

922 U0610 電腦視覺 Computer Vision

Homework 6

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 Yokoi connectivity number with following rules:

- A. Please binarize leba.bmp with threshold 128.
- B. Please down sampling binary.bmp from 512x512 to 64x64, using 8x8 blocks as unit and take the topmost-left pixel as the down sampling data.
- C. Print Yokoi connectivity number to text file.

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. Yokoi h function for 4-connectivity

$$h(b, c, d, e) = \begin{cases} q & \text{if } b = c \text{ and } (d \neq b \vee e \neq b) \\ r & \text{if } b = c \text{ and } (d = b \wedge e = b) \\ s & \text{if } b \neq c \text{ and } (d = b \wedge e = b) \end{cases}$$

2.1.2. Yokoi f function for 4-connectivity

$$f(a_1, a_2, a_3, a_4) = \begin{cases} 5, & \text{if } a_1 = a_2 = a_3 = a_4 = r \\ n, & \text{where } n = \text{number of } \{ \#a_k | a_k = q \}, \text{ otherwise} \end{cases}$$

2.2. Code Fragments

2.2.1. Code fragments of this homework

```

144 if __name__ == '__main__':
145     from PIL import Image
146     import numpy as np
147
148     # Load image from file.
149     originalImage = Image.open('lena.bmp')
150     # Get binary image.
151     binaryImage = getBinaryImage(originalImage, 128)
152     # Save binary image fo file.
153     binaryImage.save('binary.bmp')
154
155     # Get downsampling image.
156     downsamplingImage = downsampling(binaryImage, 8)
157     # Save downsampling image fo file.
158     downsamplingImage.save('downsampling.bmp')
159
160     # Get Yokoi Connectivity Number.
161     YokoiConnectivityNumber = YokoiConnectivityNumber(downsamplingImage)
162     # Save Yokoi Connectivity Number to file.
163     np.savetxt('YokoiConnectivityNumber.txt',
164               YokoiConnectivityNumber.T,
165               delimiter=',', fmt='%s')
```

Figure 2.2.1.1. Code of main function.

```

1  def getBinaryImage(originalImage, threshold):
2      """
3          :type originalImage: Image (from PIL)
4          :type threshold: int
5          :return type: Image (from PIL)
6      """
7      from PIL import Image
8      # New image with the same size and 'binary' format.
9      binaryImage = Image.new('1', originalImage.size)
10     # Scan each column in original image.
11     for c in range(originalImage.size[0]):
12         # Scan each row in original image.
13         for r in range(originalImage.size[1]):
14             # Get pixel value in original image at (c, r).
15             originalPixel = originalImage.getpixel((c, r))
16             if (originalPixel >= threshold):
17                 # Put pixel value '1' to binary image.
18                 binaryImage.putpixel((c, r), 1)
19             else:
20                 # Put pixel value '0' to binary image.
21                 binaryImage.putpixel((c, r), 0)
22     # Return binary image.
23     return binaryImage

```

Figure 2.2.1.2. Code of binarize.

```

25  def downsampling(originalImage, sampleFactor):
26      """
27          :type originalImage: Image (from PIL)
28          :type sampleFactor: int
29          :return type: Image (from PIL)
30      """
31      from PIL import Image
32      # Calculate the width and height of downsampling image.
33      downsamplingWidth = int(originalImage.size[0] / sampleFactor)
34      downsamplingHeight = int(originalImage.size[1] / sampleFactor)
35      # New image with the downsampling size and 'binary' format.
36      downsamplingImage = Image.new('1', (downsamplingWidth, downsamplingHeight))
37      # Scan each column in downsampling image.
38      for c in range(downsamplingImage.size[0]):
39          # Scan each row in downsampling image.
40          for r in range(downsamplingImage.size[1]):
41              # Get pixel value in original image at (c * sampleFactor, r * sampleFactor).
42              originalPixel = originalImage.getpixel((c * sampleFactor, r * sampleFactor))
43              # Put pixel to downsampling image.
44              downsamplingImage.putpixel((c, r), originalPixel)
45      # Return downsampling image.
46      return downsamplingImage

```

Figure 2.2.1.3. Code of down sampling.

```
48 def getNeighborhoodPixels(originalImage, position):
49     """
50     :type originalImage: Image (from PIL)
51     :type position: tuple
52     :return type: numpy array
53     """
54     # Allocate memory space of neighborhoodPixels.
55     neighborhoodPixels = np.zeros(9)
56     # Get x and y of position.
57     x, y = position
58     # Scan dx from -1 to 1.
59     for dx in range(3):
60         # Scan dy from -1 to 1.
61         for dy in range(3):
62             # Calculate destination x, y position.
63             destX = x + (dx - 1)
64             destY = y + (dy - 1)
65             # Avoid out of image range.
66             if ((0 <= destX < originalImage.size[0]) and \
67                 (0 <= destY < originalImage.size[1])):
68                 # Get neighborhood pixel values.
69                 neighborhoodPixels[3 * dy + dx] = originalImage.getpixel((destX, destY))
70             # It is out of image range.
71             else:
72                 # Padding zeros when it is out of image range.
73                 neighborhoodPixels[3 * dy + dx] = 0
74     # Original order: [[x0, x1, x2], [x3, x4, x5], [x6, x7, x8]]
75     # Sort pixels in [[x7, x2, x6], [x3, x0, x1], [x8, x4, x5]] order.
76     neighborhoodPixels = [
77         neighborhoodPixels[4], neighborhoodPixels[5], neighborhoodPixels[1],
78         neighborhoodPixels[3], neighborhoodPixels[7], neighborhoodPixels[8],
79         neighborhoodPixels[2], neighborhoodPixels[0], neighborhoodPixels[6]]
80     # Return Neighborhood Pixels.
81     return neighborhoodPixels
```

Figure 2.2.1.4. Code of getting neighborhood pixels.

```
83 def hFunctionYokoi(b, c, d, e):
84     """
85     :type b: int
86     :type c: int
87     :type d: int
88     :type e: int
89     :return type: str
90     """
91     if ((b == c) and (b != d or b != e)):
92         return 'q'
93     if ((b == c) and (b == d and b == e)):
94         return 'r'
95     if (b != c):
96         return 's'
```

Figure 2.2.1.5. Code of Yokoi h function.

```
98 def fFunctionYokoi(a1, a2, a3, a4):
99     """
100     :type a1: str
101     :type a2: str
102     :type a3: str
103     :type a4: str
104     :return type: str
105     """
106     # a1 == a2 == a3 == a4 == r
107     if ([a1, a2, a3, a4].count('r') == 4):
108         # Return label 5 (interior).
109         return 5
110     else:
111         # Return count of 'q'.
112         # 0: Isolated, 1: Edge, 2: Connecting, 3: Branching, 4: Crossing.
113         return [a1, a2, a3, a4].count('q')
```

Figure 2.2.1.6. Code of Yokoi f function.

III. RESULTS

3.1. Original Image



Figure 3.1. Original lena.bmp.

3.2. Results of binary and down sampling



Figure 3.2.1. binary.bmp.



Figure 3.2.2. downsampling.bmp.

3.3. Results of Yokoi connectivity number

1	11111111	121111111111122322221	1111111111111	0 0
2	15555551	115555555511 2 11 11	115555555511	0
3	15555551	1 2115555112 21112221	15555555551	21
4	15555551	1 2 155112 22221511	155555555511	1
5	15555551	22 2112 22 121 0 0	1555555555511	0
6	15555551	1 2 21 2 1 1	155555555551 0	0
7	15555551	12 1 121111 1321	1555555555511	
8	15111551	1322 1155551111	1555555555551	
9	111 1551	1 12155555511	15555555555511	
10	11 1551	21155555511	1551115555511	
11	21 1551	2 1555555111	1551 1155511	
12	1 1551	2 15555555511	1551 115551	1
13	1551	11211555555551	1551 15511	12
14	1551	155555555555511	1551 1111	111
15	1551	1 2221155555555511	1151 11	1151
16	1551	2 22 1 15555555555511	151 11111	1551
17	1551	2 1 115555555555551	151 115551	11551
18	1551	2 11555555555555511	11511155511	115551
19	1551	12 11555555555555555551		155551
20	1551	11 0 2215555555555555555112		1155551
21	1551	111 22 15555555555555555551 1		1555551
22	1551	1511 1 1251121111121115555555111		11555551
23	1551	15521 1 121 1 11 1 1555555111 0		15555551
24	1551	1151 132 2 115555111 0		11555551
25	1551	151 0 322 115555111 121		15555551
26	1551	1221 2 1555551 131		115555551
27	1551	2 0 1 115555511 1		1155555551
28	1551	2 0 0 115555551 0		1 15555551
29	1551	2 1155555551		2115555551
30	1551	1 0 115555555551		1555555551
31	1551	1 11511115555521 1		115555555551
32	1551	1 1 11111 1155511 2		155555555551
33	1551	131 111 15111 2		155555555551
34	1551	121 0 1121 1 111 1 2		1155555555551
35	1551	11 111 1 221 11 1 2		1555555555551
36	1551	12 0 1 21 121 11 1111 2		1555555555551
37	1551	1 1 12 22 151111111551 2		1155555555551
38	1551	1 2 1555551115511 1		15555555555551
39	1551	2 0 0 22 12555551 15551 1		15555555555551
40	1551	1 1 1555511 11511 2		11555555555551
41	1551	0 0 21 155551 1 151 2		15555555555551
42	1551	2 15555112 151 2		15555555555551
43	1551	1 1 1 1155555511111 2		155555555555551
44	1551	2 22 111511111212 2115555555555551		
45	1551	0 1 12 151 2 1 15555555111555551		
46	1551	0 0 0 1111 121 15555551 1555551		
47	1551	0 11111111 155555551 1555551		
48	1551	0 115551 155555551 1555511		
49	1551	15551 211111111 155511		
50	11521	1 12 122155511 2 11 115511		
51	1 151 0	1 1 155555111 2111 15511		
52	22 1511	1 15555555111 155111 1511		
53	22 1511	1 15555555551 155551 1151		
54	2 151	0 1 11155555555511 155511 1511		
55	2 1521	0 1 155555555555511 15551 12151		
56	2 151	121 155555555555551 155511 1551		
57	2 1511	0 1555555555555551 115551 1511		
58	21 1511	11 1555555555555551 111111151		
59	11 151	0 1155555555555551 111511		
60	11 151	15555555555555551 151		
61	11 151	0 115555555555555551 211		
62	11 151	1155555555555555511 1		
63	11 151	0 155555555555555551		
64	11 111	0 1211111111111111111		

Figure 3.3.1. Yokoi connectivity number.