HW 10

> *b05902019 資工三 蔡青邑*

`Using "lena.bmp" as input image.`

Python Packages I used

- `skimage.io`: for basic image i/o.
- `numpy`: for the convience of array manipulation.
- `math`: originally for calulating Gaussian distribution,
  turn out to be needless.
- `tqdm`: for visualizing the progress of the running code

Some Other Functions I Build

- **blank_image(height, width, value)**: return a blank image of the given height and width with all the values in it being initialize as the given value.

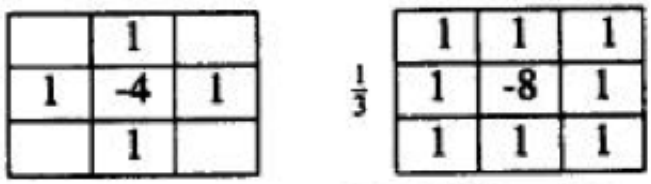
Result Image

laplace_type1.png	laplace_type2.png
	
minimum_variance_laplacian.png	laplacian_of_gaussian.png
	
difference_of_gaussian.png	
	

Laplacian

> *type1 threshold: 15, type 2 threshold 15*

Starting with `blank_image(512, 512, 255)`, I set a image with all values being 255. Using the below given masks to get the product-value. If the value > threshold, change the value of the pixel to 0. (I omitted the border of pixels of the image; as told by TA, I abandoned my original code of checking whether there's some value < -threshold near the changed pixel).




```
def laplace_type1(img, threshold):
    return_img = blank_image(height, width, 255)
    temp = lena.copy()
    gradient = blank_image(height, width, 255)
    for i in tqdm(range(1, height - 1)):
        for j in range(1, width - 1):
            gradient[i][j] = temp[i + 1][j] * 1 + temp[i][j - 1] * 1
+ temp[i][j + 1] * 1 + temp[i - 1][j] * 1 + temp[i][j] * (-4)
    for i in tqdm(range(1, height - 1)):
        for j in range(1, width - 1):
            if(gradient[i][j] > threshold):
                return_img[i][j] = 0
    return return_img


def laplace_type2(img, threshold):
    return_img = blank_image(height, width, 255)
    temp = lena.copy()
    gradient = blank_image(height, width, 0)
    io.imshow(gradient)
    for i in tqdm(range(1, height - 1)):
        for j in range(1, width - 1):
            gradient[i][j] = (temp[i + 1][j] * 1 + temp[i][j - 1] * 1
+ temp[i][j + 1] * 1 + temp[i - 1][j] * 1
                              + temp[i + 1][j + 1] * 1 + temp[i + 1]
[j - 1] * 1+ temp[i - 1][j + 1] * 1 + temp[i - 1][j - 1] * 1)
            gradient[i][j] -= temp[i][j] * 8
            gradient[i][j] = gradient[i][j] * 1/3
    for i in tqdm(range(1, height - 1)):
        for j in range(1, width - 1):
```

```
            if(gradient[i][j] > threshold):
                return_img[i][j] = 0
    return return_img
```

Minimum Variance Laplacian

> *Threshold: 19*

Starting with `blank_image(512, 512, 255)`, I set a image with all values being 255. Using the below given masks to get the product-value. If the value > threshold, change the value of the pixel to 0. (I omitted the border of pixels of the image; as told by TA, I abandoned my original code of checking whether there's some value < -threshold near the changed pixel).

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

```
def minimum_variance_laplacian(img, threshold):
    return_img = blank_image(height, width, 255)
    temp = lena.copy()
    gradient = blank_image(height, width, 255)
    for i in tqdm(range(1, height - 1)):
        for j in range(1, width - 1):
            gradient[i][j] = (temp[i + 1][j] * (-1) + temp[i][j - 1]
* (-1) + temp[i][j + 1] * (-1) + temp[i - 1][j] * (-1)
                              + temp[i + 1][j + 1] * 2 + temp[i + 1]
[j - 1] * 2 + temp[i - 1][j + 1] * 2 + temp[i - 1][j - 1] * 2)
            gradient[i][j] -= temp[i][j] * 4
            gradient[i][j] = gradient[i][j] * 1/3
    for i in tqdm(range(1, height - 1)):
        for j in range(1, width - 1):
            if(gradient[i][j] > threshold):
                return_img[i][j] = 0
    return return_img
```

Laplacian of Gaussian

> *Threshold: 8000*

Starting with `blank_image(512, 512, 255)`, I set a image with all values being 255. Using the below given mask to get the product-value. If the value > threshold, change the value of the pixel to 0. (I omitted the border of pixels of the image; as told by TA, I abandoned my original code of checking whether there's some value < -threshold near the changed pixel).

0	0	0	-1	-1	-2	-1	-1	0	0	0
0	0	-2	-4	-8	-9	-8	-4	-2	0	0
0	-2	-7	-15	-22	-23	-22	-15	-7	-2	0
-1	-4	-15	-24	-14	-1	-14	-24	-15	-4	-1
-1	-8	-22	-14	52	103	52	-14	-22	-8	-1
-2	-9	-23	-1	103	178	103	-1	-23	-9	-2
-1	-8	-22	-14	52	103	52	-14	-22	-8	-1
-1	-4	-15	-24	-14	-1	-14	-24	-15	-4	-1
0	-2	-7	-15	-22	-23	-22	-15	-7	-2	0
0	0	-2	-4	-8	-9	-8	-4	-2	0	0
0	0	0	-1	-1	-2	-1	-1	0	0	0

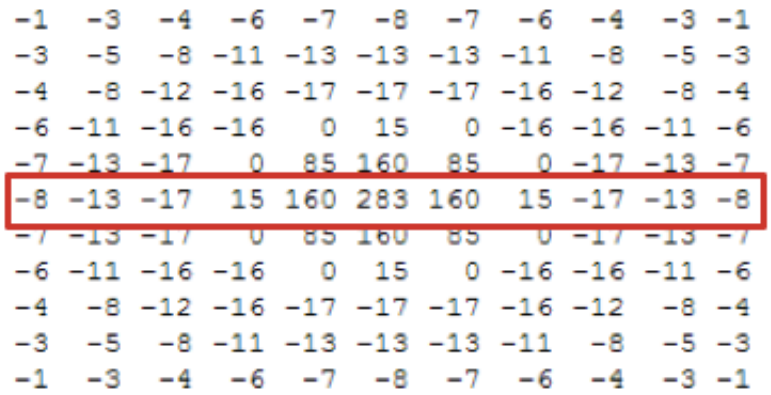
```
def laplacian_of_gaussian(img, threshold):
    return_img = blank_image(height, width, 255)
    temp = lena.copy()
    gradient = blank_image(height, width, 0)
    kernel = [
        [0, 0, 0, -1, -1, -2, -1, -1, 0, 0, 0],
        [0, 0, -2, -4, -8, -9, -8, -4, -2, 0, 0],
        [0, -2, -7, -15, -22, -23, -22, -15, -7, -2, 0],
        [-1, -4, -15, -24, -14, -1, -14, -24, -15, -4, -1],
        [-1, -8, -22, -14, 52, 103, 52, -14, -22, -8, -1],
        [-2, -9, -23, -1, 103, 178, 103, -1, -23, -9, -2],
        [-1, -8, -22, -14, 52, 103, 52, -14, -22, -8, -1],
        [-1, -4, -15, -24, -14, -1, -14, -24, -15, -4, -1],
        [0, -2, -7, -15, -22, -23, -22, -15, -7, -2, 0],
        [0, 0, -2, -4, -8, -9, -8, -4, -2, 0, 0],
        [0, 0, 0, -1, -1, -2, -1, -1, 0, 0, 0],
    ]
    for i in tqdm(range(5, height - 5)):
        for j in range(5, width - 5):
            for x in range(-5, 6):
                for y in range(-5, 6):
                    gradient[i][j] += temp[i + x][j + y] * kernel[5 +
x][5 + y] * 1
    for i in tqdm(range(5, height - 5)):
```

```
            for j in range(5, width - 5):
                if(gradient[i][j] > threshold):
                    return_img[i][j] = 0
        return return_img
```

Difference of Gaussian

> *Threshold: 10000*

Starting with `blank_image(512, 512, 255)`, I set a image with all values being 255. Using the below given mask to get the product-value. If the value > threshold, change the value of the pixel to 0. (I omitted the border of pixels of the image; as told by TA, I abandoned my original code of checking whether there's some value < -threshold near the changed pixel).




```
def difference_of_gaussian(img, threshold):
    return_img = blank_image(height, width, 0)
    temp = lena.copy()
    gradient = blank_image(height, width, 0)
    kernel = [
        [-1, -3, -4, -6, -7, -8, -7, -6, -4, -3, -1],
        [-3, -5, -8, -11, -13, -13, -13, -11, -8, -5, -3],
        [-4, -8, -12, -16, -17, -17, -17, -16, -12, -8, -4],
        [-6, -11, -16, -16, 0, 15, 0, -16, -16, -11, -6],
        [-7, -13, -17, 0, 85, 160, 85, 0, -17, -13, -7],
        [-8, -13, -17, 15, 160, 283, 160, 15, -17, -13, -8],
        [-7, -13, -17, 0, 85, 160, 85, 0, -17, -13, -7],
        [-6, -11, -16, -16, 0, 15, 0, -16, -16, -11, -6],
        [-4, -8, -12, -16, -17, -17, -17, -16, -12, -8, -4],
        [-3, -5, -8, -11, -13, -13, -13, -11, -8, -5, -3],
        [-1, -3, -4, -6, -7, -8, -7, -6, -4, -3, -1],
    ]
    for i in tqdm(range(5, height - 5)):
        for j in range(5, width - 5):
            for x in range(-5, 6):
                for y in range(-5, 6):
                    gradient[i][j] += temp[i + x][j + y] * kernel[5 +
x][5 + y] * 1
    for i in tqdm(range(5, height - 5)):
        for j in range(5, width - 5):
```

```
            if(gradient[i][j] > threshold):
                return_img[i][j] = 255
    # get the border white
    for i in range(0, height):
        for j in range(height - 5, height):
            return_img[i][j] = 255
            return_img[j][i] = 255
    for i in range(0, height):
        for j in range(0, 5):
            return_img[i][j] = 255
            return_img[j][i] = 255
    return return_img
```