Homework 10

R13525009 羅筠笙

a. Laplace Mask1 (0, 1, 0, 1, -4, 1, 0, 1, 0): 15

Description: Applies a simple Laplace operator for edge detection and further filters the edge points based on neighborhood pixel characteristics, producing the final edge-detected image.



```
def laplace1(img arr, threshold):
        mask = np.array([[0, 1, 0],
3.
                         [1, -4, 1],
                         [0, 1, 0]])
       # Expand the image
       img_arr = expand_with_replicate(img_arr, 1)
7.
8.
       # Using convolution
9.
       res1 = my conv(img arr, mask, threshold)
10.
11.
       res1 = expand_with_replicate(res1, 1)
12.
       res2 = np.ones((img_size0, img_size1))
13.
14.
```

```
# Neighbor checking
15.
       for i in range(img size0):
16.
            for j in range(img size1):
17.
                if res1[i + 1, j + 1] == 1:
18.
                    tmp = False
19.
                    for k in range(3):
20.
                         for I in range(3):
21.
                             if res1[i + k, j + l] == -1:
22.
                                 tmp = True
23.
                                 break
24.
                         if tmp:
25.
                             break
26.
                    if tmp:
27.
                         res2[i, j] = 0
28.
29.
       return res2 * 255
30.
```

b. Laplace Mask2 (1, 1, 1, 1, -8, 1, 1, 1, 1): 15

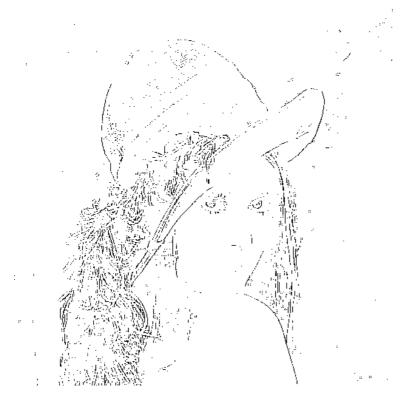
Description: Applies another Laplace operator for edge detection. Similar to question a, it filters edge points based on neighborhood characteristics, resulting in smoother edge outcomes.



```
def laplace2(img arr, threshold):
       mask = np.array([[1, 1, 1],
                         [1, -8, 1],
3.
                         [1, 1, 1]], dtype=np.float64)
       mask /= 3
6.
       # Expand input for padding using numpy
7.
       img arr = expand with replicate(img arr, 1)
8.
       # Perform convolution using numpy array
10.
       res1 = my conv(img arr, mask, threshold)
11.
12.
13.
       # Expand result for neighbor checking using numpy
       res1 = expand with replicate(res1, 1)
14.
       res2 = np.ones((img size0, img size1)) # Initialize with ones
15.
16.
       # Neighbor checking
17.
       for i in range(img size0):
           for j in range(img size1):
19.
20.
                if res1[i + 1, j + 1] == 1:
                    tmp = False
21.
                    for k in range(3):
22.
                        for I in range(3):
23.
                            if res1[i + k, j + l] == -1:
24.
                                tmp = True
25.
                                break
26.
                        if tmp:
27.
                            break
28.
                    if tmp:
29.
                        res2[i, j] = 0
30.
31.
       return res2 * 255
32.
```

c. Minimum variance Laplacian: 20

Description: Applies the minimum variance Laplace operator to detect edges, focusing on accurately representing edge features while minimizing noise, producing more stable detection results.



```
def minimum variance laplacian(img arr, threshold):
       mask = np.array([[2, -1, 2],
                        [-1, -4, -1],
3.
                        [2, -1, 2]], dtype=np.float64)
       mask /= 3
       # Expand input for padding using numpy
       img arr = expand with replicate(img arr, 1)
       # Perform convolution using numpy array
10.
       res1 = my_conv(img_arr, mask, threshold)
11.
12.
       # Expand result for neighbor checking using numpy
13.
       res1 = expand with replicate(res1, 1)
14.
       res2 = np.ones((img size0, img size1)) # Initialize with ones
15.
       # Neighbor checking
17.
       for i in range(img size0):
18.
           for j in range(img size1):
19.
               if res1[i + 1, j + 1] == 1:
20.
                   tmp = False
21.
                   for k in range(3):
22.
```

```
for I in range(3):
23.
                              if res1[i + k, j + l] == -1:
24.
                                  tmp = True
25.
                                  break
26.
                         if tmp:
27.
                              break
28.
                     if tmp:
29.
                         res2[i, j] = 0
30.
31.
        return res2 * 255
32.
```

d. Laplace of Gaussian: 3000

Description: Combines Gaussian smoothing with the Laplace operator. It removes high-frequency noise before edge detection, making it particularly suitable for noisy images.



```
1. def laplace_gauss(img_arr, threshold):
2. mask = np.array([[0, 0, 0, -1, -1, -2, -1, -1, 0, 0, 0],
3. [0, 0, -2, -4, -8, -9, -8, -4, -2, 0, 0],
4. [0, -2, -7, -15, -22, -23, -22, -15, -7, -2, 0],
5. [-1, -4, -15, -24, -14, -1, -14, -24, -15, -4, -1],
6. [-1, -8, -22, -14, 52, 103, 52, -14, -22, -8, -1],
```

```
7.
                          [-2, -9, -23, -1, 103, 178, 103, -1, -23, -9, -2],
                          [-1, -8, -22, -14, 52, 103, 52, -14, -22, -8, -1],
8.
                          [-1, -4, -15, -24, -14, -1, -14, -24, -15, -4, -1],
9.
                          [0, -2, -7, -15, -22, -23, -22, -15, -7, -2, 0],
10.
                          [0, 0, -2, -4, -8, -9, -8, -4, -2, 0, 0],
11.
                          [0, 0, 0, -1, -1, -2, -1, -1, 0, 0, 0]]
12.
13.
        # Expand input for padding using numpy
14.
       img arr = expand with replicate(img arr, 5)
15.
16.
       # Perform convolution using numpy array
17.
        res1 = my conv(img arr, mask, threshold)
18.
19.
       # Expand result for neighbor checking using numpy
20.
       res1 = expand with replicate(res1, 5)
21.
        res2 = np.ones((img size0, img size1)) # Initialize with ones
22.
23.
        # Neighbor checking
24.
        for i in range(img size0):
25.
            for j in range(img size1):
26.
                if res1[i + 1, j + 1] == 1:
27.
                    tmp = False
28.
                    for k in range(3):
29.
30.
                         for I in range(3):
                             if res1[i + k, j + l] == -1:
31.
                                 tmp = True
32.
                                 break
33.
                         if tmp:
34.
                             break
35.
                    if tmp:
36.
                         res2[i, j] = 0
37.
38.
        return res2 * 255
39.
```

e. Difference of Gaussian: 1

Description: Performs edge detection using the Difference of Gaussian method. It computes edges by applying two Gaussian filters with different standard deviations, emphasizing fine details and edge features in the image.



```
1. def difference gauss(img arr, threshold):
        mask = np.array([[-1, -3, -4, -6, -7, -8, -7, -6, -4, -3, -1],
                          [-3, -5, -8, -11, -13, -13, -13, -11, -8, -5, -3]
3.
                          [-4, -8, -12, -16, -17, -17, -17, -16, -12, -8, -4],
4.
                          [-6, -11, -16, -16, 0, 15, 0, -16, -16, -11, -6],
                          [-7, -13, -17, 0, 85, 160, 85, 0, -17, -13, -7],
6.
                          [-8, -13, -17, 15, 160, 283, 160, 15, -17, -13, -8],
7.
                          [-7, -13, -17, 0, 85, 160, 85, 0, -17, -13, -7],
8.
                          [-6, -11, -16, -16, 0, 15, 0, -16, -16, -11, -6],
9.
                          [-4, -8, -12, -16, -17, -17, -17, -16, -12, -8, -4],
10.
                          [-3, -5, -8, -11, -13, -13, -13, -11, -8, -5, -3],
11.
                          [-1, -3, -4, -6, -7, -8, -7, -6, -4, -3, -1]])
12.
13.
       # Expand input for padding using numpy
14.
       img arr = expand with replicate(img arr, 5)
15.
16.
       # Perform convolution using numpy array
17.
        res1 = my conv(img arr, mask, threshold)
18.
19.
       # Expand result for neighbor checking using numpy
20.
       res1 = expand with replicate(res1, 5)
21.
        res2 = np.ones((img size0, img size1)) # Initialize with ones
```

```
23.
       # Neighbor checking
24.
       for i in range(img_size0):
25.
           for j in range(img_size1):
26.
                if res1[i + 1, j + 1] == 1:
27.
                    tmp = False
28.
                    for k in range(3):
29.
                        for I in range(3):
30.
                            if res1[i + k, j + l] == -1:
31.
                                tmp = True
32.
                                break
33.
                        if tmp:
34.
                            break
35.
                    if tmp:
36.
                        res2[i, j] = 0
37.
38.
       return res2 * 255
39.
```