Computer Vision I _2018

Homework assignment #5

R07522717 機械所製造組碩一 林温雅

```
#使用 python
#import 套件
import cv2
import numpy as np

def GrayScale_Dilation(img, ker):
    #獲得輸入圖檔之行列數
    img_rows, img_columns = img.shape
    #獲得 kernel 之行列數
    ker_rows, ker_columns = ker.shape
    #計算 kernel 中心距離邊界有多遠，為的是擴大原始圖檔，方便後續迴圈處理
    row_dist, column_dist = int((ker_rows-1)/2), int((ker_columns-1)/2)
    #根據上述計算，製作一個比原始圖檔大的暫存圖檔，以 img 為 512*512, kernel 為 5*5 來說，暫存圖檔為 516*516，暫存圖檔為往上、往下、往左、往右分別外擴兩列/行，外擴新增的 pixel 值另為 0，中間則就是原本輸入圖檔的值

    #dilation 要找最大的，所以外擴的填 0
    temp_img = np.zeros((img_rows+2*row_dist, img_columns+2*column_dist), np.int)
    temp_img[row_dist:img_rows+row_dist, column_dist:img_columns+column_dist] = img
    #製作一個新圖檔準備接受 dilation 後的圖
    #為了 for 迴圈裡面 index 好寫，這邊一樣把 new_img 改成擴大後的，之後再來裁，和 hw4 做法有一點點不一樣
    new_img = np.zeros((img_rows+2*row_dist, img_columns+2*column_dist), np.int)

    #為了矩陣相乘，先 flip kernel，erosion 不用這樣
    kernel_flip = np.flip(ker)
```

```
    #進行 dilation 計算
    for i in range(row_dist, img_rows+row_dist):
        for j in range(column_dist, img_columns+column_dist):
            new_img[i, j] = np.nanmax(temp_img[i-row_dist:
i+row_dist+1, j-column_dist: j+column_dist+1]+kernel_flip)
    new_img = new_img[row_dist:img_rows+row_dist,
column_dist:img_columns+column_dist]

    return new_img

def GrayScale_Erosion(img, ker):
    #獲得輸入圖檔之行列數
    img_rows, img_columns = img.shape
    #獲得 kernel 之行列數
    ker_rows, ker_columns = ker.shape
    #計算 kernel 中心距離邊界有多遠，為的是擴大原始圖檔，方便後續迴圈
處理
    row_dist, column_dist = int((ker_rows-1)/2), int((ker_columns-1)/2)
    #根據上述計算，製作一個比原始圖檔大的暫存圖檔，以 img 為 512*512,
kernel 為 5*5 來說，暫存圖檔為 516*516，暫存圖檔為往上、往下、往左、往
右分別外擴兩列/行，外擴新增的 pixel 值另為 0，中間則就是原本輸入圖檔的值

    #erosion 要找最小的，所以外擴的填 255
    temp_img = 255 * np.ones((img_rows+2*row_dist,
img_columns+2*column_dist), np.int)
    temp_img[row_dist:img_rows+row_dist,
column_dist:img_columns+column_dist] = img
    #製作一個新圖檔準備接受 dilation 後的圖
    #為了 for 迴圈裡面 index 好寫，這邊一樣把 new_img 改成擴大後的，之
後再來裁，和 hw4 做法有一點點不一樣

    new_img = 255*np.ones((img_rows+2*row_dist,
img_columns+2*column_dist), np.int)
```

```
    #進行 erosion 計算
    for i in range(row_dist, img_rows+row_dist):
        for j in range(column_dist, img_columns+column_dist):
            new_img[i, j] = np.nanmin(temp_img[i-row_dist:
i+row_dist+1, j-column_dist: j+column_dist+1]-ker)

    new_img = new_img[row_dist:img_rows+row_dist,
column_dist:img_columns+column_dist]
    return new_img

def GrayScale_Opening(img, ker):
    return GrayScale_Dilation(GrayScale_Erosion(img, ker), ker)

def GrayScale_Closing(img, ker):
    return GrayScale_Erosion(GrayScale_Dilation(img, ker), ker)


original_img = cv2.imread('lena.bmp', 0)

###製作 kernel###
#dilation, erosion, opening, closing 要用的 kernel
kernel = np.array([[np.nan,0,0,0,np.nan], [0,0,0,0,0], [0,0,0,0,0], [0,0,0,0,0],
[np.nan,0,0,0,np.nan]])


###輸出圖片###
#輸出 dilation 圖片
cv2.imwrite('gray_scale_dilation_lena.bmp',
GrayScale_Dilation(original_img, kernel))

#輸出 erosion 圖片
cv2.imwrite('gray_scale_erosion_lena.bmp',
GrayScale_Erosion(original_img, kernel))

#輸出 opening 圖片
```

```
cv2.imwrite('gray_scale_opening_lena.bmp',
GrayScale_Opening(original_img, kernel))

#輸出 closing 圖片
cv2.imwrite('gray_scale_closing_lena.bmp',
GrayScale_Closing(original_img, kernel))
```