嵌入式智慧影像分析與實境界面
Fall 2021

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

Lecture 5

嵌入式基礎影像處理技術

基礎影像分析-
應用OpenCV於車道線偵測為例

車道辨識在自動車領域是非常重要的主題，除了傳統的影像處理手段之外，現今的研究也常常使用機器學習的網路模型辨識出車前攝影機影像中的車道線。

基礎影像分析-
應用OpenCV於車道線偵測為例-機器學習流程

雖然機器學習非常方便、速度快又可達到高準確率，但即使是使用機器學習的方法，仍須對影像作前處理。前處理大概包含有：

- 色彩轉換（灰階/HSV）
- 門檻值二值化
- 物件標示

而常用的影像相關機器學習架構有CNN、YOLO等。

基礎影像分析-以應用OpenCV於車道線偵測為例-傳統影像處理流程

在傳統影像處理的手段中，車道辨識大概可以包含以下步驟：

1. 將RGB影像轉為灰階/HSV影像
2. 對影像作增強/降噪、以形態學方法放大或縮小特徵
3. 以門檻值做二值化篩選出車道線
4. 以邊緣檢測方法抓出車道線的形狀

在電腦視覺領域上，最常被使用的影像套件則是OpenCV。

OpenCV

OpenCV簡介

- OpenCV：最早是由Intel公司所開發的開放原始碼電腦影像視覺函式庫(Open Source Computer Vision Library)，它由一系列 C 函數和 C++ 構成，實現了影像處理和電腦視覺方面的很多通用演算法。
- 最新穩定版本為4.4.0。
- 更多關於OpenCV：https://opencv.org/

OpenCV概述

- 在影像處理的過程中，會很頻繁地使用到矩陣數值的運算。為了更快速地實現影像處理的功能，我們使用OpenCV這個程式庫來協助影像處理的運算。
- 在以下的章節中，會介紹的內容包含有：
  - OpenCV簡介
  - OpenCV的常用語法
  - 色彩轉變與二值化
  - 常見濾波
  - 形態學

OpenCV跨平台性

- OpenCV可以在 Windows、MacOS、Linux、Android與IOS等作業系統，只要選擇OpenCV來當作影像開發函式庫，那麼所開發出來的程式將能輕鬆移植至其他作業平台。
- OpenCV支援的使用環境如下：
  - Visual C++ .net
  - Eclipse IDE
  - C++ Builder IDE
  - DevCpp IDE
  - Visual C++ and Microsoft's DirectShow
  - Xcode (Mac)
  - Python
  - Linux

OpenCV程式語言

- OpenCV使用C++語言編寫，它的主要介面也是C++語言，但是依然保留了大量的C語言介面。OpenCV也有大量的Python，Java 和 MATLAB / OCTAVE(版本2.5)的介面。這些語言的API介面函式可以通過線上文件獲得。現在也提供對於C#, Ch, Ruby的支援。
- 所有新的開發和演算法都是用C++介面。
- OpenCV從第三版開始已加入深度學習的模組 (OpenCV 3.3)。

OpenCV 常用功能

OpenCV-常用語法-導言

- 影像處理的基本元素就是圖片，而在OpenCV中，是使用Mat的資料型態儲存圖片資料。
- 每個影像處理的步驟都可以使用Mat儲存起來，顯示出來相互比較。
- 在C++中，可以使用imread將圖片讀取進Mat中，使用imwrite將圖片寫進檔案，使用imshow將圖片顯示在畫面中。 (C++)
- 在Python中，可以使用imread將圖片進行讀取，使用imwrite將圖片寫進檔案，使用imshow將圖片顯示在畫面中。 (Python)

OpenCV-常用語法(Mat)(C++)

- Mat的資料結構中，包含長、寬、像素型態、像素深度、通道數等資訊，以下為Mat之成員變數及函式。

成員變數	定義
rows	影像列的數量，也就是影像高
cols	影像行的數量，也就是影像寬

- OpenCV影像尺寸：Size Mat::size() const
  - 影像之大小 Size(cols, rows)，cols和rows分別為寬和高。
- OpenCV通道數：int Mat::channels() const
  - 影像之通道數：灰階圖為1，彩色圖為3。

OpenCV-常用語法(Mat)(C++)

- OpenCV像素型態：int Mat::type() const

CV_8U	8位元整數，無負號，通道數1
CV_8S	8位元整數，有負號，通道數1
CV_16U	16位元整數，無負號，通道數1
CV_32F	32位元浮點數，通道數1
CV_8UC3	8位元整數，無負號，通道數3

OpenCV-常用語法(Mat)(C++)

OpenCV Mat函式：

- Mat(int rows, int cols, int type, const cv::Scalar &s)
  - rows : 影像之高度。
  - cols : 影像之寬度。
  - type : 影像之型態。
  - s：像素值，為像素強度，灰階或BGR。
- 使用方法
  - Mat img1(480, 720, CV_8);
  - Mat img1(480, 720, CV_8, Scalar(128));
  - Mat img1(480, 720, CV_8UC3, Scalar(256, 128, 0));

OpenCV-常用語法(Mat)(C++)

影像複製

- OpenCV等號多載Mat& Mat::operator = (const Mat& img)。
- OpenCV影像複製：Mat::copyTo(Mat& img) const。
  - img：輸入影像，左邊影像和右邊影像相同。
- OpenCV影像複製：Mat Mat::clone() const。
  - img：輸出影像，輸出影像會有相同的長、寬、像素值。
  - Mat img1(480, 720, CV_8U);
  - Mat img2, img3, img4;
  - img2 = img1; //第一種
  - img1.copyTo(img3); //第二種
  - img4 = img1.clone(); //第三種(最常使用)

OpenCV-常用語法(Mat)(C++)

操作像素

- 我們可用at()得到或改變某個像素值，這函式使用模板，所以使用時除了輸入位置，還必須輸入影像的像素型態，使用at()函式時，輸入參數順序為at(高,寬)。
- OpenCV改變像素：template T& Mat::at(int i, int j)
- at<type> : type 有 uchar, unsigned char, float, double
- Mat Image (720, 480, CV_8U, Scalar(100));
- Image.at<uchar>(400, 100) = 255; //把位置(400,100)的值改成255

OpenCV-常用語法(resize)(C++)

OpenCV改變尺寸大小

- void resize(InputArray src, OutputArray dst, Size dsize, double fx=0, double fy=0, int interpolation=INTER_LINEAR)
  - src : 輸入影像。
  - dst : 輸出影像：型態會和輸入圖相同。
  - dsize：輸出尺寸，當輸入為0時，fx、fy皆不可為0，dsize=Size(round(fxsrc.cols), round(fysrc.rows))
  - fx：水平縮放比例，當輸入為0，fx=(double)dsize.width/src.cols。
  - fy：垂直縮放比例，當輸入為0時，fy=(double)dsize.height/src.rows。
  - interpolation：插值方式。

OpenCV-常用語法(resize)(C++)

- interpolation：插值方式
- 當影像改變大小時，要在原本的像素之間填入新的像素。
  - CV_INTER_NEAREST ：最鄰近插點法。
  - CV_INTER_LINEAR ：雙線性插值(預設)。
  - CV_INTER_AREA ：臨域像素再取樣插值。
  - CV_INTER_CUBIC ：雙立方插值，4x4大小的補點。
  - CV_INTER_LANCZOS4 ：Lanczos插值，8x8大小的補點。

影像縮放—常用語法cv2.resize()函數(Python)

- 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：
  - Ex:out = cv2.resize(img, (400, 400), interpolation=cv2.INTER_CUBIC)

影像縮放—常用語法cv2.resize()函數(Python)

- 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）

OpenCV-常用語法(waitKey)(C++)

- waitKey(ms)。
- waitKey無限制時間等待一個鍵盤輸入（當參數<= 0）或延遲毫秒等候時間（參數 > 0)。
- 由於作業系統在切換執行緒之間具有最小時間，因此該功能將不會精準的等待正確的設定的延遲(ms)，只保證等待時間至少設定的延遲（ms），取決於當時在您的平台上執行的內容。
- 如果在指定的延遲時間過去之前沒有按下任何鍵，它將返回-1，若有按下按鍵，他將返回按下的按鍵ASCII。

OpenCV-常用語法(waitKey)(Python)

- cv2.waitKey(ms)。
- waitKey無限制時間等待一個鍵盤輸入（當參數<= 0）或延遲毫秒等候時間（參數 > 0)。
- 由於作業系統在切換執行緒之間具有最小時間，因此該功能將不會精準的等待正確的設定的延遲(ms)，只保證等待時間至少設定的延遲（ms），取決於當時在您的平台上執行的內容。
- 如果在指定的延遲時間過去之前沒有按下任何鍵，它將返回-1，若有按下按鍵，他將返回按下的按鍵ASCII。

OpenCV-常用語法(imread, imwrite)(C++)

- 讀取圖片
  - imread(<路徑>, <flag>);
  - flages:
    - CV_LOAD_IMAGE_ANYDEPTH - If set, return 16-bit/32-bit image when the input has the corresponding depth, otherwise convert it to 8-bit.
    - CV_LOAD_IMAGE_COLOR - If set, always convert image to the color one(常用)
    - CV_LOAD_IMAGE_GRAYSCALE - If set, always convert image to the grayscale one
    - Ex: frame = imread("./picInput.jpg", CV_LOAD_IMAGE_COLOR)
- 儲存影像成圖片
  - imwrite(<路徑>, <影像(Mat)>);
    - Ex: imwrite("./pic1.jpg",frame);

OpenCV-常用語法(imread, imwrite)(Python)

- 讀取圖片
  - img = cv2.imread(<路徑>, <flag>);
  - flags:
    - cv2.IMREAD_UNCHANGED - If set, return 16-bit/32-bit image when the input has the corresponding depth, otherwise convert it to 8-bit.
    - cv2.IMREAD_COLOR - If set, always convert image to the color one(常用)
    - cv2.IMREAD_GRAYSCALE - If set, always convert image to the grayscale one
    - Ex: frame = cv2.imread("./picInput.jpg", cv2.IMREAD_COLOR)
- 儲存影像成圖片
  - cv2.imwrite(<路徑>, <影像>);
    - Ex: cv2.imwrite("./pic1.jpg" , frame);

OpenCV-全觀

- 目前OpenCV包含如下幾個部份:
  - core : The Core Functionality.
  - Imgproc : Image Processing Library. It include :
    - Image Filtering
    - Geometric Image Transformations
    - Miscellaneous Image Transformations
    - Histograms
    - Structural Analysis and Shape Descriptors
    - Motion Analysis and Object Tracking
    - Feature Detection
    - Object Detection
  - highgui：High-level GUI and Media I/O Library.

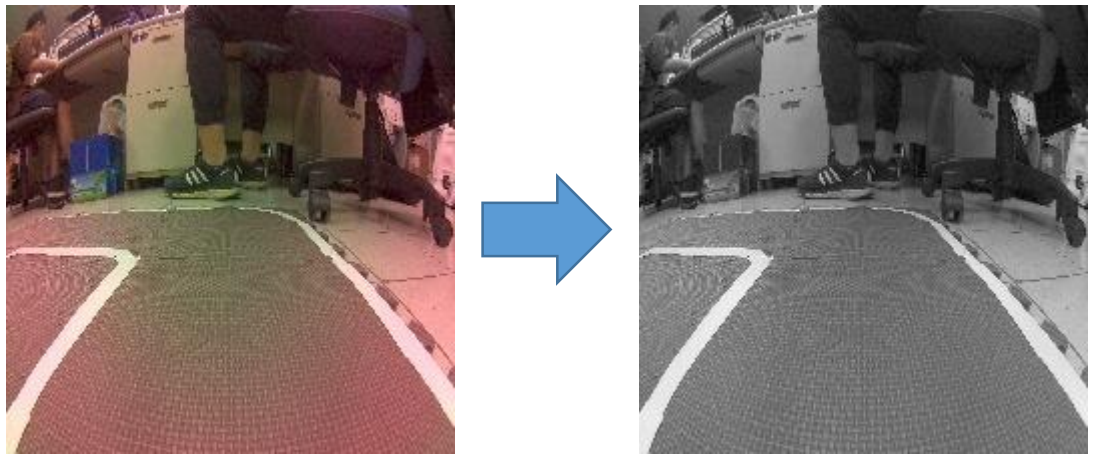
OpenCV Image Processing

灰階(BGR TO GRAY)-簡介

- 將彩色的影像轉化為灰階。
- 彩色轉灰階原理：
  - 由於人眼對綠色的敏感度最大，對藍色敏感度最小，因此綠色的權重分配會較大，藍色較小，如下公式為彩色轉灰階的標準。
- BGR to Gray公式： Y=0.299R + 0.587G + 0.114B

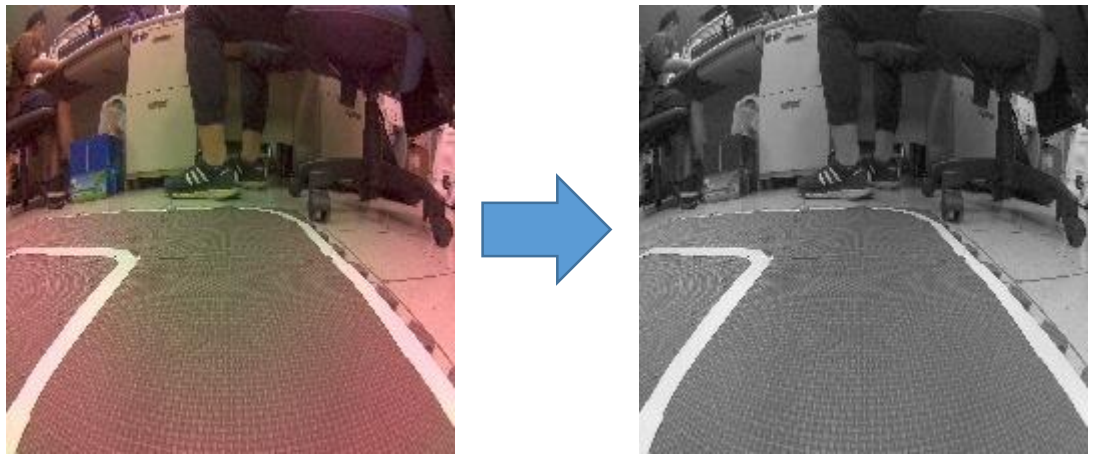
灰階(BGR TO GRAY)-OpenCV(C++)

- Mat為BGR而不是RGB格式，所以輸入參數通常使用CV_BGR2GRAY
- cvtColor(const Mat& src, Mat& dst, int code)
  - src：輸入影像。
  - dst：輸出影像，尺寸大小和深度會與輸入影像相同。
  - code：指定在何種色彩空間轉換，比如CV_BGR2GRAY、CV_GRAY2BGR、CV_BGR2HSV等。



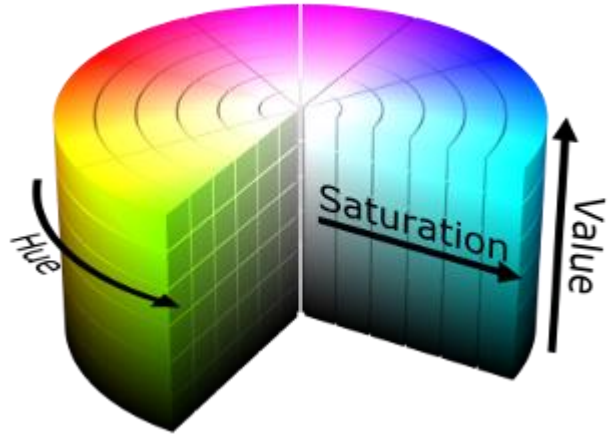
灰階(BGR TO GRAY)-OpenCV(Python)

- cv2.cvtColor(src, code[, dst[, dstCn]])
  - src：輸入影像。
  - code：色彩空間常數變數，比如cv2.COLOR_BGR2GRAY、 cv2.COLOR _GRAY2BGR、 cv2.COLOR _BGR2HSV等。
  - dst：輸出影像，尺寸大小和深度會與輸入影像相同。
  - dstCn：目標影像的頻道數。如果參數為0，則通道數和src相同。



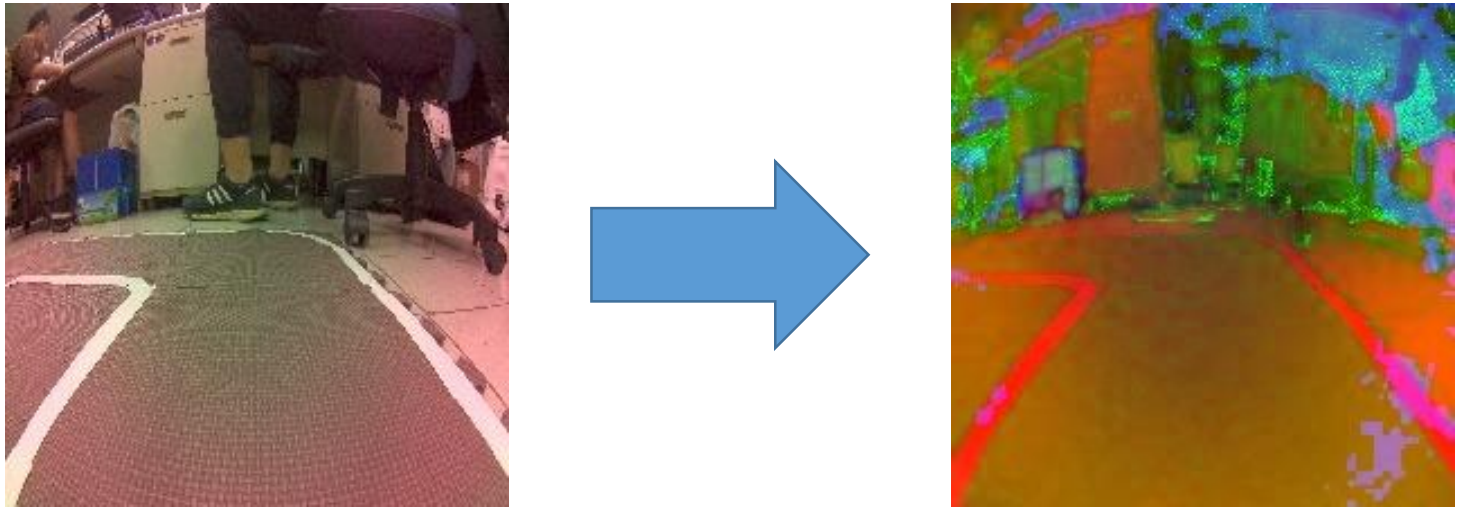
HSV(BGR TO HSV)-簡介

- 將BGR的影像轉化為HSV。
- HSV是一種將RGB色彩轉為圓柱坐標系的表示法：
  - H：色相(Hue)，是色彩的基本屬性，如紅色、黃色等。
  - S：飽和度(Saturation)，是指色彩的純度，越高色彩越純，低則逐漸變灰，取0-100%的數值。
  - V：明度(Value)，取0–100%的數值。
- 在車道辨識上，我們可以先將圖片轉為HSV，再進行後續的車道線與號誌辨識。



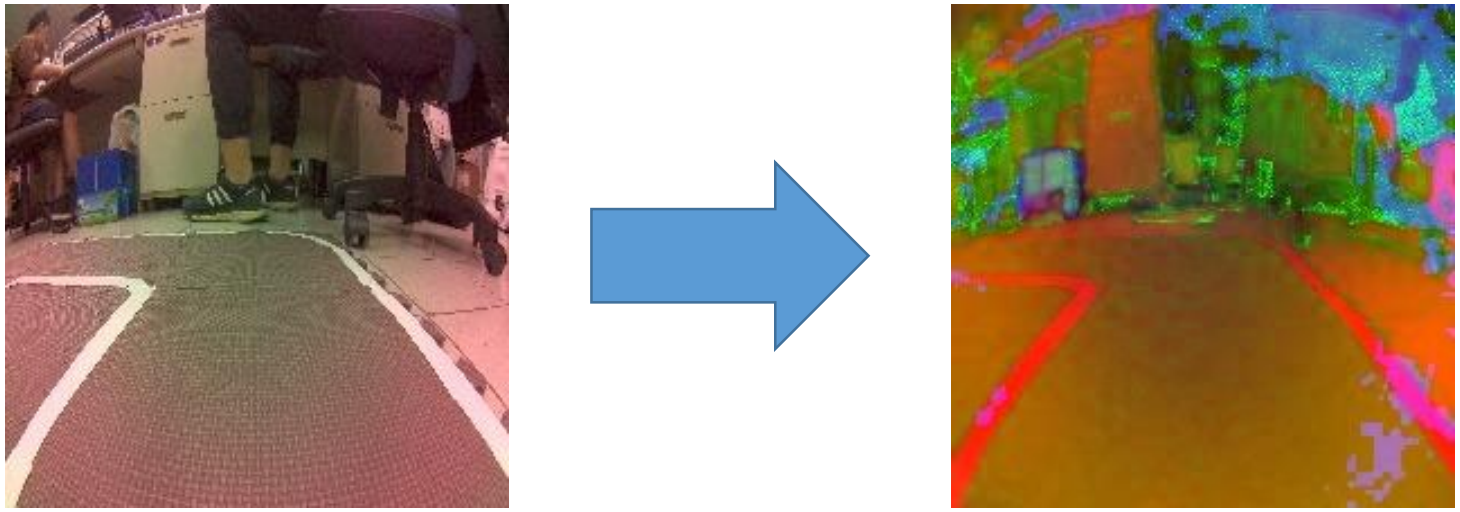
HSV(BGR TO HSV)-OpenCV(C++)

- cvtColor(const Mat& src, Mat& dst, int code)
  - src：輸入影像。
  - dst：輸出影像，尺寸大小和深度會與輸入影像相同。
  - code：指定在何種色彩空間轉換，比如CV_BGR2GRAY、CV_GRAY2BGR、CV_BGR2HSV等。



HSV(BGR TO HSV)-OpenCV(Python)

- cv2.cvtColor(src, code[, dst[, dstCn]])
  - src：輸入影像。
  - code：色彩空間常數變數，比如cv2.COLOR_BGR2GRAY、 cv2.COLOR _GRAY2BGR、 cv2.COLOR _BGR2HSV等。
  - dst：輸出影像，尺寸大小和深度會與輸入影像相同。
  - dstCn：目標影像的頻道數。如果參數為0，則通道數和src相同。



二值化-簡介

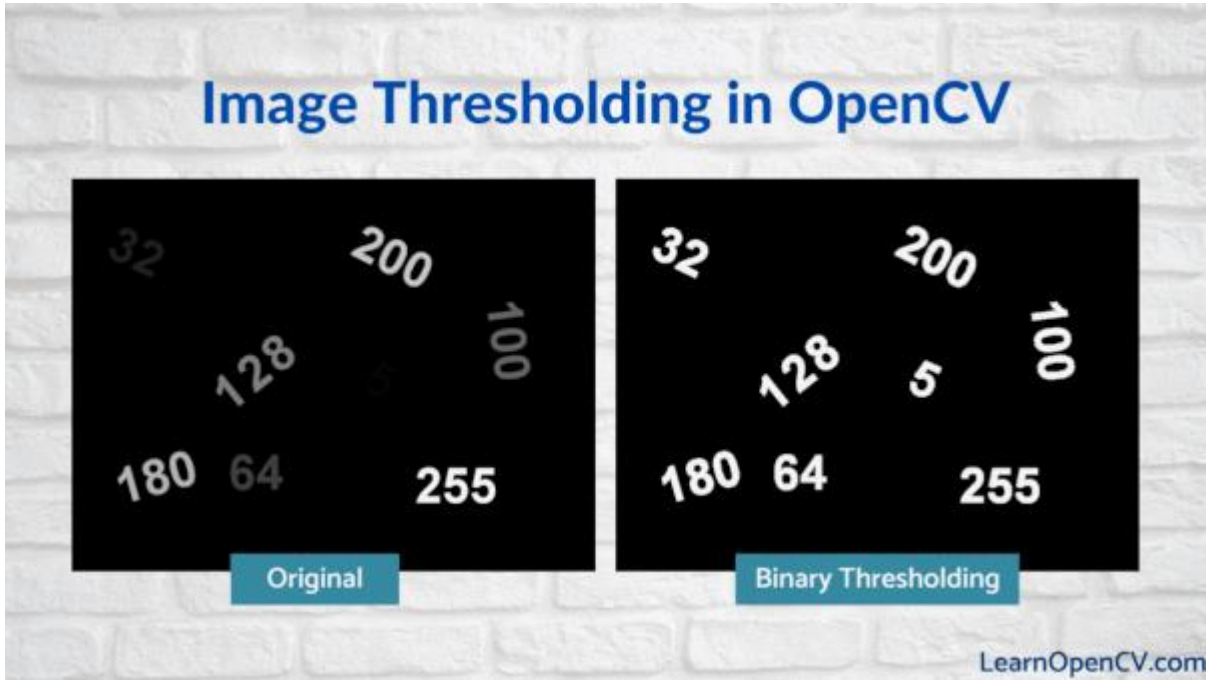
- 二值化是影像分割的一種方法，我們會將影像分為兩個部分，一個是感興趣的部分(前景)，以及不感興趣的部分(背景)，會依據門檻值(threshold)當作分割的標準，通常會以強度超過門檻值的像素當作前景，反之則為背景。
- 門檻值的算法主要分兩類：
  - 固定門檻值：直接給定一個灰階值當門檻值，再用這個門檻值進行二值化。
  - 自適應門檻值：會依據輸入影像計算出較合適的門檻值，再用這個門檻值進行二值化(在Otsu介紹)。

二值化-利用門檻值進行影像分割

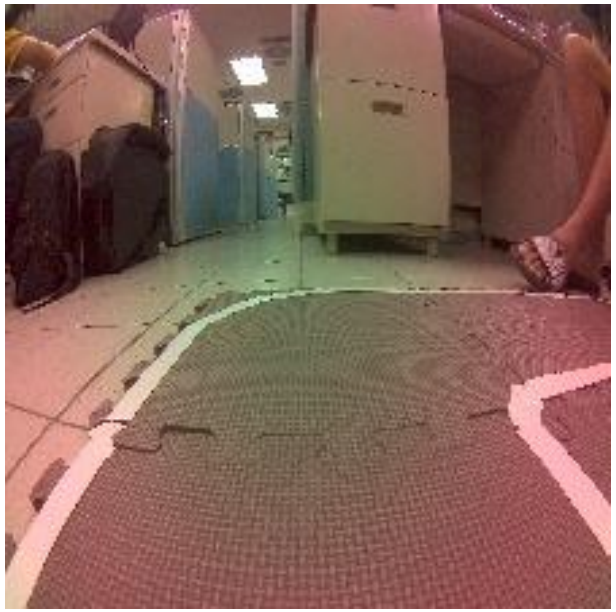
- 在灰階影像當中，每一獨立區塊的產生取決於亮度（灰階值）的不同，利用這樣的特性，我們可以設定一固定的臨界值T，以該值為條件，將影像轉換為二值影像（只包含0及1之二值影像，或0及255之灰階影像），並從中加以擷取出感興趣的影像區塊。

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

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



二值化-利用門檻值對車道影像進行分割



$g(x,y) \geq 170$

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

二值化-OpenCV(C++)

OpenCV固定門檻值二值化

- double threshold(const Mat &src, Mat &dst, double thresh, double maxval, int type)
  - src：輸入影像，只能輸入單通道，8位元或32位元浮點數影像。
  - dst：輸出影像，尺寸大小、深度會和輸入圖相同。
  - 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。

二值化-OpenCV(Python)

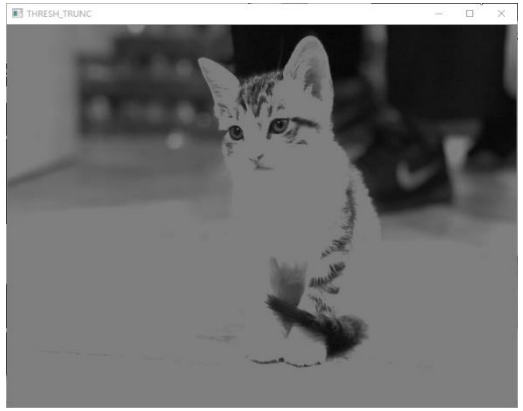
OpenCV固定門檻值(threshold)二值化

- 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。

二值化-OpenCV

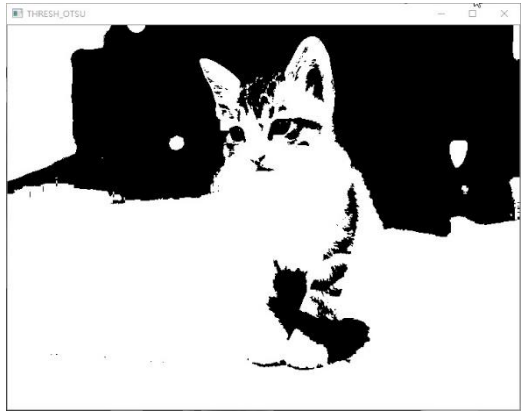
THRESH_BINARY

THRESH_BINARY_INV



THRESH_BINARY_TRUNC

THRESH_TOZERO



THRESH_TOZERO_INV

THRESH_OSTU

二值化-OpenCV

五種門檻值參數介紹：

- THRESH_BINARY：大於門檻值的像素設為最大值(maxval)，小於門檻值的設為0。
- THRESH_BINARY_INV：大於門檻值的像素設為0，小於門檻值的設為最大值(maxval)。
- THRESH_TRUNC：大於門檻值的像素設為門檻值，小於門檻值的設為0。
- THRESH_TOZERO：大於門檻值的像素值不變，小於門檻值的設為0。
- THRESH_TOZERO_INV：大於門檻值的像素值設為0，小於門檻值的不變。

影像強化

- 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，因色彩近似或亮度不夠以致融入背景之中等情況下，影像就很難看清。對於這種影像，如果把號誌或線條的色彩和亮度作適當處理，使其與背景產生微秒的差別，就可以使所需特徵更明確，以供後續辨識使用。
- 像這樣通過增強亮度或色彩等各種包含於影像中的資訊，或者轉換成其他資訊，就可以製作成清晰影像，這種處理過程稱為影像增強。

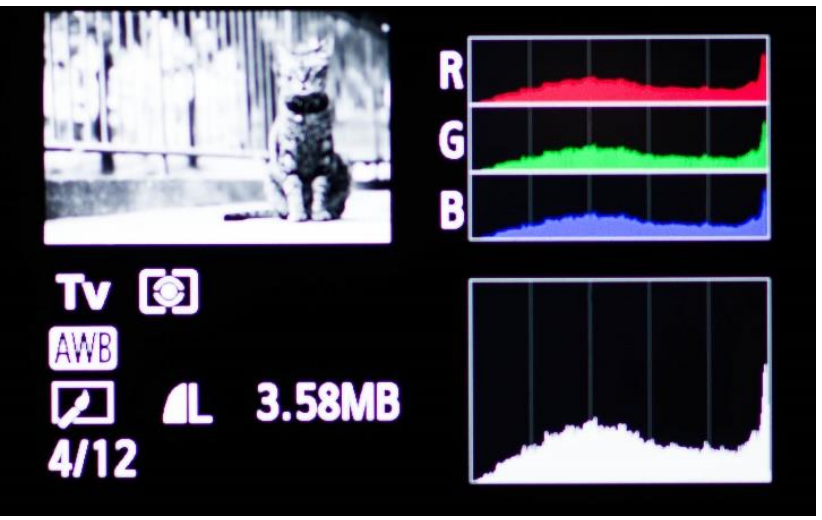
影像強化範例

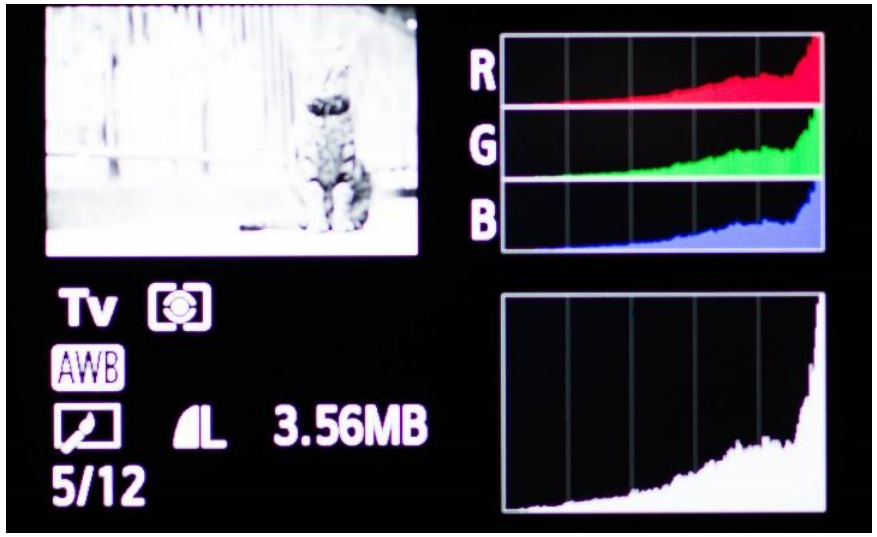
直方圖(Histogram)-簡介

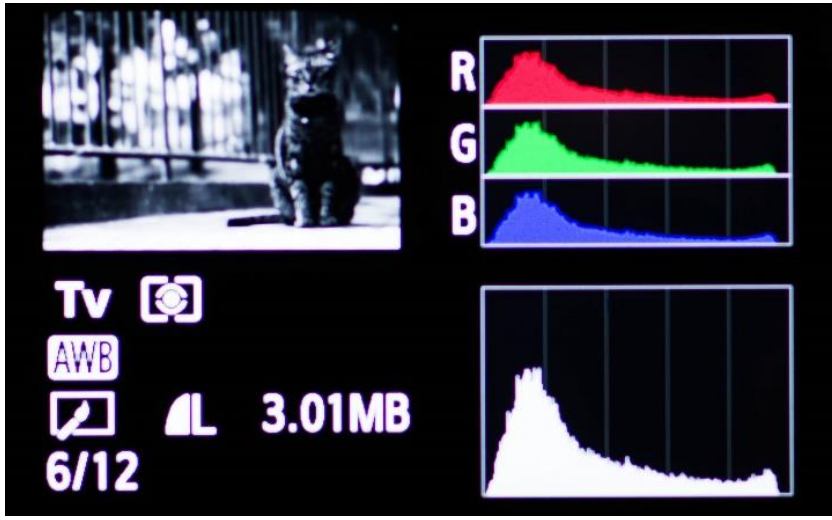
- 直方圖是一個表現影像中像素分布的統計表，橫軸為影像中所有的像素值大小(假設是8位元影像，那就是範圍為0到255)，縱軸為每個像素在影像中的個數。

- 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(Histogram)-簡介







Histogram-計算-OpenCV(C++)

OpenCV calcHist()函式：

- void calcHist(const Mat &src, int nimages, const int* channels, InputArray mask, OutputArray hist, int dims, const int* histSize, const float** ranges, bool uniform=true, bool accumulate=false)
  - src：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。
  - nimages：輸入影像之張數。
  - 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。

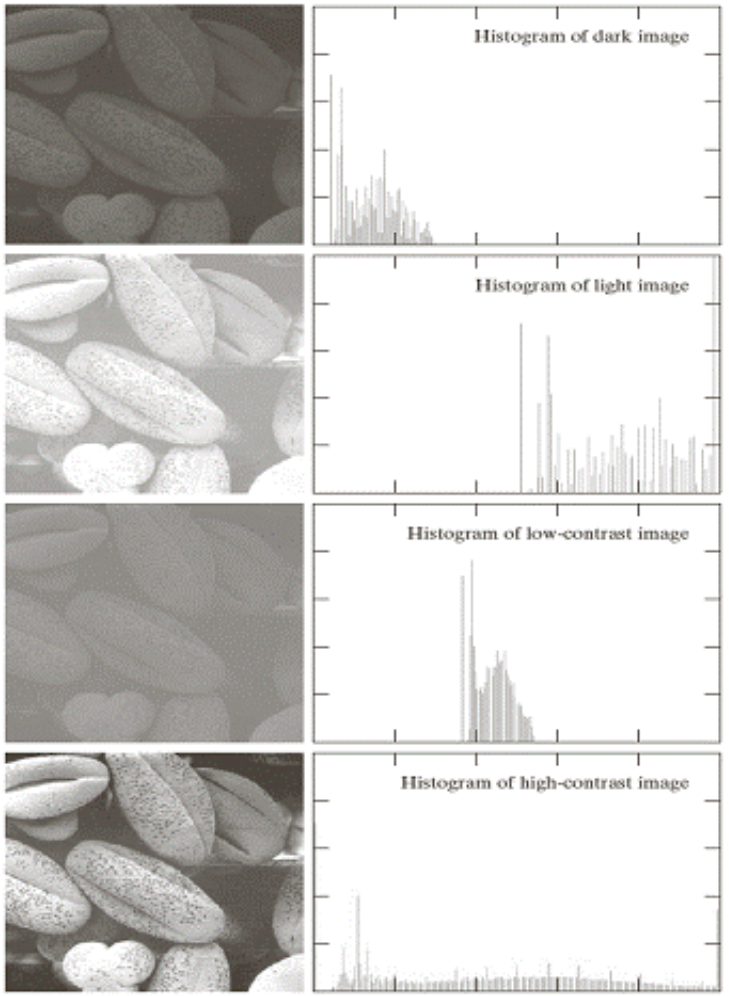
Histogram-計算-OpenCV(Python)

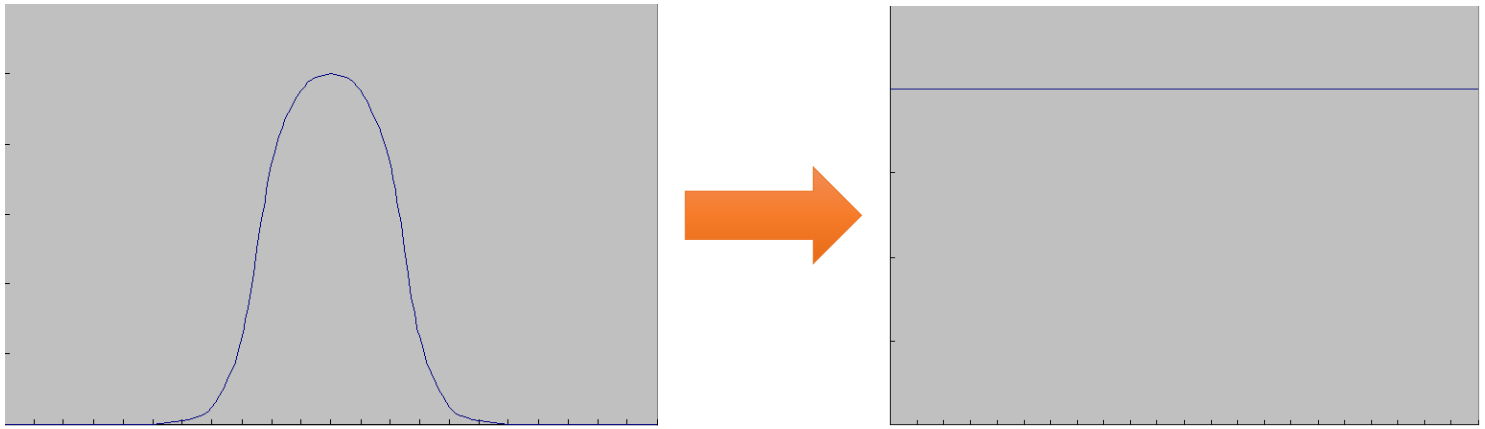
OpenCV calcHist()函式：

- 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。
  - Ex. hist = cv2.calcHist(gray_img, 0, None, 256, [0, 256])

Histogram-
直方圖等化(equalizeHist)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。





Histogram-
直方圖等化(equalizeHist)流程

- 計算輸入圖的直方圖。
- 將直方圖歸一化到總合為255。
- 計算直方圖累計表。
- 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。

Histogram-
直方圖等化(equalizeHist)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。

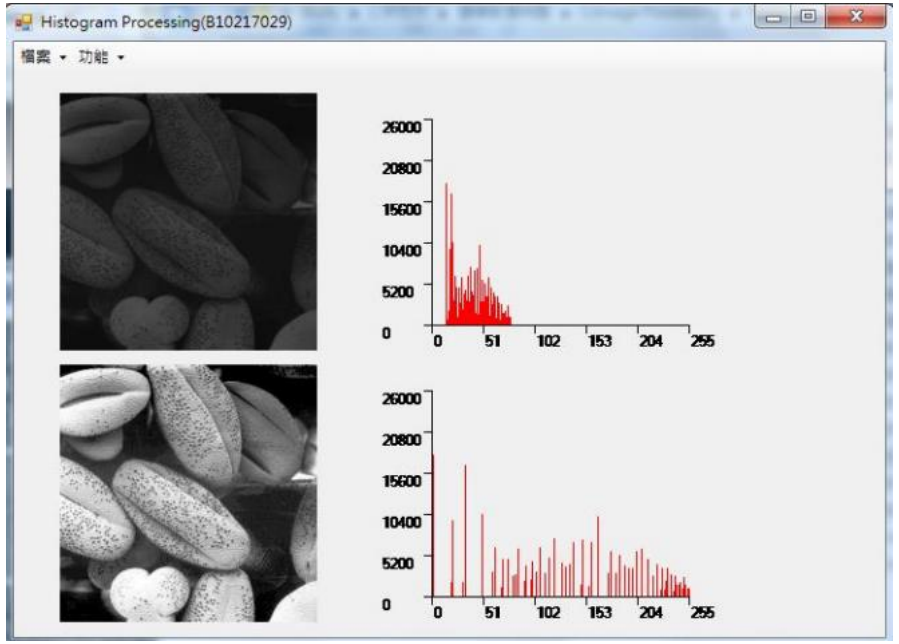
灰階級	7	6	5	4	3	2	1	0
原影像的各級像素數	0	4	9	11	5	7	4	0
等化後的各級的像素數	5	5	5	5	5	5	5	5

- 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。

Histogram-
直方圖等化(equalizeHist)-OpenCV(C++)

OpenCV直方圖等化函式

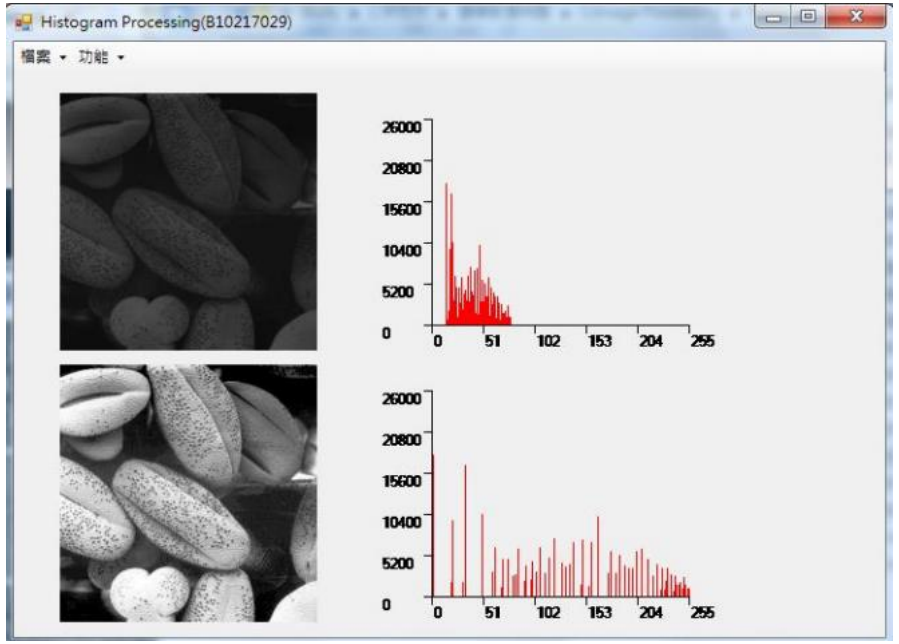
- void equalizeHist(const Mat &src, Mat &dst)
  - src：輸入影像，8位元單通道圖。
  - dst：輸出影像，和輸入影像尺寸、型態相同。



Histogram-
直方圖等化(equalizeHist)-OpenCV(Python)

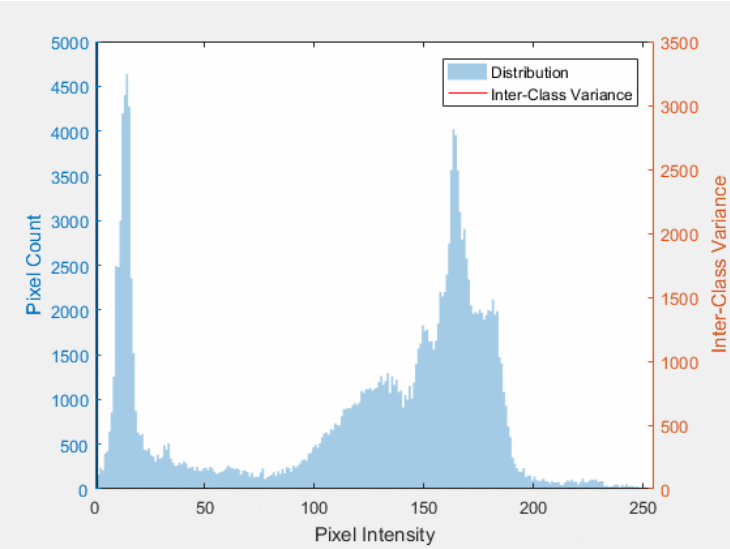
OpenCV直方圖等化函式

- 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。

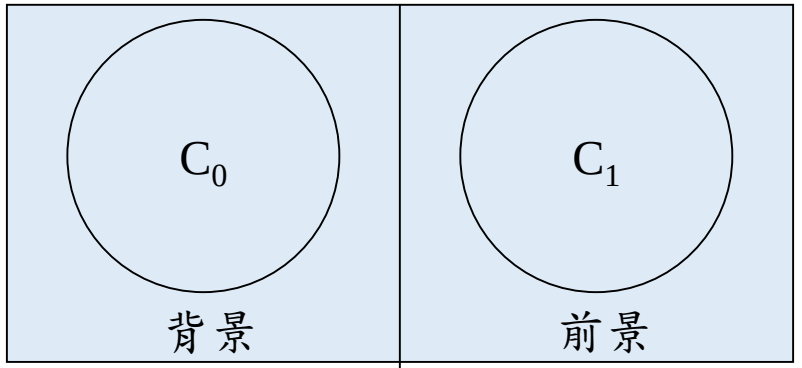


Otsu(自適應門檻值)流程

1. 計算每個強度集的直方圖和機率
2. 設定$w_i(0)$和$u_i(0)$的初始值
3. 遍歷所有可能的門檻值t=1…最大強度
   1. 更新$w_i(0)$和$u_i(0)$
   2. 計算$\sigma_b^2(t)$
4. 所需的門檻值對應於最大的$\sigma_b^2(t)$
5. 可以計算兩個最大值(和兩個對應的)。$\sigma_{b1}^2(t)$是最大值 $\sigma_{b2}^2(t)$是更大獲相等的最大值
6. 所需的門檻值=$\frac{threshold_1+threshold_2}{2}$

Otsu(自適應門檻值)決定法(1-1)

- 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。
- 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)

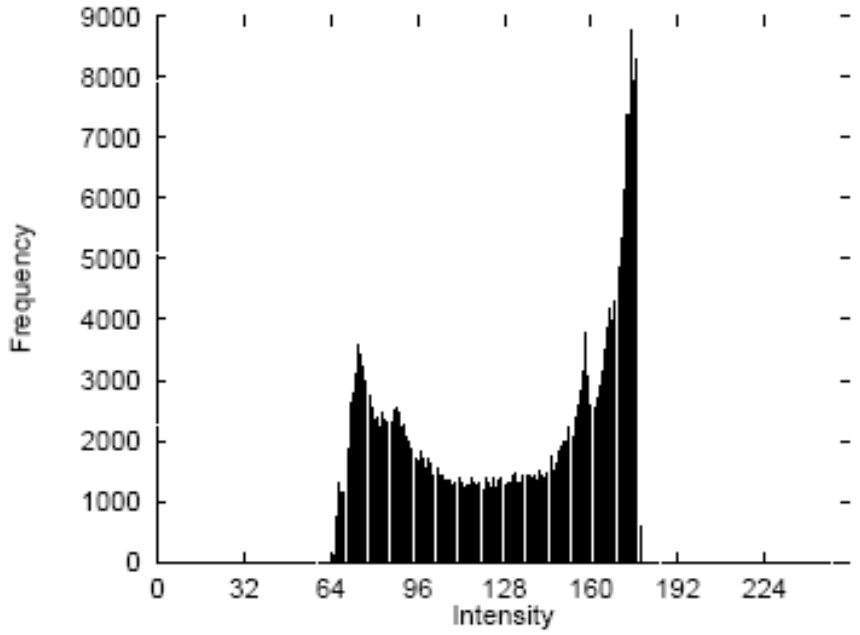


Figure 6.3: A bi-modal histogram.

Otsu(自適應門檻值)決定法(1-3)

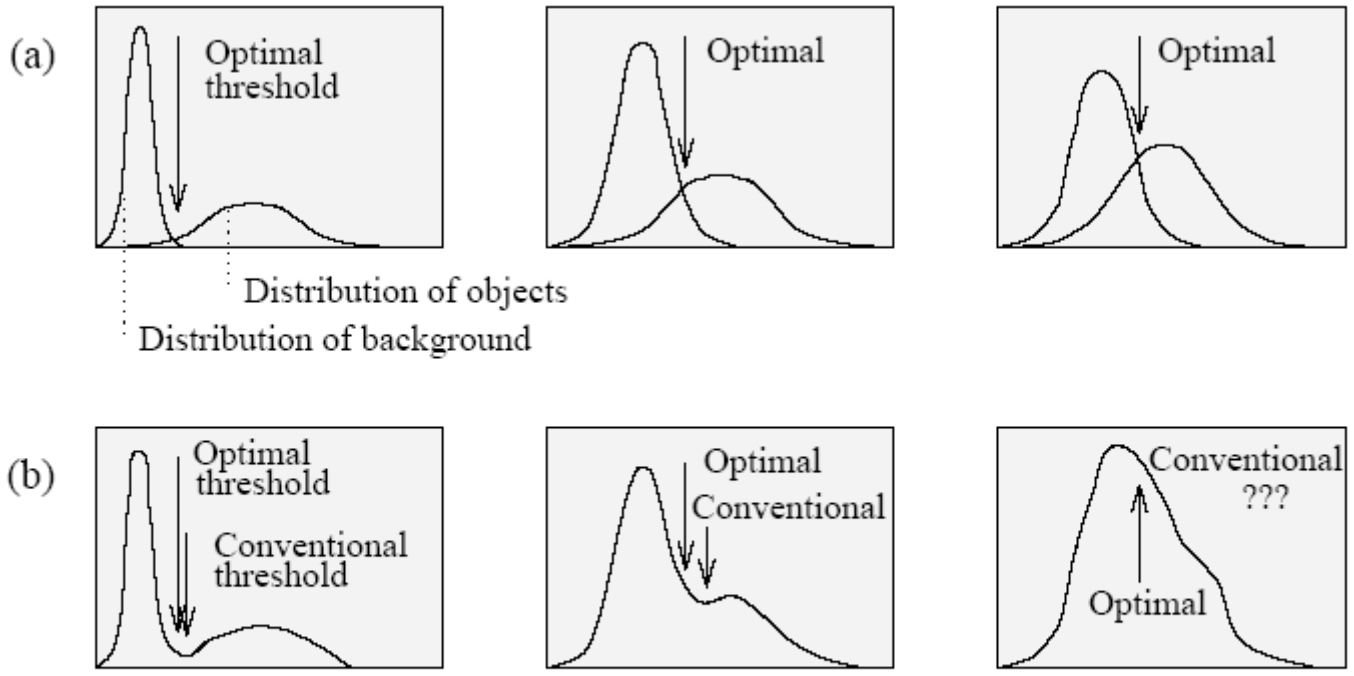


Figure 6.4: Gray-level histograms approximated by two normal distributions—the threshold is set to give minimum probability of segmentation error. (a) Probability distributions of background and objects. (b) Corresponding histograms and optimal threshold.

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$0 \le i \le I-1$，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threshold()函式，使用方式也一樣，只是最後一個參數增加CV_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、深度會和輸入圖相同。
- 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

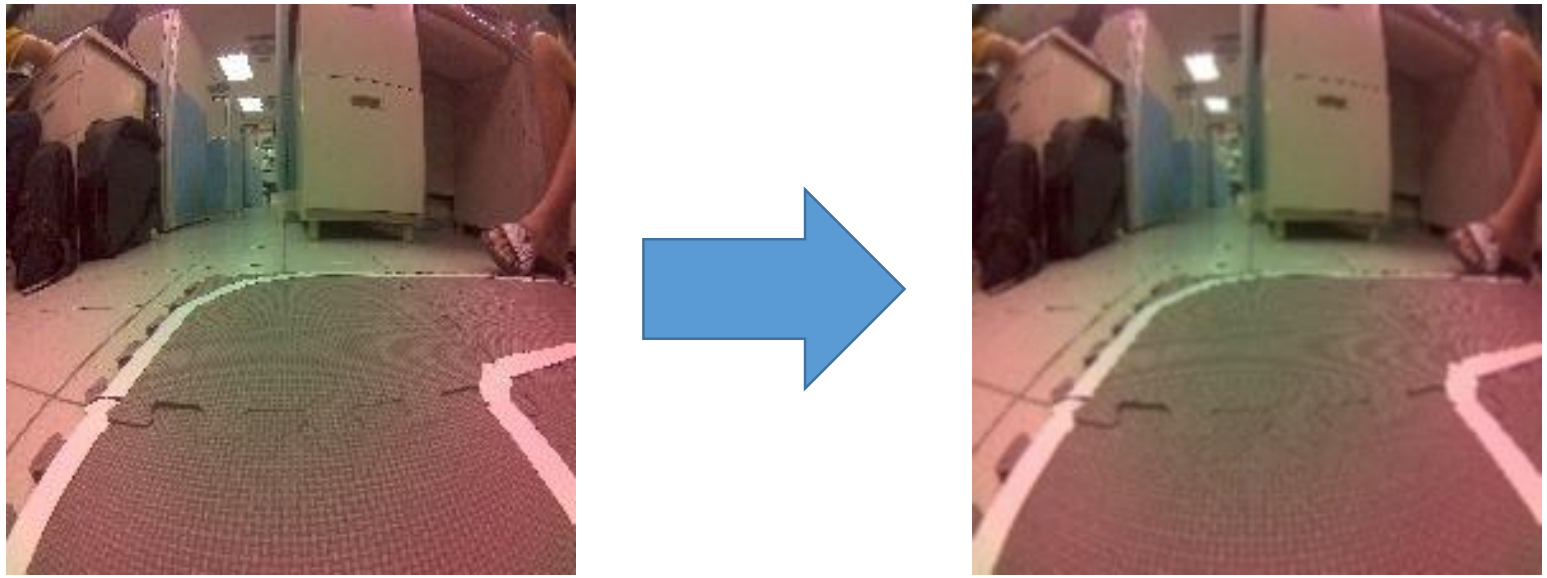
OpenCV
Image Processing Denoise

影像去雜訊(Image Denoising)

- 現實中的影像在傳輸過程中，常受到成像設備和外部環境雜訊的干擾，受到此影響產生的影像稱為雜訊影像，減少此影像中雜訊的過程即為影像降噪。
- 雜訊的多寡決定了影像訊號的品質。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，因使用車的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器去除影像中的雜訊。下圖以高斯濾波器作為示範。

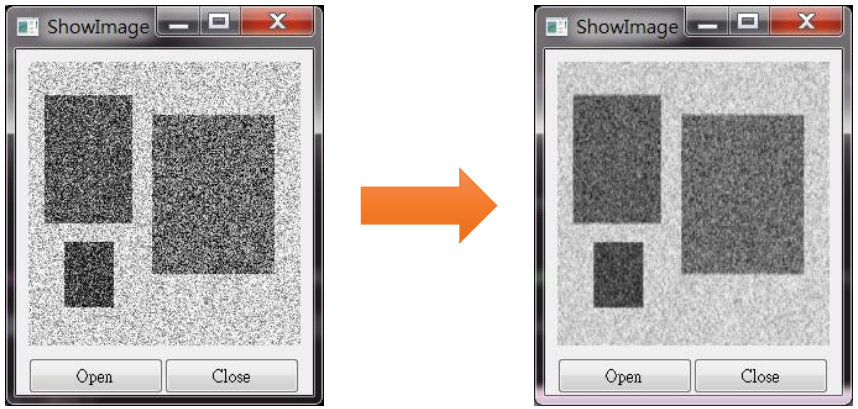


濾波器常見種類

- 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，這樣的程序使灰階圖「銳利」變化降低。隨機雜訊通常在灰階圖含有銳利的變化，所以均值濾波器常常使用於減少雜訊。
- 但是邊緣也在灰階圖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，它採取把某像素值置換為該像素周圍3x3個像素加總大小平均值的方法，將整張影像予以平滑化。

$$q = \frac{p0+p1+p2+p3+p4+p5+p6+p7+p8}{9}$$

均值濾波-OpenCV(C++)

OpenCV blur()函式：

- void blur(const Mat &src, Mat &dst, Size ksize, Point anchor = Point(-1,-1), int borderType = BORDER_DEFAULT)
  - src：輸入影像。
  - dst：輸出影像會和輸入圖尺寸、型態相同。
  - ksize：模板大小，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

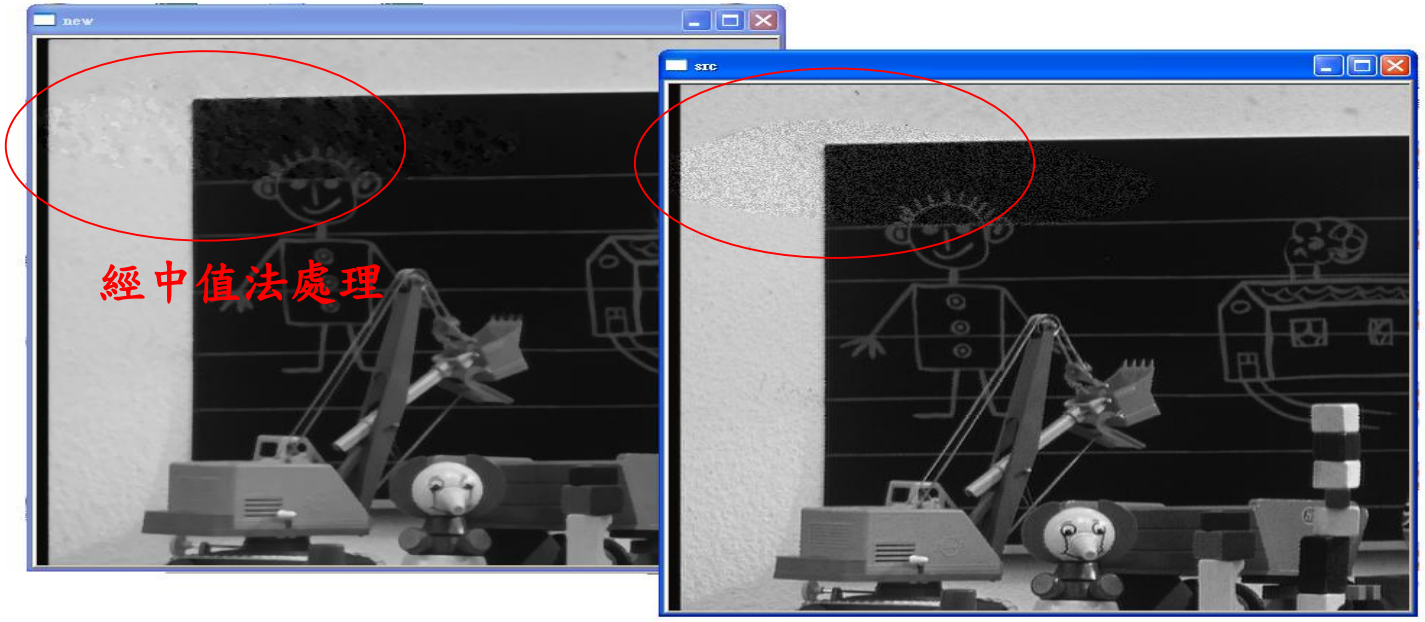
均值濾波-OpenCV(Python)

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
  - Ex:blur = cv2.blur(img, (5, 5))

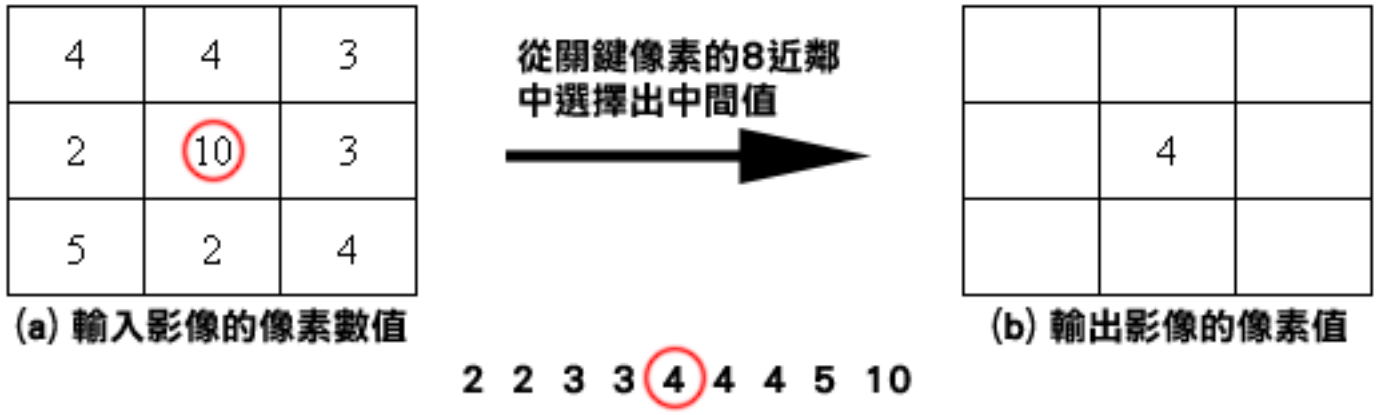
中值濾波(Median Filter)

- 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模糊化較輕微。



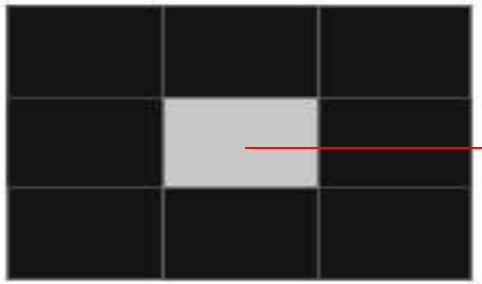
中值濾波(Median Filter)

- 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

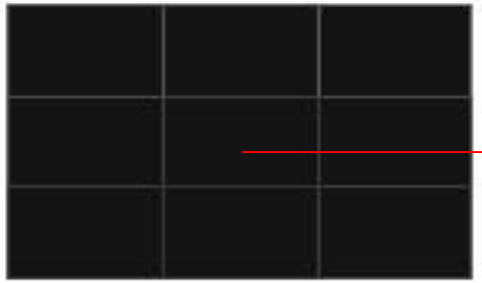


原始雜訊影像

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(C++)

OpenCV medianBlur()函式：

- void medianBlur(const Mat &src, Mat &dst, int ksize)
  - src：當ksize為3或5時，輸入影像可以為多通道的CV_8U、CV_16U或CV_32F，在更大的模板時，只能使用CV_8U的型態。
  - dst：輸出影像會和輸入圖尺寸、型態相同。
  - ksize：模板大小，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)。

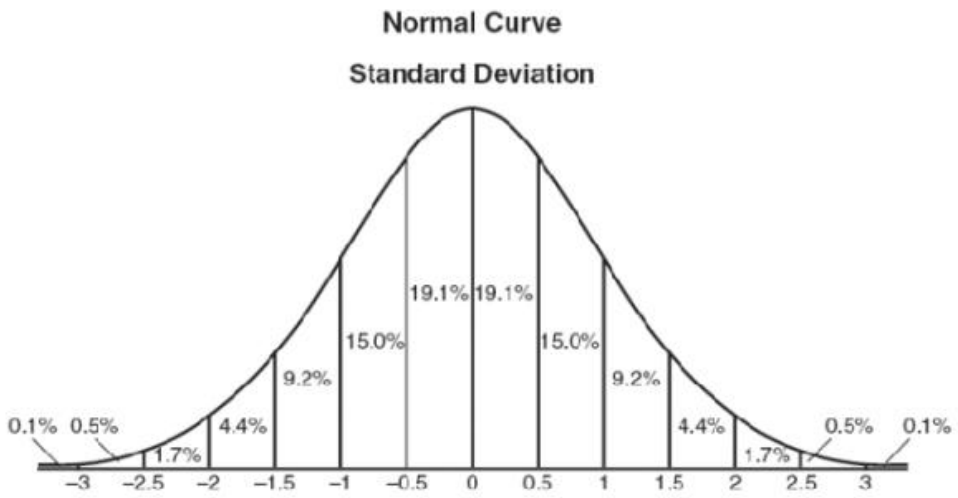
中值濾波 – OpenCV(Python)

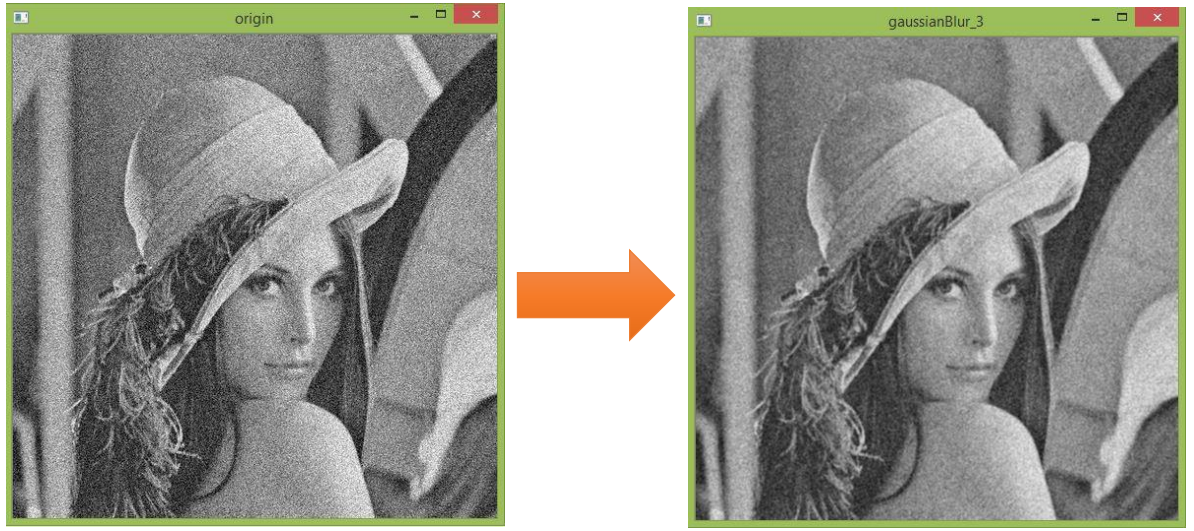
OpenCV medianBlur()函式：

- 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)。
  - Ex：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(C++)

OpenCV GaussianBlur()函式：

- void GaussianBlur(const Mat &src, Mat &dst, Size ksize, double sigmaX, double sigmaY)
  - src：輸入影像可以為多通道圖，通常使用單通道灰階圖，例如CV_8U或CV_16U。
  - dst：輸出影像會和輸入圖尺寸、型態相同。
  - ksize：模板大小，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。

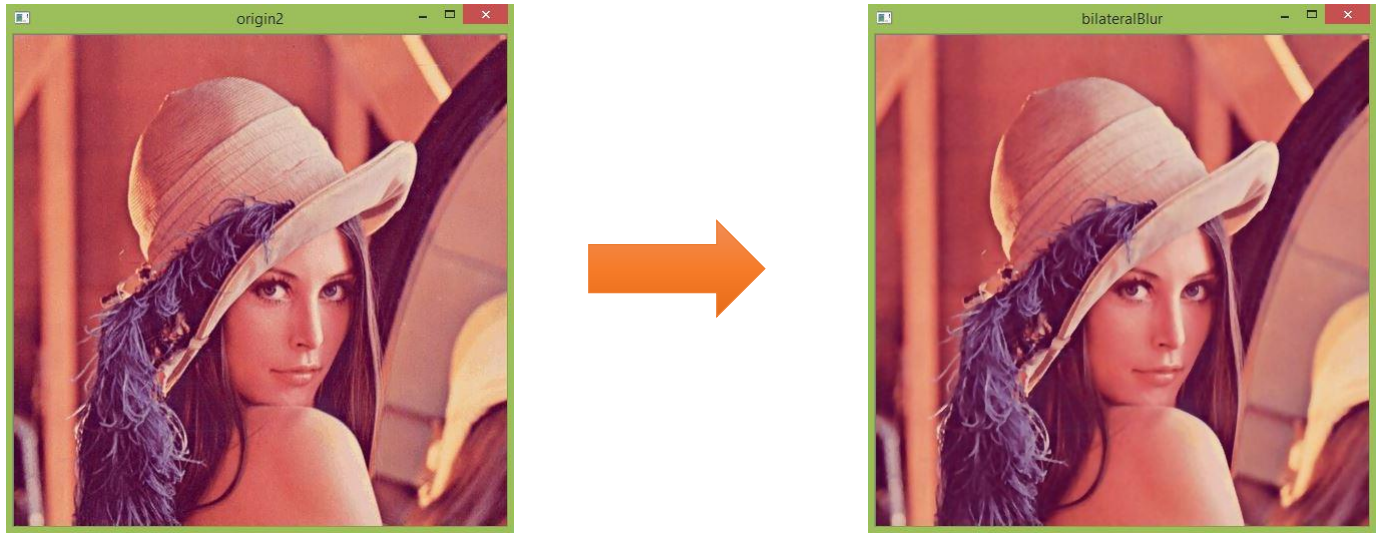
高斯濾波-OpenCV(Python)

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。
  - Ex：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，還多考慮了像素之間亮度及色彩上的差異，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(亮度/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(C++)

OpenCV bilateralFilter()函式：

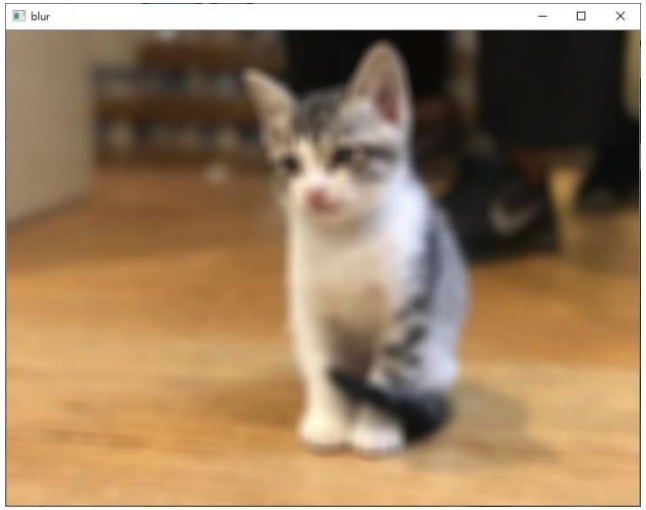
- void bilateralFilter(const Mat &src, Mat &dst, int d, double sigmaColor, double sigmaSpace)
  - src：輸入影像。
  - dst：輸出影像會和輸入影像尺寸、型態相同。
  - d：過程中各像素會使用到的鄰域直徑大小，5以上
  - sigmaColor : 該參數的較大值意味著像素鄰域內的更多顏色將被混合在一起，會得到較大的半等色區域10以上150以下。
  - sigmaSpace : 坐標空間中的過濾器sigma。參數越大意味著只要其顏色足夠近，更遠的像素就會相互影響。

雙邊濾波-OpenCV(Python)

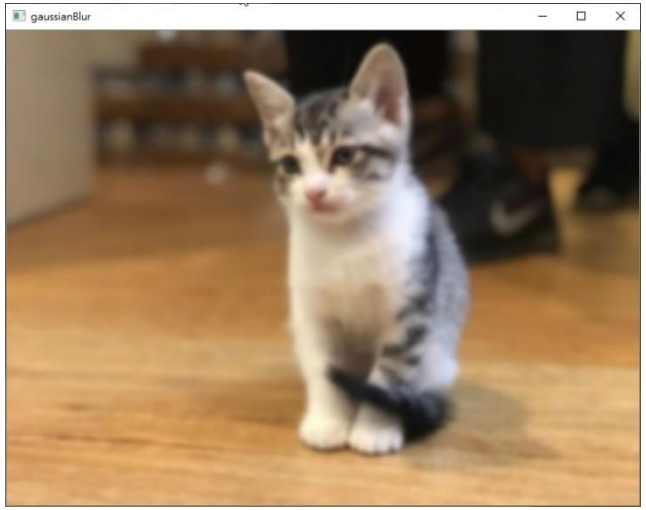
OpenCV bilateralFilter()函式：

- 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
  - Ex：blur4 = cv2.bilateralFilter(img, 19, 75, 75)

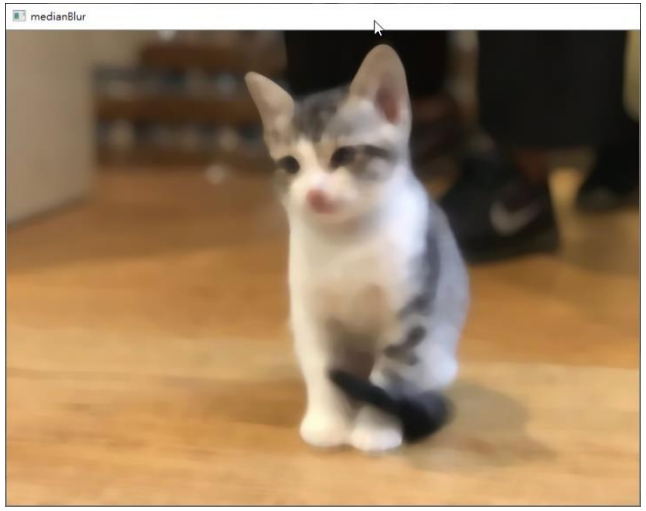
常見濾波器-OpenCV



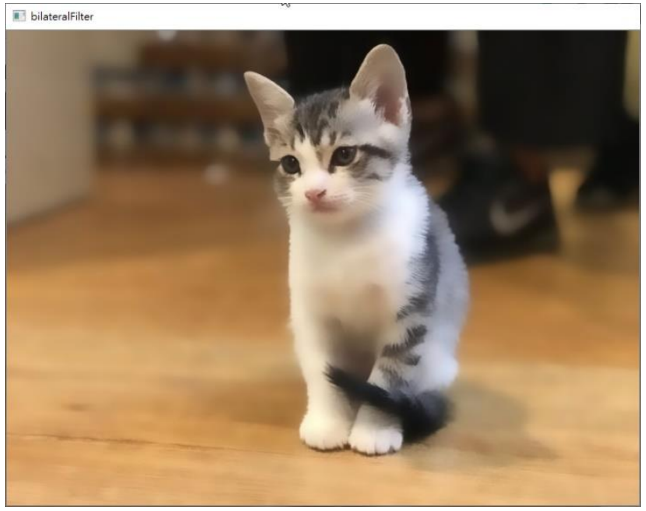
Blur(均值濾波)



GaussianBlur(高斯濾波)



MedianBlur(中值濾波)



BiliteralFilter(雙邊濾波)

膨脹、侵蝕、形態學

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

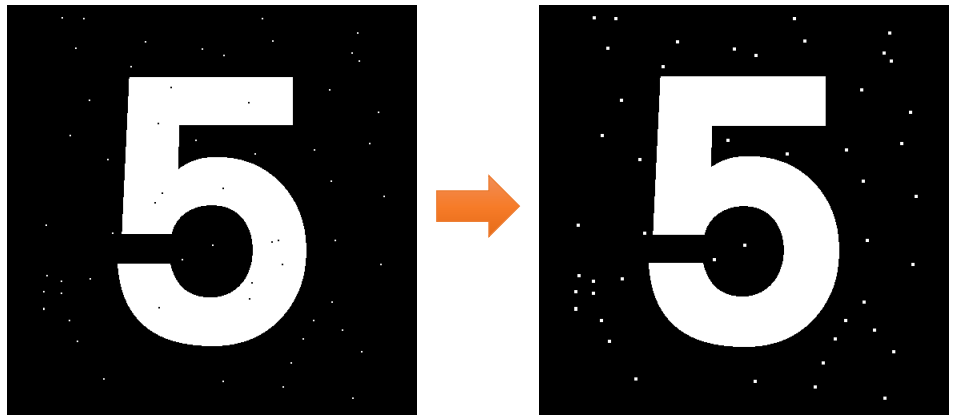
- 二值影像的雜訊，稱為椒鹽狀雜訊，它來自英語 salt and pepper noise 一詞的直譯。這種雜訊能用中值濾波法將其除去，基於此雜訊的二值性，也有膨脹、侵蝕的處理方法。
- 膨脹(dilation)是指某像素p的近鄰中，若有一個為 1，則將p置為1。
- 侵蝕(erosion)是指某像素p的近鄰中，若有一個為 0，就將p置為0。
- Closing運算：膨脹→侵蝕(除去黑色雜訊，白色雜訊依然殘留)
- Opening運算：侵蝕→膨脹(除去白色雜訊，黑色雜訊依然殘留)

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

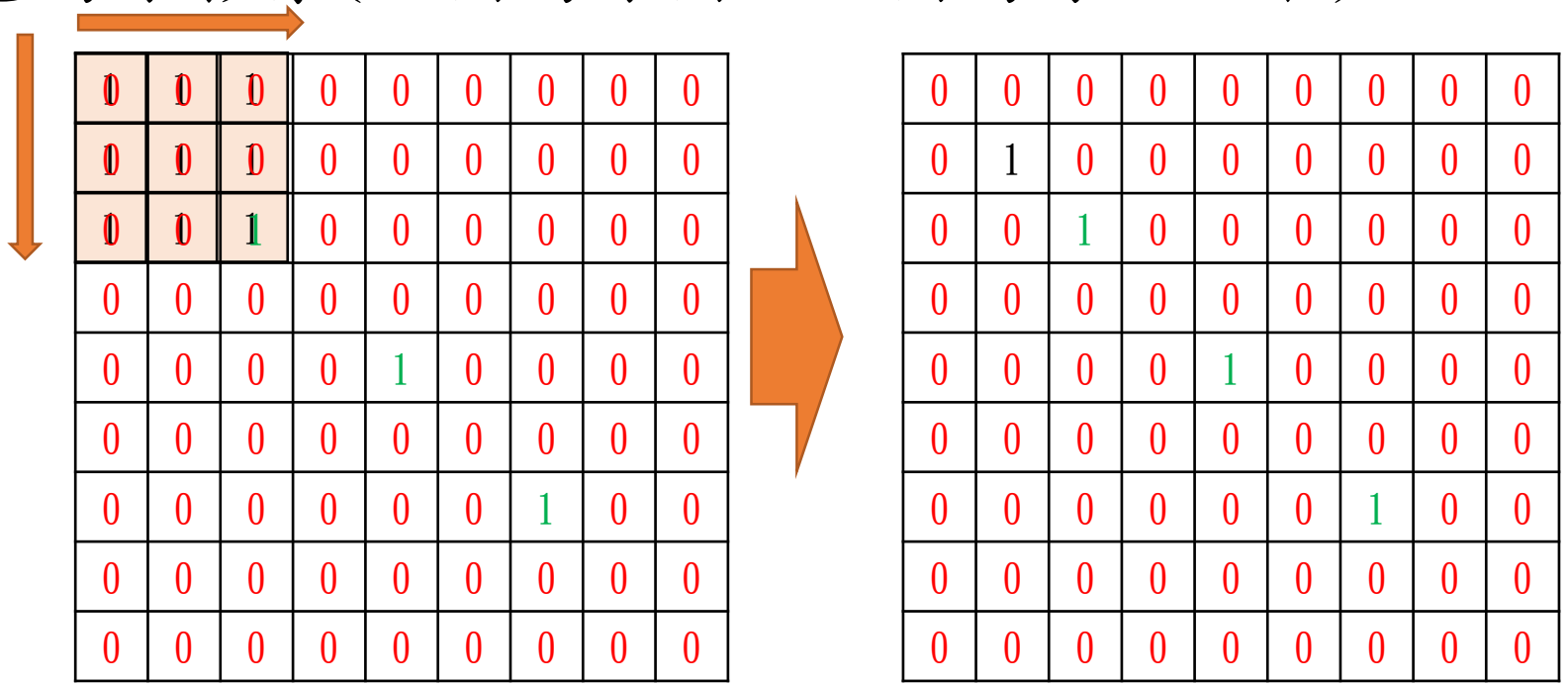
A

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

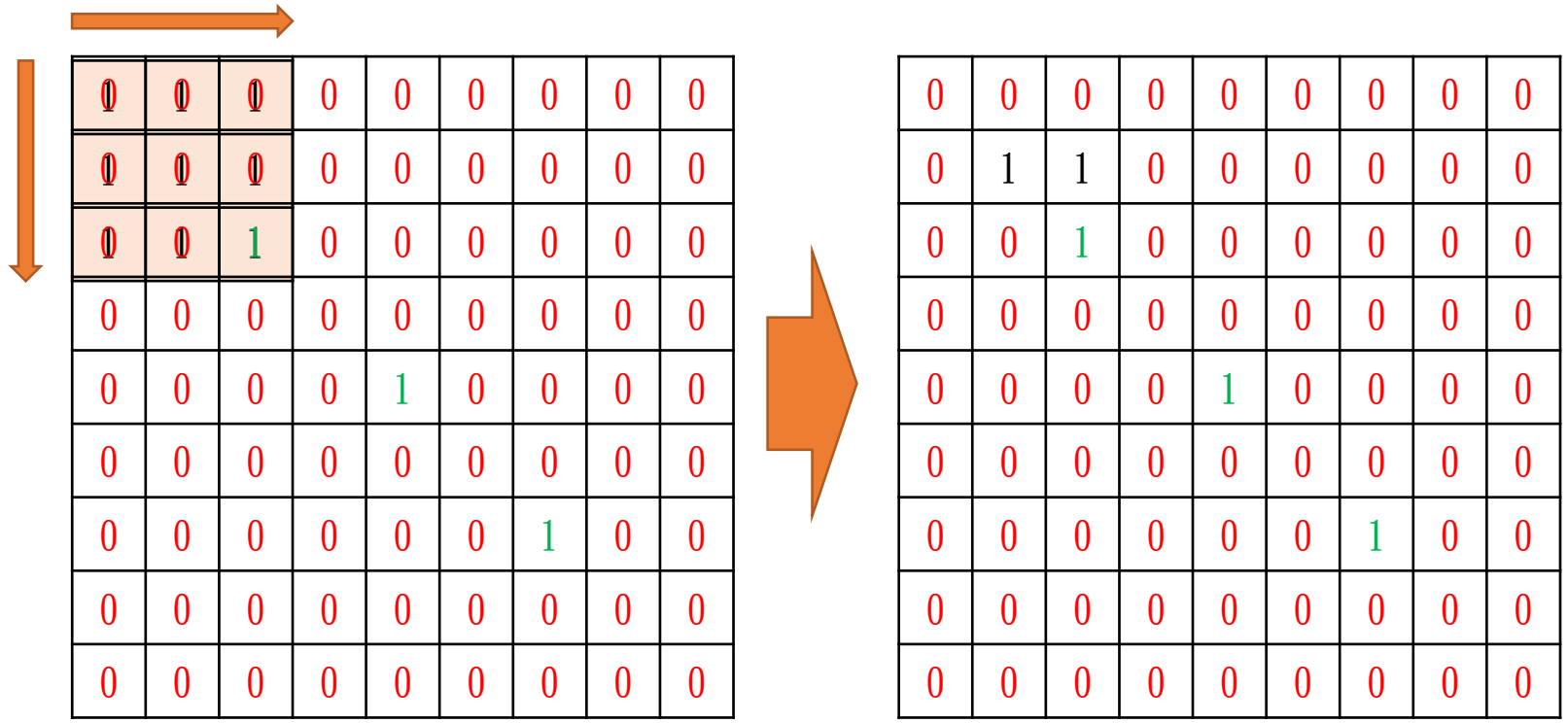
膨脹(Dilation)

- 將A由左而右，由上而下尋訪，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，右圖為尋訪結果)



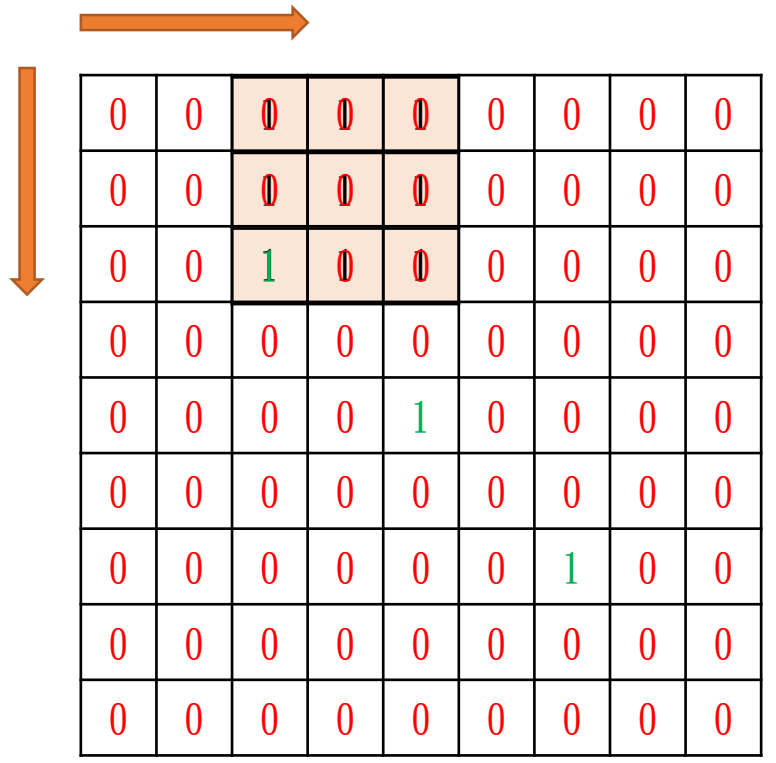
膨脹(Dilation)

- 綠色及紅色為原影像。
- 黑色為尋訪後。
- 接著繼續尋訪。



膨脹(Dilation)

- 綠色及紅色為原影像。
- 黑色為尋訪後。
- 接著繼續尋訪。

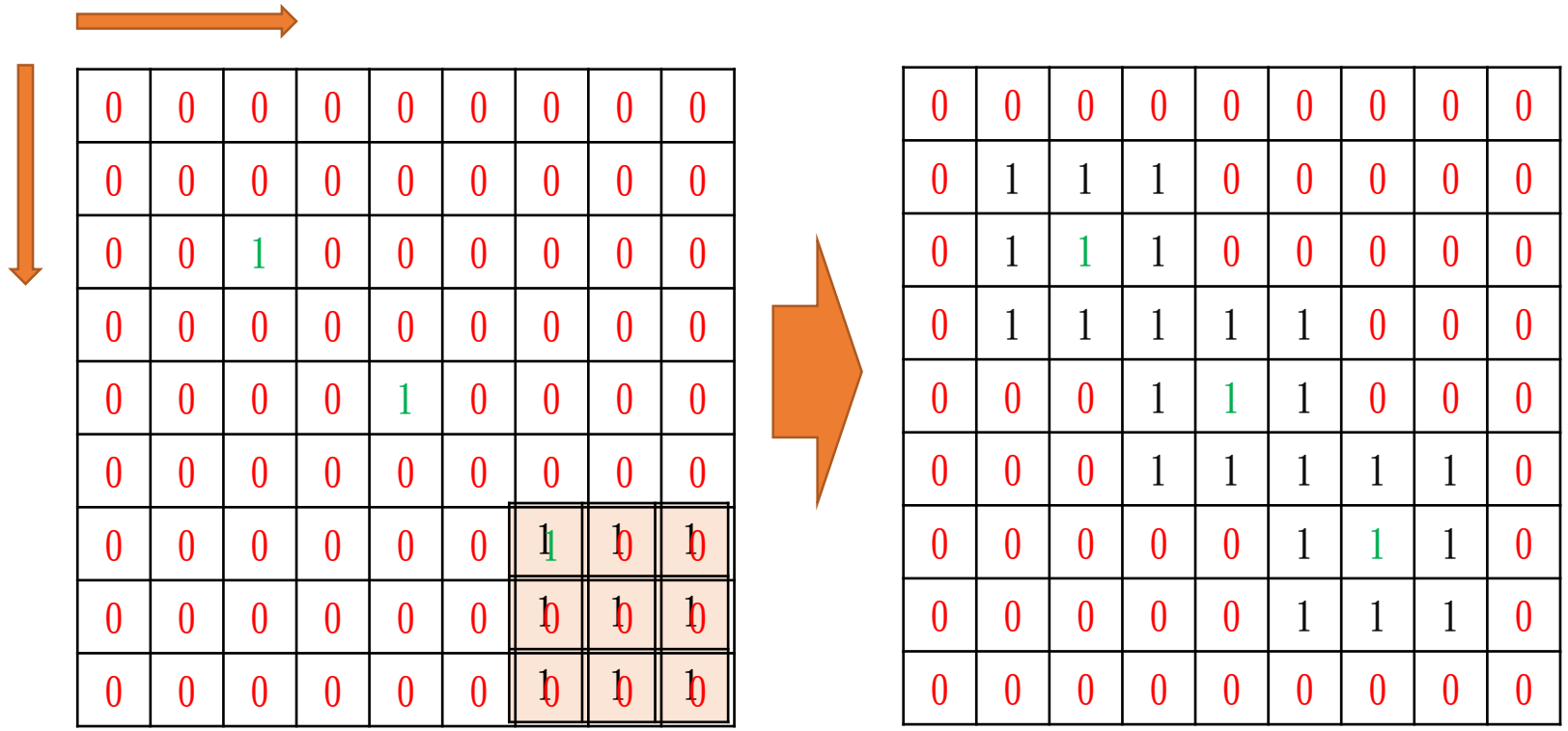


0	0	1	1	1	0	0	0	0
0	0	1	1	1	0	0	0	0
0	0	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

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

膨脹(Dilation)

- 最終尋訪結果。
- 綠色及紅色為原影像。
- 黑色為尋訪後。



膨脹-OpenCV(C++)

OpenCV dilate()函式:

- dilate(const Mat &src, Mat &dst, Mat kernel, Point anchor=Point(-1,-1), int iterations=1)
  - src：輸入影像。
  - dst：輸出影像，和輸入圖尺寸、型態相同。
  - kernel：結構元素，如果kernel=Mat()則為預設的3x3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。

膨脹-OpenCV(Python)

OpenCV dilate()函式:

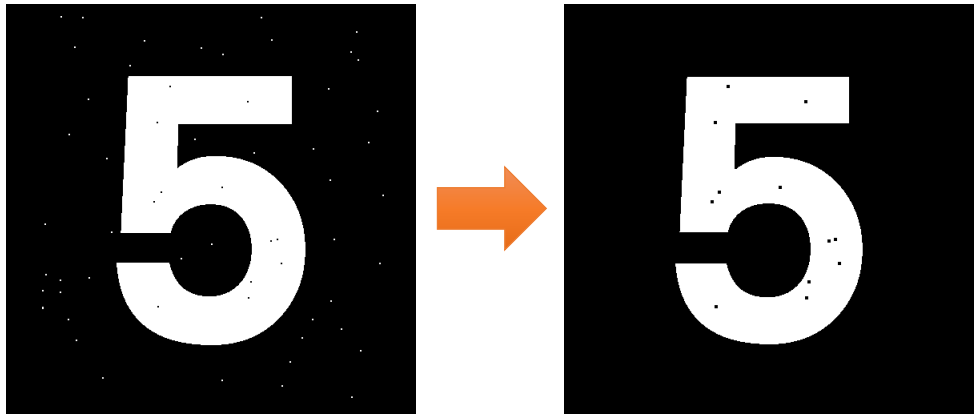
- dst = cv2.dilate(<src>, <kernel>[, <dst>[, <anchor>[, <iterations>[, <borderType>[, <borderValue>]]]]])
  - src：輸入影像。
  - dst：輸出影像，和輸入影像尺寸、型態相同。
  - kernel：結構元素，預設為3x3的矩形，越大膨脹效果越明顯。
  - anchor：原點位置，預設為結構元素的中央。
  - iterations：執行次數，執行越多次膨脹效果越明顯。
  - borderType ：邊界類型，邊界模式用來推斷影像外的像素。
  - Ex：out = cv2.dilate(img,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

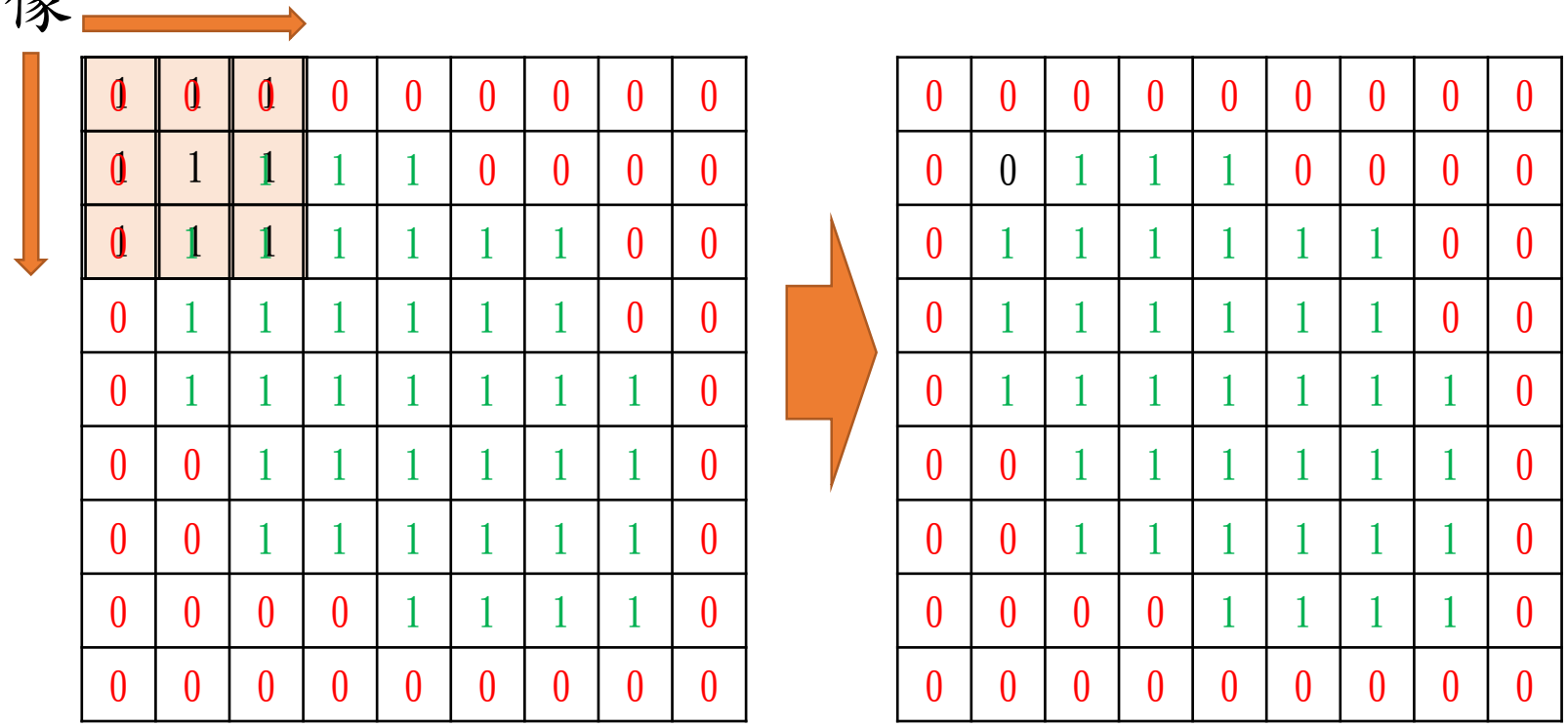
A

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

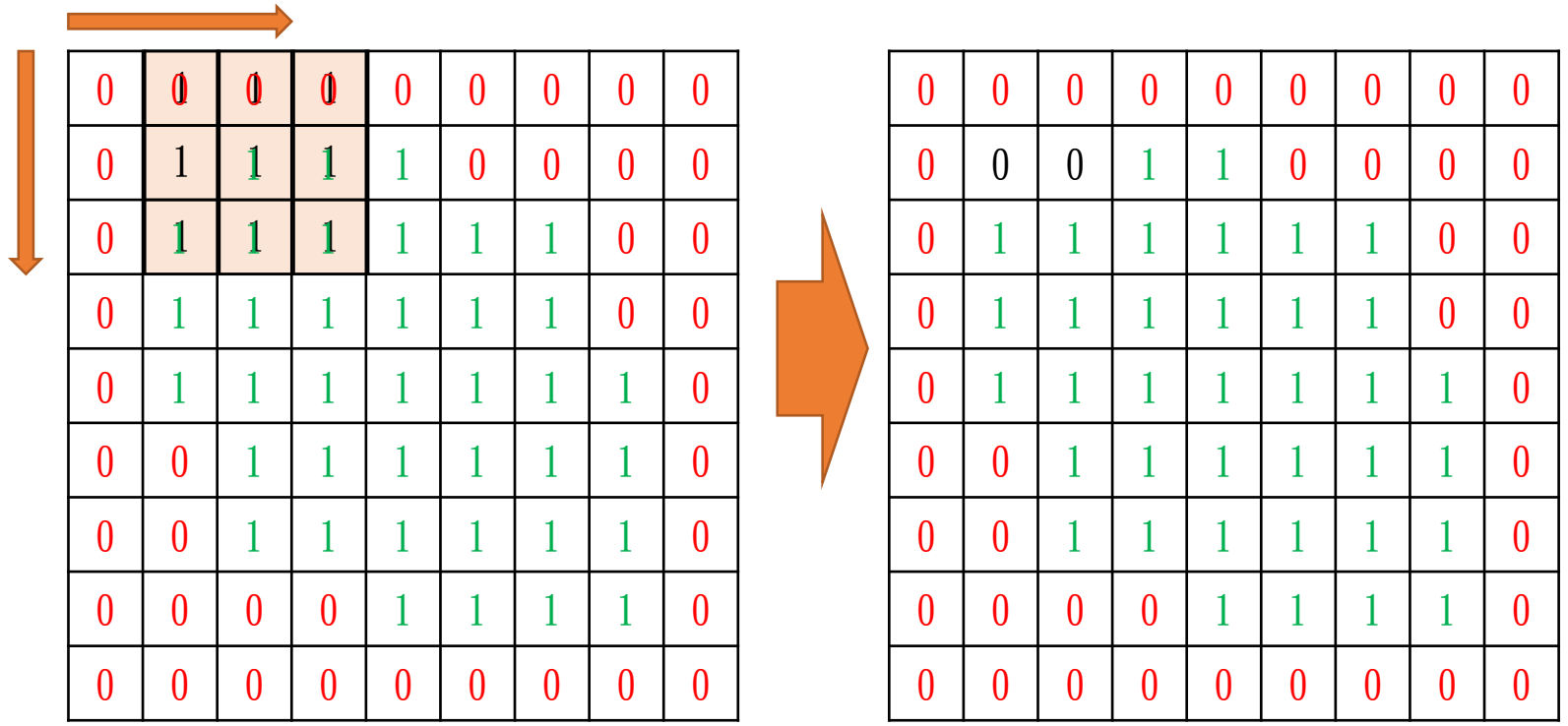
侵蝕(Erosion)

- 將A由左而右，由上而下尋訪，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



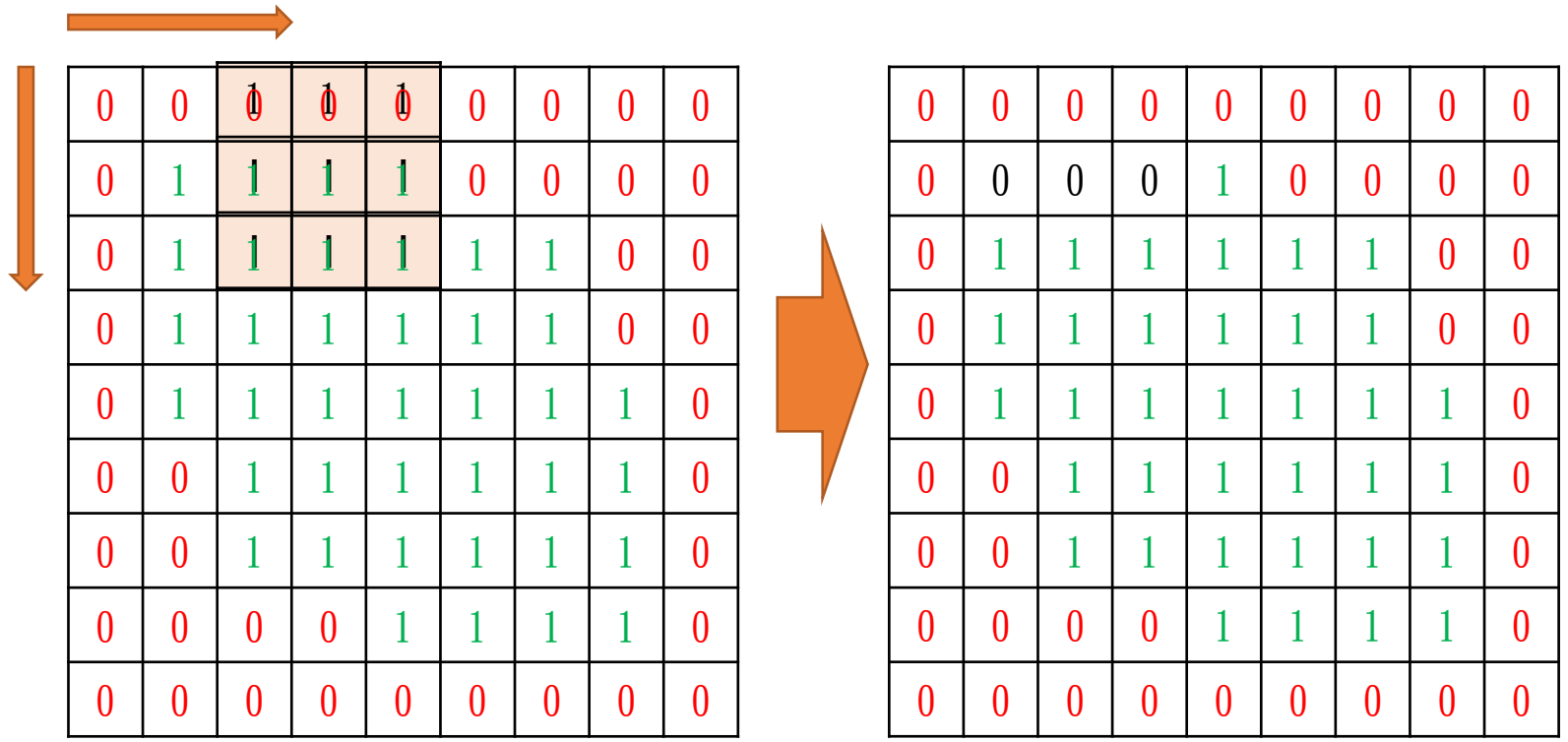
侵蝕(Erosion)

- 綠色及紅色為原影像。
- 黑色為尋訪後。
- 接著繼續尋訪。



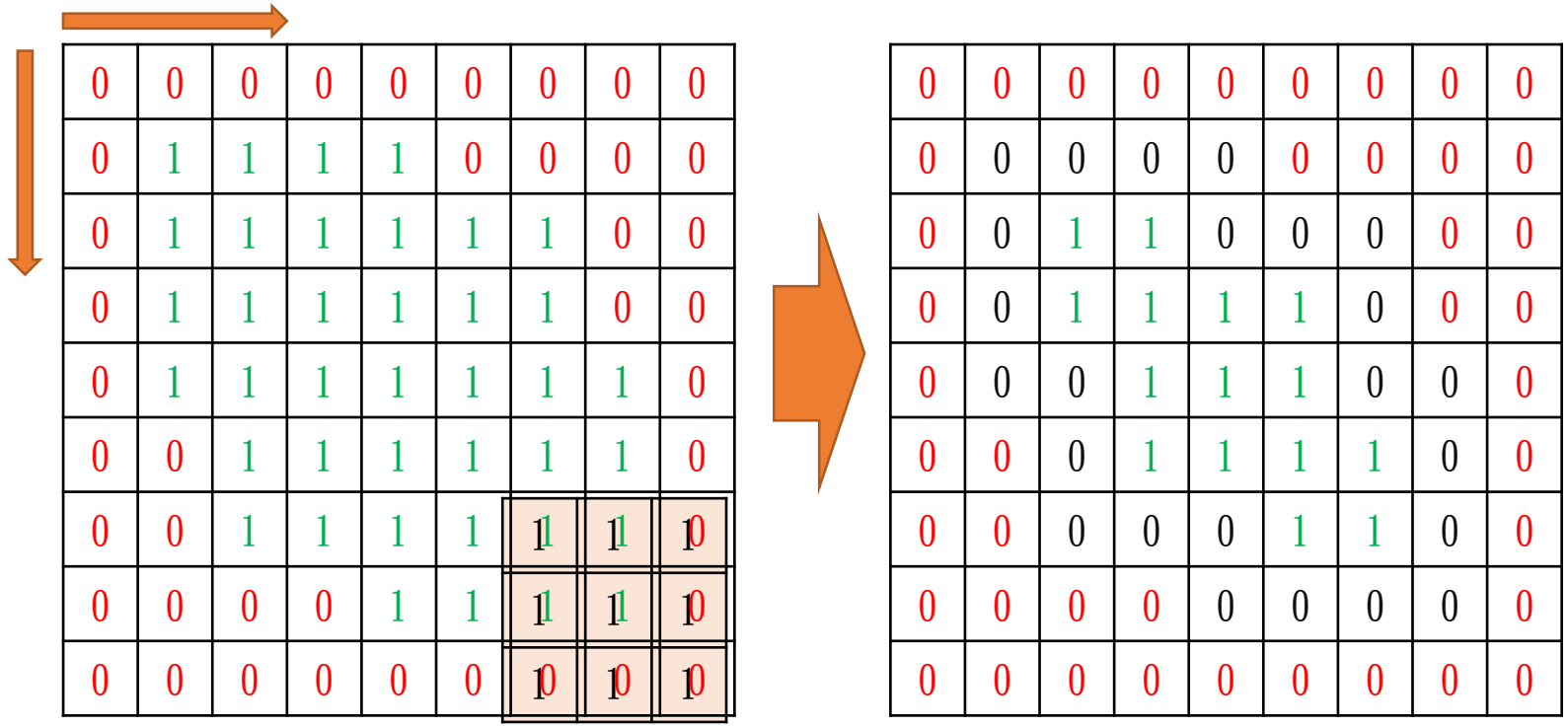
侵蝕(Erosion)

- 綠色及紅色為原影像。
- 黑色為尋訪後。
- 接著繼續尋訪。



侵蝕(Erosion)

- 完全尋訪後之結果。
- 綠色及紅色為原影像。
- 黑色為尋訪後。



侵蝕 – OpenCV(C++)

OpenCV erode()函式:

- erode(const Mat &src, Mat &dst, Mat kernel, Point anchor=Point(-1,-1), int iterations=1)
  - src：輸入影像。
  - dst：輸出圖，和輸入圖尺寸、型態相同。
  - kernel：結構元素，如果kernel=Mat()則為預設的3x3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。

侵蝕 – OpenCV(Python)

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
  - Ex：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(邊界和連通區域等)，同時細化、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 = clode(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(C++)

OpenCV morphologyEx()函式：

- morphologyEx(const Mat &src, Mat &dst, int op, Mat kernel, Point anchor=Point(-1,-1), int iterations=1)
  - src：輸入影像。
  - dst：輸出影像，和輸入圖尺寸、型態相同。
  - op：操作種類，決定要進行何種型態學操作。
  - kernel：結構元素。
  - 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。

形態學-OpenCV(C++)

操作種類如下：

- Open
  - 例： morphologyEx(inputImage, open, MORPH_OPEN, Mat(), Point(-1,-1), 2);
- Close
  - 例： morphologyEx(inputImage, close, MORPH_CLOSE, Mat(), Point(-1,-1), 2);
- Gradient
  - 例： morphologyEx(inputImage, gradient, MORPH_GRADIENT, Mat(), Point(-1,-1), 2);
- Top Hat
  - 例： morphologyEx(inputImage, tophat, MORPH_TOPHAT, Mat(), Point(-1,-1), 2);
- Black Hat
  - 例： morphologyEx(inputImage, blackhat, MORPH_BLACKHAT, Mat(), Point(-1,-1), 2);

形態學-OpenCV(Python)

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

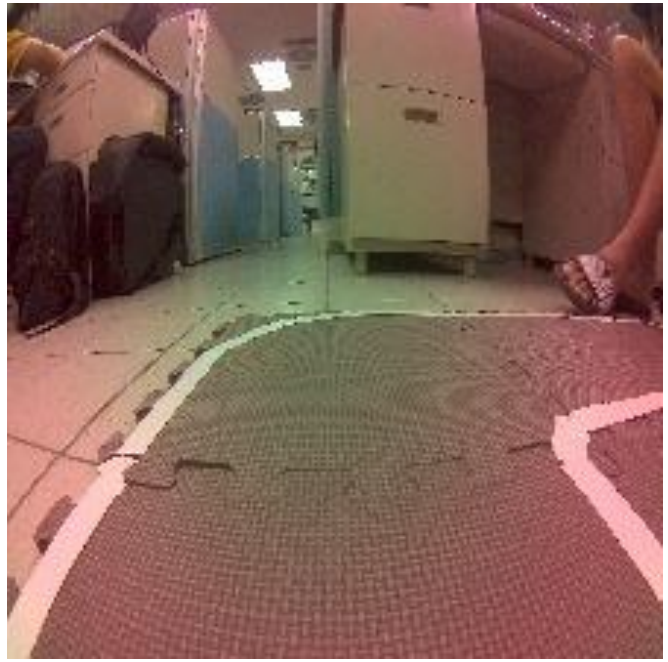
形態學-OpenCV(Python)

操作種類如下：

- Open
  - 例：opening = cv2.morphologyEx(img, cv2.MORPH_OPEN, np.ones((5, 5)))
- Close
  - 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-Open-OpenCV範例程式(C++)

- 片段程式碼：

```
//讀取測試影像
Mat inputImage = imread("pic2.jpg", CV_LOAD_IMAGE_GRAYSCALE);

//用於儲存處理結果
Mat result = Mat::zeros(inputImage.rows, inputImage.cols, inputImage.type());

morphologyEx(inputImage, result, MORPH_OPEN, Mat(), Point(-1,-1), 2);
```

形態學-Close-OpenCV範例程式(C++)

- 片段程式碼：

```
//讀取測試影像
Mat inputImage = imread("pic2.jpg", CV_LOAD_IMAGE_GRAYSCALE);

//用於儲存處理結果
Mat result = Mat::zeros(inputImage.rows, inputImage.cols, inputImage.type());

morphologyEx(inputImage, result, MORPH_CLOSE, Mat(), Point(-1,-1), 2);
```

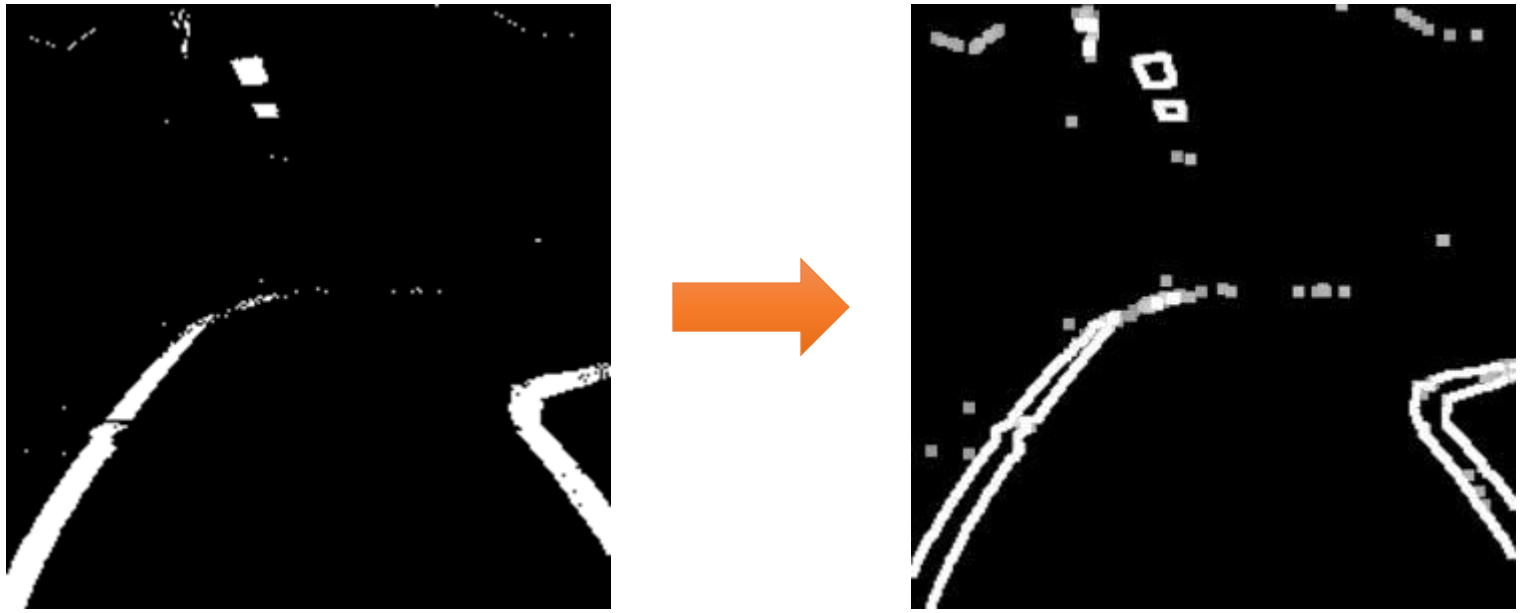
形態學-Gradient-OpenCV範例程式(C++)

- 片段程式碼：

```
//讀取測試影像
Mat inputImage = imread("pic2.jpg", CV_LOAD_IMAGE_GRAYSCALE);

//用於儲存處理結果
Mat result = Mat::zeros(inputImage.rows, inputImage.cols, inputImage.type());

morphologyEx(inputImage, result, MORPH_GRADIENT, Mat(), Point(-1,-1), 2);
```



形態學-Top Hat-OpenCV範例程式(C++)

- 片段程式碼：

```
//讀取測試影像
Mat inputImage = imread("pic2.jpg", CV_LOAD_IMAGE_GRAYSCALE);

//用於儲存處理結果
Mat result = Mat::zeros(inputImage.rows, inputImage.cols, inputImage.type());

morphologyEx(inputImage, result, MORPH_TOPHAT, Mat(), Point(-1,-1), 2);
```

形態學-Black Hat-OpenCV範例程式(C++)

- 片段程式碼：

```
//讀取測試影像
Mat inputImage = imread("pic2.jpg", CV_LOAD_IMAGE_GRAYSCALE);

//用於儲存處理結果
Mat result = Mat::zeros(inputImage.rows, inputImage.cols, inputImage.type());

morphologyEx(inputImage, result, MORPH_BLACKHAT, Mat(), Point(-1,-1), 2);
```

形態學-OpenCV(Python)

```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()
```

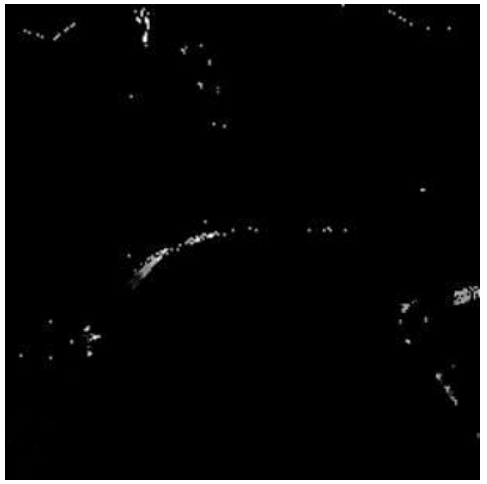
- 範例程式：
  - [5]決定一個5x5的kernel。
  - [7, 13]使用5x5的kernel進行Opening運算，迭代3次。
  - [8, 14]使用5x5的kernel進行Closing運算 ，迭代3次。
  - [9, 15]使用5x5的kernel進行Gradient運算 ，迭代3次。
  - [10, 16]使用5x5的kernel進行Top Hat運算 ，迭代3次。
  - [11, 17]使用5x5的kernel進行Black Hat運算 ，迭代3次。

形態學-OpenCV(Python)

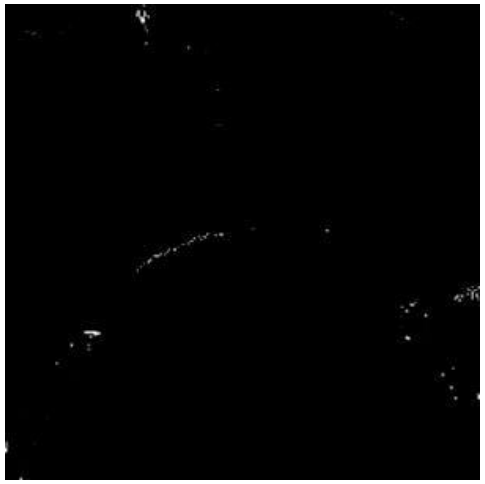
Opening運算

Closing運算

Gradient運算



Top Hat運算



Black Hat運算