

922 U0610 電腦視覺 Computer Vision

Homework 8

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 noise removal with following rules:

- A. Generate Gaussian noise with amplitude of 10 and 30.
- B. Generate salt-and-pepper noise with probability of 0.1 and 0.05.
- C. Use the 3x3 and 5x5 box filter on noise images.
- D. Use the 3x3 and 5x5 median filter on noise images.
- E. Use opening-then-closing and closing-then-opening filter on noise images.
- F. Calculate the signal-to-noise-ratio (SNR) of noise images.

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. Box filter

- 3x3 box filter
- 5x5 box filter

2.1.2. Median filter

- 3x3 median filter
- 5x5 median filter

2.1.3. Opening-then-closing

2.1.4. Closing-then-opening

2.2. Code Fragments

2.2.1. Code fragments of this homework

```

1  def getGaussianNoiseImage(originalImage, amplitude):
2      """
3      :type originalImage: Image (from PIL)
4      :type amplitude: float
5      :return type: Image (from PIL)
6      """
7      from PIL import Image
8      import random
9      # Copy image from original image.
10     gaussianNoiseImage = originalImage.copy()
11     # Scan each column in original image.
12     for c in range(originalImage.size[0]):
13         # Scan each row in original image.
14         for r in range(originalImage.size[1]):
15             # Get pixel value with gaussian noise.
16             noisePixel = int(originalImage.getpixel((c, r)) + amplitude * random.gauss(0, 1))
17             # Limit pixel value at 255.
18             if noisePixel > 255:
19                 noisePixel = 255
20             # Put pixel to noise image.
21             gaussianNoiseImage.putpixel((c, r), noisePixel)
22     return gaussianNoiseImage

```

Figure 2.2.1.1. Code of get Gaussian Noise image function.

```

24  def getSaltAndPepperImage(originalImage, probability):
25      """
26      :type originalImage: Image (from PIL)
27      :type probability: float
28      :return type: Image (from PIL)
29      """
30      from PIL import Image
31      import random
32      # Copy image from original image.
33      saltAndPepperImage = originalImage.copy()
34      # Scan each column in original image.
35      for c in range(originalImage.size[0]):
36          # Scan each row in original image.
37          for r in range(originalImage.size[1]):
38              # Get random value.
39              randomValue = random.uniform(0, 1)
40              if (randomValue <= probability):
41                  # Put black pixel(pepper) to image.
42                  saltAndPepperImage.putpixel((c, r), 0)
43              elif (randomValue >= 1 - probability):
44                  # Put white pixel(salt) to image.
45                  saltAndPepperImage.putpixel((c, r), 255)
46              else:
47                  # Put original pixel to image.
48                  saltAndPepperImage.putpixel((c, r), originalImage.getpixel((c, r)))
49      return saltAndPepperImage

```

Figure 2.2.1.2. Code of get Salt-and-pepper image function.

```

51 def boxFilter(originalImage, boxWidth, boxHeight):
52     """
53     :type originalImage: Image (from PIL)
54     :type boxWidth: integer
55     :type boxHeight: integer
56     :return type: Image (from PIL)
57     """
58     # Calculate center of kernel.
59     centerKernel = (boxWidth // 2, boxHeight // 2)
60     # Copy image from original image.
61     boxFilterImage = originalImage.copy()
62     # Scan each column in original image.
63     for c in range(originalImage.size[0]):
64         # Scan each row in original image.
65         for r in range(originalImage.size[1]):
66             # Create empty list.
67             boxPixels = []
68             # Scan each column in box.
69             for x in range(boxWidth):
70                 # Scan each row in box.
71                 for y in range(boxHeight):
72                     # Calculate destination x, y position.
73                     destX = c + (x - centerKernel[0])
74                     destY = r + (y - centerKernel[1])
75                     # Avoid out of image range.
76                     if ((0 <= destX < originalImage.size[0]) and \
77                         (0 <= destY < originalImage.size[1])):
78                         # Get pixel value in original image at (destX, destY).
79                         originalPixel = originalImage.getpixel((destX, destY))
80                         # Append pixel to list.
81                         boxPixels.append(originalPixel)
82                 boxFilterImage.putpixel((c, r), int(sum(boxPixels) / len(boxPixels)))
83     return boxFilterImage

```

Figure 2.2.1.3. Code of box filter.

```

85 def medianFilter(originalImage, boxWidth, boxHeight):
86     """
87     :type originalImage: Image (from PIL)
88     :type boxWidth: integer
89     :type boxHeight: integer
90     :return type: Image (from PIL)
91     """
92     # Calculate center of kernel.
93     centerKernel = (boxWidth // 2, boxHeight // 2)
94     # Copy image from original image.
95     medianFilterImage = originalImage.copy()
96     # Scan each column in original image.
97     for c in range(originalImage.size[0]):
98         # Scan each row in original image.
99         for r in range(originalImage.size[1]):
100             # Create empty list.
101             boxPixels = []
102             # Scan each column in box.
103             for x in range(boxWidth):
104                 # Scan each row in box.
105                 for y in range(boxHeight):
106                     # Calculate destination x, y position.
107                     destX = c + (x - centerKernel[0])
108                     destY = r + (y - centerKernel[1])
109                     # Avoid out of image range.
110                     if ((0 <= destX < originalImage.size[0]) and \
111                         (0 <= destY < originalImage.size[1])):
112                         # Get pixel value in original image at (destX, destY).
113                         originalPixel = originalImage.getpixel((destX, destY))
114                         # Append pixel to list.
115                         boxPixels.append(originalPixel)
116             # Sort pixels in box.
117             boxPixels.sort()
118             # Get Median pixel.
119             medianPixel = boxPixels[len(boxPixels) // 2]
120             # Put median pixel to image.
121             medianFilterImage.putpixel((c, r), medianPixel)
122     return medianFilterImage

```

Figure 2.2.1.4. Code of median filter.

```

124 def dilation(originalImage, kernel, centerKernel):
125     """
126     :type originalImage: Image (from PIL)
127     :type kernel: numpy array
128     :type centerKernel: tuple
129     :return type: Image (from PIL)
130     """
131     from PIL import Image
132     # New image with the same size and 'grayscale' format.
133     dilationImage = Image.new('L', originalImage.size)
134     # Scan each column in original image.
135     for c in range(originalImage.size[0]):
136         # Scan each row in original image.
137         for r in range(originalImage.size[1]):
138             # Record local max. pixel value.
139             localMaxPixel = 0
140             # Scan each column in kernel.
141             for x in range(kernel.shape[0]):
142                 # Scan each row in kernel.
143                 for y in range(kernel.shape[1]):
144                     # Only check value '1' in kernel.
145                     if (kernel[x, y] == 1):
146                         # Calculate destination x, y position.
147                         destX = c + (x - centerKernel[0])
148                         destY = r + (y - centerKernel[1])
149                         # Avoid out of image range.
150                         if ((0 <= destX < originalImage.size[0]) and \
151                             (0 <= destY < originalImage.size[1])):
152                             # Get pixel value in original image at (destX, destY).
153                             originalPixel = originalImage.getpixel((destX, destY))
154                             # Update local max. pixel value.
155                             localMaxPixel = max(localMaxPixel, originalPixel)
156             # Paste local max. pixel value on original image.
157             dilationImage.putpixel((c, r), localMaxPixel)
158     # Return dilation image.
159     return dilationImage

```

Figure 2.2.1.5. Code of dilation.

```

161 def erosion(originalImage, kernel, centerKernel):
162     """
163     :type originalImage: Image (from PIL)
164     :type kernel: numpy array
165     :type centerKernel: tuple
166     :return type: Image (from PIL)
167     """
168     from PIL import Image
169     # New image with the same size and 'grayscale' format.
170     erosionImage = Image.new('L', originalImage.size)
171     # Scan each column in original image.
172     for c in range(originalImage.size[0]):
173         # Scan each row in original image.
174         for r in range(originalImage.size[1]):
175             # Record local min. pixel value.
176             localMinPixel = 255
177             # Scan each column in kernel.
178             for x in range(kernel.shape[0]):
179                 # Scan each row in kernel.
180                 for y in range(kernel.shape[1]):
181                     # Only check value '1' in kernel.
182                     if (kernel[x, y] == 1):
183                         # Calculate destination x, y position.
184                         destX = c + (x - centerKernel[0])
185                         destY = r + (y - centerKernel[1])
186                         # Avoid out of image range.
187                         if ((0 <= destX < originalImage.size[0]) and \
188                             (0 <= destY < originalImage.size[1])):
189                             # Get pixel value in original image at (destX, destY).
190                             originalPixel = originalImage.getpixel((destX, destY))
191                             # Update local min. pixel value.
192                             localMinPixel = min(localMinPixel, originalPixel)
193             # Paste local min. pixel value on original image.
194             erosionImage.putpixel((c, r), localMinPixel)
195     # Return erosion image.
196     return erosionImage

```

Figure 2.2.1.6. Code of erosion.

```
198 def opening(originalImage, kernel, centerKernel):
199     """
200     :type originalImage: Image (from PIL)
201     :type kernel: numpy array
202     :type centerKernel: tuple
203     :return type: Image (from PIL)
204     """
205     return dilation(erosion(originalImage, kernel, centerKernel), kernel, centerKernel)
206
207 def closing(originalImage, kernel, centerKernel):
208     """
209     :type originalImage: Image (from PIL)
210     :type kernel: numpy array
211     :type centerKernel: tuple
212     :return type: Image (from PIL)
213     """
214     return erosion(dilation(originalImage, kernel, centerKernel), kernel, centerKernel)
215
216 def openingThenClosing(originalImage, kernel, centerKernel):
217     """
218     :type originalImage: Image (from PIL)
219     :type kernel: numpy array
220     :type centerKernel: tuple
221     :return type: Image (from PIL)
222     """
223     return closing(opening(originalImage, kernel, centerKernel), kernel, centerKernel)
224
225 def closingThenOpening(originalImage, kernel, centerKernel):
226     """
227     :type originalImage: Image (from PIL)
228     :type kernel: numpy array
229     :type centerKernel: tuple
230     :return type: Image (from PIL)
231     """
232     return opening(closing(originalImage, kernel, centerKernel), kernel, centerKernel)
```

Figure 2.2.1.7. Code of opening and closing.

```

234 def getSNR(signalImage, noiseImage):
235     """
236     :type signalImage: Image (from PIL)
237     :type noiseImage: Image (from PIL)
238     :return type: float
239     """
240     import math
241     # Clear mu and power of signal and noise.
242     muSignal = 0
243     powerSignal = 0
244     muNoise = 0
245     powerNoise = 0
246
247     # Scan each column in signal image.
248     for c in range(signalImage.size[0]):
249         # Scan each row in signal image.
250         for r in range(signalImage.size[1]):
251             muSignal = muSignal + signalImage.getpixel((c, r))
252     # Average mu of signal.
253     muSignal = muSignal / (signalImage.size[0] * signalImage.size[1])
254
255     # Scan each column in noise image.
256     for c in range(noiseImage.size[0]):
257         # Scan each row in noise image.
258         for r in range(noiseImage.size[1]):
259             muNoise = muNoise + (noiseImage.getpixel((c, r)) - signalImage.getpixel((c, r)))
260     # Average mu of noise.
261     muNoise = muNoise / (noiseImage.size[0] * noiseImage.size[1])
262
263     # Scan each column in signal image.
264     for c in range(signalImage.size[0]):
265         # Scan each row in signal image.
266         for r in range(signalImage.size[1]):
267             powerSignal = powerSignal + math.pow(signalImage.getpixel((c, r)) - muSignal, 2)
268     # Average power of signal.
269     powerSignal = powerSignal / (signalImage.size[0] * signalImage.size[1])
270
271     # Scan each column in noise image.
272     for c in range(noiseImage.size[0]):
273         # Scan each row in noise image.
274         for r in range(noiseImage.size[1]):
275             powerNoise = powerNoise + math.pow((noiseImage.getpixel((c, r)) - signalImage.getpixel((c, r))) - muNoise, 2)
276     # Average mu of noise.
277     powerNoise = powerNoise / (noiseImage.size[0] * noiseImage.size[1])

```

Figure 2.2.1.8. Code of SNR.

III. RESULTS

3.1. Original Image



Figure 3.1. Original lena.bmp.

3.2. Results of Gaussian Noise Image



Figure 3.2.1. gaussianNoise_10.bmp.



Figure 3.2.2. gaussianNoise_30.bmp.

3.3. Results of Salt-and-Pepper



Figure 3.3.1. saltAndPepper_0_05.bmp.



Figure 3.3.2. saltAndPepper_0_10.bmp.

3.4. Results of 3x3 Box Filter



Figure 3.4.1.
gaussianNoise_10_box_3x3.bmp.



Figure 3.4.2.
gaussianNoise_30_box_3x3.bmp.



Figure 3.4.3.
saltAndPepper_0_05_box_3x3.bmp



Figure 3.4.4.
saltAndPepper_0_10_box_3x3.bmp

3.5. Results of 5x5 Box Filter



Figure 3.5.1.
gaussianNoise_10_box_5x5.bmp.



Figure 3.5.2.
gaussianNoise_30_box_5x5.bmp.



Figure 3.5.3.
saltAndPepper_0_05_box_5x5.bmp



Figure 3.5.4.
saltAndPepper_0_10_box_5x5.bmp

3.6. Results of 3x3 Median Filter



Figure 3.6.1.
gaussianNoise_10_median_3x3.bmp.



Figure 3.6.2.
gaussianNoise_30_median_3x3.bmp.



Figure 3.6.3.
saltAndPepper_0_05_median_3x3.bmp



Figure 3.6.4.
saltAndPepper_0_10_median_3x3.bmp

3.7. Results of 5x5 Median Filter



Figure 3.7.1.
gaussianNoise_10_median_5x5.bmp.



Figure 3.7.2.
gaussianNoise_30_median_5x5.bmp.

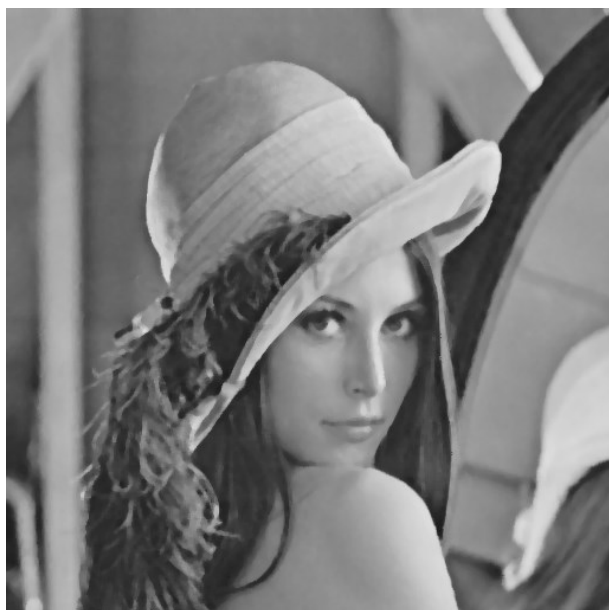


Figure 3.7.3.
saltAndPepper_0_05_median_5x5.bmp

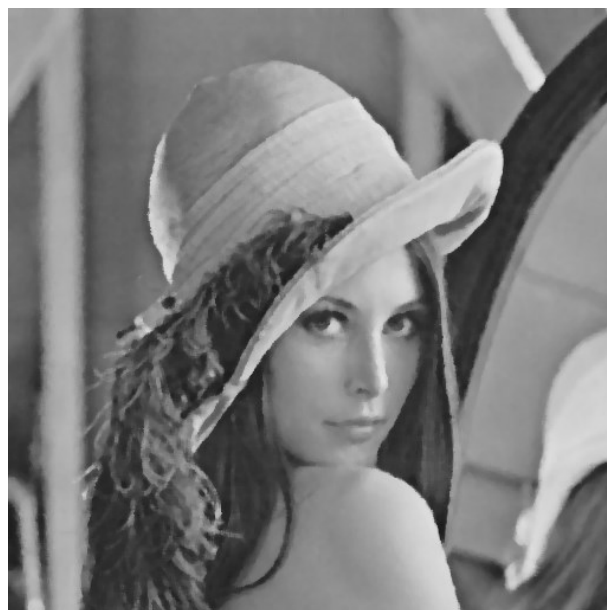


Figure 3.7.4.
saltAndPepper_0_10_median_5x5.bmp

3.8. Results of Opening-then-Closing



Figure 3.8.1.

gaussianNoise_10_openingThenClosing.bmp



Figure 3.8.2.

gaussianNoise_30_openingThenClosing.bmp



Figure 3.8.3.

saltAndPepper_0_05_openingThenClosing.bmp

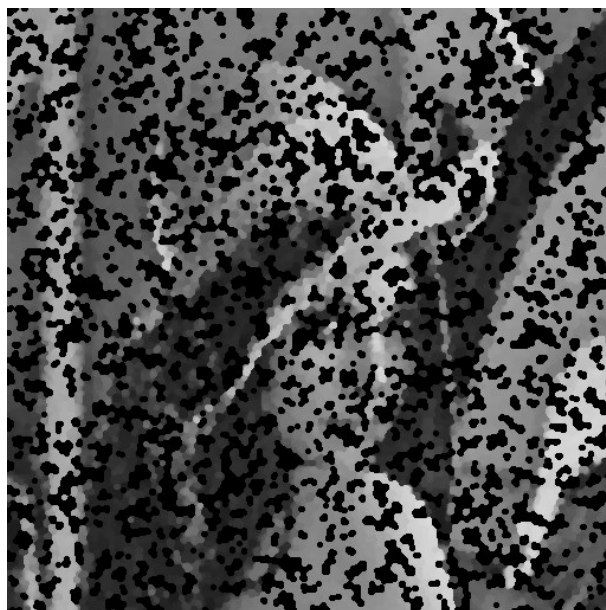


Figure 3.8.4.

saltAndPepper_0_10_openingThenClosing.bmp

3.9. Results of Closing-then-Opening



Figure 3.9.1.

gaussianNoise_10_closingThenOpening.bmp.



Figure 3.9.2.

gaussianNoise_30_closingThenOpening.bmp.

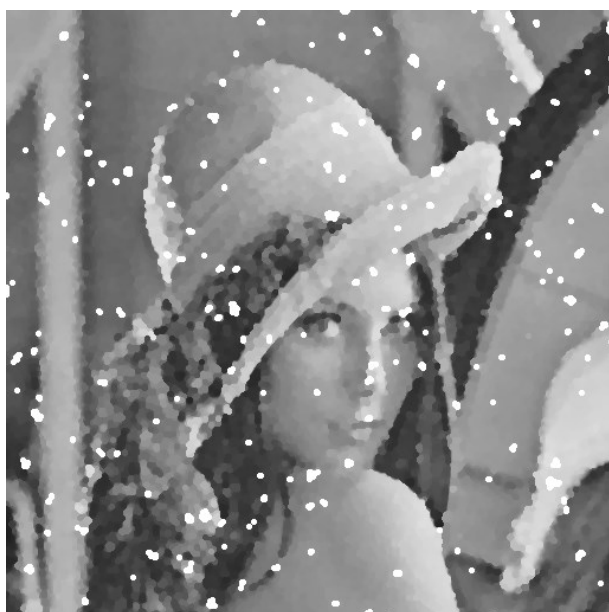


Figure 3.9.3.

saltAndPepper_0_05_closingThenOpening.bmp

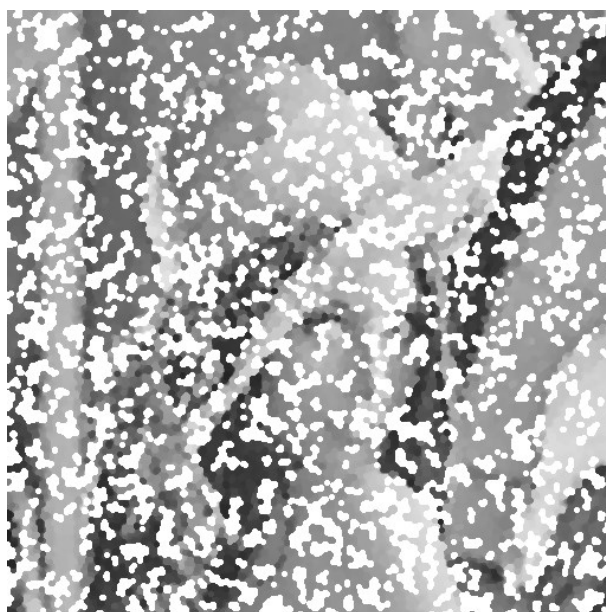


Figure 3.9.4.

saltAndPepper_0_10_closingThenOpening.bmp

3.10. Results of SNR

Filter		Gaussian Noise	
		Amplitude=10	Amplitude=30
No filter		13.577673848918188	4.179277588955975
Box	3x3	17.731140149360535	12.610573226583657
	5x5	14.857729621739733	13.311954047893185
Median	3x3	17.651055018705332	11.068588450952369
	5x5	15.986375902746365	12.923616645551498
Opening-then-Closing		13.246475409841953	11.107239975017215
Closing-then-Opening		13.587499304142945	11.161285816210508

Filter		Salt-and-Pepper	
		Probability=0.05	Probability=0.10
No filter		0.919599038789096	-2.1331972742191896
Box	3x3	9.437939435017782	6.276181353721542
	5x5	11.144625145075523	8.45026635583591
Median	3x3	19.038446617464842	14.871190557310829
	5x5	16.3801023822922	15.78842254312368
Opening-then-Closing		5.6121449335507565	-2.234575099322334
Closing-then-Opening		5.228390279049658	-2.538225164674737