```
#octal kernel 3,5,5,5,3
kernel = [[-2,-1],[-2,0],[-2,1],
          [-1,-2],[-1,-1],[-1,0],[-1,1],[-1,2],
          [0,-2],[0,-1],[0,0] ,[0,1], [0,2],
          [1,-2],[1,-1],[1,0],[1,1],[1,2],
          [2,-1],[2,0],[2,1]]
# L shaped kernel
J_kernel = [[0,-1],[0,0],[1,0]]
K_kernel = [[-1,0],[-1,1],[0,1]]
```

```
def dilation(img, kernel):
    dilation = np.zeros(img.shape)
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            if img[i][j][0] != 0:
                for element in kernel:
                    p,q = element
                    if(i+p >= 0)and(i+p <=(img.shape[0]-1))and(j+q >=
0)and(j+q)<=(img.shape[1]-1):
                        dilation[i+p][j+q] = [255, 255, 255]
    return dilation

cv2.imwrite("dilation.bmp",dilation(img,kernel))
```

- 將　　後的　　以迴圈的方式讀入，如果該　　的值非零，就依序讀入　　中的座標，計算以該位置為中心　　覆蓋的範圍中「個別位置」是否可以擴張（在圖片範圍內、不超過邊界），如果可以便將計算的覆蓋位置之值設定為　　，跑過整張　　進行擴張。

```
def erosion(img, kernel):
    erosion = np.zeros(img.shape)
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            exist = True
            for element in kernel:
                p,q = element
                if(i+p < 0)or(i+p >= img.shape[0])or(j+q < 0)or(j+q >=
img.shape[1])or(img[i+p][j+q][0]==0):
                    exist = False
                    break
            if exist:
                erosion[i][j] = [255,255,255]
    return erosion

cv2.imwrite("erosion.bmp",erosion(img, kernel))
```

- 的作法和 很類似，只是在判斷 覆蓋位置是否要侵蝕的條件更改為：以該點為中心、以 覆蓋的範圍必須「所有位置」都不超過邊界且都在圖片範圍內，才將此位置之值設為 ，否則皆設為 。

```
def opening(img, kernel):
    return dilation(erosion(img, kernel), kernel)

cv2.imwrite("opening.bmp", opening(img, kernel))
```

- 根據 的定義，是先對圖片進行 ，再對圖片進行 。

  使用在 已完成的 和 函數進行組合完成 的函數操作。

```
def closing(img, kernel):
    return erosion(dilation(img, kernel), kernel)

cv2.imwrite("closing.bmp", closing(img, kernel))
```

- 和　　剛好相反，　　是先對圖片進行　　，再對圖片進行　　。
  同樣使用在　　已完成的　　和　　函數進行組合完成　　的函數操作。

```
# L shaped kernel
J_kernel = [[0,-1],[0,0],[1,0]]
K_kernel = [[-1,0],[-1,1],[0,1]]

def hitandmiss(img, J_kernel, K_kernel):
    img_o = erosion(img, J_kernel)
    img_c = erosion(255-img, K_kernel)
    ham = np.zeros(img.shape)
    for i in range(img.shape[0]):
```

```
        for j in range(img.shape[1]):
            if img_o[i][j][0] == 255 and img_c[i][j][0] == 255:
                ham[i][j] = [255, 255, 255]
    return ham
cv2.imwrite("hitandmiss.bmp", hitandmiss(img, J_kernel, K_kernel))
```

- 使用 形的 和 ，分別對 的 以及 的 的補集做 ，分別儲存在 和

  根據 的定義，最後產生的結果應為 和 的交集，將 和 皆為白色的地方設為 ，其餘為