# DIP HW3

洪郡辰
R11944050

April 16, 2023

## 1 Problem 1

### 1.1 P1.a

(a) sample1 (b) result1

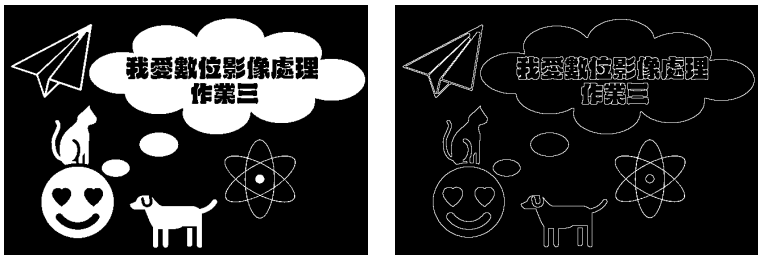


Figure 1: Problem1.a.sample and result

(a) 1 (b) 2 (c) 3

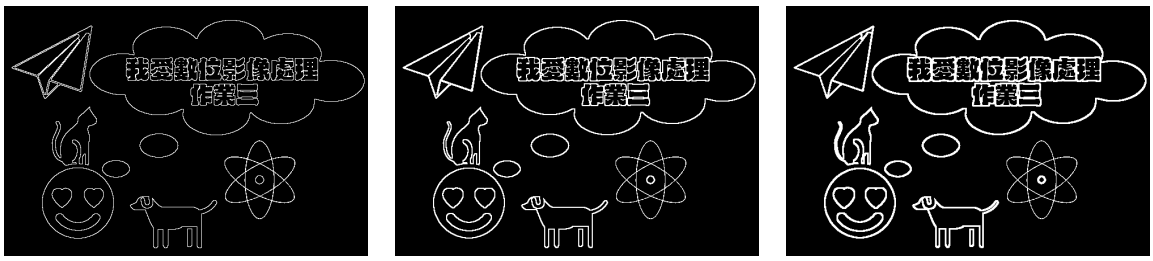


Figure 2: Problem1.a.different erosion steps

在 Fig.2 中的圖片分別為 erosion 次數為 1、2、3 的圖片。可以觀察到隨著次數增加，圖片中邊緣的輪廓會變粗。這代表可以透過設定 erosion 次數改變 boundary 線條的粗細。我選擇 erosion 次數為 1 的圖片作為我的 result。

## 1.2 P1.b

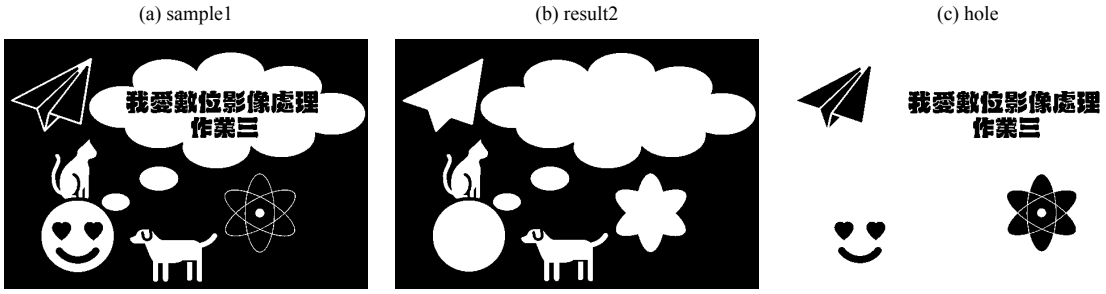


Figure 3: Problem1.b.sample and result

在 Fig.3 中是我的 result。我希望能先找到所有不被 object bound 的部分，然後剩下被認定為 background 的部分就是要填補的 hole。在比對過程中我設定一個比對點 list，將 [0,0] 加入作為初始點。我透過 1 個 3*3 的 G 對圖片進行檢測，G[1][1]=255，其餘為零。在與 structure 作用後，再根據比對點與原圖做比對。如果原圖比對到的位置為 0 且 G 中的點為 255，就將其設定為 255，並將此點加入比對的起始點 list 中。如果原圖為 255，則不到任何動作。直到所有比對點都比對過，比對即完成。在 Fig.3 中的 hole 是比對完成的結果。

比對完成後再進行 inverse，即能得到 hole 被設定為 object 的圖片。將此圖片與 sample1 做疊加，即能得到 result2。

## 1.3 P1.c

我透過 Two-Pass Connected-Component labeling 進行計算，我認定只要周圍八個方位有 object，即認定為同一個 component。

在我的演算法中，我記錄一個 label dict 及 label map，label dict 用來指出 label 所對應到的 component，label map 用來記錄每個位置的 label。如果不同 label 有相連發生時，label dict 會將其對應到相同的 min label，min label 也就代表一個 component。

在建立 label map 時，會分成兩種情形。第一種為沒有 label 為 negihbor，此時會定義一個新的 label，並將其加入到 label dict，key 及 value 皆為 label。第二種情形為有 label 為 neighbor，neighbor 只看目前位置的 N,W,NW 方位。此時會先找出 neighbor 中最小的 label，並將 neighbor 中的 label 在 dict 中所對應到的 value 都改為最小的 label，將其指向同一個 component。

在建立完成後，更新 label dict。確保所有 label 都有指向其 component，並建立一個 set，紀錄更新後的 label dict 中相異 value 的數量。set 的長度就代表有幾個 component。我最後的結果是 39 個 components。

## 1.4 P1.d

(a) sample1 (b) result3 (c) result4


Figure 4: Problem1.d.sample and result

(a) 3*3 (b) 5*5

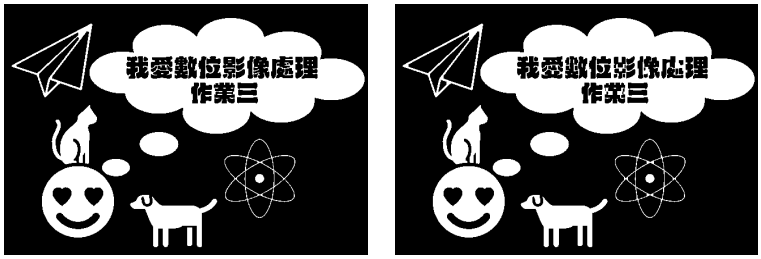


Figure 5: Problem1.d.different structure for close

(a) 3*3 (b) 5*5

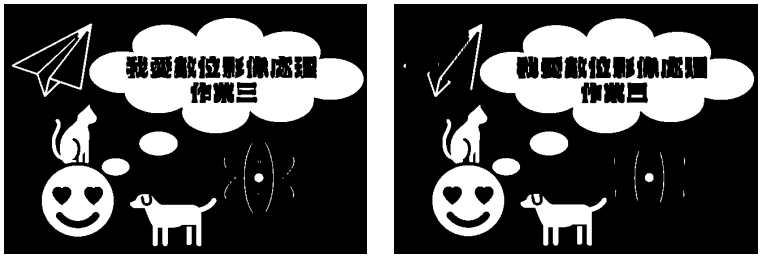


Figure 6: Problem1.d.different structure for open

structure element 會影響 apply open 或是 close operator 過後的圖片中沒有相連的部分是否會變成相連或是產生開口。

在 Fig.5 中我比較兩個不同 structure element 在 close 產生的影響。兩張圖片中的 structure element 都是在中心的 N 及 S 的方向設置為 1，其餘為零。一個為 3*3 另一個為 5*5。可以看到 3*3 的圖片中在狗的後腳部分及貓的後腳部分並未相連，跟原圖一樣。在 5*5 的圖片中狗的後腳部分有相連。從這兩個範例中能明顯看到改變 structure element 對 close 的差異。

在 Fig.6 中我比較兩個不同 structure element 在 open 產生的影響。兩張圖片中的 structure element 都是在中心的 N 及 S 的方向設置為 1，其餘為零。一個為 3*3 另一個為 5*5。可以看到 3*3 的圖片中在狗的耳朵部分及貓的後腳部分並未打通，跟原圖一樣。在 5*5 的圖片中狗的耳朵部分及貓的後腳有打通。從這兩個範例中能明顯看到改變 structure element 對 open 的差異。

# 2 Problem 2

## 2.1 P2.a

我使用三種一維 filter 交叉使用得到九個 mask。三個 filter 分別為 High-pass(HP)、Band-pass(BP)、Low-pass(LP)。在這九種 mask 下我能分別過濾掉圖片中不同的資訊，將剩下的資訊作為 feature，希望能從 feature 中進行 texture analysis。透過這九種 mask 得到圖片中不同面向的資訊，將這九種資訊作為 feature，我認為能夠有效地解析圖片。

得到九種 feature 後，我進行 energy computation，window width 為 13，我透過取 window 內的平均值作為 energy 形成用於 classification 的 feature map。完成 energy computation 代表有考慮每個 pixel 附近的情況，有將位置關係考慮進 feature，能更好地進行 texture classification。

## 2.2 P2.b

(a) sample2 (b) result5

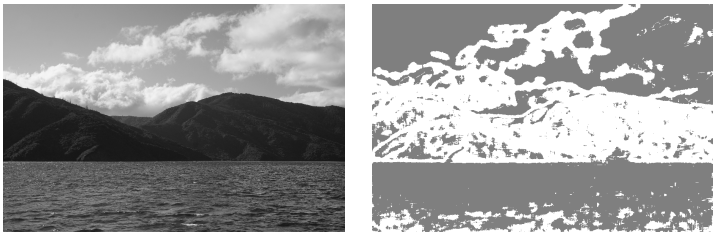

Figure 7: Problem2.b.sample and result

以下是我使用在 k-means alg. 中的 parameters：

- k: 篩選的紋理數量 (2)
- max iter.: 演算法要跑多少個 iterations(100)

在 P2.a 中求出的九種 feature map 分別代表圖片九種面向的特徵，我將相同位置的特徵形成一個 feature vector。將所有位置的 feature vector 形成一個 feature array，使用 k-means 找出 feature array 中的 representative 並將被歸類到同一個 representative 的 feature vector 視為同一種 texture。

(a) 10 (b) 100 (c) 500

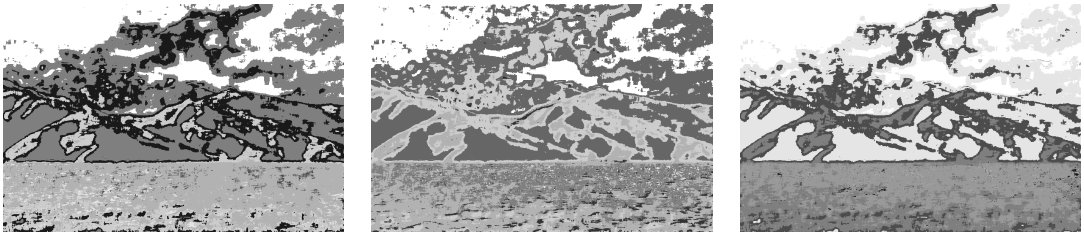

Figure 8: Problem2.b.different max.iter

在 Fig.8 中是我嘗試不同 max.iter 的結果，不同圖片中相同顏色的紋理不代表是一樣的紋理。可以看到不同 max.iter 對結果影響不大，這代表在這樣資料筆數下 k-means 不需要太高的 iter 就能很好地進行分類。

(a) 2 (b) 4 (c) 10

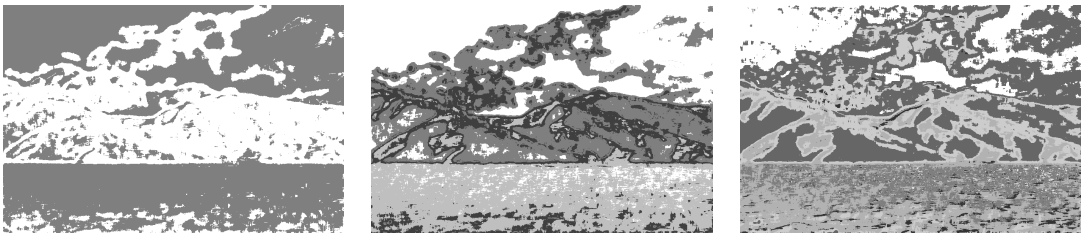

Figure 9: Problem2.b.different k

在 Fig.9 中是我固定 max iter. 為 100 嘗試不同 k 的結果，不同圖片中相同顏色的紋理不代表是一樣的紋理。可以看到不同 k 對結果影響在於會判定出有幾種 texture，我認為這個參數的設定必須基於 domain knowledge 或是實際用途。可以在 k=2 中看出圖片被分類成類似 object 及 background，白色部分被對應到比較接近雲朵及山的部分，深色部分被對應到以較接近藍天及海面的部分。在 k=4 中山峰處及及山與其他部分的交界線被分類成不同 texture。

## 2.3 P2.c

(a) sample2 (b) result5 (c) result6

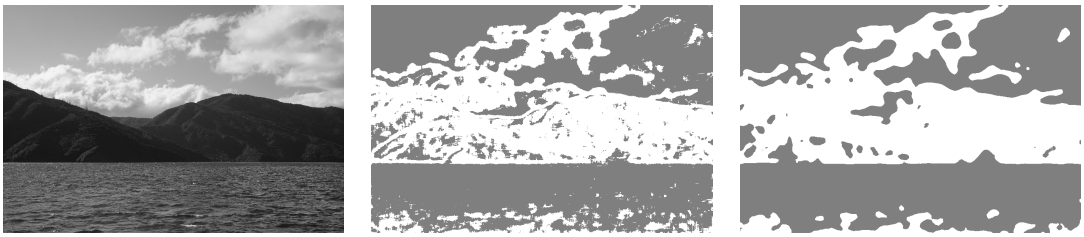

Figure 10: Problem2.c.sample and result

在觀察 result5 後，我認為可以透過 median filter 進行優化，因為我的圖片被分為 front 及 background，我認定雲朵及山是 front，湖面及藍天是 background。在 result5 中可以看到有許多 front 混在 background 中也有許多 background 混在 front 中，因此我認為可以透過 median filter 優化，減少這類現象。

(a) 5*5 (b) 11*11 (c) 21*21

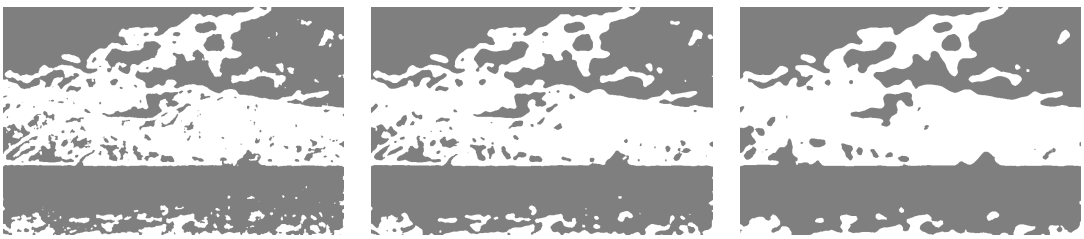

Figure 11: Problem2.b.different kernel size

在 Fig.11 中是不同 kernel size 的結果，可以看到不同 size 會造成不同程度的降噪，我目前假定是要做前後景分離，因此前後景混和的情況越少越好。在 kernel size 為 10*10 中我認為已經能夠達到很好的效果，我們可以輕鬆地透過 connect labeling 找出最大的 front，也就是山峰及雲朵的位置。