HW 9

> *b05902019 資工三 蔡青邑*

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

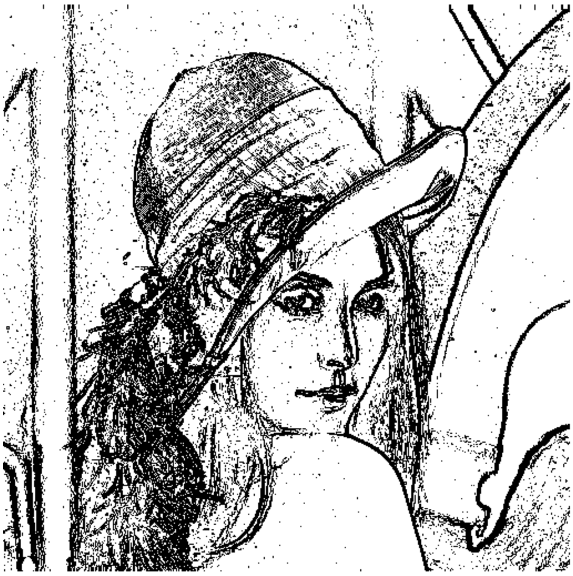
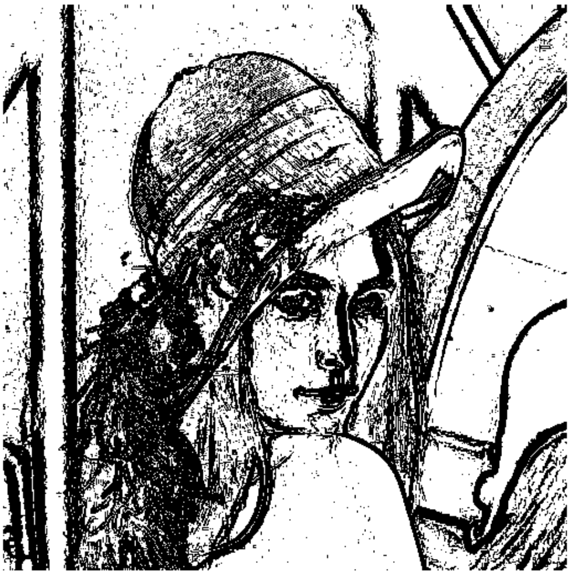
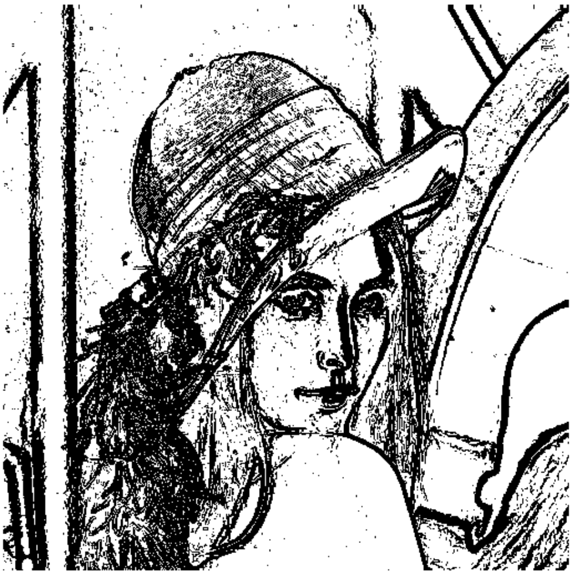
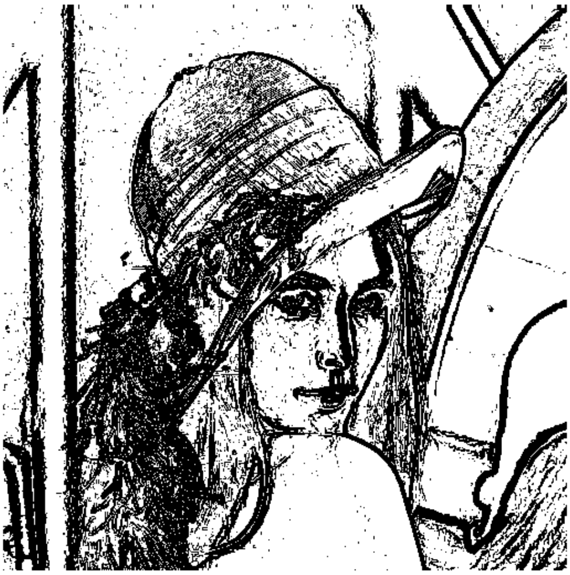
Python Packages I used

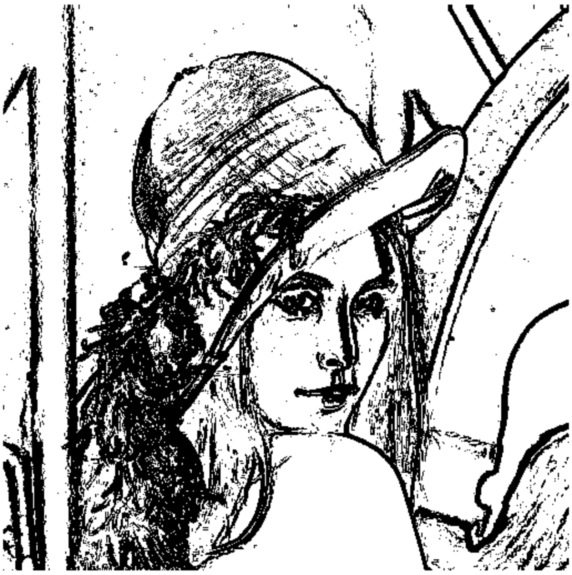
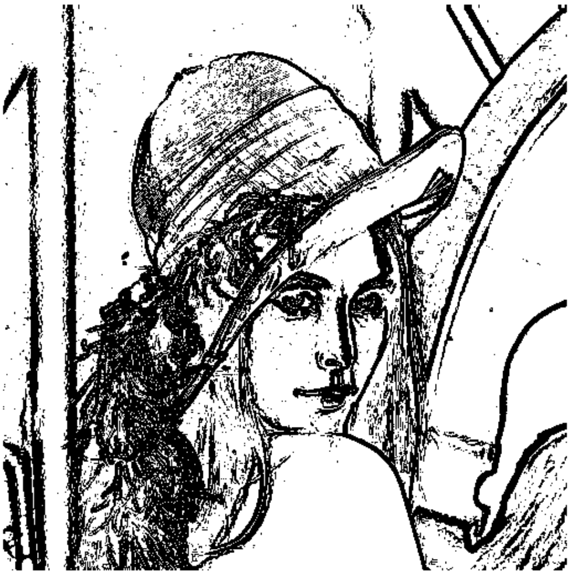
- `skimage.io`: for basic image i/o.
- `numpy`: for the convience of array manipulation.
- `math`: for calculating the value of $\sqrt{2}$
- `tqdm`: for visualizing the progress of the running code

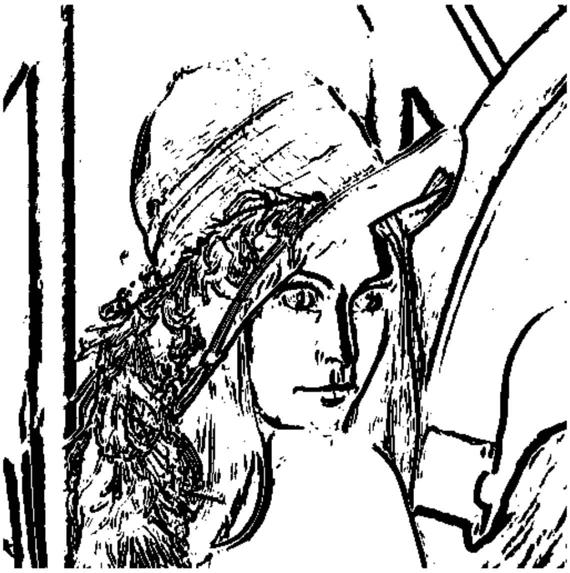
Some Other Functions I Build

- **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

robert.png	prewitt.png
	
sobel.png	frei_and_chen.png
	

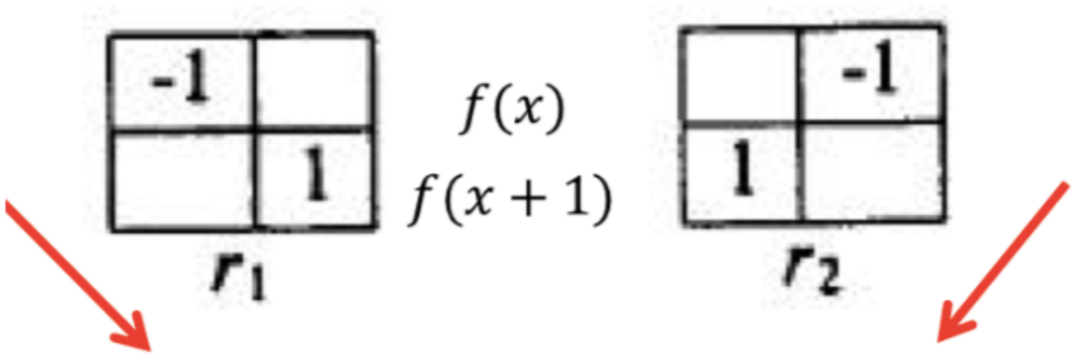
kirsch.png	robinson.png
	

nevatia_babu.png


Robert's Operator

> *threshold: 12*

Starting with `blank_image(512, 512, 255)`, I set a image with all values being 255. Using the below masks to calculate the gradient_x, and gradient_y, then I calucate the length of it. If the length > threshold, change the vaule of the pixel to 0. (I omitted the border of pixels of the image).

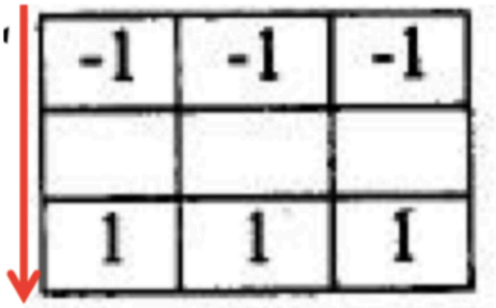


```
def robert(img, threshold):
    return_img = blank_image(height, width, 255)
    for i in tqdm(range(height - 1)):
        for j in range(width - 1):
            r1 = img[i][j] * (-1) + img[i + 1][j + 1] * 1
            r2 = img[i + 1][j] * 1 + img[i][j + 1] * (-1)
            r = r1 ** 2 + r2 ** 2
            if(r > threshold ** 2):
                return_img[i][j] = 0
    return return_img
```

Prewitt's Edge Detector

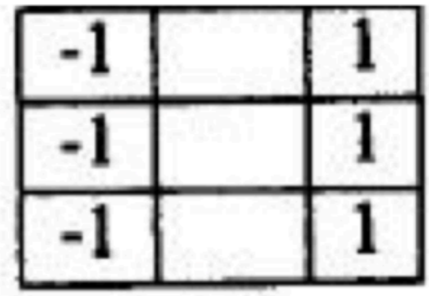
> *Threshold: 24*

Starting with `blank_image(512, 512, 255)`, I set a image with all values being 255. Using the below masks to calculate the gradient_x, and gradient_y, then I calucate the length of it. If the length > threshold, change the vaule of the pixel to 0. (I omitted the border of pixels of the image).



-1	-1	-1
1	1	1

p1



-1		1
-1		1
-1		1

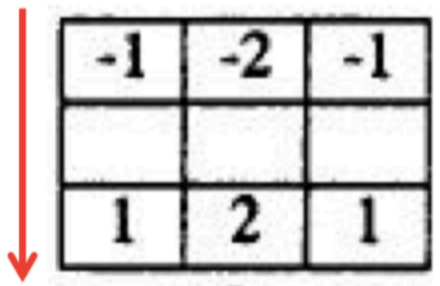
p2

```
def prewitt(img, threshold):
    return_img = blank_image(height, width, 255)
    for i in tqdm(range(1, height - 1)):
        for j in range(1, width - 1):
            p1 = img[i - 1][j - 1] * (-1) + img[i - 1][j] * (-1) +
img[i - 1][j + 1] * (-1) + img[i + 1][j - 1] + img[i + 1][j] + img[i
+ 1][j + 1]
            p2 = img[i - 1][j - 1] * (-1) + img[i][j - 1] * (-1) +
img[i + 1][j - 1] * (-1) + img[i - 1][j + 1] + img[i][j + 1] + img[i
+ 1][j + 1]
            p = p1 ** 2 + p2 ** 2
            if(p > threshold ** 2):
                return_img[i][j] = 0
    return return_img
```

Sobel's Edge Detector

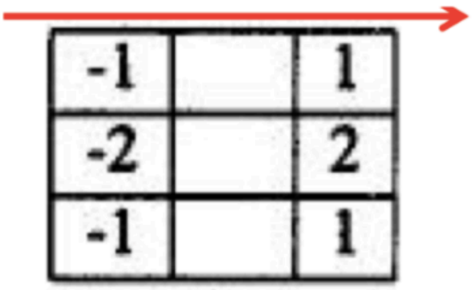
> *Threshold: 38*

Starting with `blank_image(512, 512, 255)`, I set a image with all values being 255. Using the below masks to calculate the gradient_x, and gradient_y, then I calucate the length of it. If the length > threshold, change the vaule of the pixel to 0. (I omitted the border of pixels of the image).



-1	-2	-1
1	2	1

s_1



-1		1
-2		2
-1		1

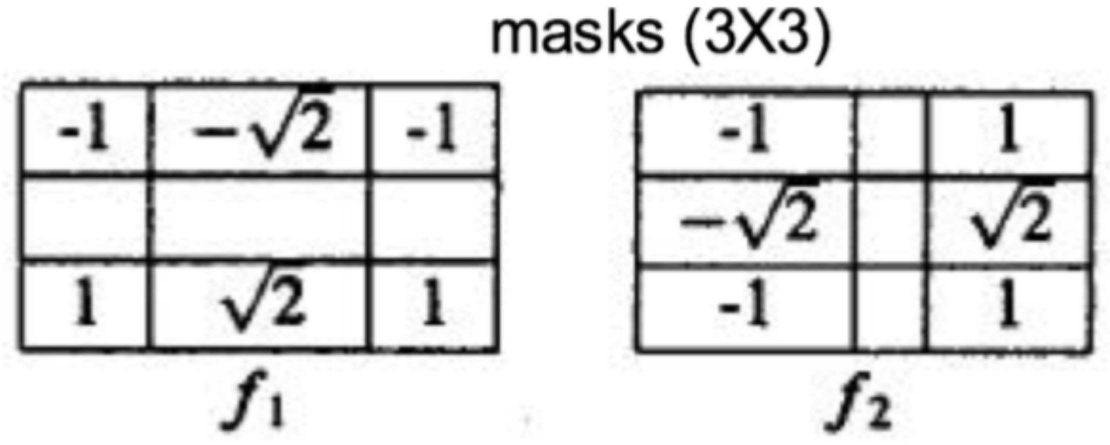
s_2

```python
def sobel(img, threshold):
    return_img = blank_image(height, width, 255)
    for i in tqdm(range(1, height - 1)):
        for j in range(1, width - 1):
            s1 = img[i - 1][j - 1] * (-1) + img[i - 1][j] * (-2) +
img[i - 1][j + 1] * (-1) + img[i + 1][j - 1] + img[i + 1][j] * 2 +
img[i + 1][j + 1]
            s2 = img[i - 1][j - 1] * (-1) + img[i][j - 1] * (-2) +
img[i + 1][j - 1] * (-1) + img[i - 1][j + 1] + img[i][j + 1] * 2 +
img[i + 1][j + 1]
            s = s1 ** 2 + s2 ** 2
            if(s > threshold ** 2):
                return_img[i][j] = 0
    return return_img
```

Frei and Chen's Gradient Operator

> *Threshold: 30*

Starting with `blank_image(512, 512, 255)`, I set a image with all values being 255. Using the below masks to calculate the gradient_x, and gradient_y, then I calucate the length of it. If the length > threshold, change the vaule of the pixel to 0. (I omitted the border of pixels of the image).



masks (3X3)

-1	$-\sqrt{2}$	-1
1	$\sqrt{2}$	1

f_1

-1		1
$-\sqrt{2}$		$\sqrt{2}$
-1		1

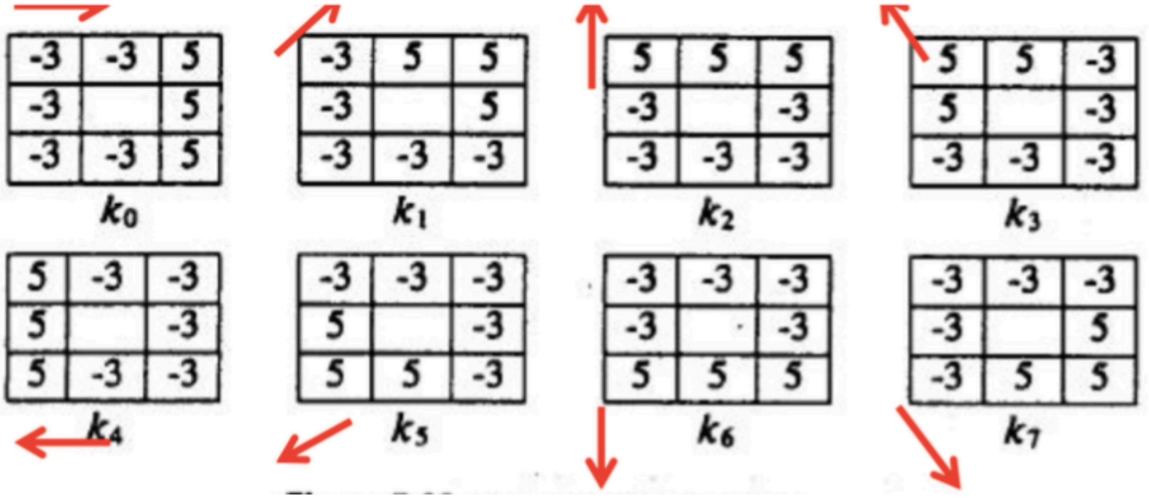
f_2

```
def frei_and_chen(img, threshold):
    return_img = blank_image(height, width, 255)
    sqrt = math.sqrt(2)
    for i in tqdm(range(1, height - 1)):
        for j in range(1, width - 1):
            f1 = img[i - 1][j - 1] * (-1) + img[i - 1][j] * (-sqrt) +
img[i - 1][j + 1] * (-1) + img[i + 1][j - 1] + img[i + 1][j] * sqrt +
img[i + 1][j + 1]
            f2 = img[i - 1][j - 1] * (-1) + img[i][j - 1] * (-sqrt) +
img[i + 1][j - 1] * (-1) + img[i - 1][j + 1] + img[i][j + 1] * sqrt +
img[i + 1][j + 1]
            f = f1 ** 2 +f2 ** 2
            if(f > threshold ** 2):
                return_img[i][j] = 0
    return return_img
```

Kirsch's Compass Operator

> *Threshold: 135*

Starting with `blank_image(512, 512, 255)`, I set a image with all values being 255. Using the below masks to calculate serveral gradients, then I choose the largest of it to compare with the threshold. If it > threshold, change the vaule of the pixel to 0. (I omitted the border of pixels of the image).



$$k_0 = \begin{bmatrix} -3 & -3 & 5 \\ -3 & & 5 \\ -3 & -3 & 5 \end{bmatrix} \quad k_1 = \begin{bmatrix} -3 & 5 & 5 \\ -3 & & 5 \\ -3 & -3 & -3 \end{bmatrix} \quad k_2 = \begin{bmatrix} 5 & 5 & 5 \\ -3 & & -3 \\ -3 & -3 & -3 \end{bmatrix} \quad k_3 = \begin{bmatrix} 5 & 5 & -3 \\ 5 & & -3 \\ -3 & -3 & -3 \end{bmatrix}$$

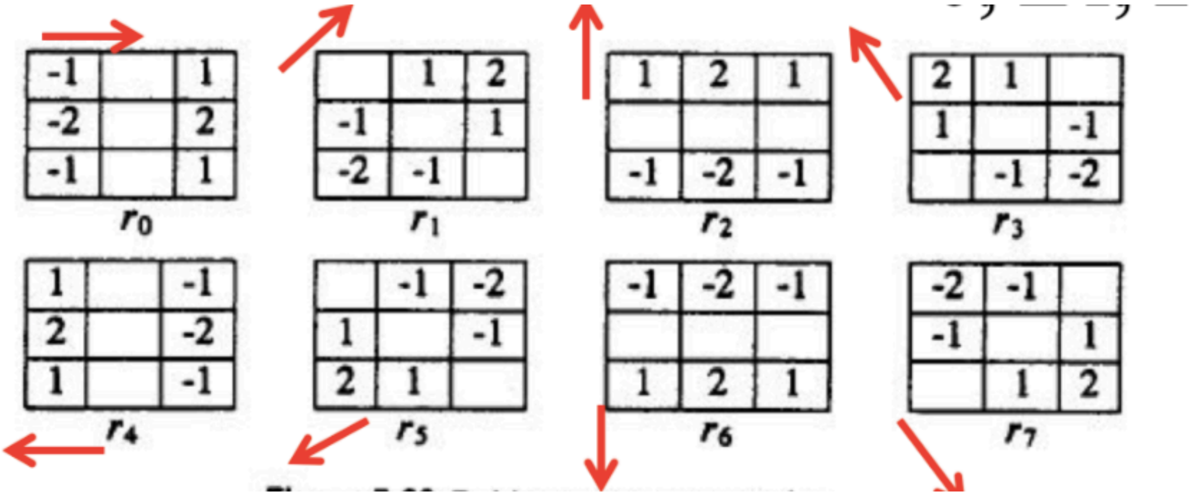
$$k_4 = \begin{bmatrix} 5 & -3 & -3 \\ 5 & & -3 \\ 5 & -3 & -3 \end{bmatrix} \quad k_5 = \begin{bmatrix} -3 & -3 & -3 \\ 5 & & -3 \\ 5 & 5 & -3 \end{bmatrix} \quad k_6 = \begin{bmatrix} -3 & -3 & -3 \\ -3 & & -3 \\ 5 & 5 & 5 \end{bmatrix} \quad k_7 = \begin{bmatrix} -3 & -3 & -3 \\ -3 & & 5 \\ -3 & 5 & 5 \end{bmatrix}$$

```
def kirsch(img, threshold):
    k = [-3, -3, -3,-3, -3, 5, 5, 5]
    return_img = blank_image(height, width, 255)
    for i in tqdm(range(1, height - 1)):
        for j in range(1, width - 1):
            k_list = []
            for idx in range(8):
                k_list.append(img[i - 1][j - 1] * k[idx % 8] + img[i
- 1][j] * k[(idx + 1) % 8] + img[i - 1][j + 1] * k[(idx + 2) % 8]
                    + img[i][j + 1] * k[(idx + 3) % 8] + img[i + 1][j
+ 1] * k[(idx + 4) % 8]
                    + img[i + 1][j] * k[(idx + 5) % 8] + img[i + 1][j
- 1] * k[(idx + 6) % 8] + img[i][j - 1] * k[(idx + 7) % 8])
            if(max(k_list) > threshold):
                return_img[i][j] = 0
```

Robinson's Compass Operator

> *Threshld: 43*

Starting with `blank_image(512, 512, 255)`, I set a image with all values being 255. Using the below mask to calculate serveral gradients, then I choose the largest of it to compare with the threshold. If it > threshold, change the vaule of the pixel to 0. (I omitted the border of pixels of the image).



-1		1
-2		2
-1		1

r_0

	1	2
-1		1
-2	-1	

r_1

1	2	1
-1	-2	-1

r_2

2	1	
1		-1
	-1	-2

r_3

1		-1
2		-2
1		-1

r_4

	-1	-2
1		-1
2	1	

r_5

-1	-2	-1
1	2	1

r_6

-2	-1	
-1		1
	1	2

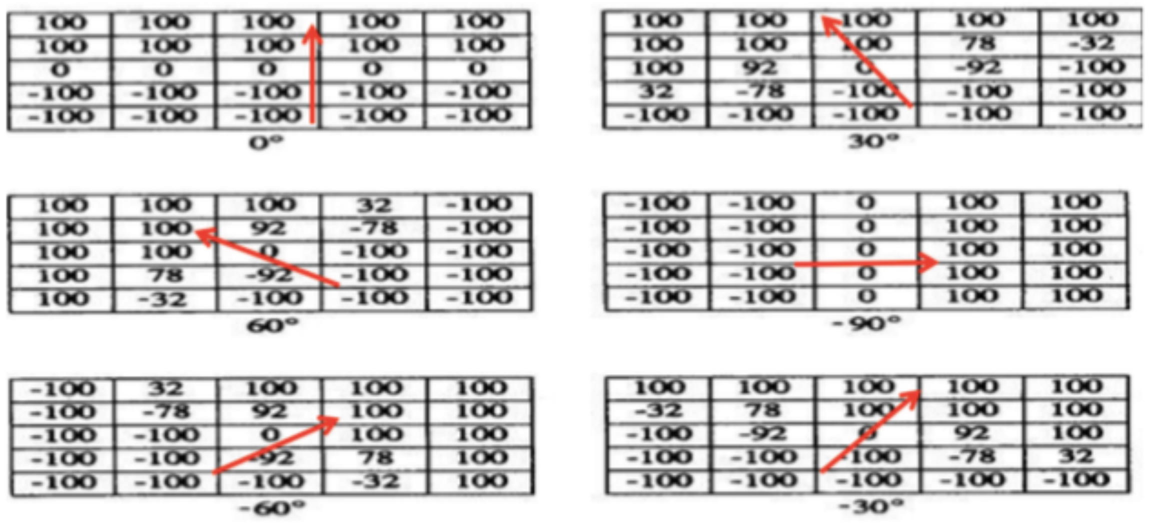
r_7

```
def robinson(img, threshold):
    r = [-1, 0, 1, 2, 1, 0, -1, -2]
    return_img = blank_image(height, width, 255)
    for i in tqdm(range(1, height - 1)):
        for j in range(1, width - 1):
            r_list = []
            for idx in range(8):
                r_list.append(img[i - 1][j - 1] * r[idx % 8] + img[i
- 1][j] * r[(idx + 1) % 8] + img[i - 1][j + 1] * r[(idx + 2) % 8]
                    + img[i][j + 1] * r[(idx + 3) % 8] + img[i + 1][j
+ 1] * r[(idx + 4) % 8]
                    + img[i + 1][j] * r[(idx + 5) % 8] + img[i + 1][j
- 1] * r[(idx + 6) % 8] + img[i][j - 1] * r[(idx + 7) % 8])
            if(max(r_list) > threshold):
                return_img[i][j] = 0
    return return_img
```

Nevatia-Babu 5x5 Operator

> *Threshold: 12500*

Starting with `blank_image(512, 512, 255)`, I set a image with all values being 255. Using the below mask to calculate serveral gradients, then I choose the largest of it to compare with the threshold. If it > threshold, change the vaule of the pixel to 0. (I omitted the border of pixels of the image).



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

0°

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

30°

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

60°

-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

-90°

-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

-60°

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

-30°

```
def nevatia_babu(img, threshold):
    kernel = [ [-2, -2], [-1, -2], [0, -2], [1, -2], [2, -2],
    [-2, -1], [-1, -1], [0, -1], [1, -1], [2, -1],
    [-2, 0], [-1, 0], [0, 0], [1, 0], [2, 0],
    [-2, 1], [-1, 1], [0, 1], [1, 1], [2, 1],
    [-2, 2], [-1, 2], [0, 2], [1, 2], [2, 2] ]
    g0 = [ 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]
    g1 = [ 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]
    g2 = [-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]
    g3 = [ 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]
    g4 = [ -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]
    g5 = [ 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]
    g = [g0, g1, g2, g3, g4, g5]
    return_img = blank_image(height, width, 255)
    for i in tqdm(range(2, height - 2)):
        for j in range(2, width - 2):
            g_list = []
            for g_idx in range(6):
                temp = 0
                for idx in range(25):
                    temp += img[i + kernel[idx][0]][j + kernel[idx]
[1]] * g[g_idx][idx]
                g_list.append(temp)
            if(max(g_list) > threshold):
                return_img[i][j] = 0
    return return_img
```