Embedded Vision Intelligent Laboratory

多媒體技術與應用
Spring 2021

Instructor： Yen-Lin Chen(陳彥霖), Ph.D.
Professor
Dept. Computer Science and Information Engineering
National Taipei University of Technology

Lecture 2

OpenCV基礎影像處理技術

OpenCV Image Processing

二值化(Binarization)-簡介

- 二值化(Binarization)是影像分割的一種方法，我們會將影像分為兩個部分，一個是感興趣的部分(前景)，以及不感興趣的部分(背景)，會依據門檻值(threshold)當作分割的標準，通常會以強度超過門檻值的像素當作前景，反之則為背景。
- 門檻值的算法主要分兩類：
  - 固定門檻值：直接給定一個灰階值當門檻值，再用這個門檻值進行二值化。
  - 自適應門檻值(Otsu決定法)：會依據輸入影像計算出較合適的門檻值，再用這個門檻值進行二值化。

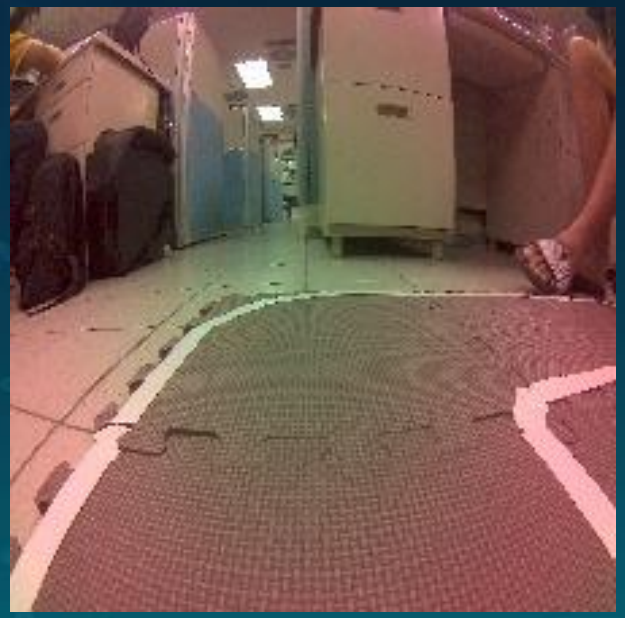
二值化-利用門檻值進行影像分割

- 在一灰階影像當中，每一獨立區塊的產生取決於亮度（灰階值）的不同，利用這樣的特性，我們可以設立一固定的門檻值T，以該值為條件，將影像轉換為二值影像（只包含0及1之二值影像，或0及255之灰階影像），並可協助我們在後續處理中擷取出感興趣的影像區塊。

$$g(x,y)=\begin{cases}1 & f(x,y)\geq T\\ 0 & f(x,y)<T\end{cases}$$

$$g(x,y)=\begin{cases}0 & f(x,y)\geq T\\ 1 & f(x,y)<T\end{cases}$$

二值化-利用門檻值對車道影像進行分割



$g(x,y) \geq 170$

$$g(x,y) = \begin{cases} 1 & f(x,y) \geq 170 \\ 0 & f(x,y) < 170 \end{cases}$$

二值化-OpenCV

OpenCV固定門檻值(threshold)二值化

- retval, dst = cv2.threshold(<src>, <thresh>, <maxval>, <type>[, <dst>])
  - retval：只是一個return值，並不會使用到
  - dst：輸出影像(ndarray)，尺寸大小、深度會和輸入影像相同。
  - src：輸入影像(ndarray) ，只能輸入**單通道**，8位元或32位元浮點數影像。
  - thresh：門檻值。
  - maxval：二值化結果的最大值。
  - type：二值化操作型態，共有THRESH_BINARY、THRESH_BINARY_INV、THRESH_TRUNC、THRESH_TOZERO、THRESH_TOZERO_INV五種。
  - 例：ret, img_out = cv2.threshold(img, 170, 255, cv2.THRESH_BINARY)
  - https://docs.opencv.org/4.1.0/d7/d1b/group__imgproc__misc.html#gae8a4a146d1ca78c626a53577199e9c57

二值化-OpenCV

常用五種門檻值參數介紹：

threshold門檻值參數	說明
cv2.THRESH_BINARY	大於門檻值的像素設為最大值(maxval)，否則設為0。
cv2.THRESH_BINARY_INV	大於門檻值的像素設為0，否則設為最大值(maxval)。
cv2.THRESH_TRUNC	大於門檻值的像素設為門檻值，否則不變。
cv2.THRESH_TOZERO	大於門檻值的像素值不變，否則設為0。
cv2.THRESH_TOZERO_INV	大於門檻值的像素值設為0，否則不變。
cv2.THRESH_OTSU	自適應決定門檻值，大於門檻值的像素設為最大值(maxval)，否則不變。

二值化-OpenCV

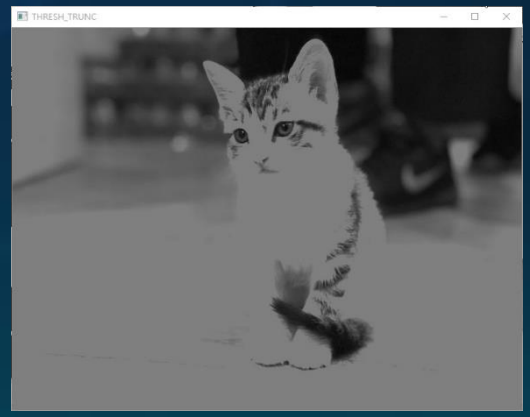
```python
import cv2
import numpy as np

img = cv2.imread("cat.jpg", 0)
ret, out1 = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)
ret, out2 = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY_INV)
ret, out3 = cv2.threshold(img, 127, 255, cv2.THRESH_TRUNC)
ret, out4 = cv2.threshold(img, 127, 255, cv2.THRESH_TOZERO)
ret, out5 = cv2.threshold(img, 127, 255, cv2.THRESH_TOZERO_INV)
ret, out6 = cv2.threshold(img, 127, 255, cv2.THRESH_OTSU)

cv2.imshow("THRESH_BINARY", out1)
cv2.imshow("THRESH_BINARY_INV", out2)
cv2.imshow("THRESH_TRUNC", out3)
cv2.imshow("THRESH_TOZERO", out4)
cv2.imshow("THRESH_TOZERO_INV", out5)
cv2.imshow("THRESH_OTSU", out6)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

- 範例程式:
  - [4]以灰階讀入圖片(後面參數0)
  - [5] THRESH_BINARY:將大於門檻值的像素設為255，否則設為0。
  - [6] THRESH_BINARY_INV:大於門檻值的像素設為0，否則設為255。
  - [7] THRESH_TRUNC:大於門檻值的像素設為127，否則不變。
  - [8] THRESH_TOZERO:大於門檻值的像素不變，否則設為0
  - [9] THRESH_TOZERO_INV:大於門檻值的像素設為0，否則不變
  - [10] THRESH_OTSU:自適應決定門檻值，大於門檻值的像素設為255，否則不變。

二值化-OpenCV

THRESH_BINARY

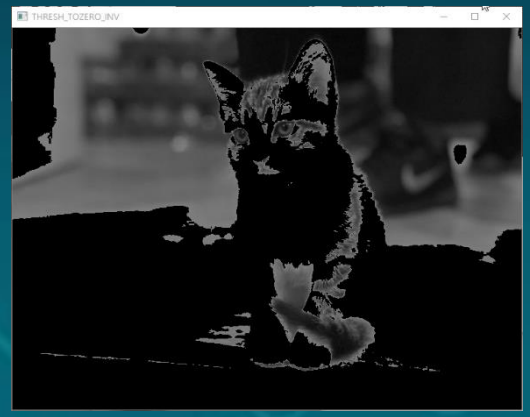
THRESH_BINARY_INV



THRESH_BINARY_TRUNC

THRESH_TOZERO

THRESH_TOZERO_INV



THRESH_OSTU

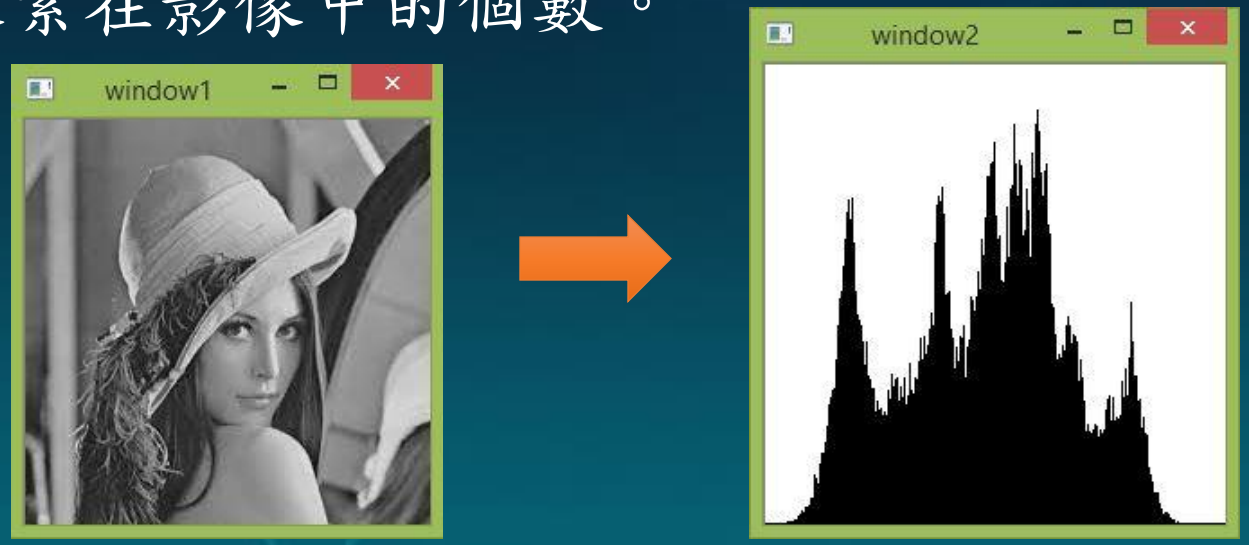
影像增強(Image Enhancement)

- 清晰的影像，是指能夠清楚反映被攝景物的明亮程度和細微色彩差別的影像。
- 應用實例：當夜間拍攝路面號誌或車道線時，因色彩相似或亮度不足以致融入背景等情況下，影像就很難看的清楚。對於這種影像，如果把號誌或線條的色彩和亮度進行適當處理，使其與背景產生微秒的差別，就可以使所需特徵更加明確，以供後續辨識作業。
- 像這樣通過增強亮度或色彩等各種包含於影像中的資訊，**如使影像顯得更美觀、易於判讀、突出某些部位等，而加強影像的某些特點**，這種處理過程稱為影像增強。

影像增強-範例

直方圖(Histogram)-簡介

- 直方圖是(Histogram)一個表現影像中像素分布的統計表，橫軸為影像中所有的像素值，假設是8位元影像，那其範圍為0到255，縱軸每個像素在影像中的個數。



- 直方圖是影像的一個重要特性，可以透過直方圖觀察影像中像素值的變化，看出影像是否太暗或過曝，又或者分布太過集中。

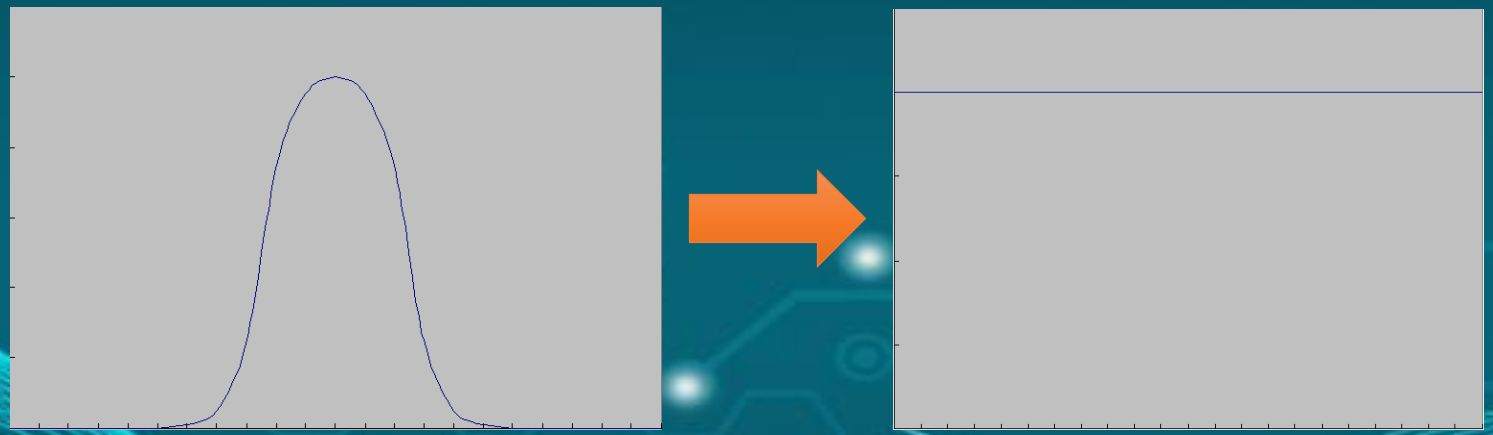
直方圖-OpenCV

OpenCV calcHist()函式：

- hist = cv2.calcHist(<images>, <channels>, <mask>, <histSize>, <ranges>[, <hist>[, <accumulate>]])
  - images：輸入影像，可以一個或多個影像，每張影像的尺寸和深度必須相同。
  - channels：直方圖通道。
  - mask：遮罩(可有可無)。
  - hist：輸出的直方圖。
  - dims：直方圖的維度，必須為正數。
  - histSize：直方圖橫軸數目。
  - ranges：直方圖的強度範圍，以8位元無負號的影像為例子:範圍為[0, 255]。
  - uniform：各維度取值是否一致。
  - accumulate：如果設定為true的話，在呼叫calcHist()這函式的時候，hist的內容不會被清掉。
  - 例: hist = cv2.calcHist(gray_img, 0, None, 256, [0, 256])

直方圖等化(Histogram Equalization)-簡介

- 我們可透過拉伸直方圖，使直方圖覆蓋所有強度範圍，這種方法的確能提高影像對比度，但是在多數情況，影像模糊不是因為過窄的強度範圍，而是某區間的像素強度比例過高，這時可以製作一個映射表，使得調整之後的影像，能平均使用所有的強度，進而增加影像的整體對比度。
- 要把灰階直條圖等化，只需要把原影像中像素少的部份進行壓縮，而將像素數較多的部份拉伸開來即可。



直方圖等化-流程

- 計算輸入影像的直方圖。
- 將直方圖歸一到所有bin的總合為255。
- 計算直方圖累計表(CDF)。
- 用直方圖累計表完成各強度的映射，所以假設強度30所累積的比例為20%，映射的強度即為255×0.2，由於我們直方圖歸一化到255，所以假設強度30所累積的值為20，映射的強度即為20。

直方圖等化-演算法

- 下面利用一個簡單的例子來說明平坦化的演算法。先考慮在灰階為0~7之處，各有如下圖所示數量的像素的情況。本例中為像素數平均值40÷8=5。然後，從原影像灰階最高處開始，每5個像素為一組，順序分配給新的灰階級，本例中是從新的灰階7開始分配。

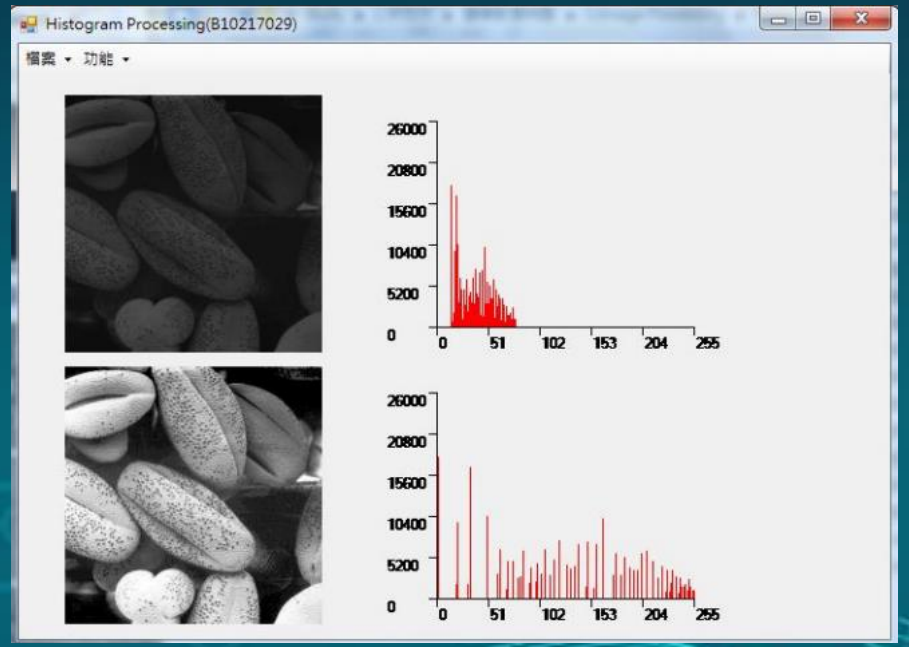
灰階級	7	6	5	4	3	2	1	0
原影像的各級像素數	0	4	9	11	5	7	4	0
琮輝階級高處開始，每5個像素為一組，依順序分配給新的灰階級	0	4	1	5	3	2	5	
等化後的各級的像素數	5	5	5	5	5	5	5	5

- 而從灰階為5處的像素中，選擇出1個像素的方法有兩種：
  1) 隨機選擇。
  2) 從周圍平均灰階高的像素開始順序選擇
- 從演算法上考慮，(1)比(2)稍顯複雜，而(2)與(1)相比，具有雜訊數較少的特點。

直方圖等化-OpenCV

OpenCV直方圖等化函式

- dst = cv2.equalizeHist(<src>[, <dst>])
  - src：輸入影像，8位元單通道圖。
  - dst：輸出影像，和輸入影像尺寸、型態相同。
  - 例：img_out = cv2.equalizeHist(img)



Otsu(自適應門檻值)-簡介

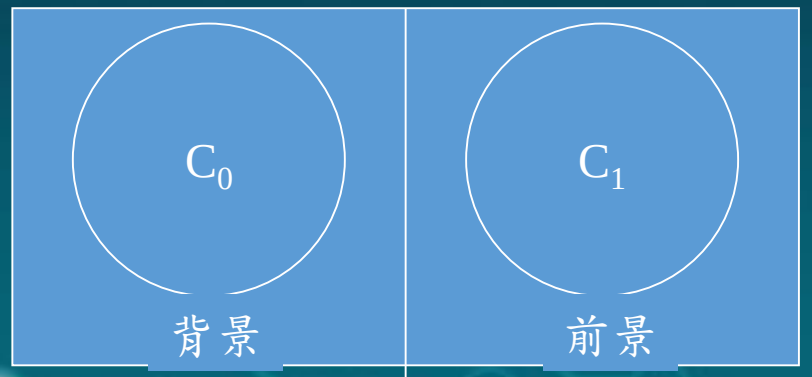
- Otsu演算法又稱為「大津演算法」，是一種自動的門檻值決定法則，常用在選擇影像二值化的門檻值上，當然Otsu並不是只能使用在影像二值化上，只要是選擇門檻值都能使用Otsu。
- Otsu 的這個方法在許多應用上時常被提起，主要是應用於影像的二值化處理，這個方法可以自動找出一個適當臨界值，讓不同的群組分開。

Otsu(自適應門檻值)-流程

- 先計算影像的直方圖。
- 把直方圖強度大於門檻值的像素分成一組，把小於門檻值的像素分成另一組。
- 分別計算這兩組的組內變異數，並把兩個組內變異數相加。
- 將0-255依序當作門檻值來計算組內變異數和，總和值最小的就是結果門檻值。

Otsu(自適應門檻值)決定法(1-1)

- 底下將以決定一個門檻值為例，來帶出Otsu的想法。假若T^*為最佳門檻值，我們利用T^*把一影像分成C_0及C_1前後景二區，而該T^*值的決定將會滿足下列兩項條件：
  1. C_0及C_1之間的「類別間變異數Between-class variance」為最大。
  2. C_0內的變異數加上C_1內的「類別內變異數Within-class variance」之和為最小。



Otsu(自適應門檻值)決定法(1-2)

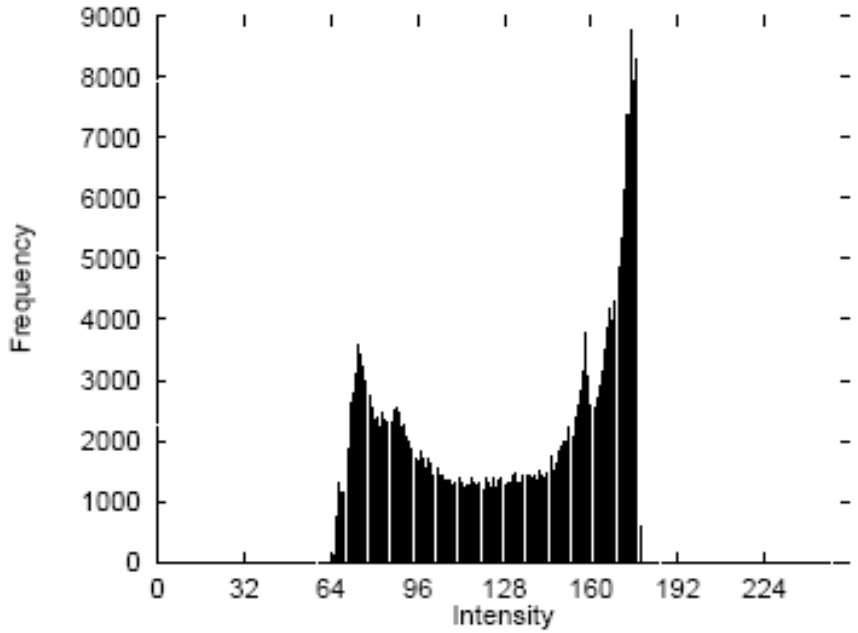


Figure 6.3: A bi-modal histogram.

Otsu(自適應門檻值)決定法(1-3)

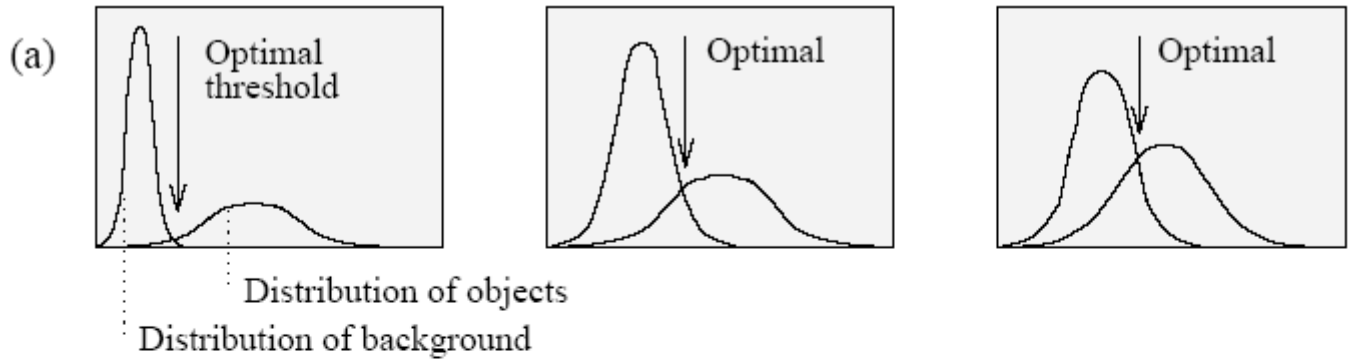


Figure 6.4: Gray-level histograms approximated by two normal distributions—the threshold is set to give minimum probability of segmentation error. (a) Probability distributions of background and objects. (b) Corresponding histograms and optimal threshold.

Otsu(自適應門檻值)決定法(2)

- 令處理影像的大小為N，而灰階值個數為I。則灰階值為i的出現機率可表示為：

$$P(i) = \frac{n_i}{N}$$

- n_i表示灰階值i出現在影像中的次數，且i的範圍介於$0 \leq i \leq I-1$，而根據機率原理又得知

$$\sum_{i=0}^{I-1} P(i) = 1$$

Otsu(自適應門檻值)決定法(3)

- 假設C_0及C_1內的像素個數佔的比率(累進機率)分別為

$$w_0 = \Pr(C_0) = \sum_{i=0}^{T^*} P(i)$$

$$w_1 = \Pr(C_1) = \sum_{i=T^*+1}^{I-1} P(i)$$

$$w_0 + w_1 = 1$$

- 接著可以求出C_0及C_1之期望值為

$$\mu_0 = \sum_{i=0}^{T^*} \frac{P(i)*i}{w_0} = \frac{\mu_*}{w_0}$$

$$\mu_1 = \sum_{i=T^*+1}^{I-1} \frac{P(i)*i}{w_1} = \frac{\mu_T - \mu_*}{1-w_0}$$

Otsu(自適應門檻值)決定法(4)

- 利用μ_0及μ_1，進一步算出C_0及C_1的變異數為

$$\sigma_0^2 = \sum_{i=0}^{T^*} (i-\mu_0)^2 \frac{P(i)}{w_0} \qquad \sigma_1^2 = \sum_{i=T^*+1}^{I-1} (i-\mu_1)^2 \frac{P(i)}{w_1}$$

- 而C_0及C_1之內變異數和為

$$\sigma_W^2 = w_0\sigma_0^2 + w_1\sigma_1^2$$

- C_0及C_1之間的類別間變異數亦可表示為

$$\sigma_B^2 = w_0(\mu_0 - \mu_{T^*})^2 + w_1(\mu_1 - u_{T^*})^2 = w_0 w_1(\mu_0 - \mu_1)^2$$

- 此處μ_{T^*}為整個原始影像的平均值，可用下式求得

$$\mu_T = \sum_{i=0}^{I-1} \frac{n_i * i}{N} = \frac{1}{N}\sum_{i=0}^{I-1} n_i * i$$

Otsu(自適應門檻值)決定法(5)

- 最後，我們可以驗證出σ_B^2、σ_W^2和$\sigma_{T^*}^2$之間存在有這樣的關係，此處$\sigma_{T^*}^2$為原始影像的變異數。
- 由於$\sigma_{T^*}^2$為一定值，C_0和C_1之間的變異數最大化問題等於C_0和C_1內的變異數和的最小化問題。那就考慮如何找到一個最佳化的T^*來使得C_0和C_1之間的變異數σ_B^2為最大就夠了。
- 我們使用的方法是在0至$I-1$之間，一個一個將灰階值代入σ_B^2式子內，等全部I個灰階值都代入完，再從可獲得最大的σ_B^2類別間變異量值所對應的灰階值做為T^*。這樣決定的T^*就是將原始影像分割為C_0和C_1兩區的最佳門檻值。

Otsu(自適應門檻值)OpenCV

一樣是用cv2.threshold()函式，使用方式也一樣，只是最後一個參數增加cv2.THRESH_OTSU，目前otsu只能使用在8位元圖。

- src：輸入影像，只能輸入單通道。
- dst：輸出影像，尺寸大小、深度會和輸入影像相同。
- thresh：門檻值。
- maxval：二值化結果的最大值。
- type：二值化操作型態，共有THRESH_BINARY、THRESH_BINARY_INV、THRESH_TRUNC、THRESH_TOZERO、THRESH_TOZERO_INV五種。
- type從上述五種結合CV_THRESH_OTSU，
- 範例：THRESH_BINARY | CV_THRESH_OTSU

OpenCV
Image Processing Denoise

影像去雜訊(Image Denoising)

- 影像在傳輸的過程中，由於受到通道、劣質取樣系統、雜訊等其他的干擾影響，導致影像變得不清晰，因此我們需要對影像使用濾波**去除雜訊**。
- 雜訊產生的原因決定了雜訊與影像訊號的關係。而減少雜訊的方法可分為兩種：一種是在空間域做處理；另一種則是在頻率域上做處理。
- 在執行影像濾波時，需要以一定的細節模糊做為代價，因此要如何濾除影像的雜訊，又可以保持影像的細節是一個重要的課題。

影像去雜訊-以車道線偵測為例

- 在車道線偵測的過程中，因所使用的車上相機不一定能保有穩定的圖片訊號，容易造成影像中留有許多雜訊，以至於難以取出車道線的特徵。
- 為了讓車道線的特徵能更完整，我們可以先使用濾波去除影像中的雜訊。下圖以高斯濾波器作為示範。

濾波器常見種類

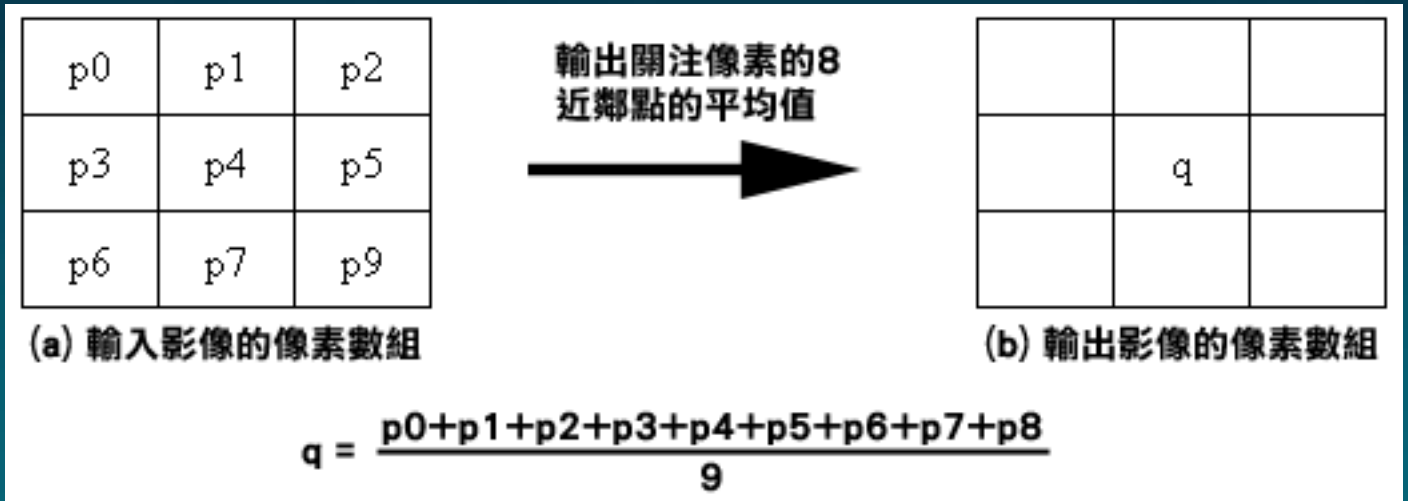
- 均值濾波器(Mean Filter)
- 中值濾波器(Median Filter)
- 高斯濾波器(Gaussian Filter)
- 雙邊濾波器(Bilateral Filter)

均值濾波(Mean Filter)

- 又稱為平滑線性濾波器(averaging filters)或低通濾波器(lowpass filters)。
- 濾波器遮罩，將鄰近的區域中的平均值(灰階)，取代區域中的每一個像素，這樣的程序產生在灰階上「銳利」變化降低的影像。隨機雜訊通常在灰階上含有銳利的變化，所以均值濾波器常常使用在減少雜訊。
- 但是邊緣也在灰階上含有銳利變化的特性，所以均值濾波器有模糊邊緣的缺點。

均值濾波(Mean Filter)

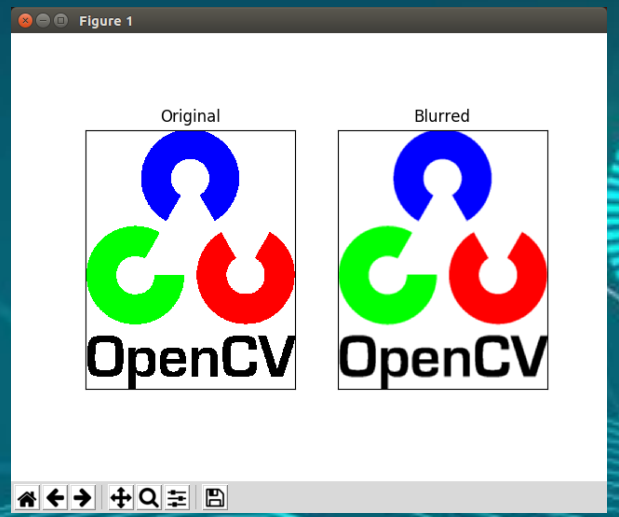
- 平滑法是最簡單的雜訊去除法，它採取把某像素的值置換為該像素周圍3x3個像素的濃度的平均值的方法，將整張影像予以平滑化。



均值濾波-OpenCV

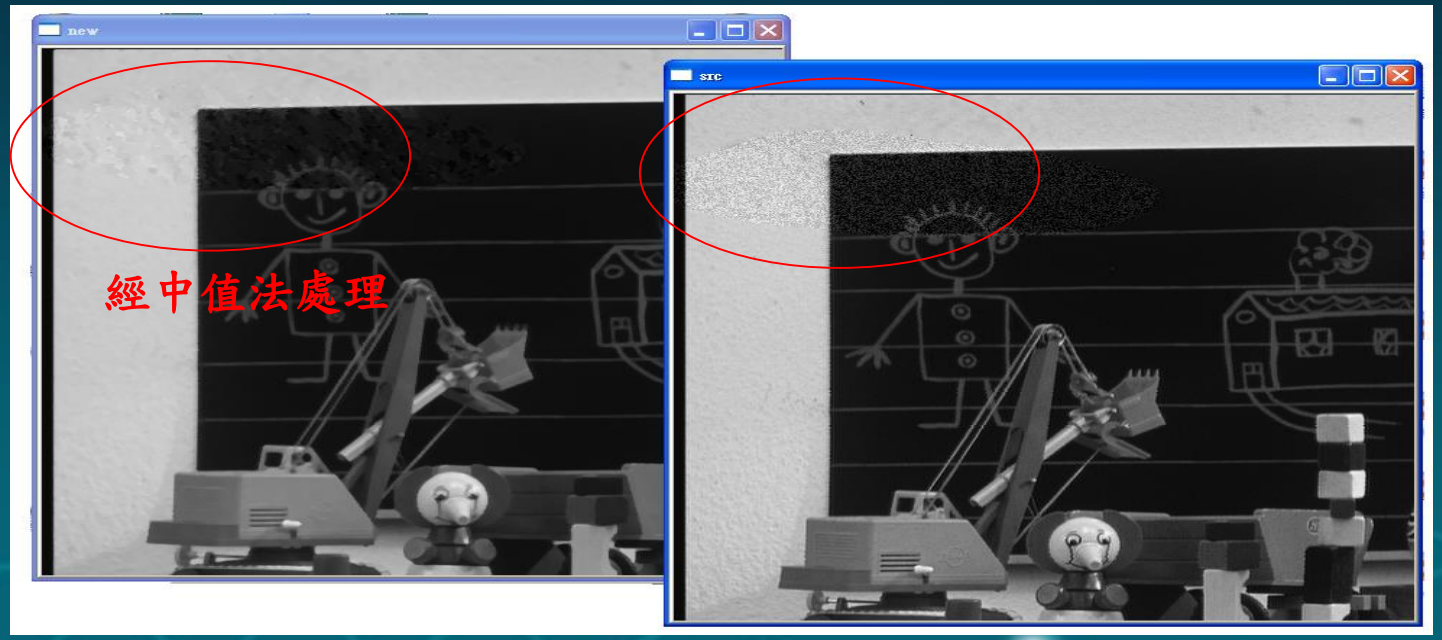
OpenCV blur()函式：

- dst = cv2.blur(<src>, <ksize>[, <dst>[, <anchor>[, <borderType>]]])
  - src：輸入影像。
  - dst：輸出影像會和輸入影像尺寸、型態相同。
  - ksize：kernel大小，可分別指定長和寬。
  - anchor : 錨點，預設為Point(-1,-1) ，代表錨點在kernel的中心
  - borderType ：邊界類型，邊界模式用來推斷影像外的像素
  - 例:blur = cv2.blur(img, (5, 5))



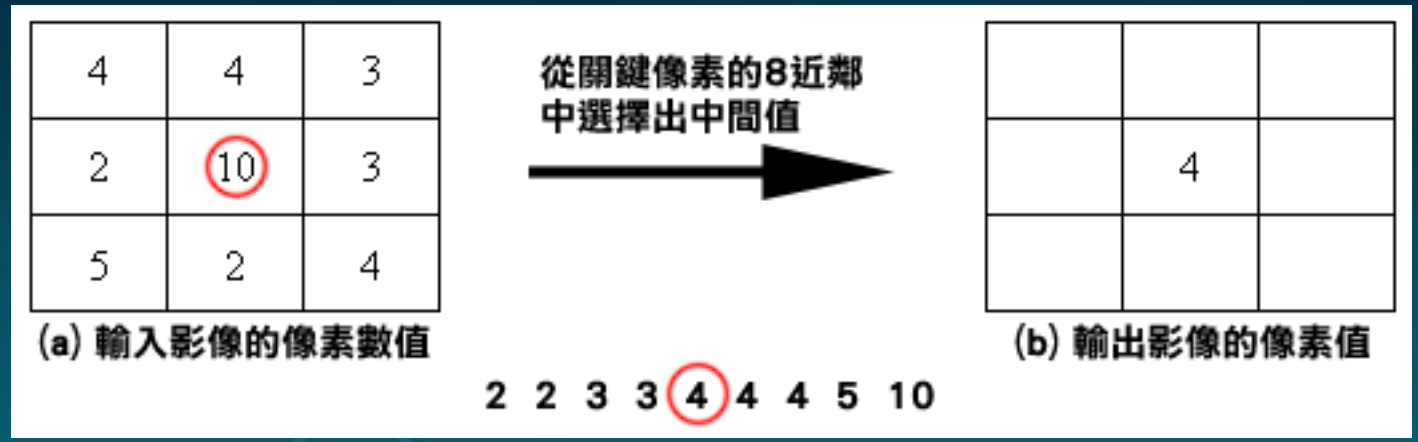
中值濾波(Median Filter)

- 將像素的值用該像素近鄰灰階的中間值來取代。
- 在脈衝雜訊(impulse noise) (又稱胡椒鹽式雜訊)出現時，中值濾波器能有效地去除雜訊，而且與均值濾波器相比有較輕微的模糊化。



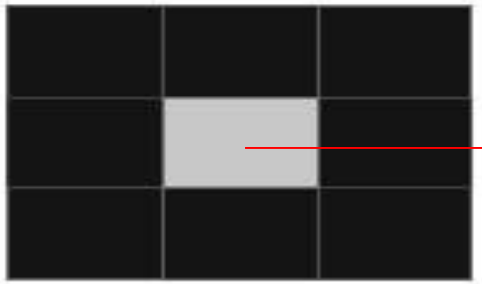
中值濾波(Median Filter)

- 將某像素的值與周圍3x3像素做大小順序排列，隨後取出中間值並取代原有像素。



中值濾波(Median Filter)

20	20	20
20	200	20
20	20	20

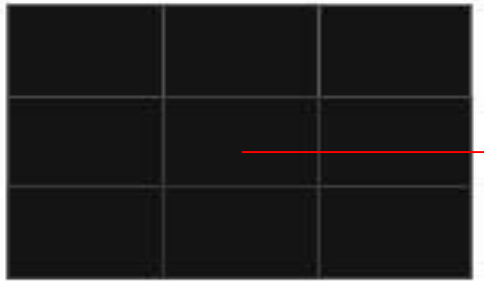


原始雜訊影像

20	20	20
20	40	20
20	20	20

使用平滑法處理之結果

20	20	20
20	20	20
20	20	20



使用中值法處理之結果

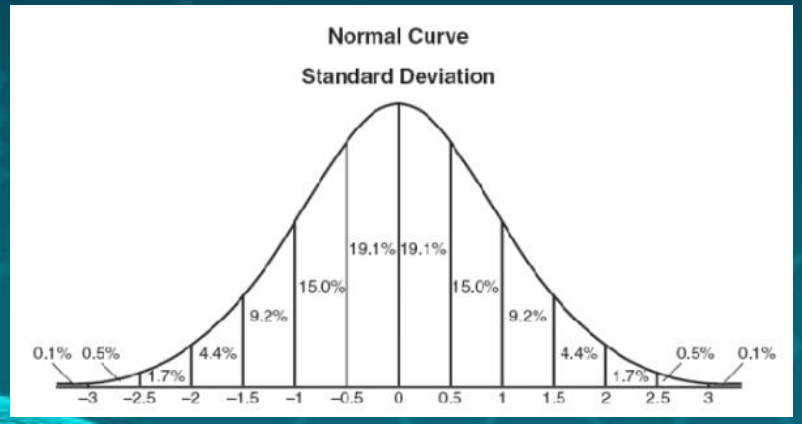
中值濾波 - OpenCV

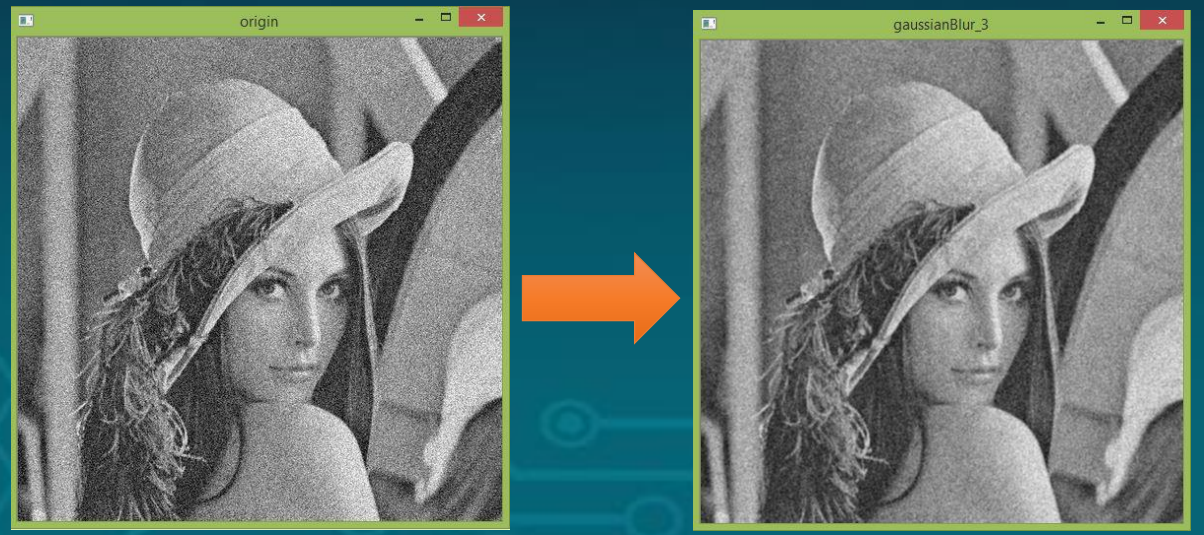
OpenCV medianBlur()函式：

- dst = cv2.medianBlur(<src>, <ksize>[, <dst>])
  - src：輸入影像。
  - dst：輸出影像會和輸入影像尺寸、型態相同。
  - ksize：kernel大小，必須為大於1的正奇數(例如給定7，就會自動使用7x7的kernel，因為kernel必須是正方形)。
  - 例：blur3 = cv2.medianBlur(img, 15)

高斯濾波(Gaussian Filter)

- 高斯濾波改變核心的參數，每個像素的值都是周圍相鄰像素值的加權平均。原始像素為中心，有最大的高斯分布值，所以有最大的權重，相鄰像素隨著距離原始像素越來越遠，其權重也越來越小如下圖。這樣進行模糊處理，跟其它的濾波器相比能更有效地保留邊緣。





高斯濾波(Gaussian Filter)

- 以下為高斯函數：

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$

- 其中μ是x的均值，σ是x的標準差。
- 每次計算時當前像素為原點，因此公式簡化為：

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-x^2/2\sigma^2}$$

高斯濾波(Gaussian Filter)

- 一般影像都是二維，所以使用二維分布，以下為二維高斯函式：

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}$$

- 計算權重值，假設中心為(0,0)：

(-1,1)	(0,1)	(1,1)
(-1,0)	(0,0)	(1,0)
(-1,-1)	(0,-1)	(1,-1)

高斯濾波(Gaussian Filter)

- 計算權重值，假設σ為2，經由高斯運算，其權重矩陣如下：

0.0309874	0.0351134	0.0309874
0.0351134	0.0397887	0.0351134
0.0309874	0.0351134	0.0309874

- 然後求其加權平均，這9個點權重總和等於 0.3041919，因此要分別除0.3041919讓總和等於1：

0.1018679	0.1154317	0.1018679
0.1154317	0.1309013	0.1154317
0.1018679	0.1154317	0.1018679

高斯濾波(Gaussian Filter)

- 有了權重就能計算高斯模糊，假設有以下9個點：

100	95	110
120	90	100
110	115	120

- 將像素點乘上權重：

100x0.1018679	95x0.1154317	110x0.1018679
120x0.1154317	90x0.1309013	100x0.1154317
110x0.1018679	115x0.1154317	120x0.1018679

高斯濾波(Gaussian Filter)

- 得到高斯模糊後的值：

10.18679	10.9660115	11.205469
13.851804	11.781117	11.54317
0.1018679	13.2746455	12.224148

- 以下為3x3常用模板σ為0.8：

1/16	2/16	1/16
2/16	4/16	2/16
1/16	2/16	1/16

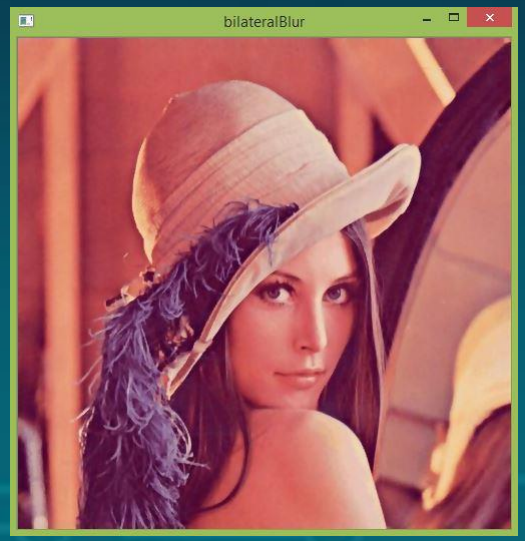
高斯濾波-OpenCV

OpenCV GaussianBlur()函式：

- dst = cv2.GaussianBlur(<src>, <ksize>, <sigmaX>[, <dst>[, <sigmaY>[, <borderType>]]])
  - src：輸入影像。
  - dst：輸出影像會和輸入影像尺寸、型態相同。
  - ksize：kernel大小，長寬可以不同，但是都必須為正的奇數。
  - sigmaX：x方向的標準差。
  - sigmaY：y方向的標準差。
  - 例：blur2 = cv2.GaussianBlur(img, (15,15), 0)

雙邊濾波(Bilateral Filter)

- 和均值濾波及中值濾波有所不同，雙邊濾波器除了使用像素之間幾何上的靠近程度之外，還多考慮像素之間光度及色彩上的差異，使得雙邊濾波器能夠有效的將影像上的雜訊濾除，同時保存影像上的邊緣資訊。



雙邊濾波(Bilateral Filter)

- 雙邊濾波包含了兩個函式，一個是採用空間幾何(和高斯濾波相似)，另一個是像素差值(光度/色彩差異)。
- 以下為雙邊濾波公式：

$$I_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(\|p-q\|) G_{\sigma_r}(|I_p - I_q|) I_q$$

  - p為目標像素。
  - q為目標像素之周圍像素。
  - I_p為目標像素之色彩。
  - I_q為目標像素之周圍像素之色彩。
  - S為目標像素之權重計算範圍。
  - G_{σ_s}為高斯濾波，加權根據距離。
  - G_{σ_r}為高斯濾波，加權根據像素色差。
  - W_p為G_{σ_s}和G_{σ_r}相乘。

雙邊濾波(Bilateral Filter)

- 假設矩陣為8x8，雙邊取值為5x5，距離σ為10，像素σ為30，我們取左上5x5先進行計算。

170	160	170	165	160	145	145	160
160	90	90	85	165	145	145	160
170	95	70	95	170	160	140	140
165	90	85	85	170	160	135	160
170	170	160	160	165	160	160	160
140	160	160	160	140	140	135	135
160	135	145	140	160	160	135	135
160	135	135	140	140	140	140	160

雙邊濾波(Bilateral Filter)

- 計算距離權重值跟高斯相同(雙邊通常模板大小大於為5x5以上，在這裡以5x5大小作為範例)：

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}$$

- 計算權重值，假設中心為(0,0)：

(-2, 2)	(-1, 2)	(0, 2)	(1, 2)	(2, 2)
(-2, 1)	(-1, 1)	(0, 1)	(1, 1)	(2, 1)
(-2, 0)	(-1, 0)	(0, 0)	(1, 0)	(2, 0)
(-2, -1)	(-1, -1)	(0, -1)	(1, -1)	(2, -1)
(-2, -2)	(-1, -2)	(0, -2)	(1, -2)	(2, -2)

雙邊濾波(Bilateral Filter)

- 計算權重值，假設σ為3，經由高斯運算，其權重矩陣如下：

0.011339	0.013395	0.014160	0.013395	0.011339
0.013395	0.015824	0.016728	0.015824	0.013395
0.014160	0.016728	0.017684	0.016728	0.014160
0.013395	0.015824	0.016728	0.015824	0.013395
0.011339	0.013395	0.014160	0.013395	0.011339

雙邊濾波(Bilateral Filter)

- 計算像素差值權重(一樣使用高斯函式，這邊使用一維)：

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-x^2/2\sigma^2}$$

- 計算權重值，假設中心為(3, 3)：

170	160	170	165	160
160	90	90	85	165
170	95	70	95	170
165	90	85	85	170
170	170	160	160	165

雙邊濾波(Bilateral Filter)

- 與(3, 3)之像素差值為：

100	90	100	95	90
90	20	20	15	95
100	25	0	25	100
95	20	15	15	100
100	100	90	90	95

- 計算權重值，假設σ為30，經由高斯運算，其權重矩陣如下：

0.000051	0.000148	0.000051	0.000088	0.000148
0.000148	0.010648	0.010648	0.011736	0.000088
0.000051	0.009397	0.013298	0.009397	0.000051
0.000088	0.010648	0.011736	0.011736	0.000051
0.000051	0.000051	0.000148	0.000148	0.000088

雙邊濾波(Bilateral Filter)

- 將距離權重值乘上像素權重值，得到計算雙邊濾波的權重。

5.78289E-07	1.98246E-06	7.2216E-07	1.17876E-06	1.67817E-06
1.98246E-06	0.000168494	0.00017812	0.00018571	1.17876E-06
7.2216E-07	0.000157193	0.000235162	0.000157193	7.2216E-07
1.17876E-06	0.000168494	0.00019632	0.00018571	6.83145E-07
5.78289E-07	6.83145E-07	2.09568E-06	1.98246E-06	9.97832E-07

- 分別將權重乘上，該位置之像素。

9.83091E-05	0.000317194	0.000122767	0.000194495	0.000268508
0.000317194	0.015164456	0.016030777	0.015785389	0.000194495
0.000122767	0.014933337	0.016461328	0.014933337	0.000122767
0.000194495	0.015164456	0.016687184	0.015785389	0.000116135
9.83091E-05	0.000116135	0.000335309	0.000317194	0.000164642

雙邊濾波(Bilateral Filter)

- 將乘上權重之像素總和除以權重之總和，得到最後計算之結果。乘上權重之像素總和為0.144046367，權重之總和為0.001651341，相除後的結果值為87.22993746。
- 得到新值如右圖(綠色)

170	160	170	165	160	145	145	160
160	90	90	85	165	145	145	160
170	95	87	95	170	160	140	140
165	90	85	85	170	160	135	160
170	170	160	160	165	160	160	160
140	160	160	160	140	140	135	135
160	135	145	140	160	160	135	135
160	135	135	140	140	140	140	160

雙邊濾波(Bilateral Filter)

- 之後再往下一個點繼續計算。

170	160	170	165	160	145	145	160
160	90	90	85	165	145	145	160
170	95	70	95	170	160	140	140
165	90	85	85	170	160	135	160
170	170	160	160	165	160	160	160
140	160	160	160	140	140	135	135
160	135	145	140	160	160	135	135
160	135	135	140	140	140	140	160

雙邊濾波-OpenCV

OpenCV bilateralFilter()函式：

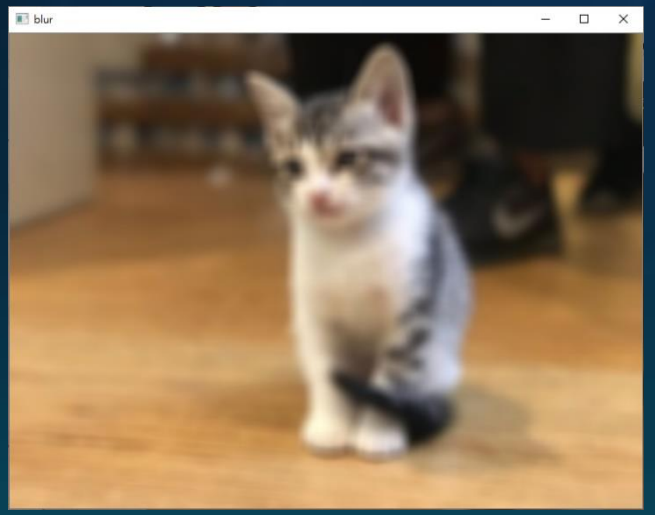
- dst = cv.bilateralFilter(<src>, <d>, <sigmaColor>, <sigmaSpace>[, <dst>[, <borderType>]])
  - src：輸入影像。
  - dst：輸出影像會和輸入影像尺寸、型態相同。
  - d：過程中各像素會使用到的鄰域直徑大小，5以上
  - sigmaColor : 該參數的較大值意味著像素鄰域內的更多顏色將被混合在一起，會得到較大的半等色區域10以上150以下。
  - sigmaSpace : 坐標空間中的過濾器sigma。參數越大意味著只要其顏色足夠近，更遠的像素就會相互影響。
  - borderType ：邊界類型，邊界模式用來推斷影像外的像素
  - 例：blur4 = cv2.bilateralFilter(img, 19, 75, 75)

常見濾波器-OpenCV

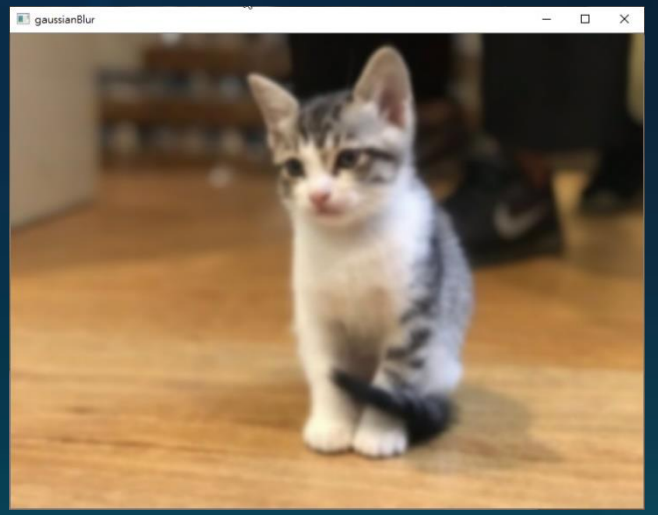
- 範例程式:
  - [4]讀入圖片
  - [6]以15x15大小的kernel(遮罩)進行均值濾波。
  - [7]以15x15大小的kernel(遮罩)進行高斯濾波。
  - [8]以15x15大小的kernel(遮罩)進行中值濾波。
  - [9]進行雙邊濾波。

```
1  import cv2
2  import numpy as np
3
4  img = cv2.imread('cat.jpg')
5
6  blur1 = cv2.blur(img,(15,15))
7  blur2 = cv2.GaussianBlur(img, (15,15), 0)
8  blur3 = cv2.medianBlur(img, 15)
9  blur4 = cv2.bilateralFilter(img, 19, 75, 75)
10
11 cv2.imshow("blur", blur1)
12 cv2.imshow("gaussianBlur", blur2)
13 cv2.imshow("medianBlur", blur3)
14 cv2.imshow("bilateralFilter", blur4)
15 cv2.waitKey(0)
16 cv2.destroyAllWindows()
```

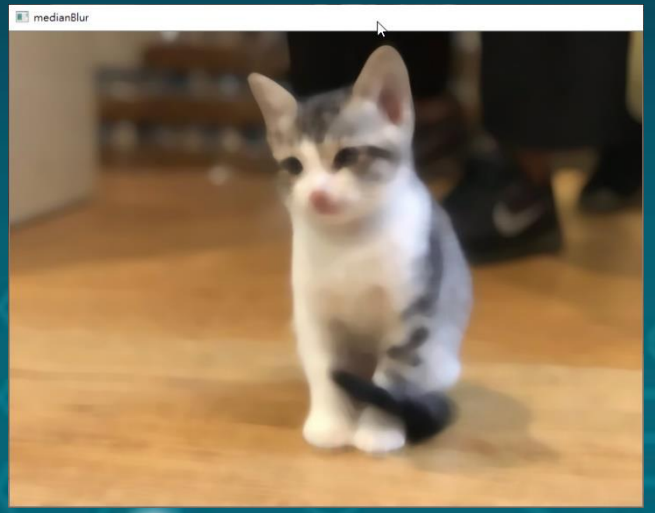
常見濾波器-OpenCV



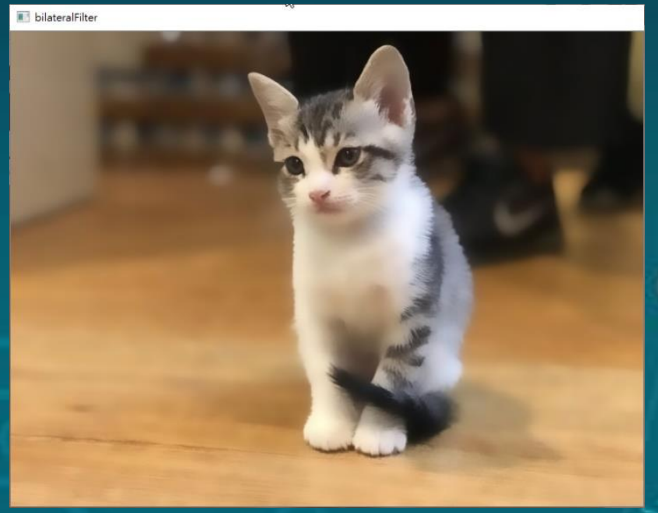
Blur(均值濾波)



GaussianBlur(高斯濾波)



MedianBlur(中值濾波)



BiliteralFilter(雙邊濾波)

應用型態學演算法進行影像雜訊濾除
二值影像的雜訊去除

- 二值影像的雜訊，稱為椒鹽雜訊，它來自英語 salt and pepper noise 一詞的翻譯。當然這種雜訊也能用中值濾波法將其除去，另外，利用它的二值性，有稱為膨脹、收縮的處理方法。
- 膨脹(Dilation)是指某像素*p*的近鄰中，若有一個為 1，則將*p*置為1。
- 收縮(Erosion)是指某像素*p*的近鄰中，若有一個為 0，就將*p*置為0。
- Closing(閉運算)：膨脹→收縮(除去黑色雜訊，白色雜訊依然殘留)
- Opening(開運算)：收縮→膨脹(除去白色雜訊，黑色雜訊依然殘留)

膨脹(Dilation)

- 膨脹法表示位於某個點時是否有偵測到物件(以A當作mask)，以下為其公式，假設A為3x3矩陣B為9x9矩陣。

$$A \oplus B = \{x | B_x \cap A \neq \emptyset\}$$

- 綠色及紅色為原影像

1	1	1
1	1	1
1	1	1

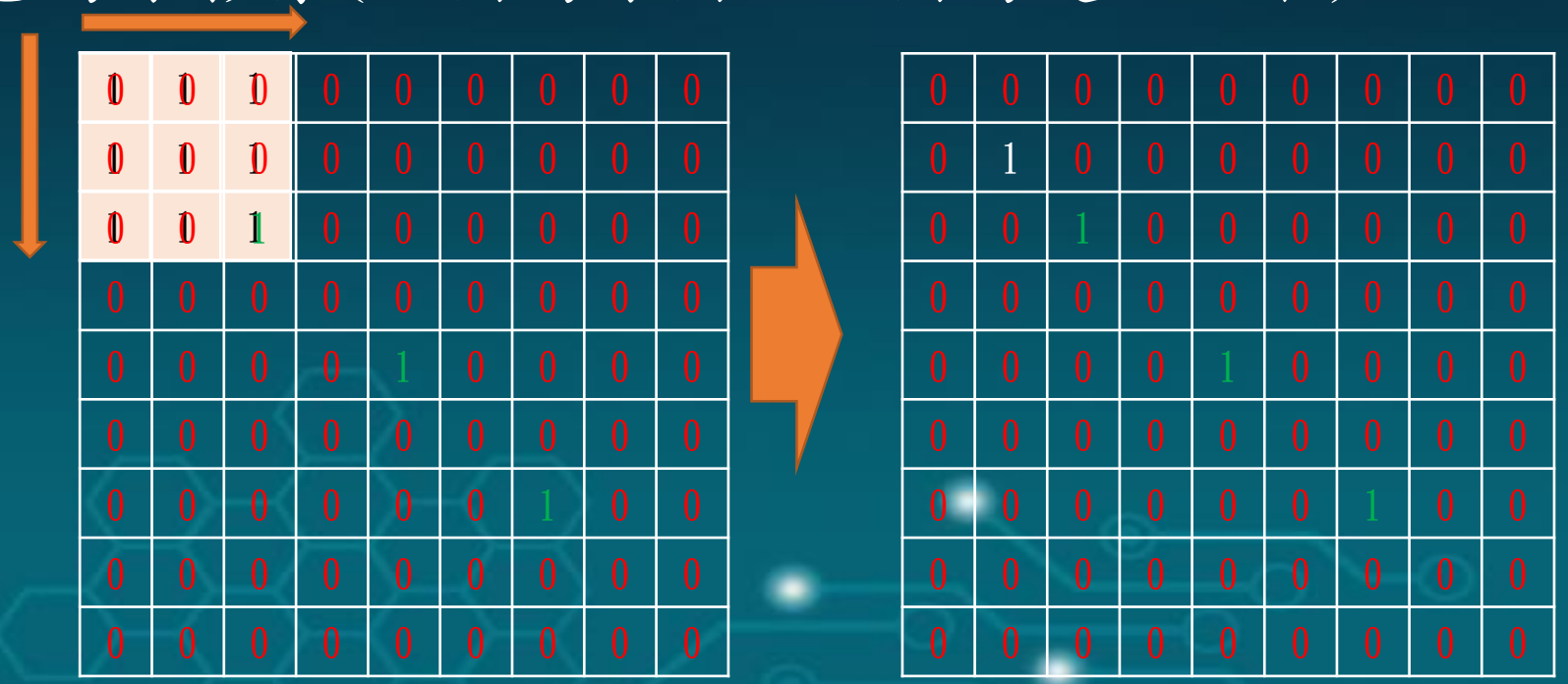
A

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

B

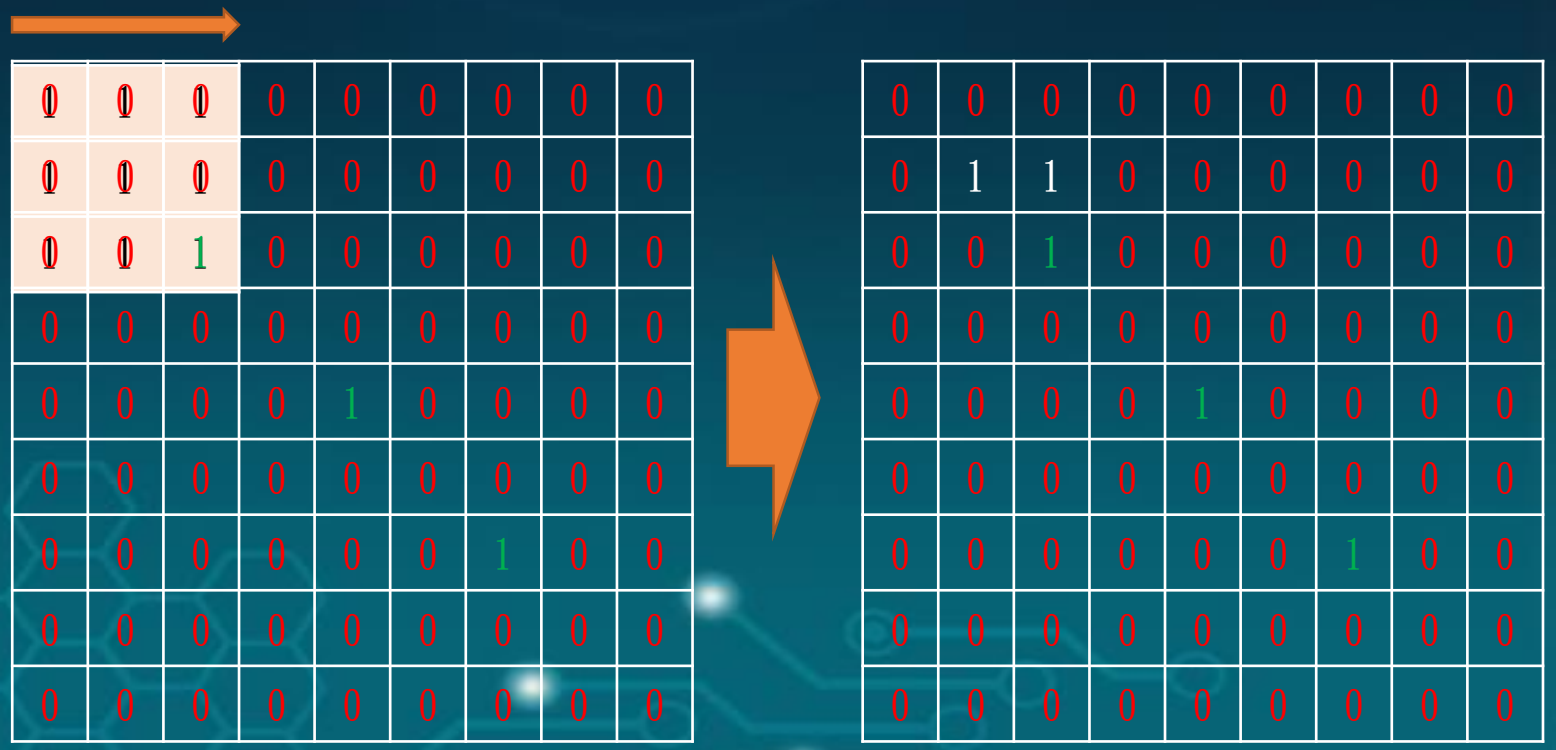
膨脹(Dilation)

- 將A由左而右，由上而下走訪，發現區域中包含了1，因此該區域中心點設為1。
- 綠色及紅色為原影像(左圖為原圖，右圖為走訪結果)



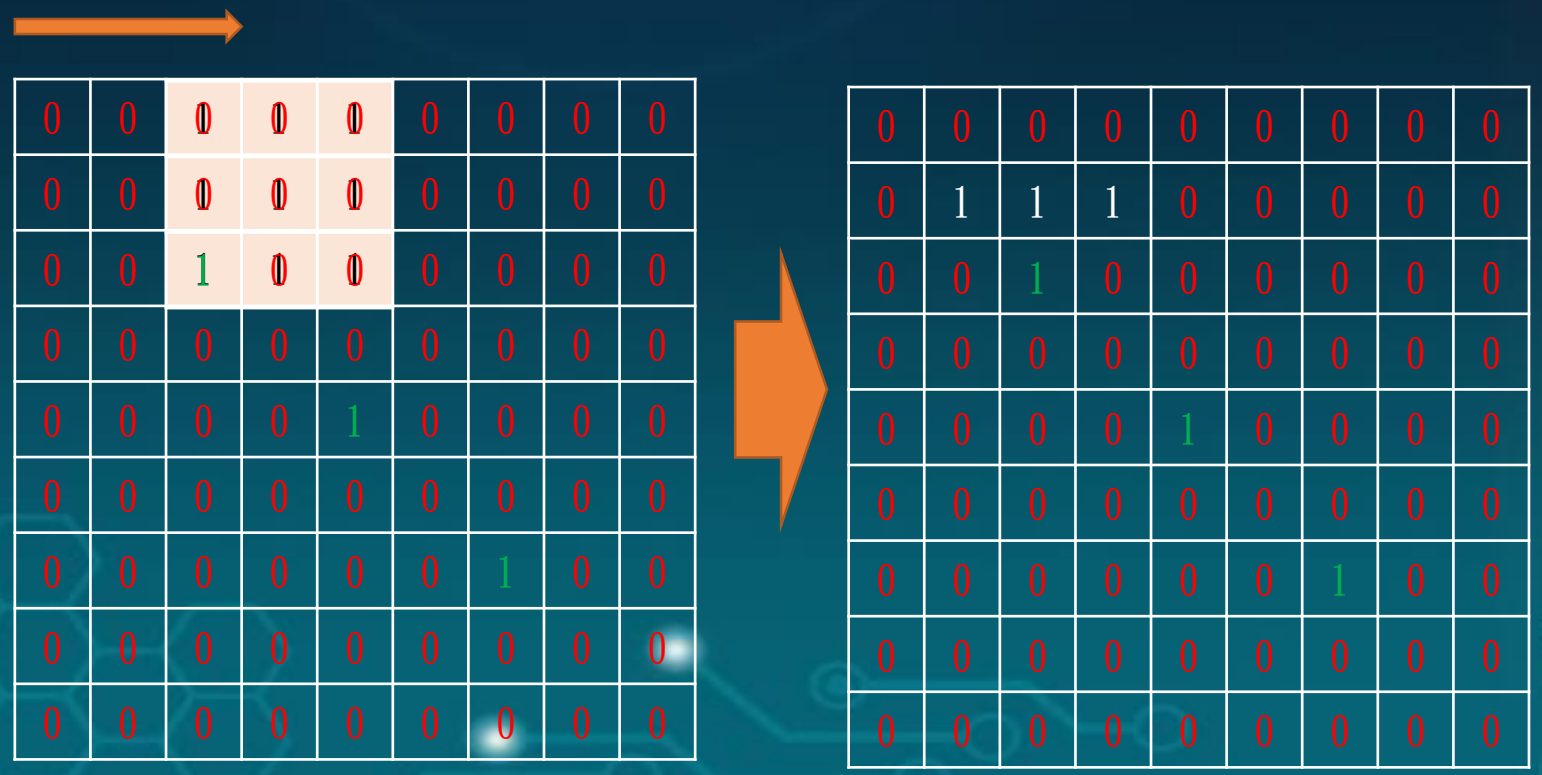
膨脹(Dilation)

- 綠色及紅色為原影像。
- 黑色為走訪後。
- 接著繼續走訪。



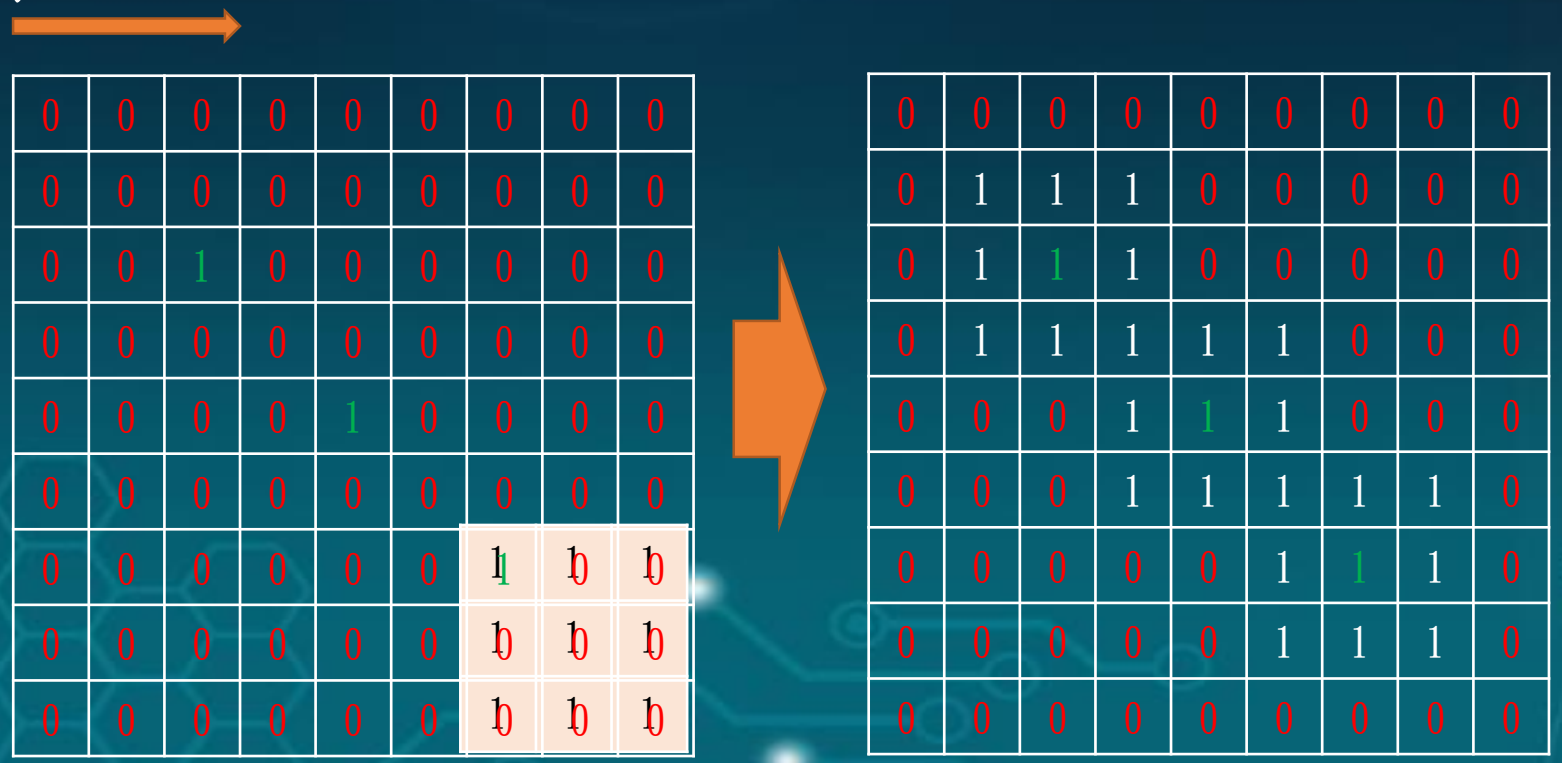
膨脹(Dilation)

- 綠色及紅色為原影像。
- 黑色為走訪後。
- 接著繼續走訪。



膨脹(Dilation)

- 最終走訪結果。
- 綠色及紅色為原影像。
- 黑色為走訪後。



膨脹-OpenCV

OpenCV dilate()函式:

- dst = cv2.dilate(<src>, <kernel>[, <dst>[, <anchor>[, <iterations>[, <borderType>[, <borderValue>]]]]])
  - src：輸入影像。
  - dst：輸出影像，和輸入影像尺寸、型態相同。
  - kernel：結構元素，預設為3x3的矩形，越大膨脹效果越明顯。
  - anchor：原點位置，預設為結構元素的中央。
  - iterations：執行次數，執行越多次膨脹效果越明顯。
  - borderType ：邊界類型，邊界模式用來推斷影像外的像素。
  - 例：out = cv2.dilate(img, np.ones((5, 5)), iterations=3)

侵蝕(Erosion)

- 侵蝕法表示位於某個點時是否有偵測到全部物件(mask A)，以下為其公式，假設A為3x3矩陣B為9x9矩陣。

$$A \ominus B = \{x | B_x \subseteq A\}$$

- 綠色及紅色為原影像

1	1	1
1	1	1
1	1	1

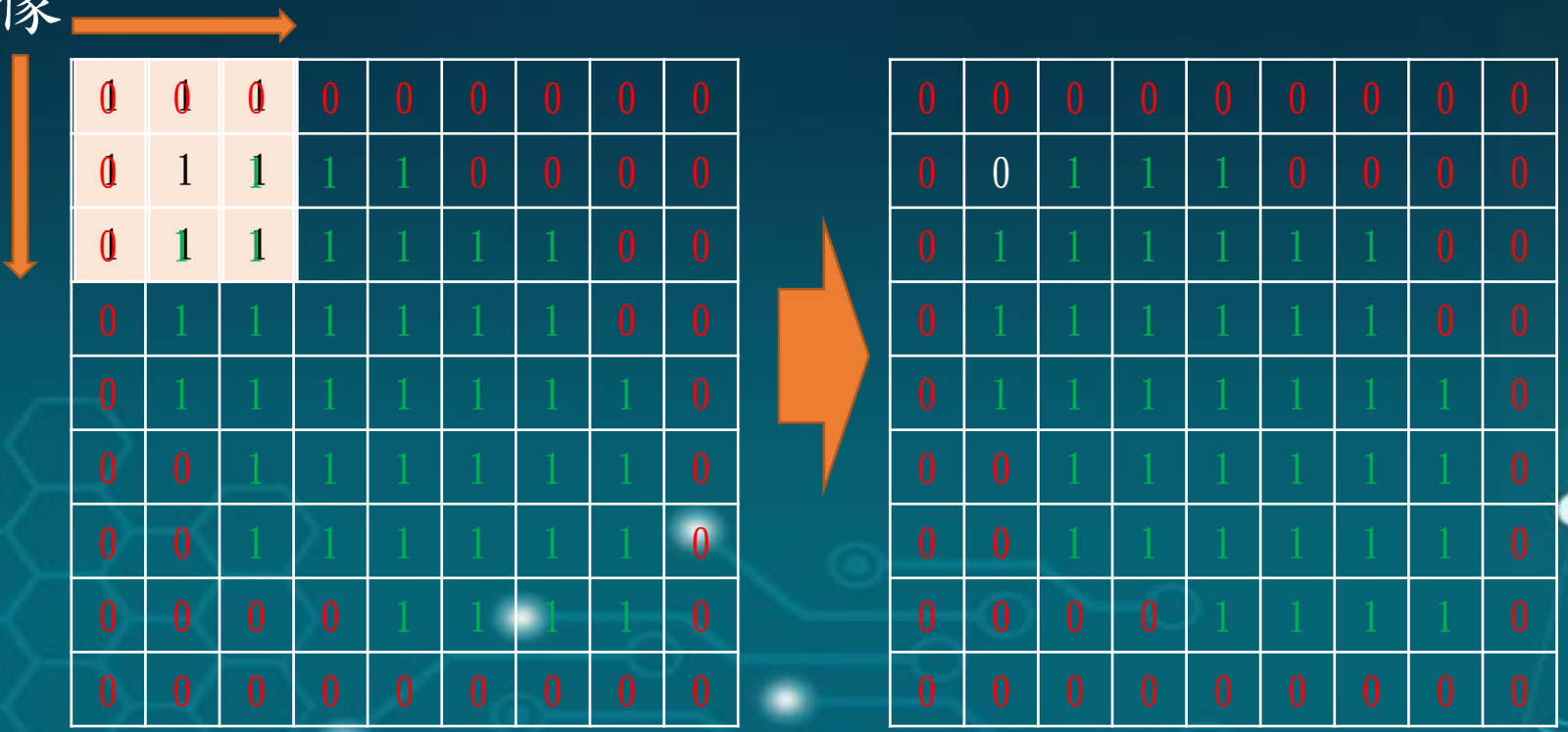
A

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

B

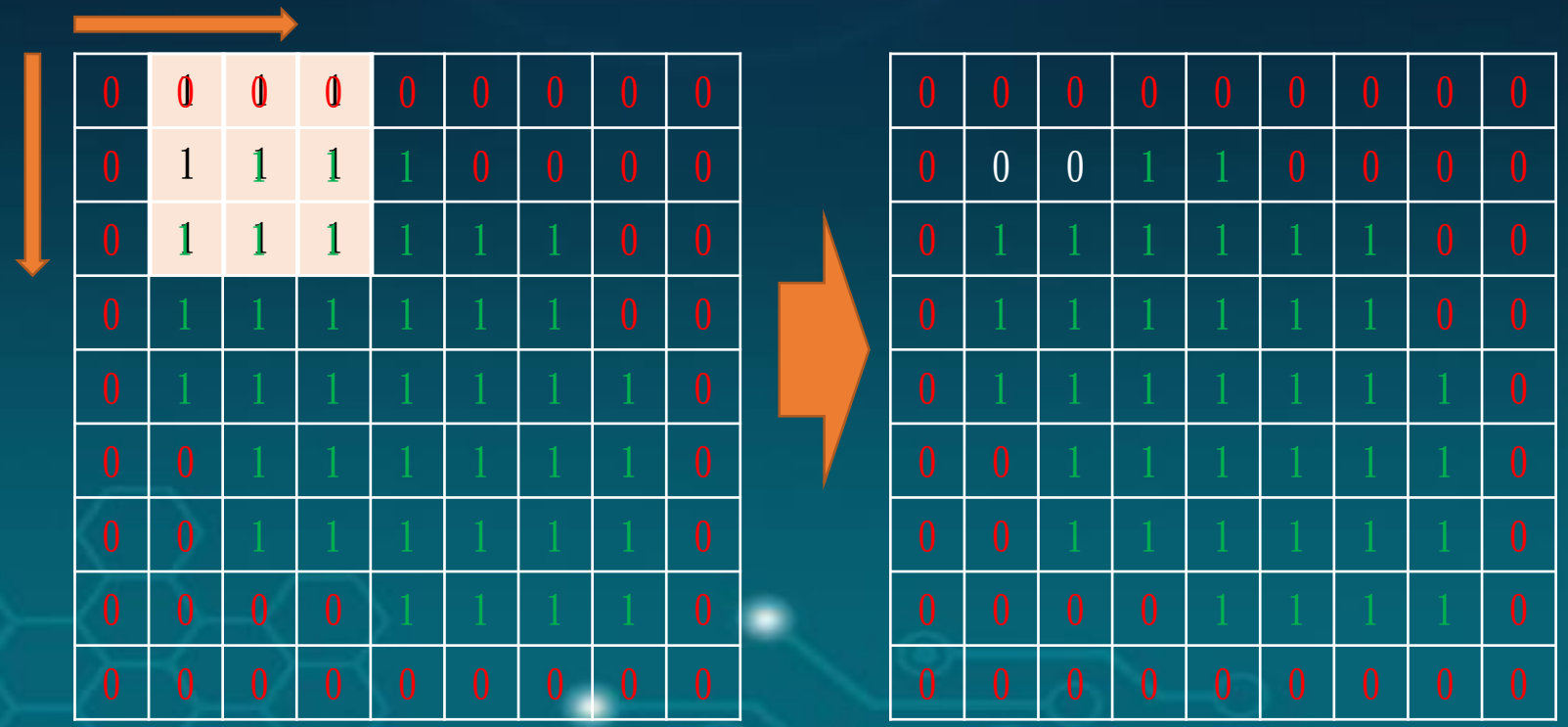
侵蝕(Erosion)

- 將A由左而右，由上而下走訪，發現區域中包含了0，因此該中心點設為0。
- 綠色及紅色為原影像



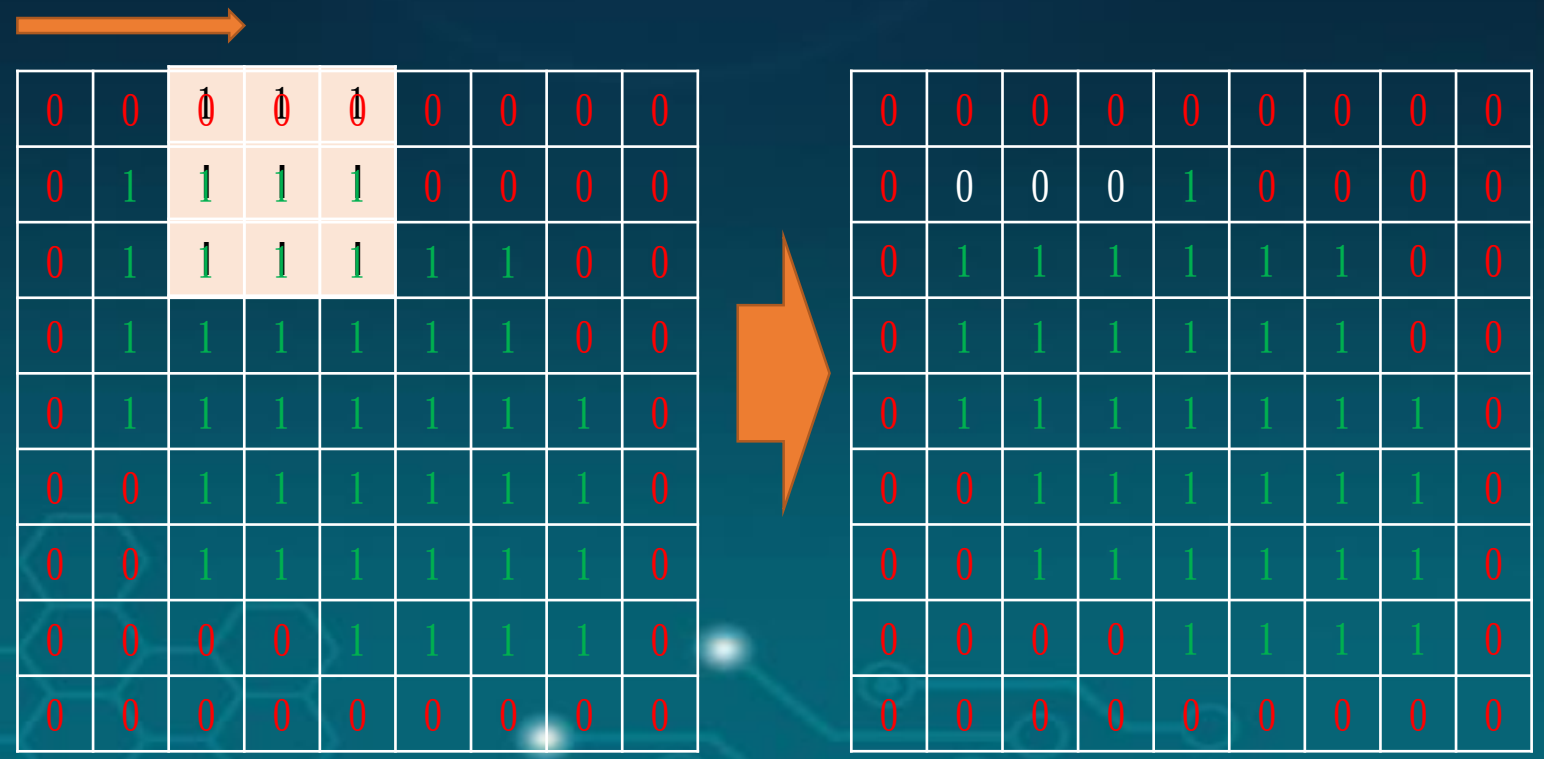
侵蝕(Erosion)

- 綠色及紅色為原影像。
- 黑色為走訪後。
- 接著繼續走訪。



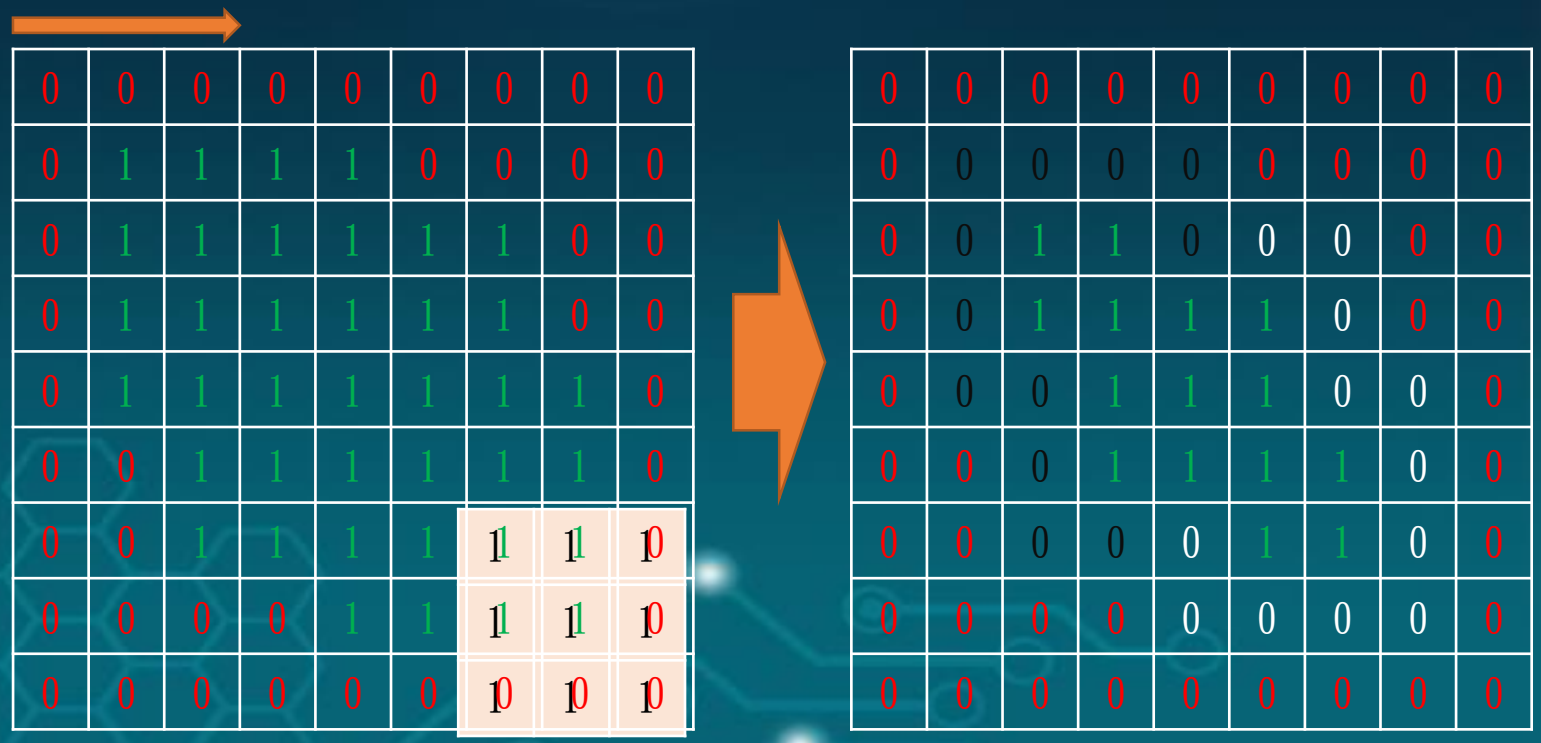
侵蝕(Erosion)

- 綠色及紅色為原影像。
- 黑色為走訪後。
- 接著繼續走訪。



侵蝕(Erosion)

- 完全走訪後之結果。
- 綠色及紅色為原影像。
- 黑色為走訪後。



侵蝕 - OpenCV

OpenCV erode()函式:

- dst = cv2.erode(<src>, <kernel>[, <dst>[, <anchor>[, <iterations>[, <borderType>[, <borderValue>]]]]])
  - src：輸入影像。
  - dst：輸出影像，和輸入影像寸、型態相同。
  - kernel：結構元素，預設為3x3的矩形，越大侵蝕效果越明顯。
  - anchor：原點位置，預設為結構元素的中央。
  - iterations：執行次數，預設為1次，執行越多次侵蝕效果越明顯。
  - borderType ：邊界類型，邊界模式用來推斷影像外的像素
  - 例：out = cv2.erode(img, np.ones((5, 5)), iterations=3)

形態學(Morphology)

- 主要用於二值化後的影像，根據使用者的目的，用來凸顯影像的形狀特徵，像邊界和連通區域等，同時像細化、像素化、修剪毛刺等技術也常用於影像的預處理和後處理，形態學操作的結果除了影像本身，也和結構元素的形狀有關，結構元素和空間域操作的濾波概念類似。
- 在機器學習與影像處理的過程中常被用來前處理。

形態學(Morphology)

有以下幾個種類其公式算法如下:

- OPEN
  - 公式：$A \circ B = (A \ominus B) \oplus B$
  - dst = open(src, element) = dilate(erode(src, element))
  - 去除小雜訊
- CLOSE
  - 公式：$A \bullet B = (A \oplus B) \ominus B$
  - dst = close(src, element) = erode(dilate(src, element))
  - 去除小洞
- GRADIENT
  - 公式：$A \oplus B$ - $A \ominus B$
  - dst = morph(src, element) = dilate(src, element) – erode(src, element)
  - 找輪廓

形態學(Morphology)

- TOPHAT
  - 公式：$T_w(f) = f - f \circ b,$
  - dst = tophat(src, element) = src – open(src, element)
  - 輸入影像及型態學Open之間的差異。
  - TOPHAT被用於各種影像處理，如特徵提取，背景均衡，影像增強等。
- BLACKHAT
  - 公式：$T_b(f) = f \bullet b - f,$
  - dst = blackhat(src, element) = close(src, element) - src
  - 型態學Close及輸入影像之間的差異。

形態學-OpenCV

OpenCV morphologyEx()函式：

- dst = cv2.morphologyEx(src, op, kernel[, dst[, anchor[, iterations[, borderType[, borderValue]]]]])
  - src：輸入影像。
  - op：操作種類，決定要進行何種型態學操作(open、close等等)。
  - kernel：結構元素。
  - dst：輸出影像，和輸入影像尺寸、型態相同。
  - anchor：原點位置，預設為結構元素的中央。
  - iterations：執行次數，預設為1次。
  - borderType ：邊界類型，邊界模式用來推斷影像外的像素
  - 例：見下頁

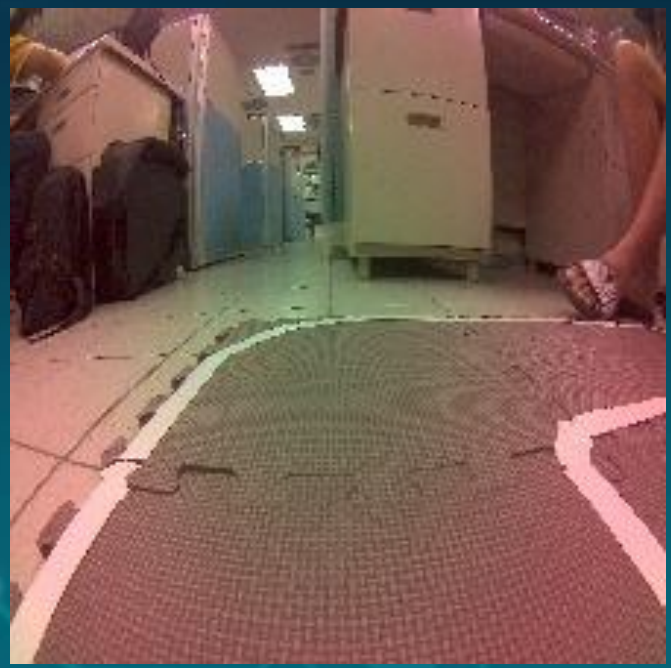
形態學-OpenCV

操作種類如下：

- Opening
  - 例：opening = cv2.morphologyEx(img, cv2.MORPH_OPEN, np.ones((5, 5)))
- Closing
  - 例：closing = cv2.morphologyEx(img, cv2.MORPH_CLOSE, np.ones((5, 5)))
- Gradient
  - 例：gradient = cv2.morphologyEx(img, cv2.MORPH_GRADIENT, np.ones((5, 5)))
- Top Hat
  - 例：tophat=cv2.morphologyEx(img, cv2.MORPH_TOPHAT, np.ones((5, 5)))
- Black Hat
  - 例：blackhat=cv2.morphologyEx(img, cv2.MORPH_BLACKHAT, np.ones((5, 5)))

形態學-以車道線擷取為例

- 在接下來的範例中，我們使用各種形態學的演算法來示範在車道線的擷取上這些演算法能提供什麼樣的幫助。
- 首先將原圖進行轉灰階與二值化的步驟，將圖片較亮的部分過濾出來，再分別使用不同演算法展示結果。



形態學-OpenCV

```python
import cv2
import numpy as np

img = cv2.imread('road.png',0)
kernel = np.ones((5,5), np.uint8)

out3 = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel, iterations=3)
out4 = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel, iterations=3)
out5 = cv2.morphologyEx(img, cv2.MORPH_GRADIENT, kernel, iterations=3)
out6 = cv2.morphologyEx(img, cv2.MORPH_TOPHAT, kernel, iterations=3)
out7 = cv2.morphologyEx(img, cv2.MORPH_BLACKHAT, kernel, iterations=3)

cv2.imshow("open", out3)
cv2.imshow("close", out4)
cv2.imshow("gradient", out5)
cv2.imshow("tophat", out6)
cv2.imshow("blackhat", out7)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

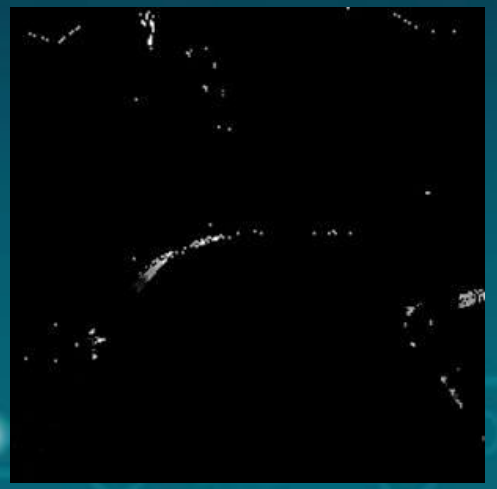
- 範例程式：
  - [5]決定一個5x5的kernel。
  - [7, 13]使用5x5的kernel進行Opening運算，迭代3次。
  - [8, 14]使用5x5的kernel進行Closing運算 ，迭代3次。
  - [9, 15]使用5x5的kernel進行Gradient運算 ，迭代3次。
  - [10, 16]使用5x5的kernel進行Top Hat運算 ，迭代3次。
  - [11, 17]使用5x5的kernel進行Black Hat運算 ，迭代3次。

形態學-OpenCV

Opening運算

Closing運算

Gradient運算



Top Hat運算

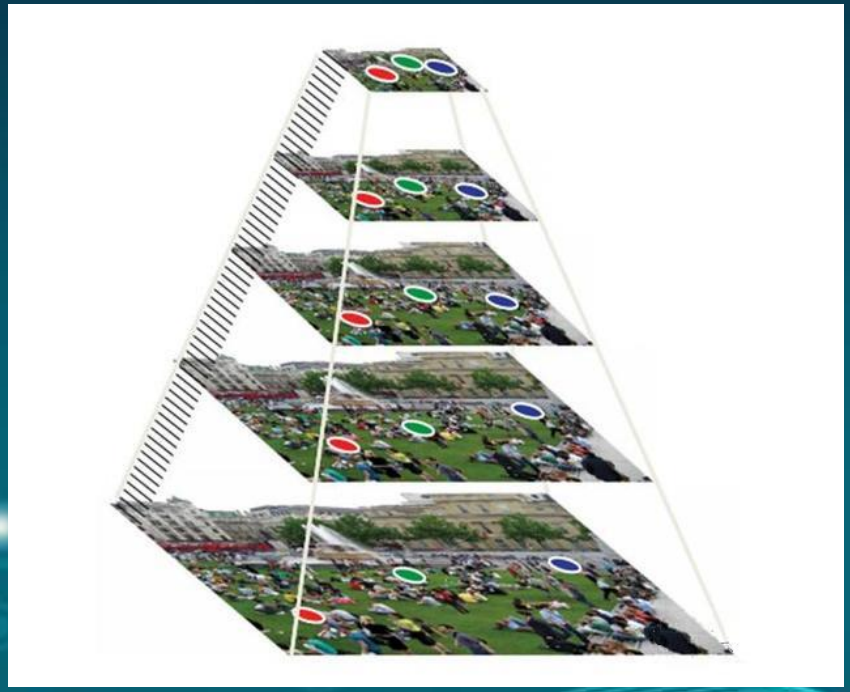
Black Hat運算

影像金字塔(Image Pyramid)

- 影像金字塔是指一組影像且不同解析度的子圖集合，它是影像多尺度表達的一種，以多解析度來解釋影像的結構，主要用於影像的分割或壓縮。

影像金字塔(Image Pyramid)

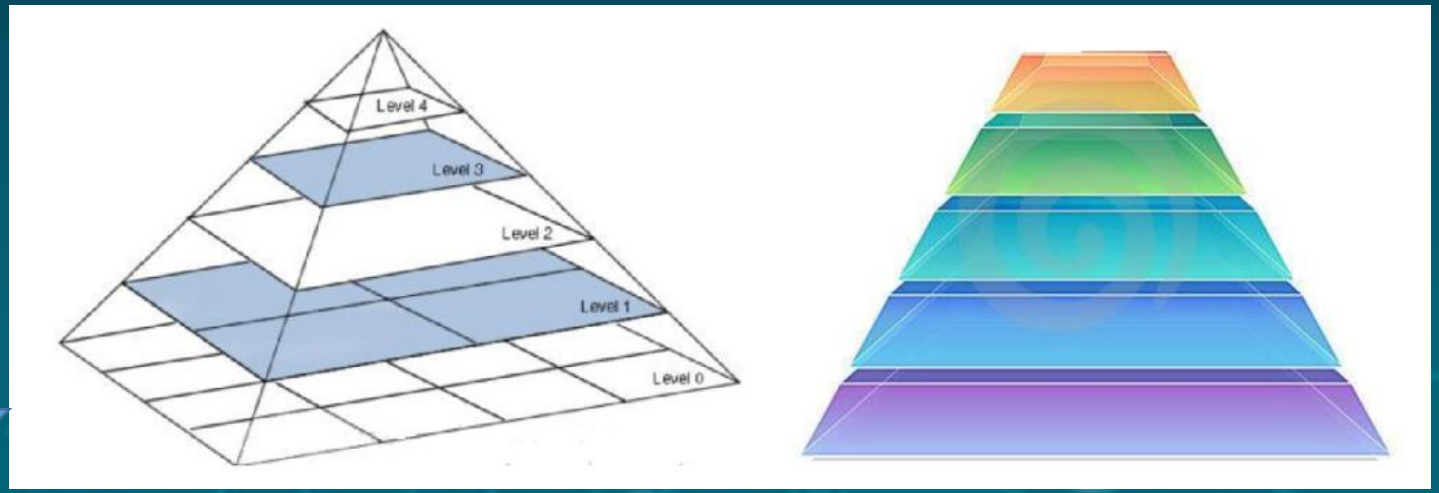
- 一幅影像的金字塔是一系列以金字塔性質排列的解析度逐步降低，且來源於同一張原始圖的影像集合，如下圖所示，它包括了五層影像，將這一層一層的影像比喻成金字塔。影像金字塔可以通過梯次向下取樣獲得，直到達到某個終止條件才停止取樣，在**向下取樣中，層次越高，解析度越低**。



影像金字塔(Image Pyramid)

- 生成影像金字塔主要包括兩種方式——**向下取樣，向上取樣。**
- 將level0級別的影像轉換為 level1，level2，level3，level4，影像**解析度不斷降低**的過程稱為**向下取樣**；
- 將level4級別的影像轉換為 level3，level2，level1，level0，影像**解析度不斷增大**的過程稱為**向上取樣**。

向上取樣



向下取樣

高斯金字塔(Gaussian Pyramid)

- 高斯金字塔用於下取樣。高斯金字塔是最基本的影像塔。原理：首先將原影像作為最底層影像 level0（高斯金字塔的第0層），利用高斯核（5*5）對其進行卷積，然後對卷積後的影像進行下取樣（去除偶數行和列）得到上一層影像G1，將此影像作為輸入，重複卷積和下取樣操作得到更上一層的影像，反覆迭代多次，形成一個金字塔形的影像數據結構，即高斯金字塔。

高斯金字塔(Gaussian Pyramid)

- 高斯金字塔是通過高斯平滑和亞取樣獲取一些列下取樣影像，也就是說第K層高斯金字塔通過平滑，亞取樣就可以獲得K+1 層高斯影像，高斯金字塔包含了一系列低通濾波器，其截止頻率從上一層到下一層是以因子 2 逐漸增加，所以高斯金字塔可以跨越很大的頻率範圍。

拉普拉斯金字塔(Laplacian Pyramid)

- 拉普拉斯金字塔用於重建圖形，也就是預測殘差，對影像進行最大程度的還原。比如一幅小影像重建為一幅大圖。原理：用高斯金字塔的每一層影像減去其上一層影像上取樣並高斯卷積之後的預測影像，得到一系列的差值影像，即為Laplacian分解影像。

拉普拉斯金字塔(Laplacian Pyramid)

- 拉普拉斯影像的形成過程大致為對原影像進行低通濾波和降取樣得到一個粗尺度的近似影像，即分解得到的低通近似影像，把這個近似影像經過插值，濾波，再計算它和原影像的插值，就得到分解的帶通分量。下一級分解是在得到的低通近似影像上進行，迭代完成多尺度分解。可以看出拉普拉斯金字塔的分解過程包括四個步驟：
  1. 低通濾波
  2. 降取樣（縮小尺寸）
  3. 內插（放大尺寸）
  4. 帶通濾波（影像相減）

影像向下取樣(高斯金字塔—縮小)

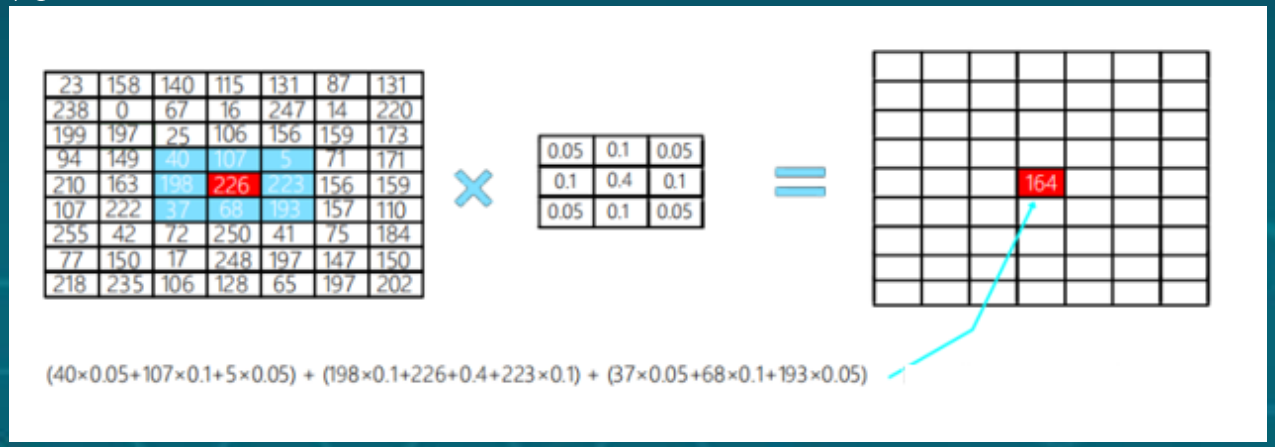
- 在影像向下取樣中，使用最多的是高斯金字塔。**它將堆影像Gi進行高斯核卷積，並刪除原圖中所有的偶數行和列，最終縮小影像。**其中，高斯核卷積運算就是對整幅影像進行加權平均的過程，每一個像素點的值，都由其本身和鄰域內的其他像素值（權重不同）經過加權平均後得到。常見的 3*3 和 5*5 高斯核如下：

$$K(3,3)=\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$$

$$K(5,5)=\frac{1}{273}\begin{bmatrix}1 & 4 & 7 & 4 & 1\\ 4 & 16 & 26 & 16 & 4\\ 7 & 26 & 41 & 26 & 7\\ 4 & 16 & 26 & 16 & 4\\ 1 & 4 & 7 & 4 & 1\end{bmatrix}\qquad \frac{1}{16}\begin{bmatrix}1 & 4 & 6 & 4 & 1\\ 4 & 16 & 24 & 16 & 4\\ 6 & 24 & 36 & 24 & 6\\ 4 & 16 & 24 & 16 & 4\\ 1 & 4 & 6 & 4 & 1\end{bmatrix}$$

影像向下取樣(高斯金字塔—縮小)

- 高斯核卷積讓臨近中心的像素點具有更高的重要度，對周圍像素計算加權平均值，如下圖所示，其中心位置權重最高為 0.4。
- 顯而易見，原始影像G_i具有 M*N 個像素，進行向下取樣之後，所得到的影像 G_{i+1} 具有 M/2 * N/2 個像素，只有原圖的四分之一。通過對輸入的原始影像不停迭代以上步驟就會得到整個金字塔。注意，由於每次向下取樣會刪除偶數行和列，所以它會不停地丟失影像的資訊。



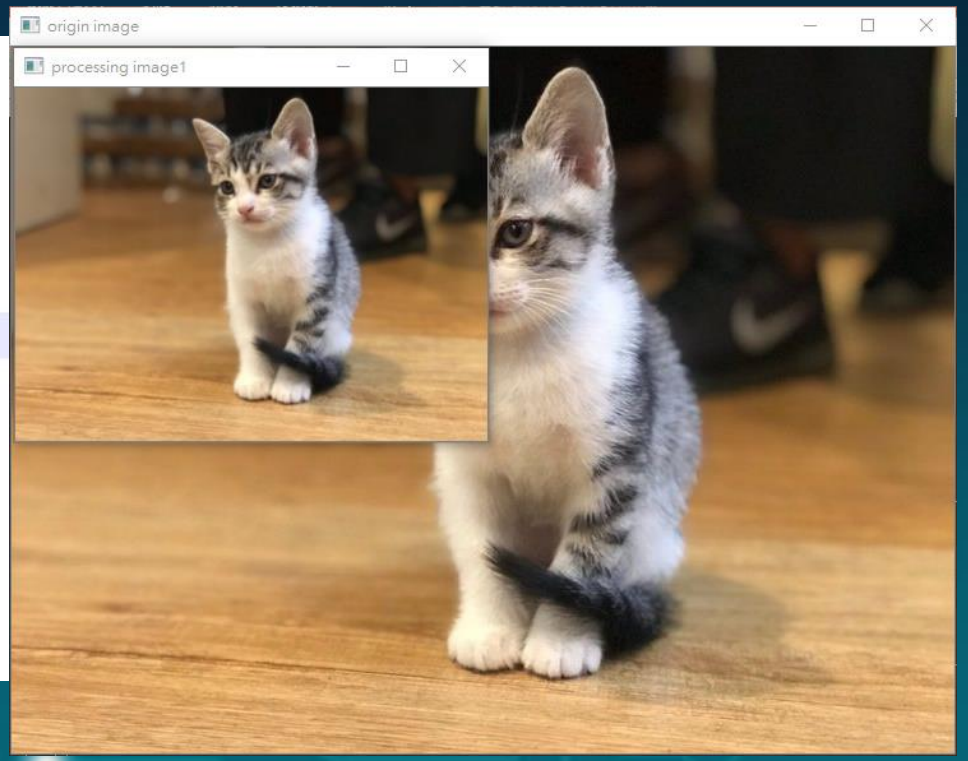
影像向下取樣—OpenCV

- dst = pyrDown(<src>[, <dst>[, <dstsize>[, <borderType>]]])
  - src:表示輸入影像，
  - dst:表示輸出影像，和輸入影像具有一樣的尺寸和類型
  - dstsize:表示輸出影像的大小，默認值為Size(5*5)
  - borderType:表示像素外推方法，詳見cv::bordertypes
  - 例：r1 = cv2.pyrDown(img)

影像向下取樣—OpenCV

- 範例程式:

```python
import cv2
import numpy as np

# 讀取原始圖片
img = cv2.imread('cat.jpg')

# 影像向下取樣
r1 = cv2.pyrDown(img)

# 顯示圖形
cv2.imshow('origin image', img)
cv2.imshow('processing image1', r1)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



影像向上取樣(高斯金字塔—放大)

- 在影像向上取樣是由小影像不斷放大影像的過程，**它將影像在每個方向上擴大為原影像的2倍，新增的行和列均使用0來填充，並使用於「向下取樣」相同的卷積核乘以4，再與放大後的影像進行卷積運算，以獲得「新增像素」的新值**。如下所示，它在原始像素45， 123， 89， 149之間各新增了一行和一列值為0的像素。

$$\begin{vmatrix} 45 & 123 \\ 89 & 149 \end{vmatrix} \rightarrow \begin{vmatrix} 45 & 0 & 123 & 0 \\ 0 & 0 & 0 & 0 \\ 89 & 0 & 149 & 0 \\ 0 & 0 & 0 & 0 \end{vmatrix}$$

```
[[ 74,  93, 120, 130],
 [ 84, 102, 127, 136],
 [ 99, 115, 138, 146],
 [104, 119, 142, 149]]
```

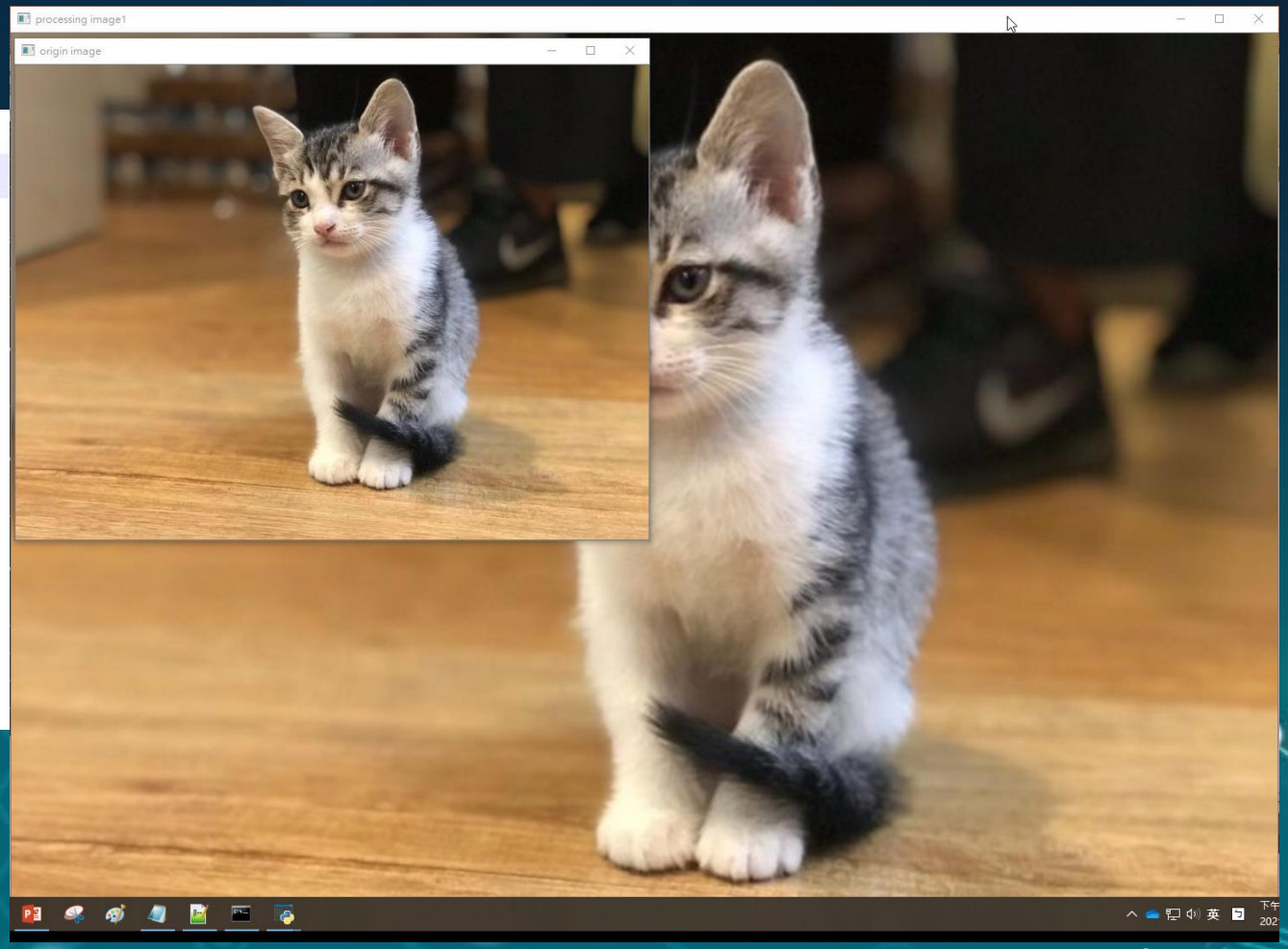
影像向上取樣—OpenCV

- dst = cv2.pyrUp(<src>[, <dst>[, <dstsize>[, <borderType>]]])
  - src:表示輸入影像，
  - dst:表示輸出影像，和輸入影像具有一樣的尺寸和類型
  - dstsize:表示輸出影像的大小，默認值為Size()
  - borderType:表示像素外推方法，詳見cv::bordertypes
  - 例：r1 = cv2.pyrUp(img)

影像向上取樣—OpenCV

- 範例程式:

```python
import cv2
import numpy as np

# 讀取原始圖片
img = cv2.imread('cat.jpg')

# 影像向上取樣
r1 = cv2.pyrUp(img)

# 顯示圖形
cv2.imshow('origin image', img)
cv2.imshow('processing image1', r1)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



影像金字塔總結

- **向上取樣**：就是圖片放大，使用pryUp函數。上取樣的步驟：先將影像在每個方向放大為原來的兩倍，新增的行和列用0填充，再使用先前同樣的內核與放大後的影像卷積，獲得新增像素的近似值。
- **向下取樣**：就是圖片縮小，使用pyrDown函數。下取樣步驟：先將圖片進行高斯內核卷積，再將所有偶數列去除。
- 注意：pryUP() 和 pyrDown() 不是互逆的，即上取樣和下取樣的不是互為逆操作。

影像金字塔總結

- 上，下取樣都存在一個嚴重的問題，那就是影像變模糊了，因為縮放的過程中發生了資訊丟失的問題。要解決這個問題，就得用拉普拉斯金字塔。
- 當然也可以直接使用 cv2裡面的resize()函數，resize()函數的效果更好，下面我們學習一下使用resize()函數進行影像縮放。

影像縮放—cv2.resize()函數

- cv2.resize()函數是opencv中專門來調整圖片的大小，改變圖片尺寸。
- dst = cv2.resize(src, dsize[, dst[, fx[, fy[, interpolation]]]])
  - src：輸入，原影像，即待改變大小的影像；
  - dsize：輸出影像的大小。如果這個參數不為0，那麼就代表將原影像縮放到這個Size(width，height)指定的大小；如果這個參數為0，那麼原影像縮放之後的大小就要通過下面的公式來計算：
  - dsize = Size(round(fx*src.cols), round(fy*src.rows))
  - fx：width方向的縮放比例，如果它是0，那麼它就會按照(double)dsize.width/src.cols來計算；
  - fy：height方向的縮放比例，如果它是0，那麼它就會按照(double)dsize.height/src.rows來計算；
  - interpolation：這個是指定插值的方式，影像縮放之後，肯定像素要進行重新計算的，就靠這個參數來指定重新計算像素的方式，請見下頁：
  - 例:out = cv2.resize(img, (400, 400), interpolation=cv2.INTER_CUBIC)

影像縮放—cv2.resize()函數

- cv2.INTER_NEAREST：最鄰近插值
- cv2.INTER_LINEAR ：雙線性插值，如果最後一個參數你不指定，默認使用這種方法
- cv2.INTER_AREA ：區域插值（使用像素區域關係進行重取樣）
- cv2.INTER_CUBIC ：三次樣條插值（超過4x4像素鄰域內的雙立方插值）
- cv2.INTER_LANCZOS4 ： Lanczos插值（超過8x8像素鄰域內的Lanczos插值）

影像縮放—cv2.resize()函數

- 對於插值方法，正常情況下使用默認的雙線性插值法就夠了。不過這裡還是有建議的：若要縮小影像，一般情形下最好用 cv2.INTER_AREA 來插值，而若要放大影像，一般情況下最好用 cv2.INTER_CUBIC（效率不高，慢，不推薦使用）或 cv2.INTER_LINEAR（效率較高，速度較快，推薦使用）