多媒體程式設計
影像資料處理

Instructor: 馬豪尚

物件偵測

邊緣檢測

- 邊緣檢測的目的是標識圖像中亮度變化明顯的點
- 圖像中的顯著變化通常反映了屬性的重要事件
  - 深度上的不連續
  - 表面方向不連續
  - 物質屬性變化
  - 場景照明變化
- 邊緣計算通常尋找圖像梯度在梯度方向上的極值點
- 只要有這個極值點出現，那就是邊緣

OpenCV邊緣檢測

- openCV提供三種邊緣檢測方式來處理
  - Laplacian、Sobel及Canny
  - 針對灰階的影像，基於每個像素灰度的不同，利用不同物體在其邊界處會有明顯的邊緣特徵來分辨

OpenCV Laplacian邊緣檢測

- cv2.Laplacian(img, ddepth, ksize, scale)
  - 有四個輸入參數
    - img為影像物件
    - ddepth 輸出影像深度，設定 -1 表示使用圖片原本影像深度
    - ksize 運算區域大小，預設 1 (必須是正奇數)
    - scale 縮放比例常數，預設 1 (必須是正奇數)
  - 針對「灰階圖片」，使用拉普拉斯運算子進行偵測邊緣的轉換
    - cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

OpenCV Laplacian邊緣檢測

- 由於Laplacian透過計算零交越點上光度的二階導數，對於雜訊（Noise）非常敏感
- 實務上要做Laplacian邊緣偵測會將影像先模糊化去除雜訊後再做處理
  - cv2.medianBlur(img, 7)

OpenCV Laplacian邊緣檢測

- 使用此函數除了傳入灰階影像之外，亦須指定輸出的影像浮點格式CV_64F，才能保留所有的邊緣資訊
  - 因為Laplacian過程需進行black-to-white及white-to-black兩種轉換，在微分的梯度計算（gradient）中black-to-white屬於正向的運算而white-to-black則是負向，灰階的8bits格式僅能儲存0-255的正值，因此建議使用64bits
  - cv2.Laplacian(img, cv2.CV_64F)
- 先輸出為64bit，再取絕對值轉為8bit
  - np.uint8(np.absolute(img))

OpenCV Sobel 邊緣檢測

- cv2.Sobel(img, ddepth, dx, dy, ksize, scale)
  - 有六個輸入參數
    - img 影像物件
    - ddepth 影像深度，設定 -1 表示使用圖片原本影像深度
    - dx 針對 x 軸抓取邊緣，設定為1或0
    - dy 針對 y 軸抓取邊緣，設定為1或0
    - ksize 運算區域大小，預設 1 (必須是正奇數)
    - scale 縮放比例常數，預設 1 (必須是正奇數)

OpenCV Sobel 邊緣檢測

- 可以單獨針對X軸、Y軸或X與Y軸抓取其邊緣
  - cv2.Sobel(image, cv2.CV_64F, 1, 0)
  - cv2.Sobel(image, cv2.CV_64F, 0, 1)
  - cv2.Sobel(image, cv2.CV_64F, 1, 1)
- 也需要先輸出為64bit，再取絕對值轉為8bit
  - np.uint8(np.absolute(img))

OpenCV Canny 邊緣檢測

- cv2.Canny(img, threshold1, threshold2, apertureSize)
  - 輸入四個參數
    - img 影像物件
    - threshold1 門檻值，範圍 0～255
    - threshold2 門檻值，範圍 0～255
    - apertureSize 計算梯度的 kernel size，預設 3
  - 傳入影像參數並指定兩個門檻參數threshold1與threshold2
  - 圖形的任一點像素，若其值大於threshold2，則認定它屬於邊緣像素，若小於threshold1則不為邊緣像素，介於兩者之間則由程式依其像素強度值運算後決定

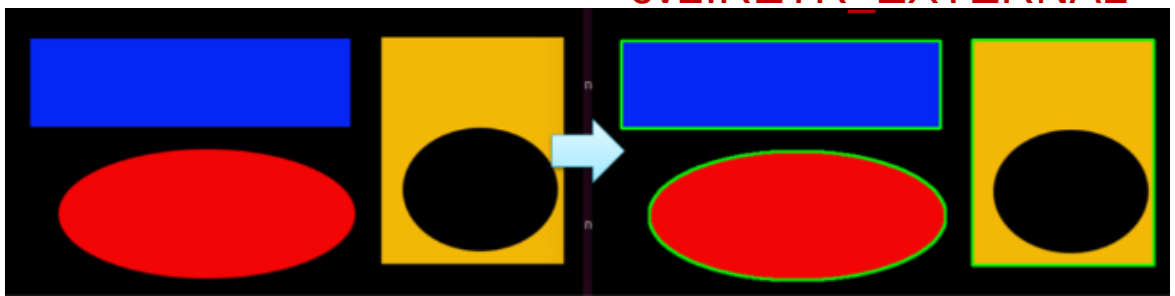
OpenCV 物件輪廓檢測

- 邊緣 vs 物件輪廓
  - 邊緣指的是圖像中像素的值有突變的地方
  - 物體間的輪廓指的是現實場景中的存在於物體之間的邊界
  - 有可能有邊緣的地方並非物體邊界，也有可能邊界的地方並無邊緣
- 在openCV的世界裏， 若邊緣線條頭尾相連形成封閉的區塊，那麼它就是輪廓，否則就只是邊緣

OpenCV 物件輪廓檢測

- cnts, hierarchy = cv2.findContours(img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
  - 輸入三個參數
    - 第一個參數為影像物件，通常是已經做完邊緣檢測的影像
    - 第二個參數為要取得物件外部輪廓或所有輪廓
      - cv2.RETR_LIST表示我們要取得圖形中所有的輪廓
      - cv2.RETR_EXTERNAL表示只取外層的輪廓
    - 第三個參數為代表壓縮取回的輪廓像素點
      - cv2.CHAIN_APPROX_SIMPLE代表只取回長寬及對角線的點，較快較省空間
      - cv2.CHAIN_APPROX_NONE代表傳回輪廓所有點

cv2.RETR_EXTERNAL



OpenCV 物件輪廓檢測

- findContours會回傳兩個結果
  - 第一個為物件輪廓的點座標cnts
  - 第二個為物件階層資訊
- 搭配繪製圖形，可以達到將物件標示出來的效果
  - cv2.boundingRect可取出輪廓的圖形，該函數會回傳每個輪廓左上角的坐標值及長寬值
    - (x, y, w, h) = cv2.boundingRect(c)
  - 將回傳的物件輪廓cnts座標繪製成為一個圖形
    - cv2.rectangle(img, (x,y), (x+w, y+h), (0,255,0), 2)

OpenCV 物件輪廓檢測

- 套用到較為複雜的實際相片，往往都不是我們希望的結果
  - 可能會檢測到很多很小的區域的輪廓，透過計算區域面積大小來忽略太小的區域輪廓
    - (cnts2, _) = contours.sort_contours(cnts)
    - cnts2 = [i for i in cnts2 if cv2.contourArea(i) > 100]

物件辨識

- 物件辨識大略上有以下幾個重要的過程
  - 物件偵測
    - 抓出影像中的物件
  - 特徵值擷取
    - 擷取物件中的特徵
  - 識別物件
    - 根據特徵辨識該物件

物件辨識例子

- 人臉辨識的重要流程
  - 偵測人臉物件
  - 擷取人臉特徵並訓練模型
  - 做特徵比對，判斷使用者身分
  - 應用在系統上
    - 門禁系統
    - FaceID

OpenCV人臉偵測

- face = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")
  - CascadeClassifier這個方法是一個分類器，可以根據所提供的模型檔案，判斷某個事件是否屬於某種結果
  - haarcascade_frontalface_default.xml為輸入的模型，設定用來辨識的分類器為人臉的模型

OpenCV人臉偵測

- faces_out = face.detectMultiScale(img, scaleFactor, minNeighbors, flags, minSize, maxSize)
  - 輸入五個參數
    - img 影像物件，建議使用灰階影像
    - scaleFactor 前後兩次掃瞄偵測畫面的比例係數，預設 1.1
    - minNeighbors 構成檢測目標的相鄰矩形的最小個數，預設 3
    - flags 通常不用設定
    - minSize, maxSize 限制目標區域的範圍，通常不用設定

OpenCV人臉偵測

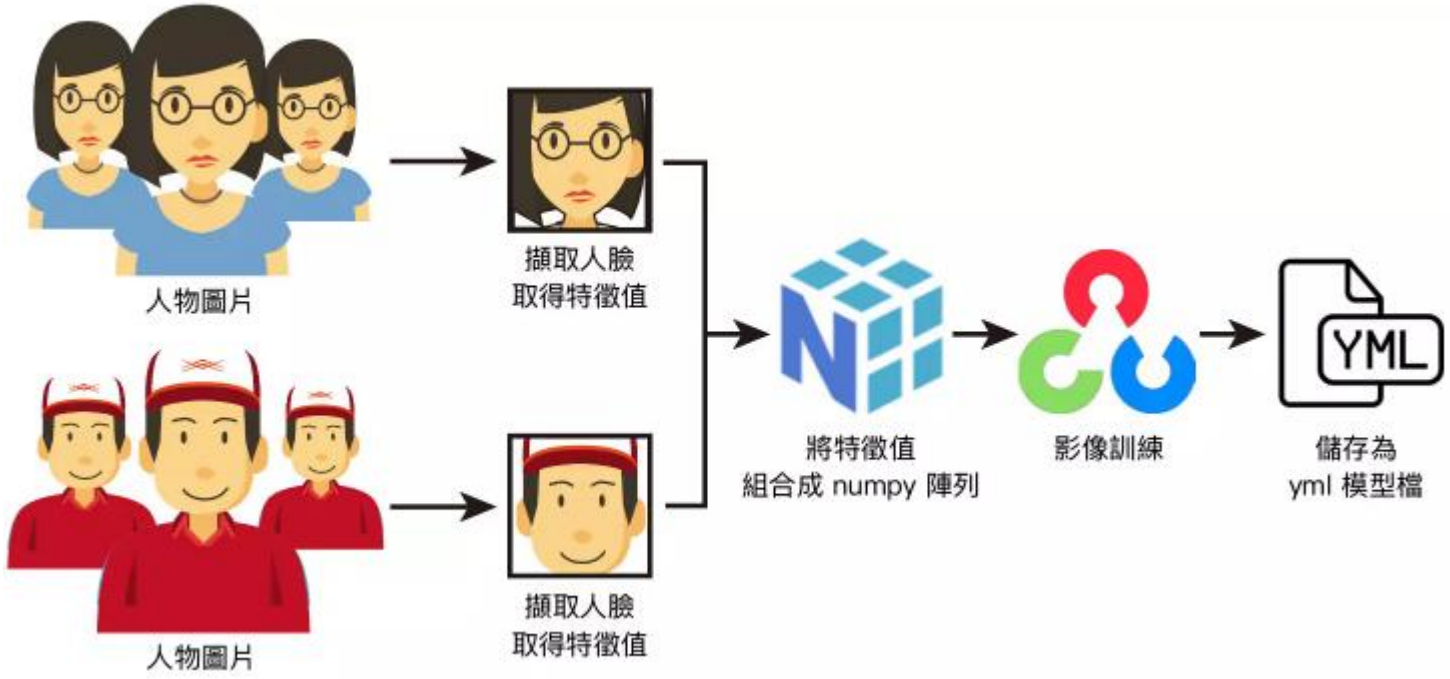
- detectMultiScale()
  - 會回傳多組的tuple(x, y, w, h)分別代表多個偵測到的人臉物件座標
  - x,y分別為物件的左上角的x和y軸座標值
  - w,h分別為物件的寬和高的值
- 搭配繪製圖形，可以達到將物件標示出來的效果
  - 將回傳的(x, y, x+w, y+h)座標繪製成為一個矩形

OpenCV 各種物件偵測模型

- https://github.com/opencv/opencv/tree/4.x/data
- 人臉五官偵測
  - 眼睛模型 → haarcascade_eye.xml
  - 嘴巴模型 → haarcascade_mcs_mouth.xml
  - 鼻子模型 → haarcascade_mcs_nose.xml
- 交通物件偵測
  - 汽車模型 → cars.xml
  - 行人(人體)模型 → haarcascade_fullbody.xml
- 動物物件偵測
  - 貓臉模型→ haarcascade_frontalcatface.xml

OpenCV人臉辨識

- 安裝模組
  - pip install opencv_contrib_python
- 訓練人臉辨識模型



OpenCV人臉辨識

- 訓練人臉辨識模型
  - 針對同一個人先蒐集多張影像，至少要蒐集兩個人以上才有辨識的意義
  - 輸入蒐集的圖片來訓練模型
- 輸入一張影像辨識是否為該使用者
  - 載入已訓練的模型
  - 擷取影像中的人臉物件
  - 比對該人臉物件和已訓練模型的特徵來判定是否為同一個人

OpenCV人臉辨識實作

- 擷取人臉物件
  - 將輸入影像色彩轉換成灰階
    - gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
  - 擷取人臉物件區域
    - face = detector.detectMultiScale(gray)
  - 記錄每一張圖片人臉物件並且標記是誰
    - (x,y,w,h)為人臉物件座標
    - tag=標記

OpenCV人臉辨識實作

- 訓練人臉辨識模型
  - 宣告模型變數
    - recog = cv2.face.LBPHFaceRecognizer_create()
  - 輸入影像和人臉物件區域進行訓練
    - recog.train(faces,np.array(ids))
    - faces為輸入的影像物件，多個影像物件儲存成一個array
    - ids為相對應影像物件是誰的標記，多個標記儲存成一個array
  - 儲存模型
    - recog.save('face.yml')

OpenCV人臉辨識實作

- 載入模型
  - recognizer = cv2.face.LBPHFaceRecognizer_create()
  - recognizer.read('face.yml')
- 根據模型辨識輸入影像的標記
  - idnum,confidence = recognizer.predict(gray[y:y+h,x:x+w])
    - gray[y:y+h,x:x+w]是輸入的人臉物件
    - idnum為模型辨識的標記
    - confidence為模型信心指數

練習/作業二

- 請完成一個人臉辨識的系統
  - 蒐集至少兩個人圖片(一人20張)
  - 訓練人臉辨識模型
    - 降低雜訊
    - 轉灰階並偵測人臉物件
    - 輸入模型訓練
  - 使用訓練好的模型來辨識
    - 輸入其他的(沒有用來訓練)影像來測試模型準確度
    - 將這些影像辨識的結果儲存起來
      - 影像、物件標示框、辨識結果(標示為誰)