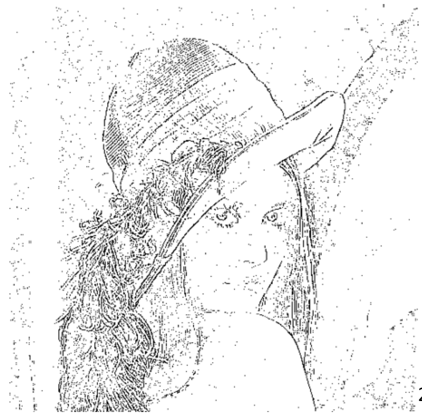
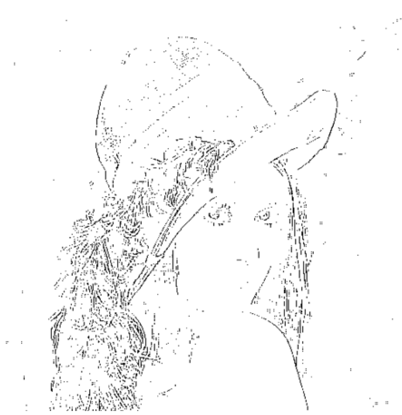
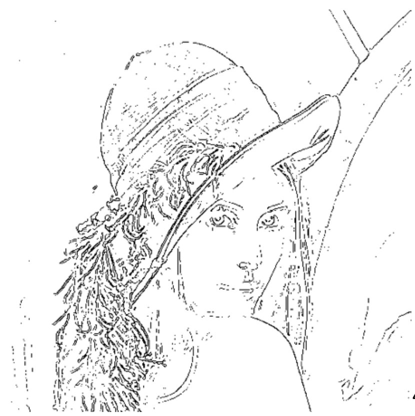
R13922136 周洛君

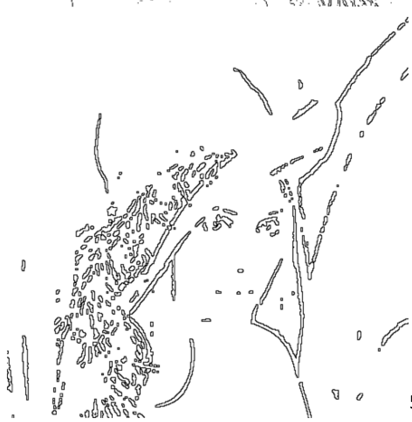
Computer Vision HW 10: Report


[1]


[2]


[3]


[4]


[5]

[1] (a) Laplacian mask 1, threshold = 15
[2] (b) Laplacian mask 2, threshold = 15
[3] (c) Minimum variance Laplacian, threshold = 20
[4] (d) Laplace of Gaussian, threshold = 3000
[5] (e) Difference of Gaussian, threshold = 1

Code Structure

1. Convolution
   - Convolution(): convolve a particular neighborhood
   - Convolution_whole_image(): iteratively apply the convolution function to each neighborhood within the image
2. Convolution result → Laplacian output
   - Convolution_result_to_Laplacian_output()
3. Laplacian result → zero crossing
   - Laplacian_output_to_result_image
4. Function that each corresponds (a) ~ (e)

Code details

For the first part, first we have the same function as used in the previous homework, but a modification is added, which is adding another Convolution_whole_image() function, so that we can call this function and get the convolution result directly, without explicitly looping though each neighborhood in the functions (a) ~ (e).

Convolution applied to the whole image

```python
def convolution_whole_image(img, convolution_mask):
    convolution_result = np.zeros((img.shape[0] - convolution_mask.shape[0] + 1, img.shape[1] - convolution_mask.shape[1] + 1))
    for i in range(convolution_result.shape[0]):
        for j in range(convolution_result.shape[1]):
            convolution_result[i, j] = convolution(img[i:i+convolution_mask.shape[0], j:j+convolution_mask.shape[1]], convolution_mask)
    return convolution_result
```

[60] ✓ 0.0s

In this function, we initialize a zero-filled array, with size that ensures the last row / column of the kernel can be placed within the original image. For example, if we have a 2x2 kernel, we cannot put the top row on the last row of our image.

Next, the convolution result of each pixel is calculated by calling the convolution function.

For the second part, we need to transform this result (the whole image after doing convolution) into Laplacian, which means the function below will output an array full of values of t:

Convolution result to Laplacian output

This function converts the convolution result into a Laplacian output array. The definition is as in the ppt, for example:

Input pixel gradient magnitude >= threshold (15) → Laplacian output pixel t = 1
Input pixel gradient magnitude <= -threshold (15) → Laplacian output pixel t = -1
Else → Laplacian output pixel t = 0

```
def convolution_result_to_Laplacian_output(convolution_result, threshold):
    laplacian_output = np.zeros(convolution_result.shape)
    for i in range(convolution_result.shape[0]):
        for j in range(convolution_result.shape[1]):
            if convolution_result[i,j] > 0 and convolution_result[i,j] >= threshold:
                laplacian_output[i,j] = 1
            elif convolution_result[i,j] < 0 and convolution_result[i,j] <= -threshold:
                laplacian_output[i,j] = -1
            else:
                laplacian_output[i,j] = 0
    return laplacian_output
```

[61] ✓ 0.0s

This function is quite simple, we just follow the definition on the ppt, to set the values of t to 1, -1, 0.

The third part is to do zero-crossing, we also follow the definition on the ppt to transform values of t into intensity values 0 / 255:

Apply zero-crossing on Laplacian output

If the Laplacian output pixel is:

1. $t = 1$, and one of its 8 neighbors is -1, then the result image pixel is 0.
2. $t = -1$ or $t = 0$, then the result image pixel is 255.

The code is as follows:

```
def Laplacian_output_to_result_image(Laplacian_output):
    result_image = np.full(Laplacian_output.shape, 255)
    for i in range(Laplacian_output.shape[0]):
        for j in range(Laplacian_output.shape[1]):
            if Laplacian_output[i,j] == 1:
                has_negative_neighbor = False
                for di in range(-1, 2):
                    for dj in range(-1, 2):
                        ni, nj = i+di, j+dj
                        if (0 <= ni < Laplacian_output.shape[0] and
                            0 <= nj < Laplacian_output.shape[1] and
                            Laplacian_output[ni, nj] == -1):
                            has_negative_neighbor = True
                            break
                    if has_negative_neighbor:
                        break
                if has_negative_neighbor:
                    result_image[i,j] = 0
    return result_image

```

✓ 0.0s

A really detailed explanation for each line is in the associated markdown cell in the source code file:

Implementation details:

`line 2`: First we initialize the result image to be a 255-filled array, so that we only change the pixel values when the above first condition is satisfied.

`line 3,4`: Then we loop though each pixel

`line 5`: if the Laplacian output pixel is 1, then we check its 8 neighbors to see if the above first condition is satisfied.

`line 6`: a flag `has_negative_neighbor` is initialized to be `False`, it is modified when a negative neighbor is found.

`line 7-9`: we loop through the 8 neighbors, as each of the neighbor's index is calculated by adding `di` to the current row, and `dj` to the current column.

`line 10-12`: if the resulting index of a neighbor is within the image, and its value is -1

`line 13`: then we set `has_negative_neighbor` to be `True`, and break the loop, so we skp rest of the columns in the current row

`line 15,16`: we also don't need to check the rest of the rows if a negative neighbor is found.

`line 17,18`: finally, set the pixel to be 0 if a negative neighbor is found.

For the last part, which is implementing the functions corresponding to functions (a) ~ (e), they all have similar structure, take (c) for example:

(c) Minimum variance Laplacian: 20

$$\frac{1}{3}\begin{bmatrix} 2 & -1 & 2 \\ -1 & -4 & -1 \\ 2 & -1 & 2 \end{bmatrix}$$

+ Code + Markdown

```python
def Minimum_Variance_Laplacian(img, threshold):
    kernel = np.array([
                [2., -1, 2],
                [-1, -4, -1],
                [2, -1, 2]
            ]) / 3
    convolution_result = convolution_whole_image(img, kernel)
    Laplacian_output = convolution_result_to_Laplacian_output(convolution_result, threshold)
    return Laplacian_output_to_result_image(Laplacian_output)

```

[67] ✓ 0.0s

The image in the markdown cell is the kernel we should use as in ppt, in the function, we first define this kernel, then what we need to do is to follow the steps of our previous 3 parts:

Get convolution result → get laplacian result → zero crossing result (the resulting image shown in the report)

```python
Minimum_Variance_Laplacian_img = Minimum_Variance_Laplacian(original_lena, 20)
cv2.imwrite('result_images/Minimum_Variance_Laplacian_20.bmp', Minimum_Variance_Laplacian_img)
```

✓ 1.2s

Finally, call the function with required threshold, and save the result by cv2.imwrite().