班級：資工三
學號：110590034
姓名：楊榮鈞

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

def rgb_to_gray(image):
    '''
    Convert color image to grayscale image
    '''
    gray_image = np.dot(image[..., :3], [0.3, 0.59, 0.11])
    return gray_image.astype(np.uint8)

def gray_to_binary(image, threshold):
    '''
    Convert image to binary image
    '''
    binary_image = np.where(image > threshold, 0, 255)
    return binary_image.astype(np.uint8)
```

Function rgb_to_gray:
使用 numpy 的 dot 去實現。
rgb 轉灰階是利用 hw1 的公式$(0.3\times R)+(0.59\times G)+(0.11\times B)$去轉成灰階。
最後用 astype(np.uint8)是因為現在的的範圍是 0~255。
回傳 gray_image。
Function gray_to_binary:
使用 numpy 的 where 去實現。
灰階轉 binary 是當大於 threshold 時，會設成 0，要是小於等於 threshold，則是設成 255，也就是背景會設為黑色。
最後用 astype(np.uint8)是因為現在的的範圍是 0~255。
回傳 binary_image。

```python
def distance_transform(image, connectivity=4):
    '''
    Distance transform with 4, 8 neighbors
    '''
    height, width = image.shape
    distance_map = np.zeros_like(image, dtype=np.uint16)
    distance_map = np.where(image == 255, 1, 0)

    # print(distance_map)
    if connectivity == 4:
        top_left_neighbors = [(-1, 0), (0, -1)]
    else:
        top_left_neighbors = [(-1, 0), (-1, -1), (-1, 1), (0, -1)]

    for i in range(height):
        for j in range(width):
            if image[i, j] == 0:
                continue
            neighbors = []
            for n_y, n_x in top_left_neighbors:
                if 0 <= (i + n_y) < height and 0 <= (j + n_x) < width:
                    neighbors.append(distance_map[i + n_y, j + n_x])
            # print(len(neighbors))
            if neighbors:
                distance_map[i, j] = min(neighbors) + 1
```

```python
    # print(distance_map)
    if connectivity == 4:
        bottom_right_neighbors = [(1, 0), (0, 1)]
    else:
        bottom_right_neighbors = [(1, 0), (1, 1), (1, -1), (0, 1)]

    for i in range(height - 1, -1, -1):
        for j in range(width - 1, -1, -1):
            if image[i, j] == 0:
                continue
            neighbors = []
            for n_y, n_x in bottom_right_neighbors:
                if 0 <= (i + n_y) < height and 0 <= (j + n_x) < width:
                    neighbors.append(distance_map[i + n_y, j + n_x])
            # print(len(neighbors), neighbors)
            if neighbors:
                distance_map[i, j] = min(min(neighbors) + 1, distance_map[i, j])

    return distance_map
```

Function distance_transform:

首先用 shape 找出圖片的高和寬，然後用 numpy 的 zeros_like 建立 distance_map 的 array，其中 dtype=np.unit16 是因為不確定會有標記的 distance 會有多少，所以先設定 np.unit16。

接著用 np.where 把 binary 的 image 的值複製到 distance_map 上，從而進行初始化，其中 255 會換成 1 來標記。

然後我用來計算 distance transform 的方式是：
根據 connectivity 的值決定 top_left_neighbors 為哪些方向的值。
如果 connectivity 為 4，top_left_neighbors 會儲存 top 和 left。
如果 connectivity 為 8，top_left_neighbors 會儲存 top、top left、top right 和 left。
先從左上 scan 到右下，如果掃到的 image[i, j]的 value 不為 0 的話，會根據 top_left_neighbors 找出 image 的 neighbors，然後把 image[i, j]的值設為 neighbors 的最小值+1。

根據 connectivity 的值決定 bottom_right_neighbors 為哪些方向的值。
如果 connectivity 為 4，bottom_right_neighbors 會儲存 bottom 和 right。
如果 connectivity 為 8，bottom_right_neighbors 會儲存 bottom、bottom right、bottom left 和 right。
再從右下 scan 到左上，如果掃到的 image[i, j]的 value 不為 0 的話，會根據 bottom_right_neighbors 找出 image 的 neighbors，然後把 image[i, j]的值設為 neighbors 最小值+1 和 image[i, j]的最小值，
（會跟自己本身的值找最小值的原因是 image[i, j]本身的值是已經從左上 scan 到右下的結果，所以在右下 scan 到左上時，需要把從左上 scan 到右下的結果考慮進去。）

最後回傳 distance_map

```python
def colorize_distance_transform(distance_image):
    '''
    Colorize distance transform image
    '''
    max_value = np.max(distance_image)
    colors_distance_image = (distance_image / max_value * 255).astype(np.uint8)
    return colors_distance_image
```

Function colorize_distance_transform:
首先會用 np.max 找出 distance_image 的 max_value。
然後用 max_value 去 normalize 再乘上 255 使 distance_image 的 value 在 0~255 內，其中有使用 astype(np.unit8)，因為現在 distance_image 的 value 範圍是 0~255。
回傳 colors_distance_image。

```python
def is_connect_point(neighbors):
    '''
    Check connectivity
    '''
    # row 1, row 3
    if sum(neighbors[0]) > 0 and sum(neighbors[2]) > 0 and sum(neighbors[1]) == 0:
        return True
    # col 1, col 3
    if sum(neighbors[:, 0]) > 0 and sum(neighbors[:, 2]) > 0 and sum(neighbors[:, 1]) == 0:
        return True
    # neighbors which is lonely
    neighbor_direction = [(-1, 0), (-1, 1), (0, 1), (1, 1),
                          (1, 0), (1, -1), (0, -1), (-1, -1)]
    for y in range(3):
        for x in range(3):
            if neighbors[y, x] != 0:
                n_neighbors = []
                for n_y, n_x in neighbor_direction:
                    if 0 <= y+n_y < 3 and 0 <= x+n_x < 3:
                        n_neighbors.append(neighbors[y+n_y, x+n_x])
                if sum(n_neighbors) == 0:
                    return True
    # hw example
    if neighbors[0, 0] != 0 and neighbors[1, 2] != 0 and neighbors[2, 1] != 0 and\
       neighbors[0, 1] == 0 and neighbors[1, 0] == 0:
        return True
    if neighbors[0, 2] != 0 and neighbors[1, 0] != 0 and neighbors[2, 1] != 0 and\
       neighbors[0, 1] == 0 and neighbors[1, 2] == 0:
        return True
    if neighbors[0, 1] != 0 and neighbors[1, 2] != 0 and neighbors[2, 0] != 0 and\
       neighbors[1, 0] == 0 and neighbors[2, 1] == 0:
        return True
    if neighbors[0, 1] != 0 and neighbors[1, 0] != 0 and neighbors[2, 2] != 0 and\
       neighbors[1, 2] == 0 and neighbors[2, 1] == 0:
        return True

    return False
```

Function is_connect_point:

funciton 主要功能是確認 image 的 pixel 是否為不可刪除的 pixel（刪除會導致沒有 connectivity）。以下 condition 都是從 neighbor 去看。

第一個 condition 是考慮到 row1 和 row3 有 value 且 row2 沒有 value 的狀況。

第二個 condition 是考慮到 column1 和 column3 有 value 且 column2 沒有 value 的狀況。

第三個 condition 是考慮到有 neighbor 周圍沒有自身的 neighbor 的狀況。（自身的 neighbor 不包括我們呼叫此 function 的點）

第四、五、六、七個 condition 是 HW3.pdf 中的作業的提示。

```python
def medial_axis(distance_image):
    '''
    Medial axis
    '''
    local_maximum_connectivity_image = np.copy(distance_image)
    rows, cols = distance_image.shape

    # find local maximum and keep connectivity
    for label in range(1, np.max(distance_image)+1):
        for i in range(1, rows-1):
            for j in range(1, cols-1):
                if distance_image[i, j] == label:
                    neighborhood = []
                    kernel = np.zeros((3, 3), dtype=np.uint16)
                    for u in range(i-1, i+2):
                        for v in range(j-1, j+2):
                            if u != i or v != j:
                                neighborhood.append((u, v))
                                kernel[u-i+1, v-j+1] = local_maximum_connectivity_image[u, v]
                            else:
                                kernel[u-i+1, v-j+1] = 0
                    max_local_distance = max([distance_image[u, v] for u, v in neighborhood])

                    if distance_image[i, j] < max_local_distance and not is_connect_point(kernel):
                        local_maximum_connectivity_image[i, j] = 0

    # remove extra redundancy for:
    # 1 1 0
    # 0 1 1
    for i in range(1, rows-1):
        for j in range(1, cols-1):
            if (distance_image[i, j-1] != 0 and distance_image[i, j] != 0) or\
               (distance_image[i, j] != 0 and distance_image[i, j+1] != 0) or\
               (distance_image[i-1, j] != 0 and distance_image[i, j] != 0) or\
               (distance_image[i, j] != 0 and distance_image[i+1, j] != 0) :
                kernel = np.zeros((3, 3), dtype=np.uint16)
                for u in range(i-1, i+2):
                    for v in range(j-1, j+2):
                        if u != i or v != j:
                            kernel[u-i+1, v-j+1] = local_maximum_connectivity_image[u, v]
                        else:
                            kernel[u-i+1, v-j+1] = 0
                if not is_connect_point(kernel):
                    local_maximum_connectivity_image[i, j] = 0

    skeleton_image = np.where(local_maximum_connectivity_image > 0, 255, 0).astype(np.uint8)
    return skeleton_image
```

Function medial_axis:
Function 的主要功能是找到 skeleton。
首先會用 np.copy()複製 distance transform 過的 image（or map）到 local_maximum_connectivity_image，然後用 shape 找到 image 的 rows（height）跟 cols（width）。

接下來有兩個大迴圈。第一個大迴圈我們會從最小的距離找 local_maximum 到最大的距離，當我們找到對應的距離的點的時候，我們會儲存他們的 neighbor 到 neighborhood 和 kernel 中，然後我們會用 neighborhood 找到 neighbor 中距離最大的值，接著我們會判斷這個點是否小於 neighbor 中距離最大的值和是否為不是連接點（不可刪除的點），如果符合條件的話，就會把這個點的 value 設

成 0（把這個點刪掉）。
第二個大迴圈會檢查是否有出現一些可以刪除的點但是沒有刪掉的狀況，其中考慮到的會是左、右、上或下有連起來的狀況，我們一樣會把 neighbors 存到 kernel 再去判斷這個檢查的點是否可以刪除，如果可以我們就會把點設成 0。

再跑完大迴圈後利用 np.where 把 local_maximum_connectivity_image 非 0 的 value 設成 255 並用 astype(np.uint8)設定 value 範圍，存到 skeleton_image 中。

最後再回傳 skeleton_image。

```
def image(number, threshold):
    '''
    For img{number}.png
    '''
    image = cv2.imread(f'images/img{number}.jpg')
    gray_image = rgb_to_gray(image)
    binary_image = gray_to_binary(gray_image, threshold)
    distance_image_4 = distance_transform(binary_image, connectivity=4)
    distance_image_8 = distance_transform(binary_image, connectivity=8)

    colors_distance_image_4 = colorize_distance_transform(distance_image_4)
    colors_distance_image_8 = colorize_distance_transform(distance_image_8)
    skeleton_image = medial_axis(distance_image_4)

    cv2.imshow('Original Image', image)
    cv2.imshow('Grayscale Image', gray_image)
    cv2.imshow('Binary Image', binary_image)
    cv2.imshow('Distance Transform Image With 4 Connectivity', colors_distance_image_4)
    cv2.imshow('Distance Transform Image With 8 Connectivity', colors_distance_image_8)
    cv2.imshow('Skeleton Image', skeleton_image)
    cv2.imwrite(f'results/img{number}_q1-1_4.jpg', colors_distance_image_4)
    cv2.imwrite(f'results/img{number}_q1-1_8.jpg', colors_distance_image_8)
    cv2.imwrite(f'results/img{number}_q1-2.jpg', skeleton_image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Function image:
先用 cv2 的 imread 讀取圖片到 image，然後利用 rgb_to_gray 把 image 轉成 gray_image，再利用 gray_to_binary 把 gray_image 轉成 binary_image。

接下來用 distance_transform 把 binary_image 轉成 distance_image_4 和 distance_image_8 (在參數設定 connectivity=4，會使用 4-connected，在參數設定 connectivity=8，會使用 8-connected)。

接著利用 colorize_distance_transform 將 distance_image 的 label 根據 normalize 的結果填上顏色，把 distance_image_4 轉成 colors_distance_image_4 還有 distance_image_8 轉成 colors_distance_image_8。

再來使用 medial_axis 找到 skeleton_image。

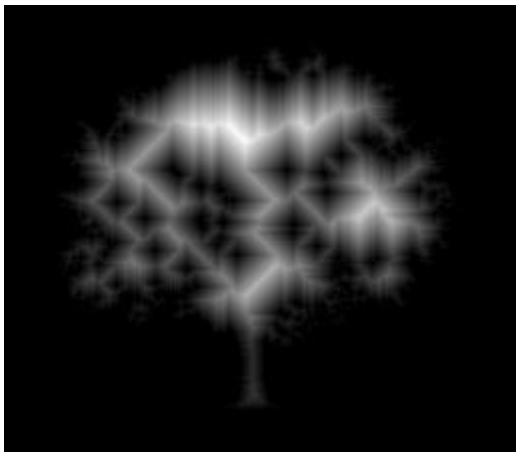
利用 cv2 的 imshow 查看 original image、gray image、binary image、colors_distance_image_4、colors_distance_image_8 和 skeleton_image。
利用 cv2 的 imwrite 將 colors_distance_image_4、colors_distance_image_8 和 skeleton_image 的結果存到 results 裡面。
利用 cv2 的 waitKey(0)和 destroyAllWindows()將 imshow 用顯示出來的圖片關閉。

```
if __name__ == '__main__':
    image(number=1, threshold=200)
    image(number=2, threshold=200)
    image(number=3, threshold=200)
    image(number=4, threshold=200)
```

執行 image，產生所有助教給的圖片的 4-connected 和 8-connected 的 distance transform 後上不同層次的顏色的圖片，還有 Medial Axis 的圖片。

Result images

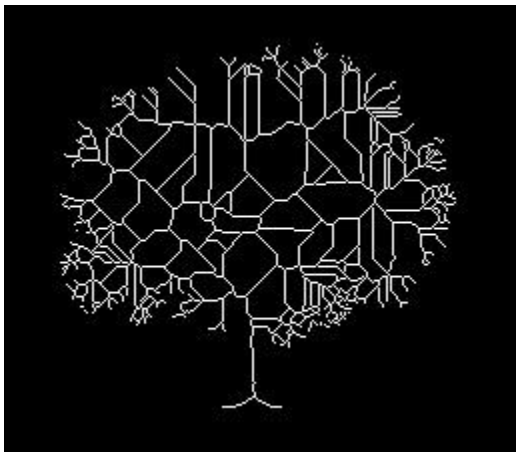
img1_q1-1_4.jpg



img1_q1-1_8.jpg

我預期的結果跟最後的輸出相同，從 4-connect 和 8-connect 的結果可以明顯看出來 distance transform 的結果，4-connect 散出來的痕跡是偏向十字，8-connect 散出來的痕跡偏向斜線。

img1_q1-2.jpg



我預期的結果跟最後的輸出相同，可以從 distance transform 的結果看到一點

medial axis 的結果的雛型。

img2_q1-1_4.jpg

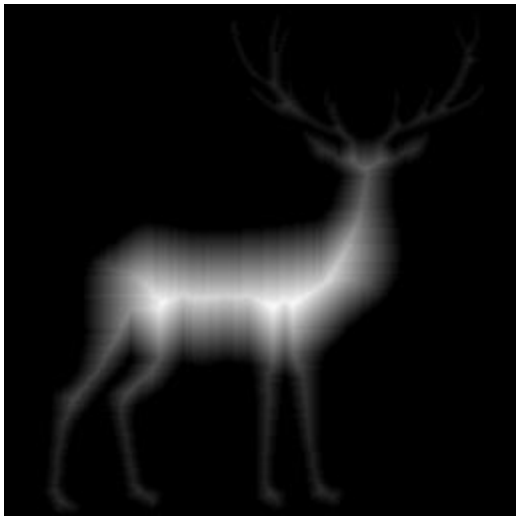

img2_q1-1_8.jpg


我預期的結果跟最後的輸出相同，8-connect 的 Machine Vision 的 V 可以明顯感受到鋸齒狀。

img2_q1-2.jpg


我預期的結果跟最後的輸出相同，可以從 distance transform 的結果看到一點 medial axis 的結果的雛型。

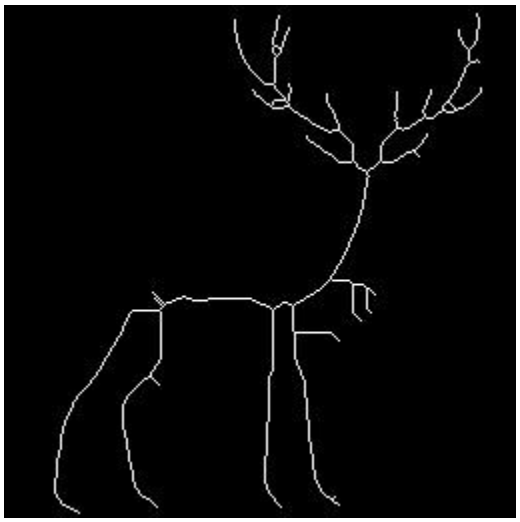
img3_q1-1_4.jpg



img3_q1-1_8.jpg

我預期的結果跟最後的輸出相同，4-connect 的鹿的身體散出來的線偏向十字，8-connect 散出來的線偏向斜線。

img3_q1-2.jpg



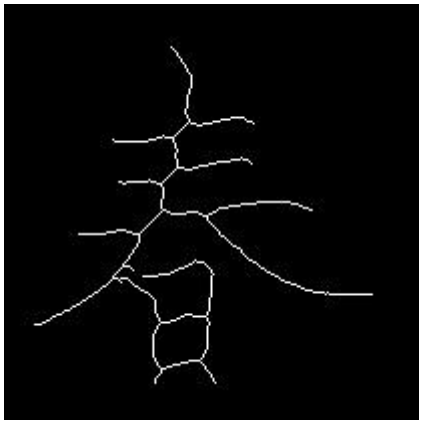
我預期的結果跟最後的輸出相同，從 distance transform 的結果中可以看到一點 medial axis 的結果的雛型。

img4_q1-1_4.jpg

img4_q1-1_8.jpg

我預期的結果跟最後的輸出相同，4-connect 的春上面的散出來的線偏向直線，8-connect 的則是斜線。

img4_q1-2.jpg



我預期的結果跟最後的輸出相同，從 distance transform 的結果中可以看到一點 medial axis 的結果的雛型。

最後的結果總結：
在一開始寫 hw3 的難點主要是找 medial axis 的部分，再找尋 medial axis 的時候有發現有些 thinning 的方式可以不用用尋找 local maximum 的方式就可以找到 skeleton，像是 Skeletonization-by-Zhang-Suen-Thinning-Algorithm，但是它的結果跟用尋找 local maximum 的方式（作業提到的做法）相差很多，從樹的圖片可以看到 Skeletonization-by-Zhang-Suen-Thinning-Algorithm 的樹葉有多捲起來形成圓圈的部分，而作業提到的方式則是以散發出去的線為居多。

從 distance transform 的結果來看 4-connect 和 8-connect 的散發結果 4-connect 會以十字的線較多，8-connect 的則以斜線較多。再 8-connect 的 distance transform 結果來看，8-connect 會比較接近原圖。

Skeletonization-by-Zhang-Suen-Thinning-Algorithm 連結
https://github.com/linbojin/Skeletonization-by-Zhang-Suen-Thinning-Algorithm/blob/master/thinning.py