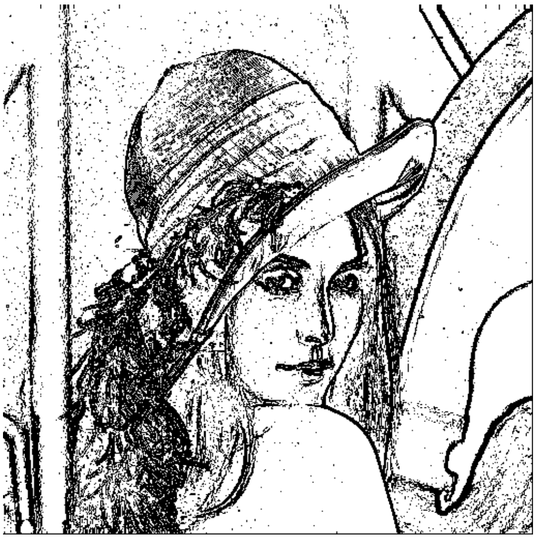
Homework 9

R13525009 羅筠笙

a. Robert's Operator: 12

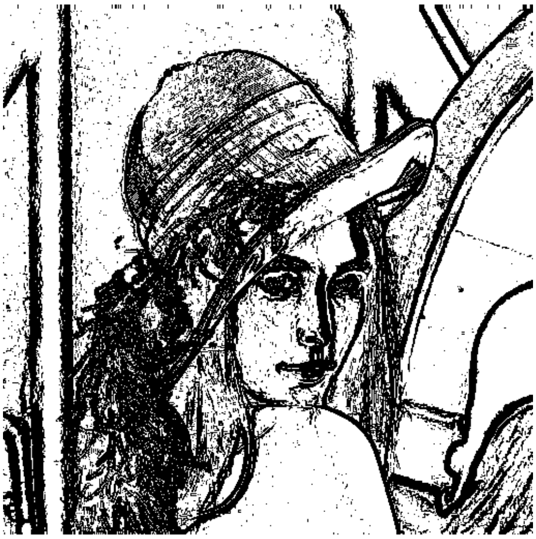
Description: Uses a 2x2 kernel to compute the gradient magnitude for edge detection by applying convolution to the input image.



```
def robert(img_arr, threshold):
    res_arr = np.zeros([img_size0, img_size1])
    for i in range(img_size0-1):
        for j in range(img_size1-1):
            '''
            r1 = [-1 0
                   0 1]
            r2 = [[0 -1
                   1   0]
            '''
            r1 =   - int(img_arr[i][j]) + int(img_arr[i+1][j+1])
            r2 = - int(img_arr[i][j+1]) + int(img_arr[i+1][j])

            grad = np.sqrt(r1**2 + r2**2)
            res_arr[i, j] = 255 if grad < threshold else 0

    return res_arr
```

b. **Prewitt's Edge Detector: 24**

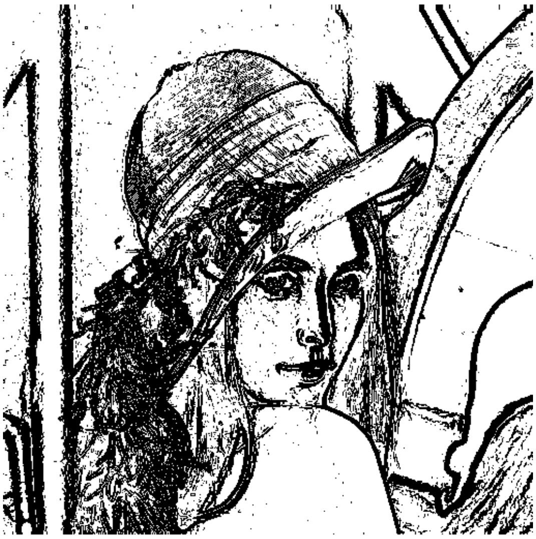
Description: Applies 3x3 kernels for horizontal and vertical gradients using convolution.



```
def prewitt_edge(img_arr, threshold):
    img_arr = expand_with_replicate(img_arr, 1)
    m, n = img_arr.shape
    res_arr = np.zeros([img_size0, img_size1])

    for i in range(img_size0):
        for j in range(img_size1):
            '''
            p1 = [-1 -1 -1
                  0 0 0
                  1 1 1]
            p2 = [-1 0 1
                  -1 0 1
                  -1 0 1]
            '''
            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])
            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])

            grad = np.sqrt(p1**2 + p2**2)
            res_arr[i, j] = 255 if grad < threshold else 0

    return res_arr
```

c. **Sobel's Edge Detector: 38**

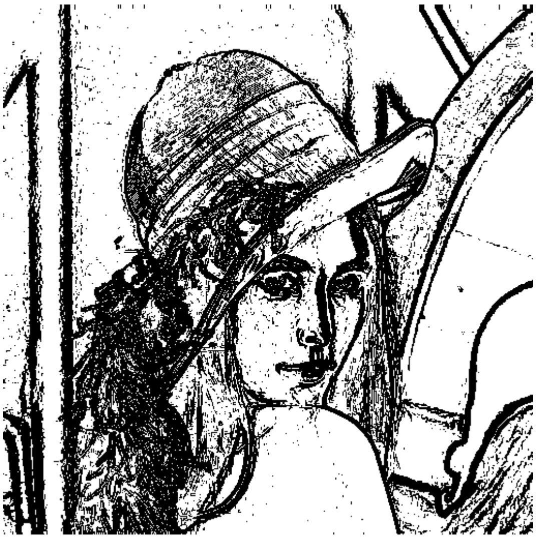
Description: Implements Sobel's operator by convolving the image with 3x3 kernels that emphasize the center pixels.



```
def sobel_edge(img_arr, threshold):
    img_arr = expand_with_replicate(img_arr, 1)
    m, n = img_arr.shape
    res_arr = np.zeros([img_size0, img_size1])

    for i in range(img_size0):
        for j in range(img_size1):
            '''
            s1 = [-1 -2 -1
                   0  0  0
                   1  2  1]
            s2 = [-1 0 1
                  -2 0 2
                  -1 0 1]
            '''
            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])
            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])

            grad = np.sqrt(s1**2 + s2**2)
            res_arr[i, j] = 255 if grad < threshold else 0

    return res_arr
```

d. **Frei and Chen's Gradient Operator: 30**

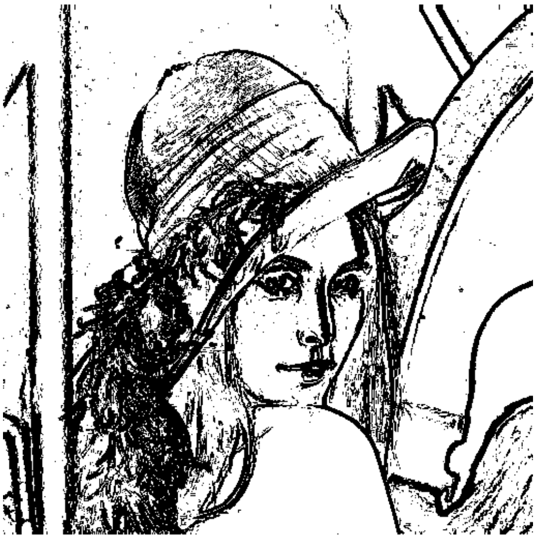
Description: Uses modified Sobel-like 3x3 kernels, applying specific weights to capture diagonal and straight-edge gradients



```
def frel_and_chen_grad(img_arr, threshold):
    img_arr = expand_with_replicate(img_arr, 1)
    m, n = img_arr.shape
    res_arr = np.zeros([img_size0, img_size1])

    for i in range(img_size0):
        for j in range(img_size1):
            '''
            f1 = [-1 sqrt(-2)    -1
                        0        0    0
                        1  sqrt(2)    1]
            f2 = [-1           0    1
                 sqrt(-)2     0 sqrt(2)
                 -1           0    1]
            '''
            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])
            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])

            grad = np.sqrt(f1**2 + f2**2)
            res_arr[i, j] = 255 if grad < threshold else 0

    return res_arr
```

e. Kirsch's Compass Operator: 135

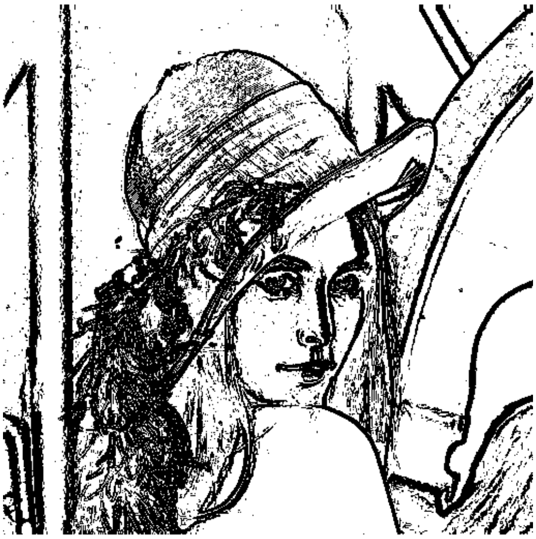
Description: Applies eight 3x3 directional masks (compass masks) to calculate gradient magnitudes in all compass directions.



```
def kirsch_comass(img_arr, threshold):
    img_arr = expand_with_replicate(img_arr, 1)
    m, n = img_arr.shape
    res_arr = np.zeros([img_size0, img_size1])

    for i in range(img_size0):
        for j in range(img_size1):
            '''
            k0 = [-3    -3    5
                  -3     0    5
                  -3    -3    5]
            k1 = [-3    5     5
                  -3    0     5
                  -3    -3    -3]
            k2 = [5     5     5
                 -3     0     -3
                 -3     -3    -3]
            k3 = [5     5     -3
                  5     0     -3
                 -3     -3    -3]
            k4 = [5     -3    -3
                  5     0     -3
                  5     -3    -3]
            k5 = [-3    -3    -3
                   5    0     -3
                   5    5     -3]
            k6 = [-3    -3    -3
                  -3    0     -3
                   5    5     5]
            k7 = [-3    -3    -3
                  -3    0     5
                  -3    5     5]
            '''
            k0 = - 3 * int(img_arr[i, j]) - 3 * int(img_arr[i, j+1]) + 5 * int(img_arr[i, j+2]) \
                - 3 * int(img_arr[i+1, j]) + 5 * int(img_arr[i+1, j+2]) \
                - 3 * int(img_arr[i+2, j]) - 3 * int(img_arr[i+2, j+1]) + 5 * int(img_arr[i+2, j+2])
            k1 = - 3 * int(img_arr[i, j]) + 5 * int(img_arr[i, j+1]) + 5 * int(img_arr[i, j+2]) \
                - 3 * int(img_arr[i+1, j]) + 5 * int(img_arr[i+1, j+2]) \
                - 3 * int(img_arr[i+2, j]) - 3 * int(img_arr[i+2, j+1]) - 3 * int(img_arr[i+2, j+2])
            k2 = 5 * int(img_arr[i, j]) + 5 * int(img_arr[i, j+1]) + 5 * int(img_arr[i, j+2]) \
                - 3 * int(img_arr[i+1, j]) - 3 * int(img_arr[i+1, j+2]) \
                - 3 * int(img_arr[i+2, j]) - 3 * int(img_arr[i+2, j+1]) - 3 * int(img_arr[i+2, j+2])
            k3 = 5 * int(img_arr[i, j]) + 5 * int(img_arr[i, j+1]) - 3 * int(img_arr[i, j+2]) \
                + 5 * int(img_arr[i+1, j]) - 3 * int(img_arr[i+1, j+2]) \
                - 3 * int(img_arr[i+2, j]) - 3 * int(img_arr[i+2, j+1]) - 3 * int(img_arr[i+2, j+2])
            k4 = 5 * int(img_arr[i, j]) - 3 * int(img_arr[i, j+1]) - 3 * int(img_arr[i, j+2]) \
                + 5 * int(img_arr[i+1, j]) - 3 * int(img_arr[i+1, j+2]) \
                + 5 * int(img_arr[i+2, j]) - 3 * int(img_arr[i+2, j+1]) - 3 * int(img_arr[i+2, j+2])
            k5 = - 3 * int(img_arr[i, j]) - 3 * int(img_arr[i, j+1]) - 3 * int(img_arr[i, j+2]) \
                + 5 * int(img_arr[i+1, j]) - 3 * int(img_arr[i+1, j+2]) \
                + 5 * int(img_arr[i+2, j]) + 5 * int(img_arr[i+2, j+1]) - 3 * int(img_arr[i+2, j+2])
            k6 = - 3 * int(img_arr[i, j]) - 3 * int(img_arr[i, j+1]) - 3 * int(img_arr[i, j+2]) \
                - 3 * int(img_arr[i+1, j]) - 3 * int(img_arr[i+1, j+2]) \
                + 5 * int(img_arr[i+2, j]) + 5 * int(img_arr[i+2, j+1]) + 5 * int(img_arr[i+2, j+2])
            k7 = - 3 * int(img_arr[i, j]) - 3 * int(img_arr[i, j+1]) - 3 * int(img_arr[i, j+2]) \
                - 3 * int(img_arr[i+1, j]) + 5 * int(img_arr[i+1, j+2]) \
                - 3 * int(img_arr[i+2, j]) + 5 * int(img_arr[i+2, j+1]) + 5 * int(img_arr[i+2, j+2])

            k_list = np.array([k0, k1, k2, k3, k4, k5, k6, k7])
            grad = np.max(k_list)

            res_arr[i, j] = 255 if grad < threshold else 0

    return res_arr
```

f. Robinson's Compass Operator: 43

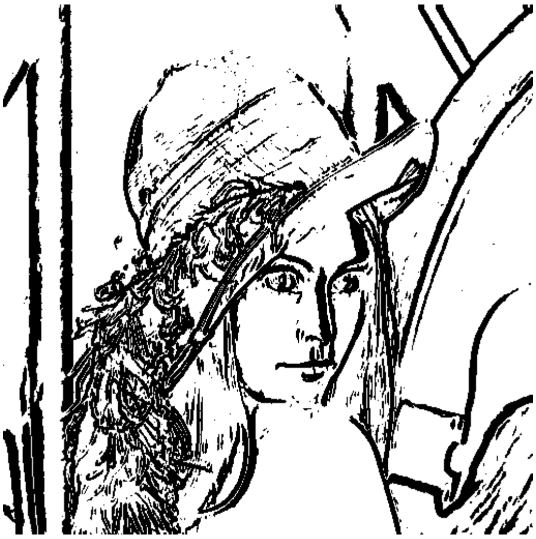
Description: Implements eight directional 3x3 masks similar to Kirsch but uses simpler calculations.



```
def robinson_compass(img_arr, threshold):
    img_arr = expand_with_replicate(img_arr, 1)
    m, n = img_arr.shape
    res_arr = np.zeros([img_size0, img_size1])

    for i in range(img_size0):
        for j in range(img_size1):
            '''
            r0 = [-1     0      1
                  -2     0      2
                  -1     0      1]
            r1 = [0      1      2
                 -1      0      1
                 -2     -1      0]
            r2 = [1      2      1
                  0      0      0
                 -1     -2     -1]
            r3 = [2      1      0
                  1      0     -1
                  0     -1     -2]
            r4 = [1      0     -1
                  2      0     -2
                  1       0      -1]
            r5 = [0      -1      -2
                  1       0      -1
                  2       1       0]
            r6 = [-1     -2      -1
                   0      0       0
                   1      2       1]
            r7 = [-2     -1       0
                  -1      0       1
                   0      1       2]
            '''
            r0 = - int(img_arr[i, j]) + int(img_arr[i, j+2]) \
                - 2 * int(img_arr[i+1, j]) + 2 * int(img_arr[i+1, j+2]) \
                - int(img_arr[i+2, j]) + int(img_arr[i+2, j+2])
            r1 = int(img_arr[i, j+1]) + 2 * int(img_arr[i, j+2]) \
                - int(img_arr[i+1, j]) + int(img_arr[i+1, j+2]) \
                - 2 * int(img_arr[i+2, j]) - int(img_arr[i+2, j+1])
            r2 = 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])
            r3 = 2 * int(img_arr[i, j]) + int(img_arr[i, j+1]) \
                + int(img_arr[i+1, j]) - int(img_arr[i+1, j+2]) \
                - int(img_arr[i+2, j+1]) - 2 * int(img_arr[i+2, j+2])
            r4 = int(img_arr[i, j]) - int(img_arr[i, j+2]) \
                + 2 * int(img_arr[i+1, j]) - 2 * int(img_arr[i+1, j+2]) \
                + int(img_arr[i+2, j]) - int(img_arr[i+2, j+2])
            r5 = - int(img_arr[i, j+1]) - 2 * int(img_arr[i, j+2]) \
                + int(img_arr[i+1, j]) - int(img_arr[i+1, j+2]) \
                + 2 * int(img_arr[i+2, j]) + int(img_arr[i+2, j+1])
            r6 = - 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])
            r7 = - 2 * int(img_arr[i, j]) - int(img_arr[i, j+1]) \
                - int(img_arr[i+1, j]) + int(img_arr[i+1, j+2]) \
                + int(img_arr[i+2, j+1]) + 2 * int(img_arr[i+2, j+2])

            r_list = np.array([r0, r1, r2, r3, r4, r5, r6, r7])
            grad = np.max(r_list)

            res_arr[i, j] = 255 if grad < threshold else 0

    return res_arr
```

g. **Nevatia-Babu 5x5 Operator: 12500**

Description: Applies multiple 5x5 kernels to analyze high-resolution images, detecting fine and directional edges.



```
def nevatia_babu_5x5(img_arr, threshold):
    img_arr = expand_with_replicate(img_arr, 2)
    m, n = img_arr.shape
    res_arr = np.zeros([img_size0, img_size1])

    for i in range(img_size0):
        for j in range(img_size1):
            '''
            n0 = [100    100    100    100    100
                   100    100    100    100    100
                     0      0      0      0      0
                  -100   -100   -100   -100   -100
                  -100   -100   -100   -100   -100]
            n1 = [100    100    100    100    100
                   100    100    100     78    -32
                   100     92      0    -92   -100
                    32    -78   -100   -100   -100
```

```
             -100   -100   -100   -100   -100]
        n2 = [100    100    100     32   -100
              100    100     92    -78   -100
              100    100      0   -100   -100
              100     78    -92   -100   -100
              100    -32   -100   -100   -100]
        n3 = [-100   -100      0    100    100
             -100   -100      0    100    100
             -100   -100      0    100    100
             -100   -100      0    100    100
             -100   -100      0    100    100]
        n4 = [-100     32    100    100    100
             -100    -78     92    100    100
             -100   -100      0    100    100
             -100   -100    -92     78    100
             -100   -100   -100    -32    100]
        n5 = [100    100    100    100    100
              -32     78    100    100    100
             -100    -92      0     92    100
             -100   -100   -100    -78     32
             -100   -100   -100   -100   -100]
        '''
        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]) \
            + 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]) \
            - 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]) \
            - 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])
        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]) \
            + 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]) \
                + 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]) \
                + 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]) \
                - 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])
            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]) \
                + 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]) \
                + 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]) \
                + 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]) \
                + 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])
            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]) \
                - 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]) \
                - 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]) \
                - 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]) \
                - 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])
            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]) \
                - 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]) \
                - 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]) \
                - 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]) \
                - 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])
            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]) \
                - 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]) \
                - 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]) \
                - 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]) \
                - 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])

            n_list = np.array([n0, n1, n2, n3, n4, n5])
            grad = np.max(n_list)

            res_arr[i, j] = 255 if grad < threshold else 0

    return res_arr
```