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.

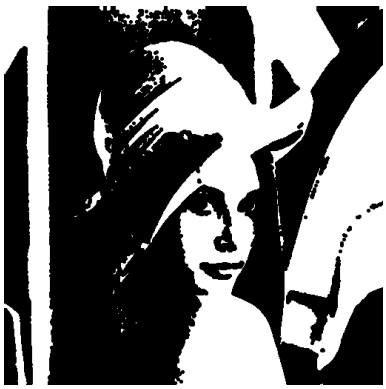
```
def dilation(img, kernel):
    kernel_size = len(kernel)
    dilation_img = np.zeros_like(img)

    for i in range(img_size0):
        for j in range(img_size1):
            max_value = 0
            for ki in range(kernel_size):
                for kj in range(kernel_size):
                    ni, nj = i + ki - kernel_size // 2, j + kj - kernel_siz
e // 2
                    if 0 <= ni < img_size0 and 0 <= nj < img_size1:
                        if kernel[ki][kj] == 1:
                            pixel_value = img[ni][nj]
                            if pixel_value > max_value:
                                max_value = pixel_value
            dilation_img[i, j] = max_value

    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.



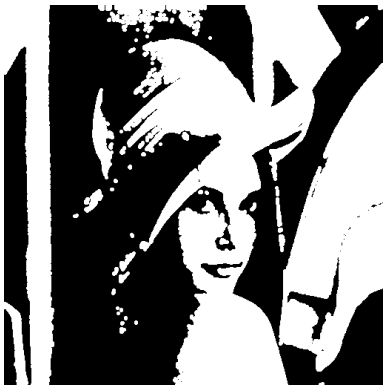
```
def erosion(img, kernel):
    kernel_size = len(kernel)
    erosion_img = np.zeros_like(img)

    for i in range(img_size0):
        for j in range(img_size1):
            min_value = 255
            for ki in range(kernel_size):
                for kj in range(kernel_size):
                    ni, nj = i + ki - kernel_size // 2, j + kj - kernel_size // 2
                    if 0 <= ni < img_size0 and 0 <= nj < img_size1:
                        if kernel[ki][kj] == 1:
                            pixel_value = img[ni][nj]
                            if pixel_value < min_value:
                                min_value = pixel_value
            erosion_img[i, j] = min_value

    return erosion_img.astype(np.uint8)
```

c. Opening

Description: First use erosion, then use dilation.



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

d. Closing

Description: First use dilation, then use erosion.

```
def closing(img, kernel):
    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.

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