Homework 8

R13525009 羅筠笙

1. Gaussian noise with amplitude = 10

|  |  |
| --- | --- |
| Gaussian noise with amplitude = 10, SNR = 13.597449 | |
| box\_3x3, SNR = 17.745861 | box\_5x5, SNR = 14.869506 |
| median\_3x3, SNR = 17.655002 | median\_5x5, SNR = 16.015192 |
| open-then-closing, SNR = 13.240792 | closing-then-opening, SNR = 13.621662 |

1. Gaussian noise with amplitude = 30

|  |  |
| --- | --- |
| Gaussian noise with amplitude = 30, SNR = 4.173514 | |
| box\_3x3, SNR = 12.633835 | box\_5x5, SNR = 13.331065 |
| median\_3x3, SNR = 11.091201 | median\_5x5, SNR = 12.932721 |
| open-then-closing, SNR = 11.175065 | closing-then-opening, SNR = 11.190557 |

1. Salt-and-pepper noise with probability = 0.1

|  |  |
| --- | --- |
| salt-and-pepper with probability = 0.1, SNR = -2.108260 | |
| box\_3x3, SNR = 6.302114 | box\_5x5, SNR = 8.470839 |
| median\_3x3, SNR = 14.836461 | median\_5x5, SNR = 15.786323 |
| open-then-closing, SNR = -2.150020 | closing-then-opening, SNR = -2.724415 |

1. Salt-and-pepper noise with probability = 0.05

|  |  |
| --- | --- |
| salt-and-pepper with probability = 0.05, SNR = 0.882124 | |
| box\_3x3, SNR = 9.414893 | box\_5x5, SNR = 11.098395 |
| median\_3x3, SNR = 19.160758 | median\_5x5, SNR = 16.277215 |
| opening-then-closing, SNR = 5.435218 | closing-then-opening, SNR = 5.602341 |

1. # Calculate the SNR
2. def SNR(origin, noisy):
3. m, n = origin.shape
4. mu = np.mean(origin)
5. mu\_n = np.mean(noisy - origin)
6. vs = np.mean((origin - mu) \*\* 2)
7. vn = np.mean((noisy - origin - mu\_n) \*\* 2)
8. return 20 \* np.log10(np.sqrt(vs) / np.sqrt(vn))
9. # Generate the gaussian noise
10. def gaussian\_noise(imgarr, amp):
11. img\_gauss\_arr = np.zeros(imgarr.shape)
12. for i in range(imgarr.shape[0]):
13. for j in range(imgarr.shape[1]):
14. img\_gauss\_arr[i, j] = max(0, min(255, imgarr[i, j] + amp \* random.gauss(0, 1)))
15. snr = SNR(imgarr, img\_gauss\_arr)
16. print(f"SNR for Gaussian noisy with amp = {amp} is {snr:.6f}")
17. img\_gauss = im.fromarray(img\_gauss\_arr.astype(np.uint8))
18. # img\_gauss.save(f"./gauss\_{amp}.bmp")
19. # img\_gauss.show()
20. return img\_gauss\_arr
21. # Add salt and pepper noise to the image
22. def salt\_and\_pepper(imgarr, threshold):
23. img\_st\_arr = np.zeros(imgarr.shape)
24. for i in range(imgarr.shape[0]):
25. for j in range(imgarr.shape[1]):
26. tmp = random.random()
27. if tmp <= threshold:
28. img\_st\_arr[i, j] = 0
29. elif tmp > 1 - threshold:
30. img\_st\_arr[i, j] = 255
31. else:
32. img\_st\_arr[i, j] = imgarr[i, j]
33. # Calculate the SNR
34. snr = SNR(imgarr, img\_st\_arr)
35. print(f"SNR for salt & pepper with threshold = {threshold} is {snr:.6f}")
36. img\_st = im.fromarray(img\_st\_arr.astype(np.uint8))
37. # img\_st.save(f"./st\_{threshold}.bmp")
38. # img\_st.show()
39. return img\_st\_arr
40. # Expand the image array using replicate padding
41. def expand\_with\_replicate(arr, pad\_size):
42. m, n = arr.shape
43. result = np.zeros((m + 2 \* pad\_size, n + 2 \* pad\_size), dtype=arr.dtype)
44. # Fill the borders
45. result[pad\_size:-pad\_size, pad\_size:-pad\_size] = arr
46. result[:pad\_size, pad\_size:-pad\_size] = arr[0, :] # Top edge
47. result[-pad\_size:, pad\_size:-pad\_size] = arr[-1, :] # Bottom edge
48. result[pad\_size:-pad\_size, :pad\_size] = arr[:, 0][:, None] # Left edge
49. result[pad\_size:-pad\_size, -pad\_size:] = arr[:, -1][:, None] # Right edge
50. # Fill the corners
51. result[:pad\_size, :pad\_size] = arr[0, 0] # Top-left corner
52. result[:pad\_size, -pad\_size:] = arr[0, -1] # Top-right corner
53. result[-pad\_size:, :pad\_size] = arr[-1, 0] # Bottom-left corner
54. result[-pad\_size:, -pad\_size:] = arr[-1, -1] # Bottom-right corner
55. return result
56. # Define box filter
57. def box\_filter(img, k, noise\_name):
58. pad\_size = (k - 1) // 2
59. img\_padded = expand\_with\_replicate(img, pad\_size)
60. m, n = img.shape
61. result = np.zeros((m, n), dtype=float)
62. for i in range(m):
63. for j in range(n):
64. result[i, j] = img\_padded[i:i + k, j:j + k].mean()
65. result\_img = im.fromarray(result.astype(np.uint8))
66. # result\_img.show()
67. result\_img.save(f"./box\_{k}\_{noise\_name}.bmp")
68. snr = SNR(img\_arr, result)
69. print(f"SNR for box filter {k}x{k}: {snr:.6f}")
70. # Define median filter
71. def median(img, k, noise\_name):
72. pad\_size = (k - 1) // 2
73. img\_padded = expand\_with\_replicate(img, pad\_size)
74. result = np.zeros([img\_size0, img\_size1])
75. for i in range(img\_size0):
76. for j in range(img\_size1):
77. result[i, j] = np.median(img\_padded[i:i+k, j:j+k])
78. result\_img = im.fromarray(result.astype(np.uint8))
79. # result\_img.show()
80. result\_img.save(f"./median\_{k}\_{noise\_name}.bmp")
81. snr = SNR(img\_arr, result)
82. print(f"SNR for mdeian filter {k}x{k}: {snr:.6f}")
83. # Define the kernel the hw asked
84. oct\_kernel = np.array([[0, 1, 1, 1, 0],
85. [1, 1, 1, 1, 1],
86. [1, 1, 1, 1, 1],
87. [1, 1, 1, 1, 1],
88. [0, 1, 1, 1, 0]])
89. def dilation(img, kernel):
90. kernel\_size = len(kernel)
91. dilation\_img = np.zeros\_like(img)
92. for i in range(img\_size0):
93. for j in range(img\_size1):
94. max\_value = 0
95. for ki in range(kernel\_size):
96. for kj in range(kernel\_size):
97. ni, nj = i + ki - kernel\_size // 2, j + kj - kernel\_size // 2
98. if 0 <= ni < img\_size0 and 0 <= nj < img\_size1:
99. if kernel[ki][kj] == 1:
100. pixel\_value = img[ni][nj]
101. if pixel\_value > max\_value:
102. max\_value = pixel\_value
103. dilation\_img[i, j] = max\_value
104. return dilation\_img.astype(np.uint8)
105. def erosion(img, kernel):
106. kernel\_size = len(kernel)
107. erosion\_img = np.zeros\_like(img)
108. for i in range(img\_size0):
109. for j in range(img\_size1):
110. min\_value = 255
111. for ki in range(kernel\_size):
112. for kj in range(kernel\_size):
113. ni, nj = i + ki - kernel\_size // 2, j + kj - kernel\_size // 2
114. if 0 <= ni < img\_size0 and 0 <= nj < img\_size1:
115. if kernel[ki][kj] == 1:
116. pixel\_value = img[ni][nj]
117. if pixel\_value < min\_value:
118. min\_value = pixel\_value
119. erosion\_img[i, j] = min\_value
120. return erosion\_img.astype(np.uint8)
121. def opening(img, kernel):
122. return (dilation(erosion(img, kernel), kernel))
123. def closing(img, kernel):
124. return (erosion(dilation(img, kernel), kernel))
125. def opening\_then\_closing(img, noise\_name="", kernel=oct\_kernel):
126. otc\_arr = closing(opening(img, kernel), kernel)
127. snr = SNR(img\_arr/255, otc\_arr/255)
128. print(f"SNR for opening-then-closing filter : {snr:.6f}")
129. otc = im.fromarray(otc\_arr)
130. otc.save(f"otc\_{noise\_name}.bmp")
131. return otc\_arr
132. def closing\_then\_opening(img, noise\_name="", kernel=oct\_kernel):
133. cto\_arr = opening(closing(img, kernel), kernel)
134. snr = SNR(img\_arr/255, cto\_arr/255)
135. print(f"SNR for closing-then-opening filter : {snr:.6f}")
136. cto = im.fromarray(cto\_arr)
137. cto.save(f"cto\_{noise\_name}.bmp")
138. return cto\_arr