卷積神經網路

Convolution Neural Network

國立政治大學金融科技研究中心
智能理財與深度學習暑期訓練營

大綱

Outline

What is CNN

Application of CNN

Component of CNN

Demo

課程地圖

深度學習三大重點

eder @edersantana · 1天

When you gotta hustle selling neural nets in China. Already profitable AI startup!

6 111

CNN 很重要

看 Twitter 就知道...

What is CNN？

CNN為何物？

Convolution Neural Network

建立在卷積計算
參數的神經網路

卷積定義

Convolution Definition

$$f * g(x) = \int_{-\infty}^{\infty} f(t)g(x-t)\,dt$$

$f(x)$及$g(x)$為兩個函數

Why is CNN so popular ?

Back to the history…

卷積神經網路

Convolutional Neural Network

1979
Fukushima
提出
Deep Neocognitron
架構

卷積神經網路

Convolutional Neural Network

1979
Fukushima
提出
Deep Neocognitron
架構

1998
Yann LeCun
提出LeNet-5
將CNN用於字元辨識

Yann LeCun 楊立昆被稱為 CNN 之父

LeCun 的 LeNet-5模型

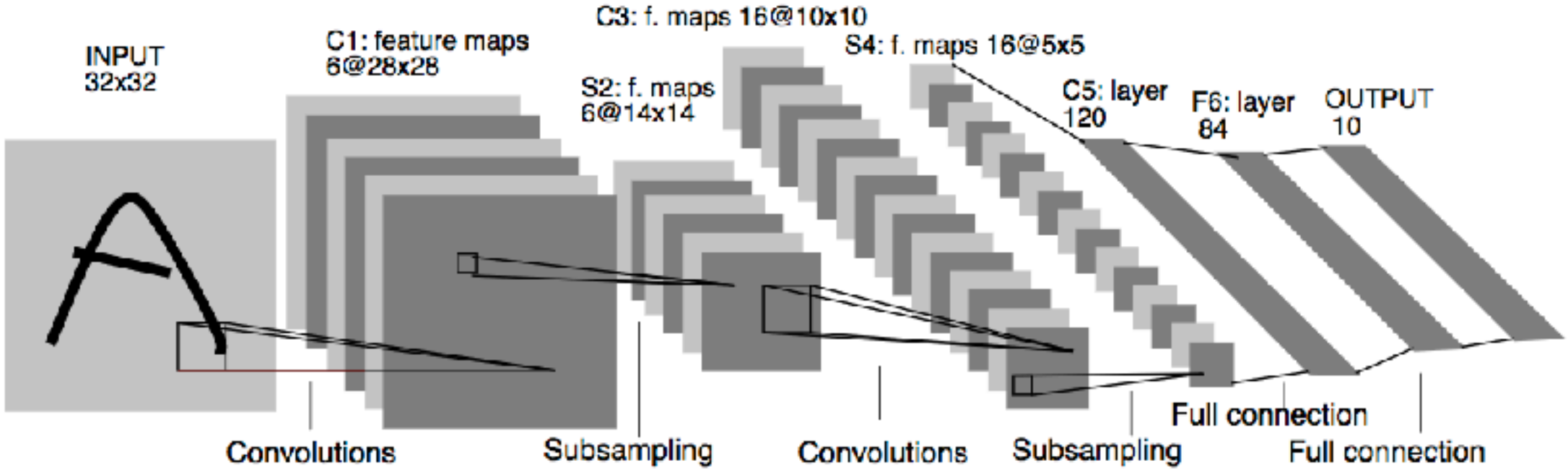


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

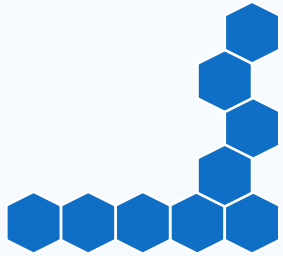
Src:http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf

雖然在1990s年代，大家期待
神經網路能帶來一些作為...

但結果像是圖形辨識
就被SVM 打趴了

2000年左右, 甚至有研究計畫出現
神經網路就不會通過的情況...

但是...



卷積神經網路

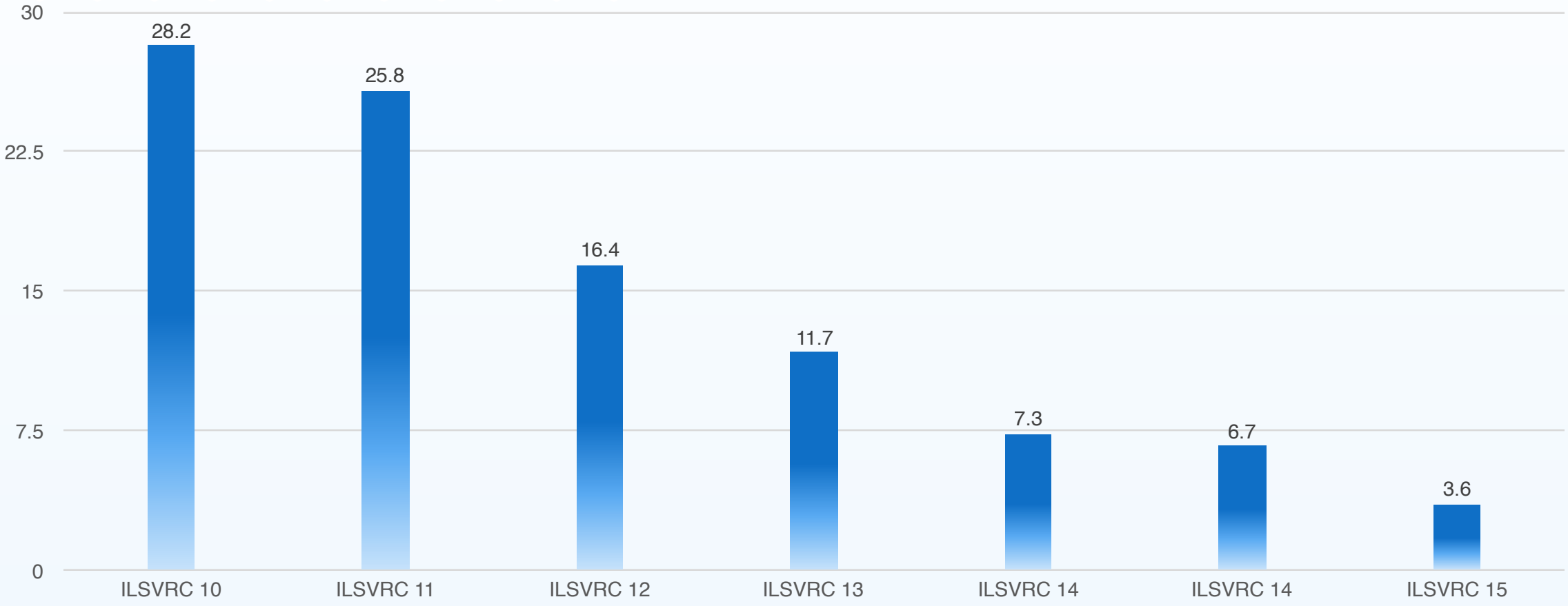
Convolutional Neural Network

1979
Fukushima
提出
Deep Neocognitron
架構

1998
Yann LeCun
提出LeNet-5
將CNN用於字元辨識

2012
CNN模型-AlexNet
在ILSVRC奪冠

ILSVRC的歷屆冠軍



從此大家在做圖形辨識, 幾乎
全用某種形式的 CNN

2015 Nature

由包括 LeCun 在內的深度學習三巨頭發表一篇就叫 “Deep Learning” 的文章

Application of CNN

CNN的應用

卷積神經網路

Convolutional Neural Network

CNN 模型其實基本上在做
特徵提取 及 特徵傳遞

卷積神經網路

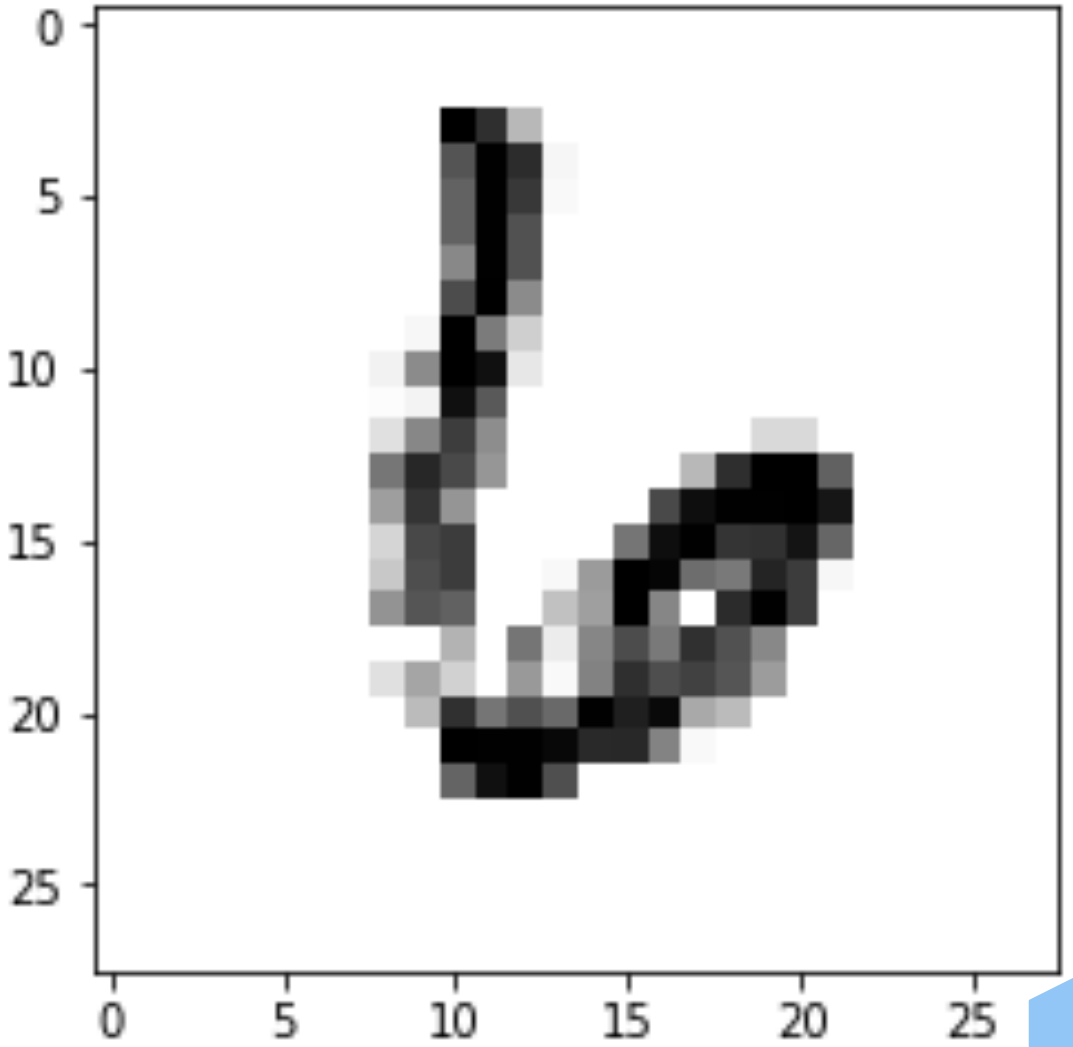
Convolutional Neural Network

通常CNN用於語音辨識與圖像辨識

但也可用於風格轉換等等應用

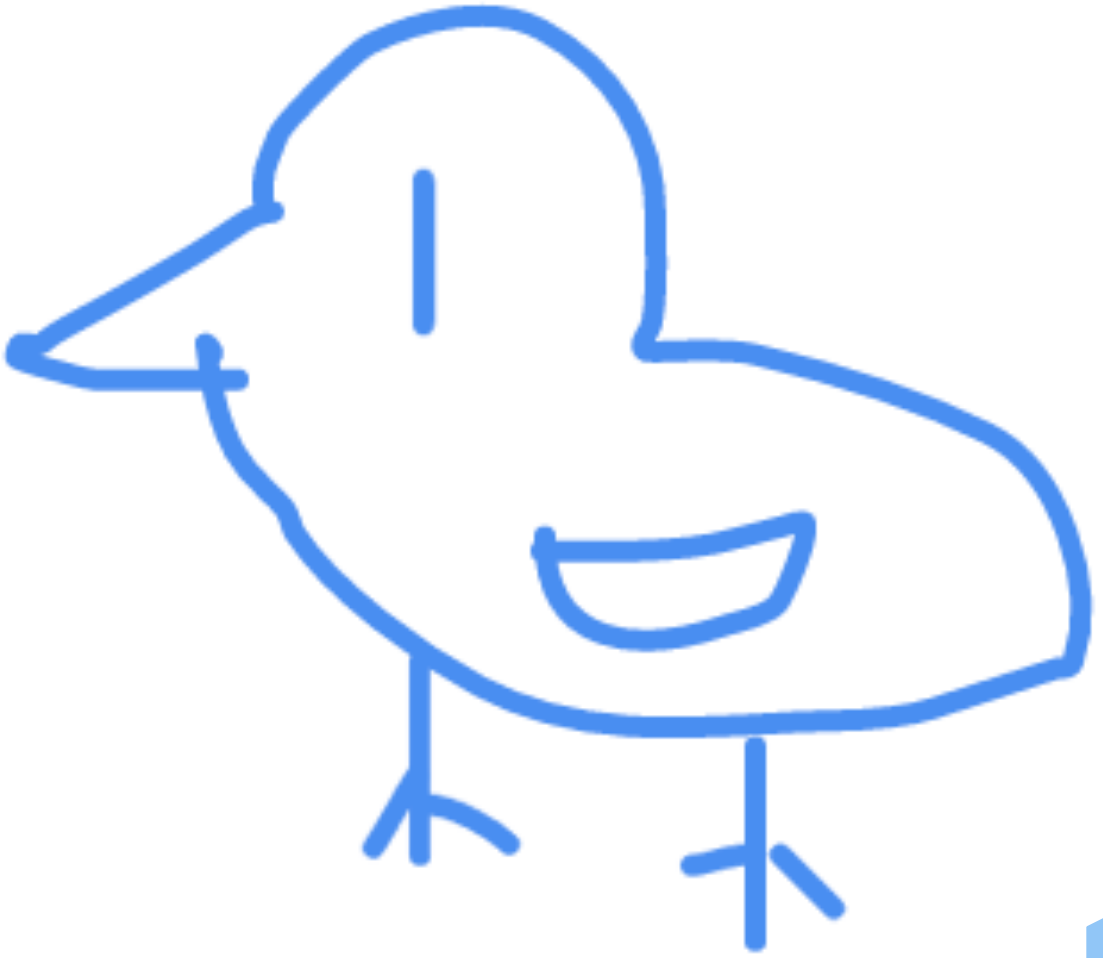
測試編號 6615

神經網路判斷為： 6



手寫辨識

MNIST data recognition

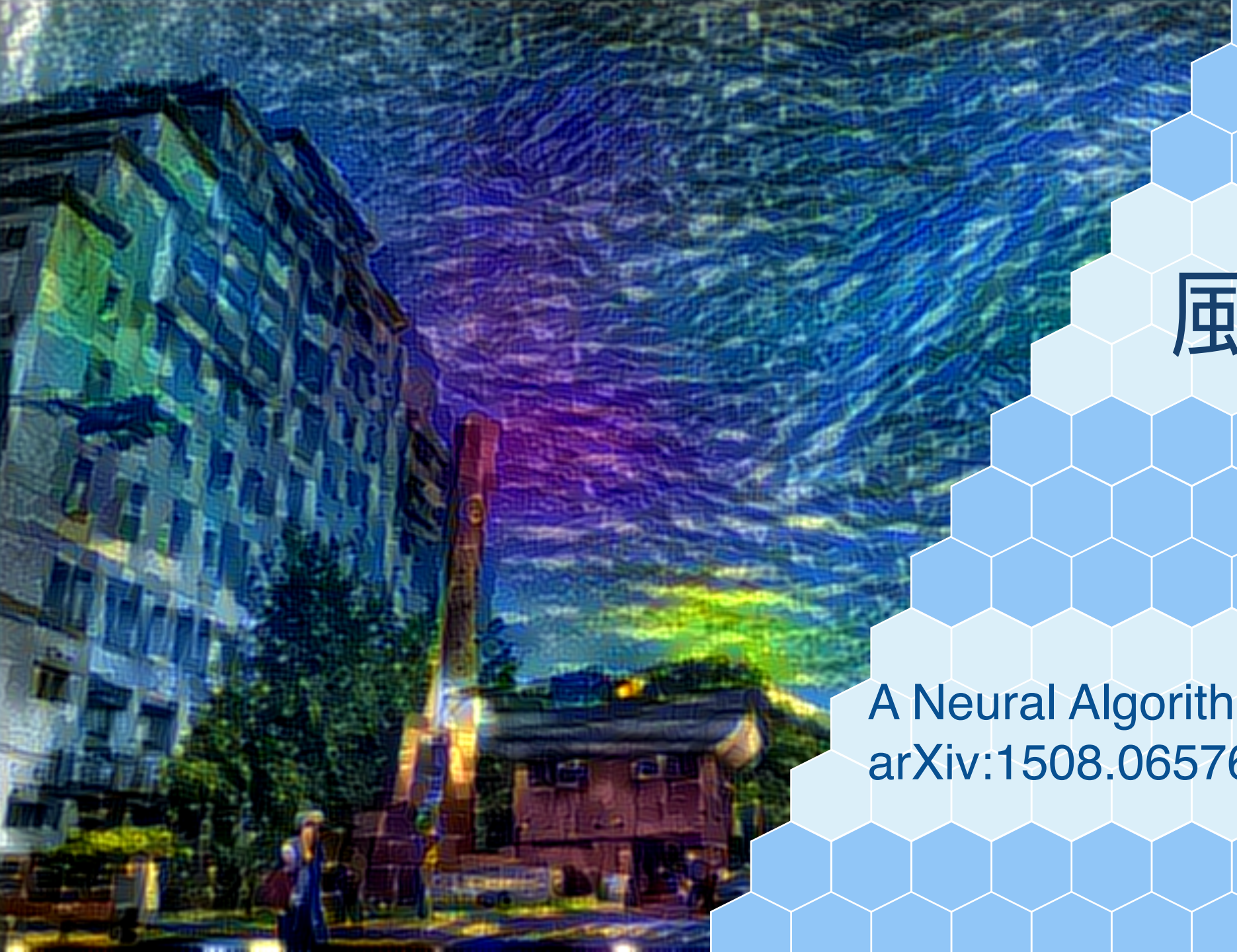


圖像辨識

Google AutoDraw
https://www.autodraw.com/

自動駕駛

自動駕駛的物件辨識是 CNN 的重要應用



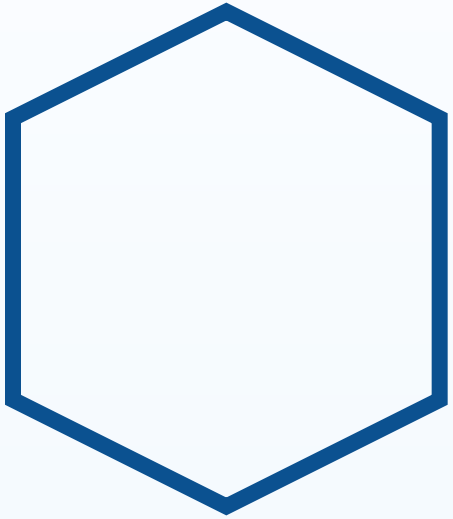
風格轉換

A Neural Algorithm of Artistic Style
arXiv:1508.06576

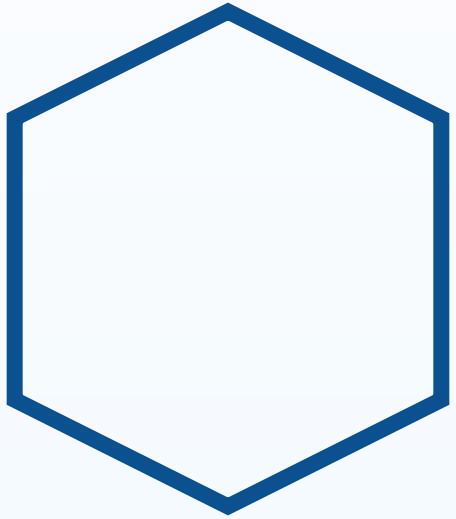
Components of CNN

CNN的組成

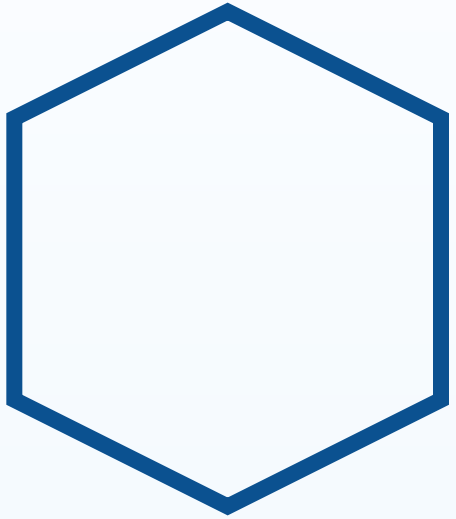
CNN 的組成



Convolutional Layer



Max-Pooling Layer



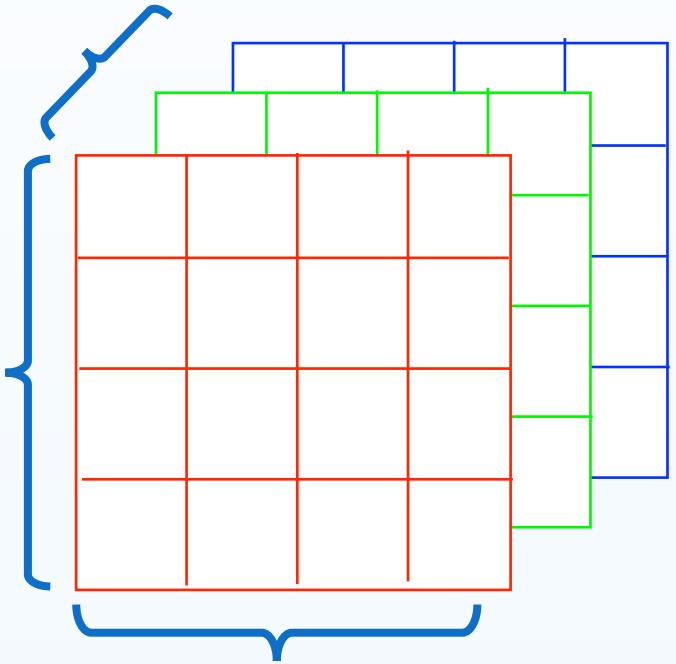
Fully Connected Neural Networks

輸入的資料形態

想像是一張圖

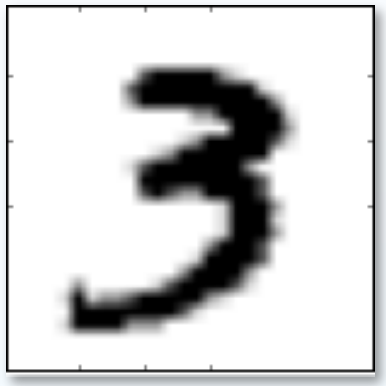
Depth
(channel)

Height



Width

輸入資料形態

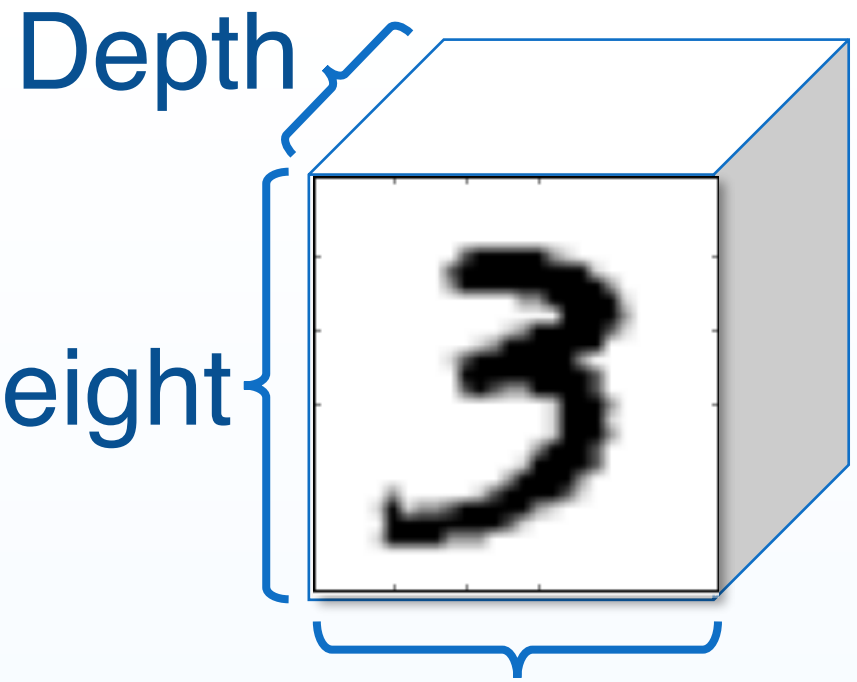


單色

Height: 28

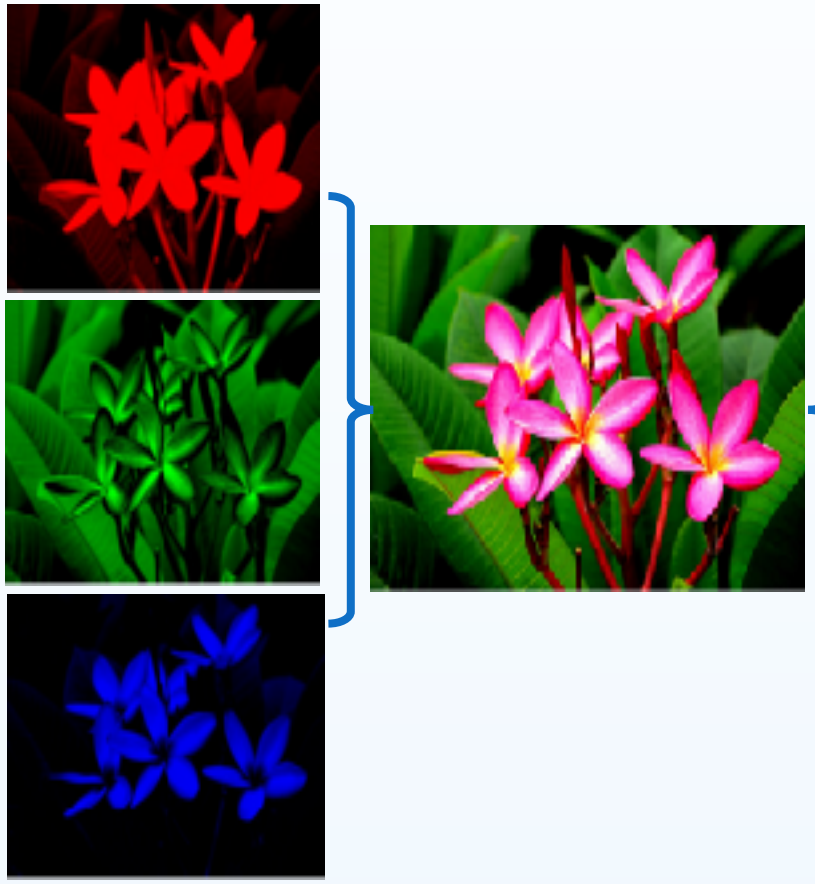
Width : 28

Depth : 1



彩色

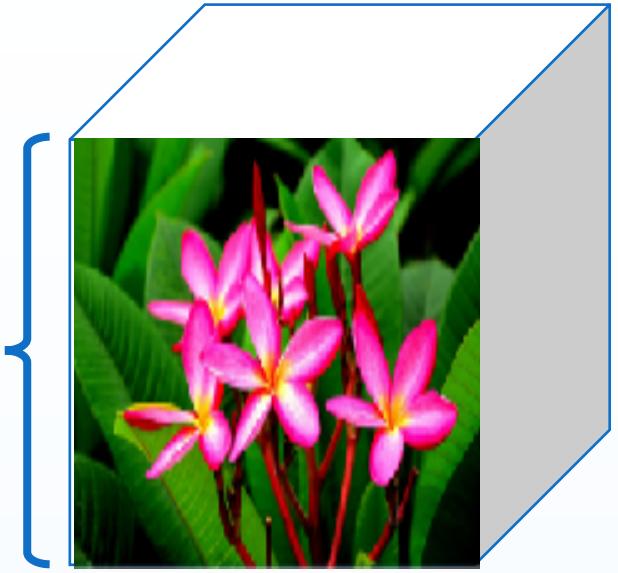
Input Data



Height: ImgH

Width : ImgW

Depth : 3



CNN
Model Structure

CNN
- Convolution Layer
- Pooling Layer (Subsampling)
- Fully Connected Layer

Convolution Layer

用 “filter” 找出圖片特徵

例如我們做 3x3 的 filters

內積

$$W = \begin{bmatrix} 1 & 0 & 1 \\ 2 & 1 & 3 \\ 1 & 1 & 2 \end{bmatrix}$$

filter

這學來的

想成這是一張圖所成的矩陣

2	5	5	2	5	2	0	1
2	3	4	0	4	2	1	5
4	3	1	3	5	5	4	3
5	3	4	5	0	2	1	5
2	3	1	1	1	0	1	3
4	4	1	1	5	1	1	4
2	3	2	2	0	4	2	4
0	5	4	5	3	4	1	4

3					
5					

Conv Layer

內積

$$W = \begin{bmatrix} 1 & 0 & 1 \\ 2 & 1 & 3 \\ 1 & 1 & 2 \end{bmatrix}$$

filter

還是一樣的矩陣

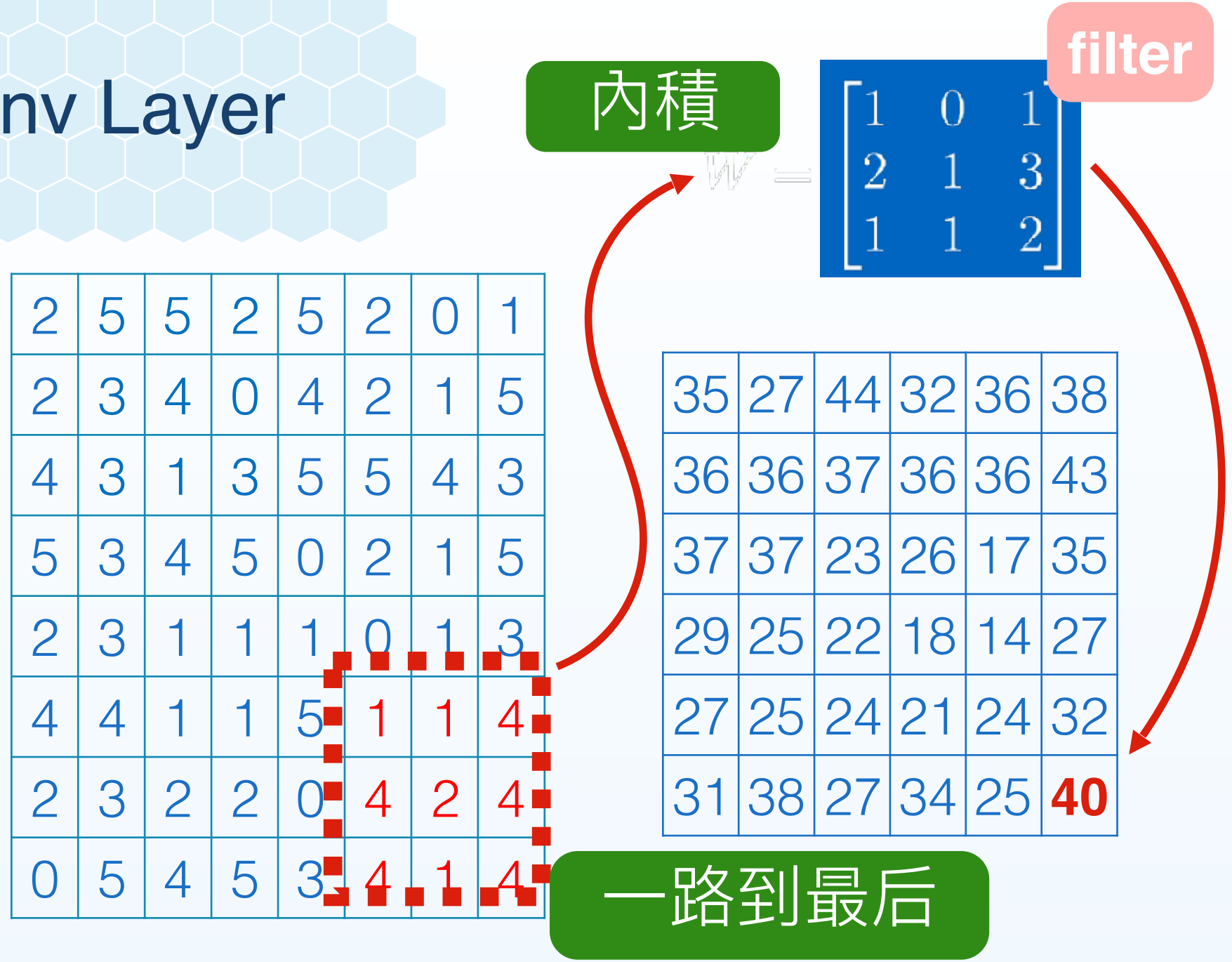
2	5	5	2	5	2	0	1
2	3	4	0	4	2	1	5
4	3	1	3	5	5	4	3
					2	1	5
2	3	1	1	1	0	1	3
4	4	1	1	5	1	1	4
2	3	2	2	0	4	2	4
0	5	4	5	3	4	1	4

右移一格

3	**2**				
5	**7**				

Conv Layer

Conv Layer


filter
$W = \begin{bmatrix} 1 & 0 & 1 \\ 2 & 1 & 3 \\ 1 & 1 & 2 \end{bmatrix}$
内積
2 5 5 2 5 2 0 1
2 3 4 0 4 2 1 5
4 3 1 3 5 5 4 3
5 3 4 5 0 2 1 5
2 3 1 1 1 0 1 3
4 4 1 1 5 1 1 4
2 3 2 2 0 4 2 4
0 5 4 5 3 4 1 4
35 27 44 32 36 38
36 36 37 36 36 43
37 37 23 26 17 35
29 25 22 18 14 27
27 25 24 21 24 32
31 38 27 34 25 40
一路到最后

神經元是這樣連結的

Conv 連結

圖片上的點是一個
個輸入層神經元

filter

$$W = \begin{bmatrix} 1 & 0 & 1 \\ 2 & 1 & 3 \\ 1 & 1 & 2 \end{bmatrix}$$

2	5	5	2	5	2	0	1
2	3	4	0	4	2	1	5
4	3	1	3	5	5	4	3
5	3	4	5	0	2	1	5
2	3	1	1	1	0	1	3
4	4	1	1	5	1	1	4
2	3	2	2	0	4	2	4
0	5	4	5	3	4	1	4

35	27	44	32	36	38
36	36	37	36	36	43
37	37	23	26	17	35
29	25	22	18	14	27
27	25	24	21	24	32
31	38	27	34	25	40

Conv 連結

filter

$$W = \begin{bmatrix} 1 & 0 & 1 \\ 2 & 1 & 3 \\ 1 & 1 & 2 \end{bmatrix}$$

2	5	5	2	5	2	0	1
2	3	4	0	4	2	1	5
4	3	1	3	5	5	4	3
5	3	4	5	0	2	1	5
2	3	1	1	1	0	1	3
4	4	1	1	5	1	1	4
2	3	2	2	0	4	2	4
0	5	4	5	3	4	1	4

35	27	44	32	36	38
36	36	37	36	36	43
37	37	23	26	17	35
29	25	22	18	14	27
27	25	24	21	24	32
31	38	27	34	25	40

Conv 層也是一個個神經元

Conv 連結

filter

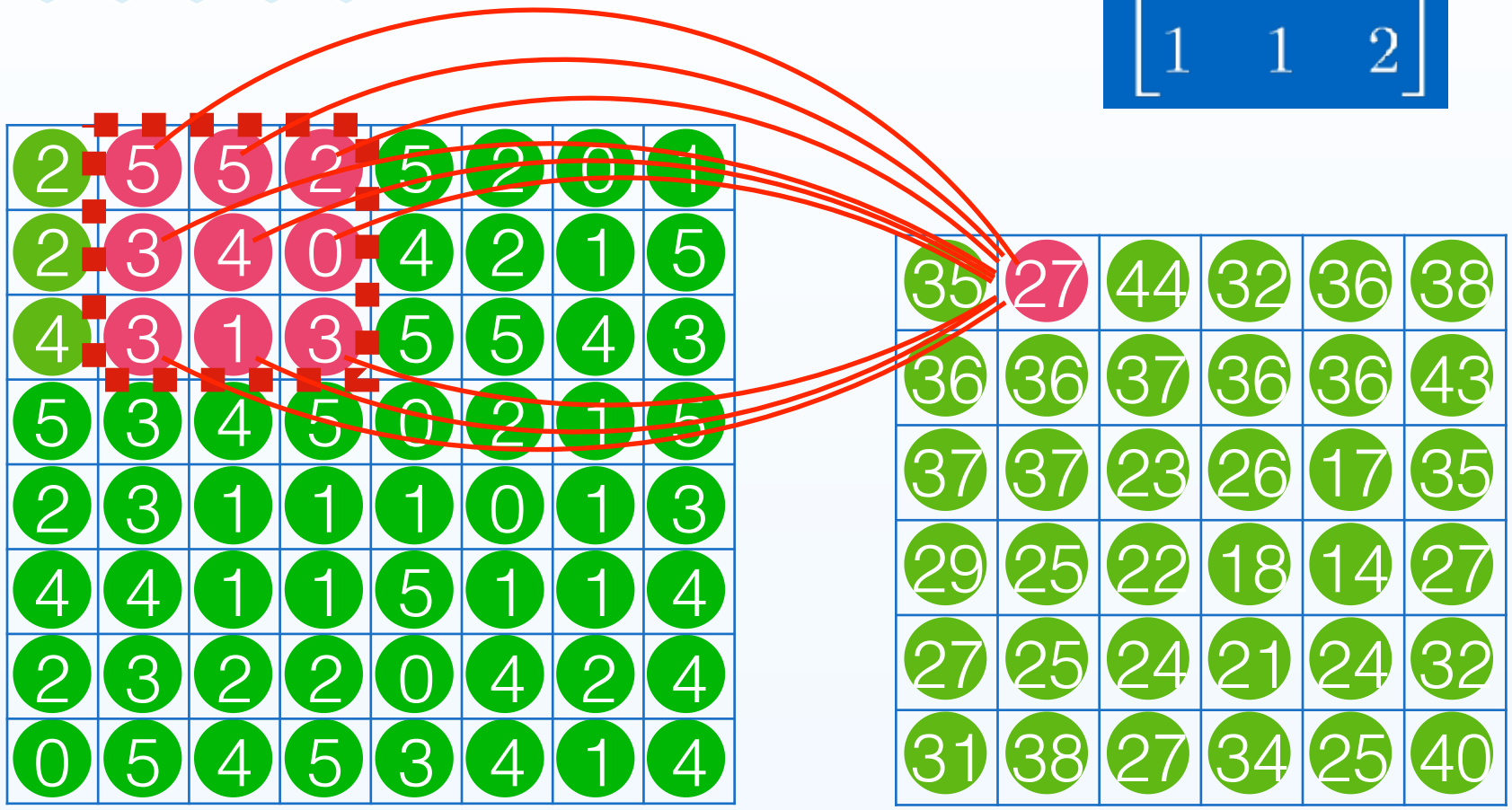
$$W = \begin{bmatrix} 1 & 0 & 1 \\ 2 & 1 & 3 \\ 1 & 1 & 2 \end{bmatrix}$$

兩層中並沒有完全相連

Conv 連結

filter

$$W = \begin{bmatrix} 1 & 0 & 1 \\ 2 & 1 & 3 \\ 1 & 1 & 2 \end{bmatrix}$$



再來 share 同樣的 weights

問題討論

- 最後就是一個 6x6 的矩陣
- 有時我們會把它弄成還是 8x8
- 基本上和原矩陣一樣大
- 而且我們通常 filter 會很多!

35	27	44	32	36	38
36	36	37	36	36	43
37	37	23	26	17	35
29	25	22	18	14	27
27	25	24	21	24	32
31	38	27	34	25	40

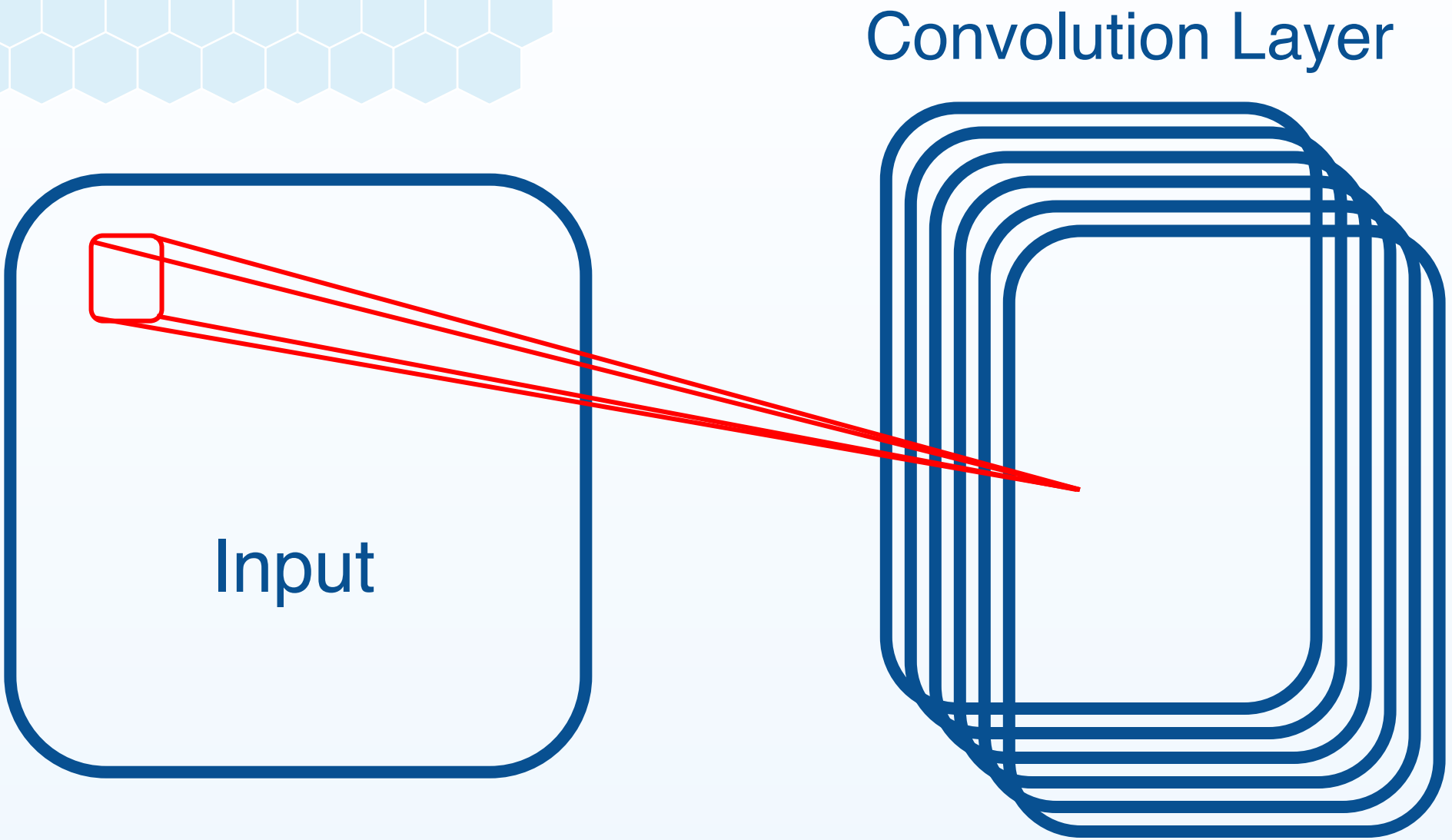
重點

CNN 模型其實基本上在做
特徵提取 及 特徵傳遞

Convolution Layer

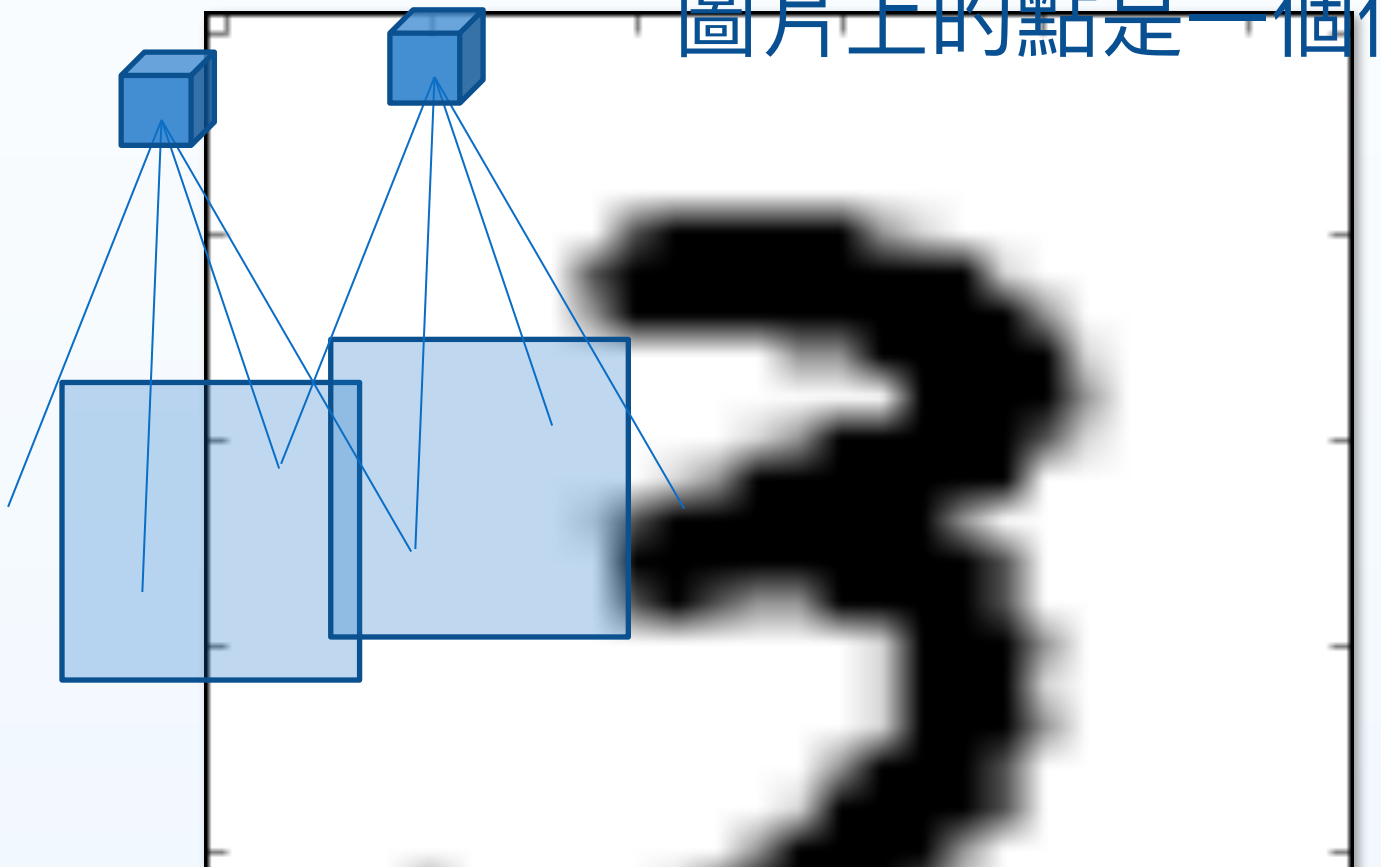
卷積層用來提取輸入值的特徵

Convolution Layer



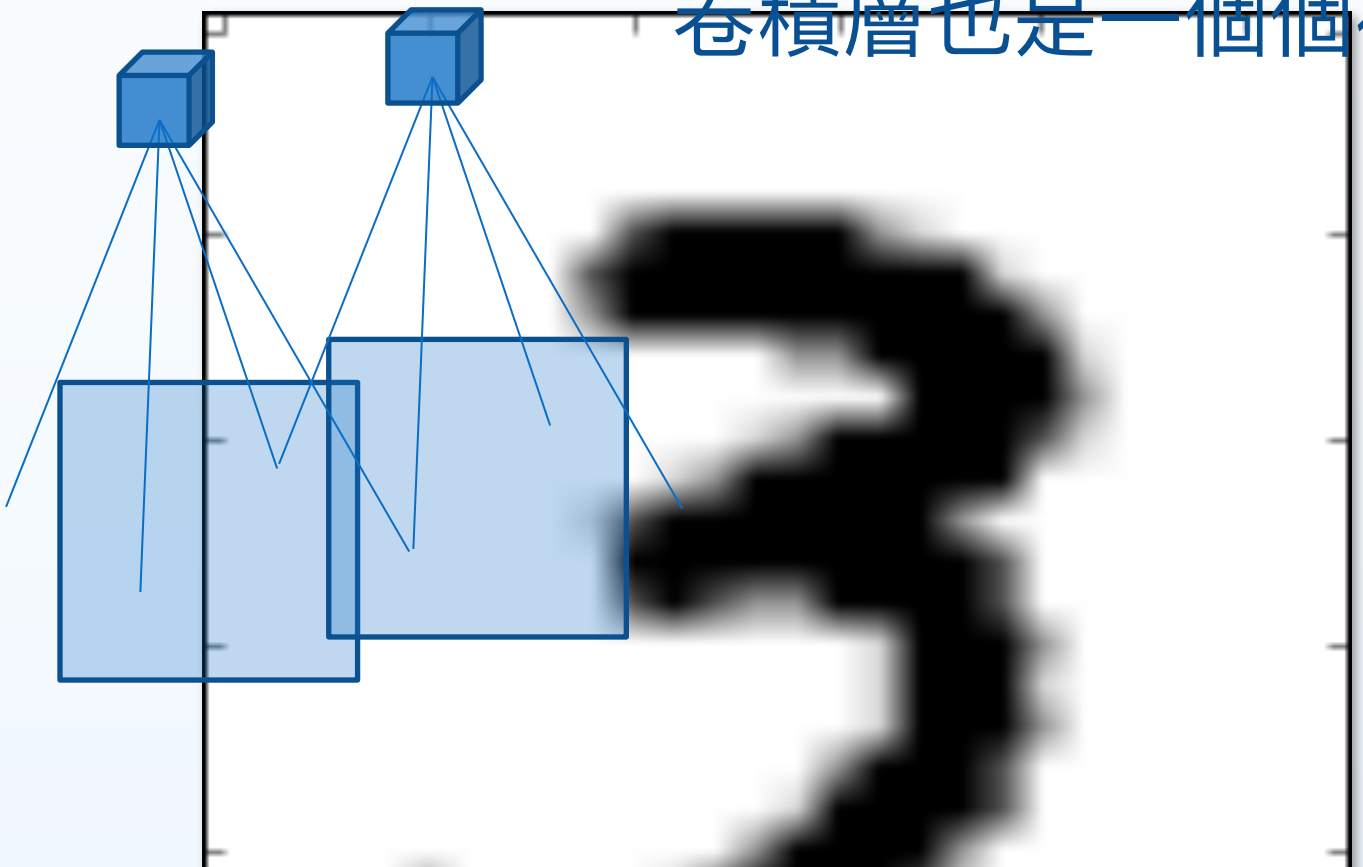
Convolution Layer

圖片上的點是一個個輸入層神經元



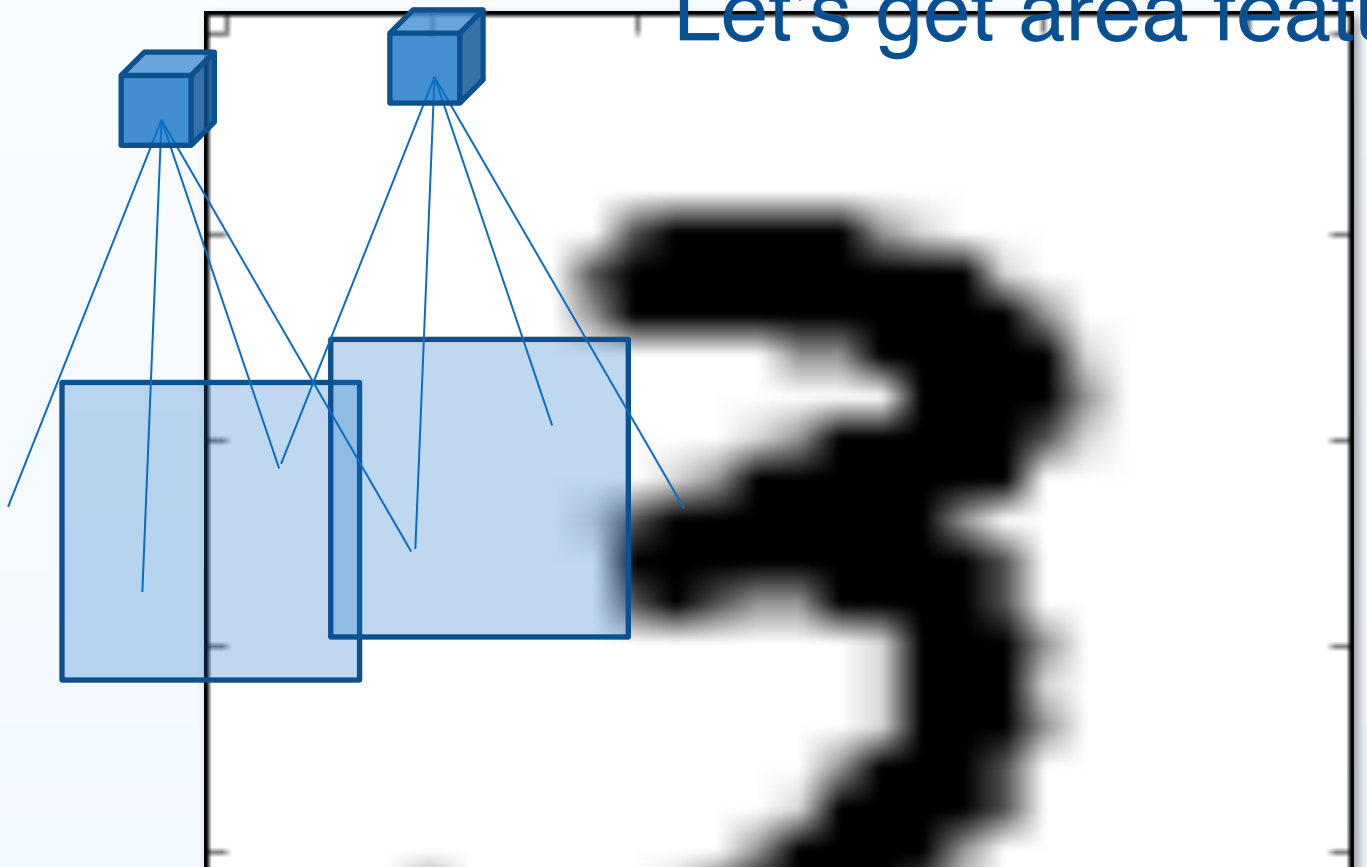
Convolution Layer

卷積層也是一個個神經元

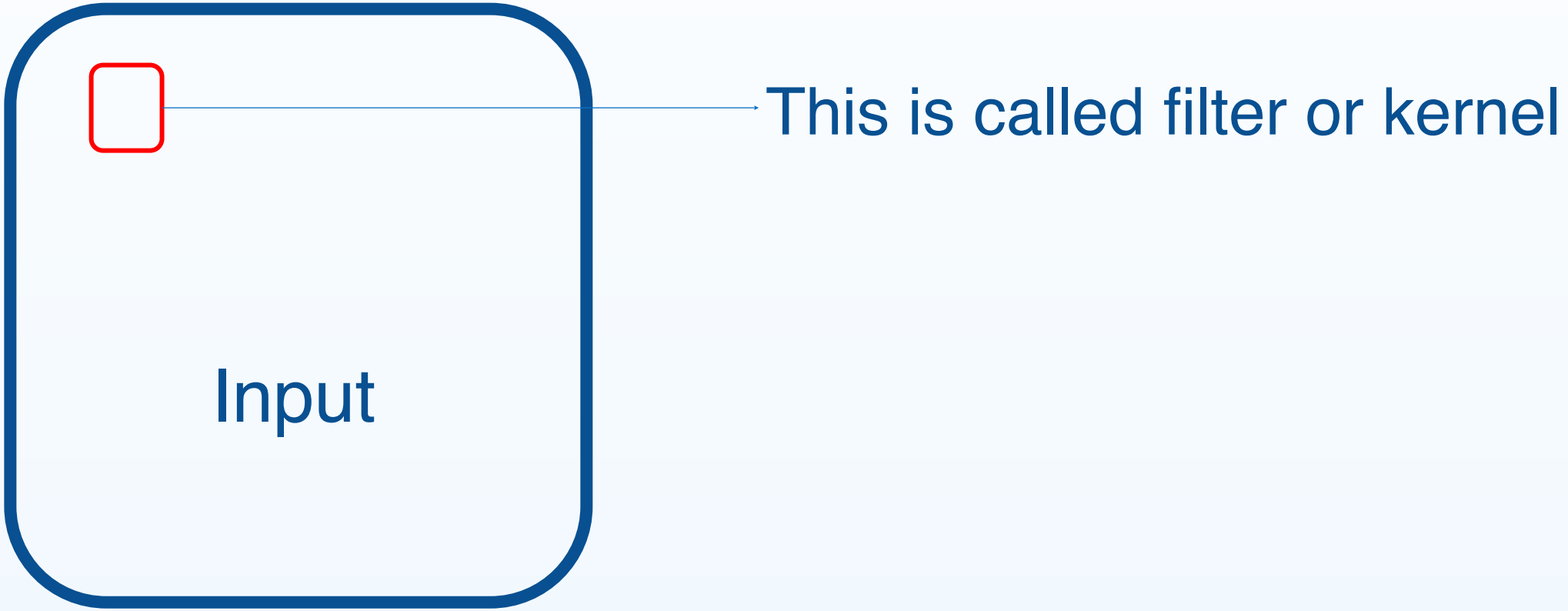


Convolution Layer

Let's get area feature!



Convolution Layer



Convolution Layer

We need choose kernel first !

Convolution Layer

Input = 4 × 4

8	**9**	**6**	**4**
9	4	8	7
1	8	8	1
7	7	4	9

Convolution Layer

卷積核 (Kernel) ＝

1	1
0	1

Convolution Layer

然後是怎麼提取特徵的？

卷積定義

Convolution Definition

$$f * g(x) = \int_{-\infty}^{\infty} f(t)g(x-t)\,dt$$

$f(x)$及$g(x)$為兩個函數

Convolution Layer

你可以換個角度想

其實卷積就很像是你學過的內積

Convolution Layer

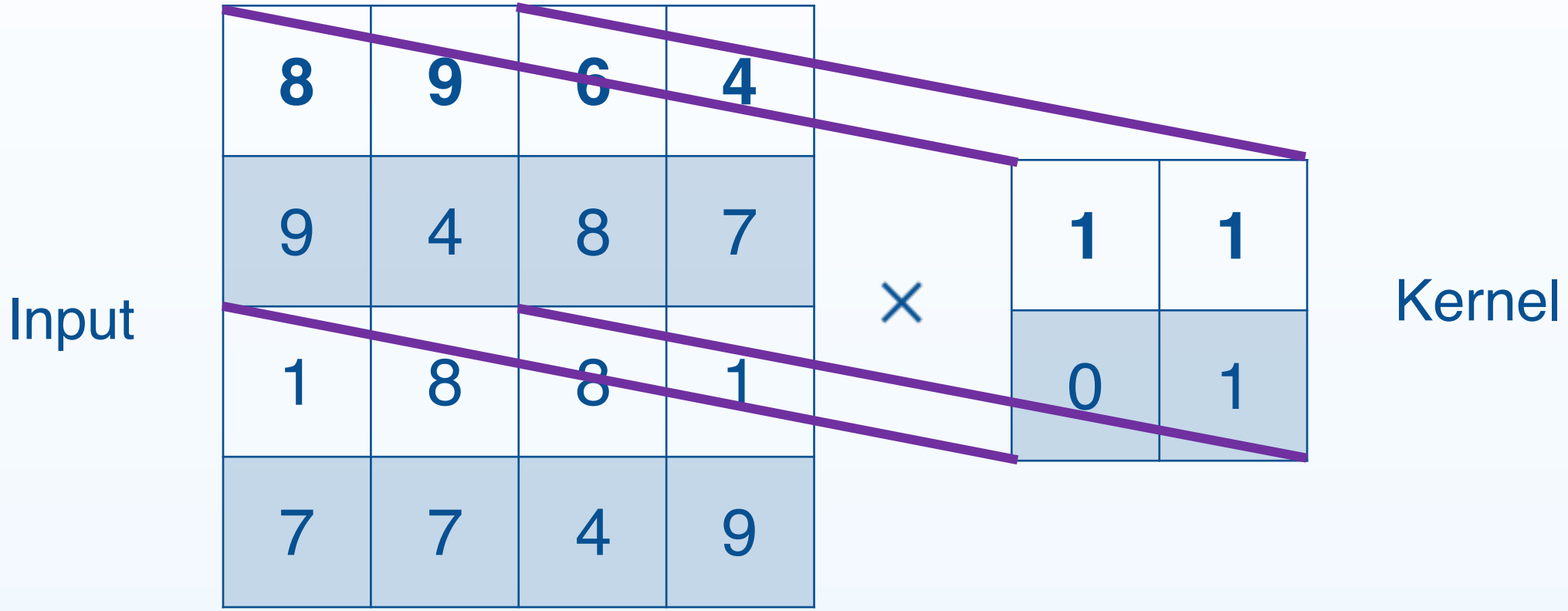
內積 ？ Why ？

我們需要的是向量不是值

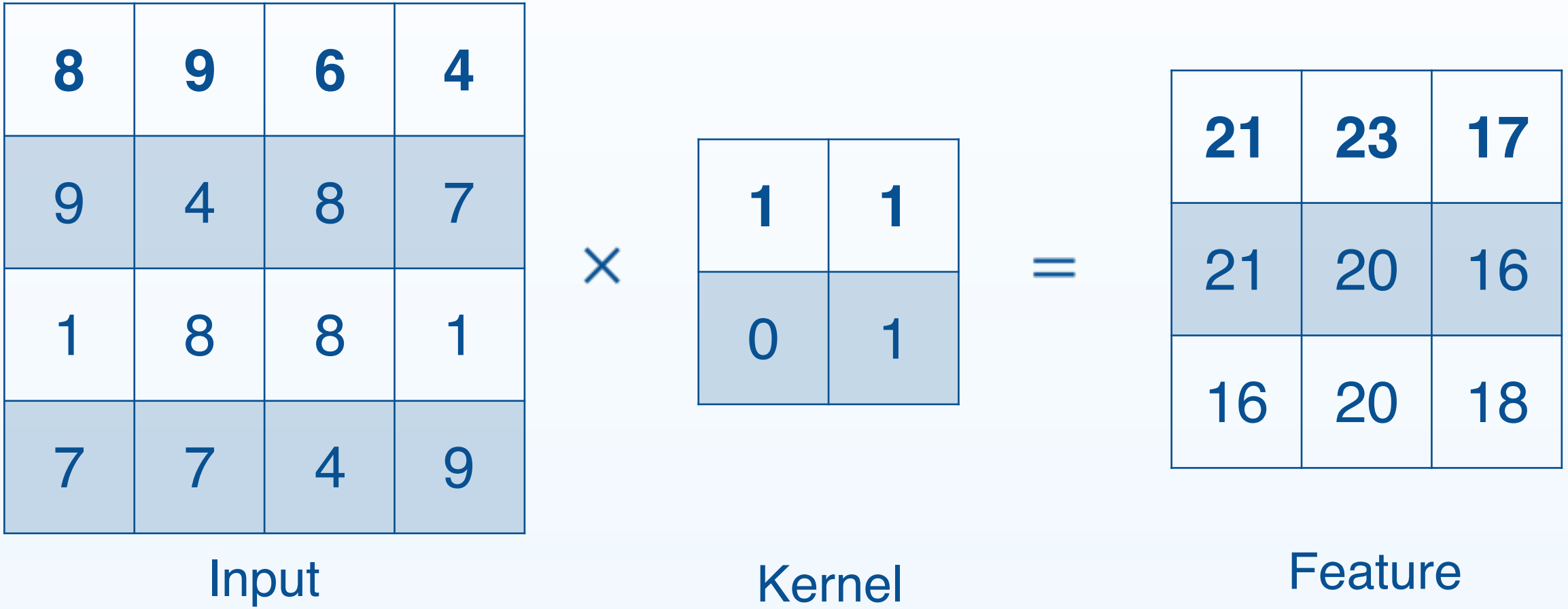
Convolution Layer

使用卷積核(kernel)來解決非向量的問題

Convolution Layer



Convolution Layer

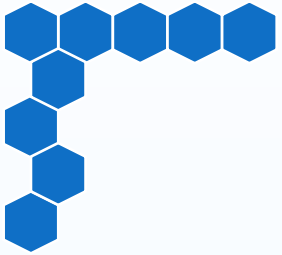


Convolution Layer

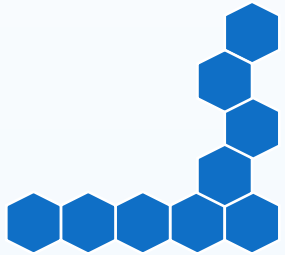
21	**23**	**17**
21	20	16
16	20	18

Feature

- Feature 最後就是一個 3x3 的矩陣
- 有時我們會把它弄成還是4x4 (Panding)



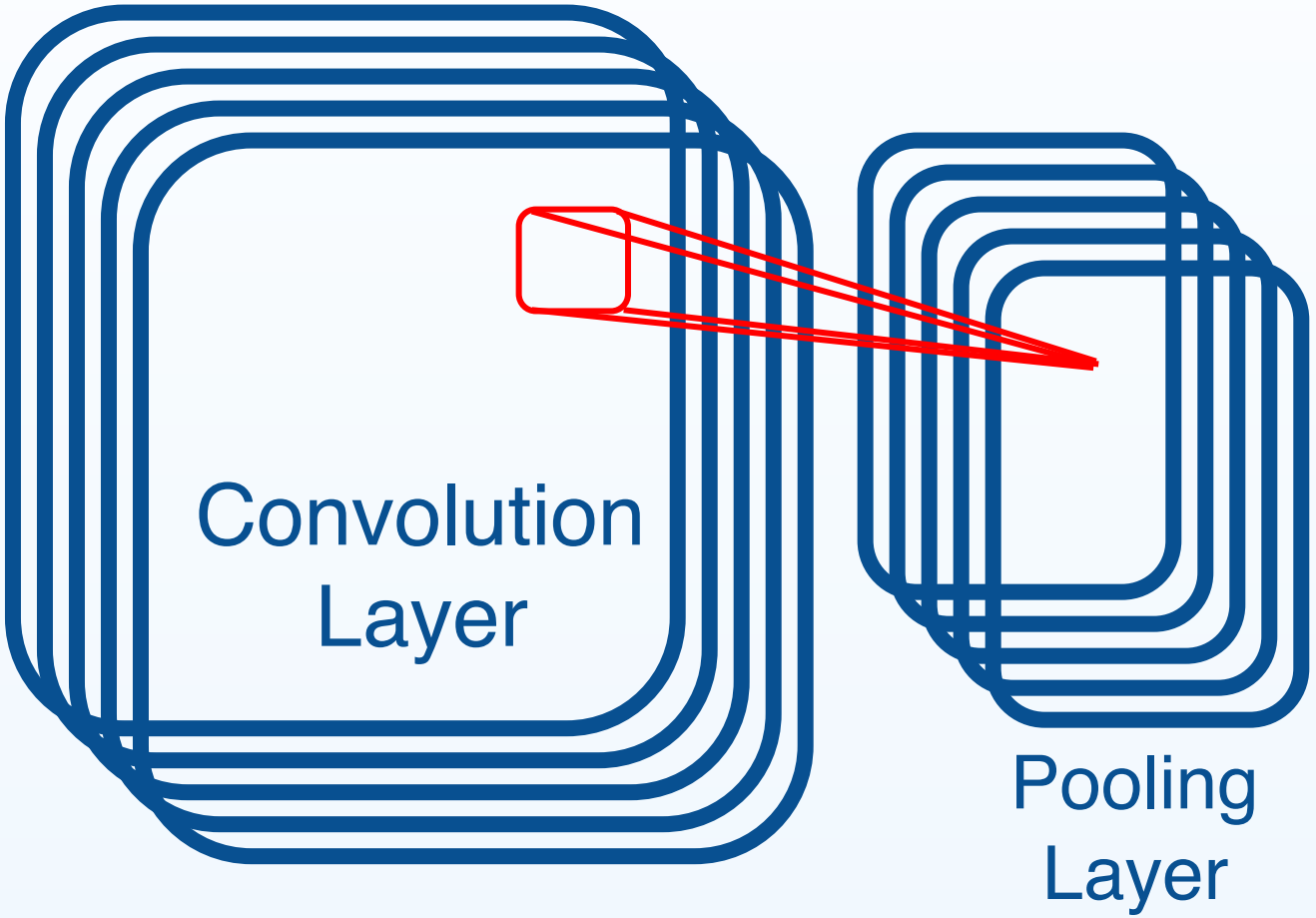
Max-Pooling Layer
(Subsampling)



Pooling Layer

池化層也是用來提取輸入值的特徵

Pooling Layer



Pooling Layer

池化層也是用來提取輸入值的特徵
我們也需要定義我們kernel 的大小

Pooling Layer

差別在於kernel怎麼提取特徵的

Pooling Layer

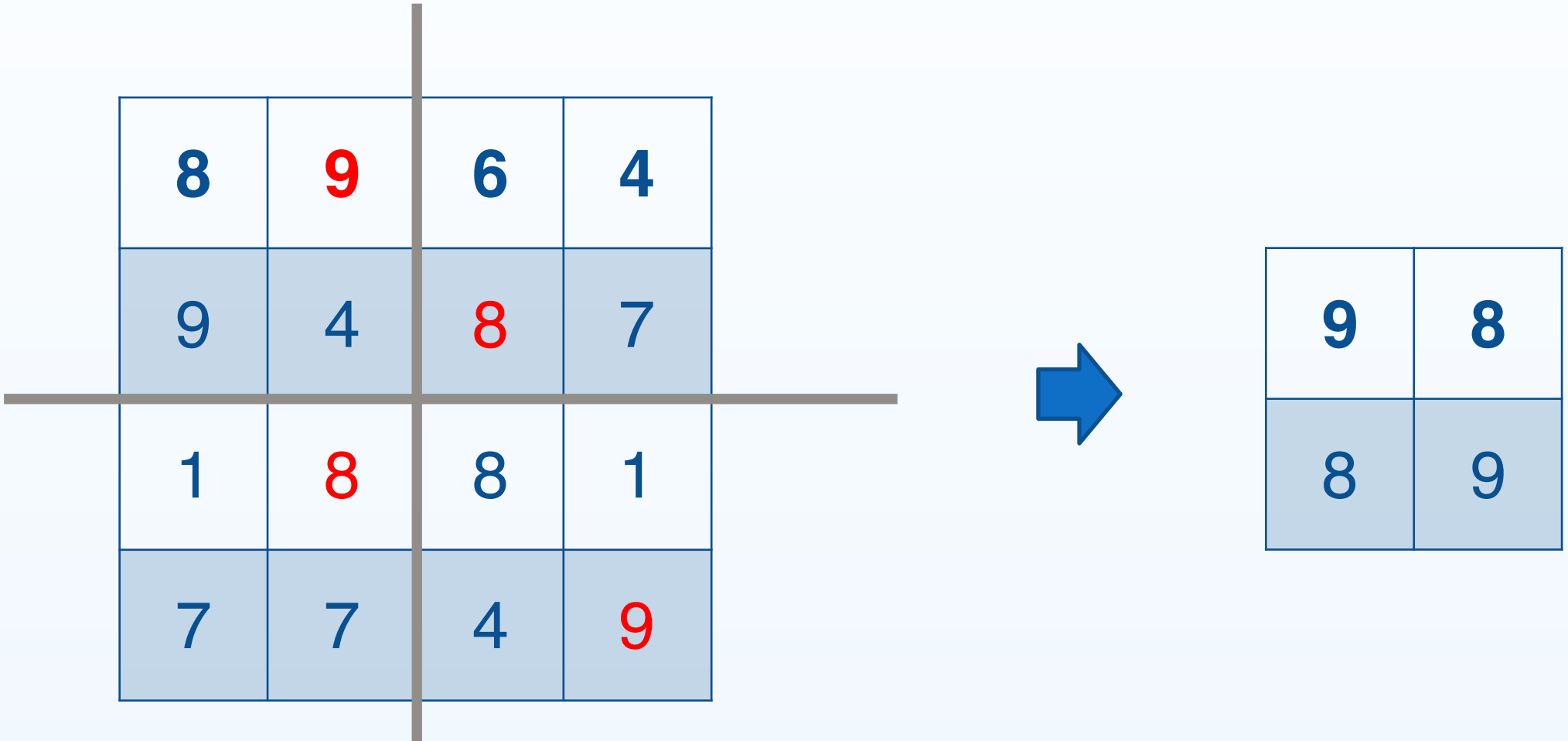
常見的兩種Pooling:

- Max Pooling
- Average Pooling

Pooling Layer

Max Pooling

Pooling Layer



Pooling Layer

Average Pooling

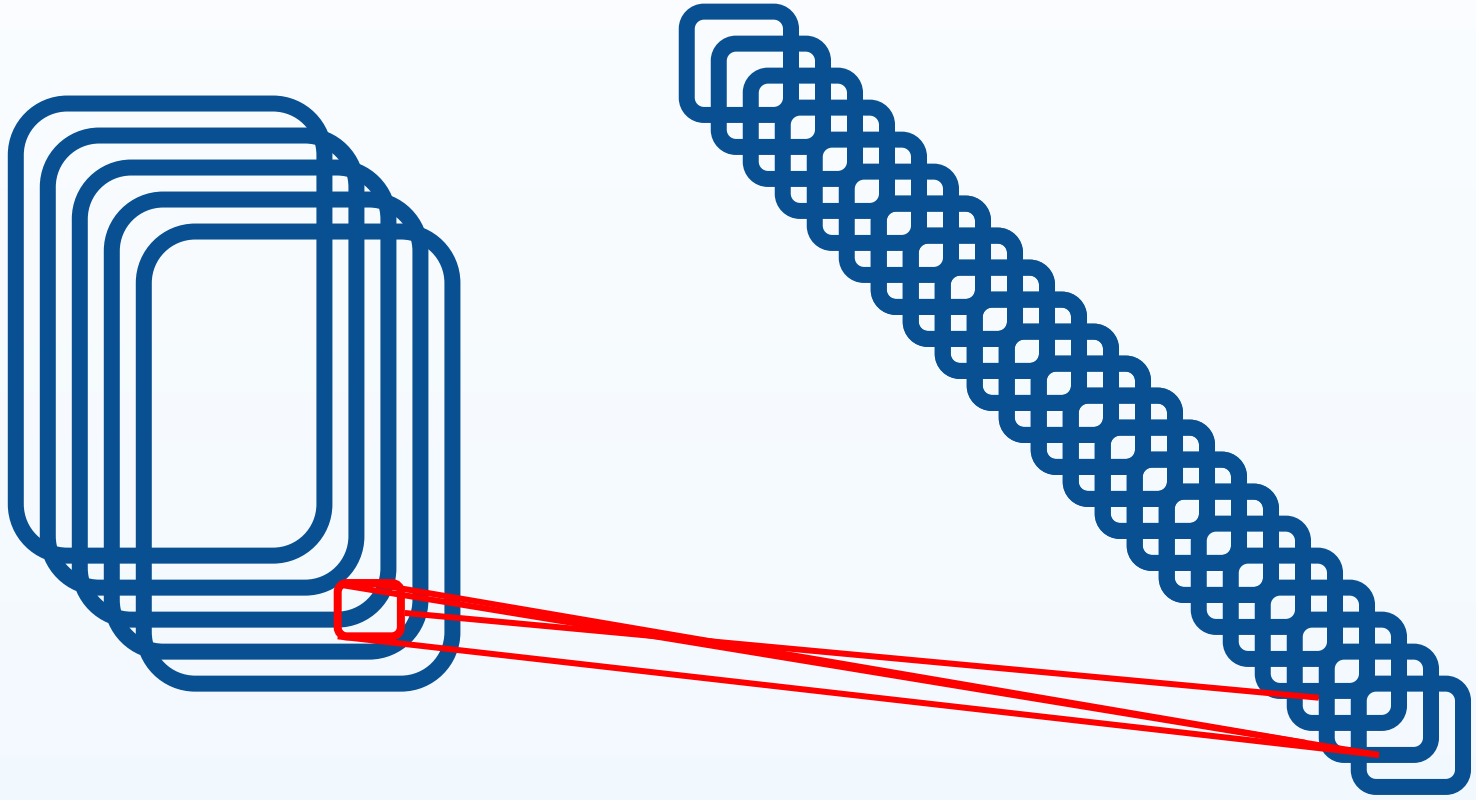
Pooling Layer

8	9	**6**	**4**
9	4	8	7
1	8	8	1
7	7	4	9

$\frac{30}{4}$	$\frac{25}{4}$	$\frac{30}{4}$	$\frac{25}{4}$
$\frac{23}{4}$	$\frac{22}{4}$	$\frac{23}{4}$	$\frac{22}{4}$
$\frac{30}{4}$	$\frac{25}{4}$	$\frac{30}{4}$	$\frac{25}{4}$
$\frac{23}{4}$	$\frac{22}{4}$	$\frac{23}{4}$	$\frac{22}{4}$

Pooling Layer

Pooling
Layer



Fully Connected Layer
(Dense Layer)

Fully Connected Layer

到目前為止，我們將複雜的圖片
汲取到最後並抽取出特徵

Fully Connected Layer

也就是此時的資料不為一維資料

Fully Connected Layer

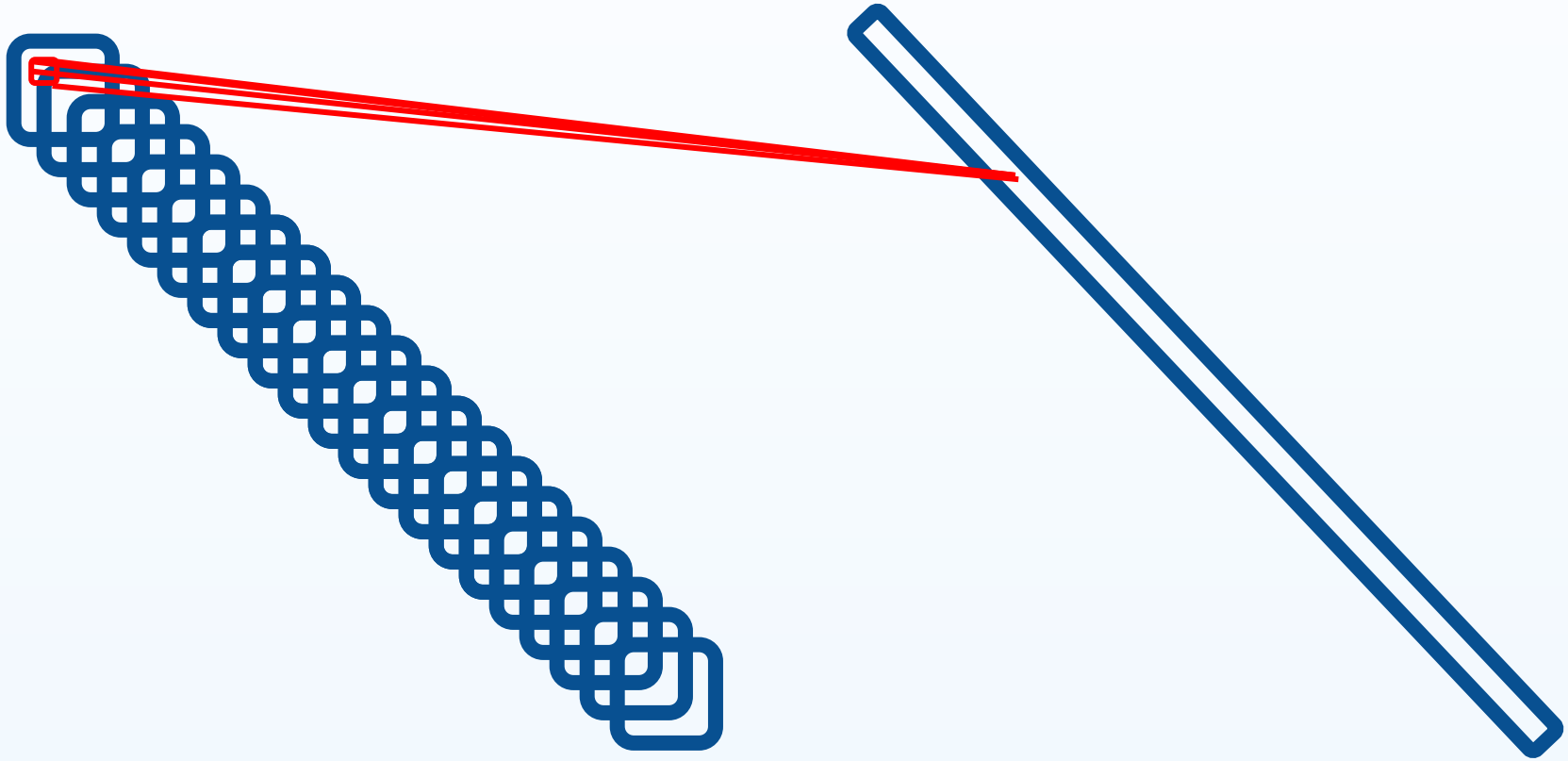
但之前的DNN不是說只吃一維資料 ？

Fully Connected Layer

所以我們通常配合使用Flatten函數將
提取之特徵向量壓平為一維之向量
在把它丟到Dense Layer裡面

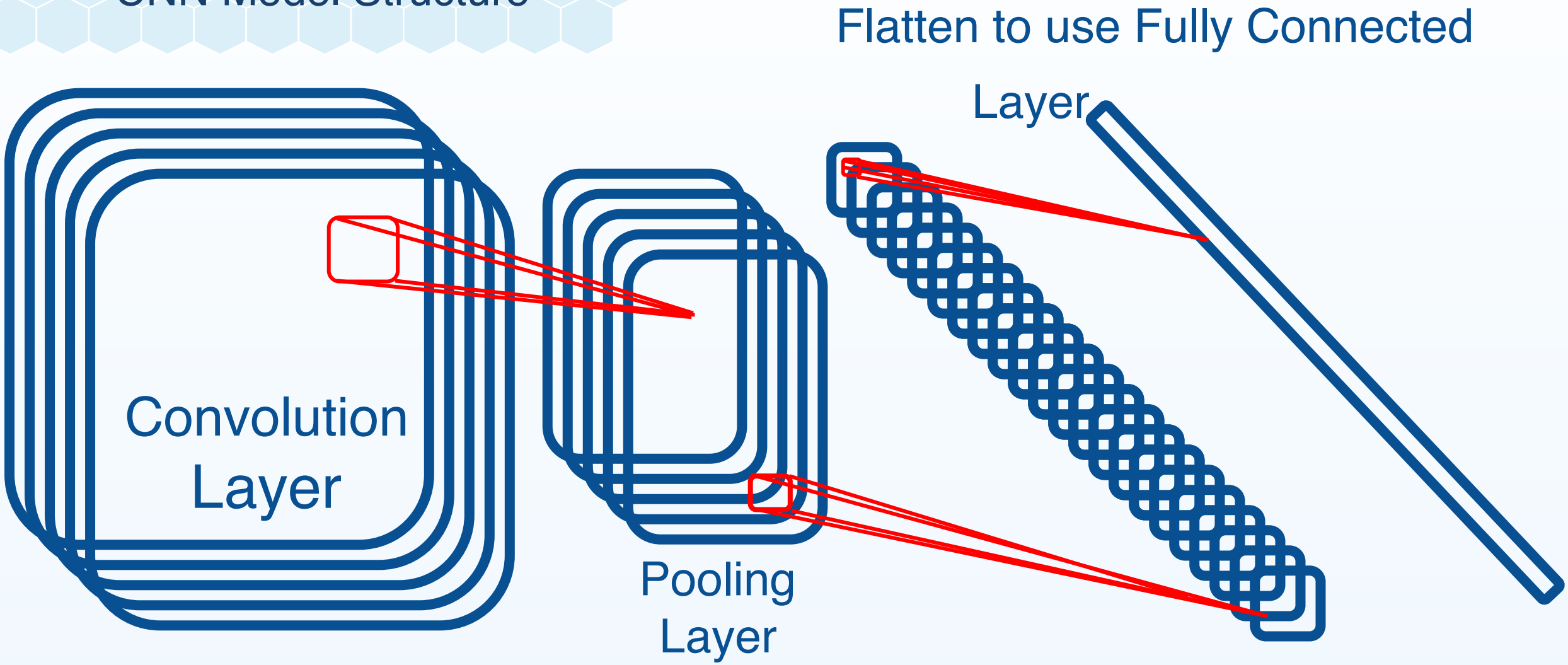
Fully Connected Layer

Flatten to use Fully Connected Layer



CNN架構

CNN Model Structure



CNN架構

CNN Model Structure

通常會重複好幾次

Convolution, Pool, Convolution, Pool, …

所以我們可以來看看當年 LeCun 的模型

LeCun 的 LeNet-5模型

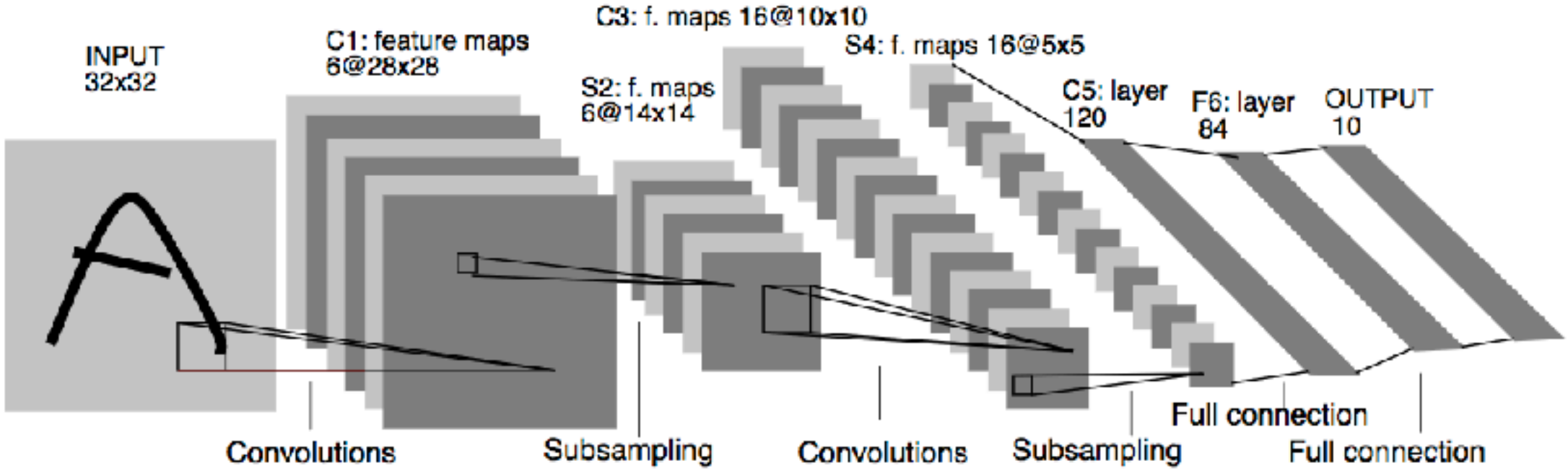


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Src:http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf

First CNN for ETF

第一個ETF CNN

表格標題

Caption

Layer (type)	Output Shape	Parameter #
conv2d_1 (Conv2D)	(None, 28, 28, 10)	100
flatten_1 (Flatten)	(None, 7840)	0
dense_1 (Dense)	(None, 10)	78410

First CNN for ETF

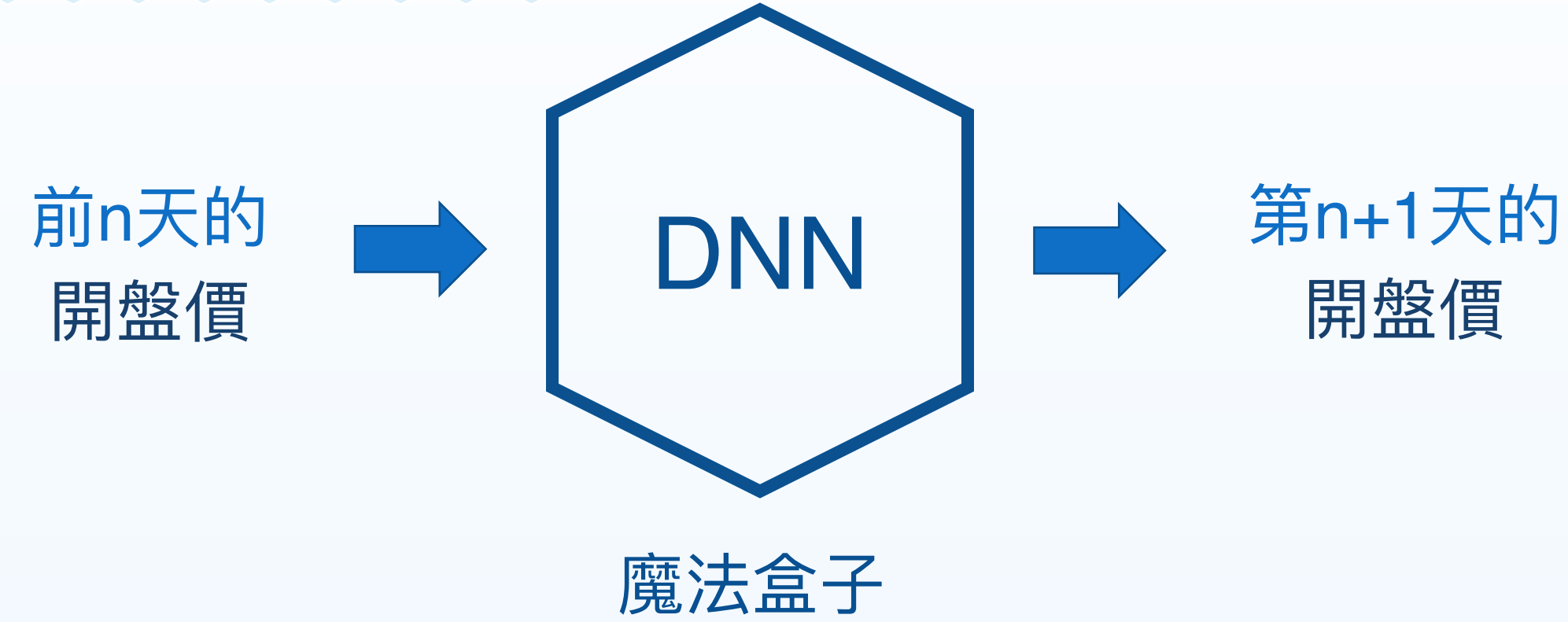
第一個ETF CNN

回憶一下
ETF每天都有下面這些資料

開盤價　收盤價　最高價　最低價　交易量　調整收盤價

大家應該還記得上次ETF在DNN的時候的做法

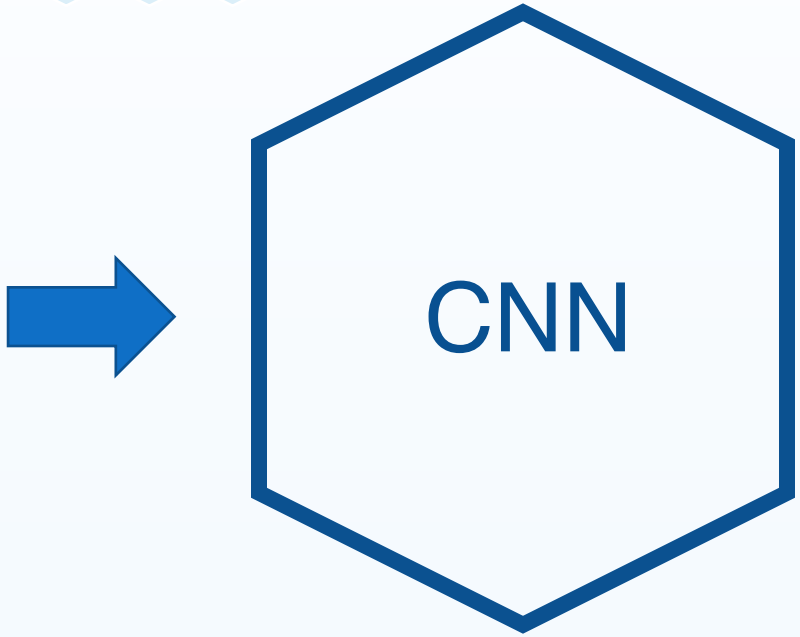
回憶DNN



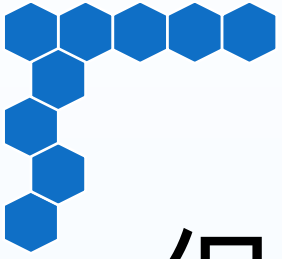
那這次我們要怎麼用？

ＣＮＮ模型

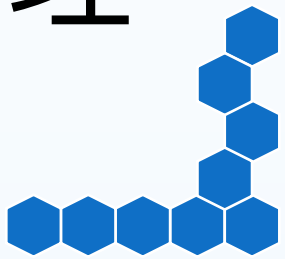
前五天的
開盤價
最高價
最低價
交易量
收盤價



第六天的
調整收盤價
漲幅



但我們這次要再多做一些前處理



這是任意一筆資料

Date	Open	High	Low	Close	Volume	Adj Close
2017-02-15	21.620001	21.870001	21.520000	21.820000	11500	21.820000
2017-02-14	21.930000	21.950001	21.469999	21.629999	15000	21.629999
2017-02-13	21.760000	21.760000	21.510000	21.670000	9800	21.670000

做哪些處理？

有些可能會出現空值 將那一天的資料先予以刪除

為了增加讀入的速度 保持資料順序，並轉換成 numpy array

收盤價的資料漲跌簡化 將資料漲跌的部分改用 0 與 1 標記

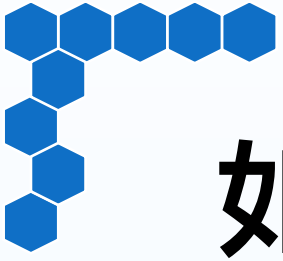
設定每次訓練時有多少天資料

首先先讀入套件跟資料

```
%matplotlib inline # 讓jupyter notebook可以出現你畫的圖片
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from keras.models import Sequential # 讀入Sequential模式以建立模型
from keras.layers import Dense, Flatten, Reshape, Dropout,
Activation
from keras.layers import Conv2D, MaxPooling2D # 這次我們CNN要特別用
的
```

接著讀入ETF資料

```
data = pd.read_csv('./etf_data/AFK.csv')
```



如果你有看過資料會發現有些欄位有0值，而這個應該要去除的

刪除空值

也就是可能會有空值

	Date	Open	High	Low	Close	Volume	Adj Close
103	2016-09-19	20.469999	20.500000	20.030001	20.030001	2800	19.493157
104	2016-09-16	20.110001	20.110001	20.110001	20.110001	300	19.571012
105	2016-09-15	20.020000	20.020000	20.020000	20.020000	0	19.483424

刪除空值

```
# 保留那些正常的欄位
data = data.loc[(data["Volume"]!=0)]
# 直接 reset index, 丟掉舊的, 直接換過去
data.reset_index(drop=True, inplace=True)
```

刪除空值

然後會變成這樣

	Date	Open	High	Low	Close	Volume	Adj Close
103	2016-09-19	20.469999	20.500000	20.030001	20.030001	2800	19.493157
104	2016-09-16	20.110001	20.110001	20.110001	20.110001	300	19.571012
105	2016-09-14	19.980000	20.120001	19.889999	20.020000	9600	19.483424

簡化處理

	Date	Open	High	Low	Close	Volume	Adj Close
0	2017-02-15	21.620001	21.870001	21.520000	21.820000	11500	21.820000
1	2017-02-14	21.930000	21.950001	21.469999	21.629999	15000	21.629999
2	2017-02-13	21.760000	21.760000	21.510000	21.670000	9800	21.670000

我們資料已經按照日期排好了，但
多這個時間欄位只是作為丟資料的順序，不作為訓練的特徵

簡化處理

```
# 因為我們欄位是按照時間排好，直接刪除即可
del data["Date"]
# 將資料轉換成 numpy 的 array
data = data.values
pd.DataFrame(data).head()
```

簡化處理

	0	1	2	3	4	5
0	21.620001	21.870001	21.520000	21.820000	11500.0	21.820000
1	21.930000	21.950001	21.469999	21.629999	15000.0	21.629999
2	21.760000	21.760000	21.510000	21.670000	9800.0	21.670000

變成這樣

資料漲跌標記

Date	Open	High	Low	Close	Volume	Adj Close
2017-02-15	21.620001	21.870001	21.520000	21.820000	11500	21.820000
2017-02-14	21.930000	21.950001	21.469999	21.629999	15000	21.629999
2017-02-13	21.760000	21.760000	21.510000	21.670000	9800	21.670000

剛剛時間的順序是這樣，由上而下由新而舊

註：Date欄位已經不存在，只是為了說明先顯示出來

資料漲跌標記

Date	Open	High	Low	Close	Volume	Adj Close
2017-02-13	21.760000	21.760000	21.510000	21.670000	9800	21.670000
2017-02-14	21.930000	21.950001	21.469999	21.629999	15000	21.629999
2017-02-15	21.620001	21.870001	21.520000	21.820000	11500	21.820000

變成這樣，由上而下由舊而新

這是為了讓我們在訓練資料的時候也是從舊的訓練到新的

資料漲跌標記

這個欄位是我們這次的目標

Open	High	Low	Close	Volume	Adj Close
21.620001	21.870001	21.520000	21.820000	11500	21.820000
21.930000	21.950001	21.469999	21.629999	15000	21.629999
21.760000	21.760000	21.510000	21.670000	9800	21.670000

資料漲跌標記

所以讓資料分開來

Open	High	Low	Close	Volume	Adj Close
21.620001	21.870001	21.520000	21.820000	11500	21.820000
21.930000	21.950001	21.469999	21.629999	15000	21.629999
21.760000	21.760000	21.510000	21.670000	9800	21.670000

簡化處理

```
X = data[:, 0:5]
Y = data[:, 5]
print("X的資料是\n", X)
print("Y的資料是\n", Y)
```

資料漲跌標記

最後我們標記他的漲跌

Open	High	Low	Close	Volume	Adj Close
21.760000	21.760000	21.510000	21.670000	9800	21.670000
21.930000	21.950001	21.469999	21.629999	15000	21.629999
21.620001	21.870001	21.520000	21.820000	11500	21.820000

資料漲跌標記

資料漲跌標記

沒得往前比的這天資料我們設為0

變成這樣

Close
0
1
0

資料漲跌標記

```
for i in range(len(Y)):
    if (i+1 < len(Y) and Y[i] - Y[i+1] > 0):
        Y[i] = 1
    else:
        Y[i] = 0
```

每多少天資料丟一次

```
day = 5   # 設定你的天數
Y = Y[day:]
XX = []
for i in range(day, len(X)):
      tmp = []
      for j in range(day-1, -1, -1):
            tmp.append(X[i-j])
      XX.append(tmp)
X = np.array(XX)
```

接著是我們的 CNN 模型

```
model = Sequential()
model.add(Conv2D(64, (3, 3), padding='same', activation='relu', input_shape=(5,5,1)))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D((2,2), strides=(2,2), padding='same'))


model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D((2,2), strides=(2,2), padding='same'))

model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D((2,2), strides=(2,2), padding='same'))

model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D((2,2), strides=(2,2), padding='same'))

model.add(Flatten())
model.add(Dense(40))
model.add(Activation('relu'))
model.add(Dropout(0.5))

model.add(Dense(1))
model.add(Activation('sigmoid'))
```

CNN model

Convolution1

Layer (type)	Output Shape	Parameter #
conv2d_1 (Conv2D)	(None, 5, 5, 64)	640
conv2d_2 (Conv2D)	(None, 5, 5, 64)	36928
conv2d_3 (Conv2D)	(None, 5, 5, 64)	36928

CNN model

Pool1

Layer (type)	Output Shape	Parameter #
max_pooling2d_1 (MaxPooling2)	(None, 3, 3, 64)	0

CNN model

Convolution2

Layer (type)	Output Shape	Parameter #
conv2d_4 (Conv2D)	(None, 3, 3, 128)	73856
conv2d_5 (Conv2D)	(None, 3, 3, 128)	147584
conv2d_6 (Conv2D)	(None, 3, 3, 128)	147584

CNN model

Pool2

Layer (type)	Output Shape	Parameter #
max_pooling2d_2 (MaxPooling2)	(None, 2, 2, 128)	0

CNN model

Convolution3

Layer (type)	Output Shape	Parameter #
conv2d_7 (Conv2D)	(None, 2, 2, 256)	295168
conv2d_8 (Conv2D)	(None, 2, 2, 256)	590080
conv2d_9 (Conv2D)	(None, 2, 2, 256)	590080

CNN model

Pool3

Layer (type)	Output Shape	Parameter #
max_pooling2d_3 (MaxPooling2)	(None, 1, 1, 256)	0

CNN model

Convolution4

Layer (type)	Output Shape	Parameter #
conv2d_10 (Conv2D)	(None, 1, 1, 512)	1180160
conv2d_11 (Conv2D)	(None, 1, 1, 512)	2359808
conv2d_12 (Conv2D)	(None, 1, 1, 512)	2359808

CNN model

Pool4

Layer (type)	Output Shape	Parameter #
max_pooling2d_4 (MaxPooling2)	(None, 1, 1, 512)	0

構建模型

序列式模型

```
model = Sequential()
```

構建模型

序列式模型

第一組Conv層及Pool層

激發函數用Relu函數

```
model = Sequential()
# Block1
model.add(Conv2D(64, (3, 3),
padding='same', activation='relu',
input_shape=(5,5,1)))
model.add(Conv2D(64, (3, 3),
activation='relu', padding='same'))
model.add(Conv2D(64, (3, 3),
activation='relu', padding='same'))
model.add(MaxPooling2D((2,2),
strides=(2,2), padding='same'))
```

構建模型

序列式模型

第二組Conv層及Pool層

激發函數用Relu函數

```
# Block2
model.add(Conv2D(128, (3, 3),
activation='relu', padding='same'))
model.add(Conv2D(128, (3, 3),
activation='relu', padding='same'))
model.add(Conv2D(128, (3, 3),
activation='relu', padding='same'))
model.add(MaxPooling2D((2,2),
strides=(2,2), padding='same'))
```

構建模型

序列式模型

第三組Conv層及Pool層

激發函數用Relu函數

```
# Block3
model.add(Conv2D(256, (3, 3),
activation='relu', padding='same'))
model.add(Conv2D(256, (3, 3),
activation='relu', padding='same'))
model.add(Conv2D(256, (3, 3),
activation='relu', padding='same'))
model.add(MaxPooling2D((2,2),
strides=(2,2), padding='same'))
```

構建模型

序列式模型

第四組Conv層及Pool層

激發函數用Relu函數

```
# Block3
model.add(Conv2D(512, (3, 3),
activation='relu', padding='same'))
model.add(Conv2D(512, (3, 3),
activation='relu', padding='same'))
model.add(Conv2D(512, (3, 3),
activation='relu', padding='same'))
model.add(MaxPooling2D((2,2),
strides=(2,2), padding='same'))
```

構建模型

序列式模型

- 將我們的資料壓平回一維
- 40個神經元的全連接層
- 1個神經元的全連接層
- 激發函數用sigmoid

```
model.add(Flatten())
model.add(Dense(40))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation('sigmoid'))

model.summary() #檢查模型形狀
```

編譯模型

```
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
```

決定損失函數和優化器
並且新增準確度作為測量記錄

訓練模型

```
model.fit(X_train, Y_train, epochs=4, batch_size=128)
```

決定批次大小以及訓練次數

試用結果

```
score = model.evaluate(X_test, Y_test, verbose=0)
print("Total Loss on Testing Set : ", score[0]) # 越低越好
print("Accuracy of Testing Set : ", score[1]) # 越高越好
```

可能更好的模型

前n天的
前n天的
前n天的
前n天的
前n天的
開盤價
最高價
最低價
交易量
收盤價

第 n+1 天的
調整收盤價
漲幅

很多份ETF