

Homework 4

R13525009 羅筠笙

a. Dilation

Description: The dilation function enlarges white regions in a binary image `img` using a structuring element kernel. It pads `img` with zeros, then checks each pixel's 5x5 window. If any overlapping pixel equals 255, the output pixel is set to 255; otherwise, it remains 0. This operation expands the white areas according to the kernel shape.



```
1. def dilation(img, kernel):
2.     kernel_size = len(kernel)
3.     pad_size = kernel_size // 2
4.     padded_img = np.pad(img, pad_size, mode='constant', constant_val
        ues=0)
5.     dilation_img = np.zeros_like(img)
6.
7.     for i in range(img_size0):
8.         for j in range(img_size1):
9.             max_value = 0
10.            for ki in range(kernel_size):
11.                for kj in range(kernel_size):
12.                    if kernel[ki][kj] == 1:
13.                        pixel_value = padded_img[i + ki][j + kj]
14.                        if pixel_value > max_value:
15.                            max_value = pixel_value
16.                        dilation_img[i, j] = max_value
17.
18.     return dilation_img.astype(np.uint8)
```

b. Erosion

Description: The erosion function performs erosion on a binary image `img` using a specified structuring element kernel. It first pads `img` with a border of 255s to prevent boundary issues. Then, for each pixel, it checks if all corresponding pixels in the kernel's area are equal to the kernel values multiplied by 255. If this condition is met, the output pixel in `erosion_img` is set to 255; otherwise, it remains 0. This operation effectively shrinks the white regions in the image.



```
1. def erosion(img, kernel):
2.     kernel_size = len(kernel)
3.     pad_size = kernel_size // 2
4.     padded_img = np.pad(img, pad_size, mode='constant', constant_val
        ues=255)
5.     erosion_img = np.zeros_like(img)
6.
7.     for i in range(img_size0):
8.         for j in range(img_size1):
9.             min_value = 255
10.            for ki in range(kernel_size):
11.                for kj in range(kernel_size):
12.                    if kernel[ki][kj] == 1:
13.                        pixel_value = padded_img[i + ki][j + kj]
14.                        if pixel_value < min_value:
15.                            min_value = pixel_value
16.            erosion_img[i, j] = min_value
17.
18.     return erosion_img.astype(np.uint8)
```

c. Opening

Description: First use erosion, then use dilation.



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

d. Closing

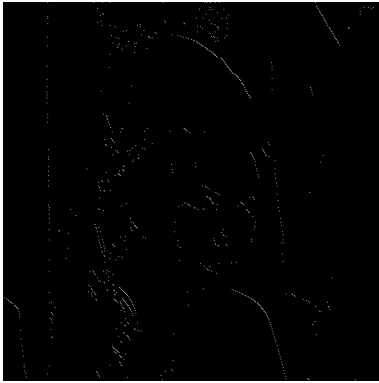
Description: First use dilation, then use erosion.



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

e. Hit-and-miss transform

Description: Identifies specific patterns in a binary image by eroding the image with kernel1 for hit detection, then eroding the inverted image with kernel2 for miss detection. It combines the results with a logical AND, marking locations that match the defined pattern.



```
1. def hit_and_miss(img, kernel1, kernel2):  
2.     hit = erosion(img, kernel1)  
3.     miss = erosion(-img+255, kernel2)  
4.     return (hit & miss).astype(np.uint8)
```