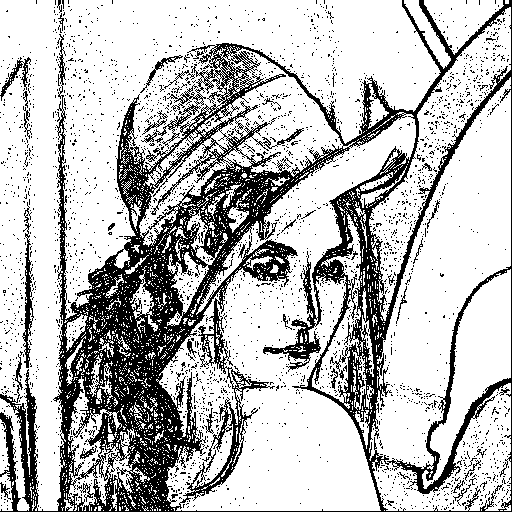
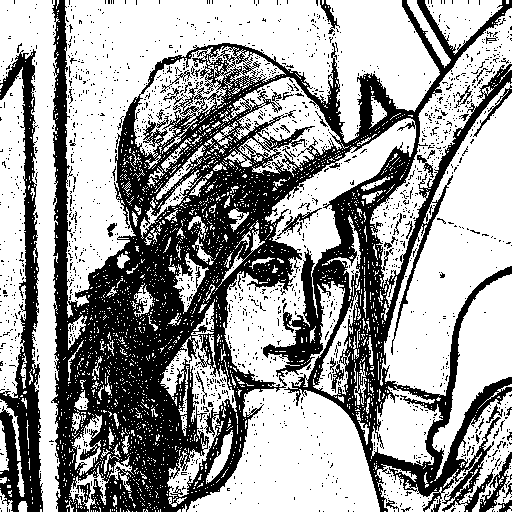
Homework 9

R13525009 羅筠笙

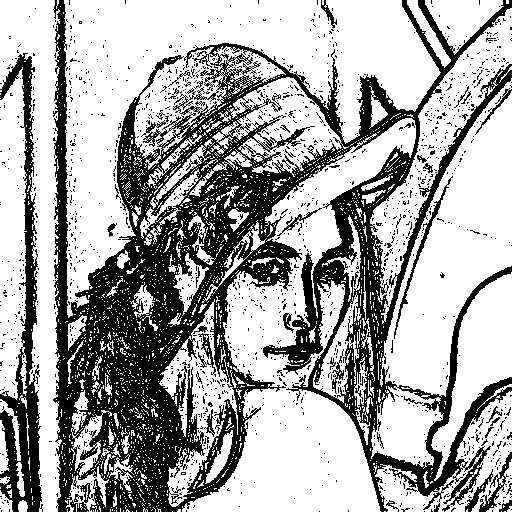
1. **Robert's Operator: 12**  
   Description: Uses a 2x2 kernel to compute the gradient magnitude for edge detection by applying convolution to the input image.



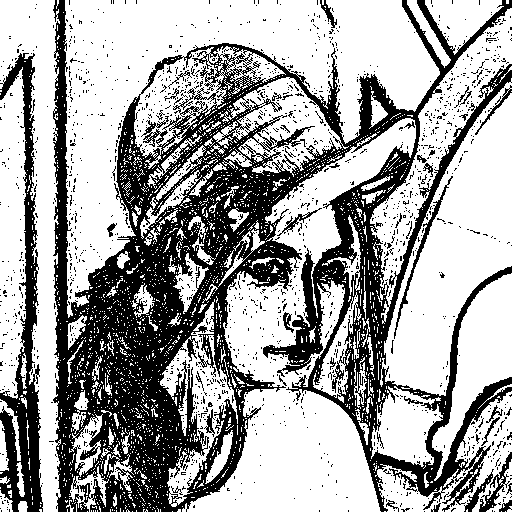
1. def robert(img\_arr, threshold):
2. res\_arr = np.zeros([img\_size0, img\_size1])
3. for i in range(img\_size0-1):
4. for j in range(img\_size1-1):
5. '''
6. r1 = [-1 0
7. 0 1]
8. r2 = [[0 -1
9. 1 0]
10. '''
11. r1 = - int(img\_arr[i][j]) + int(img\_arr[i+1][j+1])
12. r2 = - int(img\_arr[i][j+1]) + int(img\_arr[i+1][j])
13. grad = np.sqrt(r1\*\*2 + r2\*\*2)
14. res\_arr[i, j] = 255 if grad < threshold else 0
15. return res\_arr
16. **Prewitt's Edge Detector: 24**  
    Description: Applies 3x3 kernels for horizontal and vertical gradients using convolution.



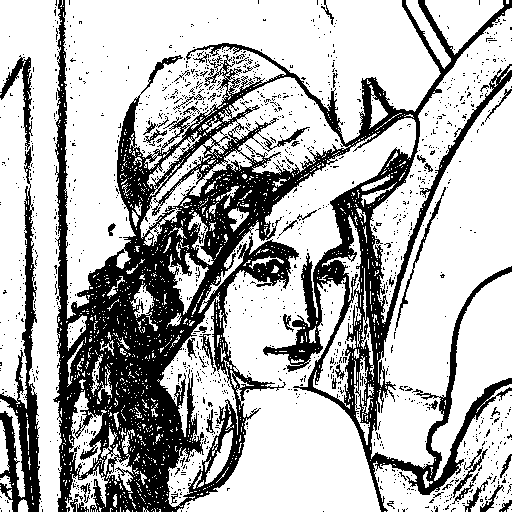
1. def prewitt\_edge(img\_arr, threshold):
2. img\_arr = expand\_with\_replicate(img\_arr, 1)
3. m, n = img\_arr.shape
4. res\_arr = np.zeros([img\_size0, img\_size1])
5. for i in range(img\_size0):
6. for j in range(img\_size1):
7. '''
8. p1 = [-1 -1 -1
9. 0 0 0
10. 1 1 1]
11. p2 = [-1 0 1
12. -1 0 1
13. -1 0 1]
14. '''
15. p1 = - int(img\_arr[i, j]) - int(img\_arr[i, j+1]) - int(img\_arr[i, j+2]) + int(img\_arr[i+2, j]) + int(img\_arr[i+2, j+1]) + int(img\_arr[i+2, j+2])
16. p2 = - int(img\_arr[i, j]) - int(img\_arr[i+1, j]) - int(img\_arr[i+2, j]) + int(img\_arr[i, j+2]) + int(img\_arr[i+1, j+2]) + int(img\_arr[i+2, j+2])
17. grad = np.sqrt(p1\*\*2 + p2\*\*2)
18. res\_arr[i, j] = 255 if grad < threshold else 0
19. return res\_arr
20. **Sobel's Edge Detector: 38**  
    Description: Implements Sobel's operator by convolving the image with 3x3 kernels that emphasize the center pixels.



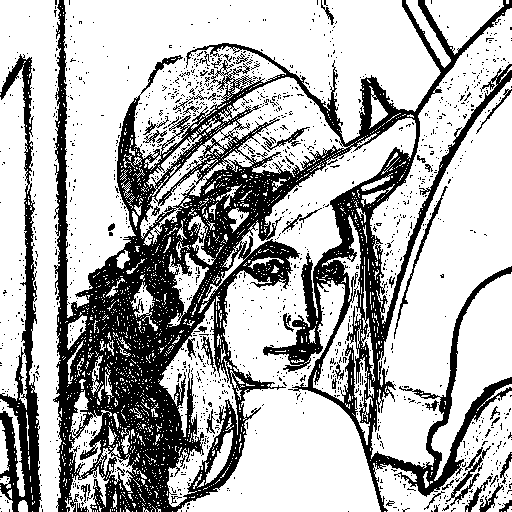
1. def sobel\_edge(img\_arr, threshold):
2. img\_arr = expand\_with\_replicate(img\_arr, 1)
3. m, n = img\_arr.shape
4. res\_arr = np.zeros([img\_size0, img\_size1])
5. for i in range(img\_size0):
6. for j in range(img\_size1):
7. '''
8. s1 = [-1 -2 -1
9. 0 0 0
10. 1 2 1]
11. s2 = [-1 0 1
12. -2 0 2
13. -1 0 1]
14. '''
15. s1 = - int(img\_arr[i, j]) - 2 \* int(img\_arr[i, j+1]) - int(img\_arr[i, j+2]) + int(img\_arr[i+2, j]) + 2 \* int(img\_arr[i+2, j+1]) + int(img\_arr[i+2, j+2])
16. s2 = - int(img\_arr[i, j]) - 2 \* int(img\_arr[i+1, j]) - int(img\_arr[i+2, j]) + int(img\_arr[i, j+2]) + 2 \* int(img\_arr[i+1, j+2]) + int(img\_arr[i+2, j+2])
17. grad = np.sqrt(s1\*\*2 + s2\*\*2)
18. res\_arr[i, j] = 255 if grad < threshold else 0
19. return res\_arr
20. **Frei and Chen's Gradient Operator: 30**  
    Description: Uses modified Sobel-like 3x3 kernels, applying specific weights to capture diagonal and straight-edge gradients



1. def frel\_and\_chen\_grad(img\_arr, threshold):
2. img\_arr = expand\_with\_replicate(img\_arr, 1)
3. m, n = img\_arr.shape
4. res\_arr = np.zeros([img\_size0, img\_size1])
5. for i in range(img\_size0):
6. for j in range(img\_size1):
7. '''
8. f1 = [-1 sqrt(-2) -1
9. 0 0 0
10. 1 sqrt(2) 1]
11. f2 = [-1 0 1
12. sqrt(-)2 0 sqrt(2)
13. -1 0 1]
14. '''
15. f1 = - int(img\_arr[i, j]) - np.sqrt(2) \* int(img\_arr[i, j+1]) - int(img\_arr[i, j+2]) + int(img\_arr[i+2, j]) + np.sqrt(2) \* int(img\_arr[i+2, j+1]) + int(img\_arr[i+2, j+2])
16. f2 = - int(img\_arr[i, j]) - np.sqrt(2) \* int(img\_arr[i+1, j]) - int(img\_arr[i+2, j]) + int(img\_arr[i, j+2]) + np.sqrt(2) \* int(img\_arr[i+1, j+2]) + int(img\_arr[i+2, j+2])
17. grad = np.sqrt(f1\*\*2 + f2\*\*2)
18. res\_arr[i, j] = 255 if grad < threshold else 0
19. return res\_arr
20. **Kirsch's Compass Operator: 135**  
    Description: Applies eight 3x3 directional masks (compass masks) to calculate gradient magnitudes in all compass directions.



1. def kirsch\_comass(img\_arr, threshold):
2. img\_arr = expand\_with\_replicate(img\_arr, 1)
3. m, n = img\_arr.shape
4. res\_arr = np.zeros([img\_size0, img\_size1])
5. for i in range(img\_size0):
6. for j in range(img\_size1):
7. '''
8. k0 = [-3 -3 5
9. -3 0 5
10. -3 -3 5]
11. k1 = [-3 5 5
12. -3 0 5
13. -3 -3 -3]
14. k2 = [5 5 5
15. -3 0 -3
16. -3 -3 -3]
17. k3 = [5 5 -3
18. 5 0 -3
19. -3 -3 -3]
20. k4 = [5 -3 -3
21. 5 0 -3
22. 5 -3 -3]
23. k5 = [-3 -3 -3
24. 5 0 -3
25. 5 5 -3]
26. k6 = [-3 -3 -3
27. -3 0 -3
28. 5 5 5]
29. k7 = [-3 -3 -3
30. -3 0 5
31. -3 5 5]
32. '''
33. k0 = - 3 \* int(img\_arr[i, j]) - 3 \* int(img\_arr[i, j+1]) + 5 \* int(img\_arr[i, j+2]) \
34. - 3 \* int(img\_arr[i+1, j]) + 5 \* int(img\_arr[i+1, j+2]) \
35. - 3 \* int(img\_arr[i+2, j]) - 3 \* int(img\_arr[i+2, j+1]) + 5 \* int(img\_arr[i+2, j+2])
36. k1 = - 3 \* int(img\_arr[i, j]) + 5 \* int(img\_arr[i, j+1]) + 5 \* int(img\_arr[i, j+2]) \
37. - 3 \* int(img\_arr[i+1, j]) + 5 \* int(img\_arr[i+1, j+2]) \
38. - 3 \* int(img\_arr[i+2, j]) - 3 \* int(img\_arr[i+2, j+1]) - 3 \* int(img\_arr[i+2, j+2])
39. k2 = 5 \* int(img\_arr[i, j]) + 5 \* int(img\_arr[i, j+1]) + 5 \* int(img\_arr[i, j+2]) \
40. - 3 \* int(img\_arr[i+1, j]) - 3 \* int(img\_arr[i+1, j+2]) \
41. - 3 \* int(img\_arr[i+2, j]) - 3 \* int(img\_arr[i+2, j+1]) - 3 \* int(img\_arr[i+2, j+2])
42. k3 = 5 \* int(img\_arr[i, j]) + 5 \* int(img\_arr[i, j+1]) - 3 \* int(img\_arr[i, j+2]) \
43. + 5 \* int(img\_arr[i+1, j]) - 3 \* int(img\_arr[i+1, j+2]) \
44. - 3 \* int(img\_arr[i+2, j]) - 3 \* int(img\_arr[i+2, j+1]) - 3 \* int(img\_arr[i+2, j+2])
45. k4 = 5 \* int(img\_arr[i, j]) - 3 \* int(img\_arr[i, j+1]) - 3 \* int(img\_arr[i, j+2]) \
46. + 5 \* int(img\_arr[i+1, j]) - 3 \* int(img\_arr[i+1, j+2]) \
47. + 5 \* int(img\_arr[i+2, j]) - 3 \* int(img\_arr[i+2, j+1]) - 3 \* int(img\_arr[i+2, j+2])
48. k5 = - 3 \* int(img\_arr[i, j]) - 3 \* int(img\_arr[i, j+1]) - 3 \* int(img\_arr[i, j+2]) \
49. + 5 \* int(img\_arr[i+1, j]) - 3 \* int(img\_arr[i+1, j+2]) \
50. + 5 \* int(img\_arr[i+2, j]) + 5 \* int(img\_arr[i+2, j+1]) - 3 \* int(img\_arr[i+2, j+2])
51. k6 = - 3 \* int(img\_arr[i, j]) - 3 \* int(img\_arr[i, j+1]) - 3 \* int(img\_arr[i, j+2]) \
52. - 3 \* int(img\_arr[i+1, j]) - 3 \* int(img\_arr[i+1, j+2]) \
53. + 5 \* int(img\_arr[i+2, j]) + 5 \* int(img\_arr[i+2, j+1]) + 5 \* int(img\_arr[i+2, j+2])
54. k7 = - 3 \* int(img\_arr[i, j]) - 3 \* int(img\_arr[i, j+1]) - 3 \* int(img\_arr[i, j+2]) \
55. - 3 \* int(img\_arr[i+1, j]) + 5 \* int(img\_arr[i+1, j+2]) \
56. - 3 \* int(img\_arr[i+2, j]) + 5 \* int(img\_arr[i+2, j+1]) + 5 \* int(img\_arr[i+2, j+2])
57. k\_list = np.array([k0, k1, k2, k3, k4, k5, k6, k7])
58. grad = np.max(k\_list)
59. res\_arr[i, j] = 255 if grad < threshold else 0
60. return res\_arr
61. **Robinson's Compass Operator: 43**  
    Description: Implements eight directional 3x3 masks similar to Kirsch but uses simpler calculations.



1. def robinson\_compass(img\_arr, threshold):
2. img\_arr = expand\_with\_replicate(img\_arr, 1)
3. m, n = img\_arr.shape
4. res\_arr = np.zeros([img\_size0, img\_size1])
5. for i in range(img\_size0):
6. for j in range(img\_size1):
7. '''
8. r0 = [-1 0 1
9. -2 0 2
10. -1 0 1]
11. r1 = [0 1 2
12. -1 0 1
13. -2 -1 0]
14. r2 = [1 2 1
15. 0 0 0
16. -1 -2 -1]
17. r3 = [2 1 0
18. 1 0 -1
19. 0 -1 -2]
20. r4 = [1 0 -1
21. 2 0 -2
22. 1 0 -1]
23. r5 = [0 -1 -2
24. 1 0 -1
25. 2 1 0]
26. r6 = [-1 -2 -1
27. 0 0 0
28. 1 2 1]
29. r7 = [-2 -1 0
30. -1 0 1
31. 0 1 2]
32. '''
33. r0 = - int(img\_arr[i, j]) + int(img\_arr[i, j+2]) \
34. - 2 \* int(img\_arr[i+1, j]) + 2 \* int(img\_arr[i+1, j+2]) \
35. - int(img\_arr[i+2, j]) + int(img\_arr[i+2, j+2])
36. r1 = int(img\_arr[i, j+1]) + 2 \* int(img\_arr[i, j+2]) \
37. - int(img\_arr[i+1, j]) + int(img\_arr[i+1, j+2]) \
38. - 2 \* int(img\_arr[i+2, j]) - int(img\_arr[i+2, j+1])
39. r2 = int(img\_arr[i, j]) + 2 \* int(img\_arr[i, j+1]) + int(img\_arr[i, j+2]) \
40. - int(img\_arr[i+2, j]) - 2 \* int(img\_arr[i+2, j+1]) - int(img\_arr[i+2, j+2])
41. r3 = 2 \* int(img\_arr[i, j]) + int(img\_arr[i, j+1]) \
42. + int(img\_arr[i+1, j]) - int(img\_arr[i+1, j+2]) \
43. - int(img\_arr[i+2, j+1]) - 2 \* int(img\_arr[i+2, j+2])
44. r4 = int(img\_arr[i, j]) - int(img\_arr[i, j+2]) \
45. + 2 \* int(img\_arr[i+1, j]) - 2 \* int(img\_arr[i+1, j+2]) \
46. + int(img\_arr[i+2, j]) - int(img\_arr[i+2, j+2])
47. r5 = - int(img\_arr[i, j+1]) - 2 \* int(img\_arr[i, j+2]) \
48. + int(img\_arr[i+1, j]) - int(img\_arr[i+1, j+2]) \
49. + 2 \* int(img\_arr[i+2, j]) + int(img\_arr[i+2, j+1])
50. r6 = - int(img\_arr[i, j]) - 2 \* int(img\_arr[i, j+1]) - int(img\_arr[i, j+2]) \
51. + int(img\_arr[i+2, j]) + 2 \* int(img\_arr[i+2, j+1]) + int(img\_arr[i+2, j+2])
52. r7 = - 2 \* int(img\_arr[i, j]) - int(img\_arr[i, j+1]) \
53. - int(img\_arr[i+1, j]) + int(img\_arr[i+1, j+2]) \
54. + int(img\_arr[i+2, j+1]) + 2 \* int(img\_arr[i+2, j+2])
55. r\_list = np.array([r0, r1, r2, r3, r4, r5, r6, r7])
56. grad = np.max(r\_list)
57. res\_arr[i, j] = 255 if grad < threshold else 0
58. return res\_arr
59. **Nevatia-Babu 5x5 Operator: 12500**  
    Description: Applies multiple 5x5 kernels to analyze high-resolution images, detecting fine and directional edges.



1. def nevatia\_babu\_5x5(img\_arr, threshold):
2. img\_arr = expand\_with\_replicate(img\_arr, 2)
3. m, n = img\_arr.shape
4. res\_arr = np.zeros([img\_size0, img\_size1])
5. for i in range(img\_size0):
6. for j in range(img\_size1):
7. '''
8. n0 = [100 100 100 100 100
9. 100 100 100 100 100
10. 0 0 0 0 0
11. -100 -100 -100 -100 -100
12. -100 -100 -100 -100 -100]
13. n1 = [100 100 100 100 100
14. 100 100 100 78 -32
15. 100 92 0 -92 -100
16. 32 -78 -100 -100 -100
17. -100 -100 -100 -100 -100]
18. n2 = [100 100 100 32 -100
19. 100 100 92 -78 -100
20. 100 100 0 -100 -100
21. 100 78 -92 -100 -100
22. 100 -32 -100 -100 -100]
23. n3 = [-100 -100 0 100 100
24. -100 -100 0 100 100
25. -100 -100 0 100 100
26. -100 -100 0 100 100
27. -100 -100 0 100 100]
28. n4 = [-100 32 100 100 100
29. -100 -78 92 100 100
30. -100 -100 0 100 100
31. -100 -100 -92 78 100
32. -100 -100 -100 -32 100]
33. n5 = [100 100 100 100 100
34. -32 78 100 100 100
35. -100 -92 0 92 100
36. -100 -100 -100 -78 32
37. -100 -100 -100 -100 -100]
38. '''
39. n0 = 100 \* int(img\_arr[i, j]) + 100 \* int(img\_arr[i, j+1]) + 100 \* int(img\_arr[i, j+2]) + 100 \* int(img\_arr[i, j+3]) + 100 \* int(img\_arr[i, j+4]) \
40. + 100 \* int(img\_arr[i+1, j]) + 100 \* int(img\_arr[i+1, j+1]) + 100 \* int(img\_arr[i+1, j+2]) + 100 \* int(img\_arr[i+1, j+3]) + 100 \* int(img\_arr[i+1, j+4]) \
41. - 100 \* int(img\_arr[i+3, j]) - 100 \* int(img\_arr[i+3, j+1]) - 100 \* int(img\_arr[i+3, j+2]) - 100 \* int(img\_arr[i+3, j+3]) - 100 \* int(img\_arr[i+3, j+4]) \
42. - 100 \* int(img\_arr[i+4, j]) - 100 \* int(img\_arr[i+4, j+1]) - 100 \* int(img\_arr[i+4, j+2]) - 100 \* int(img\_arr[i+4, j+3]) - 100 \* int(img\_arr[i+4, j+4])
43. n1 = 100 \* int(img\_arr[i, j]) + 100 \* int(img\_arr[i, j+1]) + 100 \* int(img\_arr[i, j+2]) + 100 \* int(img\_arr[i, j+3]) + 100 \* int(img\_arr[i, j+4]) \
44. + 100 \* int(img\_arr[i+1, j]) + 100 \* int(img\_arr[i+1, j+1]) + 100 \* int(img\_arr[i+1, j+2]) + 78 \* int(img\_arr[i+1, j+3]) - 32 \* int(img\_arr[i+1, j+4]) \
45. + 100 \* int(img\_arr[i+2, j]) + 92 \* int(img\_arr[i+2, j+1]) - 92 \* int(img\_arr[i+2, j+3]) - 100 \* int(img\_arr[i+2, j+4]) \
46. + 32 \* int(img\_arr[i+3, j]) - 78 \* int(img\_arr[i+3, j+1]) - 100 \* int(img\_arr[i+3, j+2]) - 100 \* int(img\_arr[i+3, j+3]) - 100 \* int(img\_arr[i+3, j+4]) \
47. - 100 \* int(img\_arr[i+4, j]) - 100 \* int(img\_arr[i+4, j+1]) - 100 \* int(img\_arr[i+4, j+2]) - 100 \* int(img\_arr[i+4, j+3]) - 100 \* int(img\_arr[i+4, j+4])
48. n2 = 100 \* int(img\_arr[i, j]) + 100 \* int(img\_arr[i, j+1]) + 100 \* int(img\_arr[i, j+2]) + 32 \* int(img\_arr[i, j+3]) - 100 \* int(img\_arr[i, j+4]) \
49. + 100 \* int(img\_arr[i+1, j]) + 100 \* int(img\_arr[i+1, j+1]) + 92 \* int(img\_arr[i+1, j+2]) - 78 \* int(img\_arr[i+1, j+3]) - 100 \* int(img\_arr[i+1, j+4]) \
50. + 100 \* int(img\_arr[i+2, j]) + 100 \* int(img\_arr[i+2, j+1]) - 100 \* int(img\_arr[i+2, j+3]) - 100 \* int(img\_arr[i+2, j+4]) \
51. + 100 \* int(img\_arr[i+3, j]) + 78 \* int(img\_arr[i+3, j+1]) - 92 \* int(img\_arr[i+3, j+2]) - 100 \* int(img\_arr[i+3, j+3]) - 100 \* int(img\_arr[i+3, j+4]) \
52. + 100 \* int(img\_arr[i+4, j]) - 32 \* int(img\_arr[i+4, j+1]) - 100 \* int(img\_arr[i+4, j+2]) - 100 \* int(img\_arr[i+4, j+3]) - 100 \* int(img\_arr[i+4, j+4])
53. n3 = - 100 \* int(img\_arr[i, j]) - 100 \* int(img\_arr[i, j+1]) + 100 \* int(img\_arr[i, j+3]) + 100 \* int(img\_arr[i, j+4]) \
54. - 100 \* int(img\_arr[i+1, j]) - 100 \* int(img\_arr[i+1, j+1]) + 100 \* int(img\_arr[i+1, j+3]) + 100 \* int(img\_arr[i+1, j+4]) \
55. - 100 \* int(img\_arr[i+2, j]) - 100 \* int(img\_arr[i+2, j+1]) + 100 \* int(img\_arr[i+2, j+3]) + 100 \* int(img\_arr[i+2, j+4]) \
56. - 100 \* int(img\_arr[i+3, j]) - 100 \* int(img\_arr[i+3, j+1]) + 100 \* int(img\_arr[i+3, j+3]) + 100 \* int(img\_arr[i+3, j+4]) \
57. - 100 \* int(img\_arr[i+4, j]) - 100 \* int(img\_arr[i+4, j+1]) + 100 \* int(img\_arr[i+4, j+3]) + 100 \* int(img\_arr[i+4, j+4])
58. n4 = - 100 \* int(img\_arr[i, j]) + 32 \* int(img\_arr[i, j+1]) + 100 \* int(img\_arr[i, j+2]) + 100 \* int(img\_arr[i, j+3]) + 100 \* int(img\_arr[i, j+4]) \
59. - 100 \* int(img\_arr[i+1, j]) - 78 \* int(img\_arr[i+1, j+1]) + 92 \* int(img\_arr[i+1, j+2]) + 100 \* int(img\_arr[i+1, j+3]) + 100 \* int(img\_arr[i+1, j+4]) \
60. - 100 \* int(img\_arr[i+2, j]) - 100 \* int(img\_arr[i+2, j+1]) + 100 \* int(img\_arr[i+2, j+3]) + 100 \* int(img\_arr[i+2, j+4]) \
61. - 100 \* int(img\_arr[i+3, j]) - 100 \* int(img\_arr[i+3, j+1]) - 92 \* int(img\_arr[i+3, j+2]) + 78 \* int(img\_arr[i+3, j+3]) + 100 \* int(img\_arr[i+3, j+4]) \
62. - 100 \* int(img\_arr[i+4, j]) - 100 \* int(img\_arr[i+4, j+1]) - 100 \* int(img\_arr[i+4, j+2]) - 32 \* int(img\_arr[i+4, j+3]) + 100 \* int(img\_arr[i+4, j+4])
63. n5 = 100 \* int(img\_arr[i, j]) + 100 \* int(img\_arr[i, j+1]) + 100 \* int(img\_arr[i, j+2]) + 100 \* int(img\_arr[i, j+3]) + 100 \* int(img\_arr[i, j+4]) \
64. - 32 \* int(img\_arr[i+1, j]) + 78 \* int(img\_arr[i+1, j+1]) + 100 \* int(img\_arr[i+1, j+2]) + 100 \* int(img\_arr[i+1, j+3]) + 100 \* int(img\_arr[i+1, j+4]) \
65. - 100 \* int(img\_arr[i+2, j]) - 92 \* int(img\_arr[i+2, j+1]) + 92 \* int(img\_arr[i+2, j+3]) + 100 \* int(img\_arr[i+2, j+4]) \
66. - 100 \* int(img\_arr[i+3, j]) - 100 \* int(img\_arr[i+3, j+1]) - 100 \* int(img\_arr[i+3, j+2]) - 78 \* int(img\_arr[i+3, j+3]) + 32 \* int(img\_arr[i+3, j+4]) \
67. - 100 \* int(img\_arr[i+4, j]) - 100 \* int(img\_arr[i+4, j+1]) - 100 \* int(img\_arr[i+4, j+2]) - 100 \* int(img\_arr[i+4, j+3]) - 100 \* int(img\_arr[i+4, j+4])
68. n\_list = np.array([n0, n1, n2, n3, n4, n5])
69. grad = np.max(n\_list)
70. res\_arr[i, j] = 255 if grad < threshold else 0
71. return res\_arr