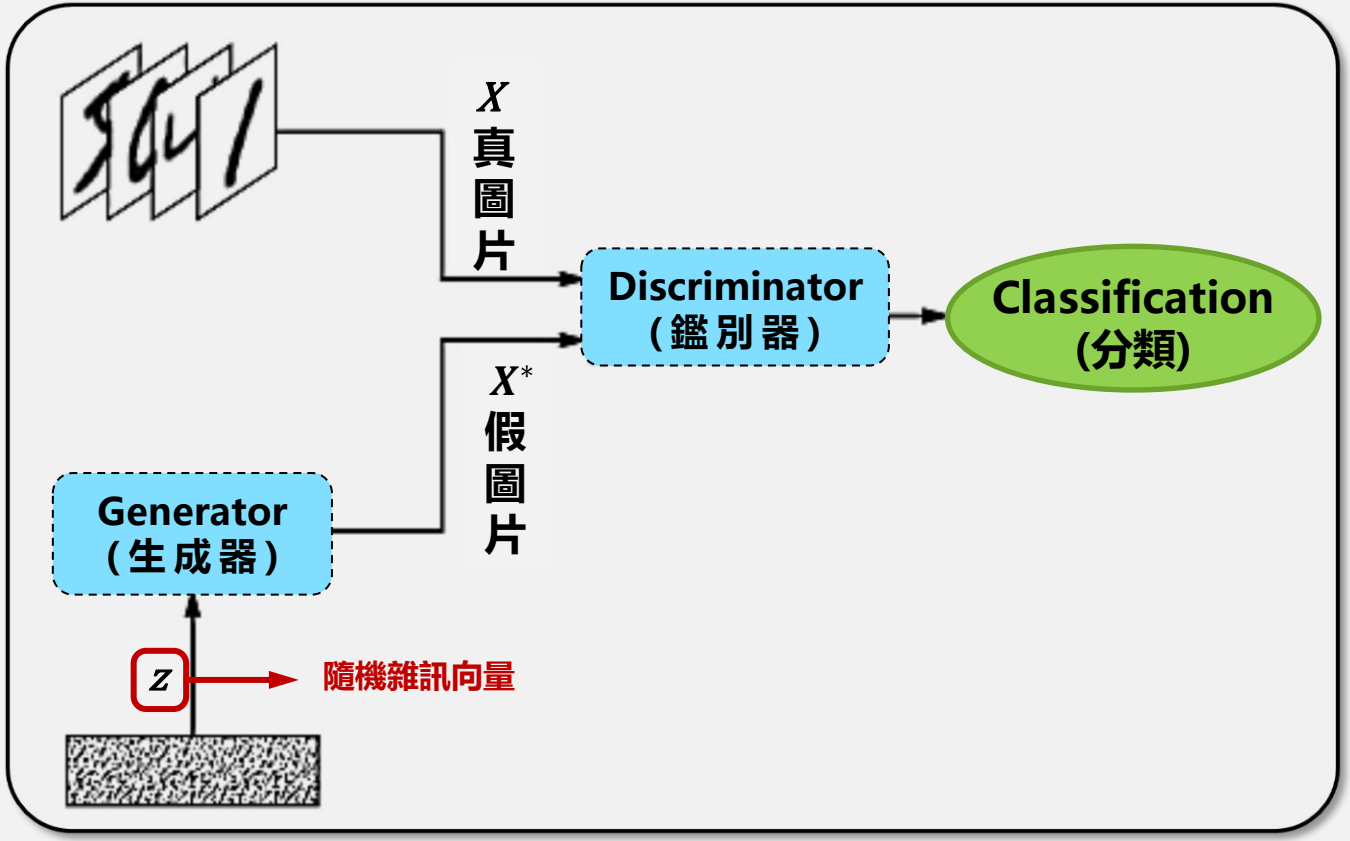
$y = \text{平均值}\,(\gamma)\hat{x} + \text{標準差}\,(\beta)$

優點

避免輸入值局限於 $\mu = 0$、$\sigma = 1$

- 加速模型收斂
- 降低該層參數調整對後層輸入資料的分佈影響

4.4 實例：用 DCGAN 生成手寫數字



DCGAN 架構圖

4.4 實例：用 DCGAN 生成手寫數字

匯入模組

```
%matplotlib inline

Import matplotlib.pyplot as plt
Import numpy as np

from keras.datasets import mnist
from keras.layers import (Activation, BatchNormalization,
                          Dense, Dropout, Flatten, Reshape)
from keras.layers.advanced_activations import LeakyReLU
from keras.layers.convolutional import Conv2D,
Conv2DTranspose
from keras.models import Sequential
from keras.optimizers import Adam
```

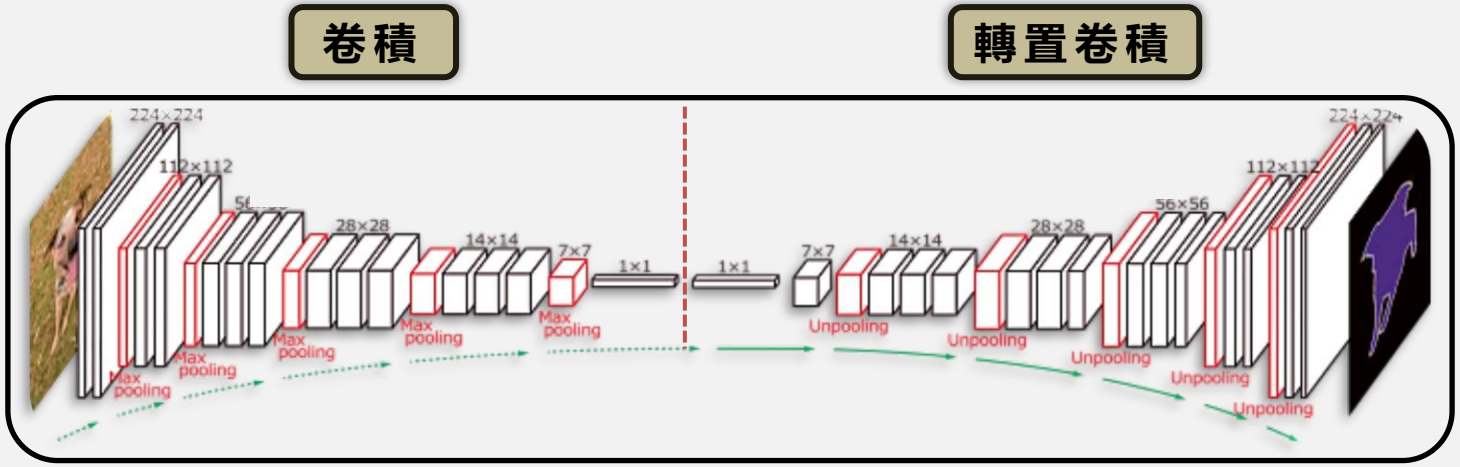
4.4 實例：用 DCGAN 生成手寫數字

設定模型輸入層維度

```
img_rows = 28
img_cols = 28
channels = 1  # MNIST圖像皆為灰階

img_shape = (img_rows, img_cols, channels)  # 輸入圖像的維度

z_dim = 100  # 指定雜訊向量的長度
```

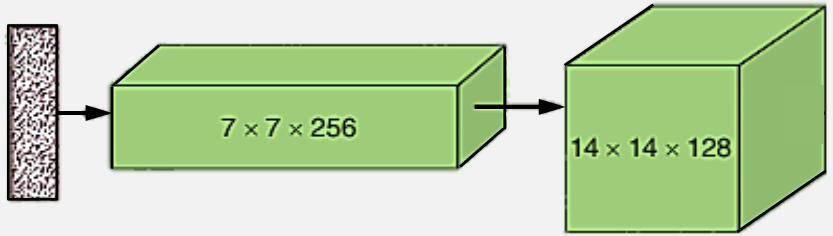
4.4 實例：用 DCGAN 生成手寫數字

	卷積 (Convolution)	轉置卷積 (Transposed Convolution)
操作方法	將圖片轉換成向量	將向量轉換成圖片
目的	增加深度，縮短寬度、高度	減少深度，增加寬度、高度
神經網路	鑑別器 (Discriminator)	生成器 (Generator)



4.4 實例：用 DCGAN 生成手寫數字

設計 DCGAN 生成器




```
def build_generator(z_dim) :
    model = Sequential()
    # 建立全連接層 ⇨ 拓展訊向量
    model.add(Dense(256 * 7 * 7, input_dim=z_dim))
    model.add(Reshape((7, 7, 256)))
    # 擴展維度：7x7x256 →14x14x128
    model.add(Conv2DTranspose(128, kernel_size=3, strides=2, padding='same'))
    # 進行批次正規化
    model.add(BatchNormalization())
    # 隱藏層激活函數：Leaky ReLU
    model.add(LeakyReLU(alpha=0.01))
```

NOTE

《轉置卷積》

→ 減少深度，增加寬度、高度

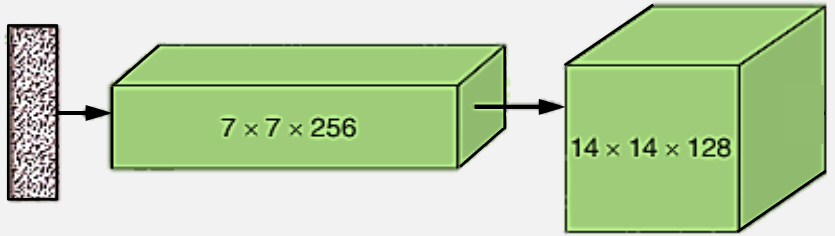
→ 將向量轉換成圖片

目的

→ 降低各參數初始值對訓練的影響

4.4 實例：用 DCGAN 生成手寫數字

設計 DCGAN 生成器

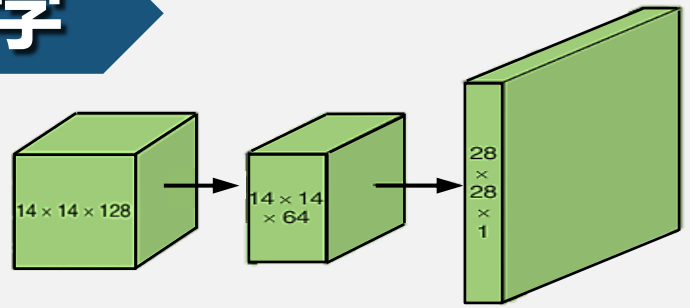



```
def build_generator(z_dim) :
    model = Sequential()
    # 建立全連接層 ⇨ 拓展訊向量
    model.add(Dense(256 * 7 * 7, input_dim=z_dim))
    model.add(Reshape((7, 7, 256)))

    # 擴展維度：7x7x256 →14x14x128
    model.add(Conv2DTranspose(128, kernel_size=3, strides=2, padding='same'))

    # 進行批次正規化
    model.add(BatchNormalization())

    # 隱藏層激活函數：Leaky ReLU
    model.add(LeakyReLU(alpha=0.01))
```

目的：決定有多少資料要進行池化
通常為奇數便於尋找中心點⇨更容易padding

目的：使輸入輸出圖片大小保持一致
方法：在原始圖片邊界補零

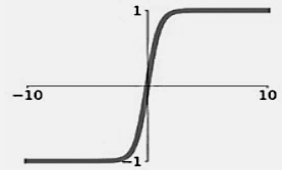
4.4 實例：用 DCGAN 生成手寫數字



設計 DCGAN 生成器

```
# 擴展維度： 14x14x128 →14x14x64
model.add(Conv2DTranspose(64, kernel_size=3, strides=1, padding='same'))

# 進行批次正規化
model.add(BatchNormalization())

# 隱藏層激活函數：Leaky ReLU
model.add(LeakyReLU(alpha=0.01))

# 擴展維度： 14x14x64 → 28x28x1
model.add(Conv2DTranspose(1, kernel_size=3, strides=2, padding='same'))

# 輸出層激活函數：tanh
model.add(Activation('tanh'))

return model
```

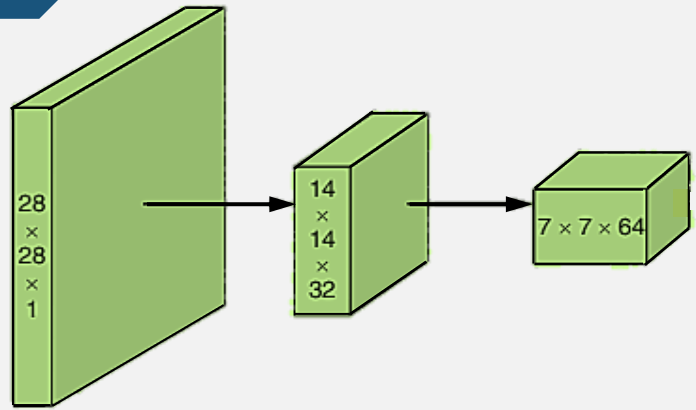
寬高不變



tanh

- 輸出值範圍：[-1,1]
- 產生較清晰的圖片

4.4 實例：用 DCGAN 生成手寫數字



設計 DCGAN 鑑別器

```
def build_discriminator (img_shape) :
    model = Sequential()
    # 擴展維度： 28x28x1 →14x14x32
    model.add( Conv2D(32, kernel_size=3, strides=2, input_shape=img_shape, padding='same'))
    # 隱藏層激活函數：Leaky ReLU
    model.add(LeakyReLU(alpha=0.01))
    # 擴展維度： 14x14x32 →7x7x64
    model.add( Conv2D(64, kernel_size=3, strides=2, input_shape=img_shape, padding='same'))
    # 進行批次正規化
    model.add(BatchNormalization())
    # 隱藏層激活函數：Leaky ReLU
    model.add(LeakyReLU(alpha=0.01))
```

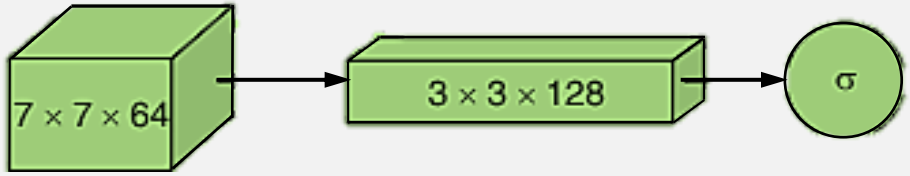
NOTE

《卷積》

→ 增加深度，縮短寬度、高度

4.4 實例：用 DCGAN 生成手寫數字

設計 DCGAN 鑑別器

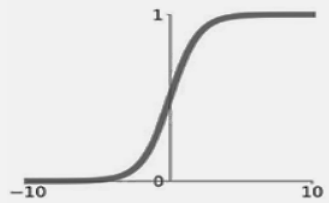



```
# 擴展維度： 7x7x64 →4x4x128
model.add( Conv2D(32, kernel_size=3, strides=2, input_shape=img_shape, padding='same'))
# 進行批次正規化
model.add(BatchNormalization())
# 隱藏層激活函數：Leaky ReLU
model.add(LeakyReLU(alpha=0.01))
```

`model.add(Flatten())` → **將輸入圖片展平成 2048 像素的一維矩陣**

`model.add(Dense(1, activation='sigmoid' ))`

return model



- **輸出值範圍：[0,1]**
- **代表輸入圖片為真的機率**

4.4 實例：用 DCGAN 生成手寫數字

建立與編譯 DCGAN

```
#定義GAN函式

def build_gan(generator, discriminator):

    model = Sequential()

    model.add(generator)
    model.add(discriminator)          # 結合生成器與鑑別器

    return model
```

4.4 實例：用 DCGAN 生成手寫數字

建立與編譯 DCGAN

```
#建立和編譯鑑別器
discriminator = build_discriminator(img_shape)
discriminator.compile(loss= 'binary_crossentropy' , optimizer=Adam(), metrics=['accuracy'])

#鎖定鑑別器參數以便訓練生成器
discriminator.trainable = False

#建立生成器
generator = build_generator(img_shape, z_dim)

#建立和編譯GAN模型用以訓練生成器
gan = build_gan(generator, discriminator)
gan.compile(loss='binary_crossentropy', optimizer=Adam())
```

二元交叉熵

- 評估二分類時預測與實際損失的差距
- 越小越好

4.4 實例：用 DCGAN 生成手寫數字

撰寫 DCGAN 訓練函式

```
# 定義用來儲存損失率、準確率及迭代次數的陣列
losses = []
accuracies = []
iteration_checkpoints = []

def train(iterations, batch_size, sample_interval):

    (X_train, _), (_, _) = mnist.load_data() # 載入MNIST資料集

    X_train = X_train / 127.5 - 1.0 # 將灰階像素質由[0, 255] 轉換為[-1, 1]
    X_train = np.expand_dims(X_train, axis=3)

    real = np.ones((batch_size, 1)) # 將真圖片標籤設為1

    fake = np.zeros((batch_size, 1)) # 將假圖片標籤設為0
```

Note

(X_train, **Y_train**), (**X_test**, **Y_test**)

將此三個不會用到的變數以底線表示

生成器輸出層激勵函數：tanh

4.4 實例：用 DCGAN 生成手寫數字

撰寫 DCGAN 訓練函式

```
for iteration in range(iterations):

    #隨機挑一批真圖片
    idx = np.random.randint(0, X_train.shape[0], batch_size)
    imgs = X_train[idx]

    #生成一批假圖片
    z = np.random.normal(0, 1, (batch_size, 100))
    gen_imgs = generator.predict(z)

    #訓練鑑別器
    d_loss_real = discriminator.train_on_batch(imgs, real)
    d_loss_fake = discriminator.train_on_batch(gen_imgs, fake)
    d_loss, accuracy = 0.5 * np.add(d_loss_real, d_loss_fake)
```

從標準常態分布中取樣

鑑別器較佳的輸出

- 真陽性
- 真陰性

4.4 實例：用 DCGAN 生成手寫數字

撰寫 DCGAN 訓練函式

```
z = np.random.normal(0, 1, (batch_size, 100))

#訓練生成器
g_loss = gan.train_on_batch(z, real)

if (iteration + 1) % sample_interval == 0:

    #紀錄損失率及準確率
    losses.append((d_loss, g_loss))
    accuracies.append(100.0 * accuracy)
    iteration_checkpoints.append(iteration + 1)

    #顯示訓練過程
    print( "%d [D loss: %f, acc.: %.2f%%] [G loss: %f]" %
           (iteration + 1, d_loss, 100.0 * accuracy, g_loss))

    #生成並顯示圖片
    sample_images(generator)
```

從標準常態分布中取樣

生成器的目的

生成出近似真圖片的擬真樣本

4.4 實例：用 DCGAN 生成手寫數字

生成並顯示圖片

```
def sample_images(generator, image_grid_rows=4, image_grid_columns=4):
    z = np.random.normal(0, 1, (image_grid_rows * image_grid_columns, z_dim)) # 取一組隨機雜訊
    gen_imgs = generator.predict(z) # 用這組隨機雜訊生成圖片
    gen_imgs = 0.5 * gen_imgs + 0.5 # 將圖片像素質由[-1, 1] 轉換為 [0, 1]
    fig, axs = plt.subplots(image_grid_rows, image_grid_columns, figsize=(4, 4),
                            sharey=True, sharex=True) # 設定圖片網格
    cnt = 0
    for i in range(image_grid_rows):
        for j in range(image_grid_columns):
            axs[i, j].imshow(gen_imgs[cnt, :, :, 0], cmap= 'gray' )
            axs[i, j].axis('off' )
            cnt += 1
```

使用4×4的子表圖來顯示16張圖片

4.4 實例：用 DCGAN 生成手寫數字

訓練模型

```
# 設定超參數
iterations = 20000
batch_size = 128
sample_interval = 1000

# 用指定的迭代次數訓練GAN
train(iterations, batch_size, sample_interval)
```

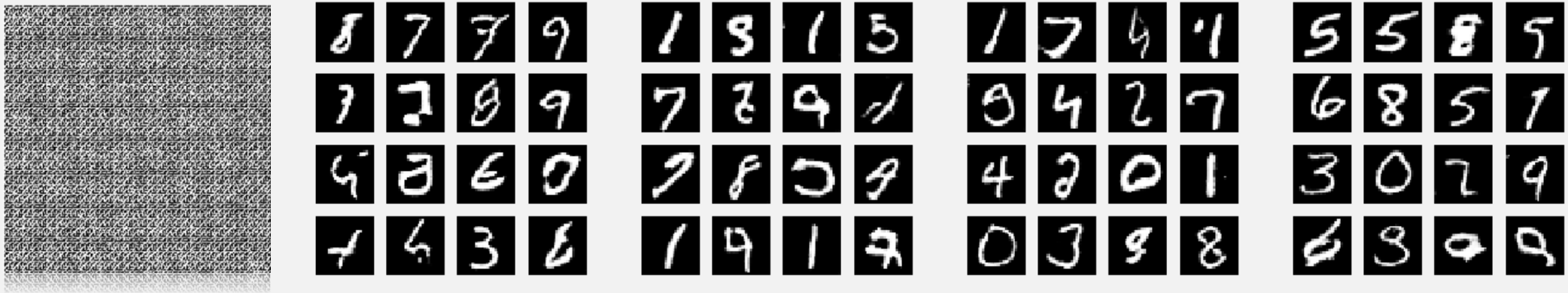
4.4 實例：用 DCGAN 生成手寫數字

訓練結果

```
1000 [D loss: 0.235200, acc.:  92.58%] [G loss: 4.601653]
2000 [D loss: 0.034661, acc.: 100.00%] [G loss: 3.741243]
3000 [D loss: 0.042863, acc.:  99.61%] [G loss: 5.250259]
4000 [D loss: 0.047563, acc.: 100.00%] [G loss: 4.974766]
5000 [D loss: 0.038144, acc.:  99.61%] [G loss: 3.358321]
6000 [D loss: 0.063527, acc.:  99.61%] [G loss: 5.743986]
7000 [D loss: 0.085559, acc.:  97.66%] [G loss: 4.646115]
8000 [D loss: 0.060515, acc.: 100.00%] [G loss: 4.075248]
9000 [D loss: 0.029112, acc.:  99.61%] [G loss: 5.616620]
10000 [D loss: 0.019951, acc.: 100.00%] [G loss: 7.076608]
11000 [D loss: 0.045042, acc.:  99.61%] [G loss: 4.535901]
12000 [D loss: 0.123534, acc.:  95.70%] [G loss: 2.141859]
13000 [D loss: 0.274708, acc.:  86.72%] [G loss: 3.151012]
14000 [D loss: 0.030535, acc.:  99.61%] [G loss: 3.976374]
15000 [D loss: 0.037902, acc.:  99.61%] [G loss: 5.177255]
16000 [D loss: 0.033344, acc.:  99.61%] [G loss: 5.022399]
17000 [D loss: 0.010645, acc.: 100.00%] [G loss: 6.817470]
18000 [D loss: 0.067494, acc.:  98.83%] [G loss: 5.317701]
19000 [D loss: 0.143730, acc.:  94.92%] [G loss: 6.451805]
20000 [D loss: 0.057865, acc.:  98.44%] [G loss: 4.254002]
```

4.4 實例：用 DCGAN 生成手寫數字

輸出圖片



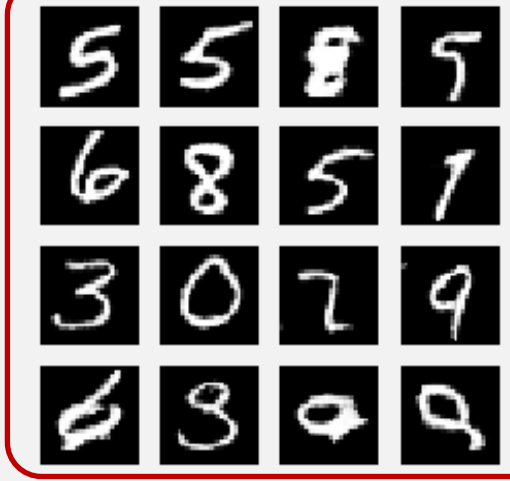
生成圖片變化　第5批　第10批　第15批　第20批

4.4 實例：用 DCGAN 生成手寫數字

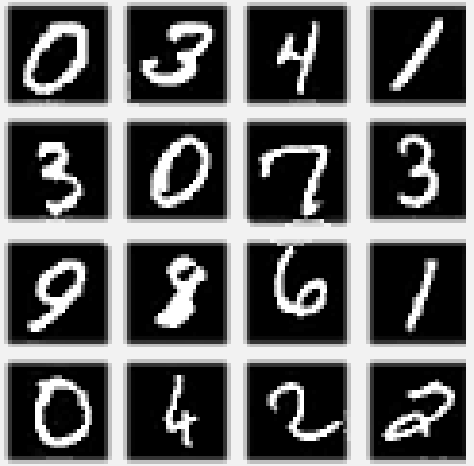
生成圖片解析度差異
GAN vs. DCGAN vs. MNIST 真樣本

趨近於相似

GAN



DCGAN



MNIST
真樣本

4.4 實例：用 DCGAN 生成手寫數字

繪製鑒別器和生成器的訓練損失

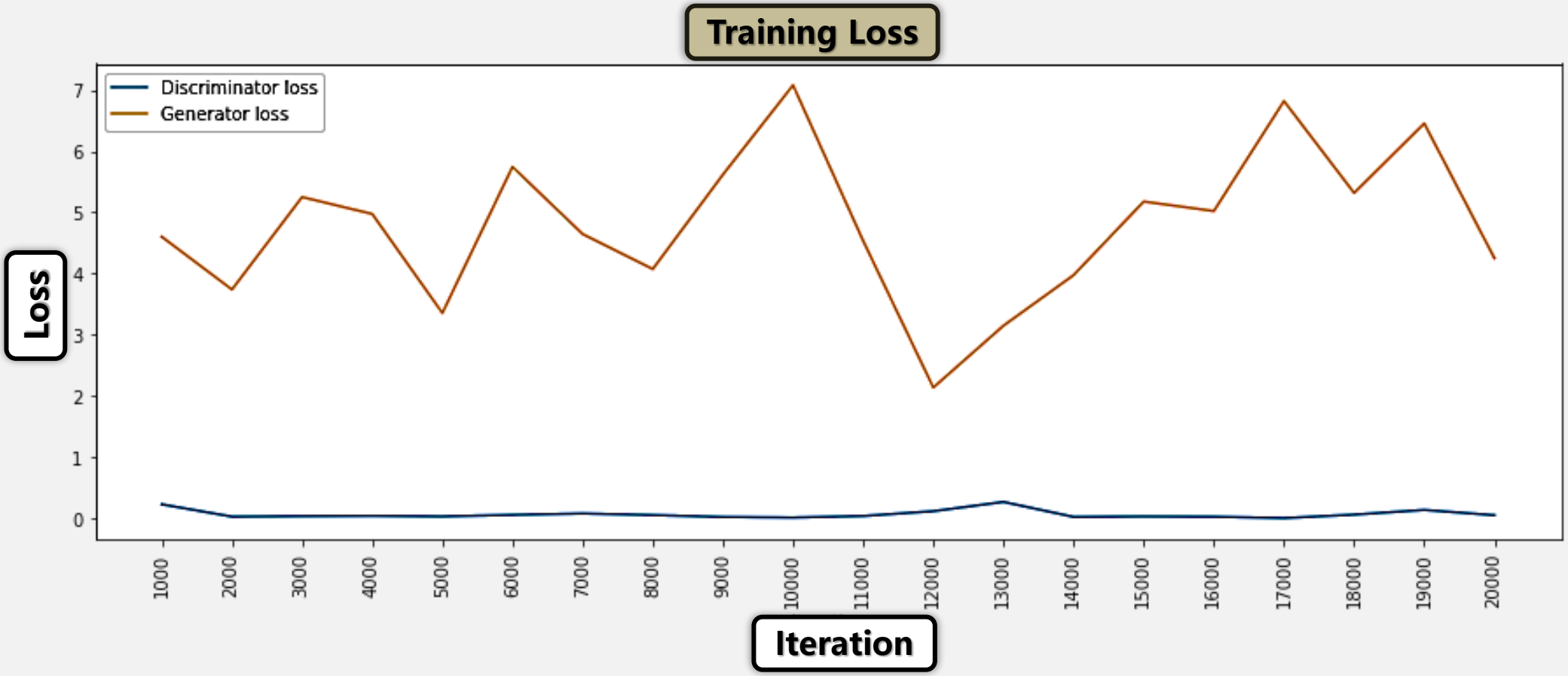
```
losses = np.array(losses)

plt.figure(figsize=(15, 5))
plt.plot(iteration_checkpoints, losses.T[0], label="Discriminator loss")
plt.plot(iteration_checkpoints, losses.T[1], label="Generator loss")

plt.xticks(iteration_checkpoints, rotation=90)

plt.title("Training Loss")
plt.xlabel("Iteration")
plt.ylabel("Loss")
plt.legend()
```

figsize = (a , b)

設置圖形大小（寬，高）

X 軸刻度向右旋轉 90 度

4.4 實例：用 DCGAN 生成手寫數字



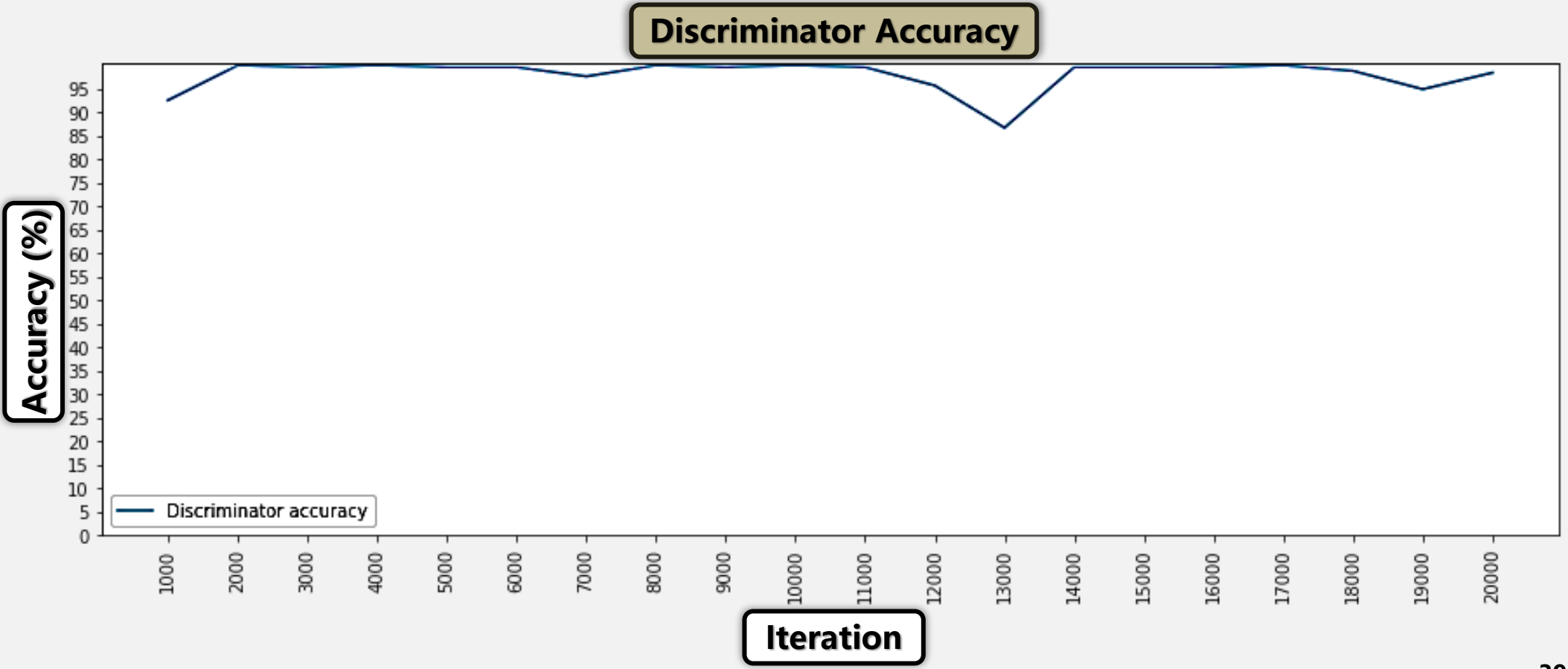
4.4 實例：用 DCGAN 生成手寫數字

繪製鑒別器的準確率

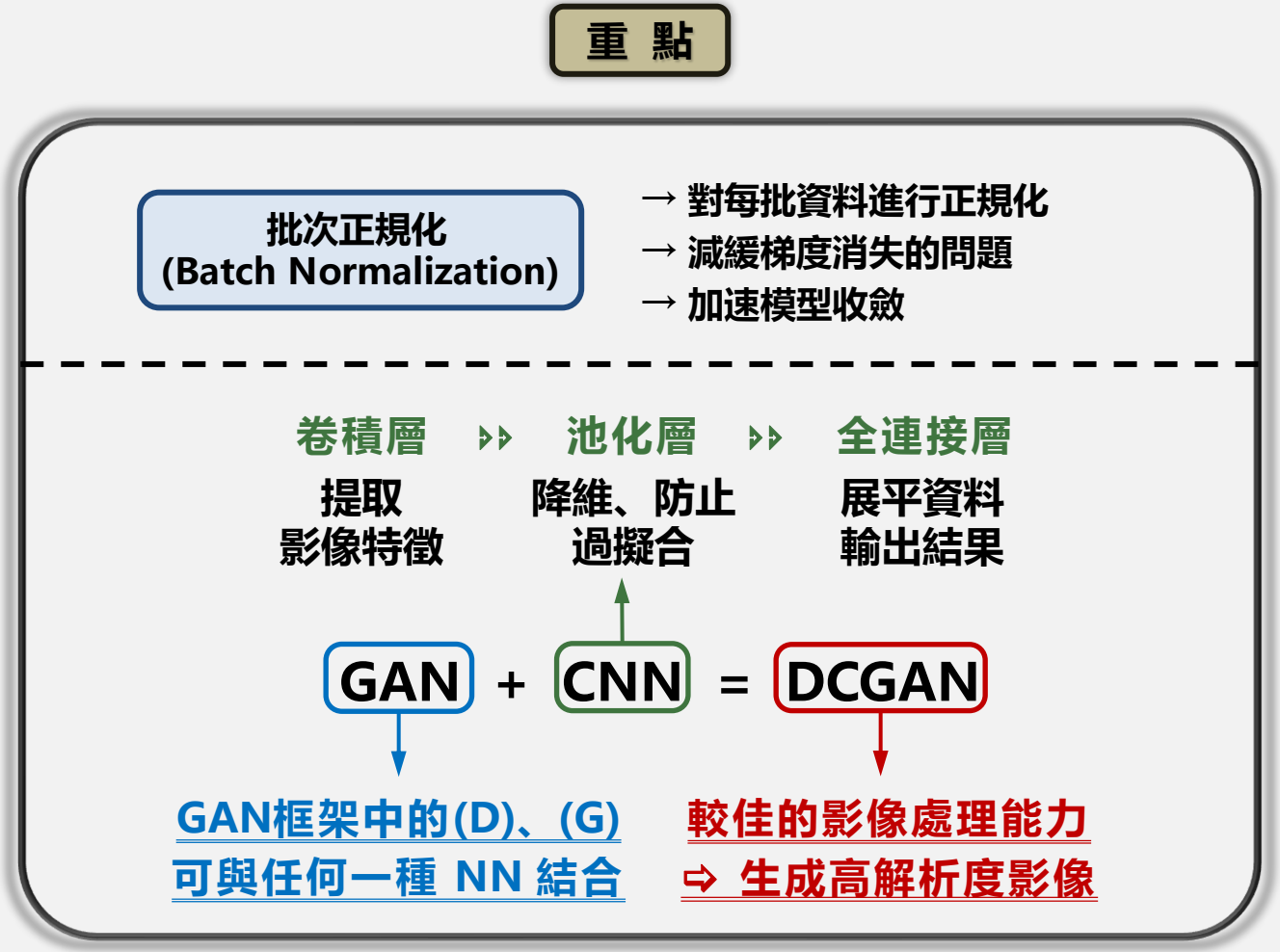
```
accuracies = np.array(accuracies)

plt.figure(figsize=(15, 5))
plt.plot(iteration_checkpoints, accuracies, label="Discriminator accuracy")

plt.xticks(iteration_checkpoints, rotation=90)
plt.yticks(range(0, 100, 5))
plt.title("Discriminator Accuracy")
plt.xlabel("Iteration")
plt.ylabel("Accuracy (%)")
plt.legend()
```

y 軸刻度值由 0 ~ 100；間隔 5

4.4 實例：用 DCGAN 生成手寫數字



4.5 重點整理


重 點
批次正規化
(Batch Normalization)
→ 對每批資料進行正規化
→ 減緩梯度消失的問題
→ 加速模型收斂
卷積層 ›› 池化層 ›› 全連接層
提取
影像特徵
降維、防止
過擬合
展平資料
輸出結果
GAN + CNN = DCGAN
GAN框架中的(D)、(G)
可與任何一種 NN 結合
較佳的影像處理能力
⇨ 生成高解析度影像