I. Problem Statement

You are to implement following edge detectors with thresholds :

(a) Robert's Operator: 12

(b) Prewitt's Edge Detector: 24

(c) Sobel's Edge Detector: 38

(d) Frei and Chen's Gradient Operator: 30

(e) Kirsch's Compass Operator: 135

(f) Robinson's Compass Operator: 43

(g) Nevatia-Babu 5x5 Operator: 12500

II. Programming Tools

- Programming language: Python 3.8.5
- Library: Numpy 1.19.1, OpenCV 4.0.1

III. Problem-Solving Process

(a) Robert's Operator

使用 r1 與 r2 為 kernel，對 img 由左到右、由上到下，算出每個 pixel 的 gradient r1 與 gradient r2，並對兩個 gradient 平方和開根號，若計算值大於 threshold，則 pixel value 為 0，否則為 255，這邊使用的 threshold 為 30。

```
def Roberts(img,threshold=30):
    img = img.astype(np.float)
    r1 = np.array([[-1,0],[0,1]],dtype=np.float)
    r2 = np.array([[0,-1],[1,0]],dtype=np.float)
    img_return = np.full_like(img,255,dtype=np.uint8)
    for r in range(img.shape[0]-1):
        for c in range(img.shape[1]-1):
            magnitude_r1 = np.sum(img[r:r+2,c:c+2]*r1)
            magnitude_r2 = np.sum(img[r:r+2,c:c+2]*r2)
            if np.sqrt(magnitude_r1**2 + magnitude_r2**2) >= threshold:
                img_return[r,c] = 0
    return img_return
```

(b) Prewitt's Edge Detector

使用 p1 與 p2 為 kernel，對 img 由左到右、由上到下，算出每個 pixel 的 gradient p1 與 gradient p2，並對兩個 gradient 平方和開根號，若計算值大於 threshold，則 pixel value 為 0，否則為 255，這邊使用的 threshold 為 24。

```python
def Prewitt(img,threshold=24):
    p1 = np.array([[-1,-1,-1],[0,0,0],[1,1,1]],dtype=np.float)
    p2 = p1.T
    img_padding = cv2.copyMakeBorder(img,1,1,1,1,cv2.BORDER_REPLICATE).astype(np.float)
    img_return = np.full_like(img,255,dtype=np.uint8)
    for r in range(img.shape[0]):
        for c in range(img.shape[1]):
            magnitude_p1 = np.sum(img_padding[r:r+3,c:c+3]*p1)
            magnitude_p2 = np.sum(img_padding[r:r+3,c:c+3]*p2)
            if np.sqrt(magnitude_p1**2 + magnitude_p2**2) >= threshold:
                img_return[r,c] = 0
    return img_return
```

(c) Sobel's Edge Detector

使用 s1 與 s2 為 kernel，對 img 由左到右、由上到下，算出每個 pixel 的 gradient s1 與 gradient s2，並對兩個 gradient 平方和開根號，若計算值大於 threshold，則 pixel value 為 0，否則為 255，這邊使用的 threshold 為 38。

```python
def Sobel(img,threshold=38):
    s1 = np.array([[-1,-2,-1],[0,0,0],[1,2,1]],dtype=np.float)
    s2 = s1.T
    img_padding = cv2.copyMakeBorder(img,1,1,1,1,cv2.BORDER_REPLICATE).astype(np.float)
    img_return = np.full_like(img,255,dtype=np.uint8)
    for r in range(img.shape[0]):
        for c in range(img.shape[1]):
            magnitude_s1 = np.sum(img_padding[r:r+3,c:c+3]*s1)
            magnitude_s2 = np.sum(img_padding[r:r+3,c:c+3]*s2)
            if np.sqrt(magnitude_s1**2 + magnitude_s2**2) >= threshold:
                img_return[r,c] = 0
    return img_return
```

(d) Frei and Chen's Gradient Operator

使用 f_c1 與 f_c2 為 kernel，對 img 由左到右、由上到下，算出每個 pixel 的 gradient f_c1 與 gradient f_c2，並對兩個 gradient 平方和開根號，若計算值大於 threshold，則 pixel value 為 0，否則為 255，這邊使用的 threshold 為 30。

```python
def Frei_and_Chen(img,threshold=30):
    f_c1 = np.array([[-1,-np.sqrt(2),-1],[0,0,0],[1,np.sqrt(2),1]],dtype=np.float)
    f_c2 = f_c1.T
    img_padding = cv2.copyMakeBorder(img,1,1,1,1,cv2.BORDER_REPLICATE).astype(np.float)
    img_return = np.full_like(img,255,dtype=np.uint8)
    for r in range(img.shape[0]):
        for c in range(img.shape[1]):
            magnitude_s1 = np.sum(img_padding[r:r+3,c:c+3]*f_c1)
            magnitude_s2 = np.sum(img_padding[r:r+3,c:c+3]*f_c2)
            if np.sqrt(magnitude_s1**2 + magnitude_s2**2) >= threshold:
                img_return[r,c] = 0
    return img_return
```

(e) Kirsch's Compass Operator

使用 k0、k1、k2、k3、k4、k5、k6、k7 為 kernel，對 img 由左到右、由上到下，算出每個 pixel 的 gradient k1 … gradient k7，並對取出 gradient 最大值，若 gradient 值大於 threshold，則 pixel value 為 0，否則為 255，這邊使用的 threshold 為 135。

```
def Kirsch(img,threshold=135):
    k0 = np.array([[-3,-3,5],[-3,0,5],[-3,-3,5]],dtype=np.float)
    k1 = np.array([[-3,5,5],[-3,0,5],[-3,-3,-3]],dtype=np.float)
    k2 = np.array([[5,5,5],[-3,0,-3],[-3,-3,-3]],dtype=np.float)
    k3 = np.array([[5,5,-3],[5,0,-3],[-3,-3,-3]],dtype=np.float)
    k4 = np.array([[5,-3,-3],[5,0,-3],[5,-3,-3]],dtype=np.float)
    k5 = np.array([[-3,-3,-3],[5,0,-3],[5,5,-3]],dtype=np.float)
    k6 = np.array([[-3,-3,-3],[-3,0,-3],[5,5,5]],dtype=np.float)
    k7 = np.array([[-3,-3,-3],[-3,0,5],[-3,5,5]],dtype=np.float)
    img_padding = cv2.copyMakeBorder(img,1,1,1,1,cv2.BORDER_REPLICATE).astype(np.float)
    img_return = np.full_like(img,255,dtype=np.uint8)
    for r in range(img.shape[0]):
        for c in range(img.shape[1]):
            magnitude_k0 = np.sum(img_padding[r:r+3,c:c+3]*k0)
            magnitude_k1 = np.sum(img_padding[r:r+3,c:c+3]*k1)
            magnitude_k2 = np.sum(img_padding[r:r+3,c:c+3]*k2)
            magnitude_k3 = np.sum(img_padding[r:r+3,c:c+3]*k3)
            magnitude_k4 = np.sum(img_padding[r:r+3,c:c+3]*k4)
            magnitude_k5 = np.sum(img_padding[r:r+3,c:c+3]*k5)
            magnitude_k6 = np.sum(img_padding[r:r+3,c:c+3]*k6)
            magnitude_k7 = np.sum(img_padding[r:r+3,c:c+3]*k7)
            if max(magnitude_k0,magnitude_k1,magnitude_k2,magnitude_k3,magnitude_k4,magnitude_k5,magnitude_k6,magnitude_k7) >= threshold:
                img_return[r,c] = 0
    return img_return
```

(f) Robinson's Compass Operator

使用 r0、r1、r2、r3、r4、r5、r6、r7 為 kernel，對 img 由左到右、由上到下，算出每個 pixel 的 gradient r0 … gradient r7，並對取出 gradient 最大值，若 gradient 值大於 threshold，則 pixel value 為 0，否則為 255，這邊使用的 threshold 為 43。

```
def Robinson(img,threshold=43):
    r0 = np.array([[-1,0,1],[-2,0,2],[-1,0,1]],dtype=np.float)
    r1 = np.array([[0,1,2],[-1,0,1],[-2,-1,0]],dtype=np.float)
    r2 = np.array([[1,2,1],[0,0,0],[-1,-2,-1]],dtype=np.float)
    r3 = np.array([[2,1,0],[1,0,-1],[0,-1,-2]],dtype=np.float)
    r4 = np.negative(r0)
    r5 = np.negative(r1)
    r6 = np.negative(r2)
    r7 = np.negative(r3)
    img_padding = cv2.copyMakeBorder(img,1,1,1,1,cv2.BORDER_REPLICATE).astype(np.float)
    img_return = np.full_like(img,255,dtype=np.uint8)
    for r in range(img.shape[0]):
        for c in range(img.shape[1]):
            magnitude_r0 = np.sum(img_padding[r:r+3,c:c+3]*r0)
            magnitude_r1 = np.sum(img_padding[r:r+3,c:c+3]*r1)
            magnitude_r2 = np.sum(img_padding[r:r+3,c:c+3]*r2)
            magnitude_r3 = np.sum(img_padding[r:r+3,c:c+3]*r3)
            magnitude_r4 = np.sum(img_padding[r:r+3,c:c+3]*r4)
            magnitude_r5 = np.sum(img_padding[r:r+3,c:c+3]*r5)
            magnitude_r6 = np.sum(img_padding[r:r+3,c:c+3]*r6)
            magnitude_r7 = np.sum(img_padding[r:r+3,c:c+3]*r7)
            if max(magnitude_r0,magnitude_r1,magnitude_r2,magnitude_r3,magnitude_r4,magnitude_r5,magnitude_r6,magnitude_r7) >= threshold:
                img_return[r,c] = 0
    return img_return
```

(g) Nevatia-Babu 5x5 Operator

使用 n0、n1、n2、n3、n4、n5 為 kernel，對 img 由左到右、由上到下，算出每個 pixel 的 gradient n0 … gradient n5，並對取出 gradient 最大值，若 gradient 值大於 threshold，則 pixel value 為 0，否則為 255，這邊使用的 threshold 為 12500。

```python
def Nevatia_Babu(img,threshold=12500):
    n0 = np.array([[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]],dtype=np.float)
    n1 = np.negative(n0.T)
    n2 = np.array([[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]],dtype=np.float)
    n3 = n2.T
    n4 = np.array([[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]],dtype=np.float)
    n5 = np.negative(n4.T)

    img_padding = cv2.copyMakeBorder(img,2,2,2,2,cv2.BORDER_REPLICATE).astype(np.float)
    img_return = np.full_like(img,255,dtype=np.uint8)
    for r in range(img.shape[0]):
        for c in range(img.shape[1]):
            magnitude_n0 = np.sum(img_padding[r:r+5,c:c+5]*n0)
            magnitude_n1 = np.sum(img_padding[r:r+5,c:c+5]*n1)
            magnitude_n2 = np.sum(img_padding[r:r+5,c:c+5]*n2)
            magnitude_n3 = np.sum(img_padding[r:r+5,c:c+5]*n3)
            magnitude_n4 = np.sum(img_padding[r:r+5,c:c+5]*n4)
            magnitude_n5 = np.sum(img_padding[r:r+5,c:c+5]*n5)
            if max(magnitude_n0,magnitude_n1,magnitude_n2,magnitude_n3,magnitude_n4,magnitude_n5) >= threshold:
                img_return[r,c] = 0
    return img_return
```

IV. Results

(a) Robert's Operator: 12

(b) Prewitt's Edge Detector: 24

(c) Sobel's Edge Detector: 38

(d) Frei and Chen's Gradient Operator: 30

(e) Kirsch's Compass Operator: 135

(f) Robinson's Compass Operator: 43

(g) Nevatia-Babu 5x5 Operator: 12500