

# Problem 1 – Morphological Processing

## ORIGINAL

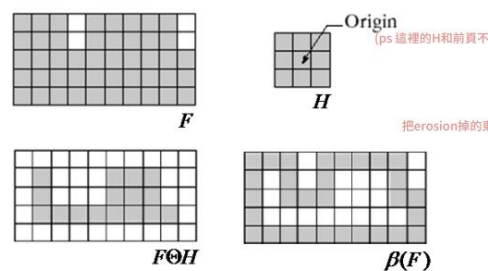


### (a) Boundary Extraction

<Step 1> Use morphological method to find the inner region of all the white pattern except for the edge

<Step 2> Subtract the original image by the result from <step 1>

$$\beta(F(j,k)) = F(j,k) - (F(j,k) \ominus H(j,k))$$



## RESULT



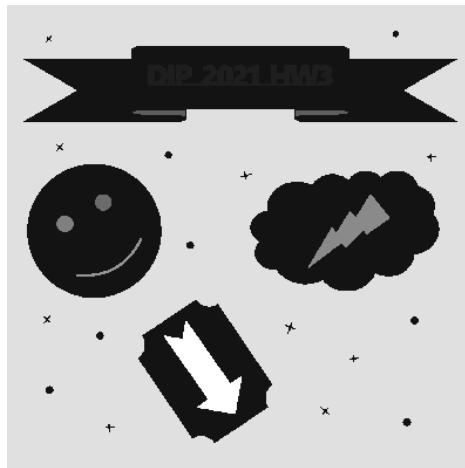
**(b) Hole filling**

這邊實作 hole filling 的方法和講義略有不同。因為 hole filling 必須先找到一個位在原圖 hole 的 starting point，再由該點做為起點進行 morphological processing。考量到找 starting point 需要先做 component labeling，而 hole filling 過程的操作和 component labelling 類似，(唯一差別是前者為 4-neighbor 的、後者是 8-neighbor，但後者也可以改做 4-neighbor)，兩步驟有所重疊，為了提高效率，本題決定先對整張圖做黑色 pixel 的 4-neighbor component label(白色作為背景)，將那些 hole 辨認成一個 component，直接將背景(排除包含 image[0, 0]的 component)以外的黑色 component 塗成白色。

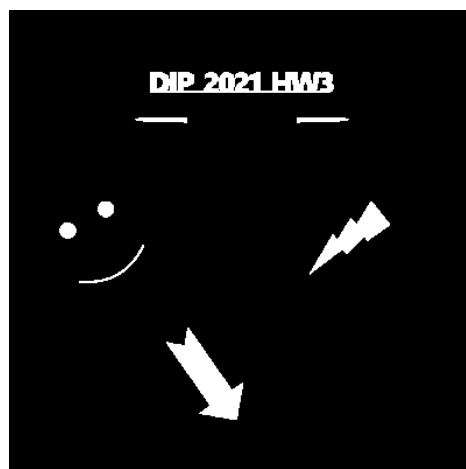
Component label 的作法在下一小題描述。

**<Step 1> Component-label (black object)**

如圖，我們將不同 object 以不同色度標示。黑色為背景。

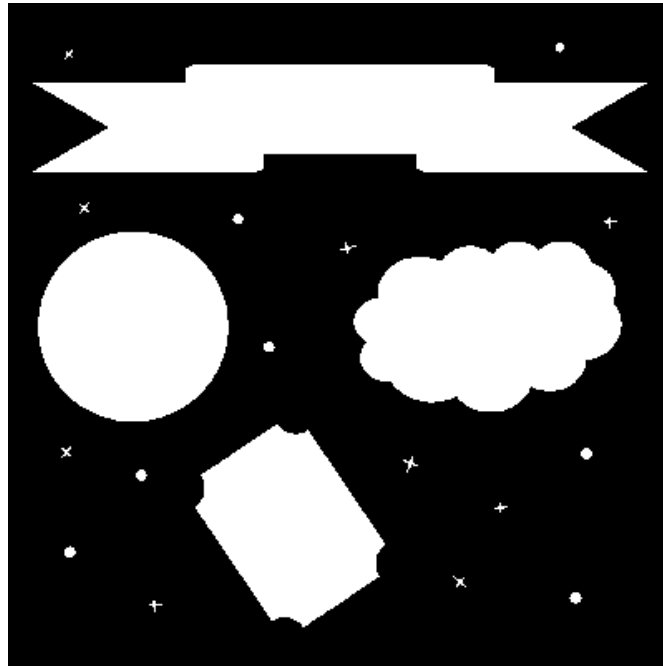
**<Step 2> Detect hole**

除了背景以外的 components 都是洞，以白色標出 object。



<Step 3> 原圖扣除 hole。

RESULT:



**(c) Component label**

這題要求我們算原圖有幾個 objects，我認為 object 和 component 的不同點在於：前者比較有主觀性，以圖中的旗子來說，我們不會認為那些空心文字獨立出來的色塊是一個新的 object，像是 D 裡面的白色色塊，他依然是旗子的一部份，所以這邊我定義 object 是被黑色背景包圍連續的白色區域（連續可以是 4-neighbor 和 8-neighbor connecting，前者跳過下述 step2 即可），而白色區域包圍的黑色區域不算背景，也不算一個新的 object，他是被包圍他的白色區域的一部份。

根據上述定義，我的步驟如下：

<Step 1> Hole filling

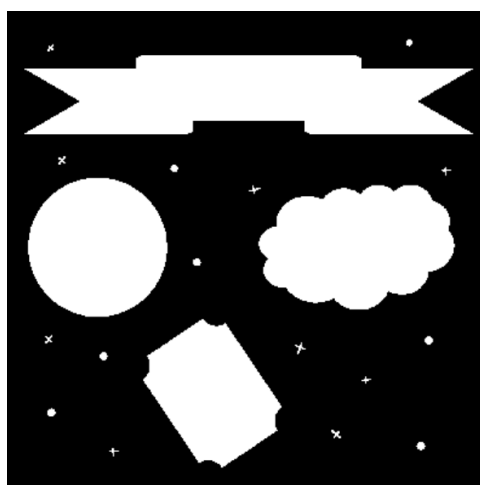
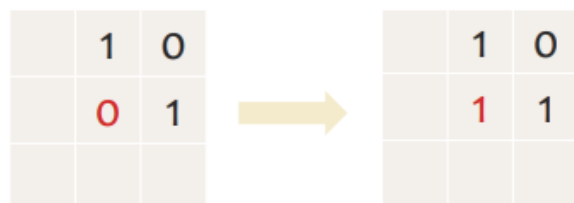
<Step 2> If using 8-neighbor, do the diagonal filling first

<Step 3> Component label and count the numbers

<Step 1> Hole filling 即上小題的結果

<Step 2>

使用下圖做四個方向的 additional operator。因為下一步我用的是 4 neighbor connect 的 component labelling，但如果想做 8 neighbor 的，左圖的就會被認為是兩個 component，所以將他相連。



Original



After diagonal fill

## &lt;Step 3&gt;

為了加快 Component label 的速度，演算法使用 sequential method 做 4 neighbor connect 的。流程如下：

定義 neighbor 為 pixel 上面(u)和左邊(l)的 pixel。

定義標籤為從 0 一直增長的整數數字。

(step 1) 掃描(scanning) p 從[0, 0]到[rows, cols]

若  $p = bg$  , p 標籤設為 -1 , 掃描下一點。

若  $p \neq bg$  ,

若  $u = l = bg$  , 給 p 一個新標籤。

若  $u \neq bg$  且  $l = bg$  , p 標籤和 u 一樣。

若  $u = bg$  且  $l \neq bg$  , p 標籤和 l 一樣。

若  $u \neq bg$  且  $l \neq bg$  :

若 u 標籤和 l 標籤一樣，把 p 也設定為該標籤。

若 u 標籤和 l 標籤不一樣，把 p 設定為其中一個標籤，並且標註 u 和 l 的那兩種標籤為同一種

(step 2)合併同類別(merging classes)

重新掃描，將上一個步驟所標示相連的標籤設定為同一個。

(step 3)重新標註

將現在有的標籤改用 sequential number 再將全部的標籤重新整理一次。

**RESULT**

4 connected neighbor: 32 個 object

8 connected neighbor: 20 個 object

左圖為 8 connected neighbor 計算結果，使用不同強度標出不同物品。

## Problem 2 – Texture Analysis

ORIGINAL:

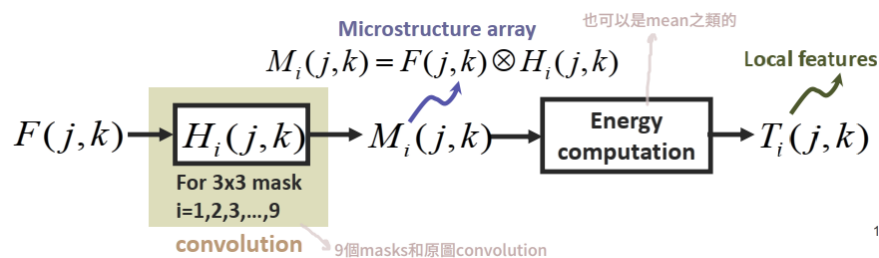


(a) Law's method to obtain the feature vector

下圖為 law's method 的流程：

<Step 1> Convolution

<Step 2> Energy computation



<Step 1>使用 9 種 mask 作為 9 種 texture 的卷積如下：

$$\begin{array}{ccc} \frac{1}{36} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} & \frac{1}{12} \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} & \frac{1}{12} \begin{bmatrix} -1 & 2 & -1 \\ -2 & 4 & -2 \\ -1 & 2 & -1 \end{bmatrix} \\ \text{Laws 1} & \text{Laws 2} & \text{Laws 3} \\ \frac{1}{12} \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} & \frac{1}{4} \begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix} & \frac{1}{4} \begin{bmatrix} -1 & 2 & -1 \\ 0 & 0 & 0 \\ 1 & -2 & 1 \end{bmatrix} \\ \text{Laws 4} & \text{Laws 5} & \text{Laws 6} \\ \frac{1}{12} \begin{bmatrix} -1 & -2 & -1 \\ 2 & 4 & 2 \\ -1 & -2 & -1 \end{bmatrix} & \frac{1}{4} \begin{bmatrix} -1 & 0 & 1 \\ 2 & 0 & -2 \\ -1 & 0 & 1 \end{bmatrix} & \frac{1}{4} \begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix} \\ \text{Laws 7} & \text{Laws 8} & \text{Laws 9} \end{array}$$

這幾個 law 是由一些像是平均、微分、二次微分的運算 **matrix** 合成。

$$L_3 = \frac{1}{6} \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} \quad E_3 = \frac{1}{2} \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \quad S_3 = \frac{1}{2} \begin{bmatrix} 1 & -2 & 1 \end{bmatrix}$$

**Local averaging**  
平均

**Edge detector**  
(1<sup>st</sup>-order gradient)  
微分

**Spot detector**  
(2<sup>nd</sup>-order gradient)  
二次微分

■ E.g.

$$L_3^T \otimes E_3 = \frac{1}{6} \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \otimes \frac{1}{2} \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} = \frac{1}{12} \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{Laws 2}^{18}$$

<Step 2> 定義 energy computation 為將整張圖的卷積結果做 2-norm。

**(b) k-means algorithm to classify each pixel**

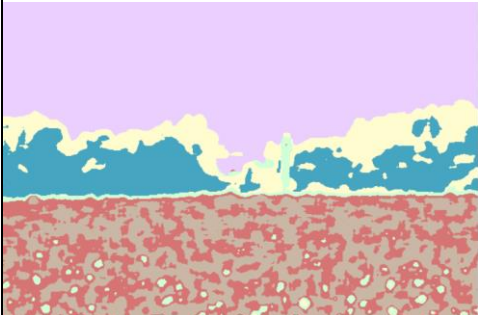
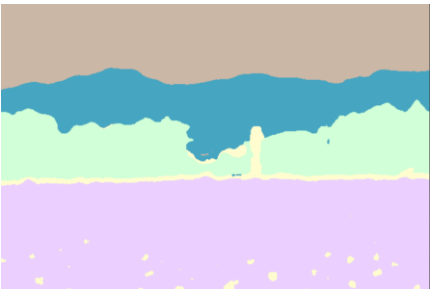
將 Law's method 跑出來的結果作為九維向量，使用 k-means algorithm 進行分類。k 個初始分類中心座標我使用 k 組 9 個 random 值，接著 k-means 會根據這個初始分類座標中心點，也就是將 pixel 和所有 k 個中心點做 2-norm，將其歸類為 2-norm 最小的那個分類。完成之後，再算分類後各類的中所有 pixels 的 mean 位子，作為下一次 k-means 的一個分類中心點，一直重複做 k-means 直到收斂(i.e.中心點位子不變。)

這題的 output 在後面一小題中表格裡的所有可能。

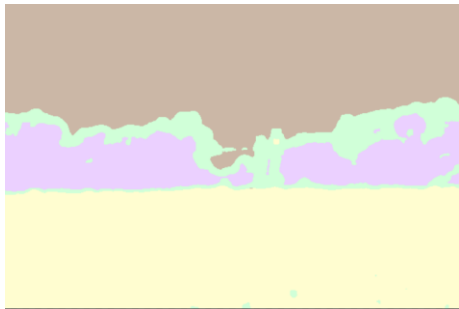
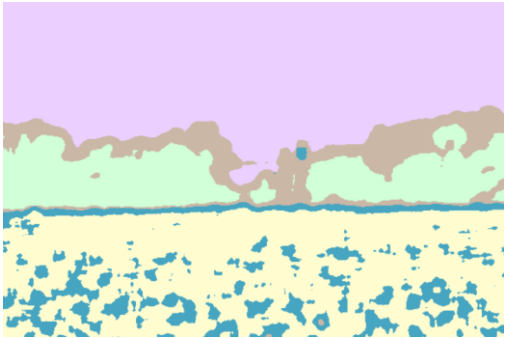
**(c) Improvement**

這題可以 improve 的有：k 的值、9 Law's method 出來的 energy 的 weighting、起始座標位子、windows size：

- (1) k 的值可以作為分幾類的依據，在這張圖片中我認為分成 5 類最適當：太陽花、樹、遠山、遠山在後面的更遠山、電線桿。k 的選擇其實蠻重要的，若選太小，可能會造成一些 pattern 沒被辨識出來。
- (2) Energy 的 weighting 可以根據比較想判斷出來的 pattern，將該特質的 weighting 加大，例如說可以加強 law-9 強化對於 spike 點的偵測。
- (3) 起始座標位子可以加速 k-means 的運算，也可以提高分類準確性，例如說不要用 random 值，而是根據原圖人工選取 k 個起始點。或者是做很多次 k-means，每次都紀錄該次 random 到的初始中心位子，再由人工的方法去看哪組初始中心位子得到的結果最好，將該值設成最佳初始中心位子。
- (4) Windows size 則可以決定 texture 的尺度，若將 window 開比較大，則一整片太陽花被當作一個 texture，若開的小，則太陽花中間的深色的花心會被辨識成另一種 pattern。

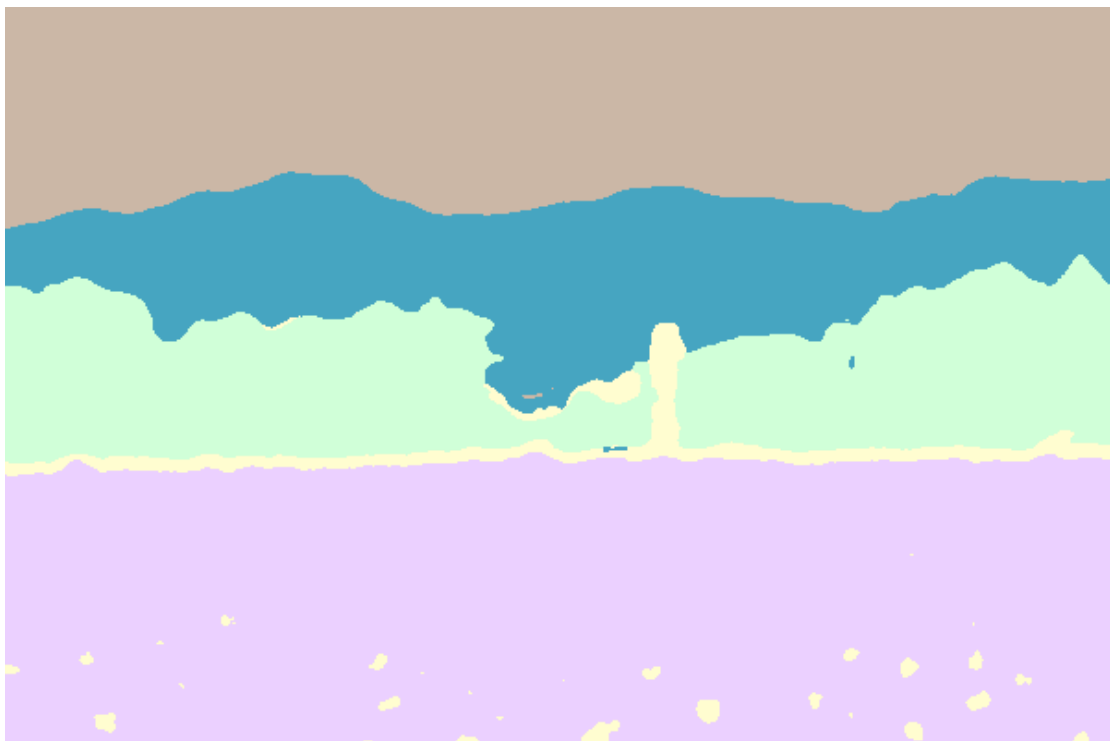
windows = 9, k = 6	windows = 13, k = 5
	
windows 較小可以分類出花芯	windows 較大則花海為同一種 texture



windows = 13, k = 4	windows = 13, k = 5
	
k 較小，可能導致分類結果不如預期。像這裡後山沒有被辨識出來，反而樹木陰影處被歸為一類。	初始中心標記點不好導致結果不佳。

**RESULT :**

這裡我放我最喜歡的 windows = 13, k = 5 的結果，搭配標記好的初始標記點。有順利將花海、電線桿、前山、後山分類出來。



**(d) Change pattern**

這題實作的方式，就是將 k-means 後的結果，將屬於花海那區的 texture 的標色的 pixel 的座標[i, j]，將 i、j 分別對 pattern\_picture 的寬、長取餘數，作為對應到 pattern\_picture 的 pixel，只是我發現會有對不齊的問題，所以可以再進一步先將 i, j 做一個 offset 再取餘數。

**RESULT :**