

# OpenCV 影像辨識-使用 Python

授課講師

党榮安 DANG Steve

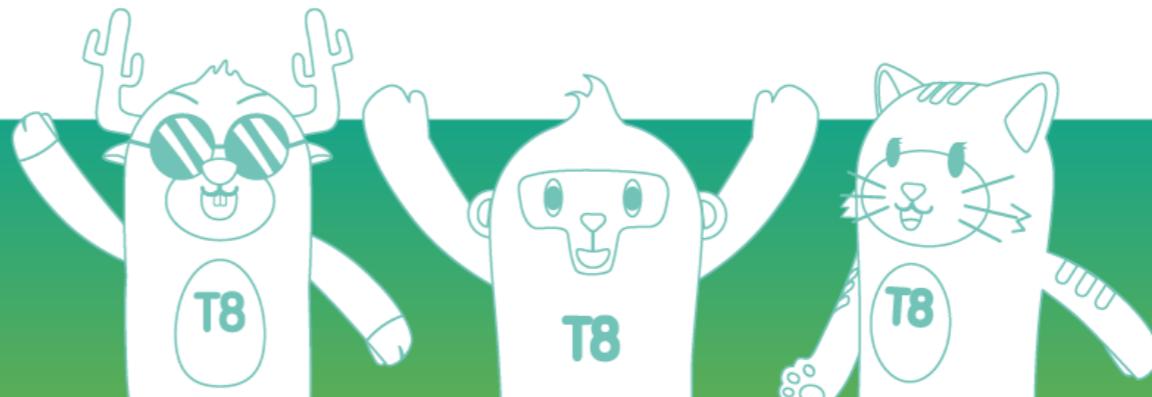
教材編寫

党榮安 DANG Steve

緯  
育 *TibaMe*

即學 · 即戰 · 即就業

<https://www.tibame.com/>



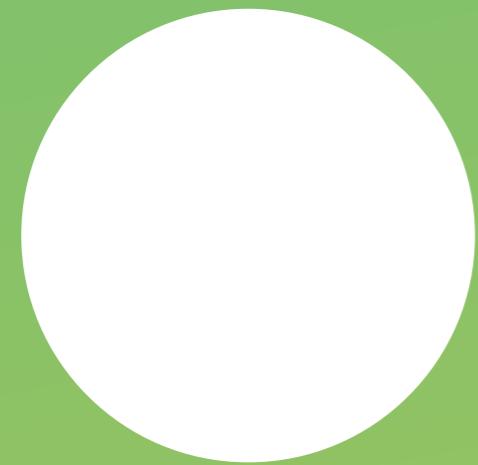
## 課程大綱

- ◆ Module 1. OpenCV套件
- ◆ Module 2. OpenCV繪圖
- ◆ Module 3. 色彩空間
- ◆ Module 4. 圖片幾何轉換
- ◆ Module 5. 濾波器
- ◆ Module 6. 設定值處理
- ◆ Module 7. 邊緣檢測
- ◆ Module 8. 輪廓偵測(contours)
- ◆ Module 9. 形態學
- ◆ Module 10. 圖像金字塔

## 課程大綱

- ◆ Module 11. 影像模板匹配
- ◆ Module 12. 特徵擷取, 匹配(SIFT)
- ◆ Module 13. 背景提取 & Hough 霍夫變換
- ◆ Module 14. 直方圖處理
- ◆ Module 15. 視訊處理
- ◆ Module 16. OpenCV 函式庫 DLib
- ◆ Module 17. OCR 光學字元識別
- ◆ Module 18. 人臉辨識

## 授課講師介紹



党榮安

### 專長

1. Data Science, AI, 統計
2. Python, SQL Server, Data Analysis, OpenCV, ML

### 簡歷

1. 曾任 MO, US 州政府 SAS Engineer
2. 曾任上市櫃公司資訊處處長
3. 電子業產品專案處長
4. 專業 Python, SQL Server, OpenCV, AI 講師, 專案指導

### 老師的話

1. 理解勝過記憶
2. 持之以恆, 功夫是一步一腳印練出來的

### 聯絡方式

<https://www.facebook.com/groups/icoding>

# 學習本課程須知

## 先備知識

1. Python
2. Numpy
3. Matplotlib
4. 基本數學概念

## 學習目標

- 了解 OpenCV 對照片, 影片的各種處理方式, 以達到預期的效果.
- 演算法的數學理論理解不易, 但演算法概念要理解, API 的運用一定要掌握.

## 學習方式

多想, 多看, 多練, 多問

## 須完成哪些作業或考試

小專題

# Module 1. OpenCV套件

- 1-1: OpenCV環境安裝
- 1-2: 彩色、灰階照片讀取/寫入
- 1-3: OpenCV影像基礎操作

## NOT suggest to use Conda and VS code

在 (命令提示字元 / Windows PowerShell \*\*系統管理員\*\*) 身分下打

- pip install jupyterlab 指令( \*\*灌程式\*\* )

在 (命令提示字元 / Windows PowerShell \*\*一般身分\*\*) 下打

- jupyter lab ( \*\*執行程式\*\* )(shell> jupyter lab)

or install Anaconda (:>jupyter notebook) (暫不建議, 為了維持一套 python 在電腦中)

**NOT suggest to use Conda and VS code**

install following in command line

- **install python \*\*3.6.8\*\***
- pip install jupyterlab
- pip install opencv-python ( pip install numpy )
- pip install opencv-contrib-python
- pip install matplotlib
- pip install scikit-image
- pip install imutils
- pip install dlib==19.8.1 ( or dlib-19.8.1-cp36-cp36m-win\_amd64.whl ) (**\*python 3.6.8\***)

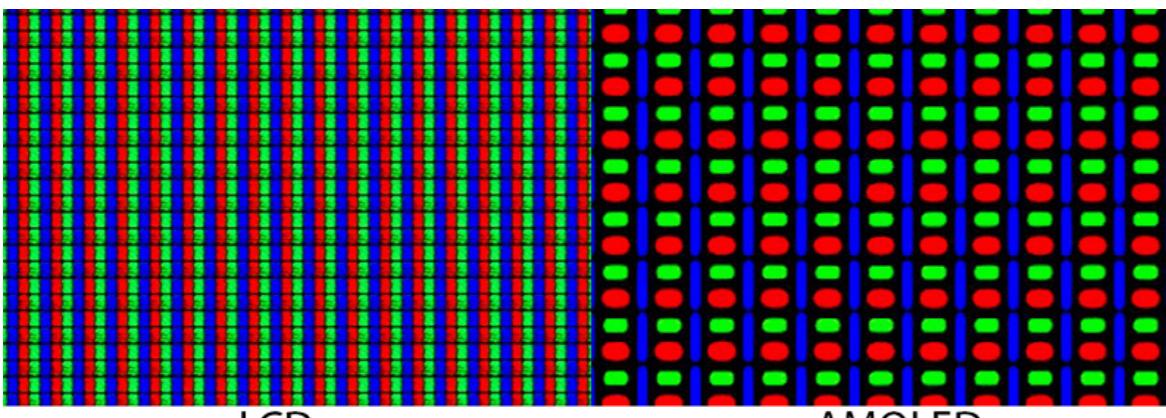
## Anaconda : 建立 python 3.6.8 虛擬環境

- or install Anaconda ( :>jupyter notebook) (暫不建議, 為了維持一套 python 在電腦中)
  - conda create --name py368 python=3.6.8 --channel conda-forge
  - conda \* install -c conda-forge jupyterlab
- :  
:  
:

## 1-2: 彩色、灰階照片讀取/寫入

Read / Open image file : B:0, G:1, R:2

讀取支援的格式 : bmp, pbm, pgm, ppm, jpeg, jpg, tiff, tif, png ....



pixel image



imread

3-channel matrix

|     |     | Blue  |     |     |    |     |
|-----|-----|-------|-----|-----|----|-----|
|     |     | Green | 255 | 134 | 93 | 22  |
| Red | 255 | 255   | 134 | 202 | 22 | 2   |
|     | 123 | 94    | 83  | 2   | 92 | 30  |
| 34  | 44  | 187   | 92  | 4   |    | 124 |
| 34  | 76  | 232   | 124 | 4   |    | 142 |
| 67  | 83  | 194   | 202 |     |    |     |

reshaped image vector

$$\begin{pmatrix} 255 \\ 231 \\ 42 \\ 22 \\ 123 \\ 94 \\ \vdots \\ 92 \\ 142 \end{pmatrix}$$

im2vector  
(or flatten)

## RGB to Gray

- 加權平均法：根據重要性及其它指標，將三個分量以不同的權值進行加權平均。由於人眼對綠色的敏感最高，對藍色敏感最低，因此，按下式對RGB三分量進行加權平均能得到較合理的灰度影像。

$$F(i, j) = 0.299 R(i, j) + 0.587 G(i, j) + 0.114 B(i, j)$$

- 分量法：將彩色影像中的三分量的亮度作為三個灰度影像的灰度值，可根據應用需要選取一種灰度影像。

$$F1(i, j) = R(i, j) \quad F2(i, j) = G(i, j) \quad F3(i, j) = B(i, j)$$

- 最大值法：將彩色影像中的三分量亮度的最大值作為灰度圖的灰度值。

$$F(i, j) = \max(R(i, j), G(i, j), B(i, j))$$

- 平均值法：將彩色影像中的三分量亮度求平均得到一個灰度值。

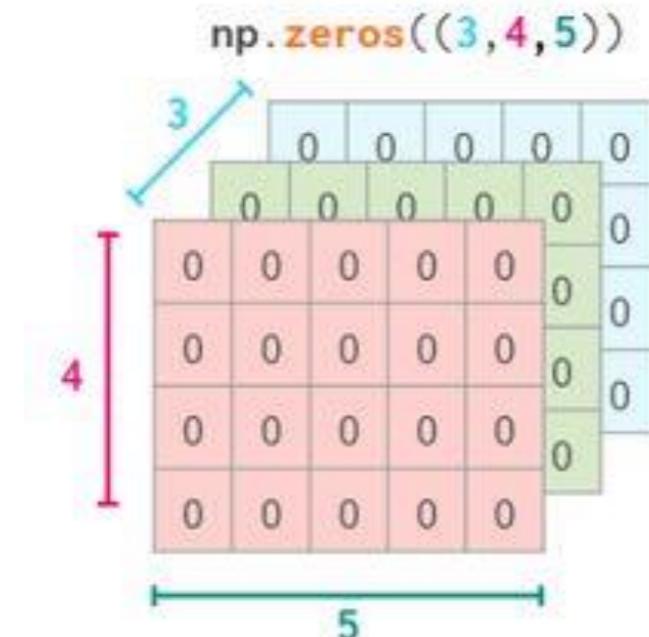
$$F(i, j) = (R(i, j) + G(i, j) + B(i, j)) / 3$$

讀取 : `cv2.imread('./image/SpongeBob.jpg' , 0)`

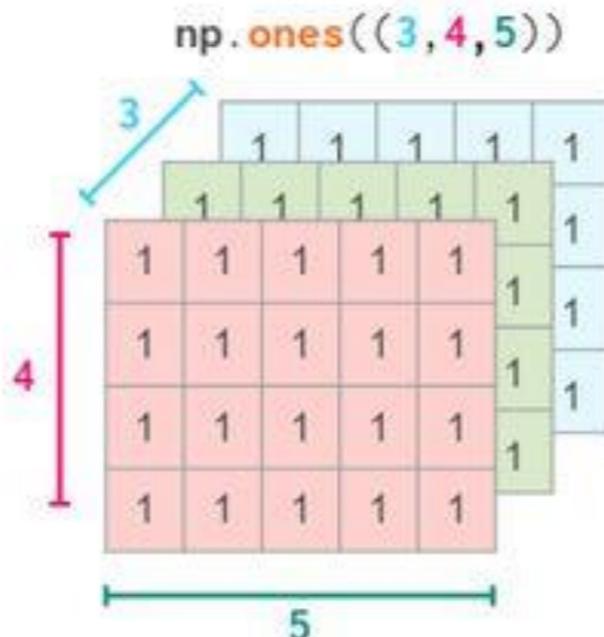
| 數值 | 含意         | 語法                                |
|----|------------|-----------------------------------|
| -1 | 保持原格式不便    | <code>cv2.IMREAD_UNCHANGED</code> |
| 0  | 單通道灰階      | <code>cv2.IMREAD_GRAYSCALE</code> |
| 1  | 3通道BGR(預設) | <code>cv2.IMREAD_COLOR</code>     |

寫入 : `cv2.imwrite('./image/baby01.jpg', img0)`

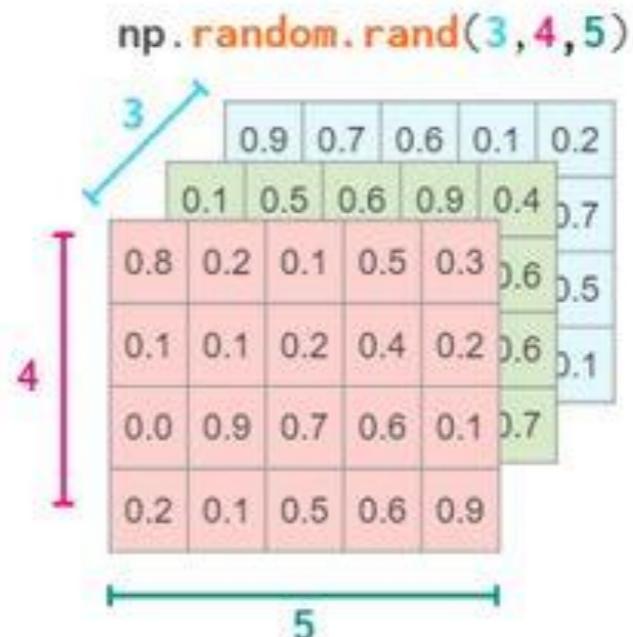
### Generic 3D arrays creation



`np.zeros((4, 5, 3))`



`np.ones((4, 5, 3))`



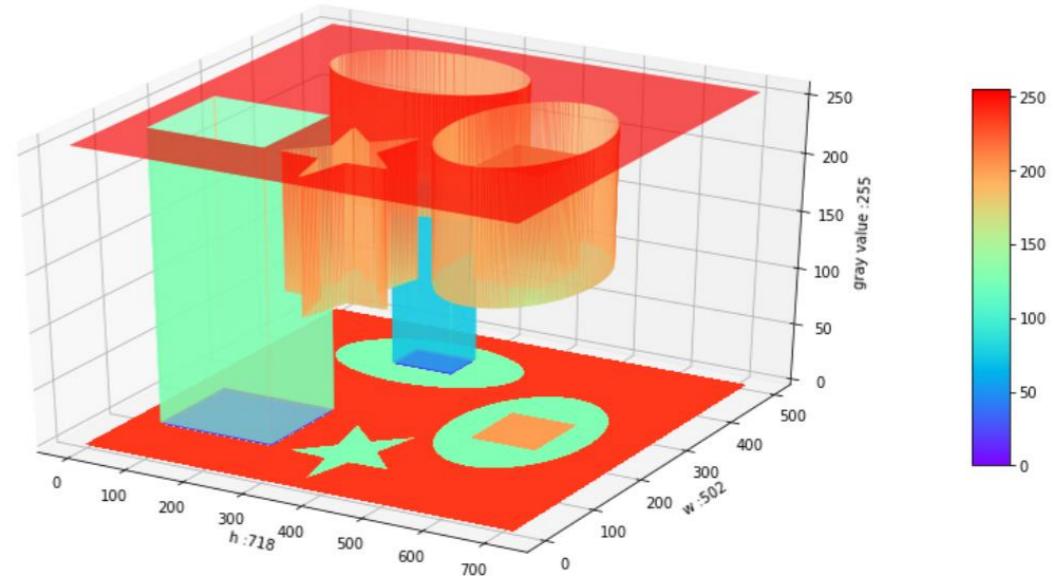
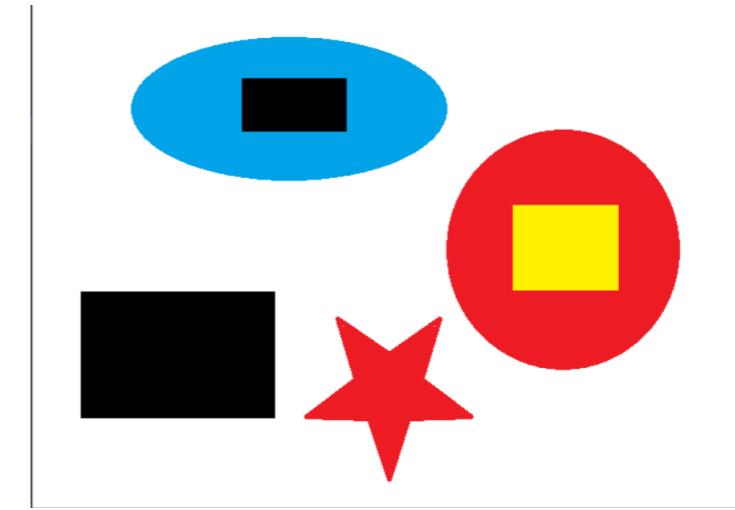
`np.random.rand(4, 5, 3)`

### RGB images creation

## Data Types for ndarrays

| 名稱            | 描述                            | 簡寫    |
|---------------|-------------------------------|-------|
| np.bool       | 用一個位元組儲存的布爾型別 ( True或False )  | 'b'   |
| np.int8       | 一個位元組大小，-128 至 127            | 'i'   |
| np.int16      | 整數，-32768 至 32767             | 'i2'  |
| np.int32      | 整數，-2^31 至 2^32 -1            | 'i4'  |
| np.int64      | 整數，-2^63 至 2^63 -1            | 'i8'  |
| np.uint8      | 無符號整數，0 至 255                 | 'u'   |
| np.uint16     | 無符號整數，0 至 65535               | 'u2'  |
| np.uint32     | 無符號整數，0 至 2^32 -1             | 'u4'  |
| np.uint64     | 無符號整數，0 至 2^64 -1             | 'u8'  |
| np.float16    | 半精度浮點數：16位元，正負號1位，指數5位，精度10位  | 'f2'  |
| np.float32    | 單精度浮點數：32位元，正負號1位，指數8位元，精度23位 | 'f4'  |
| np.float64    | 雙精度浮點數：64位元，正負號1位，指數11位，精度52位 | 'f8'  |
| np.complex64  | 複數，分別用兩個32位元浮點數表示實部和虛部        | 'c8'  |
| np.complex128 | 複數，分別用兩個64位元浮點數表示實部和虛部        | 'c16' |
| np.object_    | python物件                      | 'O'   |
| np.string_    | 字串                            | 'S'   |
| np.unicode_   | unicode型別                     | 'U'   |

- ROI ( Region of Interest )
- 指定pix位置 x, y
- 分割與合併，色彩通道
- image 3D data

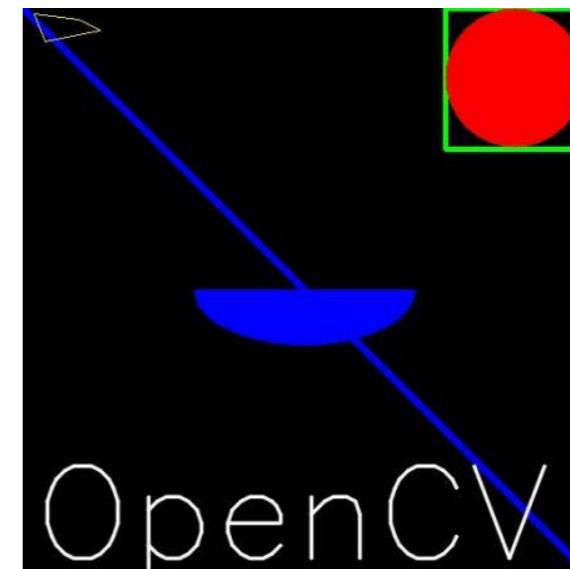


# Module 2. OpenCV繪圖

- 2-1: 基礎繪圖
- 2-2: 滑鼠交互
- 2-3: 滾動條

OpenCV 在圖片上加上線條等幾何圖案以及文字標示。在影像處理的程式中，若要比較清楚呈現處理的結果，時常會需要在圖片上加上一些標示的幾何圖形或是文字，比方說在物件辨識的問題上，可能會使用方框將辨識出來的物件框起來，並加註一些文字描述等

- line : cv2.line ( 影像, 開始座標, 結束座標, 顏色, 線條寬度 )
- rectangle : cv2.rectangle(影像, 頂點座標, 對向頂點座標, 顏色, 線條寬度)
- circle : cv2.circle ( 影像, 圓心座標, 半徑, 顏色, 線條寬度 )
- ellipse : cv2.ellipse ( 影像, 中心座標, 軸長, 旋轉角度, 起始角度, 結束角度, 顏色, 線條寬度 )
- cv2.polyline ( 影像, 頂點座標, 封閉型, 顏色, 線條寬度 )
- Bitwise
  - bitwise\_and
  - bitwise\_or
  - bitwise\_xor
  - bitwise\_not



Font : cv2.putText(影像, 文字, 座標, 字型, 大小, 顏色, 線條寬度, 線條種類)

- cv2.FONT\_HERSHEY\_SIMPLEX,
- cv2.FONT\_HERSHEY\_PLAIN,
- cv2.FONT\_HERSHEY\_DUPLEX,
- cv2.FONT\_HERSHEY\_COMPLEX,
- cv2.FONT\_HERSHEY\_TRIPLEX,
- cv2.FONT\_HERSHEY\_COMPLEX\_SMALL,
- cv2.FONT\_HERSHEY\_SCRIPT\_SIMPLEX,
- cv2.FONT\_HERSHEY\_SCRIPT\_COMPLEX

FontHersheySimplex  
FontHersheyPlain  
FontHersheyDuplex  
FontHersheyComplex  
FontHersheyTriplex  
FontHersheyComplexSmall  
FontHersheyScriptSimplex  
FontHersheyScriptComplex

## onmouse (event, x, y, flags, param)

事件代號 (int event), 座標 (int x,int y), 旗標代號 (int flag), 滑鼠事件的代號名稱 (param)

- event : 代表的是滑鼠回傳的事件號碼，每當滑鼠有動作，event就會回傳訊息到 onMouse()，也順便回傳滑鼠移動的座標
- flag : 代表的是拖曳事件
- param : 則是自己定義 onMouse() 事件的ID，就跟 GUI 介面的視窗介面 ID 一樣 (cvGetWindowHandle())，不過這邊是自己給的編號，而視窗介面的 ID 則是系統自動隨機分配的 ID，而滑鼠事件的執行可以細分為：

## onmouse (event, x, y, flags, param)

| 事件 event                   | 值 | 動作   |
|----------------------------|---|------|
| CV_EVENT_MOUSEMOVE         | 0 | 滑動   |
| CV_EVENT_LBUTTONDOWN       | 1 | 左鍵點擊 |
| CV_EVENT_RBUTTONDOWN       | 2 | 右鍵點擊 |
| CV_EVENT_MBUTTONDOWN       | 3 | 中鍵點擊 |
| CV_EVENT_LBUTTONUP         | 4 | 左鍵放開 |
| CV_EVENT_RBUTTONUP         | 5 | 右鍵放開 |
| CV_EVENT_MBUTTONUP         | 6 | 中鍵放開 |
| CV_EVENT_LBUTTONDOWNDBLCLK | 7 | 左鍵雙擊 |
| CV_EVENT_RBUTTONDOWNDBLCLK | 8 | 右鍵雙擊 |
| CV_EVENT_MBUTTONDOWNDBLCLK | 9 | 中鍵雙擊 |

## onmouse (event, x, y, flags, param)

| 旗標 flag                | 值  | 動作                |
|------------------------|----|-------------------|
| CV_EVENT_FLAG_LBUTTON  | 1  | 左鍵拖曳              |
| CV_EVENT_FLAG_RBUTTON  | 2  | 右鍵拖曳              |
| CV_EVENT_FLAG_MBUTTON  | 4  | 中鍵拖曳              |
| CV_EVENT_FLAG_CTRLKEY  | 8  | (8~15)按Ctrl不放事件   |
| CV_EVENT_FLAG_SHIFTKEY | 16 | (16~31)按Shift不放事件 |
| CV_EVENT_FLAG_ALTHOOK  | 32 | (32~39)按Alt不放事件   |

Python-opencv使用捲軸即時調整圖片的亮度、強度和飽和度, PythonOpenCV, 利用, 調節

- 創建滑動條

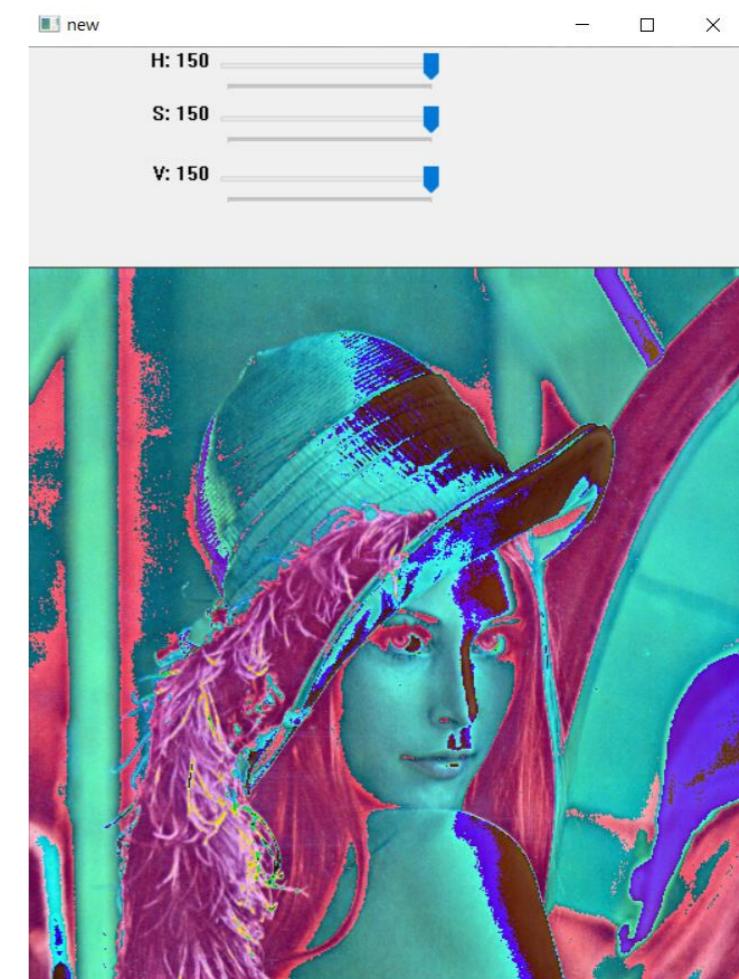
`cv2.createTrackbar(trackbarName, windowName, value, count, onChange)-> None`

- `trackbarName` : 滑動條名稱
- `windowName` : 所在窗口名
- `value` : 初始值
- `count` : 最大值
- `onChange` : 回呼函數名稱
- 返回值 : 無

- 獲取滑動條資料

`cv2.getTrackbar(trackbarname, winname)-> retval`

- `trackbarname` : 滑動條名稱
- `winname` : 所在窗口名
- 返回值 : 捲軸所在位置的值



# Module 3. 色彩空間

3-1: RGB  
3-2: Gray  
3-3: HSV

色彩空間是描述使用一組值（通常使用三個、四個值或者顏色成分）表示顏色方法的抽象數學模型，也被稱作「色域」。

通常使用RGB（紅色、綠色、藍色）色彩空間定義，這是另外一種生成同樣顏色的方法，紅色、綠色、藍色被當作X、Y和Z坐標軸。另外一個生成同樣顏色的方法是使用色相（X軸）、飽和度（色度）（Y軸）和明度（Z軸）表示，這種方法稱為HSV色彩空間。另外還有許多其它的色彩空間，許多可以按照這種方法用三維（X、Y、Z）、更多或者更少維表示，但是有些根本不能用這種方法表示。

RGB色彩空間是RGB色彩空間之一，以單色（單一波長）原色的特定集合著稱

最常用的是24-位實現方法，也就是紅綠藍每個通道有8位元或者256色級。基於這樣的24-位RGB模型的色彩空間可以表現 $256 \times 256 \times 256 \approx 1677$ 萬色。一些實現方法採用每原色16位元，能在相同範圍內實現更高更精確的色彩密度。這在寬域色彩空間中尤其重要，因為大部分通常使用的顏色排列的相對更緊密。

為什麼RGB不夠好!!!

任何的色彩都可由紅、綠、及藍三原色光混合而成，而CRT、LCD顯示及電漿螢幕等，確實都透過套用不同的紅、綠、及藍色次像素（Sub-Pixel）陣列的密度，而產生彩色影像。但事實上，有許多色彩卻是沒辦法以這樣的方法產生。尤其是，沒辦法產生各式各樣的青綠色（Blue-Green） – 可以完美地由攝影機捕捉，並在膠捲重現的色彩。

基礎：對於彩色轉灰度，有一個很著名的心理學公式：

$$\text{Gray} = R * 0.299 + G * 0.587 + B * 0.114$$

## 整數算法

而實際應用時，希望避免低速的浮點運算，所以需要整數算法。注意到係數都是3位精度，我們可以將它們縮放1000倍來實現整數運算算法：

$$\text{Gray} = (R * 299 + G * 587 + B * 114 + 500) / 1000$$

RGB一般是8位精度，現在縮放1000倍，所以上面的運算是32位整型的運算。注意後面那個除法是整數除法，所以需要加上500來實現四捨五入。就是由於該算法需要32位運算，所以該公式的另一個變種很流行：

$$\text{Gray} = (R * 30 + G * 59 + B * 11 + 50) / 100$$



HSV即色相、飽和度、明度（英語：Hue, Saturation, Value），又稱HSB，其中B即英語：Brightness。另外一個生成同樣顏色的方法是使用色相（X軸）、飽和度（色度）（Y軸）和明度（Z軸）表示，這種方法稱為HSV色彩空間

- **色調(Hue)**: 色彩的顏色名稱，如紅色、黃色等

在HSL中，色調(Hue)決定了的顏色，用360度來劃分，就像傳統的色輪。

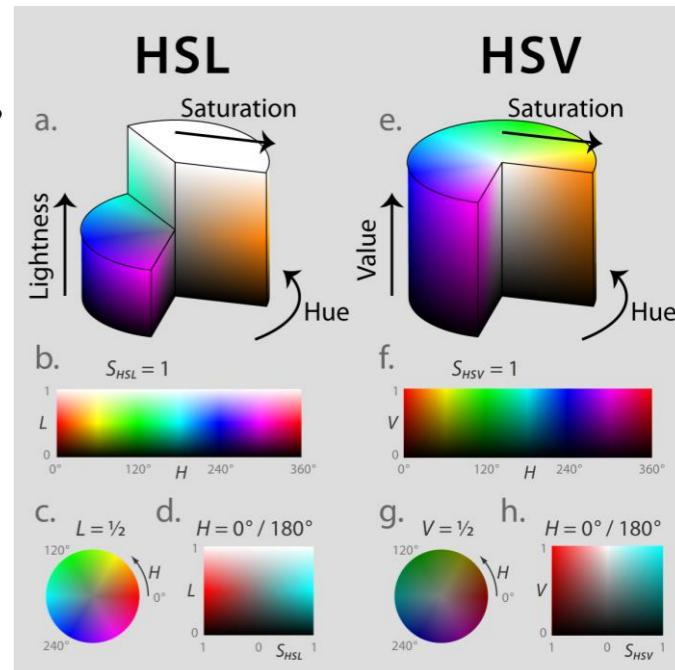
HSL相對於RGB顏色的主要優勢之一是互補色彼此相對，這使得整個系統非常直觀。

- **飽和度(Saturation)**: 色彩的純度，越高色彩越純，低則逐漸變灰，數值為0-100%

與色輪中心的距離稱為"飽和度"。仔細觀察上面的色輪，距離圓心越遠，顏色更明亮，更鮮豔。

- **明度 (Value)**，亮度(Lightness)：數值為0-100%。

HSL即色相、飽和度、亮度（英語：Hue, Saturation, Lightness）。



## RGB to XYZ

RGB轉換至CIE XYZ的公式如下：

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

## Lab

- L – Lightness ( Intensity ).
- a – color component ranging from Green to Magenta.
- b – color component ranging from Blue to Yellow

Lab顏色空間與RGB顏色空間完全不同。在RGB顏色空間中，顏色信息分為三個通道，但是相同的三個通道也對亮度信息進行編碼。另一方面，在Lab顏色空間中，L通道與顏色信息無關，並且僅編碼亮度。其他兩個通道對顏色進行編碼。它具有以下屬性。

- 感知均勻的色彩空間，近似我們對色彩的感知方式。
- 與設備無關（捕獲或顯示）。
- 在Adobe Photoshop中廣泛使用。
- 通過複雜的轉換方程與RGB顏色空間相關。

# Module 4. 圖片幾何轉換

- 4-1: 線性代數複習
- 4-2: 圖片幾何轉換介紹
- 4-3: OpenCV圖片幾何  
轉換

## 矩陣相乘 dot

$$\begin{bmatrix} 4 & 7 \\ 2 & 6 \end{bmatrix} \begin{bmatrix} 0.6 & -0.7 \\ -0.2 & 0.4 \end{bmatrix} = \begin{bmatrix} 4 \times 0.6 + 7 \times -0.2 & 4 \times -0.7 + 7 \times 0.4 \\ 2 \times 0.6 + 6 \times -0.2 & 2 \times -0.7 + 6 \times 0.4 \end{bmatrix}$$
$$= \begin{bmatrix} 2.4 - 1.4 & -2.8 + 2.8 \\ 1.2 - 1.2 & -1.4 + 2.4 \end{bmatrix}$$
$$= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

## 矩陣轉置 transpose

Transposing a 2x3 matrix to create a 3x2 matrix

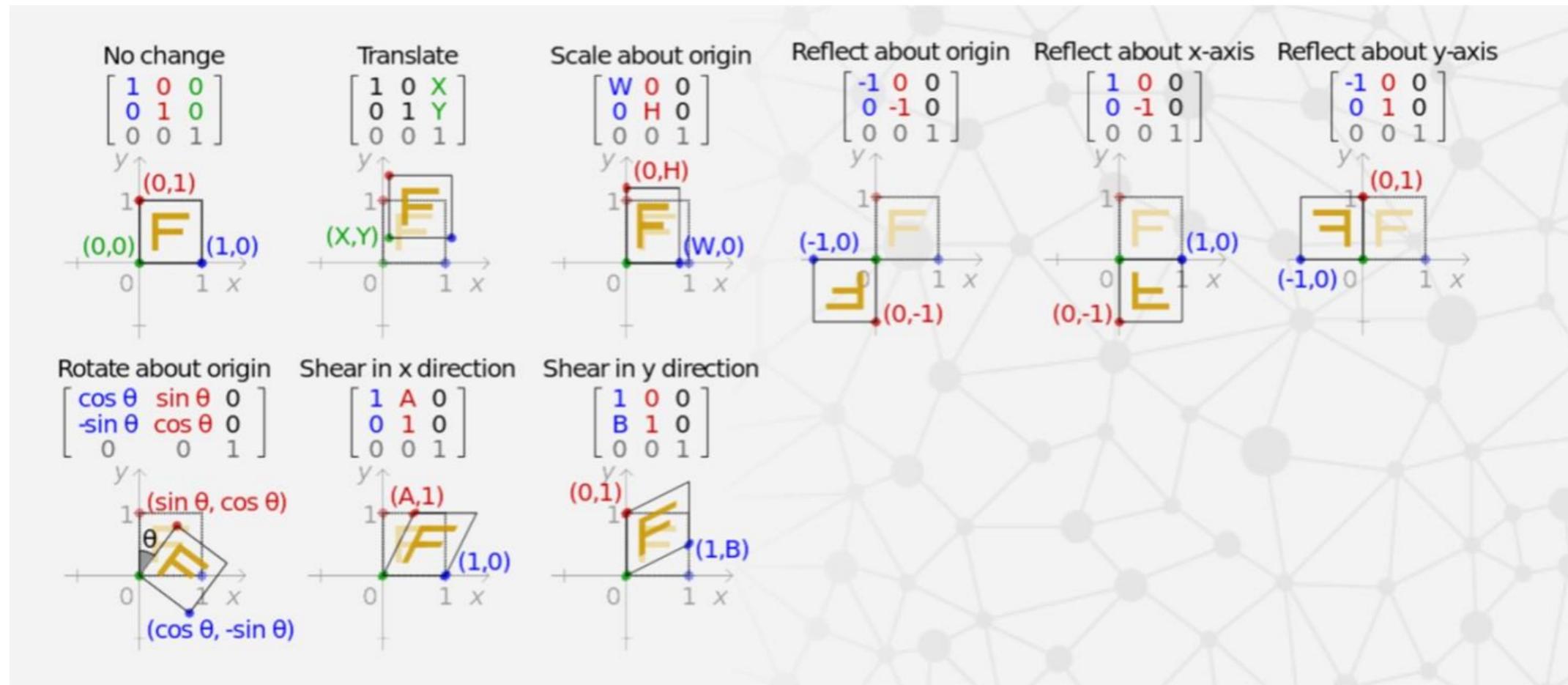
$$\begin{bmatrix} 6 & 4 & 24 \\ 1 & -9 & 8 \end{bmatrix}^\top = \begin{bmatrix} 6 & 1 \\ 4 & -9 \\ 24 & 8 \end{bmatrix}$$

座標位置 row, column 轉換 (非 RGB 轉換) :

- 縮放 (resize)
- 翻轉 (flip)
- 平移 (translate)
- 旋轉 (Rotate)
- 仿射 (Affine)

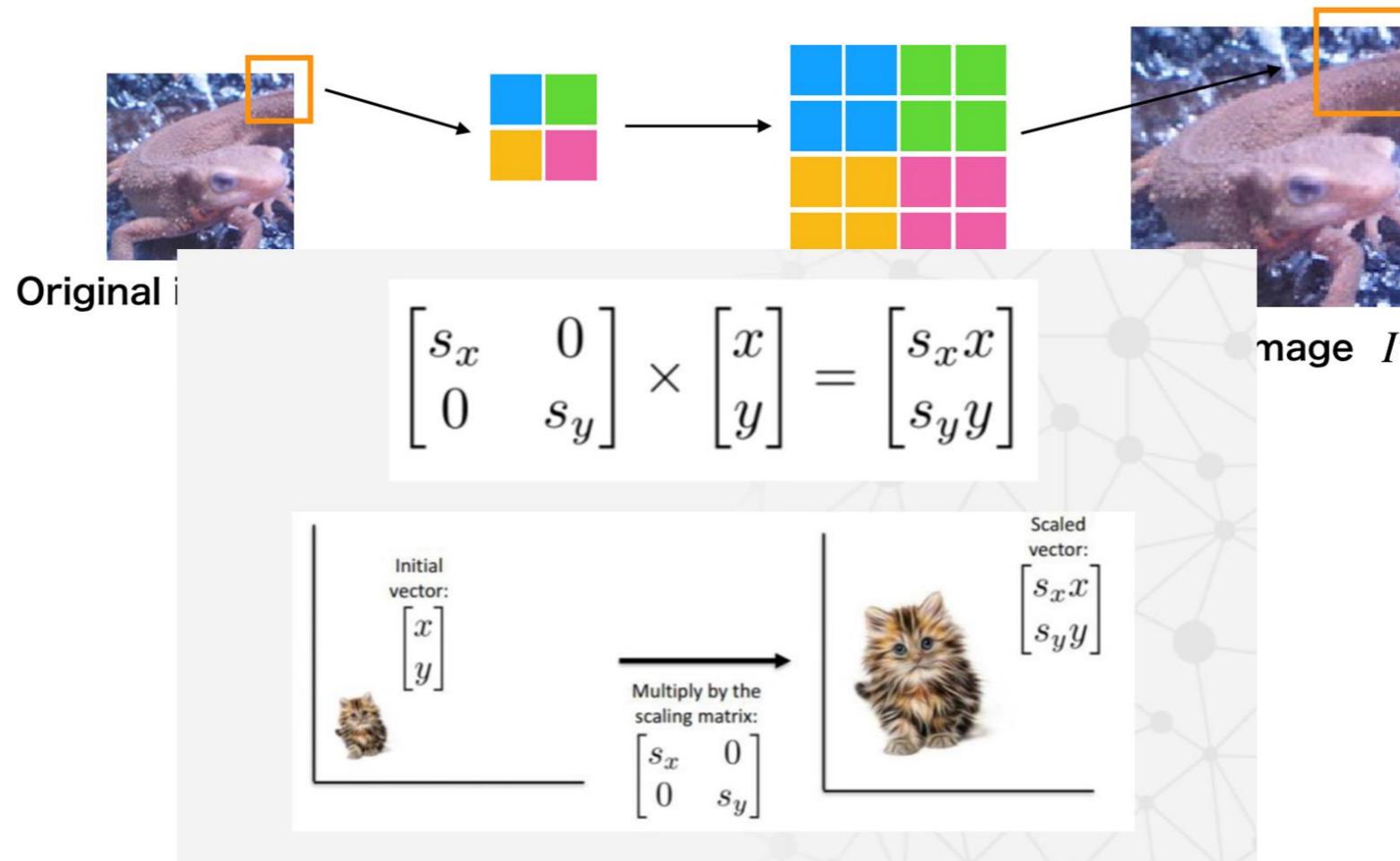
## 4-2: 圖片幾何轉換介紹

## 轉換矩陣 transformMatrix



## 4-2: 圖片幾何轉換介紹

## 縮放 (resize)

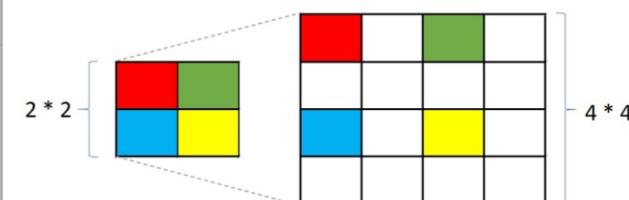


## 4-3: OpenCV圖片幾何轉換

**Scaling - resize()** : 縮放只是調整影像大小

```
cv2.resize(img, (250, 200), , interpolation=cv2.INTER_AREA)
```

| 插值方式           | 名稱        | 說明   |
|----------------|-----------|--|
| INTER_NEAREST  | 最近鄰插值     | 邊緣不會出現緩慢的漸慢過度區域，這也導致放大的圖像容易出現鋸齒的現像                                     |
| INTER_LINEAR   | 線性插值 (預設) | 線性插值是以距離為權重的一種插值方式。  |
| INTER_AREA     | 區域插值      | 使用圖元區域關係進行重採樣。它可能是圖像抽取的首選方法，因為它會產生無雲紋理的結果。但是當圖像縮放時，它類似於INTER_NEAREST方法 |
| INTER_CUBIC    | 三次樣條插值    | 三次樣條插值，可以有效避免出現鋸齒的現像，4x4圖元鄰域的雙三次插值                                     |
| INTER_LANCZOS4 | Lanczos插值 | 8x8圖元鄰域的Lanczos插值  |



## Flip

```
cv2.flip(src, flipCode)
```

src : 原始影像。

flipCode : 翻轉方向。

flipCode = 0 , 則以 X (水平) 軸為對稱軸翻轉

flipCode > 0 , 則以 Y (垂直) 軸為對稱軸翻轉

flipCode < 0 , 則在 X (水平) 軸、Y (垂直) 軸方向同時翻轉

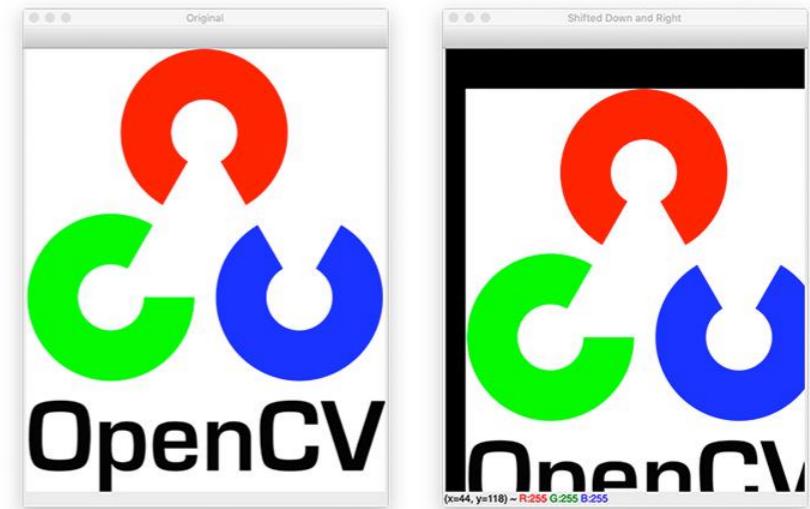


## Translation

`cv2.warpAffine(src, dst, transformMatrix, size)`

在影像平移中我們會用到前三個引數：

- `src` 對像，表示此操作的源（輸入圖像）。
- `dst` 表示此操作的目標（輸出圖像）的對像。
- 表示轉換矩陣的 `transformMatrix` 對像。
- `size`-整數類型的變量，表示輸出圖像的大小

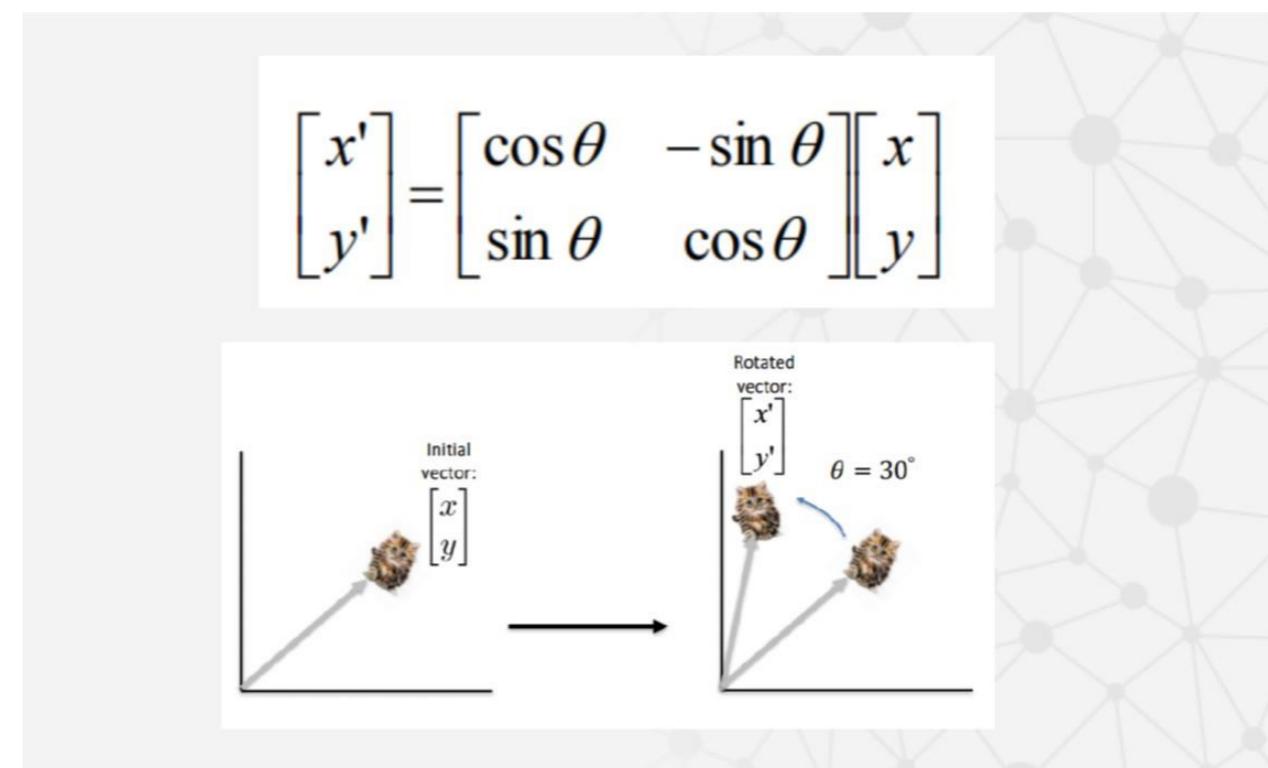


## Rotation

`cv2.getRotationMatrix2D((w/2, h/2), 45, 0.8)`

計算出一個二維旋轉的仿射矩陣

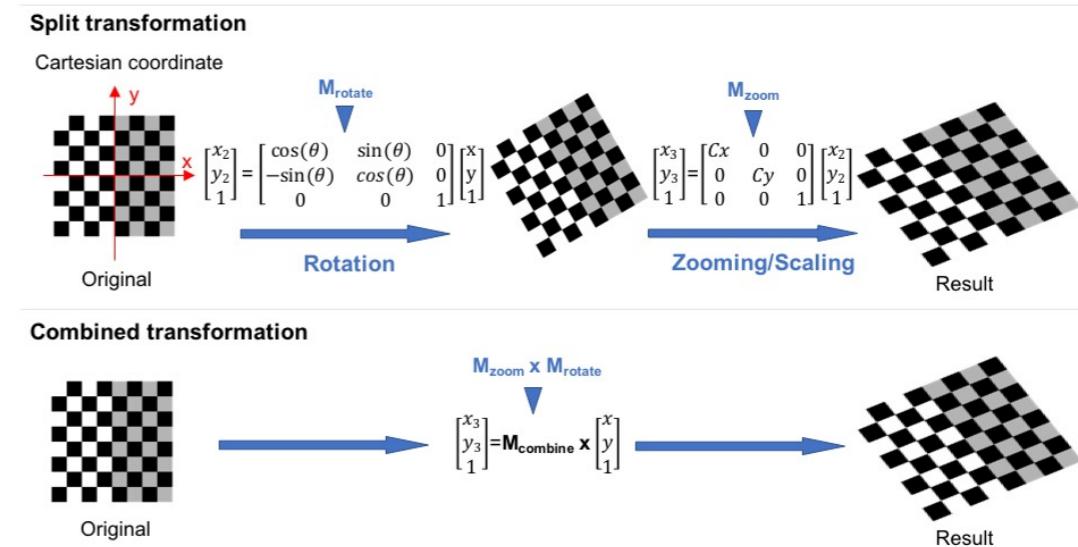
- `center` : 旋轉中心座標
- `angle` : 旋轉角度，正值意味著逆時針旋轉，座標原點為左上角
- `scale` : 縮放比例



## Affine Transformation

在仿射變換中，原始影像中的所有平行線在輸出影像中仍然是平行的(平行四邊形的概念)。為了找到變換矩陣，我們需要從輸入影像中得到三個點，以及它們在輸出影像中的對應位置。然後 `cv.getAffineTransform` 將會建立一個 $2 \times 3$ 矩陣，它將被傳遞給 `cv.warpAffine`。

```
M = cv2.getAffineTransform(pts1, pts2)  
cv2.warpAffine(img, M, (cols,rows))
```



## 透視變換

要完成透視變換，你需要一個  $3 \times 3$  的對映矩陣，直線會在對映之後保持筆直。要找到這個對映矩陣，你需要四個原圖上的點，以及它們在轉換後圖像上對應的位置。在這四個點中，其中任意三個不能共線。

```
M = cv2.getPerspectiveTransform(pts1,pts2)  
affine_img = cv2.warpPerspective(img, M, (w, h))
```

# Module 5. 濾波器

- 5-1: 卷積運算介紹
- 5-2: 卷積運算物理意義
- 5-3: 平均濾波器、高斯濾波器、中值濾波器、雙邊濾波器

卷積一詞最開始出現在信號與系統中，是指兩個原函數產生一個新的函數的一種算子。卷積運算在運算過程可以概括為翻轉、平移再加權求和三個步驟，其中的加權求和就是乘加操作。另外，卷積運算還有一個重要的特性：空間域卷積=頻域乘積，這一點可以解釋為什麼卷積運算可以自動地提取圖像的特徵。

在卷積神經網絡中，對數字圖像做卷積操作其實就是利用卷積核（黃底部分）在圖像（綠底部分）上滑動，將圖像上的像素灰度值與對應卷積核上的數值相乘，然後將所有相乘後的值相加作為此時的輸出值（紅底部分），並最終滑動遍歷完整副圖像的過程。

|   |   |   |   |   |
|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |

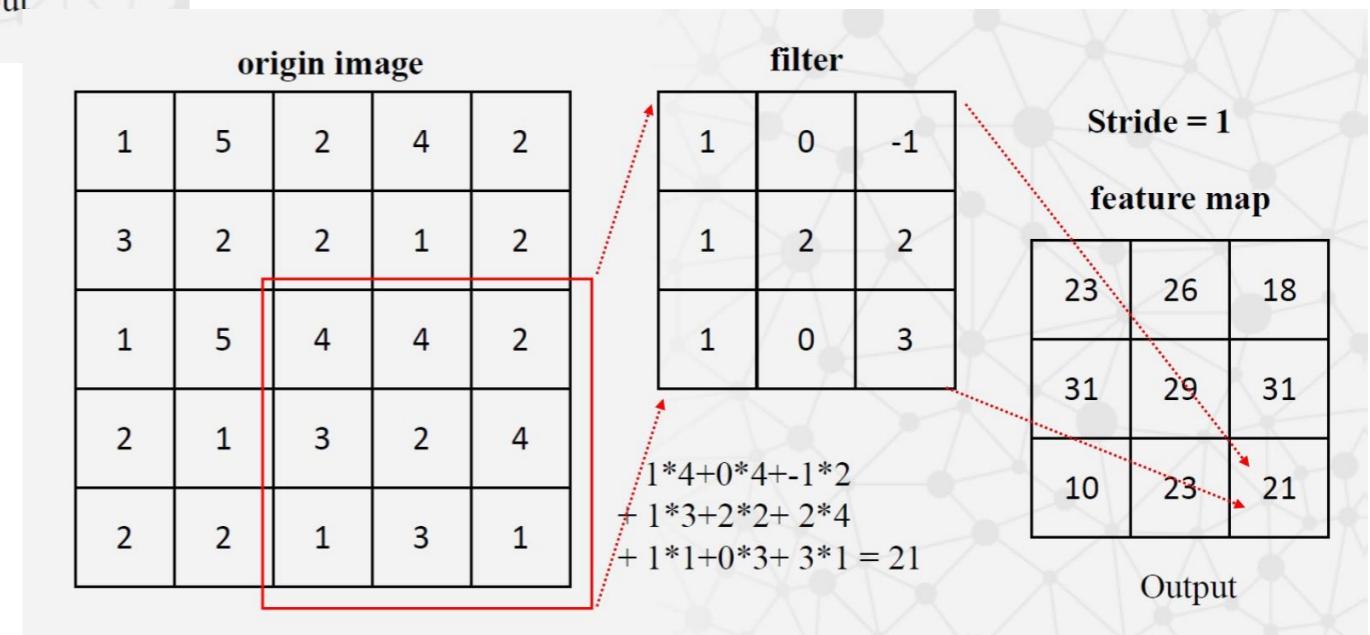
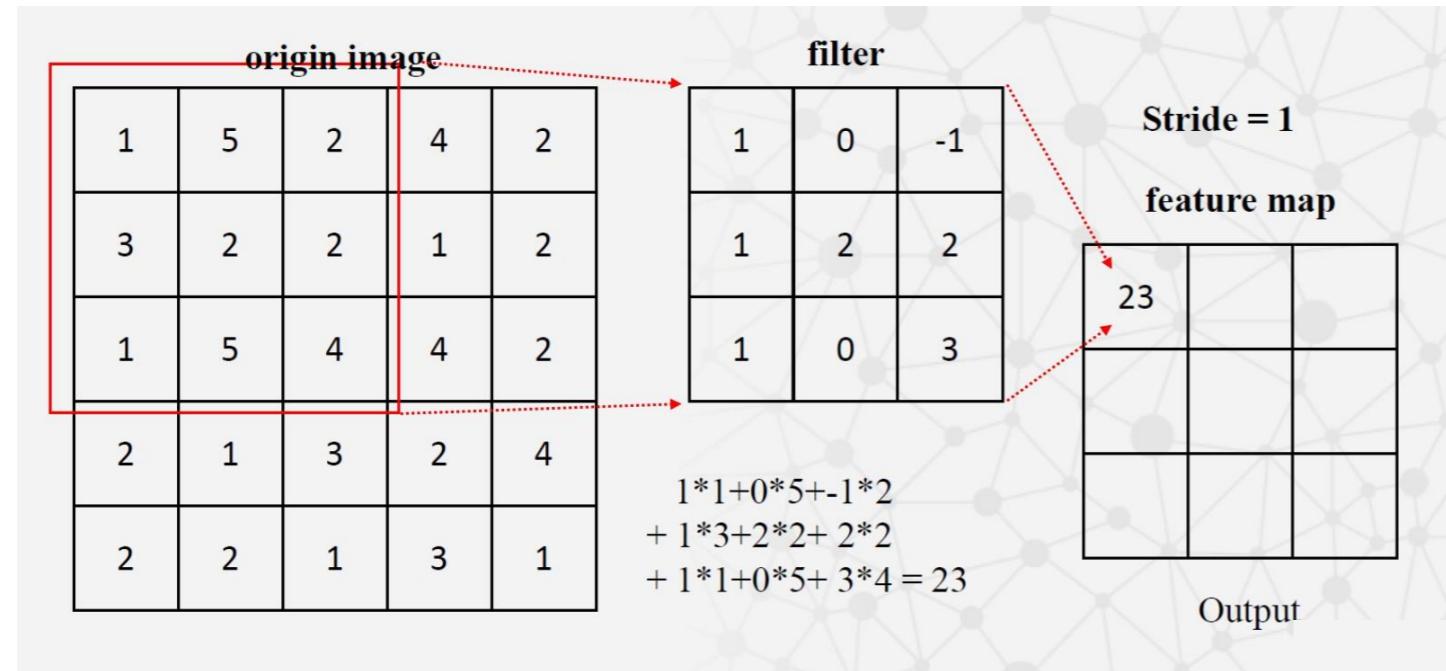
Image

|   |  |  |
|---|--|--|
| 4 |  |  |
|   |  |  |
|   |  |  |
|   |  |  |

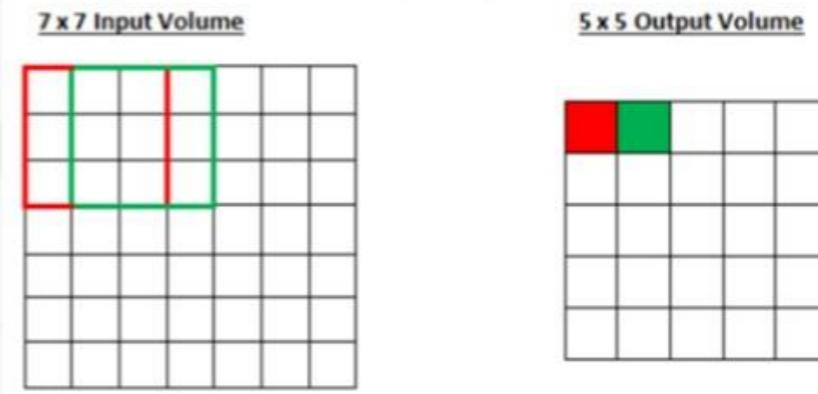
Convolved Feature

原文網址：<https://kknews.cc/tech/6322omp.html>

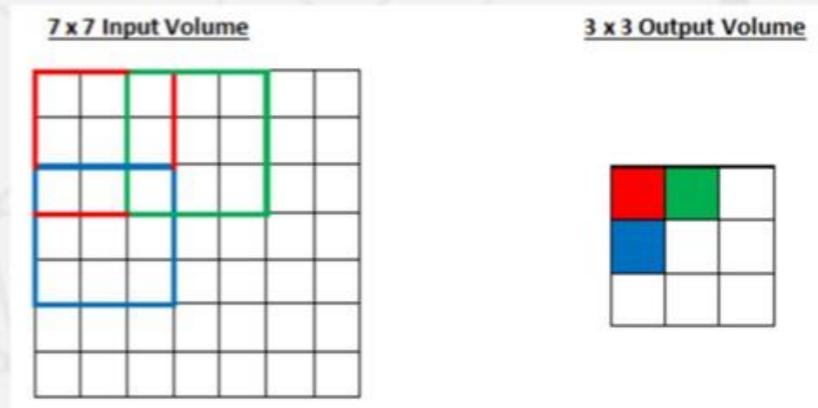
# 5-1: 卷積運算介紹



- Filter size
- Stride
  - Amount of filter shift
- Padding
  - Padding zero on image boundary
  - Remain the same size after convolution



Size = 3\*3 stride = 1

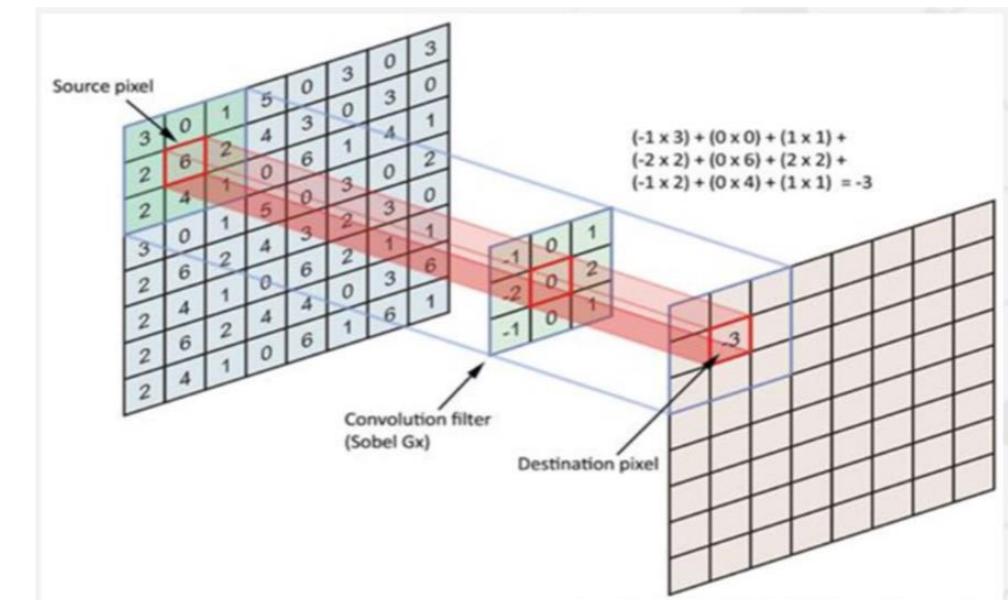


Size = 3\*3 stride = 2

## 5-2: 卷積運算物理意義

我們計算系統輸出時就必須考慮現在時刻的信號輸入的響應以及之前若干時刻信號輸入的響應之「殘留」影響的一個疊加效果。再拓展點，某時刻的系統響應往往不一定是由當前時刻和前一時刻這兩個響應決定的，也可能是再加上前前時刻，前前前時刻，前前前前時刻，等等

影像平滑模糊化是透過使用低通濾波器進行影像卷積來實現的。這對於消除雜訊很有用。實際上使用此濾波器時，它會從影像中去除高頻內容（例如，雜訊，邊緣），也會導致影像邊緣變得模糊（也有其他濾波器不會造成影像邊緣模糊）。OpenCV主要提供四種類型的平滑模糊化技術



## 5-3: 平均濾波器、高斯濾波器、 中值濾波器、雙邊濾波器

### Blur & boxFilter - Average & Sum

boxFilter()函數方框濾波所用的核為：

- 當flow algorithms 中用到的圖像倒數的協方差矩陣（ normalize = true 時，盒式濾波就變成了均值濾波。也就是說，均值濾波是盒式濾波歸一化（ normalized ）後的特殊情況。其中，歸一化就是把要處理的量都縮放到一個範圍內，比如(0,1)，以便統一處理和直觀量化。
- 當normalize = false 時，為非歸一化的盒式濾波，用於計算每個圖元鄰域內的積分特性，比如密集光流演算法（ dense optical covariance matrices of image derivatives ）。

$$\text{kernel} = \alpha \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix} \quad \text{where } \alpha = \begin{cases} \frac{1}{\text{width} \times \text{height}} & \text{normalize == True} \\ 1 & \text{otherwise} \end{cases}$$

## 5-3: 平均濾波器、高斯濾波器、 中值濾波器、雙邊濾波器

### Define other Filter

#### 2D Convolution ( Image Filtering )

filter2D()函數

```
dst = cv.filter2D (src, ddepth, kernel[, dst[, anchor[, delta[, borderType]]]])
```

OpenCV provides a function `cv.filter2D()` to convolve a kernel with an image. As an example, we will try an averaging filter kernel will look like the below:

$$K = \frac{1}{25} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

The operation works like this: keep this kernel above a pixel, add all the 25 pixels below this kernel, take the average, and replace the central pixel with the new average value. This operation is continued for all the pixels in the image. Try this code and check the result:

## 5-3: 平均濾波器、高斯濾波器、 中值濾波器、雙邊濾波器

### filter2D()函數

```
dst = cv.filter2D (src, ddepth, kernel[, dst[, anchor[, delta[, borderType]]]])
```

| 參數     | 描述  |
|--------|---|
| src    | 原圖像   |
| dst    | 目標圖像，與原圖像尺寸和通道數相同   |
| ddepth | 目標圖像的所需深度，當ddepth輸入值為-1時，靶心圖表像和原圖像深度保持一致                            |
| kernel | 卷積核（或相當於相關核），單通道浮點矩陣；如果要將不同的內核應用於不同的通道，請使用拆分將圖像拆分為單獨的顏色平面，然後單獨處理它們。 |
| anchor | 內核的錨點，指示內核中過濾點的相對位置；錨應位於內核中；默認值（-1，-1）表示錨位於內核中心。                    |
| delta  | 在將它們存儲在dst中之前，將可選值添加到已過濾的像素中。類似於偏置。                                 |

## 5-3: 平均濾波器、高斯濾波器、 中值濾波器、雙邊濾波器

其中 `ddepth` 表示目標圖像的所需深度，它包含有關圖像中存儲的數據類型的信息，可以是 `unsigned char ( CV_8U )`，`signed char ( CV_8S )`，`unsigned short ( CV_16U )` 等等...

| Input depth (src.depth())  | Output depth (ddepth)                |
|----------------------------|--------------------------------------|
| <code>CV_8U</code>         | <code>-1/CV_16S/CV_32F/CV_64F</code> |
| <code>CV_16U/CV_16S</code> | <code>-1/CV_32F/CV_64F</code>        |
| <code>CV_32F</code>        | <code>-1/CV_32F/CV_64F</code>        |
| <code>CV_64F</code>        | <code>-1/CV_64F</code>               |

Note : 當 `ddepth = -1` 時，表示輸出圖像與原圖像有相同的深度。

## 5-3: 平均濾波器、高斯濾波器、中值濾波器、雙邊濾波器

### Boundary Padding

- BORDER\_REPLICATE**：複製法，也就是複製最邊緣像素。
- BORDER\_REFLECT**：反射法，對感興趣的圖像中的像素在兩邊進行複製例如：  
`fedcba|abcdefg|hgfedcb`
- BORDER\_REFLECT\_101**：反射法，也就是以最邊緣像素為軸，對稱，  
`gfedcb|abcdefg|gfedcba`
- BORDER\_WRAP**：外包裝法  
`cdefgh|abcdefg|abcdefg`
- BORDER\_CONSTANT**：常量法，常數值填充。

| Enumerator   |   |
|--|---|
| <b>BORDER_CONSTANT</b><br>Python: cv.BORDER_CONSTANT       | <code>iiiiii abcdefg iiiiii</code> with some specified <code>i</code> |
| <b>BORDER_REPLICATE</b><br>Python: cv.BORDER_REPLICATE     | <code>aaaaaa abcdefg hhhhhh</code>                                    |
| <b>BORDER_REFLECT</b><br>Python: cv.BORDER_REFLECT         | <code>fedcba abcdefg hgfedcb</code>                                   |
| <b>BORDER_WRAP</b><br>Python: cv.BORDER_WRAP               | <code>cdefgh abcdefg abcdefg</code>                                   |
| <b>BORDER_REFLECT_101</b><br>Python: cv.BORDER_REFLECT_101 | <code>gfedcb abcdefg gfedcba</code>                                   |
| <b>BORDER_TRANSPARENT</b><br>Python: cv.BORDER_TRANSPARENT | <code>uvwxyz abcdefg ijklmno</code>                                   |
| <b>BORDER_REFLECT101</b><br>Python: cv.BORDER_REFLECT101   | same as BORDER_REFLECT_101  |
| <b>BORDER_DEFAULT</b><br>Python: cv.BORDER_DEFAULT         | same as BORDER_REFLECT_101  |
| <b>BORDER_ISOLATED</b><br>Python: cv.BORDER_ISOLATED       | do not look outside of ROI  |

<https://blog>

## 5-3: 平均濾波器、高斯濾波器、 中值濾波器、雙邊濾波器

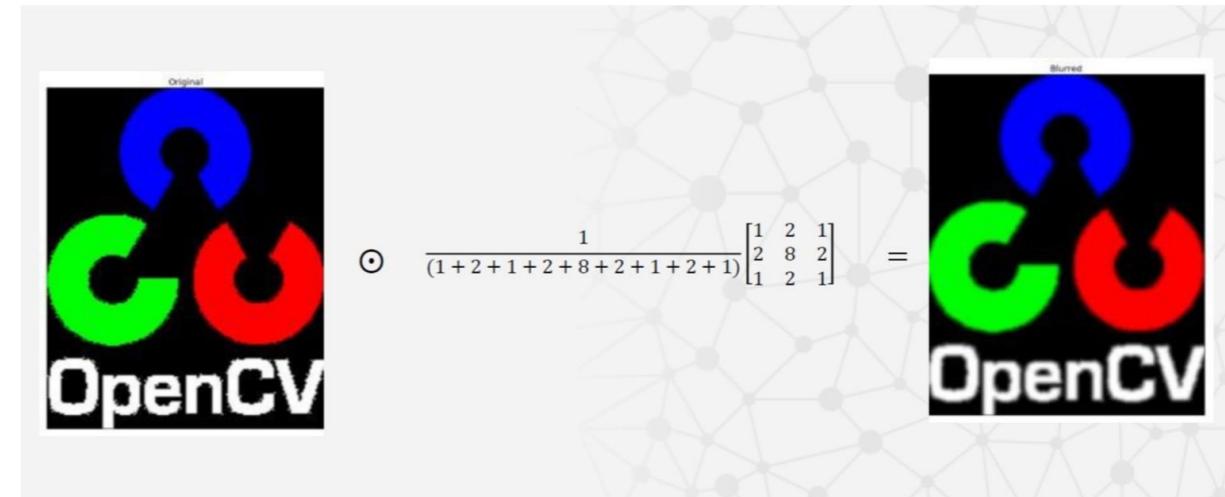
### Gaussian Filter

它的運作方式與 Averaging Filter 類似，但差別在於中間那個點的計算方式不同，Gaussian Filter 的作法是將給予各點不同的權值，愈靠近中央點的權值愈高，最後再以平均方式計算出中央點，因此，Gaussian Filter 的模糊化效果比起 Averaging 會比較明顯，但是效果卻更為自然。

`GaussianBlur(src, ksize, sigmaX, sigmaY=None, borderType=None)`

$$\text{SigmaX} = 0.3 * [(\text{ksize.width} - 1) * 0.5 - 1] + 0.8$$

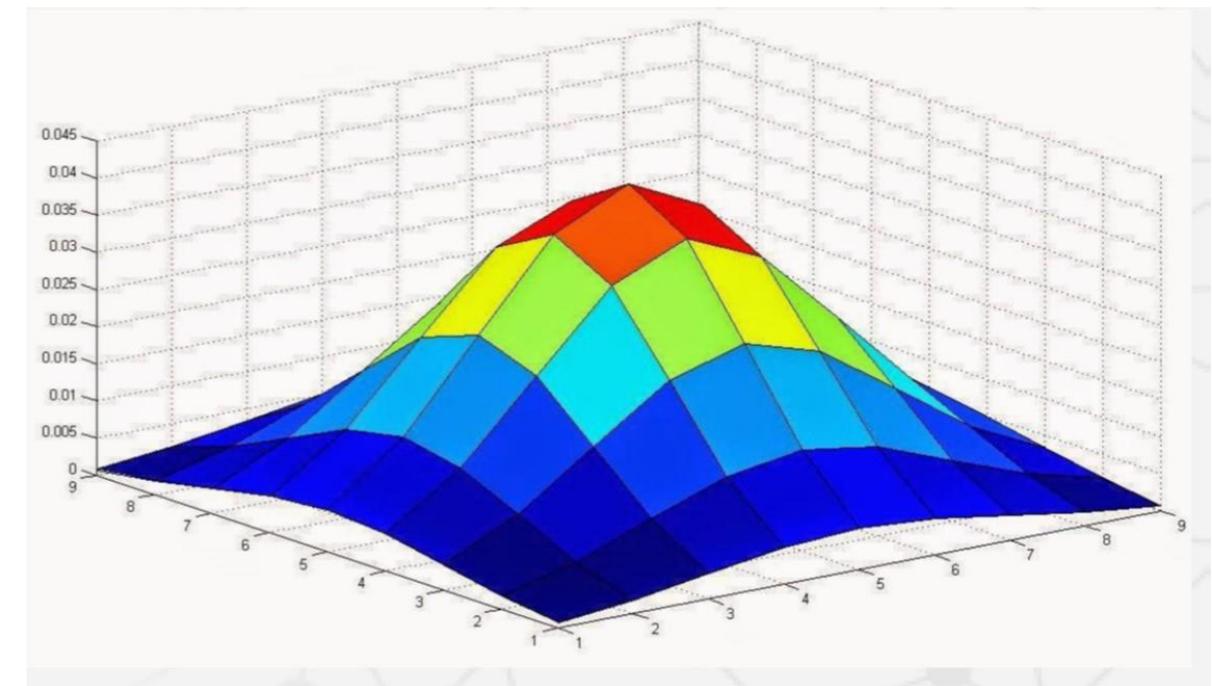
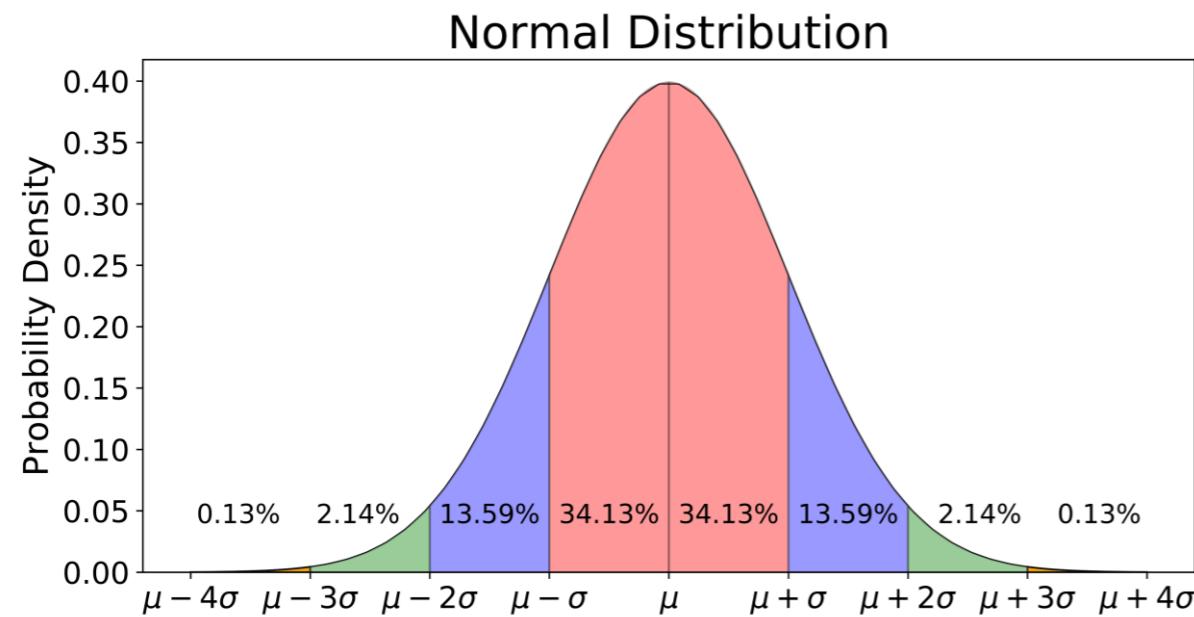
$$\text{SigmaY} = 0.3 * [(\text{ksize.width} - 1) * 0.5 - 1] + 0.8$$



`SigmaY` : 垂直方向的標準差預設為 0 表示與水平方向相同.

## 5-3: 平均濾波器、高斯濾波器、 中值濾波器、雙邊濾波器

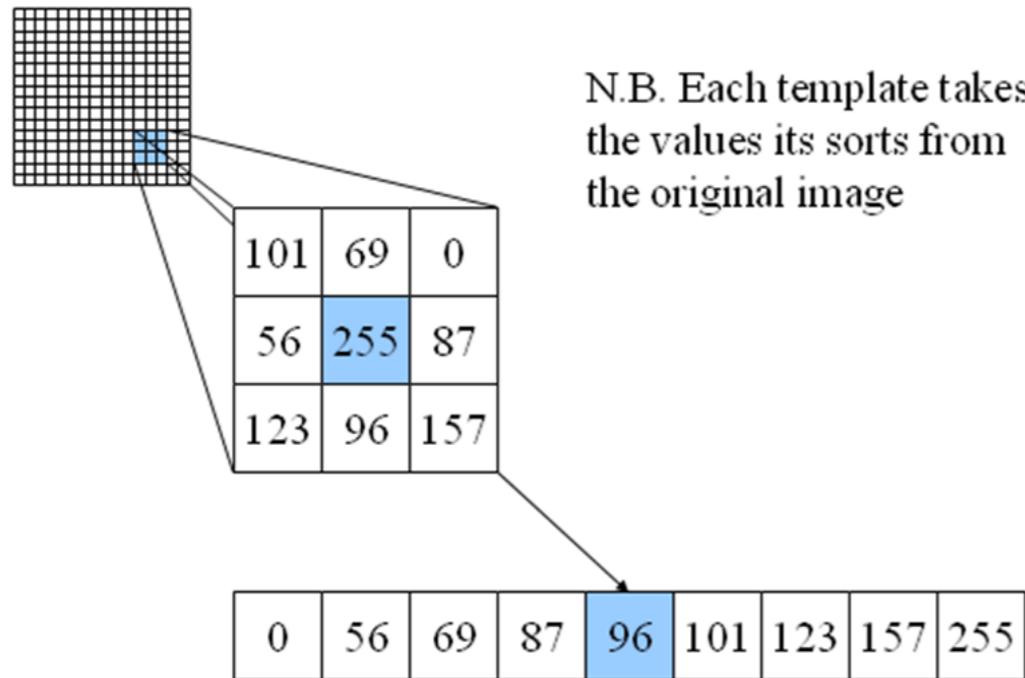
高斯分佈(也叫常態分佈)的特點為,標準差越大,分佈越分散,標準差越小,分佈越集中



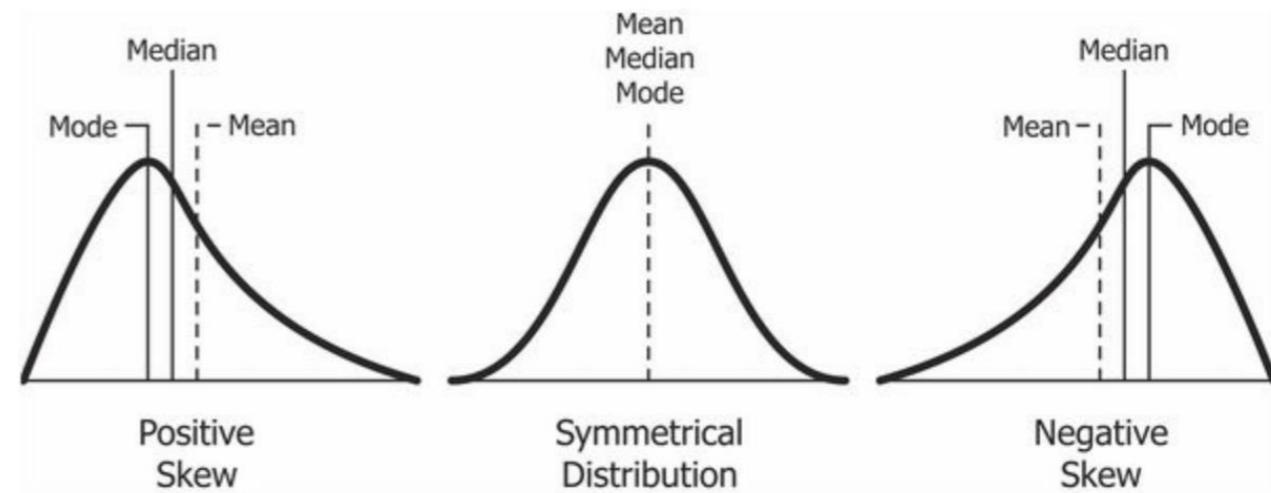
## 5-3: 平均濾波器、高斯濾波器、 中值濾波器、雙邊濾波器

### Median Filter 中位數

計算內核窗口下所有像素的中位數，並將中心像素替換為該中位數而不是平均值



### Mean, mode vs. Median



## 5-3: 平均濾波器、高斯濾波器、 中值濾波器、雙邊濾波器

### Bilateral Filter

```
cv2.bilateralFilter( src, d, sigmaColor, sigmaSpace[, dst[, borderType]])
```

雙邊濾波能在保持邊界清晰的情況下有效的去除噪音。但是這種操作與其他濾波器相比會比較慢，我們知道高斯濾波器是求中心點鄰近區域畫素的高斯加權平均值。這種高斯濾波器只考慮畫素之間的空間關係，而不會考慮畫素值之間的關係（畫素的相似度）。所以這種方法不會考慮一個畫素是否位於邊界。因此邊界也會被模糊掉。

雙邊濾波在同時使用“空間高斯權重”和“灰度值相似性高斯權重”。空間高斯函式確保只有鄰近區域的畫素對中心點有影響，灰度值相似性高斯函式確保只有“與中心畫素灰度值相近的才會被用來做模糊運算”。所以這種方法會確保邊界不會被模糊掉，因為邊界處的灰度值變化比較大。

簡單說就是，在生成周邊畫素的權重矩陣時，“如果發現旁邊的畫素值和當前的畫素值差異很大，就只給差異很大的那個元素分配很小的權重，這樣“大的突變差異就被保留了””。

## 5-3: 平均濾波器、高斯濾波器、 中值濾波器、雙邊濾波器

### Bilateral Filter

```
cv2.bilateralFilter( src, d, σ_Color, σ_Space[, dst[, borderType]])
```

此種方法的好處是，它不但擁有 Median filter 的除噪效果，又能保留圖片中的不同物件的邊緣(其它三種方式均會造成邊緣同時被模糊化)，但缺點是，Bilateral Filter 執行的效率較差，運算需要的時間較長。

```
cv2.bilateralFilter( src, d, σ_Color, σ_Space[, dst[, borderType]])
```

- src : 影象矩陣
- d : 鄰域直徑
- sigmaColor : 顏色標準差，愈大代表在計算時需要考慮更多的顏色
- sigmaSpace : 空間標準差，這個參數與Guassian filter使用的相同，數值越大，代表越遠的像素有較大的權值。

# Module 6. 設定值處理

- 6-1:什麼是 Threshold  
二值化處理
- 6-2:自我調節設定
- 6-3: otsu 處理

## 6-1: 什麼是 Threshold 二值化處理

### Thresh 門檻, 闕

threshold 是門檻值的意思，OpenCV 提供的 threshold 工具包裡面有影像門檻值的功能，當畫素值高於門檻值時，我們給這個畫素賦予一個新值（可能是白色），否則我們給它賦予另一種顏色（也許是黑色）。這個函式就是 cv2.threshold()

圖像的二值化就是將圖像上的圖元點的灰度值設置為0或255，這樣將使整個圖像呈現出明顯的黑白效果。在數位影像處理中，二值圖像佔有非常重要的地位，圖像的二值化使圖像中資料量大為減少，從而能凸顯出目標的輪廓。

`ret, dst = cv2.threshold(src, thresh, maxval, type)`

`ret` = `thresh`

`dst` : 輸出圖,

`src` : 輸入圖 , 只能輸入單通道影像 , 通常來說為灰度圖

`thresh` : 閾, 門檻值

`maxval` : 當畫素值超過了門檻值 ( 或者小於門檻值 , 根據 `type` 來決定 ) , 所賦予的值

`type` :

| 語法 <code>type</code>           | 值 | 說明                                     |
|--------------------------------|---|--|
| <code>THRESH_BINARY</code>     | 0 | 即二值化 , 將大於門檻值的灰度值設為最大灰度值 , 小於門檻值的值設為 0 |
| <code>THRESH_BINARY_INV</code> | 1 | 將大於門檻值的灰度值設為 0 , 其他值設為最大灰度值            |
| <code>THRESH_TRUNC</code>      | 2 | 將大於門檻值的灰度值設為門檻值 , 小於門檻值的值保持不變。(灰黑)     |
| <code>THRESH_TOZERO</code>     | 3 | 將小於門檻值的灰度值設為 0 , 大於門檻值的值保持不變。(黑灰白對比)   |
| <code>THRESH_TOZERO_INV</code> | 4 | 將大於門檻值的灰度值設為 0 , 小於門檻值的值保持不變。(黑灰強烈)    |

# 6-1:什麼是 Threshold 二值化處理

- **THRESH\_BINARY**

$$dst(x, y) = \begin{cases} \text{maxval} & \text{if } \text{src}(x, y) > \text{thresh} \\ 0 & \text{otherwise} \end{cases}$$

- **THRESH\_BINARY\_INV**

$$dst(x, y) = \begin{cases} 0 & \text{if } \text{src}(x, y) > \text{thresh} \\ \text{maxval} & \text{otherwise} \end{cases}$$

- **THRESH\_TRUNC**

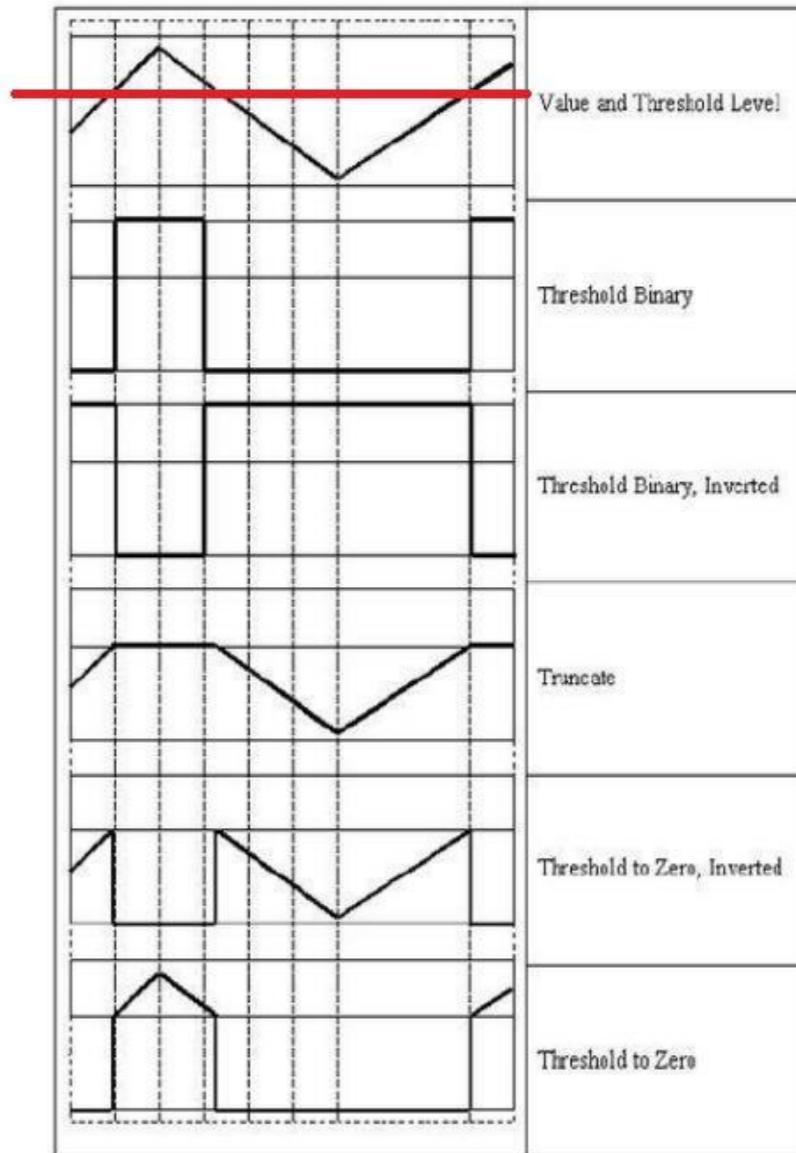
$$dst(x, y) = \begin{cases} \text{threshold} & \text{if } \text{src}(x, y) > \text{thresh} \\ \text{src}(x, y) & \text{otherwise} \end{cases}$$

- **THRESH\_TOZERO**

$$dst(x, y) = \begin{cases} \text{src}(x, y) & \text{if } \text{src}(x, y) > \text{thresh} \\ 0 & \text{otherwise} \end{cases}$$

- **THRESH\_TOZERO\_INV**

$$dst(x, y) = \begin{cases} 0 & \text{if } \text{src}(x, y) > \text{thresh} \\ \text{src}(x, y) & \text{otherwise} \end{cases}$$



## threshold adaptive \*\*局部\*\* 自我調節設定

自我調整閾值二值化函數根據圖片一小塊區域的值來計算對應區域的門檻, 閾值, 從而得到也許更為合適的圖片。

`thresh_type` : 閾值的計算方法, 包含以下2種類型 :

`cv2.ADAPTIVE_THRESH_MEAN_C`: 鄰域面積的平均值

`cv2.ADAPTIVE_THRESH_GAUSSIAN_C`: 高斯窗口的鄰域值的加權和

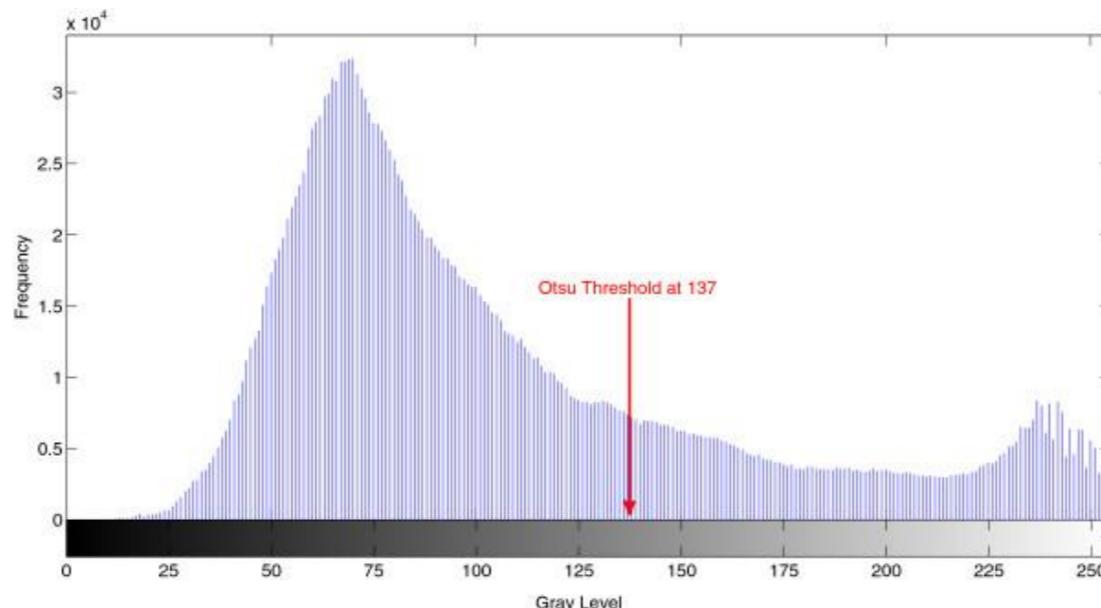
`Block Size` : 圖片中分塊的大小

`C` : 閾值計算方法中的常數項src-類的對像表示源(輸入)圖像。`offset ( thresh - c )`

threshold otsu : 是一種自動門檻值決定法則

Otsu 過程 :

1. 計算圖像長條圖；
2. 設定一閾值，把長條圖強度大於閾值的圖元分成一組，把小於閾值的圖元分成另外一組；
3. 分別計算兩組內的偏移數，並把偏移數相加；
4. 把  $0 \sim 255$  依照順序多為閾值，重複 1-3 的步驟，直到得到最小偏移數，其所對應的值即為結果閾值。



# Module 7. 邊緣檢測

- 7-1: 什麼是邊緣檢測
- 7-2: Sobel、Scharr、Laplacian
- 7-3: Canny

## 邊緣檢測

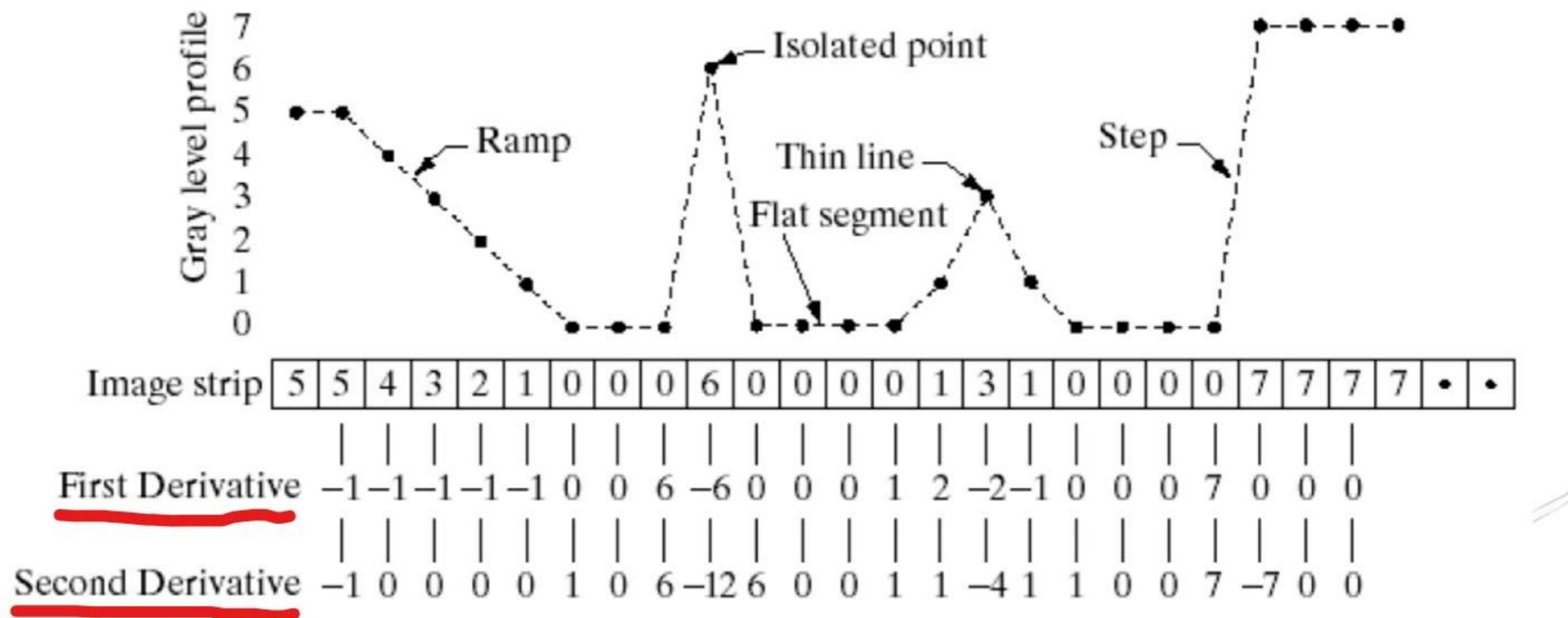
Edge detection 邊緣偵測是 Computer vision 中最重要的步驟，它讓電腦能準確的抓取圖中的物體，這項技術運用到相當複雜的數學運算，以檢查影像中各像素點的顏色變化程度來區分邊界。有了邊緣之後，這些交錯的線段中會有所謂的輪廓，而這也是電腦取得影像中物件的依據。

openCV 提供三種邊緣檢測方式來處理 \*\*Laplacian、Sobel 及 Canny\*\*，這些技術皆是使用 \*\*灰階\*\* 的影像，基於每個像素灰度的不同，利用不同物體在其邊界處會\*\*有明顯的邊緣特徵來分辨\*\*。這三種方法皆使用了一維甚至於二維的微分，嚴格來說，若依其使用技術原理的不同可分為兩種：

而 Sobel 和 Canny 使用的則是 Gradient methods ( 梯度原理 )，它是透過計算像素光度的一階導數差異 ( detect changes in the first derivative of intensity ) 來進行邊緣檢測。

Laplacian 原稱為 Laplacian method，透過計算零交越點上光度的二階導數 ( detect zero crossings of the second derivative on intensity changes )

## 離散導數

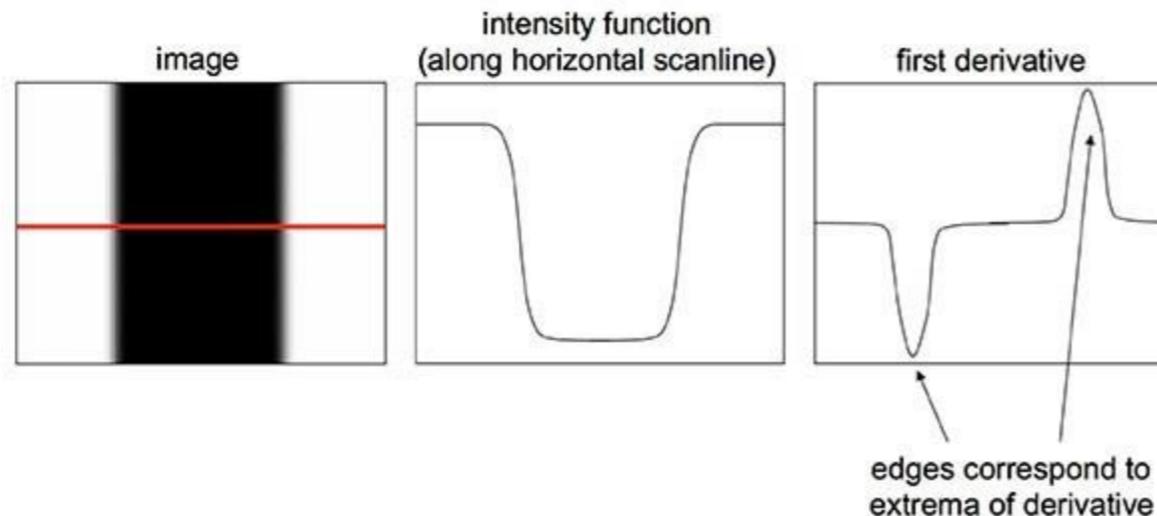


Sobel : 是一種過濾器，只是其是帶有方向的

結合了高斯平滑與微分運算的結合方法，所以它的抗噪聲能力很強

Sobel與Canny兩者雖然使用相同的底層技術，但執行方式有些差異。Sobel以簡單的卷積過濾器（convolutional filter）偵測圖像上水平及縱向光度的改變，以加權平均方式計算各點的數值來決定邊緣。

光影變化，這光影變化在術語上就是所謂「梯度」(gradient)。



Vertical Sobel =

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Horizontal Sobel =

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

$$F(x) = \sqrt{\text{Vertical Sobel}^2 + \text{Horizontal Sobel}^2}$$

## Sobel

是一種過濾器，只是其是帶有方向的

若以此公式算梯度值： $\nabla f(x, y) = \sqrt{G_x^2 + G_y^2}$

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 6 | 6 | 6 | 6 | 6 | 0 |
| 0 | 6 | 6 | 6 | 6 | 6 | 0 |
| 0 | 6 | 6 | 6 | 6 | 6 | 0 |
| 0 | 6 | 6 | 6 | 6 | 6 | 0 |
| 0 | 6 | 6 | 6 | 6 | 6 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |



|   |    |    |    |    |    |   |
|---|----|----|----|----|----|---|
| x | x  | x  | x  | x  | x  | x |
| x | 25 | 24 | 24 | 24 | 25 | x |
| x | 24 | 0  | 0  | 0  | 24 | x |
| x | 24 | 0  | 0  | 0  | 24 | x |
| x | 24 | 0  | 0  | 0  | 24 | x |
| x | 25 | 24 | 24 | 24 | 25 | x |
| x | x  | x  | x  | x  | x  | x |

Gx

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

Gy

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

## 7-2: Sobel、Scharr、Laplacian

```
dst = cv2.Sobel(src, ddepth, dx, dy[, dst[, ksize[, scale[, delta[, borderType]]]]])
```

src : 需要處理的影像；

ddepth : 影像的深度，-1表示採用的是與原影像相同的深度。

CV\_8U : 8-bit unsigned integers ( 0~255 )

CV\_8S : 8-bit signed integers ( -128~127 )

CV\_16U : 16-bit unsigned integers ( 0~65535 )

→大小相當於short

CV\_16S : 16-bit signed integers ( -32768~32767 )

→大小相當於short

CV\_32S : 32-bit signed integers ( -2147483648~2147483647 )

→大小相當於long

CV\_32F : 32-bit floating-point numbers

CV\_64F : 64-bit floating-point numbers

目標影像的深度必須大於等於原影像的深度；

| src.depth()   | ddepth                     |
|---------------|----------------------------|
| CV_8U         | -1, CV_16S, CV_32F, CV_64F |
| CV_16U/CV_16S | -1, CV_32F, CV_64F         |
| CV_32F        | -1, CV_32F, CV_64F         |
| CV_64F        | -1, CV_64F                 |

```
dst = cv2.Sobel(src, ddepth, dx, dy[, dst[, ksize[, scale[, delta[, borderType]]]]])
```

- **dx** 和 **dy**表示的是求導的階數，0表示這個方向上沒有求導，一般為0、1。
- **ksize** : 是 Sobel 運算元的大小，必須為1、3、5、7。
- **scale** : 是縮放導數的比例常數，預設情況下沒有伸縮係數；
- **delta**是：一個可選的增量，將會加到最終的dst中，同樣，預設情況下沒有額外的值加到dst中；
- **borderType** : 是判斷影像邊界的模式。這個引數預設值為cv2.BORDER\_DEFAULT。

## Scharr

濾波器設計用於消除小核Sobel導數誤差

$$dx \geq 0$$

$$dy \geq 0$$

$$dx+dy = 1$$

$$\begin{bmatrix} +3 & 0 & -3 \\ +10 & 0 & -10 \\ +3 & 0 & -3 \end{bmatrix}$$

$$\begin{bmatrix} +3 & +10 & +3 \\ 0 & 0 & 0 \\ -3 & -10 & -3 \end{bmatrix}$$

注意參數要用 cv2.CV\_64F

**Scharr-x**

**Scharr-y**

## Laplacian

先用Sobel 運算元計算二階x和y導數

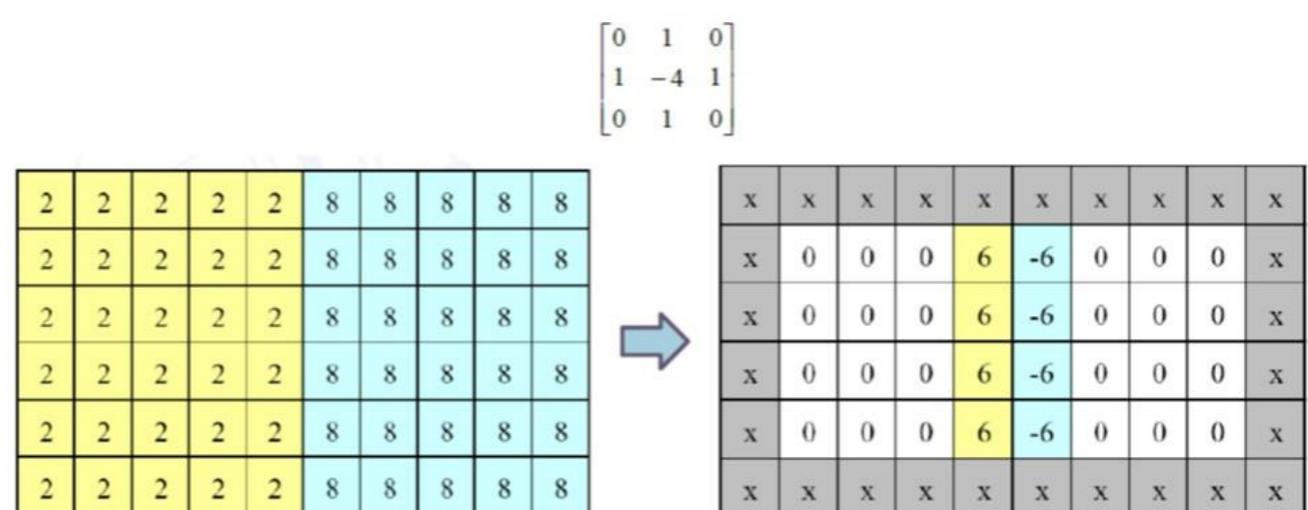
Laplacian對於雜訊 ( Noise ) 非常敏感，因此在實用上都會將影像\*\*先模糊化\*\*後再處理 (LoG Laplacian of Gaussian)。

|   |    |   |
|---|----|---|
| 0 | 1  | 0 |
| 1 | -4 | 1 |
| 0 | 1  | 0 |

|    |    |    |
|----|----|----|
| P1 | P2 | P3 |
| P4 | P5 | P6 |
| P7 | P8 | P9 |

$$P5_{\text{new}} = (P2 + P4 + P6 + P8) - 4 * P5$$

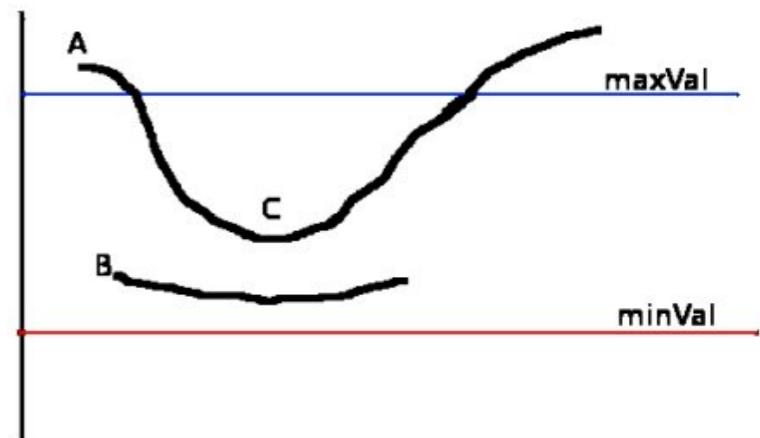
使用 Laplacian 找出邊緣。注意使用此函數除了傳入\*\*灰階\*\*影像之外，亦須指定輸出的影像浮點格式 CV\_64F



## Canny

Canny邊緣檢測，其實Canny不能被單獨稱為一種方法，因為它是一連串的過程加上其它方法，先模糊化去除不必要的像素、再使用類似Sobel方式取得XY軸邊緣。

Canny則較為複雜，它先將影像模糊化再進行非極大值抑制（non-maxima suppression），因此Canny比起Sobel較能處理雜訊問題，但是需要花費較多的硬體資源來處理。在下方的實作中我們可以看到它們輸出的差異。不過這部份技術原理已超出本人能力範圍無法深入解釋，若您對其技術原理有興趣，可再詳查其相關技術文件。



Canny邊緣檢測，其實Canny不能被單獨稱為一種方法，因為它是一連串的過程加上其它方法，先模糊化去除不必要的像素、再使用類似Sobel方式取得XY軸邊緣。

- 
1. Filter out any noise. The Gaussian filter is used for this purpose. An example of a Gaussian kernel of `size = 5` that might be used is shown below:

$$K = \frac{1}{159} \begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

2. Find the intensity gradient of the image. For this, we follow a procedure analogous to Sobel:

- a. Apply a pair of convolution masks (in  $x$  and  $y$  directions):

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

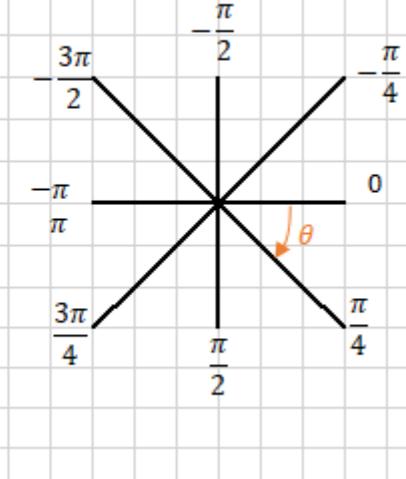
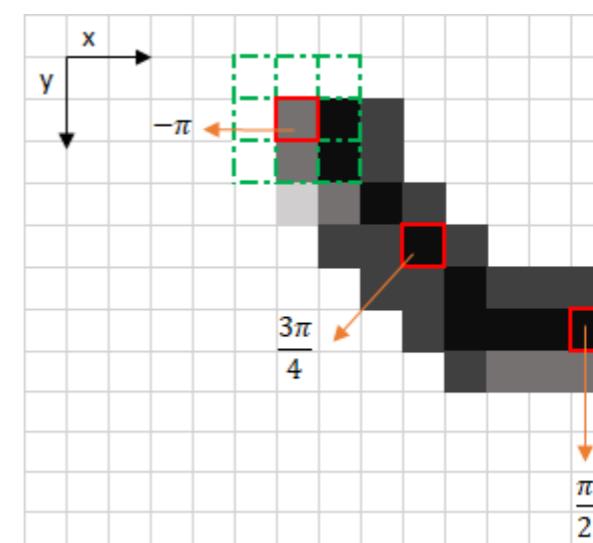
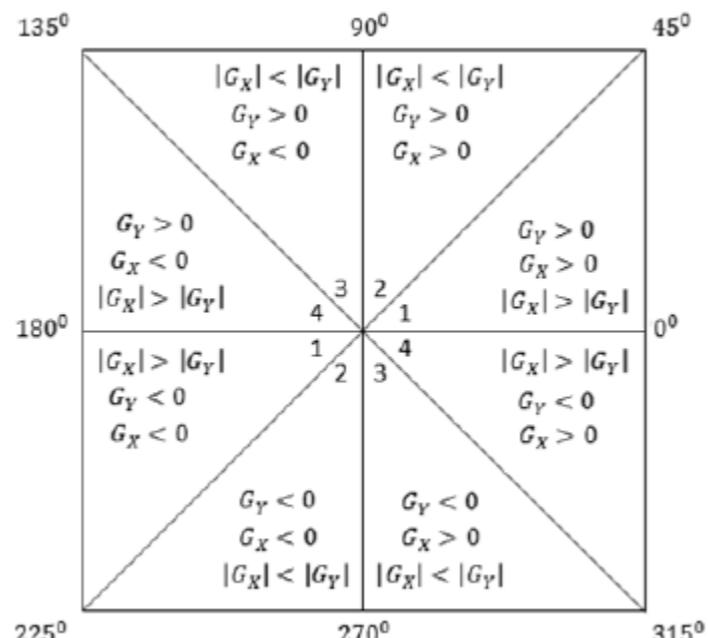
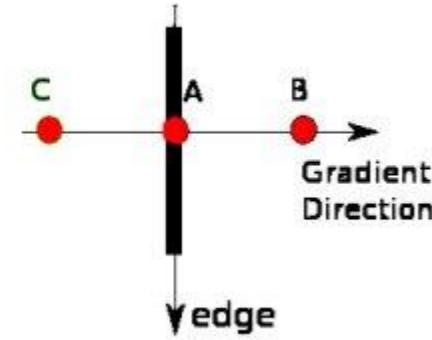
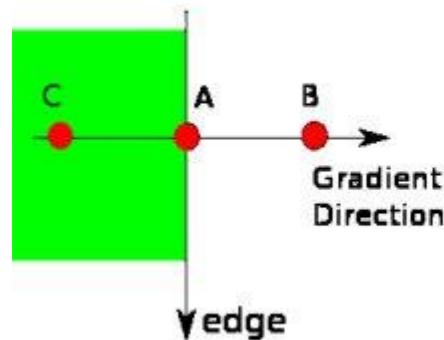
$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

把當前位置的梯度值與梯度方向上兩側的梯度值進行比較

$$\text{EdgeGradient}(G) = \sqrt{G_x^2 + G_y^2}$$

$$\text{Angle}(\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

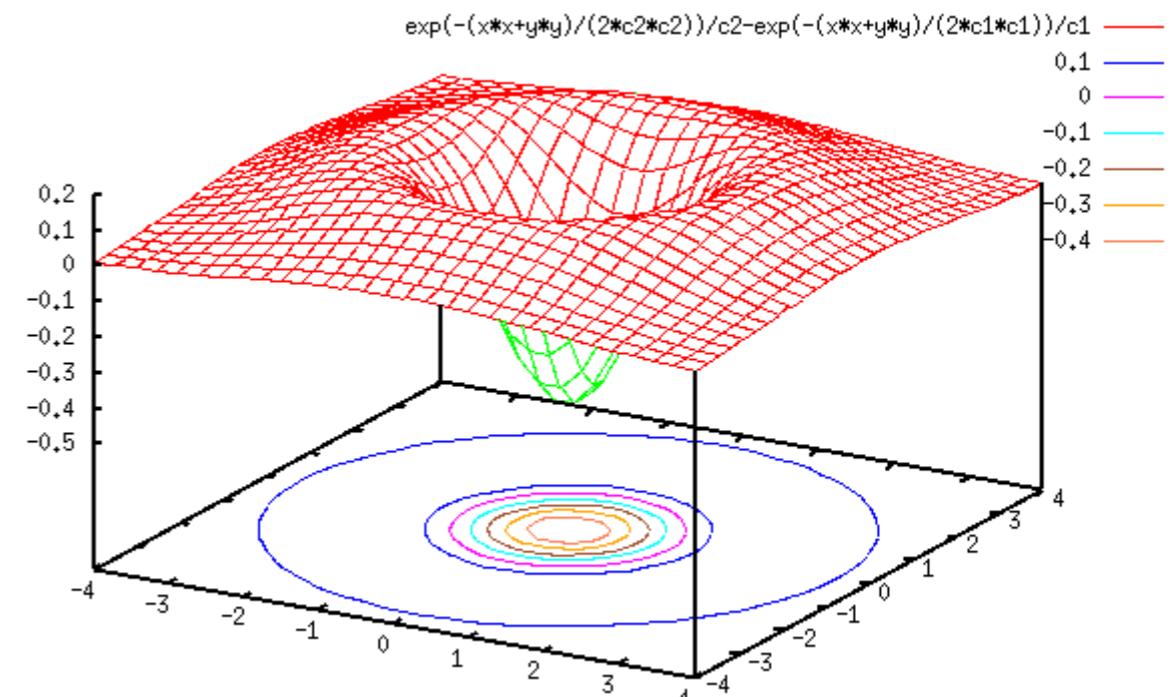
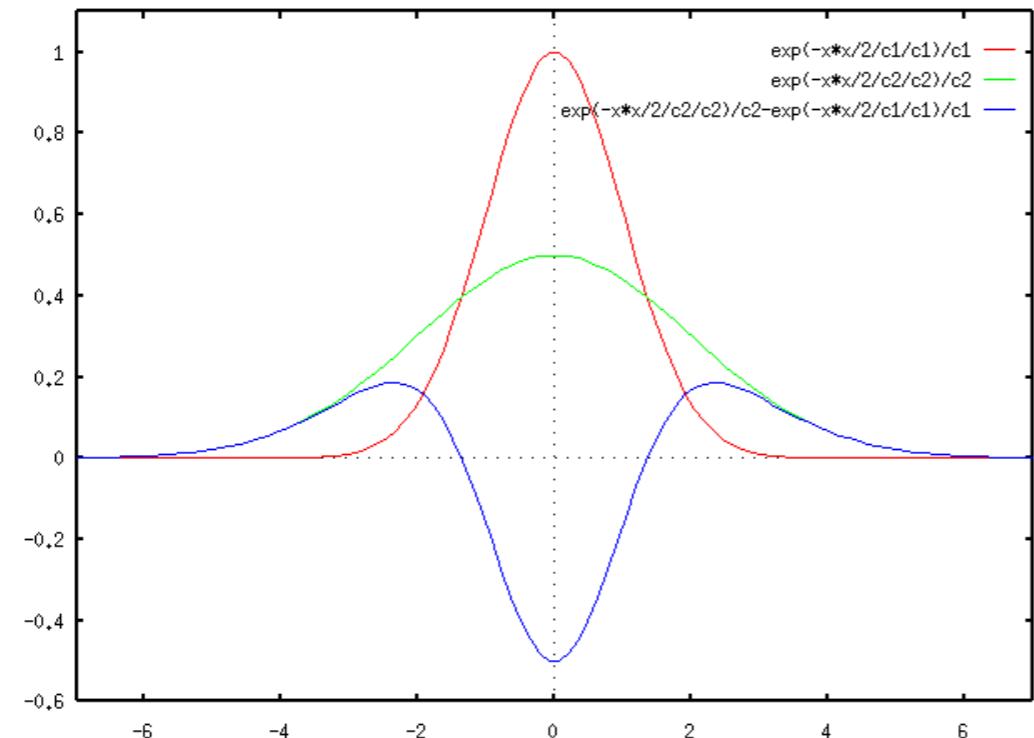
梯度方向垂直於邊緣方向



## DoG (Difference of Gaussian)高斯函數的差分

是灰度圖像增強和角點檢測的方法，其做法較簡單，證明較複雜，具體講解如下：

我們已經知道可以通過將圖像與高斯函數進行卷積得到一幅圖像的低通濾波結果，即去噪過程，這裡的 Gaussian 和高斯低通濾波器的高斯一樣，是一個函數，即為正態分佈函數。



# Module 8. 輪廓偵測 (contours)

- 8-1: cv.findContours  
函數**
- 8-2: cv.drawContours  
函數**
- 8-3: 配合邊緣檢測**

若 Edge 線條頭尾相連形成封閉的區塊，那麼它就是 Contour，否則就只是 Edge。Contours 是由一連串沒有間斷的點所組成的曲線，我們在針對影像進行分析及識別時，Contours 的使用是很重要的一個步驟。

結構簡單物體且背景色單純的圖片，我們可以直接使用灰階圖形取得該物體的 Contour，但如果是一張複雜背景的圖片，就需要先透過 edge detection 或 threshold 預處理才行

一般在對圖像取 contour 前，都會先轉黑白，做 threshold, canny 等 edge detection 處理，能提高 contour 的辨識效果。物件必須是白色，背景必須是黑色

## 8-1: cv.findContours函數 -- 獲取輪廓

```
contour, hierarchy = cv2.findContours(fgmask.copy(), cv2.RETR_EXTERNAL,  
cv2.CHAIN_APPROX_SIMPLE)
```

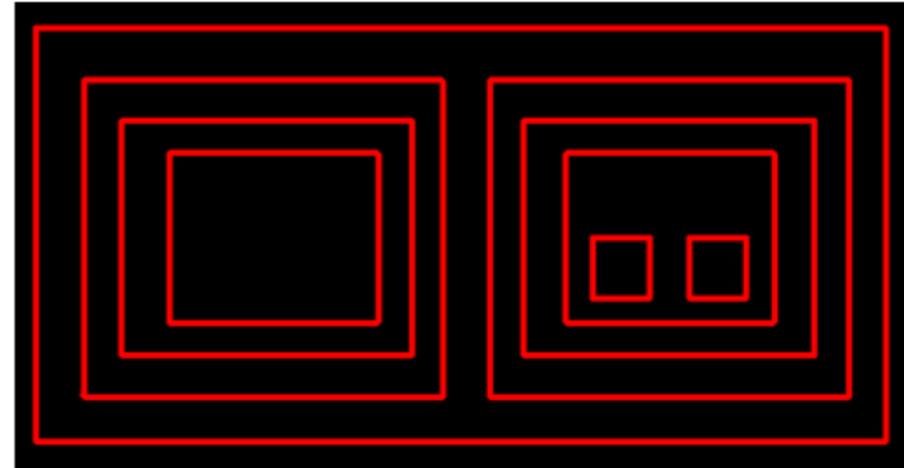
- contour返回值 : cv2.findContours()函式首先返回一個list，list中每個元素都是影像中的一個輪廓，用numpy中的ndarray表示。
- hierarchy返回值 : 該函式還可返回一個可選的hiararchy結果，這是一個ndarray，其中的元素個數和輪廓個數相同，每個輪廓contours[i]對應4個hierarchy元素hierarchy[i][0] ~hierarchy[i][3]，分別表示後一個輪廓、前一個輪廓、父輪廓、內嵌輪廓的索引編號，如果沒有對應項，則該值為負數。

## 8-1: cv.findContours函數 -- 獲取輪廓

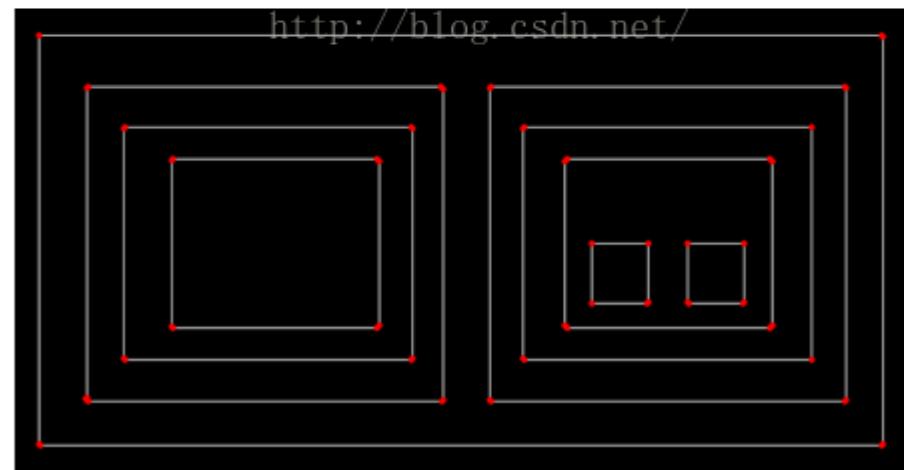
- cv2.RETR\_LIST : 這是最簡單的一個，它獲取所有輪廓，但是不建立父子關係，他們都是一個層級。
- 所以，層級屬性第三個和第四個欄位（父子）都是-1，但是Next和Previous還是有對應值。
- cv2.RETR\_EXTERNAL : 則表示只取外輪廓的 Contour (如果有其它 Contour 包在內部) 所有孩子輪廓都不要
- cv2.RETR\_TREE : 建立一個等級樹結構的輪廓
- cv2.RETR\_CCOMP : 建立兩個等級的輪廓，上面的一層為外邊界，裡面的一層為內孔的邊界資訊。如果內孔內還有一個連通物體，這個物體的邊界也在頂層。

## 8-1: cv.findContours函數 -- 獲取輪廓

- cv2.CHAIN\_APPROX\_NONE 儲存所有的輪廓點，相鄰的兩個點的畫素位置差不超過1，即 $\max(\text{abs}(x_1-x_2), \text{abs}(y_2-y_1)) == 1$
- cv2.CHAIN\_APPROX\_SIMPLE 壓縮水平方向，垂直方向，對角線方向的元素，只保留該方向的終點座標，例如一個矩形輪廓只需4個點來儲存輪廓資訊
- cv2.CHAIN\_APPROX\_TC89\_L1，CV\_CHAIN\_APPROX\_TC89\_KCOS 使用 Teh-Chin chain 近似演算法



(a) CV\_CHAIN\_APPROX\_NONE



(b) CV\_CHAIN\_APPROX\_SIMPLE

## 8-2: cv.drawContours函數 -- 繪出輪廓

```
cv2.drawContours(image, contours, contourIdx, color[, thickness[, lineType[,  
hierarchy[, maxLevel[, offset ]]]]])
```

**image** : 是指明在哪幅影像上繪製輪廓；

**contours** : 是輪廓本身，在Python中是一個list。

**contourIdx** : 指定繪製輪廓list中的哪條輪廓，如果是-1，則繪製其中的所有輪廓。

**color** : 繪製輪廓的顏色

**thickness** : 表明輪廓線的寬度，如果是-1 ( cv2.FILLED )，則為填充模式。

**cv2.RETR\_EXTERNAL** : 則表示只取外輪廓的 Contour (如果有其它 Contour 包在內部)  
所有孩子輪廓都不要

cv2.drawContours 可協助我們找出 Contours

- cv2.RETR\_TREE : 建立一個等級樹結構的輪廓
- cv2.RETR\_CCOMP : 建立兩個等級的輪廓，上面的一層為外邊界，裡面的一層為內孔的邊界資訊。如果內孔內還有一個連通物體，這個物體的邊界也在頂層。
- hull : Convex Hull 概念上是取一個物件的凸多邊形框.

有了 Convex Hull 後, 它和原圖像的 deviation (偏差), 也就是那些凹陷的部份, 就叫做 Convexity Defects (凸多邊形缺陷?), OpenCV 有個叫做 convexityDefect 的指令, 能找出圖像的 Convexity Defects.

## 矩的計算 moment()

`cv2.moments(cnts[i])`

空間矩  $m : m_{00}, m_{10}, m_{01}, m_{20}, m_{11}, m_{02}, m_{30}, m_{21}, m_{12}, m_{03}$

中心矩  $\mu : \mu_{20}, \mu_{11}, \mu_{02}, \mu_{30}, \mu_{21}, \mu_{12}, \mu_{03}$

規一化中心矩  $Hu$   $\nu : \nu_{20}, \nu_{11}, \nu_{02}, \nu_{30}, \nu_{21}, \nu_{12}, \nu_{03}$

`contourArea (), arcLength(cnts[i], True)`

輪廓 0 的 center (323, 412), 面積 : 23242.0, 週長 : 607.66

輪廓 1 的 center (561, 330), 面積 : 16244.5, 週長 : 620.03

輪廓 2 的 center (185, 184), 面積 : 19033.0, 週長 : 516.22

輪廓 3 的 center (375, 299), 面積 : 449849.0, 週長 : 2700.0



# Module 9. 形態學

- 9-1: 形態學介紹
- 9-2: 什麼是形態學
- 9-3: 膨脹、侵蝕、開運算、畢運算

影像處理中指的形態學，往往表示的是數學形態學。

數學形態學（Mathematical morphology）是一門建立在格論和拓撲學基礎之上的影像分析學科，是數學形態學影像處理的基本理論。其基本的運算包括：二值腐蝕和膨脹、二值開閉運算、骨架抽取、極限腐蝕、擊中擊不中變換、形態學梯度、Top-hat變換、顆粒分析、流域變換、灰值腐蝕和膨脹、灰值開閉運算、灰值形態學梯度等。

簡單來講，形態學操作就是基於形狀的一系列影像處理操作。OpenCV為進行影像的形態學變換提供了快捷、方便的函式。最基本的形態學操作有二種，他們是：膨脹與腐蝕(Dilation與Erosion)。

膨脹與腐蝕能實現多種多樣的功能，主要如下：

- 消除噪聲
- 分割(isolate)出獨立的影像元素，在影像中連線(join)相鄰的元素。
- 尋找影像中的明顯的極大值區域或極小值區域
- 求出影像的梯度

## 9-2: 什麼是形態學

形態學操作是根據影像形狀進行的簡單操作，一般情況下對二值化影像進行的操作。需要輸入兩個引數，一個是原始影像，第二個被稱為結構化元素或核，它是用來決定操作的性質的。兩個基本的形態學操作是腐蝕和膨脹。他們的變體構成了開運算，閉運算，梯度等。

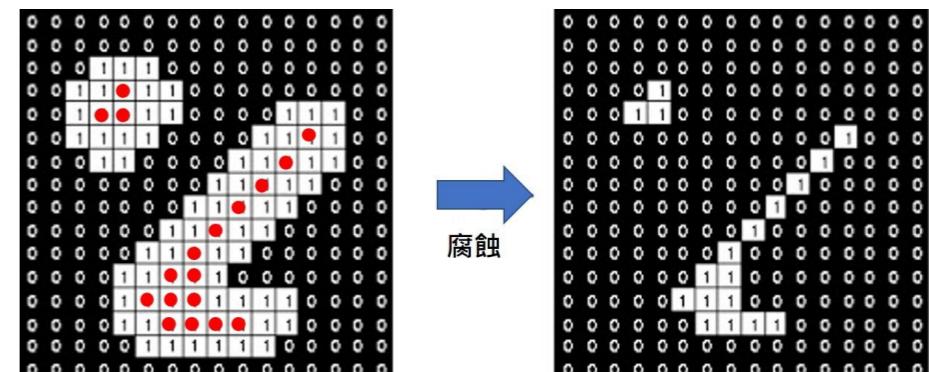
## Erode 腐蝕：以 Kernel 中心點移動

卷積核沿著圖像滑動，如果與卷積核對應的原圖像的所有圖元值與 Kernel 相同，那麼中心元素就給 1，否則就變為零。根據卷積核的大小靠近前景的所有圖元都會被腐蝕掉（變為0），所以前景物體會變小，整幅圖像的白色區域會減少。這對於去除白色雜訊很有用，也可以用來斷開兩個連在一塊的物體等。

`cv2.erode(img, kernel=None, iterations =1)`

- `img` : 指需要腐蝕的圖
- `kernel` : 指腐蝕操作的內核，默認是一個簡單的 3X3 全 1 的矩陣，我們也可以利用 `getStructuringElement( )` 函數指明它的形狀
- `iterations` : 指的是腐蝕次數，省略是默認為1

在進行腐蝕和膨脹的講解之前，首先需要注意，腐蝕和膨脹是對白色部分（高亮部分）而言的，不是黑色部分。膨脹就是影像中的高亮部分進行膨脹，“領域擴張”，效果圖擁有比原圖更大的高亮區域。腐蝕就是原圖中的高亮部分被腐蝕，“領域被蠶食”，效果圖擁有比原圖更小的高亮區域



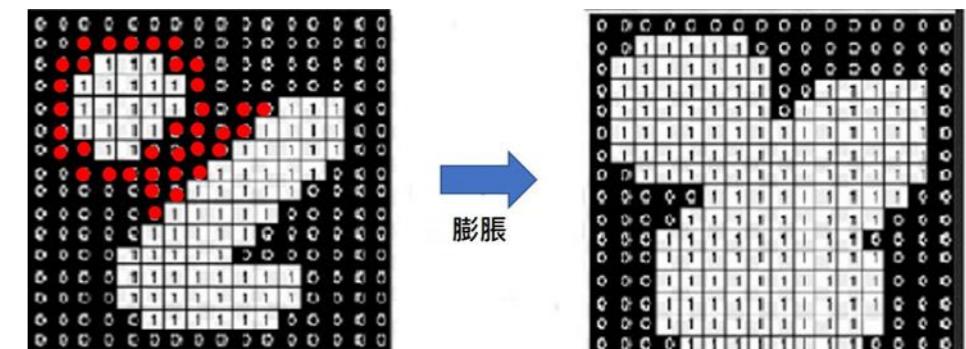
## Dilate 膨脹

與腐蝕相反，與卷積核對應的原圖像的圖元值中只要有一個是1，中心元素的圖元值就是1。所以這個操作會增加圖像中的白色區域（前景）。一般在去雜訊時先用腐蝕再用膨脹。因為腐蝕在去掉白色雜訊的同時，也會使前景對像變小。所以我們再對他進行膨脹。這時雜訊已經被去除了，不會再回來了，但是前景還在並會增加。

膨脹也可以用來連接兩個分開的物體。其實，膨脹就是求區域性最大值的操作。按數學方面來說，膨脹或者腐蝕操作就是將影像（或影像的一部分割槽域，我們稱之為A）與核（我們稱之為B）進行卷積。

核可以是任何的形狀和大小，它擁有一個單獨定義出來的參考點，我們稱其為錨點（anchorpoint）。多數情況下，核是一個小的中間帶有參考點和實心正方形或者圓盤，其實，我們可以把核視為模板或者掩碼。

而膨脹就是求區域性最大值的操作，核B與圖形卷積，即計算核B覆蓋的區域的畫素點的最大值，並把這個最大值賦值給參考點指定的畫素。這樣就會使影像中的高亮區域逐漸增長。如下圖所示，這就是膨脹操作的初衷。



### Open : MORPH\_OPEN

先 erode 再 dilate 叫 open 運算，作用能消除圖片上的小標點  
降噪, 計數

### Close : MORPH\_CLOSE

先 dilate, 再 erode : 它經常被用來填充前景物體中的小洞，或者前景物體上的小黑點。不同前景影像連接

### Gradient 梯度 (膨脹 - 腐蝕)

取得前景原始影像的邊緣  
用於獲取圖片的輪廓，形態梯度圖 = 膨脹圖 - 腐蝕圖

### Tophat 禮帽 ( 原圖 - Open )

取得原影像雜訊資訊

取得比原影像邊緣更 "亮" 的邊緣

Top Hat = 原圖 - open 運算圖，顯示出原圖去除掉的白色部分。

### Blackhat( close - 原圖 )

取得原影內部小孔, 或前景中的小黑點

取得比原影像邊緣更 "暗" 的邊緣

Black Hat = 原圖 - 閉運算，顯示出原圖去除掉的黑色部分。

## Define Own Kernel

**MORPH\_RECT** : 矩形結構，所有元素是 1

**MORPH\_CROSS** : 十字結構，對角線元素是 1

**MORPH\_ELLIPSE** : 橢圓形結構，所有元素是 1

| 中文名   | 英文名                 | api  | 原理                           | 個人理解            |
|-------|---------------------|--|------------------------------|-----------------|
| 腐蝕    | erode               | <code>erosion = cv2.erode(src=girl_pic, kernel=kernel)</code>                  | 在窗中，只要含有 0，則窗內全變為 0，可以去淺色噪點  | 淺色成分被腐蝕         |
| 膨脹    | dilate              | <code>dilation = cv2.dilate(src=girl_pic, kernel=kernel)</code>                | 在窗中，只要含有 1，則窗內全變為 1，可以增加淺色成分 | 淺色成分得膨脹         |
| 開運算   | morphology-open     | <code>opening = cv2.morphologyEx(girl_pic, cv2.MORPH_OPEN, kernel)</code>      | 先腐蝕，後膨脹，去白噪點                 | 先合再開，對淺色成分不利    |
| 閉運算   | morphology-close    | <code>closing = cv2.morphologyEx(girl_pic, cv2.MORPH_CLOSE, kernel)</code>     | 先膨脹，後腐蝕，去黑噪點                 | 先開再合，淺色成分得勢     |
| 形態學梯度 | morphology-gradient | <code>gradient = cv2.morphologyEx(girl_pic, cv2.MORPH_GRADIENT, kernel)</code> | 一幅影像腐蝕與膨脹的區別，可以得到輪廓          | 數值上解釋為：膨脹減去腐蝕   |
| 禮帽    | tophat              | <code>tophat = cv2.morphologyEx(girl_pic, cv2.MORPH_TOPHAT, kernel)</code>     | 原影像減去開運算的差                   | 數值上解釋為：原影像減去開運算 |
| 黑帽    | blackhat            | <code>blackhat = cv2.morphologyEx(girl_pic, cv2.MORPH_BLACKHAT, kernel)</code> | 閉運算減去原影像的差                   | 數值上解釋為：閉運算減去原影像 |

# Module 10. 圖像金字塔

- 10-1: 圖像金字塔理論
- 10-2: PyrDown 及 PyrUp
- 10-3: 實作圖像金字塔

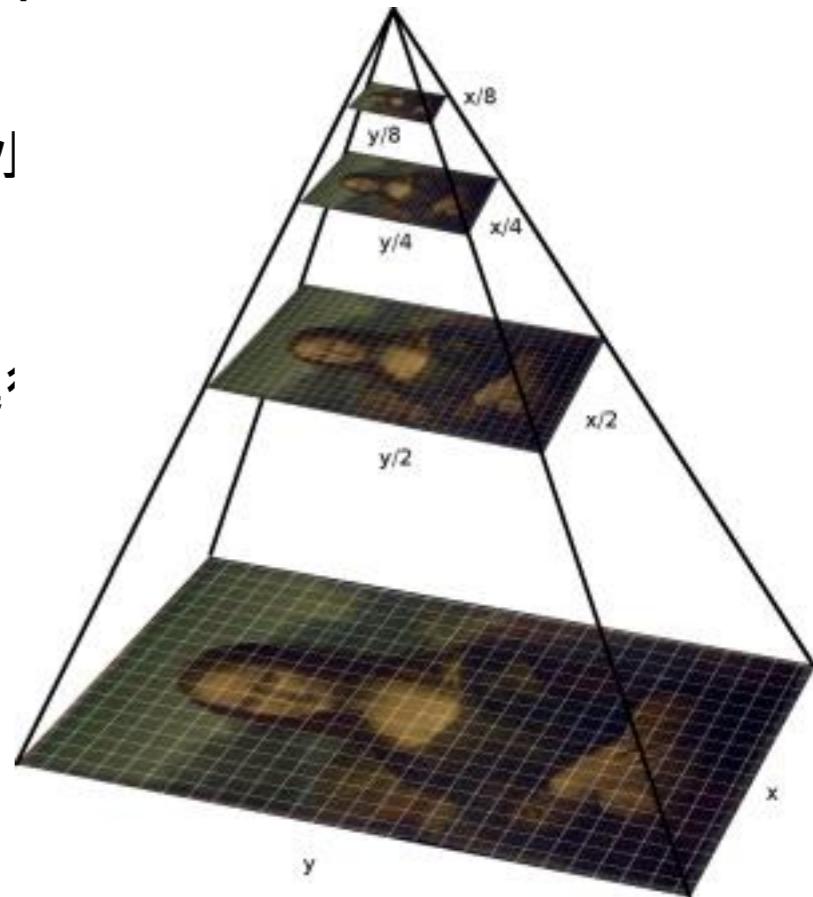
## 10-1: 圖像金字塔理論

影像金字塔是影像多尺度表達的一種，是一種以多解析度來解釋影像的有效但概念簡單的結構。一幅影像的金字塔是一系列以金字塔形狀排列的解析度逐步降低，且來源於同一張原始圖的影像集合。其通過梯次向下取樣獲得，直到達到某個終止條件才停止取樣。我們將一層一層的影像比喻成金字塔，層級越高，則影像越小，解析度越低。將參加融合的每幅影像分解為多尺度的金字塔影像序列，將低解析度的影像在上層，高解析度的影像在下層，上層影像的大小為前一層影像大小的  $1/4$

我們經常會將某種尺寸的圖像轉換為其他尺寸的圖像，如果放大或者縮小以使用OpenCV為我們提供的如下兩種方式：

`resize`函數。這是最直接的方式，

`pyrUp()`、`pyrDown()`函數。即圖像金字塔相關的兩個函數，對圖像進



## 高斯金字塔

高斯金字塔的生成，主要用到pyrDown函數，其生成原理：

- 輸入圖片
- 對圖像進行高斯內核卷積
- 將所有偶數行和列去除，得到 1. 中圖片的 $1/4$ 大小
- 將 3. 中得到的圖片重複 2. 和 3. 步驟，直到得到  $n$  級圖像金字塔

`pyrDown(src, dst=None, dstsize=None, borderType=None)`

注意：`dstsize` 參數是有默認值的，調用函數的時候不指定第三個參數，那麼這個值是按照  $\text{Size}((\text{src.cols}+1)/2, (\text{src.rows}+1)/2)$  計算的。而且不管如何指定這個參數，一定必須保證滿足以下關係式： $|\text{dstsize.width} * 2 - \text{src.cols}| \leq 2$ ;  $|\text{dstsize.height} * 2 - \text{src.rows}| \leq 2$ 。也就是說降取樣的意思其實是把影像的尺寸縮減一半，行和列同時縮減一半。

## 拉普拉斯金字塔

### 高斯差分金字塔的構建

使用組和層的結構構建了一個具有線性關係的金字塔（尺度空間），這樣可以在連續的高斯核尺度上查找圖像的特徵點；另外，它使用一階的高斯差分來近似高斯的拉普拉斯核，大大的減少了運算量。

### 尺度空間的極值檢測及特徵點的定位

### 鄰域圖元的資訊。

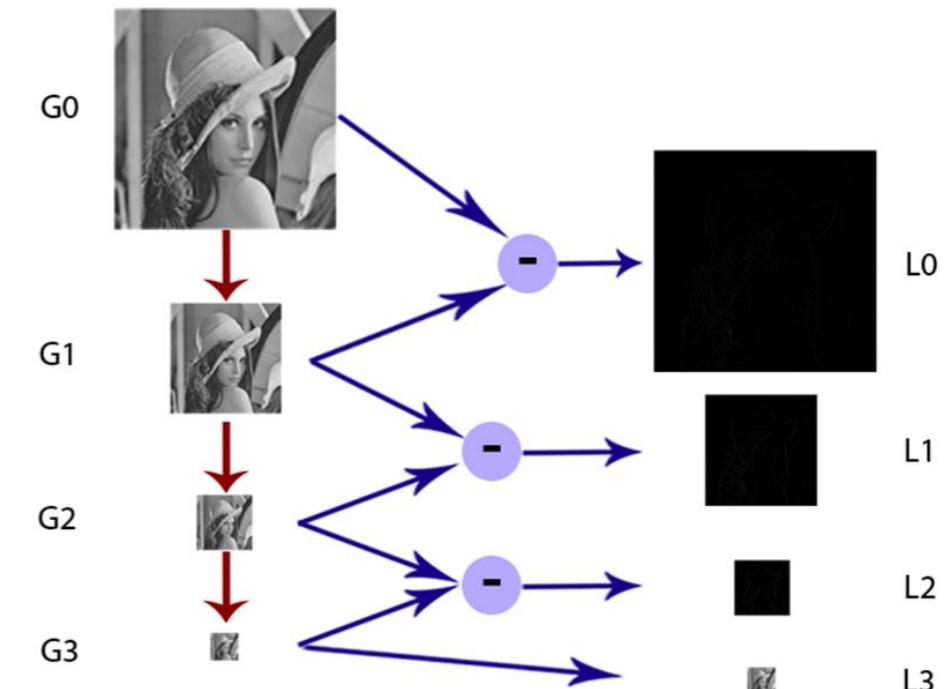
搜索上一步建立的高斯尺度空間，通過高斯差分來識別潛在的對尺度和旋轉不變的特徵點。但是，在離散空間中，局部極值點可能並不是真正意義的極值點，真正的極值點有可能落在離散點的間隙中，SIFT通過尺度空間DoG函數進行曲線擬合尋找極值點。

### 特徵方向賦值

基於圖像局部的梯度方向，分配給每個關鍵點位置一個或多個方向，後續的所有操作都是對於關鍵點的方向、尺度和位置進行變換，從而提供這些特徵的不變性。

### 特徵描述子的生成

通過上面的步驟已經找到的SIFT特徵點的位置、方向、尺度資訊，最後使用一組向量來描述特徵點及其周圍鄰域圖元的資訊。



# Module 11.

## 影像模板匹配

11-1: 什麼是影像模板  
匹配

11-2: 影像模板匹配介  
紹

11-3: 實作影像模板匹  
配

## 11-1: 什麼是影像模板匹配

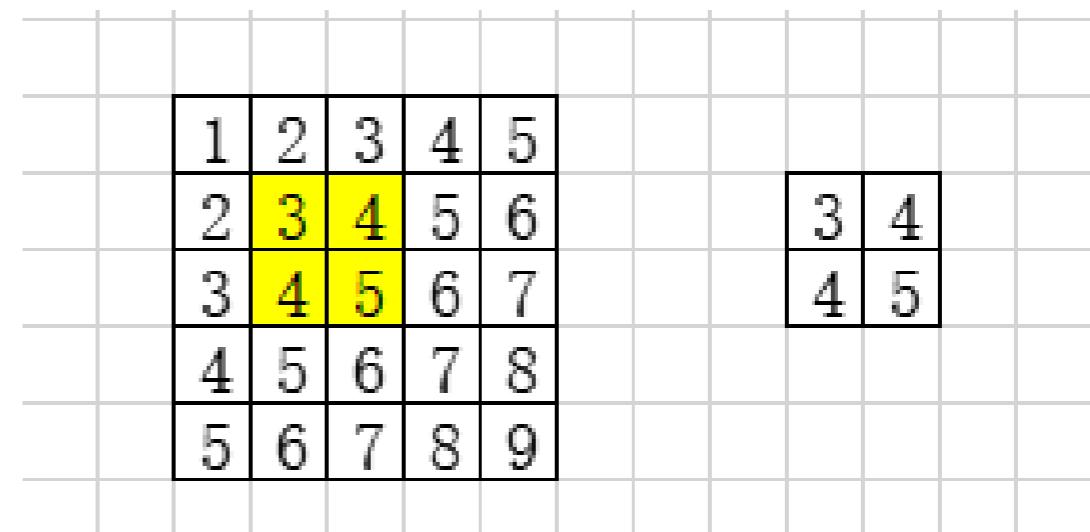
範本匹配是一種最原始、最基本的模式識別方法，研究某一特定物件物的圖案位於圖像的什麼地方，進而識別物件物，這就是一個匹配問題。它是影像處理中最基本、最常用的匹配方法。

範本就是一副已知的小圖像，而範本匹配就是在一副大圖像中搜尋目標，已知該圖中有要找的目標，且該目標同範本有相同的尺寸、方向和圖像元素，通過一定的演算法可以在圖中找到目標，確定其座標位置。

這就說明，我們要找的範本是裡圖像裡標準存在的，這裡說的標準，就是說，一旦圖像或者範本發生變化，比如旋轉，修改某幾個圖元，圖像翻轉等操作之後，我們就無法進行匹配了，這也是這個演算法的弊端。

所以這種匹配演算法，相當於“人工智障式”匹配，就是在待檢測圖像上，從左到右，從上向下對範本圖像與小東西的圖像進行比對。

模板匹配，是一種在給定的目標影像中尋找給定的模板影像的技術，原理很簡單，就是利用一些計算相似度的公式來判斷兩張影像之間有多相似



模板影像小於目標影像的話，就需要用 sliding window 的方式來得到多個匹配的結果，可以選擇取最佳匹配或是設定一個門檻值，只要比這個門檻值好的結果都認為是有效的匹配

Square difference 平方差，這是最常見的數學公式

Correlation 相關性：計算 dot product (內積)，可以想成是計算兩個向量在空間中的距離有多近，就是用 cosine 去算夾角，cosine 值越大代表夾角越小，代表越接近

(假設模板影像是 $10 \times 10$ 的影像，可以被看作是100維的向量，每一維是像素的值)

Correlation coefficient 與 Correlation 差別只在於計算內積時還要減去各個向量的平均值，如此一來相關性就會被放大



## 11-3: 實作影像模板匹配

| 說明      | 語法 method        | 值 |
|---------|------------------|---|
| 平方差匹配   | TM_SQDIFF        | 1 |
| 標準平方差匹配 | TM_SQDIFF_NORMED | 2 |
| 相關匹配    | TM_CCORR         | 3 |
| 標準相關匹配  | TM_CCORR_NORMED  | 4 |
| 相關匹配    | TM_CCOEFF        | 5 |
| 標準相關匹配  | TM_CCOEFF_NORMED | 6 |

### TM\_SQDIFF 平方差匹配 : minVal

從名字來理解，平方差匹配就是通過計算每個圖元點的差的平方的和，和數學中統計裡面的平方差類似。但是因為我們要的只是一個值，所以我們最後不需要求平均。

### TM\_SQDIFF\_NORMED 標準平方差匹配 : minVal

這個只是對上面的進行了標準化處理，經過處理後，上面的值就不會太大

### TM\_CCORR 相關匹配 : maxVal

這類方法採用範本和圖像間的乘法操作，所以較大的數表示匹配程度較高，0標識最壞的匹配效果。

### TM\_CCORR\_NORMED 標準相關匹配 : maxVal

這個只是對上面的進行了標準化處理，經過處理後，上面的值就不會太大。

### TM\_CCOEFF 相關匹配 : maxVal

這類方法將模版對其均值的相對值與圖像對其均值的相關值進行匹配,1表示完美匹配,-1表示糟糕的匹配,0表示沒有任何相關性(隨機序列)

### TM\_CCOEFF\_NORMED 標準相關匹配 : maxVal

這個只是對上面的進行了標準化處理，經過處理後，上面的值就不會太大。

# Module 12. 特徵擷取, 匹配 (SIFT)

12-1: 擷取原理  
12-2: Keypoint Descriptor  
12-3: Keypoint Matching

要辨識某物體的條件就是先掌握其特徵！由於我們要辨識的是某個物件而非整張相片，因此需要提取所謂稱為「Local features」的特徵，作法是先在影像中選取重要的特徵點，接著以其為 base 取得周圍的特徵（即local features），這些來自不同相片的local features會透過稍後會說明的Feature matching功能來比對是否有相同的物件。

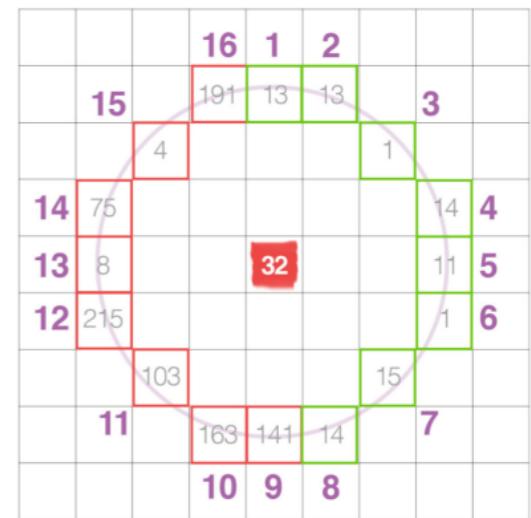
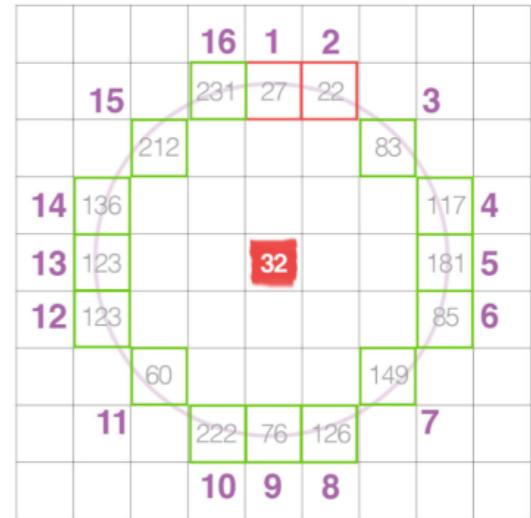
特徵點 → 局部特徵 → Feature matching 這些特性可用 edges、corners、blobs (斑點) 等組合來描述  
Keypoint detection、Feature extraction 以及 Feature matching

- Keypoint detection：在圖片中取得感興趣的關鍵點（可能為 edges、corners 或 blobs）。
- Feature extraction：針對各關鍵點提取該區域的 features（我們稱為 local features）。
- 關鍵點篩選並進行 Feature matching。

## FAST – FastFeatureDetector

原理簡單，執行速度相當快，主要用於偵測corners，亦可偵測 blob。適用於要求速度的 real-time analysis，適用於速度慢或較低階的執行環境。使用度極高，尤其在需要即時的 realtime 環境。

- FAST 方法認為，一個以  $c$  為中心、半徑為  $r$  的圓形，若它位於一個所謂的 corner 上，那麼該圓的圓周上必有連續  $n$  個點，其強度值大於或小於中心點  $c$  加上一個指定的 threshold 門檻的強度值。如果是的話，該中心點  $c$  便被認為是 keypoint。
- 考慮是否應該將中心像素  $p$  視為關鍵點。中心像素  $p$  具有灰度強度值  $p = 32$ 。為了使該像素成為關鍵點，必須在圓的邊界上具有  $n = 12$  (一般  $3/4$ ) 個連續像素，這些像素要不就比  $p + t$  亮，要不就比  $p - t$  暗。在此示例中，假設  $t = 16$ 。

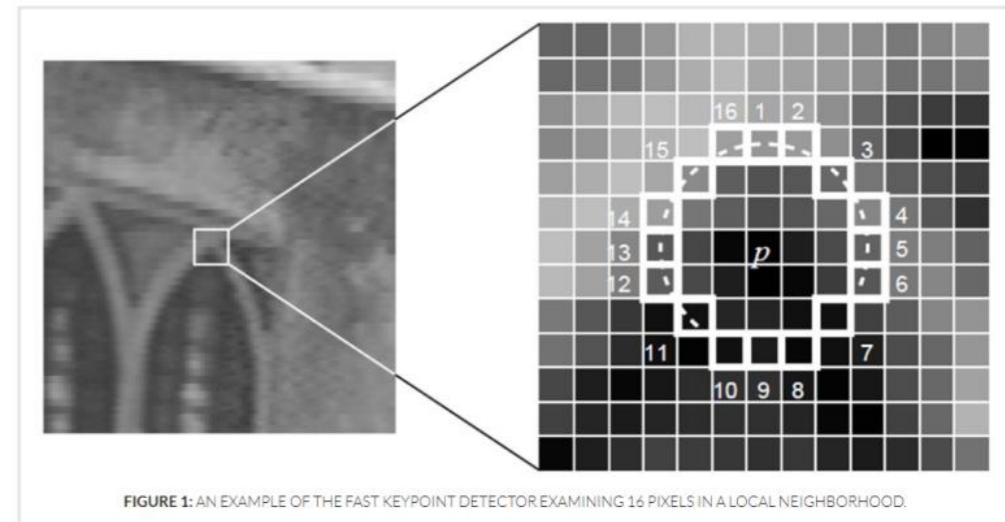


FAST演算法提取角點的步驟：

- 在圖像中選擇圖元 $p$ ，假設其灰度值為： $I_p$
- 設置一個閾值 $T$ ，例如： $I_p$  的20%
- 選擇 $p$ 周圍半徑為3的圓上的16個圖元，作為比較圖元
- 假設選取的圓上有連續的 $N$ 個圖元大於 $I_p+T$ 或者小於 $I_p-T$ ，那麼可以認為圖元 $p$ 就是一個特徵點。（ $N$ 通常取12，即為FAST-12；常用的還有FAST-9, FAST-11）。

缺點：

- 檢測到的特徵點過多並且會出現“繁堆”的現像。這可以在第一遍檢測完成後，使用非最大值抑制（Non-maximal suppression），在一定區域內僅保留回應極大值的角點，避免角點集中的情況。
- FAST提取到的角點沒有方向和尺度資訊
- SIFT和SURF演算法都包含有各自的特徵點描述子的計算方法，而FAST不包含特徵點描述子的計算，僅僅只有特徵點的提取方法，這就需要一個特徵點描述方法來描述FAST提取到的特徵點，以方便特徵點的匹配



## 12-2: Keypoint Descriptor

SIFT : Scale Invariant Feature Transform，尺度不變特徵變換。

SIFT特徵對旋轉、尺度縮放、亮度變化等保持不變性，是一種非常穩定的局部特徵。

角點在影像旋轉的情況下也可以檢測到，但是如果減小(或者增加)影像的大小，可能會丟失影像的某些部分，甚至導致檢測到的角點發生改變。這樣的損失現象需要一種與影像比例無關的角點檢測方法來解決。尺度不變特徵變換 (Scale-Invariant Feature Transform, SIFT) 可以解決這個問題。

SIFT 特徵是基於物體上的一些區域性外觀的興趣點而與影像的大小和旋轉無關。對於光線、噪聲、些微視角改變的容忍度也相當高。基於這些特性，它們是高度顯著而且相對容易擷取，在母數龐大的特徵資料庫中，很容易辨識物體而且鮮有誤認。使用 SIFT 特徵描述對於部分物體遮蔽的偵測率也相當高，甚至只需要3個以上的 SIFT 物體特徵就足以計算出位置與方位。在現今的電腦硬體速度下和小型的特徵資料庫條件下，辨識速度可接近即時運算。SIFT特徵的資訊量大，適合在海量資料庫中快速準確匹配。

## 12-2: Keypoint Descriptor

SIFT 演算法利用 DoG (差分高斯)來提取關鍵點(或者說成特徵點)，DoG 的思想是用不同的尺度空間因子(高斯正態分佈的標準差 $\sigma$ )對影像進行平滑，然後比較平滑後圖像的區別，差別大的畫素就是特徵明顯的點，即可能是特徵點。對得到的所有特徵點，我們剔除一些不好的，SIFT運算元會把剩下的每個特徵點用一個128維的特徵向量進行描述

- 原理較為複雜。
- 主要針對 blob 偵測，不過亦可偵測 corners。
- 可適應物件的 scale (大小變化)及 angle ( 旋轉角度 ) 等情況。
- 在DoG模型中，使用了不同尺寸影像並套用高斯模糊，比較不同模糊比例之間的變化，來決定是否為 keypoint。
- 由於計算量大，DoG 的執行速度較慢，不適用於 realtime 的環境。
- SIFT已廣泛的應用於電腦視覺領域，並成為評定新 keypoint detecter 的效率指標。
- 速度上ORB>SURF>SIFT，SURF的魯鈍性(抗干擾能力 ) 更好一些。

SURF : This algorithm is patented

Speeded Up Robust Features。加速版的SIFT。

SURF的流程和SIFT比較類似，這些改進體現在以下幾個方面：

- 特徵點檢測是基於Hessian矩陣，依據Hessian矩陣行列式的極值來定位特徵點的位置。並且將Hessian特徵計算與高斯平滑結合在一起，兩個操作通過近似處理得到一個核範本。
- 在構建尺度空間時，使用box filter與源圖像卷積，而不是使用DoG運算元。
- SURF使用一階Haar小波在x、y兩個方向的回應作為構建特徵向量的分佈資訊。
- SURF演算法比SIFT快好幾倍，它吸收了SIFT演算法的思想。SURF採用Hessian演算法檢測關鍵點。SURF需要提供閾值，特徵隨著閾值的增加而減少。
- ORB是用來取代SIFT和SURF的，與兩者相比，ORB有更快的速度。ORB用FAST來檢測關鍵點，用BRIEF來進行關鍵點特徵描述。
- 有點迷糊，現在總結一下

## Brief

過程如下：

- 為減少雜訊干擾，先對圖像進行高斯濾波（方差為2，高斯窗口為 $9 \times 9$ ）
- 以特徵點為中心，取 $S \times S$ 的鄰域大視窗。在大視窗中隨機選取一對（兩個） $5 \times 5$ 的子視窗，比較子視窗內的圖元和（可用積分圖像完成），進行二進位賦值。（一般 $S=31$ ）其中， $p(x)$ ， $p(y)$ 分別隨機點 $x=(u_1, v_1), y=(u_2, v_2)$ 所在 $5 \times 5$ 子視窗的圖元和。
- 在大視窗中隨機選取N對子視窗，重複步驟2的二進位賦值，形成一個二進位編碼，這個編碼就是對特徵點的描述，即特徵描述子。（一般 $N=256$ ）
- 非常重要的一點是：BRIEF 是一種特徵描述符，它不提供查找特徵的方法。所以我們不得不使用其他特徵檢測器，比如 SIFT 和 SURF 等。原始文獻推薦使用 CenSurE 特徵檢測器，這種演算法很快。而且 BRIEF 演算法對 CenSurE 關鍵點的描述效果要比 SURF 關鍵點的描述更好。
- 簡單來說 BRIEF 是一種對特徵點描述符計算和匹配的快速方法。這種演算法可以實現很高的識別率，除非出現平面內的大旋轉。

## ORB : ORiented Brief / Oriented FAST and Rotated BRIEF

ORB亦是針對 FAST 的強化，但除了scale space invariance，亦加入了旋轉不變性(rotation invariance)。

ORB原理有三大步驟：

- Pyramid圖像尺寸並進行各尺寸的FAST計算。
- 使用Harris keypoint detector的方法計算每個keypoint分數（是否近似corner？），並進行排序，最多僅取500個keypoints，其餘則丟棄。
- 於此第三步中加入旋轉不變性，使用「intensity centroid」計算每個keypoint的rotation。ORB與BRISK相同，繼承了FAST運算快速的特性，可適用於realtime分析。

## Harris

速度相當快，但仍較 FAST 慢，不過偵測 corner 比起 FAST 準確一些。廣泛應用於偵測 edges 及 corners, Harris可用於識別角點。此函數可以很好的檢測角點，這些角點在圖像旋轉的情況下也能被檢測到。但是如果減少或者增加圖像的尺寸，可能會丟失圖像的某些部分，也有可能增加圖像的角點。

## match knn

**BFmatcher** (暴力匹配)：計算匹配圖層的一個特徵描述子與待匹配圖層的所有特徵描述子的距離返回最近距離。

**FlannBasedMatcher**：是目前最快的特徵匹配演算法（最近鄰搜尋）

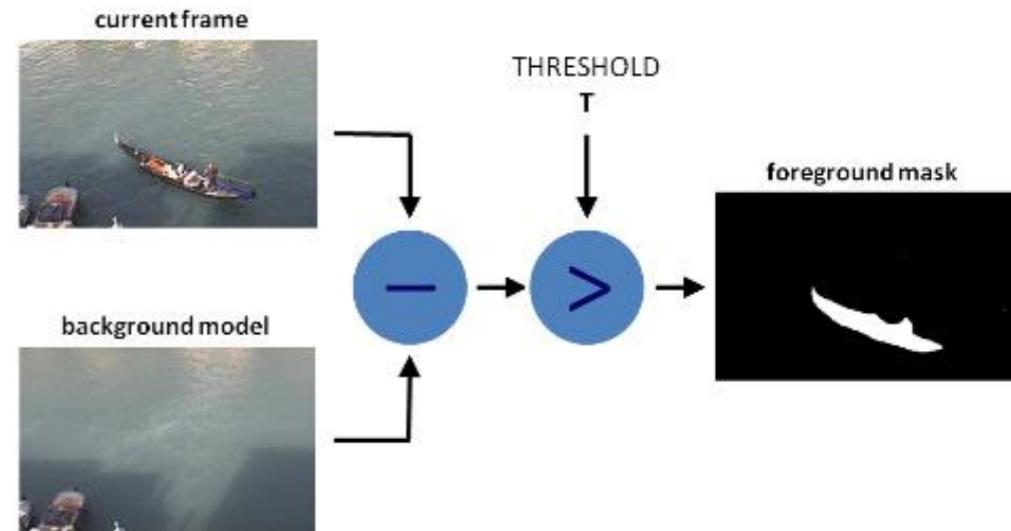
# Module 13. 背景 提取 & Hough 霍 夫變換

13-1: MOG 背景提取  
13-2: KNN 鄰近演算法  
13-3: Hough 霍夫轉換

背景減除(Background Subtraction)是許多基於計算機視覺的任務中的主要預處理步驟。如果我們有完整的靜止的背景幀,那麼我們可以通過幀差法來計算畫素差從而獲取到前景物件。但是在大多數情況下,我們可能沒有這樣的影象,所以我們需要從我們擁有的任何影象中提取背景。當運動物體有陰影時,由於陰影也在移動,情況會變的變得更加複雜。為此引入了背景減除演算法,通過這一方法我們能夠從視訊中分離出運動的物體前景,從而達到目標檢測的目的。OpenCV已經實現了幾種非常容易使用的演算法。

幀差法：

將連續兩幀的圖像資料進行差分法，即進行相減操作，如果其相減後的絕對值大於閾值，則圖元點變為255，否則變為0，通過這種方法來找出視頻中運動的物體



## 高斯混合模型分離演算法,全稱 Gaussian Mixture-based Background

它使用  $K$  ( $K=3$  或  $5$ ) 個高斯分佈混合對背景圖元進行建模。混合的權重表示這些顏色停留在場景中的時間比例，背景顏色是那些保持更長時間和更靜態的顏色。

在編寫代碼時，我們需要使用函數：`cv2.createBackgroundSubtractorMOG()`創建一個背景物件。這個函數有些可選參數，比如要進行建模場景的時間長度，高斯混合成分的數量，閾值等。將他們全部設置為預設值。然後在整個視頻中我們是需要使用 `backgroundsubtractor.apply()`就可以得到前景的遮罩圖。

## MOG2：高斯混合模型分離演算法,是MOG的改進演算法

這個演算法的一個特點是它為每一個圖元選擇一個合適數目的高斯分佈。（上一個方法中我們使用是 K 高斯分佈）。它能更好地適應光照不同等各種場景。和前面一樣我們需要創建一個背景物件。但在這裡我們我們可以選擇是否檢測陰影。如果 `detectShadows = True`（預設值），它就會檢測並將影子標記出來，但是這樣做會降低處理速度。影子會被標記為灰色。

這個也是以高斯混合模型為基礎的背景/前景分割演算法。它是以 2004 年和 2006 年 Z.Zivkovic 的兩篇文章為基礎的，分別是 "Improved adaptive Gaussian mixture model for background subtraction" 和 "Efficient Adaptive Density Estimation per Image Pixel for the Task of Background Subtraction"。這個演算法的一個特點是它為每一個圖元選擇一個合適數目的高斯分佈。（上一個方法中我們使用是 K 高斯分佈）。它能更好地適應光照不同等各種場景。和前面一樣我們需要創建一個背景物件。但在這裡我們我們可以選擇是否檢測陰影。如果 `detectShadows = True`（預設值），它就會檢測並將影子標記出來，但是這樣做會降低處理速度。影子會被標記為灰色。

**GMG** : 此演算法結合了靜態背景圖像估計和每個圖元的貝葉斯分割

它使用前面很少的圖像（預設為前 120 帖）進行背景建模。它採用概率前景分割演算法，使用貝葉斯推斷識別可能的前景物件。這是一種自我調整的估計，新觀察到的物件比舊的物件具有更高的權重，從而對光照變化產生適應。一些形態學操作如開運算閉運算等被用來除去不需要的噪音。在開始的前幾幀圖像中你會看到一個黑色視窗。

## 鄰近演算法，或者說K最近鄰(kNN，k-NearestNeighbor)分類演算法

是資料探勘分類技術中最簡單的方法之一。所謂K最近鄰，就是 $k$ 個最近的鄰居的意思，說的是每個樣本都可以用它最接近的 $k$ 個鄰居來代表。

kNN演算法的核心思想是如果一個樣本在特徵空間中的 $k$ 個最相鄰的樣本中的大多數屬於某一個類別，則該樣本也屬於這個類別，並具有這個類別上樣本的特性。該方法在確定分類決策上只依據最鄰近的一個或者幾個樣本的類別來決定待分樣本所屬的類別。kNN方法在類別決策時，只與極少量的相鄰樣本有關。由於kNN方法主要靠周圍有限的鄰近的樣本，而不是靠判別類域的方法來確定所屬類別的，因此對於類域的交叉或重疊較多的待分樣本集來說，kNN方法較其他方法更為適合。

## gradient – circles

```
circles = cv2.HoughCircles(
```

- image: 8位，單通道影象。如果使用彩色影象，需要先轉換為灰度影象。
- method : 定義檢測影象中圓的方法。目前唯一實現的方法是cv2.HOUGH\_GRADIENT。
- dp : 累加器解析度與影象解析度的反比。dp獲取越大，累加器陣列越小。
- minDist : 檢測到圓的中心，(x,y)座標之間的最小距離。如果minDist太小，則可能導致檢測到多個相鄰的圓。如果minDist太大，則可能導致很多圓檢測不到。
- param1 : 用於處理邊緣檢測的梯度值方法。
- param2 : cv2.HOUGH\_GRADIENT方法的累加器閾值。閾值越小，檢測到的圈子越多。
- minRadius : 半徑的最小大小（以畫素為單位）。
- maxRadius : 半徑的最大大小（以畫素為單位）。
- )

# Module 14. 直方圖處理

- 14-1: 直方圖的含意
- 14-2: 直方圖均衡化
- 14-3: 實作直方圖

## 14-1: 直方圖的含意

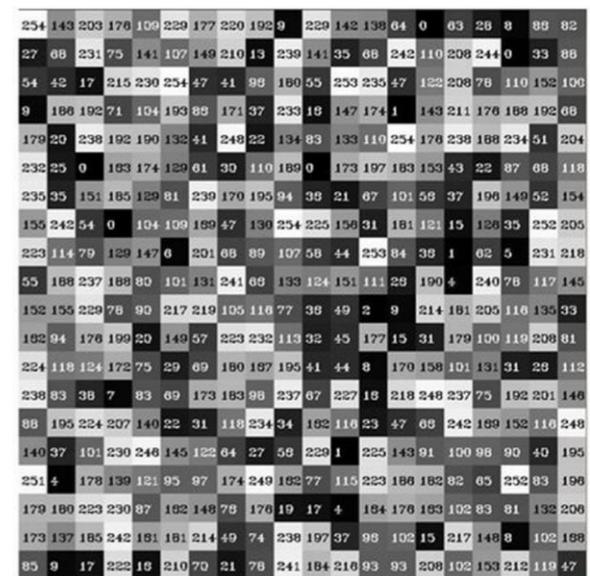
定義：直方圖反應了一張圖像內所有像素強度從0至255的分佈情形

特色：能掌握圖像的明暗、對比及像素強度分布

Histograms 是直方圖的意思，在攝影領域一般稱為曝光直方圖，主要用以分析一張照片的曝光是否正確。應用在電腦視覺則主要利用它來判斷白平衡、處理thresholding（閾值，請參考之前的文章OPENCV – more on contours #2），也可用於物體追蹤（例如CamShift演算法即使用彩色直方圖的變化達到跟蹤目的），另外也可用於圖像檢索目的（例如Bag-of-words詞袋演算法）、或應用於機器學習運算...等等。因為Histograms為我們統計出各像素強度頻率的資訊來呈現整張相片的色彩分佈情況，因此我們得以透過此資訊來猜測該圖片的物件及特性。

直方圖均衡化，，用於提高影像的質量，直方圖均衡化是通過拉伸畫素強度分佈範圍來增強影像對比度的一種方法。

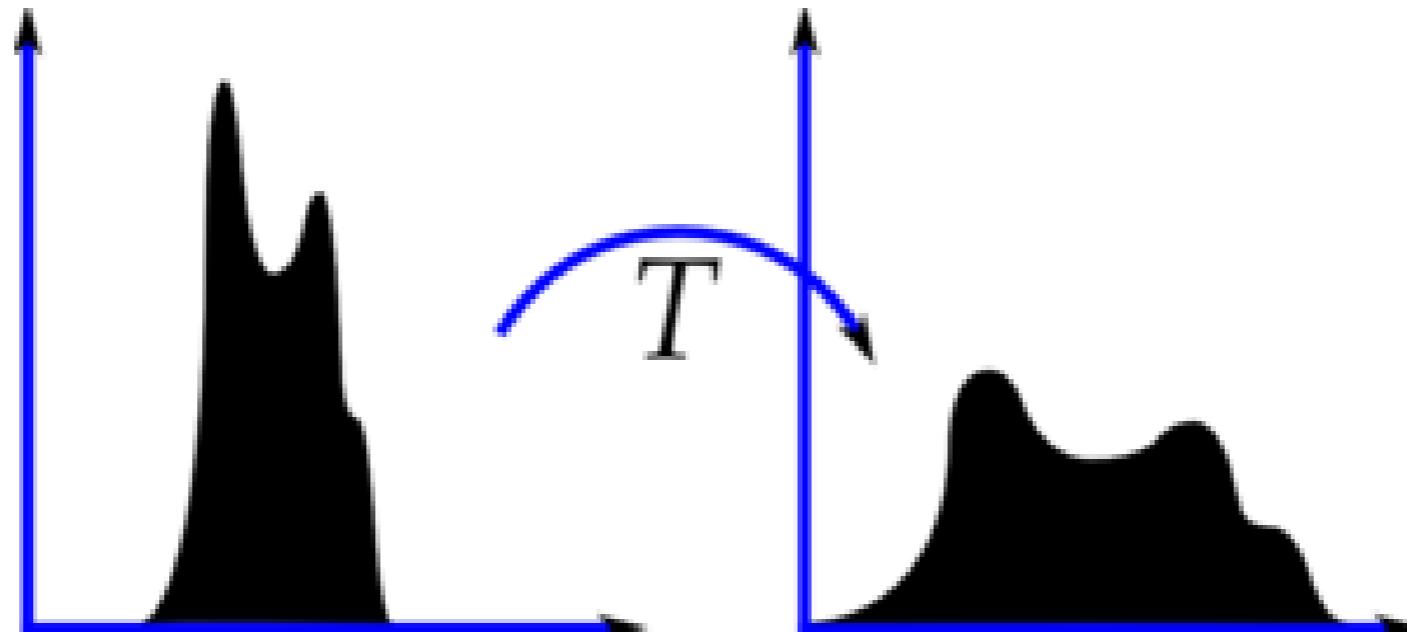
- 所謂直方圖就是對影像的中的這些畫素點的值進行統計，得到一個統一的整體的灰度概念。直方圖的好處就在於可以清晰瞭解影像的整體灰度分佈，這對於後面依據直方圖處理影像來說至關重要。
- 在開發影像處理的程式時，我們時常會需要觀察影像像素值的分佈與特性，以便選用適合的演算法、制定門檻值、設計出適合的影像處理流程。



## 14-1: 直方圖的含意

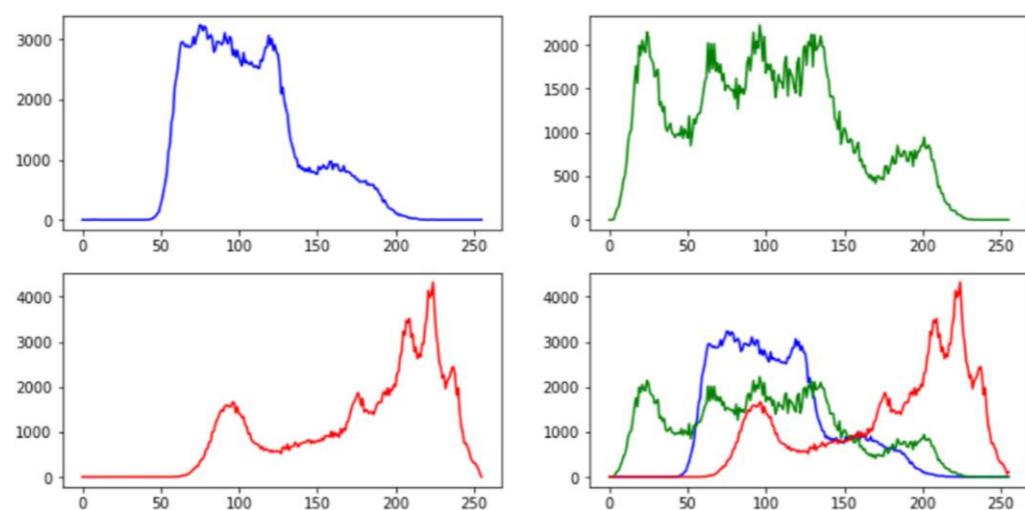
直方圖均衡化，，用於提高影像的質量，直方圖均衡化是通過拉伸畫素強度分佈範圍來增強影像對比度的一種方法。

- 所謂直方圖就是對影像的中的這些畫素點的值進行統計，得到一個統一的整體的灰度概念。直方圖的好處就在於可以清晰瞭解影像的整體灰度分佈，這對於後面依據直方圖處理影像來說至關重要。
- 在開發影像處理的程式時，我們時常會需要觀察影像像素值的分佈與特性，以便選用適合的演算法、制定門檻值、設計出適合的影像處理流程。



`cv2.calcHist(images, channels, mask, histSize, ranges)`

- `images`：要分析的圖片檔，其型別可以是 `uint8` 或 `float32`，變數必須放在中括號當中，例如：`[img]`。
- `channels`：產生的直方圖類型指定影像的通道（`channel`）。，同樣必須放在中括號當中例：`[0]→灰階，[0, 1, 2]→RGB三色`。
- `mask`：optional，若有提供則僅計算`mask`的部份。，若指定為 `None` 則會計算整張圖形的所有像素。
- `histSize`：要切分的像素強度值範圍，預設為256。每個`channel`皆可指定一個範圍（`bins`）。例如，`[32,32,32]` 表示RGB三個`channels`皆切分為32區段，也就是圖形畫出來要有幾條長方形。
- `ranges`：X軸(像素強度)的範圍，預設為`[0,256]`（非筆誤，`calcHist`函式的要求，最後那個值是表示`<256`）。

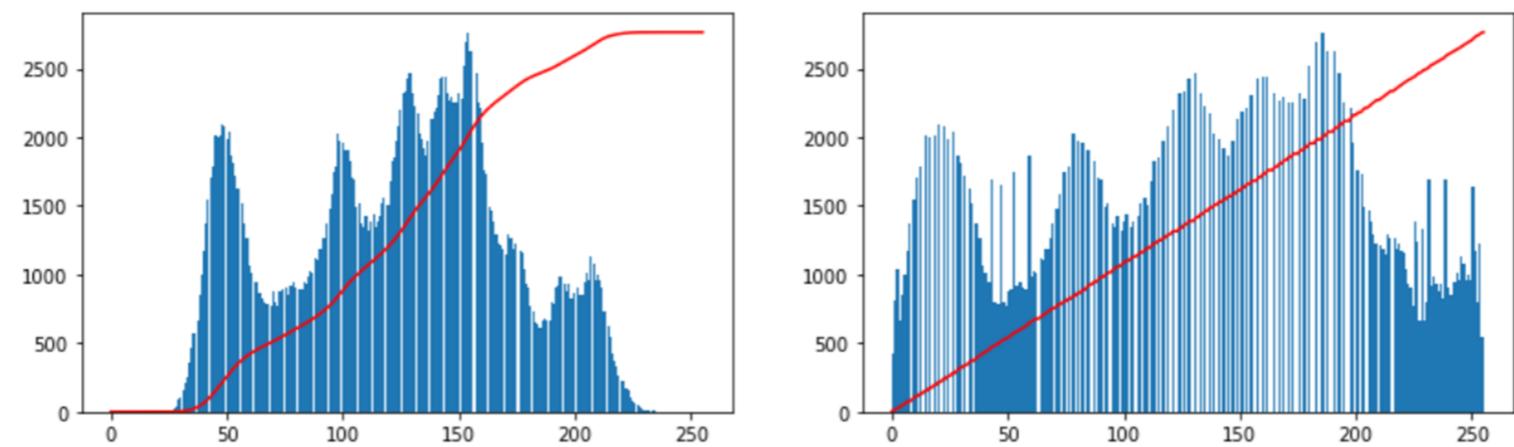


## Gray Level 直方圖均衡化處理實例 Histogram Equalization (HE)

定義：通過拉伸影像的像素強度分佈範圍來增強圖像對比度，適用於過曝或背光的圖片

缺點：對處理的數據不加選擇(全局處理)，如此一來會增加背景雜訊的對比度並且降低有用訊號(特別亮或暗)的對比度

經過比較可以發現均衡化後的圖像明亮的部分出現了過曝，失去了原本的細節。這是因為原圖整體暗的部份較多，而經過直方圖均衡化後使原本亮的地方更亮，



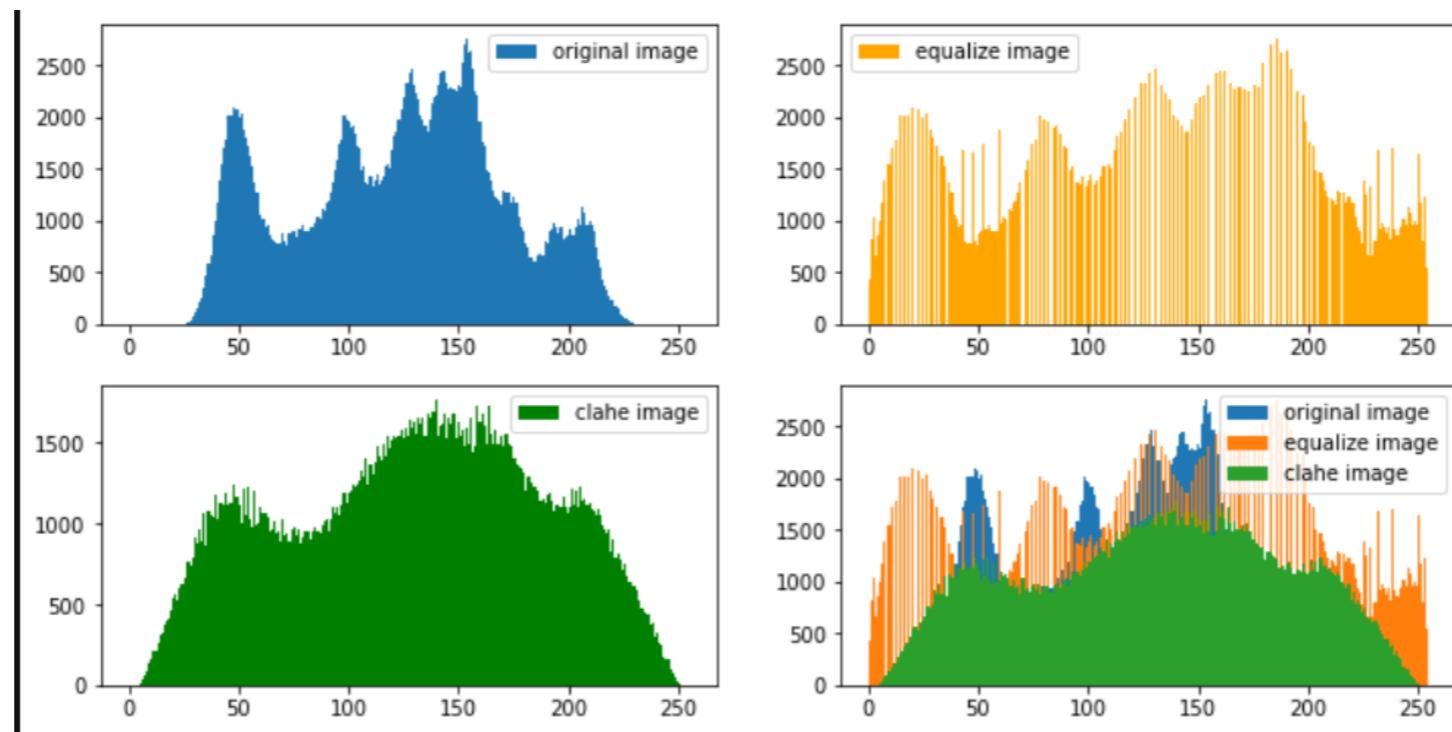
## 自適應直方圖均衡 Adaptive Histogram Equalization AHE

為了提高影像的區域性對比度，將影像分成若干子塊對子塊進行HE處理，這便是AHE（自適應直方圖均衡化）它在每一個小區域內（預設  $8 \times 8$ ）進行直方圖均衡化。當然，如果有噪點的話，噪點會被放大，需要對小區域內的對比度進行了限制。使用 AHE 處理上圖得到：

- 演算法：與一般的直方圖均衡（全局）相比，AHE 透過計算圖像每一個顯著區域的直方圖，重新分佈圖像的亮度值來改變影像對比度。
- 優點：適合於改善影像的區域性對比度以及獲得更多的影像細節。
- 缺點：在對比度增強的同時，也放大了影像的噪音

## 限制對比度自適應直方圖均衡 Contrast Limited Adaptive Histogram Equalization CLAHE

演算法：CLAHE與AHE都是局部均衡化，也就是把整個圖像分成許多小塊Tiles (OpenCV default為 $8 \times 8$ )，對每個小塊進行均衡化。這種方法主要對於圖像直方圖不是那麼單一的圖像(e.g. 多峰情況)比較實用。所以在每一個的區域中，直方圖會集中在某一個小的區域中



# Module 15. 視訊video處理

15-1:視訊預覽  
15-2:視訊讀寫  
15-3:視訊物件ROI, 追  
蹤和去背景

介紹如何使用 OpenCV 擷取網路攝影機影像，處理與顯示即時的畫面影像，並將連續的畫面影像寫入影片檔案中儲存起來。網路攝影機的串流影像，可以透過 OpenCV 模組的 VideoCapture 影片擷取功能來達成，至於寫入影片檔則可使用 VideoWriter

## play video file or camera preview

- `cap = VideoCapture([index, file])` 開啟相機裝置 index, 視訊檔案 file
- `retval = VideoCapture.isOpened() : True, False` 判斷視訊捕獲是否初始化成功。初始化成功返回 true
- `ret, frame = cap.read()` :
- `cap.release()`

Play in jupyter lab

```
from IPython.display import HTML  
  
HTML("""  
    <video alt="test" controls>  
        <source src=".//video/chaplin.mp4" type="video/mp4">  
    </video>  
""")
```

**imWrite : image writer**

- `cv2.WINDOW_NORMAL` : "可以" 讓使用者改變視窗大小
- `cv2.WINDOW_AUTOSIZE` : 使用者 "不可" 改變視窗大小
- `CAP_PROP_FRAME_WIDTH`,  
`CAP_PROP_FRAME_HEIGHT` : 取的影像的長寬
- `cv2.VideoCapture.set( propid, value )`

| 類別                                      | 值               | 說明  |
|---|-----------------|---|
| <code>cv2.CAP_PROP_POS_MSEC</code>      | <code>0</code>  | 視訊檔案的當前位置 (ms)  |
| <code>cv2.CAP_PROP_POS_FRAMES</code>    | <code>1</code>  | 從 <code>0</code> 開始索引幀，幀位置。                           |
| <code>cv2.CAP_PROP_POS_AVI_RATIO</code> | <code>2</code>  | 視訊檔案的相對位置 ( <code>0</code> 表示開始， <code>1</code> 表示結束) |
| <code>cv2.CAP_PROP_FRAME_WIDTH</code>   | <code>3</code>  | 視訊流的幀寬度   |
| <code>cv2.CAP_PROP_FRAME_HEIGHT</code>  | <code>4</code>  | 視訊流的幀高度   |
| <code>cv2.CAP_PROP_FPS</code>           | <code>5</code>  | 幀率  |
| <code>cv2.CAP_PROP_FOURCC</code>        | <code>6</code>  | 編解碼器四字元程式碼  |
| <code>cv2.CAP_PROP_FRAME_COUNT</code>   | <code>7</code>  | 視訊檔案的幀數   |
| <code>cv2.CAP_PROP_FORMAT</code>        | <code>8</code>  | <code>retrieve()</code> 返回的Mat物件的格式                   |
| <code>cv2.CAP_PROP_MODE</code>          | <code>9</code>  | 後端專用的值，指示當前捕獲模式                                       |
| <code>cv2.CAP_PROP_BRIGHTNESS</code>    | <code>10</code> | 影像的亮度，僅適用於支援的相機                                       |
| <code>cv2.CAP_PROP_CONTRAST</code>      | <code>11</code> | 影像對比度，僅適用於相機  |
| <code>cv2.CAP_PROP_SATURATION</code>    | <code>12</code> | 影像飽和度，僅適用於相機  |
| <code>cv2.CAP_PROP_HUE</code>           | <code>13</code> | 影像色調，僅適用於相機   |
| <code>cv2.CAP_PROP_GAIN</code>          | <code>14</code> | 影像增益，僅適用於支援的相機  |
| <code>cv2.CAP_PROP_EXPOSURE</code>      | <code>15</code> | 曝光，僅適用於支援的相機  |
| <code>cv2.CAP_PROP_CONVERT_RGB</code>   | <code>16</code> | 布林標誌，指示是否應將影像轉換為RGB                                   |

- track one obj in video file
- Track one obj in live Camera
- ROI (Region Of Interest) 只處理部分有興趣的
- framediff : 去背景 video

# Module 16. OpenCV 函式庫 DLib 介紹 ( Python )

- 16-1: DLib 影像辨識應用
- 16-2: DLib 套件應用
- 16-3: DLib 特徵點描述

Dlib 使用的人臉偵測演算法是以方向梯度直方圖（ HOG ）的特徵加上線性分類器（ linear classifier ）、影像金字塔（ image pyramid ）與滑動窗格（ sliding window ）來實作的

dlib是一套包含了機器學習、計算機視覺、圖像處理等的函式庫，使用C++開發而成，目前廣泛使用於工業及學術界，也應用在機器人、嵌入式系統、手機、甚至於大型的運算架構中，而且最重要的是，它不但開源且完全免費，而且可跨平台使用（ Linux 、 Mac OS 、 Windows ），並且除了C++之外還提供了Python API，因此如果我們想要建立一套物件偵測系統，dlib是相當適合的平台。

```
pip install dlib==19.8.1 ( or dlib-19.8.1-cp36-cp36m-win_amd64.whl ) (**python 3.6.8**)
```

### Python 3.6.8 + dlib 19.8.1

在 (命令提示字元 / Windows PowerShell \*\*系統管理員\*\*) 身分下打

`pip install dlib==19.8.1`

( or `dlib-19.8.1-cp36-cp36m-win_amd64.whl` ) (\*\*python 3.6.8\*\*)

## 16-3: DLib 特徵點描述

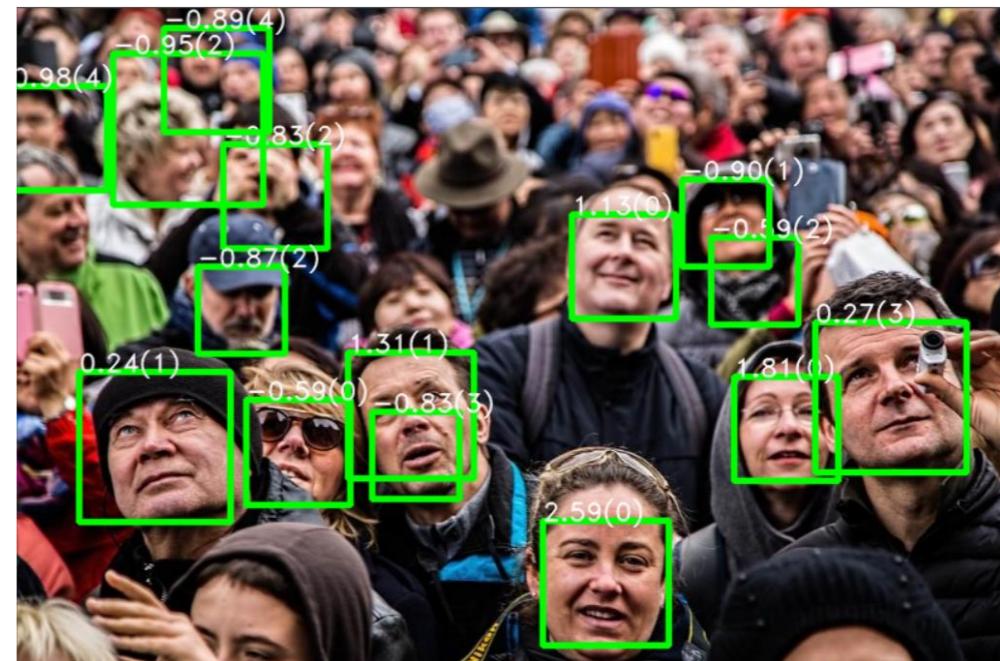
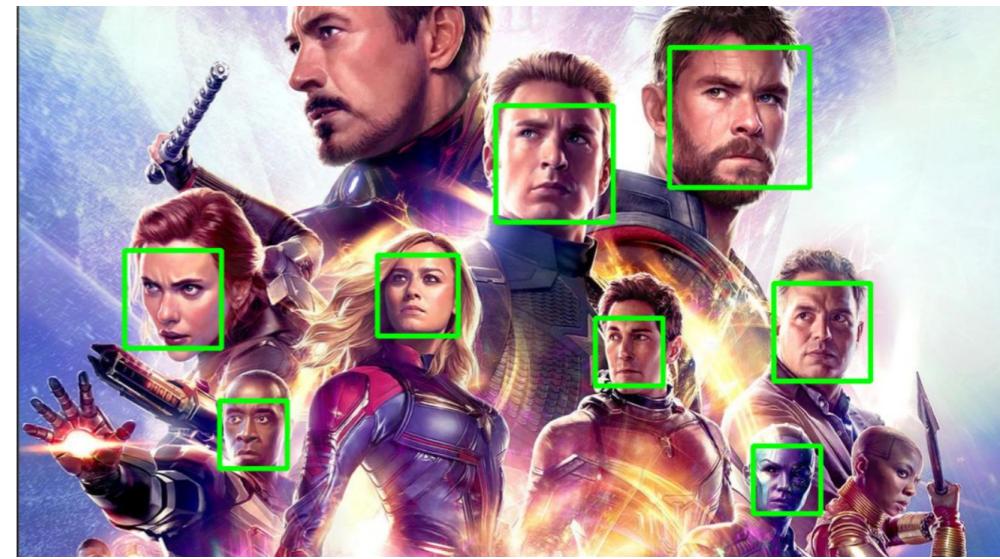
### 照片人臉偵測

Dlib 的人臉偵測器

```
detector = dlib.get_frontal_face_detector()
```

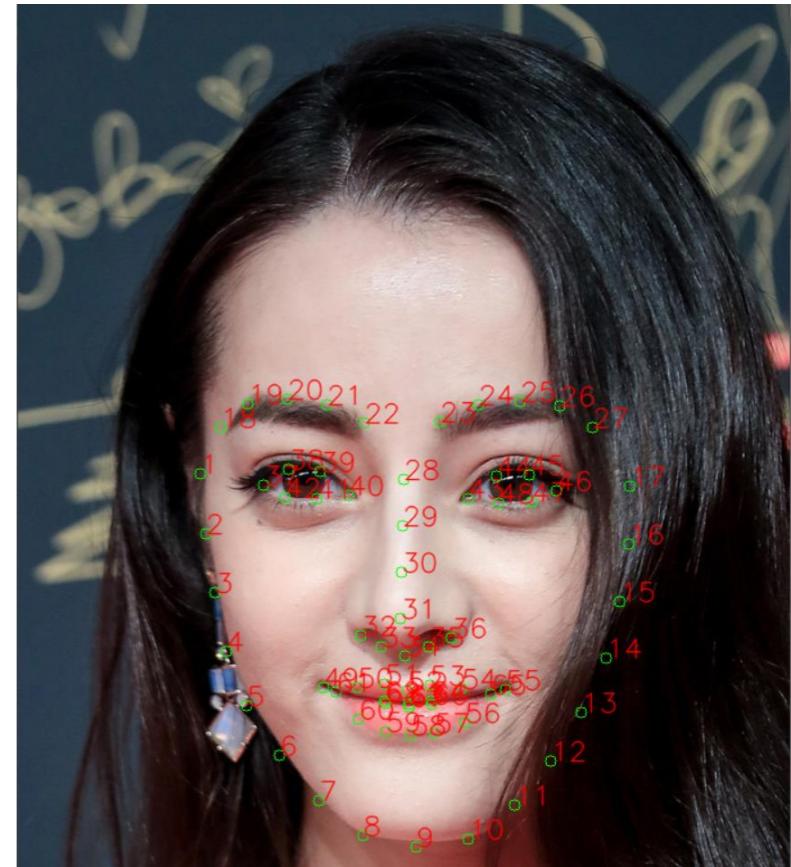
### 偵測結果與分數

改用 `detector.run` 來偵測人臉，它的第三個參數是指定分數的門檻值，所有分數超過這個門檻值的偵測結果都會被輸出，而傳回的結果除了人臉的位置之外，還有分數 ( `scores` ) 與子偵測器的編號 ( `idx` )，子偵測器的編號可以用來判斷人臉的方向，請參考 Dlib 的說明



- 影片人臉偵測
- 即時串流影像人臉偵測
- 實現人臉68個關鍵點檢測

另外dlib也提供訓練好的模型，可以辨識出人臉的68的特徵點，68特徵點包括鼻子、眼睛、眉毛，以及嘴巴等等，如上圖紅點就是偵測出人臉的68個特徵點。



## dlib - Obj detect

- 先收集欲辨識的人正面照片 (甚至可以收集多角度照片) 於特定資料夾 (此處設為 ./rec ) 中，並將檔名設為人名。
- 使用 `dlib.get_frontal_face_detector()` 擷取資料夾中照片的人臉，再利用 `dlib.shape_predictor()` 取出臉部 68 個關鍵點。
- `dlib.face_recognition_model_v1().compute_face_descriptor()` 將 68 個關鍵點進行嵌入成一個 128 維的向量  $v_1, v_2, \dots$ 。
- 相同的方式將鏡頭中的人臉也嵌入成 128 維的向量  $v$ 。
- 計算  $v$  與  $v_1, v_2, \dots$  個別計算歐式距離，最接近者及判定其身分。

# Module 17. OCR 光學字元 識別

- 17-1 OCR 介紹
- 17-2 Tesseract 安裝
- 17-3 實作光學字元辨識

OCR 為光學文字識別的縮寫 ( Optical Character Recognition , OCR ) , 白話一點就是將圖片翻譯為文字。而 **Tesseract** 是一個 **OCR 模組** , 目前由 Google 贊助。Tesseract 已經有 30 年歷史 , 一開始它是惠普實驗室的一款專利軟體 , 於 2005 年開源 , 從 2006 年後由 Google 贊助進行後續的開發和維護 , Tesseract 也是目前公認最優秀、最精準的開源 OCR 系統



Tesseract目前已作為開源項目發佈在Google Project , 其最新版本 3.0已經支持中文OCR , 並提供了一個命令行工具。

主要使用在辨識掃描文件/圖片的文字 , 包含契約、發票等等 , 能夠輕鬆地減少需要人力的工作 , 例如像 RPA(Robotic Process Automation)類型的專案可能都會使用到。

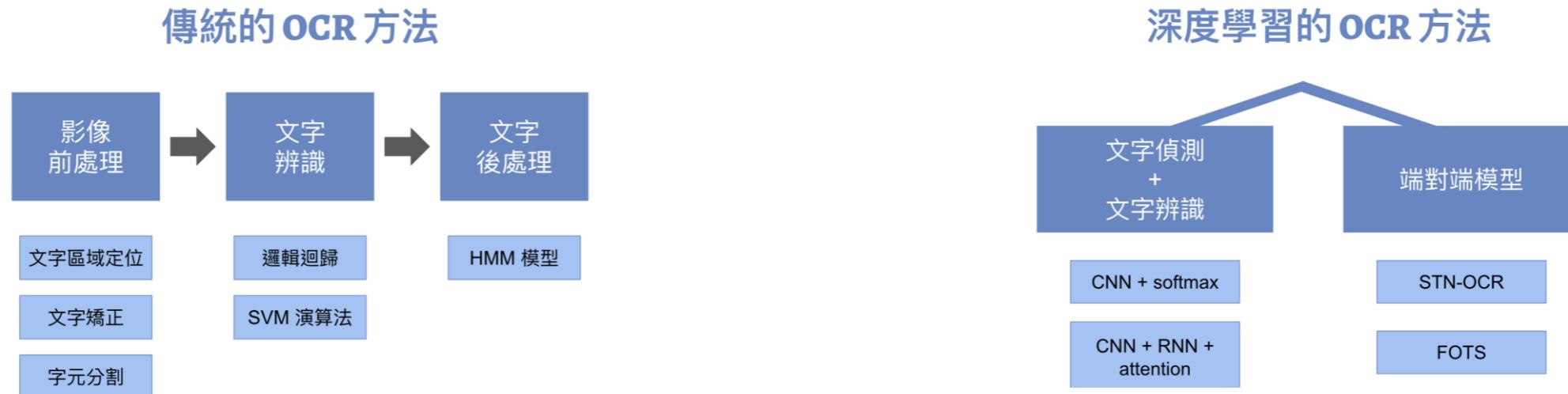
如何除錯或利用輔助資訊提高識別正確率 , 是OCR最重要的課題 , ICR ( Intelligent Character Recognition ) 的名詞也因此而產生。衡量一個OCR系統性能好壞的主要指標有 : 拒識率、誤識率、識別速度、使用者介面的友好性 , 產品的穩定性 , 易用性及可行性等。

OCR 光學字元辨識



在一個24位彩色影像中，每個畫素由三個位元組表示，通常表示為RGB。通常，許多24位彩色影像儲存為32點陣圖像，每個畫素多餘的位元組儲存為一個alpha值，表現有特殊影響的資訊。在RGB模型中，如果R=G=B時，則彩色表示一種灰度顏色，其中R=G=B的值叫灰度值，因此，灰度影像每個畫素只需一個位元組存放灰度值（又稱強度值、亮度值），灰度範圍為0-255。這樣就得到一幅圖片的灰度圖。

除了極高的精準度外，Tesseract 也有很高的靈活性，能夠通過訓練識別出任何字體（只要這些字體的風格不變就可以），也能識別出任何 Unicode 字符，是不是非常厲害呢？我們待會會用到的 pytesseract 模組就像是Tesseract的 python 包裝器。



Tesseract目前已作為開源項目發佈在Google Project，其最新版本3.0已經支持中文OCR，並提供了一個命令行工具。

主要使用在辨識掃描文件/圖片的文字，包含契約、發票等等，能夠輕鬆地減少需要人力的工作，例如像RPA(Robotic Process Automation)類型的專案可能都會使用到。

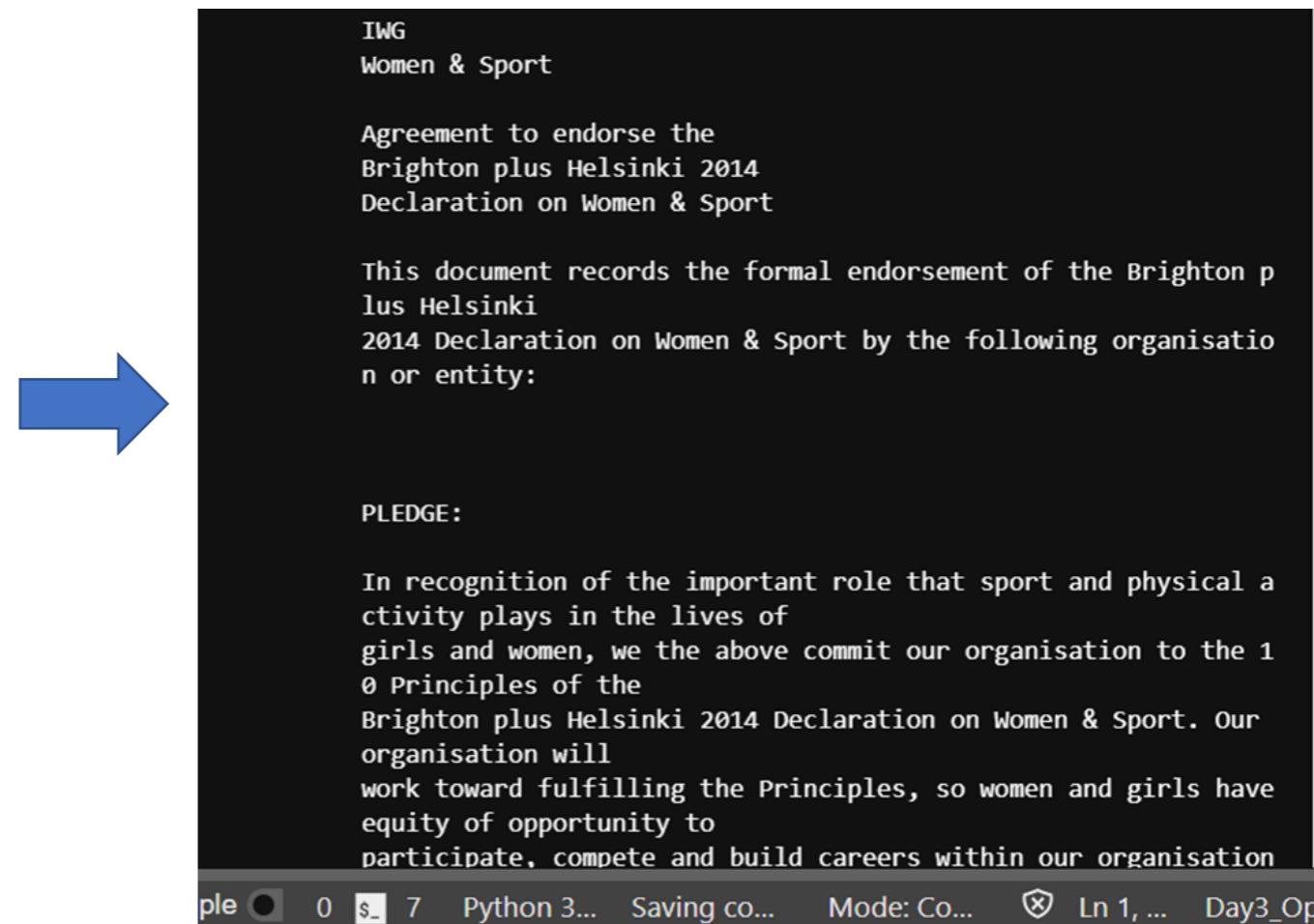
到此網站下載中文的語言辨識包

[https://github.com/tesseract-ocr/tessdata\\_best](https://github.com/tesseract-ocr/tessdata_best)

- `chi_sim.traineddata` -> 簡體中文包
- `chi_tra.traineddata` -> 繁體中文包

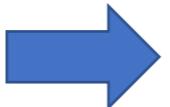
- pip install pytesseract
- <https://github.com/UB-Mannheim/tesseract/wiki>
- run : **tesseract-ocr-w64-setup-v5.0.0-alpha.20201127.exe**
- 安裝好後找到 pytesseract.exe 的位置，並複製其絕對路徑，通常會在 C:\Program Files\Tesseract-OCR\tesseract.exe 。

## 英文辨識



## 繁中辨識

「春遊浩蕩，是年年寒食，梨花時節。白錦無紋香爛漫，玉樹瓊苞堆雪。靜夜沉沉，浮光靄靄，冷浸溶溶月。人間天上，爛銀霞照通微。潭似姑射真人，天姿靈秀，意氣殊高潔。萬蕊參差誰信道，不與群芳同列。浩氣清英，仙才卓犖，下土難分別。瑤台歸去，洞天方看清楚。」



作這一首《無俗念》詞的，乃南宋末年一位武學名家，有道之士。此人姓丘，名處機，道號長春子，名列全真七子之一，是全真教中出類拔萃的人物。《詞品》評論此詞道：「長春，世之所謂仙人也，而詞之清拔如此」。這首詞誦的似是梨花，其實詞中真意卻是讚譽一位身穿白衣的美貌少女，說她「潭似姑射真人，天姿靈秀，意氣殊高潔」，又說她「浩氣清英，仙才

上註 『丁卯御批詩同列』，即宋丘處機詩集，四十首，注音，小字注解，此一卷，共四百首。

「春遊浩蕩，是年年寒食，梨花時節。白錦無紋香爛漫，玉樹瓊苞堆雪。靜夜深沉，浮光靄靄，冷浸溶溶月。人間天上，爛銀霞照通微。潭似姑射真人，天姿靈秀，意氣殊高潔。萬蕊參差誰信道，不與群芳同列。浩氣清英，仙才卓犖，下土難分別。瑤台歸去，洞天方看清楚。」

作這一首《無俗念》詞的，乃南宋末年一位武學名家，有道之士。此人姓丘，名處機，道號長春子，名列全真七子之一，是全真教中出類拔萃的人物。《詞品》評論此詞道：「長春，世之所謂仙人也，而詞之清拔如此」。這首詞誦的似是梨花，其實詞中真意卻是讚譽一位身穿白衣的美貌少女，說她「潭似姑射真人，天姿靈秀，意氣殊高潔」，又說她「浩氣清英，仙才卓犖」。「不與群芳同列」。詞中所領這美女，乃古墓派傳人小龍女，她一生愛穿白衣，當真如風拂玉樹，雪裏瓊苞，兼之生性清冷，實當得起「冷浸溶溶月」的形容。以「無俗念」三字贈之，可說十分貼切。長春子丘處機和她在終南山上比鄰而居，當年一見，便寫下這首詞來。

Tesseract 命令列參數最重要的三個是 -l, --oem, --psm

-l : 控制輸入文本的語言，用 eng 表示英文（預設語言），用 chi\_sim 表示中文簡體，用 chi\_tra 表示中文繁體。點擊 [這裡](#) 查看並下載更多支援的語言。

--oem : OCR Engine modes，Tesseract 有兩個OCR引擎，使用 -oem 選擇演算法類型，有四種操作模式可供選擇。

| 值 | 說明                                  |
|---|-------------------------------------|
| 0 | Legacy engine only                  |
| 1 | Neural nets LSTM engine only        |
| 2 | Legacy + LSTM engines               |
| 3 | Default, based on what is available |

用 --oem 1 表示我們希望只使用LSTM neural network。

--psm : Page segmentation modes，控制 Tesseract 使用的自動頁面分割模式。

## Page segmentation modes:

| 值  | 說明  |
|----|---|
| 0  | Orientation and script detection (OSD) only   |
| 1  | Automatic page segmentation with OSD  |
| 2  | Automatic page segmentation, but no OSD, or OCR (not implemented)                             |
| 3  | Fully automatic page segmentation, but no OSD (Default)                                       |
| 4  | Assume a single column of text of variable sizes  |
| 5  | Assume a single uniform block of vertically aligned text                                      |
| 6  | Assume a single uniform block of text   |
| 7  | Treat the image as a single text line   |
| 8  | Treat the image as a single word  |
| 9  | Treat the image as a single word in a circle  |
| 10 | Treat the image as a single character   |
| 11 | Sparse text. Find as much text as possible in no particular order                             |
| 12 | Sparse text with OSD  |
| 13 | Raw line. Treat the image as a single text line, bypassing hacks that are Tesseract-specific. |

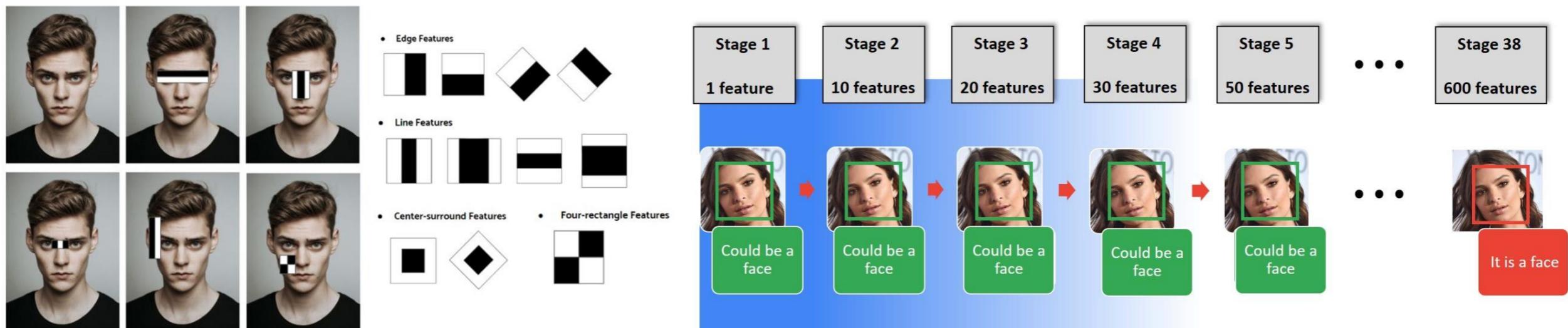
# Module 18. 人臉辨識

18-1: 人臉偵測介紹  
18-2: 人臉辨識介紹  
18-3: OpenCV 實作人  
臉辨識

# 18-1: 人臉偵測介紹

一張臉，會有超過 6000 個 classifiers. (這數字是在某篇文章看到的，真假不知). 如果要辨識一張圖裡面有幾張人臉，不就等於是要把圖裡面的全部區塊，都分別跑完 6000 個 classifiers 比對. 這會需要很多很多的運算資源與時間.

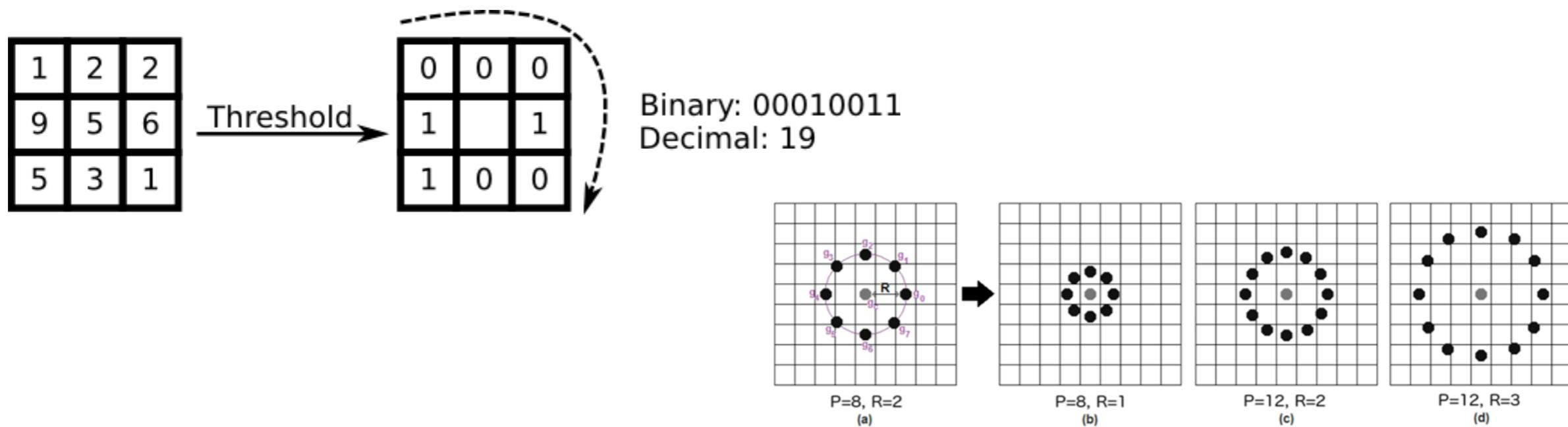
Cascades (瀑布式) 是把人臉辨識的 6000 個 classifiers 分成數個階段. 比對圖區塊時，都從第一個階段做辨識，若沒通過就淘汰不用往下做，有通過（可能是臉）才繼續做下階段的辨識，直到通過全部的辨識，才判斷為人臉。這種類似瀑布一段一段往下做法的好處是～節省運算資源與時間，不用每個圖區塊都得做足 6000 個辨識。



## HaarCascade

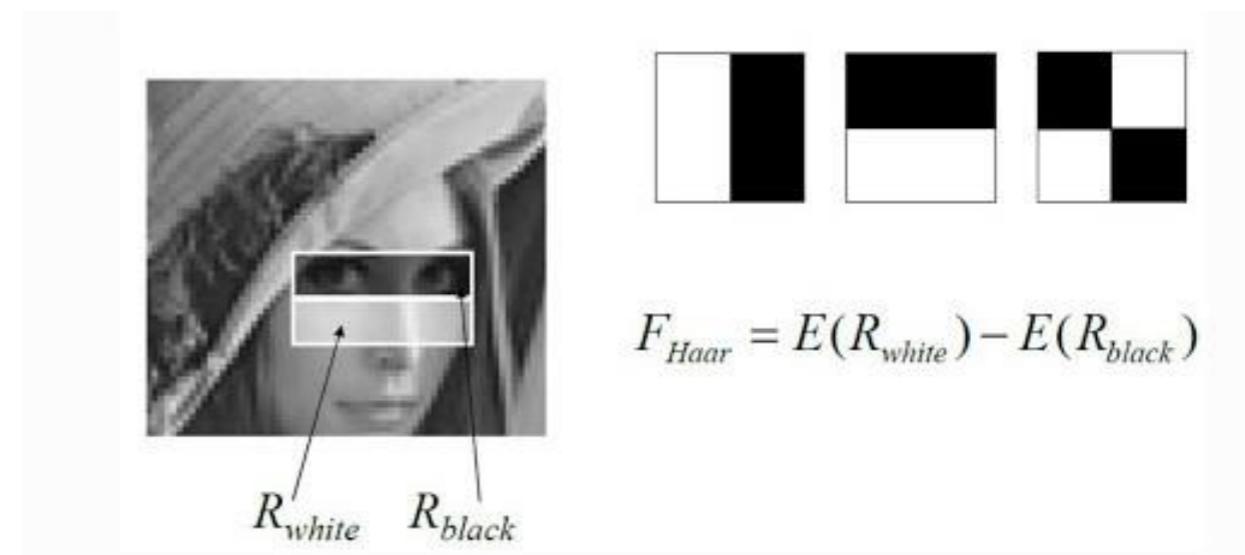
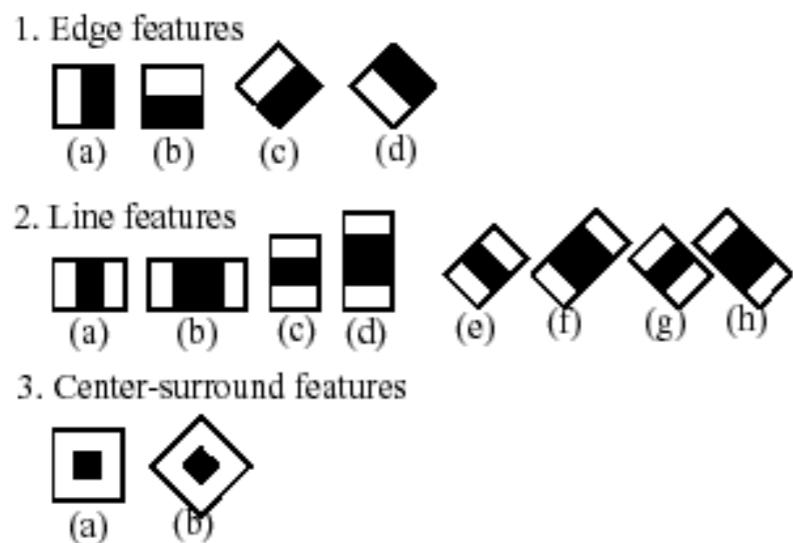
LBPH : 愈低愈好, 低於 50 可接受

原始的LBP算子定義為在  $3 \times 3$  的窗口內，以窗口中心像素為閾值，將相鄰的8個像素的灰度值與其進行比較，若周圍像素值大於或等於中心像素值，則該像素點的位置被標記為1，否則為0。這樣， $3 \times 3$  鄰域內的8個點經比較可產生8位二進位數(通常轉換為十進位數即LBP碼，共256種)，即得到該窗口中心像素點的LBP值，並用這個值來反映該區域的紋理特徵。如下圖所示：



## EigenFaceRecognier

參數num\_components :在PCA中要保留的分量個數。當然,該參數值通常要根據輸入資料來具體確定,並沒有一定之規。一般來說,85個分量就足夠了



Cascade classifier for Haar features 有很多種類; 人臉, 眼睛, 耳朵, 嘴... 甚至可以自己訓練一套專屬用途的。

### haar cascades 資源

- haarcascade\_frontalface\_default.xml 人臉正面與側面
- haarcascade\_frontalface\_alt2.xml 人臉正面效果較好
- haarcascade\_profileface.xml 人臉側面效果較好
- haarcascade\_eye.xml 眼睛偵測

# Thank you !