922 U0610 電腦視覺
Computer Vision

Homework 5

授課教師： 傅楸善 教授

學生系級： 資工所一年級

學生姓名： 姚嘉昇

學生學號： R06922002

I. INTRODUCTION

1.1. Descriptions of Problem

This homework is to do gray scaled morphology with following rules:

A. Please use the octagonal 3-5-5-5-3 kernel.
B. Please take the local maxima or local minima respectively.
C. Four images should be included in your report: Dilation, Erosion, Opening, and Closing.

1.2. Programming Tools

1.2.1. Programming Language: Python3

1.2.2. Programming IDE: Visual Studio Code

II. METHOD

2.1. Algorithms

2.1.1. Grayscaled Dilation

$$(f \oplus b)(x) = \sup_{y \in E}[f(y) + b(x - y)]$$

2.1.2. Grayscaled Erosion

$$(f \ominus b)(x) = \inf_{y \in B}[f(x + y) - b(y)]$$

2.1.3. Opening

$$A \circ B = (A \ominus B) \oplus B$$

2.1.4. Closing

$$A \bullet B = (A \oplus B) \ominus B$$

2.2. Code Fragments

2.2.1. Code fragments of dilation

```python
if __name__ == '__main__':
    from PIL import Image
    import numpy as np
    import JasonDIP

    # Define kernel for dilation.
    kernel = np.array([\
        [0, 1, 1, 1, 0], \
        [1, 1, 1, 1, 1], \
        [1, 1, 1, 1, 1], \
        [1, 1, 1, 1, 1], \
        [0, 1, 1, 1, 0]])
    # Define center of kernel for dilation.
    centerKernel = (2, 2)
    # Load image from file.
    originalImage = Image.open('lena.bmp')
    # Get dilation image.
    dilationImage = JasonDIP.dilation(originalImage, kernel, centerKernel)
    # Save image fo file.
    dilationImage.save('dilation.bmp')
```

Figure 2.2.1.1. Code of main of dilation.

```python
def dilation(originalImage, kernel, centerKernel):
    """
    :type originalImage: Image (from PIL)
    :type kernel: numpy array
    :type centerKernel: tuple
    :return type: Image (from PIL)
    """
    from PIL import Image
    # New image with the same size and 'grayscale' format.
    dilationImage = Image.new('L', originalImage.size)
    # Scan each column in original image.
    for r in range(originalImage.size[0]):
        # Scan each row in original image.
        for c in range(originalImage.size[1]):
            # Record local max. pixel value.
            localMaxPixel = 0
            # Scan each column in kernel.
            for x in range(kernel.shape[0]):
                # Scan each row in kernel.
                for y in range(kernel.shape[1]):
                    # Only check value '1' in kernel.
                    if (kernel[x, y] == 1):
                        # Calculate destination x, y position.
                        destX = r + (x - centerKernel[0])
                        destY = c + (y - centerKernel[1])
                        # Avoid out of image range.
                        if ((0 <= destX < originalImage.size[0]) and \
                            (0 <= destY < originalImage.size[1])):
                            # Get pixel value in original image at (destX, destY).
                            originalPixel = originalImage.getpixel((destX, destY))
                            # Update local max. pixel value.
                            localMaxPixel = max(localMaxPixel, originalPixel)
            # Paste local max. pixel value on original image.
            dilationImage.putpixel((r, c), localMaxPixel)
    # Return dilation image.
    return dilationImage
```

Figure 2.2.1.2. Code of function of dilation.

2.2.2. Code fragments of erosion

```python
if __name__ == '__main__':
    from PIL import Image
    import numpy as np
    import JasonDIP

    # Define kernel for erosion.
    kernel = np.array([\
        [0, 1, 1, 1, 0], \
        [1, 1, 1, 1, 1], \
        [1, 1, 1, 1, 1], \
        [1, 1, 1, 1, 1], \
        [0, 1, 1, 1, 0]])
    # Define center of kernel for erosion.
    centerKernel = (2, 2)
    # Load image from file.
    originalImage = Image.open('lena.bmp')
    # Get erosion image.
    erosionImage = JasonDIP.erosion(originalImage, kernel, centerKernel)
    # Save image fo file.
    erosionImage.save('erosion.bmp')
```

Figure 2.2.2.1. Code of main of erosion.

```python
def erosion(originalImage, kernel, centerKernel):
    """
    :type originalImage: Image (from PIL)
    :type kernel: numpy array
    :type centerKernel: tuple
    :return type: Image (from PIL)
    """
    from PIL import Image
    # New image with the same size and 'grayscale' format.
    erosionImage = Image.new('L', originalImage.size)
    # Scan each column in original image.
    for r in range(originalImage.size[0]):
        # Scan each row in original image.
        for c in range(originalImage.size[1]):
            # Record local min. pixel value.
            localMinPixel = 255
            # Scan each column in kernel.
            for x in range(kernel.shape[0]):
                # Scan each row in kernel.
                for y in range(kernel.shape[1]):
                    # Only check value '1' in kernel.
                    if (kernel[x, y] == 1):
                        # Calculate destination x, y position.
                        destX = r + (x - centerKernel[0])
                        destY = c + (y - centerKernel[1])
                        # Avoid out of image range.
                        if ((0 <= destX < originalImage.size[0]) and \
                            (0 <= destY < originalImage.size[1])):
                            # Get pixel value in original image at (destX, destY).
                            originalPixel = originalImage.getpixel((destX, destY))
                            # Update local min. pixel value.
                            localMinPixel = min(localMinPixel, originalPixel)
            # Paste local min. pixel value on original image.
            erosionImage.putpixel((r, c), localMinPixel)
    # Return erosion image.
    return erosionImage
```

Figure 2.2.2.2. Code of function of erosion.

2.2.3. Code fragments of opening

```python
if __name__ == '__main__':
    from PIL import Image
    import numpy as np
    import JasonDIP

    # Define kernel for opening.
    kernel = np.array([\
        [0, 1, 1, 1, 0], \
        [1, 1, 1, 1, 1], \
        [1, 1, 1, 1, 1], \
        [1, 1, 1, 1, 1], \
        [0, 1, 1, 1, 0]])
    # Define center of kernel for opening.
    centerKernel = (2, 2)
    # Load image from file.
    originalImage = Image.open('lena.bmp')
    # Get opening image.
    openingImage = JasonDIP.opening(originalImage, kernel, centerKernel)
    # Save image fo file.
    openingImage.save('opening.bmp')
```

Figure 2.2.3.1. Code of main of opening.

```python
def opening(originalImage, kernel, centerKernel):
    """
    :type originalImage: Image (from PIL)
    :type kernel: numpy array
    :type centerKernel: tuple
    :return type: Image (from PIL)
    """
    return dilation(erosion(originalImage, kernel, centerKernel), kernel, centerKernel)
```

Figure 2.2.3.2. Code of function of opening.

2.2.4. Code fragments of closing

```python
if __name__ == '__main__':
    from PIL import Image
    import numpy as np
    import JasonDIP

    # Define kernel for closing.
    kernel = np.array([\
        [0, 1, 1, 1, 0], \
        [1, 1, 1, 1, 1], \
        [1, 1, 1, 1, 1], \
        [1, 1, 1, 1, 1], \
        [0, 1, 1, 1, 0]])
    # Define center of kernel for closing.
    centerKernel = (2, 2)
    # Load image from file.
    originalImage = Image.open('lena.bmp')
    # Get closing image.
    closingImage = JasonDIP.closing(originalImage, kernel, centerKernel)
    # Save image fo file.
    closingImage.save('closing.bmp')
```

Figure 2.2.4.1. Code of main of closing.

```python
def closing(originalImage, kernel, centerKernel):
    """
    :type originalImage: Image (from PIL)
    :type kernel: numpy array
    :type centerKernel: tuple
    :return type: Image (from PIL)
    """
    return erosion(dilation(originalImage, kernel, centerKernel), kernel, centerKernel)
```

Figure 2.2.4.2. Code of function of closing.

III. RESULTS

3.1. Original Image

Figure 3.1. Original lena.bmp.

3.2. Results of dilation and erosion

Figure 3.2.1. dilation.bmp.

Figure 3.2.2. erosion.bmp.

3.3. Results of opening and closing

Figure 3.3.1. Original lena.bmp.

Figure 3.3.2. opening.bmp.

Figure 3.3.3. closing.bmp.