922 U0610 電腦視覺
Computer Vision

Homework 4

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 binary morphology with following rules:

A. Please use the octagonal 3-5-5-5-3 kernel.

B. Please use the “L” shaped kernel to detect the upper-right corner for hit-and-miss transform.

C. Please process the white pixels (operating on white pixels).

D. Five images should be included in your report: Dilation, Erosion, Opening, Closing, and Hit-and-Miss.

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. Dilation

$$A \oplus B = \{c \in E^N \mid c = a + b \text{ for some } a \in A \text{ and } b \in B\}$$

2.1.2. Erosion

$$A \ominus B = \{x \in E^N | x + b \in A \text{ for every } b \in B\}$$

2.1.3. Opening

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

2.1.4. Closing

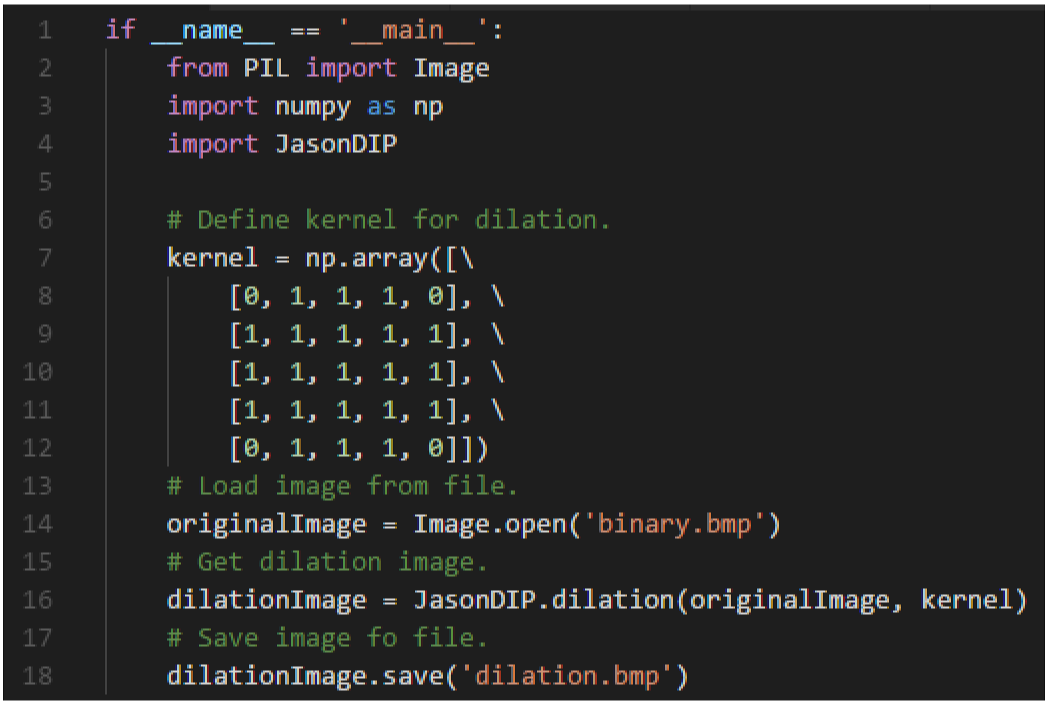
$$B \bullet K = (B \oplus K) \ominus K$$

2.1.5. Hit-and-Miss

$$A \otimes (J, K) = (A \ominus J) \cap (A^c \ominus K)$$

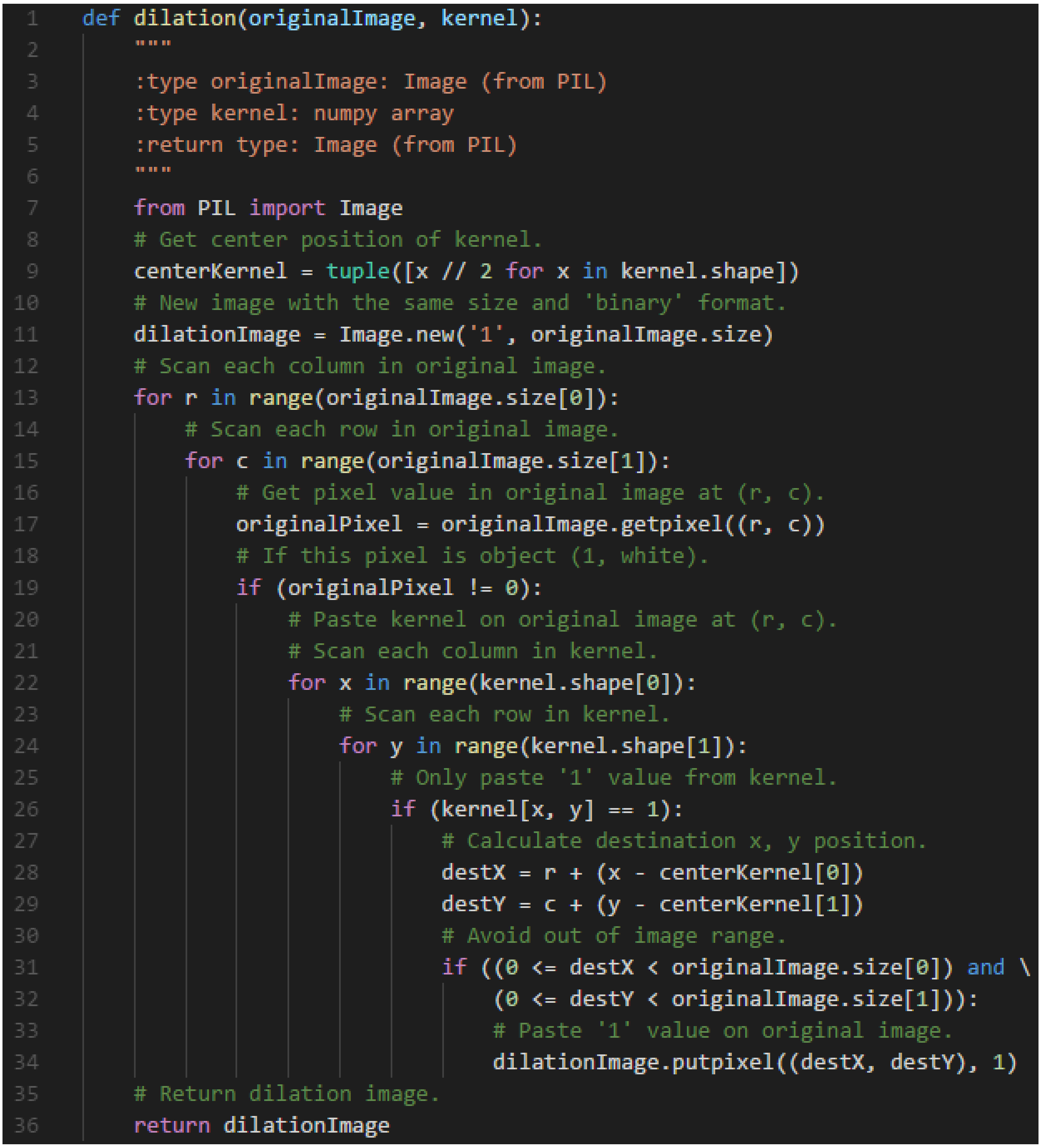
2.2. Code Fragments

2.2.1. Code fragments of dilation



```
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]])
    # Load image from file.
    originalImage = Image.open('binary.bmp')
    # Get dilation image.
    dilationImage = JasonDIP.dilation(originalImage, kernel)
    # Save image fo file.
    dilationImage.save('dilation.bmp')
```

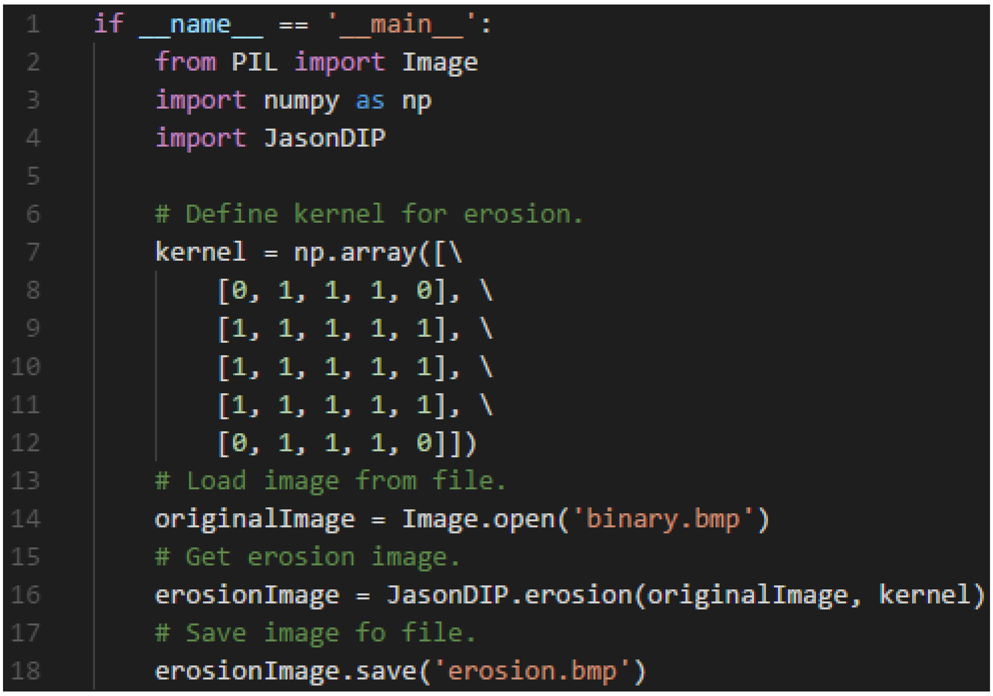
Figure 2.2.1.1. Code of main of dilation.



```python
def dilation(originalImage, kernel):
    """
    :type originalImage: Image (from PIL)
    :type kernel: numpy array
    :return type: Image (from PIL)
    """
    from PIL import Image
    # Get center position of kernel.
    centerKernel = tuple([x // 2 for x in kernel.shape])
    # New image with the same size and 'binary' format.
    dilationImage = Image.new('1', 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]):
            # Get pixel value in original image at (r, c).
            originalPixel = originalImage.getpixel((r, c))
            # If this pixel is object (1, white).
            if (originalPixel != 0):
                # Paste kernel on original image at (r, c).
                # 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 paste '1' value from 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])):
                                # Paste '1' value on original image.
                                dilationImage.putpixel((destX, destY), 1)
    # Return dilation image.
    return dilationImage
```

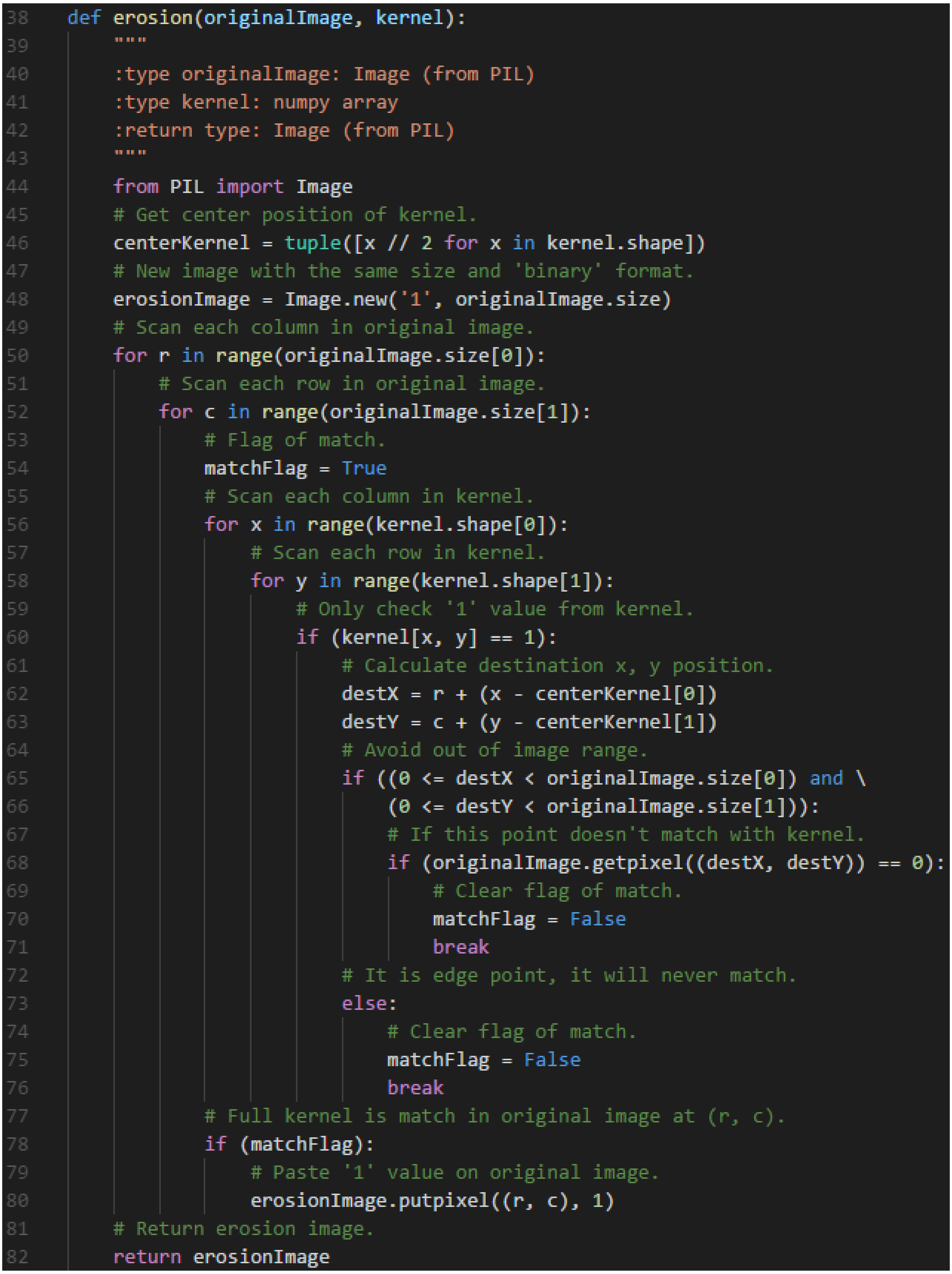
Figure 2.2.1.2. Code of function of dilation.

2.2.2. Code fragments of erosion



```
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]])
    # Load image from file.
    originalImage = Image.open('binary.bmp')
    # Get erosion image.
    erosionImage = JasonDIP.erosion(originalImage, kernel)
    # Save image fo file.
    erosionImage.save('erosion.bmp')
```

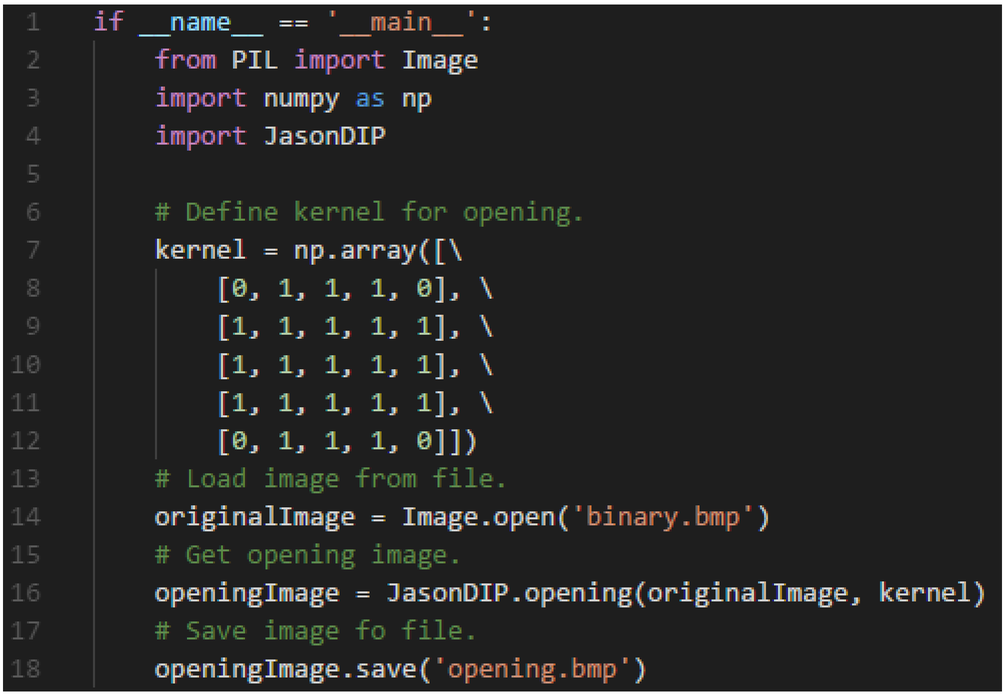
Figure 2.2.2.1. Code of main of erosion.



```
def erosion(originalImage, kernel):
    """
    :type originalImage: Image (from PIL)
    :type kernel: numpy array
    :return type: Image (from PIL)
    """
    from PIL import Image
    # Get center position of kernel.
    centerKernel = tuple([x // 2 for x in kernel.shape])
    # New image with the same size and 'binary' format.
    erosionImage = Image.new('1', 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]):
            # Flag of match.
            matchFlag = True
            # 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 '1' value from 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])):
                            # If this point doesn't match with kernel.
                            if (originalImage.getpixel((destX, destY)) == 0):
                                # Clear flag of match.
                                matchFlag = False
                                break
                        # It is edge point, it will never match.
                        else:
                            # Clear flag of match.
                            matchFlag = False
                            break
            # Full kernel is match in original image at (r, c).
            if (matchFlag):
                # Paste '1' value on original image.
                erosionImage.putpixel((r, c), 1)
    # Return erosion image.
    return erosionImage
```

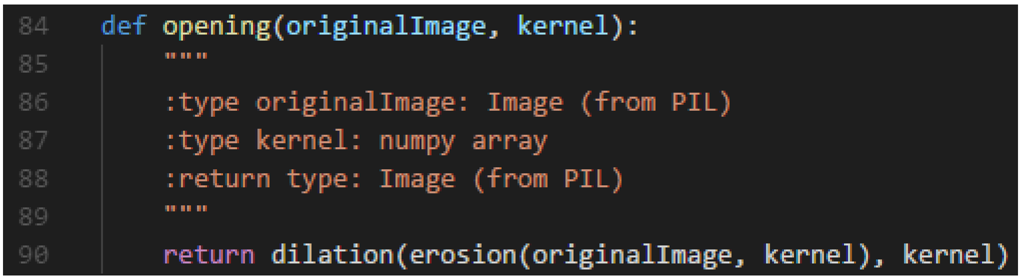
Figure 2.2.2.2. Code of function of erosion.

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]])
    # Load image from file.
    originalImage = Image.open('binary.bmp')
    # Get opening image.
    openingImage = JasonDIP.opening(originalImage, kernel)
    # Save image fo file.
    openingImage.save('opening.bmp')
```

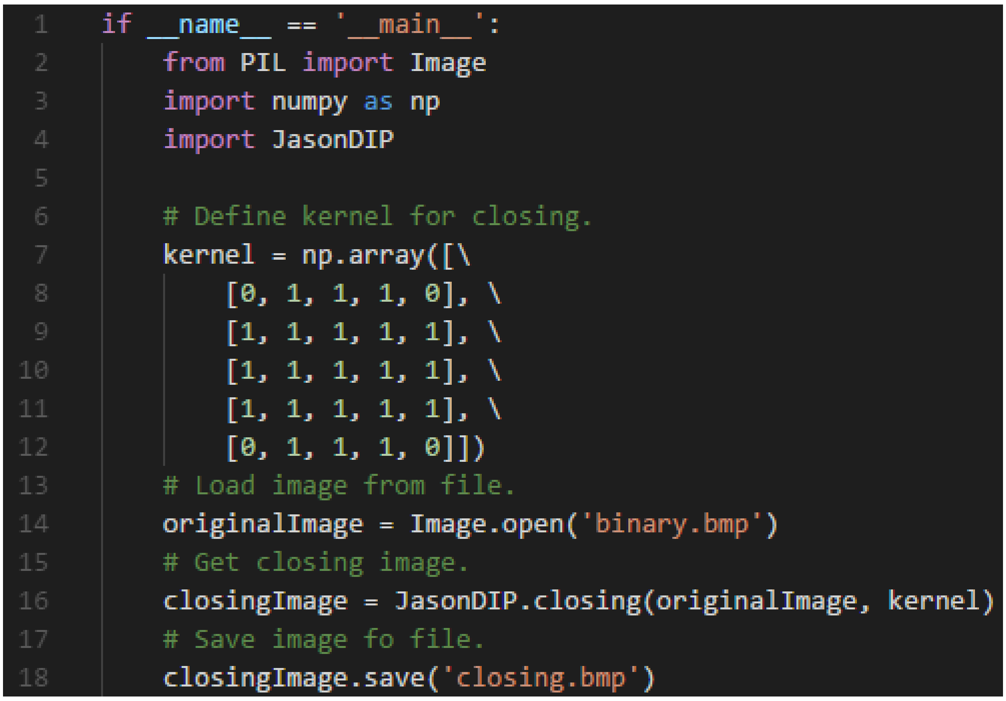
Figure 2.2.3.1. Code of main of opening.



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

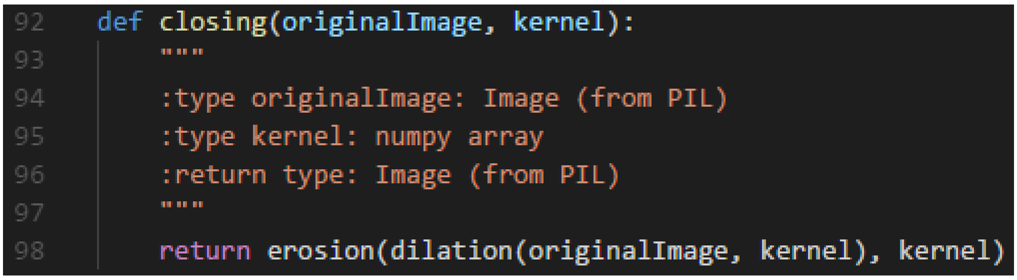
Figure 2.2.3.2. Code of function of opening.

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]])
    # Load image from file.
    originalImage = Image.open('binary.bmp')
    # Get closing image.
    closingImage = JasonDIP.closing(originalImage, kernel)
    # Save image fo file.
    closingImage.save('closing.bmp')
```

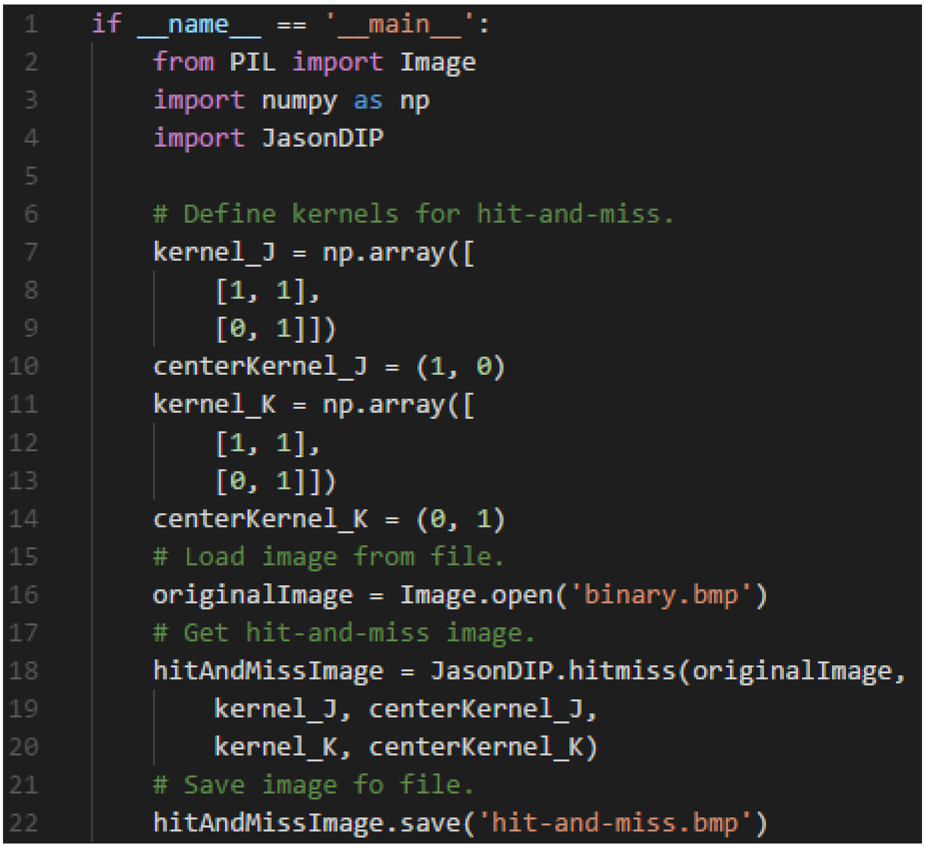
Figure 2.2.4.1. Code of main of closing.



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

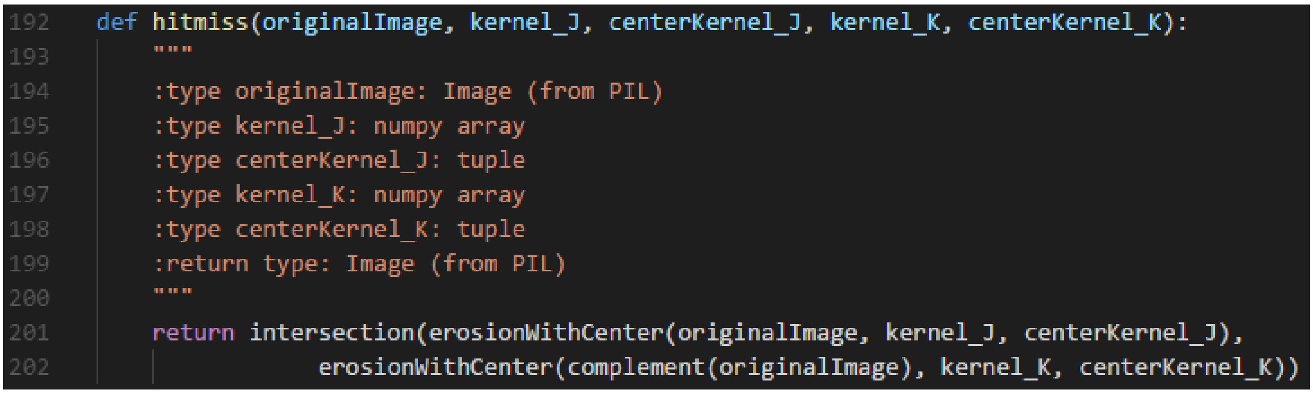
Figure 2.2.4.2. Code of function of closing.

2.2.5. Code fragments of hit-and-miss



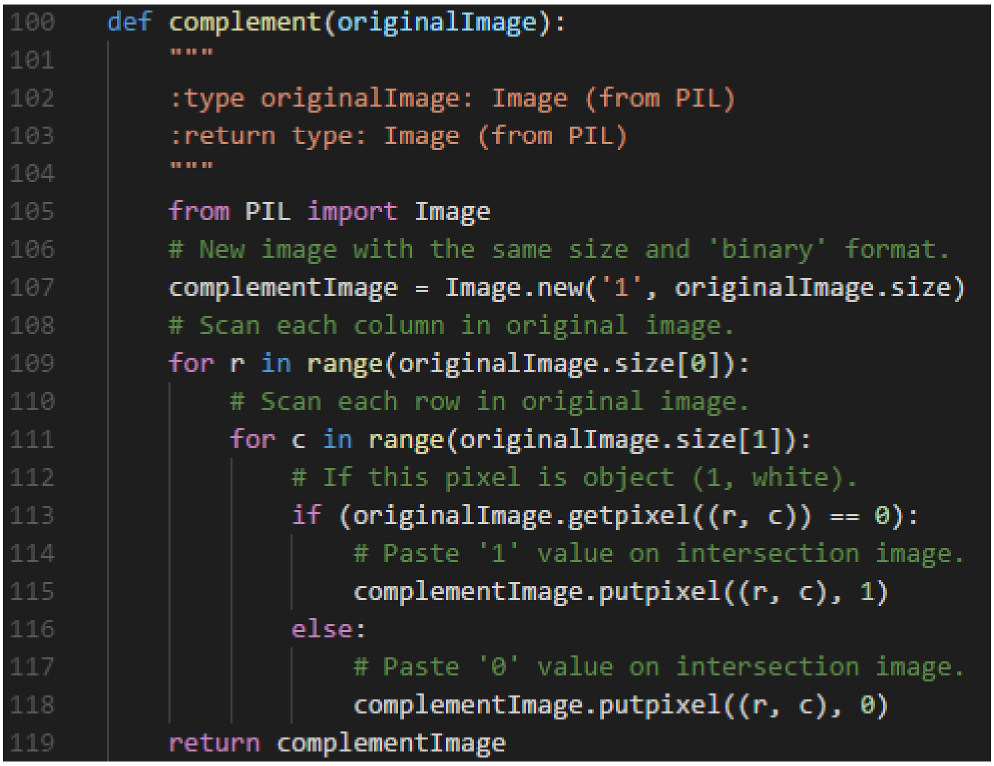
```python
if __name__ == '__main__':
    from PIL import Image
    import numpy as np
    import JasonDIP

    # Define kernels for hit-and-miss.
    kernel_J = np.array([
        [1, 1],
        [0, 1]])
    centerKernel_J = (1, 0)
    kernel_K = np.array([
        [1, 1],
        [0, 1]])
    centerKernel_K = (0, 1)
    # Load image from file.
    originalImage = Image.open('binary.bmp')
    # Get hit-and-miss image.
    hitAndMissImage = JasonDIP.hitmiss(originalImage,
        kernel_J, centerKernel_J,
        kernel_K, centerKernel_K)
    # Save image fo file.
    hitAndMissImage.save('hit-and-miss.bmp')
```

Figure 2.2.5.1. Code of main of hit-and-miss.



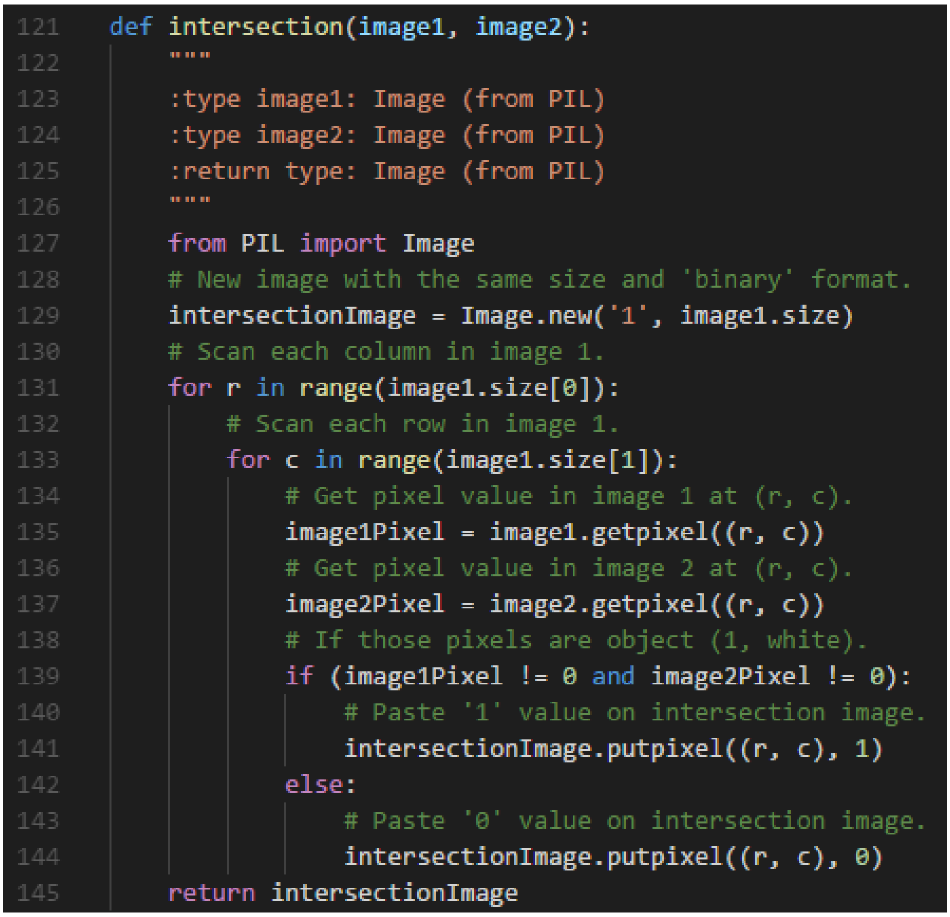
```python
def hitmiss(originalImage, kernel_J, centerKernel_J, kernel_K, centerKernel_K):
    """
    :type originalImage: Image (from PIL)
    :type kernel_J: numpy array
    :type centerKernel_J: tuple
    :type kernel_K: numpy array
    :type centerKernel_K: tuple
    :return type: Image (from PIL)
    """
    return intersection(erosionWithCenter(originalImage, kernel_J, centerKernel_J),
                erosionWithCenter(complement(originalImage), kernel_K, centerKernel_K))
```

Figure 2.2.5.2. Code of function of hit-and-miss.



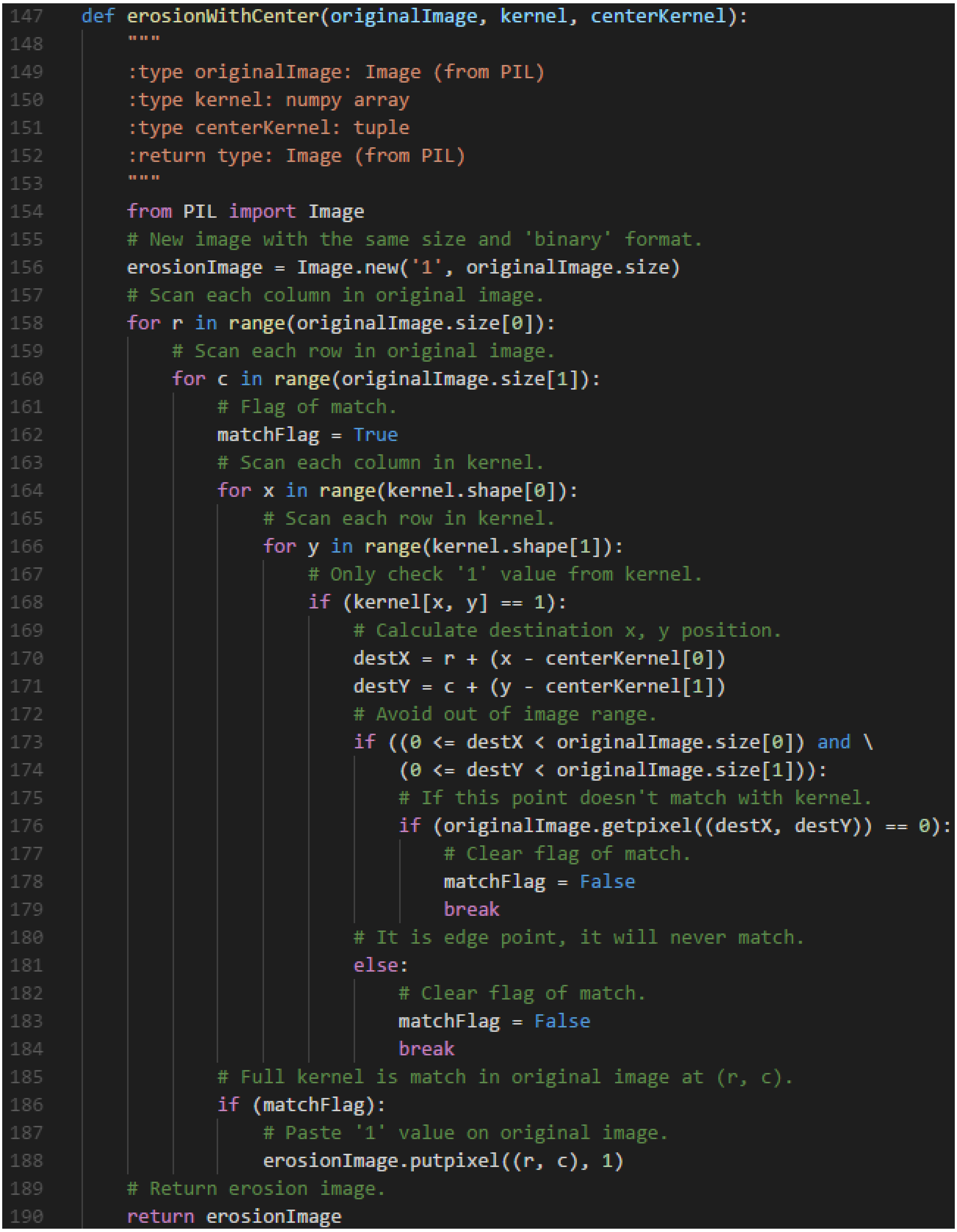
```python
def complement(originalImage):
    """
    :type originalImage: Image (from PIL)
    :return type: Image (from PIL)
    """
    from PIL import Image
    # New image with the same size and 'binary' format.
    complementImage = Image.new('1', originalImage.size)
    # Scan each column in original image.
    for r in range(originalImage.size[0]):
        # Scan each row in original image.
        for c in range(originalImage.size[1]):
            # If this pixel is object (1, white).
            if (originalImage.getpixel((r, c)) == 0):
                # Paste '1' value on intersection image.
                complementImage.putpixel((r, c), 1)
            else:
                # Paste '0' value on intersection image.
                complementImage.putpixel((r, c), 0)
    return complementImage
```

Figure 2.2.5.3. Code of function of complement.



```python
def intersection(image1, image2):
    """
    :type image1: Image (from PIL)
    :type image2: Image (from PIL)
    :return type: Image (from PIL)
    """
    from PIL import Image
    # New image with the same size and 'binary' format.
    intersectionImage = Image.new('1', image1.size)
    # Scan each column in image 1.
    for r in range(image1.size[0]):
        # Scan each row in image 1.
        for c in range(image1.size[1]):
            # Get pixel value in image 1 at (r, c).
            image1Pixel = image1.getpixel((r, c))
            # Get pixel value in image 2 at (r, c).
            image2Pixel = image2.getpixel((r, c))
            # If those pixels are object (1, white).
            if (image1Pixel != 0 and image2Pixel != 0):
                # Paste '1' value on intersection image.
                intersectionImage.putpixel((r, c), 1)
            else:
                # Paste '0' value on intersection image.
                intersectionImage.putpixel((r, c), 0)
    return intersectionImage
```

Figure 2.2.5.4. Code of function of intersection.



```python
def erosionWithCenter(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 'binary' format.
    erosionImage = Image.new('1', 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]):
            # Flag of match.
            matchFlag = True
            # 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 '1' value from 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])):
                            # If this point doesn't match with kernel.
                            if (originalImage.getpixel((destX, destY)) == 0):
                                # Clear flag of match.
                                matchFlag = False
                                break
                        # It is edge point, it will never match.
                        else:
                            # Clear flag of match.
                            matchFlag = False
                            break
            # Full kernel is match in original image at (r, c).
            if (matchFlag):
                # Paste '1' value on original image.
                erosionImage.putpixel((r, c), 1)
    # Return erosion image.
    return erosionImage
```

Figure 2.2.5.5. Code of function of erosionWithCenter.

III. RESULTS

3.1. Original Image

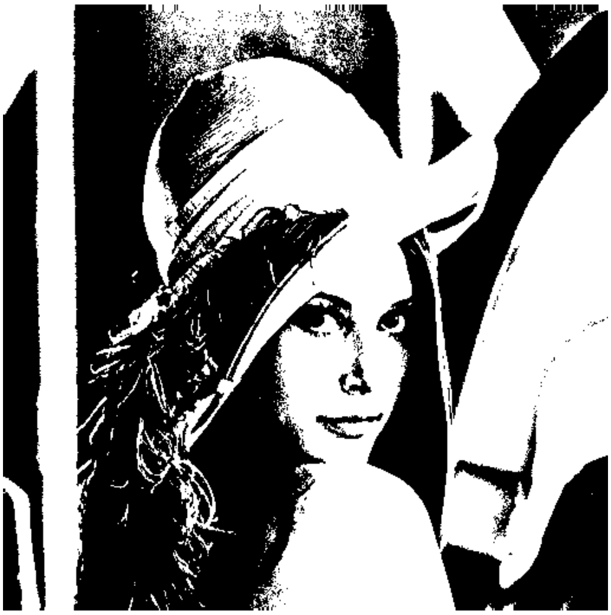


Figure 3.1. Original binary.bmp.

3.2. Results of this homework

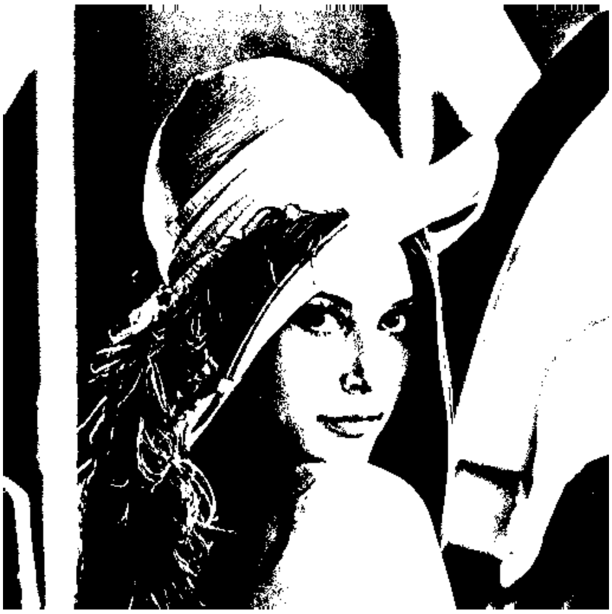


Figure 3.2.1. Original binary.bmp.

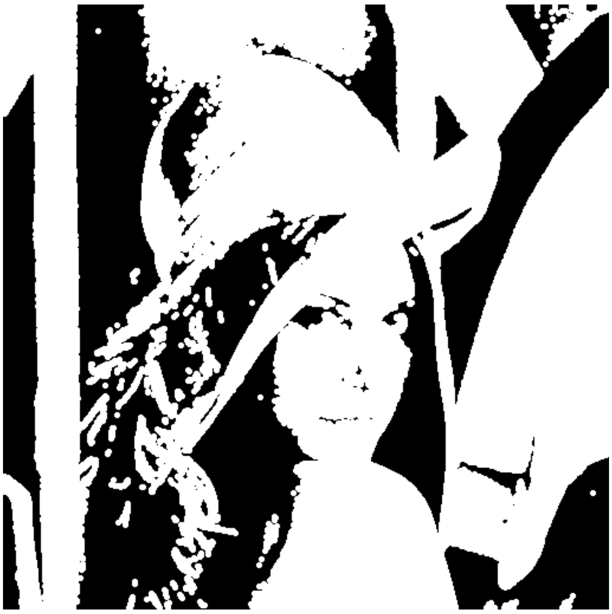


Figure 3.2.2. dilation.bmp.

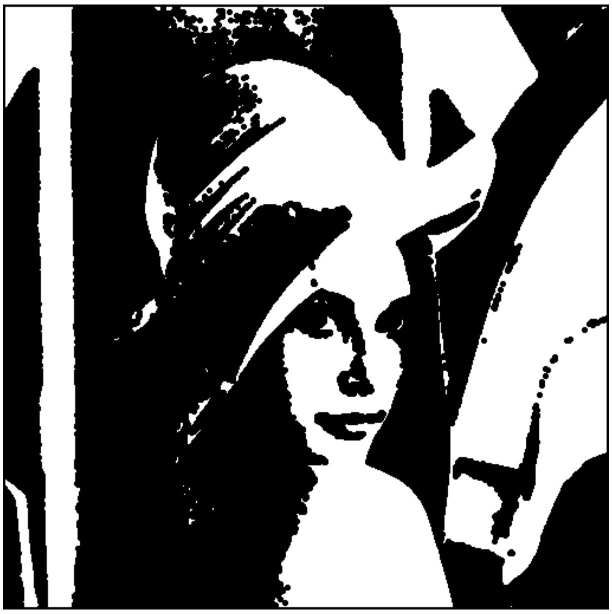


Figure 3.2.4. erosion.bmp.

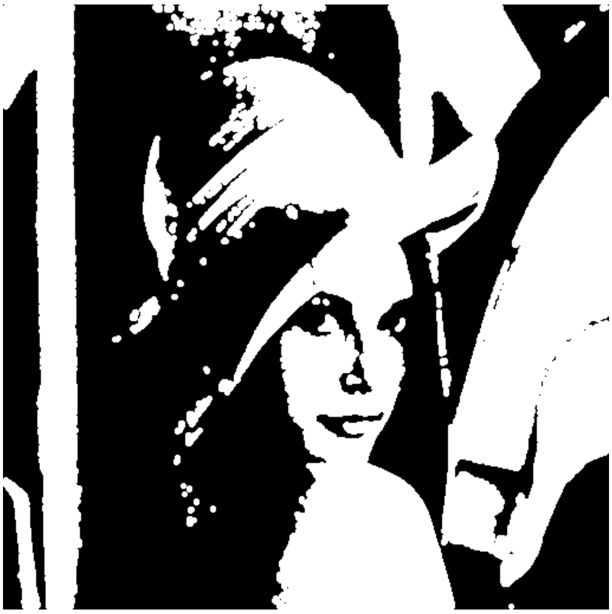


Figure 3.2.5. opening.bmp.

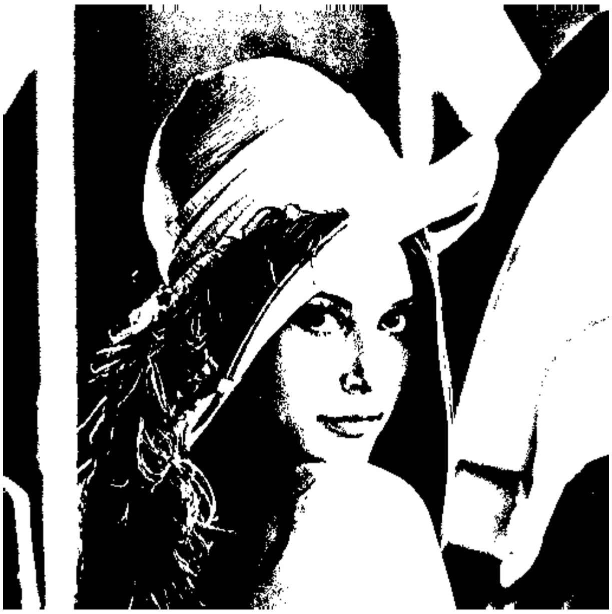


Figure 3.2.5. Original binary.bmp.

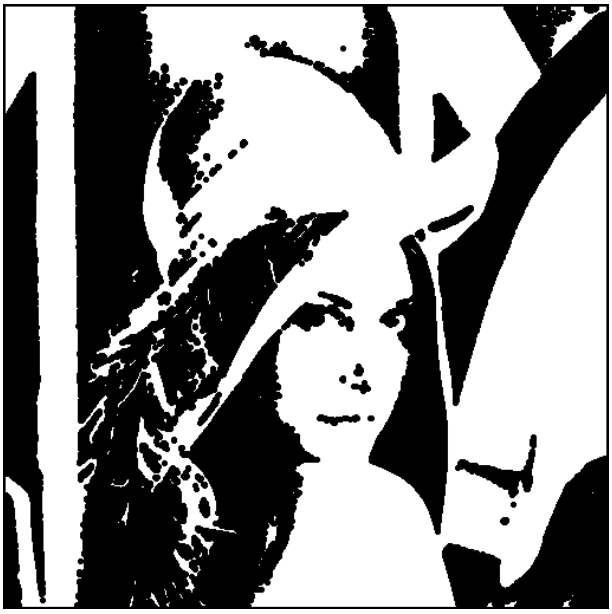


Figure 3.2.6. closing.bmp.

Figure 3.2.7. hit-and-miss.bmp.