Computer Vision HW5

R08922156 黃劍韜

(a) Dilation

```python
# Get the maxinum of dilation
def maxnum_dilation(image,kernel,i,j,width,height):
    max_num = 0
    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):
            max_num = max(max_num,image[new_i,new_j])
    return max_num

# (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]!=0):
                max_value = maxnum_dilation(image,kernel,i,j,width,height)
                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] = max_value
    return img_result
```

第一題的部分是做灰階圖像的 Dilation 膨脹，相較於二值化圖像的 Dilation，灰階圖像的作法是一個雙層迴圈走訪圖像的每一個像素點，並把一個 kernel 的原點與該點重合，然後去找出其 local maximum 的亮度值。接著，去走訪其 kernel 中的每一個點，然後如果這些點在圖像上面，則將這些點的亮度值賦予 local maximum 的亮度值。

(b) Erosion

```python
# Get the mininum of erosion
def mininum_erosion(image,kernel,i,j,width,height):
    correct_num = 0
    min_num = 999999
    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]!=0):
                correct_num = correct_num + 1
                min_num = min(min_num,image[new_i,new_j])
            else:
                break
        else:
            break
    return min_num

# (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):
            if(image[i,j]!=0):
                min_value = mininum_erosion(image,kernel,i,j,width,height)
                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]!=0):
                            img_result[new_i,new_j] = min_value
                        else:
                            break
                    else:
                        break
    return img_result
```

第二題的部分是做灰階圖像的 Erosion 侵蝕，它的概念也是只要全部的點都符合 kernel 的部分亮度才會變成 local minimum，否則變成 0。相較於二值化圖像的 Erosion，灰階圖像的作法是一個雙層迴圈走訪圖像的每一個像素點，並把一個 kernel 的原點與該點重合，然後去找出其 local minimum 的亮度值。接著，再去走訪其 kernel 中的每一個點，然後如果這些點在圖像上面且不為 0，則將這些點的亮度值賦予 local minimum 的亮度值。

(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
```

第三題的部分是做灰階圖像的 Opening，做法和二值化圖像做 Opening 的時候類似，是先做 Erosion 侵蝕再做 Dilation 膨脹。其公式為$(A \ominus B) \oplus B$。

(d) Closing

```python
# (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
```

第四題的部分是做灰階圖像的 Closing，做法和二值化圖像做 Closing 的時候類似都是先做 Dilation 膨脹 Erosion 侵蝕再做。其計算公式為 $(A \oplus B) \ominus B$。