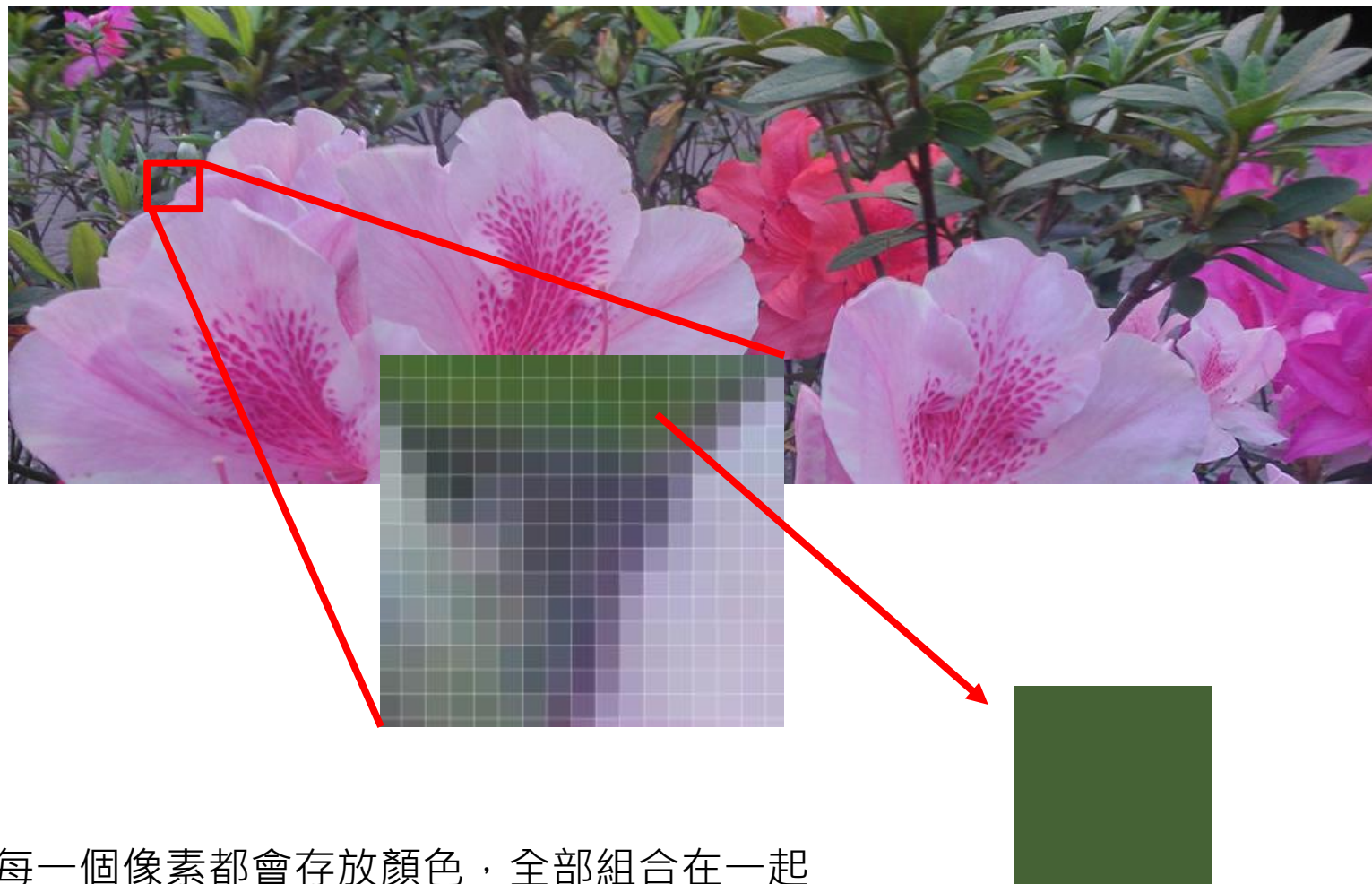


# 電腦影像數據處理

蘇柏原(teaching@bo-yuan.net)

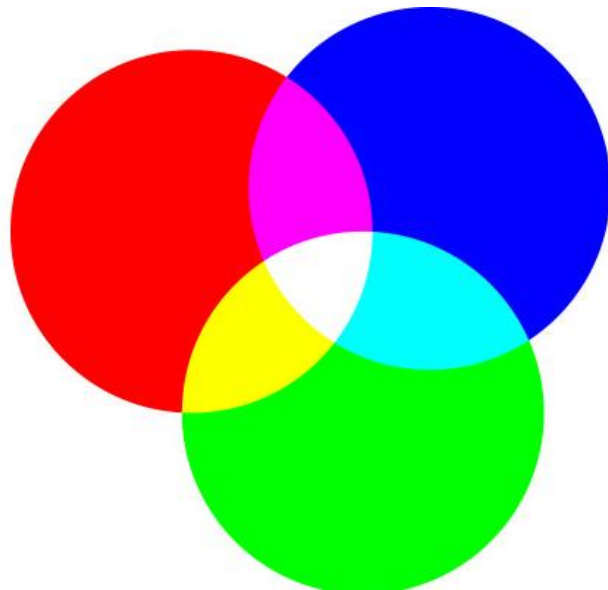
# 影像的構成



每一個像素都會存放顏色，全部組合在一起  
就會變成一張圖片

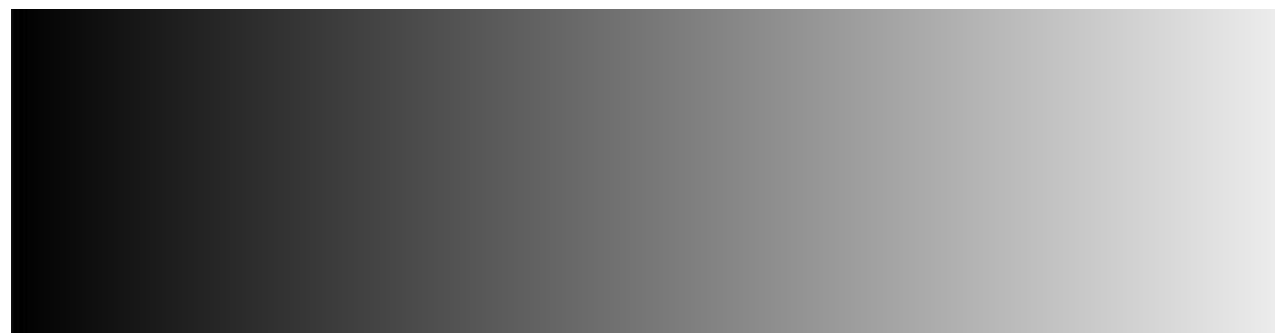
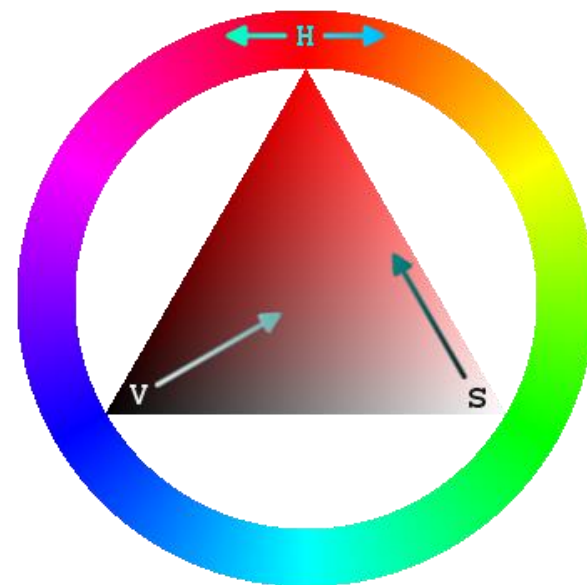
像素

# 色彩空間



RGB(紅, 綠, 藍)

HSV(色相, 飽和度, 明度)



GRAY(灰階)

其他還有許多，如CMYK, CIE, xvYCC等等

# 軟體安裝

- 開啟命令提示字元安裝「OpenCV」函式庫：
  - `pip install opencv-python`
  - `pip install opencv-contrib-python`

# 使用OpenCV

- 在Python使用OpenCV :

```
import cv2
import numpy as np
import IPython.display as dp
from google.colab.patches import cv2_imshow
```
- 「cv2」是OpenCV的函式庫
- 「numpy」是一個矩陣與數學運算的函式庫

# 讀取並顯示圖片

- 讀取一張圖像：

圖像變數=cv2.imread("圖片路徑", 讀取方式)

- 顯示圖片(COLAB使用)：

cv2.imshow(圖片變數)

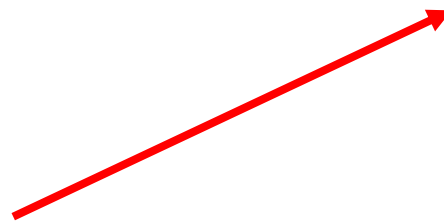
1 => 一般(不含透明度)  
-1 => 完整(包含透明度)  
0 => 灰階

# 圖片變數

- 「cv2.imread」返回的圖片變數會以多維陣列形式表示：
  - 第一維：圖片的高
  - 第二維：圖片的寬
  - 第三維：色彩空間的各個值(灰階沒有第三維)
- 返回的多維陣列是numpy函式庫的類型，所以可以透過「shape」屬性取得維度大小，譬如：
  - image.shape[0] => 取得圖片高
  - image.shape[1] => 取得圖片寬
  - image.shape[2] => 取得當前色彩空間的通道數量

# 色彩空間轉換

- 預設讀進來的圖片色彩空間都是BGR
- 轉換函式：  
轉換後的圖像變數=cv2.cvtColor(圖片變數, 轉換類型)
- 轉換類型(常用的)：
  - cv2.COLOR\_BGR2HSV
  - cv2.COLOR\_BGR2GRAY
  - cv2.COLOR\_HSV2BGR
  - cv2.COLOR\_GRAY2BGR





# 儲存圖片

- 儲存圖片：  
`cv2.imwrite("圖片路徑", 圖片變數, 設定參數)`
- 圖片格式(常用的):
  - PNG => 支援透明色(陣列的第三維的第四個值會當作透明色值)
  - JPG => 不支援透明色，但可調整壓縮率，加在設定參數上：
    - `[cv2.IMWRITE_JPEG_QUALITY, 畫質比率]`

0~100



# 建立一張圖片

- 由於OpenCV裡面是靠numpy陣列在記錄圖像資訊，所以只要建立一個numpy陣列就等同於建立一個圖像了：

變數=np.full((維度一長度, 維度二長度...), 初始值, 陣列型態)

- RGB影像固定會是一個三維陣列，且值的表示為0~255(8bit)：

圖像變數=np.full((高, 寬, 3), 初始顏色值, np.uint8)

除了Gray以外其他常用的  
色彩空間都需要第三維

基本上各個色彩空間都是8bit

可以透過tuple類  
型分別設定顏色值

# 繪圖

- 除繪製多邊形外其他圖形的座標點都只能是Tuple類型，用以表示(x, y)
- 所有的顏色參數都依照色彩空間的不同可傳進陣列或單一數值

- 繪直線：

`cv2.line`(圖像變數, 線的起點, 線的終點, 顏色, 線粗細)

- 繪矩形：

`cv2.rectangle`(圖像變數, 矩形左上點, 矩形右下點, 顏色, 線粗細)

- 繪圓形：

`cv2.circle`(圖像變數, 中心點, 半徑, 顏色, 線粗細)

繪製矩形跟圓形如果粗細設為「-1」則會繪製實心圖形

# 寫字

- 如果要載入自訂的字型，需使用PIL函式庫的Image, ImageFont, ImageDraw模組(**pip install Pillow**)：  
`from PIL import ImageFont, ImageDraw, Image`

`PIL圖像變數=Image.fromarray(OpenCV圖像變數)`

`ImageFont.truetype(TTF字型檔位置, 文字大小)`

`ImageDraw.Draw(PIL圖像變數).text(文字位置, 要寫的文字, 顏色, 設定)`

這裡的Y軸是從上往下算

`OpenCV圖像變數=np.array(PIL圖像變數)`

# 動畫(COLAB)

- 清除畫面：

`dp.clear_output(等待新輸出)`

True或False

- 等待時間：

`time.sleep(秒數)`

可以有小數點

# 影片儲存

- 建立儲存控制變數：

控制變數=cv2.VideoWriter(檔案路徑, 格式, 每秒影格數, 大小)

格式不同會有不同的副檔名

- 影片格式設定：

cv2.VideoWriter\_fourcc(\*'編碼格式')

Tuple類型：(寬, 高)

MP4V => MP4格式  
XVID => AVI格式

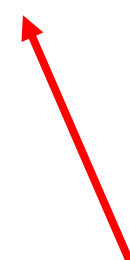
- 寫入影片影格：

控制變數.write(圖像變數)

- 釋放控制變數：

控制變數.release()

# 影片播放 (COLAB)

- 檔案類型轉換：  
`ffmpeg -i 來源檔案 -vcodec libx264 目標檔案`
  - 將影片檔案轉成base64格式：  
`base64.b64encode(檔案內容).decode()`
  - 顯示影片播放器：  
`dp.HTML(  
 f'<video src="data:video/mp4;base64,{影片}" controls />'  
)`
- 

# 影片讀取

- 取得影像來源：

控制變數=cv2.VideoCapture(來源)

- 判斷影像來源是否有開啟：

變數=控制變數.isOpened()

輸入檔案路徑，則讀取指定路徑的影片檔案

- 取得影像：

變數一, 變數二=控制變數.read()

讀到的畫面(圖像)變數

如果有讀到畫面則返回True，否則False



# 影片讀取

- 取得來源資訊：

控制變數.get(參數)

1 => 當前的影格

3 => 影像寬度

4 => 影像高度

5 => 每秒的影格數

7 => 影片的總影格數

- 設定來源資訊：

控制變數.set(參數, 設定)

# 作業1

- 寫一個可以動畫的程式。
- 程式中有一個方形會在左右兩端來回的跑。
- 直到按下任意按鍵為止，否則程式不會結束。
- 完成後將程式碼繳交上傳到系統上。

# 運算

- 圖像相加(大於255的會直接等於255) :  
結果圖像=cv2.add(圖像變數一, 圖像變數二)
- 圖像相減(小於0的會直接等於0) :  
結果圖像=cv2.subtract(圖像變數一, 圖像變數二)
- 圖像相減(小於0的會做絕對值運算) :  
結果圖像=cv2.absdiff(圖像變數一, 圖像變數二)

運算用的「圖像變數二」也可以是一個數字(彩色圖像則要四個數字)

# 運算

- 圖像相除(未整除的會四捨五入)：  
結果圖像=cv2.divide(圖像變數一, 圖像變數二)
- 圖像相乘(大於255的會直接等於255)：  
結果圖像=cv2.multiply(圖像變數一, 圖像變數二)
- 各像素值在二進位表示下做not運算：  
結果圖像=cv2.bitwise\_not(圖像變數)

除了使用函式外，numpy陣列也可以直接做運算(而若運算結果值低於0或者高於255，則超過的部分會從另一端繼續計算)

# 圖像變換

- 圖像縮放：

結果圖像=cv2.resize(圖像變數, 新圖像大小)

↑  
Tuple類型：(寬, 高)

- 圖像翻轉：

結果圖像=cv2.flip(圖像變數, 翻轉方式)

↑  
1 => 左右翻轉  
0 => 上下翻轉  
-1 => 左右與上下皆翻轉

# 運算與計算

`cv2.getRotationMatrix2D(旋轉中心, 角度, 縮放比率)`

- 圖像旋轉：

`結果圖像 = cv2.warpAffine(圖像變數, 變換矩陣, 輸出的圖像大小)`

Tuple類型：(寬, 高)

# 區域裁切、複製和貼上

- 透過numpy的矩陣變數功能可以作到裁切效果：
  - 圖像變數[Y軸範圍起始:Y軸範圍結束, X軸範圍起始: X軸範圍結束]

numpy函式庫可以透過這種  
形式存取或讀取矩陣範圍

# 影像白平衡

- 白平衡的計算公式：
  - $R \times \frac{(R_{avg} + G_{avg} + B_{avg})}{3 \times R_{avg}}$
  - $G \times \frac{(R_{avg} + G_{avg} + B_{avg})}{3 \times G_{avg}}$
  - $B \times \frac{(R_{avg} + G_{avg} + B_{avg})}{3 \times B_{avg}}$

取平均：  
圖像變數.mean()



## 作業2

- 剔除指定照片上除文字以外的景物。
- 不可以使用IF判斷式或者迴圈等指令。
- 完成後將程式碼繳交上傳到系統上。

# 影像二值化

- 透過計算，定義一個門檻值來區隔圖像中所有像素的顏色，使其成為一張由兩個顏色組成的圖像。



# 影像二值化

傳回門檻值(使用THRESH\_OTSU  
方法時才会有不變化)

二值化後的最大值

單一數值，色彩空間內的所  
有值都會依照此門檻值篩選

結果圖像

• 變數一, 變數二=cv2.threshold(圖像變數, 門檻值, 最大值, 方法)

- cv2.THRESH\_BINARY => 超過門檻值的像素設為最大值，小於的設為0
- cv2.THRESH\_BINARY\_INV => 超過門檻值的像素設為0，小於的設為最大值
- cv2.THRESH\_OTSU => 自動計算門檻值來做二值化，可配合其他方法使用  
(只接受單一通道的色彩空間)

# 影像二值化

- 「adaptiveThreshold」會自動計算門檻值，跟「threshold」函式的「THRESH\_OTSU」方法不同點在於他會將整張圖像分成數個小區塊分別去計算(只接受單一通道的色彩空間)：

結果圖像=cv2.adaptiveThreshold(

圖像變數,  
最大值,  
方法一,  
方法二,  
區塊大小,  
微調值

- cv2.ADAPTIVE\_THRESH\_MEAN\_C :  
計算區塊大小內的平均值再減去微調值
- cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C :  
計算區塊大小內的高斯加權平均值再減去微調值

)

- cv2.THRESH\_BINARY
- cv2.THRESH\_BINARY\_INV

# 影像邊緣偵測

- 透過一大一小的門檻值，來計算圖象中的邊線

結果圖像=cv2.Canny(  
圖像變數,  
門檻值1,  
門檻值2  
)

先用大的門檻值檢測邊緣，再用小的門檻值將檢測出來的結果線條連起來

# 影像模糊化

- 平均值模糊法(統計範圍內的色彩值平均)：

結果圖像=cv2.blur(

圖像變數,

範圍大小

)

Tuple類型：(寬, 高)



# 影像模糊化

- 中值模糊法(將處理範圍內的色彩值做排序，取順序在中間的)：

結果圖像=cv2.medianBlur(

圖像變數,

處理數量

)

純數值，必須是單數

# 影像銳利化

- 直方圖均衡化法：  
結果圖像=cv2.equalizeHist(圖像變數)

只接受單一通道色彩空間





# 形態學

- 侵蝕(色彩值低的會侵蝕色彩值高的) :  
結果圖像=cv2.erode(圖像變數, 結構陣列)

- 膨脹(色彩值高的會侵蝕色彩值低的) :  
結果圖像=cv2.dilate(圖像變數, 結構陣列)

np.ones(範圍大小)

Tuple類型 : (高, 寬)

# 形態學

結果圖像 = cv2.morphologyEx(圖像變數, 方法, 結構陣列)

np.ones(範圍大小)

- cv2.MORPH\_OPEN : 先執行侵蝕後執行膨脹
- cv2.MORPH\_CLOSE : 先執行膨脹後執行侵蝕
- cv2.MORPH\_GRADIENT : 執行膨脹與侵蝕產生的變化差

# 色彩篩選

- 判斷圖像裡的各項素是否在指定色彩範圍內：  
結果圖像=cv2.inRange(圖像變數, 顏色下限, 顏色上限)

傳回一張與傳入變數相同  
大小的黑白圖像，在範圍  
內的像素會被設白色，否  
為則黑色

依照色彩空間的不同可傳  
進陣列或單一數值

# 取得輪廓

是個多維陣列，第一維指向不同的輪廓，第二維指向該輪廓對應的所有輪廓點

是個多維陣列，用來記錄各個輪廓的關係，第二維指向各個輪廓，第三維會有四個值，紀錄輪廓的索引（如果為-1代表沒有），分別代表：

1. 相鄰的下一個輪廓
2. 相鄰的上一個輪廓
3. 被其包覆的第一個輪廓
4. 包覆他的輪廓

輪廓點, 輪廓階層資料=cv2.findContours(  
圖像變數(灰階圖像),  
類型,  
方法  
)

- cv2.RETR\_EXTERNAL：只儲存最外層的輪廓
- cv2.RETR\_LIST：儲存所有輪廓，但不建立階層資料
- cv2.RETR\_CCOMP：儲存所有輪廓，但階層資料只包留兩層，首階層為物件外圍，第二階層為內部空心部分的輪廓，如果更內部有其餘物件，包含於首階層
- cv2.RETR\_TREE：儲存所有輪廓與其對應的階層資料

- cv2.CHAIN\_APPROX\_NONE：儲存所有輪廓點
- cv2.CHAIN\_APPROX\_SIMPLE：簡化輪廓點，一條線只儲存頭尾

# 取得輪廓

- 繪製輪廓：

`cv2.drawContours(`

圖像變數,

存取全部輪廓的變數,

要繪製的輪廓索引,

顏色,

粗細

)

如果為「-1」代表全部

Tuple類型

- 取得包覆指定輪廓點的最小正矩形：

X座標, Y座標, 寬度, 高度 = `cv2.boundingRect(指定的輪廓)`

# 作業3

- 抓取指定影片的**前景輪廓**。
- 完成後將程式碼繳交上傳到系統上。

# 文字辨識

- 安裝文字辨識系統tesseract(COLAB) :  
`apt install tesseract-ocr`
- 安裝Python文字辨識Library :  
`pip install pytesseract`
- 在python載入pytesseract :  
`import pytesseract as pt`

# 文字辨識

辨識結果=pt.image\_to\_string(圖片變數, 語言包名稱)

輸入語言包的檔名





# 文字辨識 學習字型

- 執行老師提供的「training.py」檔案，輸入你要的**字型名稱**與作為讓識別系統學習的**字型圖像**檔案

# 條碼偵測

- 偵測條碼可以使用zbar函式庫，該函式庫支援多種條碼的辨識，包含QR Code和Code 39等(COLAB)：

```
apt install libzbar-dev
```

- 安裝對應的Python模組：

```
pip install pyzbar
```

- 載入需要的模組：

```
from pyzbar import pyzbar
```

# 條碼偵測

- 辨識方式：

結果變數 = pyzbar.decode(圖像變數)

串列(list)類型，每個索引  
值指向一個條碼

- 結果變數中各個索引值的內容屬性：

type：條碼類型

rect：包覆條碼的最小矩形資訊

data：結果文字

由於是日本人寫的，所以如果要顯示UTF-8的中文必須：  
decode成UTF-8後再encode成sjis，最後再decode成UTF-8

# 辨識

- 分類器：

控制變數=cv2.CascadeClassifier(分類器文件)

- 辨識目標：

結果變數=控制變數.detectMultiScale(

圖像變數,

minNeighbors=檢測門檻數,

minSize=最小尺寸

)

(寬, 高)

整數值，如：1, 2, 3等

- 結果變數會是個二維矩陣：

- 第一維：每一個辨識到的目標
- 第二維：每一個目標的X座標、Y座標、寬和高

# 作業4

- 訓練自己的分類器文件：
  1. 訓練自己的分類器文件。
  2. 完成後將建立好的分類器文件以及作為訓練用的照片繳交上傳到系統上。