電腦視覺 HW4 report

D10922012 資工所 王傳啟

Write programs which do binary morphology on a binary image:

(a)~(d) 所共用的 kernel，存放為需要處理的座標，相對應到原點之位置

```
    kernel = np.array(
        [
            [-2, -1], [-2,  0], [-2,  1],
            [-1, -2], [-1, -1], [-1,  0], [-1,  1], [-1,  2],
            [ 0, -2], [ 0, -1], [ 0,  0], [ 0,  1], [ 0,  2],
            [ 1, -2], [ 1, -1], [ 1,  0], [ 1,  1], [ 1,  2],
            [ 2, -1], [ 2,  0], [ 2,  1]
        ]
    )
```

(a) Dilation:

做法是迴圈掃完整張圖，當遇到 pixel 為空白時，根據 kernel 的形狀與邊緣判斷拓展到原圖之中

```
def dilation(np_img, kernel):
    dilation_img = np.zeros((row, col), dtype=int)
    for i in range(row):
        for j in range(col):
            if np_img[i][j]: # pixel 為空白
                for k in kernel: #獲取其他要處理的位置
                    new_i = i + k[0]
                    new_j = j + k[1]
                    if new_i >= 0 and new_i < row and \
                        new_j >= 0 and new_j < col: # 若在邊緣內就強迫為白
                        dilation_img[new_i][new_j] = 255
    return dilation_img
```

(b) Erosion:

作法一樣是迴圈掃完整張圖，若 pixel 周遭能構成 kernel 的形狀才設定為白

```
def erosion(np_img, kernel):
    erosion_img = np.zeros((row, col), dtype=int)
    for i in range(row):
        for j in range(col):
            contained = True
            for k in kernel:
                new_i = i + k[0]
                new_j = j + k[1]
                if new_i < 0 or new_i >= row or \
                    new_j < 0 or new_j >= col or \
                    not np_img[new_i][new_j]:
                        contained = False
                        break
            if contained == True:
                erosion_img[i][j] = 255
    return erosion_img
```

(c) Opening:

Opening 就是對原圖執行 kernel 的 Erosion 再做 Dilation (物件變背景)

```
np_img_c = dilation(erosion(np_img, kernel), kernel)
```

(d) Closing:

Closing 就是對原圖執行 kernel 的 Dilation 再做 Erosion (背景變成物件)

```
np_img_d = erosion(dilation(np_img, kernel), kernel)
```

(e) Hit-and-Miss:

先根據講義建構 kernel j 與 k (也是相對座標的表示方式)

```
kernel_j = np.array([[ 0, -1], [ 0,  0], [1,  0]])
kernel_k = np.array([[-1,  0], [-1,  1], [0,  1]])
```

根據下述公式, 運算結果 (其中, A^c 為 A 的補集, 透過 255-原圖亮度 可得)

$$A \otimes (J, K) = (A \ominus J) \cap (A^c \ominus K)$$

```
def get_complement(np_img): # 獲取補集合
    row = np_img.shape[0]
    col = np_img.shape[1]
    np_img_complement = np.zeros((row, col), dtype=int)
    for i in range(row):
        for j in range(col):
            np_img_complement[i][j] = 255 - np_img[i][j]
    return np_img_complement

def hit_and_miss(np_img, kernel_j, kernel_k):
    row = np_img.shape[0]
    col = np_img.shape[1]
    res = np.zeros((row, col), dtype=int)
    img_A  = copy.deepcopy(np_img)
    img_Ac = get_complement(img_A) # 得到原圖的補集
    first_half  = erosion(img_A, kernel_j) # 前半部的括號
    second_half = erosion(img_Ac, kernel_k) # 後半部的括號
```

```
    for i in range(row):
        for j in range(col):
            # 取交集，有值就設為 255 (即白)
            if first_half[i][j] and second_half[i][j]:
                res[i][j] = 255
    return res
```