
嵌入式智慧影像分析與實境界面

Fall 2018

INSTRUCTOR : YEN-LIN CHEN(陳彥霖), PH.D.
PROFESSOR
DEPT. COMPUTER SCIENCE AND INFORMATION
ENGINEERING
NATIONAL TAIPEI UNIVERSITY OF TECHNOLOGY

Lecture2-

嵌入式基礎影像處理技術開發

本課程討論群組

- 本課程課後討論FB社群：
- <https://www.facebook.com/groups/381460865722688/>

本課程作業/報告上傳

- 位址：北科i學園
- 作業上傳格式，以組別命名，壓縮為zip，例如：組別X.zip
- 其zip裡要包含如下資料夾
 - 報告 //存放報告 (PDF檔or WORD)
- 請依照此規則上傳，否則會造成助教改作業的困擾，成績有問題要自行負責
- 若有其他有關此作業的備註訊息要跟TA告知，可在根目錄附上文字檔(Readme.txt)，並留下你的E-Mail

大綱

- OpenCV
- OpenCV常用功能
- OpenCV Image Processing
- OpenCV Image Processing Denoise
- Qt with OpenCV
- TK1上執行Qt with 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，實現了圖像處理和電腦視覺方面的很多通用演算法。
- 當前最新穩定版本為3.4.3
- 更多關於OpenCV <https://opencv.org/>

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。
- 最新的OpenCV已經加入深度學習的模組 (OpenCV 3.3)。

OpenCV 常用功能

OpenCV - 常用語法(Mat)

- 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)

- 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

- https://docs.opencv.org/2.4/modules/core/doc/basic_structures.html?highlight=mat

OpenCV - 常用語法(Mat)

- 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)

- 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)

- 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)(1)

- 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 : 水平縮放比率，當輸入為0，`fx=(double)dsize.width/src.cols`。
 - fy : 垂直縮放比率，當輸入為0時，`fy=(double)dsize.height/src.rows`。
 - interpolation : 插值方式。

OpenCV - 常用語法(resize)(2)

- interpolation：插值方式
- 當影像改變大小時，要將像素填入新的影像。
 - 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，4×4大小的補點。
 - CV_INTER_LANCZOS4：Lanczos插值，8×8大小的補點。
- [https://docs.opencv.org/2.4/modules/imgproc/doc/geometric_transformations.html#void%20resize\(InputArray%20src,%20OutputArray%20dst,%20Size%20dsize,%20double%20fx,%20double%20fy,%20int%20interpolation\)](https://docs.opencv.org/2.4/modules/imgproc/doc/geometric_transformations.html#void%20resize(InputArray%20src,%20OutputArray%20dst,%20Size%20dsize,%20double%20fx,%20double%20fy,%20int%20interpolation))

OpenCV - 常用語法(waitKey)

- `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線程之間具有最小時間，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運行的內容。
- 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。
- http://docs.opencv.org/2.4/modules/highgui/doc/user_interface.html?highlight=waitkey#waitkey

OpenCV - 常用語法(讀檔寫檔)

- 讀取圖片
 - `imread(<路徑>, <flag>);`
 - `flags:`
 - `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 - 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

- Mat內存為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。
- 範例：`cvtColor(src, dst, CV_BGR2GRAY) //彩色轉灰階`

二值化 - 簡介

- 二值化是圖像分割的一種方法，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)，會依據某個強度(閾值(threshold))當作分割的標準，通常會以強度超過閾值的像素當作前景，反之則為背景。
- 閾值的算法主要分兩類：
 - 固定閾值：直接給定一個灰階值當閾值，再用這個閾值進行二值化。
 - 自適應閾值：會依據輸入影像計算出較合適的閾值，再用這個閾值進行二值化(在Otsu介紹)。

二值化 - 利用臨界進行值影像分割

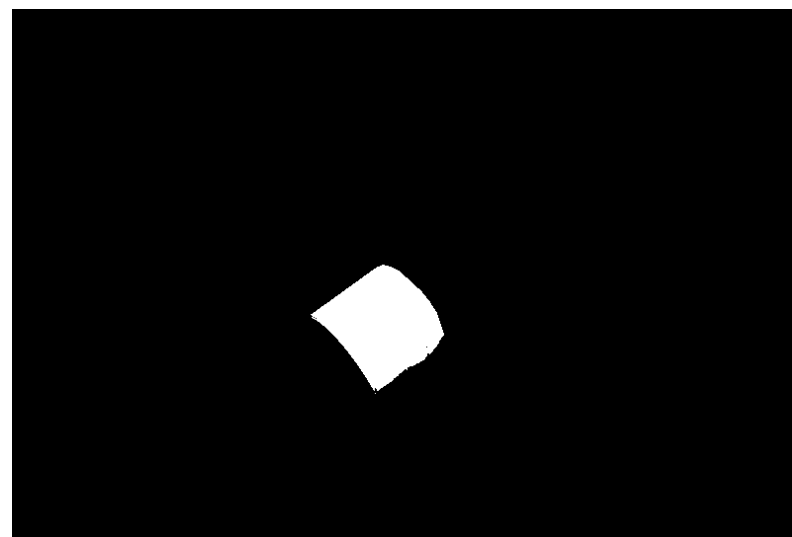
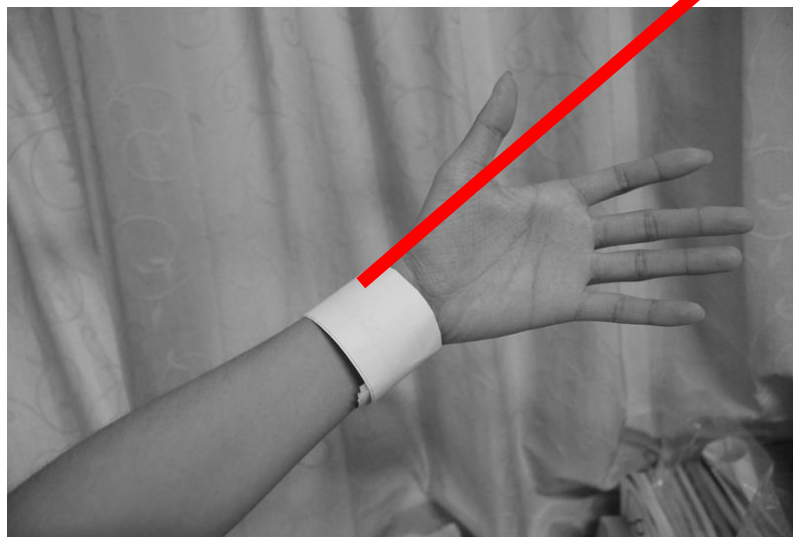
- 在一灰階影像當中，每一獨立區塊的產生取決於亮度（灰階值）的不同，利用這樣的特性，我們可以設立一固定的臨界值T，以該值為條件，將影像轉換為二值影像（只包含0及1之二值影像，或0及255之灰階影像），並從中加以擷取出感興趣的影像區塊。

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

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

二值化 - 利用臨界值進行影像分割

≥ 170



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

二值化 - OpenCV

- OpenCV固定閾值二值化
- `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：閾值。
 - 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。
 - https://docs.opencv.org/2.4.13/modules/imgproc/doc/miscellaneous_transformations.html?highlight=otsu

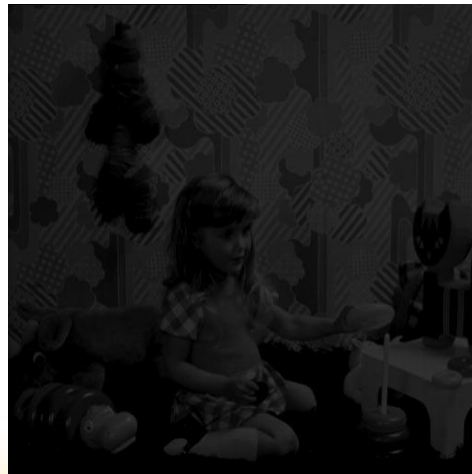
二值化 - OpenCV

- 五種Type介紹：
- THRESH_BINARY：大於閾值的像素設為最大值(maxval)，小於閾值的設為0。
- THRESH_BINARY_INV：大於閾值的像素設為0，小於閾值的設為最大值(maxval)。
- THRESH_TRUNC：大於閾值的像素設為閾值，小於閾值的設為0。
- THRESH_TOZERO：大於閾值的像素值不變，小於閾值的設為0。
- THRESH_TOZERO_INV：大於閾值的像素值設為0，小於閾值的不變。

影像強化

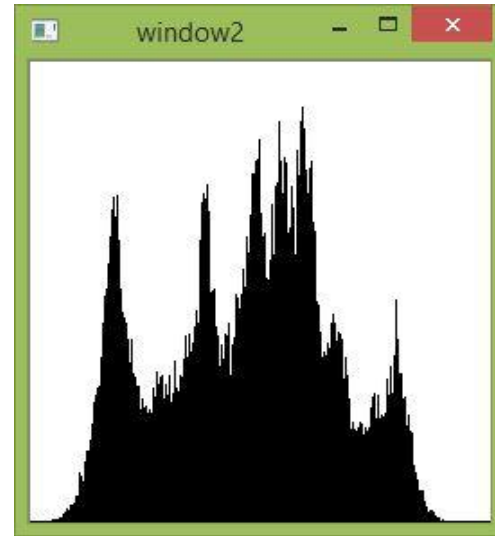
- 清晰的影像，是指能夠清楚反映被攝景物的明亮程度和細微色彩差別的影像。
- 當拍攝黑夜中的動物或草叢中的螞蟻等影像時，因色彩近似或亮度不夠以致溶入背景之中等情況下，影像就很難看清。對於這種影像，如果把動物或螞蟻的色彩和亮度作適當處理，使其與背景產生微秒的差別，就可以使動物或小蟲的英姿從背景中浮現突出，從而變的清晰起來。
- 像這樣通過增強亮度或色彩等各種包含於影像中的資訊，或者轉換成其他資訊，就可以製作成清晰影像，這種處理過程稱為影像增強。

影像強化



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，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。



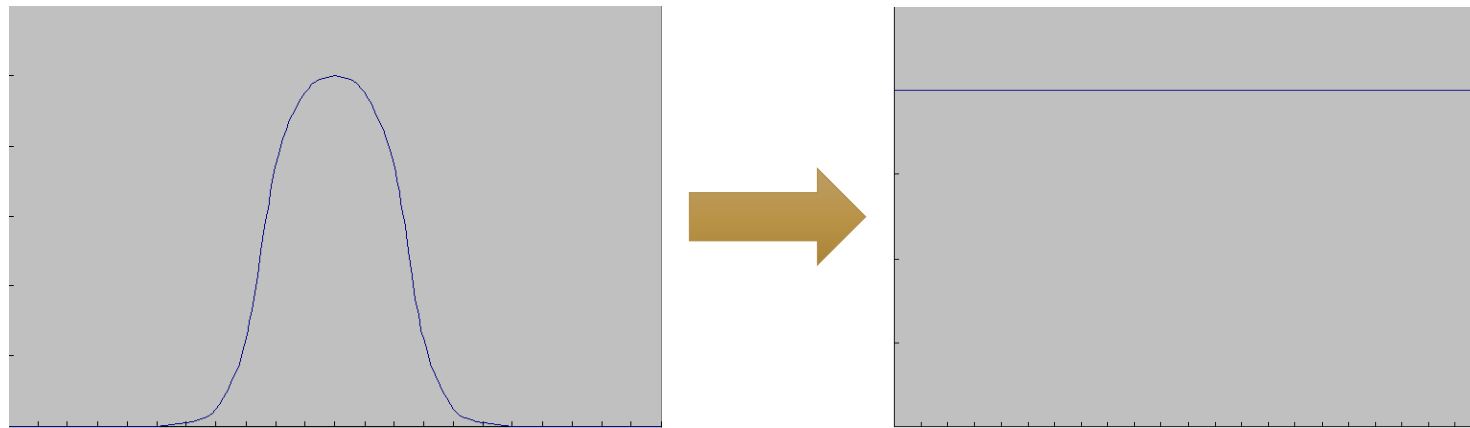
- 直方圖是影像的一個重要特性，可以透過直方圖觀察影像中像素值的變化，看出影像是否太暗或過曝，又或者分布太過集中。

Histogram - OpenCV

- 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。

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。

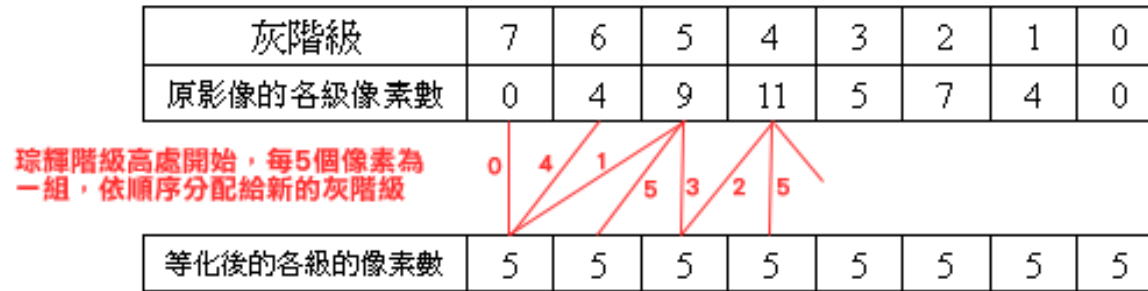


Histogram - 直方圖等化(equalizeHist)流程

- 計算輸入圖的直方圖。
- 將直方圖歸一到所有bin的總合為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 $40 \div 8 = 5$ 。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。



- 而從灰階為5處的像素中，選擇出1個像素的方法有兩種：

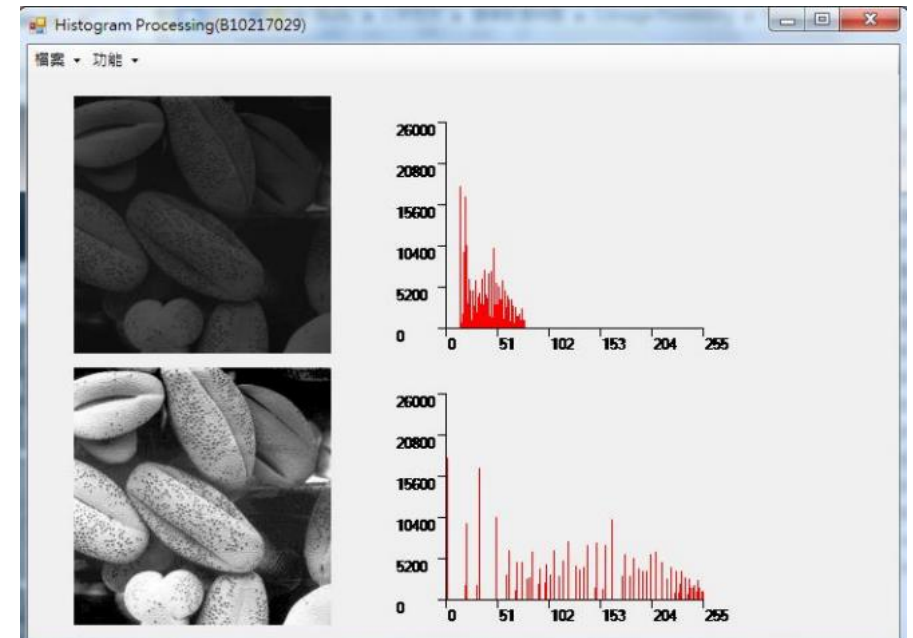
(1)隨機選擇。

(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

- 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。

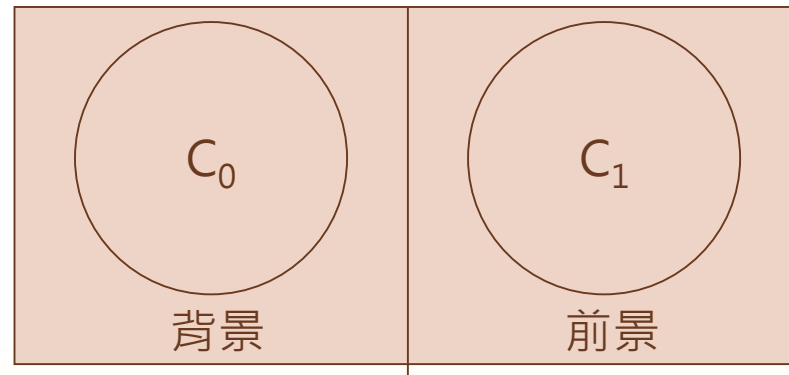


Otsu(自適應閾值)流程

- 先計算影像的直方圖。
- 把直方圖強度大於閾值的像素分成一組，把小於閾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依序當作閾值來計算組內變異數和，總和值最小的就是結果閾值。

Otsu(自適應閾值)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」之合為最小



Otsu(自適應閾值)門檻值決定法(1-2)

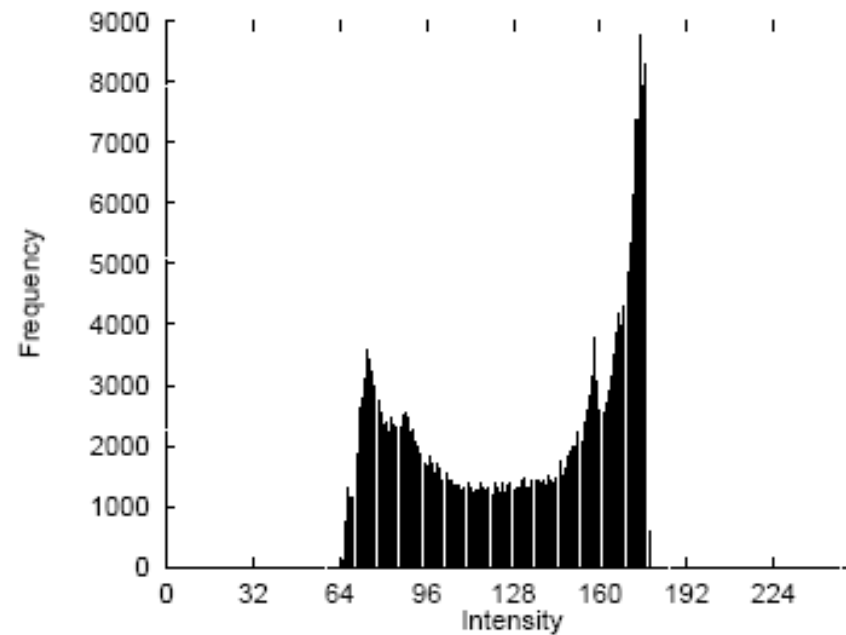


Figure 6.3: A bi-modal histogram.

Otsu(自適應閾值)門檻值決定法(1-3)

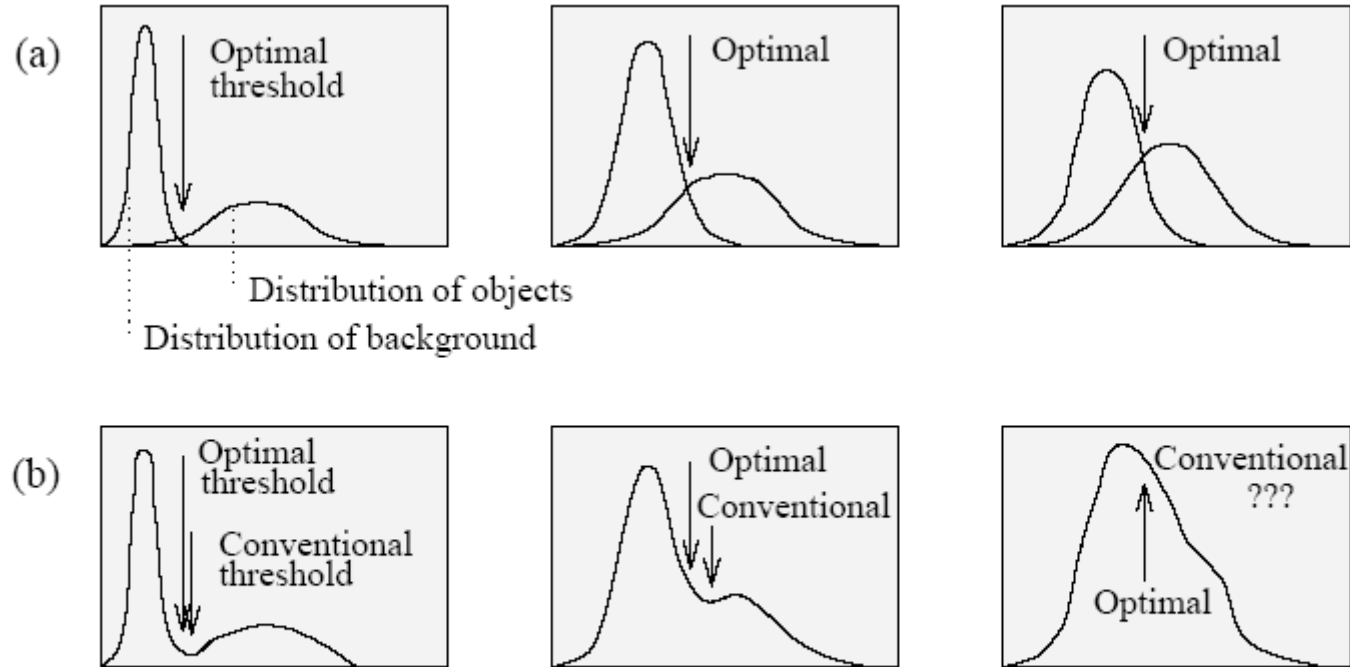


Figure 6.4: Gray-level histograms approximated by two normal distributions—the threshold is set to give minimum probability of segmentation error. (a) Probability distributions of background and objects. (b) Corresponding histograms and optimal threshold.

Otsu(自適應閾值)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(自適應閾值)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(自適應閾值)門檻值決定法(4)

- 利用 u_1 及 u_2 ，進一步算出 C_0 及 C_1 的變異數為

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

- 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

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

- 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(自適應閾值)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
 $\sigma_W^2 + \sigma_B^2 = \sigma_{T^*}^2$ 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 $L-1$ 之間，一個一個將灰階值代入 σ_B^2 式子內，等全部 L 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(自適應閾值)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：閾值。
 - 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

濾波簡介

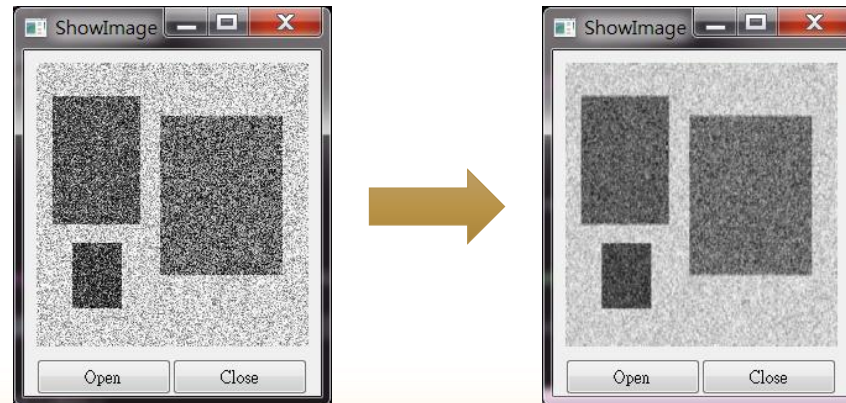
- 影像在傳輸的過程中，由於受到通道、劣質取樣系統、雜訊等其他的干擾影響，導致影像變得不清晰，因此我們需要對影像進行濾波。
- 雜訊產生的原因決定了雜訊與影像訊號的關係。而減少雜訊的方法可分為兩種：一種是在空間域做處理；另一種則是在頻率域上做處理。
- 在執行影像濾波時，需要以一定的細節模糊做為代價，因此要如何濾除影像的雜訊，又可以保持影像的細節是一個重要的課題。

濾波器常見種類

- 均值濾波器
- 中值濾波器
- 高斯濾波器
- 雙邊濾波器

均值濾波

- 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

- 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。

p0	p1	p2
p3	p4	p5
p6	p7	p9

(a) 輸入影像的像素數組

輸出關注像素的8
近鄰點的平均值



	q	

(b) 輸出影像的像素數組

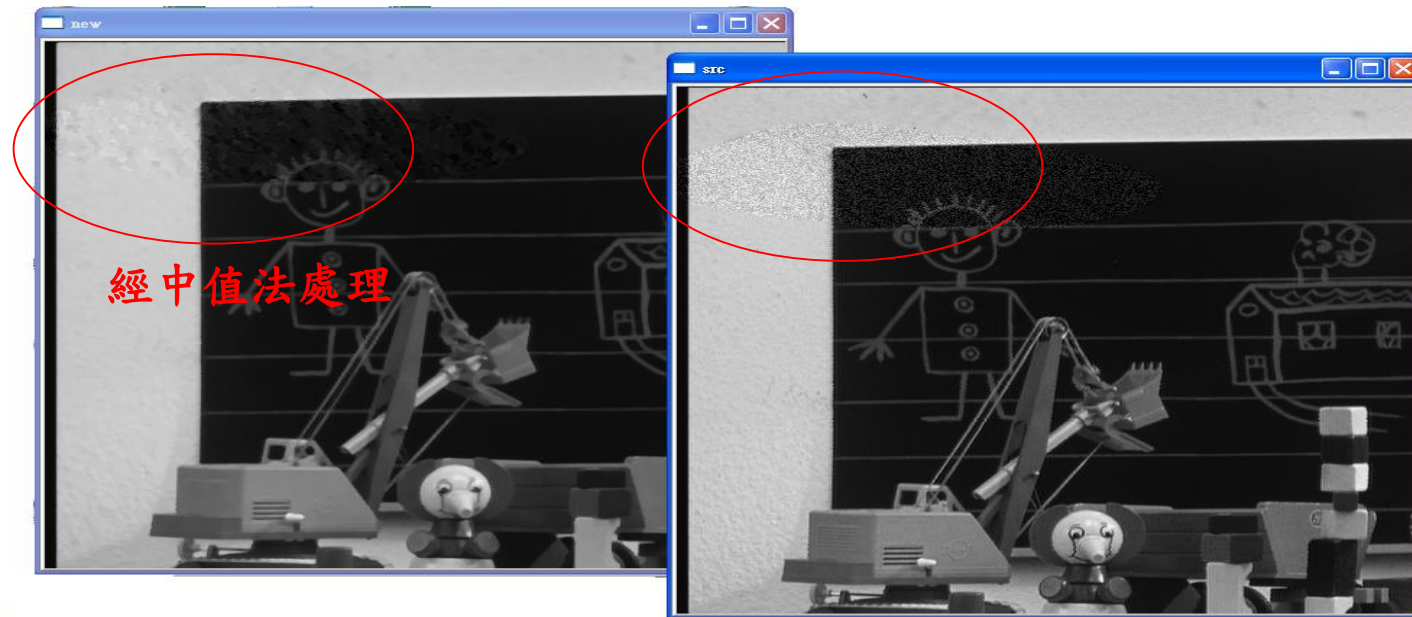
$$q = \frac{p0+p1+p2+p3+p4+p5+p6+p7+p8}{9}$$

均值濾波 - OpenCV

- 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，邊界模式用來推斷圖像外的像素

中值濾波

- 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

- 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。

4	4	3
2	10	3
5	2	4

(a) 輸入影像的像素數值

從關鍵像素的8近鄰
中選擇出中間值



	4	

(b) 輸出影像的像素值

2 2 3 3 4 4 4 5 10

中值濾波

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



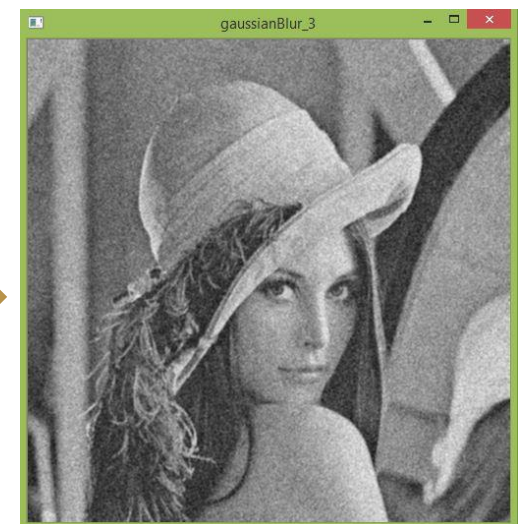
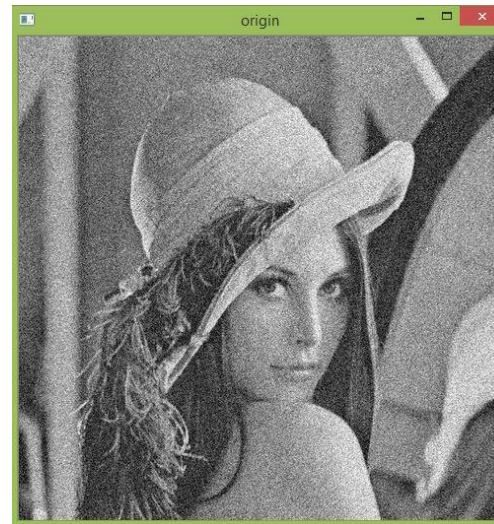
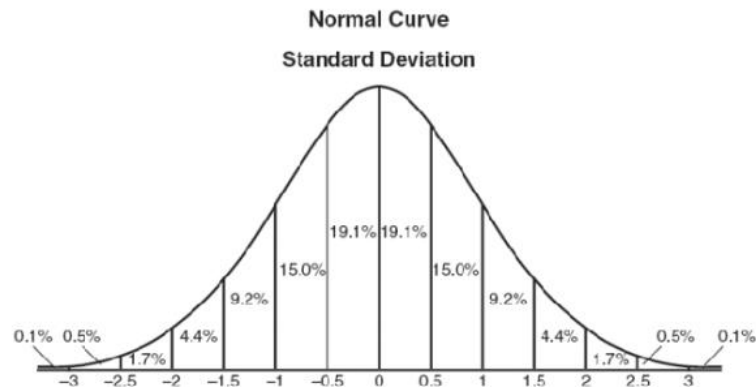
使用中值法處理之結果

中值濾波 - OpenCV

- OpenCV medianBlur()函式:
- **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。

高斯濾波

- 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

- 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

- 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

- 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

- 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：

100×0.1018679	95×0.1154317	110×0.1018679
120×0.1154317	90×0.1309013	100×0.1154317
110×0.1018679	115×0.1154317	120×0.1018679

高斯濾波

- 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 :

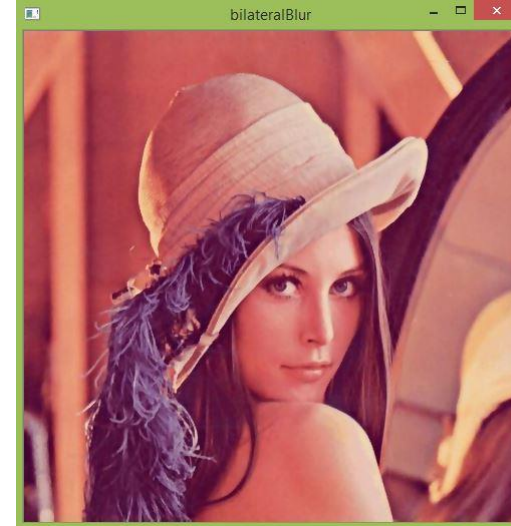
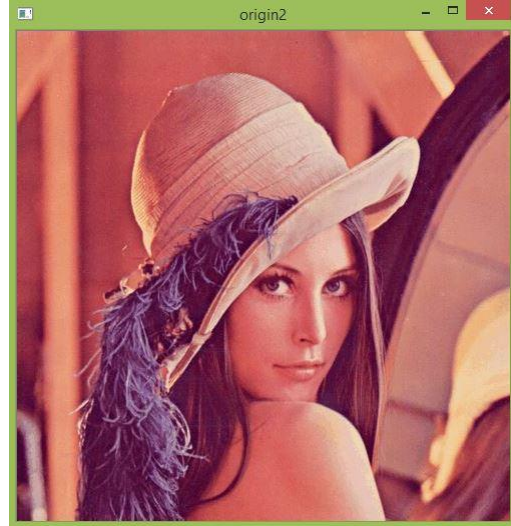
1/16	2/16	1/16
2/16	4/16	2/16
1/16	2/16	1/16

高斯濾波 - OpenCV

- OpenCV GaussianBlur()函式:
- `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。

雙邊濾波

- 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

- 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_s 為高斯濾波，加權根據距離。
- G_r 為高斯濾波，加權根據像素色差。
- W_p 為 G_s 和 G_r 相乘。

雙邊濾波

- 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

- 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

- 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

- 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

- 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

- 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

- 最後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。得到新值為(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

- 之後在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:
- `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。

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

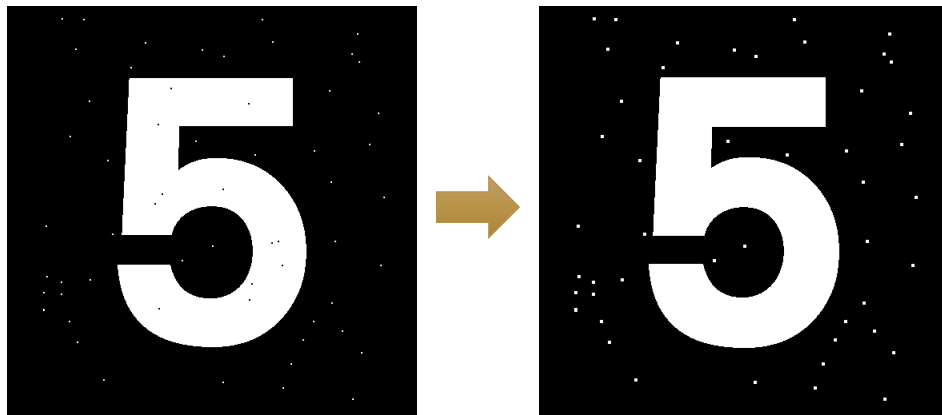
- 二值影像的雜訊，稱為椒鹽狀雜訊，它來自英語 salt and pepper noise 一詞的直譯。當然這種雜訊也能用中值濾波法將其除去，另外，利用它的二值性，有稱為膨脹、收縮的處理方法。
- 所謂膨脹(dilation)，是指某像素的近鄰中，若有一個為 1，則將該像素置為 1，其他的均置為 0。
- 所謂收縮(erosion)，是指某像素的近鄰中，若有一個為 0，就將該像素置為 0，而將其他均置為 1 的處理。
- Closing 運算：膨脹→收縮(除去黑色雜訊，白色雜訊依然殘留)
- Opening 運算：收縮→膨脹(除去白色雜訊，黑色雜訊依然殘留)

膨脹

- 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

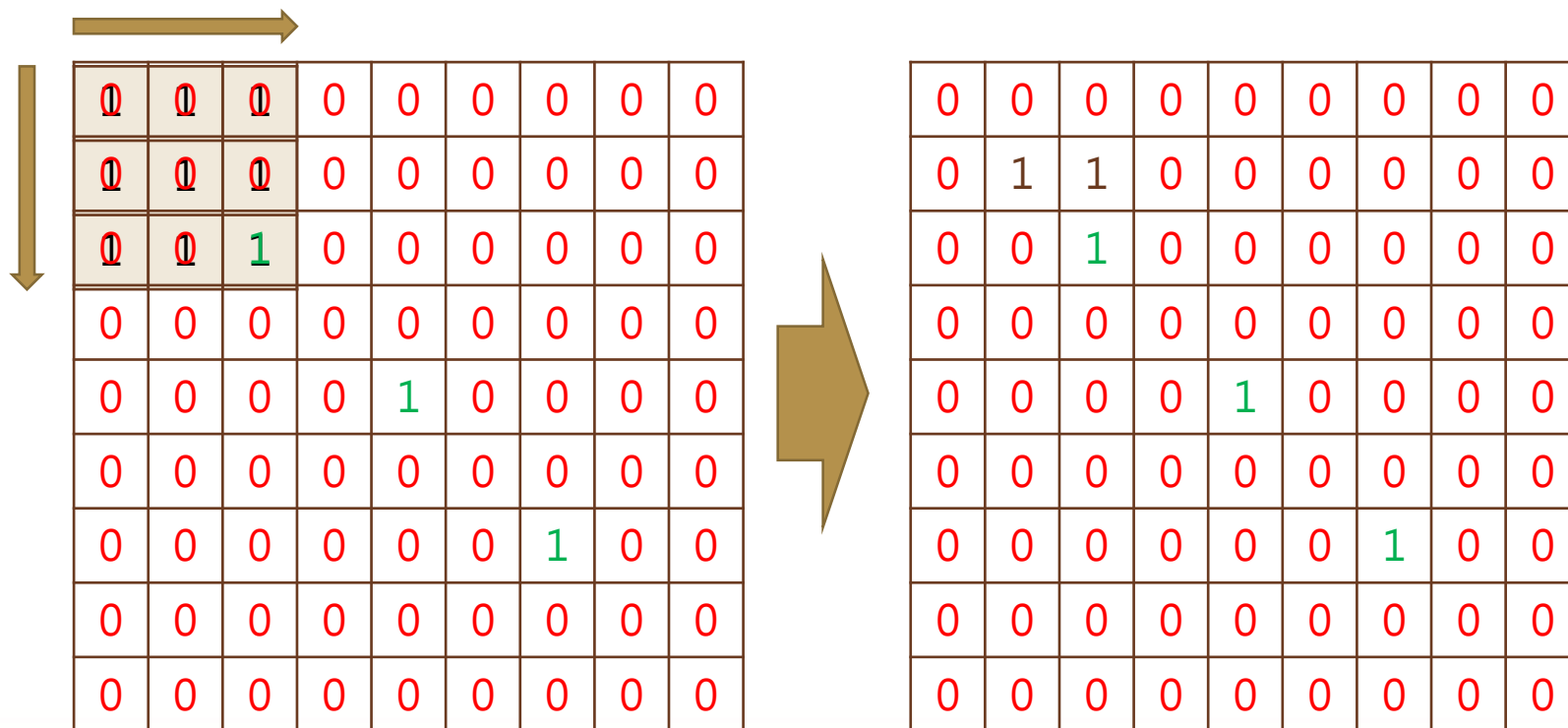
A

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

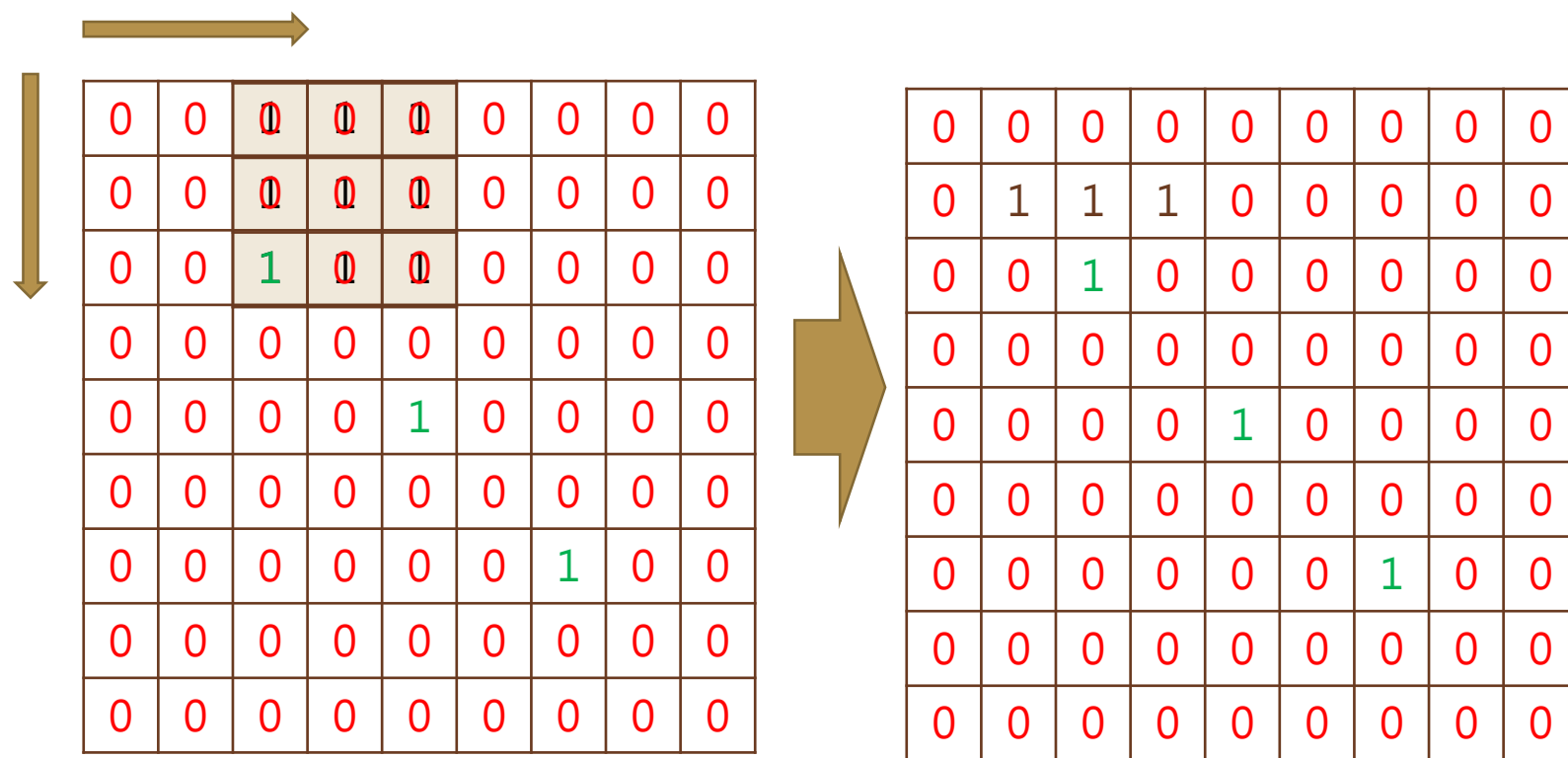
膨脹

- 綠色及紅色為原影像。
- 黑色為尋訪後。
- 接著繼續尋訪。



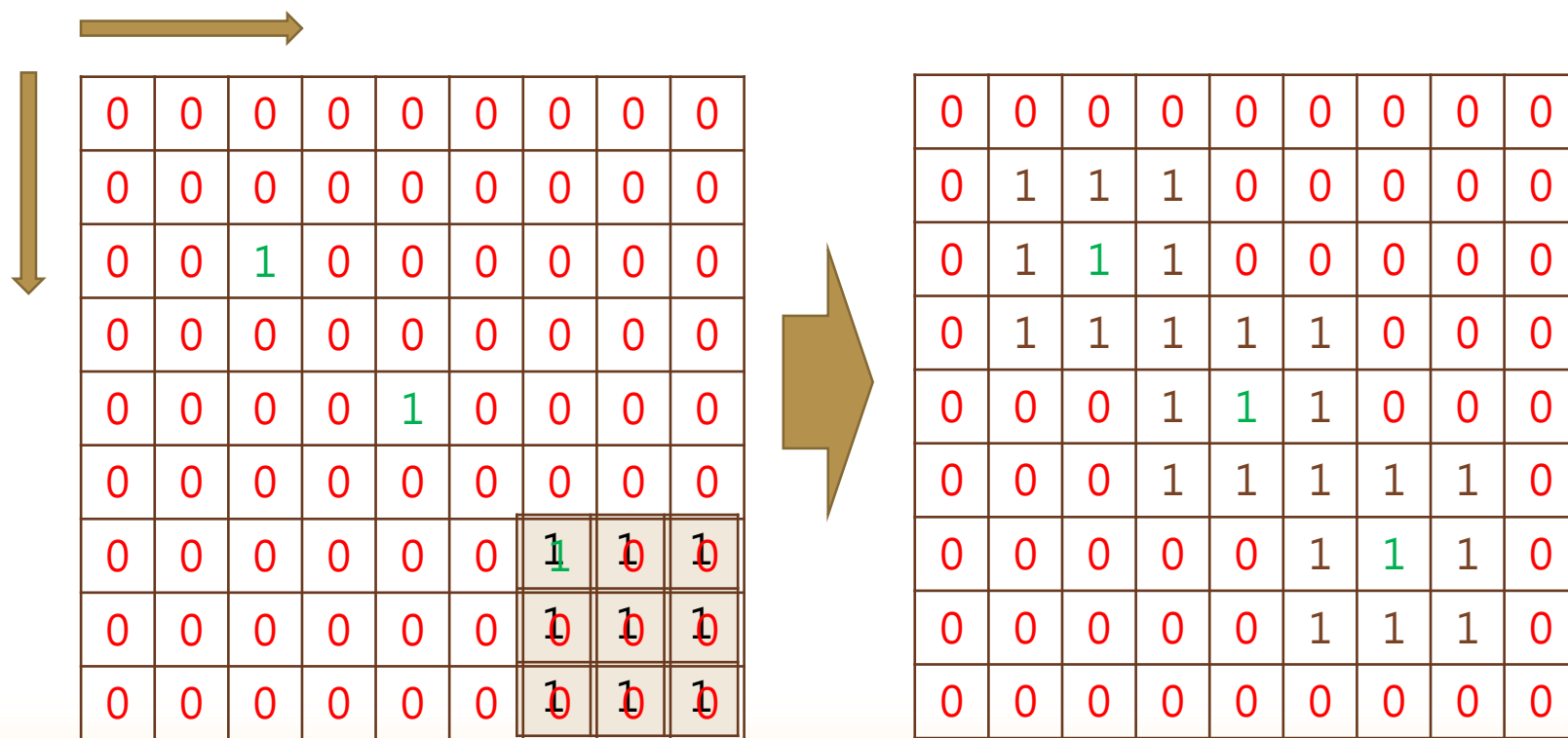
膨脹

- 綠色及紅色為原影像。
- 黑色為尋訪後。
- 接著繼續尋訪。



膨脹

- 最終尋訪結果。
- 綠色及紅色為原影像。
- 黑色為尋訪後。



膨脹 - OpenCV

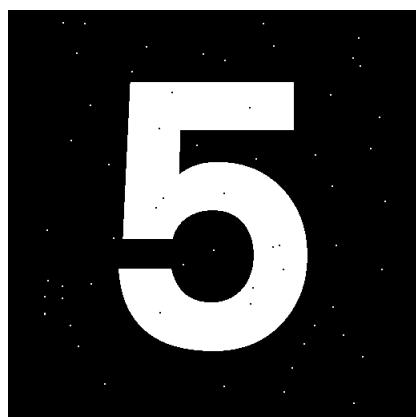
- OpenCV dilate()函式:
- `dilate(const Mat &src, Mat &dst, Mat kernel, Point anchor=Point(-1,-1), int iterations=1)`
 - src：輸入影像。
 - dst：輸出影像，和輸入圖尺寸、型態相同。
 - kernel：結構元素，如果kernel=Mat()則為預設的3×3矩形，越大膨脹效果越明顯。
 - anchor：原點位置，預設為結構元素的中央。
 - iterations：執行次數，執行越多次膨脹效果越明顯。

侵蝕

- 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

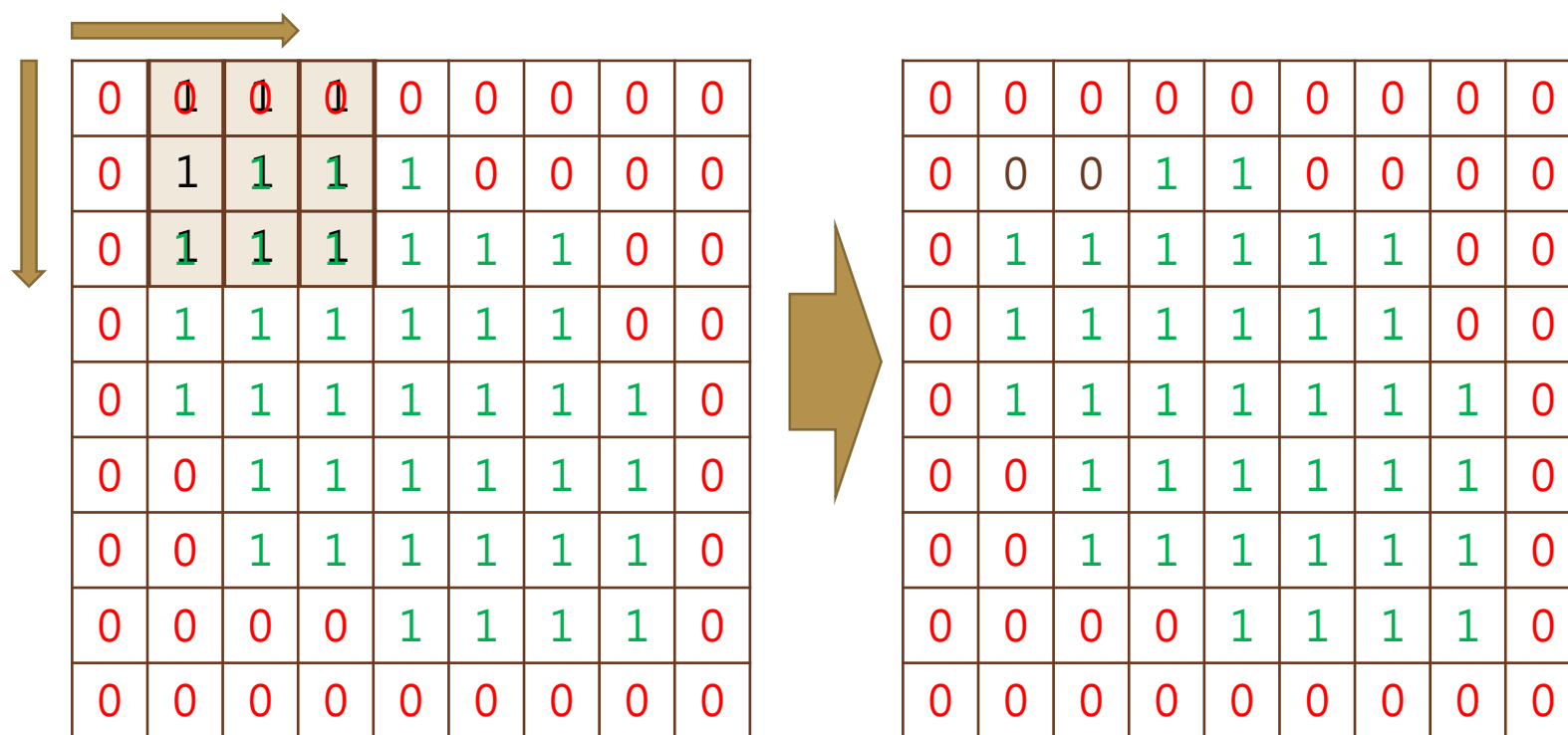
A

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

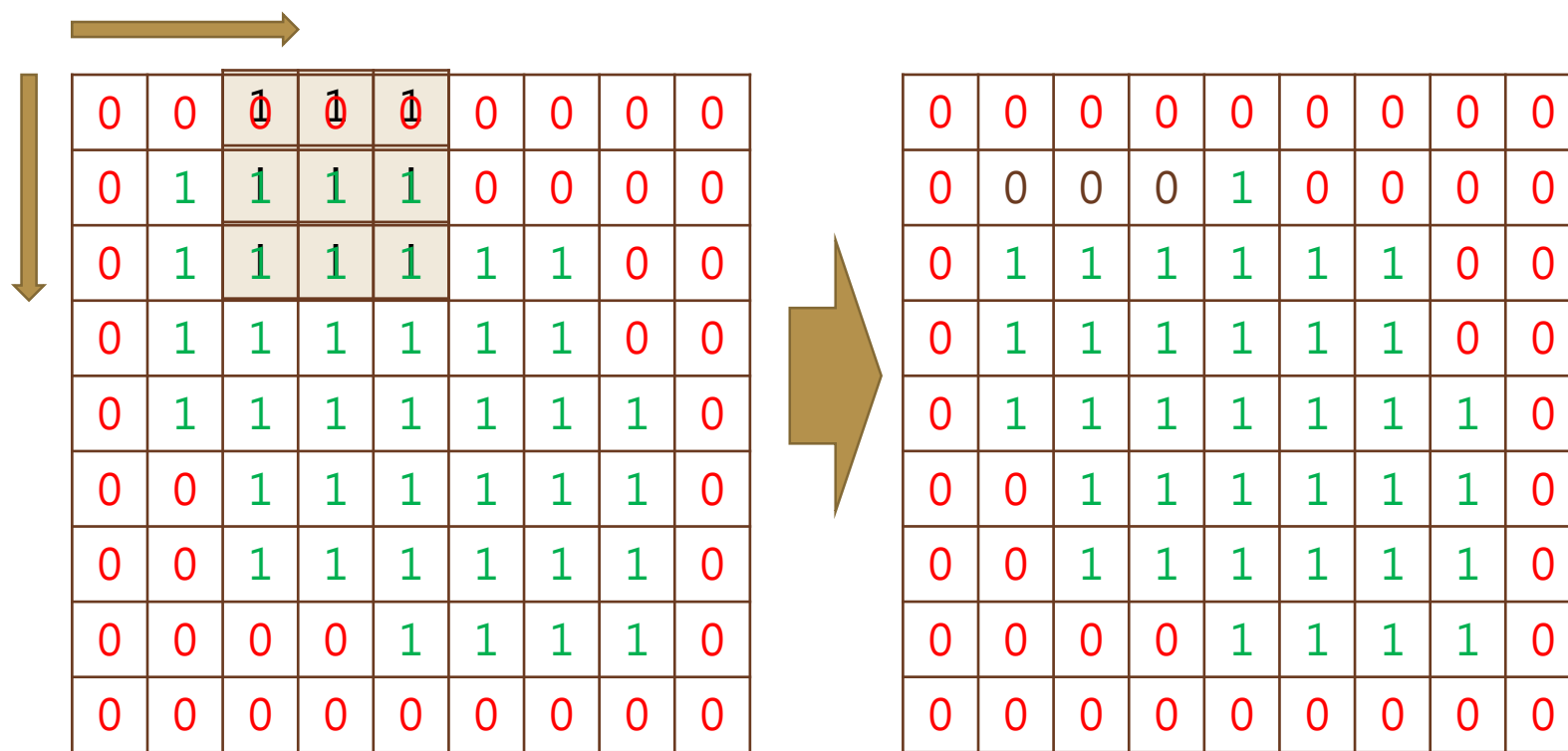
侵蝕

- 綠色及紅色為原影像。
- 黑色為尋訪後。
- 接著繼續尋訪。



侵蝕

- 綠色及紅色為原影像。
- 黑色為尋訪後。
- 接著繼續尋訪。



侵蝕 - OpenCV

- 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3×3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。

形態學

- 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圖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。

形態學

- 有以下幾個種類其公式算法如下:

- OPEN

- 公式 $A \circ B = (A \ominus B) \oplus B$.

- $\text{dst} = \text{open}(\text{src}, \text{element}) = \text{dilate}(\text{erode}(\text{src}, \text{element}))$

- 去除小雜訊

- CLOSE

- 公式 $A \bullet B = (A \oplus B) \ominus B$.

- $\text{dst} = \text{close}(\text{src}, \text{element}) = \text{erode}(\text{dilate}(\text{src}, \text{element}))$

- 去除小洞

- GRADIENT

- 公式 $A \oplus B - A \ominus B$

- $\text{dst} = \text{morph}(\text{src}, \text{element}) = \text{dilate}(\text{src}, \text{element}) - \text{erode}(\text{src}, \text{element})$

- 找輪廓

形態學

- TOPHAT

- 公式 $T_w(f) = f - f \circ b$.
- $\text{dst} = \text{tophat}(\text{src}, \text{element}) = \text{src} - \text{open}(\text{src}, \text{element})$
- 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，圖像增強等。

- BLACKHAT

- 公式 $T_b(f) = f \bullet b - f$.
- $\text{dst} = \text{blackhat}(\text{src}, \text{element}) = \text{close}(\text{src}, \text{element}) - \text{src}$
- 型態學Close及輸入影像之間的差異。

形態學 - OpenCV

- 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

- op : 操作種類如下:

//Open

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

//Close

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

//Gradient

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

//Top Hat

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

//Black Hat

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

形態學 - Open – OpenCV範例程式

- 片段程式碼：

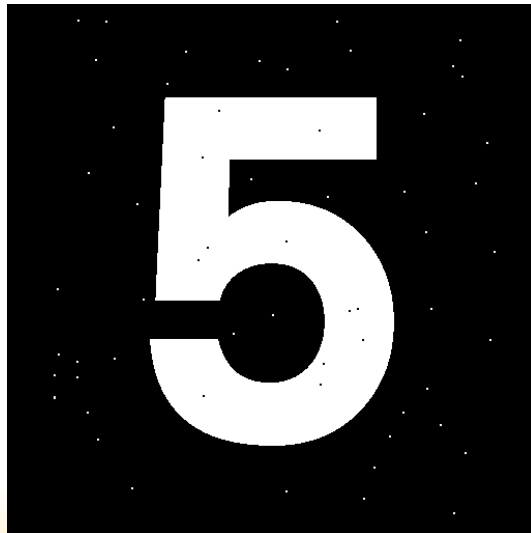
```
//讀取測試影像  
Mat inputImage = imread("pic2.jpg", CV_LOAD_IMAGE_GRAYSCALE);  
  
//用於儲存處理結果  
Mat result = Mat::zeros(inputImage.rows, inputImage.cols, inputImage.type());  
  
morphologyEx(inputImage, result, MORPH_OPEN, Mat(), Point(-1,-1), 2);
```



形態學 - Close – OpenCV範例程式

- 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

- 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

- 片段程式碼：

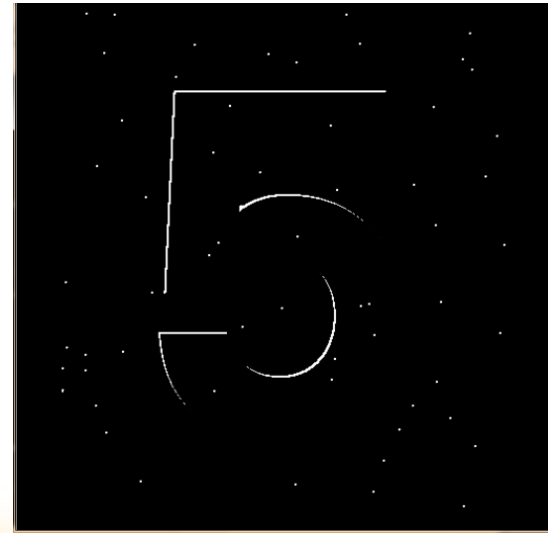
```
//讀取測試影像  
Mat inputImage = imread("pic2.jpg", CV_LOAD_IMAGE_GRAYSCALE);  
  
//用於儲存處理結果  
Mat result = Mat::zeros(inputImage.rows, inputImage.cols, inputImage.type());  
  
morphologyEx(inputImage, result, MORPH_TOPHAT, Mat(), Point(-1,-1), 2);
```



形態學 – Black Hat OpenCV範例程式

- 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);
```



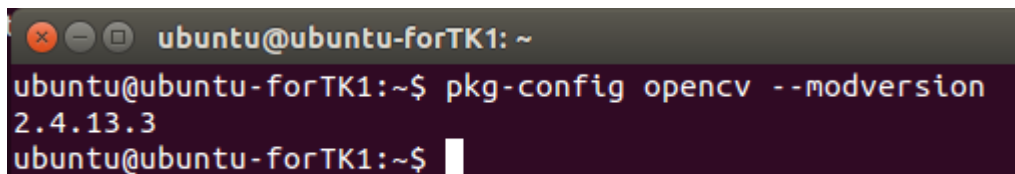
Qt with OpenCV part 1

Qt with OpenCV

- OpenCV安裝在系統中的路徑在每個平台可能不一樣，於課程的Ubuntu虛擬機我們是安裝在 `/usr/local/` 下的include與lib裡面，在TK1上預設是安裝在 `/usr/` 下的include與lib中，學生在不同平台或環境操作時應注意OpenCV安裝位置在哪裡!

Qt with OpenCV

- 請先輸入 `pkg-config opencv --modversion` 查看opencv版本



```
ubuntu@ubuntu-forTK1: ~  
ubuntu@ubuntu-forTK1:~$ pkg-config opencv --modversion  
2.4.13.3  
ubuntu@ubuntu-forTK1:~$
```

Qt with OpenCV (查看OpenCV Library)

- 輸入 `ls /usr/local/lib/*opencv*` (虛擬機上)
- 輸入 `ls /usr/lib/*opencv*` (TK1上)

```
ubuntu@ubuntu-forTK1: ~  
ubuntu@ubuntu-forTK1:~$ ls /usr/local/lib/*opencv*  
/usr/local/lib/libopencv_calib3d.so  
/usr/local/lib/libopencv_calib3d.so.2.4  
/usr/local/lib/libopencv_calib3d.so.2.4.13  
/usr/local/lib/libopencv_contrib.so  
/usr/local/lib/libopencv_contrib.so.2.4  
/usr/local/lib/libopencv_contrib.so.2.4.13  
/usr/local/lib/libopencv_core.so  
/usr/local/lib/libopencv_core.so.2.4  
/usr/local/lib/libopencv_core.so.2.4.13  
/usr/local/lib/libopencv_features2d.so
```


Qt with OpenCV (QT使用OpenCV)

- 在Qt Creator專案的.pro檔案，新增OpenCV 路徑及使用的Library。

```
LIBS += /usr/local/lib/libopencv_imgproc.so \
        /usr/local/lib/libopencv_highgui.so \
        /usr/local/lib/libopencv_core.so \
        /usr/local/lib/libopencv_calib3d.so \
        /usr/local/lib/libopencv_ml.so \
        /usr/local/lib/libopencv_contrib.so \
        /usr/local/lib/libopencv_photo.so \
        /usr/local/lib/libopencv_legacy.so \
        /usr/local/lib/libopencv_stitching.so \
        /usr/local/lib/libopencv_features2d.so \
        /usr/local/lib/libopencv_flann.so \
        /usr/local/lib/libopencv_objdetect.so \
        /usr/local/lib/libopencv_gpu.so \
        /usr/local/lib/libopencv_video.so \
        /usr/local/lib/libopencv_videostab.so \
        /usr/local/lib/libopencv_superres.so
```

```
QT      += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = testOpenCVWithQt
TEMPLATE = app

SOURCES += main.cpp\
           mainwindow.cpp

HEADERS  += mainwindow.h

FORMS    += mainwindow.ui

INCLUDEPATH += /usr/include

LIBS += /usr/local/lib/libopencv_imgproc.so \
        /usr/local/lib/libopencv_highgui.so \
        /usr/local/lib/libopencv_core.so \
        /usr/local/lib/libopencv_calib3d.so \
        /usr/local/lib/libopencv_ml.so \
        /usr/local/lib/libopencv_contrib.so \
        /usr/local/lib/libopencv_photo.so \
        /usr/local/lib/libopencv_legacy.so \
        /usr/local/lib/libopencv_stitching.so \
        /usr/local/lib/libopencv_features2d.so \
        /usr/local/lib/libopencv_flann.so \
        /usr/local/lib/libopencv_objdetect.so \
        /usr/local/lib/libopencv_gpu.so \
        /usr/local/lib/libopencv_video.so \
        /usr/local/lib/libopencv_videostab.so \
        /usr/local/lib/libopencv_superres.so
```

←此為opencv include檔的目錄位置

←此為需要用到的opencv library檔案
(Linux下share library檔名為.so結尾)

*在此我們把所有的OpenCV library都連結上來，
當需要用到時不需要另外回來link

Qt Creator 上使用OpenCV– 基本環境配置

- 接著在程式上include所需要用到的標頭檔即可，以下是常用include的標頭檔

- #include "opencv2/core/core.hpp"
- #include "opencv2/highgui/highgui.hpp"

```
#include "opencv2/core/core.hpp"  
#include "opencv2/highgui/highgui.hpp"
```

Qt Creator 上使用OpenCV – HelloWorld

- 當我們配置好環境後我們可以寫個簡單的HelloWorld來測試環境是否設定正確，此簡單範例為當Qt程式執行後，直接透過OpenCV讀取一張圖片(Lena.jpg)然後直接用OpenCV的window將圖片顯示出來

使用opencv的物件及function
(可使用using namespace cv
那就不用再前面加上cv:: 但不推薦)



```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "opencv2/core/core.hpp"
#include "opencv2/highgui/highgui.hpp" ←include OpenCV (常用的建議放在標頭檔)

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    //declare a mat to open an image
    cv::Mat image = cv::imread("Lena.jpg"); ←讀取圖片並顯示(路徑記得自己改)

    //show image
    cv::imshow("Hello world !!", image);

    //delay
    //cv::waitKey(0); ←由於顯示問題(一般會加入一個delay)

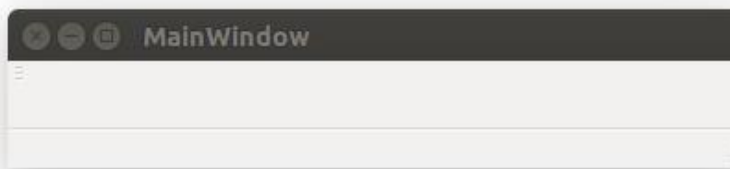
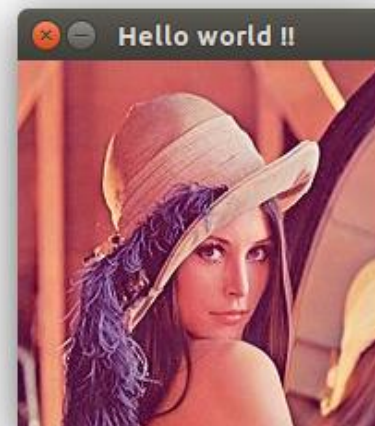
    ui->setupUi(this);
}

MainWindow::~MainWindow()
{
    delete ui;
}
```

Qt Creator 上使用OpenCV – HelloWorld

- 執行結果
- 此範例有放到TK1上，
資料夾名稱為
testOpenCVWithQt

```
#include "mainwindow.h"  
#include "ui_mainwindow.h"  
#include "opencv2/core/core.hpp"  
#include "opencv2/highgui/highgui.hpp"  
  
MainWindow::MainWindow(QWidget *parent) :  
    QMainWindow(parent),  
    ui(new Ui::MainWindow)  
{  
    //declare a mat to open an image  
    cv::Mat image = cv::imread("Lena.jpg");  
  
    //show image  
    cv::imshow("Hello world !!", image);  
  
    //delay  
    //cv::waitKey(0);  
  
    ui->setupUi(this);  
}  
  
MainWindow::~MainWindow()  
{  
    delete ui;  
}
```



Qt with OpenCV part 2

Qt Creator 上使用OpenCV

- 程式流程

按下 Open 按鈕



按下 Close 按鈕

Qt Creator 上使用OpenCV

- Mat 轉 QImage 說明：

- 在建立Qimage影像時，有下列方式：

`QImage ()`

`QImage (const QSize & size, Format format)`

`QImage (int width, int height, Format format)`

`QImage (uchar * data, int width, int height, Format format)`

`QImage (const uchar * data, int width, int height, Format format)`

`QImage (uchar * data, int width, int height, int bytesPerLine, Format format)`

`QImage (const uchar * data, int width, int height, int bytesPerLine, Format format)`

`QImage (const char * const[] xpm)`

`QImage (const QString & fileName, const char * format = 0)`

`QImage (const char * fileName, const char * format = 0)`

`QImage (const QImage & image)`

Qt Creator 上使用OpenCV

- 在Mat與QImage之間格式轉換，主要部份為：
 - unsigned char : 主要影像內容
 - width : 影像寬
 - height : 影像高
 - widthStep : 每一列影像像素的Bytes總數
 - 影像格式及位元深度
- 在OpenCV，以彩色影像為例：每一個Pixel其格式為BGR，而QImage為RGB，故在轉換時必須透過
cvCvtColor進行R與B交換

Qt Creator 上使用OpenCV

- 在影像格式上，彩色影像使用RGB表示三種顏色，每種顏色有256階，共 $3 * 8 = 24$ bits則該影像為24位元深度，透過Mat所讀取的影像，其位元深度為24位元。

- 彩色Mat轉換成QImage影像格式時，使用

`Format::RGB888`

可於下列網址查詢QImage所支援的影像格式

<http://qt-project.org/doc/qt-5.0/qtgui/qimage.html#Format-enum>

- 灰色Mat轉換成QImage影像格式時，使用

`Format::Indexed8`

在QT上使用OpenCV範例程式

- 首先使用QT建立使用者介面。
 - 有開啟圖片和關閉程式的按鈕
- 使用imread讀取影像。
- 將Mat轉成QImage，接著輸出影像結果。
- Include

```
#include <QImage>
#include <QPixmap>
#include <opencv2/core/core.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
```

Qt Creator 上使用OpenCV

- 片段程式碼：

```
connect(ui->openPushButton,SIGNAL(clicked(bool)), this, SLOT(openImage()));  
connect(ui->closePushButton,SIGNAL(clicked(bool)), this, SLOT(closeApp()));
```

按下Open後
→ Open →

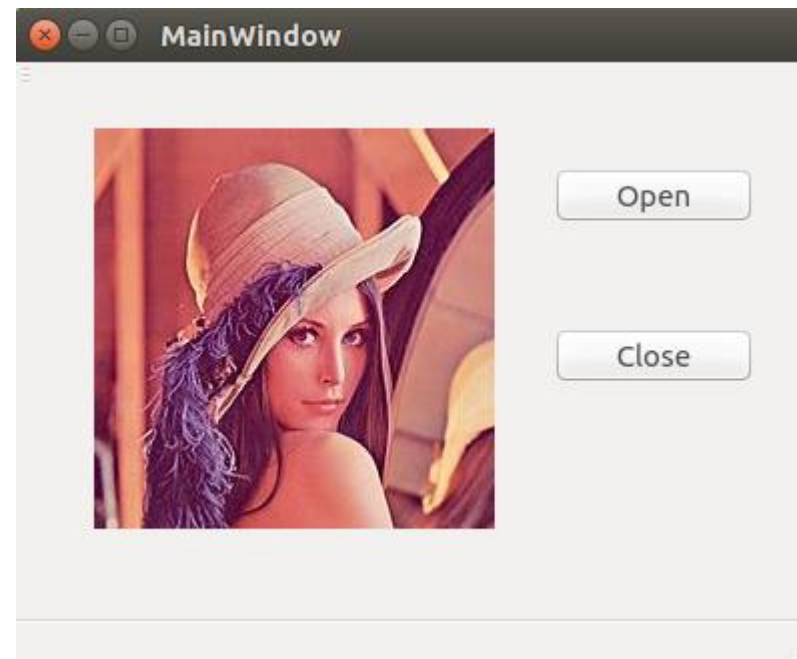
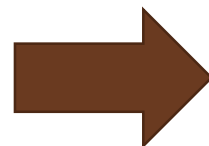
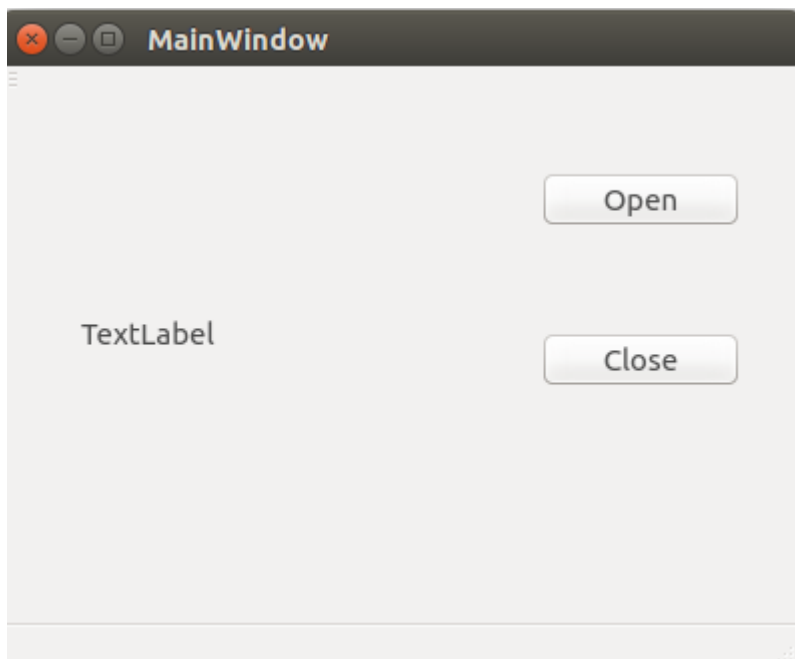
```
void MainWindow::openImage()  
{  
    cv::Mat image = cv::imread("Lena.jpg");  
    QImage myShowImage;  
    myShowImage = convertProcess(image);  
    ui->label->setPixmap(QPixmap::fromImage(myShowImage).scaled(this->ui->label->size()));  
}
```

其中convertProcess()

```
QImage MainWindow::convertProcess(cv::Mat image)  
{  
    if(image.type() == CV_8UC1)  
    {  
        return QImage((unsigned char *)image.data, image.cols, image.rows, image.step, QImage::Format_Indexed8);  
    }  
    else  
    {  
        cvtColor(image, image, CV_BGR2RGB);  
        return QImage((unsigned char *)image.data, image.cols, image.rows, image.step, QImage::Format_RGB888);  
    }  
}
```

Qt Creator + OpenCV

- 程式範例圖



TK1 上執行 Qt with OpenCV

更改Makefile

- 打開QT做完專案，在虛擬機端編譯完成後，在資料夾中會產生Makefile，將x86_64-linux-gnu 全部換成 arm-linux-gnueabihf。
- 將Makefile中OpenCV部分，usr/local/ 改成 usr/如下圖 (可全選直接取代)。

```
LIBS          = $(SUBLIBS) -L/usr/X11R6/lib64 /usr/lib/  
libopencv_imgproc.so /usr/lib/libopencv_highgui.so /usr/lib/  
libopencv_core.so /usr/lib/libopencv_calib3d.so /usr/lib/libopencv_ml.so /  
usr/lib/libopencv_contrib.so /usr/lib/libopencv_photo.so /usr/lib/  
libopencv_legacy.so /usr/lib/libopencv_stitching.so /usr/lib/  
libopencv_features2d.so /usr/lib/libopencv_flann.so /usr/lib/  
libopencv_objdetect.so /usr/lib/libopencv_gpu.so /usr/lib/  
libopencv_video.so /usr/lib/libopencv_videostab.so /usr/lib/  
libopencv_superres.so -l0t5Widoets -L/usr/lib/arm-linux-gnueabihf -l0t5Gui -
```


更改Makefile

- 將專案.pro檔INCLUDEPATH和LIBS改成下圖把原本的usr/local/改成usr/。
- 用filezilla傳到TK1上
- 用ssh連線
- 之後執行 make clean, qmake 及 make

```
ubuntu@tegra-ubuntu:~/testOpenCVWithQt$ ls
Lena.jpg      main.o          mainwindow.ui    testOpenCVWithQt.pro
Makefile      mainwindow.cpp moc_mainwindow.cpp testOpenCVWithQt.pro.user
Makefile~    mainwindow.h    moc_mainwindow.o ui_mainwindow.h
main.cpp      mainwindow.o    testOpenCVWithQt
```

- 就會看到執行檔

```
!
QT      += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widget

TARGET = testOpenCVWithQt
TEMPLATE = app

SOURCES += main.cpp\
           mainwindow.cpp

HEADERS  += mainwindow.h

FORMS    += mainwindow.ui

INCLUDEPATH += /usr/include

LIBS += /usr/lib/libopencv_imgproc.so \
        /usr/lib/libopencv_highgui.so \
        /usr/lib/libopencv_core.so \
        /usr/lib/libopencv_calib3d.so \
        /usr/lib/libopencv_ml.so \
        /usr/lib/libopencv_contrib.so \
        /usr/lib/libopencv_photo.so \
        /usr/lib/libopencv_legacy.so \
        /usr/lib/libopencv_stitching.so \
        /usr/lib/libopencv_features2d.so \
        /usr/lib/libopencv_flann.so \
        /usr/lib/libopencv_objdetect.so \
        /usr/lib/libopencv_gpu.so \
        /usr/lib/libopencv_video.so \
        /usr/lib/libopencv_videostab.so \
        /usr/lib/libopencv_superres.so
```