深度學習

第 7 章 卷積神經網路
(Convolutional Neural Networks)

講師：紀俊男

本章大綱

- 卷積神經網路原理
- 卷積神經網路範例
- 常見的卷積神經網路經典模型
- 遷移學習

卷積神經網路原理
Concepts of CNN

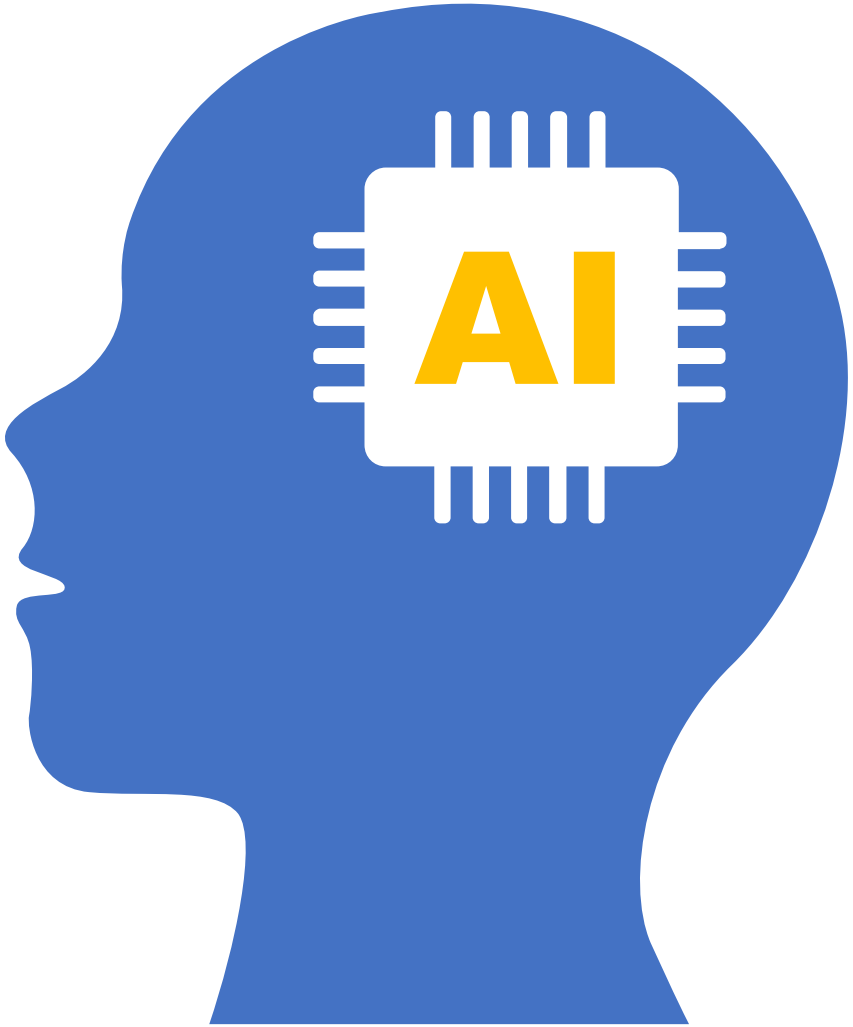
人類如何「辨認」東西

樣式辨認

抽象略化

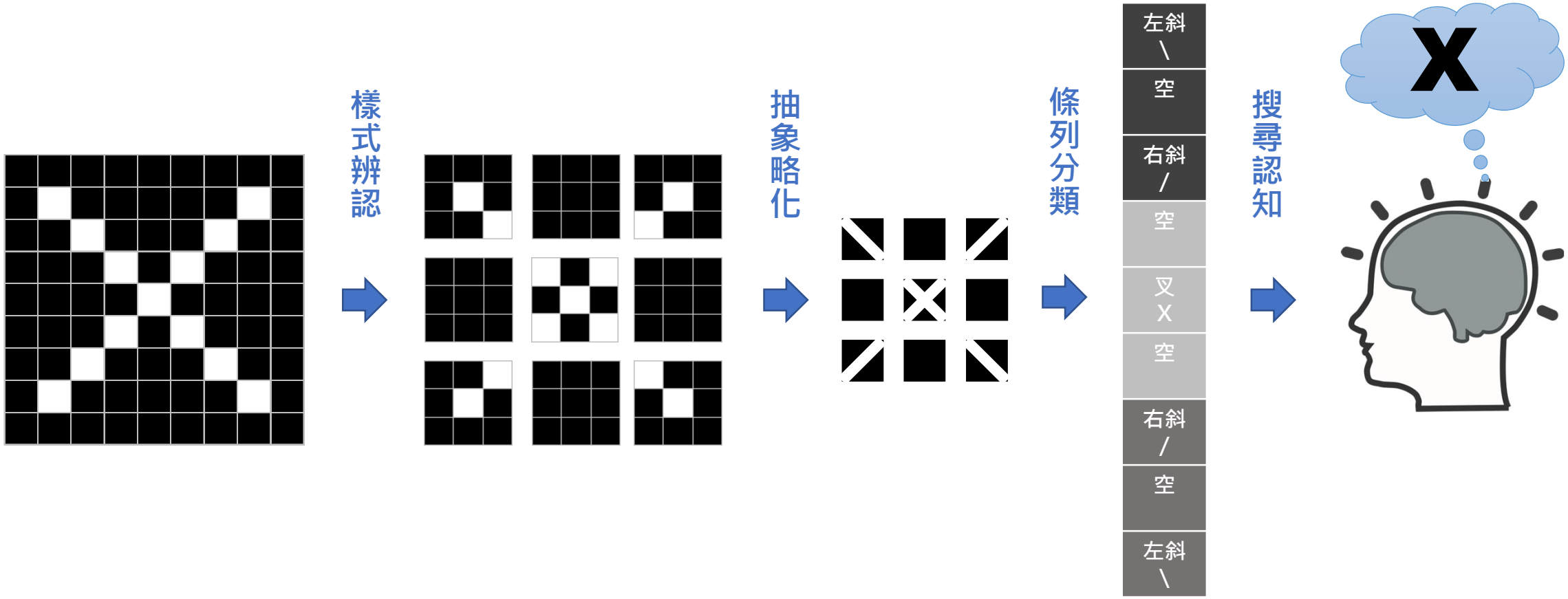
條列分類

搜尋認知



什麼是「卷積神經網路」

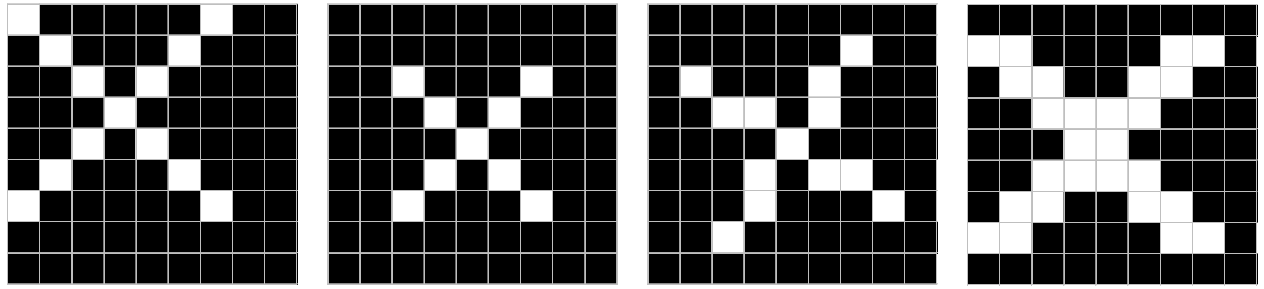
- 先瞭解人類如何「辨認」一樣東西

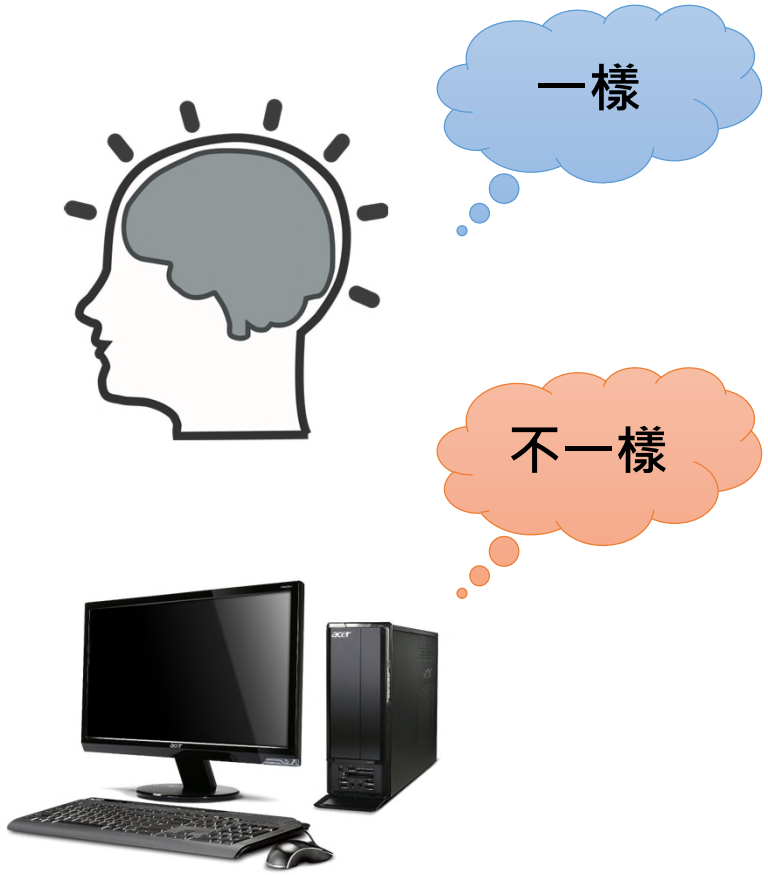


辨認東西，很難嗎？

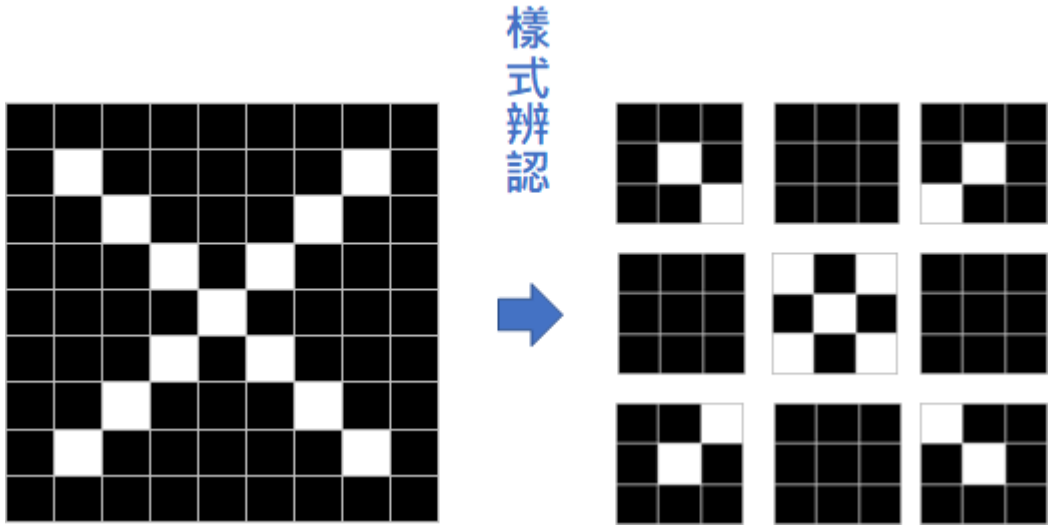
- 對電腦而言，**很難**！

平移 縮放 歪斜 粗細

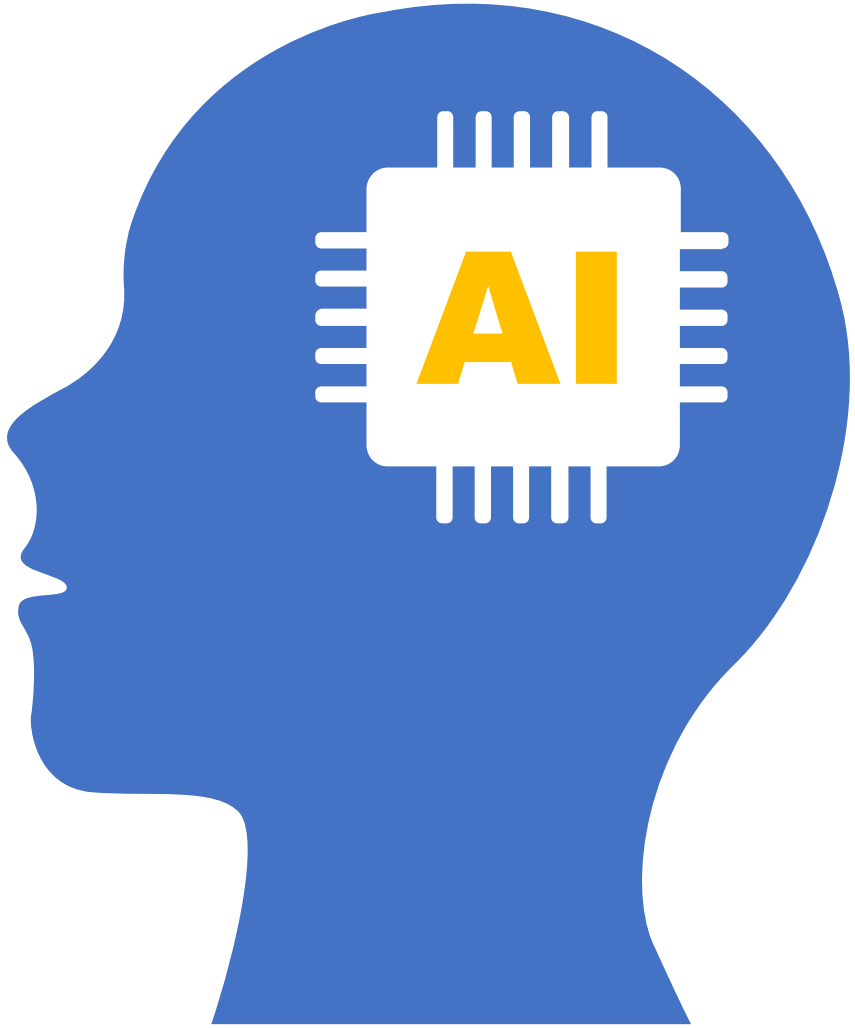




樣式辨認

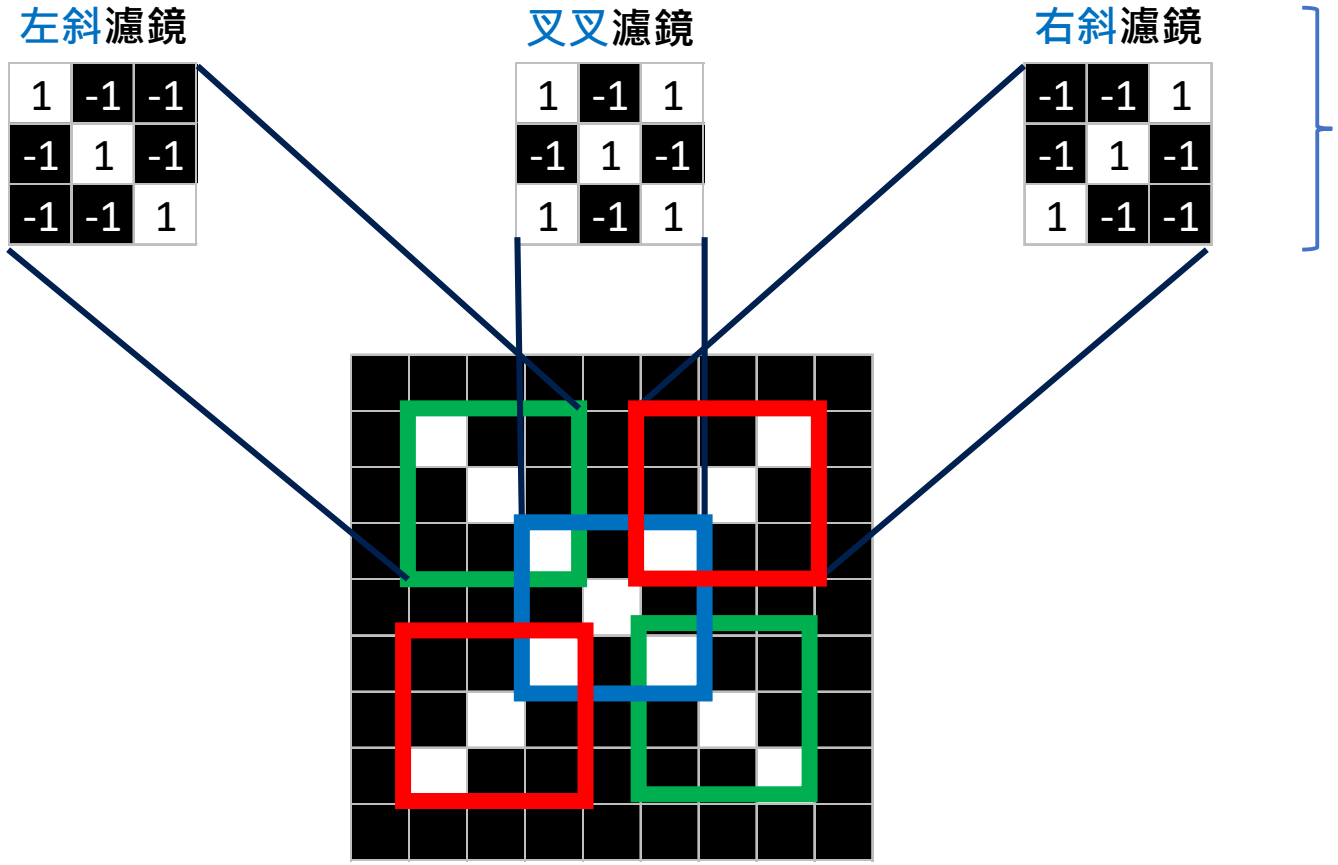


卷積運算 + ReLU 運算



電腦如何做「樣式辨認」？

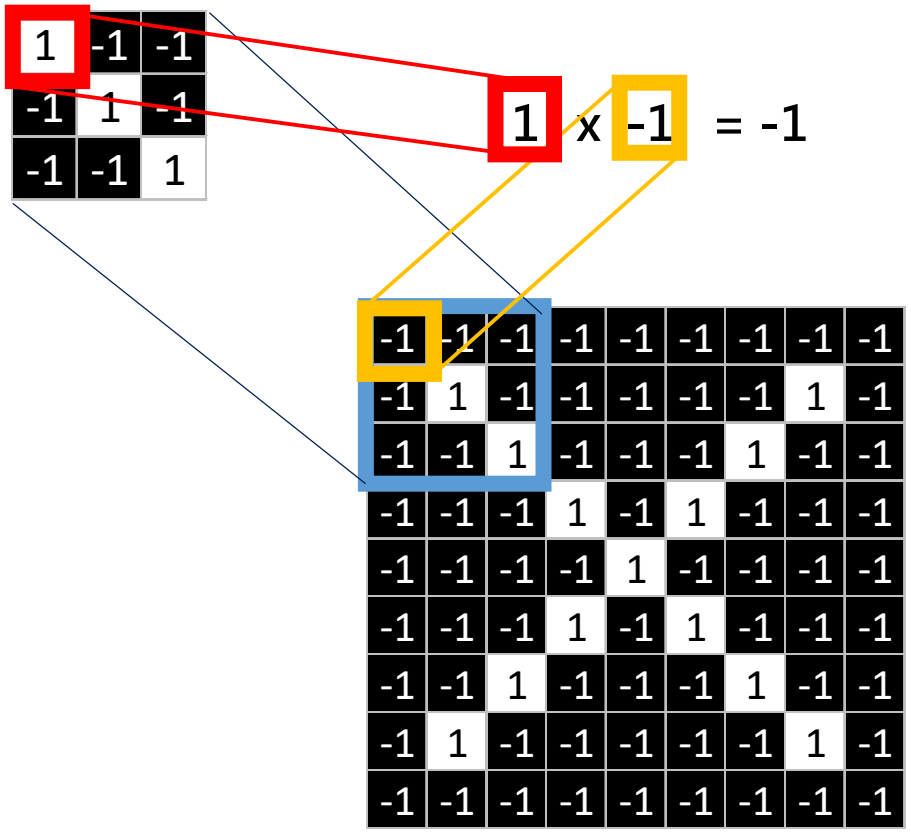
- 靠「卷積運算（Convolutional Operations）」



濾鏡（Filter）
= 尋找與之相似的樣式

電腦如何做「樣式辨認」？

- 卷積運算：第一次



所有格子的計算

濾鏡

1	-1	-1
-1	1	-1
-1	-1	1

⊗

原圖

-1	-1	-1
-1	1	-1
-1	-1	1

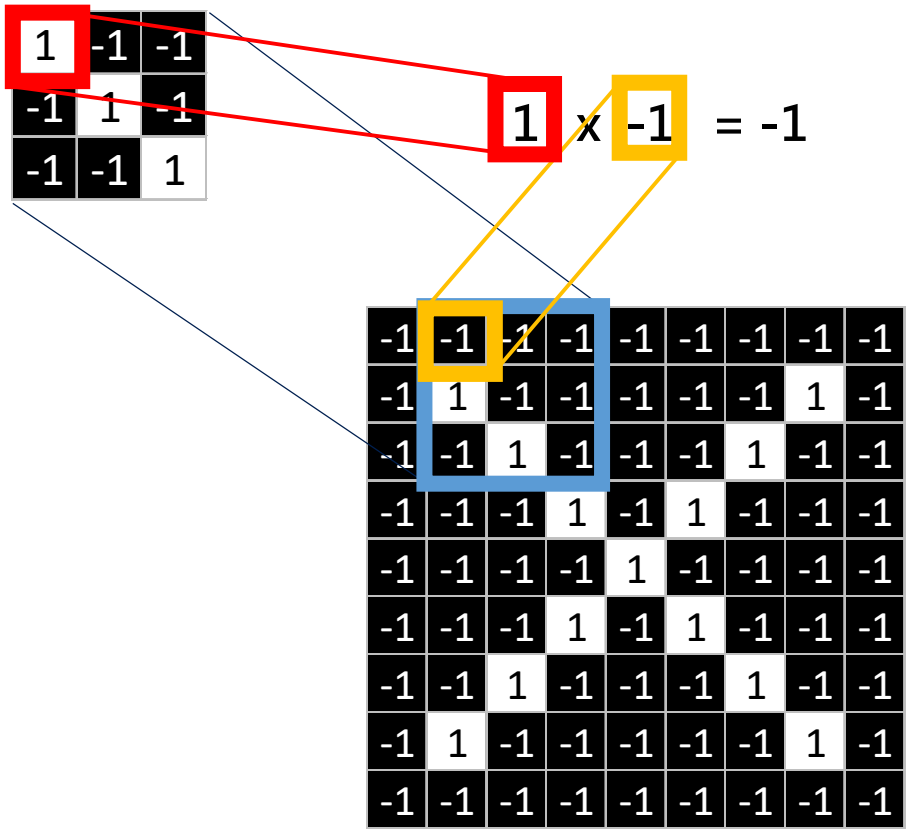
$1 \times -1 = -1$
$-1 \times -1 = 1$
$-1 \times -1 = 1$
$-1 \times -1 = 1$
$1 \times 1 = 1$
$-1 \times -1 = 1$
$-1 \times -1 = 1$
$-1 \times -1 = 1$
$1 \times 1 = 1$

$= \frac{7}{9} \cong 0.77$

取平均　濾鏡 vs. 原圖相似程度

電腦如何做「樣式辨認」？

- 卷積運算：第二次（+右移一格）



所有格子的計算

濾鏡

1	-1	-1
-1	1	-1
-1	-1	1

⊗

原圖

-1	-1	-1
1	-1	-1
-1	1	-1

1 x -1 = -1
-1 x -1 = 1
-1 x -1 = 1
-1 x 1 = -1
1 x -1 = -1
-1 x -1 = 1
-1 x -1 = 1
-1 x 1 = -1
1 x -1 = -1

$$= \frac{-1}{9} \cong -0.11$$

取平均　濾鏡 vs. 原圖相似程度

電腦如何做「樣式辨認」？

- 卷積運算：左斜濾鏡完整運算

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

⊗

1	-1	-1
-1	1	-1
-1	-1	1

=

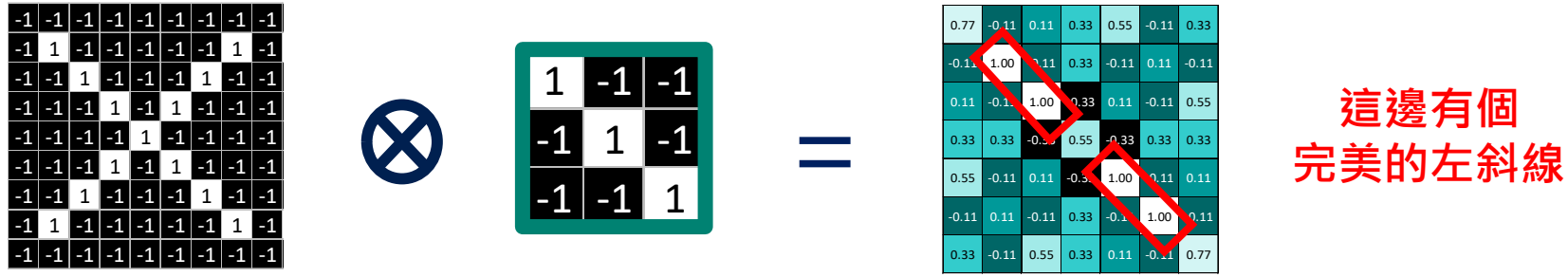
暗示！

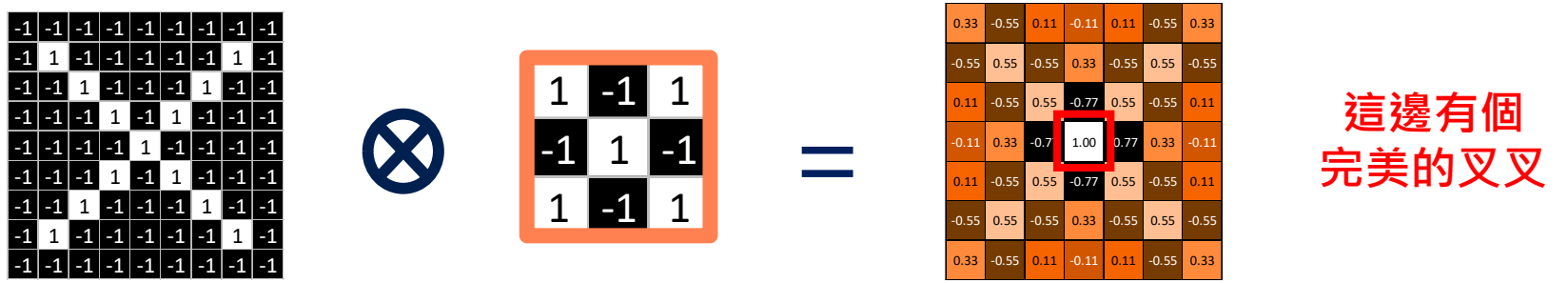
這邊有個完美的左斜線

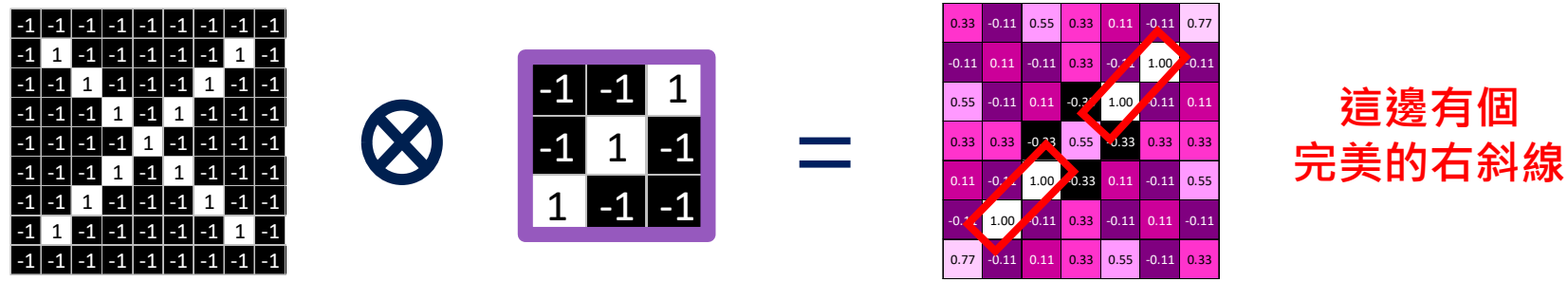
0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

電腦如何做「樣式辨認」？

- 卷積運算：左斜、叉叉、右斜濾鏡完整運算



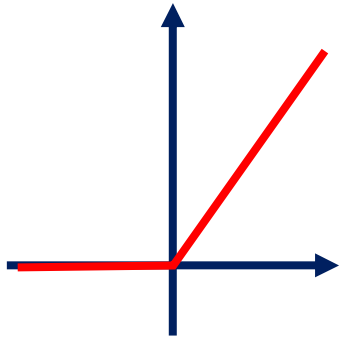




電腦如何做「樣式辨認」？

- 線性整流運算：機率不可能是負的 → 負機率=0 → 防止機率正負相消

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77



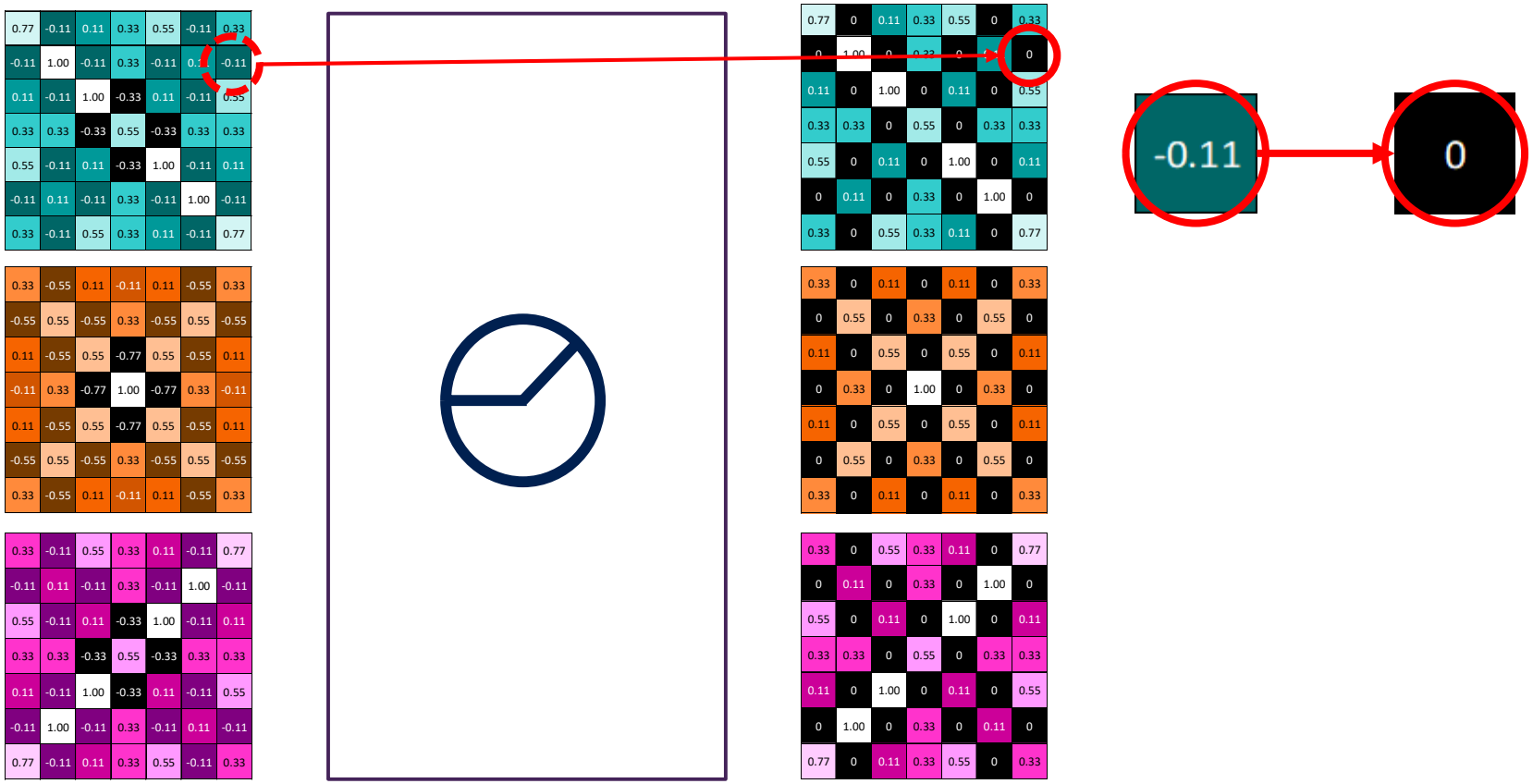
ReLU

Rectified **L**inear **U**nits

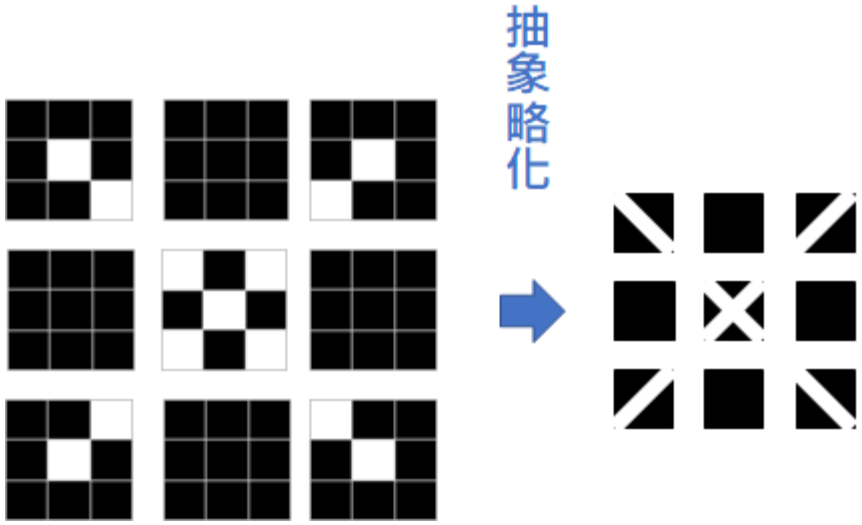
0.77	0	0.11	0.33	0.55	0	0.33

電腦如何做「樣式辨認」？

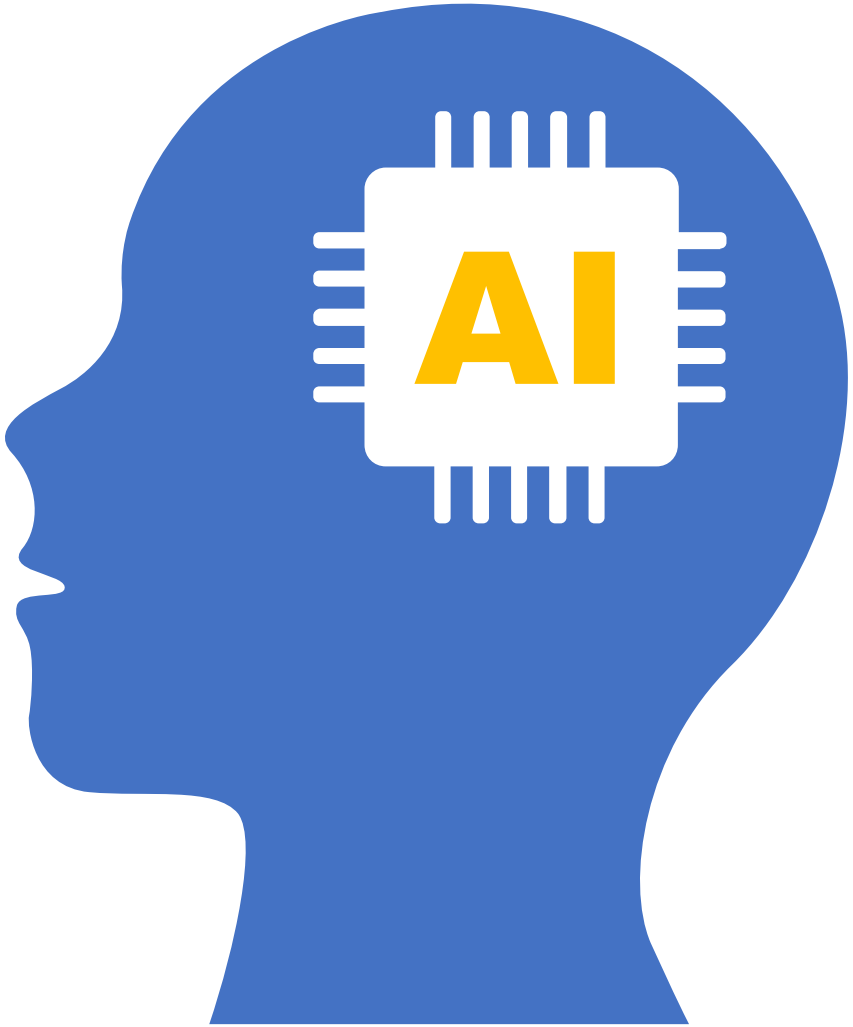
- **線性整流運算**：所有**濾鏡**的 **ReLU 運算**



抽象略化

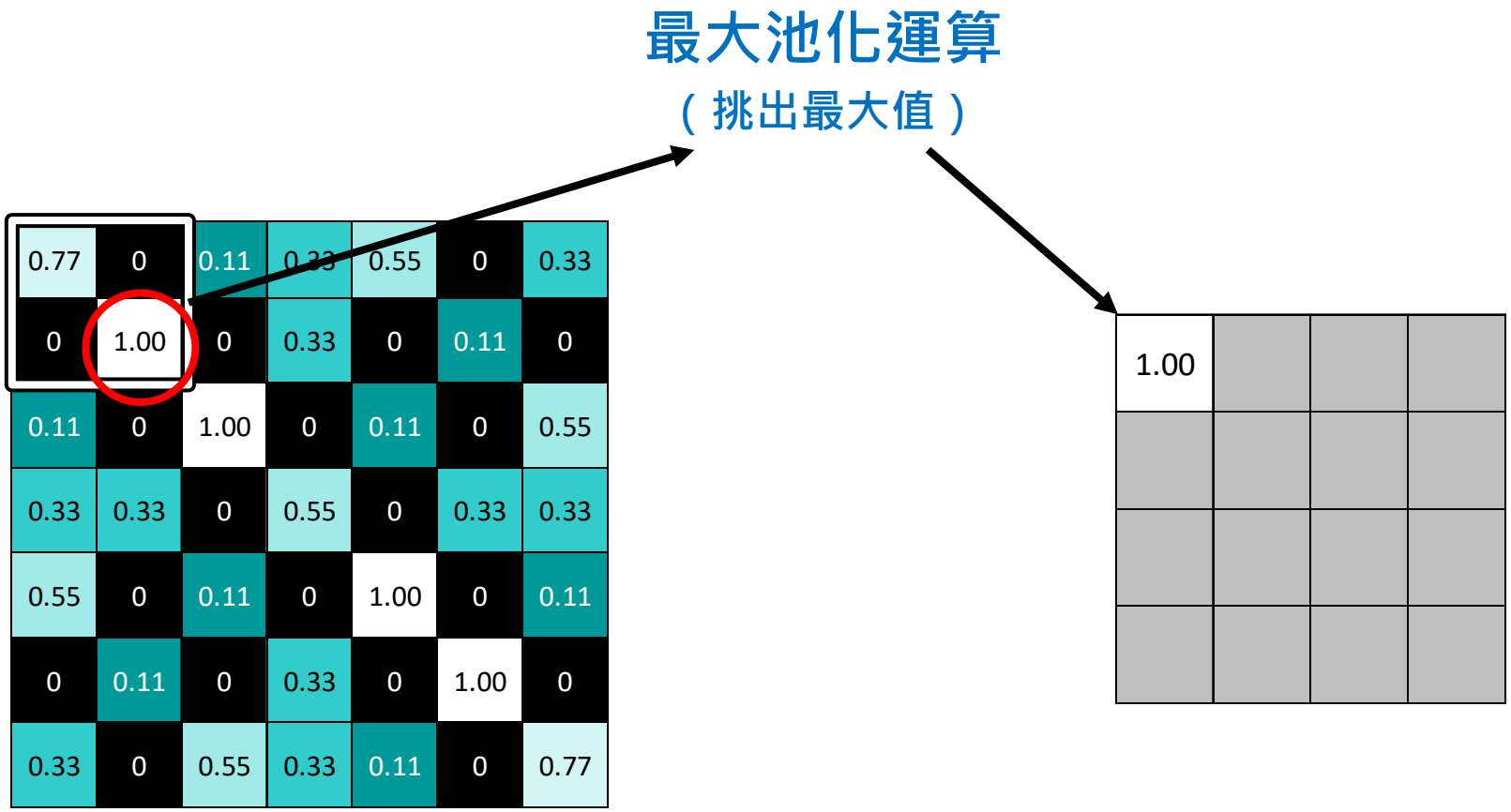


池化運算



電腦如何做「抽象略化」？

- 使用「池化運算（Pooling Operations）」

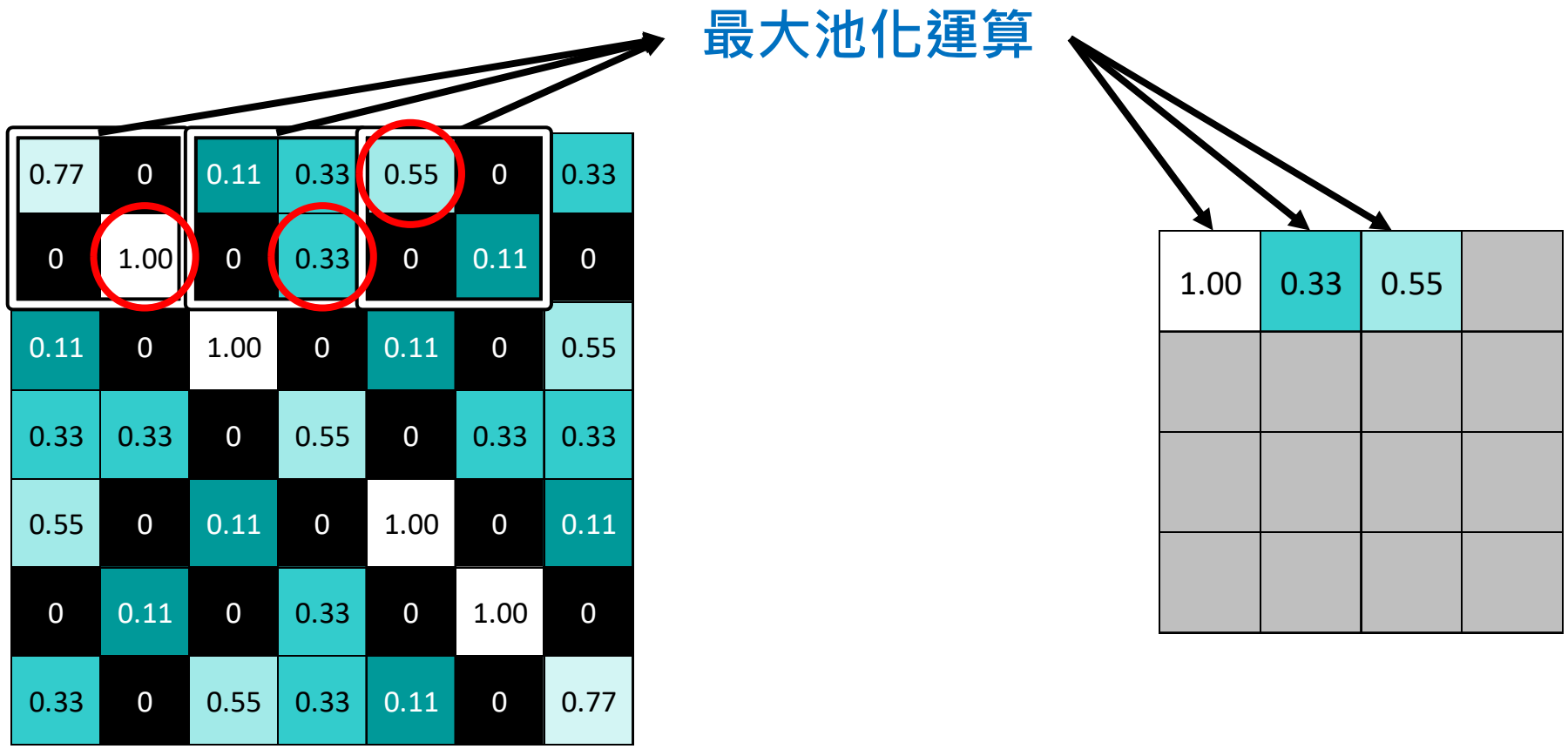


池化（Pooling）

- 在不損失特徵的情況下，降低維度。
- 種類：最大池化、平均池化、最小池化...。

電腦如何做「抽象略化」？

- 池化運算過程（每次右移兩格）



電腦如何做「抽象略化」？

- 池化運算：左斜濾鏡「最大池化」完整運算

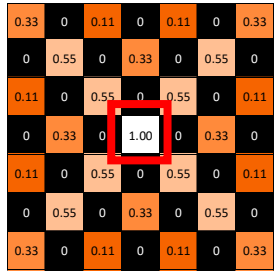
0.77	0	0.11	0.33	0.55	0	0.33
0	1.00	0	0.33	0	0.11	0
0.11	0	1.00	0	0.11	0	0.55
0.33	0.33	0	0.55	0	0.33	0.33
0.55	0	0.11	0	1.00	0	0.11
0	0.11	0	0.33	0	1.00	0
0.33	0	0.55	0.33	0.11	0	0.77

最大池化運算

1.00	0.33	0.55	0.33
0.33	1.00	0.33	0.55
0.55	0.33	1.00	0.11
0.33	0.55	0.11	0.77

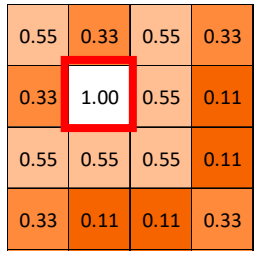
電腦如何做「抽象略化」？

- 池化運算：所有濾鏡的池化運算

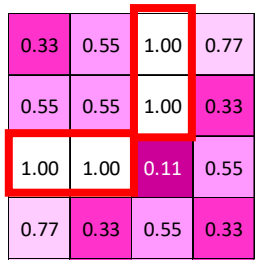


最大池化運算

這邊有個
完美的左斜線



這邊有個
完美的叉叉

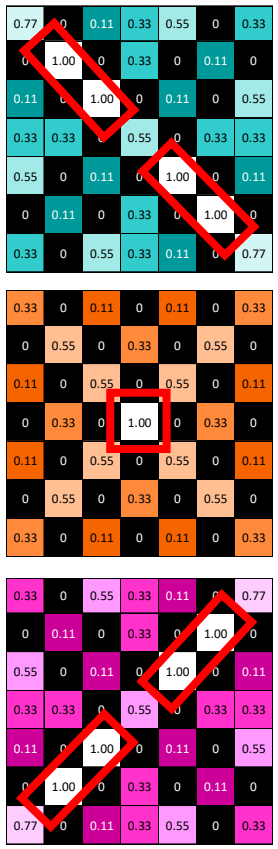


這邊有個
完美的右斜線

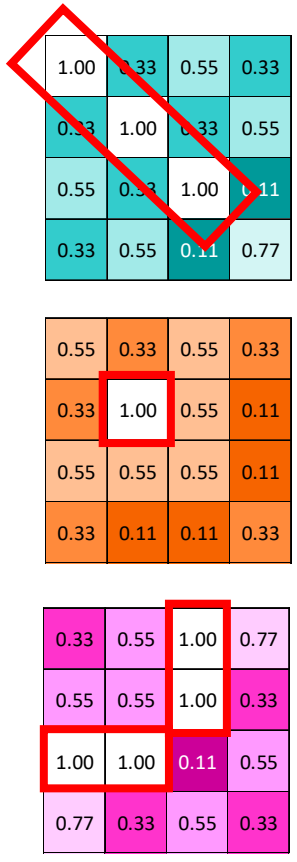
池化
= 不損失特徵下，
減低維度。

電腦如何做「抽象略化」？

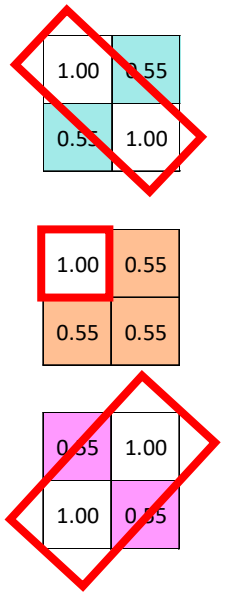
- 池化運算可以不止一次



最大池化運算

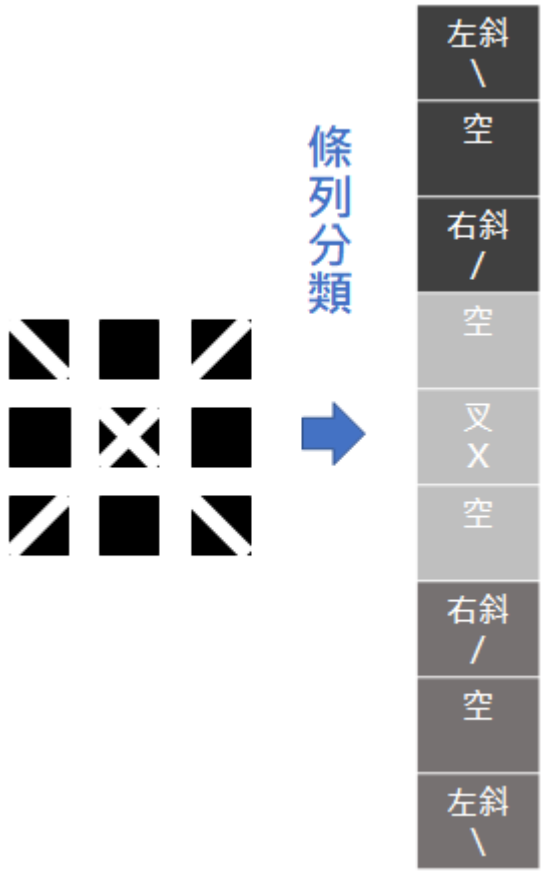


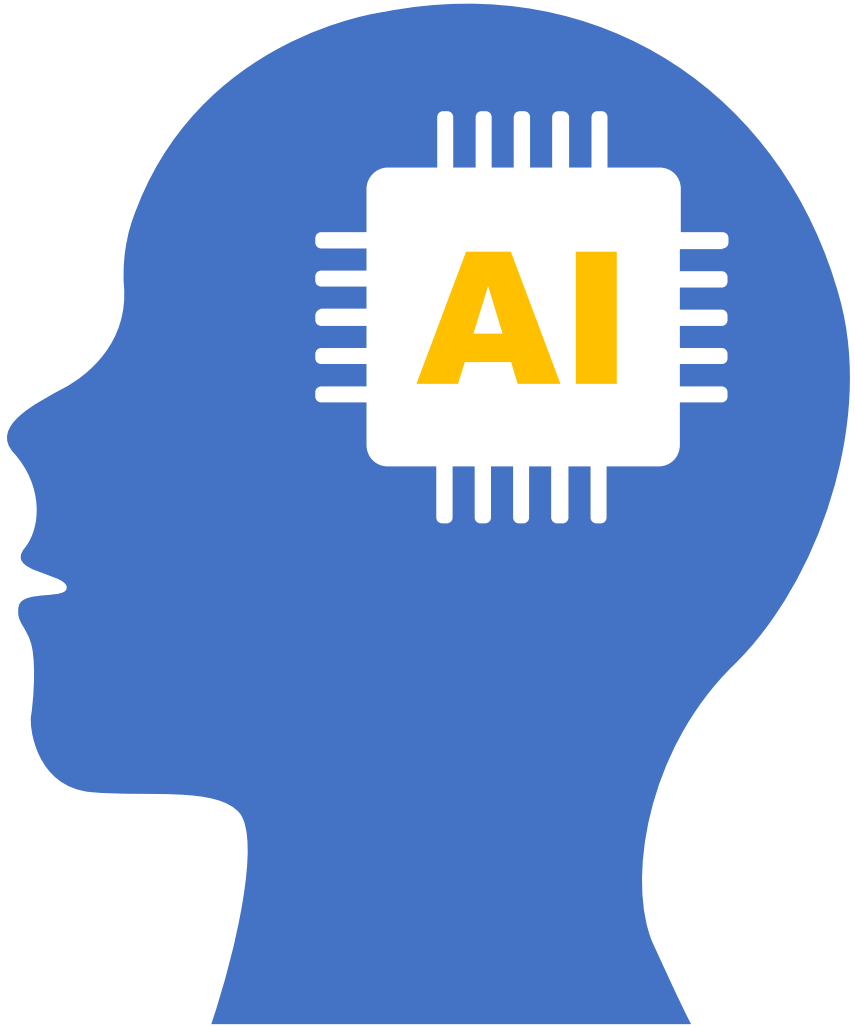
最大池化運算



特徵維持，
維度降低！

條列分類

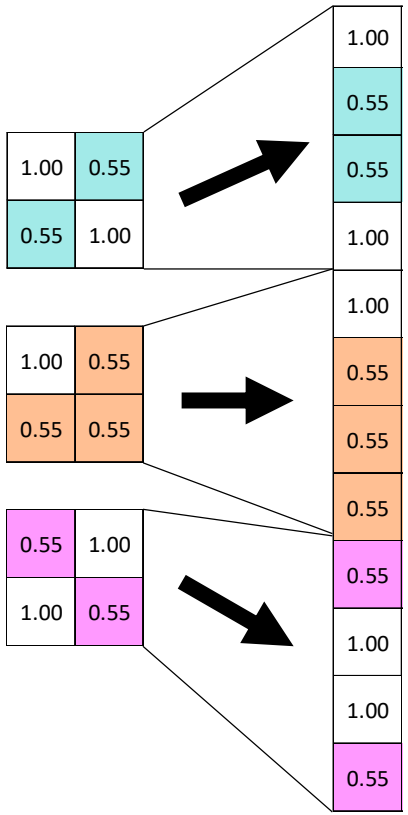




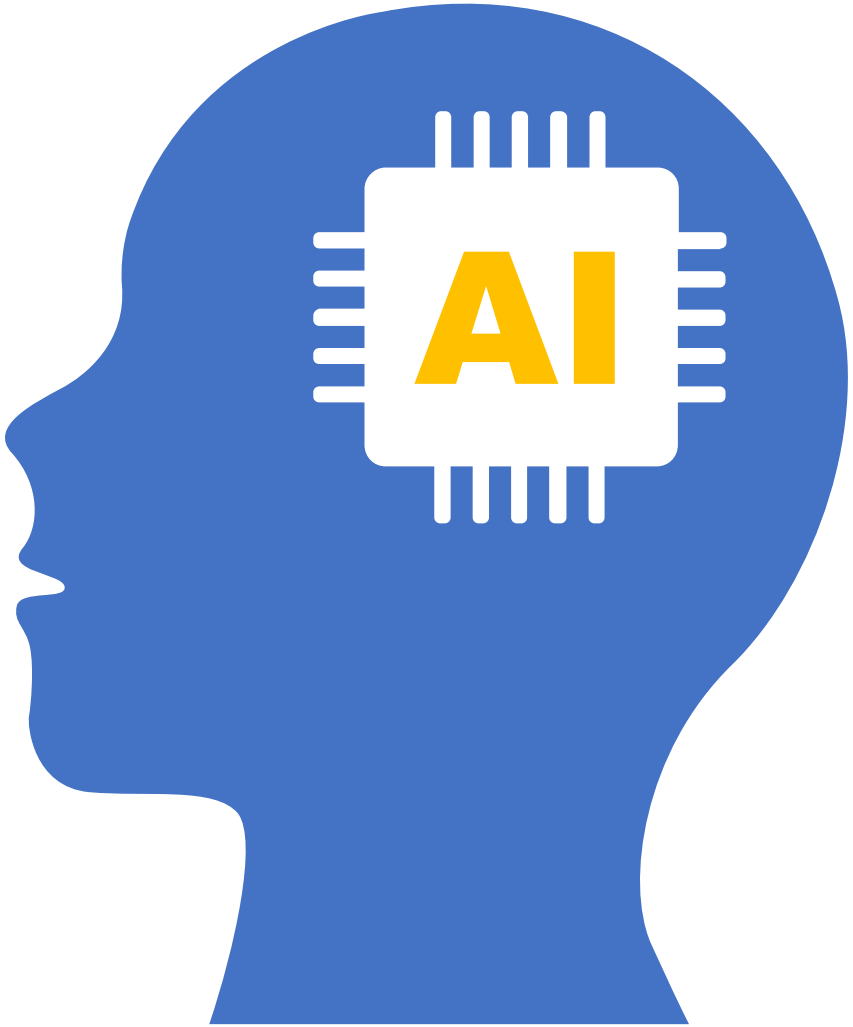
扁平化運算

電腦如何做「條列分類」？

- 對結果執行「扁平化運算（Flatten Operations）」



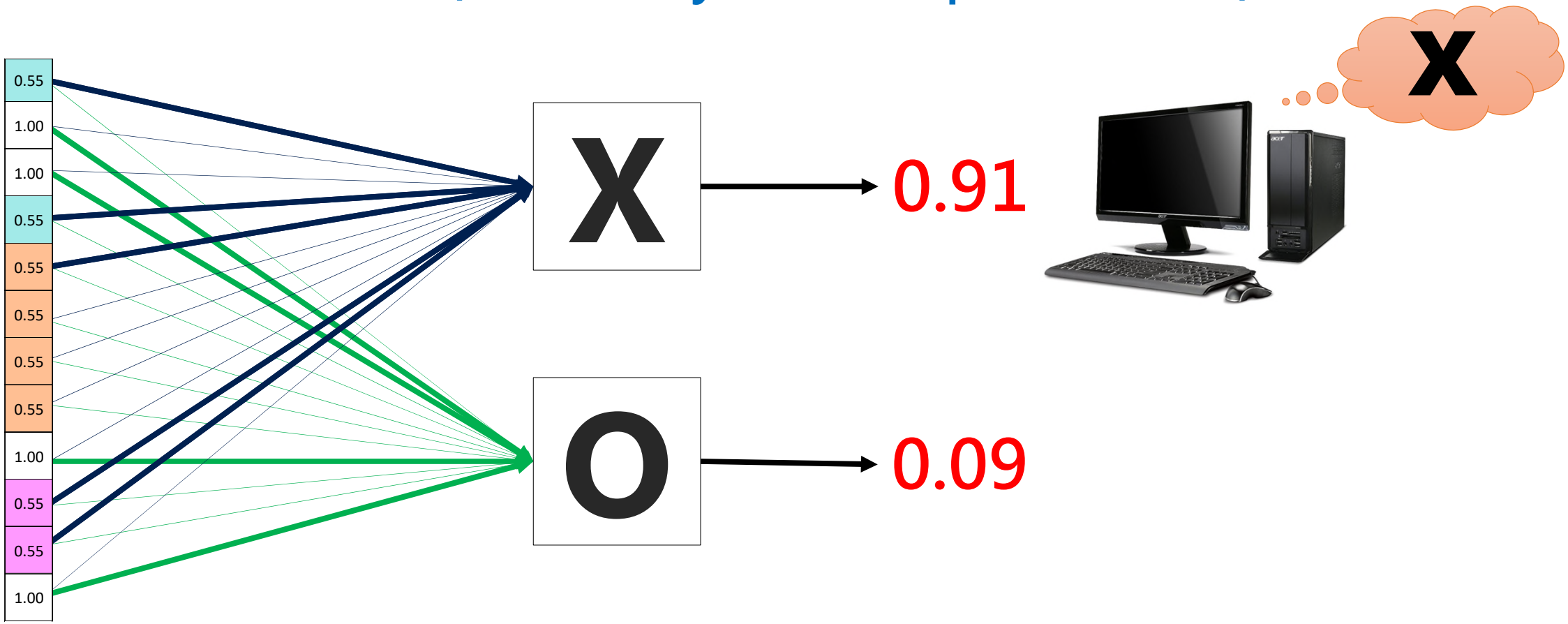
搜尋認知



全連接層運算

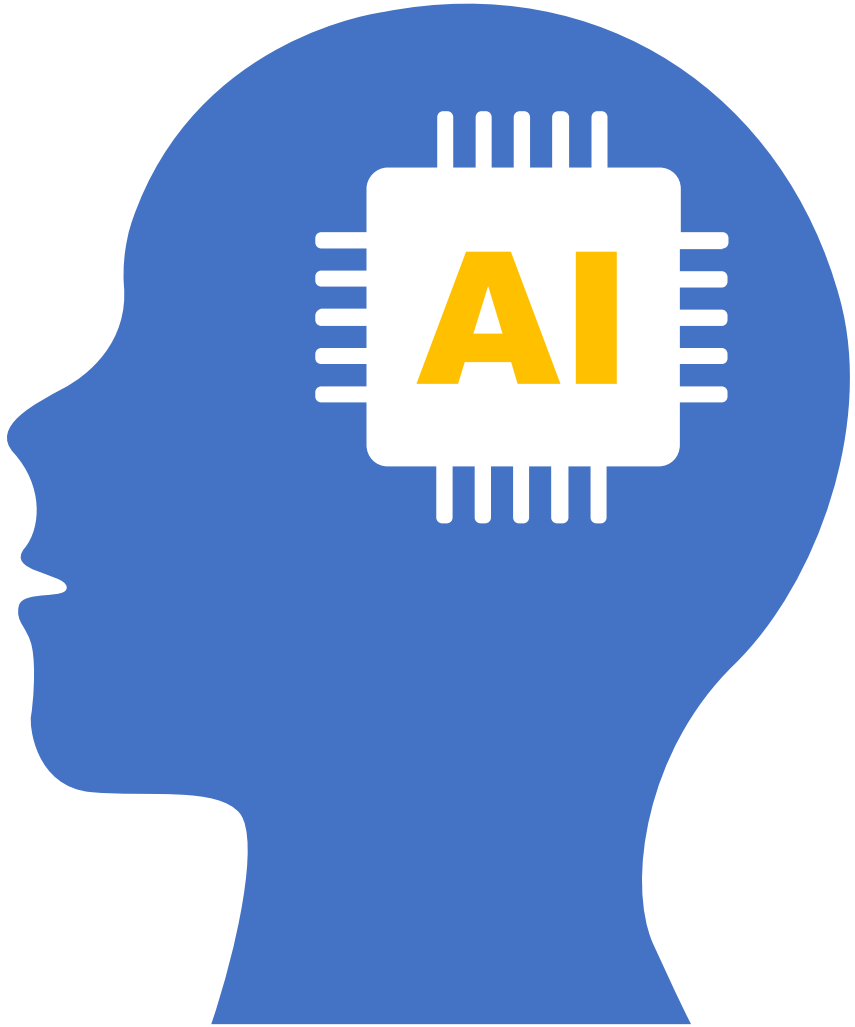
電腦如何做「搜尋認知」？

- 使用「多層次感知器（Multi-Layers Perceptron, MLP）」分類

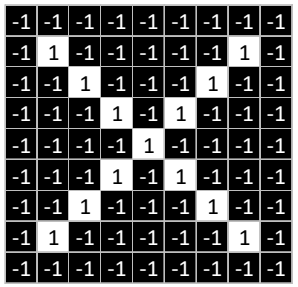


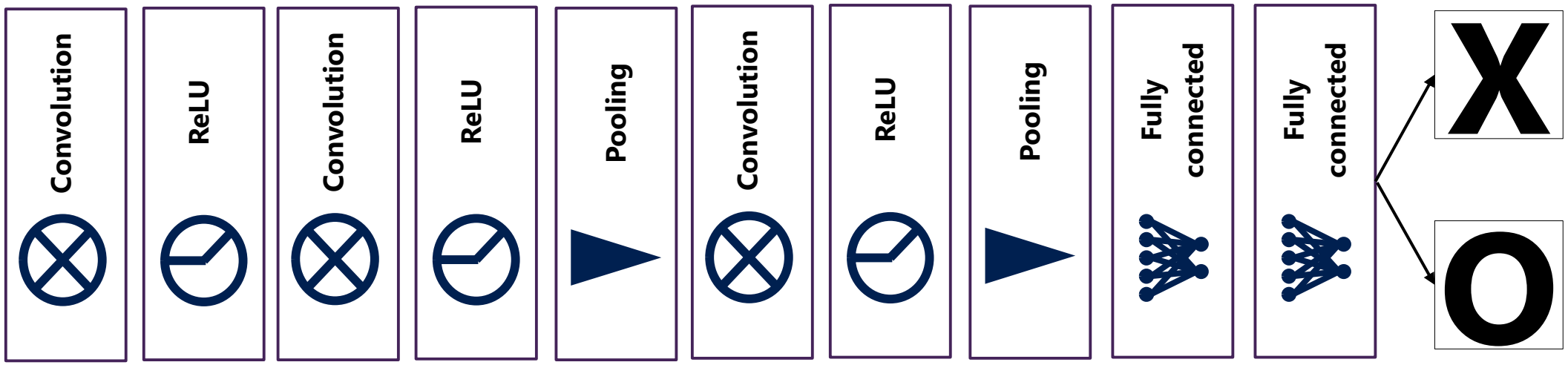
其它卷積神經網路概念

- 卷積神經網路完整架構
- 超參數
- 適用 / 不適用場合



完整的「卷積神經網路」架構

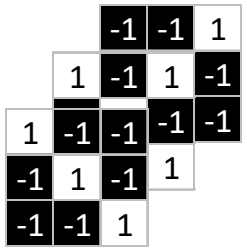




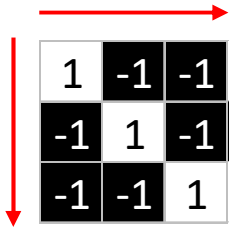
卷積神經網路的「超參數」有哪些？

「卷積運算」相關

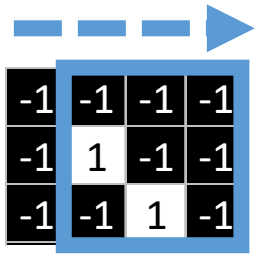
濾鏡種類



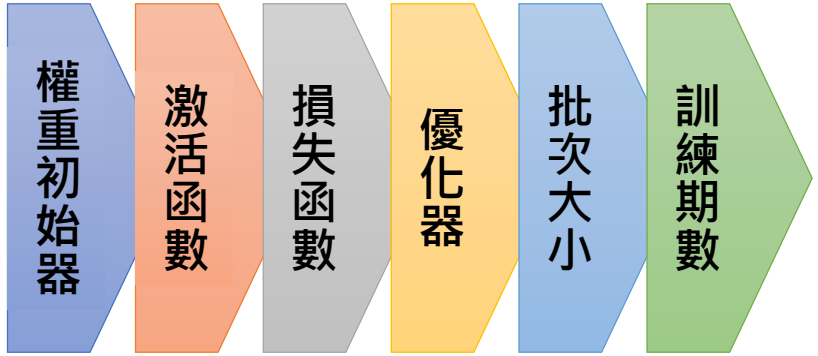
濾鏡大小



滑動步長

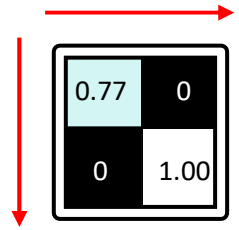


「全連接層」相關

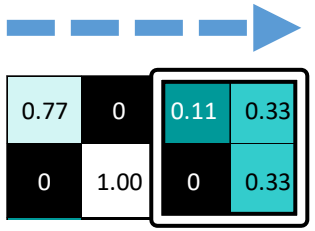


「池化運算」相關

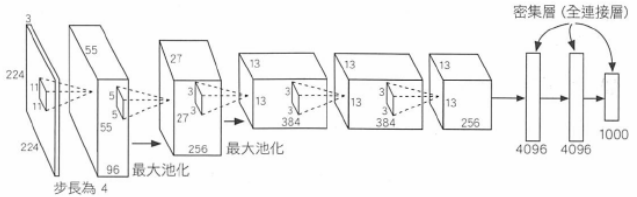
池化範圍



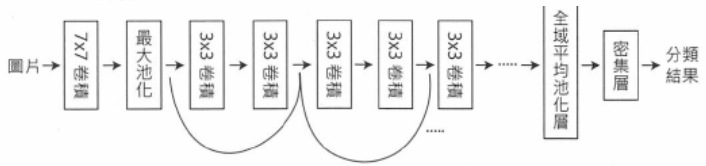
滑動步長



「網路架構」相關



AlexNet



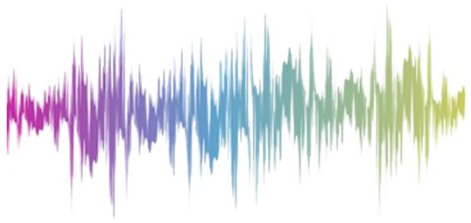
ResNet

卷積神經網路適用場合

- 適用時機

鄰近資料，有結構上的關連意義

圖片



聲音

CHRISTIANITY, like other religions, provides a world-view—that is, a particular way of explaining and interpreting experience. But so does science, and so do political theories such as Marxism; so does our very humanity. Such world-views are only partial. Science provides a purely descriptive view, with no values; but combined with the view derived from our humanity, which does have values, it gives the most satisfactory available world-view — scientific humanism. But this article is about Christianity, not humanism, and in particular about the (to me) rather marginal question of why I am not a Christian.

The Christian world-view was once the best available. Now it is woefully inadequate. Like the phlogiston theory, Christianity explained a great deal by a not-too-incredible hypothesis. But different, better explanations are now given by science. Phlogiston has long since been rejected. God, however, still clings on. Christianity is stretched this way and that to accommodate new knowledge — Copernicus and Galileo, Darwin and Einstein — and these distortions cause it to fragment.

Some try to reconcile science and Christianity. The Bishop of Woolwich has discovered how difficult this is. "Ultimate Reality" simply *is not* personal. "To call the world 'God' is not to explain it; it is only to enrich our language with a superfluous synonym for the word 'world'." (Schopenhauer). If the Bishop were honest with himself, surely he would renounce such mystical self-indulgence and label himself "atheist."

Others reject either science (fundamentalists) or else religion (atheists). The latter at least generally experience an enormous relief. They are released from the considerable intellectual strain of not allowing their religion to be confronted by common-sense or a scientific scepticism.

For God finds no place in scientific thought. No one can explain in scientifically defined terms the nature of Heaven, Hell, the soul, the Trinity, or God. Christians generally try to make a virtue of this failing. "That's the great mystery!" they exclaim, as if this were compelling proof,

文字

- 不適用時機

任兩列 / 兩欄交換，不影響資料集完整性

交換失去完整性

適合用 CNN

Age	Height	Salary	Education	Matching
25	179	75	College	N
33	190	95	College	Y
28	180	90	Master	Y
25	178	90	Master	Y
30	180	175	Bachelor	Y
40	170	350	Bachelor	N
34	174	100	Master	Y
36	181	220	Bachelor	N
35	170	125	Master	Y
46	177	500	Master	N
28	174	150	Bachelor	N
29	176	180	Bachelor	Y

交換不失完整性

不適合用 CNN

小節整理：卷積神經網路概念

- **人類如何「辨認」東西？**
  - 樣式辨認 → 抽象略化 → 條列分類 → 搜尋認知

- **「卷積神經網路」基本運算**
  - 卷積運算：找出特定樣式，不論它們是否被平移、縮放、歪斜、加粗
  - 線性整流運算：防止「負機率」出現
  - 池化運算：在不損失特徵的情況下，降低維度（抽象化）
  - 扁平化運算：讓 2D 特徵轉 1D，好進入神經網路做分類
  - 全連接運算：執行分類動作

小節整理：卷積神經網路概念

- **卷積神經網路的「超參數」**
  - 卷積運算：濾鏡種類、濾鏡大小、滑動步長
  - 池化運算：池化範圍、滑動步長
  - 全連接層運算：權重初始器、激活函數、損失函數、優化器、批次大小、訓練期數
  - 網路架構：卷積、池化、扁平、全連接...各種排列組合

- **何時適用「卷積神經網路」？**
  - 適用：資料前後，有結構上的關連性（所以不能前後調換）
  - 不適用：資料前後，沒有結構上的關連性（所以可以前後調換）

卷積神經網路範例

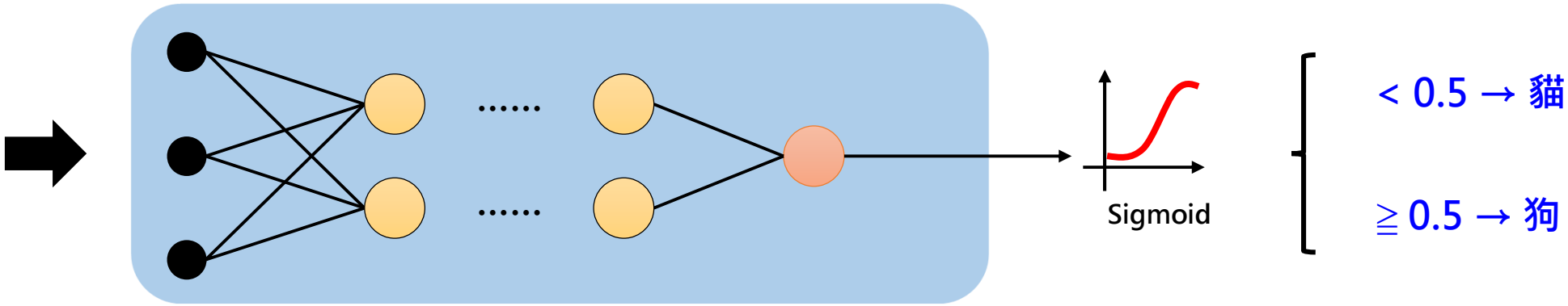
貓 vs. 狗圖片辨識

範例完整原始碼：
https://lurl.cc/5WQtzo

題目說明

卷積神經網路

(Convolutional Neural Networks)



資料集安裝＆說明

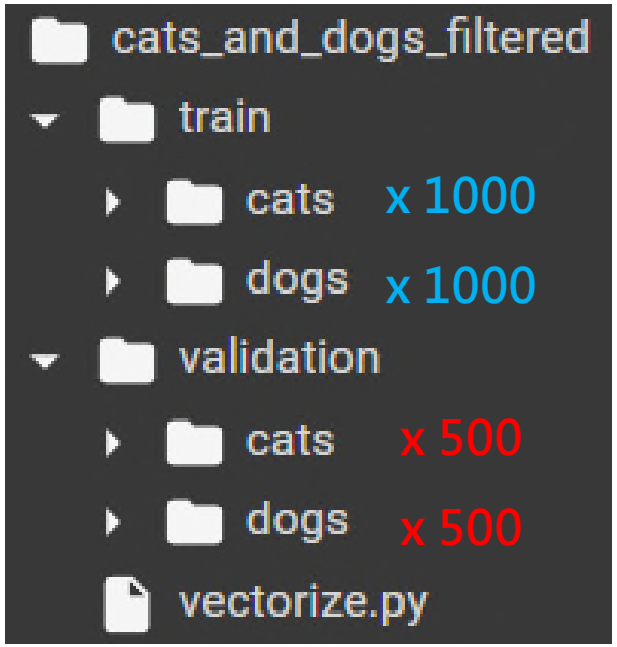
- 請用下列原始碼，下載「貓狗照片」資料集

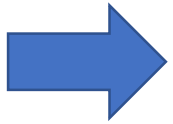
```
1  # Dataset Download
2  !wget https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip
3
4  # Unzip the file
5  import os
6  import zipfile
7
8  zip_file = zipfile.ZipFile('cats_and_dogs_filtered.zip', 'r')
9  zip_file.extractall()
10 zip_file.close()
```

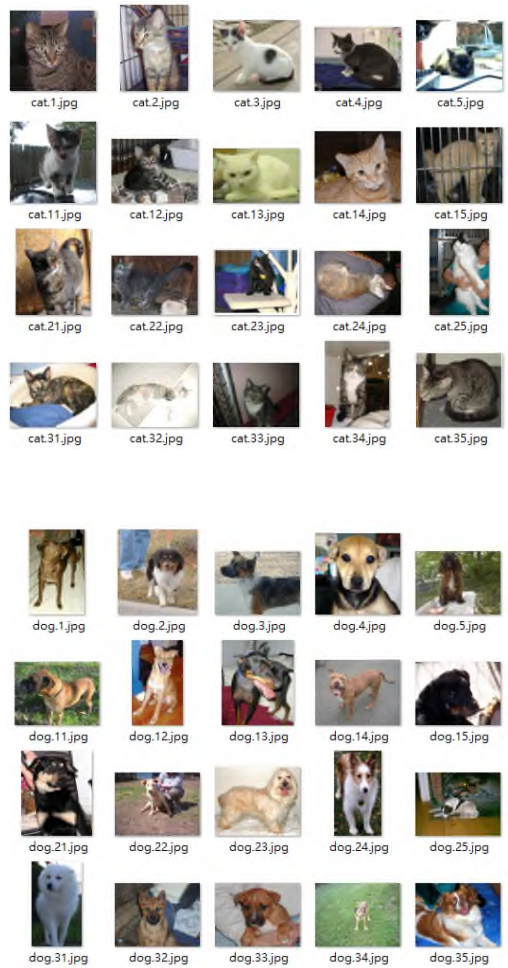
Open for Read

資料集安裝&說明

- 展開資料集並觀察之







精細度高，大小不一

隨堂練習：資料集安裝

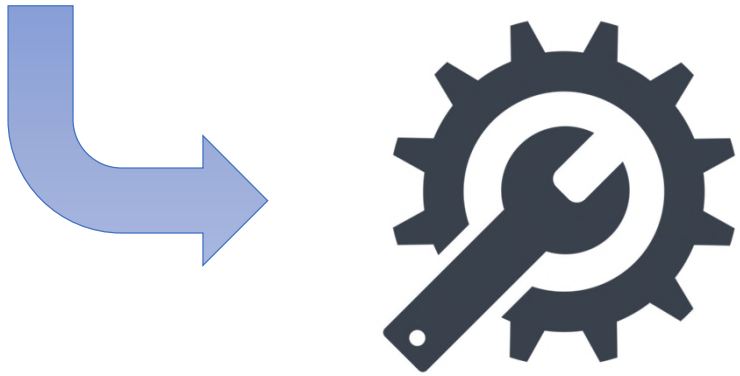
- 請撰寫下列原始碼並執行，將資料集下載並解壓縮。
- 解壓縮之後，瀏覽一下資料集的目錄架構。
- 隨手點開一張圖片，感受一下其中的內容。

```
# Dataset Download
!wget https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip

# Unzip the file
import os
import zipfile

zip_file = zipfile.ZipFile('cats_and_dogs_filtered.zip', 'r')
zip_file.extractall()
zip_file.close()
```

函式庫&資料集設定

- 安裝 HappyML 函式庫

```
1  import os
2
3  if not os.path.isdir("HappyML"):
4      os.system("git clone https://github.com/cnchi/HappyML.git")
```

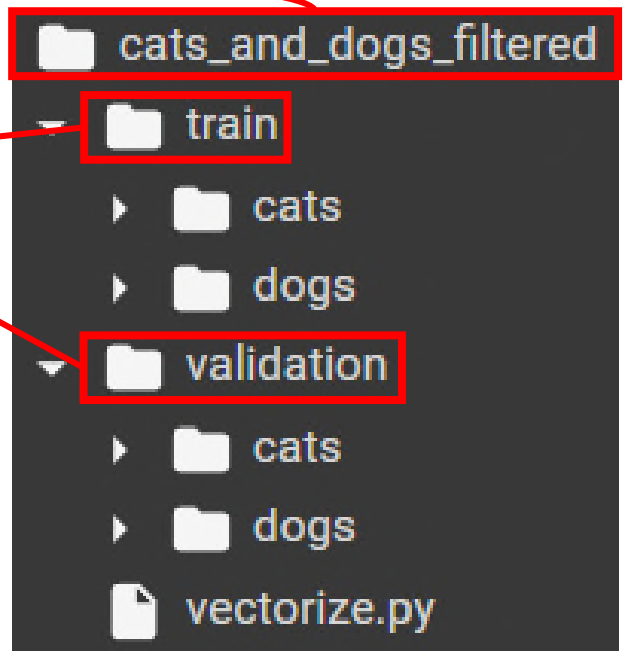


HappyML

函式庫＆資料集設定

- 設定資料集路徑

```
1  # Dataset Path Setting
2  base_dir = 'cats_and_dogs_filtered'
3  train_dir = os.path.join(base_dir, 'train')
4  test_dir = os.path.join(base_dir, 'validation')
```



名稱需一致！

隨堂練習：函式庫＆資料集設定

- 請撰寫前兩頁的下列程式碼
  - 安裝 HappyML 函式庫
  - 設定資料集路徑
- 執行看看，看是否專案資料夾內，會多出 HappyML 這個資料夾？

```
# Install HappyML
import os

if not os.path.isdir("HappyML"):
  os.system("git clone https://github.com/cnchi/HappyML.git")

# Dataset Path Setting
base_dir = 'cats_and_dogs_filtered'
train_dir = os.path.join(base_dir, 'train')
test_dir = os.path.join(base_dir, 'validation')
```

照片前處理

- 為何照片需要前處理？

無法接受

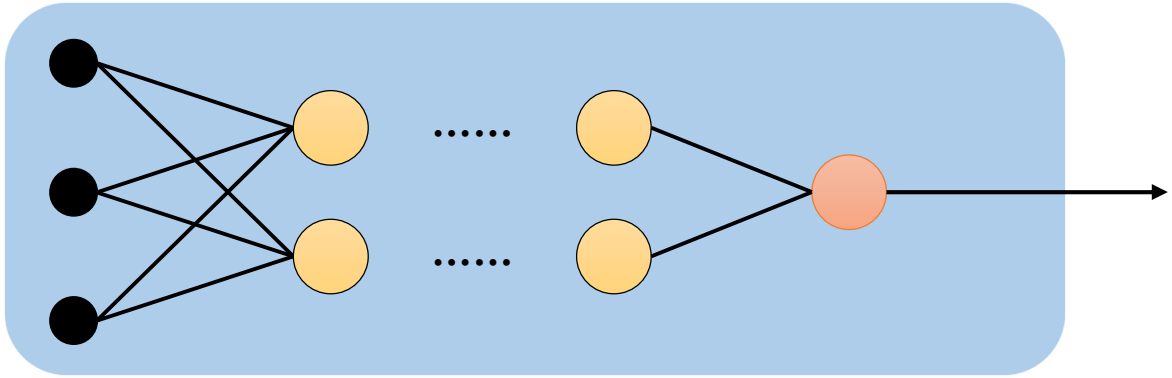
```
[[207, 199, 181],
 [203, 195, 175],
 [203, 196, 173],
 ...,
 [135, 132, 127],
 [162, 158, 150],
 [168, 163, 151]]
```

[R, G, B]

可以接受

卷積神經網路

(Convolutional Neural Networks)



照片前處理

- (1) 建造「影像資料陣列生成器」

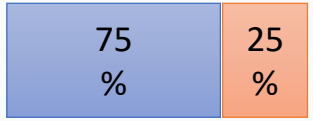
1 引入必要套件

```
1  from tensorflow.keras.preprocessing.image import ImageDataGenerator
2
3  train_datagen = ImageDataGenerator(
4      rescale=1.0/255,
5      validation_split=0.25,
6      shear_range=10,
7      zoom_range=0.2,
8      horizontal_flip=True)
```

2 縮放至 0~1

```
[ 59,  62,  63],   [0.23, 0.24, 0.25],
[ 43,  46,  45]    [0.17, 0.18, 0.18],
[ 50,  48,  43]    [0.2 , 0.19, 0.17]
```

3 切分驗證集



4 資料集擴張

正確率↑ 過擬合↓

隨機切變
shear_range

隨機縮放
zoom_range

隨機翻轉
horizontal_flip

照片前處理

- (2) 生成「訓練集」&「驗證集」

```
1  train_set = train_datagen.flow_from_directory(
2    1 directory=train_dir,
3    2 target_size=(64, 64),
4    3 batch_size=10,
5    4 class_mode="binary",
6    5 subset="training")
```

```
1  val_set = train_datagen.flow_from_directory(
2      directory=train_dir,
3      target_size=(64, 64),
4      batch_size=10,
5      class_mode="binary",
6    6 subset="validation")
```

1. directory: 該資料集來源路徑
2. 統一縮放至 64x64
3. 每次轉換 10 張照片（避免 CPU 塞住）
4. 二元分類（cats 資料夾=0，dogs 資料夾=1）
   "categorical" → One-Hot Encoding
5. 這一批（75%）命名為 "Training"

6. 這一批（25%）命名為 "validation"

照片前處理

- (3) 生成「測試集」

```
1  test_datagen = ImageDataGenerator(rescale=1.0/255)
2
3  test_set = test_datagen.flow_from_directory(
4      directory=test_dir,
5      target_size=(64, 64),
6      batch_size=10,
7      class_mode="binary")
```

測試集不需做任何「影像擴增」
（Image Augmentation）

除了來源資料夾不同，
其餘參數都相同

隨堂練習：照片前處理

- 請輸入前幾頁的程式碼，並執行看看。
- 完整程式碼如下所示：

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(
    rescale=1.0/255,
    validation_split=0.25,
    shear_range=10,
    zoom_range=0.2,
    horizontal_flip=True)

train_set = train_datagen.flow_from_directory(
    directory=train_dir,
    target_size=(64, 64),
    batch_size=10,
    class_mode="binary",
    subset="training")
```

```
val_set = train_datagen.flow_from_directory(
    directory=train_dir,
    target_size=(64, 64),
    batch_size=10,
    class_mode="binary",
    subset="validation")

test_datagen = ImageDataGenerator(rescale=1.0/255)

test_set = test_datagen.flow_from_directory(
    directory=test_dir,
    target_size=(64, 64),
    batch_size=10,
    class_mode="binary")
```

驗證照片載入成功

- 原始程式碼

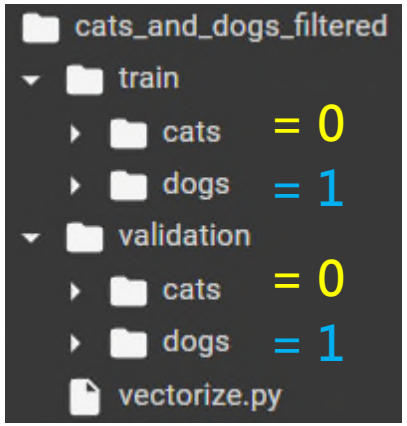
每呼叫一次，會產生 10 張轉換過的照片

```python
import HappyML.model_drawer as md

# Print the first 10 images of Training Set
for data, label in train_set:
    md.show_first_n_images(x_ary=data, y_real=label, first_n=10, font_size=12)
    break

# Print the first 10 images of Testing Set
for data, label in test_set:
    md.show_first_n_images(x_ary=data, y_real=label, first_n=10, font_size=12)
    break
```

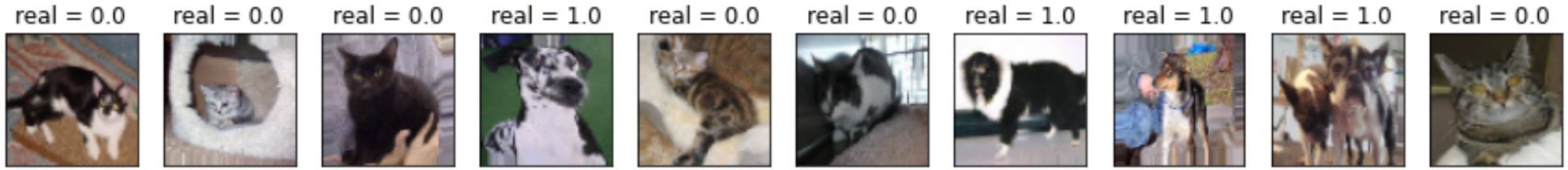
迴圈轉一圈（10 張照片）就停

隨堂練習：驗證照片載入成功

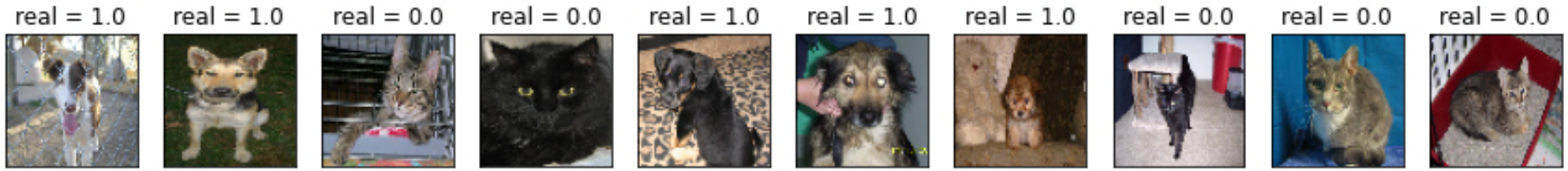
- 請輸入前一頁投影片的程式碼，並執行看看。
- 完成後，應該可以看到如下類似的結果：



訓練集（Training Set）



測試集（Testing Set）



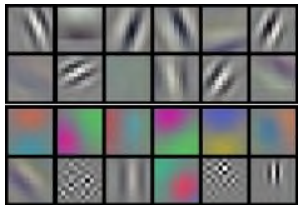
建造卷積神經網路

- (1) 輸入層 & 第一隱藏層

```
1  from tensorflow.keras.models import Sequential
2  from tensorflow.keras import layers
3
4  # CNN Model
5  model = Sequential()
6
7  # Input & 1st Hidden Layer: 1 nvolutional 2 Max Pooling                    3
8  model.add(layers.Conv2D(filters=32, kernel_size=(3, 3), activation="relu", padding="same", input_shape=[64, 64, 3]))
9  model.add(layers.Dropout(0.25))4
10 model.add(layers.MaxPool2D(pool_size=(2, 2), strides=2))5
```

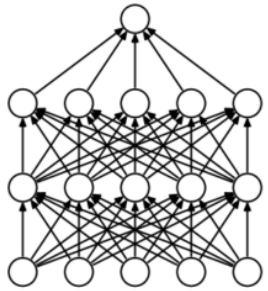
輸入
64x64 x RGB

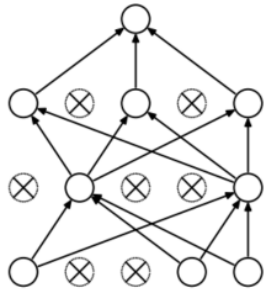
1 系統挑選 32 種濾鏡　2 每個濾鏡大小 3x3　3 不足外圍補 0　4 丟棄25%防止過擬合　5 最大池化2x2 x 2 步



1	1	1
0	0	0
-1	-1	-1

0	0	0	0	0	0	0
0	60	113	56	139	85	0
0	73	121	54	84	128	0
0	131	99	70	129	127	0
0	80	57	115	69	134	0
0	104	126	123	95	130	0
0	0	0	0	0	0	0





1	3	2	9
7	4	1	5
8	5	2	3
4	2	1	4

建造卷積神經網路

- (2) 第二、第三隱藏層

=32

```
12  # 2nd Hidden Layer: Convolutional + Max Pooling
13  model.add(layers.Conv2D(filters=64, kernel_size=(3, 3), activation="relu", padding="same"))
14  model.add(layers.Dropout(0.25))
15  model.add(layers.MaxPool2D(pool_size=(2, 2), strides=2))
16
17  # 3rd Hidden Layer: Convolutional + Max Pooling
18  model.add(layers.Conv2D(filters=128, kernel_size=(3, 3), activation="relu", padding="same"))
19  model.add(layers.Dropout(0.25))
20  model.add(layers.MaxPool2D(pool_size=(2, 2), strides=2))
```

“通常”卷積層越後面，濾鏡越多
32 → 64 → 128...
（要求系統學的細節越多）

建造卷積神經網路

- (3) 展平層、全連接層、輸出層

```
22  # Flatten layer
23  model.add(layers.Flatten())
24
25  # Fully Connected Layer
26  model.add(layers.Dense(units=512, activation="relu"))
27
28  # Output layer
29  model.add(layers.Dense(units=1, activation="sigmoid"))
30
31  # Compile
32  model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy"])
```

建造卷積神經網路

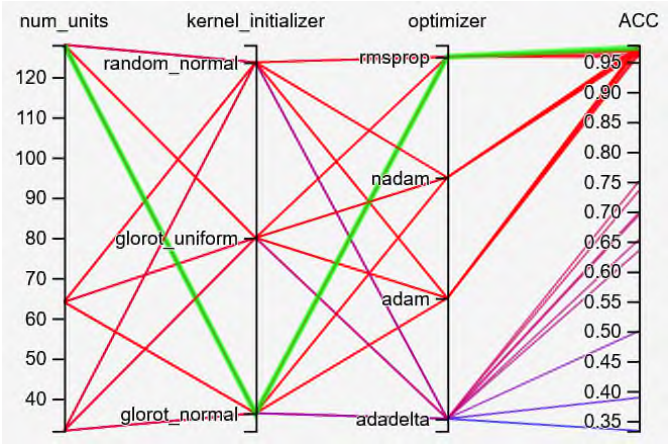
- 該有幾層？參數怎麼下？

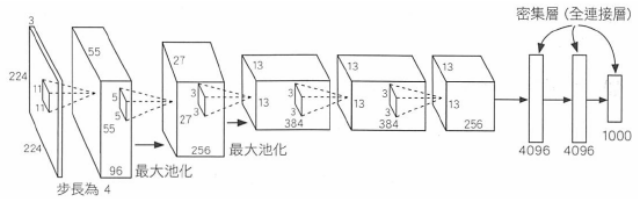
沒有規則！

用 TensorBoard 窮舉

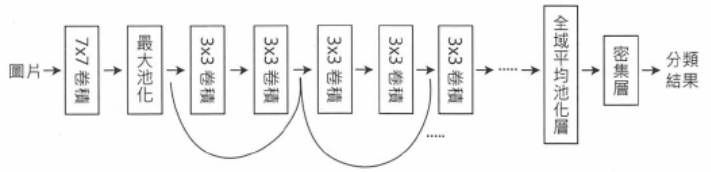
或

直接使用各種有效配方





AlexNet



ResNet

隨堂練習：建造卷積神經網路

- 請輸入前幾頁的程式碼，並且執行看看。
- 執行完畢之後，請用 model.summary() 檢查是否建造成功？
- 輸出應如下所示：

```
In [10]: model.summary()
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 64, 64, 32)        896
_________________________________________________________________
dropout (Dropout)            (None, 64, 64, 32)        0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 64)        18496
_________________________________________________________________
dropout_1 (Dropout)          (None, 32, 32, 64)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 128)       73856
_________________________________________________________________
dropout_2 (Dropout)          (None, 16, 16, 128)       0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 8192)              0
_________________________________________________________________
dense (Dense)                (None, 512)               4194816
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 513
=================================================================
Total params: 4,288,577
Trainable params: 4,288,577
Non-trainable params: 0
_________________________________________________________________
```

模型訓練＆調整

- TensorBoard 設定與啟動

```
# Load the TensorBoard notebook extension
%load_ext tensorboard

# Create TensorBoard log directory
import os
from datetime import datetime
from tensorflow.keras.callbacks import TensorBoard

logdir = os.path.join("logs", datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = TensorBoard(logdir, histogram_freq=1)

# Start the TensorBoard
%tensorboard --logdir logs
```

模型訓練&調整

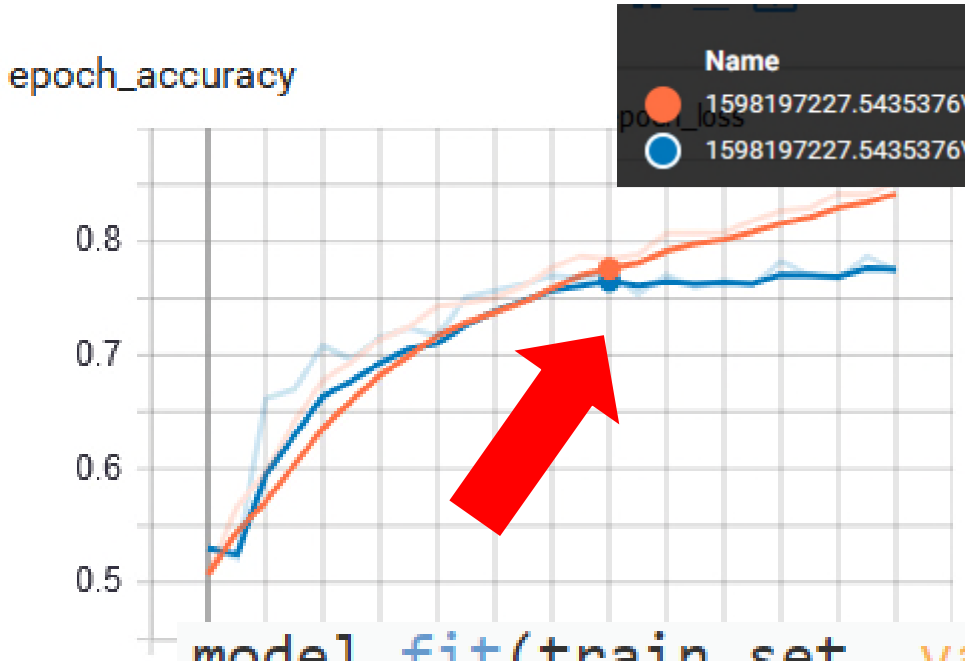
- 模型訓練

```
1  # Training Model
2  model.fit(train_set, validation_data=val_set, batch_size=10, epochs=25, callbacks=[tensorboard_callback])
```

```
Epoch 19/25
600/600 [==============================] - 20s 33ms/step - loss: 0.4169 - accuracy: 0.8080 - val_loss: 0.4903 - val_accuracy: 0.7655
Epoch 20/25
600/600 [==============================] - 20s 33ms/step - loss: 0.4049 - accuracy: 0.8185 - val_loss: 0.4959 - val_accuracy: 0.7615
Epoch 21/25
600/600 [==============================] - 20s 33ms/step - loss: 0.3910 - accuracy: 0.8270 - val_loss: 0.4576 - val_accuracy: 0.7825
Epoch 22/25
600/600 [==============================] - 20s 33ms/step - loss: 0.3753 - accuracy: 0.8297 - val_loss: 0.4716 - val_accuracy: 0.7700
Epoch 23/25
600/600 [==============================] - 20s 33ms/step - loss: 0.3591 - accuracy: 0.8423 - val_loss: 0.4759 - val_accuracy: 0.7670
Epoch 24/25
600/600 [==============================] - 20s 33ms/step - loss: 0.3566 - accuracy: 0.8412 - val_loss: 0.4467 - val_accuracy: 0.7870
Epoch 25/25
600/600 [==============================] - 20s 33ms/step - loss: 0.3365 - accuracy: 0.8530 - val_loss: 0.4650 - val_accuracy: 0.7740
```

模型訓練＆調整

- 找到合適的訓練週期（Epochs）



Name	Smoothed	Value	Step	Time	Relative
1598197227.5435376\train	0.7761	0.7848	14	Sun Aug 23, 23:45:45	4m 36s
1598197227.5435376\validation	0.7652	0.7715	14	Sun Aug 23, 23:45:45	4m 36s

0 ~ 14 → 15 個週期就夠了！

```
model.fit(train_set, validation_data=val_set, epochs=15)
```

隨堂練習：模型訓練&調整

- 請將前幾頁的程式碼寫好，並執行看看。
- 看完找到最佳訓練週期之後，請修改訓練週期，並重新訓練模型。
- 透過 HParams 找到其它超參數最佳值的方法，就留給各位自行練習。

```
# Load the TensorBoard notebook extension
%load_ext tensorboard

# Create TensorBoard log directory
import os
from datetime import datetime
from tensorflow.keras.callbacks import TensorBoard

logdir = os.path.join("logs", datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = TensorBoard(logdir, histogram_freq=1)

# Start the TensorBoard
%tensorboard --logdir logs
```

```
# Training Model
model.fit(train_set, validation_data=val_set, batch_size=10, epochs=25, callbacks=[tensorboard_callback])
```

模型預測

- 程式碼

```
1  import numpy as np
2  import HappyML.model_drawer as md
3
4  Y_pred = np.rint(model.predict(test_set))
5
6  for data, label in test_set: ← 一次產生十筆
7      md.show_first_n_images(x_ary=data, y_real=label, y_pred=Y_pred[:10], first_n=10, font_size=12)
8      break ← 只做一圈（十筆）就停
```

↑ 取前十筆丟進去

model.predict()

```
[[0.8474125 ],
 [0.5022847 ],
 [0.66769755],
 ...,
 [0.7356534 ],
 [0.43187016],
 [0.07690324]]
```

np.rint()

```
[[0.0],
 [1.0],
 [0.0],
 ...,
 [0.0],
 [0.0],
 [1.0]]
```

猜錯 x3

隨堂練習：模型預測

- 請將前幾頁的程式碼寫好，並執行看看。
- 您的模型猜對了幾張圖片呢？

模型評估

- 程式碼

```
1  test_loss, test_acc = model.evaluate(test_set)
2  print("Loss of Test:", test_loss)
3  print("Accuracy of Test:", test_acc)
```

```
In [13]: runcell('[] 模型評估', 'D:/demo/DeepLearning/Ch05 CNN/02cnn_catsdogs.py')
200/200 [==============================] - 7s 34ms/step - loss: 0.4394 - accuracy: 0.8030
Loss of Test: 0.439406216144456177
Accuracy of Test: 0.802999732971191
```

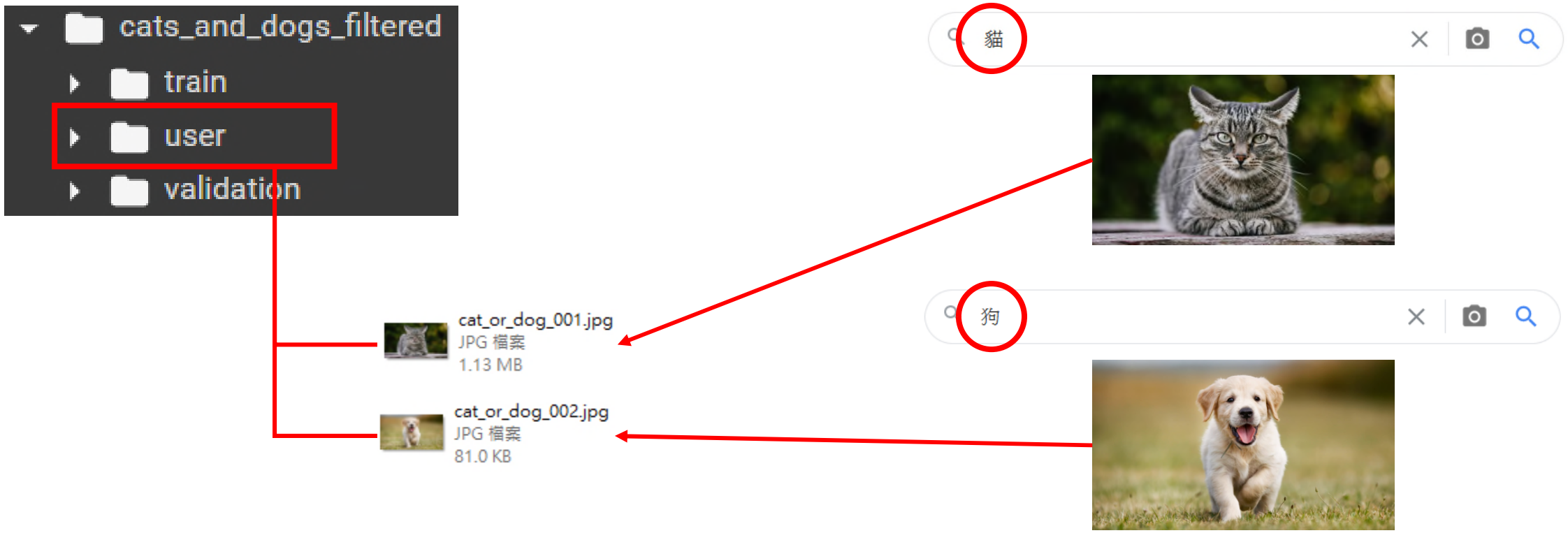
≒ 80.30%

隨堂練習：模型評估

- 請將前幾頁的程式碼寫好，並執行看看。
- 您的模型套用在測試集上，正確率大概有多少呢？

預測使用者提供的圖片

- 搜尋屬於你自己的「貓」、「狗」照片



預測使用者提供的圖片

- 程式碼

① 載入需要的套件

```
1  from tensorflow.keras.preprocessing import image
2
3  user_image = image.load_img("cats_and_dogs_filtered/user/cat_or_dog_001.jpg", target_size=(64, 64))
4  image_data = image.img_to_array(user_image) / 255
5  image_data = np.expand_dims(image_data, axis=0)
6
7  isDog = (model.predict(image_data) >= 0.5)
8  print("Dog" if isDog[0][0] else "Cat")
9
10 import matplotlib.pyplot as plt
11 plt.imshow(user_image)
12 plt.show()
```

② 將指定圖片，以 64x64 大小，載入到記憶體中

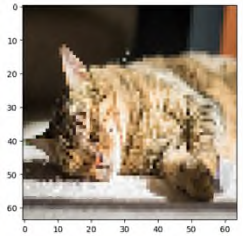
③ 將 0/1 的圖片，轉成 0.0~1.0 x RGB 色階表示

④ 將維度，從 (64, 64) → (1, 64, 64)，以符合訓練集

⑤ 預測結果。如：[[0.8754]] 是狗， [[0.8754]]>=0.5 → True

⑥ 印出預測結果

⑦ 將圖片印出來驗證



可以將 cat_or_dog_001.jpg，改成 cat_or_dog_002.jpg 再試一次

隨堂練習：預測使用者的圖片

- 請先在照片資料夾下，建立一個新資料夾，叫做 user。
- 到 Google 圖片搜尋，找到「貓」或「狗」圖片若干。
- 將它們依序命名如下，放入 user 資料夾中：
  - cat_or_dog_001.jpg
  - cat_or_dog_002.jpg
  - …
- 將前一頁的程式碼寫好，執行看看。
- 更換不同檔名，重新執行程式看看。
- 您的模型能猜對多少張照片呢？

補充影片

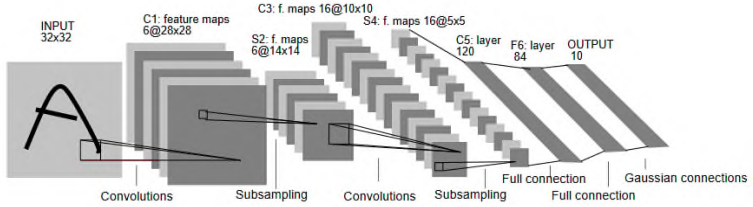
常見的卷積神經網路經典模型

何謂 CNN 經典模型

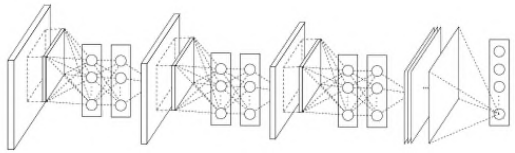
CNN 界的成藥

- 經過專家研究、對特定問題有效的 CNN 層級&參數

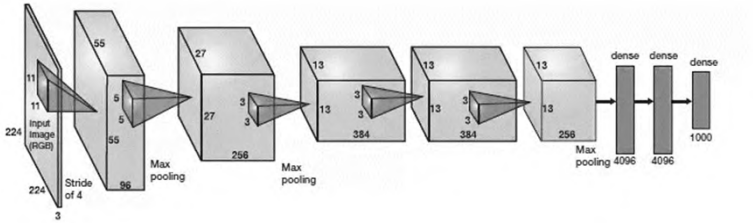
LeNet (1998)



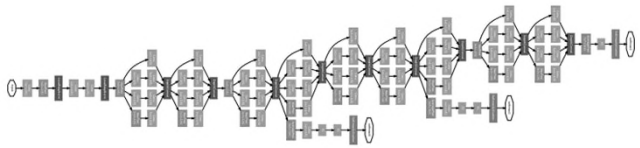
NiN (2014)



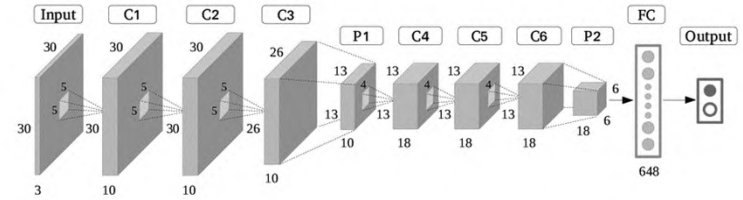
AlexNet (2012)



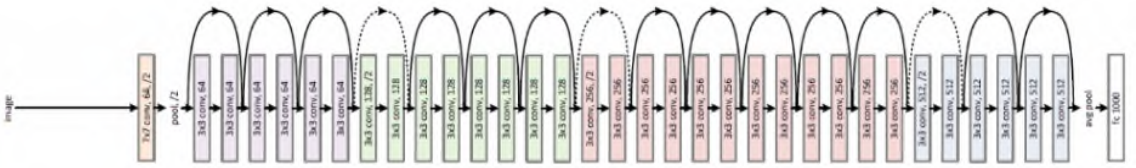
GoogLeNet (2014)



VGGNet (2014)

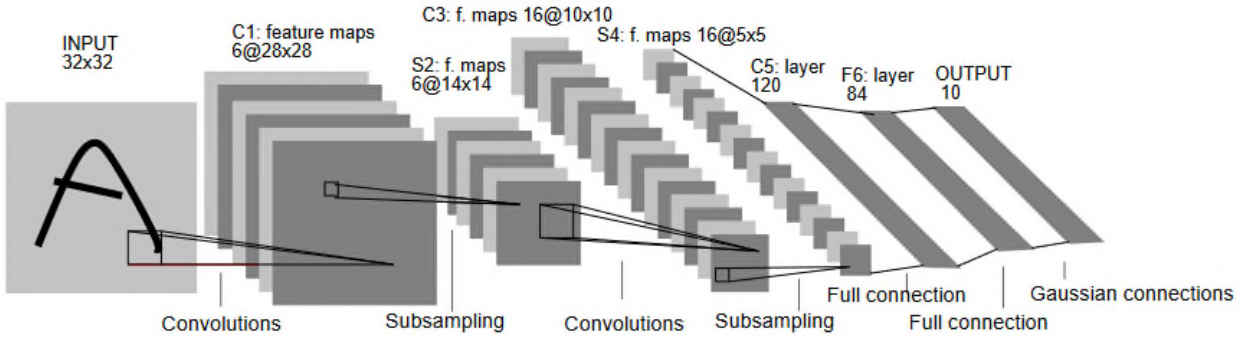


ResNet (2015)



LeNet

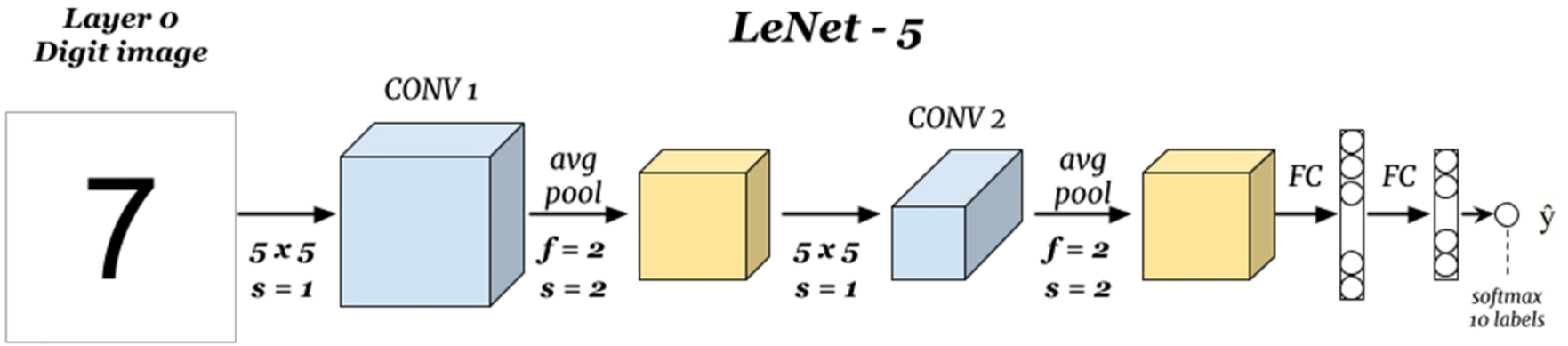
- 簡介



- 發表時間：1998
- 發表者
  - Yann LeCun (CNN 之父)
- 貢獻
  - 最早的 CNN 啟蒙模型
- 用途
  - 辨識 32x32 大小的圖片

LeNet

- 模型


Layer 0
Digit image
LeNet - 5
7
CONV 1
5 x 5
s = 1
avg
pool
f = 2
s = 2
CONV 2
5 x 5
s = 1
avg
pool
f = 2
s = 2
FC
FC
ŷ
softmax
10 labels
32 x 32 x 1
28 x 28 x 6
14 x 14 x 6
10 x 10 x 16
5 x 5 x 16
120
84
輸入層
卷積層
池化層
卷積層
池化層
全連接層
全連接層
輸出層
32x32 圖片
濾鏡：6 個
大小：5x5
步長：1 步
類型：平均
大小：2x2
步長：2 步
濾鏡：16 個
大小：5x5
步長：1 步
類型：平均
大小：2x2
步長：2 步
120個
84個
10類

LeNet

- 範例程式碼

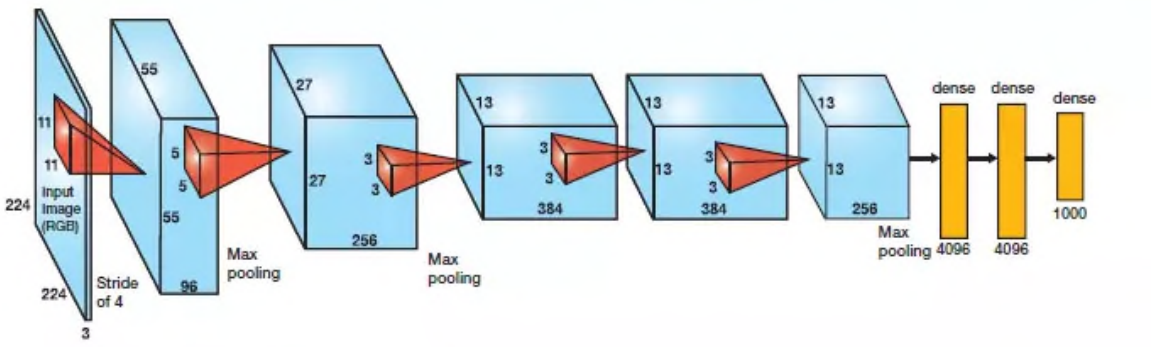
```
from keras.models import Sequential
from keras.layers import Dense,Conv2D,AveragePooling2D,Flatten

batch_size = 32
epoch = 20

model = Sequential()
model.add(Conv2D(32,(5,5),strides=(1,1),input_shape=(28,28,1),padding='valid',activation='sigmoid',kernel_initializer='uniform'))
model.add(AveragePooling2D(pool_size=(2,2)))
model.add(Conv2D(64,(5,5),strides=(1,1),padding='valid',activation='sigmoid',kernel_initializer='uniform'))
model.add(AveragePooling2D(pool_size=(2,2)))
model.add(Flatten())
model.add(Dense(100,activation='sigmoid'))
model.add(Dense(10,activation='softmax'))
model.compile(optimizer='sgd',loss='categorical_crossentropy',metrics=['accuracy'])
model.summary()

model.fit(train_x,train_y,validation_data=(valid_x,valid_y),batch_size=batch_size,epochs=epoch,verbose=1)
print(model.evaluate(test_x,test_y,batch_size=batch_size,verbose=1))
```

LeNet

- **優點**
  - 簡單、易理解
- **缺點**
  - 使用「平均池化」，明顯特徵會被周遭沖淡，無法突出。
  - 使用「Sigmoid」，層數一多，容易有「梯度消失」問題。
- **何時用**
  - 懷舊、實驗、做為基準範例（baseline case）
- **何時不要用**
  - 圖片超過 32x32 大小時

AlexNet

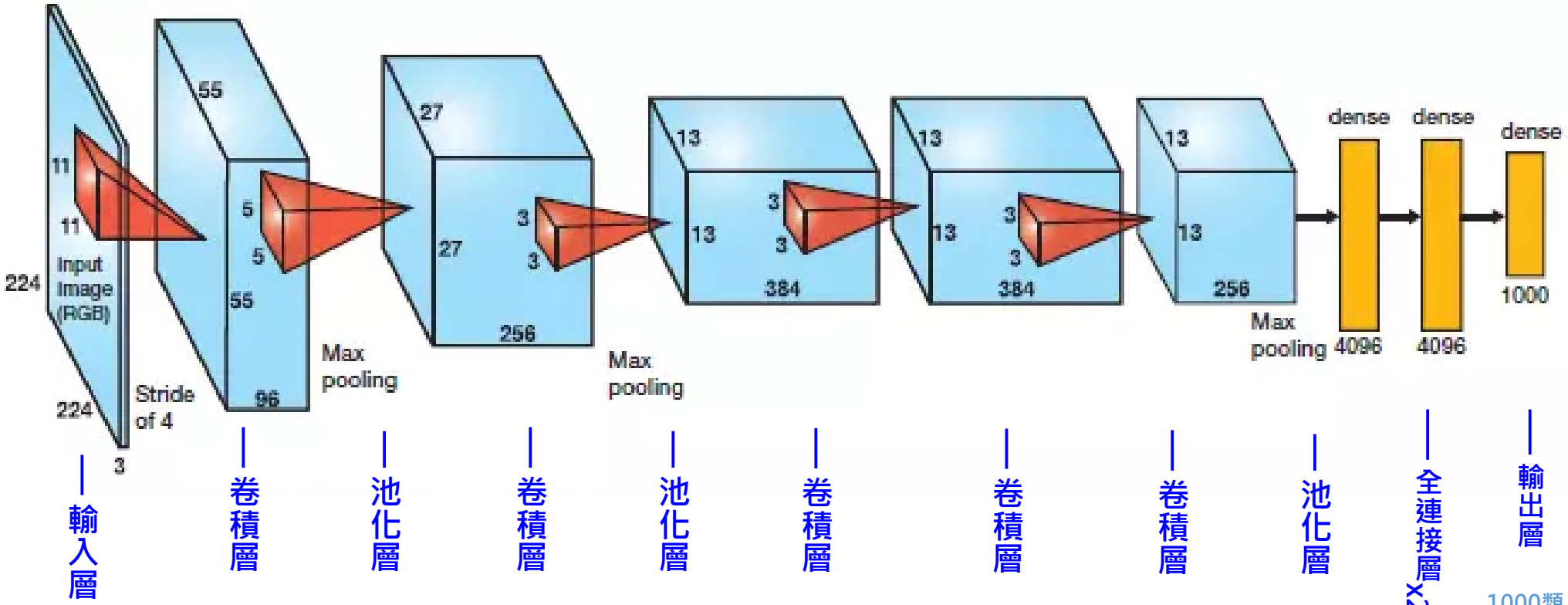
- 簡介



- **發表時間**：2012
- **發表者**
  - **Alex** Krizhevsky
  - 2012 ImageNet 大賽冠軍得主
- **貢獻**
  - 提出「資料擴張」、「丟棄層」
- **用途**
  - 辨識 224x224 大小彩色圖片

AlexNet

- 模型：卷積層 x5 + 全連接層 x3



AlexNet

- 程式碼

```
from keras.models import Sequential
from keras.layers import Dense,Conv2D,MaxPooling2D,Flatten,Dropout

model = Sequential()
model.add(Conv2D(96,(11,11),strides=(4,4),input_shape=(227,227,3),padding='valid',activation='relu',kernel_initializer='uniform'))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))
model.add(Conv2D(256,(5,5),strides=(1,1),padding='same',activation='relu',kernel_initializer='uniform'))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))
model.add(Conv2D(384,(3,3),strides=(1,1),padding='same',activation='relu',kernel_initializer='uniform'))
model.add(Conv2D(384,(3,3),strides=(1,1),padding='same',activation='relu',kernel_initializer='uniform'))
model.add(Conv2D(256,(3,3),strides=(1,1),padding='same',activation='relu',kernel_initializer='uniform'))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))
model.add(Flatten())
model.add(Dense(4096,activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(4096,activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1000,activation='softmax'))
model.compile(loss='categorical_crossentropy',optimizer='sgd',metrics=['accuracy'])
model.summary()
```

AlexNet

- **優點**
  - 引入「**資料擴張**」、「**丟棄層**」概念，減少**過擬合**機會。
  - 用「**最大池化**」取代「**平均池化**」，更能留下重要特徵。
  - 池化**步長** < 池化**視窗大小**，可**重複檢視**各特徵，確保沒有遺漏。
  - 用 **ReLU** 取代 **Sigmoid**，減低「梯度消失」問題。

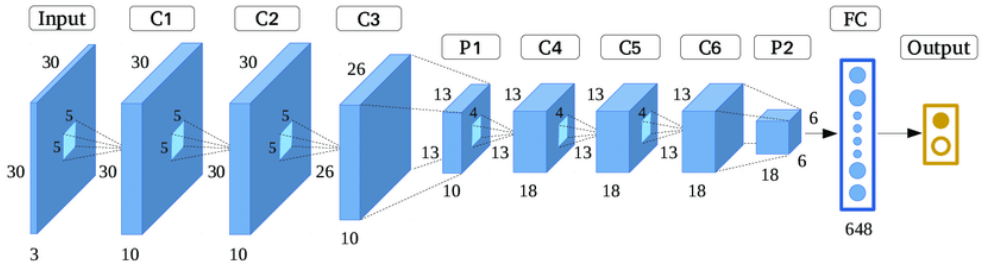
- **缺點**
  - 除了耗費記憶體有點多（參數多）外，無重大缺點。

- **何時用**
  - 大型彩色圖片（224x224）

- **何時不要用**
  - 系統記憶體不足、在意訓練速度時。

VGGNet

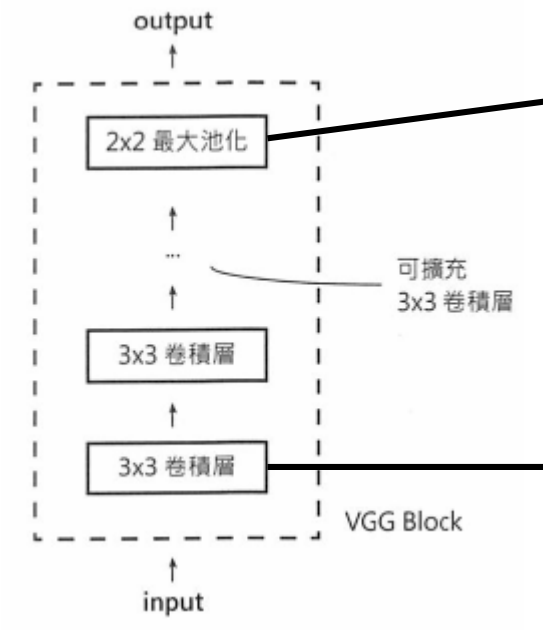
- **簡介**



- **發表時間**：2014
- **發表者**
  - 牛津大學 Visual Geometry Group (VGG) 實驗室
- **貢獻**
  - 提出「**VGG Block**」組件概念。
  - 提出「**多個小卷積**」取代「**一個大卷積**」概念。參數變少，效率變好。
- **用途**
  - 辨識 224x224 大小彩色圖片

VGGNet

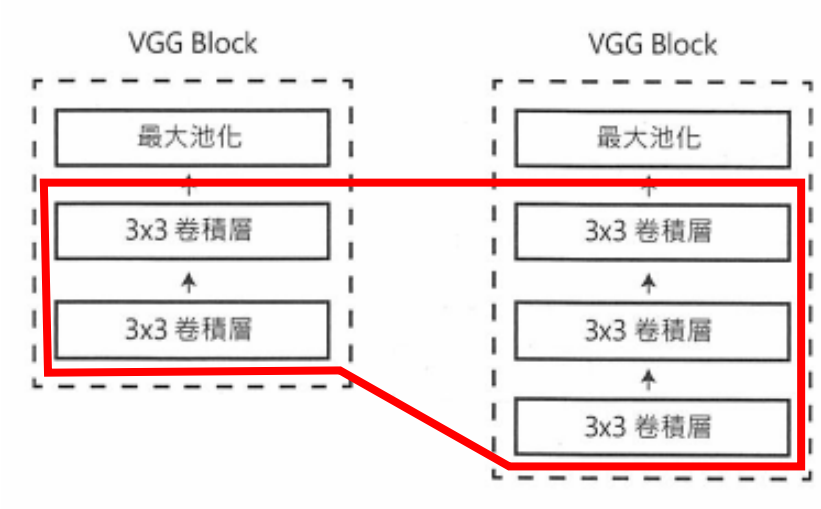
- 何謂「VGG Block」（VGG 組件）



- **池化窗口**：2x2（固定）
- **滑動步長**：2 步

- **濾鏡數目**：N（可自訂）
- **濾鏡大小**：3x3（固定）
- **Padding**：Same（維持維度）
- **滑動步長**：1 步
- **激活函數**：ReLU

一個「VGG 組件」



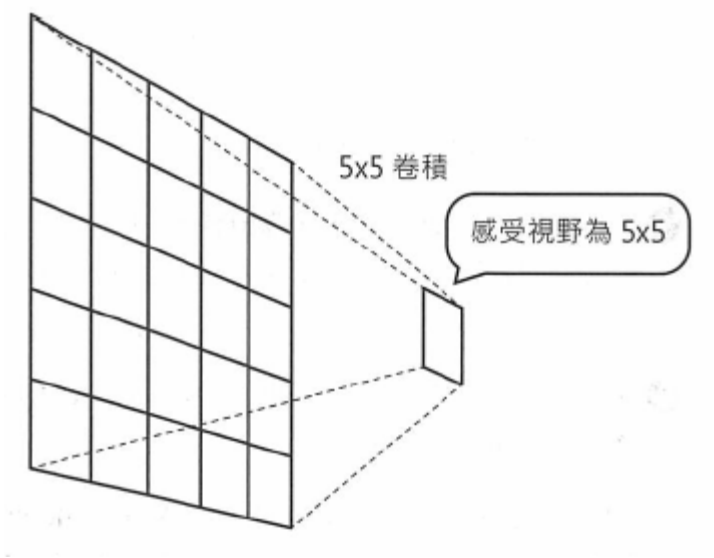
卷積層個數可自訂

自行定義各種不同的「VGG 組件」

VGGNet

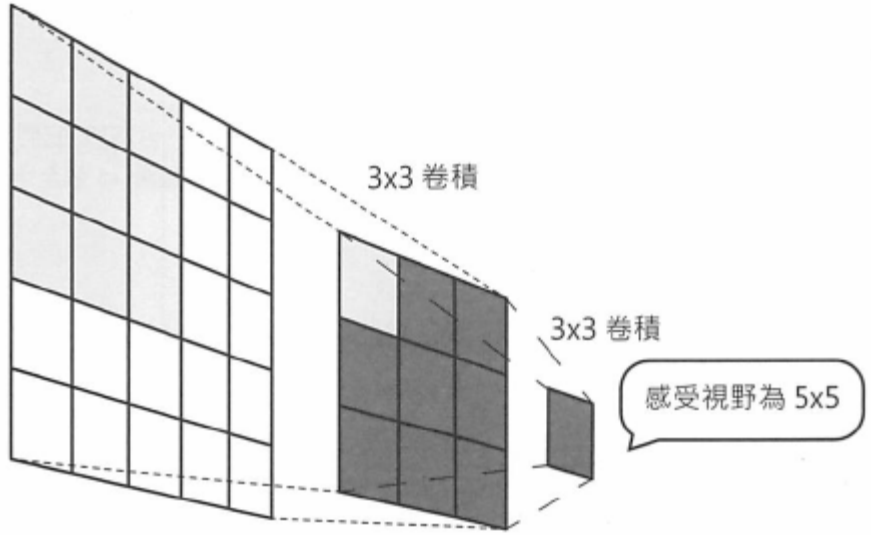
- 重要貢獻：「多個小卷積」優於「一個大卷積」

5x5 卷積 x1

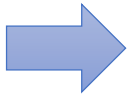


- 視野=5x5 → 5x5 卷積 → 1x1 格
- 5x5 卷積參數量 = 5x5 = 25 個

3x3 卷積 x2



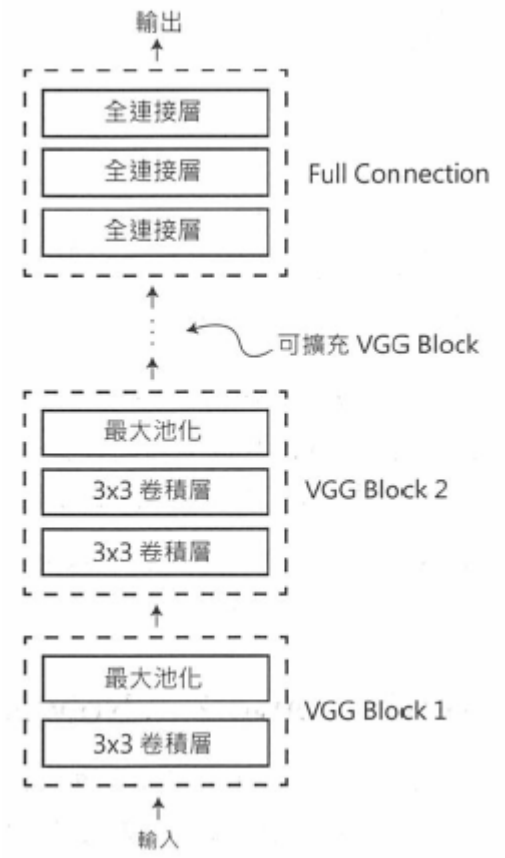
- 視野=5x5 → 3x3 卷積 → 3x3 格
- 視野=3x3 → 3x3 卷積 → 1x1 格
- (3x3 卷積)x2 參數量 = (3x3)x2 = 18 個



- 深度增加，抽象程度增加！
- 參數量變少，計算量減優！

VGGNet

一個完整的 VGGNet 範例



不同的 VGGNet 配置
(VGG11、13、16、19)

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

VGGNet

- 程式碼

VGG16

```
1  from tensorflow.keras.applications import VGG16
2
3  model = VGG16()
4  model.summary()
```

VGG19

```
1  from tensorflow.keras.applications import VGG19
2
3  model = VGG19()
4  model.summary()
```

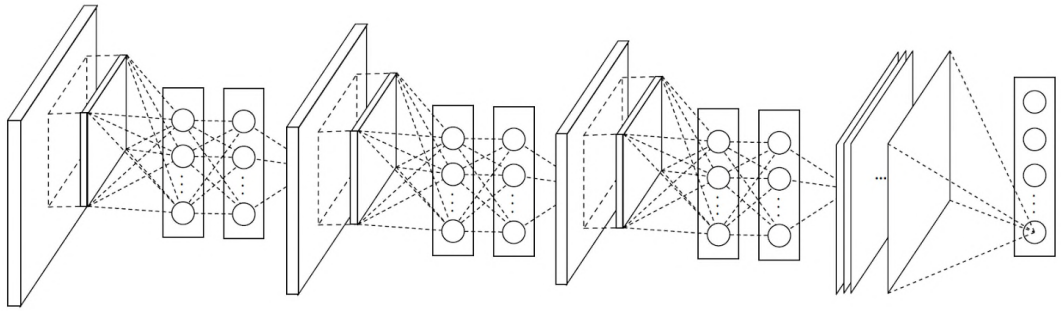
- 其它模式：自己寫！

VGGNet

- **優點**
  - 靈活、好調整、能處理多種情況。
- **缺點**
  - 網路深，參數多，訓練時間長。
- **何時用**
  - 想要擁有靈活、客製化的 CNN 網路架構時。
- **何時不要用**
  - 機器效能差、不希望訓練時間太長時。

Network in Network (NiN)

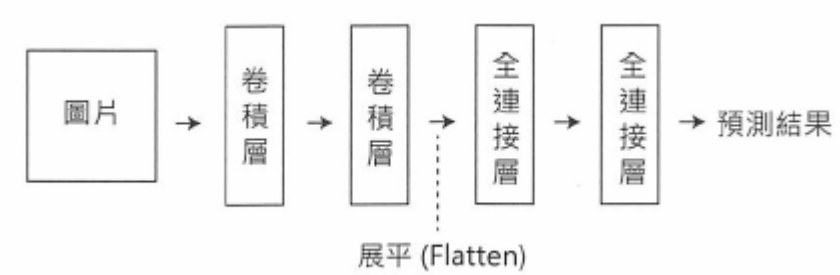
- 簡介



- 發表時間：2014
- 發表者
  - 新加坡大學 Min Lin（林敏）
- 貢獻
  - 提出 MLPConv（MLP 卷積層），透過 1x1 卷積層，做到「降維」（減低參數）。
  - 提出「全域平均池化」代替「全連接層」，減低大量參數。
- 用途
  - 將參數降低 10 倍以上，且精準度沒有喪失。

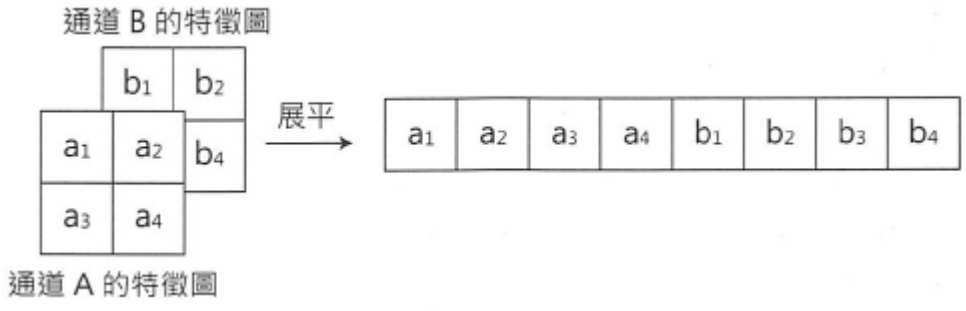
Network in Network (NiN)

- 一般 CNN 架構的缺點



一個一般的 CNN 架構

為了取得「跨通道」特徵而「展平」，讓特徵混雜在一起




```
In [10]: model.summary()
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 64, 64, 32)        896
_________________________________________________________________
dropout (Dropout)            (None, 64, 64, 32)        0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 64)        18496
_________________________________________________________________
dropout_1 (Dropout)          (None, 32, 32, 64)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 128)       73856
_________________________________________________________________
dropout_2 (Dropout)          (None, 16, 16, 128)       0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 8192)              0
_________________________________________________________________
dense (Dense)                (None, 512)               4194816
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 513
=================================================================
Total params: 4,288,577
Trainable params: 4,288,577
Non-trainable params: 0
_________________________________________________________________
```

卷積層參數
= 93221 (2.17%)

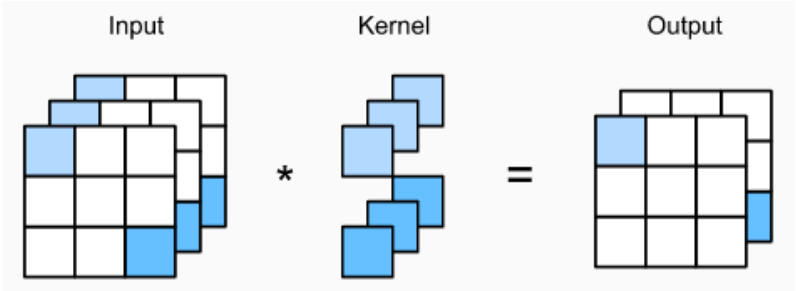
全連接層參數
= 4194816 (97.83%)

「全連接層」是天底下最耗費參數的一層

Network in Network (NiN)

- NiN 的思路
  - 能夠「不展平」，就能取得「跨通道」特徵嗎？
  - 「全連接層」有不要那麼多參數的替代品嗎？

問題一解法：1x1 縱向卷積層

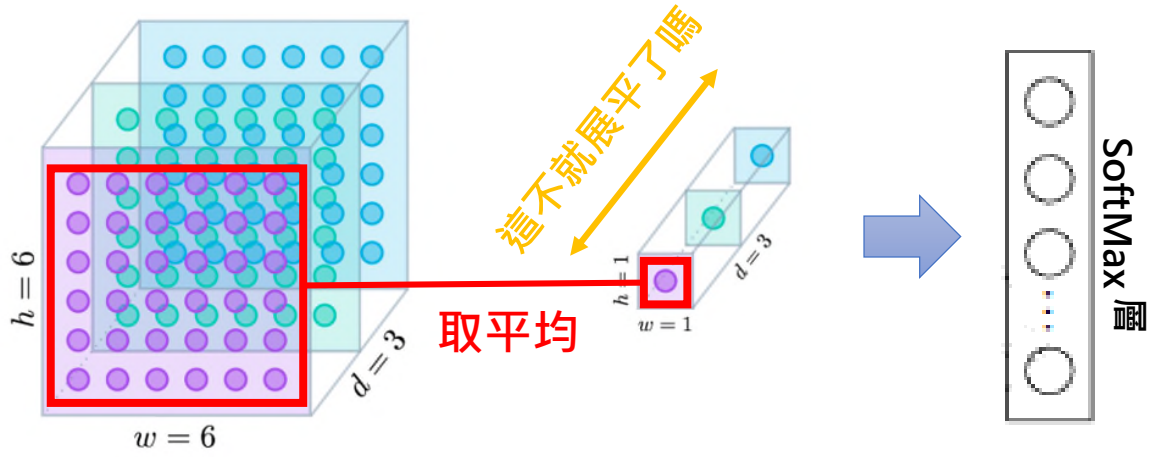


(3, 3) x 3 濾鏡特徵

- * 2 縱向濾鏡 = (3, 3) x 2 濾鏡特徵
- * 4 縱向濾鏡 = (3, 3) x 4 濾鏡特徵

縱向濾鏡 = 跨通道 + 升 / 降維

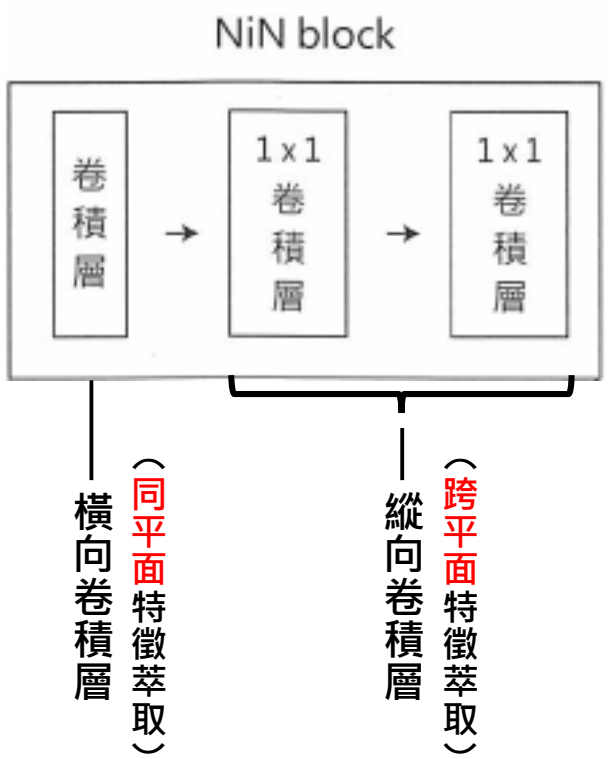
問題二解法：全域平均池化



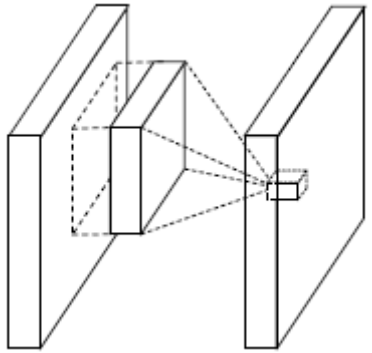
全域平均池化 = 通道不混雜 + 能展平

Network in Network (NiN)

- NiN Block（又稱 MLP 卷積層，MLPConv）

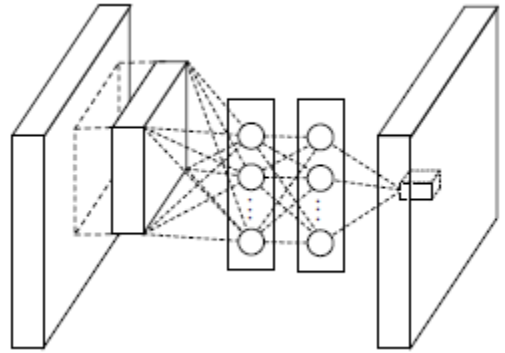


一般的 Conv



僅萃取同一通道之特徵
（同一通道 = R or G or B）

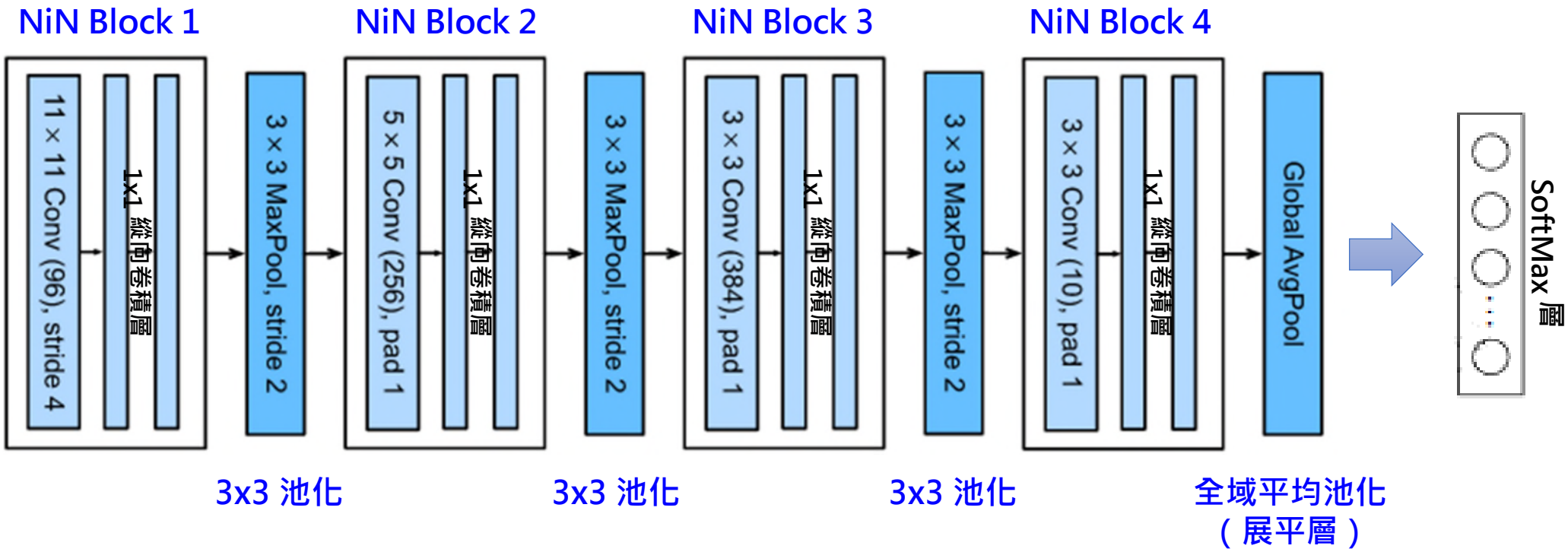
MLPConv



萃取同一通道 & 不同通道之特徵

Network in Network (NiN)

- NiN 完整模型



Network in Network (NiN)

- 程式碼

```
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers

def nin_block(num_channels, kernel_size, strides, padding):
    return Sequential([
        layers.Conv2D(num_channels, kernel_size, strides=strides,
                      padding=padding, activation='relu'),
        layers.Conv2D(num_channels, kernel_size=1, activation='relu'),
        layers.Conv2D(num_channels, kernel_size=1, activation='relu')])

def net():
    return Sequential([
        nin_block(96, kernel_size=11, strides=4, padding='valid'),
        layers.MaxPool2D(pool_size=3, strides=2),
        nin_block(256, kernel_size=5, strides=1, padding='same'),
        layers.MaxPool2D(pool_size=3, strides=2),
        nin_block(384, kernel_size=3, strides=1, padding='same'),
        layers.MaxPool2D(pool_size=3, strides=2),
        layers.Dropout(0.5),
        # There are 10 label classes
        nin_block(10, kernel_size=3, strides=1, padding='same'),
        layers.GlobalAveragePooling2D(),
        layers.Reshape((1, 1, 10)),
        # Transform the four-dimensional output into two-dimensional output
        # with a shape of (batch size, 10)
        layers.Flatten()])
```

1x1 縱向卷積

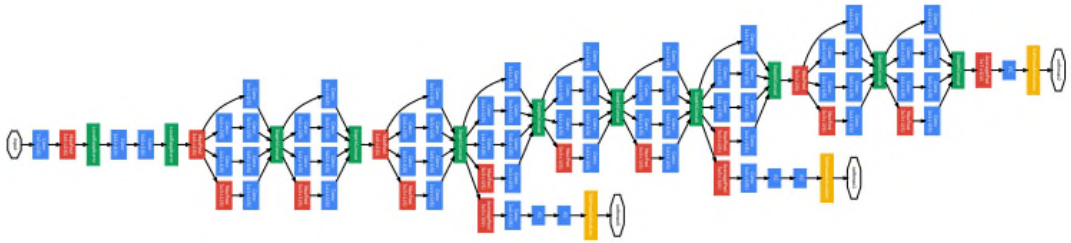
全域平均池化

Network in Network (NiN)

- **優點**
  - 使用 1x1 縱向卷積 & 全域平均池化，降低參數量。
  - 參數量下降 10 倍，準確率更高。
- **缺點**
  - 無非常明顯的缺點。
- **使用時機**
  - 實務上較少用，學術上的概念突破貢獻較大。

GoogLeNet

- 簡介

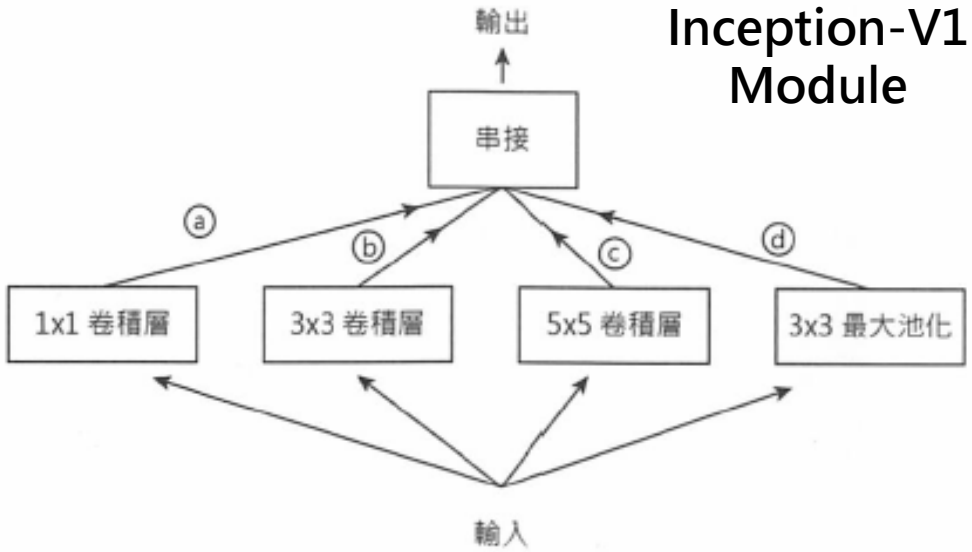


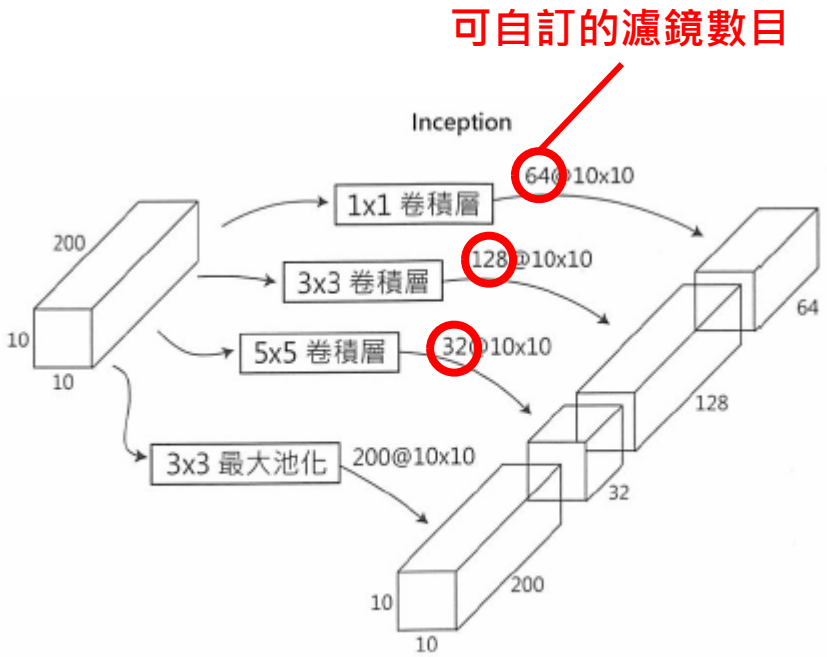
又稱「Inception-V1」

- 發表時間：2014
- 發表者
  - Google
- 貢獻
  - 利用 **Functional API 模型**，計算出最佳特徵。
- 用途
  - 將網路加深，但參數卻變少，準確率變高。

GoogLeNet

- 想法





不知道哪個大小的卷積層（1x1, 3x3, 5x5...）比較好

→ 用 Functional API 架構全都做，並把特徵「跨通道」加以混合

如何把特徵「跨通道」混合

GoogLeNet

- Inception-V1 Module 的程式碼

```
1  from tensorflow.keras import Input, Model
2  from tensorflow.keras.layers import Conv2D, MaxPooling2D, concatenate
3
4  ipt = Input(shape=(10,10,200))
5
6  ka, kb, kc = 64, 128, 32    # 自訂濾鏡數目（i.e. 卷積核數量、輸出通道數）
7  a = Conv2D(ka, kernel_size=1, padding='same', activation='relu')(ipt) # 分支 a
8  b = Conv2D(kb, kernel_size=3, padding='same', activation='relu')(ipt) # 分支 b
9  c = Conv2D(kc, kernel_size=5, padding='same', activation='relu')(ipt) # 分支 c
10 d = MaxPooling2D(pool_size=3, strides=1, padding='same')(ipt)          # 分支 d
11
12 opt = concatenate([a, b, c, d], axis=-1) # 串接 4 個分支的輸出
13
14 model = Model(ipt, opt)
15 model.summary()
```

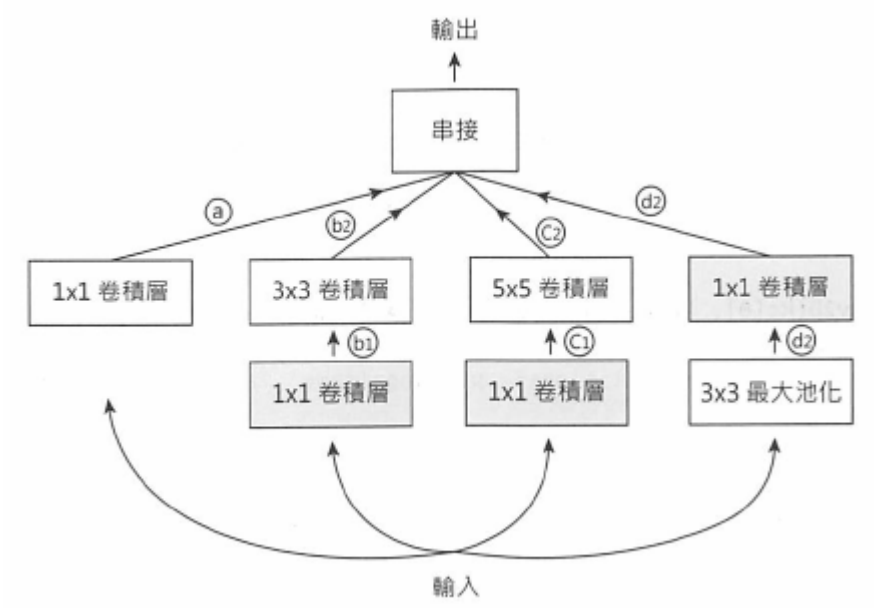
403,424

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 10, 10, 200)	0	
conv2d (Conv2D)	(None, 10, 10, 64)	12864	input_1[0][0]
conv2d_1 (Conv2D)	(None, 10, 10, 128)	230528	input_1[0][0]
conv2d_2 (Conv2D)	(None, 10, 10, 32)	160032	input_1[0][0]
max_pooling2d (MaxPooling2D)	(None, 10, 10, 200)	0	input_1[0][0]
concatenate (Concatenate)	(None, 10, 10, 424)	0	conv2d[0][0] conv2d_1[0][0] conv2d_2[0][0] max_pooling2d[0][0]

Total params: 403,424
Trainable params: 403,424
Non-trainable params: 0

GoogLeNet

- Inception Module-V1 缺點：參數仍嫌太多



改良版！用 1x1 縱向卷積層
降低維度

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 10, 10, 200)	0	
conv2d_1 (Conv2D)	(None, 10, 10, 64)	12864	input_1[0][0]
conv2d_3 (Conv2D)	(None, 10, 10, 16)	3216	input_1[0][0]
max_pooling2d (MaxPooling2D)	(None, 10, 10, 200)	0	input_1[0][0]
conv2d (Conv2D)	(None, 10, 10, 64)	12864	input_1[0][0]
conv2d_2 (Conv2D)	(None, 10, 10, 128)	73856	conv2d_1[0][0]
conv2d_4 (Conv2D)	(None, 10, 10, 32)	12832	conv2d_3[0][0]
conv2d_5 (Conv2D)	(None, 10, 10, 32)	6432	max_pooling2d[0][0]
concatenate (Concatenate)	(None, 10, 10, 256)	0	conv2d[0][0] conv2d_2[0][0] conv2d_4[0][0] conv2d_5[0][0]

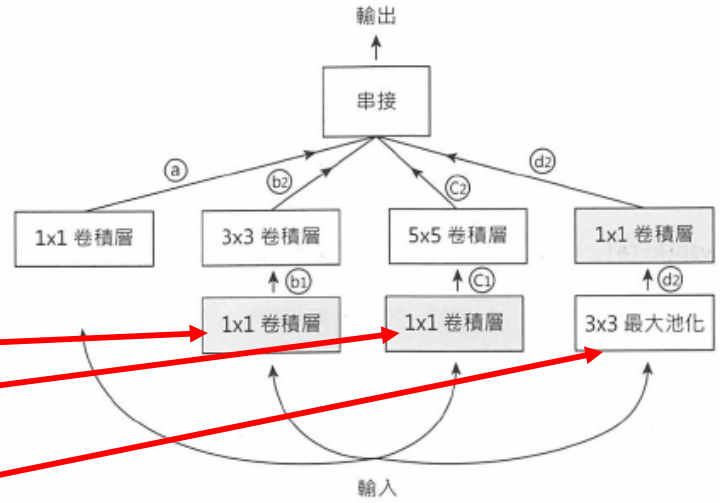
Total params: 122,064
Trainable params: 122,064
Non-trainable params: 0

403,424 → 122,064
下降 70% 左右！！

GoogLeNet

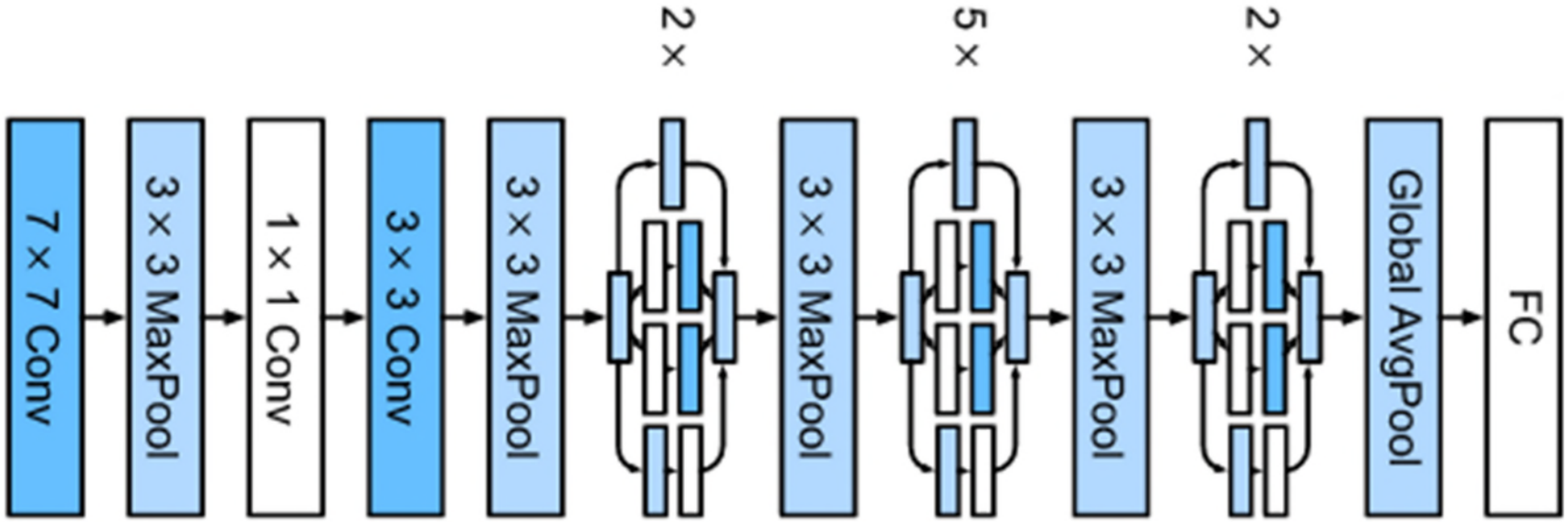
- Inception-V1 Module 改良程式碼

```
1  from tensorflow.keras import Input, Model
2  from tensorflow.keras.layers import Conv2D, MaxPooling2D, concatenate
3
4  ipt = Input(shape=(10,10,200))
5
6  ka, kb, kc, kd = 64, (64,128), (16,32), 32    # 設定輸出通道數 (卷積核數量)
7  a = Conv2D(ka, kernel_size=1, padding='same', activation='relu')(ipt) # 分支 a
8
9  b1 = Conv2D(kb[0], kernel_size=1, padding='same', activation='relu')(ipt) # 分支 b
10 b2 = Conv2D(kb[1], kernel_size=3, padding='same', activation='relu')(b1) # 分支 b
11
12 c1 = Conv2D(kc[0], kernel_size=1, padding='same', activation='relu')(ipt) # 分支 c
13 c2 = Conv2D(kc[1], kernel_size=5, padding='same', activation='relu')(c1) # 分支 c
14
15
16 d1 = MaxPooling2D(pool_size=3, strides=1, padding='same')(ipt)        # 分支 d
17 d2 = Conv2D(kd, kernel_size=1, padding='same', activation='relu')(d1) # 分支 d
18
19 output = concatenate([a, b2, c2, d2], axis=-1) # 串接 4 個分支的輸出
20
21 model = Model(ipt, output)
22 model.summary()
```



GoogLeNet

- 模型：一個完整的 GoogLeNet

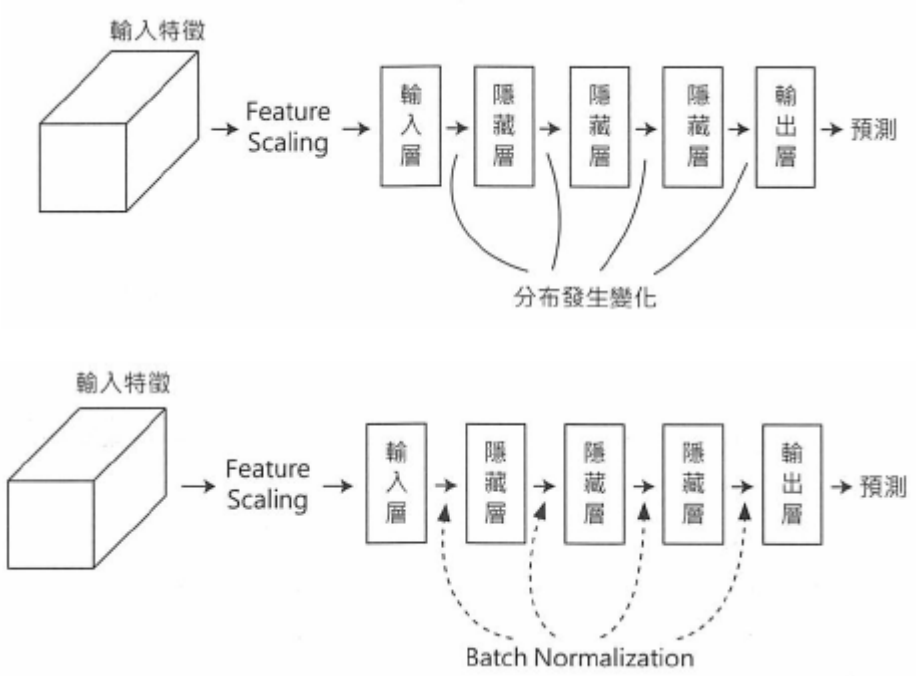


GoogLeNet

- Inception-V2（2015）：解決「梯度消失問題」&「削減參數」

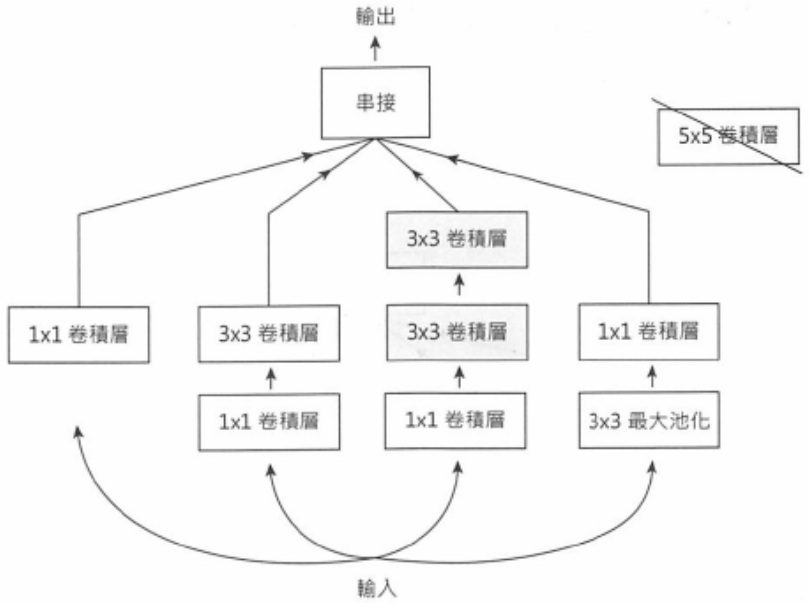
解決「梯度消失」

每層做完，將資料執行「平均=0，標準差=1」之正規化



「削減參數」

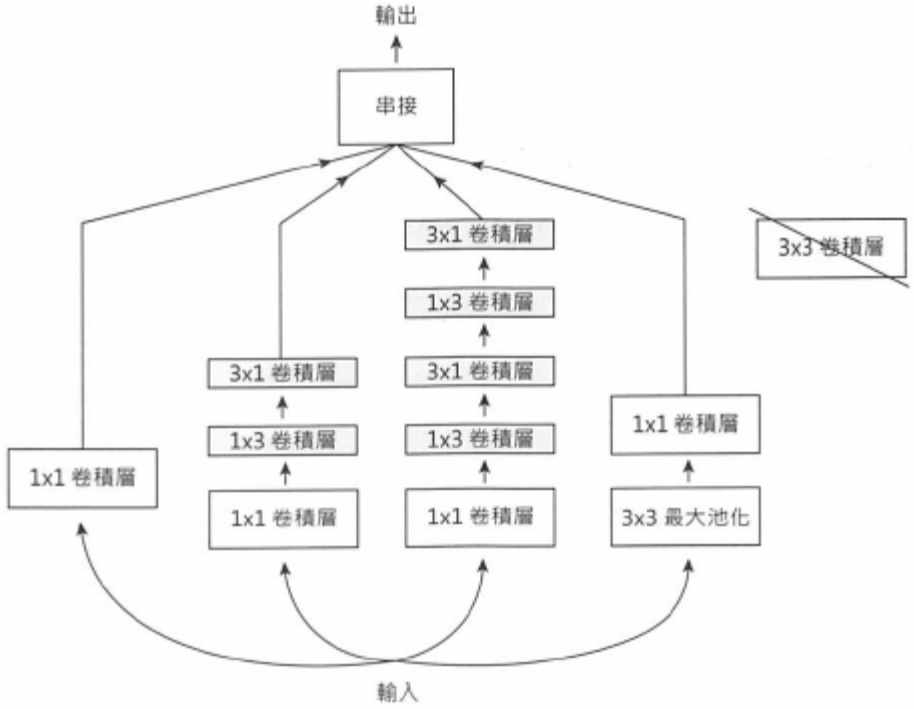
用兩個 3x3 小卷積層，代替一個 5x5 大卷積層



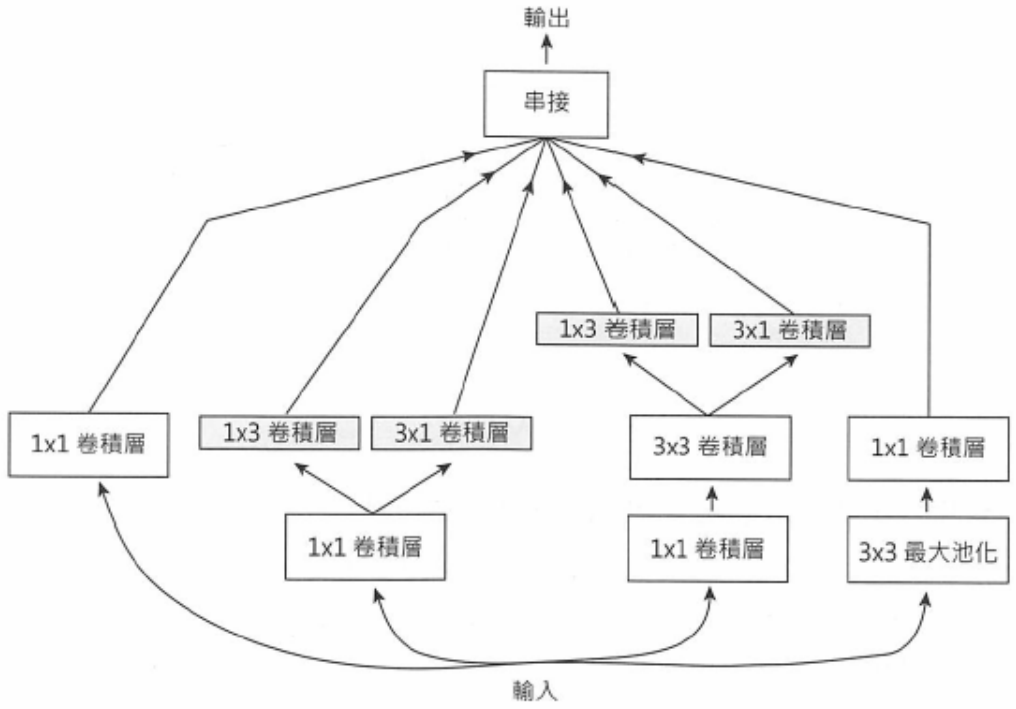
GoogLeNet

- Inception-V3（2015）：再次「削減參數量」

用 1x3 & 3x1 代替 3x3 卷積層，降低參數量



用橫向的方式，避免過深的網路，減少參數量



GoogLeNet

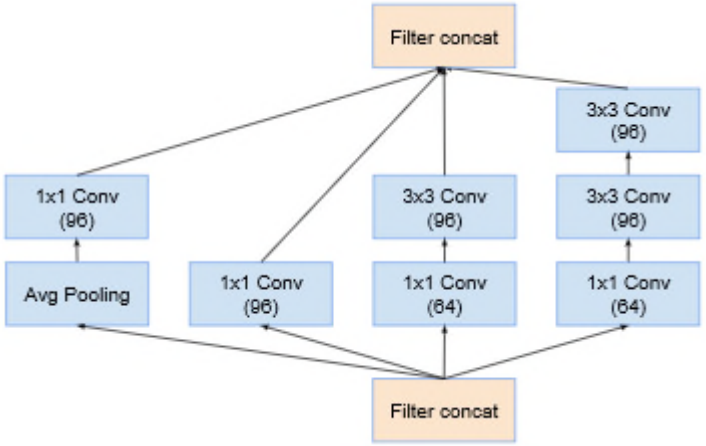
- Inception-V3 程式碼

```
1  from tensorflow.keras.applications import InceptionV3
2
3  model = InceptionV3()
4  model.summary()
```

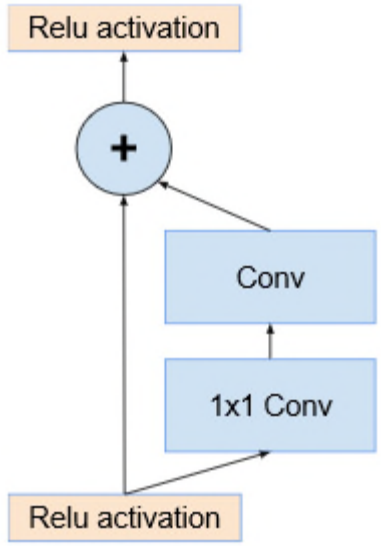
GoogLeNet

- Inception-V4（Inception-ResNet）（2016）

結合 Inception 與 ResNet（後述）優點的架構



+



=

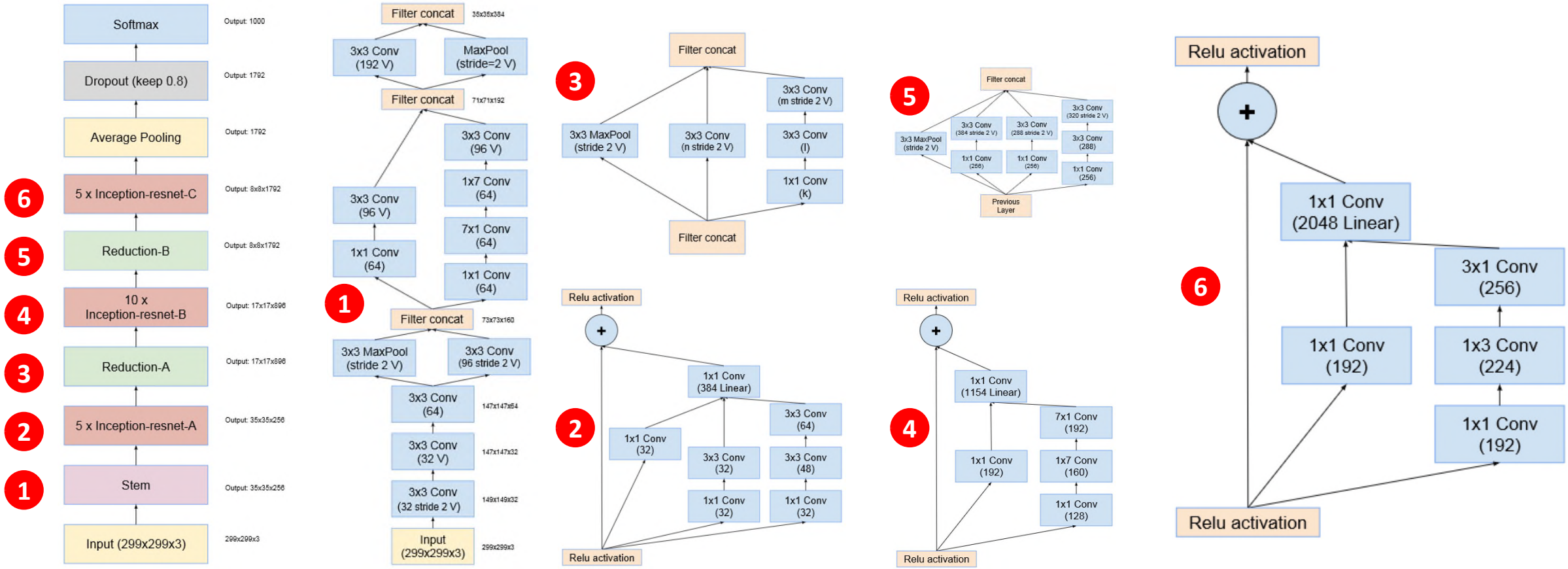
又寬（特徵多）
又深（更抽象）

Inception Block（網路變寬）

ResNet Block（網路變深）

GoogLeNet

- 完整的 Inception-V4（Inception-ResNet）模型



GoogLeNet

- Inception-V4（Inception-ResNet）的程式碼：

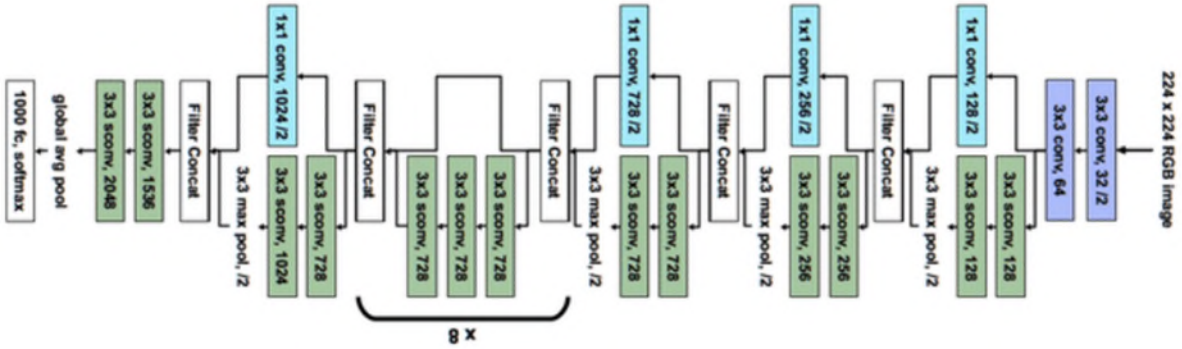
```
1  from tensorflow.keras.applications import InceptionResNetV2
2
3  model = InceptionResNetV2()
4  model.summary()
```

GoogLeNet

- **優點**
  - 是所有模型中，運算量 vs. 精確度取得較佳平衡的一款 CNN 經典模型。
- **缺點**
  - 網路結構複雜，不好理解。
- **使用時機**
  - 想辨識複雜照片，又不想運算量過高時。

Xception（eXtreme Inception）

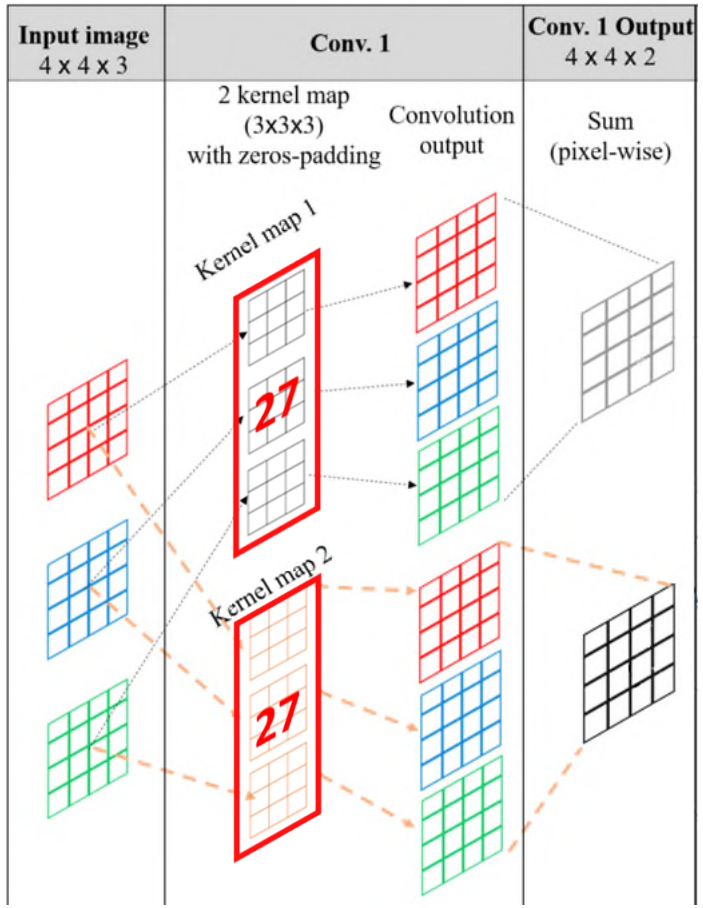
- **簡介**



- **發表時間**：2017
- **發表者**
  - Google
- **貢獻**
  - 改良 Inception-V3，讓它參數更少。
  - 提出「**深度分離卷積運算**」，來降低參數。
- **用途**
  - 需要**高度抽象化**後才能解開的問題，可以用它。

Xception

- 先看最原始的「卷積運算」



Channel #1

原圖： 4x4 x RGB

Channel #2

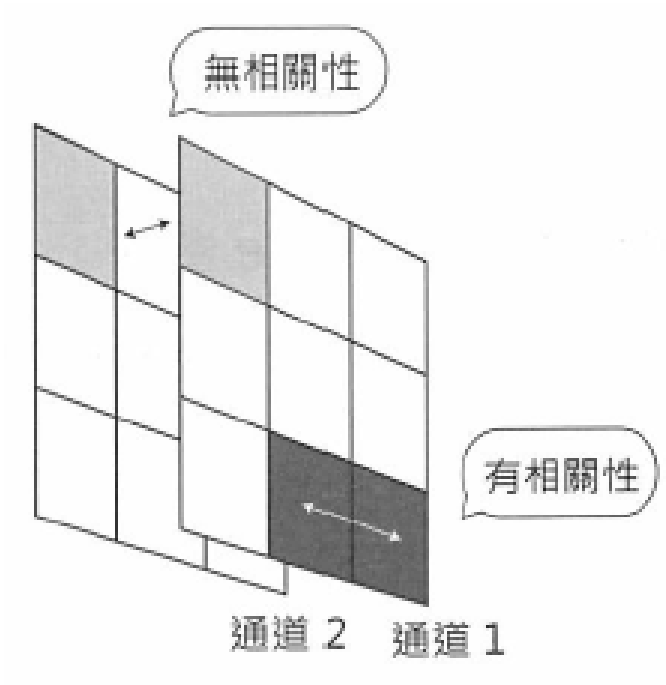
濾鏡： 3x3 x 2 個

- 濾鏡 #1 參數： 3x3 x RGB = 27
- 濾鏡 #2 參數： 3x3 x RGB = 27

參數： 27 + 27 = 54

Xception

- Xception 的假設



假設

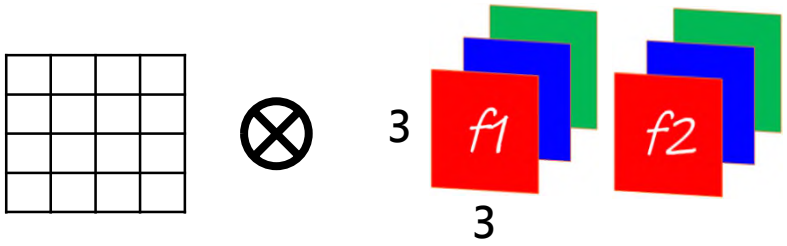
特徵僅「空間相關」
卻常常「頻道獨立」

→ 各通道可獨立做卷積，不必急著做「跨通道」特徵抽取
（可有效降低參數數量）

Xception

- Xception 的「深度分離卷積運算」（Depthwise Separable Convolution）

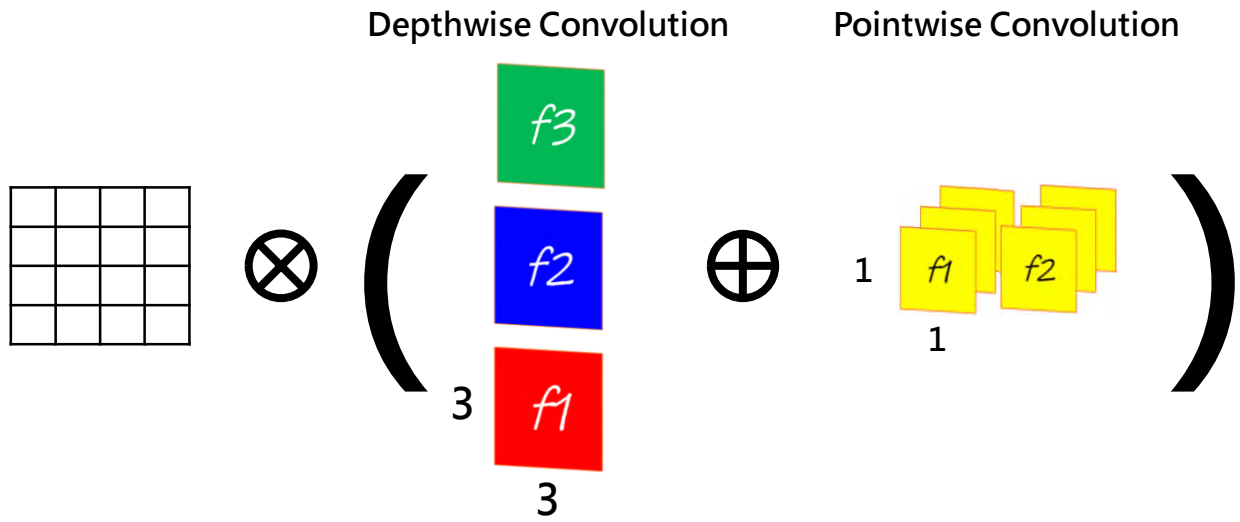
原始的卷積運算



RGB 濾鏡數

參數 = 3x3 x3 x2 = 54

深度分離卷積運算

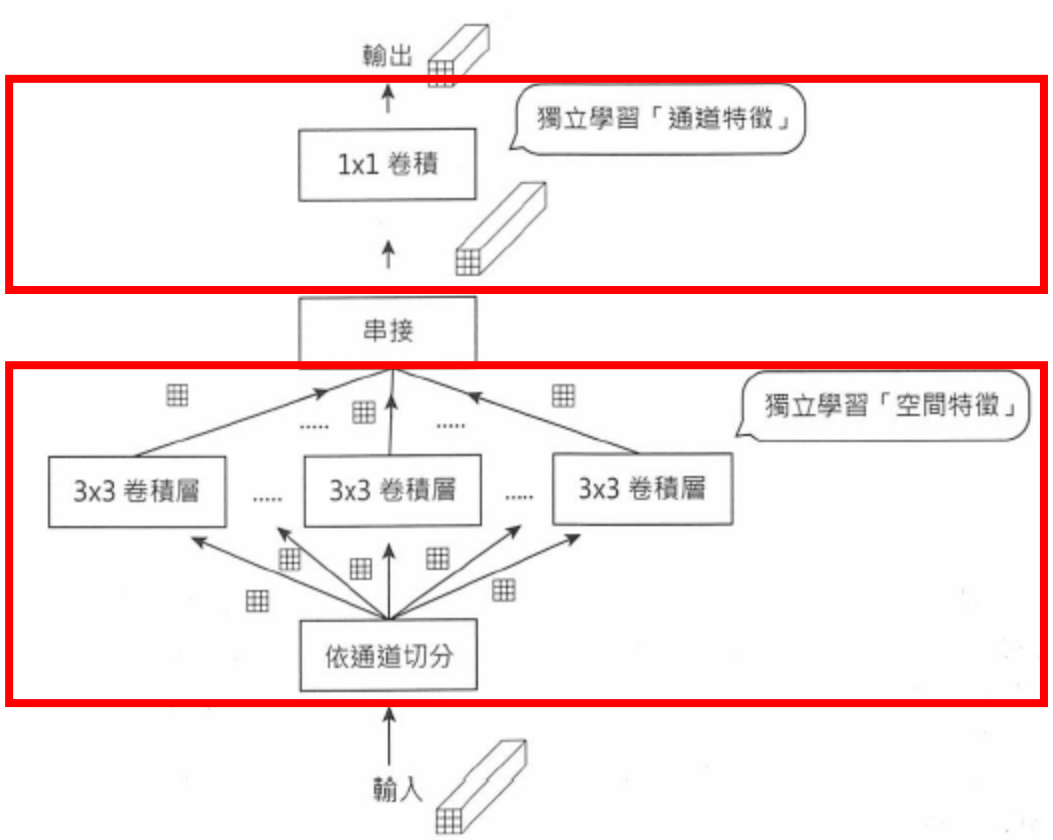


RGB 濾鏡數 RGB 濾鏡數

參數 = 3x3 x1 x3 + 1x1 x3 x2 = 33

Xception

- 一個完整的 Xception Block

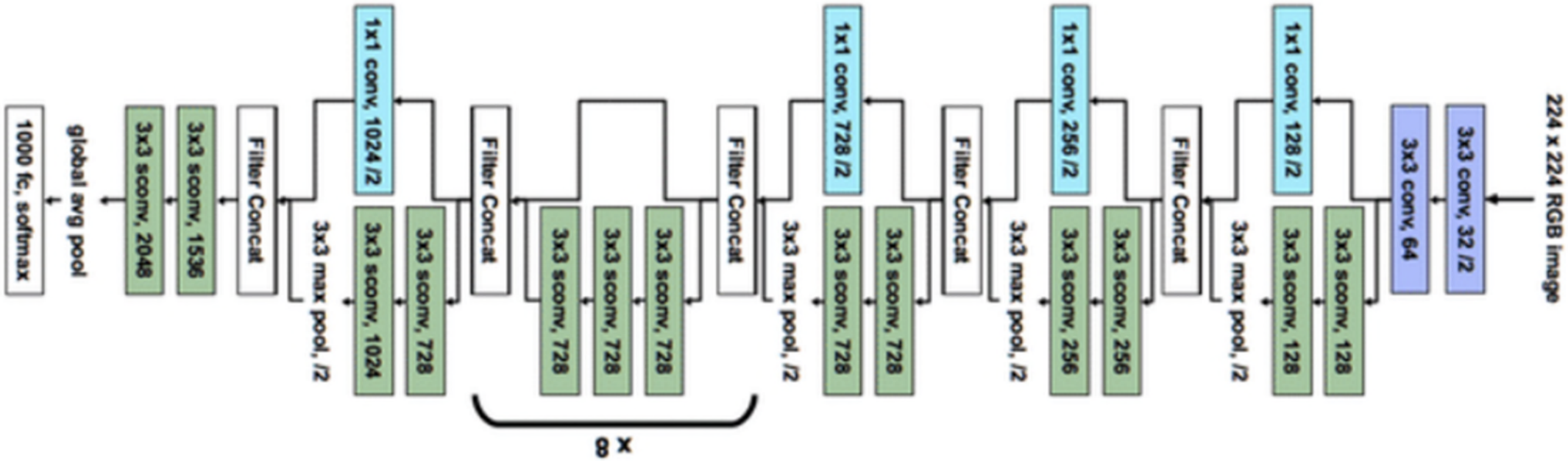


Pointwise Convolution

Depthwise Convolution

Xception

- 完整的 Xception 模型



Xception

- 程式碼

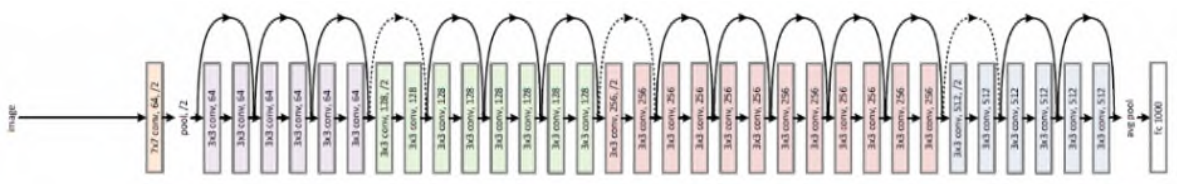
```
1  from tensorflow.keras.applications import Xception
2
3  model = Xception()
4  model.summary()
```

Xception

- **優點**
  - 參數少，計算速度快。
- **缺點**
  - 如果特徵並非「跨通道獨立」，這個方法就不能用了。
- **使用時機**
  - 該用：想要讓參數減少的時候。
  - 不該用：特徵「跨通道相依」時。

ResNet（殘差網路）

- **簡介**

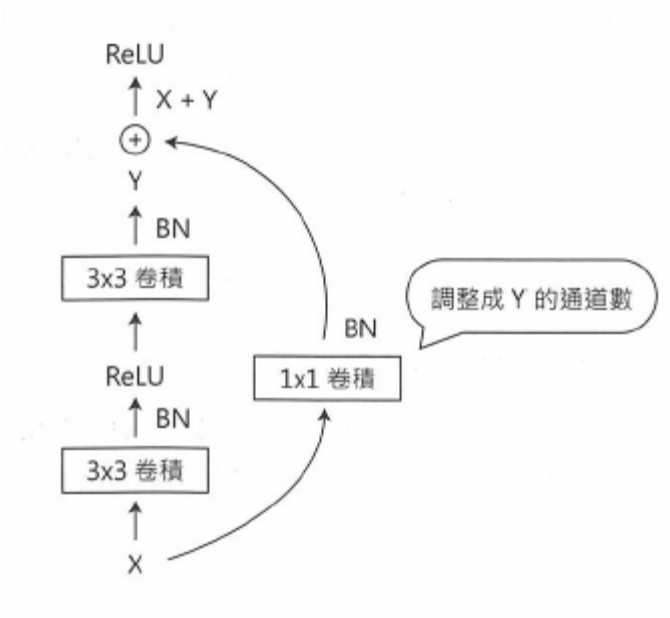


- **發表時間**：2015
- **發表者**
  - 何愷明（Kaiming He）
- **貢獻**
  - 用「**短路**」手法，解決超深層神經網路，「**特徵退化**」問題（比喻：**喝水傳話，越傳越糟**）。
  - 實際示範一個 152 層「可以動」的超深層網路。
  - 奪得 2015 ImageNet 大賽冠軍。
- **用途**
  - 需要**高度抽象化**後才能解開的問題，可以用它。

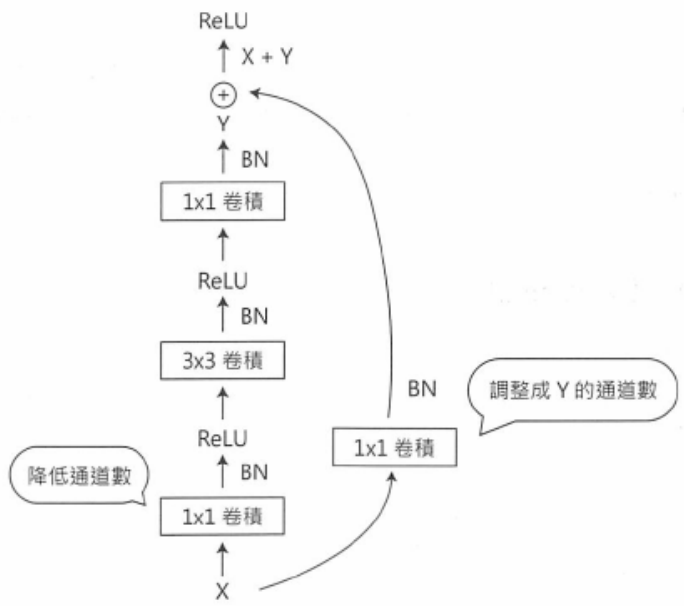
ResNet

- ResNet Block

前 34 層

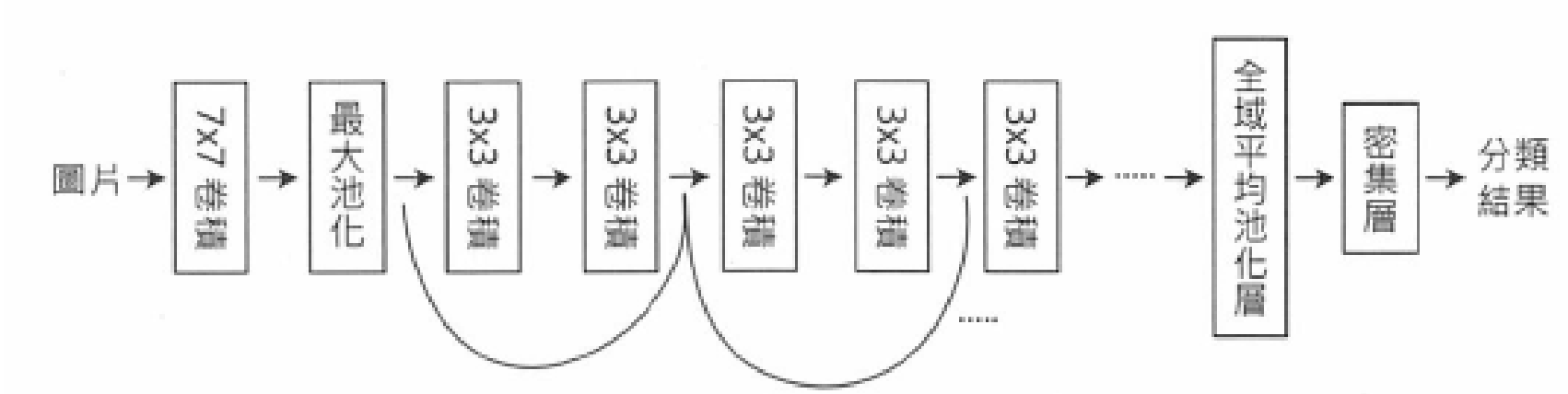


50/101/152 層



ResNet

- 完整架構


圖片
7x7 卷積
最大池化
3x3 卷積
3x3 卷積
3x3 卷積
3x3 卷積
3x3 卷積
全域平均池化層
密集層
分類結果

ResNet

- 程式碼

50 層的 ResNet

```
1  from tensorflow.keras.applications import ResNet50
2
3  model = ResNet50()
4  model.summary()
```

101 層的 ResNet

```
1  from tensorflow.keras.applications import ResNet101
2
3  model = ResNet101()
4  model.summary()
```

152 層的 ResNet

```
1  from tensorflow.keras.applications import ResNet152
2
3  model = ResNet152()
4  model.summary()
```

ResNet

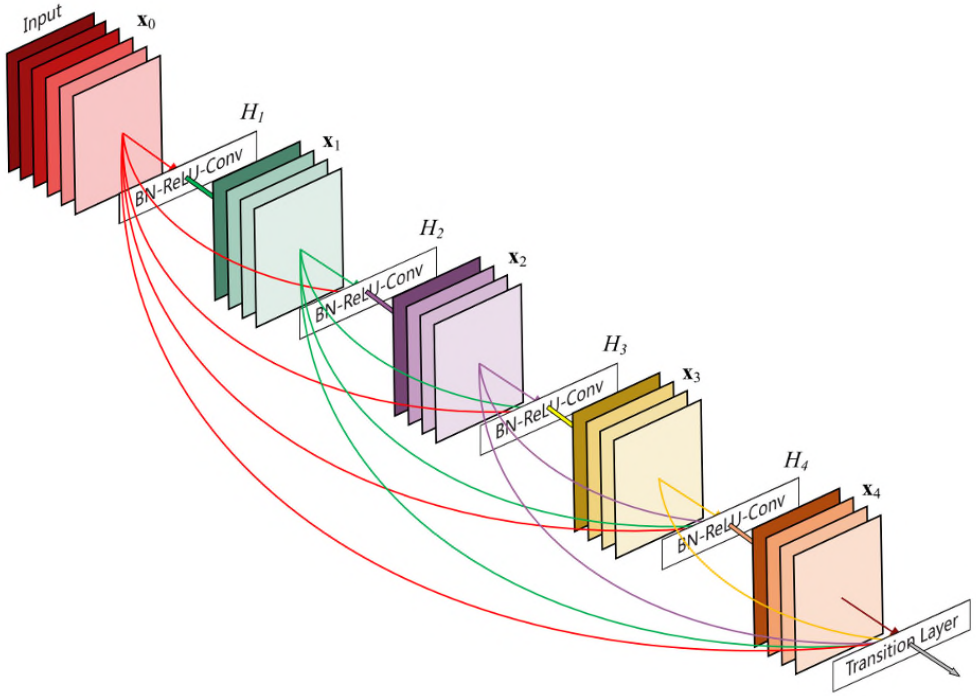
- **優點**
  - 用「短路」解決了「超深層網路特徵退化」問題。
  - 從此之後，可以解開網路深度無法太深的枷鎖。

- **缺點**
  - 計算量龐大

- **使用時機**
  - 當你想要使用「超深層網路」，辨識複雜圖形時。

DenseNet

- **簡介**

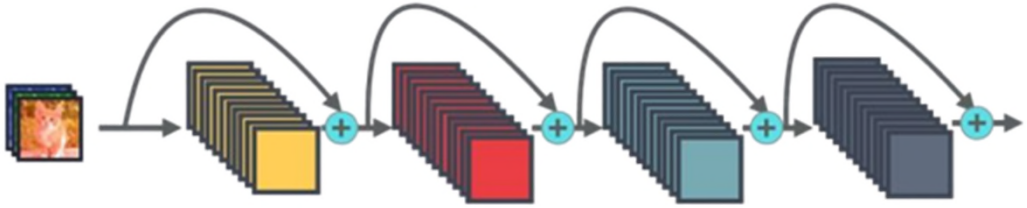


- **發表時間**：2016
- **發表者**
  - Gao Huang (康乃爾大學)
  - Zhuang Liu (北京清華大學)
  - Laurens van der Maaten (Facebook AI Research)
- **貢獻**
  - 基於 **ResNet** 突破超深層網路「特徵退化」基礎之下，保留**更多特徵**，且讓**參數更少**。
- **用途**
  - **超深層網路** + **保留特徵**時，可以用它。

DenseNet

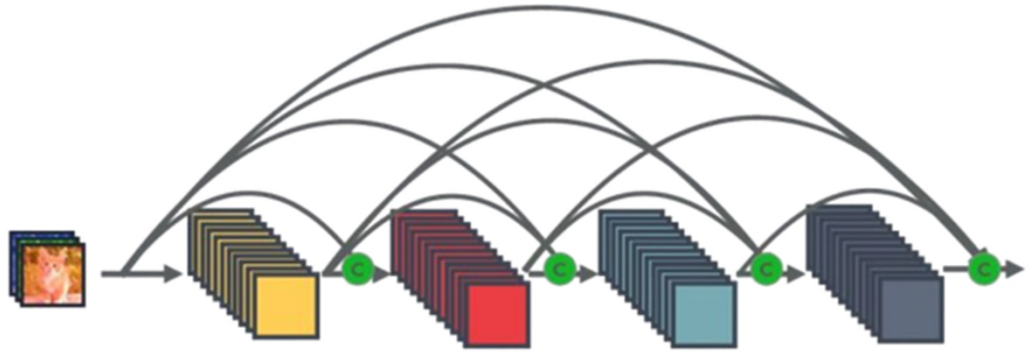
- ResNet Block vs. DenseNet Block 的不同

ResNet Block



每一層都加入上一層的特徵

DenseNet Block

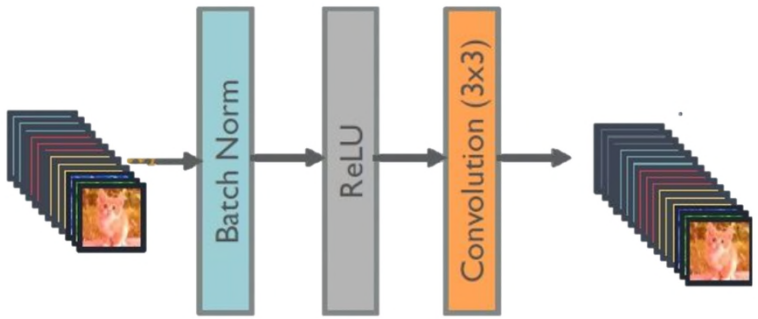


每一層都加入所有層的特徵
（特徵保存做得更好）

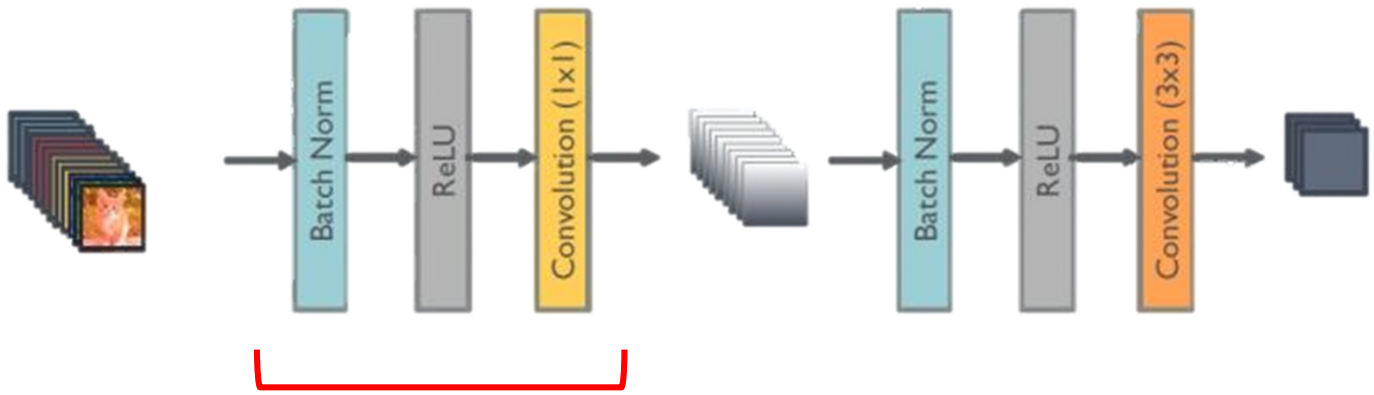
DenseNet

- 兩種不同的 DenseNet Block Layers

DesnseNet Block Layer
(Type A)



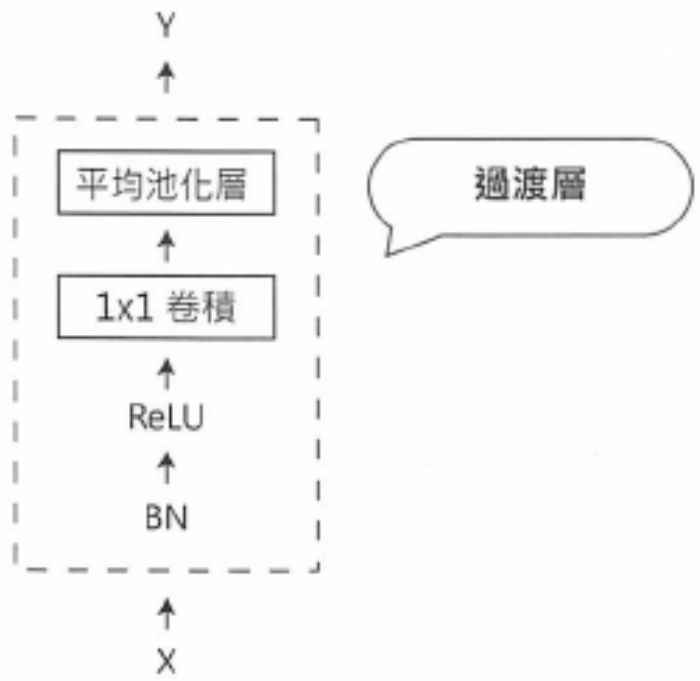
DesnseNet Block Layer
(Type B)



Bottleneck 層
(用 1x1 卷積降低維度用)

DenseNet

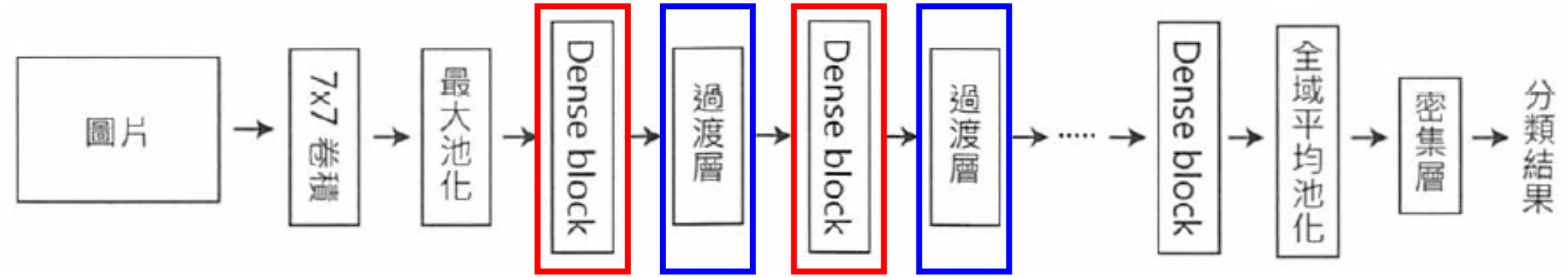
- DenseNet 的「過渡層（Transition Layer）」



- 位於兩個 DenseNet Block 之間
- 負責用 1x1 卷積縮小特徵圖

DenseNet

- 完整的 DenseNet 模型



DenseNet

- 程式碼

DenseNet121

```
from tensorflow.keras.applications import DenseNet121

model = DenseNet121()
model.summary()
```

DenseNet169

```
from tensorflow.keras.applications import DenseNet169

model = DenseNet169()
model.summary()
```

DenseNet201

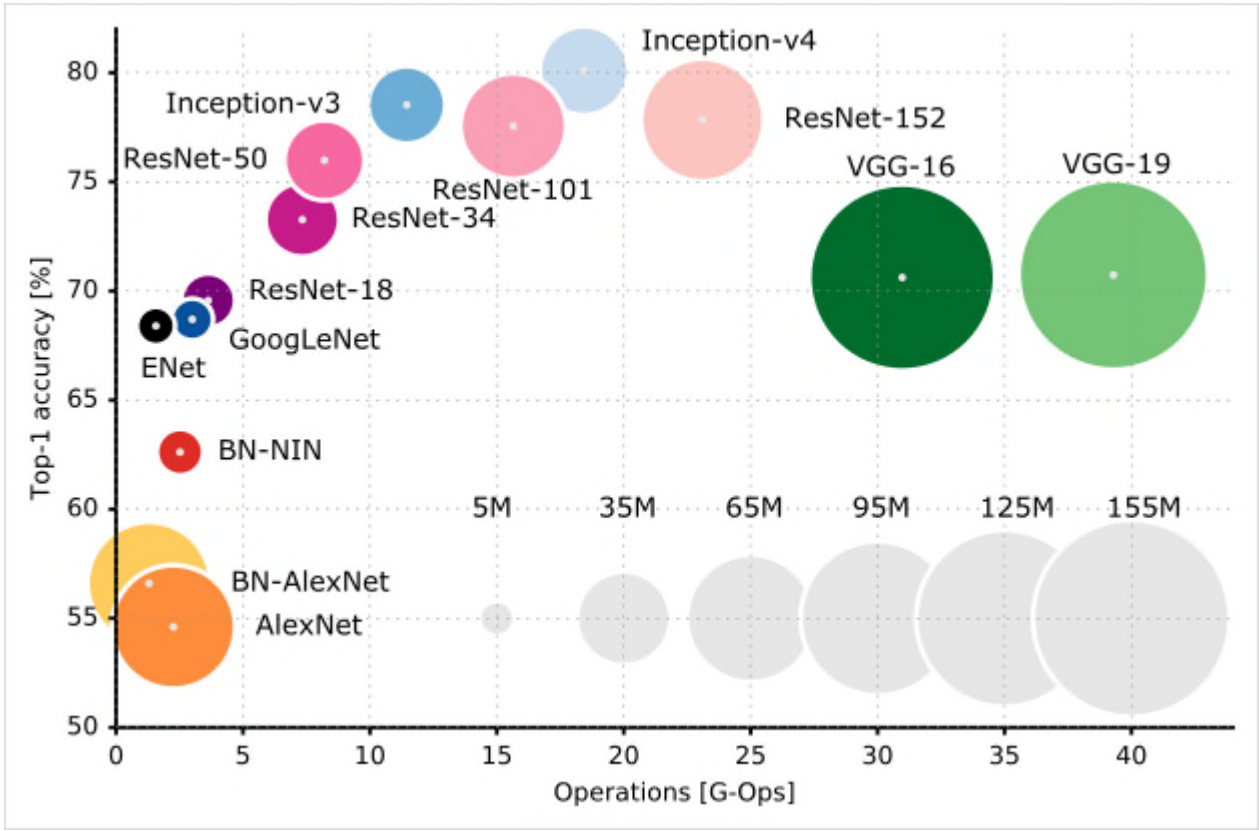
```
from tensorflow.keras.applications import DenseNet201

model = DenseNet201()
model.summary()
```

DenseNet

- **優點**
  - 更有效地保留特徵，防止梯度消失。
  - 與 ResNet 相比，參數更少。
- **缺點**
  - 計算量仍然很大
- **使用時機**
  - 超深層網路 + 想保留特徵 + 想要更少參數時

CNN 經典模型總結

- 運算量 vs. 精確度對照圖



小節整理：CNN 經典模型

- **懂得這些，你也可能發明出自己的「經典模型」**

- **關於「有效保存特徵」**
  - 用「最大池化」比「平均池化」，更能保存突出的特徵。
  - 池化步長 < 池化視窗大小，可重複檢視各特徵，確保沒有遺漏。

- **關於「防止梯度或特徵消失」**
  - 用「ReLU」做為激活函數，可有效防止「梯度消失」。
  - 在每一層結束後，做一下「Batch Normalization（平均=0，標準差=1）」，可有效防止梯度消失。
  - 用「短路」手法，可以防止特徵在「超深層網路」裡正向傳播時，逐漸消失的問題。

- **關於「減低參數」**
  - 「多個小卷積」優於「一個大卷積」。參數更少，深度更深。
  - 1x1 卷積層，可以「降維」，也能「升維」。
  - 用「全域平均池化」代替「全連接層」，可大量減低參數。
  - 使用 Functional API 橫向架構，比 Sequential 縱向架構，更能有效削減參數。
  - 如果特徵在頻道間獨立，可以考慮用「深度分離卷積運算」，降低參數量。

遷移學習

Transfer Learning

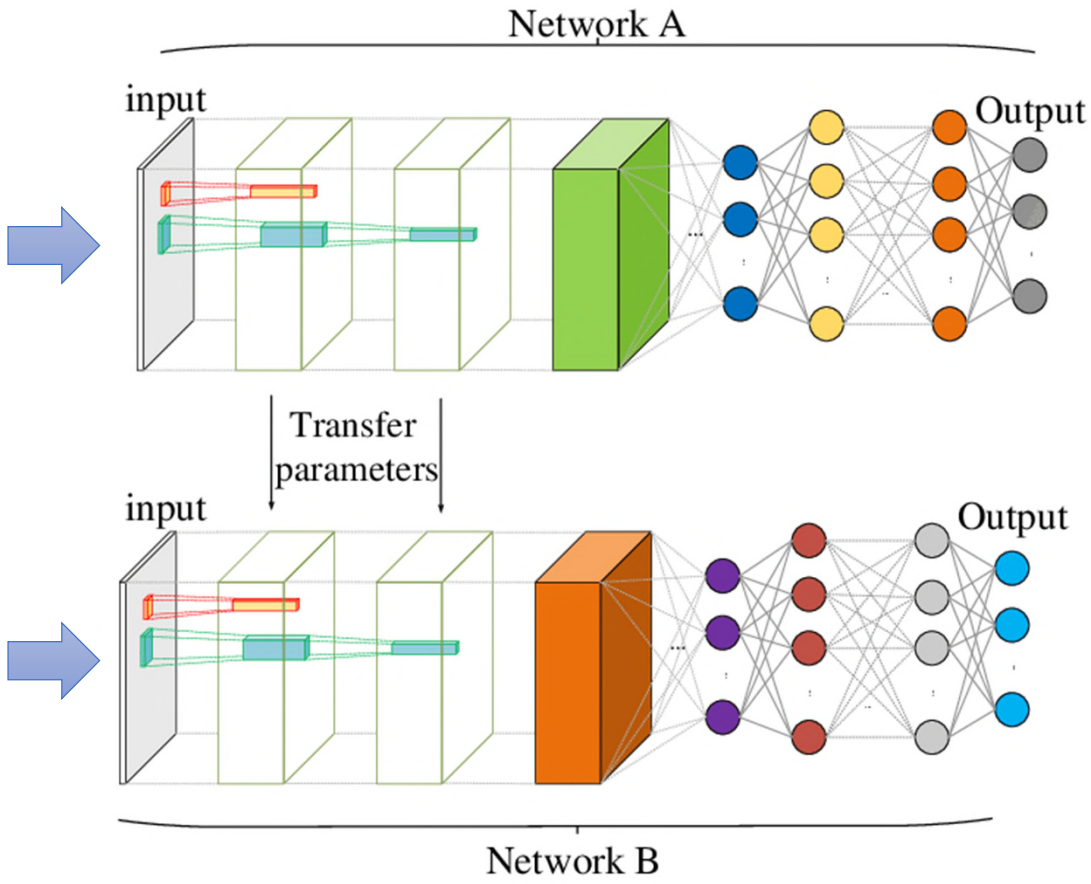
範例完整原始碼：
https://lurl.cc/4D0Wlo

何謂「遷移學習」

- 將其它資料集訓練好的「卷積層」模型，移花接木到自己的程式內使用

ImageNet
(Source Data)

Medical Pictures
(Target Data)



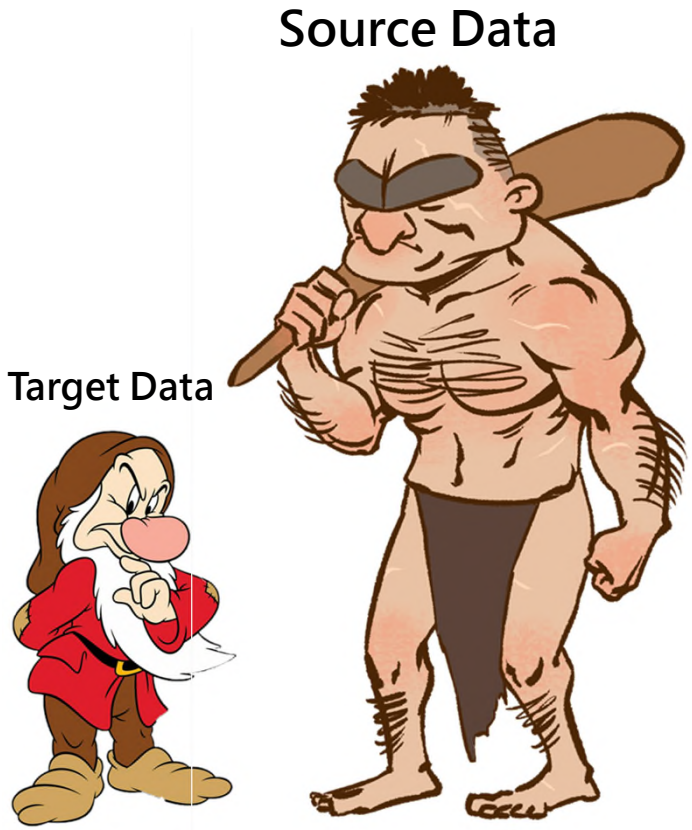
2 用一般照片訓練出一個模型

3 模型轉移

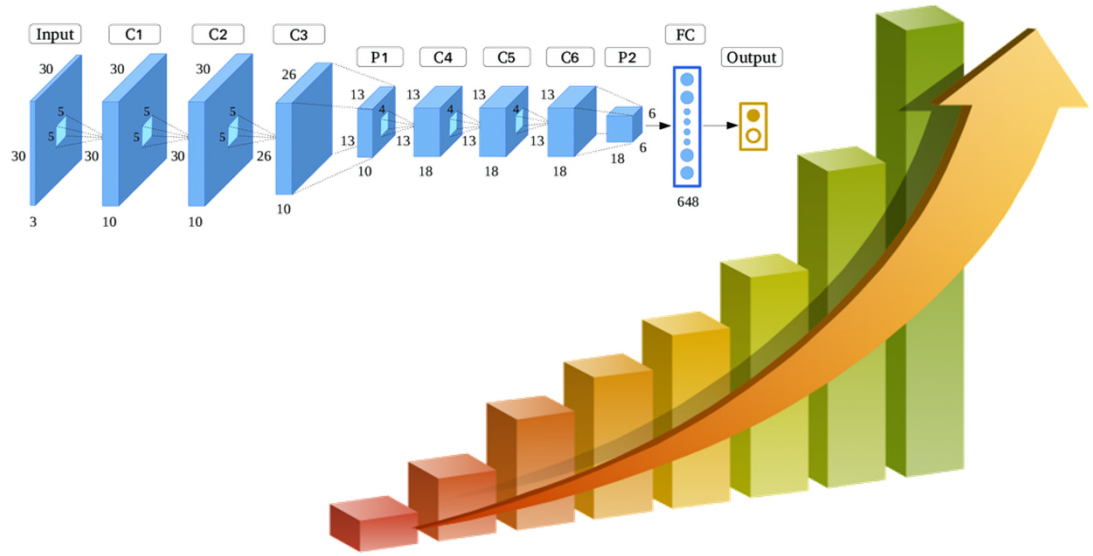
1 想做出能辨識「腫瘤」、「骨折」等醫學照片的模型

何時用「遷移學習」

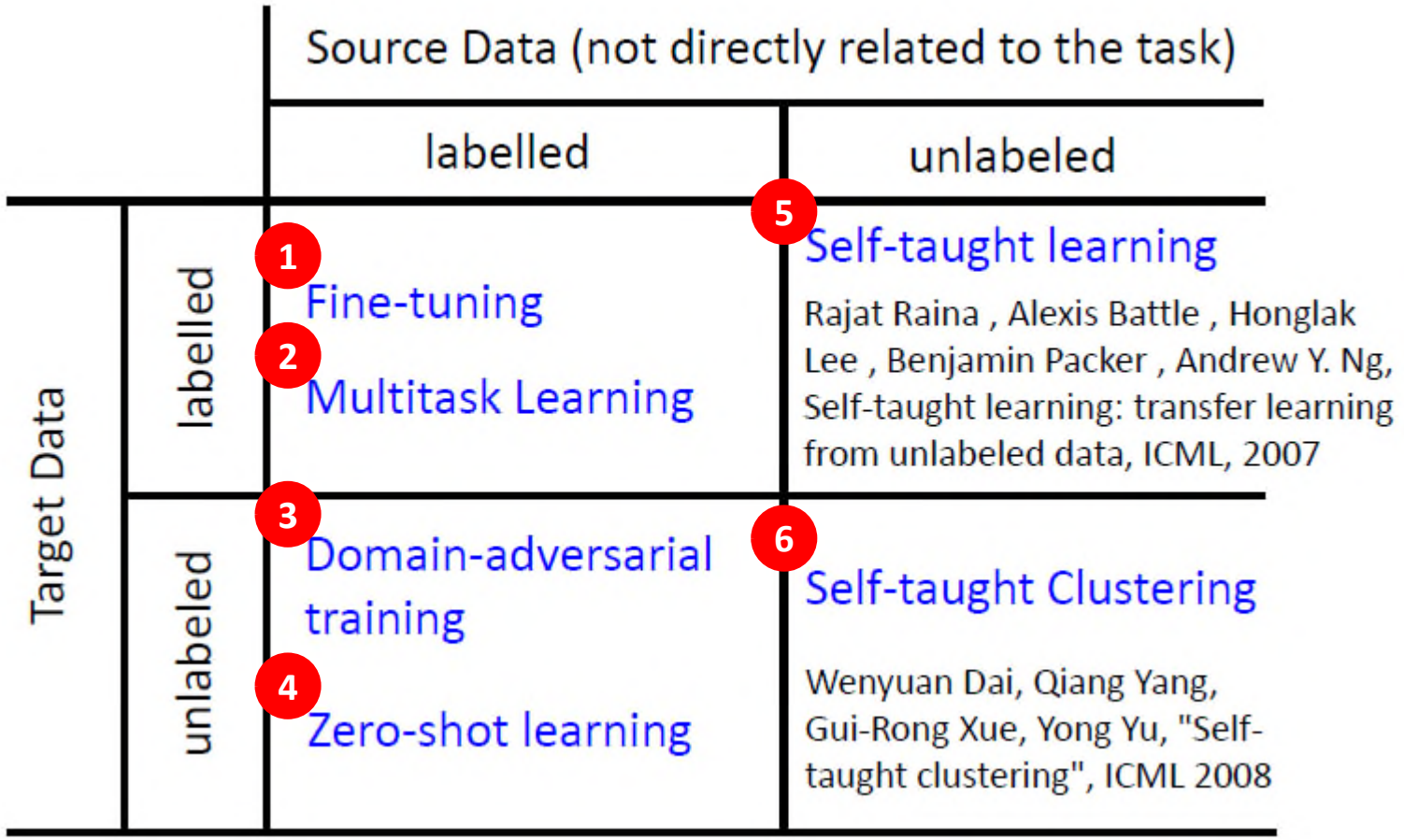
- 「目標資料」太少，無法訓練
- 想藉助「經典模型」，提高正確率



VGG16



「遷移學習」的種類



詳細解說：
台大李宏毅老師
「機器學習 #19：遷移學習」
https://youtu.be/qD6iD4TFsdQ

本小節討論範圍：
Fine-Tuning 之下的
「**Layers Transferring**」一法

程式碼

- (1) HappyML 與資料集下載

```
1  # Install HappyML
2  import os
3
4  if not os.path.isdir("HappyML"):
5    os.system("git clone https://github.com/cnchi/HappyML.git")
6
7  # Dataset Download
8
9  if not os.path.isfile("cats_and_dogs_filtered.zip"):
10   !wget https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip
11
12   # Unzip the file
13   import zipfile
14
15   zip_file = zipfile.ZipFile('cats_and_dogs_filtered.zip', 'r')
16   zip_file.extractall()
17   zip_file.close()
```

安裝 HappyML

下載/解壓資料集

程式碼

- (2) 重要變數設定

```
1  # Dataset Path Setting
2  base_dir = 'cats_and_dogs_filtered'
3  train_dir = os.path.join(base_dir, 'train')
4  test_dir = os.path.join(base_dir, 'validation')
5
6  # Model Related Setting
7  to_train = True  ← =True：這次要訓練 = False：這次要讀入模型檔
8  image_size = 224 ← 照片統一縮放成 224x224 ( 這也是 ImageNet 的大小 )
9  model_file = "CatsDogs_VGG16_224x224.h5" ← 模型檔檔名
```

資料集路徑

程式碼

- （3）照片集前處理

```python
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(
    rescale=1.0/255,
    validation_split=0.75,
    width_shift_range = 0.2,
    height_shift_range = 0.2,
    shear_range=10,
    zoom_range=0.2,
    horizontal_flip=True)

train_set = train_datagen.flow_from_directory(
    directory=train_dir,
    target_size=(image_size, image_size),
    batch_size=10,
    class_mode="binary",
    subset="training")
```

← 模擬 Target Data 量少的樣子

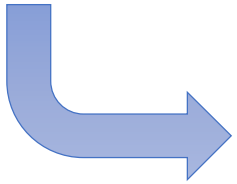
隨機水平/垂直平移 ±20%

```python
val_set = train_datagen.flow_from_directory(
    directory=train_dir,
    target_size=(image_size, image_size),
    batch_size=10,
    class_mode="binary",
    subset="validation")

test_datagen = ImageDataGenerator(rescale=1.0/255)

test_set = test_datagen.flow_from_directory(
    directory=test_dir,
    target_size=(image_size, image_size),
    batch_size=10,
    class_mode="binary")
```

程式碼

- （４）驗證照片載入成功

```
# In[] 驗證照片載入成功
import HappyML.model_drawer as md

# Print the first 10 images of Training Set
for data, label in train_set:
    md.show_first_n_images(x_ary=data, y_real=label, first_n=10, font_size=12)
    break

# Print the first 10 images of Testing Set
for data, label in test_set:
    md.show_first_n_images(x_ary=data, y_real=label, first_n=10, font_size=12)
    break
```



隨堂練習：函式庫&前處理

- 請開啟 Colab，建立一個新檔案 CNN_CatsDogs_VGG16.py。
- 將前述投影片的程式碼撰寫好。可以拷貝前一小節原始碼 CNN_CatsDogs.ipynb 的內容，再加以修改。
- 執行程式碼，看看結果是否如您所預期。

程式碼

- （5）用 VGG16 建造模型

```
# In[] 建造模型
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers
from tensorflow.keras.applications import VGG16

# CNN Model
model = Sequential()

# Use VGG16 as Convolutional Layers
vgg16 = VGG16(
    include_top=False,
    weights="imagenet",
    input_shape=(image_size, image_size, 3))
#vgg16.summary()
```

VGG16 參數詳解：

- include_top：
  是否包含「全連接層」部分。
- weights="imagenet"：
  使用 ImageNet 訓練好的權重，
  做為初始值。
- input_shape：輸入層維度。
- pooling：是否在卷積層之後，
  全連接層之前，使用池化層。
  - None：不使用池化層
  - "avg"：使用「全域平均池化」
    （GlobalAveragePooling2D）
  - "max"：使用「全域最大池化」
    （GlobalMaxPooling2D）

程式碼

• （5）用 VGG16 建造模型

- 在互動模式中，使用 vgg16.summary() 瞭解 VGG16 架構。
- 凍結所有「**低階特徵**」權重，開放以下三層「**高階特徵**」權重可訓練：
  - "block5_conv1"
  - "block5_conv2"
  - "block5_conv3"
- 不同任務，**凍結層級**也不同：
  - 圖形辨識：**低階特徵**相似（邊角），**高階特徵**不同（貓狗）
  - 聲音辨識：**低階特徵**不同（聲調），**高階特徵**相似（文字）

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

程式碼

- （5）用 VGG16 建造模型

```
16  # Set the last 3 layers of VGG16 as trainable
17  is_trainable = ["block{}_conv{}".format(5, i) for i in range(1, 4)]
18     ↑ is_trainable = [ “block5_conv1” , “block5_conv2” , “block5_conv3” ]
19  for layer in vgg16.layers:
20      layer.trainable = True if layer.name in is_trainable else False
21  #vgg16.summary() ↑ 凍結所有層級的權重（=False），除了上述三個層級之外（=True）
22
23  # In[] Add each layers & Compile
24  model.add(vgg16)
25  model.add(layers.Flatten())
26  model.add(layers.Dense(units=512, activation="relu"))
27  model.add(layers.Dense(units=1, activation="sigmoid"))
28
29  from tensorflow.keras.optimizers import RMSprop
30  model.compile(optimizer=RMSprop(lr=1e-5), loss="binary_crossentropy", metrics=["accuracy"])
31  #model.summary()                    ↑ 注意：將學習速率縮小一點（原 2e-5），抵達終點附近才有辦法「收斂」
```

將 VGG16 後方，加上一個「全連接層」做分類。

隨堂練習：建造模型

- 請先將 VGG16 必要的參數填好，並執行程式碼看看。
- 用 vgg16.summary() 瞭解 VGG16 所有層級的名稱。
- 除了最後三個層級之外，將 VGG16 所有層級的權重凍結。
- 將 VGG16 後方接上一個全連接層，並且編譯模型。
- 用 model.summary() 看看，模型是否編譯成功？

程式碼

- （6）模型訓練

```
1  # In[] Train & Save Model or Load Model
2  from tensorflow.keras.models import load_model
3
4  if to_train:  ← 如果 to_train == True：訓練模型，並將之儲存
5      model.fit(train_set, validation_data=val_set, epochs=10)
6      model.save(model_file)
7  else:         ← 如果 to_train == False：載入模型檔，繼續下一段程式碼
8      model = load_model(model_file)
```

程式碼

- （7）模型評估

```
1   # In[] 模型評估
2   test_loss, test_acc = model.evaluate(test_set)
3   print("Loss of Test:", test_loss)
4   print("Accuracy of Test:", test_acc)
```

大進步！

串接 VGG16

```
In [15]: runcell('[] 模型評估', 'D:/demo/DeepLearning/Ch05 CNN/CNN_CatsDogs_VGG16.py')
200/200 [==============================] - 18s 92ms/step - loss: 0.1582 - accuracy: 0.9500
Loss of Test: 0.15822544693946838
Accuracy of Test: 0.949999988079071
```

≒ 95.00%

與先前模型對比

```
In [13]: runcell('[] 模型評估', 'D:/demo/DeepLearning/Ch05 CNN/02cnn_catsdogs.py')
200/200 [==============================] - 7s 34ms/step - loss: 0.4394 - accuracy: 0.8030
Loss of Test: 0.43940621614456177
Accuracy of Test: 0.8029999732971191
```

≒ 80.30%

隨堂練習：模型訓練＆評估

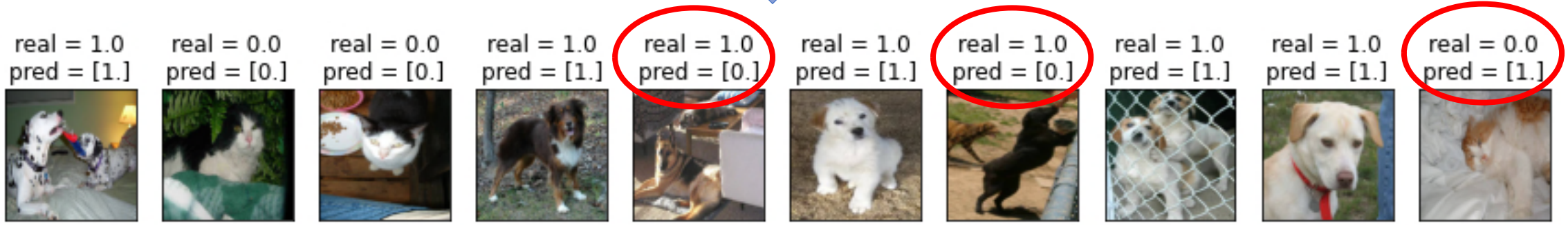
- 請將前兩頁投影片的程式碼寫好，執行看看。
- 你的模型正確率有多高？

程式碼

- （8）實際預測「測試集」圖片（顯示前 10 張的結果）

```python
# In[] 模型預測
import numpy as np

Y_pred = np.rint(model.predict(test_set))

for data, label in test_set:
    md.show_first_n_images(x_ary=data, y_real=label, y_pred=Y_pred[:10], first_n=10, font_size=12)
    break
```

猜錯的部分



程式碼

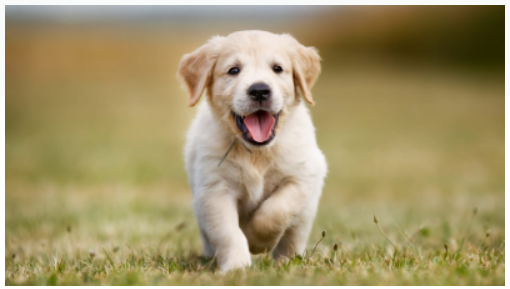
- （9）測試使用者提供的照片

```
# In[] 使用者照片預測
from tensorflow.keras.preprocessing import image

user_image = image.load_img("cat_dog/user/cat_or_dog_002.jpg", target_size=(image_size, image_size))
user_image = image.img_to_array(user_image)
user_image = np.expand_dims(user_image, axis=0)
result = model.predict(user_image)

if result[0][0] < 0.5:
    print("Cat")
else:
    print("Dog")
```

cat_or_dog_001.jpg

“Cat”

cat_or_dog_002.jpg



“Dog”

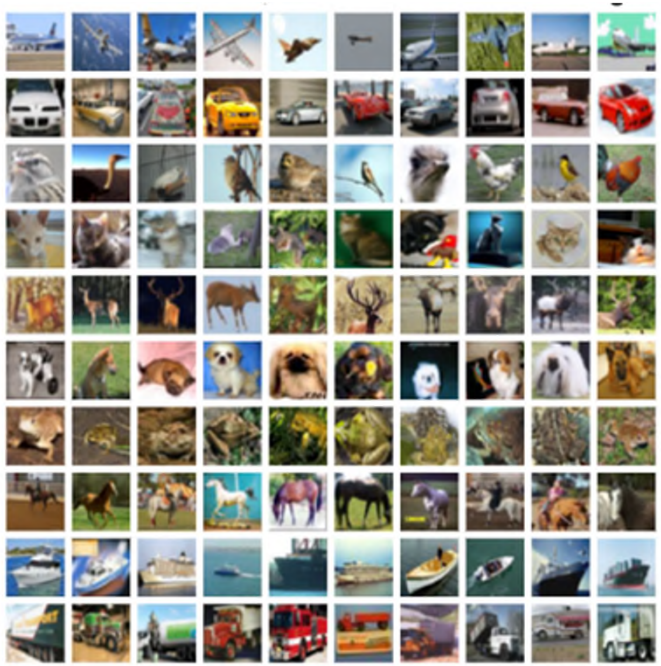
隨堂練習：測試集&使用者圖片

- 請將前兩頁投影片的程式碼寫好，執行看看。
- 你的「測試集」與「使用者圖片」的預測結果，正確率有多高？

課後作業

- 請做一個能將 CIFAR-10 圖片資料集分類的 CNN 模型
  - CIFAR-10 簡介
    - 32x32 彩色圖片。訓練集 50000 筆 + 測試集 10000 筆
    - 共分 10 類，每類 6000 張圖片

class = 0 airplane
class = 1 automobile
class = 2 bird
class = 3 cat
class = 4 deer
class = 5 dog
class = 6 frog
class = 7 horse
class = 8 ship
class = 9 truck

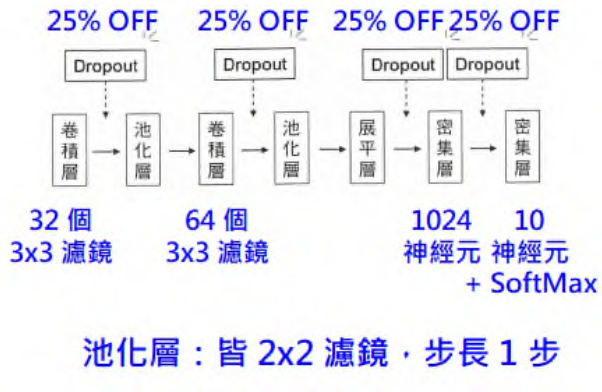


資料集載入方式

```
1  from tensorflow.keras.datasets import cifar10
2
3  (x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

課後作業

- 題目要求：
  - 請做出一個「架構自製」的 CNN 分類器。參考架構如下：



  - 請自行尋找「最佳超參數」，想辦法讓自己的模型正確率越高越好。
  - 請用 .evaluate() 函數，將測試集丟入模型中，看看正確率有多高？
  - 重新嫁接一個「經典模型」（如：VGG、Inception、Xception、ResNet…），看看有沒有辦法將正確率再往上拉高？

本章總結

- **CNN 原理**
  - 卷積層：抽取圖形特徵（樣式辨認）
  - 池化層：降低維度，維持特徵
  - 展平層：將特徵攤平
  - 全連接層：分類

- **CNN 經典模型**
  - LeNet：最古老的 CNN 模型
  - AlexNet：提出「資料擴張」概念，減低過擬合
  - VGGNet：提出「多個小卷積」，勝過「一個大卷積」
  - NiN：提出「縱向卷積」降維，以及「全域平均池化」取代全連接層，降低參數量
  - GoogLeNet：用 Functional API 模型進行 CNN 運算
  - Xception：深度分離卷積運算。各通道分離計算，降低參數。
  - ResNet：解決「超深層網路」的「特徵退化」問題
  - DenseNet：將低階特徵，跳接至高階層級，更好地保留特徵。

- **遷移學習**
  - 將別人訓練好的模型，移花接木給自己