



DIP Course

# Homework Report week #3

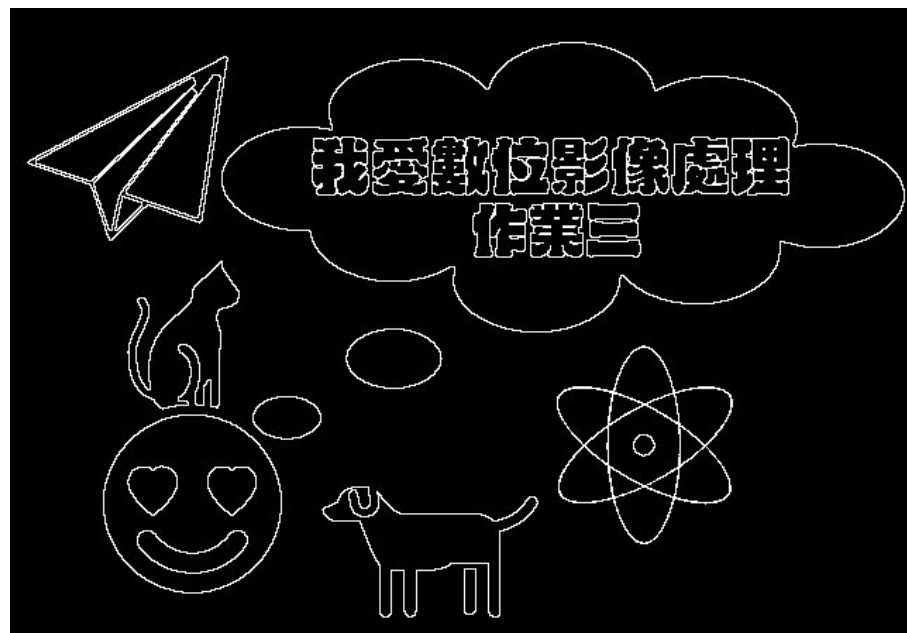
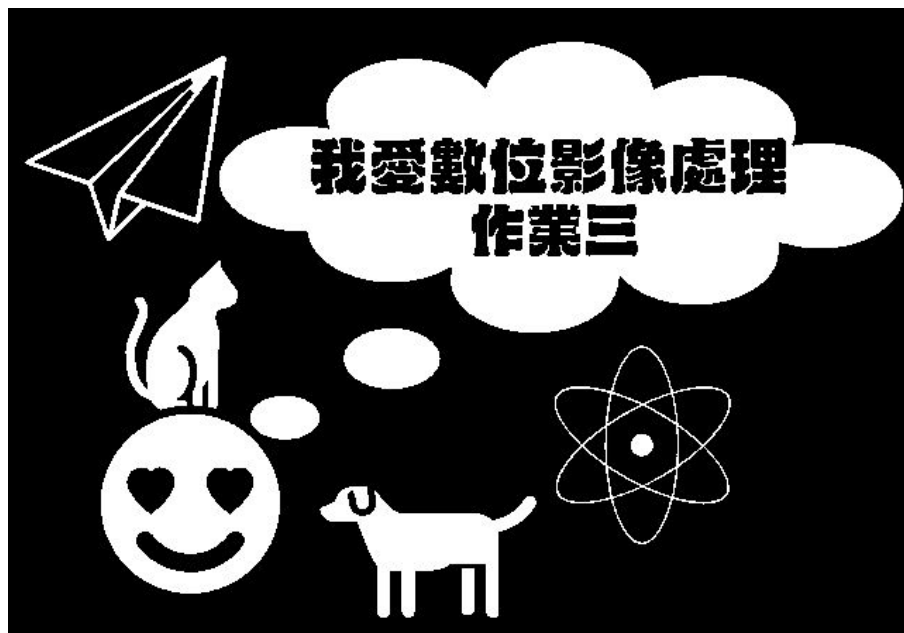
Student: p11922004 任祖頤

---

# Outline

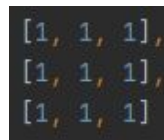
- MORPHOLOGICAL PROCESSING
- TEXTURE ANALYSIS

## Morphological Processing: extract

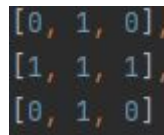


# Morphological Processing: extract

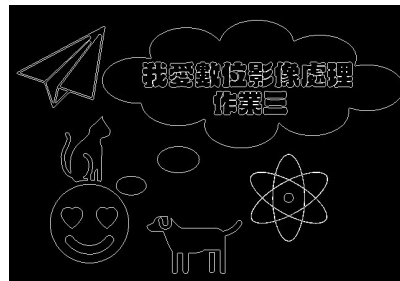
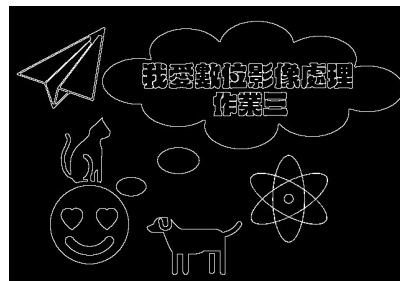
- 演算法
  - 因為 sample 圖片是 binary image, 這邊改成計算 filter 中數值的總計值來簡化運算成本, 加總數 值有超過 threshold 就會上白色, 之後再跟原圖相減得到邊界
  - 這個演算法適合給 8-neighbor 的 filter 使用, 4-neighbor 因為會分不清楚 0-1 的位置, 因此要使用傳統的 dilation
- 觀察
  - 右圖可看出 8-neighbor 對邊界的要求比較寬鬆, 因此得到的邊也更寬



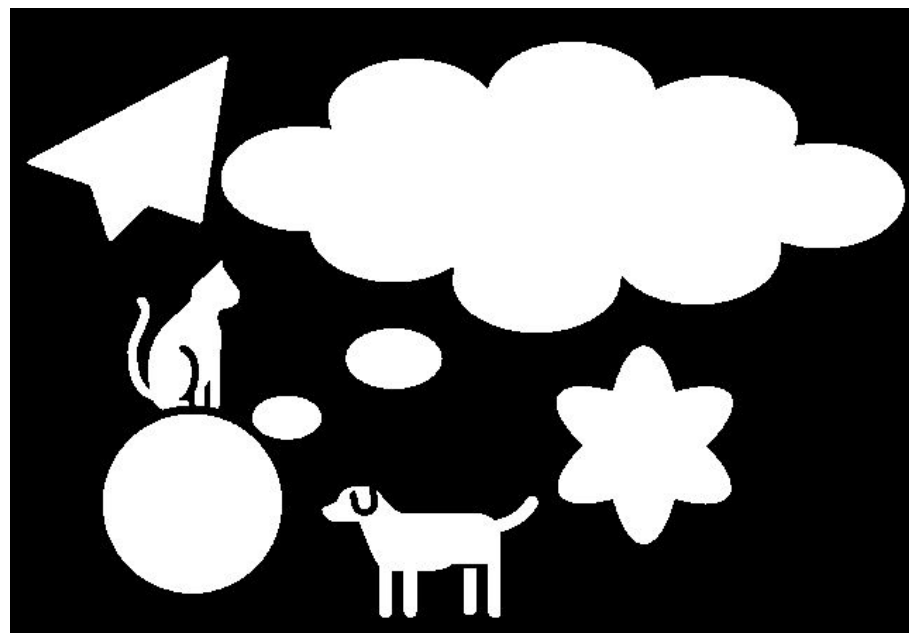
[1, 1, 1]  
[1, 1, 1]  
[1, 1, 1]



[0, 1, 0]  
[1, 1, 1]  
[0, 1, 0]



## Hole Filling



# Hole Filling

filled\_img



invert\_filled\_img(b)

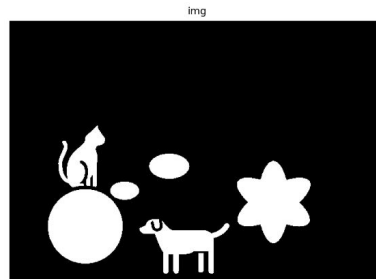
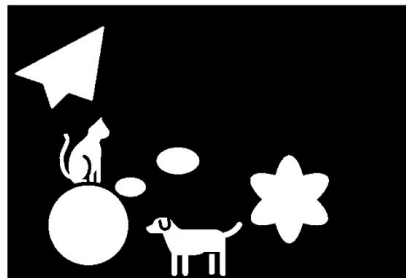
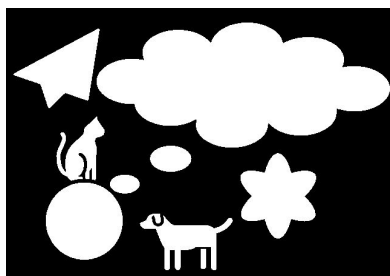


$G = F - b$

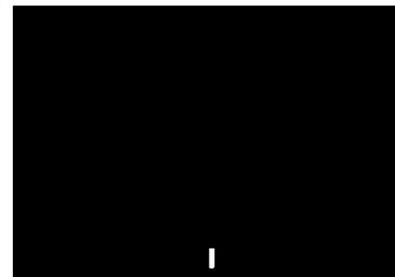
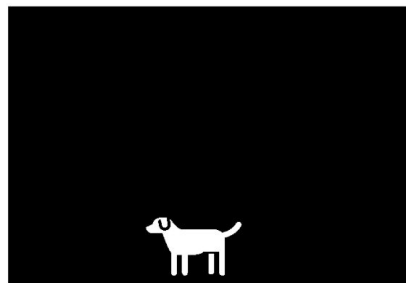


- 定義
  - 我這邊是把黑色背景 (第1層) 遇到的第一層白色 (2) 視為邊界, 再往裡面的黑色 (3) 是要作 Hole filling 的內容。如果還有再往裡面的白色 (4) 跟黑色 (5) 則不動
  - (5) 的情況在這張範例圖中並沒有出現, 因此畫出來的圖 內容會變全白
- 操作步驟
  - 先對畫面左上角作 hole filling, 接著將 filling 的結果反轉, 在用原圖減去反轉圖得到結果
- 討論
  - 本來想寫一個演算法來找出所有跟背景相連的物件逐一作 Hole Filling, 但用傳統的方法作迴圈太消耗運算資源, 一直 crash, 最後才想出上面這個做法

## Count the number of objects



...



Total number: 9

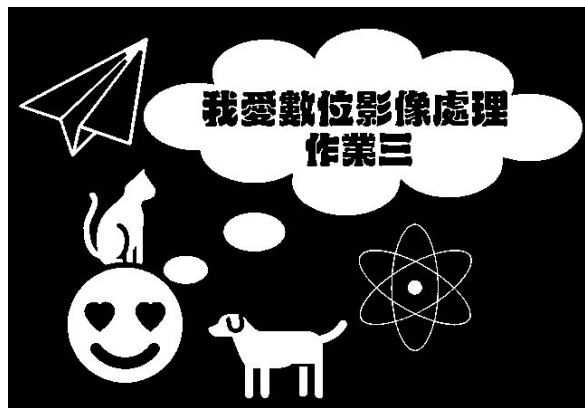


# Count the number of objects

- 實作步驟
  - 每個點逐一尋訪，看到白點就作 hole filling 成黑色並計算 1 次，直到所有點都被看完為止
- 討論
  - 因為要使用 hole filling 的技巧來計算 object 數量還是得要調整上一步的演算法，最後參考 <https://github.com/dani-amirtharaj/ImageSegmentation-Clustering-MorphologicalProcessing> 看到一個一次 loop 就能調整完的作法，便參考這個來實作，速度差超級多（應該有幾十倍）



## Apply open operator and close operator



open operator



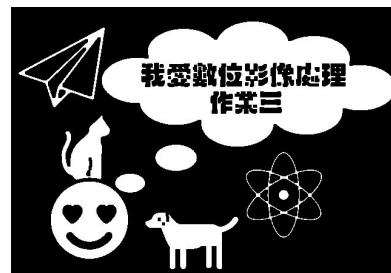
close operator

```
[[1. 1. 1. 1. 1.]  
 [1. 1. 1. 1. 1.]  
 [1. 1. 1. 1. 1.]  
 [1. 1. 1. 1. 1.]  
 [1. 1. 1. 1. 1.]]
```

applied structure element

# Apply open operator and close operator

- 實作步驟
  - open operator
    - 以白色的部分當作有內容，然後將 structure element 放到白色區域中滾動，可以正常放入的區域就會在結果顯示白色，無法進去的區域則為黑色，如右上圖
  - close operator
    - 這邊反過來將 structure 置於黑色區域滾動，可以正常滾動的區域維持黑色，其餘無法到達的區域則為白色，如右下圖



## Law's method



## Law's method: structure elements

```
[[1, 2, 1], #1  
 [2, 4, 2],  
 [1, 2, 1]],
```

```
[[1, 0, -1], #2  
 [2, 0, -2],  
 [1, 0, -1]],
```

```
[[ -1, 2, -1], #3  
 [-2, 4, -2],  
 [-1, 2, -1]],
```

```
[[ -1, -2, -1], #4  
 [0, 0, 0],  
 [1, 2, 1]],
```

```
[[1, 0, -1], #5  
 [0, 0, 0],  
 [-1, 0, 1]],
```

```
[[ -1, 2, -1], #6  
 [0, 0, 0],  
 [1, -2, 1]],
```

```
[[1, -2, 1], #7  
 [-2, 4, -2],  
 [-1, -2, -1]],
```

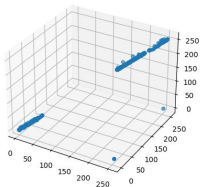
```
[[ -1, 0, 1], #8  
 [2, 0, -2],  
 [-1, 0, 1]],
```

```
[[1, -2, 1], #9  
 [-2, 4, -2],  
 [1, -2, 1]]
```

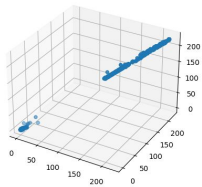
這邊使用課本所提供的filters

# Law's method: feature distribution

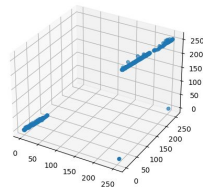
feat: [0, 1, 2]



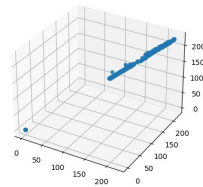
feat: [3, 6, 7]



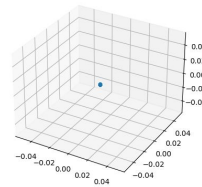
feat: 1, 2, 3



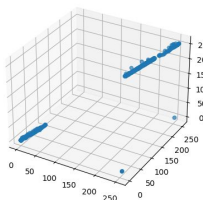
feat: [2, 4, 6]



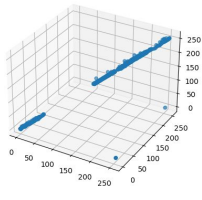
feat: [2, 5, 8]



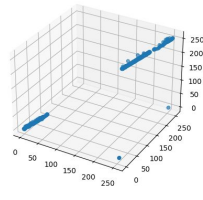
feat: [0, 1, 5]



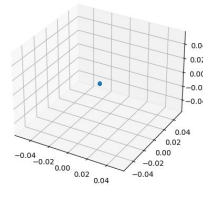
feat: [0, 1, 6]



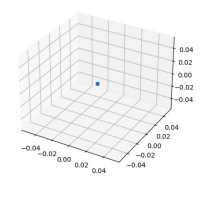
feat: [0, 1, 8]



feat: [1, 2, 4]



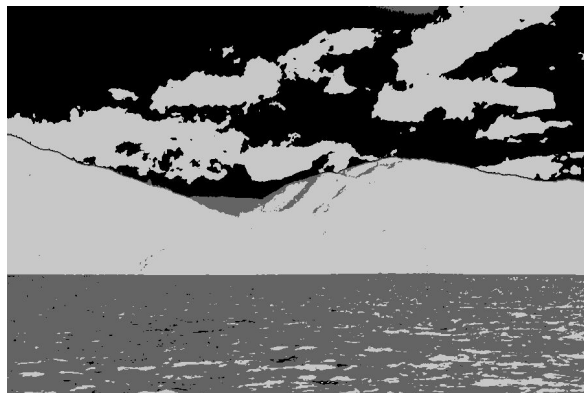
feat: [2, 4, 5]



上面各圖是選擇不同 feature 時產生的 3d point distribution

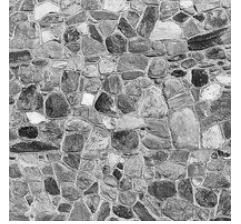
## Law's method: results

- 討論
  - 從上一頁的視覺圖中可以看出, 類似 1st-derivative 或 2nd-derivative 的 structure element, 其求出來的 feature 都趨近於 0, 在 3d 空間中看不出什麼差異, 目前沒有找到比較適合的使用方式
  - 最後取 visualization 後三群分布比較明顯一點的 0, 6, 7 來作 k-mean clustering
  - k=3 時雲跟山不能有效區分 (右上圖)  
但在 k=4 時就能分出差異 (右下圖)





# Bonus

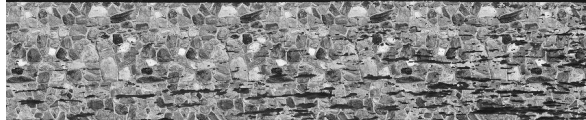
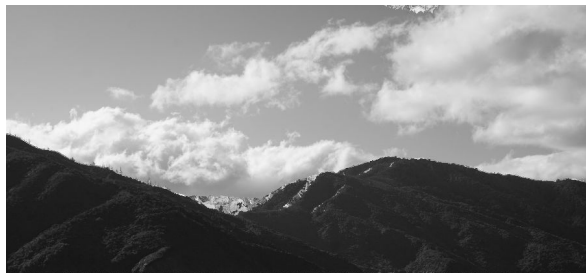
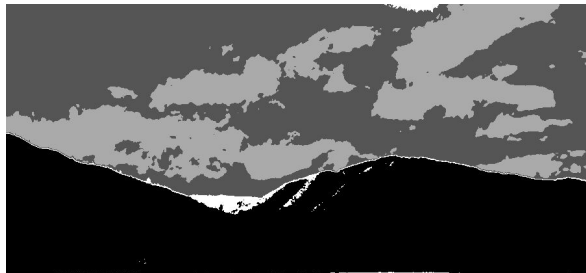


## Bonus

- 討論

- 這邊使用前一個步驟所得到的海平面區域 ( $p=255$ ), 將其替換為 sample3.png 的石板圖
- 石板圖的取樣方式為

```
sample3[y % sample3.shape[0]][x % sample3.shape[1]]
```





---

**Thank you**