

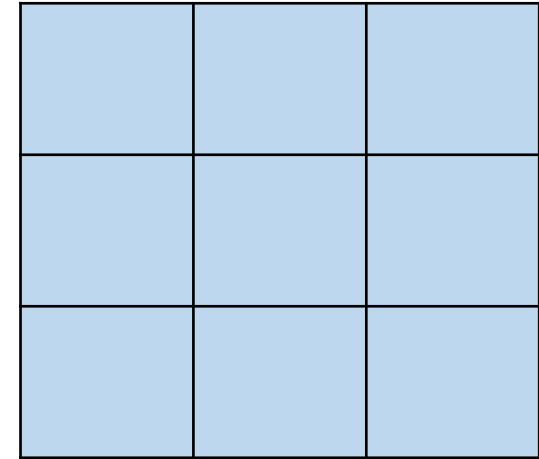
# CNN

Convolutional neural network

講師:隋建德

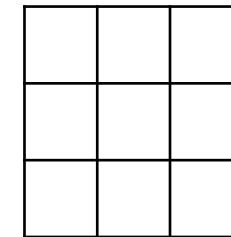
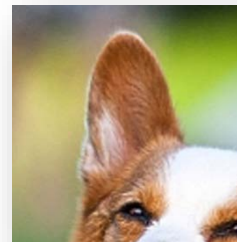
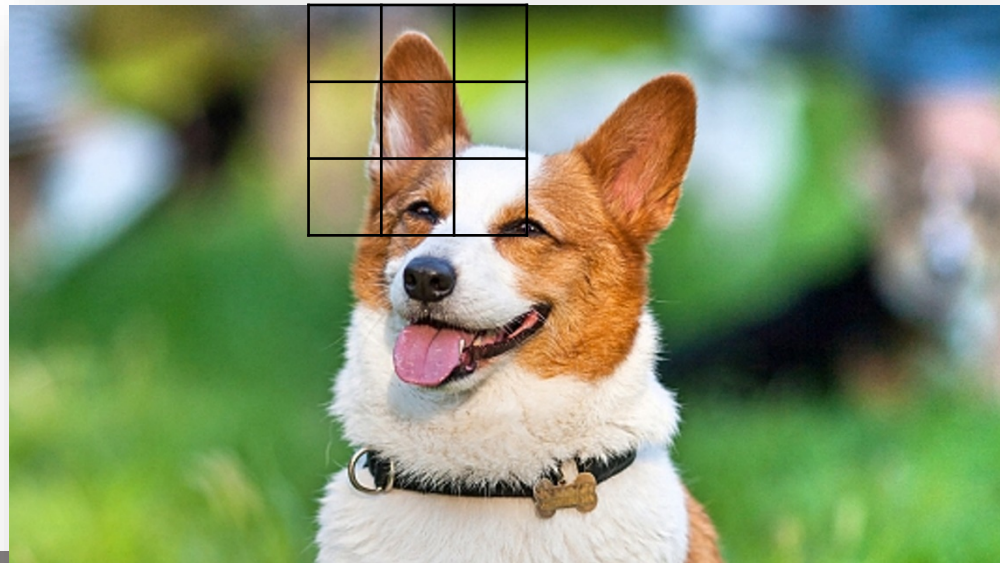
# 卷積 Convolution

- 模仿人的視覺
- 使用方塊的filter去看圖案
- 只有fully connect無法找到圖案上下關係
- 在圖像辨識上有很大成效



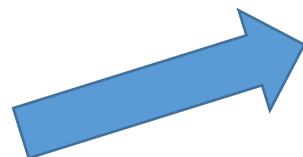
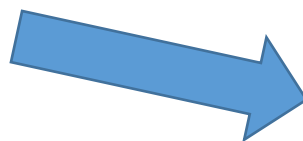
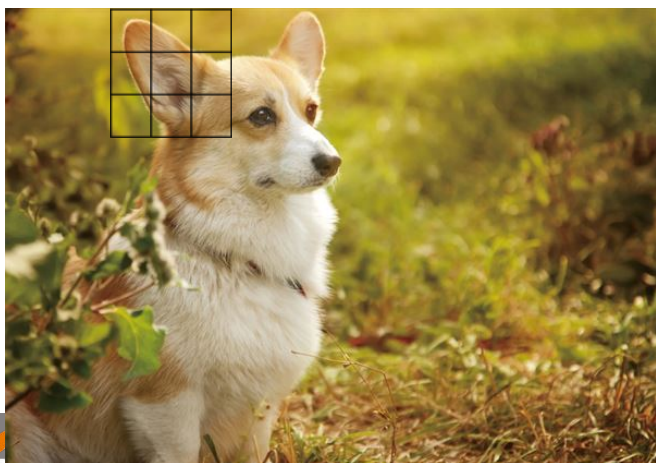
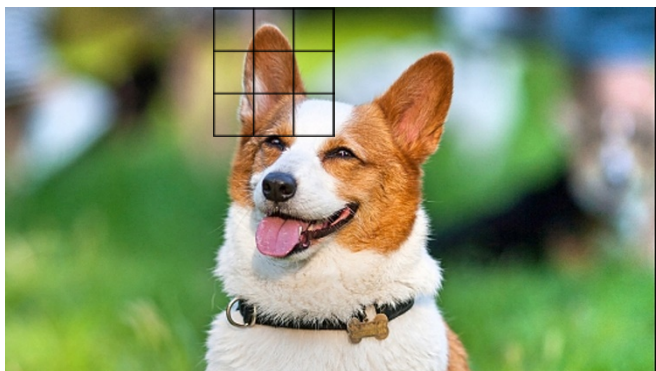
# 卷積(Convolution)的目的

- 圖片中主要特徵遠小於整張圖片
  - 不用看整張圖片去找特徵
  - 可以減少參數量

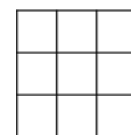


# 卷積(Convolution)的目的

- 同樣的特徵在不同圖案不同位置



狗耳朵

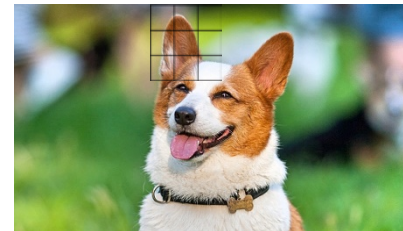
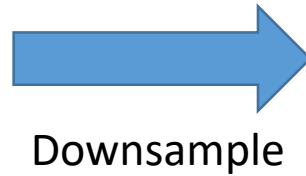
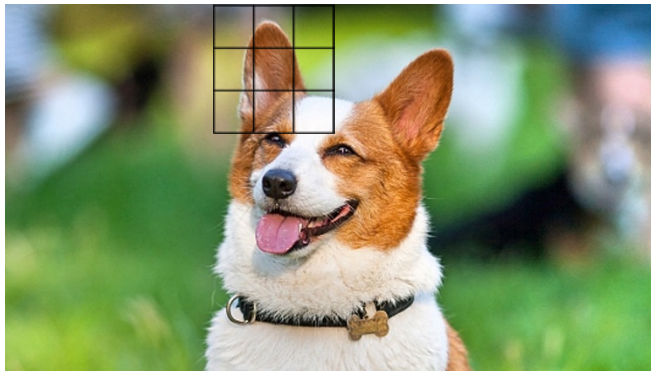


同一個狗耳  
偵測器

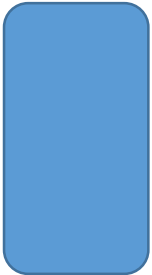
- 擁有相同參數
- 效用都是偵測狗耳


# 卷積(Convolution)的目的

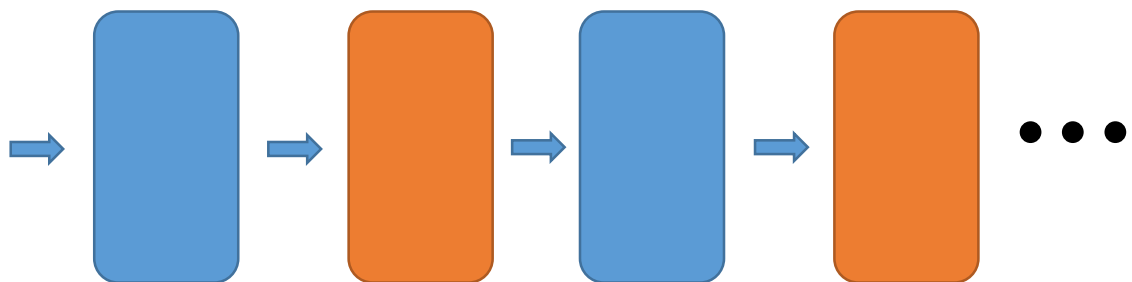
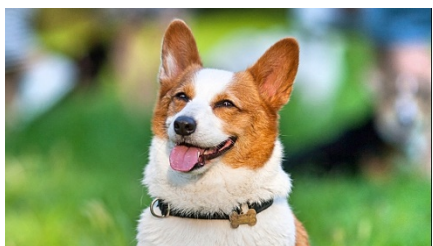
- 圖案解析度下降(downsample)也可達到相同效果



# 卷積神經網路

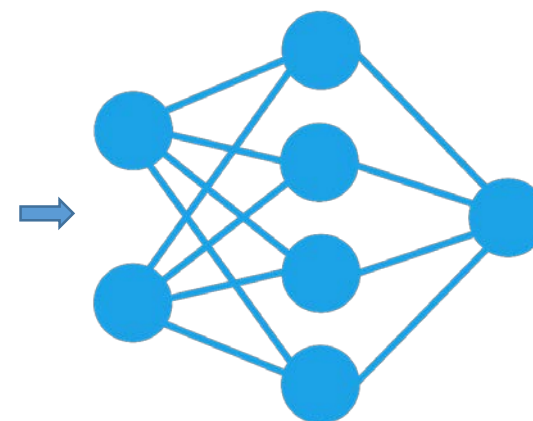
  
Convolution

  
Max pooling



可以加深  
許多層

展開  
(flatten)



狗?

# Convolution

- 相較整張圖，明顯的特徵遠小於長張圖片
- 特徵會出現在不同區塊

# Max pooling

- 取較大的特徵點
- 可以拿到較重要的特徵

# Convolution

Input

1	0	0	1	0	1
0	0	1	1	1	0
1	1	1	0	0	0
1	0	1	0	1	1
0	1	0	0	1	1
0	0	0	1	1	1

圖片

Convolution  
layer

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

Filter 1

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

Filter 2

●  
●  
●

裡面的參數都是學  
出來的



# Convolution

Stride = 1

Stride 是 filter 在圖片  
上移動的距離

1	0	0	1	0	1
0	0	1	1	1	0
1	1	1	0	0	0
1	0	1	0	1	1
0	1	0	0	1	1
0	0	0	1	1	1

圖片

Filter 1

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

對應位置相乘相加

1      -1

# Convolution

Stride = 2

Stride 是 filter 在圖片  
上移動的距離

1	0	0	1	0	1
0	0	1	1	1	0
1	1	1	0	0	0
1	0	1	0	1	1
0	1	0	0	1	1
0	0	0	1	1	1

圖片

Filter 1

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

對應位置相乘相加

1      -1

# Convolution

Stride = 1

Stride 是 filter 在圖片  
上移動的距離

1	0	0	1	0	1
0	0	1	1	1	0
1	1	1	0	0	0
1	0	1	0	1	1
0	1	0	0	1	1
0	0	0	1	1	1

圖片

Filter 1

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

對應位置相乘相加

Feature Map  
特徵圖

1	-1	-1	2
-1	1	0	2
0	0	2	2
1	1	1	-1

# Convolution

Stride = 1

Stride 是 filter 在圖片  
上移動的距離

1	0	0	1	0	1
0	0	1	1	1	0
1	1	1	0	0	0
1	0	1	0	1	1
0	1	0	0	1	1
0	0	0	1	1	1

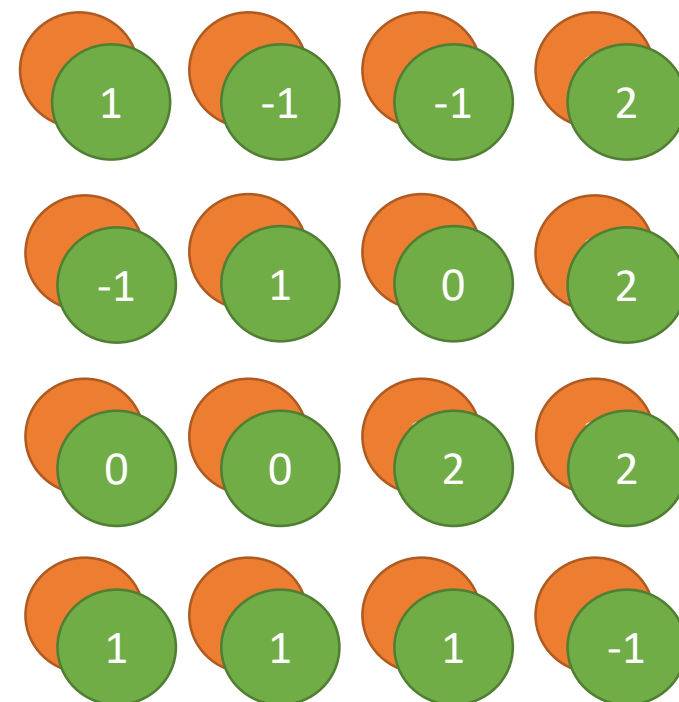
圖片

Filter 2

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

對應位置相乘相加

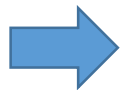
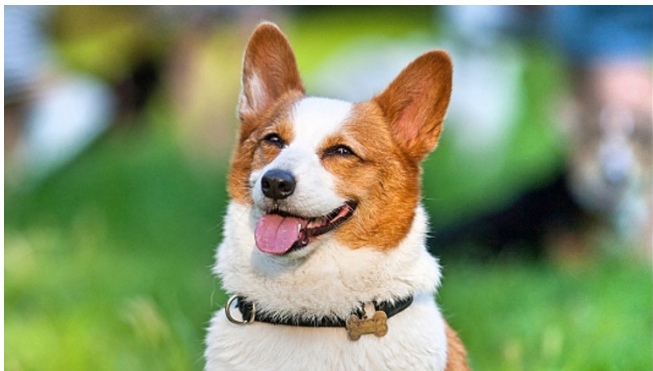
Feature Map  
特徵圖



- 每個filter都做一次相同動作→N個filter 會產生N層特徵圖，稱為channel數

# 彩色&黑白圖片

- 黑白圖片只會有一層
  - $1 \times 6 \times 6$
- 彩色圖片會有三層 RGB
  - $3 \times 6 \times 6$
  - 一個filter(立體filter)會在三層都做運算

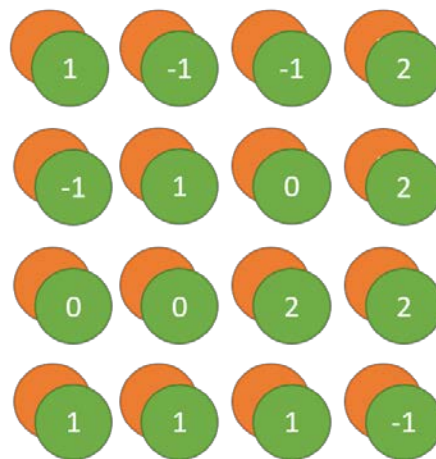


1	0	0	1	0	1
0	0	1	1	1	0
1	1	1	0	0	0
1	0	1	0	1	1
0	1	0	0	1	1
0	0	0	1	1	1

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



Convolution

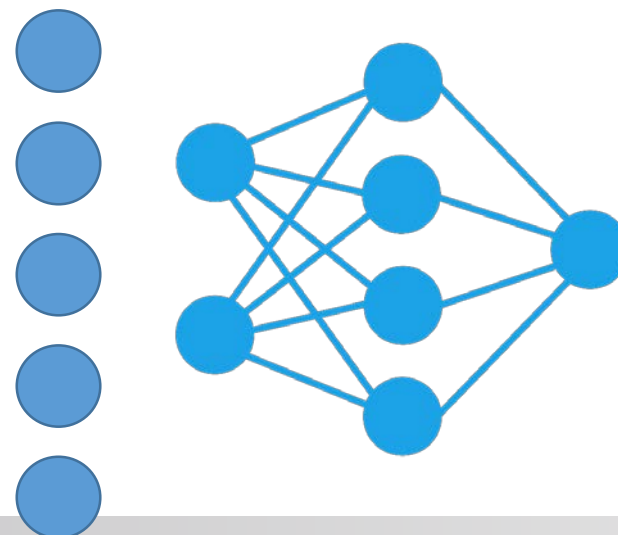


CNN

NN

1	0	0	1	0	1
0	0	1	1	1	0
1	1	1	0	0	0
1	0	1	0	1	1
0	1	0	0	1	1
0	0	0	1	1	1

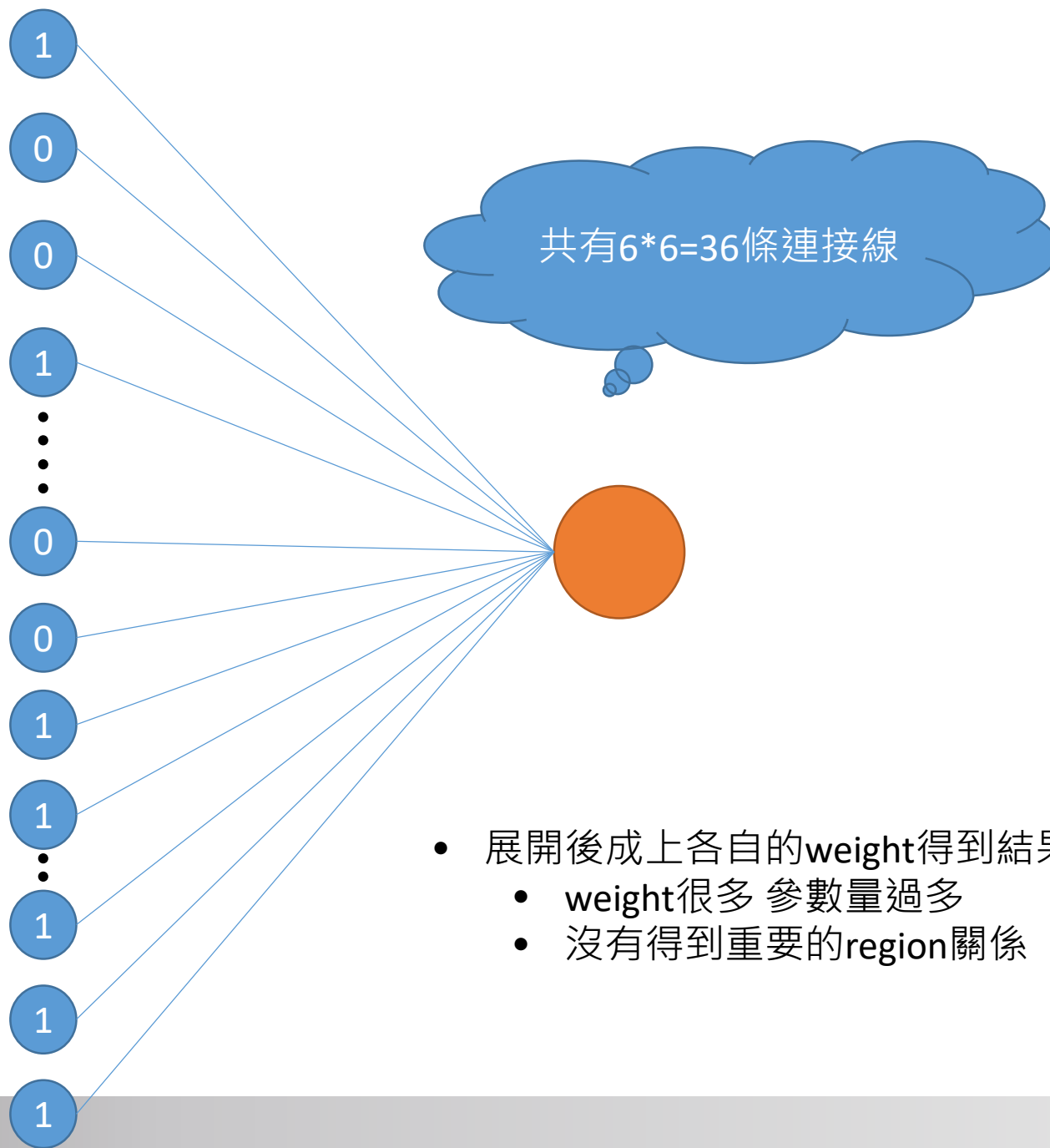
展開



## Fully connected

1	0	0	1	0	1
0	0	1	1	1	0
1	1	1	0	0	0
1	0	1	0	1	1
0	1	0	0	1	1
0	0	0	1	1	1

展開

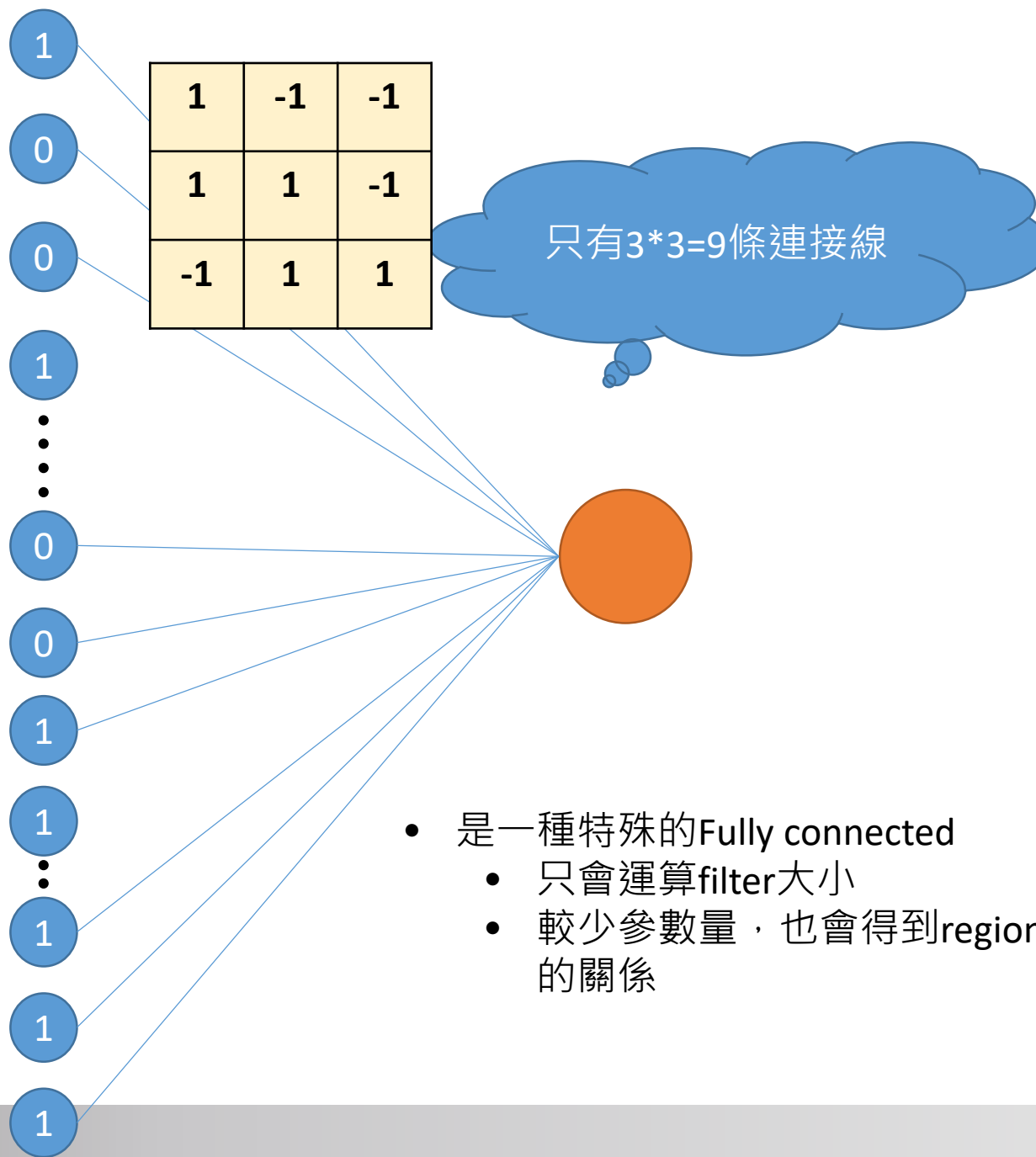


- 展開後成上各自的weight得到結果
  - weight很多 參數量過多
  - 沒有得到重要的region關係

## Convolution

1	0	0	1	0	1
0	0	1	1	1	0
1	1	1	0	0	0
1	0	1	0	1	1
0	1	0	0	1	1
0	0	0	1	1	1

展開

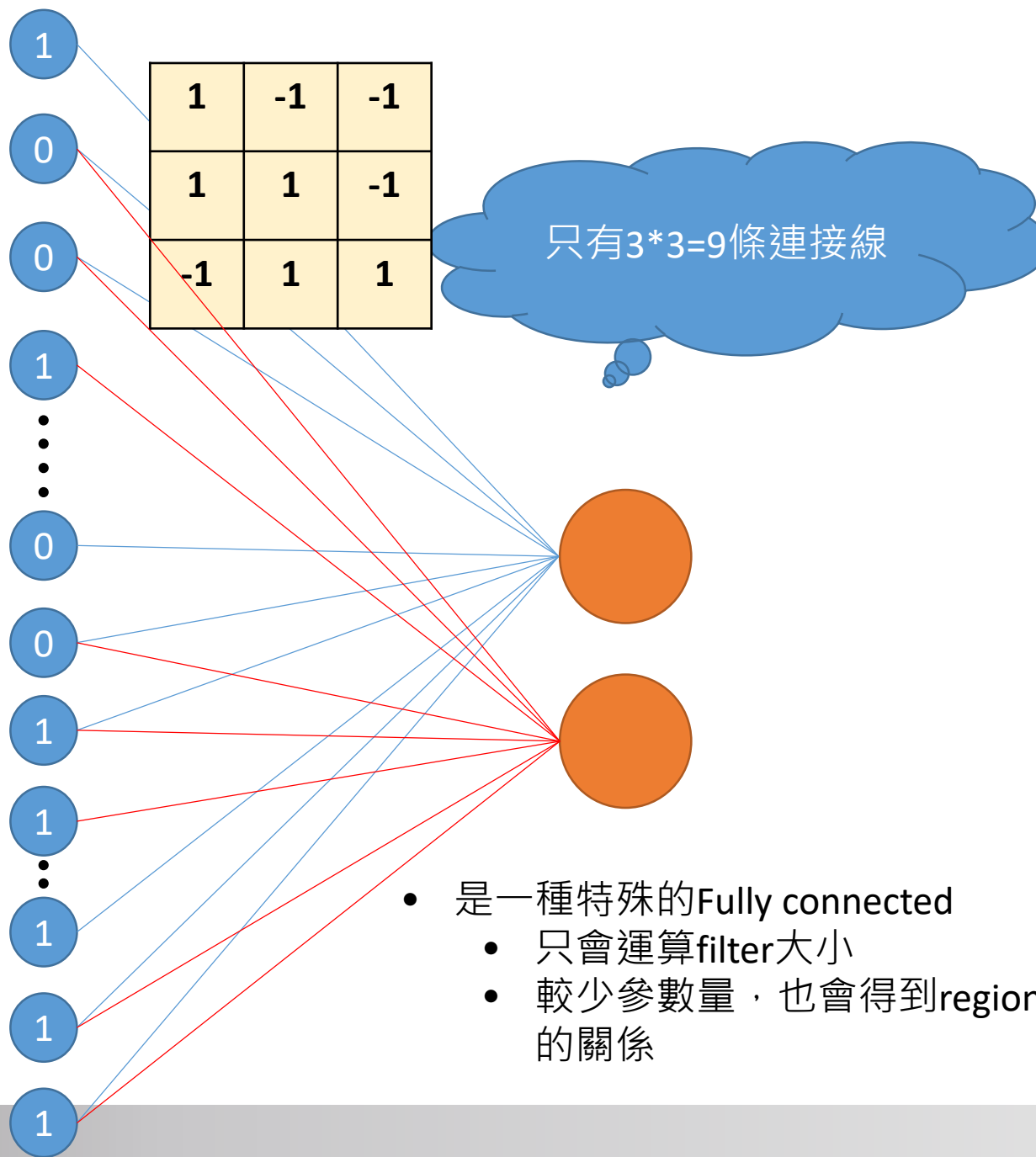




## Convolution

1	0	0	1	0	1
0	0	1	1	1	0
1	1	1	0	0	0
1	0	1	0	1	1
0	1	0	0	1	1
0	0	0	1	1	1

展開



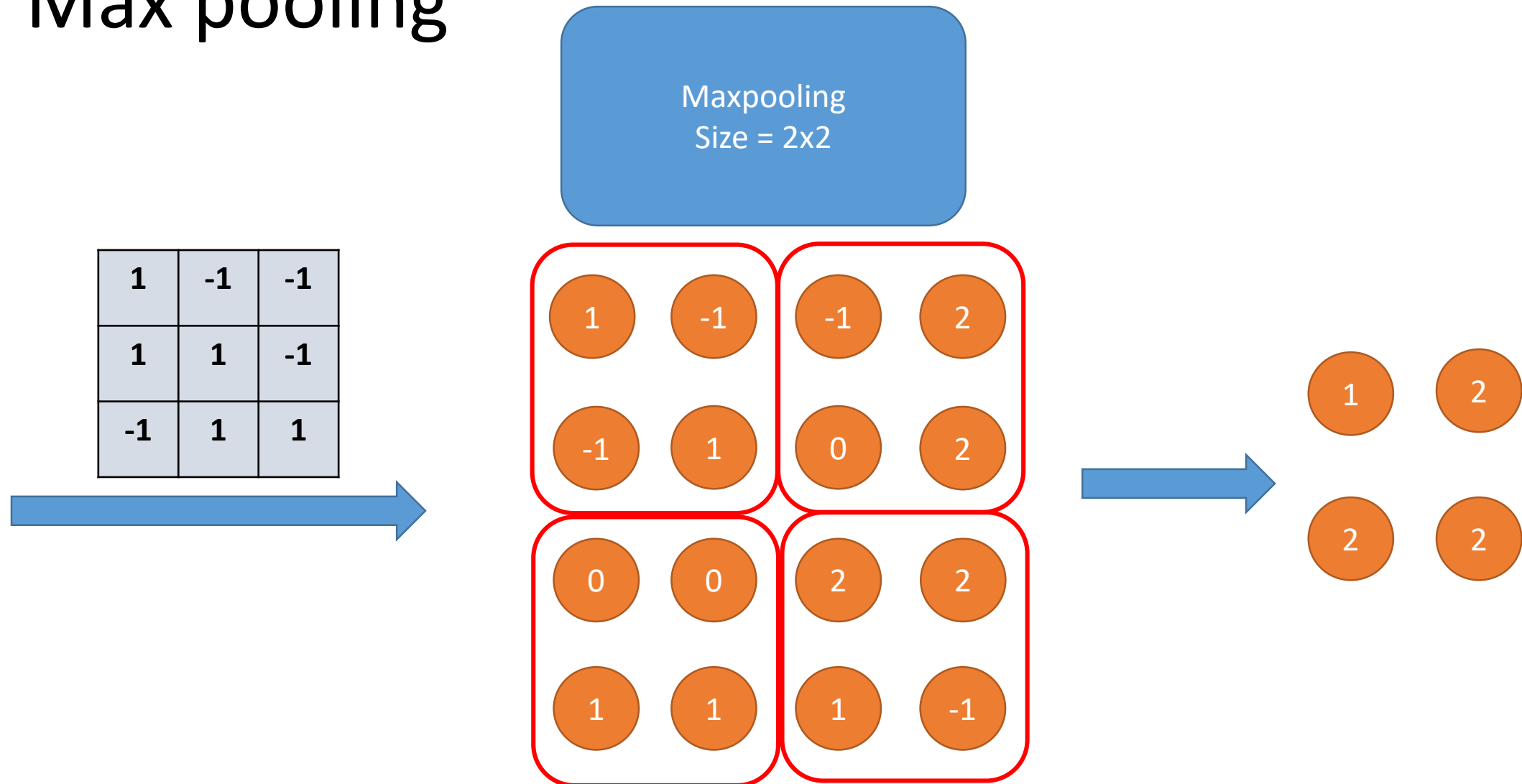
# Convolution

- 相較整張圖，明顯的特徵遠小於長張圖片
- 特徵會出現在不同區塊

# Max pooling

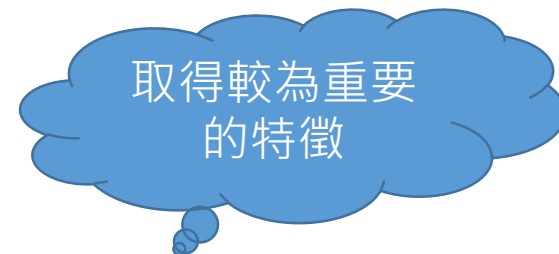
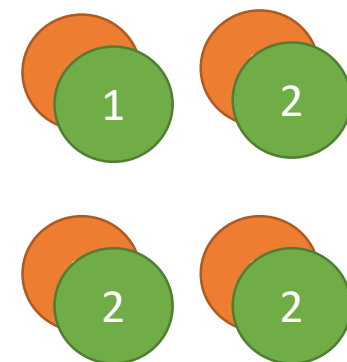
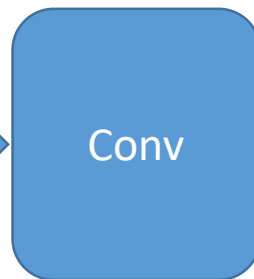
- 取較大的特徵點
- 可以拿到較重要的特徵

# Max pooling



# Max pooling

1	0	0	1	0	1
0	0	1	1	1	0
1	1	1	0	0	0
1	0	1	0	1	1
0	1	0	0	1	1
0	0	0	1	1	1



每一個filter會新增一個channel

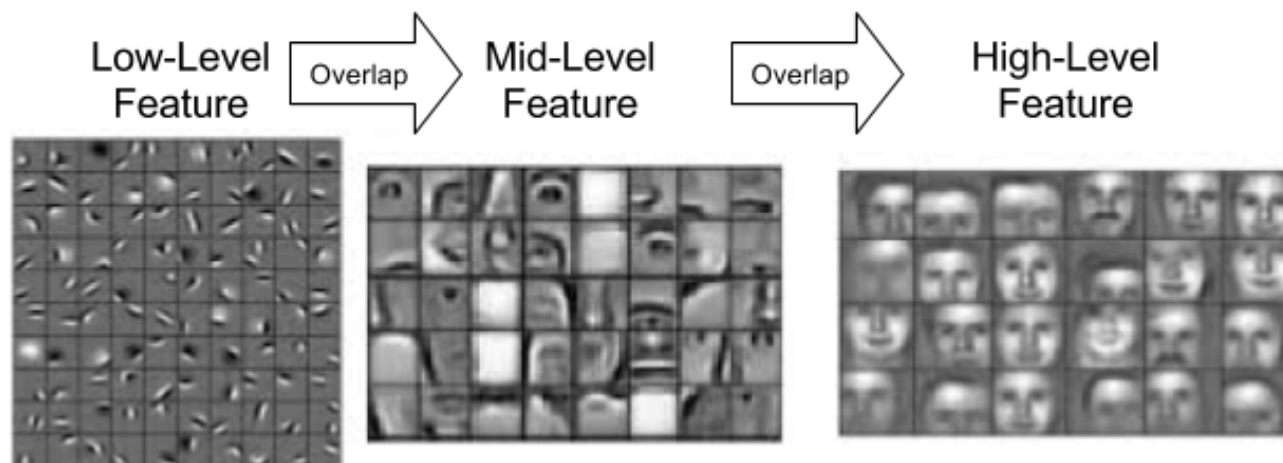
# Feature map

- 前幾層
  - 只看到local的特徵、feature
- 後幾層
  - 看到global的特徵、feature
    - 因為maxpooling downsample過

代表太淺層無法看到global特徵，帶出一個觀念

**Receptive field:** 當層可看到原始圖片的多大範圍

Feature Map in Convolutional Neural Networks (CNN)



<https://www.quora.com/How-does-a-convolutional-neural-network-recognize-an-occluded-face>

# Receptive field

- 當層可看到原圖的多少範圍
  - 經過三層Conv(3x3)+三層max(2x2)
    - $2*2*2*3 = 24$
    - 最終看到原始圖片24\*24的大小
    - 若重要的feature可能是48\*48，則遠不夠

# 卷積神經網路

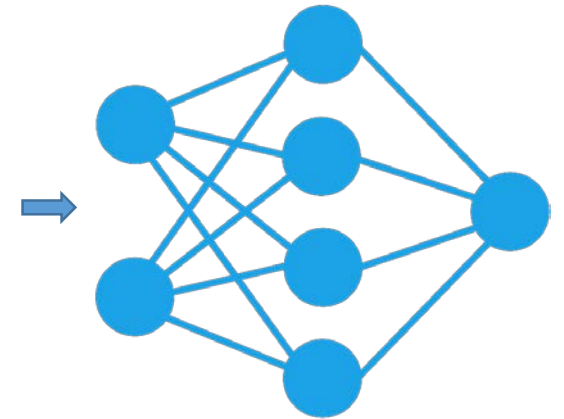
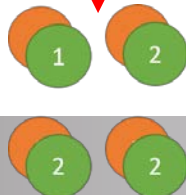
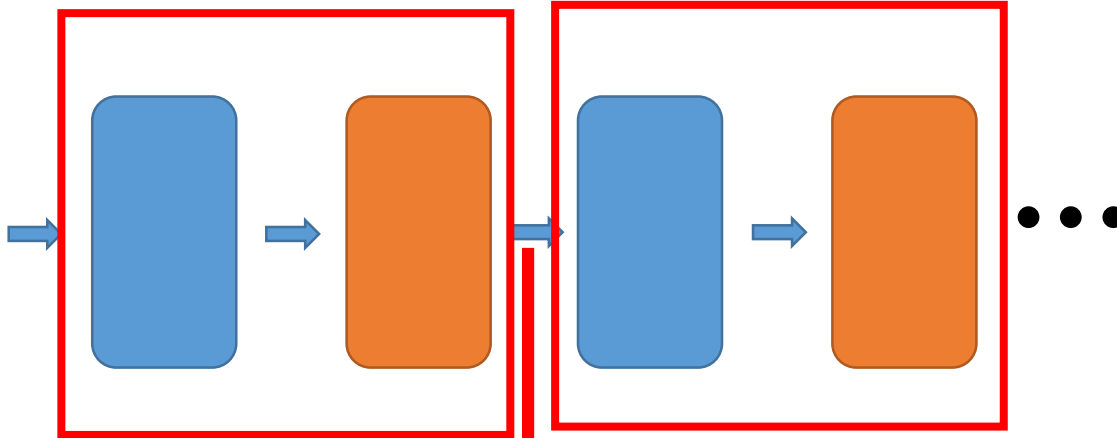
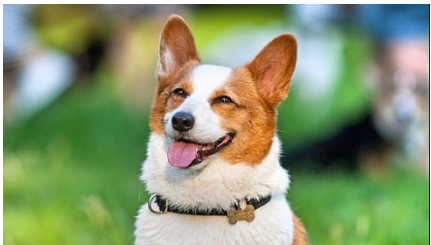
Convolution

Max pooling

展開  
(flatten)

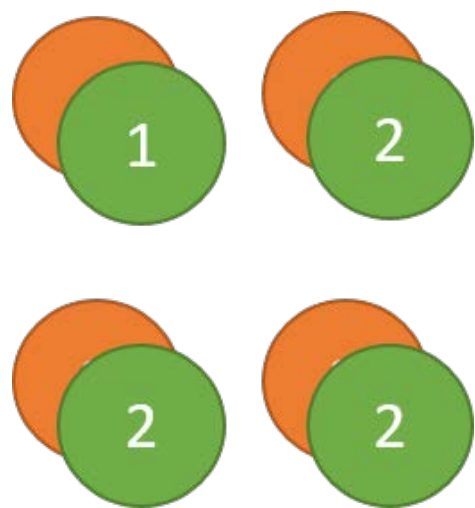
狗?

產生新的圖片，會小  
於原本圖片



# Flatten

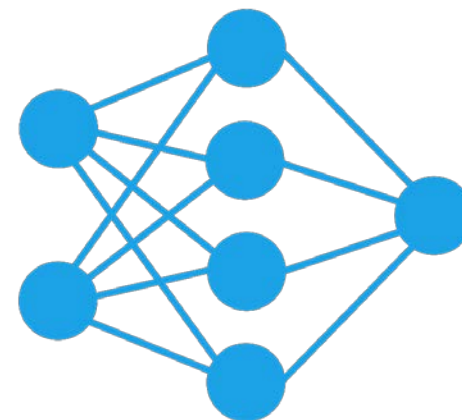
- Convolution+Max pooling後 會產生新的圖片
  - 如何接到fully connected?
    - 將圖片展開來



Flatten

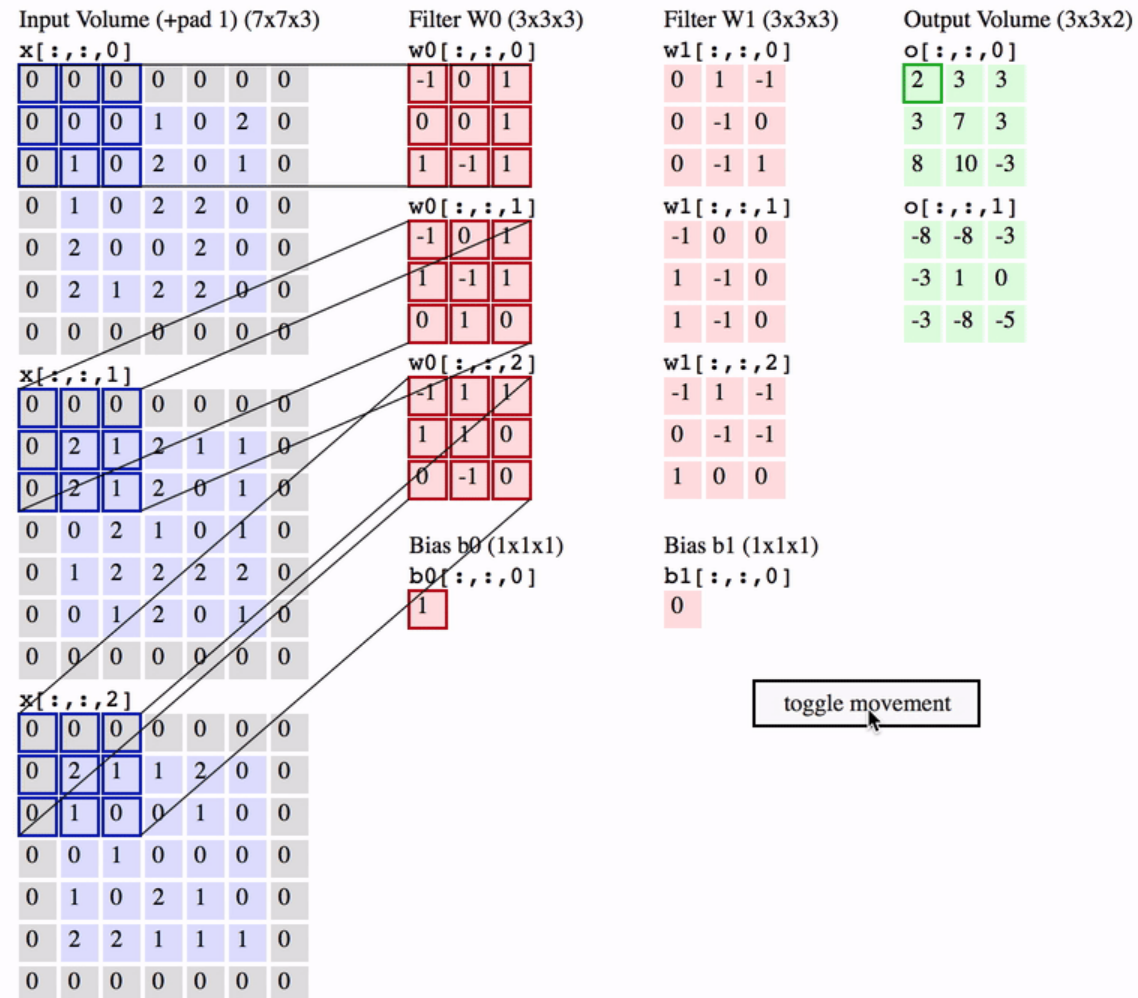


Fully Connected layer





# 動畫



# 應用

- Convolution 主要應用於圖像辨識
  - 比起單純的NN，準確率提升、參數量下降
- Signal訊號
  - 聲音訊號:speech
  - Sensor訊號:Rabboni

# 實戰教學

CNN with Keras

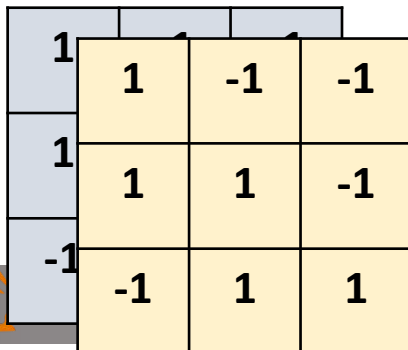
# CNN with Keras

```
from keras.models import Sequential
from keras.layers.core import Dense, Activation, Conv2D
from keras.optimizers import SGD

# 宣告這是一個 Sequential 次序性的深度學習模型
model = Sequential()

model.add(Conv2D(32, (3, 3), input_shape=(28, 28, 1)))
```

32個filter(通常是2的幕次方),每個都3x3



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

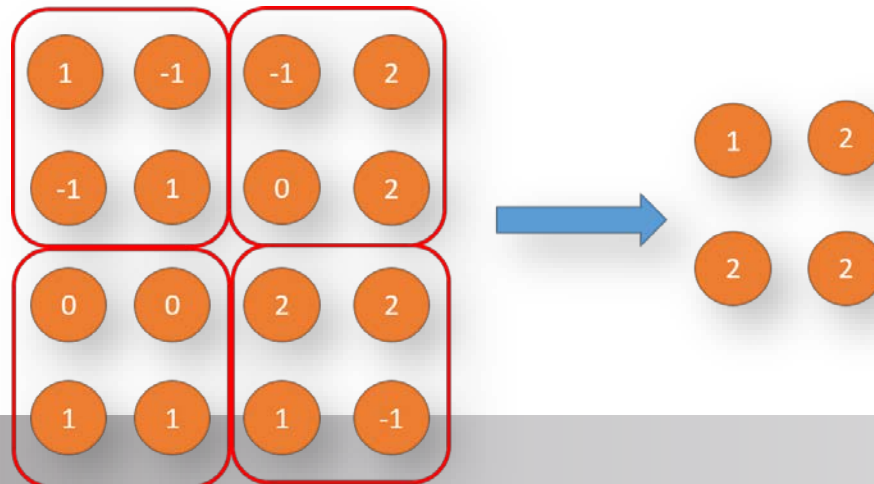
圖案:28x28x1, 1是灰階圖, 而RGB彩色圖則為3

# CNN with Keras

```
from keras.models import Sequential
from keras.layers import Dense, Activation, Conv2D, MaxPool2D
from keras.optimizers import SGD

# 宣告這是一個 Sequential 次序性的深度學習模型
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(28, 28, 1)))
Model.add(MaxPool2D((2, 2)))
```

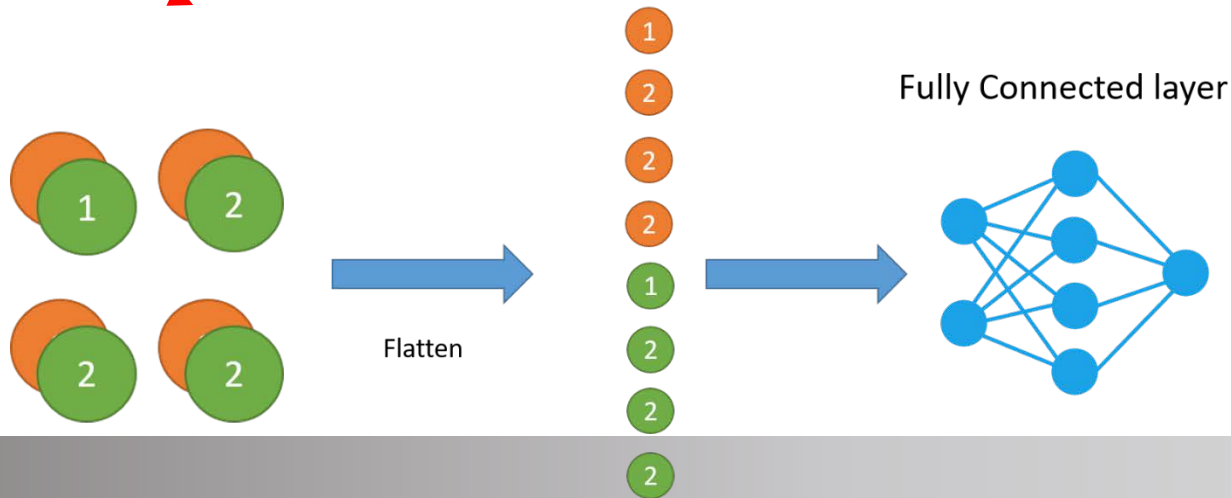
Maxpooling 2x2



# CNN with Keras

```
from keras.models import Sequential
from keras.layers import Dense, Activation, Conv2D, MaxPool2D, Flatten
from keras.optimizers import SGD

model = Sequential()
model.add(Conv2D(32, (3,3), input_shape=(28,28,1)))
model.add(MaxPool2D((2,2))
model.add(Flatten())
model.add(Dense(256))
```



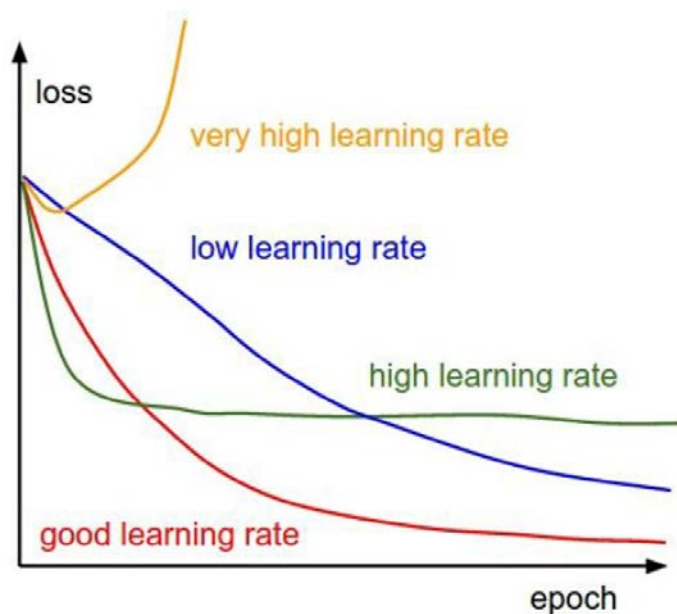
# 訓練技巧

# 選擇 Loss Function

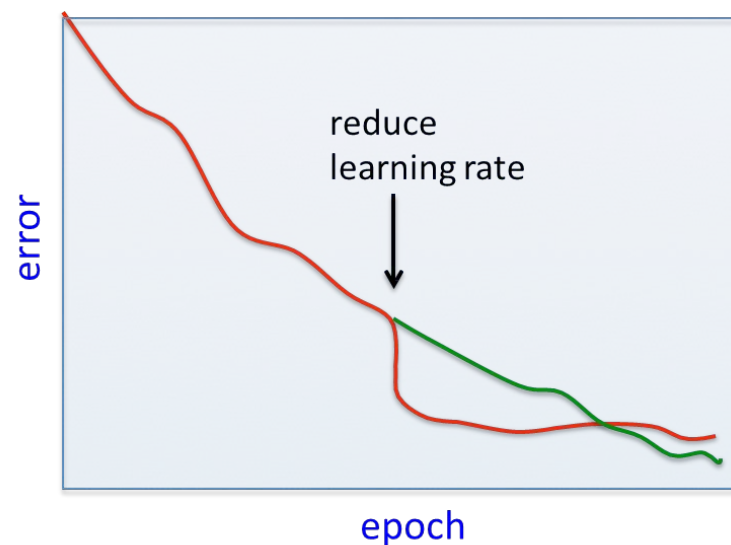
- Classification 常用 cross-entropy
  - 搭配 softmax 當作 output layer 的 activation function
- Regression 常用 mean absolute/squared error
- 對特定問題定義 loss function
  - Unbalanced dataset, class 0 : class 1 = 99 : 1
  - Total loss = 99 \* Class 1 loss + Class 0 loss



# Learning Rate

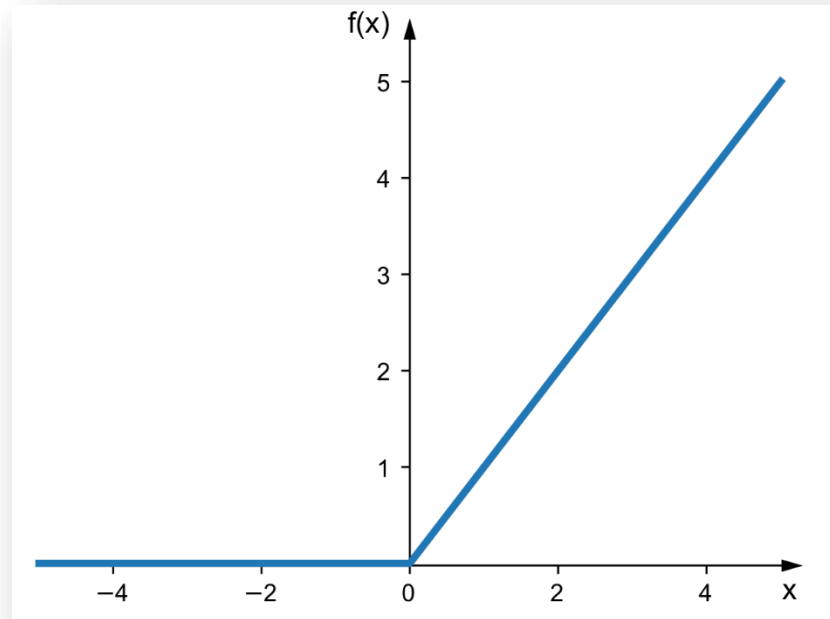


- 建議一次降一個數量級，如: 0.01 vs 0.001
- 一開始較大，後面較小避免上下震盪



# Activation function

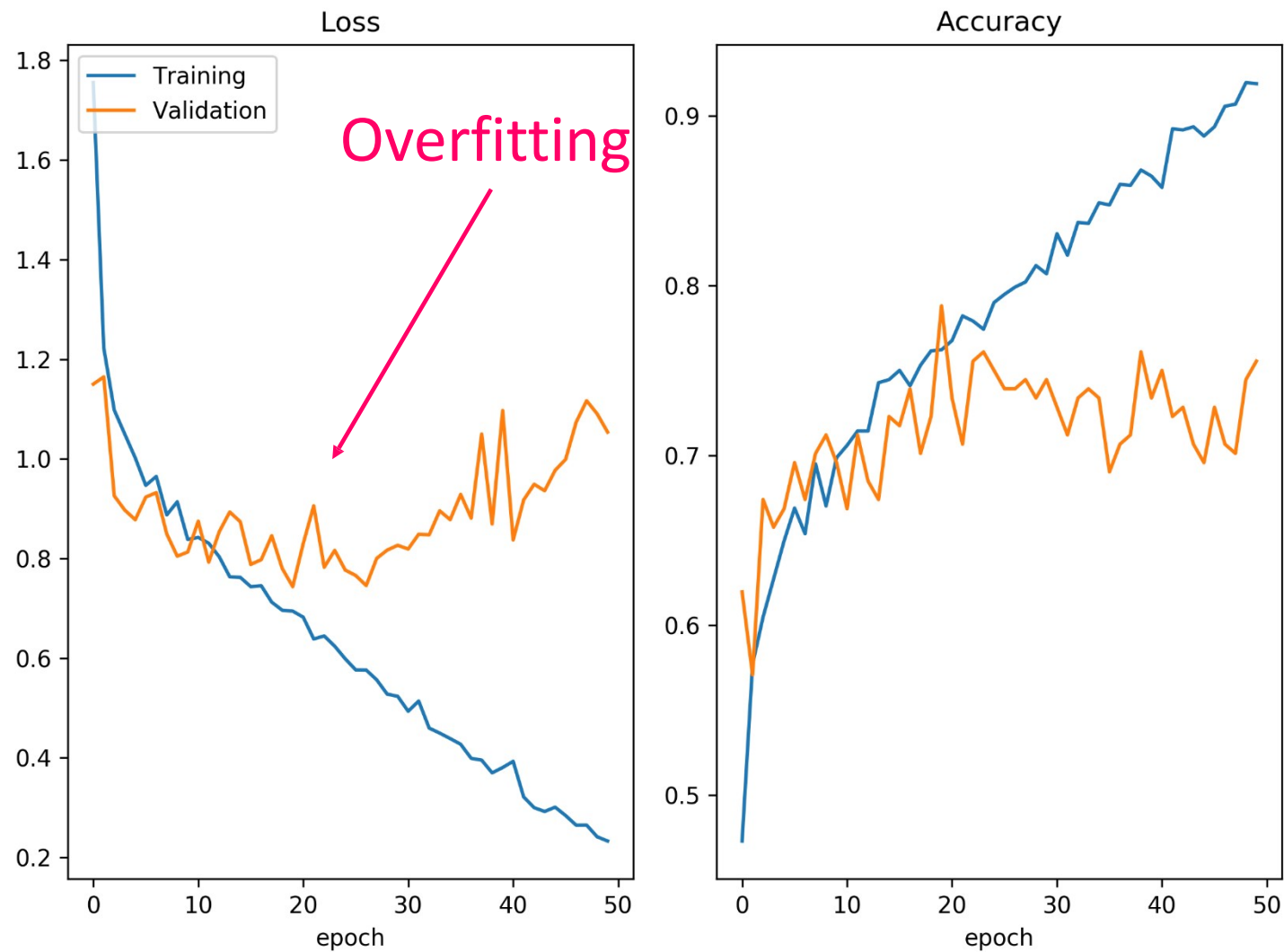
- Hidden layers
  - 通常會用 ReLU
  - 過去也有人使用tanh
- Output layer
  - Regression: linear
  - Classification: softmax



# Optimizer

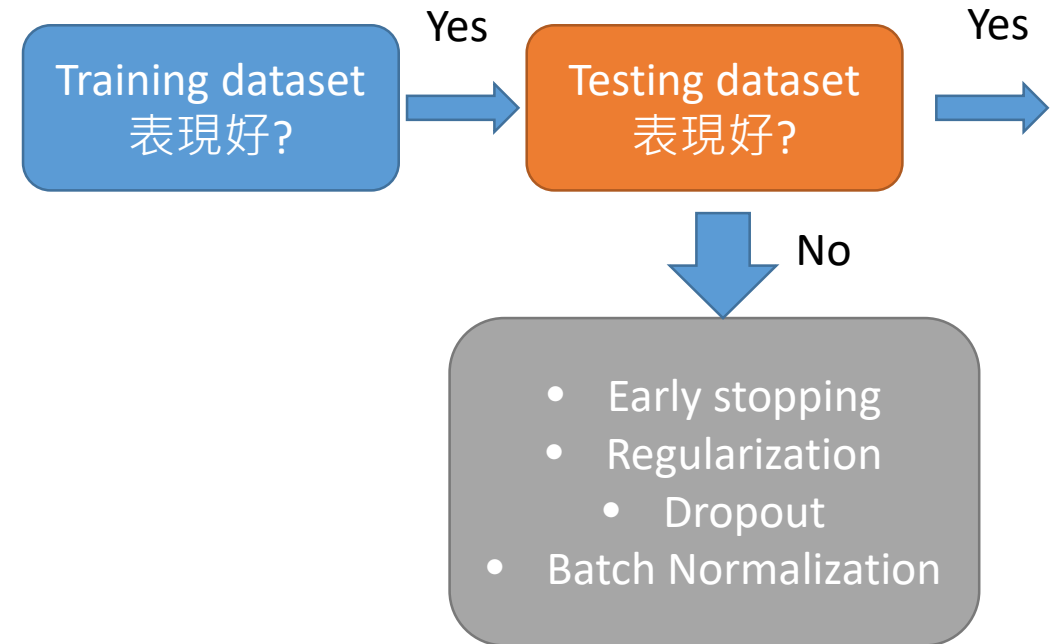
- 一般的起手式：Adam
  - Adaptive learning rate for every weights
  - Momentum included

# Validation/Testing結果如何?



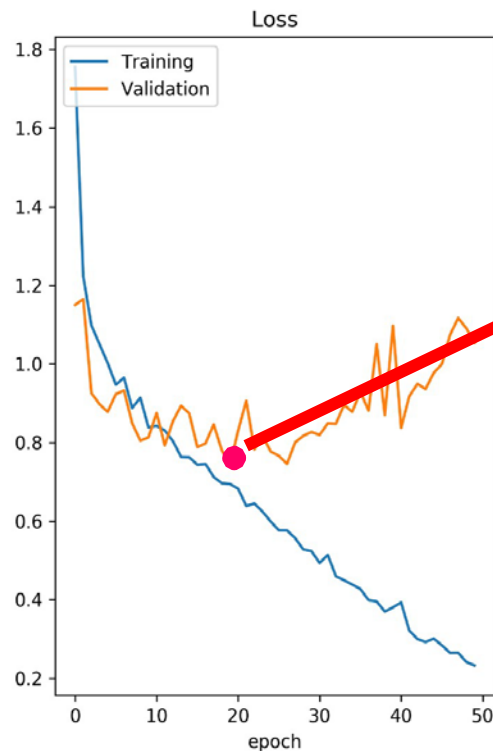
# 過擬合Overfitting?

- Training result 很好，但是在validation或是testing上卻越來越差
  - Model的參數適合training dataset
  - Model的參數不適合testing dataset
- 方法
  - 增加資料量、變異、多樣 – 根本
  - Early stopping
  - Regularization
  - Dropout
  - Batch Normalization



# Early Stopping

- 在validation變差前停下來

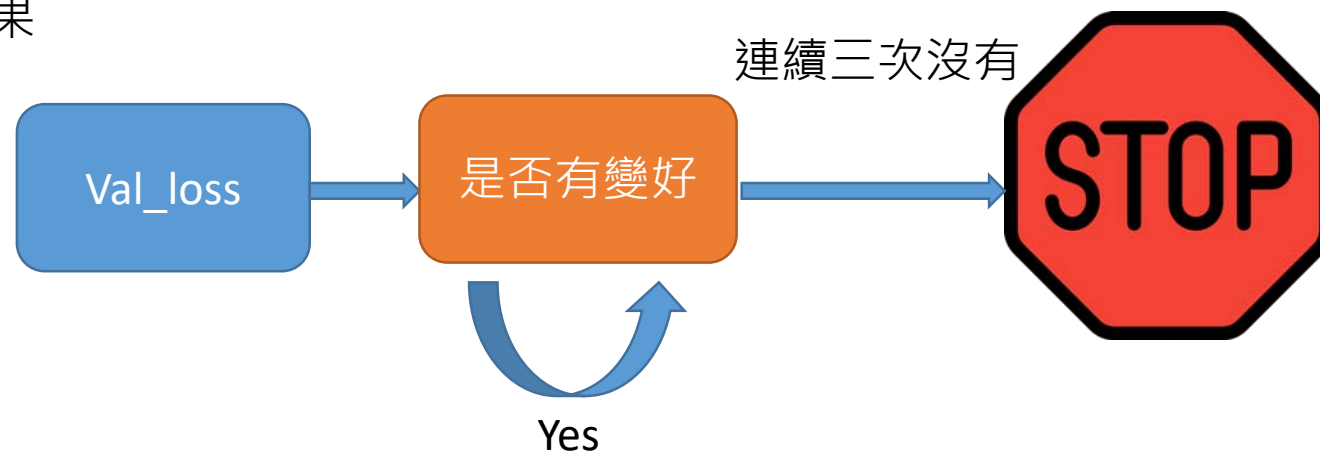


- 在此處停下來是最好的，因為接下來的 validation error 都在上升
- 會設定一定的次數，連續幾次的變差來觀察未來是否有更好的結果
- 可觀察 loss 或是 accuracy(分類)

# Early Stopping

```
from keras.callbacks import EarlyStopping  
earlyStopping=EarlyStopping(  
    monitor = 'val_loss',  
    patience = 3)
```

- monitor: 觀察的目標
- Patience: 可以容忍連續幾次不好的結果



# Regularization

$\lambda$  = regularization strength  
(hyperparameter)

- 限制weight自由度，降低overfitting機率
- 作法: 加入loss function

$$L = \frac{1}{N} \sum_{i=1}^N \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) + \lambda R(W)$$

In common use:

**L2 regularization**

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

**L1 regularization**

$$R(W) = \sum_k \sum_l |W_{k,l}|$$



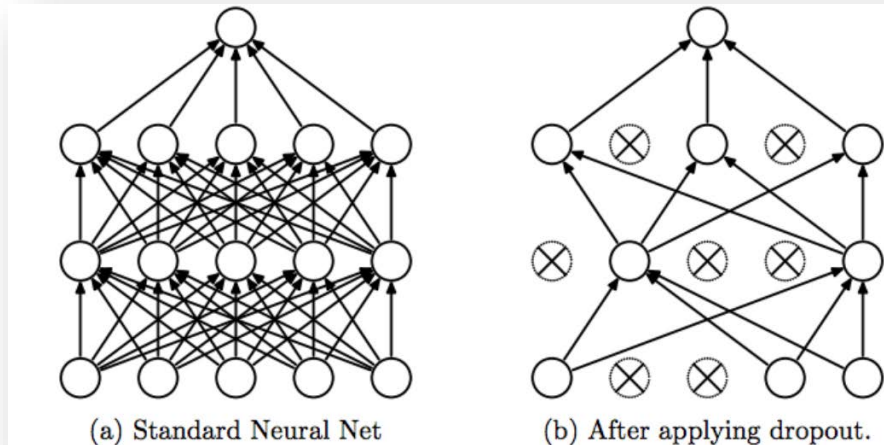
```
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import SGD
from keras.regularizers import l1, l2
```

```
# 宣告這是一個 Sequential 次序性的深度學習模型
model = Sequential()
```

```
model.add(Dense(128, input_dim=200, kernel_regularizer=l2(0.01)))
```

# Dropout

- 使用在fully connected layer中間
- 原本layer間的neurons & neurons 為全部連接
- training時，隨機拿掉一些連接不訓練
- testing時，還原成全部連接
- 效用
  - 訓練難度增加(training loss比較高)，但在testing會比較好
  - 等效於訓練會有  $2^N$  種 network structures
  - 會造成training performance變差
    - 不建議一開始就用dropout
    - 在有overfitting時再用

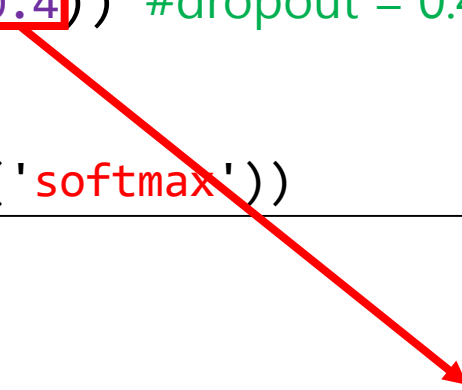


# Dropout

```
from keras.layers import Dense, Activation, Dropout
model = Sequential()

# 加入第一層 hidden layer
model.add(Dense(128, input_dim=200))
model.add(Activation('sigmoid'))
model.add(Dropout(0.4)) #dropout = 0.4
model.add(Dense(256))
model.add(Activation('sigmoid'))
model.add(Dropout(0.4)) #dropout = 0.4

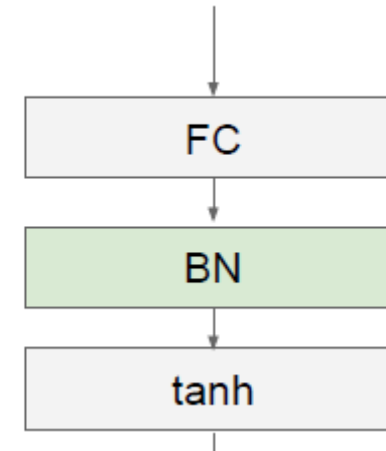
model.add(Dense(5))
model.add(Activation('softmax'))
```



捨棄的機率

# Batch Normalization

- 輸入前建議做normalization
  - 比較好訓練，容易收斂
- 如果網路很深，中間層的輸出會因為nonlinear activation而無法控制
  - 中間每一層都做normalization
- 加在FC後，activation前
- 好處
  - 取代 dropout & regularization
  - 加速訓練
  - 可以用比較大的 learning rate
- 目前大多數都會加 (基本組合之一)



```
from keras.layers import Dense, Activation, BatchNormalization
model = Sequential()

# 加入第一層 hidden layer
model.add(Dense(128, input_dim=200))
model.add(BatchNormalization())
model.add(Activation('sigmoid'))

model.add(Dense(256))
model.add(BatchNormalization())
model.add(Activation('sigmoid'))

model.add(Dense(5))
model.add(Activation('softmax'))
```

# 根本-增加資料量 Data augmentation

- 資料量有限
  - 時間、經費等等因素無法蒐集大量資料
- “**假造**”資料
- 圖像為例
  - 旋轉、鏡射、縮放、變色、雜訊去增加資料量



# 資料的重要性

# 資料種類

原則：接近目標使用情境

多樣性

人：各種年齡、男女、身高

環境：水泥地、上下坡、地板材質、

資料量

越多越好

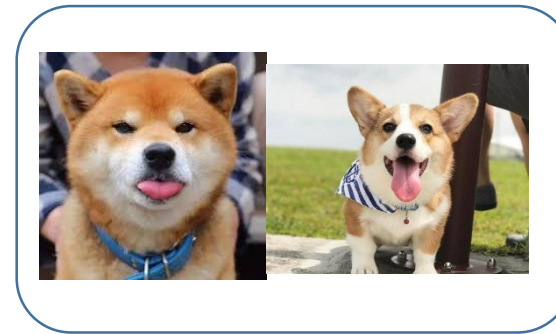
每一分析種類數量平均



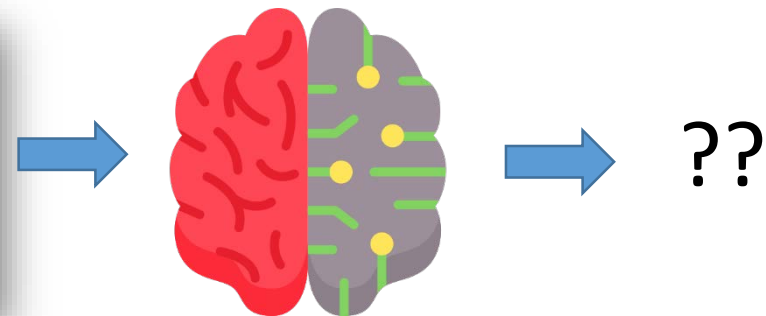
# 多樣性

- 分類
  - 各種血統的貓狗
  - 不同年齡的男女
  - 不同身高的男女
- 為何需要多樣的資料?
  - 單一類型資料
    - 模型只學到**單一類型**的特徵
    - 實際應用在**不同類型**會失敗

Training dataset



只含有柴犬與柯基跟貓



??

# 資料量

- 越多越好
  - 可以增加模型的穩定與準確
- 類型平均
  - 若數量偏向某類型
    - 訓練出來的模型可能只會針對該類型優化
- 不夠?
  - Data augmentation
  - Pre-train & fine-tuning

# Pre-train & Fine-tuning

- 目的
  - 利用強大的網路
  - GoogLeNet、VGG16等等
  - 重新訓練需要長久時間
  - 資料量不夠
- 方式
  - 使用前人的傑作
  - 使用已訓練過的模型參數(Pre-train)
  - 只改變幾層進行訓練(Fine-tuning)
- 前提
  - 類型相同 容易得到相同的參數
  - 拿classification pre-train model 運用在regression上
    - 不太適合