Computer Vision HW9 Report
蕭恩慈 / B07902095

The task is to implement the following edge detectors with each of the thresholds:

a. Robert's Operator with threshold 12
b. Prewitt's Edge Detector with threshold 24
c. Sobel's Edge Detector with threshold 38
d. Frei and Chen's Gradient Operator with threshold 30
e. Kirsch's Compass Operator with threshold 135
f. Robinson's Compass Operator with threshold 43
g. Nevatia-Babu 5x5 Operator with threshold 12500

To complete the task, the program is written with Python languages, with numpy and math library for the calculation process in each function, and PIL library for uploading and formatting the image. The image that is going to be implemented on is the original image of lena.bmp. First, upload the original image and set its width and height as the global variable, that way it will be easier to call later in each function. Using the guide from the class slides, (note: I noticed in the slides there are two results from the implementations from Robert's operator with threshold 12 and 30, but since the homework description only lists one, which is with threshold 12, I decided to not do the one with threshold 30.) implement the calculation of each edge detectors to create the functions that can be processed into driver's code.

Code snippet of Robert's Operator:

```
def Roberts(ori, th):
    ber = Image.new("1", ori.size)
    for c in range(w):
        for r in range(h):
            x0 = c
            y0 = r
            x1 = min(c + 1, w - 1)
            y1 = min(r + 1, h - 1)

            r1 = -ori.getpixel((x0, y0)) + ori.getpixel((x1, y1))
            r2 = -ori.getpixel((x1, y0)) + ori.getpixel((x0, y1))

            mag = int(math.sqrt(r1 ** 2 + r2 ** 2))

            if(mag >= th):
                ber.putpixel((c, r), 0)
            else:
                ber.putpixel((c, r), 1)
    return ber
```

Code snippet of Prewitt's Edge Detector:

```
def Prewitt(ori, th):
    prw = Image.new("1", ori.size)
    for c in range(w):
        for r in range(h):
            x0 = max(c - 1, 0)
            y0 = max(r - 1, 0)
            x1 = c
            y1 = r
            x2 = min(c + 1, w - 1)
            y2 = min(r + 1, h - 1)

            p1 = -ori.getpixel((x0, y0)) - ori.getpixel((x1, y0)) - ori.getpixel((x2, y0)) \
                + ori.getpixel((x0, y2)) + ori.getpixel((x1, y2)) + ori.getpixel((x2, y2))

            p2 = -ori.getpixel((x0, y0)) - ori.getpixel((x0, y1)) - ori.getpixel((x0, y2)) \
                + ori.getpixel((x2, y0)) + ori.getpixel((x2, y1)) + ori.getpixel((x2, y2))

            mag = int(math.sqrt(p1 ** 2 + p2 ** 2))

            if(mag >= th):
                prw.putpixel((c, r), 0)
            else:
                prw.putpixel((c, r), 1)
    return prw
```

Code snippet of Sobel's Edge Detector:

```
def Sobel(ori, th):
    sob = Image.new("1", ori.size)
    for c in range(w):
        for r in range(h):
            x0 = max(c - 1, 0)
            y0 = max(r - 1, 0)
            x1 = c
            y1 = r
            x2 = min(c + 1, w - 1)
            y2 = min(r + 1, h - 1)

            p1 = -ori.getpixel((x0, y0)) - 2 * ori.getpixel((x1, y0)) - ori.getpixel((x2, y0)) \
                + ori.getpixel((x0, y2)) + 2 * ori.getpixel((x1, y2)) + ori.getpixel((x2, y2))
            p2 = -ori.getpixel((x0, y0)) - 2 * ori.getpixel((x0, y1)) - ori.getpixel((x0, y2)) \
                + ori.getpixel((x2, y0)) + 2 * ori.getpixel((x2, y1)) + ori.getpixel((x2, y2))

            mag = int(math.sqrt(p1 ** 2 + p2 ** 2))

            if(mag >= th):
                sob.putpixel((c, r), 0)
            else:
                sob.putpixel((c, r), 1)
    return sob
```

Code snippet of Frei and Chen's Gradient Operator:

```
def FreiChen(ori, th):
    fre = Image.new("1", ori.size)
    for c in range(w):
        for r in range(h):
            x0 = max(c - 1, 0)
            y0 = max(r - 1, 0)
            x1 = c
            y1 = r
            x2 = min(c + 1, w - 1)
            y2 = min(r + 1, h - 1)

            p1 = -ori.getpixel((x0, y0)) - math.sqrt(2) * ori.getpixel((x1, y0)) - ori.getpixel((x2, y0)) \
                + ori.getpixel((x0, y2)) + math.sqrt(2) * ori.getpixel((x1, y2)) + ori.getpixel((x2, y2))
            p2 = -ori.getpixel((x0, y0)) - math.sqrt(2) * ori.getpixel((x0, y1)) - ori.getpixel((x0, y2)) \
                + ori.getpixel((x2, y0)) + math.sqrt(2) * ori.getpixel((x2, y1)) + ori.getpixel((x2, y2))

            mag = int(math.sqrt(p1 ** 2 + p2 ** 2))

            if(mag >= th):
                fre.putpixel((c, r), 0)
            else:
                fre.putpixel((c, r), 1)
    return fre
```

Code snippet of Kirsch's Compass Operator:

```
def Kirsch(ori, th):
    kir = Image.new("1", ori.size)
    for c in range(w):
        for r in range(h):
            x0 = max(c - 1, 0)
            y0 = max(r - 1, 0)
            x1 = c
            y1 = r
            x2 = min(c + 1, w - 1)
            y2 = min(r + 1, h - 1)

            k = np.zeros(8)
            k[0] = -3 * ori.getpixel((x0, y0)) - 3 * ori.getpixel((x1, y0)) + 5 * ori.getpixel((x2, y0)) - 3 * ori.getpixel((x0, y1)) \
                + 5 * ori.getpixel((x2, y1)) - 3 * ori.getpixel((x0, y2)) - 3 * ori.getpixel((x1, y2)) + 5 * ori.getpixel((x2, y2))
            k[1] = -3 * ori.getpixel((x0, y0)) + 5 * ori.getpixel((x1, y0)) + 5 * ori.getpixel((x2, y0)) - 3 * ori.getpixel((x0, y1)) \
                + 5 * ori.getpixel((x2, y1)) - 3 * ori.getpixel((x0, y2)) - 3 * ori.getpixel((x1, y2)) - 3 * ori.getpixel((x2, y2))
            k[2] = 5 * ori.getpixel((x0, y0)) + 5 * ori.getpixel((x1, y0)) + 5 * ori.getpixel((x2, y0)) - 3 * ori.getpixel((x0, y1)) \
                - 3 * ori.getpixel((x2, y1)) - 3 * ori.getpixel((x0, y2)) - 3 * ori.getpixel((x1, y2)) - 3 * ori.getpixel((x2, y2))
            k[3] = 5 * ori.getpixel((x0, y0)) + 5 * ori.getpixel((x1, y0)) - 3 * ori.getpixel((x2, y0)) + 5 * ori.getpixel((x0, y1)) \
                - 3 * ori.getpixel((x2, y1)) - 3 * ori.getpixel((x0, y2)) - 3 * ori.getpixel((x1, y2)) - 3 * ori.getpixel((x2, y2))
            k[4] = 5 * ori.getpixel((x0, y0)) - 3 * ori.getpixel((x1, y0)) - 3 * ori.getpixel((x2, y0)) + 5 * ori.getpixel((x0, y1)) \
                - 3 * ori.getpixel((x2, y1)) + 5 * ori.getpixel((x0, y2)) - 3 * ori.getpixel((x1, y2)) - 3 * ori.getpixel((x2, y2))
            k[5] = -3 * ori.getpixel((x0, y0)) - 3 * ori.getpixel((x1, y0)) - 3 * ori.getpixel((x2, y0)) + 5 * ori.getpixel((x0, y1)) \
                - 3 * ori.getpixel((x2, y1)) + 5 * ori.getpixel((x0, y2)) + 5 * ori.getpixel((x1, y2)) - 3 * ori.getpixel((x2, y2))
            k[6] = -3 * ori.getpixel((x0, y0)) - 3 * ori.getpixel((x1, y0)) - 3 * ori.getpixel((x2, y0)) - 3 * ori.getpixel((x0, y1)) \
                - 3 * ori.getpixel((x2, y1)) + 5 * ori.getpixel((x0, y2)) + 5 * ori.getpixel((x1, y2)) + 5 * ori.getpixel((x2, y2))
            k[7] = -3 * ori.getpixel((x0, y0)) - 3 * ori.getpixel((x1, y0)) - 3 * ori.getpixel((x2, y0)) - 3 * ori.getpixel((x0, y1)) \
                + 5 * ori.getpixel((x2, y1)) - 3 * ori.getpixel((x0, y2)) + 5 * ori.getpixel((x1, y2)) + 5 * ori.getpixel((x2, y2))

            mag = max(k)

            if(mag >= th):
                kir.putpixel((c, r), 0)
            else:
                kir.putpixel((c, r), 1)
    return kir
```

Code snippet of Robinson's Compass Operator:

```
def Robinson(ori, th):
    rob = Image.new("1", ori.size)
    for c in range(w):
        for r in range(h):
            x0 = max(c - 1, 0)
            y0 = max(r - 1, 0)
            x1 = c
            y1 = r
            x2 = min(c + 1, w - 1)
            y2 = min(r + 1, h - 1)

            k = np.zeros(8)
            k[0] = -1 * ori.getpixel((x0, y0)) - 2 * ori.getpixel((x0, y1)) - 1 * ori.getpixel((x0, y2)) \
                + 1 * ori.getpixel((x2, y0)) + 2 * ori.getpixel((x2, y1)) + 1 * ori.getpixel((x2, y2))
            k[1] = -1 * ori.getpixel((x0, y1)) - 2 * ori.getpixel((x0, y2)) - 1 * ori.getpixel((x1, y2)) \
                + 1 * ori.getpixel((x1, y0)) + 2 * ori.getpixel((x2, y0)) + 1 * ori.getpixel((x2, y1))
            k[2] = -1 * ori.getpixel((x0, y2)) - 2 * ori.getpixel((x1, y2)) - 1 * ori.getpixel((x2, y2)) \
                + 1 * ori.getpixel((x0, y0)) + 2 * ori.getpixel((x1, y0)) + 1 * ori.getpixel((x2, y0))
            k[3] = -1 * ori.getpixel((x1, y2)) - 2 * ori.getpixel((x2, y2)) - 1 * ori.getpixel((x2, y1)) \
                + 1 * ori.getpixel((x0, y1)) + 2 * ori.getpixel((x0, y0)) + 1 * ori.getpixel((x1, y0))
            k[4] = -1 * ori.getpixel((x2, y0)) - 2 * ori.getpixel((x2, y1)) - 1 * ori.getpixel((x2, y2)) \
                + 1 * ori.getpixel((x0, y0)) + 2 * ori.getpixel((x0, y1)) + 1 * ori.getpixel((x0, y2))
            k[5] = -1 * ori.getpixel((x1, y0)) - 2 * ori.getpixel((x2, y0)) - 1 * ori.getpixel((x2, y1)) \
                + 1 * ori.getpixel((x0, y1)) + 2 * ori.getpixel((x0, y2)) + 1 * ori.getpixel((x1, y2))
            k[6] = -1 * ori.getpixel((x0, y0)) - 2 * ori.getpixel((x1, y0)) - 1 * ori.getpixel((x2, y0)) \
                + 1 * ori.getpixel((x0, y2)) + 2 * ori.getpixel((x1, y2)) + 1 * ori.getpixel((x2, y2))
            k[7] = -1 * ori.getpixel((x0, y1)) - 2 * ori.getpixel((x0, y0)) - 1 * ori.getpixel((x1, y0)) \
                + 1 * ori.getpixel((x1, y2)) + 2 * ori.getpixel((x2, y2)) + 1 * ori.getpixel((x2, y1))

            mag = max(k)
            if(mag >= th):
                rob.putpixel((c, r), 0)
            else:
                rob.putpixel((c, r), 1)
    return rob
```

Code snippet of Nevatia-Babu 5x5 Operator:

```
def NevatiaBabu(ori, th):
    nev = Image.new("1", ori.size)
    for c in range(w):
        for r in range(h):
            x0 = max(c - 2, 0)
            y0 = max(r - 2, 0)
            x1 = max(c - 1, 0)
            y1 = max(r - 1, 0)
            x2 = c
            y2 = r
            x3 = min(c + 1, w - 1)
            y3 = min(r + 1, h - 1)
            x4 = min(c + 2, w - 1)
            y4 = min(r + 2, h - 1)

            neighbors = [
                ori.getpixel((x0, y0)), ori.getpixel((x1, y0)), ori.getpixel((x2, y0)), ori.getpixel((x3, y0)), ori.getpixel((x4, y0)),
                ori.getpixel((x0, y1)), ori.getpixel((x1, y1)), ori.getpixel((x2, y1)), ori.getpixel((x3, y1)), ori.getpixel((x4, y1)),
                ori.getpixel((x0, y2)), ori.getpixel((x1, y2)), ori.getpixel((x2, y2)), ori.getpixel((x3, y2)), ori.getpixel((x4, y2)),
                ori.getpixel((x0, y3)), ori.getpixel((x1, y3)), ori.getpixel((x2, y3)), ori.getpixel((x3, y3)), ori.getpixel((x4, y3)),
                ori.getpixel((x0, y4)), ori.getpixel((x1, y4)), ori.getpixel((x2, y4)), ori.getpixel((x3, y4)), ori.getpixel((x4, y4))]
```

```
            k = np.zeros(6)
            k[0] = (100) * neighbors[0] + (100) * neighbors[1] + (100) * neighbors[2] + (100) * neighbors[3] + (100) * neighbors[4] + \
                   (100) * neighbors[5] + (100) * neighbors[6] + (100) * neighbors[7] + (100) * neighbors[8] + (100) * neighbors[9] + \
                   (0) * neighbors[10] + (0) * neighbors[11] + (0) * neighbors[12] + (0) * neighbors[13] + (0) * neighbors[14] + \
                   (-100) * neighbors[15] + (-100) * neighbors[16] + (-100) * neighbors[17] + (-100) * neighbors[18] + (-100) * neighbors[19] + \
                   (-100) * neighbors[20] + (-100) * neighbors[21] + (-100) * neighbors[22] + (-100) * neighbors[23] + (-100) * neighbors[24]
            k[1] = (100) * neighbors[0] + (100) * neighbors[1] + (100) * neighbors[2] + (100) * neighbors[3] + (100) * neighbors[4] + \
                   (100) * neighbors[5] + (100) * neighbors[6] + (100) * neighbors[7] + (78) * neighbors[8] + (-32) * neighbors[9] + \
                   (100) * neighbors[10] + (92) * neighbors[11] + (0) * neighbors[12] + (-92) * neighbors[13] + (-100) * neighbors[14] + \
                   (32) * neighbors[15] + (-78) * neighbors[16] + (-100) * neighbors[17] + (-100) * neighbors[18] + (-100) * neighbors[19] + \
                   (-100) * neighbors[20] + (-100) * neighbors[21] + (-100) * neighbors[22] + (-100) * neighbors[23] + (-100) * neighbors[24]
            k[2] = (100) * neighbors[0] + (100) * neighbors[1] + (100) * neighbors[2] + (32) * neighbors[3] + (-100) * neighbors[4] + \
                   (100) * neighbors[5] + (100) * neighbors[6] + (92) * neighbors[7] + (-78) * neighbors[8] + (-100) * neighbors[9] + \
                   (100) * neighbors[10] + (100) * neighbors[11] + (0) * neighbors[12] + (-100) * neighbors[13] + (-100) * neighbors[14] + \
                   (100) * neighbors[15] + (78) * neighbors[16] + (-92) * neighbors[17] + (-100) * neighbors[18] + (-100) * neighbors[19] + \
                   (100) * neighbors[20] + (-32) * neighbors[21] + (-100) * neighbors[22] + (-100) * neighbors[23] + (-100) * neighbors[24]
            k[3] = (-100) * neighbors[0] + (-100) * neighbors[1] + (0) * neighbors[2] + (100) * neighbors[3] + (100) * neighbors[4] + \
                   (-100) * neighbors[5] + (-100) * neighbors[6] + (0) * neighbors[7] + (100) * neighbors[8] + (100) * neighbors[9] + \
                   (-100) * neighbors[10] + (-100) * neighbors[11] + (0) * neighbors[12] + (100) * neighbors[13] + (100) * neighbors[14] + \
                   (-100) * neighbors[15] + (-100) * neighbors[16] + (0) * neighbors[17] + (100) * neighbors[18] + (100) * neighbors[19] + \
                   (-100) * neighbors[20] + (-100) * neighbors[21] + (0) * neighbors[22] + (100) * neighbors[23] + (100) * neighbors[24]
            k[4] = (-100) * neighbors[0] + (32) * neighbors[1] + (100) * neighbors[2] + (100) * neighbors[3] + (100) * neighbors[4] + \
                   (-100) * neighbors[5] + (-78) * neighbors[6] + (92) * neighbors[7] + (100) * neighbors[8] + (100) * neighbors[9] + \
                   (-100) * neighbors[10] + (-100) * neighbors[11] + (0) * neighbors[12] + (100) * neighbors[13] + (100) * neighbors[14] + \
                   (-100) * neighbors[15] + (-100) * neighbors[16] + (-92) * neighbors[17] + (78) * neighbors[18] + (100) * neighbors[19] + \
                   (-100) * neighbors[20] + (-100) * neighbors[21] + (-100) * neighbors[22] + (-32) * neighbors[23] + (100) * neighbors[24]
            k[5] = (100) * neighbors[0] + (100) * neighbors[1] + (100) * neighbors[2] + (100) * neighbors[3] + (100) * neighbors[4] + \
                   (-32) * neighbors[5] + (78) * neighbors[6] + (100) * neighbors[7] + (100) * neighbors[8] + (100) * neighbors[9] + \
                   (-100) * neighbors[10] + (-92) * neighbors[11] + (0) * neighbors[12] + (92) * neighbors[13] + (100) * neighbors[14] + \
                   (-100) * neighbors[15] + (-100) * neighbors[16] + (-100) * neighbors[17] + (-78) * neighbors[18] + (32) * neighbors[19] + \
                   (-100) * neighbors[20] + (-100) * neighbors[21] + (-100) * neighbors[22] + (-100) * neighbors[23] + (-100) * neighbors[24]

            mag = max(k)
            if(mag >= th):
                nev.putpixel((c, r), 0)
            else:
                nev.putpixel((c, r), 1)
    return nev
```

Results:

Implementation of Robert's Operator:

Implementation of Prewitt's Edge Detector:

Implementation of Sobel's Edge Detector:

Implementation of Frei and Chen's Gradient Operator:

Implementation of Robinson's Compass Operator:

Implementation of Nevatia-Babu 5x5 Operator: