

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

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

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

```
1  if __name__ == '__main__':
2      from PIL import Image
3      import numpy as np
4      import JasonDIP
5
6      # Define kernel for dilation.
7      kernel = np.array([\
8          [0, 1, 1, 1, 0], \
9          [1, 1, 1, 1, 1], \
10         [1, 1, 1, 1, 1], \
11         [1, 1, 1, 1, 1], \
12         [0, 1, 1, 1, 0]])
13     # Load image from file.
14     originalImage = Image.open('binary.bmp')
15     # Get dilation image.
16     dilationImage = JasonDIP.dilation(originalImage, kernel)
17     # Save image fo file.
18     dilationImage.save('dilation.bmp')
```

Figure 2.2.1.1. Code of main of dilation.

```
1  def dilation(originalImage, kernel):
2      """
3      :type originalImage: Image (from PIL)
4      :type kernel: numpy array
5      :return type: Image (from PIL)
6      """
7      from PIL import Image
8      # Get center position of kernel.
9      centerKernel = tuple([x // 2 for x in kernel.shape])
10     # New image with the same size and 'binary' format.
11     dilationImage = Image.new('1', originalImage.size)
12     # Scan each column in original image.
13     for r in range(originalImage.size[0]):
14         # Scan each row in original image.
15         for c in range(originalImage.size[1]):
16             # Get pixel value in original image at (r, c).
17             originalPixel = originalImage.getpixel((r, c))
18             # If this pixel is object (1, white).
19             if (originalPixel != 0):
20                 # Paste kernel on original image at (r, c).
21                 # Scan each column in kernel.
22                 for x in range(kernel.shape[0]):
23                     # Scan each row in kernel.
24                     for y in range(kernel.shape[1]):
25                         # Only paste '1' value from kernel.
26                         if (kernel[x, y] == 1):
27                             # Calculate destination x, y position.
28                             destX = r + (x - centerKernel[0])
29                             destY = c + (y - centerKernel[1])
30                             # Avoid out of image range.
31                             if ((0 <= destX < originalImage.size[0]) and \
32                                 (0 <= destY < originalImage.size[1])):
33                                 # Paste '1' value on original image.
34                                 dilationImage.putpixel((destX, destY), 1)
35     # Return dilation image.
36     return dilationImage
```

Figure 2.2.1.2. Code of function of dilation.

2.2.2. Code fragments of erosion

```
1  if __name__ == '__main__':
2      from PIL import Image
3      import numpy as np
4      import JasonDIP
5
6      # Define kernel for erosion.
7      kernel = np.array([\
8          [0, 1, 1, 1, 0], \
9          [1, 1, 1, 1, 1], \
10         [1, 1, 1, 1, 1], \
11         [1, 1, 1, 1, 1], \
12         [0, 1, 1, 1, 0]])
13     # Load image from file.
14     originalImage = Image.open('binary.bmp')
15     # Get erosion image.
16     erosionImage = JasonDIP.erosion(originalImage, kernel)
17     # Save image fo file.
18     erosionImage.save('erosion.bmp')
```

Figure 2.2.2.1. Code of main of erosion.

```

38 def erosion(originalImage, kernel):
39     """
40     :type originalImage: Image (from PIL)
41     :type kernel: numpy array
42     :return type: Image (from PIL)
43     """
44     from PIL import Image
45     # Get center position of kernel.
46     centerKernel = tuple([x // 2 for x in kernel.shape])
47     # New image with the same size and 'binary' format.
48     erosionImage = Image.new('1', originalImage.size)
49     # Scan each column in original image.
50     for r in range(originalImage.size[0]):
51         # Scan each row in original image.
52         for c in range(originalImage.size[1]):
53             # Flag of match.
54             matchFlag = True
55             # Scan each column in kernel.
56             for x in range(kernel.shape[0]):
57                 # Scan each row in kernel.
58                 for y in range(kernel.shape[1]):
59                     # Only check '1' value from kernel.
60                     if (kernel[x, y] == 1):
61                         # Calculate destination x, y position.
62                         destX = r + (x - centerKernel[0])
63                         destY = c + (y - centerKernel[1])
64                         # Avoid out of image range.
65                         if ((0 <= destX < originalImage.size[0]) and \
66                             (0 <= destY < originalImage.size[1])):
67                             # If this point doesn't match with kernel.
68                             if (originalImage.getpixel((destX, destY)) == 0):
69                                 # Clear flag of match.
70                                 matchFlag = False
71                                 break
72                         # It is edge point, it will never match.
73                     else:
74                         # Clear flag of match.
75                         matchFlag = False
76                         break
77             # Full kernel is match in original image at (r, c).
78             if (matchFlag):
79                 # Paste '1' value on original image.
80                 erosionImage.putpixel((r, c), 1)
81 # Return erosion image.
82 return erosionImage

```

Figure 2.2.2.2. Code of function of erosion.

2.2.3. Code fragments of opening

```
1  if __name__ == '__main__':
2      from PIL import Image
3      import numpy as np
4      import JasonDIP
5
6      # Define kernel for opening.
7      kernel = np.array([\
8          [0, 1, 1, 1, 0], \
9          [1, 1, 1, 1, 1], \
10         [1, 1, 1, 1, 1], \
11         [1, 1, 1, 1, 1], \
12         [0, 1, 1, 1, 0]])
13     # Load image from file.
14     originalImage = Image.open('binary.bmp')
15     # Get opening image.
16     openingImage = JasonDIP.opening(originalImage, kernel)
17     # Save image fo file.
18     openingImage.save('opening.bmp')
```

Figure 2.2.3.1. Code of main of opening.

```
84  def opening(originalImage, kernel):
85      """
86      :type originalImage: Image (from PIL)
87      :type kernel: numpy array
88      :return type: Image (from PIL)
89      """
90      return dilation(erosion(originalImage, kernel), kernel)
```

Figure 2.2.3.2. Code of function of opening.

2.2.4. Code fragments of closing

```
1  if __name__ == '__main__':
2      from PIL import Image
3      import numpy as np
4      import JasonDIP
5
6      # Define kernel for closing.
7      kernel = np.array([\
8          [0, 1, 1, 1, 0], \
9          [1, 1, 1, 1, 1], \
10         [1, 1, 1, 1, 1], \
11         [1, 1, 1, 1, 1], \
12         [0, 1, 1, 1, 0]])
13     # Load image from file.
14     originalImage = Image.open('binary.bmp')
15     # Get closing image.
16     closingImage = JasonDIP.closing(originalImage, kernel)
17     # Save image fo file.
18     closingImage.save('closing.bmp')
```

Figure 2.2.4.1. Code of main of closing.

```
92  def closing(originalImage, kernel):
93      """
94      :type originalImage: Image (from PIL)
95      :type kernel: numpy array
96      :return type: Image (from PIL)
97      """
98      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

```

1  if __name__ == '__main__':
2      from PIL import Image
3      import numpy as np
4      import JasonDIP
5
6      # Define kernels for hit-and-miss.
7      kernel_J = np.array([
8          [1, 1],
9          [0, 1]])
10     centerKernel_J = (1, 0)
11     kernel_K = np.array([
12         [1, 1],
13         [0, 1]])
14     centerKernel_K = (0, 1)
15     # Load image from file.
16     originalImage = Image.open('binary.bmp')
17     # Get hit-and-miss image.
18     hitAndMissImage = JasonDIP.hitmiss(originalImage,
19         kernel_J, centerKernel_J,
20         kernel_K, centerKernel_K)
21     # Save image fo file.
22     hitAndMissImage.save('hit-and-miss.bmp')

```

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

```

192 def hitmiss(originalImage, kernel_J, centerKernel_J, kernel_K, centerKernel_K):
193     """
194     :type originalImage: Image (from PIL)
195     :type kernel_J: numpy array
196     :type centerKernel_J: tuple
197     :type kernel_K: numpy array
198     :type centerKernel_K: tuple
199     :return type: Image (from PIL)
200     """
201     return intersection(erosionWithCenter(originalImage, kernel_J, centerKernel_J),
202         erosionWithCenter(complement(originalImage), kernel_K, centerKernel_K))

```

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

```

100 def complement(originalImage):
101     """
102     :type originalImage: Image (from PIL)
103     :return type: Image (from PIL)
104     """
105     from PIL import Image
106     # New image with the same size and 'binary' format.
107     complementImage = Image.new('1', originalImage.size)
108     # Scan each column in original image.
109     for r in range(originalImage.size[0]):
110         # Scan each row in original image.
111         for c in range(originalImage.size[1]):
112             # If this pixel is object (1, white).
113             if (originalImage.getpixel((r, c)) == 0):
114                 # Paste '1' value on intersection image.
115                 complementImage.putpixel((r, c), 1)
116             else:
117                 # Paste '0' value on intersection image.
118                 complementImage.putpixel((r, c), 0)
119     return complementImage

```

Figure 2.2.5.3. Code of function of complement.

```

121 def intersection(image1, image2):
122     """
123     :type image1: Image (from PIL)
124     :type image2: Image (from PIL)
125     :return type: Image (from PIL)
126     """
127     from PIL import Image
128     # New image with the same size and 'binary' format.
129     intersectionImage = Image.new('1', image1.size)
130     # Scan each column in image 1.
131     for r in range(image1.size[0]):
132         # Scan each row in image 1.
133         for c in range(image1.size[1]):
134             # Get pixel value in image 1 at (r, c).
135             image1Pixel = image1.getpixel((r, c))
136             # Get pixel value in image 2 at (r, c).
137             image2Pixel = image2.getpixel((r, c))
138             # If those pixels are object (1, white).
139             if (image1Pixel != 0 and image2Pixel != 0):
140                 # Paste '1' value on intersection image.
141                 intersectionImage.putpixel((r, c), 1)
142             else:
143                 # Paste '0' value on intersection image.
144                 intersectionImage.putpixel((r, c), 0)
145     return intersectionImage

```

Figure 2.2.5.4. Code of function of intersection.

```

147 def erosionWithCenter(originalImage, kernel, centerKernel):
148     """
149     :type originalImage: Image (from PIL)
150     :type kernel: numpy array
151     :type centerKernel: tuple
152     :return type: Image (from PIL)
153     """
154     from PIL import Image
155     # New image with the same size and 'binary' format.
156     erosionImage = Image.new('1', originalImage.size)
157     # Scan each column in original image.
158     for r in range(originalImage.size[0]):
159         # Scan each row in original image.
160         for c in range(originalImage.size[1]):
161             # Flag of match.
162             matchFlag = True
163             # Scan each column in kernel.
164             for x in range(kernel.shape[0]):
165                 # Scan each row in kernel.
166                 for y in range(kernel.shape[1]):
167                     # Only check '1' value from kernel.
168                     if (kernel[x, y] == 1):
169                         # Calculate destination x, y position.
170                         destX = r + (x - centerKernel[0])
171                         destY = c + (y - centerKernel[1])
172                         # Avoid out of image range.
173                         if ((0 <= destX < originalImage.size[0]) and \
174                             (0 <= destY < originalImage.size[1])):
175                             # If this point doesn't match with kernel.
176                             if (originalImage.getpixel((destX, destY)) == 0):
177                                 # Clear flag of match.
178                                 matchFlag = False
179                                 break
180                         # It is edge point, it will never match.
181                     else:
182                         # Clear flag of match.
183                         matchFlag = False
184                         break
185             # Full kernel is match in original image at (r, c).
186             if (matchFlag):
187                 # Paste '1' value on original image.
188                 erosionImage.putpixel((r, c), 1)
189     # Return erosion image.
190     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.

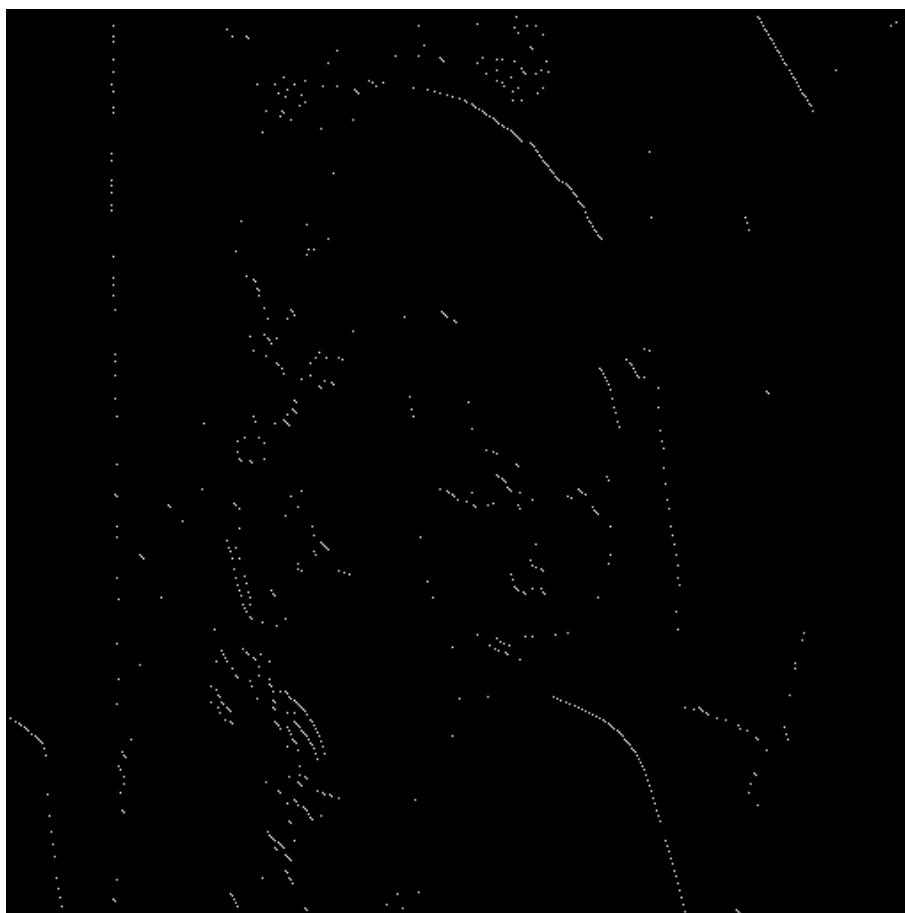


Figure 3.2.7. hit-and-miss.bmp.