Computer Vision HW05 旁聽生 陳又蕾

Brief Description: This homework used Python, OpenCV, numpy to uses gray-scale morphology to transform the Lena image to:

(a) dilation (b) erosion (c) opening (d) closing

Please use python ./main.py to run the program.

The kernel used for the first four homework problems is a 3-5-5-5-3 octagonal. The tuples indicate the directions/change in rows and columns of all white points neighboring the origin pixel:

```
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)]
```

Dilation

```
def dilation(img, kernel):
    res = np.zeros((img.shape[0], img.shape[1]), np.int)
    for row in range(img.shape[0]):
        for col in range(img.shape[1]):
            max_val = 0
            for (px, py) in kernel:
                new_row = row +  px
                new_col = col + py
                if(new_row >= 0 and new_row < img.shape[0] and new_col >= 0
                    and new_col < img.shape[1]):
                    if(img[new_row][new_col] > max_val):
                        max_val = img[new_row][new_col]
            res[row][col] = max_val
    return res
```

The algorithm used can be explained by the definition of grayscale dilation in Wikipedia. It involves assigning to each pixel, the **maximum value** found over the neighborhood of the structuring element.

I first created an array of black pixels with the size of the input image. I later loop through each pixel and generate the rows and columns of all the neighboring pixels using the given kernel. The value of its respective pixel in the array will be set to maximum value found over the neighboring pixels. The resulting array of pixels is at last returned from this function.

Erosion

```
def erosion(img, kernel):
    res = np.zeros((img.shape[0], img.shape[1]), np.int)
    for row in range(img.shape[0]):
        for col in range(img.shape[1]):
            min_val = 256
            for (px, py) in kernel:
                new_row = row +  px
                new_col = col + py
                if(new_row >= 0 and new_row < img.shape[0] and new_col >= 0
                        and new_col < img.shape[1]):
                    if(img[new_row][new_col] < min_val):
                        min_val = img[new_row][new_col]
            res[row][col] = min_val
    return res
```

The algorithm used can be explained by the definition of grayscale dilation in Wikipedia. It involves assigning to each pixel, the **minimum value** found over the neighborhood of the structuring element.

I first created an array of black pixels with the size of the input image. I later loop through each pixel and generate the rows and columns of all the neighboring pixels using the given kernel. The value of its respective pixel in the array will be set to minimum value found over the neighboring pixels. The resulting array of pixels is at last returned from this function.

Opening

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

The input image is first eroded then dilated using the grayscale dilation and erosion method I wrote previously

Closing

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

The input image is first dilated then eroded using the grayscale dilation and erosion I wrote previously.