ML Lecture 10: Convolutional Neural Network

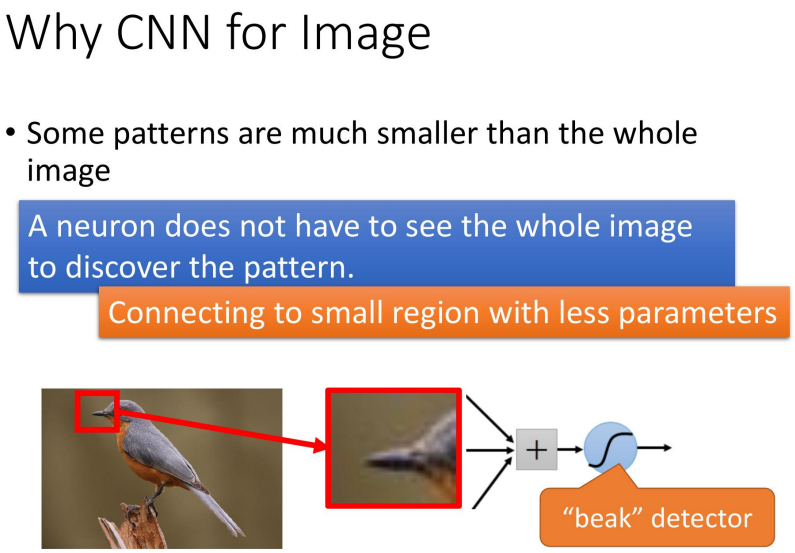
> 臺灣大學人工智慧中心 科技部人工智慧技術暨全幅健康照護聯合研究中心
> http://ai.ntu.edu.tw

Why CNN for Image ?

當我們使用一般的 DNN，如 Fully connected Feedforward network 做影像辨識時，往往會需要太多參數； 而透過以下幾點觀察（Property），CNN 可**簡化** neural network 的架構，**使用較少的參數**。

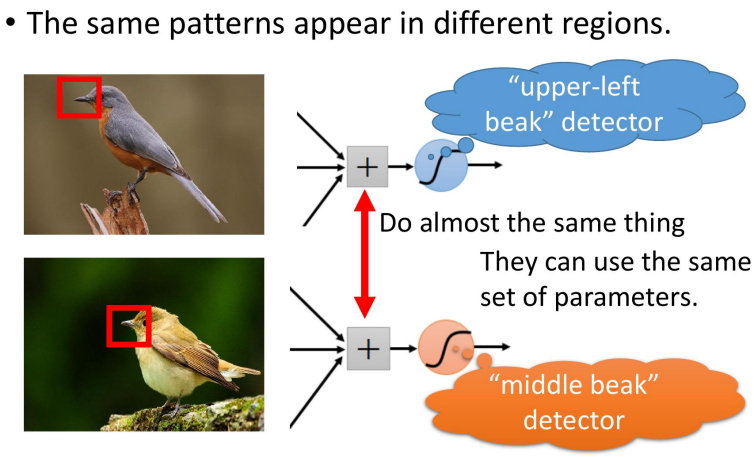
1. 偵測 pattern 是否存在時，不需要看整張 image，只需看圖片的一小部分即可決定，故可簡化參數。

   > 例：只需看下圖的紅色框框，就可決定圖中是否有鳥嘴的存在。



2. 同樣的 pattern 可能會出現在 image 的不同地方，只需用相同的 neuron 就可偵測出來，故可簡化參數。

   > 例：偵測上圖鳥嘴的 neuron 跟偵測下圖鳥嘴的 neuron 可能是一樣的，故我們可共用同一組參數。

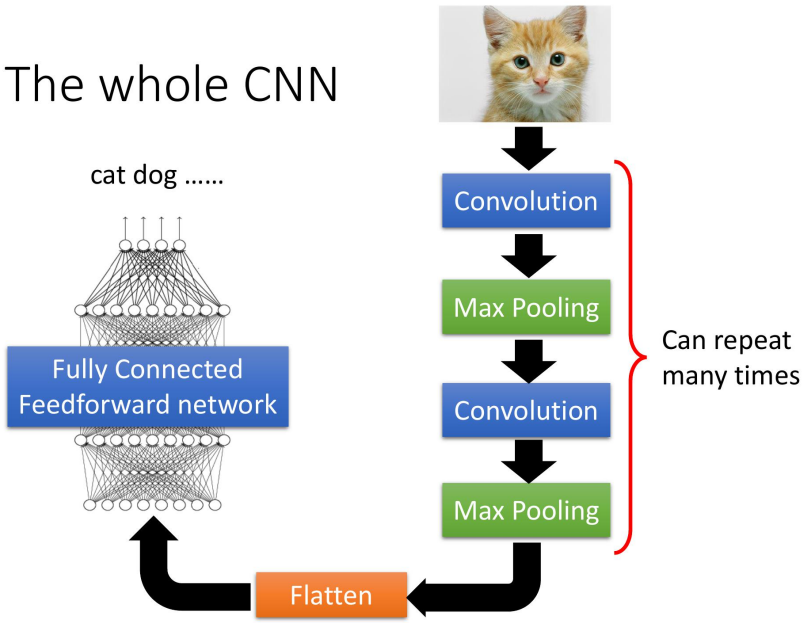


3. 對 image 做 **subsampling**，這件事對影像辨識沒有太大影響，也可簡化參數

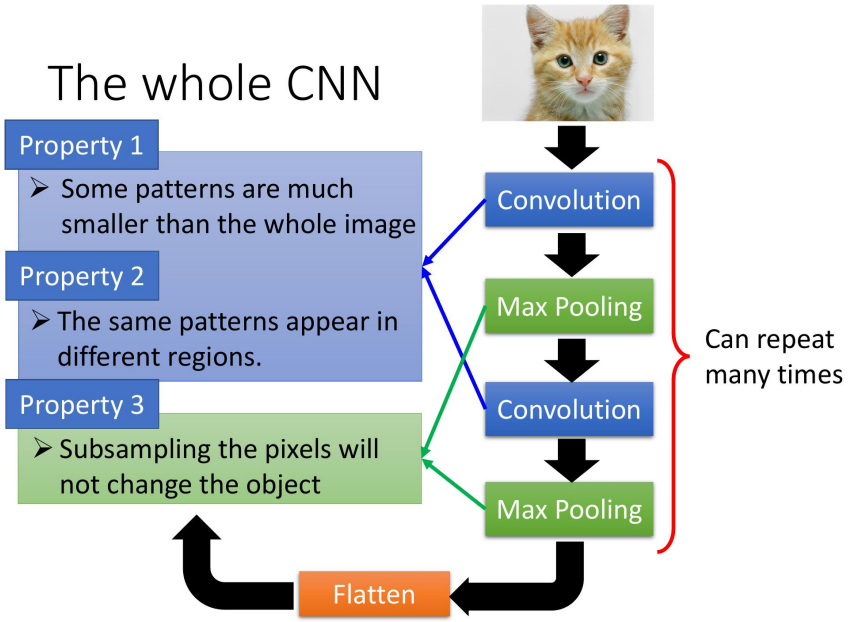
> subsampling：拿掉一張 image 的奇數行及偶數列，使其大小變為原本的 1/10

Structure of CNN

- Input 一張 image ➡ 反覆做 [Convolution 及 Max Pooling] * (事先決定好的次數) ➡ flatten ➡ output 的結果再 input 至一般的 Fully connected Feedforward network 中



- Convolution layer：處理上述第一、二個與 pattern 有關的 property Max Pooling：處理 Subsampling



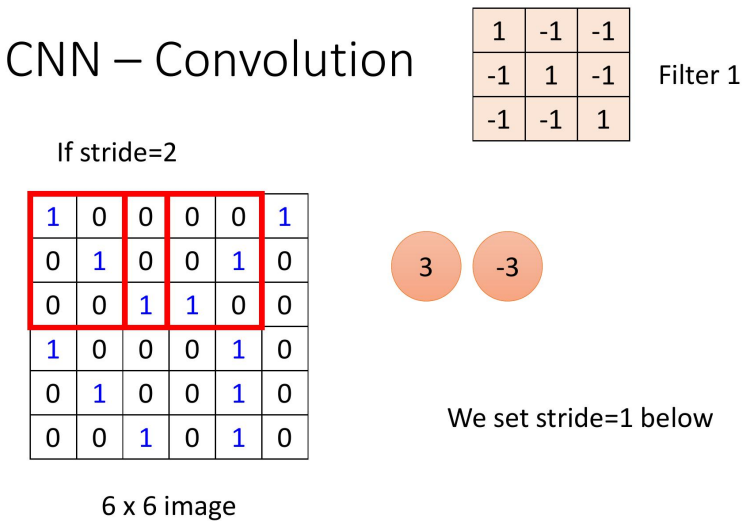
Part 1 - Convolution

黑白圖像（0 代表白色，1 代表黑色）

- **Filter**：Convolution layer 裡有許多組 filter，一個 filter 其實就是一個 matrix

  Filter 中的參數是根據 training data 學得的，3*3 的 size 代表偵測一個 3*3 的 pattern

- **內積**：filter 從左上角開始放，與 image 中對應的值做內積；如下圖，最左上角與 filter 做內積後得到 3
- **Stride**：挪動 filter 的位置，移動多少稱作 stride；如下圖，紅色框框一次偏移 2 格，stride = 2

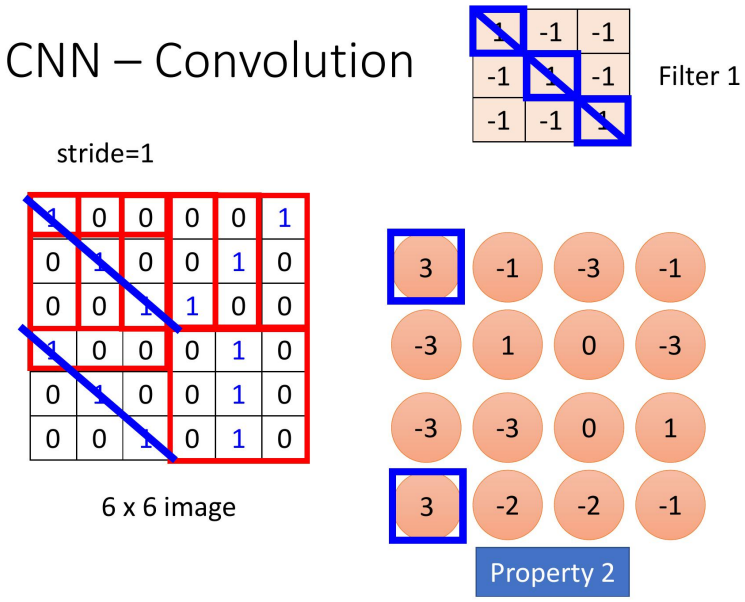


- **偵測**：觀察 filter，他要偵測的 pattern 為圖中是否有出現「左上到右下斜線的值為 (1, 1, 1)」

  亦即，內積完後，出現最大值的地方，就代表這個 filter 要偵測的 pattern

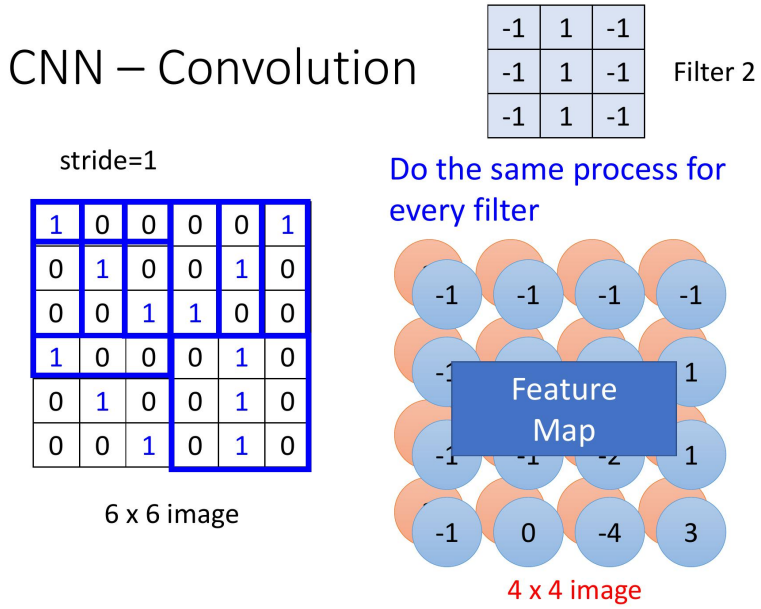
- **Property 2**：做此步驟時，有考慮 Property 2（同樣的 pattern 可用相同的 filter 偵測出來）

  > 如下圖，左上角及左下角出現了同樣的 pattern，我們用同一個 Filter1 即可偵測出來



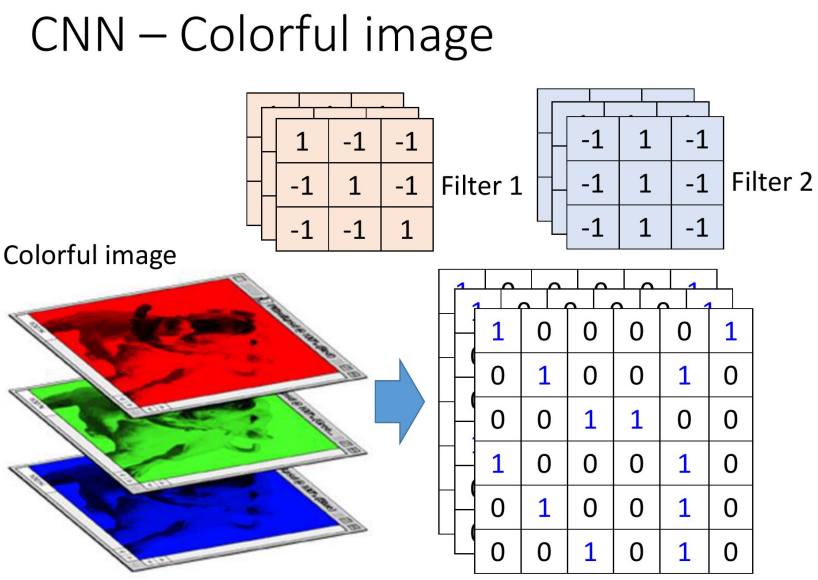
- **Feature Map**：一層 convolution layer 中有許多 filter，每個 filter 做完內積後的 image 合稱 Feature Map

> 如下圖，Filter 2 做完後，得到另一個藍色 4*4 的 matrix，與紅色 4*4 的 matrix 合稱 Feature Map



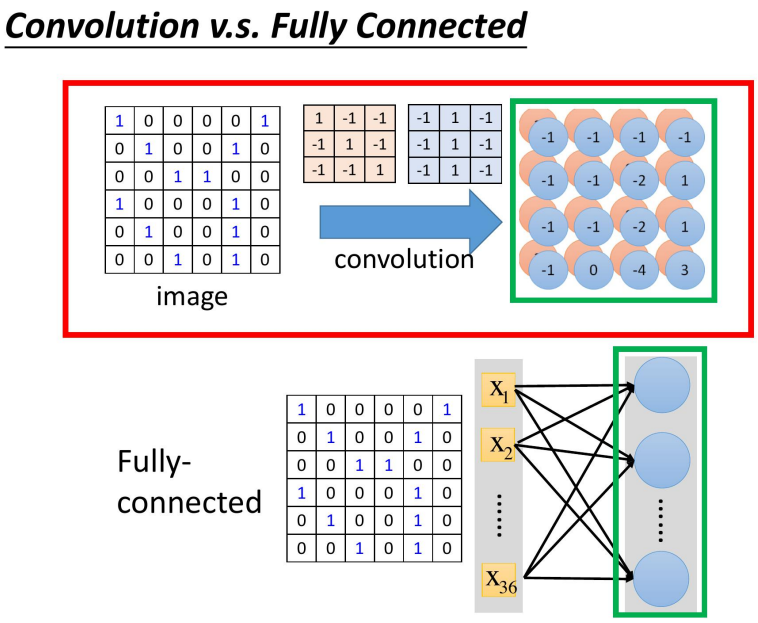
彩色圖像（RGB）

- 彩色圖像就是好幾個 matrix 疊在一起，是一個立方體；故我們使用的 filter 也是「立方體」。計算時，並非每個顏色（channel）分開算，而是每個 filter 同時考慮了不同顏色代表的 channel

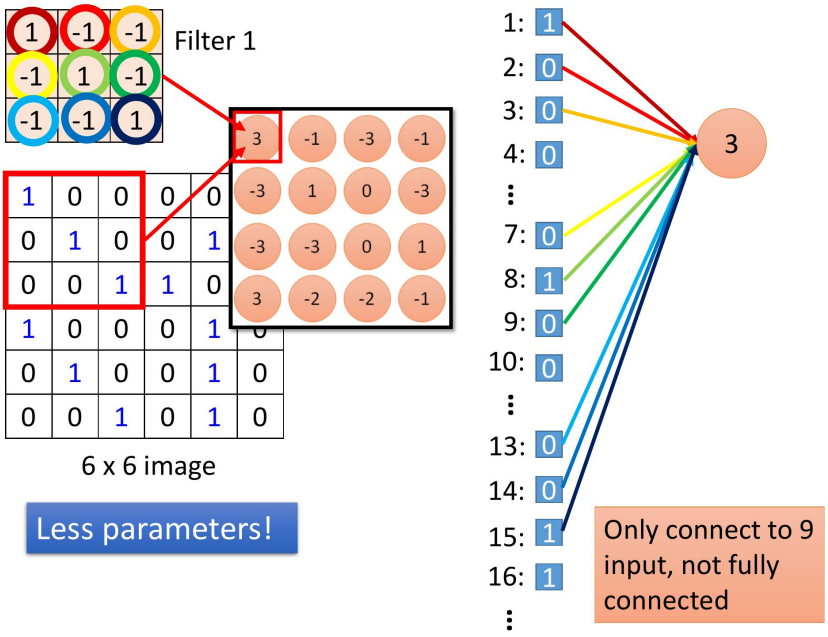


- Convolution 與 Fullu Connected 的關係
  - Convolution 其實就是一個 neural network
  - Convolution 就是 Fully Connected 的 layer 將一些 weight 拿掉

- Feature Map 這個 output 就是 Fully Connected neuron 的 output（綠色框框互相對應）

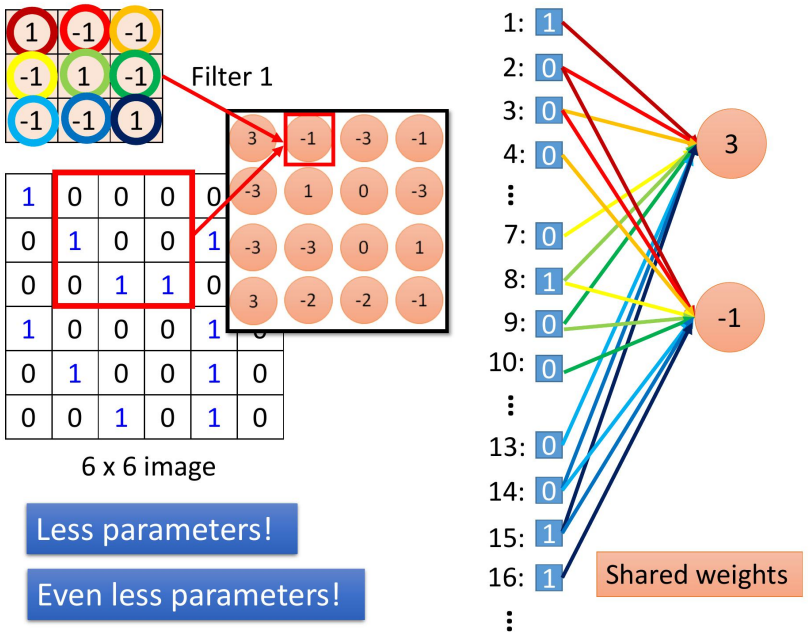


- Convolution 與 Fullu Connected 相對應
  - Convolution 的 filter 放在左上角，考慮的 pixel 是 [1, 2, 3, 7, 8, 9, 13, 14, 15]
  - 將左邊 6*6 的 Image 拉成直的，filter 做內積後得到 3，即右圖某個 neuron 的 output 為 3
  - 左圖 filter 代表的意思為，右圖 neuron 的 weight 只連接到 [1, 2, 3, 7, 8, 9, 13, 14, 15] 這些 pixel 而已



- Stride = 1
  - filter 移動到下一格，得 -1，亦即另外一個 neuron 的 output 為 -1

    - 這個 neuron 就只連接到 [2, 3, 4, 8, 9, 10, 14, 15, 16] 這些 pixel
- 在 Fully Connected layer 中，每個 neuron 本來都有自己獨立的 weight；當我們做 convolution 時
    1. 減少每個 neuron 前面連接的 weight
    2. 強迫某些 neuron 要共用某一組 weight（如下圖中的 pixel 2, pixel 3 等）即 **Shared weights**



Part 2 - Max Pooling

- 根據 Filter 1, Filter 2，我們可以得到兩個 4*4 的 matrix

  將他們 4 個一組（如下圖），接著每組中選出一個數（可選平均、最大值.....自行定義）

- 這裡，我們選最大值，故做完一次 Convolution + Max Pooling

  我們將原來 **6*6** 的圖像化簡成一個 **2*2** 的圖像

  > $$Filter1 = \begin{bmatrix} 3 & 0 \\ 3 & 1 \end{bmatrix} \quad Filter2 = \begin{bmatrix} -1 & 1 \\ 0 & 3 \end{bmatrix}$$

- **image 深度(維度)**：由「filter 個數」決定

  > 例：這裡只有 Filter 1 及Filter 2，故右下圖中的維度就是 2

- 反覆「Convolution + Max Pooling」的動作 我們可以重複「Convolution + Max Pooling」的動作，讓 image 越來越小

Part 3 - Flatten

將 Feature Map 拉直，丟到一個 Fully Connected Feedforward Network 中，就結束了

CNN in Keras

- 用 Keras implement CNN 時
  - 相同：training, compile 及 fitting 的部分是一模一樣的
  - 不同：network 的 structure 及 input 的 format
- Convolution 時

  > structure：dense 改成 Convolution2D，(25, 3, 3) 代表 25 個 filter，每個 filter 是 3*3 的 matrix input：(28, 28, 1) 代表 28*28 的 image，黑白 image 放 1，彩色 image 放 3

- Max Pooling 時

  > 多 add 一個 2*2 的 layer，代表 2*2 個 pixel 一組，每組中挑一個數出來

- 實作
  - Input：1*28*28 的 image
  - **Convolution**：通過 25 個 3*3 的 filter

- Output：25*26*26（25 是 channel 的數目，28*28 通過 3*3 的 filter 不考慮邊得 26*26 image）
- **Max Pooling**：2*2 的 pixel 圈成一組，挑其中最大的值
- Output：25*13*13
- **Convolution**：通過 50 個 3*3 的 filter
- Output：50*11*11
- **Max Pooling**：2*2 的 pixel 圈成一組，挑其中最大的值
- Output：50*5*5

- - **Flatten**：將 50*5*5 拉直成一個 1250 維的 vector，

    丟到一個 Fully Connected Feedforward Network 裡面
  - Output：result

What does CNN Learn ?

> 分析 CNN 的每個步驟

- Convolution - 第一層 filter
  - **理解容易**：只要分析 filter 內的值，很容易就知道它在 detect 什麼 pattern
  - **較難想像**：但是，比較難想像第一層 filter 在做什麼事情
- Convolution - 第二層 filter
  - **轉化目標**：轉而分析這些 3*3 filter 的 output，亦即那些 11*11 的矩陣
  - **作法**：
    1. 取出第 k 個 11*11 的矩陣，裡面每一個元素我們以 $a_{i,j}^k$ 表示 k 表示第 k 個 filter，下標 i, j 代表矩陣中的第 i 個 row，第 j 個 $column$
    2. 定義 **Degree of activation of the k^{th} filter** $=$ 所有元素的和 代表第 k 個 filter 被 activate 的程度，亦即現在的 input 跟第 k 個 filter 有多 match
    3. 找一張 image(x)，它可以最大化第 k 個 filter 被 activate 的程度，使用 gradient ascent 找到這張 image，可以讓 degree 最大，我們觀察 image 就可以知道 filter 在 detect 什麼就結束了
  - **舉例**：取前 12 個 filter 為例，這些圖共同的特徵就是有線條紋的 pattern 故，這裡每一個 filter 的工作就是 detect 圖上是否有某一種線條的 pattern
- Flatten - 每一個 neuron
  - **作法**：定義 a_j 為第 j 個 neuron 的 output，再用 Gradient Descent 找一張 image,x 使 a_j 被 maximize

- **舉例**：可使前 9 個 neuron 最大化的圖如下，與 filter 不同的是，filter 觀察的是圖的一小部分，neuron 觀察 的是整張圖；故，這裡的 neuron 工作就是偵測較大的 pattern，如有圓圈有線條等

- Output
  - **作法**：找一張 image，讓該維度的 output 最大
  - **舉例**：下面八張圖就是機器學習到的數字 1~8，顯然地，機器學習到的東西跟人類不一樣

    > Deep Neural Network are Easily Fooled 一段影片中有不少例子

  - **改進**：對 image, x 做一些限制，讓機器了解雖然有一些 x 可以最大化 output，但這些 x 不是數字

    > 這裡加的限制是 maximize y 的同時，$\Sigma_{ij}|x_{ij}|$ 越小越好，亦即找出來的 image，只有少部分是筆畫

Deep Dream

- 精神：給機器一張 image，它會在 image 裡面加上它看到的東西

- 原理：
  - 將 image 丟入 CNN 後，把某一個 hidden layer 拿出來，將這個 filter 的值調大
  - 讓 positive 的越大，negative 的越小（正的越正，負的越負）
  - 再將這個調整後的 vector 當作新 image 的目標

  亦即，讓 CNN 誇大化它看到的東西（如下圖，有一些念獸出現）

Deep Style

- 精神：給機器一張 image，讓機器修改這張圖，給予另一張圖的風格

  > 例：在原本的照片基礎上，增加吶喊一幅畫的風格

- 原理：

  1. 原圖丟入 CNN 處理後，得到的 output 代表這張 image 的 content
  2. 欲成為的風格圖也丟入 CNN 處理後，得到的 output 代表這張 image 的 style
  3. 用同一個 CNN，找一張能同時 maximize 左右兩邊的 image 即為 output

> *reference: A Neural Algorithm od Artistic Style*

- 成果

More Application of CNN

- 圍棋（Go）

  > 其實一般的 neural network 也可讓 machine 學會下圍棋，但 CNN 可獲得更好的表現

  - 對 CNN 來說，棋盤可以當成一張 19*19 的 image 來看
  - training data 就是棋譜，而 machine 將學會 output 落子的地方
  - 使用 CNN 可以獲得較佳表現的原因，是因為圍棋有一些特性，跟講義開頭講的三種 property 類似
    1. 圍棋有一些 pattern 就如同鳥喙，只需看整張圖的一小部分即可偵測出來
    2. 同樣的 pattern 可能重複出現在一張棋盤的其他地方，並代表相同的意義
  - **但是**，第三點 subsampling（Max pooling）在圍棋不適用，那怎麼解釋 AlphaGo 優異的表現呢？
  - **其實**，AlphaGo 沒有使用 Max Pooling

    > 下圖為 AlphaGo 論文附錄的內容，底線處描述了它 neural network 的 structure

- 音像處理（Speech）

  > 將音像以頻譜表示（橫軸：一段時間、縱軸：那段時間裡聲音的頻率）

  - 我們可以將聲音頻譜當作 input 的 image，讓 CNN 判斷這對應到什麼樣的聲音訊號
  - 特別的是，我們移動 filter 的時候，只移動 frequency 的方向，在時間方向移動沒有太大幫助

- 文字處理（Text）

  > 每一個 word 都用一個 vector 表示，一個 sentence 中所有的 word 排在一起，就變成一張 image

- 丟入一個 word sequence，我們希望 CNN output 這個句子的含義是正面的還是負面的
- Filter 的高和 image 的高相同，沿著 word 順序移動； 不同的 filter 會得到不同的 vector，再做 Max Pooling； 丟入 Fully Connected Feeddorward layer，最後得到 output。