DIP Homework Assignment #3

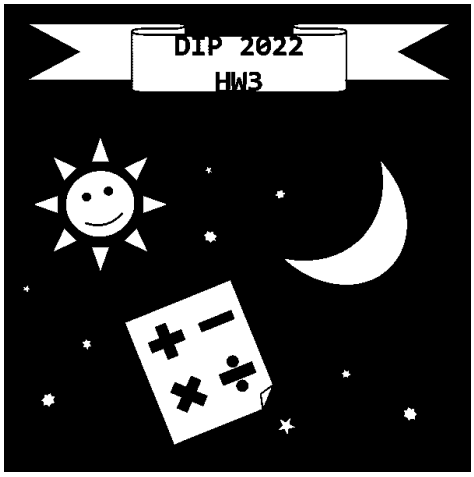
Name: 方郁婷

ID #: r10922196

email: r10922196@ntu.edu.tw

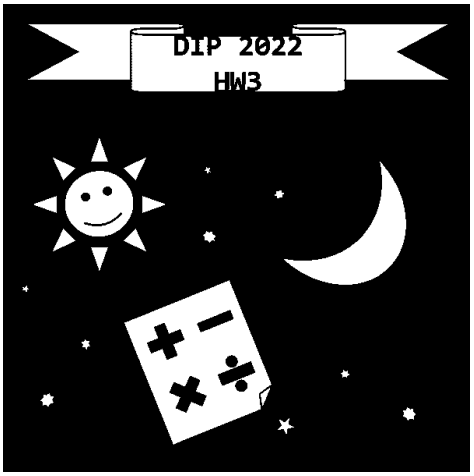
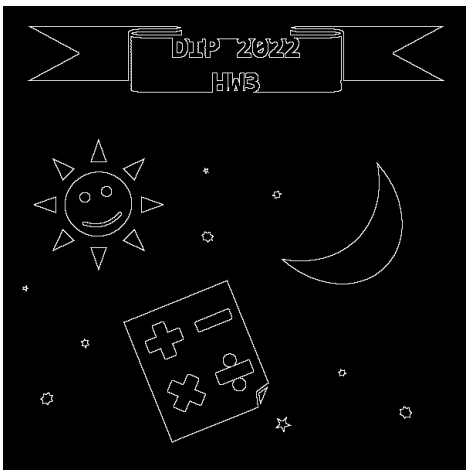
Problem 1: MORPHOLOGICAL PROCESSING

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.)



sample1.png

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

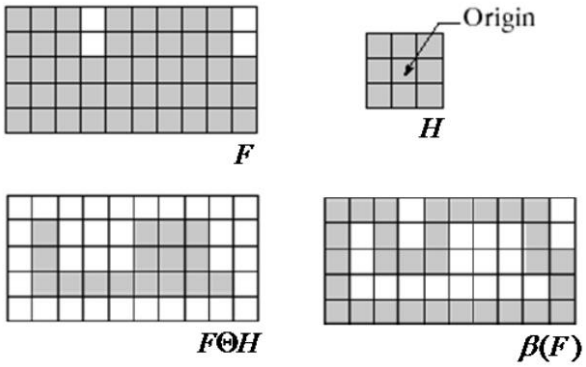
	
Original image	Output image

Motivation and approach (include parameters):

如同 ppt 的作法，利用 H 將全圖掃一遍，之後再跟原圖相減

- **Boundary Extraction**

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

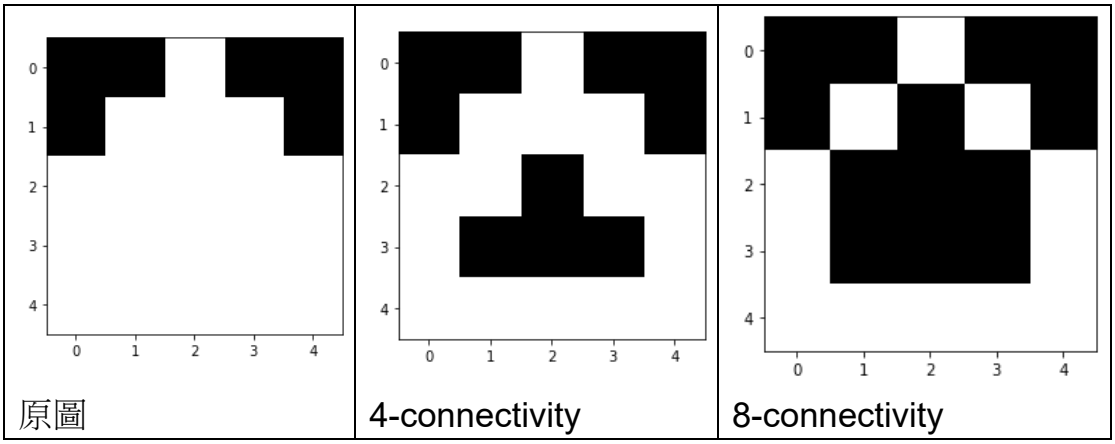


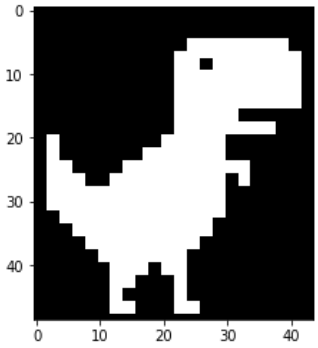
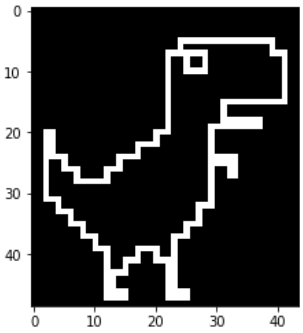
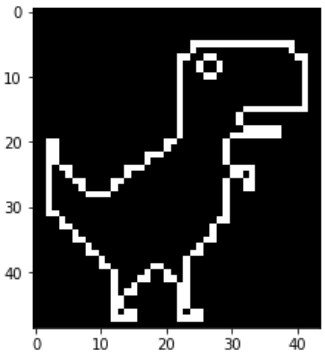

```
H = np.ones((3,3))
h,w = sample1.shape
result1 = np.zeros((h,w))

for i in range(1,h-1):
    for j in range(1,w-1):
        patch = sample1[i-1:i+2, j-1:j+2]
        if (patch*H).sum() == 9: result1[i][j] = 1

result1 = sample1-result1
```

Discussion of results:

成功取得邊緣。

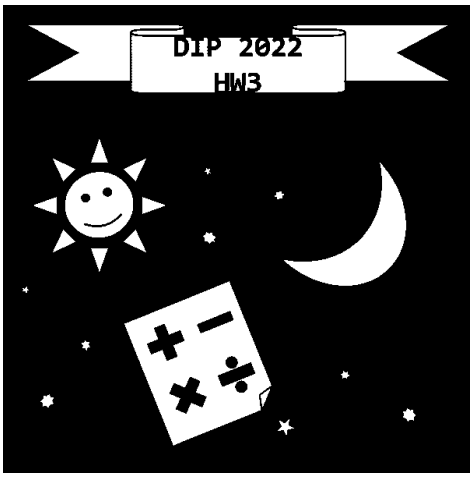
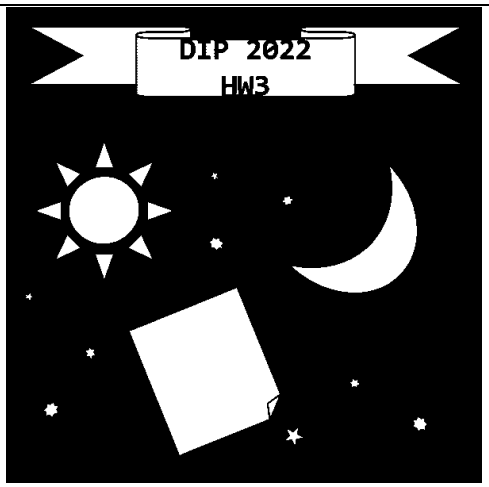
以上的方法是使用 4-connectivity，我還有試過 8-connectivity



		
原圖	4-connectivity	8-connectivity

說實話當像數一多，這些細節就會變得很不明顯，所以也是沒什麼差。真要說的話，小恐龍的部分，外觀線條我喜歡的是 4-connectivity(手跟腳沒有洞洞)，眼睛的部分我比較喜歡 8-connectivity(眼睛比較圓，比較可愛)

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

	
Original image	Output image

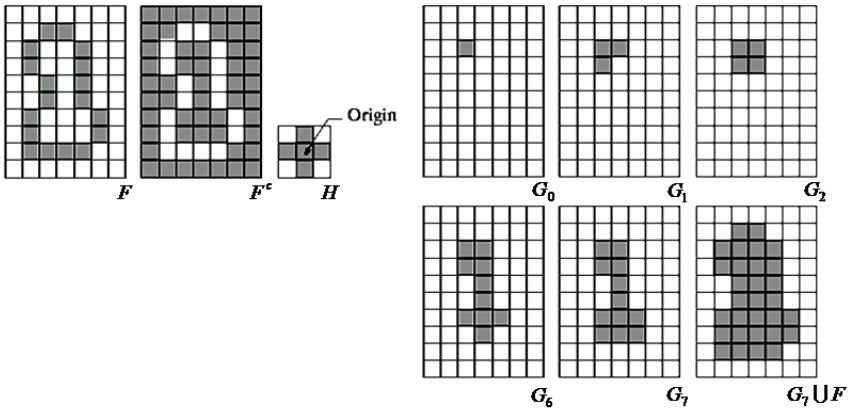
motivation and approach (include parameters):

使用 H 讓 pixel 的四周(上下左右)長出 pixel，再與 Fc 取交集，會把邊界的 pixel(錯誤的 pixel)去掉，保留正確長出的 pixel，一直重複這個步驟，

直到 pixel 不再長出為止，之後再與原圖相加就能填充回去。

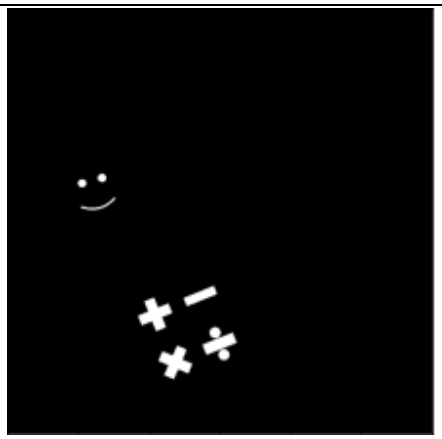
- **Hole Filling**

$$G_i(j,k) = (G_{i-1}(j,k) \oplus H(j,k)) \cap F^c(\mathrm{j,k}) \quad i = 1,2,3...$$
$$G(j,k) = G_i(j,k) \cup F(\mathrm{j,k})$$



洞洞的初始化座標是用小畫家找的，為了加速長出的速度，我特意把洞洞座標選在洞洞中心點(這樣上下左右都能同時長)，並且讓 9 個洞洞同時生長，以加快完成的時間。

```
H = np.array([[0,1,0],[1,1,1],[0,1,0]])
h,w = sample1.shape
G1 = np.zeros((h,w))
# 初始化洞洞座標(9個點)
holes = ((247,104),(239,134),(281,130),(433,209),
for hole in holes:
    G1[hole] = 1

while True:
    G2 = np.zeros((h,w))
    for i in range(1,h-1):
        for j in range(1,w-1):
            patch = G1[i-1:i+2, j-1:j+2]
            if (patch*H).sum() > 0: G2[i][j] = 1
    G2 = G2 * Fc
    if (G1==G2).all(): break
    G1 = G2
```



result

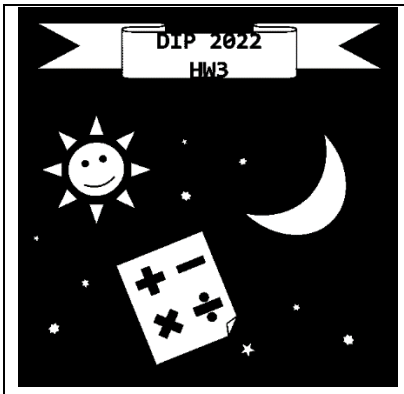
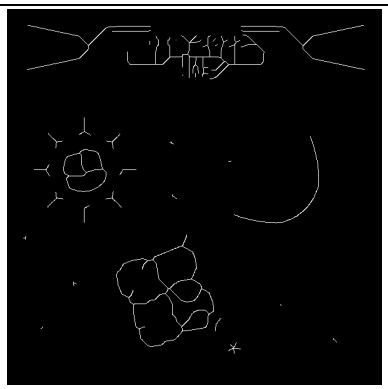
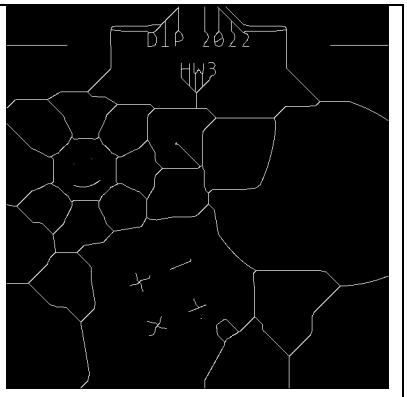
最後再將圖片相加即可 `G1+sample1`

Discussion of results:

成功填滿洞洞。

雖然是 9 個洞一起填的，但必須要 9 個洞都找完才會停止，只要有其中一個洞還未填完，已經找完的洞依舊會重複讓 pixel 的四周(上下左右)長出 pixel，與 Fc 取交集，把邊界的 pixel(錯誤的 pixel)去掉這些過程。

(c) (15 pt) Apply skeletonizing to sample1.png and output the result as result3.png. Reverse foreground and background pixels and apply skeletonizing to the resultant image. Output the skeletonized result as result4.png. What can you observe from result3.png and result4.png?

		
Original image	result3.png	result4.png

Motivation and approach (include parameters):

```
def neighbours(x,y,image):
    "Return 8-neighbours of image point P1(x,y), in a clockwise order"
    img = image
    x_1, y_1, x1, y1 = x-1, y-1, x+1, y+1
    return [ img[x_1][y], img[x_1][y1], img[x][y1], img[x1][y1],     # P2,P3,P4,P5
                img[x1][y], img[x1][y_1], img[x][y_1], img[x_1][y_1] ]    # P6,P7,P8,P9

def transitions(neighbours):
    "No. of 0,1 patterns (transitions from 0 to 1) in the ordered sequence"
    n = neighbours + neighbours[0:1]      # P2, P3, ... , P8, P9, P2
    return sum( (n1, n2) == (0, 1) for n1, n2 in zip(n, n[1:]) )  # (P2,P3), (P3,P4), ... , (P8,P9), (P9,P2)
```

分為兩個 step

```
# Step 1
changing1 = []
rows, columns = Image_Thinned.shape               # x for rows, y for columns
for x in range(1, rows - 1):                     # No. of  rows
    for y in range(1, columns - 1):            # No. of columns
        P2,P3,P4,P5,P6,P7,P8,P9 = n = neighbours(x, y, Image_Thinned)
        if (Image_Thinned[x][y] == 1 and    # Condition 0: Point P1 in the object regions
            2 <= sum(n) <= 6 and       # Condition 1: 2<= N(P1) <= 6
            transitions(n) == 1 and    # Condition 2: S(P1)=1
            P2 * P4 * P6 == 0  and     # Condition 3: P2 , P4 , P6 中有0 L
            P4 * P6 * P8 == 0):        # Condition 4: P4 , P6 , P8 中有0 D
            changing1.append((x,y))
            print(np.array([[P9,P2,P3],[P8,1,P4],[P7,P6,P5]]))
            print()
for x, y in changing1:
    Image_Thinned[x][y] = 0
```

```
# Step 2
changing2 = []
for x in range(1, rows - 1):
    for y in range(1, columns - 1):
        P2,P3,P4,P5,P6,P7,P8,P9 = n = neighbours(x, y, Image_Thinned)
        if (Image_Thinned[x][y] == 1    and        # Condition 0
            2 <= sum(n) <= 6  and        # Condition 1
            transitions(n) == 1 and      # Condition 2
            P8 * P2 * P4 == 0 and        # Condition 3 U
            P6 * P8 * P2 == 0):          # Condition 4 R
            changing2.append((x,y))
              print(np.array([[P9,P2,P3],[P8,1,P4],[P7,P6,P5]]))
              print()
for x, y in changing2:
    Image_Thinned[x][y] = 0
```

只要符合以下 pattern 就可移除中心的 1

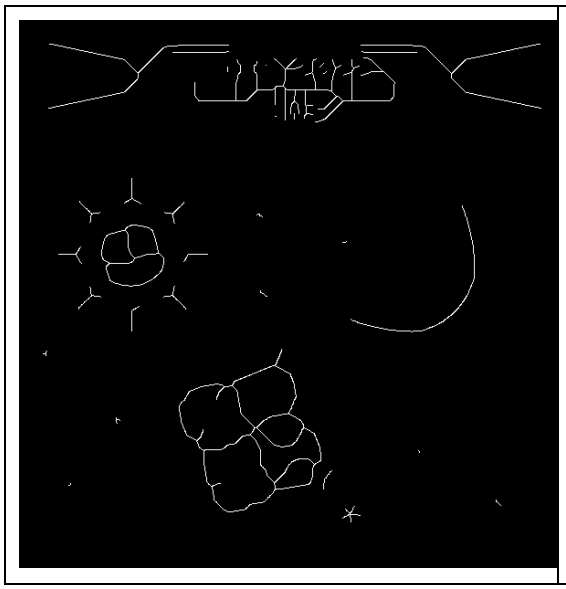
[[0 0 0] [0 1 1] [0 1 1]]	[[0 0 0] [1 1 0] [1 1 0]]	[[1 1 0] [1 1 0] [0 0 0]]	[[0 1 1] [0 1 1] [0 0 0]]

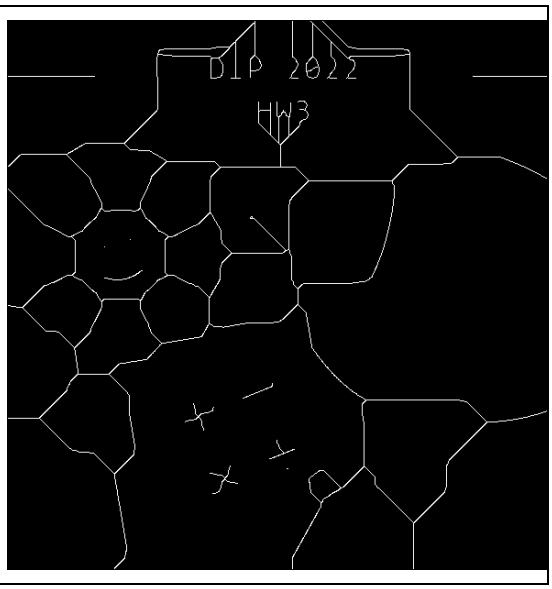
這樣當移除中心的 1 時，會形成一個凹邊

原理就是只要移除中心點時能形成 4-conectivity 的邊即可，所以還有很多其他的 case(詳細請看 code 的 Condition3、4)

以小恐龍為例: 

Discussion of results:

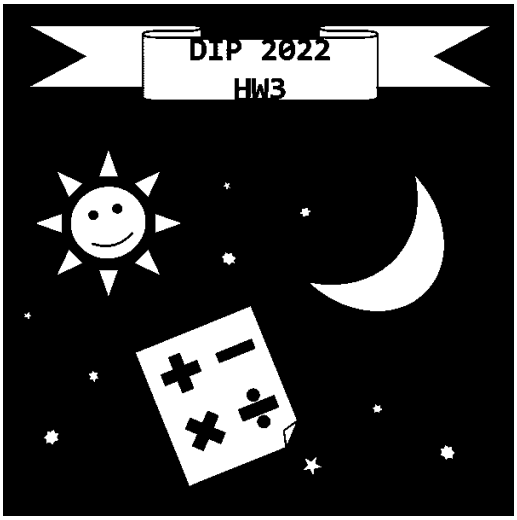
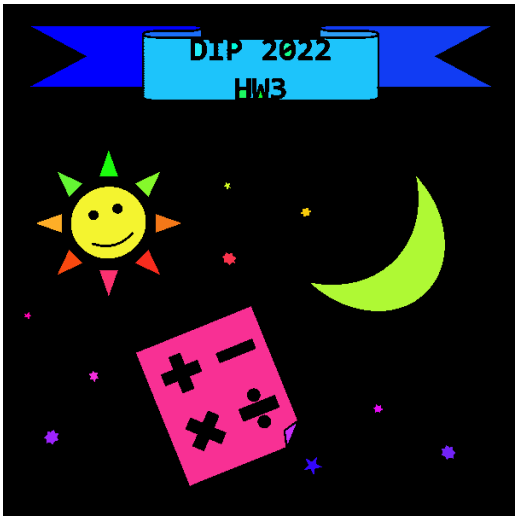




這題跑超久的(大概要十分鐘)
因為 skeletonizing 是依照白點侵蝕，所以 result3.png 是往內縮(很像脫水的樣子)，result4.png 是往外侵蝕白點(很像水吸太多膨脹的樣子)。

我比較喜歡 result4.png，因為小太陽變得很可愛(像向日葵)，文字和加減乘除都很清楚，尤其是 DIP 2022 那裡，很有藝術氣息。

(d) (15 pt) Perform connected component labeling on sample1.png to count the number of objects and obtain an image result5.png where different objects are labeled with different colors. Describe the steps in detail and specify the corresponding parameters.

	
Original image	Output image

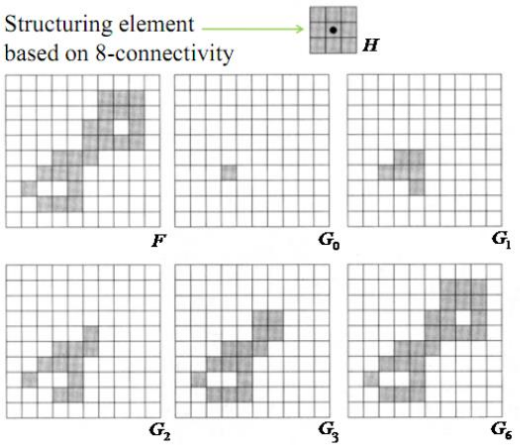
Motivation and approach (include parameters):

因為要計算個數，所以只能每碰到一個 object 就執行 ppt 上的步驟(無法同時進行)，找完後要將 object 從原圖移除，再繼續找下一個 object

```
for i in range(1,h-1):
        for j in range(1,w-1):
            if F[i][j] == 1: # 找到object
                object_count += 1
                # G 代表object
                G = find_object(F, i, j)
                # 將 object 從原圖去掉
                F = F - G
                # 標記 object
                label = object_count + 1
                result5 = result5 + G * label
```

Connected Component Labeling

$$G_i(j,k) = (G_{i-1}(j,k) \oplus H(j,k)) \cap F(j,k) \quad i = 1,2,3,...$$



為了加速所以是從起始點(obj_i)開始偵測，並且只要一偵測到 object 斷開就不往下計算了。

其實跟補洞洞的原理很像，只是這次改成向八個方向長 pixel(H 不同)，再與原圖做交集，將不對的 pixel 去掉。

```
def find_object(F, obj_i, obj_j):
    H = np.ones((3,3))
    h,w = F.shape
    G1 = np.zeros((h,w))
    G1[obj_i, obj_j] = 1

    while True:
        G2 = np.zeros((h,w))
        for i in range(obj_i-1,h-1):
            # 只要一偵測到object斷開就不往下計算
            if (G1[i-1] == 0).all() and (G1[i] == 0).all() and (G1[i+1] == 0).all() and (G2 > 0).any() : break
            for j in range(1,w-1):
                patch = G1[i-1:i+2, j-1:j+2]
                if (patch*H).sum() > 0: G2[i][j] = 1
        G2 = G2 * F
        if (G1==G2).all(): break
        G1 = G2
    return G1
```

顏色的分配是先使用 HSL 色彩空間，以每 360/32 度選取顏色，之後再轉到 RGB 色彩空間得來的。

Discussion of results:
object 是以有無相連來判斷，總共找了 32 個 object

Problem 2: TEXTURE ANALYSIS

In this problem, an image sample2.png of a natural scene is given in Figure. 2(a).

Original image: sample2.png

(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.

Motivation and approach (include parameters):
依照 ppt 的方法依序產生 law1~law9

- **Laws' Method**
  - **//Step 1// Convolution** $M_i(j,k) = F(j,k) \otimes H_i(j,k)$
    - **Micro-structure impulse response arrays (a basis set)**

$H_i(j,k)$

for 3x3 mask, i=1,2,3,…,9

for 5x5 mask, i=1,2,3,…,25

How to choose the mask size?

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

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

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

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

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

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

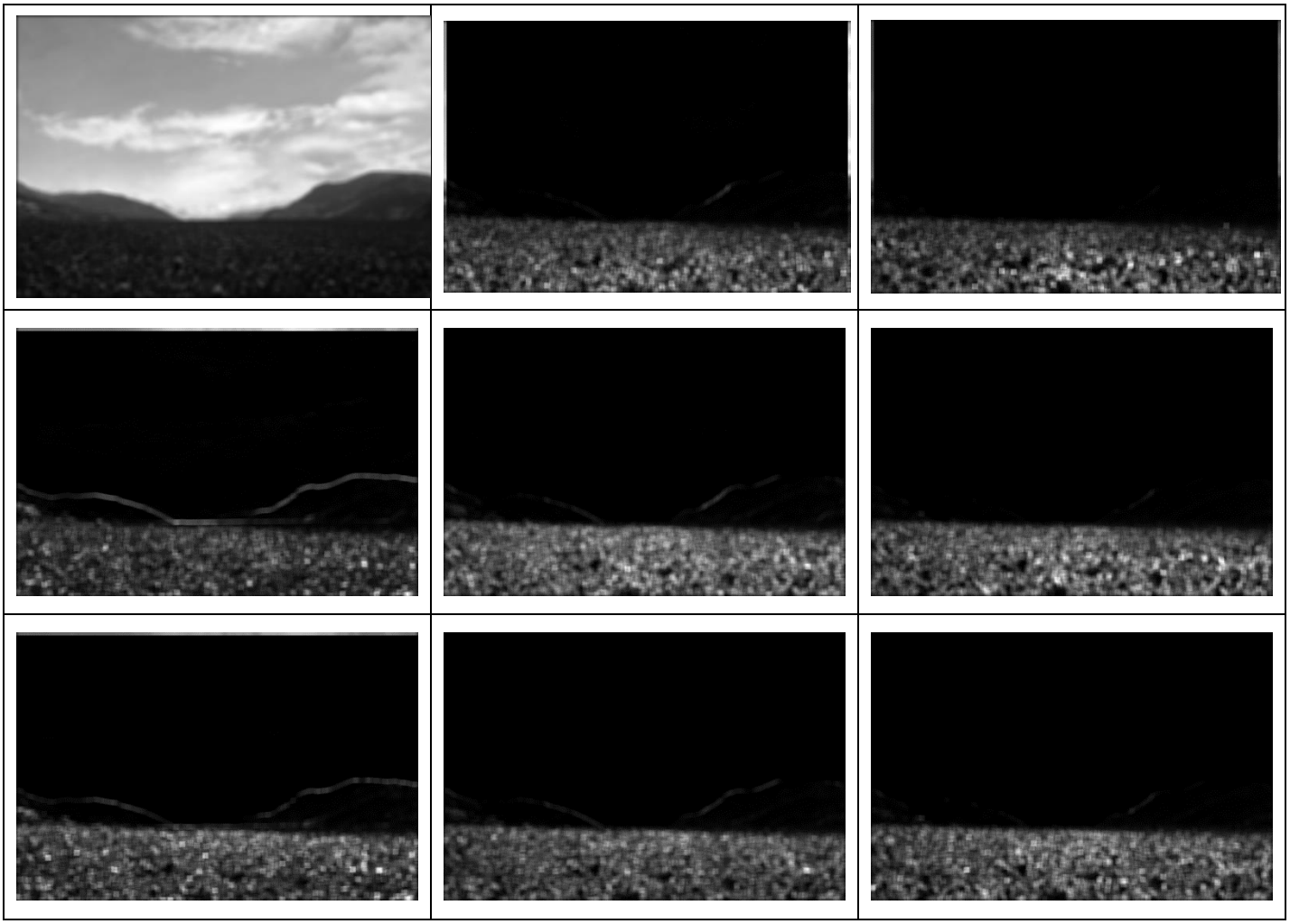
Laws 7: $\frac{1}{12}\begin{bmatrix} -1 & -2 & -1 \\ 2 & 4 & 2 \\ -1 & -2 & -1 \end{bmatrix}$

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

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

17

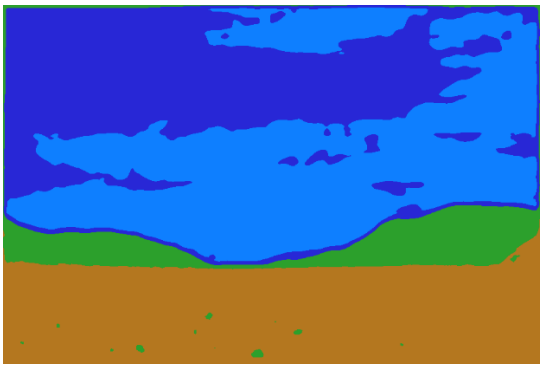
以下的圖從上到下、左到右依序為 law1~law9



Discussion of results:

除了第一章圖(law1)比較明顯以外，其餘的圖大概只能看到花的部分而已，因此我認為最重要的 feature 應該是 law1

(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 result6.png.

	
Original image	Output image

Motivation and approach (include parameters):

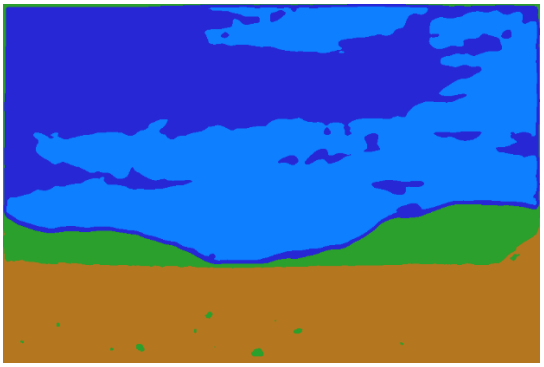
```
# k-means algorithm: https://gist.github.com/tvwerkhoven/4fdc9baad760240741a09292901d3abd
def kMeans(X, K, iterations):
    # Select k vectors as the initial centroids
    centroids = X[np.random.choice(np.arange(len(X)), K)]
    for i in range(iterations):
        # 找出與centroids距離最相近的向量(相減平方最小)
        C = np.array([np.argmin([np.dot(x_i-y_k, x_i-y_k) for y_k in centroids]) for x_i in X])
        print(np.unique(C))
        # check if there are fewer than K clusters.(如果centroids有重複就有可能發生)
        if(len(np.unique(C)) < K):
            centroids = X[np.random.choice(np.arange(len(X)), K)]
        else: #以平均向量作為中心點
            centroids = [X[C == k].mean(axis = 0) for k in range(K)]
    return C

cluster = kMeans(law_method.reshape(-1, 9), 4, 10)
cluster = cluster.reshape(law_method.shape[0], law_method.shape[1])
result6 = np.zeros((cluster.shape[0], cluster.shape[1], 3))
```

觀察完圖後，我將圖片分成 4 個區塊(雲、天空、山、花)，因此我將 K 設為 4。

Discussion of results:

我覺得規劃得還蠻漂亮的，但右邊山與花的邊界沒有取好，花的部分有洞洞。

(c) (20 pt) Based on result6.png, design a method to improve the classification result and output the updated result as result7.png. Describe the modifications in detail and explain the reason why.

	
Original image	Output image

Motivation and approach (include parameters):

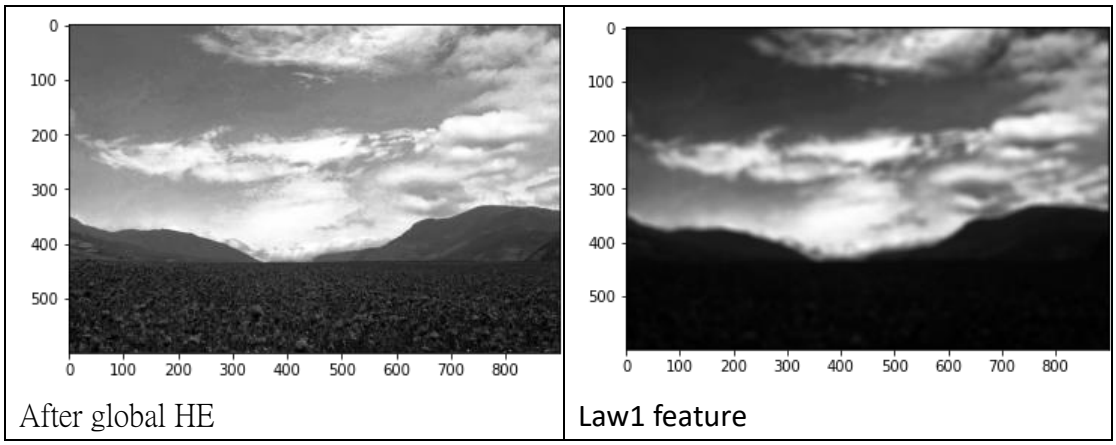
一開始想用用看高斯模糊看能不能把花的洞洞補起來(因為高斯模糊實際上有擴散的作用，應該能將 vector 的資訊與周圍分享，但最後的結果卻不盡人意

花的洞洞不但沒補起來，山與花的界線也快要分不清了，應該是因為高斯將分界上的 vector 也混和了，所以分界變得更加不清楚。

這張照片最關鍵的部分就是要將花與山分開，所以我使用了 global HE 希望能將花與山的差異拉大。

雖然花的洞洞被補起來了，但山與花的界線依舊不明顯。

分析:



global HE 後 Law1 feature 反而變得更不明顯了

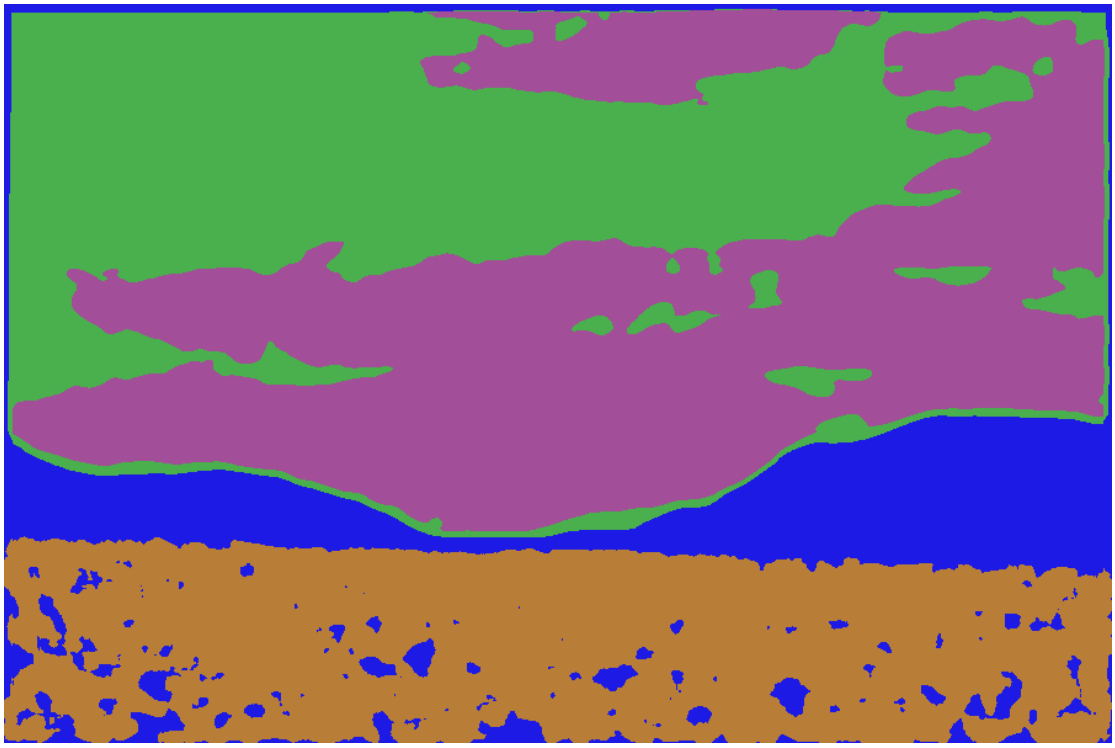
global HE 的結果是亮變更亮，暗變更暗，但對本來對比度就很高的照片就沒什麼用了(如 sample2)，因此這次我決定要使用 pow law 專注提升暗區的亮度

結果就是花的洞洞變得更多了!

最後我想說在算 Energy 時需要平方，數字也蠻大的，不同數據間的標準差也不同，所以我對數據做了一下映射，將數據映射到[0,1]之間

```
def normal(v):
    normalized_v = (v-v.min())/(v.max()-v.min())
    return normalized_v
T1 = normal(T1)
T2 = normal(T2)
T3 = normal(T3)
T4 = normal(T4)
T5 = normal(T5)
T6 = normal(T6)
T7 = normal(T7)
T8 = normal(T8)
T9 = normal(T9)
```

結果:



山與花的邊界終於分開了! 但花的洞洞又變多了…

Discussion of results:
我認為花的莖葉與山也算是同一種 texture(同為綠色植被)，因此也不太能算分錯吧 XD

其實我也有想過老師上課講的，不要一次只看一個 pixel 的 vector，可以連周圍也一起看，但我想說再計算 Energy 時，已經包含了周圍的資訊了，所以覺得意義不大，實做也比較複雜，也很可能會讓山與花又混在一起。

K 的個數我試過設為 5，但反而更糟(山與花合在一起了)。

另外也有想過填洞洞的方法，比如使用 NN 算法，周圍的哪個 label 比較多就分到哪個 label，但這張圖的洞還蠻大的(並不是雜訊或 outlier)，所以我想效果應該也不是很好。

(d) (`Bonus`) Try to perform image quilting, replacing the flowers in sample2.png with sample3.png or other texture you prefer by using the result from (c), and output it as result8.png. It's allowed to utilize external libraries to help you accomplish it, but you should specify the implementation detail and functions you used in the report.

	
Original image	Output image

Motivation and approach (include parameters):

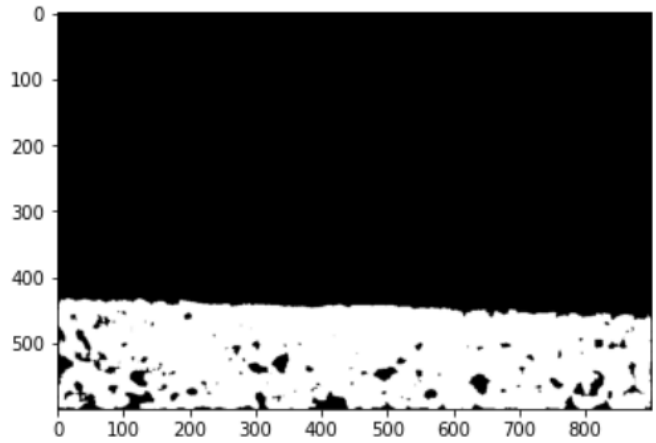
首先先取得 texture 的 label(因為 KMeans 會隨機分配 label 號碼)
作法是取下半部分的眾數

```
counts = np.bincount(cluster[500,400:500])
#返回眾數
label = np.argmax(counts)
print(label)
```

取得 label 後，就可利用 label 選取 texture 的區域，將花的 texture 設為 1，其餘設為 0，即可生成 mask

```
mask = np.where(cluster == label, 1, 0)
plt.imshow(mask, cmap='gray')
```

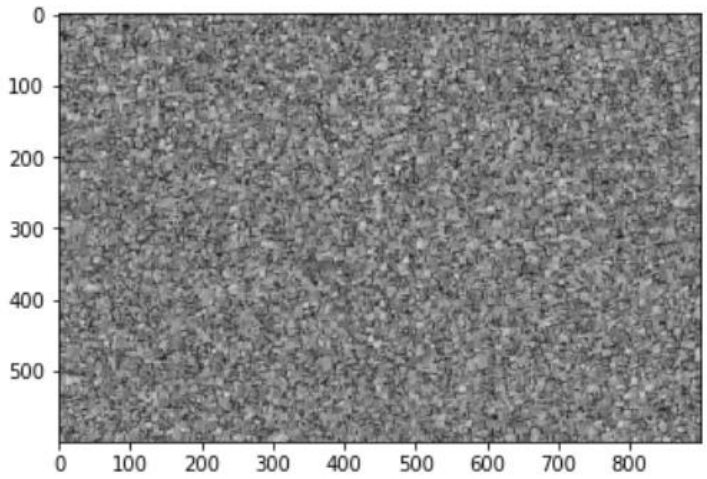
```
<matplotlib.image.AxesImage at 0x1c2396f7f40>
```



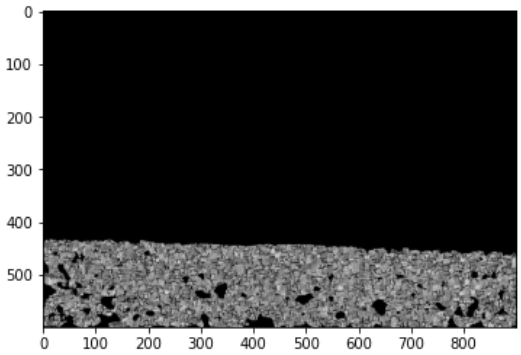
之後我使用了 https://github.com/bglogowski/ImageQuilting 的 quilt_random 方法(因為最簡單且速度最快，做法就是隨機取 sample3 的某一區塊填入)，將原本寬只有 200 的 smaple3 隨機生成到 900。

```
plt.imshow(sample3_texture, cmap='gray')
```

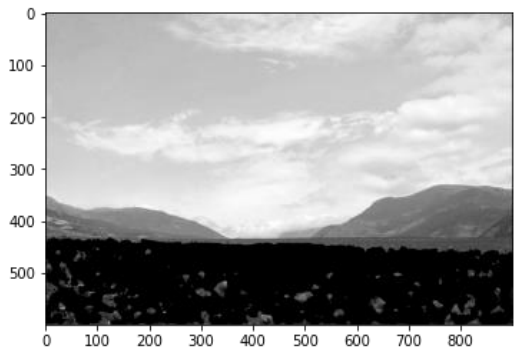
```
<matplotlib.image.AxesImage at 0x1c239acdb20>
```



乘上 mask 後



texture



sample2

再相加就完成了

Discussion of results:

因為 texture 分類沒有分得很完美所以還是有幾朵花殘留 XD