Computer Vision HW4

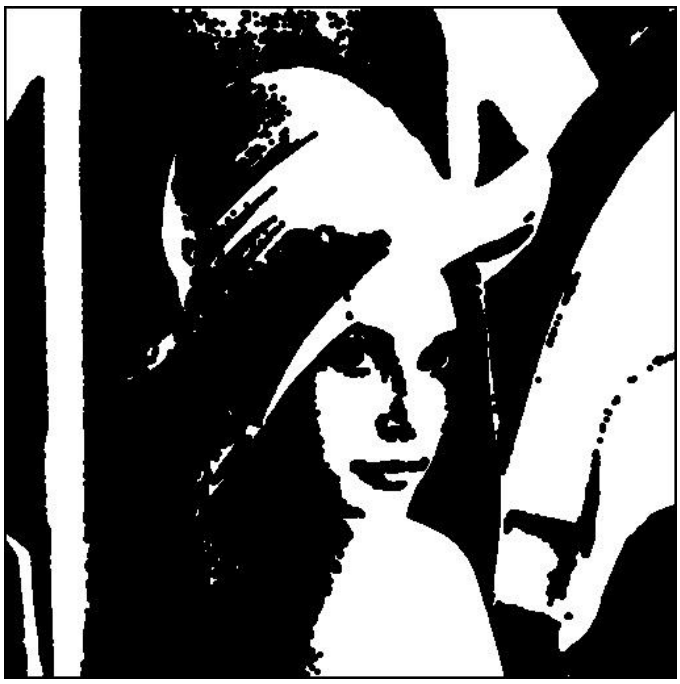
R08922156 黃劍韜

(a) Dilation

```
# (a) Dilation image 往外膨脹
def dilation_img(image,kernel,width,height):
    img_result = image.copy()
    for i in range(width):
        for j in range(height):
            if(image[i,j]==255):
                for k in kernel:
                    new_i = i + k[0]
                    new_j = j + k[1]
                    if(new_i>=0 and new_i<=width-1 and new_j>=0 and new_j<=height-1):
                        img_result[new_i,new_j] = 255
    return img_result
```

第一題的部分是做膨脹，只要有符合 kernel 的部分亮度都變成 255。我創建了一個 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]]，然後創建兩層的 for 迴圈去讀取每一個點，如果這個點的亮度為 255,則將 kernel 的原點與之重合，並去比對其他的點。如果這些點沒有超出圖像範圍，則將這些點的亮度變成 255。

(b) Erosion



```python
# (b) Erosion image 往內侵蝕
def erosion_img(image,kernel,width,height):
    img_result = np.zeros(image.shape)
    for i in range(width):
        for j in range(height):
            correct_num = 0
            if(image[i,j]==255):
                for k in kernel:
                    new_i = i + k[0]
                    new_j = j + k[1]
                    if(new_i>=0 and new_i<=width-1 and new_j>=0 and new_j<=height-1):
                        if(image[new_i,new_j]==255):
                            correct_num = correct_num + 1
                        else:
                            break
                    else:
                        break
                if(correct_num == len(kernel)):
                    img_result[i,j] = 255
    return img_result
```

第二題的部分是做侵蝕，只要全部的點都符合 kernel 的部分亮度才會保留 255，否則變成 0。我創建了一個 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]] 。創建兩層的 for 迴圈去讀取每一個點，如果這個點的亮度為 255, 則將 kernel 的原點與之重合，並去比對其他的點。只有 kernel 中所有的點都符合的話，這個點的亮度才為 255，反之為 0。

(c) Opening

```
# (c) Opening image 先Erosion再Dilation
def opening_img(image,kernel,width,height):
    e_img = erosion_img(image,kernel,width,height)
    img_result = dilation_img(e_img,kernel,width,height)
    return img_result
```

第三題的部分是計算 Opening，做法就是先做 Erosion 侵蝕再做 Dilation 膨脹。其公式為$(A \ominus B) \oplus B$。

(d) Closing

```
# (d) Closing image 先Dilation再Erosion
def closing_img(image,kernel,width,height):
    d_img = dilation_img(image,kernel,width,height)
    img_result = erosion_img(d_img,kernel,width,height)
    return img_result
```

第四題的部分是計算 Closing，做法就是先做 Dilation 膨脹再做 Erosion 侵蝕。其公式為$(A \oplus B) \ominus B$。

(e) Hit-and-miss transform

```
kernel2 = [ [0, -1], [0, 0], [1, 0] ]

kernel3 = [ [-1, 0], [-1, 1], [0, 1] ]
```

```
# (e) Hit-and-miss transform
def hit_and_miss(image,Jkernel,Kkernel,width,height):
    img1 = erosion_img(image,Jkernel,width,height)
    img2 = erosion_img_v2(255-image,Kkernel,width,height)
    img_result = np.zeros(image.shape)
    for i in range(width):
        for j in range(height):
            if(img1[i,j]==255 and img2[i,j]==255):
                img_result[i,j] = 255
    #show_img(img_result)
    return(img_result)
```

第四題的部分是計算 Hit and miss，創建兩個 L 型 kernel [[0, -1], [0, 0], [1, 0]]和[[-1, 0], [-1, 1], [0, 1]]。首先將原本二值化圖像用第一個 L 型 kernel 做 Erosion 侵蝕。第二步是將原圖像做補集，然後和第二個 L 型 kernel 做 Erosion 侵蝕。然後將這兩個得到的結果做一個交集的動作，即取兩個圖形中共同為 255 的點。