

OpenCV & python

- OpenCV 介紹
- python + OpenCV 安裝與環境設定
- 影像幾何轉換
- 影像比較度
- 影像增強
- 形態學
- 邊緣檢測
- 輪廓、重心
- 設定值切割

OpenCV & python

概述與環境建置

■ What is OpenCV?

OpenCV 概述

OpenCV 模組

OpenCV 應用

■ python + OpenCV 安裝與環境設定

Anaconda套件

Anaconda安裝

Conda語法

Opencv模組安裝

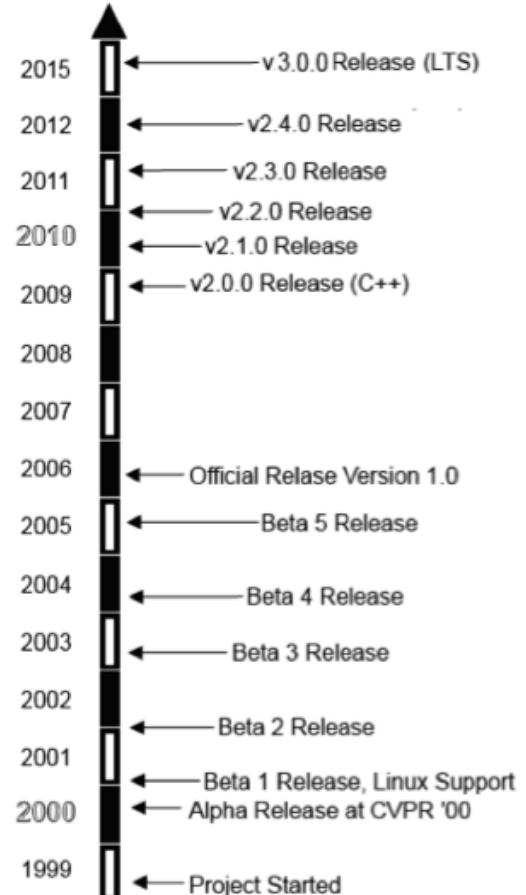
■ OpenCV

全名是Open Source Computer Vision Library，是一個影像處理函式庫，由Intel發起並參與開發，以BSD授權條款發行，可在商業和研究領域中免費使用，目前是非營利的基金組織OpenCV.org在維護

■ OpenCV-開放原始碼之電腦視覺

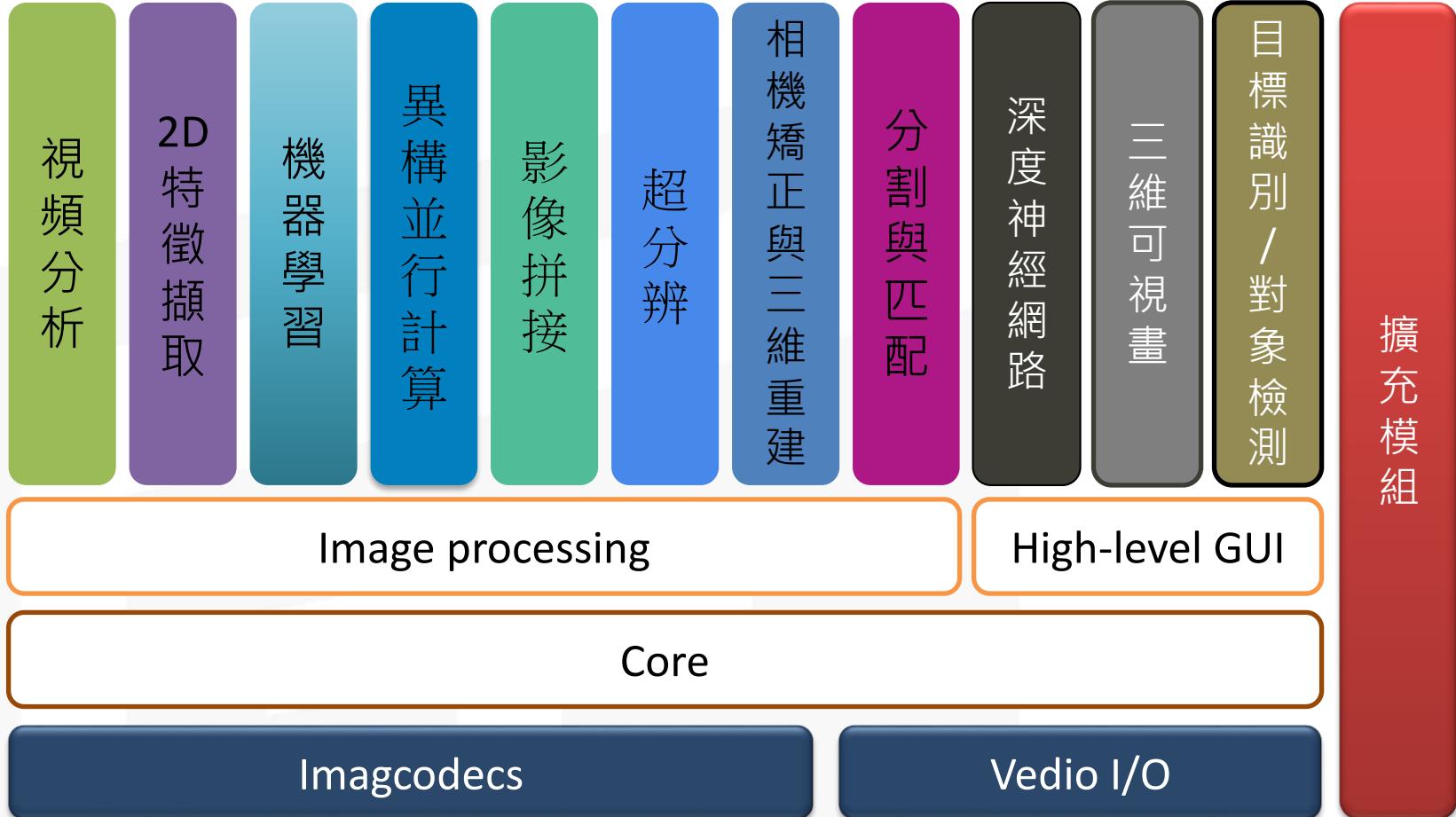
主要是針對在Real-time計算機視覺庫的一種編程功能。

■ Timeline



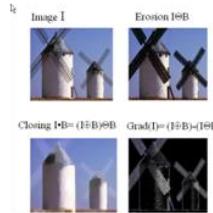
- 從英特爾1999年發展OpenCV以來，至今仍在持續更新中，動機起源於讓電腦視覺有更低的門檻，並充分利用 Intel 處理器的運算效能
- 目前最新的版本為3.0 alpha (Nov, 2014)
- 一開始原始碼是用C語言撰寫，目前支援以下語言：
C, C++、Python、Java、Matlab
- 跨平台: Windows, xNIX, MacOS, iOS, Android, etc...
- 超過2500個函式

OpenCV模組



OpenCV應用

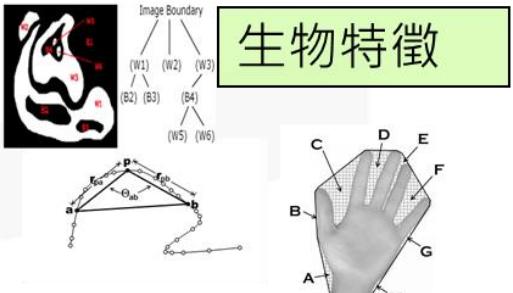
Robot support
> 2500 functions



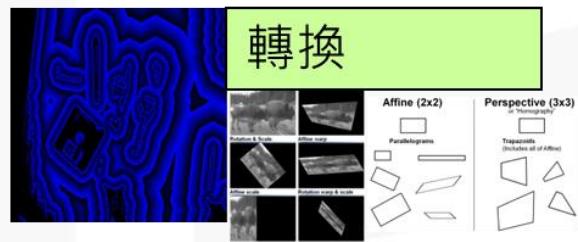
通用圖像函式



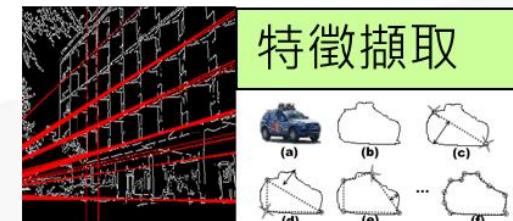
切割



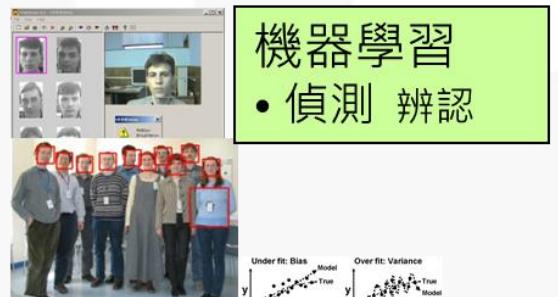
生物特徵



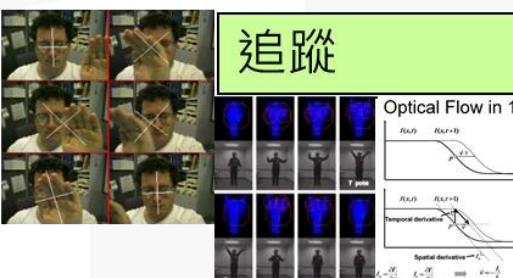
轉換



特徵擷取



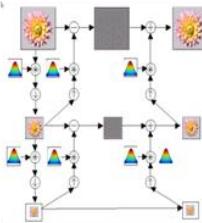
機器學習 • 偵測 辨認



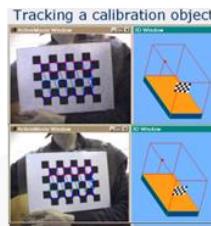
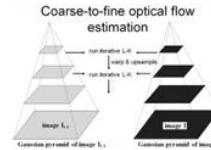
追蹤



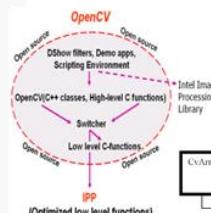
矩陣數學



影像金字塔



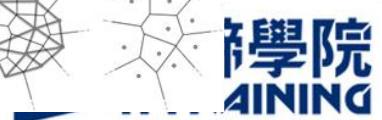
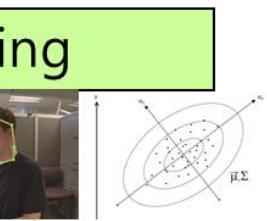
攝影機校準



工具和資料結構



Fitting



海
學院
AINING

■ Anaconda ，森蚺(ㄤ一ㄙˋ)

- ✓ 包含了眾多流行的科學、數學、工程、數據分析的 Python包，完全開源和免費，額外的加速、優化是收費的，但對於學術用途可以申請免費的 License 。
- ✓ 全平台支持：Linux、Windows、Mac5.
- ✓ 支持 Python 2.6、2.7、3.3、3.4，可自由切換
- ✓ 內帶spyder 編譯器與jupyter notebook 環境
(舊版的ipython notebook)

■ 優點：省時！

一鍵安裝完90%一般人這一輩子會用到的Python套件，剩下的再用pip install個別去安裝即可

■ 缺點：占空間

包含了一堆用不到的Python的套件(可安裝另一種miniconda)

另外，安裝完Anaconda之後你會獲得一個叫Conda的指令可以用，可以讓你用conda管理不同Python的版本以及套件。

Anaconda套件

NOW PLAY WITH THE WORLD'S MOST AWESOME DATA SCIENCE PACKAGES

Packaged included in Anaconda 4+, or get with "conda install PACKAGENAME"

1. NumPy | numpy.org

N-dimensional array for numerical computation

2. SciPy | scipy.org

Collection of numerical algorithms and toolboxes,
including signal processing and optimization

3. Matplotlib | matplotlib.org

Plotting library for Python

4. Pandas | pandas.pydata.org

Powerful Python data analysis toolkit

5. Seaborn | stanford.edu/~mwaskom/software/seaborn/

Statistical data visualization

6. Bokeh | bokeh.pydata.org

Interactive web visualization library

7. SciKit-Learn | scikit-learn.org/stable

Python modules for machine learning and data mining

8. NLTK | nltk.org

Natural language toolkit

9. Notebook | jupyter.org

Web-based interactive computational
environment combines code execution, rich
text, mathematics, plots and rich media

10. R essentials | conda.pydata.org/docs/r-with-conda.html

R with 80+ of the most used R packages for data science
"conda install -c r r-essentials"

10 (以上為Anaconda內含的重要套件，完整套件可點此[參考](#))

- Step01、Anaconda官方網站
<https://www.anaconda.com/download/>

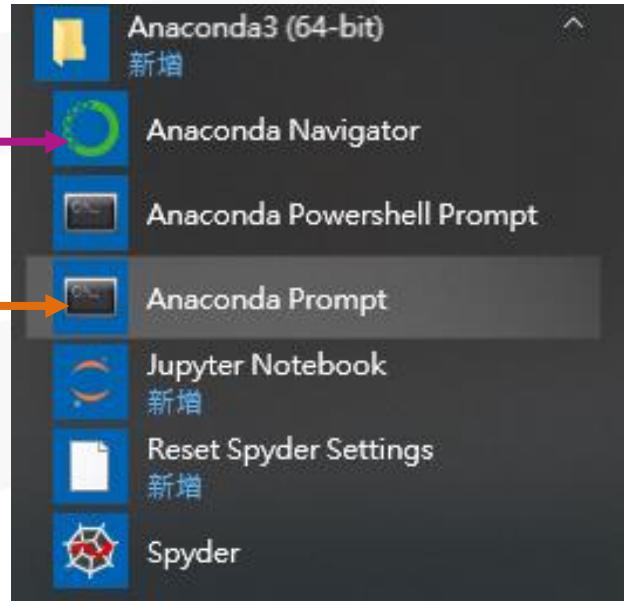
- Step02、選擇作業系統與python版本(建議Python3.6 or 3.5)，可以先安裝3.7之後用指令修改

- Step03、點擊下載的檔案，依序下一步安裝
*請勾選Add path to the system PATH

- Step04、確認是否安裝成功
win+r，輸入cmd，>後輸入python
如下python3.7.3安裝成功

```
C:\Users\shangyu>python
Python 3.7.3 (default, Mar 27 2019, 17:13:21) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32
```

- 另一個驗證方式，從Windows開始選單(Start menu)中選擇Anaconda Navigator，如果Navigator可以打開，表示您已成功安裝Anaconda。



- 透過 Anaconda Prompt 可以進入設好環境變數的終端機其主要語法與命令提示字元cmd內，大致上一樣。

Conda是什麼？

- 如果慣用pip的人對於conda的指令一定也可以馬上上手，因為它和pip指令非常非常的相似
- Conda是套件管理系統，也可用來建立虛擬環境，不過因為Anaconda本身專注於數據分析，所以我們會使用到的像是pandas、NumPy、Scipy的python package在安裝完成時就已經包含在裡面不需要另外安裝了
- 記得不管要做什麼，先確認一下環境的 conda 是最新版、資訊也是最新的，所以都先 update conda 自己。

conda update conda

Conda vs. pip

	Conda	pip
依賴項檢查	<ul style="list-style-type: none">① 列出所需其他依賴包。② 安裝包時自動安裝其依賴項。③ 可以便捷地在包的不同版本中自由切換。	<ul style="list-style-type: none">① 不一定會展示所需其他依賴包。② 安裝包時或許會直接忽略依賴項而安裝，僅在結果中提示錯誤。
環境管理	比較方便地在不同環境之間進行切換，環境管理較為簡單。	維護多個環境難度較大。
對系統本身 python影響	在系統自帶Python中包的更新/回退版本/卸載將影響其他程式。	不會影響系統自帶Python。
適用語言	僅適用於Python。	適用於Python, R, Ruby, Lua, Scala, Java, JavaScript, C/C++, FORTRAN。

■ 安裝package：

```
conda install packageName
```

■ 安裝特定版本的package：

```
conda install packageName = version
```

```
conda install python=3.5
```

■ 移除package：

```
conda remove packageName
```

■ 若想知道目前電腦內安裝了哪些套件

```
conda list
```

使用conda做虛擬環境的建置(1/2)

■ 創建虛擬環境：

```
conda create --name env_name
```

■ 建立特定python版本的虛擬環境

```
conda create --name myenv python=3.5
```

■ 啟動/離開 虛擬環境：

```
Activate myenv
```

```
deactivate
```

■ 若想知道目前Anaconda內已安裝幾個虛擬環境 **conda env list**

■ 若想知道某個虛擬環境下，安裝了哪些套件

- ①先啟動某個虛擬環境
- ②再輸入**conda list**

使用conda做虛擬環境的建置(2/2)

- 創建虛擬環境(name後可加上想安裝的套件包)：

```
conda create --name env_name python numpy ...
```

- 根據.txt需求，建置虛擬環境：

```
conda create --name myenv --file environment.txt
```

- 刪除虛擬環境的package/刪除整個虛擬環境：

```
conda remove --name myenv numpy
```

```
conda env remove --name myenv
```

- 儲存當前虛擬環境的package版本

1 檢視包的列表

```
conda list --explicit
```

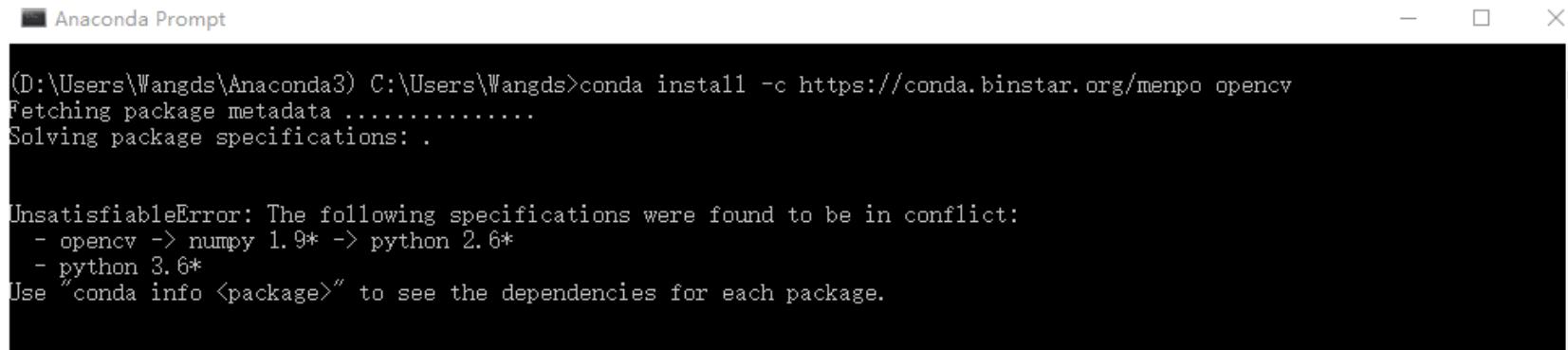
2 將該列表輸出到當前目錄下的txt檔

```
conda list --explicit > Filename.txt
```

■ 快速安裝opencv

```
conda install -c https://conda.binstar.org/menpo opencv
```

如果出現以下錯誤，有另一個安裝方法



```
Anaconda Prompt

(D:\Users\Wangds\Anaconda3) C:\Users\Wangds>conda install -c https://conda.binstar.org/menpo opencv
Fetching package metadata .....
Solving package specifications: .

UnsatisfiableError: The following specifications were found to be in conflict:
  - opencv -> numpy 1.9* -> python 2.6*
  - python 3.6*
Use "conda info <package>" to see the dependencies for each package.
```

■ 下載一個whl檔，關於opencv的，下載位址在這裡：

<https://www.lfd.uci.edu/~gohlke/pythonlibs/#opencv>

進入到Anadonda3\Lib\site-packages資料夾

把下載的whl檔複製到此路徑下，再需要的環境內輸入

```
pip install opencv_python-4.1.0-cp36-cp36m-win_amd64.whl
```

Spyder IDLE(1/3)

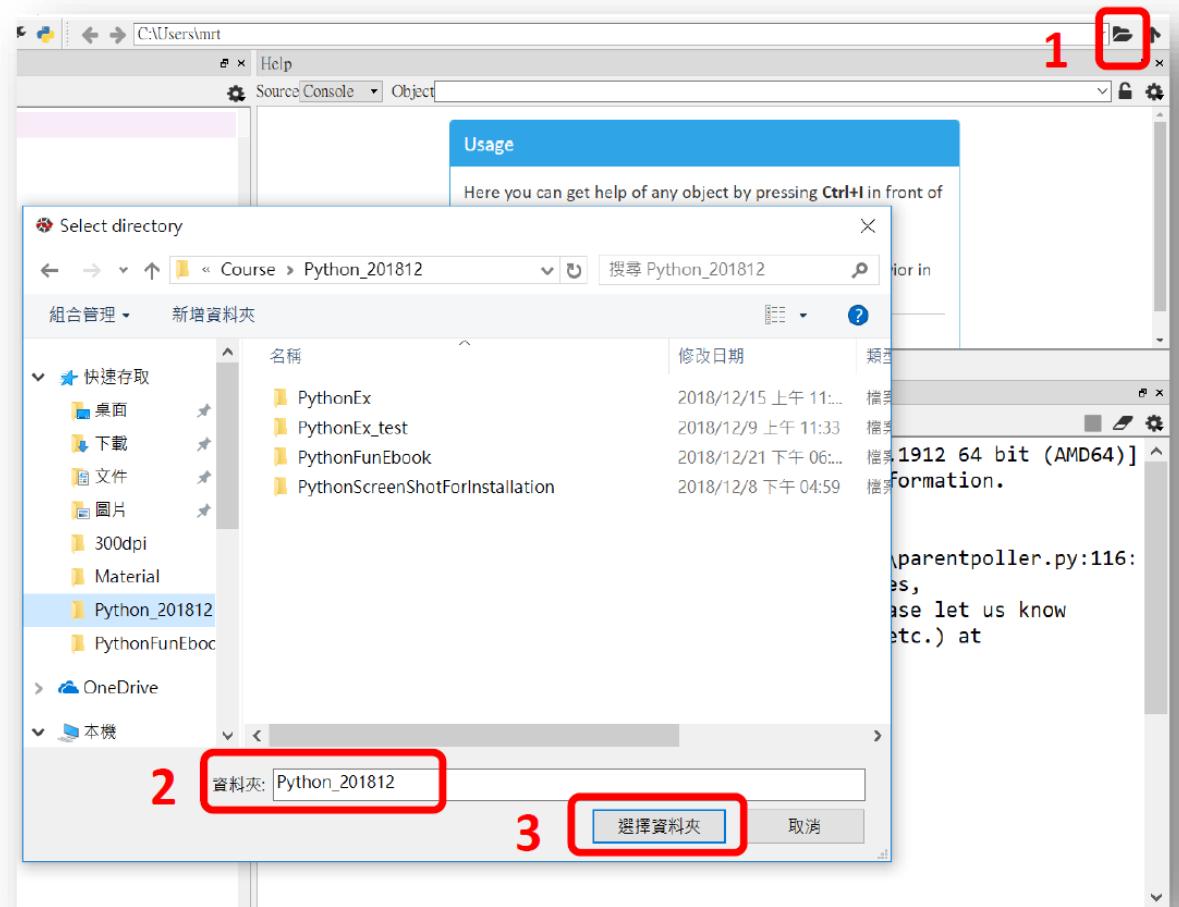


Spyder IDLE(2/3)

環境中的預設路徑不是很方便程式的管理

- 點選面板右上角”瀏覽工作目錄”的圖示
- 選擇事先建立好的目標資料夾位置

* 點選 File explorer 頁籤即可方便管理目標工作資料夾內的檔案



■ 測試spyder環境，套件是否安裝成功

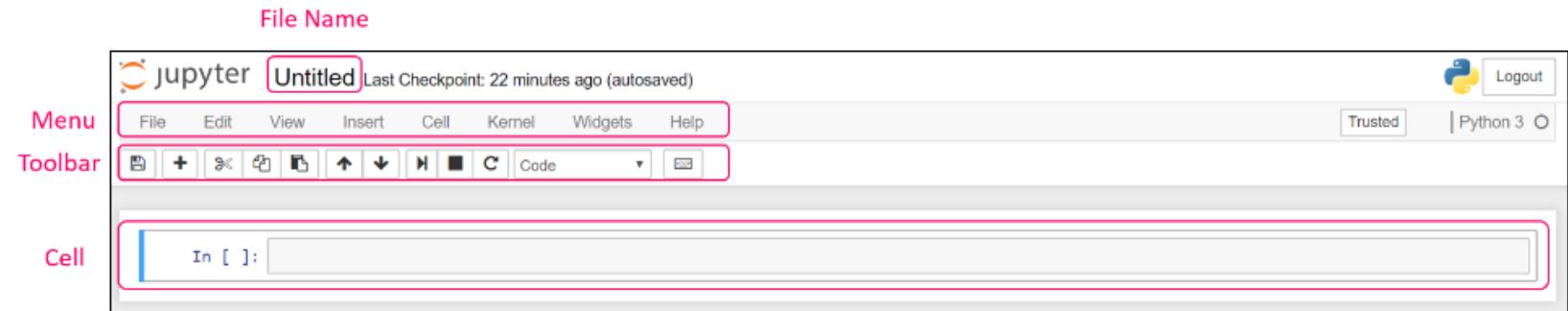
```
In [1]: import cv2  
  
In [2]: import matplotlib  
  
In [3]: import numpy
```

■ 嘗試讀一張圖檔

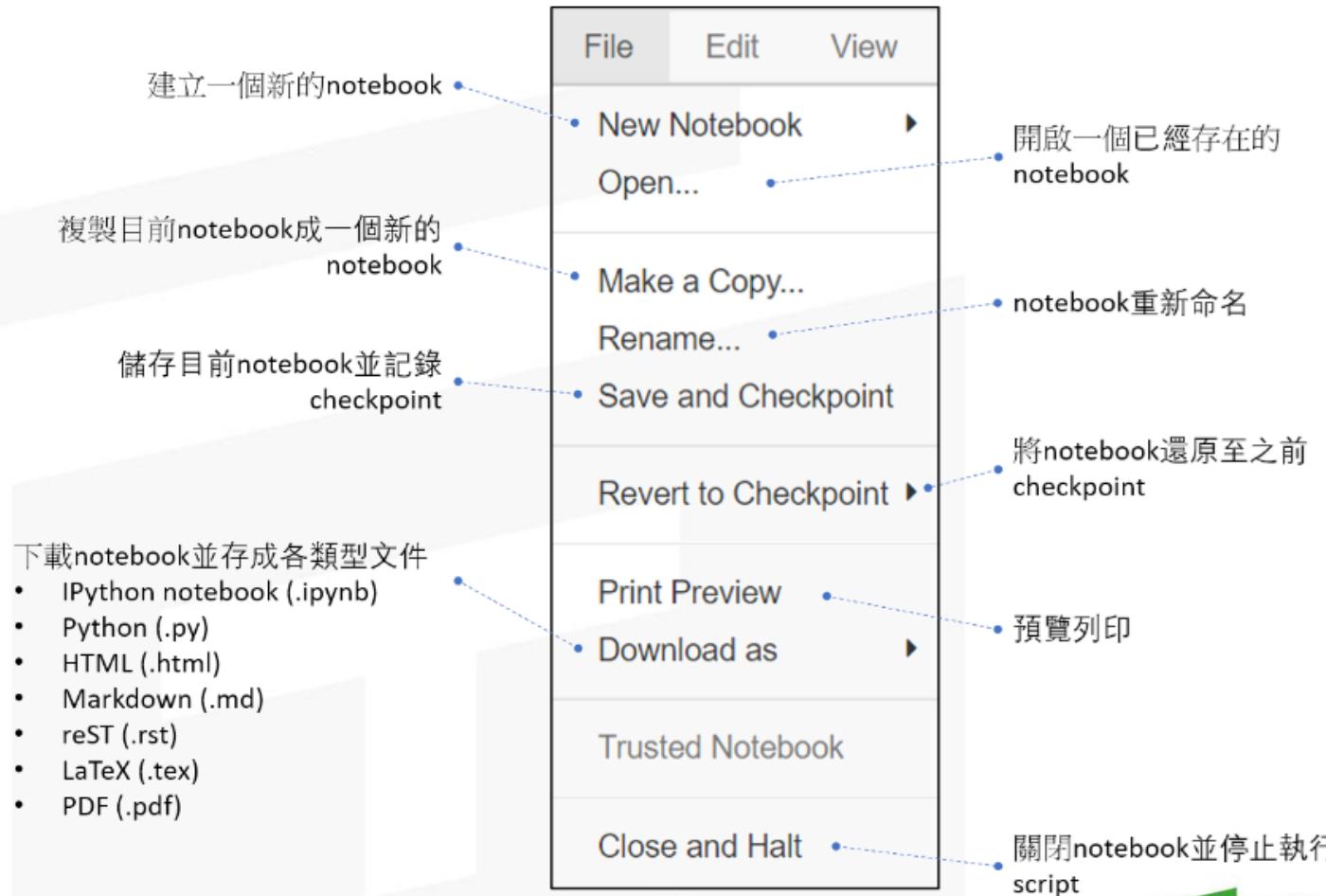
```
import cv2  
  
image2 = cv2.imread('D:\\spyder.jpg')  
cv2.imshow('image',image2)  
  
cv2.waitKey(0)  
cv2.destroyAllWindows()
```

讀取圖檔
顯示圖檔
按下任意鍵則關閉所有視窗

Jupyter Notebook的編輯介面主要分為四部分：檔名(File Name)、主選單(Menu)、工具列(Toolbar)及編輯單元(Cell)。



■ 儲存及載入(Saving/Loading)



■ 程式碼和文本是由3種基本cells類型所包裝起來： Markdown cells、Code cells及Raw NBConvert cells。



Insert

在目前Cell上方
插入一個新Cell



在目前Cell下方
插入一個新Cell

Executing

執行選定的Cell(s)

執行目前Cell(s)並且在下方插入
一個新的Cell

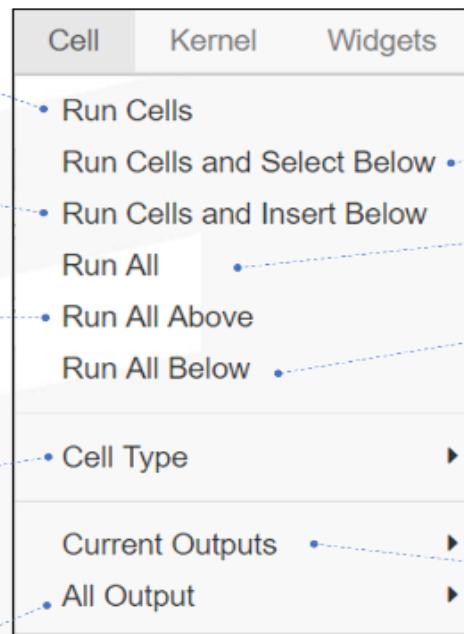
執行目前Cell上方所有的Cells

更改目前Cell的類型

- Code
- Markdown
- Raw NBConvert

對所有Cell輸出做下列動作

- Toggle(隱藏/顯示切換)
- Toggle Scrolling(上下切換)
- Clear(清除)



執行目前Cell(s)並且移動至下
方Cell

執行全部的Cells

執行目前Cell下方所有的Cells

對目前Cell輸出做下列動作

- Toggle(隱藏/顯示切換)
- Toggle Scrolling(上下切換)
- Clear(清除)

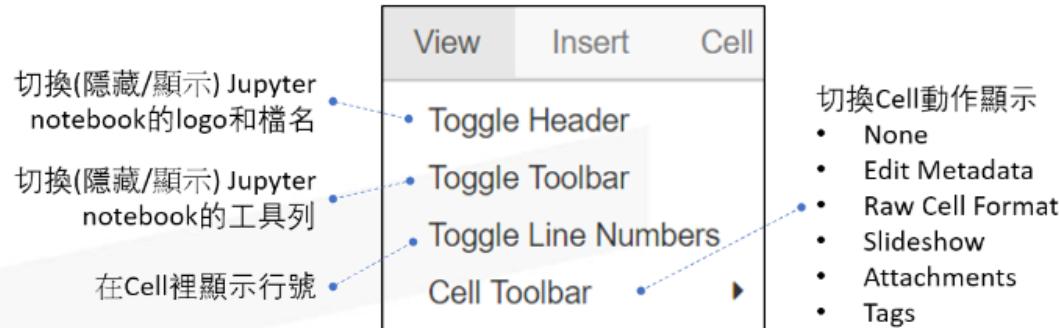
Help

Jupyter Notebook提供了許多線上的使用指南，包括在Data Science中常用的packages。

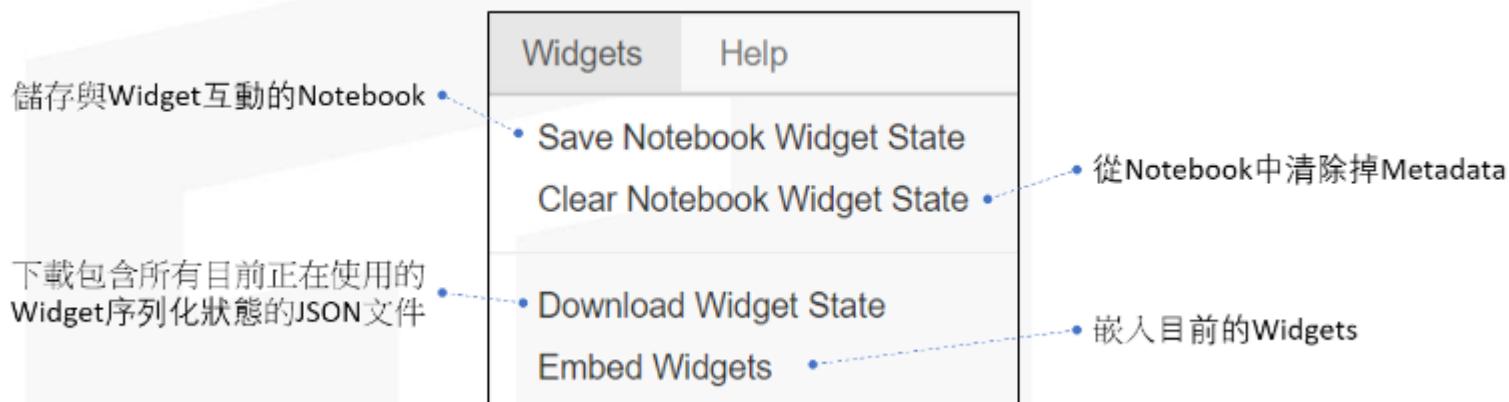


Jupyter Notebook(6/7)

View



Widgets - 提供了可視覺化及控制數據變化的能力



Jupyter Notebook(7/7)

■ 執行程式

- 1.按工具列的Run按鍵
- 2.按Ctrl+Enter
- 3.按Shift+Enter

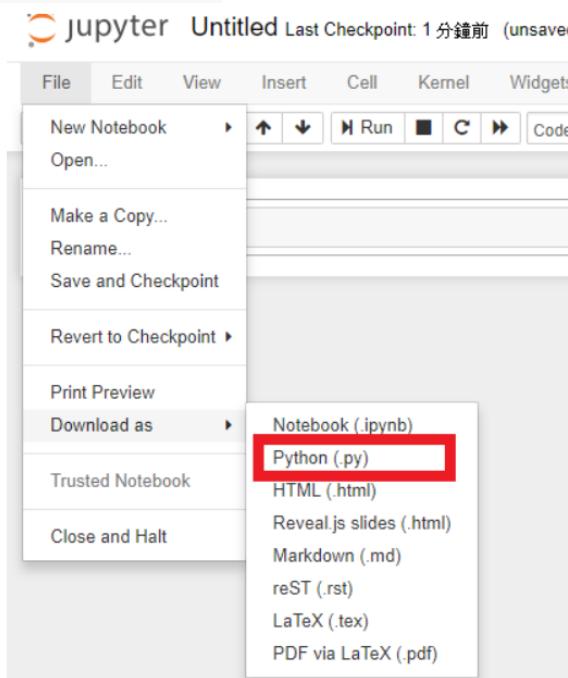


■ Jupyter Notebook

建立檔案的副檔名為「.ipynb」
如果要另外存成「.py」

■ 列出所有的快捷鍵

Esc+h



■ Install OpenCV on Mac OS with PYTHON

<https://www.youtube.com/watch?v=iluST-V757A>

基礎影像python API 資料結構

NumPy、Matplotlib

■ 基礎opencv的python API資料結構

■ 認識NumPy

■ 建置ndarray的物件

ndarray基本運算

ndarray其他運算

■ OpenCV讀取圖片 – NumPy

■ OpenCV讀取圖片 – Matplotlib

- C++版本的OpenCV有已定義的**Class Mat**，作為最基本的影像處理的資料結構。
- Python版本的OpenCV中，則是直接**numpy.ndarray**，以**numpy**為基礎的運算，達成OpenCV的函式。
- 所以本章的練習，除了基本的IO以外，我們將會使用**numpy**直接進行運算，做出OpenCV函式的功能。



- NumPy是python的一種開放原始碼的數值計算擴充，用來儲存和處理多維陣列，核心的資料結構是ndarray

- ndarray

ndarray, a fast and space-efficient multidimensional array providing vectorized arithmetic operations and sophisticated broadcasting capabilities

→ ndarray是一個**快速的**且可以**節省空間的****多維度陣列**
提供**向量運算**以及**複雜的廣播功能**

- ndarray建置的方法

1. 引入numpy的套件
2. 建立資料Data
3. 資料基本運算

```
import numpy as np
```

ndarray資料型別

說明

bool

布林型別，True或False

intc

與C語言中的int型別一致，一般是int32或int64

intp

用於索引的整數，與C語言中ssize_t一致，int32或int64

int8

位元組長度的整數，取值：[-128, 127]

int16

16位長度的整數，取值：[-32768, 32767]

int32

32位長度的整數，取值：[-231, 231-1]

int64

64位長度的整數，取值：[-263, 263-1]

uint8

8位無符號整數，取值：[0, 255]

uint16

16位無符號整數，取值：[0, 65535]

uint32

32位無符號整數，取值：[0, 232-1]

uint64

32位無符號整數，取值：[0, 264-1]

float16

16位半精度浮點數：1位符號位，5位指數，10位尾數

float32

32位半精度浮點數：1位符號位，8位指數，23位尾數

float64

64位半精度浮點數：1位符號位，11位指數，52位尾數

complex64

複數型別，實部和虛部都是32位浮點數

complex128

複數型別，實部和虛部都是64位浮點數

■ 建立ndarray的方法

方法	說明
<code>np.array(list/tuple, dtype)</code>	從列表、元組等中建立
<code>np.arange(n, dtype)</code>	類似range()函式，返回 ndarray 型別，元素從0到n-1
<code>np.ones(shape, dtype)</code>	根據shape生成一個全1陣列，shape是元組型別
<code>np.zeros(shape, dtype)</code>	根據shape生成一個全0陣列，shape是元組型別
<code>np.full(shape, val, dtype)</code>	根據shape生成一個數組，每個元素值都是val
<code>np.eye(n, dtype)</code>	建立n*n單位矩陣，即對角線為1，其餘為0
<code>np.ones_like(a, dtype)</code>	根據陣列a的形狀生成一個全1陣列
<code>np.zeros_like(a, dtype)</code>	根據陣列a的形狀生成一個全0陣列
<code>np.full_like(a, val, dtype)</code>	根據陣列a的形狀生成一個數組，每個元素值都是val
<code>np.linspace(start, end, num)</code>	從start到end等間距地填充num各資料，形成陣列
<code>np.concatenate((a,b))</code>	將兩個或多個數組合併成一個新的陣列

2. 建置二維的 ndarray 資料 ★ 建置一個2行4列，數值全是0

- 行數(高) · 列數(寬) 類型為 uchar 二維陣列(矩陣)

```
z = np.zeros((2,4),np.uint8)
```

- 資料類型

★ np.array() 建置一組2行3列數值

```
data = np.array([[ 0.226, -0.23 , -0.86 ],  
[ 0.5639, 0.2379, 0.904 ]])
```

```
data2 = np.array([1, 2, 3], dtype=complex)
```



np.array() 也可以提供我們要的資料類型給它

只需要在括號內加上 dtype= 資料類型

★ 使用 list 創建 ndarray 資料

先建立一個 list，使用 ndarray()

```
data1 = [6, 7.5, 8, 0, 1]
```

```
arr1 = np.array(data1)
```

輸出 arr1 就可以看到輸出結果：

```
array([ 6., 7.5, 8., 0., 1.])
```

★ 也可以給一個二維的陣列，一樣可以轉成 ndarrays

```
data2 = [[1, 2, 3, 4], [5, 6, 7, 8]]
```

```
arr2 = np.array(data2)
```

★ .ndim - 資料(ndarray)有多少維度

```
data2 = [[1, 2, 3, 4], [5, 6, 7, 8]]  
arr2 = np.array(data2)  
arr2.ndim
```

使用ndim，上面的程式碼可以得到2。
記得要是ndarray的型態才可以哦

如果資料型態list忘記轉成ndarray回出現下列錯誤：

★ .shape - 矩陣的行列數

★ .dtype - 查看資料類型

```
-----  
AttributeError  
 11 last)  
<ipython-input-27-78cb2f6fe29a> in <module>()  
----> 1 data1.ndim  
AttributeError: 'list' object has no attribute 'ndim'
```



ndarray的操作索引 & 局部資料(切片)

方法

.[index]

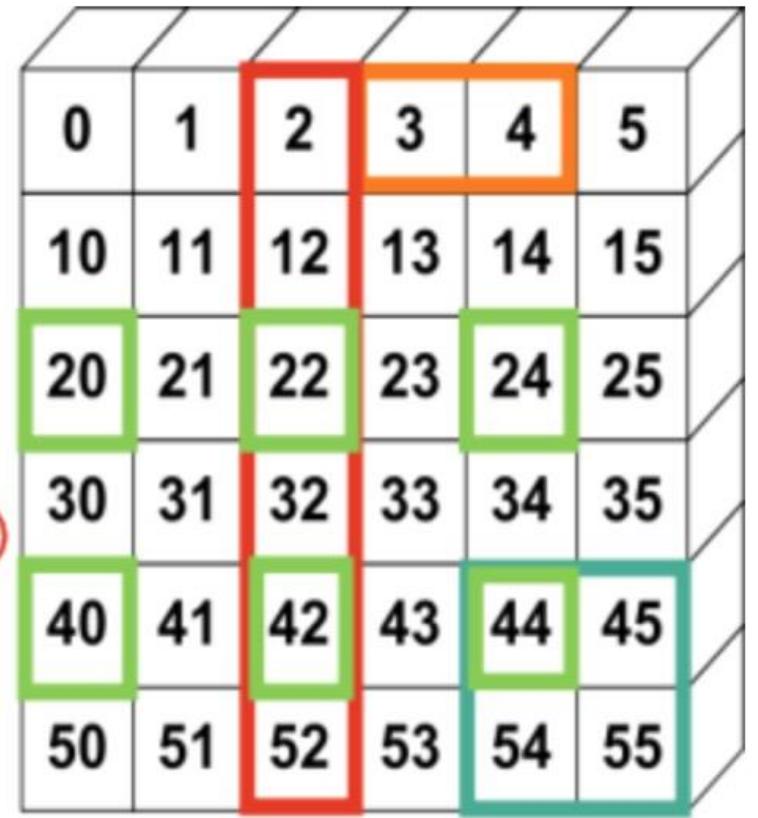
說明

索引 · index 從 0 開始 · -1 表示倒數第一個

.[start, end, dist]

切片 · 從 start 到 end · 間距為 dist

```
>>> a[0,3:5]
array([3,4])
>>> a[4:,4:]
array([[44,45],[54,55]])
>>> a[:,2]
array([2,12,22,32,42,52])
>>> a[2::2,:2]
array([[20,22,24],
       [40,42,44]])
```



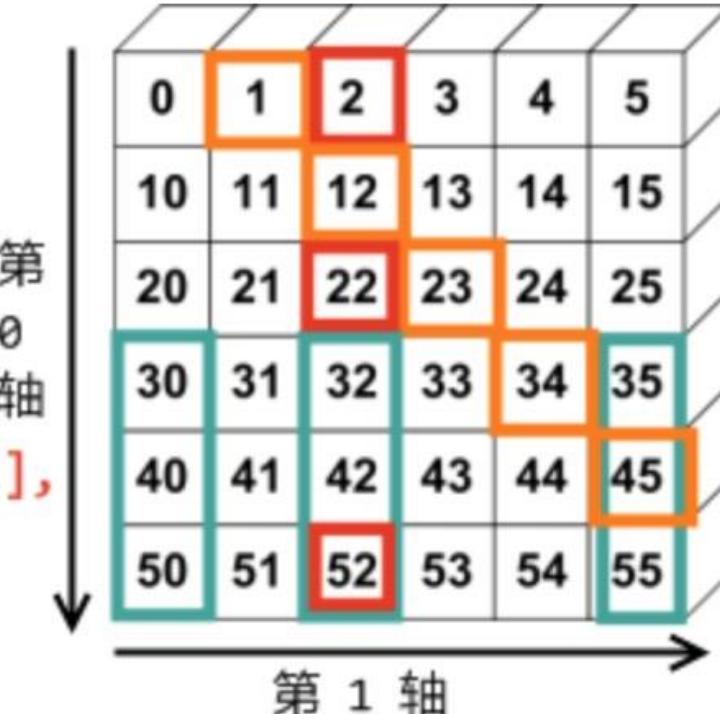
ndarray的操作索引 & 局部資料(切片)

```
>>> a[(0,1,2,3,4),(1,2,3,4,5)]  
array([1,12,23,34,45])
```

```
>>> a[3:,[0,2,5]]  
array([[30,32,35],  
       [40,42,45],  
       [50,52,55]])
```

```
>>> mask=np.array([1,0,1,0,0,1],  
                  dtype=np.bool)
```

```
>>> a[mask,2]  
array([2,22,52])
```



一元函式

函式	說明
<code>np.abs(x)、np.fabs(x)</code>	計算陣列各元素的絕對值
<code>np.sqrt(x)</code>	計算陣列各元素的平方根
<code>np.square(x)</code>	計算陣列各元素的平方
<code>np.log(x)、np.log10(x)、np.log2(x)</code>	計算陣列各元素的自然對數、10底對數和2底對數
<code>np.ceil(x)</code>	<code>np.floor(x)</code> 計算陣列各元素的ceiling值或floor值
<code>np.rint(x)</code>	計算陣列各元素的四捨五入值
<code>np.modf(x)</code>	將陣列各元素的小數和整數部分以兩個獨立陣列形式返回
<code>np.cos(x)、np.cosh(x)</code>	
<code>np.sin(x)、np.sinh(x)</code>	
<code>np.tan(x)、np.tanh(x)</code>	計算陣列各元素的普通型和雙曲型三角函式
<code>np.exp(x)</code>	計算陣列各元素的指數值
<code>np.sign(x)</code>	計算陣列各元素的符號值 · 1(+), 0, -1(-)

二元函式

+ - * / **

兩個陣列各元素進行對應運算

np.maximum(x,y) np.fmax()

np.minimum(x,y) np.fmin()

元素級的最大值/最小值計算

np.mod(x,y)

元素級的模運算

np.copysign(x,y)

將陣列y中各元素值的符號賦值給陣列x對應元素

> < >= <= == !=

兩個陣列各元素進行算術比較，產生布爾型陣列

梯度函式

函式

說明

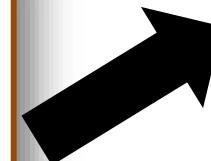
np.gradient(f)

計算陣列f中元素的梯度，當f為多維時，返回每個維度梯度，

中間元素的梯度=(後-前)/2，最左/上邊元素=(後-自己)，最右/下邊元素=(自己-前)，

■ 加法運算

```
import numpy as np  
src1 = np.array([[1,2,3],[4,5,6]],np.uint8)  
src2 = np.array([[1,1,1],[2,2,2]],np.uint8)  
dst = src1+src2  
print(dst)  
print(dst.dtype)
```



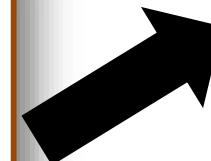
```
[[2 3 4]  
 [6 7 8]]  
uint8
```

$$\star \text{ Src3} = \begin{bmatrix} 23 & 123 & 90 \\ 100 & 250 & 0 \end{bmatrix}, \text{ Src4} = \begin{bmatrix} 125 & 150 & 60 \\ 100 & 10 & 40 \end{bmatrix}$$

```
src3 = np.array([[23,123,90],[100,250,0]],np.uint8)  
src4 = np.array([[125,150,60],[100,10,40]],np.uint8)  
dst2 = src3+src4
```

■ 加法運算

```
import numpy as np  
src1 = np.array([[1,2,3],[4,5,6]],np.uint8)  
src2 = np.array([[1,1,1],[2,2,2]],np.uint8)  
dst = src1+src2  
print(dst)  
print(dst.dtype)
```



```
[[2 3 4]  
 [6 7 8]]  
uint8
```

★ $\text{Src3} = \begin{bmatrix} 23 & 123 & 90 \\ 100 & 250 & 0 \end{bmatrix}, \text{Src4} = \begin{bmatrix} 125 & 150 & 60 \\ 100 & 10 & 40 \end{bmatrix}$

$$\text{dst2} = \begin{bmatrix} 148 & 17 & 150 \\ 200 & 4 & 40 \end{bmatrix}$$

array 對大於 255 的 uchar 類型，將該數 255 取模運算後減 1
$273 \% 255 - 1 = 17$

■ 加法運算(不同資料類型)

```
src3 = np.array([[23,123,90],[100,250,0]],np.uint8)
src4 = np.array([[125,150,60],[100,10,40]],np.float32)
dst3 = src3+src4
print(dst3,dst3.dtype)
```

■ 加法運算(Opencv中add函式)

```
dst4 = cv2.add(src3,src4,dtype=cv2.CV_32F)
print('dst4:',dst4)
print(dst4.dtype)
```

■ 加法運算(不同資料類型)

```
src3 = np.array([[23,123,90],[100,250,0]],np.uint8)
src4 = np.array([[125,150,60],[100,10,40]],np.float32)
dst3 = src3+src4
print(dst3,dst3.dtype)
```

■ 加法運算(Opencv中add函式)

```
dst4 = cv2.add(src3,src4,dtype=cv2.CV_32F)
print('dst4:',dst4)
print(dst4.dtype)
```



```
[[148. 273. 150.]
 [200. 260. 40.]] float32
```

* 記得使用cv2時
要import cv2模組
*cv2 dtype注意大小寫

■ 減法運算

大致上與加法類似，比較一下c的Mat的減法會有一些不同

#減法運算-1

```
src1 = np.array([[23,123,90],[100,250,0]],np.uint8)
src2 = np.array([[125,150,60],[100,10,40]],np.uint8)
dst = src1-src2
print('dst:',dst)
print(dst.dtype)
```

Numpy 的處理方式，將該數255取模運算後加1

-102 % 255 + 1 = 154

#Mat處理轉為uchar類型時，數值自動截斷為0

#[[0,0,30][0,240,0]]

■ 減法運算(Opencv中subtract函式)

#減法運算-2

```
src1 = np.array([[23,123,90],[100,250,0]],np.uint8)
src2 = np.array([[125,150,60],[100,10,40]],np.uint8)
dst2 = cv2.subtract(src1,src2)
print('dst2:',dst2)
print(dst2.dtype)
```

#減法運算-3

```
dst3 = cv2.subtract(src1,src2,dtype = cv2.CV_32F)
print('dst3:',dst3)
print(dst3.dtype)
```

■ 減法運算

```
dst: [[154 229 30]
      [ 0 240 216]]
      uint8
```

■ 減法運算(Opencv中subtract函式)

```
dst2: [[ 0  0 30]
        [ 0 240  0]]
        uint8
dst3: [[-102. -27. 30.]
        [ 0. 240. -40.]]
        float32
```

■ np加、減、乘與除法(np函式)

$$x = \begin{bmatrix} 2 & 3 \\ 1 & 4 \end{bmatrix}, \quad y = \begin{bmatrix} 4 & 2 \\ 1 & 3 \end{bmatrix}$$

```
import numpy as np
x = np.array([[2,3],[1,4]], dtype=np.float64)
y = np.array([[4,2],[1,3]], dtype=np.float64)
print('x + y:',np.add(x, y),sep='\n')
print('x - y:',np.subtract(x, y),sep='\n')
print('x * y:',np.multiply(x, y),sep='\n')
print('x / y:',np.divide(x, y),sep='\n')
```

ndarray的基本運算

■ np點加、點減、點乘與點除法(np函式)

$$x = \begin{bmatrix} 2 & 3 \\ 1 & 4 \end{bmatrix}, y = \begin{bmatrix} 4 & 2 \\ 1 & 3 \end{bmatrix}$$

x + y:
 [[6. 5.]
 [2. 7.]]

x - y:
 [[-2. 1.]
 [0. 1.]]

x * y:
 [[8. 6.]
 [1. 12.]]
 x / y:
 [[0.5 1.5]
 [1. 1.33333333]]

NumPy在處理分母為0時
 *如果兩個ndarray為uint8
 類型，則回傳值為0
 *其他情況傳回inf

```
x2 = np.array([[2,3],[1,4]], dtype=np.uint8)
y2 = np.array([[4,2],[1,0]], dtype=np.uint8)
z = x2/y2
z=z.astype(np.uint8)
print('x / y:',z,z.dtype,sep='\n')

y = np.array([[4,2],[1,0]], dtype=np.float64)
print('x / y:',np.divide(x, y),sep='\n')
```

NumPy廣播

NumPy is smart enough to use the original scalar value without actually making copies, so that broadcasting operations are as memory and computationally efficient as possible



廣播的使用原來的向量而不需要實際的複製
它可以有效的運用記憶體和運算

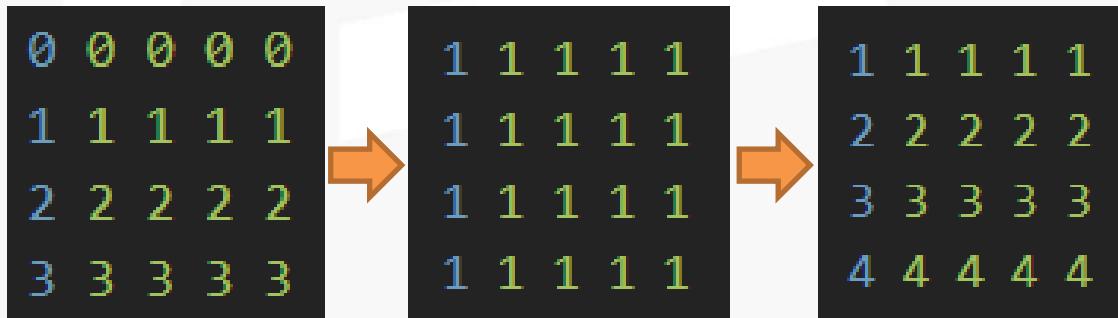
相當於d也是一個array
想像是NumPy會把d給放大
變成和c一樣的大小 (shape)
而不需要自己去複製和c一樣多個2

```
import numpy as np
c = np.array([1.0, 2.0, 3.0])
d = 2
dst = c * d
print(dst)
```

NumPy廣播

```
x = np.arange(4)
print('x:',x,sep='\n')
x2 = x.reshape(4,1)
print('x2:',x2,sep='\n')
y = np.ones(5)
print('y:',y,sep='\n')
print('x2+y:',x2+y,sep='\n')
```

我們想像 $x2 + y$ 的時候 x 會延伸，而 y 亦是結果
則為 5×5 矩陣



x:

[0 1 2 3]

x2:

[[0]]

[1]

[2]

[3]]

y:

[1. 1. 1. 1. 1.]

x2+y:

[[1. 1. 1. 1. 1.]]

[2. 2. 2. 2. 2.]

[3. 3. 3. 3. 3.]

[4. 4. 4. 4. 4.]

■ 矩陣乘法運算

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

矩陣乘法：

$$A \times B = \begin{bmatrix} 1[5 \ 6] + 2[7 \ 8] \\ 3[5 \ 6] + 4[7 \ 8] \end{bmatrix}$$

$$= \begin{bmatrix} [5 \ 6] + [14 \ 16] \\ [15 \ 18] + [28 \ 32] \end{bmatrix}$$

$$= \begin{bmatrix} 19 & 32 \\ 43 & 50 \end{bmatrix}$$

矩陣相對應位置相乘：

$$= \begin{bmatrix} 1 \times 5 & 2 \times 6 \\ 3 \times 7 & 4 \times 8 \end{bmatrix} = \begin{bmatrix} 5 & 12 \\ 21 & 32 \end{bmatrix}$$

■ 矩陣乘法運算

```
import numpy as np
x = np.array([[1,2,3],[4,5,6]], dtype=np.uint8)
y = np.array([[6,5],[4,3],[2,1]], dtype=np.uint8)
z = np.dot(x,y)
print('np.dot(x,y):',z,sep='\n')
print(z.dtype)

y = np.array([[6,5],[4,3],[2,1]], dtype=np.float32)
z = np.dot(x,y)
print('np.dot(x,y):',z,sep='\n')
print(z.dtype)
#傳回的資料型態與數值範圍大的類型相同
```

np.dot(x,y):
[[20 14]
 [56 41]]
uint8
np.dot(x,y):
[[20. 14.]
 [56. 41.]]
float32

轉置矩陣

```
In [18]: x = np.zeros((10, 2))  
x
```

建立一個都是0的矩陣，矩陣形狀為 10×2

```
Out[18]: array([[ 0.,  0.],  
                 [ 0.,  0.],  
                 [ 0.,  0.],  
                 [ 0.,  0.],  
                 [ 0.,  0.],  
                 [ 0.,  0.],  
                 [ 0.,  0.],  
                 [ 0.,  0.],  
                 [ 0.,  0.],  
                 [ 0.,  0.]])
```

T這個是將x整個選轉變成 2×10

```
In [25]: y = x.T  
y
```

```
Out[25]: array([[ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],  
                  [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.]])
```

■ ndarray陣列的變換

方法	說明
.reshape(shape)	不改變陣列元素，返回一個 shape 形狀的陣列，原陣列不變
.resize(shape)	與.reshape()功能一致，但修改原陣列
.swapaxes(ax1,ax2)	將陣列n個維度中兩個維度進行調換
.flatten()	對陣列進行降維，返回摺疊後的一維陣列，原陣列不變
.astype(new_type)	不改變陣列元素，改變元素的資料型別
.tolist()	轉化為列表 list 型別

■ 改變矩陣size

簡單來說 `reshape()` 就是用來改變array的shape，但並不會改變它的數值。

```
s = np.arange(6)
```

建立一個 `s = [0,1,2,3,4,5]`，接著將 `s` 做 `reshape()`

```
s = s.reshape((3, 2))
```

`s` 就會變成 `3 X 2` 了，如圖：

```
In [53]: s = np.arange(6)
s = s.reshape((3, 2))
s
```

```
Out[53]: array([[0, 1],
                 [2, 3],
                 [4, 5]])
```

■ 指數和對數

```
import numpy as np

#Log是以e為底
a = np.array([[6,5],[4,3]], dtype=np.uint8)
dst_a = np.log(a)
print('np.log(a):',dst_a,sep='\n')
#ndarray可以為任意資料類型，回傳值型態為float或double
print(dst_a.dtype)
```

np.log(a):
[[1.792 1.609]
 [1.387 1.099]]
float16

■ 指數和對數

```
x = [1,2,9,10, 16]
print("exp()---->", np.exp(x))          # e^x
print("exp2()---->", np.exp2(x))         # 2^x
print("power()---->", np.power(3, x))     # 3^x
print("log()---->", np.log(x))           # Log自然對數
print("log2()---->", np.log2(x))          # 以2為基底的對數
print("log10()---->", np.log10(x))        # 以10為基底的對數
```

```
exp()----> [2.71828183e+00 7.38905610e+00 8.10308393e+03 2.20264658e+04
8.88611052e+06]
exp2()----> [2.0000e+00 4.0000e+00 5.1200e+02 1.0240e+03 6.5536e+04]
power()----> [ 3 9 19683 59049 43046721]
log()----> [0. 0.69314718 2.19722458 2.30258509 2.77258872]
log2()----> [0. 1. 3.169925 3.32192809 4. ]
log10()----> [0. 0.30103 0.95424251 1. 1.20411998]
```

ndarray的其他運算

■ 幂指數

```
x1 = range(6)
print('x1^2 = ',np.power(x1, 2))
```

幂指數的資料類型，對於power回傳的值影響很大

```
src = np.array([[25,40],[10,100]],np.uint8)
dst1 = np.power(src,2)
print('dst1:',dst1,sep='\n')
print(dst1.dtype)
dst2 = np.power(src,2.0)
print('dst2:',dst2,sep='\n')
print(dst2.dtype)
```

x1^2 = [0 1 4 9 16 25]

dst1:

[[113 64]
 [100 16]]

float16

dst2:

[[625. 1600.]
 [100. 10000.]]

float16

統計函式

函式	說明
sum(a, axis=None)	根據給定軸axis計算陣列a相關元素之和，axis整數或元組
mean(a, axis=None)	根據給定軸axis計算陣列a相關元素的期望，axis整數或元組
average(a, axis=None, weights=None)	根據給定軸axis計算陣列a相關元素的加權平均值
std(a, axis=None)	根據給定軸axis計算陣列a相關元素的標準差
var(a, axis=None)	根據給定軸axis計算陣列a相關元素的方差
min(a)、max(a)	計算陣列a中元素的最小值、最大值
argmin(a)、argmax(a)	計算陣列a中元素最小值、最大值的降一維後下標
unravel_index(index, shape)	根據shape將一維下標index轉換成多維下標
ptp(a)	計算陣列a中元素最大值與最小值的差
median(a)	計算陣列a中元素的中位數（中值）

隨機函式

函式

說明

np.random.rand(d0,d1,...,dn)

根據d0-dn建立隨機數陣列，浮點數，[0,1]，均勻分佈

np.random.randn(d0,d1,...,dn)

根據d0-dn建立隨機數陣列，標準正態分佈

np.random.randint(low[,high,shape])

根據shape建立隨機整數或整數陣列，範圍是[low, high)

np.random.seed(s)

隨機數種子，s是給定的種子值

np.random.shuffle(a)

根據陣列a的第1軸進行隨排列，改變陣列x

np.random.permutation(a)

根據陣列a的第1軸產生一個新的亂序陣列，不改變陣列x

np.random.choice(a[size,replace,p])

從一維陣列a中以概率p抽取元素，形成size形狀新陣列，
replace表示是否可以重用元素，預設為False

np.random.uniform(low,high,size)

產生具有均勻分佈的陣列,low起始值,high結束值,size形狀

np.random.normal(loc,scale,size)

產生具有正態分佈的陣列,loc均值,scale標準差,size形狀

np.random.poisson(lam,size)

產生具有泊松分佈的陣列,lambda隨機事件發生率,size形狀

..

二元函式

+ - * / **

兩個陣列各元素進行對應運算

np.maximum(x,y) np.fmax()

np.minimum(x,y) np.fmin()

元素級的最大值/最小值計算

np.mod(x,y)

元素級的模運算

np.copysign(x,y)

將陣列y中各元素值的符號賦值給陣列x對應元素

> < >= <= == !=

兩個陣列各元素進行算術比較，產生布爾型陣列

- 引入python模組

```
import numpy as np  
import cv2
```

- OpenCV 本身就有提供讀取圖片檔的函數可用，讀取一般圖檔，只要呼叫cv2.imread

```
img = cv2.imread('spyder.JPG')
```

- cv2.imread 讀進來的資料，會儲存成一個 NumPy 的陣列，可以使用 type 查看一下

```
print(type(img))
```

- NumPy 陣列的前兩個維度分別是圖片的高度與寬度，#第三個維度則是圖片的 channel (RGB 彩色圖片的 channel 是 3，灰階圖片則為 1)

```
print(img.shape)
```

- OpenCV 的 cv2.imread 在讀取圖片時，可以在第二個參數指定圖片的格式，可用的選項有三種：

cv2.IMREAD_COLOR

此為預設值，這種格式會讀取 RGB 三個 channels 的彩色圖片，而忽略透明度的 channel

cv2.IMREAD_GRAYSCALE

以灰階的格式來讀取圖片。

cv2.IMREAD_UNCHANGED

讀取圖片中所有的 channels，包含透明度的 channel。

- 以灰階的方式讀取圖檔

```
img_gray = cv2.imread('image.jpg', cv2.IMREAD_GRAYSCALE)
```

■ 顯示圖片

```
cv2.imshow('My Image', img)
```

■ 按下任意鍵則關閉所有視窗

```
cv2.waitKey(0)  
cv2.destroyAllWindows()
```

*cv2.waitKey 函數是用來等待與讀取使用者按下的按鍵，而其參數是等待時間（單位為毫秒），若**設定為 0**就表示**持續等待至使用者按下按鍵為止**，當我們按下任意按鍵之後，就會呼叫 cv2.destroyAllWindows 關閉所有OpenCV 的視窗。

■ 關閉特定視窗

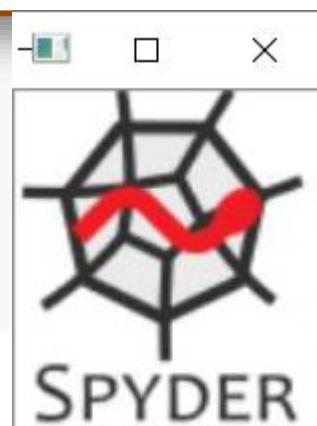
```
cv2.destroyWindow('My Image')
```

■ 讓視窗可以自由縮放大小

```
img = cv2.imread('spyder.JPG')
print(type(img))
print(img.shape)
```

讓視窗可以自由縮放大小

```
cv2.namedWindow('My Image', cv2.WINDOW_NORMAL)
cv2.imshow('My Image', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



- 若要將 NumPy 陣列中儲存的圖片寫入檔案，可以使用 OpenCV 的 cv2.imwrite

```
img = cv2.imread('spyder.JPG')
cv2.imwrite('output.jpg', img)
```

- cv2.imwrite 可透過圖片的副檔名來指定輸出的圖檔格式

```
cv2.imwrite('output.png', img)
cv2.imwrite('output.tiff', img)
```

- 輸出圖片檔案時，也可以調整圖片的品質或壓縮率

```
# 設定 JPEG 圖片品質為 90 (可用值為 0 ~ 100)
cv2.imwrite('output.jpg', img, [cv2.IMWRITER_JPEG_QUALITY, 90])
# 設定 PNG 壓縮層級為 5 (可用值為 0 ~ 9)
cv2.imwrite('output.png', img, [cv2.IMWRITER_PNG_COMPRESSION, 5])
```

■ Matplotlib

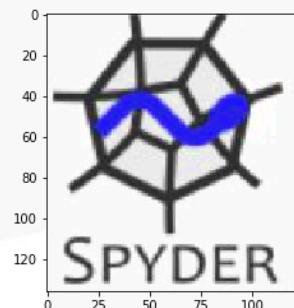
Python 之中很常被使用的繪圖工具，如何以 Matplotlib 顯示 OpenCV 讀入或產生的圖片？

```
from matplotlib import pyplot as plt
```

■ Matplotlib 顯示圖片

只要呼叫 `imshow` 就可以了，但是由於 OpenCV 讀取進來的圖片會以 **BGR** 的方式儲存三個顏色的 channel

如果直接把 OpenCV 讀入的圖片放進 Matplotlib 來顯示，就會出現**顏色錯誤問題**



■ 將 BGR 圖片轉為 RGB 圖片

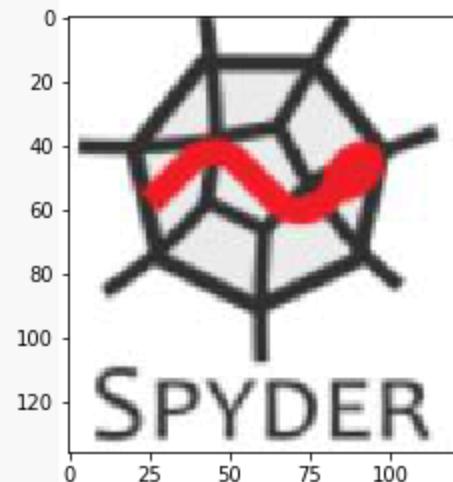
```
img_rgb = img_bgr[:, :, ::-1]
```

或是這樣亦可

```
img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
```

■ 再使用 Matplotlib 顯示圖片

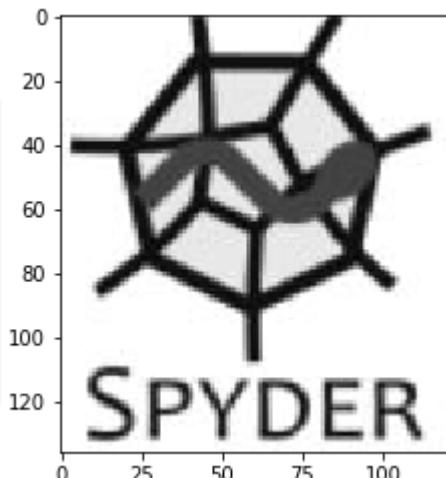
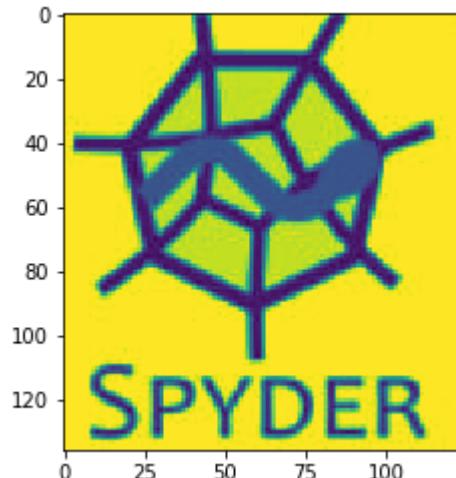
```
plt.imshow(img_rgb)
```



■ 使用 Matplotlib 顯示灰階圖片(如最底圖右)

```
# 使用 OpenCV 讀取灰階圖檔  
img_gray = cv2.imread('spyder.jpg', cv2.IMREAD_GRAYSCALE)  
  
# 使用 Matplotlib 顯示圖片  
plt.imshow(img_gray,cmap = 'gray')
```

* Matplotlib 在顯示一個 channel 的圖片時，會用預設的 colormap 上色(若無指定cmap，則回出現下左圖之錯誤)



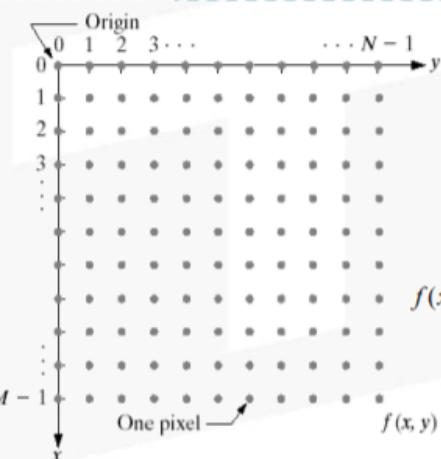
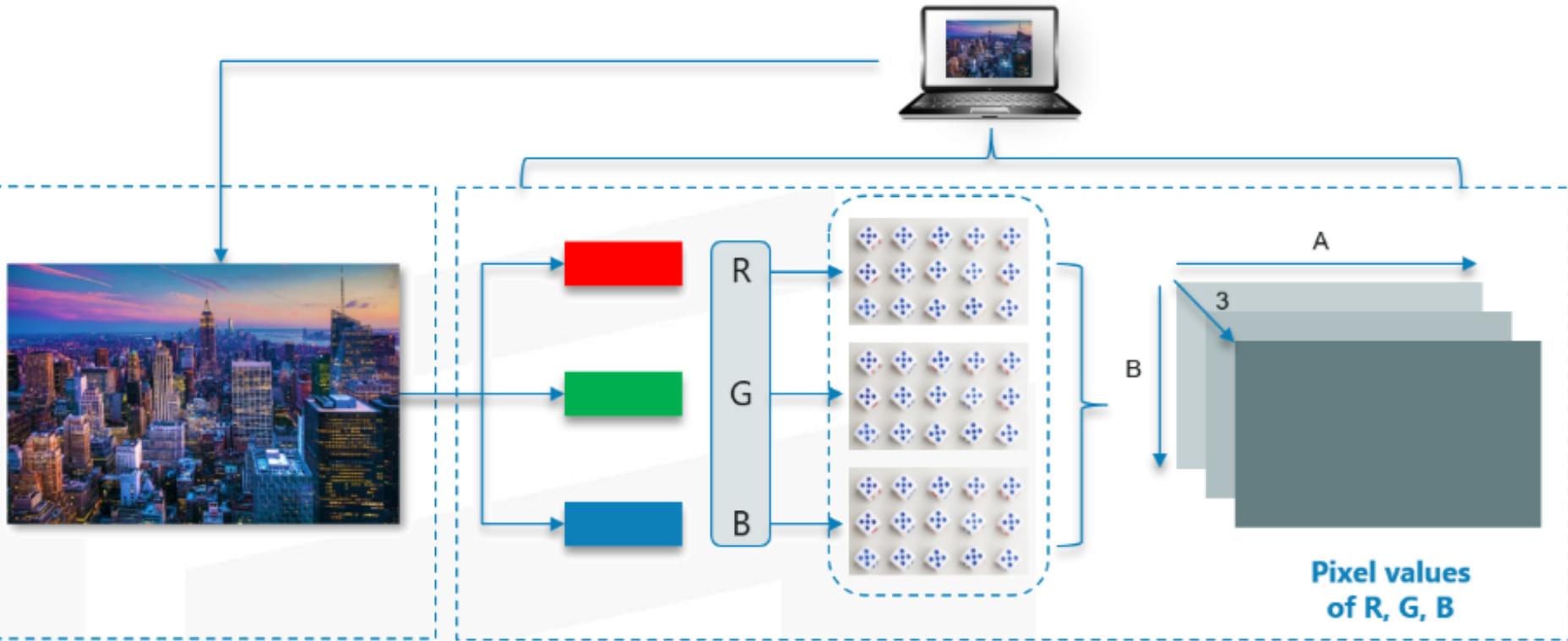
影像處理

基本影像介紹、灰階與彩色影像數位化
案例探討—即時擷取攝影機影像

■ 影像處理 (image processing)

簡單說就是處理影像，讓人或機器(電腦)可以從處理後的影像中獲得更多、更有用的資訊，做更可靠的後續判斷、分析、及應用

- ★ 什麼是影像？(what)
- ★ 什麼是處理？(what)
- ★ 怎麼處理？(how)



每個像素的
■ 座標
■ 顏色值

$$f(x, y) = \begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0, N-1) \\ f(1,0) & f(1,1) & \cdots & f(1, N-1) \\ \vdots & \vdots & \ddots & \vdots \\ f(M-1,0) & f(M-1,1) & \cdots & f(M-1, N-1) \end{bmatrix}$$

數位影像的內容

影像基本元件
(image element)

Pixel
像素

描述每個像素的性質

(1) 座標(x,y)

(2) 顏色值 $f(x,y)$

單色影像:

亮度值(Intensity)
8 bit: 0~255

origin

x





MichelinBaby.bmp



影像讀取



處理結果

$$f = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,N} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,N} \\ \vdots & \vdots & \vdots & \vdots \\ a_{M,1} & a_{M,2} & \cdots & a_{M,N} \end{bmatrix}$$

相關演算法、OpenCV處理函式

程式運算

■ 轉換顏色空間

在 OpenCV 中有 超過150 種進行顏色空間轉換的方法。但是你以後就會發現我們經常用到的也就兩種：BGR \leftrightarrow Gray 和 BGR \leftrightarrow HSV 。

■ 我們用到的函數是cv2.cvtColor(input_imageflag) ，其中 flag就是轉換類型 。

BGR \leftrightarrow Gray的轉換，使用的flag就是cv2.COLOR_BGR2GRAY 。

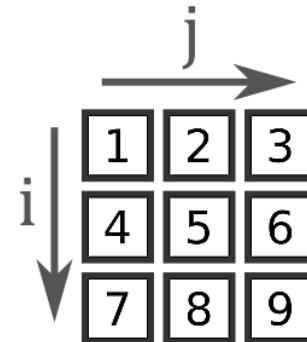
BGR \leftrightarrow HSV的轉換我們用的flag就是cv2.COLOR_BGR2HSV 。

■ 下列程式碼命令能得到所有可用的 flag

```
import cv2
flags=[i for i in dir(cv2) if i.startswith('COLOR_')]
print (flags)
```

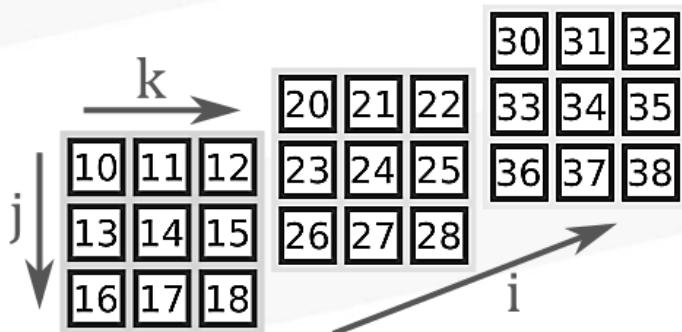
■ 灰階影像數位化

將**灰階**影像圖檔轉成**ndarray**



■ 彩色影像數位化

將**RGB**彩色圖檔轉成**3D的ndarray**



Array RGB								
Page 3 -	blue	intensity	values	0.689	0.706	0.118	0.884	...
Page 2 -	green	intensity	values	0.535	0.532	0.653	0.925	...
Page 1 -	red	intensity	values	0.314	0.265	0.159	0.101	...
				0.553	0.633	0.528	0.493	...
				0.441	0.465	0.512	0.512	...
				0.342	0.647	0.515	0.816	...
				0.111	0.300	0.205	0.526	...
				0.523	0.428	0.712	0.929	...
				0.214	0.604	0.918	0.344	...
				0.100	0.121	0.113	0.126	...
				0.112	0.986	0.234	0.432	...
				0.765	0.128	0.863	0.521	...
				1.000	0.985	0.761	0.698	...
				0.455	0.783	0.224	0.395	...
				0.021	0.500	0.311	0.123	...
				1.000	1.000	0.867	0.051	...
				1.000	0.945	0.998	0.893	...
				0.990	0.941	1.000	0.876	...
				0.902	0.867	0.834	0.798	...
			

■ 數位影像處理的本質就是要操作矩陣，透過特定的演算法改變矩陣**ndarray**中的值，進一步改變影像的顯示，或是達成預期的目標與效果。

■ 設定色彩的門檻閥值做遮罩(mask) p13

設定色彩的門檻閥值

根據門檻值(threshold)做遮罩

對原圖和mask進行運算

■ 調整飽和度和亮度的控制項 p14

圖像歸一化，圖檔RGB轉成HLS

■ 人臉區域檢測(將圖檔RGB轉成HSV) p15

Pitas等人提出在HSV空間建立肤色模型。不要求颜色归一化并且对光照鲁棒性很强，条件同时满足才会被分割成皮肤。实现条件如下：

$$\begin{cases} (0^\circ \leq H \leq 25^\circ) \vee (335^\circ \leq H \leq 360^\circ) \\ 0.2 \leq S \leq 0.6 \wedge 0.4 \leq V_{\text{shadow_guo}} \end{cases}$$