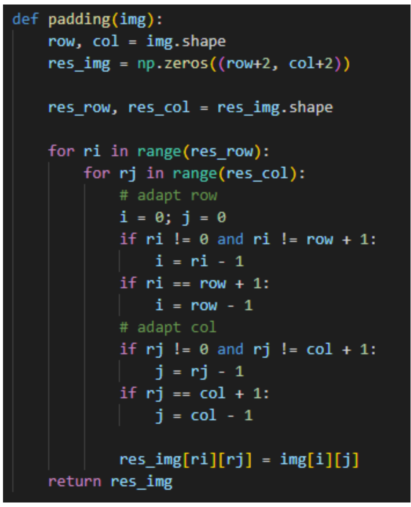
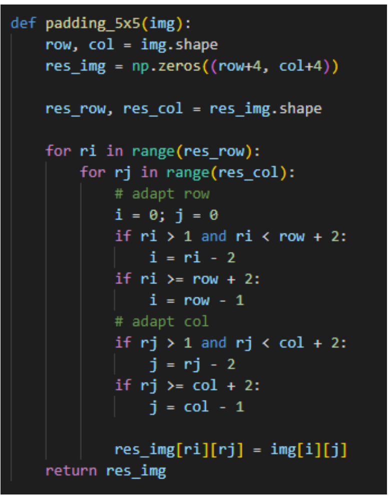
R10922123 周昱豪

Code:

做(a)~(g)前都有先將原圖做 padding，以下分別為往外補一格與補兩格的 code



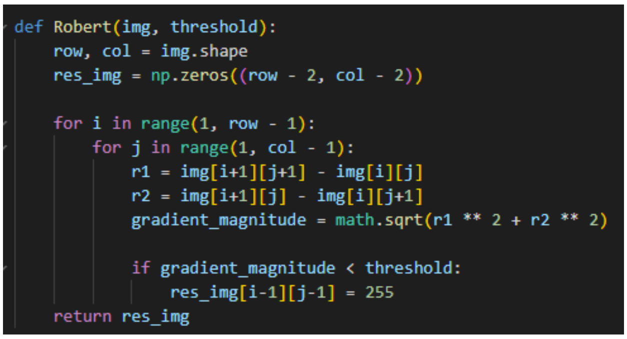
```
def padding(img):
    row, col = img.shape
    res_img = np.zeros((row+2, col+2))

    res_row, res_col = res_img.shape

    for ri in range(res_row):
        for rj in range(res_col):
            # adapt row
            i = 0; j = 0
            if ri != 0 and ri != row + 1:
                i = ri - 1
            if ri == row + 1:
                i = row - 1
            # adapt col
            if rj != 0 and rj != col + 1:
                j = rj - 1
            if rj == col + 1:
                j = col - 1

            res_img[ri][rj] = img[i][j]
    return res_img
```



```
def padding_5x5(img):
    row, col = img.shape
    res_img = np.zeros((row+4, col+4))

    res_row, res_col = res_img.shape

    for ri in range(res_row):
        for rj in range(res_col):
            # adapt row
            i = 0; j = 0
            if ri > 1 and ri < row + 2:
                i = ri - 2
            if ri >= row + 2:
                i = row - 1
            # adapt col
            if rj > 1 and rj < col + 2:
                j = rj - 2
            if rj >= col + 2:
                j = col - 1

            res_img[ri][rj] = img[i][j]
    return res_img
```

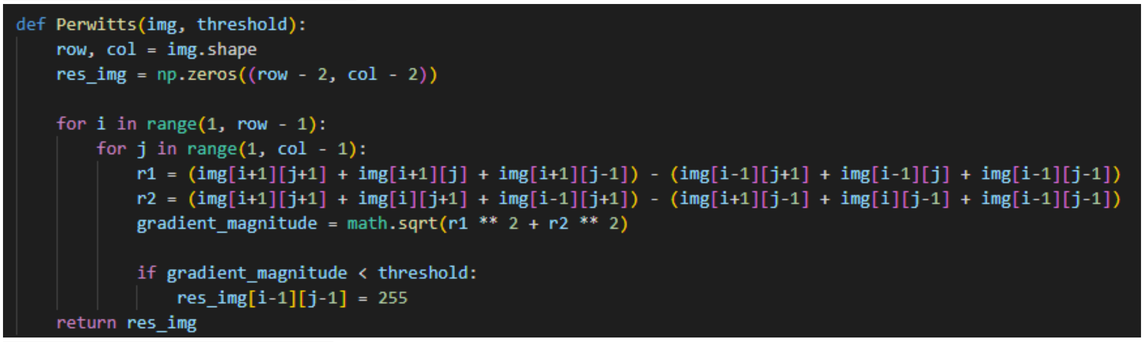
基本上(a) ~ (g)就是對 padding 後的圖，依照講義上給的方向取值計算 gradient magnitude

Robert's Operator



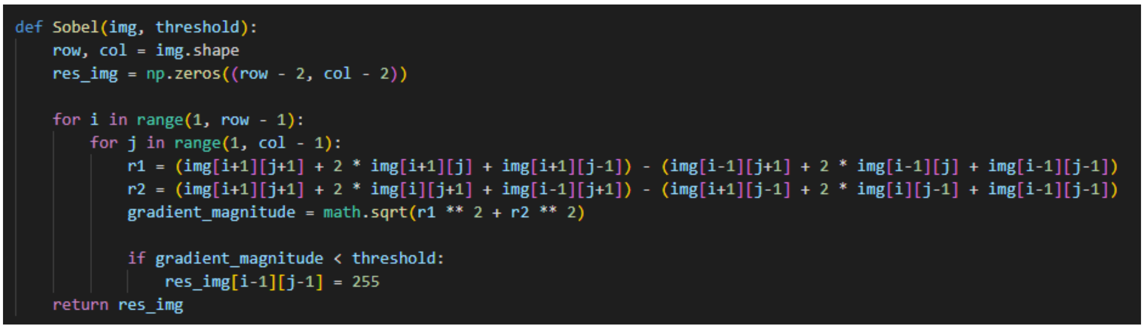
```
def Robert(img, threshold):
    row, col = img.shape
    res_img = np.zeros((row - 2, col - 2))

    for i in range(1, row - 1):
        for j in range(1, col - 1):
            r1 = img[i+1][j+1] - img[i][j]
            r2 = img[i+1][j] - img[i][j+1]
            gradient_magnitude = math.sqrt(r1 ** 2 + r2 ** 2)

            if gradient_magnitude < threshold:
                res_img[i-1][j-1] = 255
    return res_img
```

Prewitt's Edge Detector



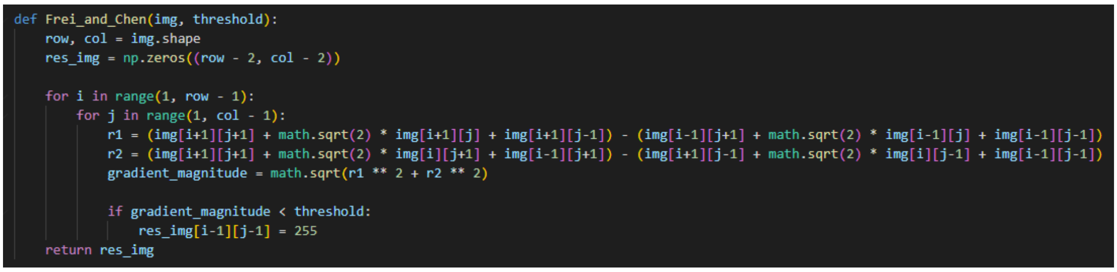
```
def Perwitts(img, threshold):
    row, col = img.shape
    res_img = np.zeros((row - 2, col - 2))

    for i in range(1, row - 1):
        for j in range(1, col - 1):
            r1 = (img[i+1][j+1] + img[i+1][j] + img[i+1][j-1]) - (img[i-1][j+1] + img[i-1][j] + img[i-1][j-1])
            r2 = (img[i+1][j+1] + img[i][j+1] + img[i-1][j+1]) - (img[i+1][j-1] + img[i][j-1] + img[i-1][j-1])
            gradient_magnitude = math.sqrt(r1 ** 2 + r2 ** 2)

            if gradient_magnitude < threshold:
                res_img[i-1][j-1] = 255
    return res_img
```

Sobel's Edge Detector



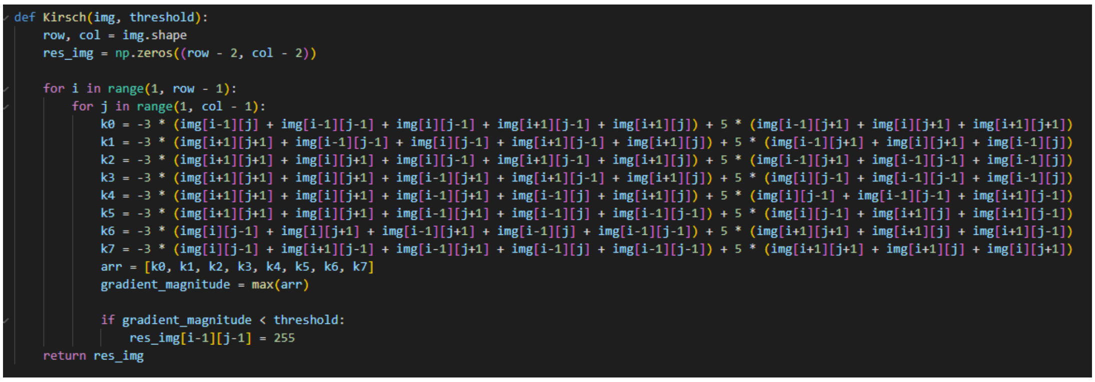
```
def Sobel(img, threshold):
    row, col = img.shape
    res_img = np.zeros((row - 2, col - 2))

    for i in range(1, row - 1):
        for j in range(1, col - 1):
            r1 = (img[i+1][j+1] + 2 * img[i+1][j] + img[i+1][j-1]) - (img[i-1][j+1] + 2 * img[i-1][j] + img[i-1][j-1])
            r2 = (img[i+1][j+1] + 2 * img[i][j+1] + img[i-1][j+1]) - (img[i+1][j-1] + 2 * img[i][j-1] + img[i-1][j-1])
            gradient_magnitude = math.sqrt(r1 ** 2 + r2 ** 2)

            if gradient_magnitude < threshold:
                res_img[i-1][j-1] = 255
    return res_img
```

Frei and Chen's Gradient Operator



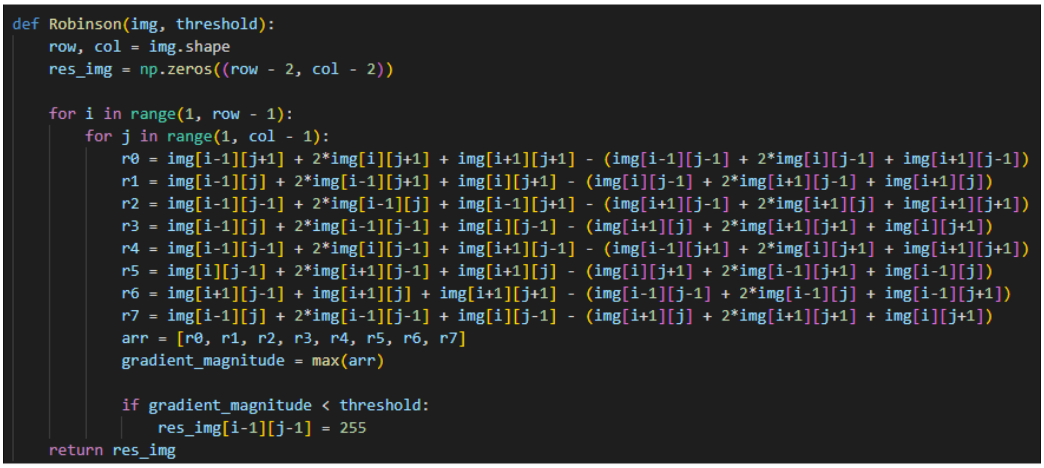
```python
def Frei_and_Chen(img, threshold):
    row, col = img.shape
    res_img = np.zeros((row - 2, col - 2))

    for i in range(1, row - 1):
        for j in range(1, col - 1):
            r1 = (img[i+1][j+1] + math.sqrt(2) * img[i+1][j] + img[i+1][j-1]) - (img[i-1][j+1] + math.sqrt(2) * img[i-1][j] + img[i-1][j-1])
            r2 = (img[i+1][j+1] + math.sqrt(2) * img[i][j+1] + img[i-1][j+1]) - (img[i+1][j-1] + math.sqrt(2) * img[i][j-1] + img[i-1][j-1])
            gradient_magnitude = math.sqrt(r1 ** 2 + r2 ** 2)

            if gradient_magnitude < threshold:
                res_img[i-1][j-1] = 255
    return res_img
```

Kirsch's Compass Operator



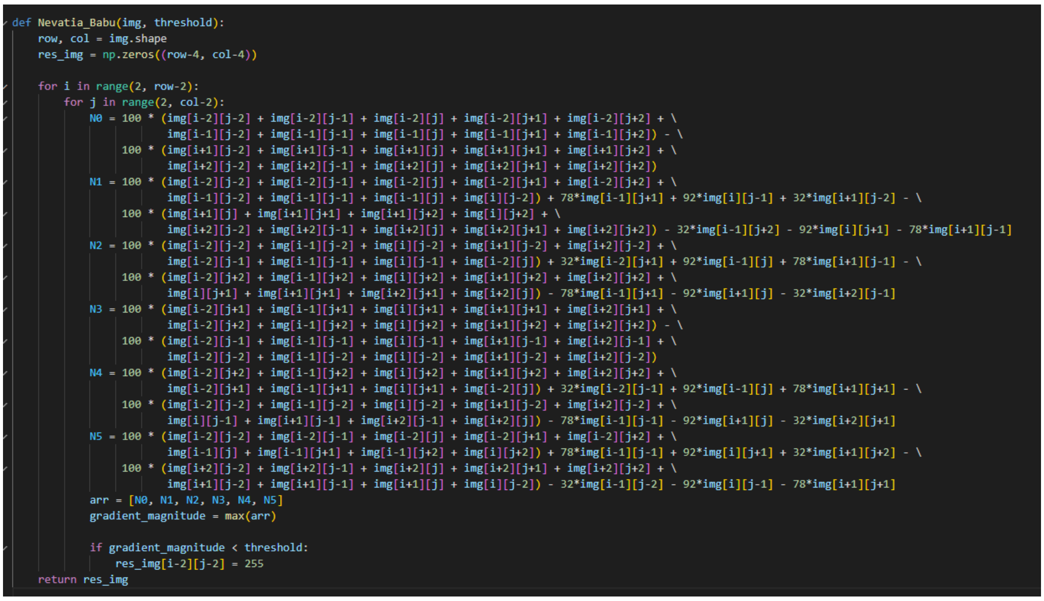
```python
def Kirsch(img, threshold):
    row, col = img.shape
    res_img = np.zeros((row - 2, col - 2))

    for i in range(1, row - 1):
        for j in range(1, col - 1):
            k0 = -3 * (img[i-1][j] + img[i-1][j-1] + img[i][j-1] + img[i+1][j-1] + img[i+1][j]) + 5 * (img[i-1][j+1] + img[i][j+1] + img[i+1][j+1])
            k1 = -3 * (img[i+1][j+1] + img[i-1][j-1] + img[i][j-1] + img[i+1][j-1] + img[i+1][j]) + 5 * (img[i-1][j+1] + img[i][j+1] + img[i-1][j])
            k2 = -3 * (img[i+1][j+1] + img[i][j+1] + img[i][j-1] + img[i+1][j-1] + img[i+1][j]) + 5 * (img[i-1][j+1] + img[i-1][j-1] + img[i-1][j])
            k3 = -3 * (img[i+1][j+1] + img[i][j+1] + img[i-1][j+1] + img[i+1][j-1] + img[i+1][j]) + 5 * (img[i][j-1] + img[i-1][j-1] + img[i-1][j])
            k4 = -3 * (img[i+1][j+1] + img[i][j+1] + img[i-1][j+1] + img[i-1][j] + img[i+1][j]) + 5 * (img[i][j-1] + img[i-1][j-1] + img[i+1][j-1])
            k5 = -3 * (img[i+1][j+1] + img[i][j+1] + img[i-1][j+1] + img[i-1][j] + img[i-1][j-1]) + 5 * (img[i][j-1] + img[i+1][j] + img[i+1][j-1])
            k6 = -3 * (img[i][j-1] + img[i][j+1] + img[i-1][j+1] + img[i-1][j] + img[i-1][j-1]) + 5 * (img[i+1][j+1] + img[i+1][j] + img[i+1][j-1])
            k7 = -3 * (img[i][j-1] + img[i+1][j-1] + img[i-1][j+1] + img[i-1][j] + img[i-1][j-1]) + 5 * (img[i+1][j+1] + img[i+1][j] + img[i][j+1])
            arr = [k0, k1, k2, k3, k4, k5, k6, k7]
            gradient_magnitude = max(arr)

            if gradient_magnitude < threshold:
                res_img[i-1][j-1] = 255
    return res_img
```

Robinson's Compass Operator



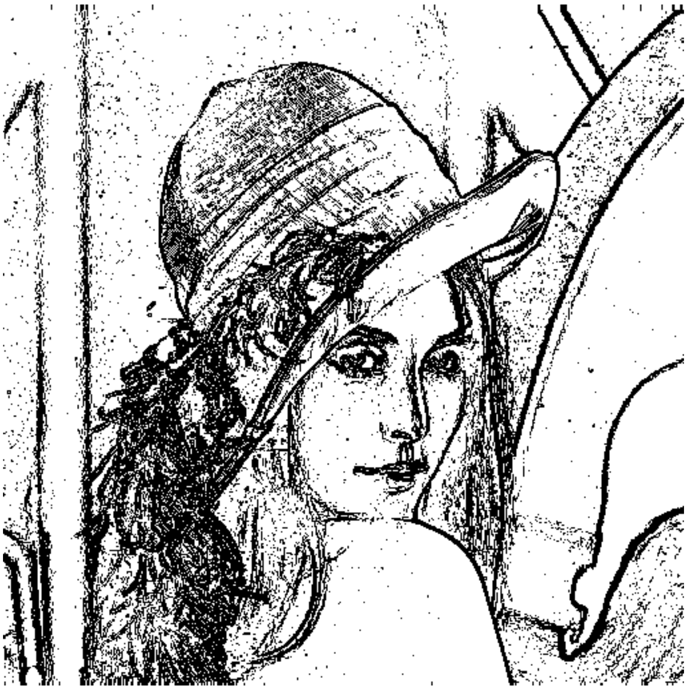
```python
def Robinson(img, threshold):
    row, col = img.shape
    res_img = np.zeros((row - 2, col - 2))

    for i in range(1, row - 1):
        for j in range(1, col - 1):
            r0 = img[i-1][j+1] + 2*img[i][j+1] + img[i+1][j+1] - (img[i-1][j-1] + 2*img[i][j-1] + img[i+1][j-1])
            r1 = img[i-1][j] + 2*img[i-1][j+1] + img[i][j+1] - (img[i][j-1] + 2*img[i+1][j-1] + img[i+1][j])
            r2 = img[i-1][j-1] + 2*img[i-1][j] + img[i-1][j+1] - (img[i+1][j-1] + 2*img[i+1][j] + img[i+1][j+1])
            r3 = img[i-1][j] + 2*img[i-1][j-1] + img[i][j-1] - (img[i+1][j] + 2*img[i+1][j+1] + img[i][j+1])
            r4 = img[i-1][j-1] + 2*img[i][j-1] + img[i+1][j-1] - (img[i-1][j+1] + 2*img[i][j+1] + img[i+1][j+1])
            r5 = img[i][j-1] + 2*img[i+1][j-1] + img[i+1][j] - (img[i][j+1] + 2*img[i-1][j+1] + img[i-1][j])
            r6 = img[i+1][j-1] + img[i+1][j] + img[i+1][j+1] - (img[i-1][j-1] + 2*img[i-1][j] + img[i-1][j+1])
            r7 = img[i-1][j] + 2*img[i-1][j-1] + img[i][j-1] - (img[i+1][j] + 2*img[i+1][j+1] + img[i][j+1])
            arr = [r0, r1, r2, r3, r4, r5, r6, r7]
            gradient_magnitude = max(arr)

            if gradient_magnitude < threshold:
                res_img[i-1][j-1] = 255
    return res_img
```

Nevatia-Babu 5x5 Operator

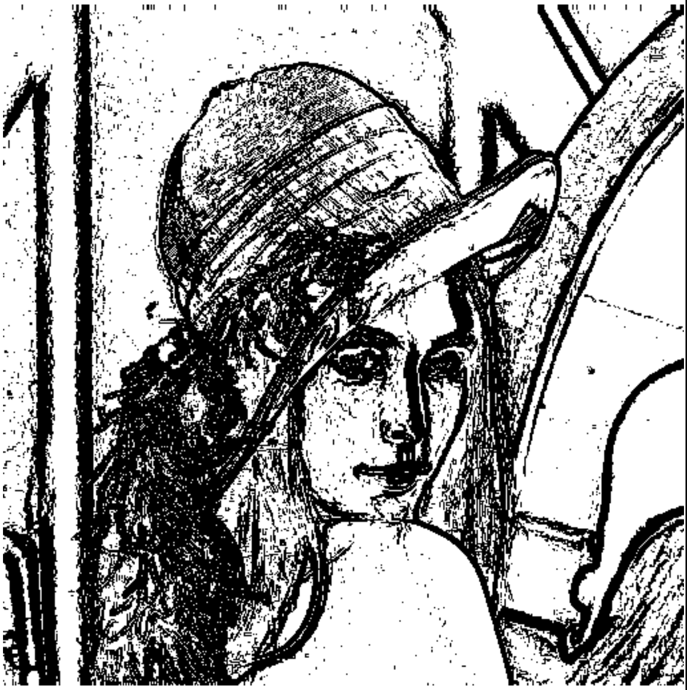


```python
def Nevatia_Babu(img, threshold):
    row, col = img.shape
    res_img = np.zeros((row-4, col-4))

    for i in range(2, row-2):
        for j in range(2, col-2):
            N0 = 100 * (img[i-2][j-2] + img[i-2][j-1] + img[i-2][j] + img[i-2][j+1] + img[i-2][j+2] + \
                        img[i-1][j-2] + img[i-1][j-1] + img[i-1][j] + img[i-1][j+1] + img[i-1][j+2]) - \
                 100 * (img[i+1][j-2] + img[i+1][j-1] + img[i+1][j] + img[i+1][j+1] + img[i+1][j+2] + \
                        img[i+2][j-2] + img[i+2][j-1] + img[i+2][j] + img[i+2][j+1] + img[i+2][j+2])
            N1 = 100 * (img[i-2][j-2] + img[i-2][j-1] + img[i-2][j] + img[i-2][j+1] + img[i-2][j+2] + \
                        img[i-1][j-2] + img[i-1][j-1] + img[i-1][j] + img[i][j-2]) + 78*img[i-1][j+1] + 92*img[i][j-1] + 32*img[i+1][j-2] - \
                 100 * (img[i+1][j] + img[i+1][j+1] + img[i+1][j+2] + img[i][j+2] + \
                        img[i+2][j-2] + img[i+2][j-1] + img[i+2][j] + img[i+2][j+1] + img[i+2][j+2]) - 32*img[i-1][j+2] - 92*img[i][j+1] - 78*img[i+1][j-1]
            N2 = 100 * (img[i-2][j-2] + img[i-1][j-2] + img[i][j-2] + img[i+1][j-2] + img[i+2][j-2] + \
                        img[i-2][j-1] + img[i-1][j-1] + img[i][j-1] + img[i-2][j]) + 32*img[i-2][j+1] + 92*img[i-1][j] + 78*img[i+1][j-1] - \
                 100 * (img[i-2][j+2] + img[i-1][j+2] + img[i][j+2] + img[i+1][j+2] + img[i+2][j+2] + \
                        img[i][j+1] + img[i+1][j+1] + img[i+2][j+1] + img[i+2][j]) - 78*img[i-1][j+1] - 92*img[i+1][j] - 32*img[i+2][j-1]
            N3 = 100 * (img[i-2][j+1] + img[i-1][j+1] + img[i][j+1] + img[i+1][j+1] + img[i+2][j+1] + \
                        img[i-2][j+2] + img[i-1][j+2] + img[i][j+2] + img[i+1][j+2] + img[i+2][j+2]) - \
                 100 * (img[i-2][j-1] + img[i-1][j-1] + img[i][j-1] + img[i+1][j-1] + img[i+2][j-1] + \
                        img[i-2][j-2] + img[i-1][j-2] + img[i][j-2] + img[i+1][j-2] + img[i+2][j-2])
            N4 = 100 * (img[i-2][j+2] + img[i-1][j+2] + img[i][j+2] + img[i+1][j+2] + img[i+2][j+2] + \
                        img[i-2][j+1] + img[i-1][j+1] + img[i][j+1] + img[i-2][j]) + 32*img[i-2][j-1] + 92*img[i-1][j] + 78*img[i+1][j+1] - \
                 100 * (img[i-2][j-2] + img[i-1][j-2] + img[i][j-2] + img[i+1][j-2] + img[i+2][j-2] + \
                        img[i][j-1] + img[i+1][j-1] + img[i+2][j-1] + img[i+2][j]) - 78*img[i-1][j-1] - 92*img[i+1][j] - 32*img[i+2][j+1]
            N5 = 100 * (img[i-2][j-2] + img[i-2][j-1] + img[i-2][j] + img[i-2][j+1] + img[i-2][j+2] + \
                        img[i-1][j] + img[i-1][j+1] + img[i-1][j+2] + img[i][j+2]) + 78*img[i-1][j-1] + 92*img[i][j+1] + 32*img[i+1][j+2] - \
                 100 * (img[i+2][j-2] + img[i+2][j-1] + img[i+2][j] + img[i+2][j+1] + img[i+2][j+2] + \
                        img[i+1][j-2] + img[i+1][j-1] + img[i+1][j] + img[i][j-2]) - 32*img[i-1][j-2] - 92*img[i][j-1] - 78*img[i+1][j+1]
            arr = [N0, N1, N2, N3, N4, N5]
            gradient_magnitude = max(arr)

            if gradient_magnitude < threshold:
                res_img[i-2][j-2] = 255
    return res_img
```

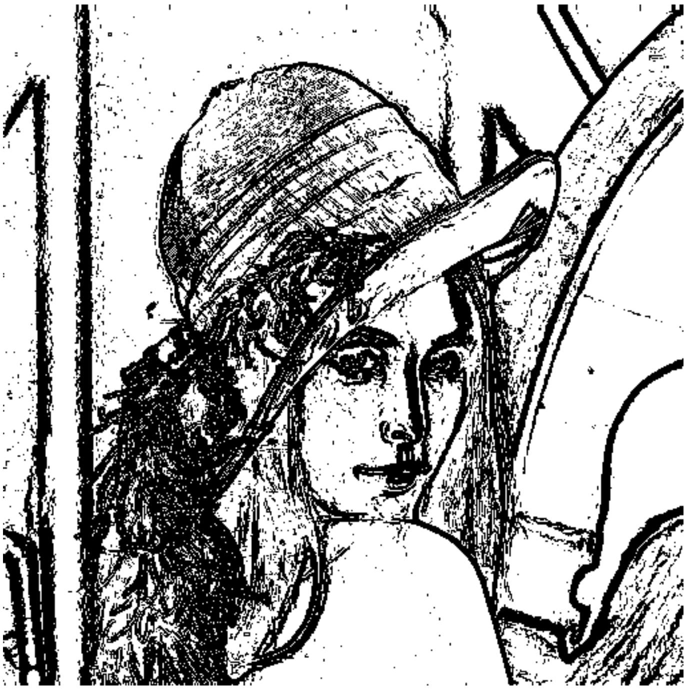
Result



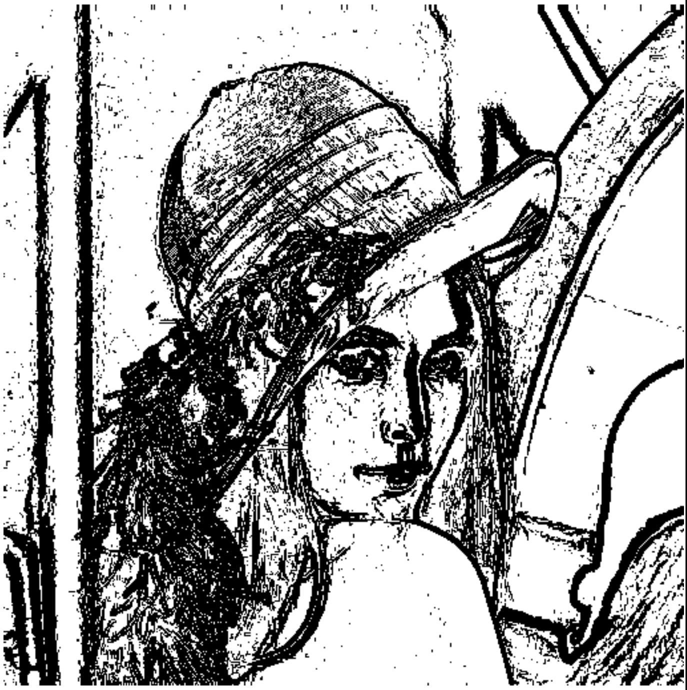
Robert's Operator: 12



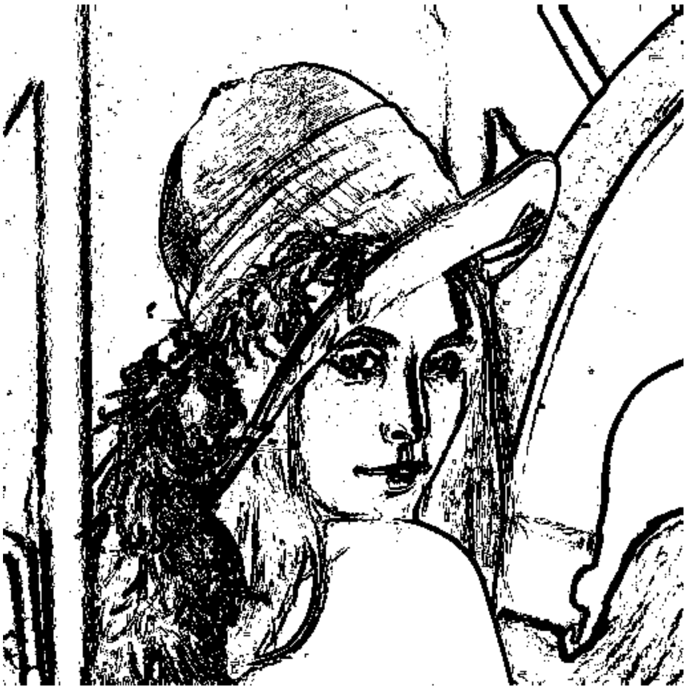
Prewitt's Edge Detector: 24



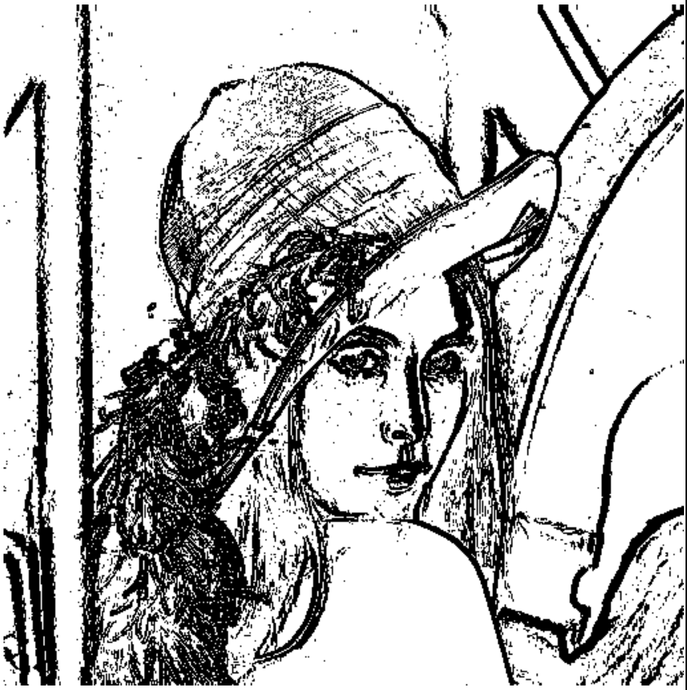
Sobel's Edge Detector: 38



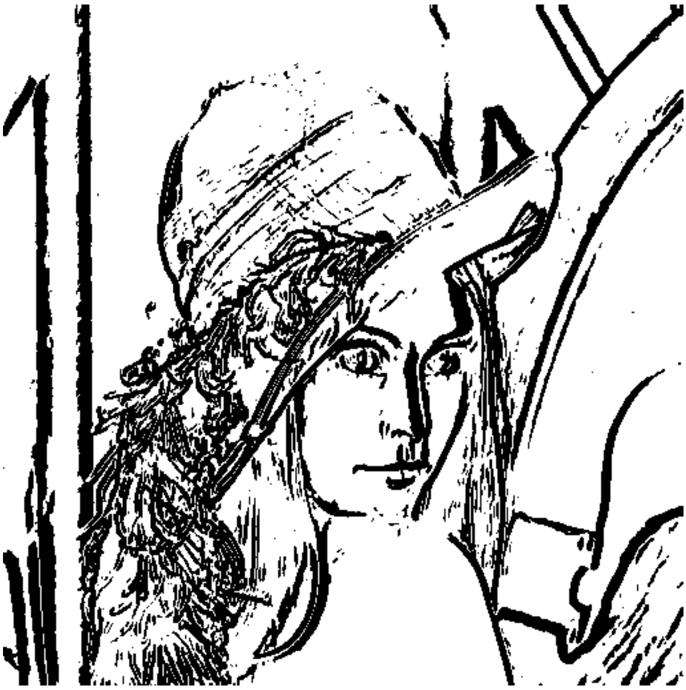
Frei and Chen's Gradient Operator: 30



Kirsch's Compass Operator: 135



Robinson's Compass Operator: 43



Nevatia-Babu 5x5 Operator: 12500