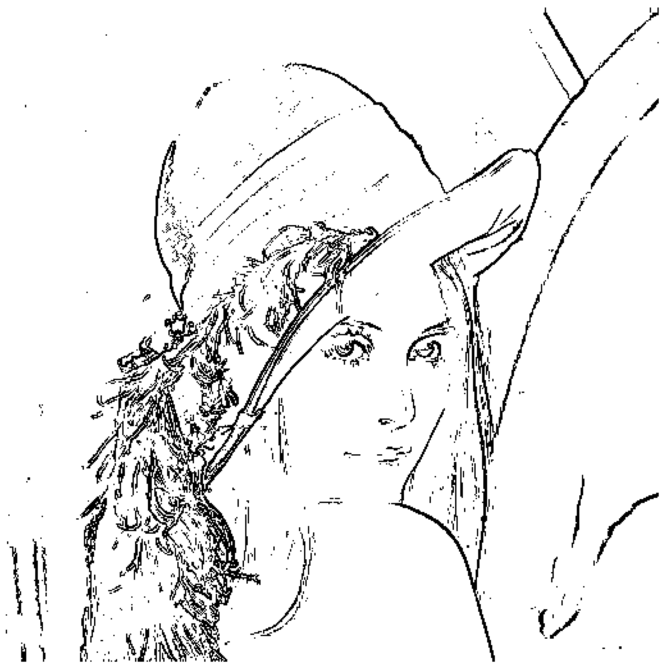
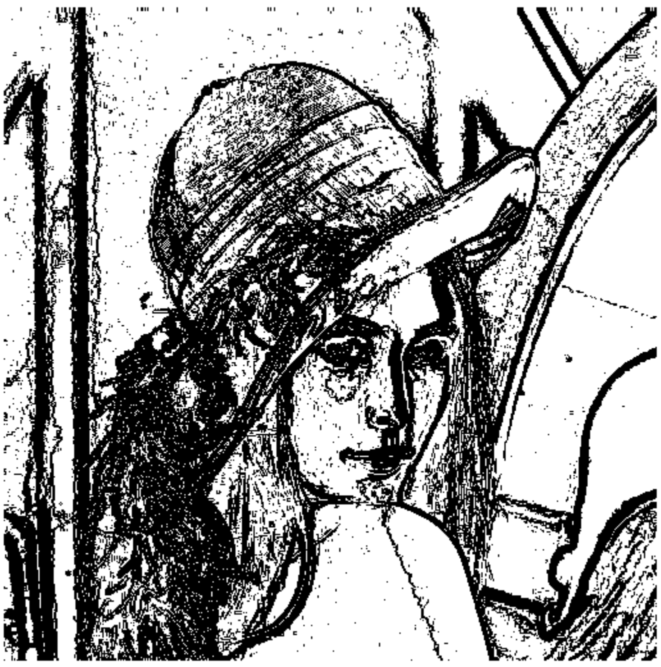
```
def Roberts(img, threshold):
    img_rob = np.zeros((img.shape[0], img.shape[1]))
    Gx = np.zeros((img.shape[0], img.shape[1]))
    Gy = np.zeros((img.shape[0], img.shape[1]))

    for i in range(img.shape[0]-1):
        for j in range(img.shape[1]-1):
            Gx[i][j] = (-1)*img[i][j][0] + 0*img[i][j+1][0] + 0*img[i+1][j][0]
+ 1*img[i+1][j+1][0]
            Gy[i][j] = 0*img[i][j][0] + (-1)*img[i][j+1][0] + 1*img[i+1][j][0]
+ 0*img[i+1][j+1][0]

    for i in range(img_rob.shape[0]):
        for j in range(img_rob.shape[1]):
            if sqrt(Gx[i][j]**2 + Gy[i][j]**2) < threshold:
                img_rob[i][j] = 255
    return img_rob
```

- 使用兩個mask(r1 = [-1, 0, 0, 1], r2 = [0, -1, 1, 0])對鄰域進行計算，將使用r1計算的值寫入Gx、r2計算的值寫入Gy。計算完所有pixel的Gx和Gy後，如果該pixel的$\sqrt{Gx^2 + Gy^2}$比閥值大則將pixel的數值更新為0，若小於閥值則更新為255。
- Roberts operator with threshold = 30



```
# Prewitt edge detector
def Prewitt(img, threshold):
    img_pre = np.zeros((img.shape[0], img.shape[1]))
    Gx = np.zeros((img.shape[0], img.shape[1]))
    Gy = np.zeros((img.shape[0], img.shape[1]))
    for i in range(1, img.shape[0]-1):
        for j in range(1, img.shape[1]-1):
            Gx[i][j] = (-1)*(img[i-1][j-1][0]+img[i-1][j][0]+img[i-1][j+1]
[0])+1*(img[i+1][j-1][0]+img[i+1][j][0]+img[i+1][j+1][0])
            Gy[i][j] = (-1)*(img[i-1][j-1][0]+img[i][j-1][0]+img[i+1][j-1]
[0])+1*(img[i-1][j+1][0]+img[i][j+1][0]+img[i+1][j+1][0])

    for i in range(img_pre.shape[0]):
        for j in range(img_pre.shape[1]):
            if sqrt(Gx[i][j]**2 + Gy[i][j]**2) < threshold:
                img_pre[i][j] = 255
    return img_pre
```

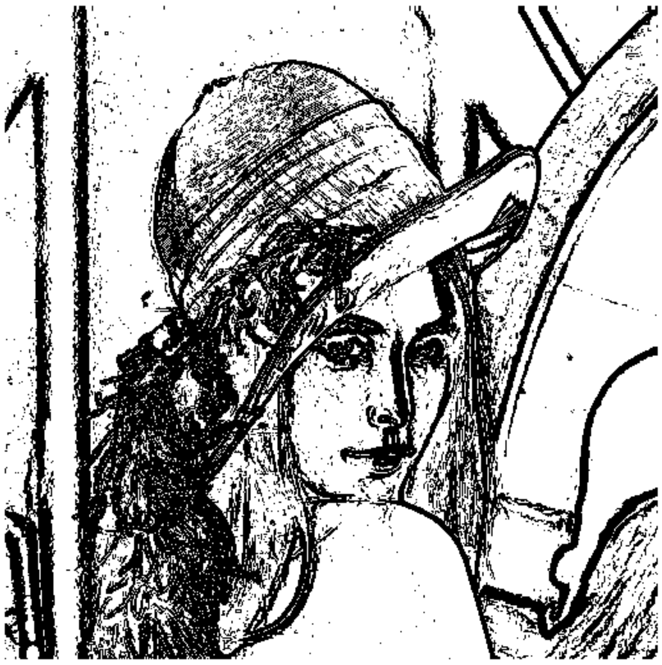
- 使用兩個mask(r1 = [-1, -1, -1, 0, 0, 0, 1, 1, 1], r2 = [-1, 0, 1, -1, 0, 1, -1, 0 1])對鄰域進行計算，將使用r1計算的值寫入Gx、r2計算的值寫入Gy。計算完所有pixel的Gx和Gy後，如果該pixel的$\sqrt{Gx^2 + Gy^2}$比閥值大則將pixel的數值更新為0，若小於閥值則更新為255。
- Prewitt edge detector with threshold = 24



```
# Sobel edge detector
def Sobel(img, threshold):
    img_sob = np.zeros((img.shape[0], img.shape[1]))
    Gx = np.zeros((img.shape[0], img.shape[1]))
    Gy = np.zeros((img.shape[0], img.shape[1]))

    for i in range(1, img.shape[0]-1):
        for j in range(1, img.shape[1]-1):
            Gx[i][j] = (-1)*(img[i-1][j-1][0]+2*img[i-1][j][0]+img[i-1][j+1]
[0])+1*(img[i+1][j-1][0]+2*img[i+1][j][0]+img[i+1][j+1][0])
            Gy[i][j] = (-1)*(img[i-1][j-1][0]+2*img[i][j-1][0]+img[i+1][j-1]
[0])+1*(img[i-1][j+1][0]+2*img[i][j+1][0]+img[i+1][j+1][0])
```

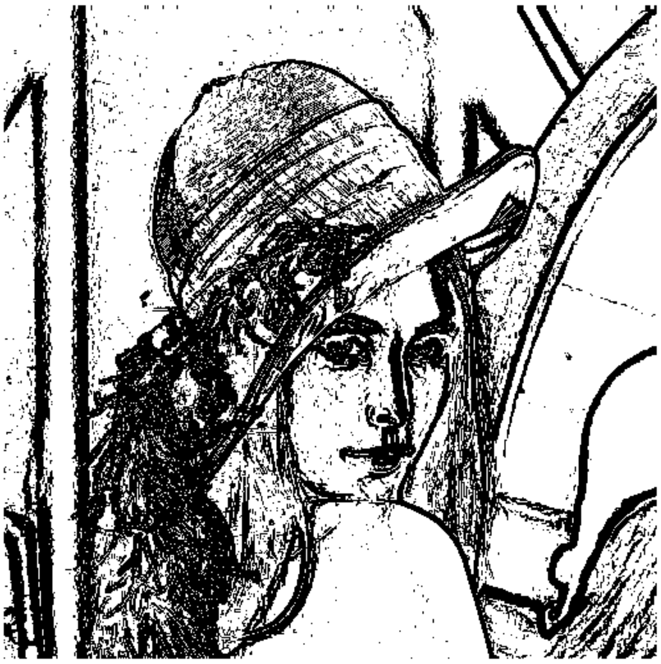
```
    for i in range(img_sob.shape[0]):
        for j in range(img_sob.shape[1]):
            if sqrt(Gx[i][j]**2 + Gy[i][j]**2) < threshold:
                img_sob[i][j] = 255
    return img_sob
```

- 使用兩個mask(r1 = [-1, -2, -1, 0, 0, 0, 1, 2, 1], r2 = [-1, 0, 1, -2, 0, 2, -1, 0 1])對鄰域進行計算，將使用r1計算的值寫入Gx、r2計算的值寫入Gy。計算完所有pixel的Gx和Gy後，如果該pixel的$\sqrt{Gx^2+Gy^2}$比閥值大則將pixel的數值更新為0，若小於閥值則更新為255。
- Sobel edge detector with threshold = 38



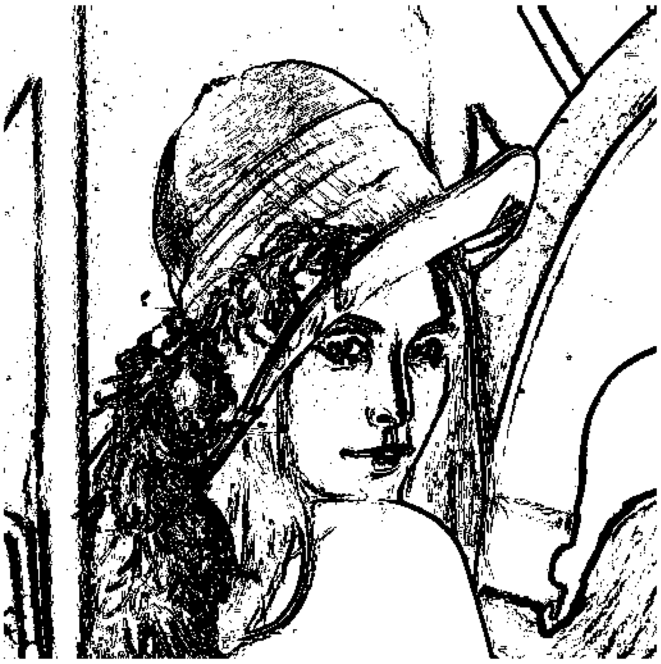
```
# Frei and Chen gradient operator
def FreiandChen(img, threshold):
    img_fac = np.zeros((img.shape[0], img.shape[1]))
    Gx = np.zeros((img.shape[0], img.shape[1]))
    Gy = np.zeros((img.shape[0], img.shape[1]))

    for i in range(1, img.shape[0]-1):
        for j in range(1, img.shape[1]-1):
            Gx[i][j] = (-1)*(img[i-1][j-1][0]+sqrt(2)*img[i-1][j][0]+img[i-1]
[j+1][0])+1*(img[i+1][j-1][0]+sqrt(2)*img[i+1][j][0]+img[i+1][j+1][0])
            Gy[i][j] = (-1)*(img[i-1][j-1][0]+sqrt(2)*img[i][j-1][0]+img[i+1]
[j-1][0])+1*(img[i-1][j+1][0]+sqrt(2)*img[i][j+1][0]+img[i+1][j+1][0])

    for i in range(img_fac.shape[0]):
        for j in range(img_fac.shape[1]):
            if sqrt(Gx[i][j]**2 + Gy[i][j]**2) < threshold:
                img_fac[i][j] = 255
    return img_fac
```

- 使用兩個mask(r1 = [-1, -$\sqrt{2}$, -1, 0, 0, 0, 1, $\sqrt{2}$, 1], r2 = [-1, 0, 1, -$\sqrt{2}$, 0, $\sqrt{2}$, -1, 0 1])對鄰域進行計算，將使用r1計算的值寫入Gx、r2計算的值寫入Gy。計算完所有pixel的Gx和Gy後，如果該pixel的$\sqrt{Gx^2 + Gy^2}$比閥值大則將pixel的數值更新為0，若小於閥值則更新為255。
- Frei and Chen gradient operator with threshold = 30



```
# Kirsch compass operator
def Kirsch(img, threshold):
    img_kir = np.zeros((img.shape[0], img.shape[1]))
    mask = [[-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]]

    for i in range(1, img.shape[0]-1):
        for j in range(1, img.shape[1]-1):
            neighbor = [img[i-1][j-1][0], img[i-1][j][0], img[i-1][j+1][0],
                        img[i][j-1][0], img[i][j][0], img[i][j+1][0],
                        img[i+1][j-1][0], img[i+1][j][0], img[i+1][j+1][0]]
            kmaps = np.zeros(8)

            for k in range(8):
                kmaps[k] = sum([mask[k][tmp] * neighbor[tmp] for tmp in
range(9)])

            img_kir[i][j] = max(kmaps)

    img_kir = np.where(img_kir >= threshold, 0, 255)
    return img_kir
```

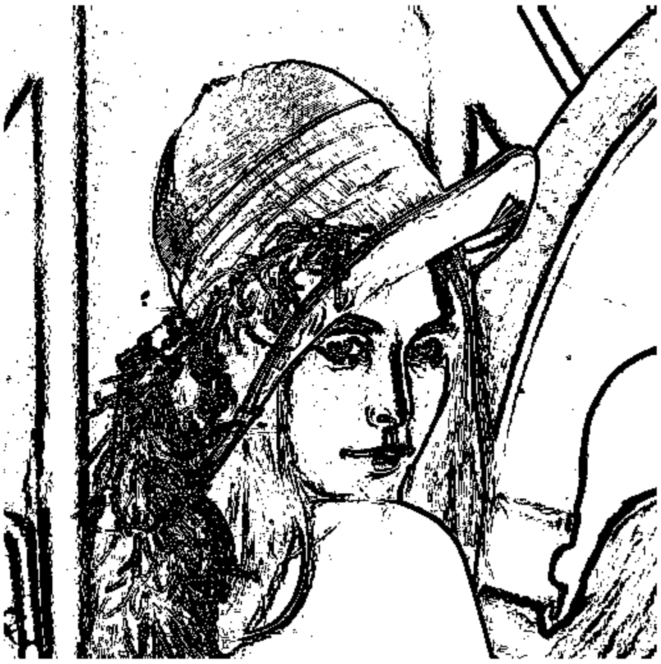
- 使用8張3*3的mask(如程式碼中所示)，將pixel用這8張mask所計算的值寫入kmaps中，並將pixel的數值紀錄為kmaps的最大值。
- 計算完所有pixel後，如果該pixel的數值比閥值大則將pixel的數值更新為0，若小於閥值則更新為255。
- Kirsch compass operator with threshold = 135



```
# Robinson compass operator
def Robinson(img, threshold):
    img_robi = np.zeros((img.shape[0], img.shape[1]))
    mask = [[-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]]

    for i in range(1, img.shape[0]-1):
        for j in range(1, img.shape[1]-1):
            neighbor = [img[i-1][j-1][0], img[i-1][j][0], img[i-1][j+1][0],
                        img[i][j-1][0], img[i][j][0], img[i][j+1][0],
                        img[i+1][j-1][0], img[i+1][j][0], img[i+1][j+1][0]]
            kmaps = np.zeros(8)

            for k in range(8):
                kmaps[k] = sum([mask[k][tmp] * neighbor[tmp] for tmp in
range(9)])
```

```
                img_robi[i][j] = max(kmaps)

        img_robi = np.where(img_robi >= threshold, 0, 255)

        return img_robi
```

- 使用8張3*3的mask(如程式碼中所示)，將pixel用這8張mask所計算的值寫入kmaps中，並將pixel的數值紀錄為kmaps的最大值。
- 計算完所有pixel後，如果該pixel的數值比閥值大則將pixel的數值更新為0，若小於閥值則更新為255。
- Robinson compass operator with threshold = 43



```
# Nevatia and Babu 5×5 operator
def NevatiaandBabu(img, threshold):
    img_nab = np.zeros((img.shape[0], img.shape[1]))
    mask = [[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],
            [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],
            [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],
            [-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],
            [-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],
            [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]]

    for i in range(2, img.shape[0]-2):
        for j in range(2, img.shape[1]-2):
            neighbor = [img[i-2][j-2][0], img[i-2][j-1][0], img[i-2][j][0], img[i-2][j+1][0], img[i-2][j+2][0],
                        img[i-1][j-2][0], img[i-1][j-1][0], img[i-1][j][0], img[i-1][j+1][0], img[i-1][j+2][0],
                        img[i][j-2][0], img[i][j-1][0], img[i][j][0], img[i][j+1][0], img[i][j+2][0],
                        img[i+1][j-2][0], img[i+1][j-1][0], img[i+1][j][0], img[i+1][j+1][0], img[i+1][j+2][0],
                        img[i+2][j-2][0], img[i+2][j-1][0], img[i+2][j][0], img[i+2][j+1][0], img[i+2][j+2][0]]
            kmaps = np.zeros(6)

            for k in range(6):
                kmaps[k] = sum([mask[k][tmp] * neighbor[tmp] for tmp in range(25)])

            img_nab[i][j] = max(kmaps)

    img_nab = np.where(img_nab >= threshold, 0, 255)

    return img_nab
```

- 使用6張5*5的mask(如程式碼中所示)，將pixel用這6張mask所計算的值寫入kmaps中，並將pixel的數值紀錄為kmaps的最大值。
- 計算完所有pixel後，如果該pixel的數值比閥值大則將pixel的數值更新為0，若小於閥值則更新為255。
- Nevatia-Babu 5×5 operator threshold = 12500