**Boundary Extraction & Region Filling**

4110056030 資工三 鄭詠謙

* **Code**

import numpy as np

import cv2

def dilate(img\_matrix=None, kernel=np.ones((3, 3))):

    img\_matrix = np.asarray(img\_matrix)

    shape = np.asarray(kernel).shape

    dilated\_img = np.zeros((img\_matrix.shape[0], img\_matrix.shape[1]), dtype=img\_matrix.dtype)

    origin = (int((kernel.shape[0] - 1) / 2), int((kernel.shape[1] - 1) / 2))

    for i in range(len(img\_matrix)):

        for j in range(len(img\_matrix[0])):

            overlap = img\_matrix[check(i - origin[0]):i + (shape[0] - origin[0]),check(j - origin[1]):j + (shape[1] - origin[1])]

            shp = overlap.shape

            first\_row = int(np.fabs(i - origin[0])) if i - origin[0] < 0 else 0

            first\_col = int(np.fabs(j - origin[1])) if j - origin[1] < 0 else 0

            last\_row = shape[0] - 1 - (i + (shape[0] - origin[0]) - img\_matrix.shape[0]) if i + (shape[0] - origin[0]) > img\_matrix.shape[0] else shape[0] - 1

            last\_col = shape[1] - 1 - (j + (shape[1] - origin[1]) - img\_matrix.shape[1]) if j + (shape[1] - origin[1]) > img\_matrix.shape[1] else shape[1] - 1

            if shp[0] != 0 and shp[1] != 0 and np.logical\_and(kernel[first\_row:last\_row + 1, first\_col:last\_col + 1],overlap).any():

                dilated\_img[i, j] = 255

    return dilated\_img

def erode(img\_matrix=None, kernel=np.ones((3, 3))):

    img\_matrix = np.asarray(img\_matrix)

    shape = np.asarray(kernel).shape

    eroded\_img = np.zeros((img\_matrix.shape[0], img\_matrix.shape[1]), dtype=img\_matrix.dtype)

    origin = (int(np.ceil((kernel.shape[0] - 1) / 2.0)), int(np.ceil((kernel.shape[1] - 1) / 2.0)))

    for i in range(len(img\_matrix)):

        for j in range(len(img\_matrix[0])):

            overlap = img\_matrix[check(i - origin[0]):i + (shape[0] - origin[0]),check(j - origin[1]):j + (shape[1] - origin[1])]

            shp = overlap.shape

            first\_row = int(np.fabs(i - origin[0])) if i - origin[0] < 0 else 0

            first\_col = int(np.fabs(j - origin[1])) if j - origin[1] < 0 else 0

            last\_row = shape[0] - 1 - (i + (shape[0] - origin[0]) - img\_matrix.shape[0]) if i + (shape[0] - origin[0]) > img\_matrix.shape[0] else shape[0] - 1

            last\_col = shape[1] - 1 - (j + (shape[1] - origin[1]) - img\_matrix.shape[1]) if j + (shape[1] - origin[1]) > img\_matrix.shape[1] else shape[1] - 1

            if shp[0] != 0 and shp[1] != 0 and np.array\_equal(np.logical\_and(overlap, kernel[first\_row:last\_row + 1,first\_col:last\_col + 1]),kernel[first\_row:last\_row + 1,first\_col:last\_col + 1]):

                eroded\_img[i, j] = 255

    return eroded\_img

def check(index):

    return 0 if index < 0 else index

def boundary\_extraction():

    img\_path = "pika.jpg"

    input\_img = cv2.imread(img\_path)

    # 調整圖像大小

    height, width = input\_img.shape[:2]

    new\_dimensions = (width // 2, height // 2)

    input\_img = cv2.resize(input\_img, new\_dimensions, interpolation=cv2.INTER\_AREA)

    cv2.imshow("input", input\_img)

    gray\_img = cv2.cvtColor(input\_img, cv2.COLOR\_BGR2GRAY)

    # 二值化

    thresh = 128

    img\_binary = cv2.threshold(gray\_img, thresh, 255, cv2.THRESH\_BINARY)[1]

    origin\_el = np.ones((3, 3))

    # Erosion

    erosion = erode(img\_binary, origin\_el)

    # Dilation

    dilation = dilate(img\_binary, origin\_el)

    boundary = dilation - erosion

    # 二值圖畫素取反

    result = 255 - boundary

    cv2.imshow("erode", erosion)

    cv2.imshow("dilate", dilation)

    cv2.imshow("boundary", result)

    # 結束程式

    cv2.waitKey(0)

    cv2.destroyAllWindows()

def region\_filling():

    img\_path = "images.jpg"

    input\_img = cv2.imread(img\_path)

    # 調整圖像大小

    height, width = input\_img.shape[:2]

    new\_dimensions = (width // 3, height // 3)

    input\_img = cv2.resize(input\_img, new\_dimensions, interpolation=cv2.INTER\_AREA)

    cv2.imshow("input", input\_img)

    gray\_img = cv2.cvtColor(input\_img, cv2.COLOR\_BGR2GRAY)

    # 二值化

    thresh = 128

    img\_binary = cv2.threshold(gray\_img, thresh, 255, cv2.THRESH\_BINARY)[1]

    # 構造Marker圖像

    marker = np.zeros\_like(img\_binary)

    marker[0, :] = 255

    marker[-1, :] = 255

    marker[:, 0] = 255

    marker[:, -1] = 255

    mask = 255 - img\_binary

    cv2.imshow("mask", mask)

    origin\_el = np.ones((3, 3))

    while True:

        marker\_pre = marker.copy()

        dilation = dilate(marker, origin\_el)

        marker = np.minimum(dilation, mask)

        if np.array\_equal(marker\_pre, marker):

            break

    result = 255 - marker

    cv2.imshow("result", result)

    filling = result - img\_binary

    cv2.imshow("region filling", filling)

    # 結束程式

    cv2.waitKey(0)

    cv2.destroyAllWindows()

if \_\_name\_\_ == '\_\_main\_\_':

    boundary\_extraction()

    region\_filling()

* **Result**

圖片***Boundary Extraction***

邊界提取的應用公式:

    erosion = erode(img\_binary, origin\_el)

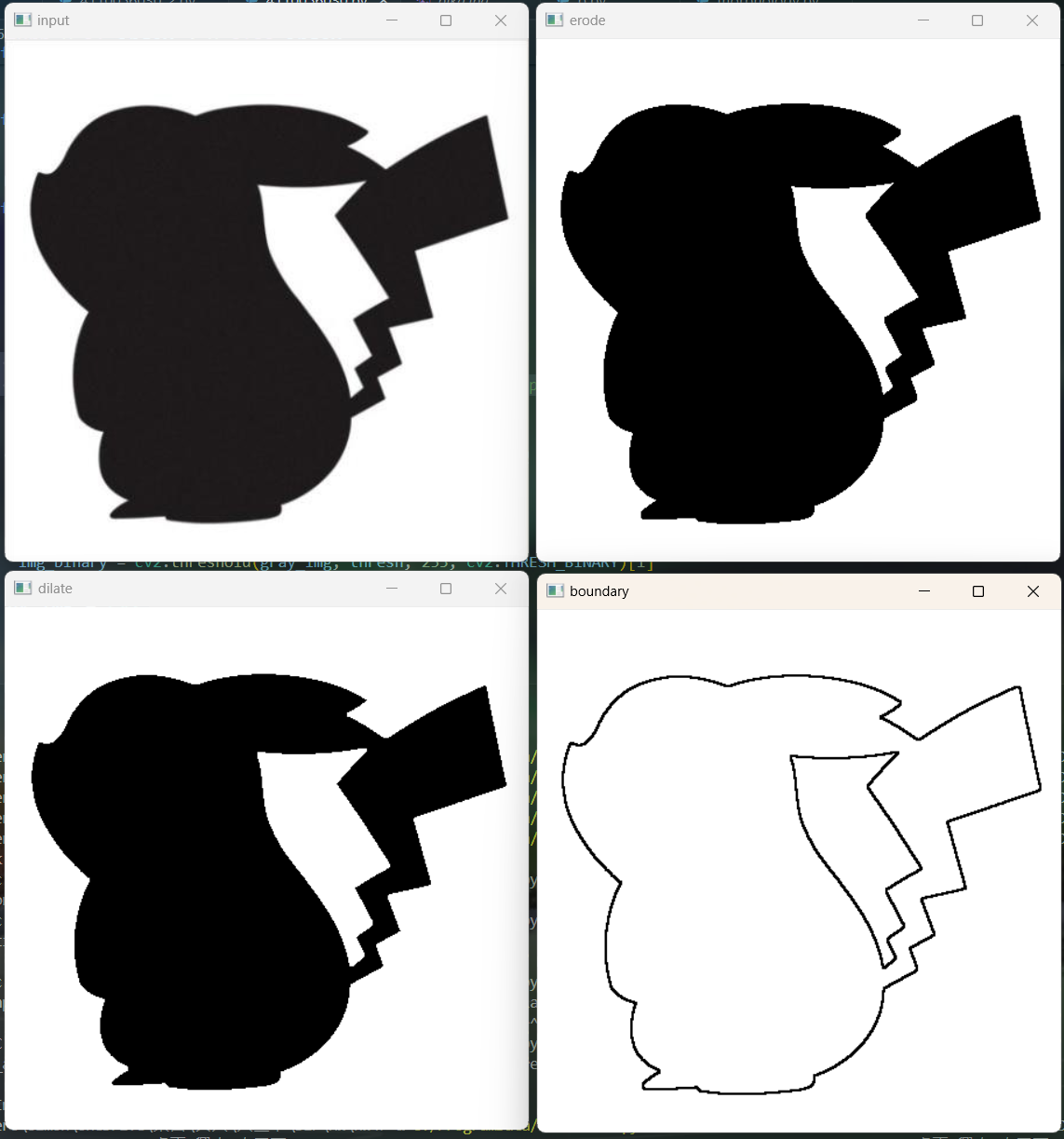
    dilation = dilate(img\_binary, origin\_el)

    boundary = dilation - erosion

利用膨脹後的圖減掉侵蝕後的圖可以得到一條輪廓，也可以用原圖-侵蝕圖或是用膨脹圖-原圖，只是邊界會稍微細一點。

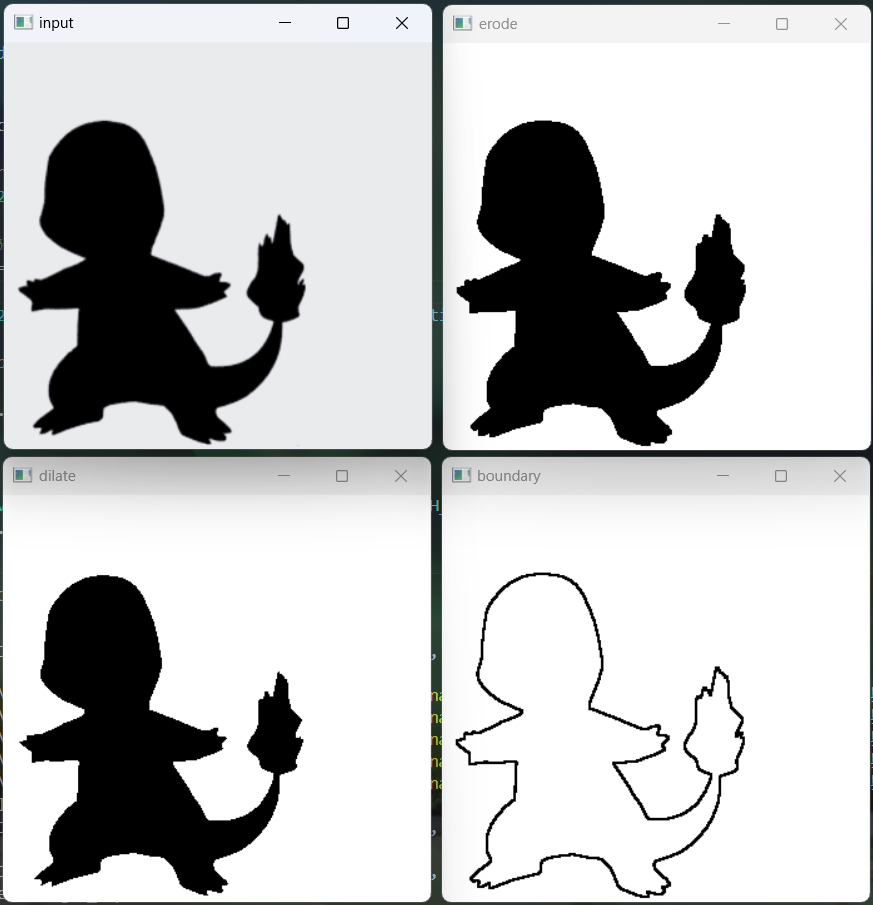
左上為input原圖，右上為侵蝕圖，左下為膨脹圖，右下為邊界提取結果。

Result1:



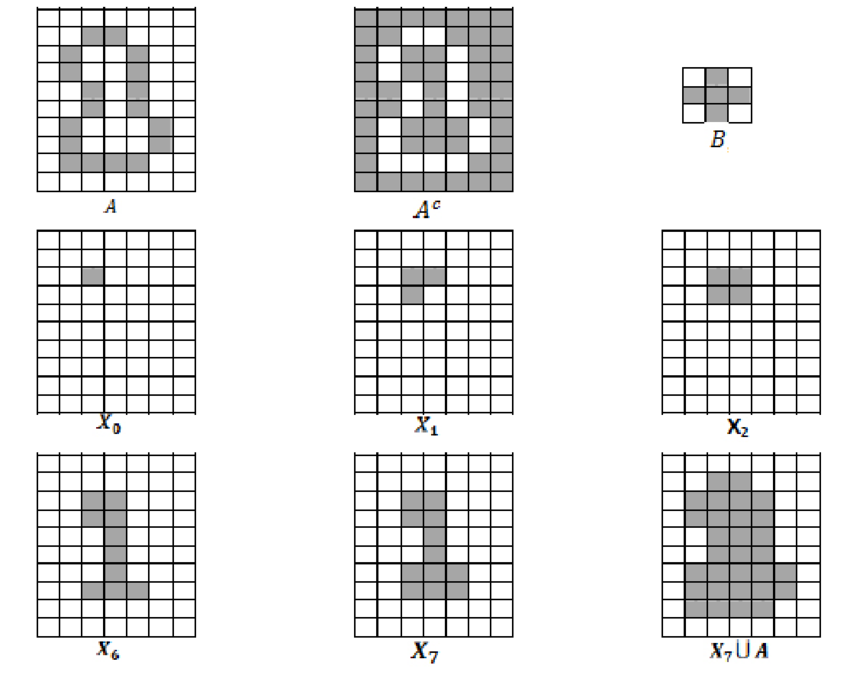
以下是用不同的圖片提取邊界，可以看到最後的結果在右下。

Result2:



***Region Filling***

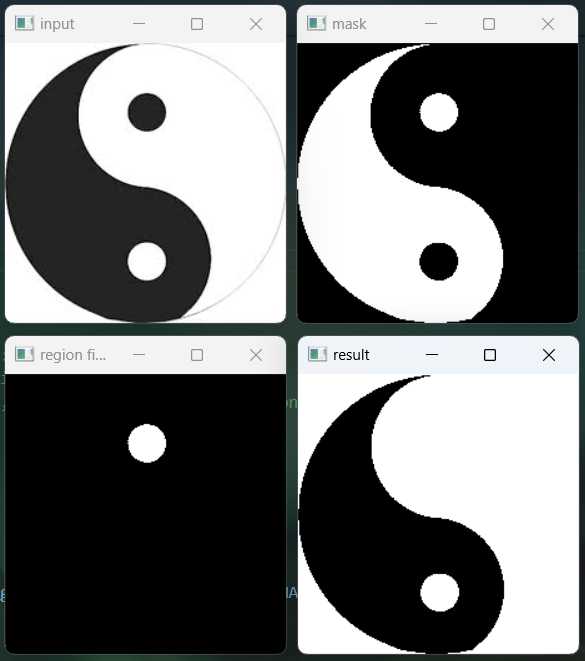
利用B圖這種Structures Element逐步移動，直到撞到A補集的邊界就往下一行填滿，直到範圍內都跑完。把X7跟A作聯集就是填滿後的樣子。



圖源:https://www.researchgate.net/figure/Figure-33-Representation-of-the-region-filling-algorithm\_fig7\_297585117

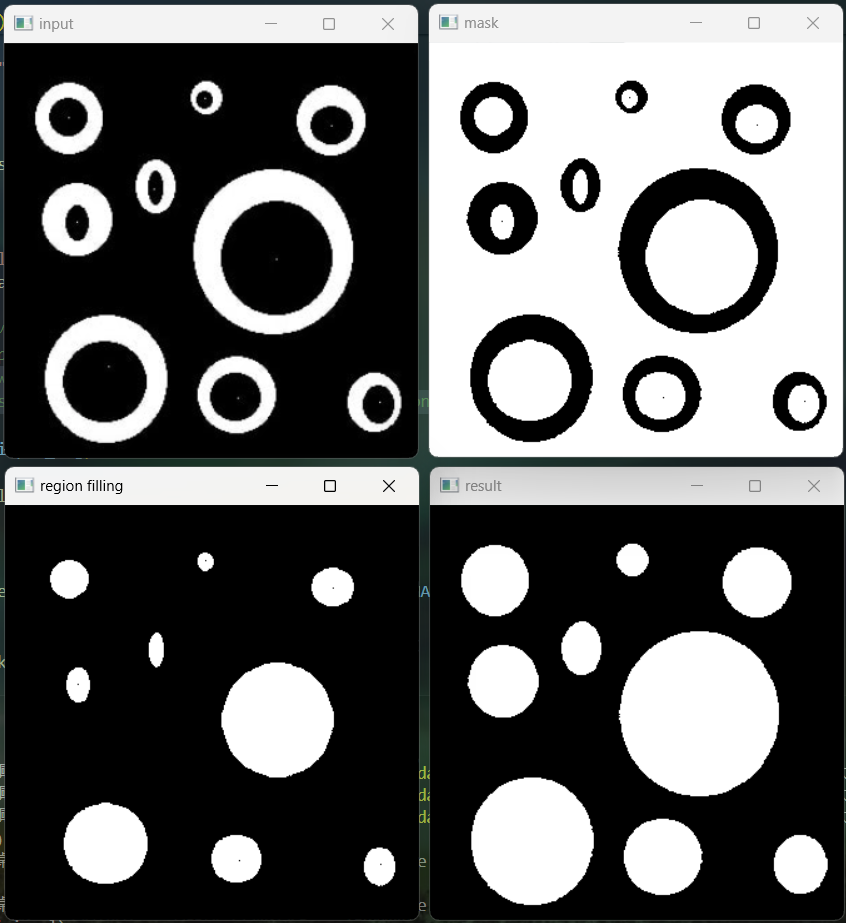
Result1:

左上為input原圖，右上為他的補集，左下為類似上圖X7欲填滿的區域，右下是填滿後的結果。

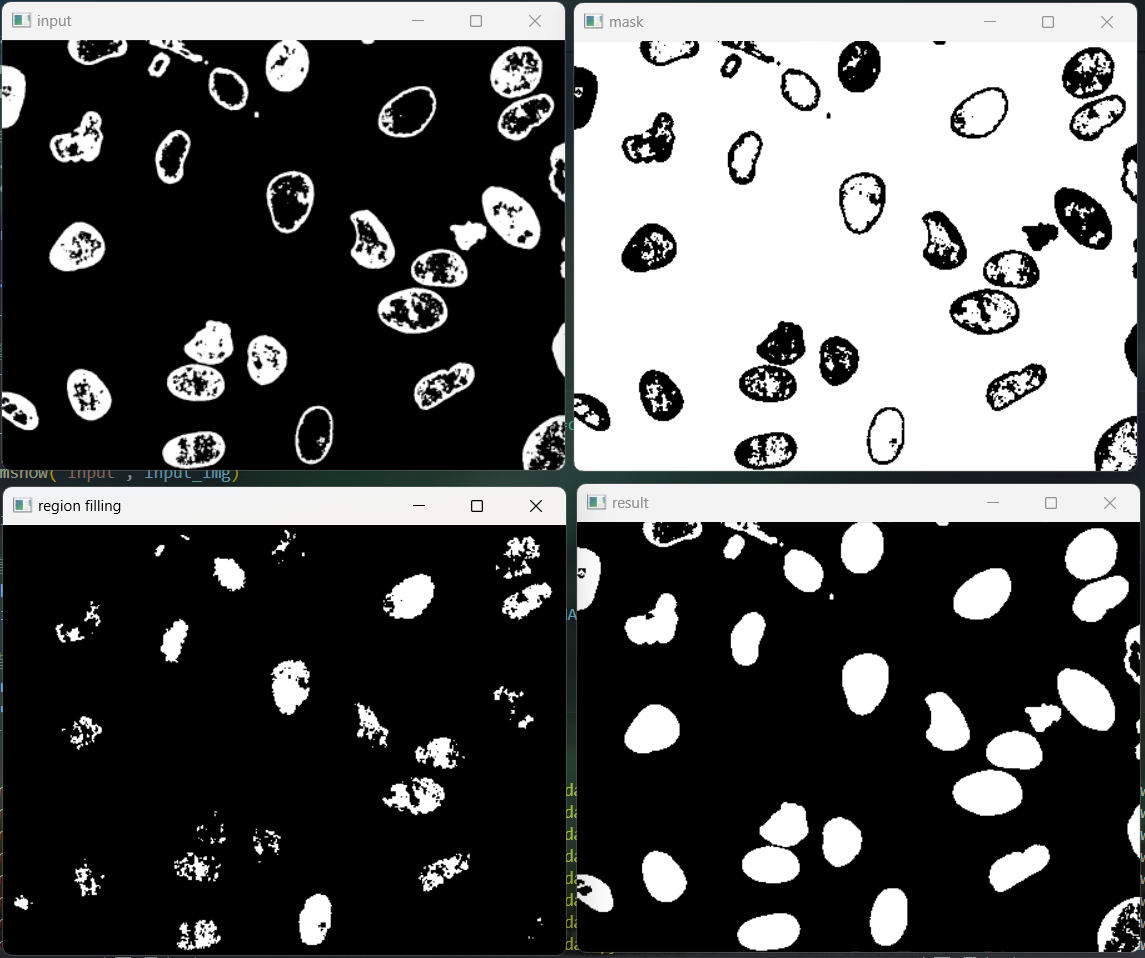


下圖為課本中給的圖片，可以看到右下角是把圈圈填滿的結果。

Result2:

****

Result3: 填補更加細節的區域。

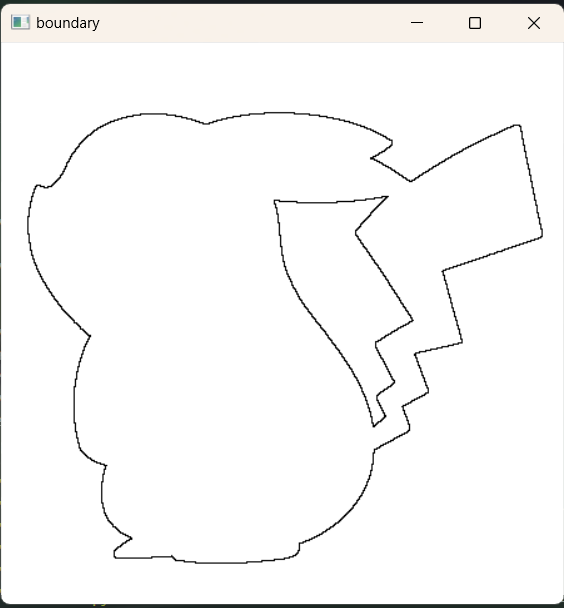


* **Discussion**

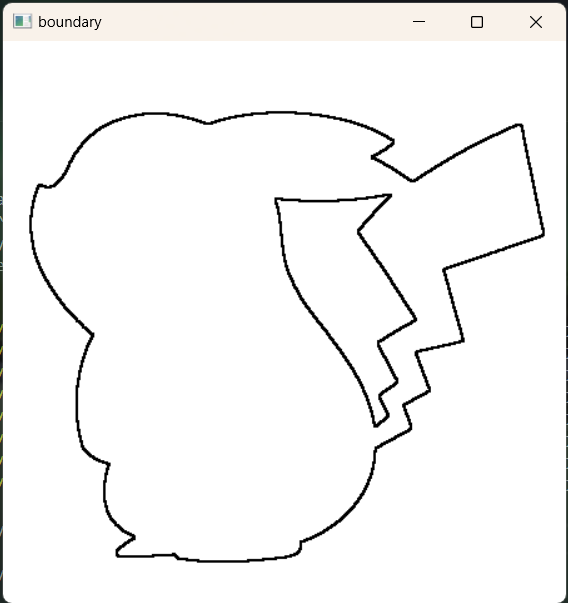
***Boundary Extraction***

調整線條粗細，改用原圖減侵蝕或是用膨脹減原圖。

使用原圖減去侵蝕的結果:



用膨脹減去侵蝕的結果:

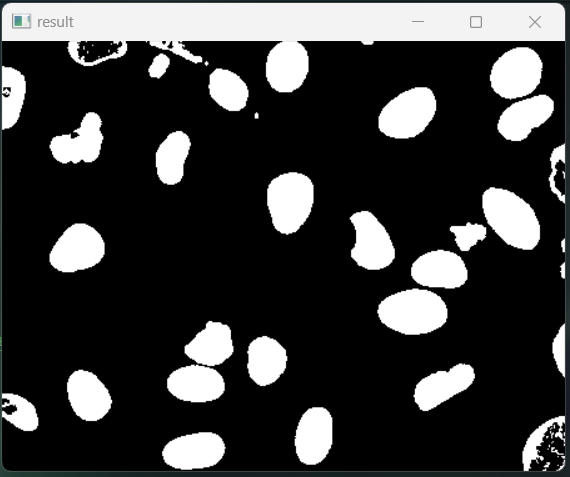


粗細會有明顯的不同。

***Region Filling***

用不同大小的element去填滿區域會有不同的效果

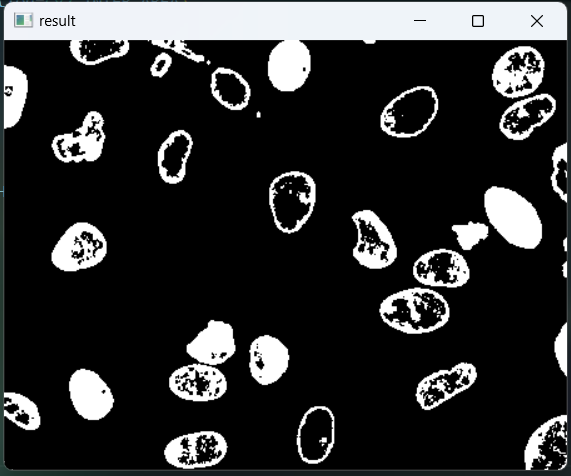
np.ones((3, 3))

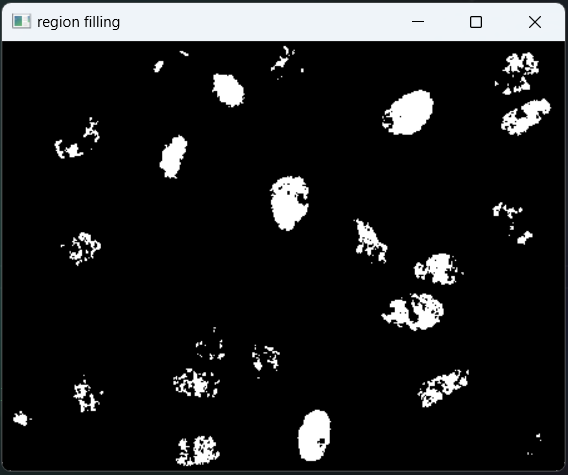


np.ones((5, 5))



np.ones((7, 7))

  
可以由上三種不同大小的結構elements看出3x3的會是最細節的，而7x7的填滿結果有很多地方都沒有填補到。以下是3x3跟7x7兩種element的filling。

 3x3

 7x7

總結:

Boundary Extraction主要是利用大圖減去小圖，邊緣偵測其實有許多方法例如sobel、canny等等，但是都不如使用Morphology 方法計算來的簡單。

Region Filling也是有用到dilation 跟 mask做比較，以達到filling的效果。