R07922106 曾俊為 CV_HW7 Report

一開始先將 len.bmp 讀進來之後，先做 binarize 的動作，用迴圈將 pixel value 大於 127 的部分設為 1，其他則為 0。

```
for i in range(512):
    for j in range(512):
        if image_original[i][j] >= 128:
            image_original[i][j] = 1
        else:
            image_original[i][j] = 0
```

再來以 8*8 的 kernel 取 topmost-left 的 pixel 對 binary image 做 downsampling。

```
downsample_image = np.zeros([64,64])
for i in range(64):
    for j in range(64):
        downsample_image[i][j] = binary_image[8*i][8*j]
```

接下來，Yokoi operator 的部分則和 HW6 一樣，先從剛剛 downsampling 過後的圖，針對值為 1 的 pixel 一個一個去做。每個 pixel 要先找到他 neighborhood，並將之排序好。

```
kernel = [[-1,-1], [-1,0], [-1,1], [0,-1], [0,0], [0,1], [1,-1], [1,0], [1,1]]
neighborpixels = np.zeros(9)
x, y = pos
for element in kernel:
    i, j = element
    if 0 <= x+i < 64 and 0 <= y+j < 64:
        neighborpixels[(i+1)+3*(j+1)] = image[x+i][y+j]
    else :
        neighborpixels[(i+1)+3*(j+1)] = 0
neighborpixels1 = [neighborpixels[4], neighborpixels[5], neighborpixels[1],
neighborpixels[3], neighborpixels[7], neighborpixels[8], neighborpixels[2],
neighborpixels[0], neighborpixels[6]]
```

之後，照公式求出 4 個分別經過 h_function 跑出來的結果之後，再將這些值帶入 f_function，最後得到 0~5 的值。

```
def function_h(b, c, d, e):
    if b == c and (b != d or b != e):
        return("q")
    if b == c and (b == d and b == e):
        return("r")
    if b != c:
        return("s")

def function_f(a1, a2, a3, a4):
    count_list = [a1, a2, a3, a4]
    if count_list.count("r") == 4:
        return(5)
    else :
        return(count_list.count("q"))
```

以上都是做 thinning 的前置作業，也就是 HW6 的部分。
再來則是做 pair relationship operator 的部分，針對剛剛做 Yokoi operator 得到的結果中 value 為 1 的部分去找他的 neighbor 是否也有 Yokoi connectivity number 為 1 的，將之標註之後，得到 marked_pixel。

```
if matrix_yokoi[i, j] == '1':
    flag = False
    for element in NEIGHBORS:
        if 0 <= i + element[0] < height and 0 <= j + element[1] < width:
            if matrix_yokoi[i + element[0], j + element[1]] == '1':
                flag = True
                break
    if flag:
        indices_p.append((i, j))
```

接下來對做過 binarize 和 downsampling 過後的原圖做 connected shrink operator，針對剛剛標註的 marked pixel 的位置，做 remove 的動作，只要其 neighbor 超過圖的範圍或是其中一個 pixel 不相等，也將之標註。

```
if 0 <= i + n[0] < height and 0 <= j + n[1] < width:
    neighbor = image[i + n[0], j + n[1]]
    if neighbor:
        h1, w1 = i + d[0][0], j + d[0][1]
        h2, w2 = i + d[1][0], j + d[1][1]
        # 只要超出範圍或是其中一個pixel不相等，則標註起來
        if h1 < 0 or h1 == height or w1 < 0 or w1 == width or \
            h2 < 0 or h2 == height or w2 < 0 or w2 == width:
            count += 1
        elif image[h1, w1] != neighbor or image[h2, w2] != neighbor:
            count += 1
```

只要原圖 marked_pixel 相對應位置的值為 1 且做完 connected shrink operator 有標註，則將此位置的值 delete 掉，設為 0，也就是由白點變黑點，呈現 thinning 的效果。
最後反覆動作，直到結果圖不再改變為止，以下為最後結果 ：