Convolutional Neural Network (CNN)

進行圖像處理時，我們往往需要將其拉成 vector 當作 input 進行分析，然而即便是僅有 100*100 pixel 的彩色圖片，也需要 100*100*3 dims.的 vector 才能表示這張圖片，因此直接用一般的 fully connected 的 feedforward network 來做圖像處理的時候，往往會需要太多的參數。這種時候會傾向使用 CNN 來解決該問題。

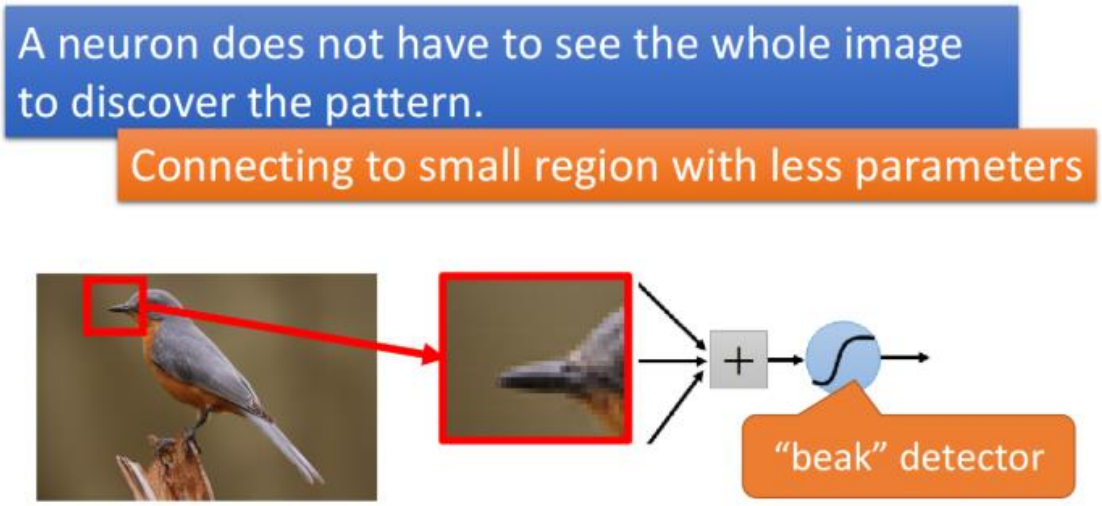
Three Property for CNN

並非所有參數過多的 DNN 問題都能用 CNN 簡化，僅有在處理的問題具有圖像的特性時，CNN 才能成立。主要有三項假設。

一、Some Patterns Are Much Smaller Than the Whole Image

在圖像處理的問題中，第一層 hidden layer 的 neuron 主要的工作是偵測有沒有一種 pattern（圖案樣式）出現該 image 中。然而大部分的 pattern 其實是比整張 image 要小的，因此對一個 neuron 來說，其實並不需要看整張 image 才能偵測有沒有某一個 pattern 出現，只需要看這張 image 的一小部分即可。

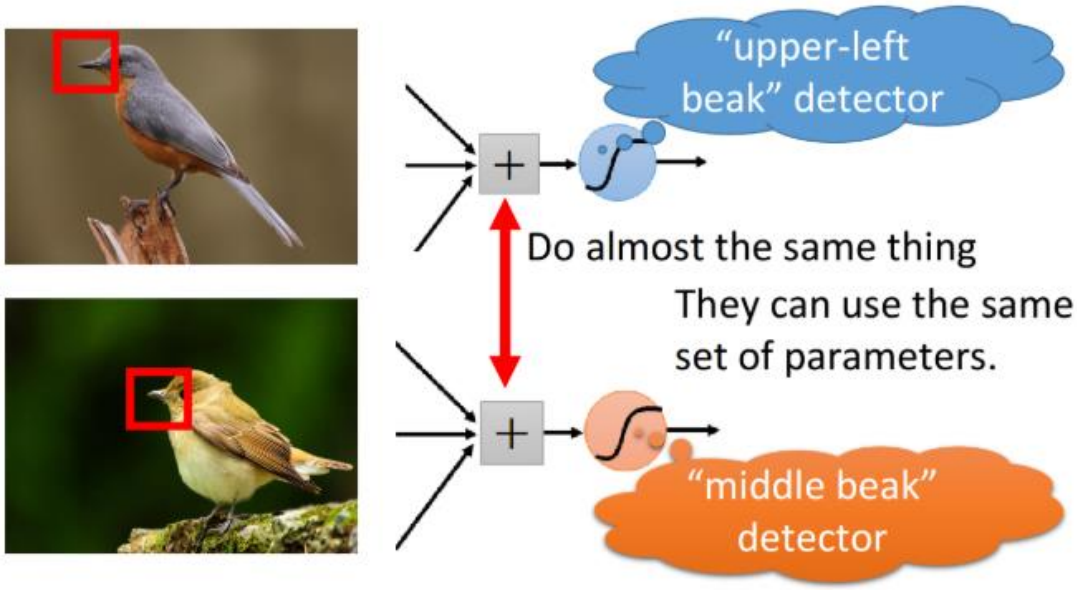
例如要確認一張 image 中的生物是否為鳥類，第一層的 hidden layer 可能就會去找有沒有「鳥喙」出現，然而「鳥喙」只會出現在鳥的頭部，因此沒有必要看完整張 image，只需要限定在該生物的頭部即可。



二、The Same Patterns Appear in Different Regions

同樣的 pattern，可能會出現在 image 的不同地方，但是它們有同樣的形狀且代表的是同樣的含義，因此它們也可以用同樣的 neuron、同樣的參數，被同一個 detector 檢測出來。所以我們可以要求這些功能幾乎一致的 neuron 共用一組參數，它們 share 同一組參數就可以幫助減少總參數的量。

舉例來說，我們已經知道找到「鳥喙」就能代表有一隻鳥在 image 中，無關他所在的位置，因此並不需要訓練兩個不同的 detector 去分別偵測 image 的各個地方有沒有「鳥喙」這件事情。



三、Subsampling the Pixels Will Not Change the Object

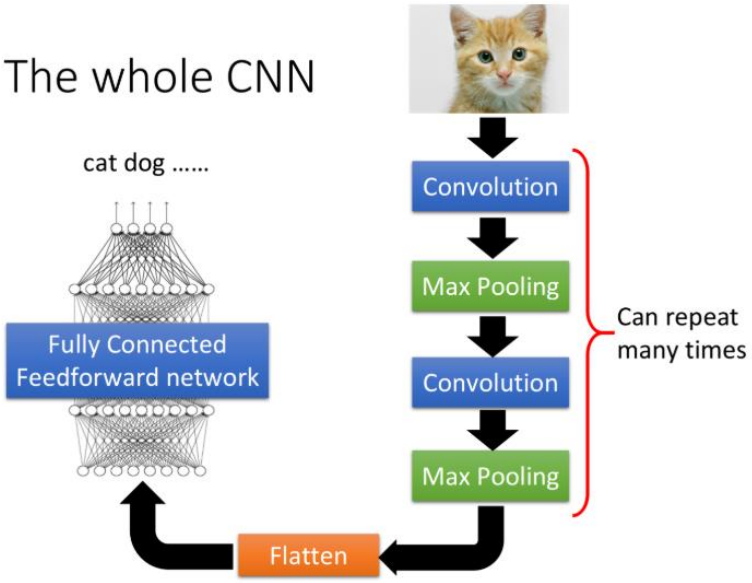
假設對一張 image 做 subsampling，例如把它奇數行與偶數列的 pixel 拿掉，image 就可以變成原來的十分之一大小，而且並不會影響人對這張 image 的理解。所以我們可以利用 subsampling 這個概念把 image 變小，從而減少需要用到的參數量。

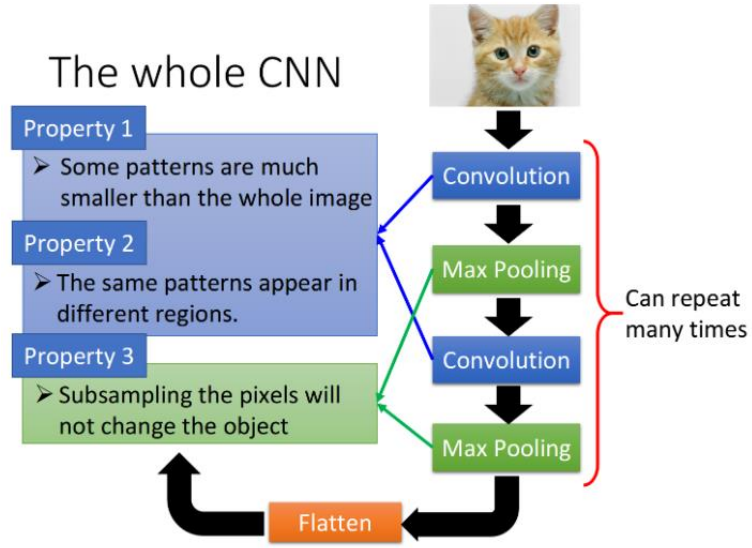
We can subsample the pixels to make image smaller

→ Less parameters for the network to process the image

The Whole CNN Structure

綜上所述，我們可以利用這些特性建立出 CNN 的結構藉此減少參數量。
首先 input 一張 image 以後，它會先通過 Convolution 的 layer，接下來再做 Max Pooling 這件事，上述的兩層 layer 可以疊加多次，最後透過 Flatten layer 拉成 vector 後再丟到一般的 Fully connected network 裡面。



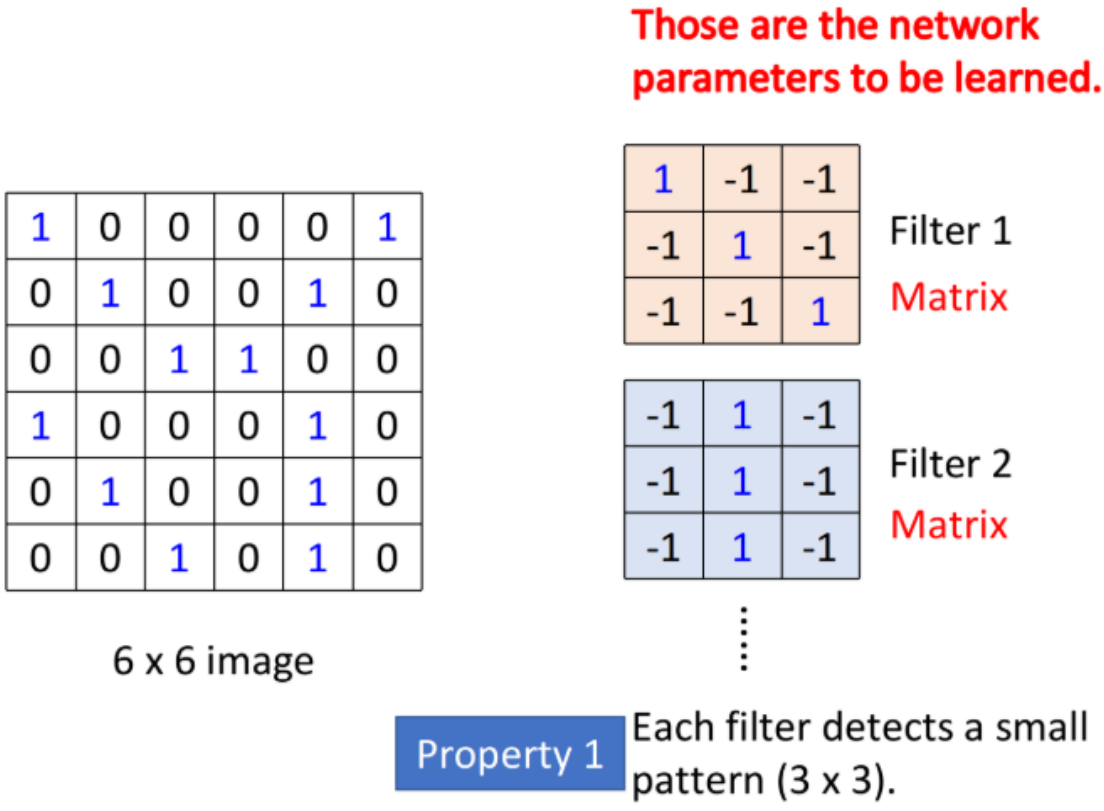


一、Convolution Layer

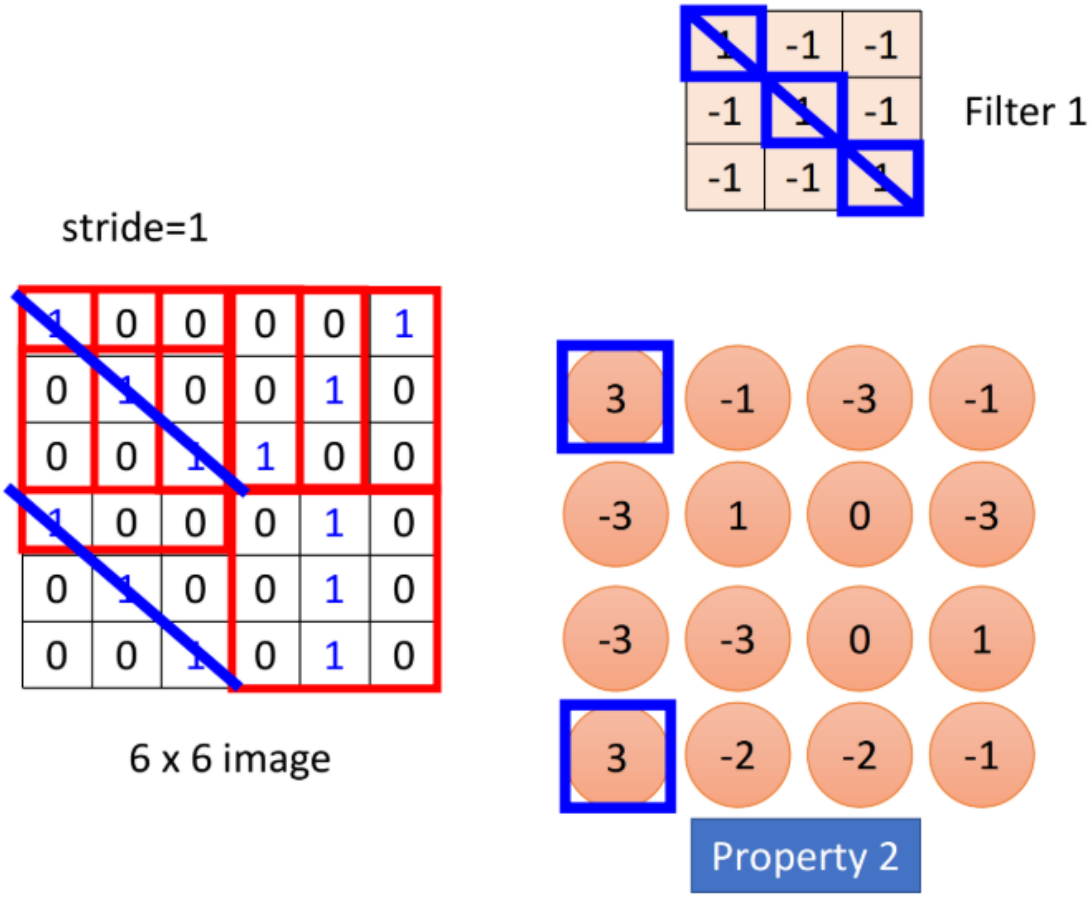
該層 layer 利用了前述的第一與第二個特性，意即從 image 中找出特定的 pattern，並對偵測同一個 pattern 但偵測範圍不同的 neuron 共用相同的參數。

假設現在 input 是一張 6*6 的黑白圖片，因此每個 pixel 只需要用一個 value 來表示，而在 convolution layer 裡面，有一堆 filter matrix，裡面每一個 element 的值就是 network 的 parameter，它們的值都是通過 training data 學出來的；而這邊的每一個 filter，其實就等同於是 Fully connected layer 裡的一個 neuron。

該例中每一個 filter 的 dim.為 3*3，意味著它就是在偵測一個 3*3 的 pattern，當它偵測的時候，並不會去看整張 image，它只看一個 3*3 範圍內的 pixel，就可以判斷某一個 pattern 有沒有出現，這就考慮了 property 1。

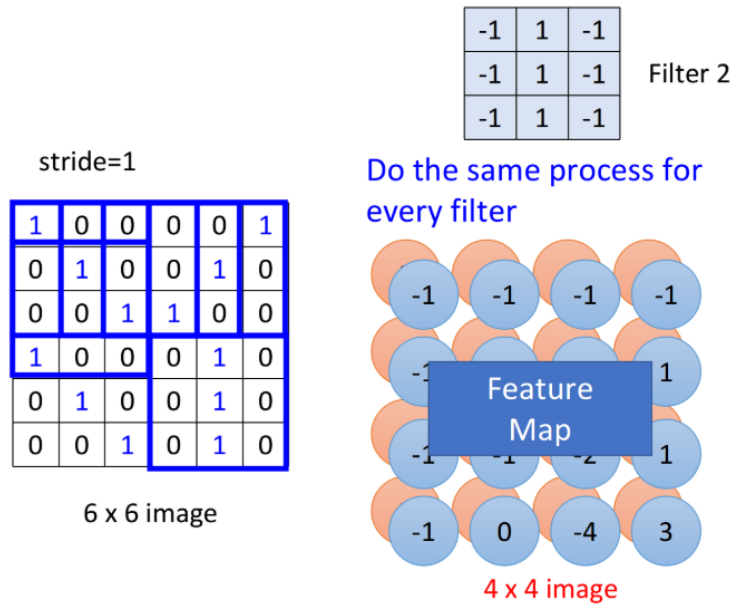


再者，每個 filter 都是從 image 的左上角開始，做一個 slide window，每次向右挪動一定的距離，該距離就叫做 stride（由使用者決定）。
在此例中，每次 filter 停下的時候就跟 image 裡對應的 3*3 的 matrix 做內積，接著再挪動一格 (stride = 1)。直到當它碰到 image 最右邊的時候，就從下一行的最左邊開始重複進行上述操作。
經過一整個 convolution 的 process，最終得到下圖所示的紅色的 4*4 matrix。

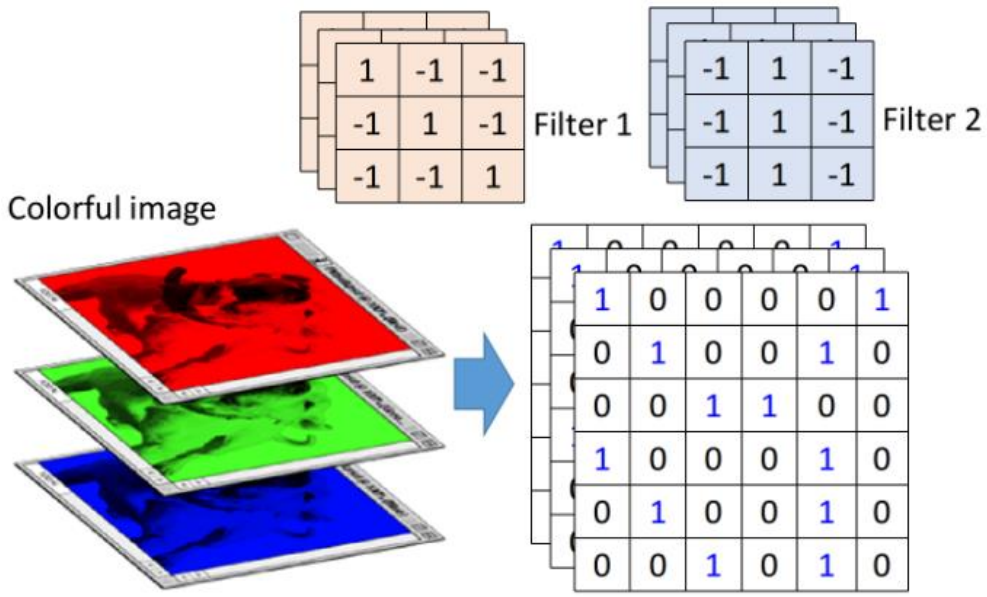


此外，以此例的 filter1 為例，它斜對角的地方是 1,1,1，所以它的工作就是 detect 有沒有連續的從左上角到右下角的 1,1,1 出現在這個 image 裡面，如果有其內積便會呈現最大值（圖中藍色部分）。
而同一個 pattern 出現在 image 左上角的位置和左下角的位置，並不需要用到不同的 filter，我們用 filter1 就可以偵測出來，這就考慮了 property 2。

最終經過數個 filter 處理後得到的多個 4*4 的 matrix 我們將其合在一起，並稱一片 matrix 為一個 channel，合稱該 matrix 的集合為 Feature Map。



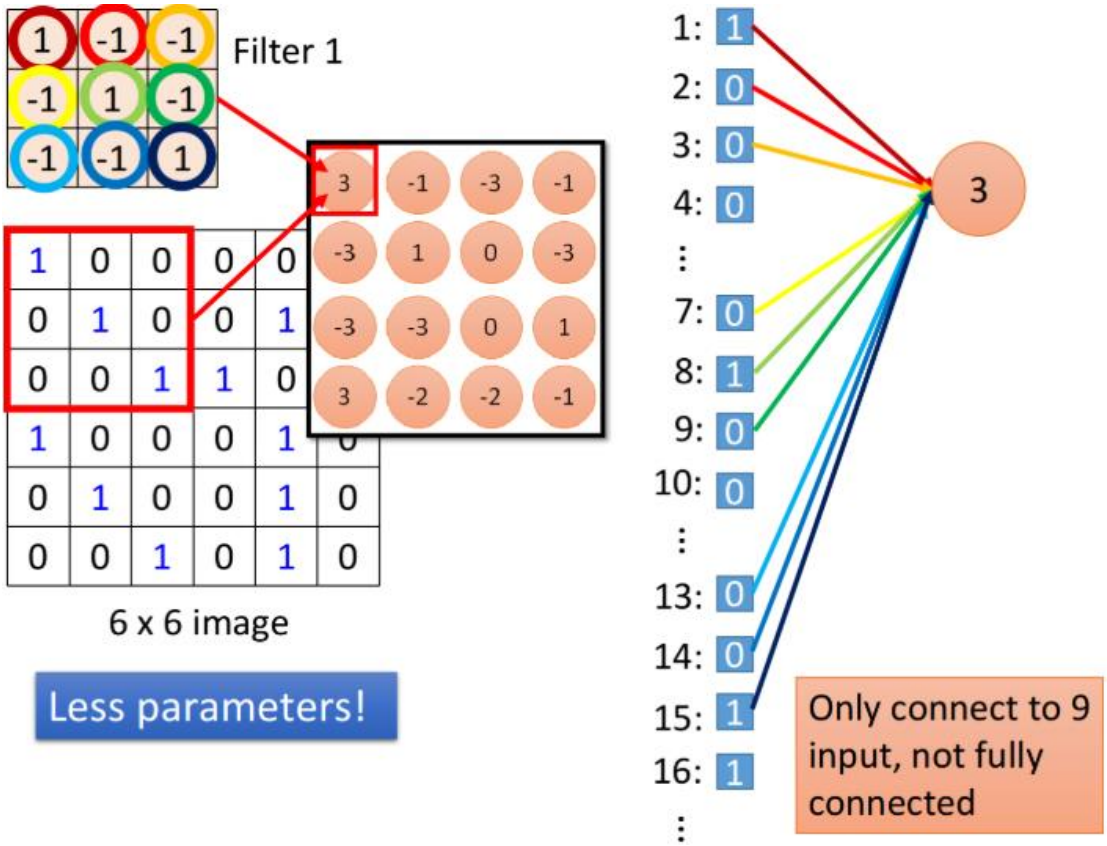
同理，若今天處理的是彩色圖片的話，則 input 共有三個 channel 且 filter 也有三個 channel；filter 每完成一次 detect 就會產生三個 channel 的 Feature Map。



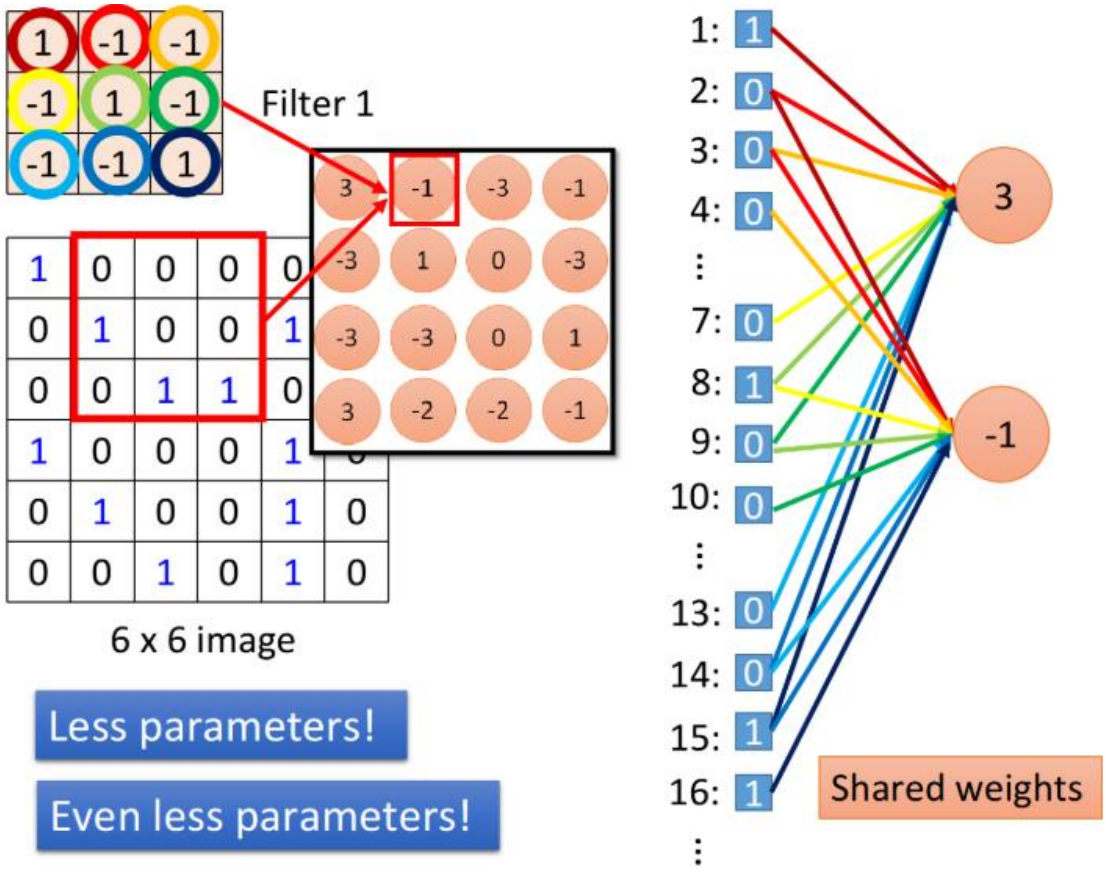
然而如果今天同一個 pattern 它有不同的 size，例如 input 的鳥類有大的鳥喙，也有小的鳥喙，則 CNN 並不能夠自動處理這個問題，需要事先將 image 經過適當的旋轉、縮放才能使用 CNN 處理之。

1. Convolution VS. Fully Connected

在進行 convolution 時，其實就是把 fully connected layer 的一些 weight 拿掉。
如下圖所示，我們把 filter 放在 image 的左上角，再去做內積，得到一個值 3。
這件事情等同於，我們現在把這個 image 的 6*6 的 matrix 拉直變成右邊這個 vector。然後這些 input 經過紅色的 neuron 之後，得到的 output 是 3。
而這個 neuron 就只有連接到編號為 1，2，3，7，8，9，13，14，15 的這 9 個 pixel 而已，其餘 pixel 的 weight 皆為 0。



接著當我們把 filter 做 stride = 1 的移動的時候，通過 filter 和 image matrix 的內積得到另外一個 output 值-1，我們假設這個-1 是另外一個 neuron 的 output，對應到編號 2，3，4，8，9，10， 14，15，16 這 9 個 pixel；而他們 connect 的與前一個 neuron 相同顏色的線，則表示具有相同的 weight（皆是 filter matrix 的 element）。



2. 補充：如何 train 此 network

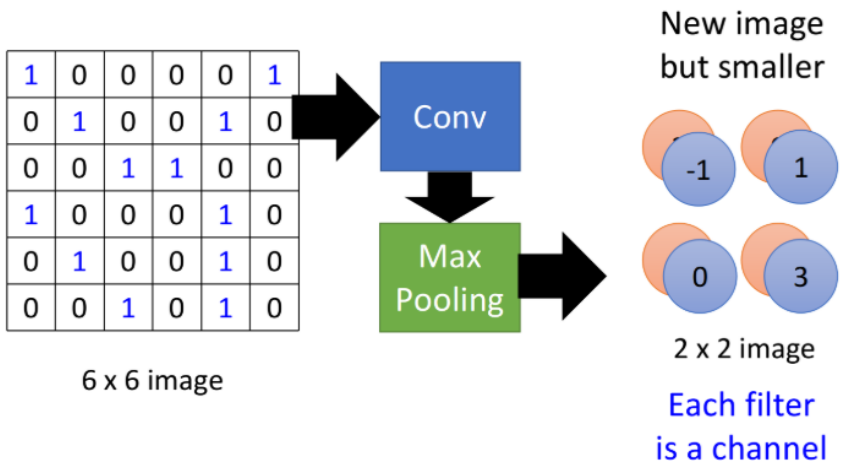
（節錄自影片逐字檔）
看到這裡你可能會問，這樣的 network 該怎麼搭建，又該怎麼去 train 呢？

首先，第一件事情就是這都是用 toolkit 做的，所以你大概不會自己去寫；如果你要自己寫的話，它其實就是跟原來的 Backpropagation 用一模一樣的做法，只是有一些 weight 就永遠是 0，你就不去 train 它，它就永遠是 0

然後，怎麼讓某些 neuron 的 weight 值永遠都是一樣呢？你就用一般的 Backpropagation 的方法，對每個 weight 都去算出 gradient，再把本來要 tight 在一起、要 share weight 的那些 weight 的 gradient 平均，然後，讓他們 update 同樣值就 ok 了。

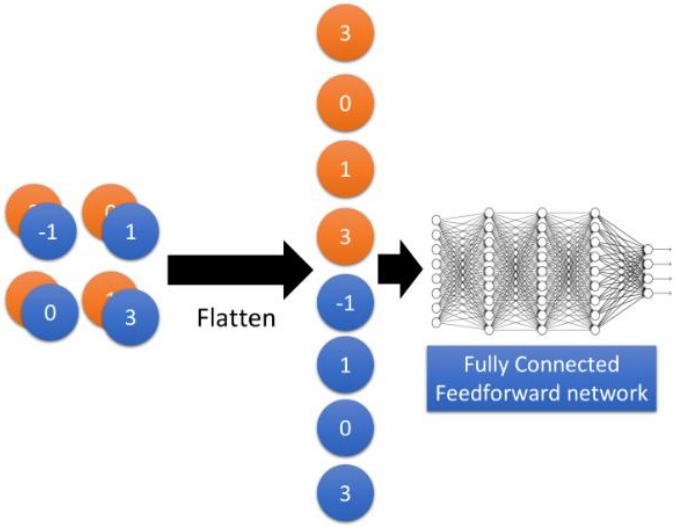
二、Max Pooling

該層 layer 就是進行 subsampling 的任務。
同樣以上述例子來說，根據 filter 1，我們得到一個 4*4 的 matrix，再根據 filter 2，得到另外一個 4 * 4 的 matrix。
接著把這兩個 output 每四個 element 分為一組，每一組裡面通過選取平均值或最大值的方式，把原來 4 個 value 合成一個 value，這件事情相當於在 image 每相鄰的四塊區域內都挑出一塊來檢測。



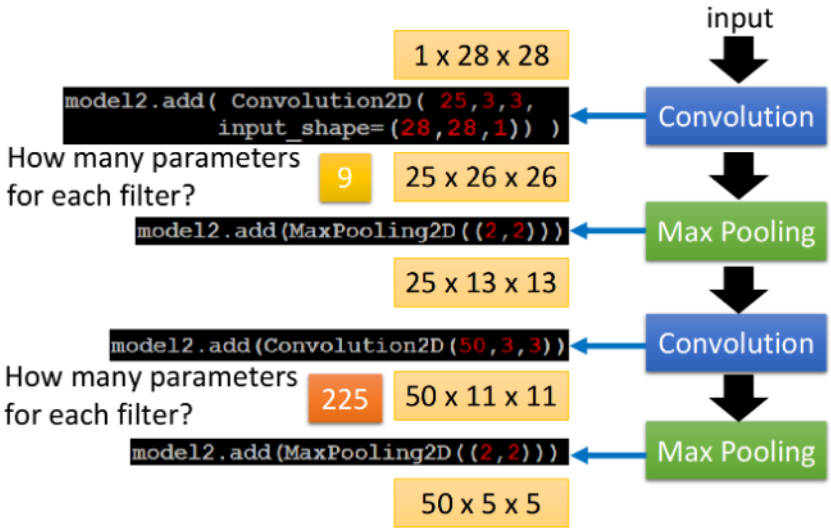
三、Flatten

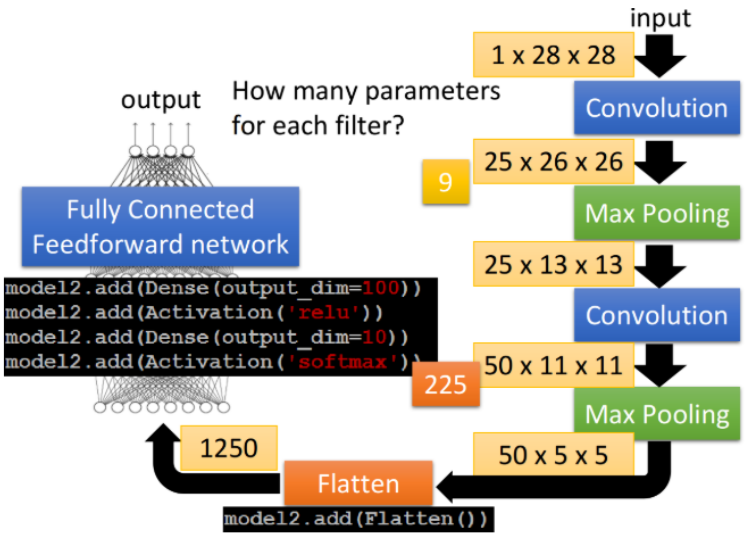
最後 Flatten layer 的工作就是把 feature map 拉直，然後丟進一個 Fully connected Feedforward network。



What Does CNN Learn？

下圖是用 Keras 表示 CNN 的結構的流程圖。

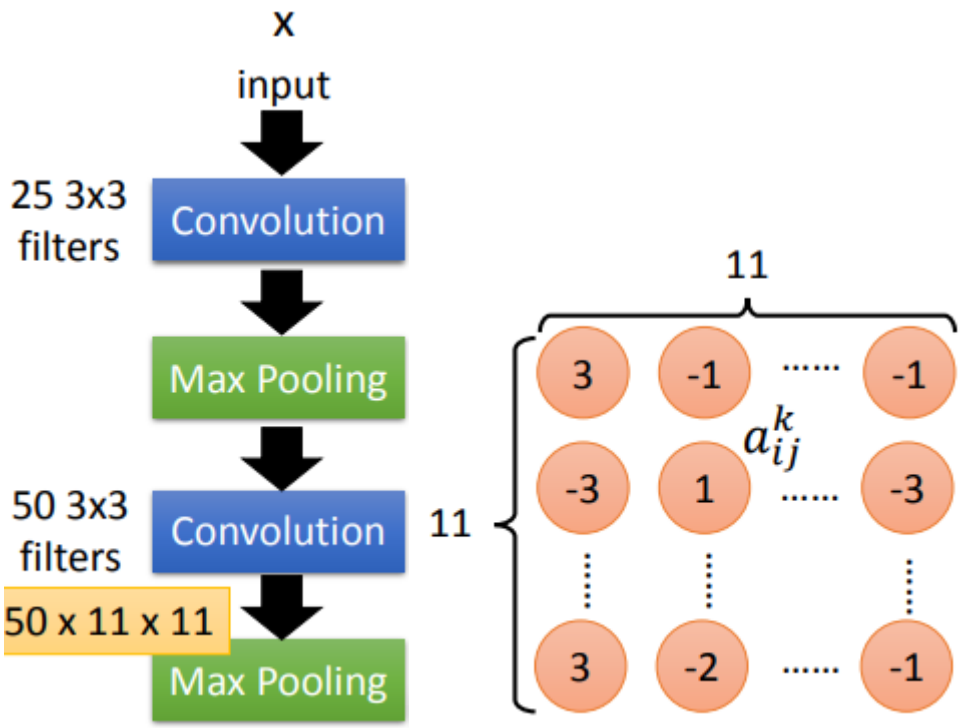




而假設我們想要得知在 CNN 中機器究竟學到了什麼，可以分成 filter 與最後丟進的 NN 的 neuron 還有 output 來分析。三者所採用的方式皆相似。

一、Filter

舉上圖例子來說，在第二個 convolution layer 裡面的 50 個 filter，每一個 filter 的 output 就是一個 11*11 的 matrix。
假設我們現在把第 k 個 filter 的 output 拿出來，如下圖所示，這個 matrix 裡的每一個 element，我們叫它a_{ij}^k，上標 k 表示這是第 k 個 filter，下標 ij 表示它在這個 matrix 裡的第 i 個 row，第 j 個 column。



接下來我們定義一個叫做 Degree of the activation of the k-th filter，這個值描述現在 input 的東西跟第 k 個 filter 有多接近，意即它對 filter 的激活程度有多少。
寫作下列式子：

$$a^k = \sum_{i=1}^{11} \sum_{j=1}^{11} a_{ij}^k$$

也就是說，我們 input 一張 image，然後把這個 filter 和 image 進行 convolution output 的 11*11 個值全部加起來，當作現在這個 filter 被 activate 的程度。
接著找一個 image x*，它可以讓我們定義的 activation 的 degree 最大，即：

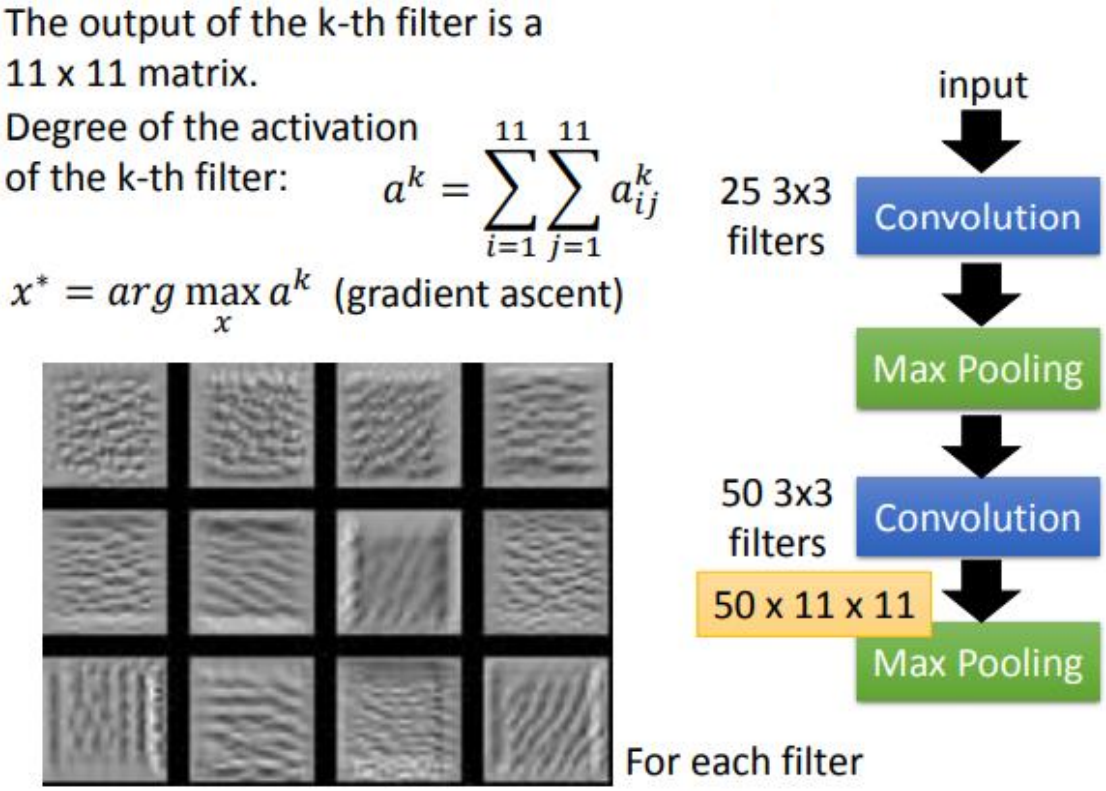
$$x^* = arg \max_x a^k$$

該步驟可用 gradient ascent 實現。

所以 50 個 filter 理論上可以分別找 50 張 image 使對應的 activation 最大。
而這些 image 有一個共同的特徵，它們裡面都是一些反覆出現的某種 texture。

比如說，下圖第三張 image 上佈滿了小小的斜條紋，這意味著第三個 filter 的工作就是 detect 圖上有沒有斜條紋，所以圖中一旦出現一個小小的斜條紋，這個 filter 就會被 activate，相應的 output 也會比較大。

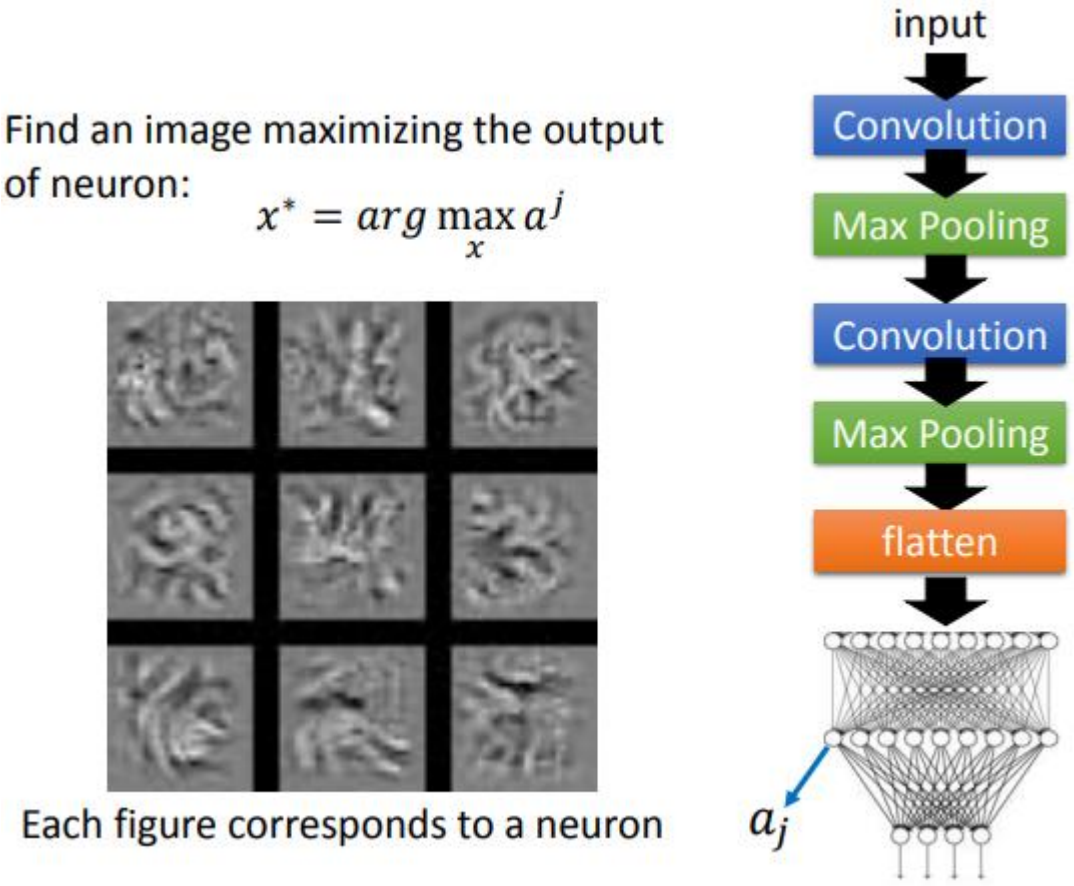
所以如果整張 image 上佈滿這種斜條紋的話，這個時候 filter 的 activation 程度是最大的，相應的 output 值也會達到最大。



因此每個 filter 的工作就是去 detect 某一種 pattern，detect 某一種線條，上圖所示的 filter 所 detect 的就是不同角度的線條，所以今天 input 有不同線條的話，某一個 filter 會去找到讓它 activation 程度最大的匹配對象，這個時候它的 output 就是最大的。

二、Neuron

方法同 filter，定義出 activation function 之後，用 gradient ascent 的方法去找一張 image x，把它丟到 neural network 裡面就讓a^j的值可以被 maximize。
結果為下圖所示。



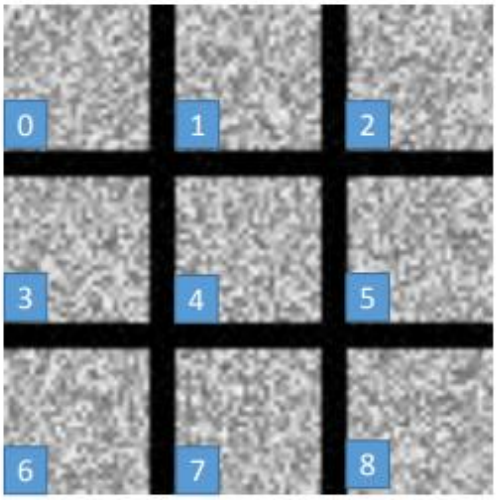
比較與 filter 所觀察到的情形，此處的 image 不再只是 texture，而是類似一張完整的圖形。
那是因為每個 filter 考慮的只是圖上一部分的 vision，所以它 detect 的是一種 texture；但是在做完 flatten 以後，每一個 neuron 不再是只看整張圖的一小部分，它現在的工作是看整張圖，所以對每一個 neuron 來說， activation 最大的 image，不再是 texture，而是一個完整的圖形。

三、Output

同樣的找一張 image x，使 activation function y*最大。

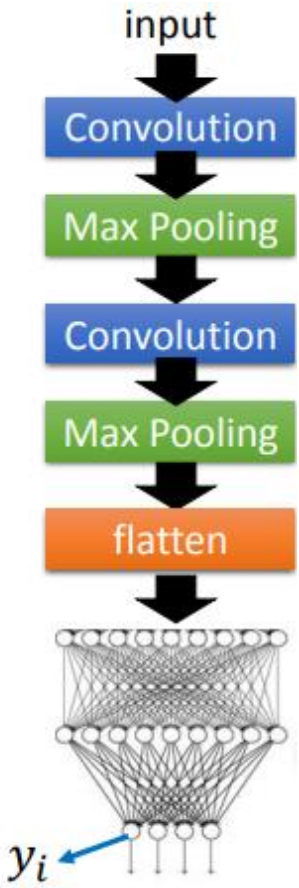
$$x^* = arg \max_x y^i$$

Can we see digits?



Deep Neural Networks are Easily Fooled

https://www.youtube.com/watch?v=M2IebCN9Ht4



然而理論上每一個 output 的每一個 dimension 對應到一個數字，因此如果去找一張 image x，讓它對應到數字 1 的那個 output layer，它的 neuron 的 output 值最大，那這張 image 顯然應該看起來會像是數字 1，但是結果卻並非是如此。

因為 neural network 經訓練之後，他所學到的東西一般都會與人類的認知相異，若要讓 x*呈現的樣子更像是數字，我們必須對找出來的 x 做一些 constraint。

例如可以使用 L1 的 regularization，希望找出來的 image 在大部分的地方是沒有塗顏色的，只有少數數字筆劃在的地方才有顏色出現。

$$x^* = arg\max_x (y^i - \sum_{i,j} |x_{ij}|)$$

$$x^* = arg\max_x y^i$$

$$x^* = arg\max_x \left(y^i - \boxed{\sum_{i,j} |x_{ij}|} \right)$$

Over all pixel values