班級：資工三
學號：110590034
姓名：楊榮鈞

```
'''
Modules import
'''
import numpy as np
import cv2

from heapq import heappush, heappop

def mark_on_image(image_path, number):
    '''
    Mark on image by mouse event
    '''
    def draw_circle(event, x, y, flags, param):
        '''
        Draw circle
        '''
        nonlocal img, drawing, radius, color_idx
        if event == cv2.EVENT_LBUTTONDOWN:
            drawing = True
            cv2.circle(img, (x, y), radius, colors[number-1][color_idx], -1)
        elif event == cv2.EVENT_MOUSEMOVE:
            if drawing:
                cv2.circle(img, (x, y), radius, colors[number-1][color_idx], -1)
        elif event == cv2.EVENT_LBUTTONUP:
            drawing = False
```

```
    img = cv2.imread(image_path)

    # Create a window and bind the function to window
    cv2.namedWindow('image')
    cv2.setMouseCallback('image', draw_circle)

    drawing = False  # True if mouse is pressed
    radius = 3  # Initial radius
    colors = [
                [(255, 0, 0), (0, 128, 0), (0, 0, 255), (255, 255, 0)],
                [(255, 0, 0), (0, 128, 0), (0, 0, 255), (255, 255, 0),
                 (255, 0, 255), (0, 255, 255), (255, 140, 0), (128, 128, 128),
                 (128, 0, 0), (128, 0, 128), (0, 128, 128), (192, 192, 192),
                 (255, 165, 0), (255, 192, 203)],
                [(255, 0, 0), (0, 128, 0), (0, 0, 255)]
            ]
    color_idx = 0  # Initial color index
```

```
while True:
    img_with_text = img.copy()
    # Display current color
    cv2.putText(img_with_text, f'Color: {colors[number-1][color_idx]}',
                (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2)
    cv2.imshow('image', img_with_text)
    k = cv2.waitKey(1) & 0xFF
    if k == ord('q'):  # Press 'q' to quit
        break
    elif k == ord('c'):  # Press 'c' to clear canvas
        img = cv2.imread(image_path)
    elif k == ord('+'):  # Increase radius
        radius += 1
    elif k == ord('-'):  # Decrease radius (minimum is 1)
        radius = max(1, radius - 1)
    elif k == ord('n'):  # Change color (next)
        color_idx = (color_idx + 1) % len(colors[number-1])
    elif k == ord('s'):  # Save image
        cv2.imwrite(f'results\img{number}_q1-1.png', img)
        print("Image saved.")

cv2.destroyAllWindows()
```

Function mark_on_image:
在此 Function 中有一個 Function draw_circle 是要讓使用者用滑鼠畫畫時，畫出一個實心圓，其中利用 cv2.EVENT_LBUTTONDOWN、cv2.EVENT_MOUSEMOVE 和 cv2.EVENT_LBUTTONUP 去偵測滑鼠的行為去決定現在是否再畫畫。

接著用 cv2.imread 讀取 image_path 的圖片(原圖)，利用 cv2.namedWindow 和 cv2.setMouseCallback 建立畫畫的視窗。

初始化 drawing(畫畫的狀態)、radius(畫出實心圓的半徑)、colors(三張原圖所需標記的顏色)和 color_idx(colors 的 index)。

接下來用 while 迴圈來執行畫畫的相關行為，首先用 copy()複製一份原圖到 img_with_text 上，然後用 cv2.putText 顯示現在在使用的顏色(RGB)，用 cv2.imshow 顯示圖片。接著使用 cv2.waitKey 和 ord 去根據使用者所輸入的行為(keyboard input)做出不同的動作。
最後在使用 cv2.destroyAllWindows 關掉視窗。

按下 q 時，會關掉視窗；按下 c 時，會清空畫布(還原成沒有標記的狀態)；按下+時，會增加畫筆畫的實心圓半徑；按下-時，會減少畫筆畫的實心圓半徑；按下 n 時，切換到下一個顏色；按下 s 時，會儲存圖片(標記的圖片)。

```python
def calculate_priority(y, x, origin_image, seeds, method=1):
    '''
    Calculate priority
    '''
    height = origin_image.shape[0]
    width = origin_image.shape[1]
    priority = 0
    neighbors = []
    for i in range(-1, 2):
        for j in range(-1, 2):
            if 0 <= y+i < height and 0 <= x+j < width\
              and (i != 0 or j != 0):
                neighbors.append((y+i, x+j))
    neighbors_rgb = [list(origin_image[i][j]) for i, j in neighbors]
    neighbors_rgb = np.array(neighbors_rgb)
    mean_rgb = np.mean(neighbors_rgb, axis=0)
    variance_rgb = np.var(neighbors_rgb, axis=0)

    # first priority method
    if method == 1:
        priority = 0.5 * mean_rgb.sum() + 0.5 * variance_rgb.sum()
    # second priority method
    elif method == 2:
        neighbors_distance = np.linalg.norm(neighbors_rgb - origin_image[y][x], axis=1)
        seeds_rgb = [list(origin_image[i][j]) for i, j in seeds]
        seeds_rgb = np.array(seeds_rgb)
        seeds_distance = np.linalg.norm(seeds_rgb - origin_image[y][x], axis=1)
        priority = 0.5 * mean_rgb.sum() + 0.5 * variance_rgb.sum() + 0.5 * np.min(neighbors_distance) + 0.75 * np.min(seeds_distance)
    # third priority method
    else:
        seeds_rgb = [list(origin_image[i][j]) for i, j in seeds]
        seeds_rgb = np.array(seeds_rgb)
        seeds_distance = np.linalg.norm(seeds_rgb - origin_image[y][x], axis=1)
        priority = 0.5 * variance_rgb.sum() + 0.5 * np.min(seeds_distance)
    # print(priority)
    return priority
```

Function calculate_priority:
首先用 shape 找出 origin_image 的 height 和 width，然後初始化 priority 和建立 neighbors，接著用 for 迴圈找到 nieghbors。

然後我們找出 neighbor 的 RGB 的平均值和 variance (mean_rgb、variance_rgb)，然後再根據 method 決定接下來的步驟。
當 method 為 1，priority = 0.5 * mean_rgb.sum() + 0.5 * variance_rgb.sum()。
當 method 為 2，會先計算當前 pixel 對 neighbors 的 RGB value 的 distance 到 neighbors_distance，然後再計算當前 pixel 對 seeds 的 RGB value 的 distance 到 seeds_distance，priority = 0.5 * mean_rgb.sum() + 0.5 * variance_rgb.sum() + 0.5 * np.min(neighbors_distance) + 0.75 * np.min(seeds_distance)。
當 method 為其他值(ex: 3)，會計算當前 pixel 對 seeds 的 RGB value 的 distance 到 seeds_distance，priority = 0.5 * variance_rgb.sum() + 0.5 * np.min(seeds_distance)。

最後回傳 priority。

```python
def colorize_watershed(origin_image, label_map, colors, number):
    '''
    colorize watershed
    '''
    height = origin_image.shape[0]
    width = origin_image.shape[1]
    o_img = np.array(origin_image).astype(np.float64)
    colors = np.array(colors[number-1]).astype(np.float64)
    region_image = np.zeros((height, width, 3))
    for row in range(height):
        for col in range(width):
            if label_map[row][col] > 0:
                region_image[row][col] = colors[label_map[row][col]-1]
            else:
                region_image[row][col] = [0, 0, 0]
    watershed_image = np.zeros((height, width, 3))
    region_image = region_image.astype(np.float64)
    print(colors)
    print(np.unique(label_map))
    for row in range(height):
        for col in range(width):
            if label_map[row][col] > 0:
                watershed_image[row][col] = o_img[row][col] * 0.5 + region_image[row][col] * 0.5
    cv2.imshow('Region Image', region_image.astype(np.uint8))
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    return watershed_image
```

Function colorize_watershed:
利用 colors(標記用的顏色)和 label_map 先產生出 region_image，在利用 region_image 和 origin_image 在 watershed_image 上產生 origin_image 加上 region 的薄霧顏色。

首先用 shape 找出 origin_image 的 height 和 width，在將 origin_image 用 astype 轉成 np.float64 到 o_img 上，然後將 colors 轉乘 np.array 並使用 np.float64。

接著利用 np.zeros 建立 region_image 的 array，然後用 for 迴圈根據 label_map 的值決定 region_image 的顏色(當 label_map[row][col]>0 時會將當前的 pixel 轉成 label 對應的 colors 中的顏色，若是小於等於 0 會直接設定為 RGB(0,0,0))。

利用 np.zeros 建立 watershed_image 的 array，將 region_image 用 astype 設定成 np.float64，然後用 for 迴圈將 label 大於 0(label_map[row][col]>0)的 pixel 設定成 o_img 對應的 pixel*0.5+region_image 對應的 pixel*0.5，使 label 小於等於 0 的值直接設成 RGB(0,0,0)。

用 cv2.imshow 觀察 region_image，然後用 cv2.waitKey(0)和 cv2.destroyAllWindows()關閉視窗

最後回傳 watershed_image。

```python
def watershed(origin_image, marked_image, number, priority_method=1):
    '''
    watershed
    '''
    colors = [
        [[255, 0, 0], [0, 128, 0], [0, 0, 255], [255, 255, 0]],
        [[255, 0, 0], [0, 128, 0], [0, 0, 255], [255, 255, 0], [255, 0, 255], [0, 255, 255], [255, 140, 0],
         [128, 128, 128], [128, 0, 0], [128, 0, 128], [0, 128, 128], [192, 192, 192], [255, 165, 0], [255, 192, 203]],
        [[255, 0, 0], [0, 128, 0], [0, 0, 255]]
    ]
    height = origin_image.shape[0]
    width = origin_image.shape[1]
    label_map = np.zeros((height, width), dtype=np.int64)
    print(label_map.shape[0])
    print(label_map.shape[1])
    seeds = []
    # 1-1 mark area
    for row in range(height):
        for col in range(width):
            if list(marked_image[row][col]) in colors[number-1]:
                label_map[row][col] = colors[number-1].index(list(marked_image[row][col])) + 1
                seeds.append((row, col))
    print('mark area done')
    # print(seeds)
```

```python
    # 1-2 region growing
    # create priority queue needed for region growing
    priority_queue = []
    for seed in seeds:
        heappush(priority_queue, (calculate_priority(seed[0], seed[1], origin_image, seeds, priority_method), seed))
    print('add seeds in priority queue done')
    neighbors_direction = [(0, 1), (1, 0), (0, -1), (-1, 0)]
    count  = 0
    print('start region growing:')
    while len(priority_queue) != 0:
        priority, (y, x) = heappop(priority_queue)
        count += 1
        print(count)
        neighbors = []
        for direction in neighbors_direction:
            n_y = y + direction[0]
            n_x = x + direction[1]
            if 0 <= n_y < height and 0 <= n_x < width:
                neighbors.append((n_y, n_x))
        neighbors_label = [label_map[i][j] for i, j in neighbors]
        unique_mark_label = [label for label in np.unique(neighbors_label) if label > 0]
        if label_map[y][x] == -2:
            # mark label count > 2 or = 0
            if len(unique_mark_label) > 1 or len(unique_mark_label) == 0:
                label_map[y][x] = -1
                continue
            if len(unique_mark_label) == 1:
                label_map[y][x] = unique_mark_label[0]
        for i, j in neighbors:
            if label_map[i][j] == 0:
                label_map[i][j] = -2
                heappush(priority_queue, (calculate_priority(i, j, origin_image, seeds, priority_method), (i, j)))

    watershed_image = colorize_watershed(origin_image, label_map, colors, number)
    return watershed_image
```

Function watershed:

初始化 colors(建立標記的顏色)，利用 shape 找到 origin_image 的 height 和 width，使用 np.zeros 建立 label_map，建立 seeds 儲存一開始標記的位子。

用 for 迴圈找到 marked_image 標記的位置，並將對應到此位置的 label_map 設為對應 colors 的顏色的 index+1，並將此位置存入 seeds。

建立 priority_queue，然後先將 seeds 裡面 seed 的值用 heapq 的 heappush 將 calculate_priority 的值和當前的 pixel 位置存入 priority_queue 中。

建立 neighbors_direction(4 neighbor，所以是右上左下)，並初始化 count(用來觀察 priority_queue 處理了多少 pixel)。

利用 while 迴圈 label，當 priority_queue 為 empty 時跳出迴圈。首先先使用 heapq 的 heappop 將 priority 最高(值最小)的 pixel 的位置取出，然後用 for 迴圈尋找這 pixel 的 neighbor 並將 neighbor 的位置存入 neighbors 中。
將 neighbor 對應到 label_map 的 label 存入 neighbors_label 中。
利用 neighbors_label 找出大於 0 的 unique label 存入 unique_mark_label 中。
如果當前的 pixel 對應到 label_map 的值為-2 (in queue)時，會判斷周圍是否有被標記的 label(>0)，當被標記的 label 的數量超過 1 個或是沒有被標記的 label 時，當前 pixel 對應到 label_map 的值會設為-1 (edge)，並進入下個迴圈；若是 neighbor 中被標記的 label 數量只有 1 個，那當前 pixel 對應到 label_map 的值會設為那個 label 的值。
然後用 for 迴圈判斷 neighbor 對應到 label_map 的值是否為 0 (unmark)，如果是的話會將對應到 label_map 的值設為-2 (in queue)，並使用 heapq 的 heappush 將 calculate_priority 的值和 neighbor 的 pixel 位置存入 priority_queue 中。

當 priority_queue 為 empty，然後跳出迴圈後，用 colorize_watershed 將 origin_image、標記好的 label_map、colors 和 number(當前原圖的編號)轉換成 watershed_image。
最後回傳 watershed_image。

```
def image(number, priority_method=1):
    '''
    For img{number}.png
    '''
    origin_image = cv2.imread(f'images/img{number}.jpg')
    mark_on_image(f'images/img{number}.jpg', number)
    marked_image = cv2.imread(f'results/img{number}_q1-1.png')
    watershed_image = watershed(origin_image, marked_image, number, priority_method)
    cv2.imwrite(f'results/img{number}_q1.jpg', watershed_image)
    segmented_image = cv2.imread(f'results/img{number}_q1.jpg')

    cv2.imshow('Origin Image', origin_image)
    cv2.imshow('Marked Image', marked_image)
    cv2.imshow('Watershed Image', segmented_image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Function image:
先用 cv2 的 imread 讀取圖片到 origin_image，然後傳入 origin_image 的 path 到 mark_on_image 中。

接著用 cv2 的 imread 讀取 marked_image，然後將 origin_image 和 marked_image、number 還有 priority_method 傳入 watershed 進行 watershed segmentation，並將輸出結果存到 watershed_image。

接下來用 cv2 的 imwrite 儲存圖片 ，再用 cv2.imread 讀取 segmented_image(用 watershed 過後的 image)。

利用 cv2 的 imshow 查看 original image、marked image 和 watershed image。
利用 cv2 的 waitKey(0)和 destroyAllWindows()將 imshow 顯示出來的圖片關閉。

```
if __name__ == '__main__':
    image(number=1, priority_method=3)
    image(number=2, priority_method=3)
    image(number=3, priority_method=2)
```

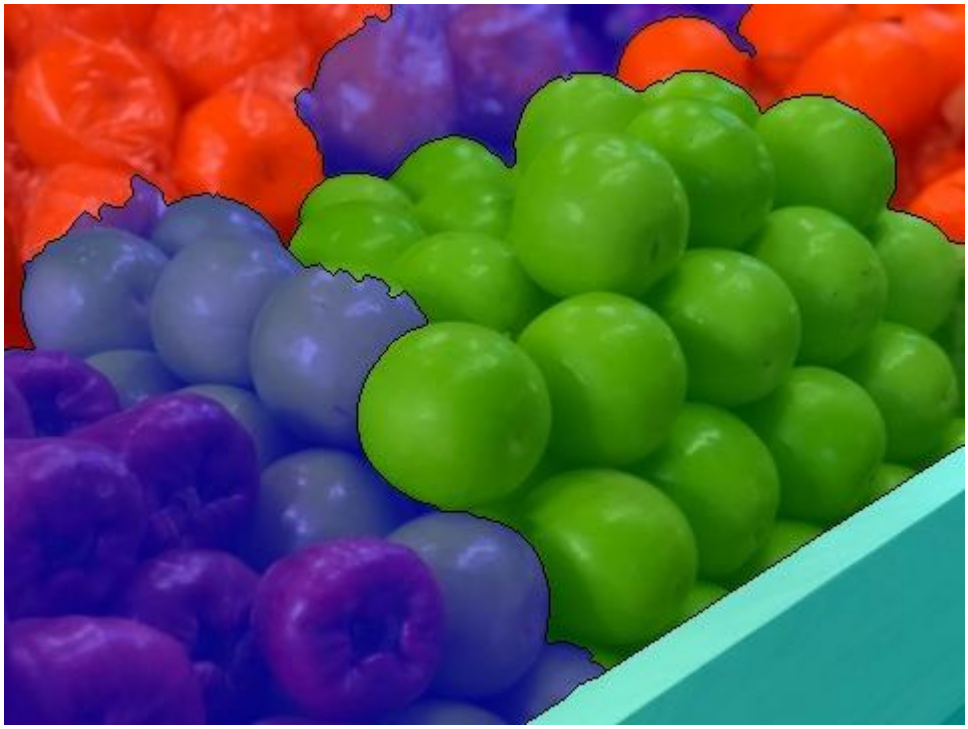
執行 image，產生所有助教給的圖片的 marked image 和 watershed image。

Result images

img1_q1-1.png

img1_q1.jpg (results 裡面的是第三種 priority 算法)

priority = 0.5 * mean_rgb.sum() + 0.5 * variance_rgb.sum()



我預期的結果跟最後的輸出不太相同，藍色的 region 超出的範圍有點多。

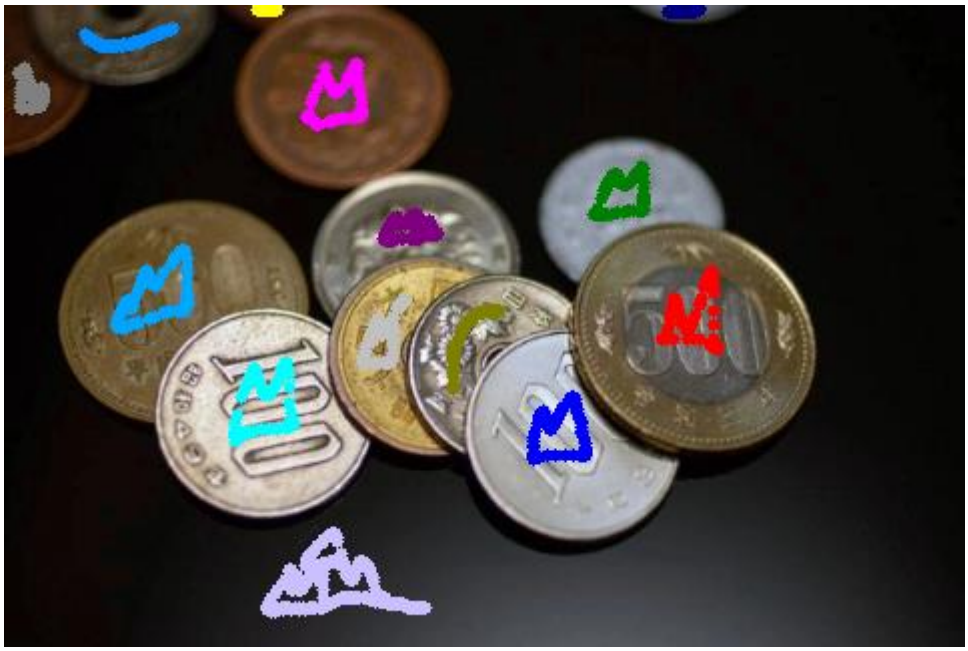
priority = 0.5 * mean_rgb.sum() + 0.5 * variance_rgb.sum() + 0.5 * np.min(neighbors_distance) + 0.75 * np.min(seeds_distance)

我預期的結果跟最後的輸出不太相同，相比於第一種的 priority 的圖上面藍色 region 的部分標示的範圍已經很漂亮了，但是又下的藍色 region 仍然有超出範圍的問題，且有一部份的 region 被綠色 region 給覆蓋掉了。

priority = 0.5 * variance_rgb.sum() + 0.5 * np.min(seeds_distance)

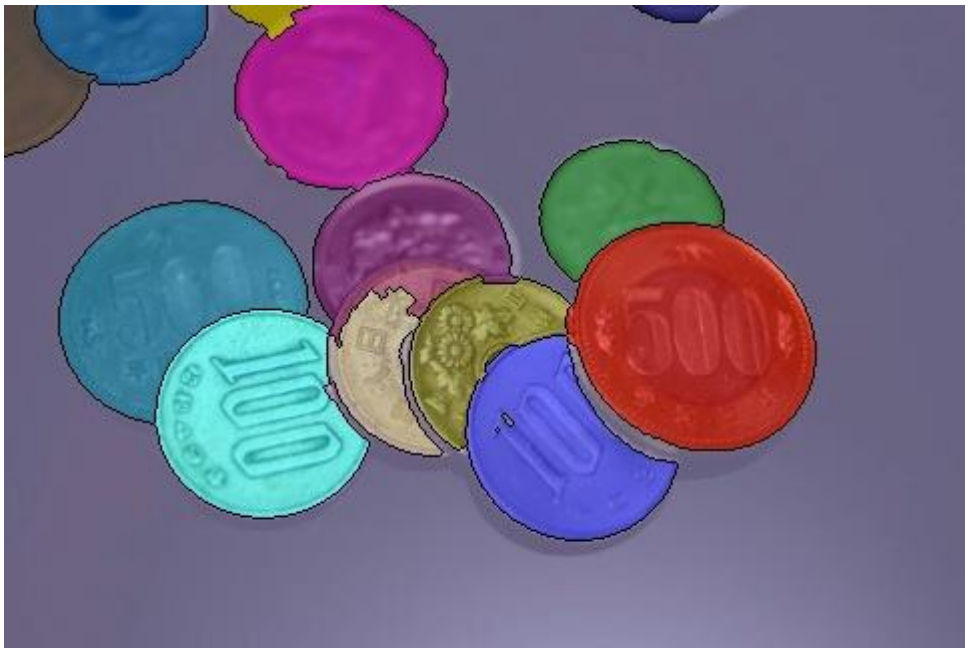
我預期的結果跟最後的輸出大致相同，只有下面的藍色 region 有覆蓋到一點綠色 region，綠色 region 也有覆蓋到一點藍色 region。

img2_q1-1.png



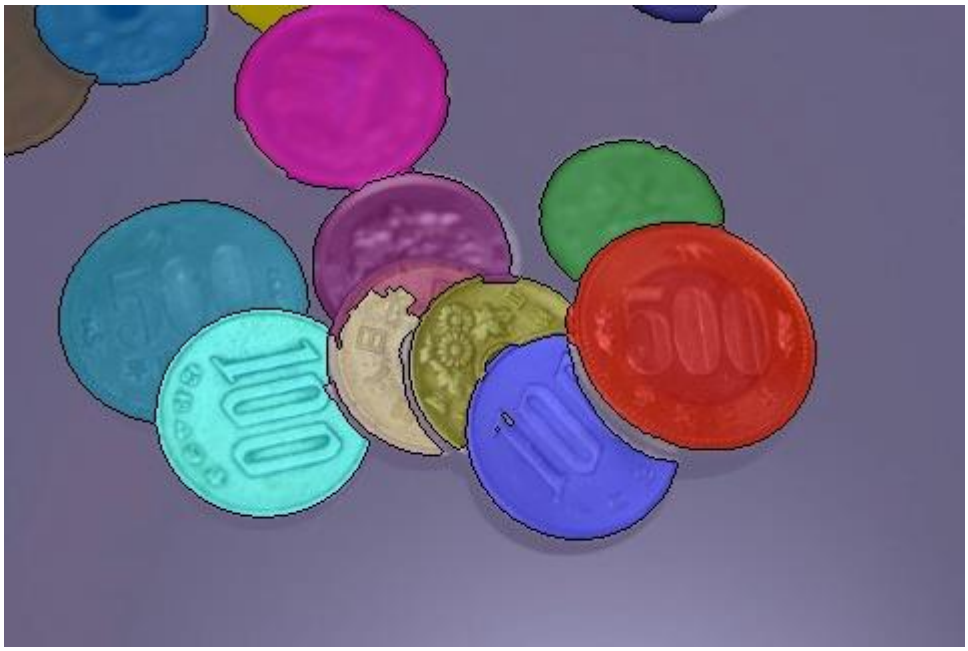
img2_q1.jpg (results 裡面的是第三種 priority 算法)

priority = 0.5 * mean_rgb.sum() + 0.5 * variance_rgb.sum()



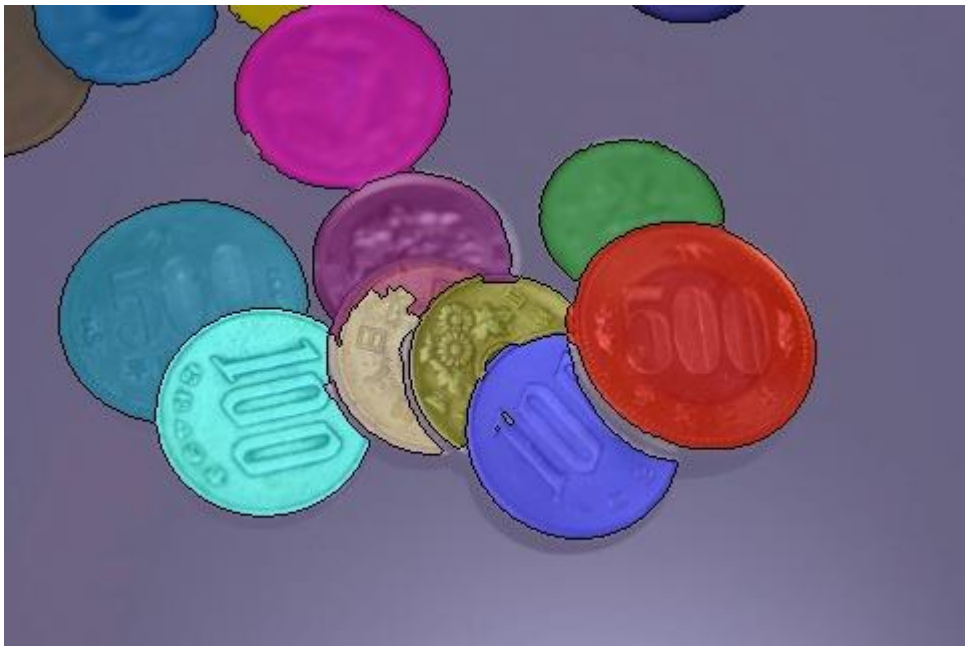
我預期的結果跟最後的輸出大致上相同，只有中間紫色 region 的硬幣有點超出範圍。

priority = 0.5 * mean_rgb.sum() + 0.5 * variance_rgb.sum() + 0.5 * np.min(neighbors_distance) + 0.75 * np.min(seeds_distance)



我預期的結果跟最後的輸出大致上相同，只有中間紫色 region 的硬幣有點超出範圍，上面粉紅 region 的硬幣的 region 有較完整一點。

priority = 0.5 * variance_rgb.sum() + 0.5 * np.min(seeds_distance)



我預期的結果跟最後的輸出大致上相同，只有中間紫色 region 的硬幣有點超出範圍，上面粉紅 region 的硬幣的 region 有較完整一點，右上的硬幣 region 有完整覆蓋到(相對於前兩個 priority 算法)。

img3_q1-1.png

img3_q1.jpg (results 裡面的是第二種 priority 算法)

priority = 0.5 * mean_rgb.sum() + 0.5 * variance_rgb.sum()

我預期的結果跟最後的輸出大致上相同，樹的只有樹葉較稀疏的有些部分被分到紅色的 region。

priority = 0.5 * mean_rgb.sum() + 0.5 * variance_rgb.sum() + 0.5 * np.min(neighbors_distance) + 0.75 * np.min(seeds_distance)

我預期的結果跟最後的輸出大致上相同，左邊的樹相比於第一個 priority 的算法有被藍色的 region 更完整的覆蓋到。

priority = 0.5 * variance_rgb.sum() + 0.5 * np.min(seeds_distance)

我預期的結果跟最後的輸出不太相同，左邊的樹的部分沒有像第二個 priority method 這麼好。

最後的結果總結：

這次 hw4 的第一個難點，我覺得是 mark image 的部分，因為不熟悉 cv2 畫圖的功能，所以花了很多時間在上面。第二個難點是 q1-2 的 region growing 的部分，其中最難的部分是 priority queue 的 priority 要如何設定。而這次作業我覺

得最精華的部分也是 priority 的部分，首先我 priority 有先用過 neighbor 的 RGB 平均值*0.5+neighbor 的 RGB 的 variance*0.5，這樣的結果在作業的第一張圖上會看到有些 region 會超出範圍，第二張圖中桌面的另一邊會沒有標記到，而在第三張圖中 region 超出範圍的影響較小。
第二次我用的 priority 是 neighbor 的 RGB 平均值*0.5+neighbor 的 RGB 的 variance*0.5+min(當前 pixel 的 RGB 到 neighbors 的 RGB 的 distance)*0.5+min(當前 pixel 的 RGB 到 seeds 的 RGB 的 distance)*0.75，作業的第一張圖的結果的上半部分的藍色 region 有變的沒超出範圍，但是底下藍色 region 的部分仍有些許超出範圍，且有些被綠色 region 覆蓋到，第三張圖的左邊的樹的 region 有比較完整。
第三次我用的 priority 是 neighbor 的 RGB 的 variance*0.5 + min(當前 pixel 的 RGB 到 seeds 的 RGB 的 distance)*0.5，我覺得前兩張圖有 region 的比較好一點，但是第三張圖就沒有第二次用的 priority 這麼好了。

其中使用第二種和第三種方法花的時間會比較久，除非畫筆在 image 上 mark 的時候把圓變小一點，不然在 priority 算 seeds distance 時會算很久，因為畫筆比較粗的話，那 seed 就會變多，而 priority_method 2 和 3 都會計算 pixel 到每個 seed 的 RGB value 的 distance，目前 img1_q1-1.png、img2_q1-1.png 和 img3_q1-1.png 產生每張個圖片的時間為 40~50min。

我還有發現 priority 也可以配合老師剛剛教到的 sebol 來去做 priority 的計算。
我的計算 priority 沒有使用到降階，如果使用降階，可以減少產生圖片的時間，然後 mark image 的部分也要注意轉成 jpg 還是 png，轉成 jpg 時，會有些標記的 label 顏色會變淡，而 png 則沒有問題。
除了 priority 的部分外，後來也有發現當 Mark 的位置不同時，也會影響到 watershed 的結果(因為 priority 的算法會導致結果整個不一樣)。