DIP Report HW3

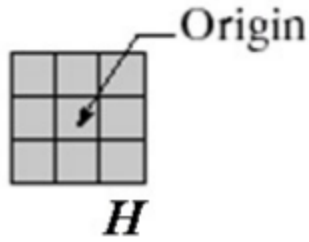
Problem 1: MORPHOLOGICAL PROCESSING

A binary image, sample1.png, is given in Figure 1. Please implement several morphological operations to meet the following requirements and provide discussions on each of the results. (Note that the white pixels represent foreground objects and the black pixels are background.) For each subproblem, please specify the parameters that you used.

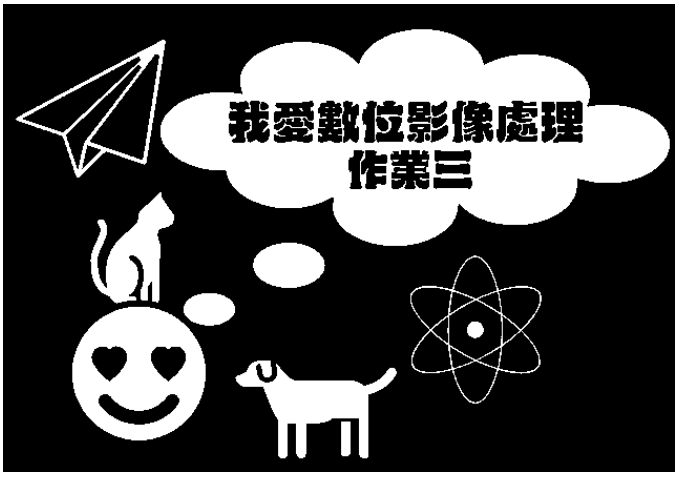
(a) (10 pt) Design a morphological processing to extract the objects' boundaries in sample1.png and output the result as result1.png.

Ans: 將原圖減去作完erosion的結果，即可得原圖中object品之boundary。

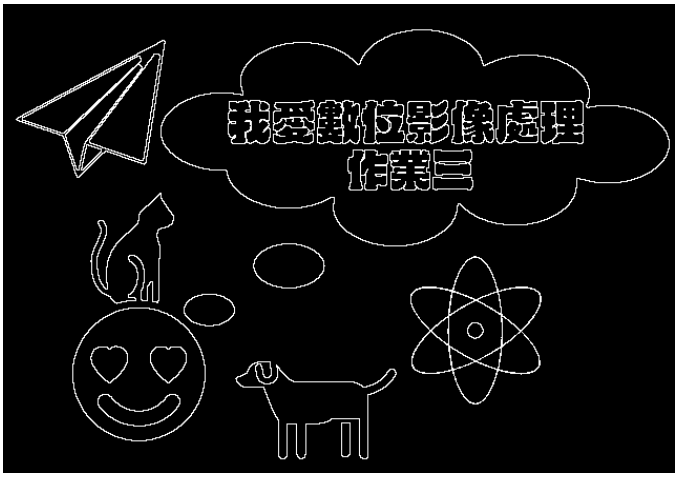
我使用的structuring element為：



original image:



output image:



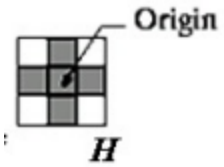
(b) (10 pt) Perform hole filling on sample1.png and output the result as result2.png.

Ans: 宣告一張空的圖，只放上想填滿的hole當中的pixel（這邊我用手動的點幾個位於hole的座標），不斷重複下列步驟直到指定區塊沒有空隙需填補（圖片做完這一輪後與上一輪結果無異）：

1. 對其做一輪dilation後
2. 和原圖之complement做交集(理解為使其不會超出原圖中指定要填補的區塊)

```
def hole_filling(img,start_i,start_j):
    kernel = np.array([[0,1,0],[1,1,1],[0,1,0]])
    kernel_size=3
    height,width=img.shape
    G= np.zeros((height,width))
    G[start_i][start_j]=1
    img_C=complement(img)
    while True:
        new_G=dilation(G,kernel,kernel_size)
        new_G[img_C==0]=0
        if (G==new_G).all():
            break
        G = new_G
    new_G[img==1]=1
    return new_G
```

我使用的structuring element:



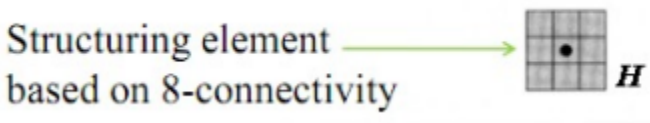
output image: 因題目沒有要求要將每個hole都填滿，這邊我選擇幾個hole(笑臉中的兩個眼睛和嘴巴、原子中的幾個hole)來作填滿。

(c) (10 pt) Design an algorithm to count the number of objects in sample1.png. Describe the steps in detail and specify the corresponding parameters.

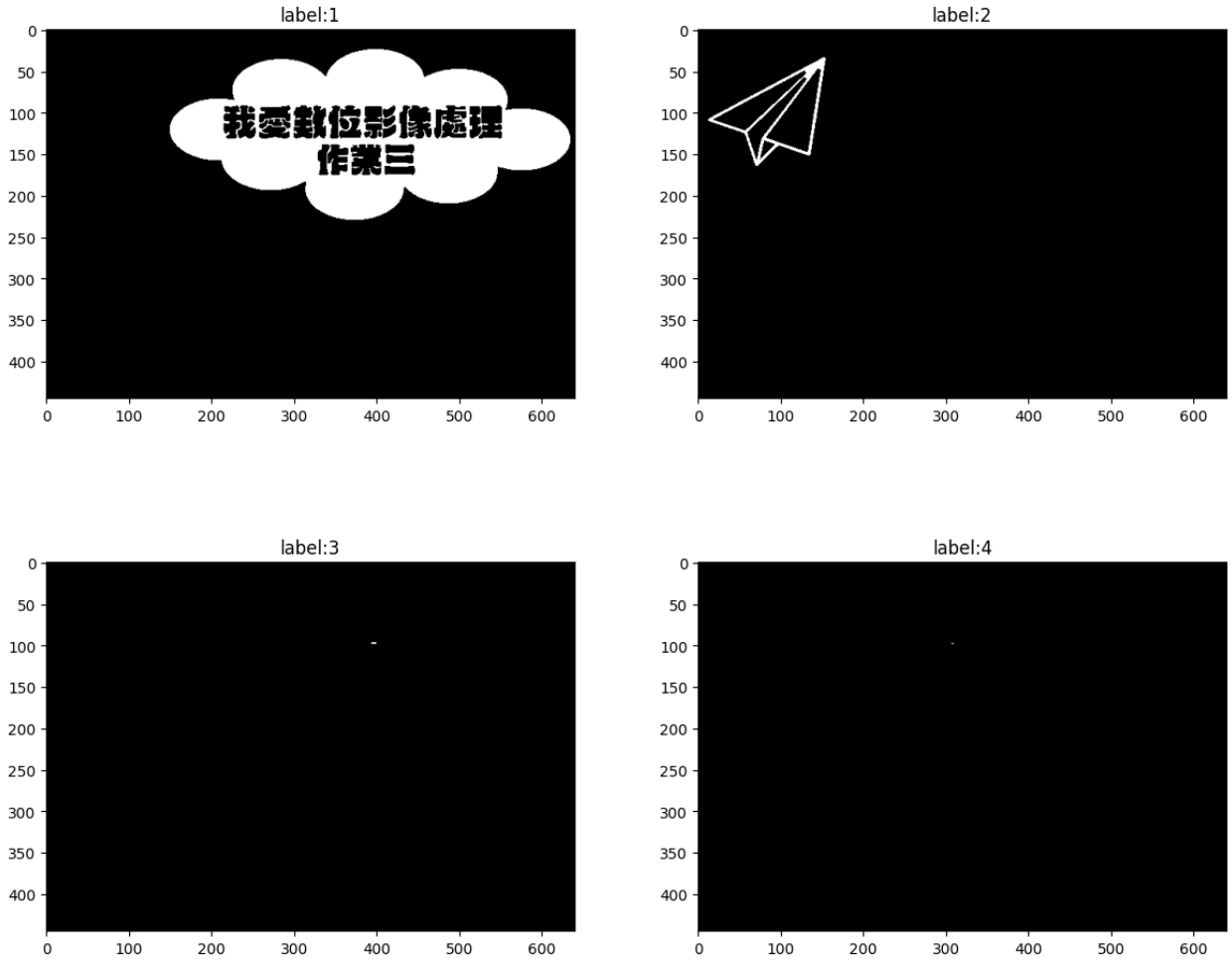
Ans:先對component中的一點作dilation（與前面hole filling實做方法類似），不斷重複下列步驟：

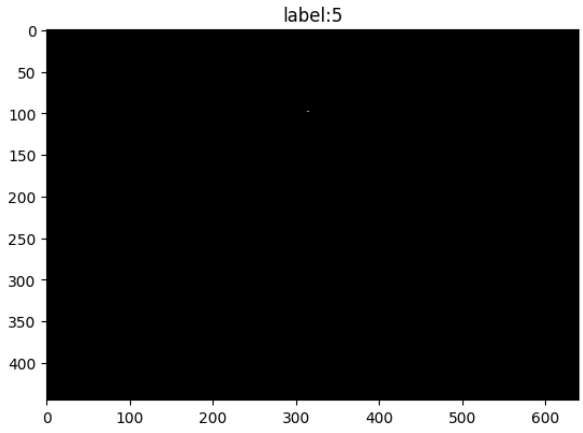
1. 找出connected component範圍：先取出component中的pixel在空白的圖片上，並重複下列步驟，直到擴張到整個該connected component的範圍（用此輪結果與上一輪結果無異判斷）：
    a. 對其做dilation
    b. 與原圖做交集
2. 將該connected component從圖中移除，並累計object數量。

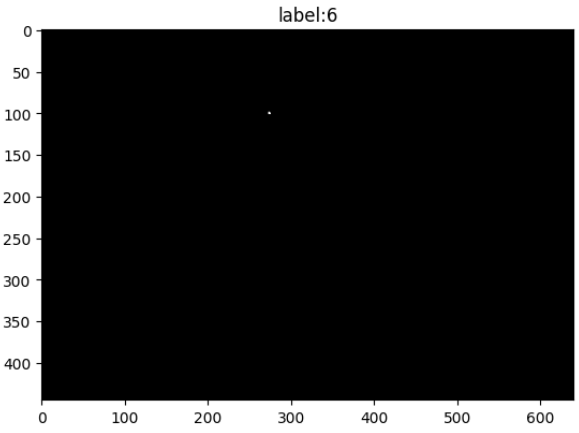
我使用的structuring element:

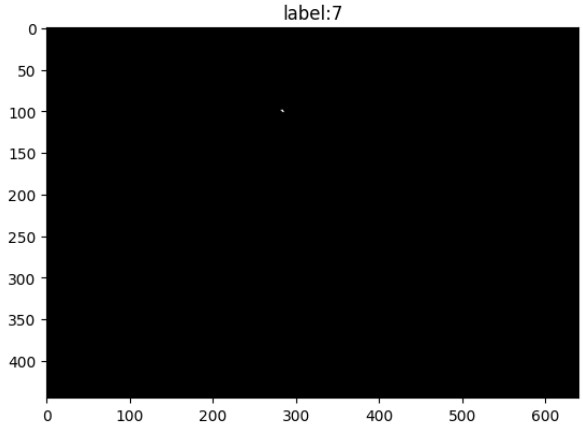


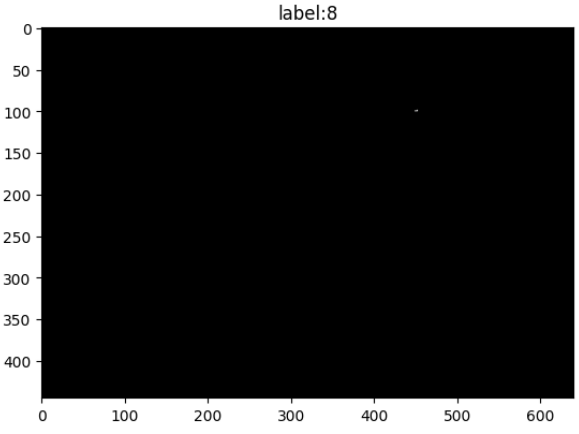
output:共有38個object

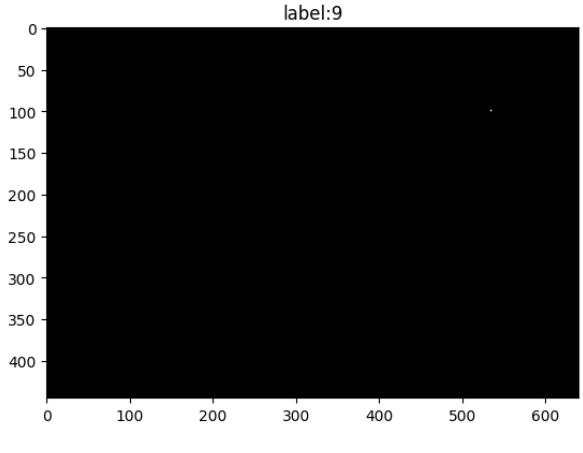


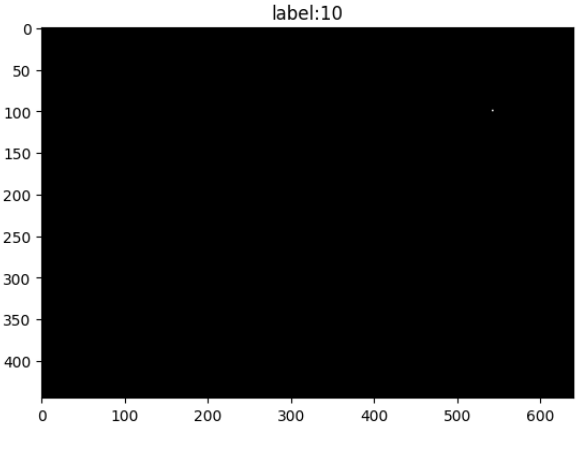

label:5

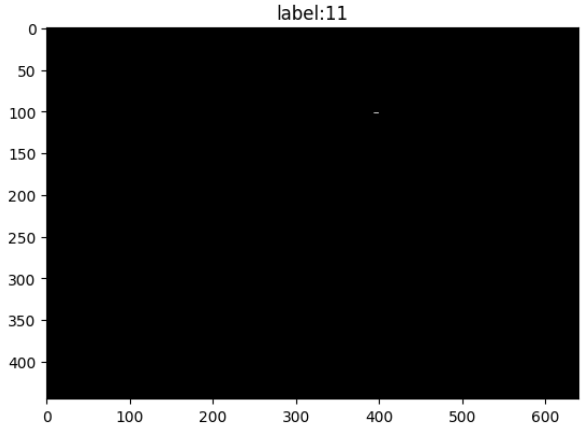

label:6

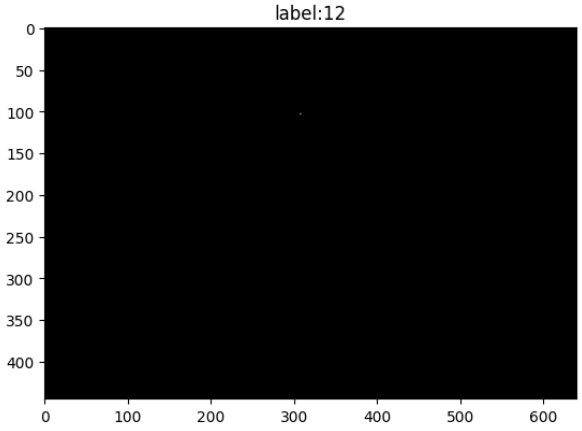

label:7

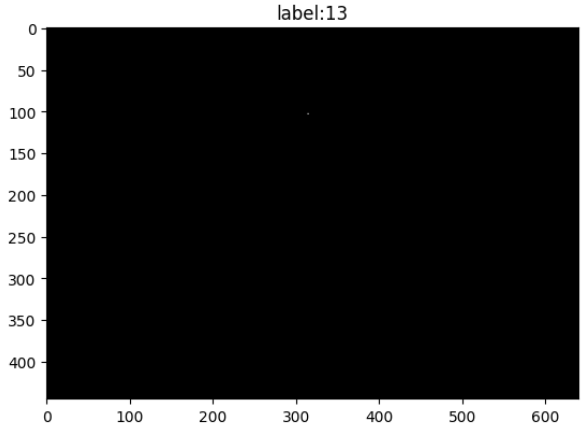

label:8

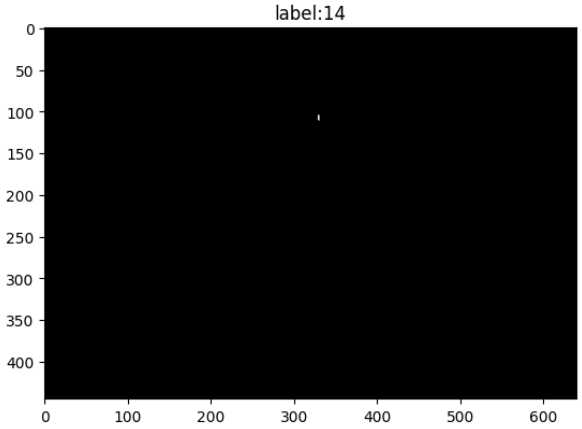

label:9

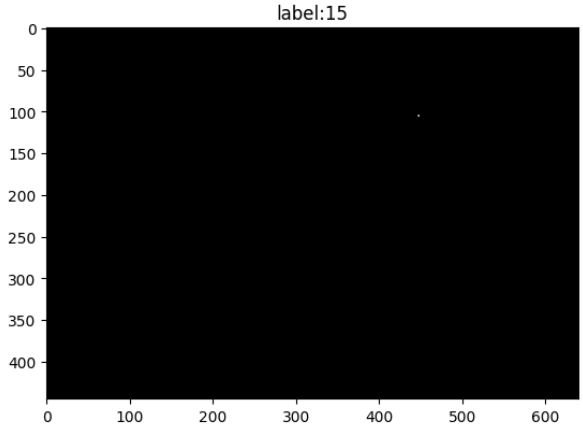

label:10

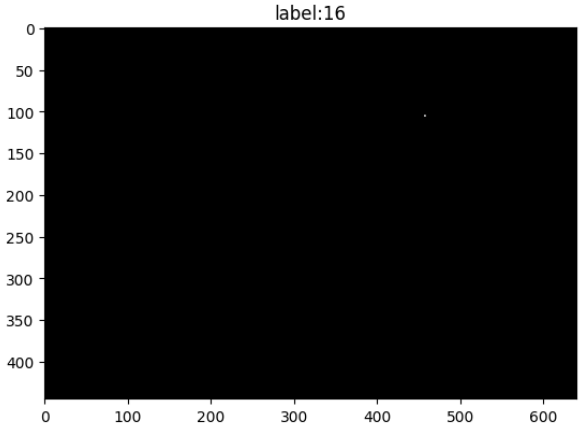

label:11

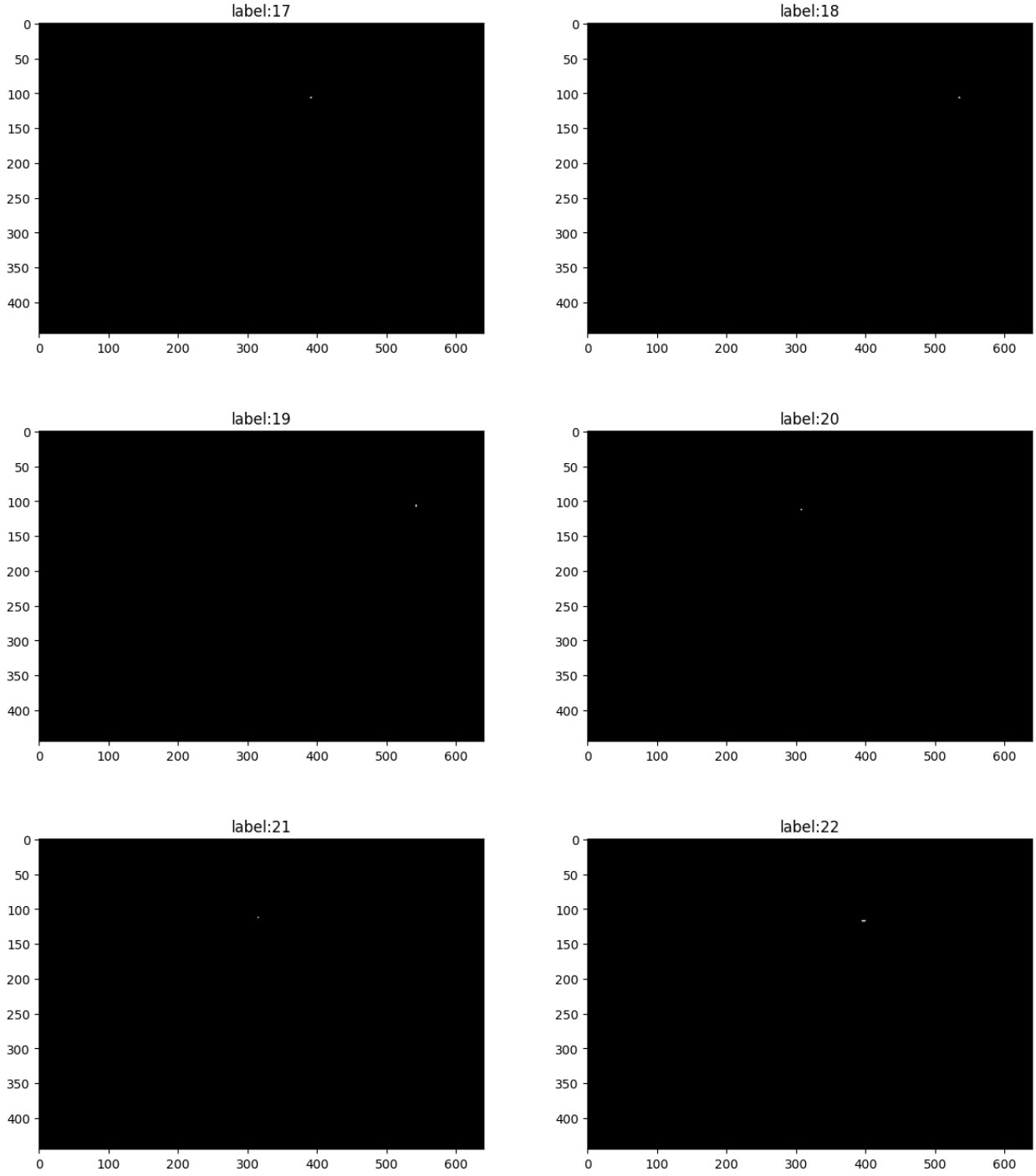

label:12

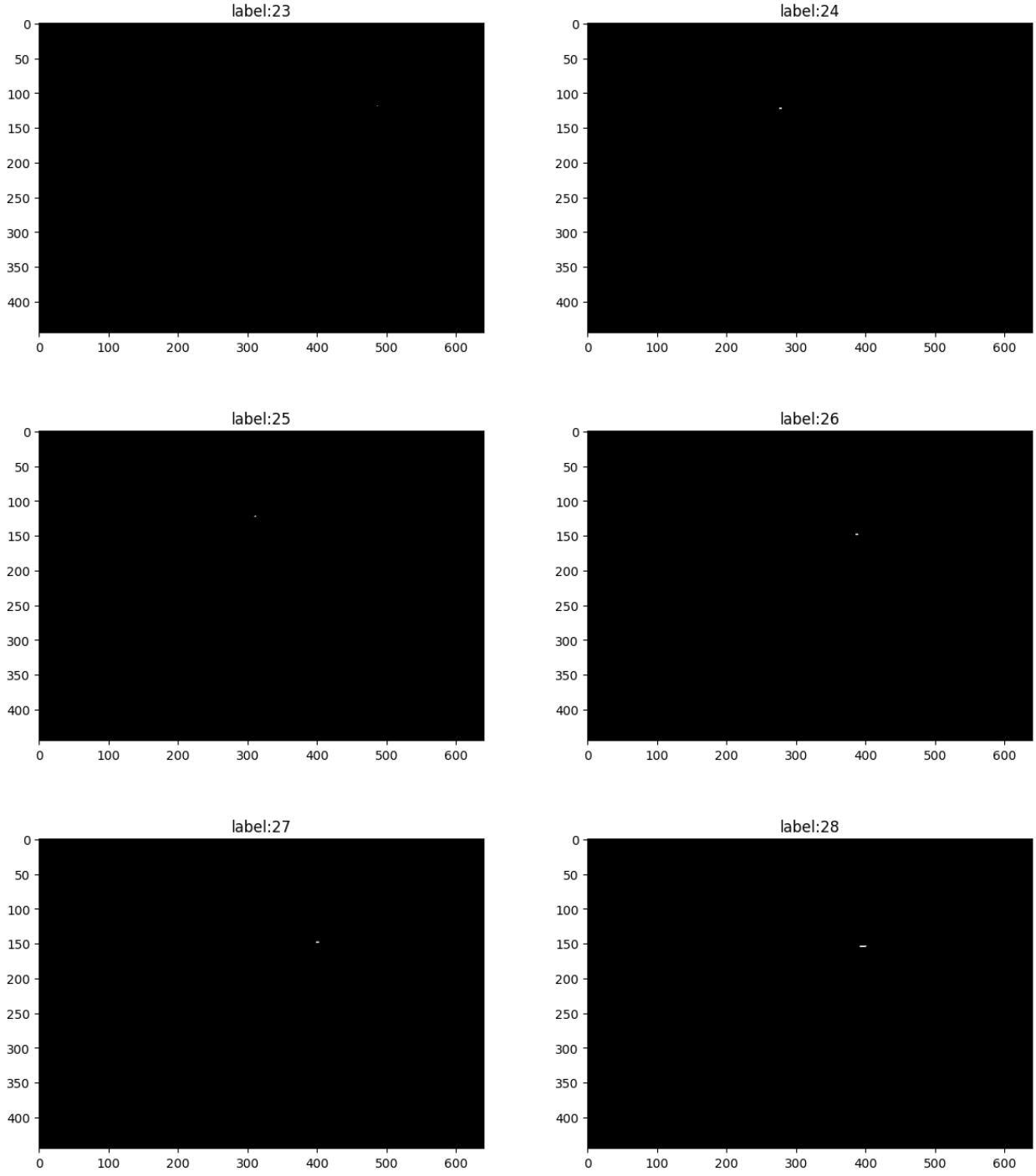

label:13

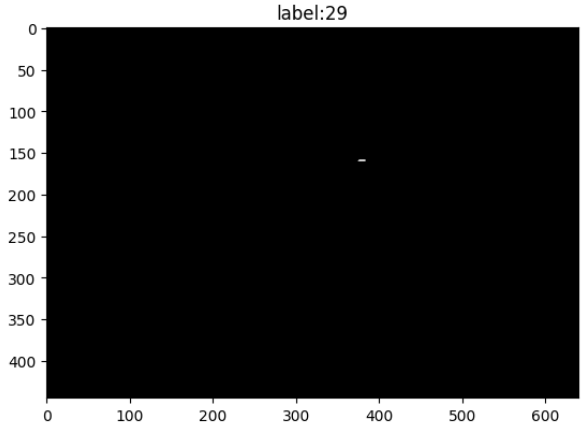

label:14

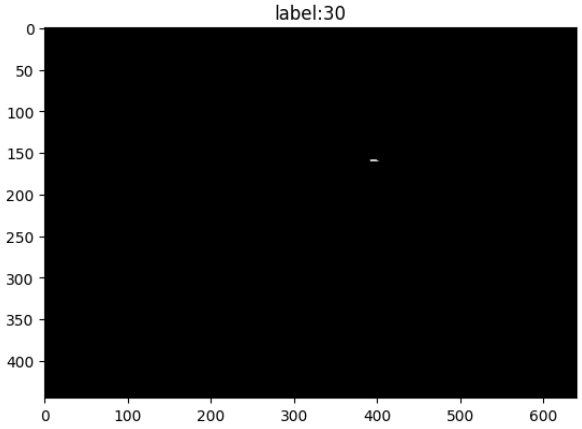

label:15

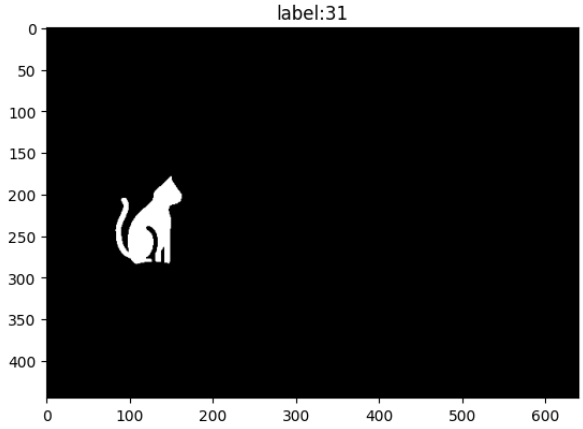

label:16

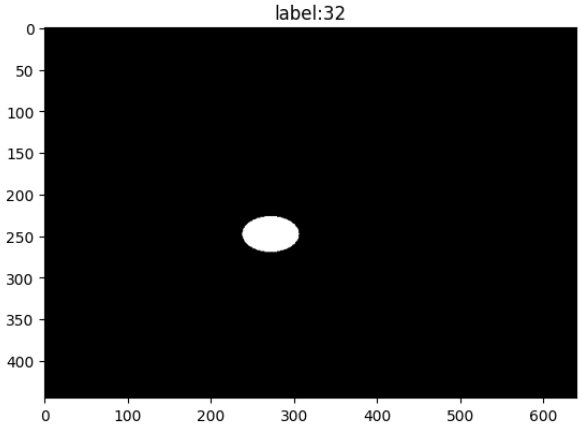

label:17
label:18
label:19
label:20
label:21
label:22

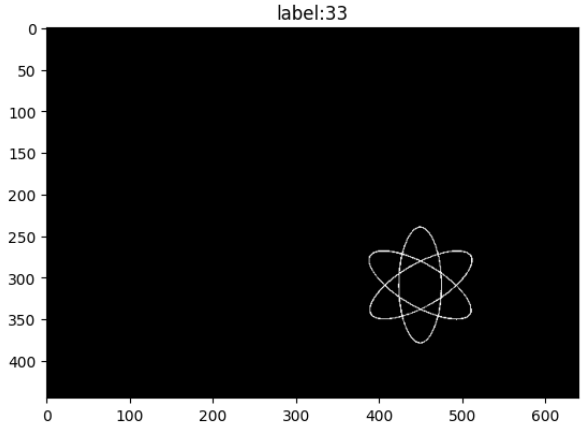

label:23
label:24
label:25
label:26
label:27
label:28

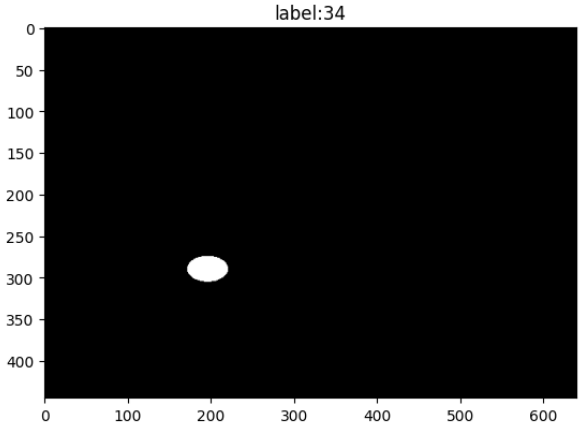

label:29

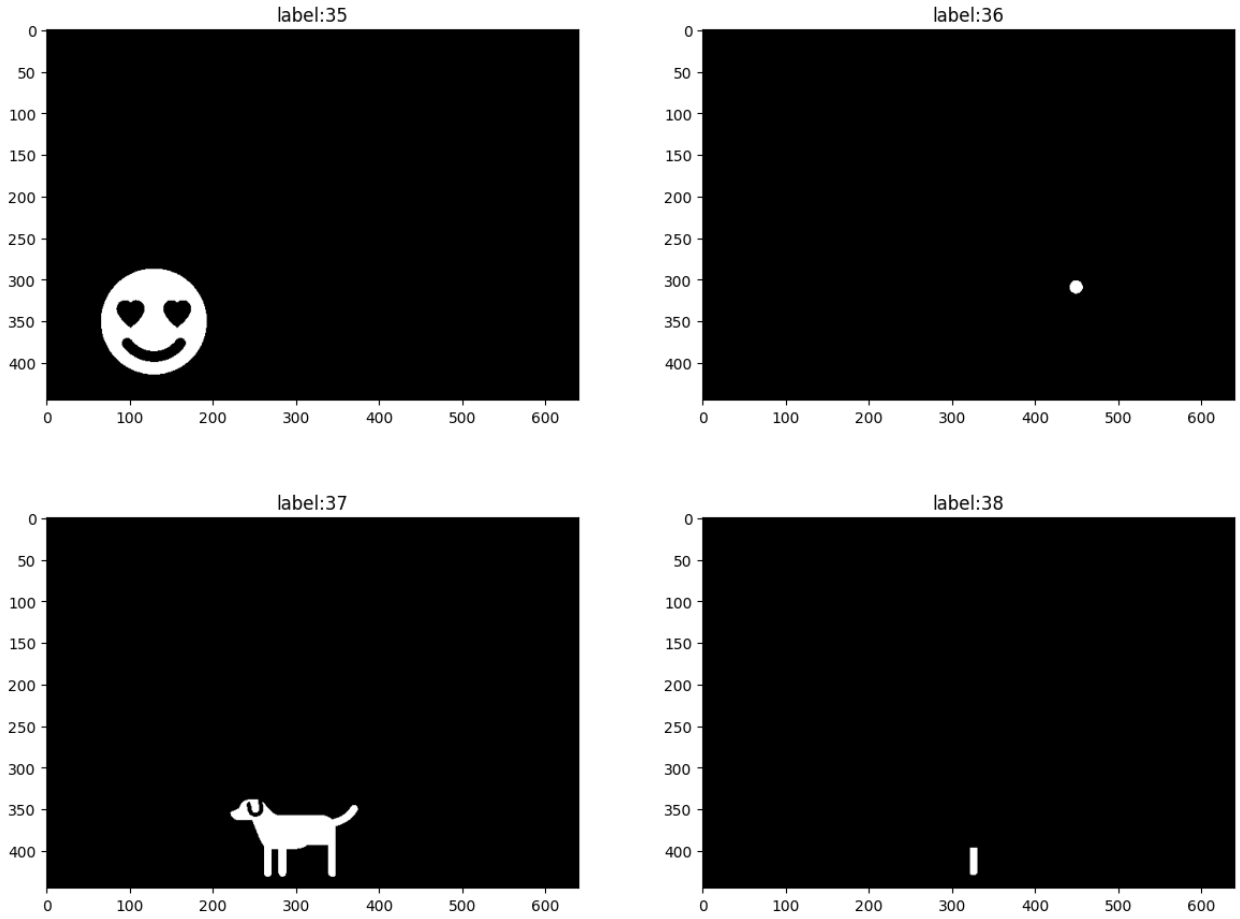

label:30


label:31


label:32


label:33


label:34



其中object label 3-30均為對話泡泡中中文字的一個小點，有些pixel數量較少，將圖片放大會看的較為清楚。

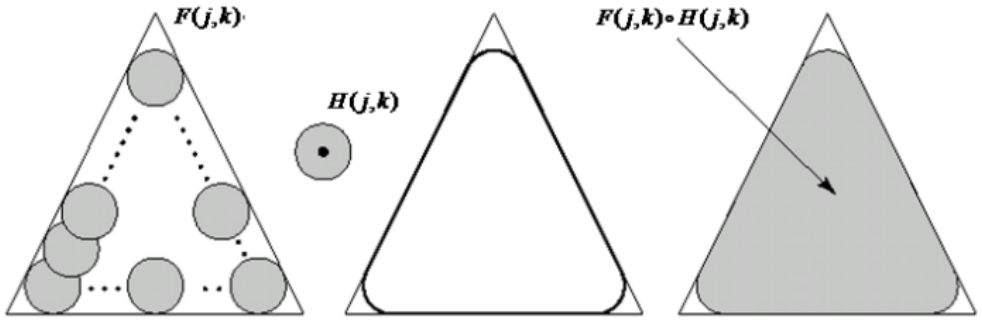
(d) (20 pt) Apply open operator and close operator to sample1.png and output the results as result3.png and result4.png, respectively. How will it affect the result images if you change the
shape of the structuring element?

- open operator:

- **Open operator**

$$G(j,k) = F(j,k) \circ H(j,k) = [F(j,k) \Theta \tilde{H}(j,k)] \oplus H(j,k)$$

  - **With a compact structuring element**
    - **Smoothes contours of objects**
    - **Eliminates small objects**
    - **Breaks narrow strokes**

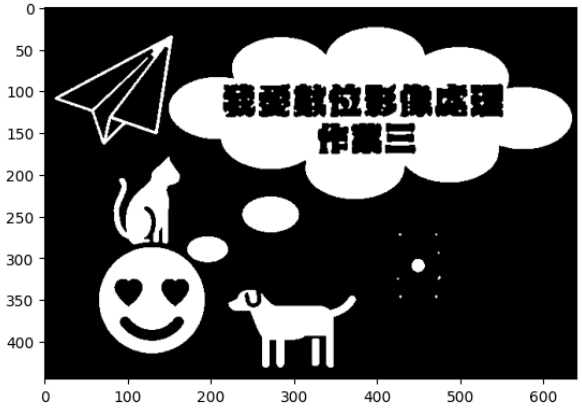


44

structuring element 1:

```
np.array([[0,1,0],[1,1,1,],[0,1,0]])
```

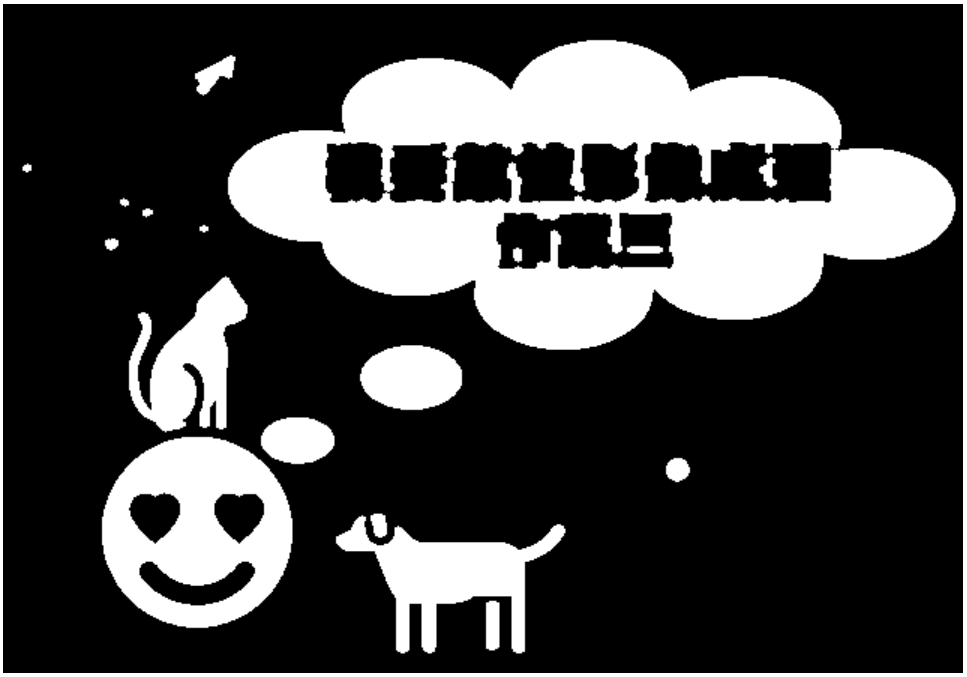
output:



structuring element 2:

```
np.array([[0, 0, 1, 0, 0],[1, 1, 1, 1, 1],[1, 1, 1, 1, 1],[1, 1, 1, 1, 1], [0, 0, 1, 0, 0]])
```

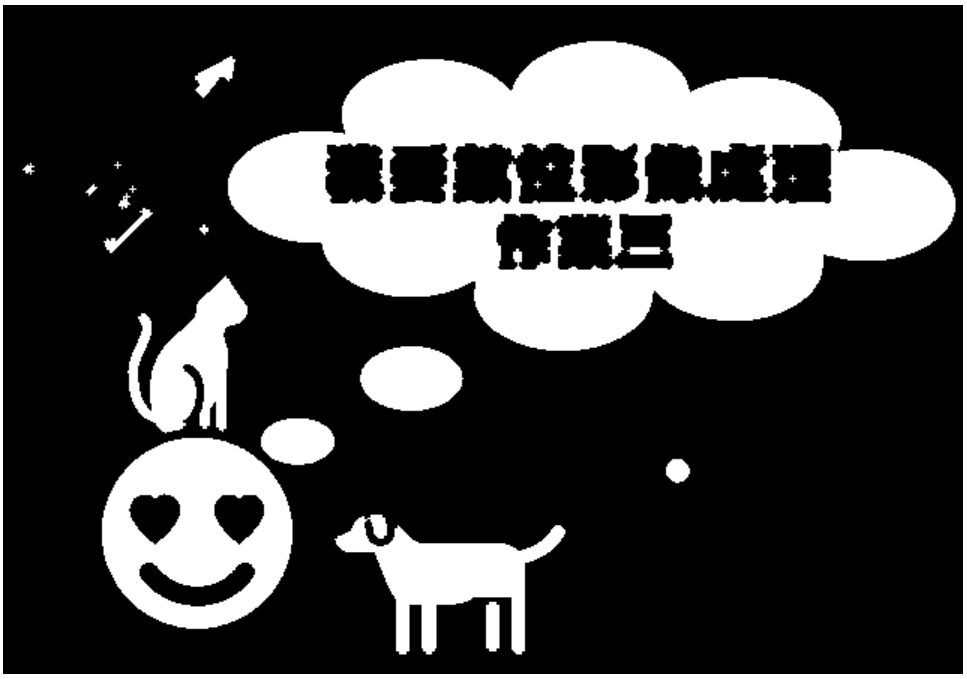
output:



structuring element 3:

```
,np.array([[0, 0, 1, 0, 0],[0, 0, 1, 0, 0],[1, 1, 1, 1, 1],[0, 0, 1, 0, 0],[0, 0, 1, 0, 0]])
```

output:



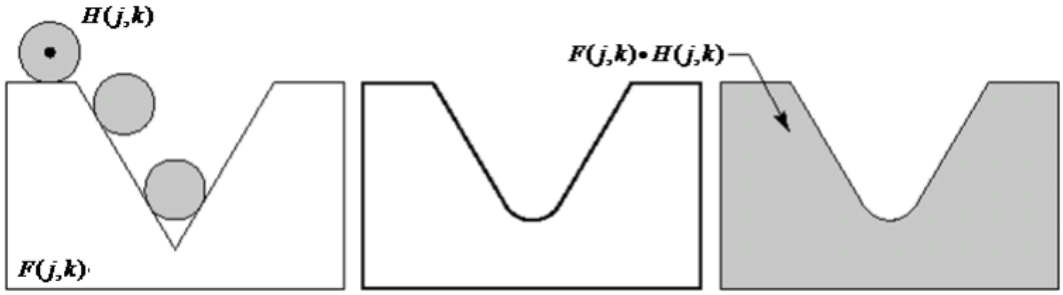
由以上結果可得structuring element 決定了滾過的球之形狀，設定size較小的structuring element，能滾過的範圍相較下較多，而size較大的structuring element能滾過的stroke相對更少，因此break的stroke比較多。其中structuring element 的形狀差異也顯現在smoothes contours 的形狀，可由structuring element 2和3 發驗：因structuring element 2形狀較element 3寬扁， 推測是因此導致2的contours smoothes結果較寬扁，3的結果較為有（從狗後右腳和身體接縫處看出）。其中對話泡泡中的文字裡面一些小object也可能被忽略（白色處）。

- close operator:

- **Close operator**

$$G(j,k) = F(j,k) \bullet H(j,k) = [F(j,k) \oplus H(j,k)] \Theta \tilde{H}(j,k)$$

  - **With a compact structuring element**
    - **Smoothes contours of objects**
    - **Eliminate small holes**
    - **Fuses short gaps between objects**

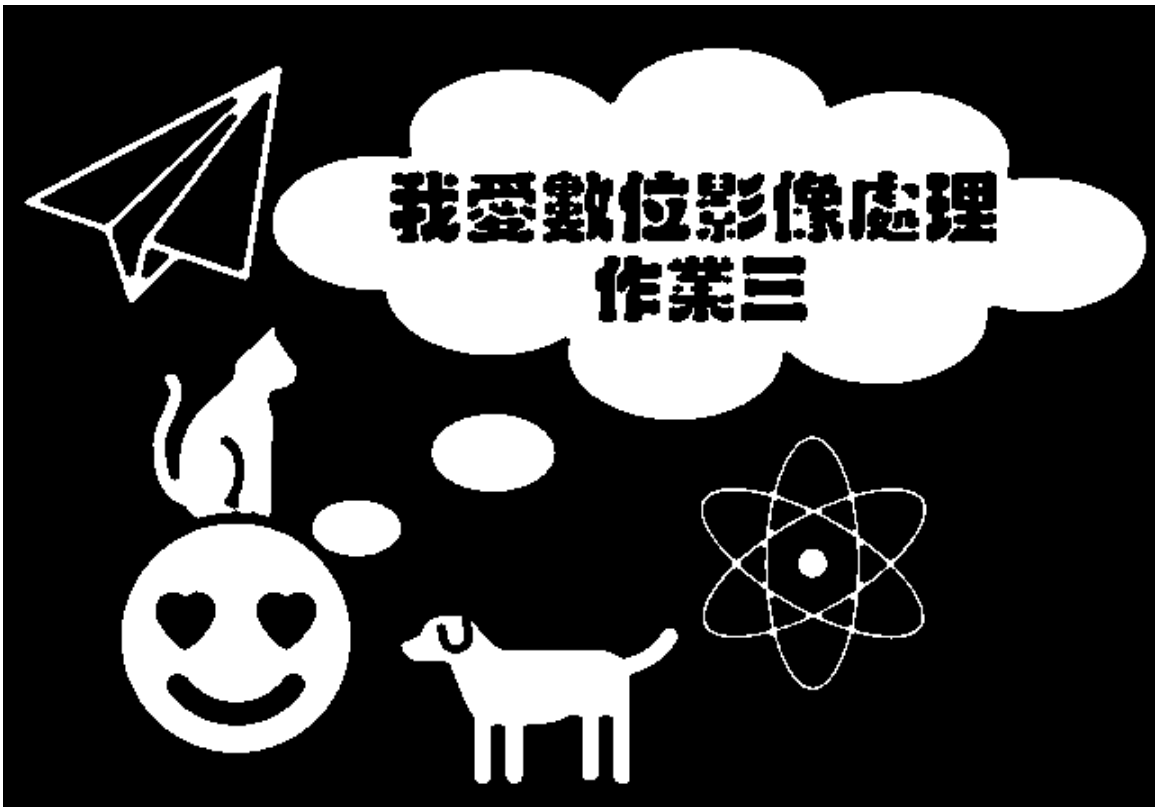


45

structuring element 1:

```
,np.array([[0, 0, 1, 0, 0],
           [1, 1, 1, 1, 1],
           [1, 1, 1, 1, 1],
           [1, 1, 1, 1, 1],
           [0, 0, 1, 0, 0]]),
```

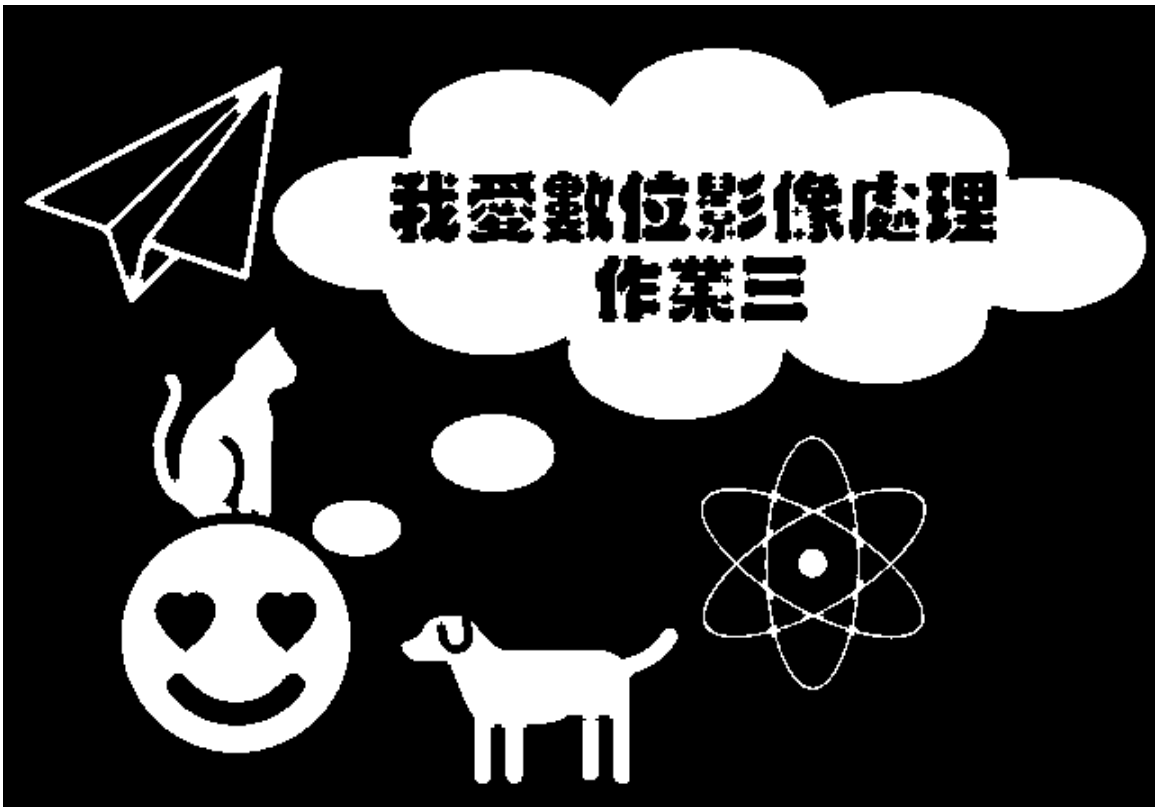
output:



structuring element 2:

```
np.array([[0, 0, 1, 0, 0],[0, 0, 1, 0, 0],[1, 1, 1, 1, 1],[0, 0, 1, 0, 0],[0, 0, 1, 0, 0]])
```

output:



structuring element 1的高度比structuring element 2長且形狀扁寬一些，因此在狗狗右後腿和身體接縫處連接（Fuses short gaps between objects）的較為完整一些，sturcturing element兩側還是能看見縫隙。

此外作完close operator後，在物品邊界外較為銳利的角也有向外被填的較為圓潤（幾個在紙飛機內部邊界夾角可見）對話泡泡裡面文字的一些小縫隙（黑色區塊為縫隙）也有被填滿（例如：『像』）。

Problem 2: TEXTURE ANALYSIS

(a) (10 pt) Perform Law's method on sample2.png to obtain the feature vectors. Please describe how you obtain the feature vectors and provide the reason why you choose it in this way.

Ans:我使用Micro-structure(multi-channel)method，因使用單一方法難以將各類texture分出，因此使用了3*3九種mask，步驟為：

1. Convolution:
   我使用的9個mask為講義上提供的9個mask:

$$\frac{1}{36}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$ Laws 1

$$\frac{1}{12}\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$ Laws 2

$$\frac{1}{12}\begin{bmatrix} -1 & 2 & -1 \\ -2 & 4 & -2 \\ -1 & 2 & -1 \end{bmatrix}$$ Laws 3

$$\frac{1}{12}\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$ Laws 4

$$\frac{1}{4}\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$$ Laws 5

$$\frac{1}{4}\begin{bmatrix} -1 & 2 & -1 \\ 0 & 0 & 0 \\ 1 & -2 & 1 \end{bmatrix}$$ Laws 6

$$\frac{1}{12}\begin{bmatrix} -1 & -2 & -1 \\ 2 & 4 & 2 \\ -1 & -2 & -1 \end{bmatrix}$$ Laws 7

$$\frac{1}{4}\begin{bmatrix} -1 & 0 & 1 \\ 2 & 0 & -2 \\ -1 & 0 & 1 \end{bmatrix}$$ Laws 8

$$\frac{1}{4}\begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix}$$ Laws 9

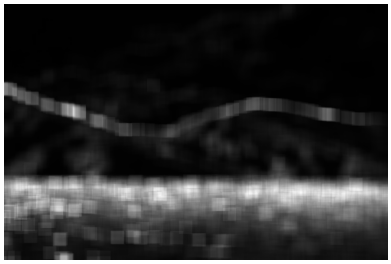
2. Energy Computation:
   使用15*15的window size計算分別計算9個套用mask後的local feature，並使用windows裡面每個pixel之feautre平方的總和作為energy。

original image:

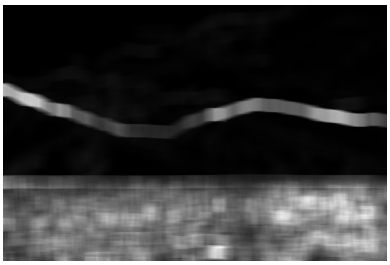
Law 1 ~ 9:（對應到0-255的值後之示意圖片）

law 1

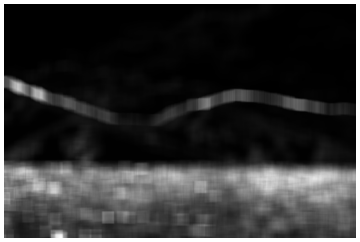
law 2



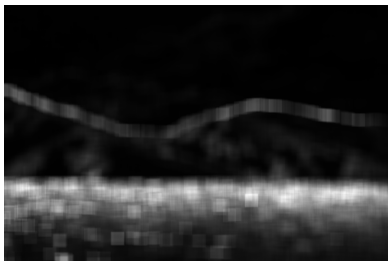
law 3



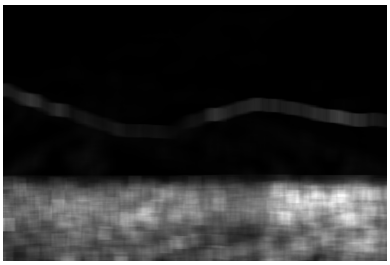
law 4



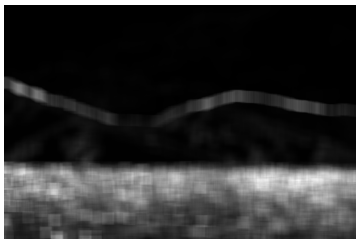
law 5



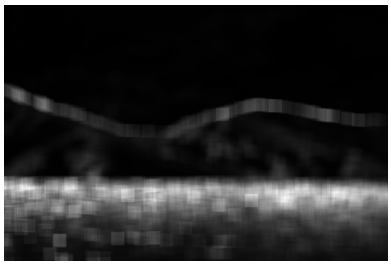
law 6



law 7



law 8



law 9

(b) (20 pt) Use k-means algorithm to classify each pixel with the feature vectors you obtained from(a). Label the pixels of the same texture with the same color and output it as result5.png. Please describe how you use the features in k-means algorithm and all the chosen parameters in detail.

Ans:將上題結果的9張照片做轉換，使9張(600*900)的feature變為一張(600,900,9)的feature，後對這600*900個9維的向量執行k-means。步驟如下：

- // Initialization //
  - Select k vectors as the initial centroids
  - Do the following iterations
- // step1 // Form k clusters using the NN rule
- // step2 // re-compute the centroid of each cluster

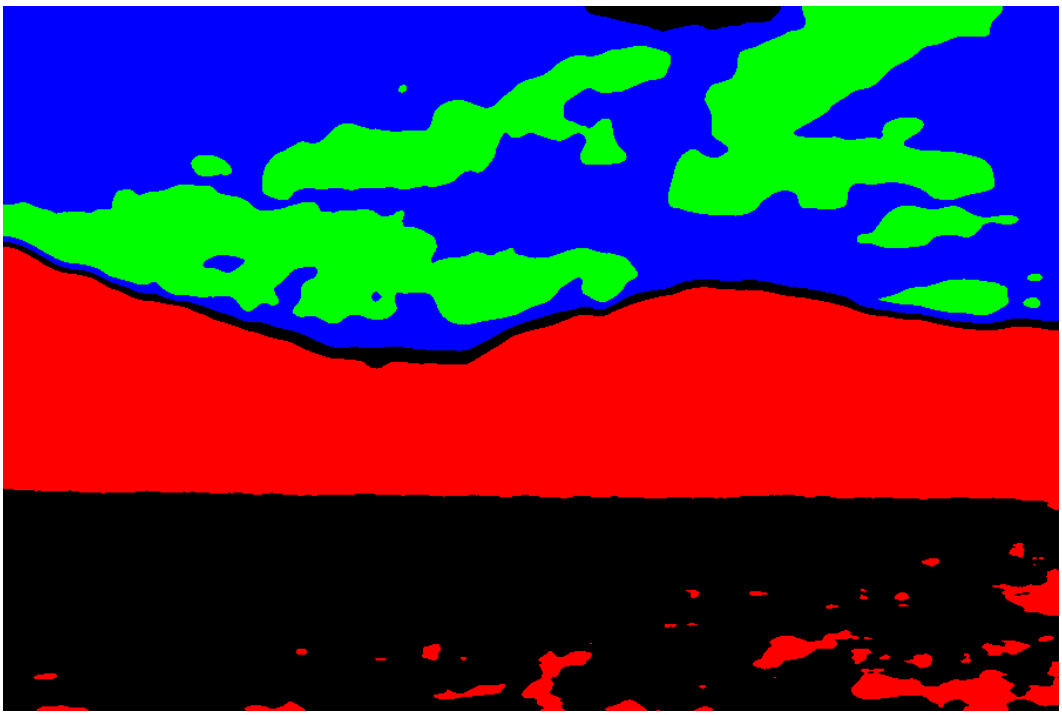
```python
def K_means(feature,K,iteration):
    # height,width,dimension=feature.shape
    height=feature.shape[0]
    width=feature.shape[1]
    # feature.reshape(-1,9)
    C=np.ones((height,width))
    C*=-1
    K=4
    # initialization
    # i_list=random.sample(range(height),K)
    # j_list=random.sample(range(width),K)
    # centroids=np.array([feature[i_list[0]][j_list[0]]])
    # for k in range(1,K):
    #     centroids=np.append(centroids,[feature[i_list[k]][j_list[k]]],axis=0)
    centroids=np.array([feature[100][100],feature[50][680],feature[350][500],feature[500][400]])
    for _ in range(iteration):
        # step1: form k clusters using the NN rule
        group=[[]for i in range(K)]
        for i in range(0,height):
            for j in range(0,width):
                k=-1
                min_diff=float('inf')
                for centroid in centroids:
                    k+=1
                    diff=np.dot(feature[i][j]-centroid,feature[i][j]-centroid)
                    if diff <min_diff:
                        min_diff=diff
                        C[i][j] = k
                group[int(C[i][j])].append(feature[i][j])
        # step2: re-compute the centroid of each cluster
        for k in range(K):
            group_k=np.array(group[k])
            centroids[k]=group_k.mean(axis=0)
    return C
```

經觀察後決定分為4群，10次iteration即可。

我一開始的方法是隨機取出圖片中的k個點作初始的centroids（k個9維的向量），但是效果並不理想，執行過程中可能有centroid重疊，導致無法分為k群的問題；後來改為肉眼判斷圖片中大約分為4群：天空、雲朵、山、海，因此在初始時將4個點分別上述的4個群中取出，效果就變好許多。

original image:

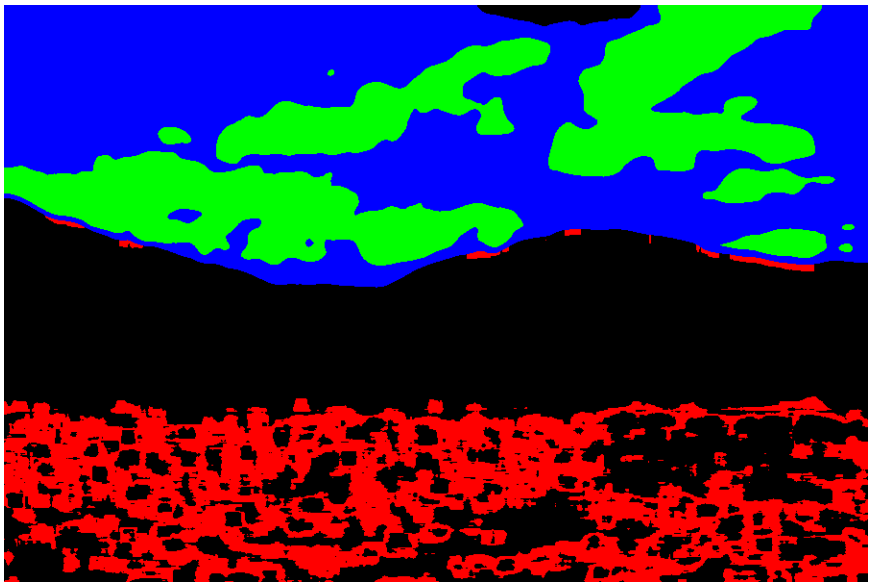
output image:



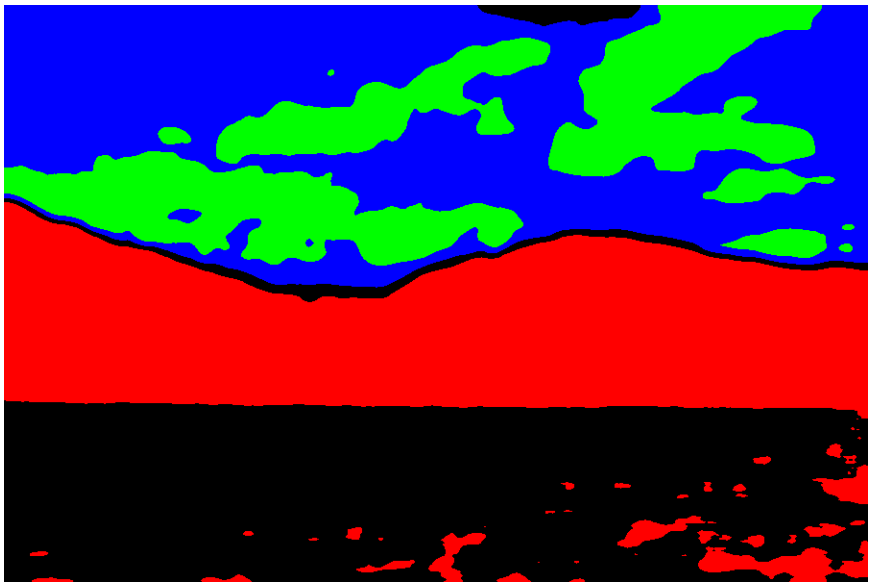
大部分都有分類正確，除了少面積的海被判斷為山，和山和天空的交界有細細的部份被判斷為海（猜想是部份filter為計算gradient的關係。）

(c) (20 pt) Based on result5.png, design a method to improve the classification result and output the updated result as result6.png. Describe the modifications in detail and explain the reason why.

- 將feature每一張圖分別做k-means，再以投票方式分類，但是結果不是太好，猜是中間很多mask的output結果都無法正確的區分區域，導致分開判斷時大多數mask的分類結果不正確。
- 以肉眼判斷並調整的方式：用肉眼看過每個filter的特徵後，在特定類別用特定的filter作判斷，天空和山的邊界判斷錯誤的部份是變窄變少了，但是海的部份空缺卻變多了：



- 嘗試把某幾個mask的feature結果加大，也並沒有什麼肉眼可見的顯著影響：（我將laws 1,4,7權重加大非常多倍）



- 整體來說，最改善有感的是將energy computation中計算local feature的window size調大：

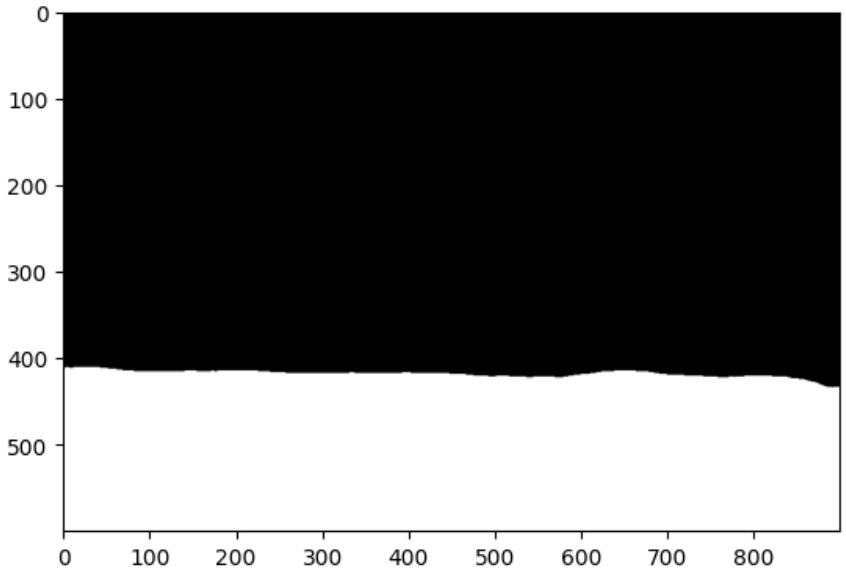
  嘗試把energy computation中計算local feature的window size調大，海中右下角誤判為山的部份就減少了，猜想是當feature考慮的範圍變大，能減少region interior中holes的情形。

  下面是window size設為31*31的output:

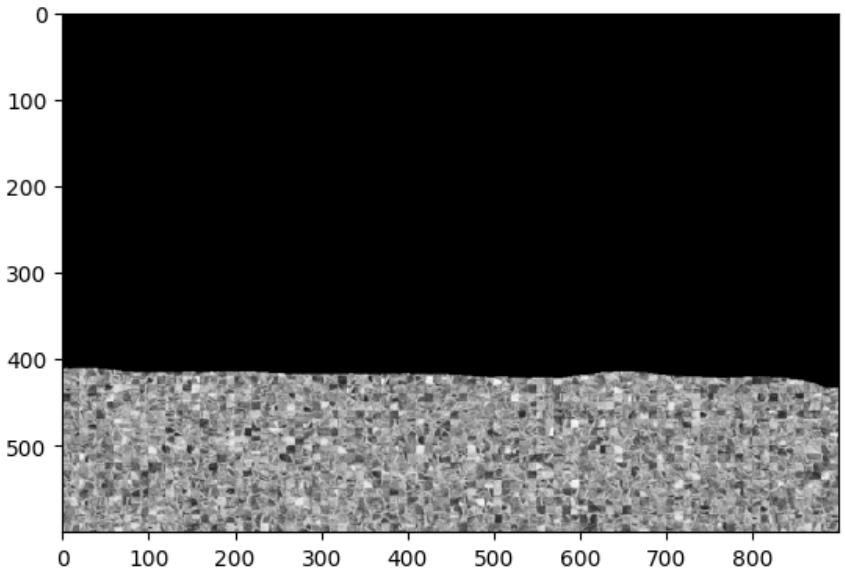
但是圖中天空和山的邊界誤判為海的部份也變寬了，可能是因feature的考量範圍變大時可能會使邊界因計算local feature的window中有不只一種材質而不易判定的區塊擴大。

(d) (Bonus) TA can’t swim. Try to perform image quilting, replacing the sea in sample2.png with
sample3.png or other texture you prefer by using the result from (c), and output it as result7.png. It’s allowed to utilize external libraries to help you accomplish it, but you should specify the im-plementation detail and functions you used in the report.

Ans: 首先將我將上面laws’ method的window_size再擴大成51(為將海中的holes盡量填滿)，再將label為海的部份設定為我要填補的區域，其餘設定為0，為我後續要蓋上新材質補丁的區域之mask。



之後再使用https://github.com/bglogowski/ImageQuilting 的 quilt_random 方法，其作法為隨機取sample3之的部份區塊作填入，將寬為200填補至900後蓋上mask取出填補範圍。



再將上面填補之區塊在原圖中清零，兩圖合併即完成。

output: