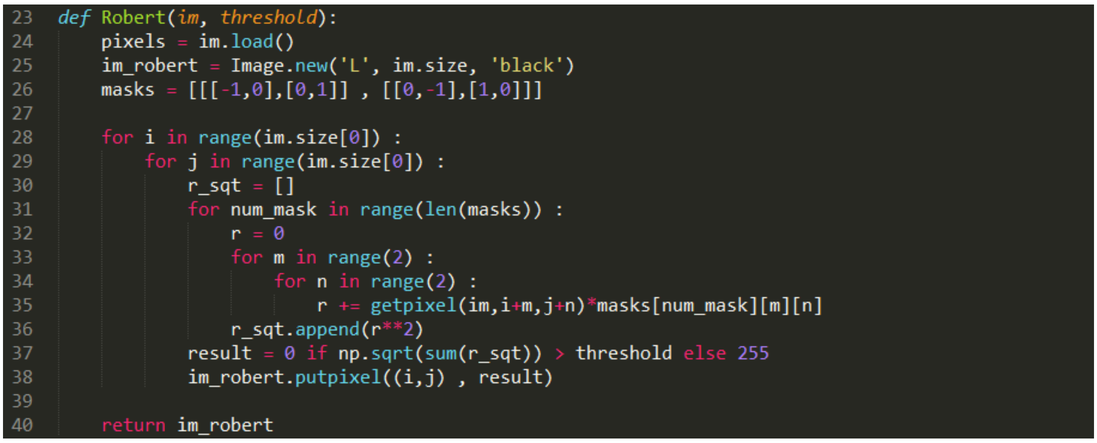
Computer Vision HW9

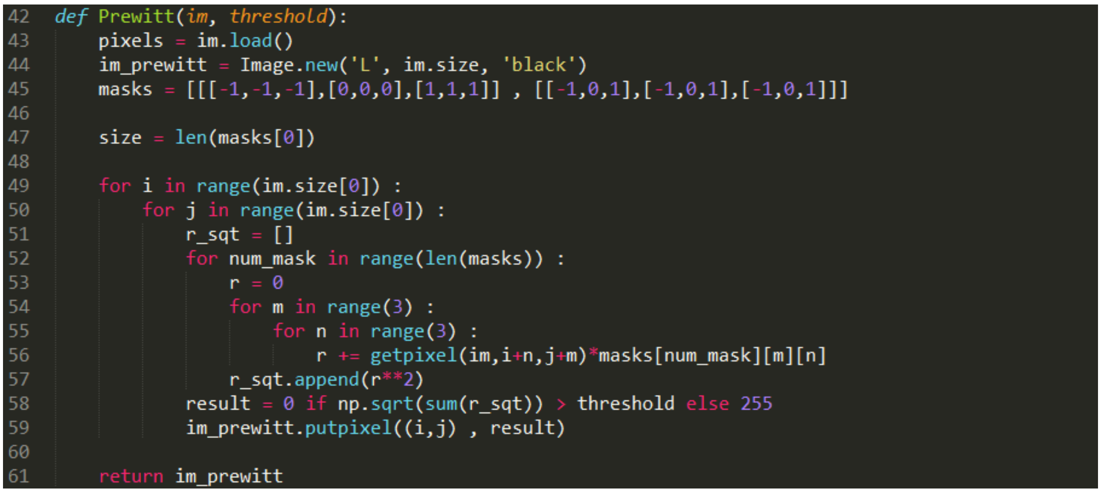
R06922075 翁瑋

- **Write programs to generate the following gradient magnitude images and choose proper thresholds to get the binary edge images: Generate additive white Gaussian noise**
  1. Roberts operator
  2. Prewitt edge detector
  3. Sobel edge detector
  4. Frei and Chen gradient operator
  5. Kirsch compass operator
  6. Robinson compass operator
  7. Nevatia-Babu 5X5 operator



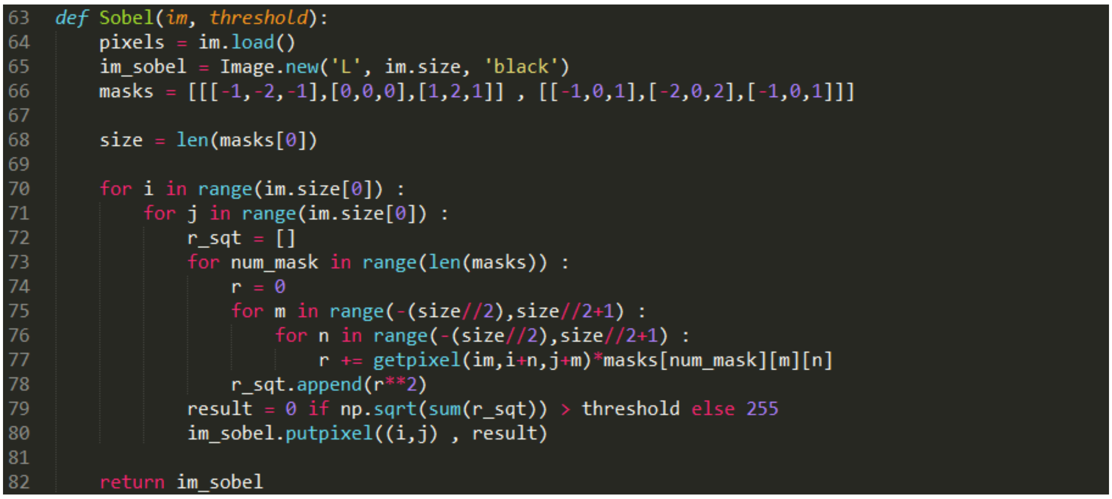
```python
def Robert(im, threshold):
    pixels = im.load()
    im_robert = Image.new('L', im.size, 'black')
    masks = [[[-1,0],[0,1]] , [[0,-1],[1,0]]]

    for i in range(im.size[0]) :
        for j in range(im.size[0]) :
            r_sqt = []
            for num_mask in range(len(masks)) :
                r = 0
                for m in range(2) :
                    for n in range(2) :
                        r += getpixel(im,i+m,j+n)*masks[num_mask][m][n]
                r_sqt.append(r**2)
            result = 0 if np.sqrt(sum(r_sqt)) > threshold else 255
            im_robert.putpixel((i,j) , result)

    return im_robert
```

Robert Operator



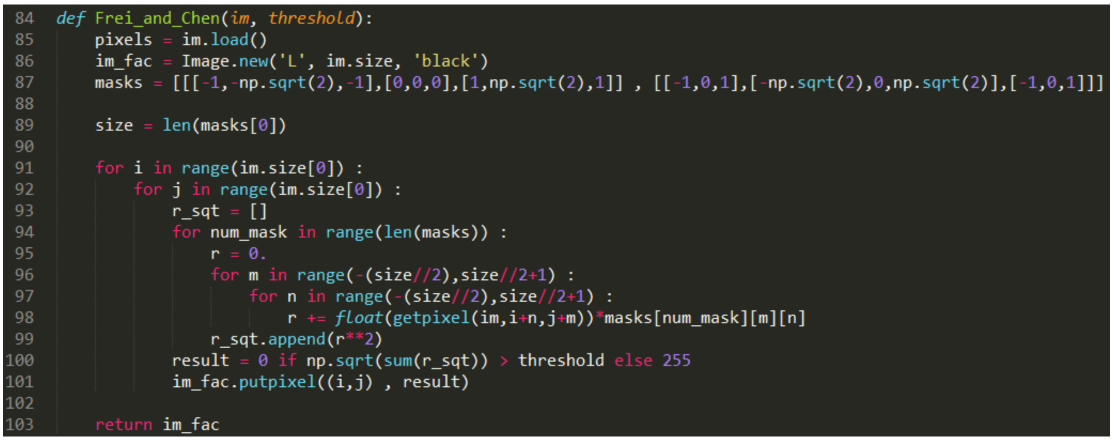
```python
def Prewitt(im, threshold):
    pixels = im.load()
    im_prewitt = Image.new('L', im.size, 'black')
    masks = [[[-1,-1,-1],[0,0,0],[1,1,1]] , [[-1,0,1],[-1,0,1],[-1,0,1]]]

    size = len(masks[0])

    for i in range(im.size[0]) :
        for j in range(im.size[0]) :
            r_sqt = []
            for num_mask in range(len(masks)) :
                r = 0
                for m in range(3) :
                    for n in range(3) :
                        r += getpixel(im,i+n,j+m)*masks[num_mask][m][n]
                r_sqt.append(r**2)
            result = 0 if np.sqrt(sum(r_sqt)) > threshold else 255
            im_prewitt.putpixel((i,j) , result)

    return im_prewitt
```

Prewitt Operator



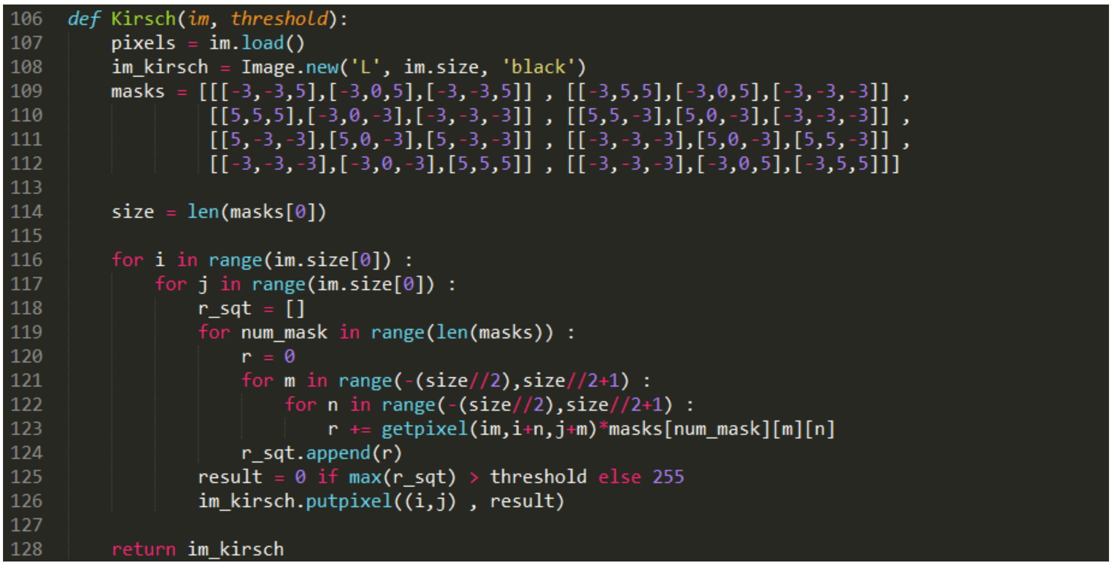
```python
def Sobel(im, threshold):
    pixels = im.load()
    im_sobel = Image.new('L', im.size, 'black')
    masks = [[[-1,-2,-1],[0,0,0],[1,2,1]] , [[-1,0,1],[-2,0,2],[-1,0,1]]]

    size = len(masks[0])

    for i in range(im.size[0]) :
        for j in range(im.size[0]) :
            r_sqt = []
            for num_mask in range(len(masks)) :
                r = 0
                for m in range(-(size//2),size//2+1) :
                    for n in range(-(size//2),size//2+1) :
                        r += getpixel(im,i+n,j+m)*masks[num_mask][m][n]
                r_sqt.append(r**2)
            result = 0 if np.sqrt(sum(r_sqt)) > threshold else 255
            im_sobel.putpixel((i,j) , result)

    return im_sobel
```

Sobel Operator



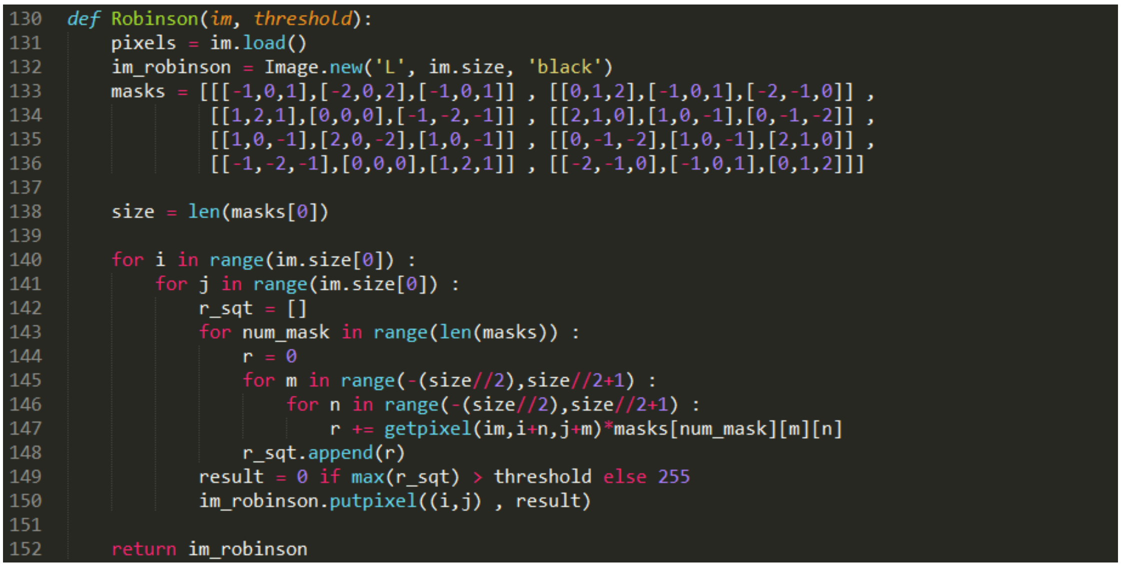
```python
def Frei_and_Chen(im, threshold):
    pixels = im.load()
    im_fac = Image.new('L', im.size, 'black')
    masks = [[[-1,-np.sqrt(2),-1],[0,0,0],[1,np.sqrt(2),1]] , [[-1,0,1],[-np.sqrt(2),0,np.sqrt(2)],[-1,0,1]]]

    size = len(masks[0])

    for i in range(im.size[0]) :
        for j in range(im.size[0]) :
            r_sqt = []
            for num_mask in range(len(masks)) :
                r = 0.
                for m in range(-(size//2),size//2+1) :
                    for n in range(-(size//2),size//2+1) :
                        r += float(getpixel(im,i+n,j+m))*masks[num_mask][m][n]
                r_sqt.append(r**2)
            result = 0 if np.sqrt(sum(r_sqt)) > threshold else 255
            im_fac.putpixel((i,j) , result)

    return im_fac
```

Frei and Chen Operator



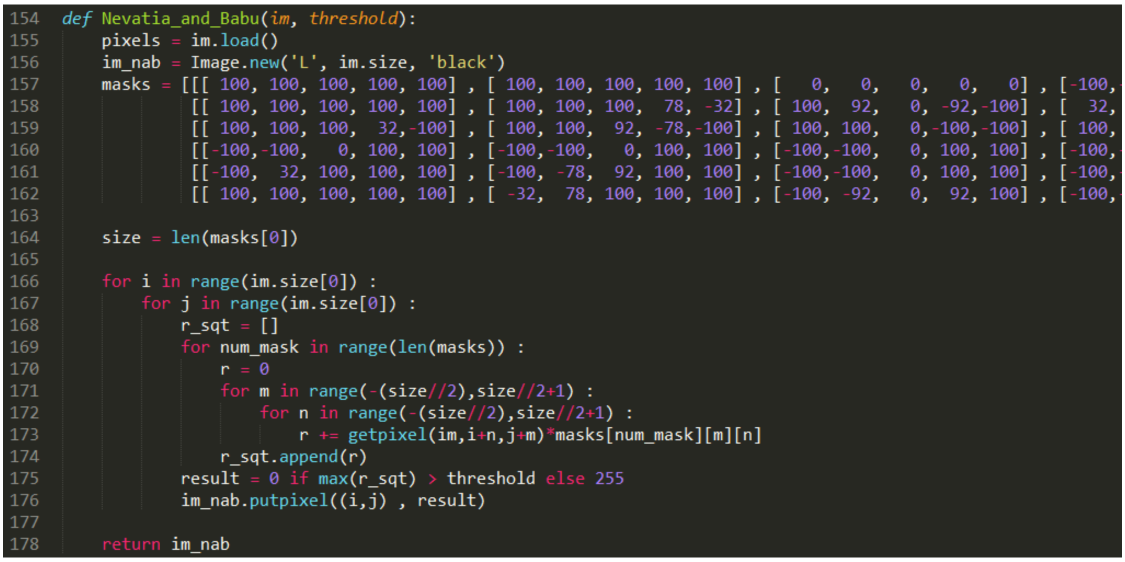
```python
def Kirsch(im, threshold):
    pixels = im.load()
    im_kirsch = Image.new('L', im.size, 'black')
    masks = [[[-3,-3,5],[-3,0,5],[-3,-3,5]] , [[-3,5,5],[-3,0,5],[-3,-3,-3]] ,
             [[5,5,5],[-3,0,-3],[-3,-3,-3]] , [[5,5,-3],[5,0,-3],[-3,-3,-3]] ,
             [[5,-3,-3],[5,0,-3],[5,-3,-3]] , [[-3,-3,-3],[5,0,-3],[5,5,-3]] ,
             [[-3,-3,-3],[-3,0,-3],[5,5,5]] , [[-3,-3,-3],[-3,0,5],[-3,5,5]]]

    size = len(masks[0])

    for i in range(im.size[0]) :
        for j in range(im.size[0]) :
            r_sqt = []
            for num_mask in range(len(masks)) :
                r = 0
                for m in range(-(size//2),size//2+1) :
                    for n in range(-(size//2),size//2+1) :
                        r += getpixel(im,i+n,j+m)*masks[num_mask][m][n]
                r_sqt.append(r)
            result = 0 if max(r_sqt) > threshold else 255
            im_kirsch.putpixel((i,j) , result)

    return im_kirsch
```

Kirsch Operator



```
def Robinson(im, threshold):
    pixels = im.load()
    im_robinson = Image.new('L', im.size, 'black')
    masks = [[[-1,0,1],[-2,0,2],[-1,0,1]] , [[0,1,2],[-1,0,1],[-2,-1,0]] ,
             [[1,2,1],[0,0,0],[-1,-2,-1]] , [[2,1,0],[1,0,-1],[0,-1,-2]] ,
             [[1,0,-1],[2,0,-2],[1,0,-1]] , [[0,-1,-2],[1,0,-1],[2,1,0]] ,
             [[-1,-2,-1],[0,0,0],[1,2,1]] , [[-2,-1,0],[-1,0,1],[0,1,2]]]

    size = len(masks[0])

    for i in range(im.size[0]) :
        for j in range(im.size[0]) :
            r_sqt = []
            for num_mask in range(len(masks)) :
                r = 0
                for m in range(-(size//2),size//2+1) :
                    for n in range(-(size//2),size//2+1) :
                        r += getpixel(im,i+n,j+m)*masks[num_mask][m][n]
                r_sqt.append(r)
            result = 0 if max(r_sqt) > threshold else 255
            im_robinson.putpixel((i,j) , result)

    return im_robinson
```

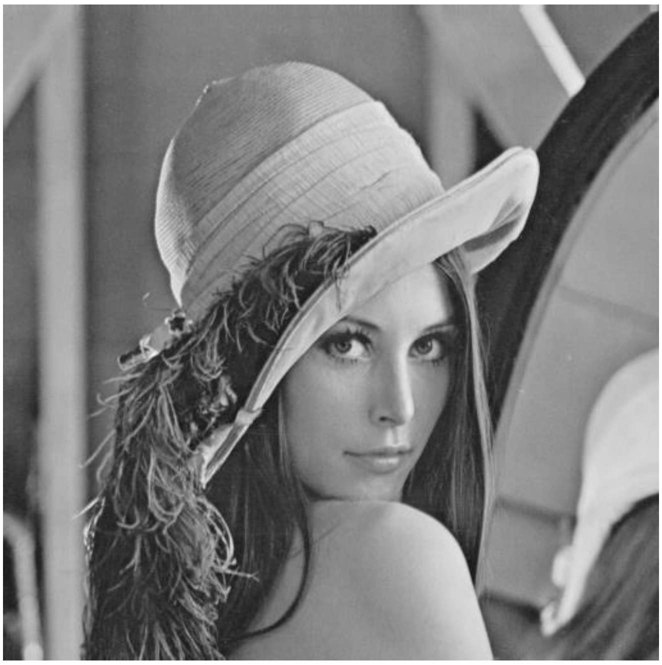
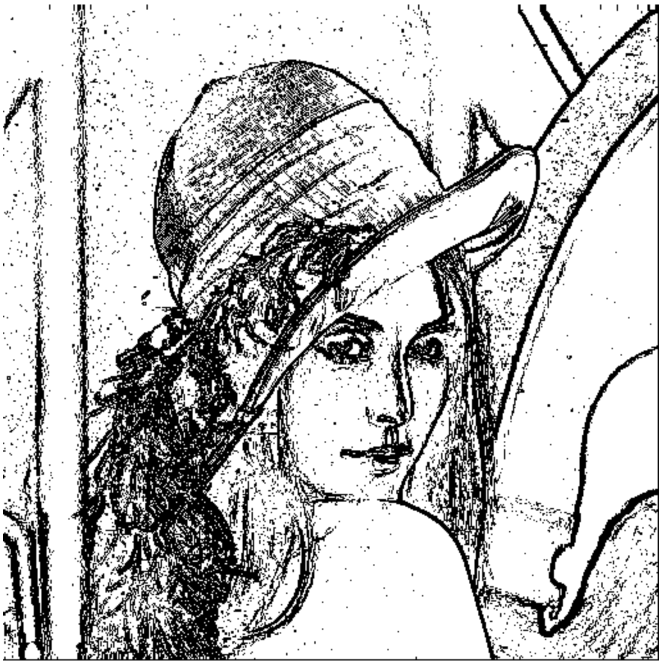
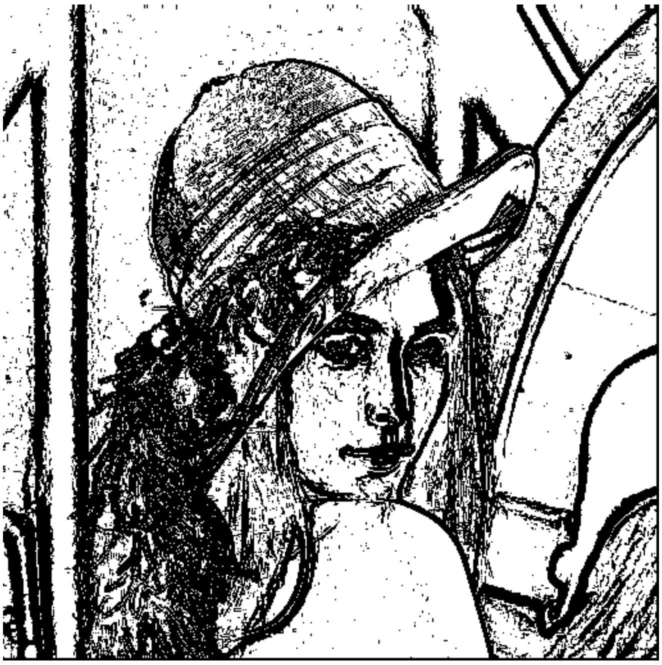
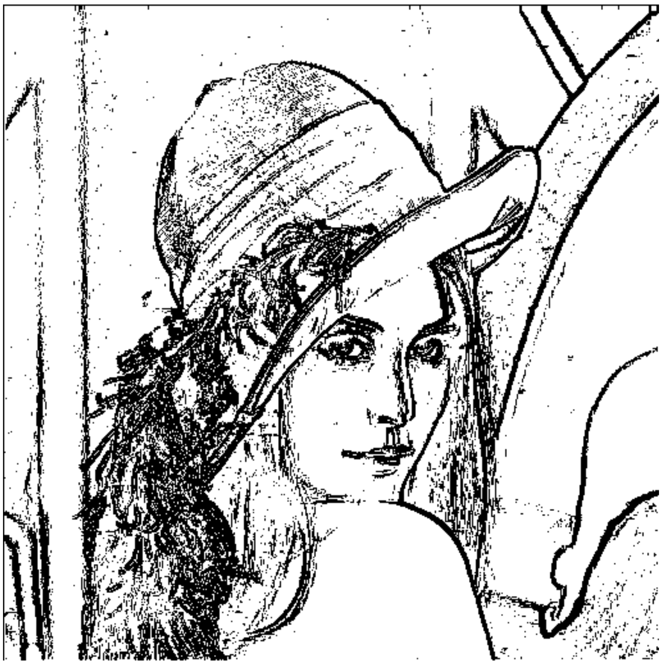
Robinson Operator

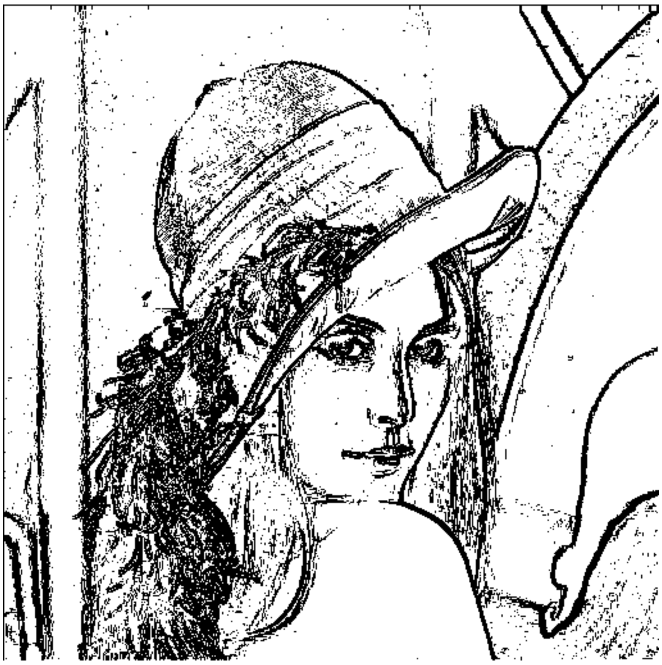
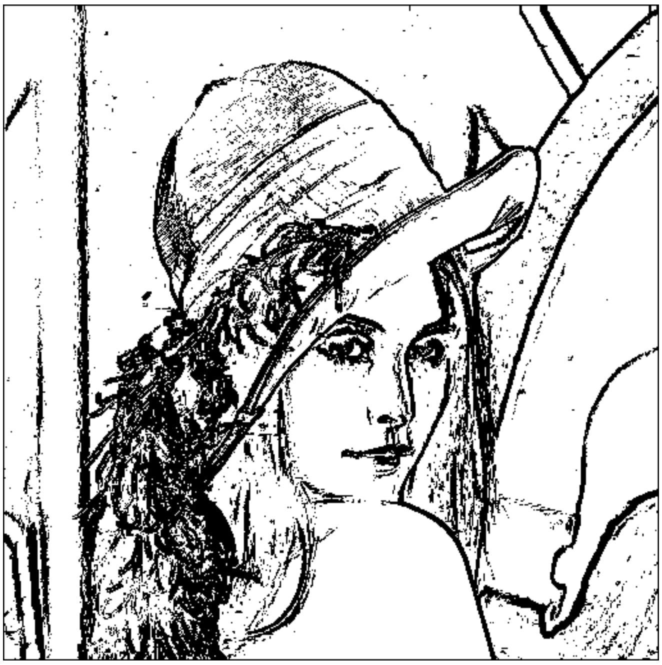


```
def Nevatia_and_Babu(im, threshold):
    pixels = im.load()
    im_nab = Image.new('L', im.size, 'black')
    masks = [[[ 100, 100, 100, 100, 100] , [ 100, 100, 100, 100, 100] , [   0,   0,   0,   0,   0] , [-100,
              [[ 100, 100, 100, 100, 100] , [ 100, 100, 100,  78, -32] , [ 100,  92,   0, -92,-100] , [  32,
              [[ 100, 100, 100,  32,-100] , [ 100, 100,  92, -78,-100] , [ 100, 100,   0,-100,-100] , [ 100,
              [[-100,-100,   0, 100, 100] , [-100,-100,   0, 100, 100] , [-100,-100,   0, 100, 100] , [-100,
              [[-100,  32, 100, 100, 100] , [-100, -78,  92, 100, 100] , [-100,-100,   0, 100, 100] , [-100,
              [[ 100, 100, 100, 100, 100] , [ -32,  78, 100, 100, 100] , [-100, -92,   0,  92, 100] , [-100,

    size = len(masks[0])

    for i in range(im.size[0]) :
        for j in range(im.size[0]) :
            r_sqt = []
            for num_mask in range(len(masks)) :
                r = 0
                for m in range(-(size//2),size//2+1) :
                    for n in range(-(size//2),size//2+1) :
                        r += getpixel(im,i+n,j+m)*masks[num_mask][m][n]
                r_sqt.append(r)
            result = 0 if max(r_sqt) > threshold else 255
            im_nab.putpixel((i,j) , result)

    return im_nab
```

Nevatia and Babu Operator

Result :

Original	Roberts operator(12)
	
Prewitt edge detector(24)	Sobel edge detector(38)
	

Frei and Chen gradient operator(30)	Kirsch compass operator(135)
	
Robinson compass operator(43)	Nevatia-Babu 5X5 operator(12500)