Computer Vision - Assignment 9

R09922A04 資工所人工智慧組 黃品硯

[Code]

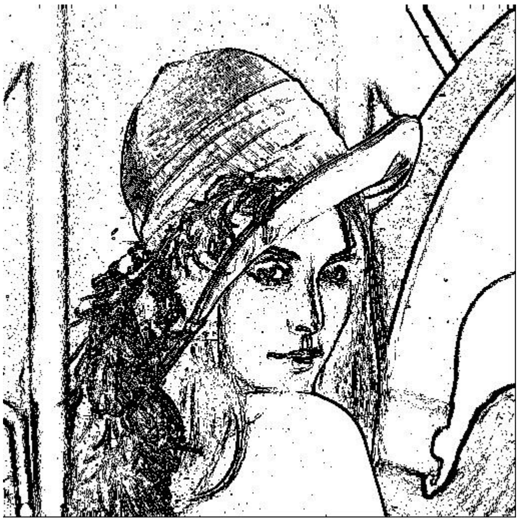
```
def apply_mask(img, y, x, mask):
    r = 0
    for m_y in range(mask_height):
        for m_x in range(mask_width):
            if y + m_y < img_height and x + m_x < img_width:
                r += img[y + m_y, x + m_x] * mask[m_y, m_x]
    return r


def edge_detector1(img, mask1, mask2, threshold):
    for y in range(height):
        for x in range(width):
            r1 = apply_mask(img, y, x, mask1)
            r2 = apply_mask(img, y, x, mask2)
            g = math.sqrt(r1 ** 2 + r2 ** 2)
            if g >= threshold:
                img[y, x] = 0
            else:
                img[y, x] = 255

    return img
```

(a) Robert's Operator, Threshold: 12

Apply mask1 and mask2 on each of the pixels and get the corresponding value r1 and r2. Get value g by the formula: $\sqrt{r1^2 + r2^2}$. If g larger or equal to the threshold then set the pixel to black (value 0), otherwise set it to white (value 255).

[Code]

```
mask1 = np.array([[-1, 0], [0, 1]])
mask2 = np.array([[0, -1], [1, 0]])
edge_detector1(img, mask1, mask2, 12)
```

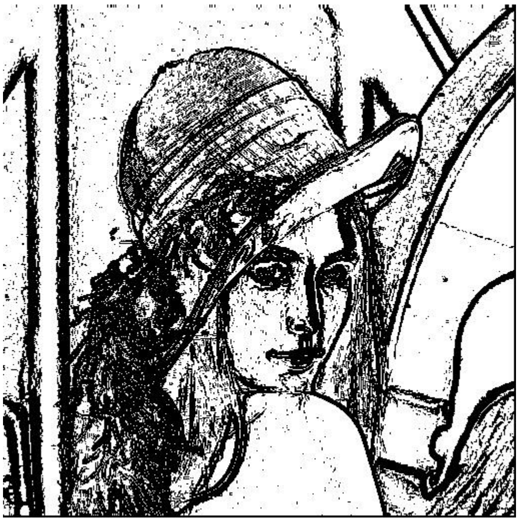


(b) Prewitt's Edge Detector, Threshold: 24

Same procedure with (a) but with a different threshold and masks.

[Code]

```
mask1 = np.array([[-1, -1, -1], [0, 0, 0], [1, 1, 1]])
mask2 = np.array([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]])
edge_detector1(img, mask1, mask2, 24)
```

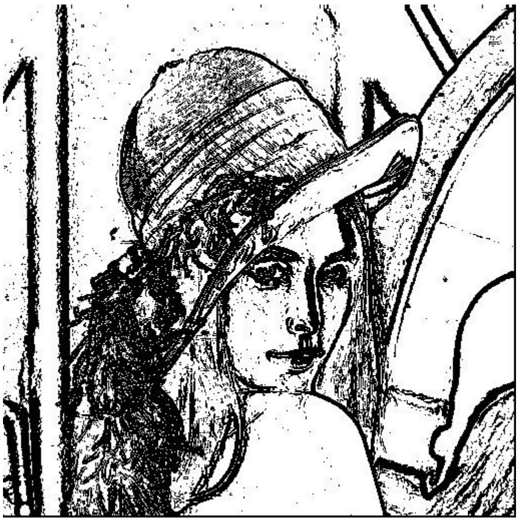


(c) Sobel's Edge Detector, Threshold: 38

Same procedure with (a) but with a different threshold and masks.

[Code]

```
mask1 = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])
mask2 = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
edge_detector1(img, mask1, mask2, 38)
```

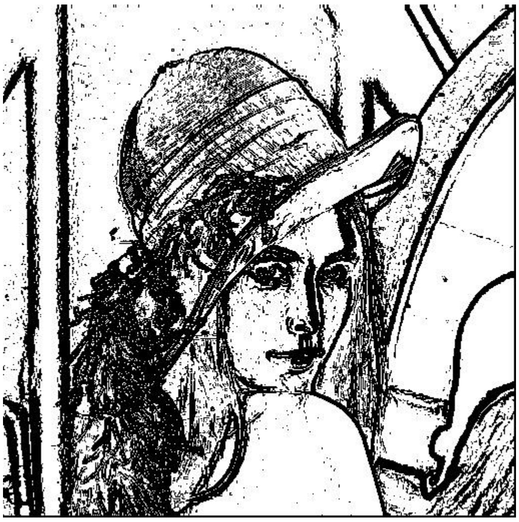


(d) Frei and Chen's Gradient Operator, Threshold: 30

Same procedure with (a) but with a different threshold and masks.

[Code]

```
mask1 = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])
mask2 = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
edge_detector1(img, mask1, mask2, 38)
```

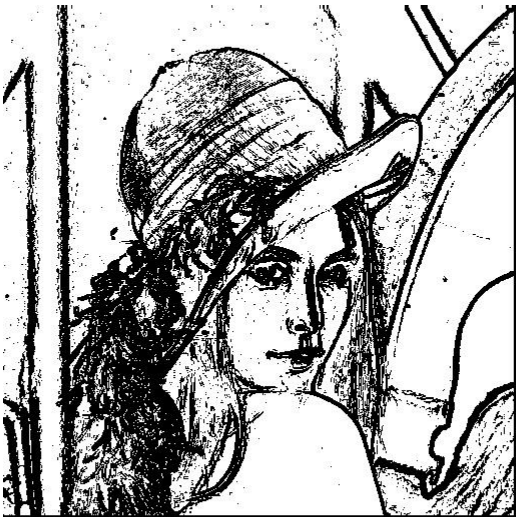


[Code]

```
def edge_detector2(img, masks, threshold):
    for y in range(height):
        for x in range(width):
            all_g = []
            for mask in masks:
                g = apply_mask(img, y, x, mask)
                all_g.append(g)
            max_g = max(all_g)

            if max_g >= threshold:
                img[y, x] = 0
            else:
                img[y, x] = 255

    return img
```

(e) Kirsch's Compass Operator, Threshold: 135

Apply a list of n masks on each of the pixels and get the corresponding value . Get value g by the formula: $Max(m_1, m_2 ... m_n)$. If g larger or equal to the threshold then set the pixel to black (value 0), otherwise set it to white (value 255).

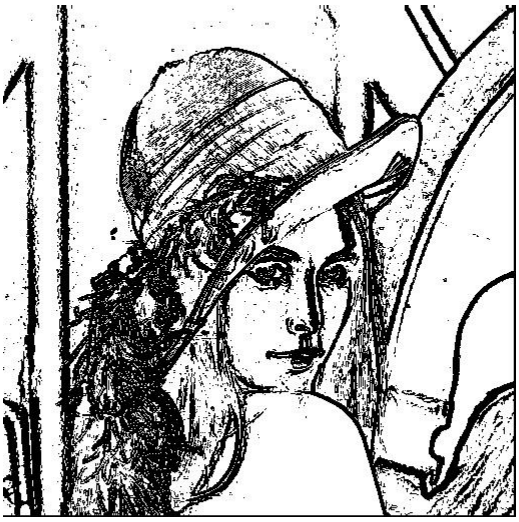


[Code]

```
masks = [
    np.array([[-3, -3, 5], [-3, 0, 5], [-3, -3, 5]]),
    np.array([[-3, 5, 5], [-3, 0, 5], [-3, -3, -3]]),
    np.array([[5, 5, 5], [-3, 0, -3], [-3, -3, -3]]),
    np.array([[5, 5, -3], [5, 0, -3], [-3, -3, -3]]),
    np.array([[5, -3, -3], [5, 0, -3], [5, -3, -3]]),
    np.array([[-3, -3, -3], [5, 0, -3], [5, 5, -3]]),
    np.array([[-3, -3, -3], [-3, 0, -3], [5, 5, 5]]),
    np.array([[-3, -3, -3], [-3, 0, 5], [-3, 5, 5]]),
]
edge_detector2(img, masks, 125)
```

(f) Robinson's Compass Operator, Threshold: 43

Same procedure with (e) but with a different threshold and masks.

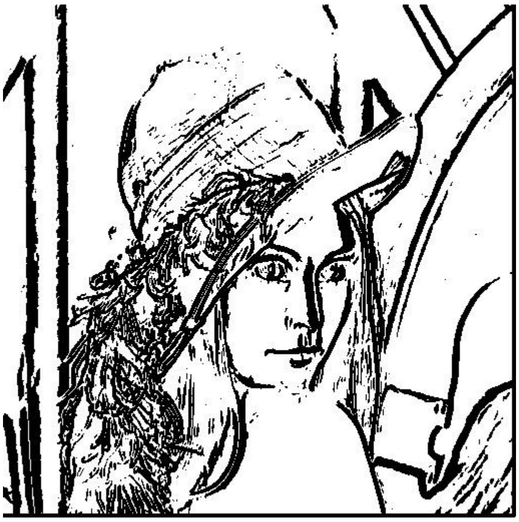


[Code]

```
masks = [
    np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]]),
    np.array([[0, 1, 2], [-1, 0, 1], [-2, -1, -0]]),
    np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]]),
    np.array([[2, 1, 0], [1, 0, -1], [0, -1, -2]]),
    np.array([[1, 0, -1], [2, 0, -2], [1, 0, -1]]),
    np.array([[0, -1, -2], [1, 0, -1], [2, 1, 0]]),
    np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]]),
    np.array([[-2, -1, 0], [-1, 0, 1], [0, 1, 2]]),
]
edge_detector2(img, masks, 43)
```

(g) Nevatia-Babu 5x5 Operator, Threshold: 12500

Same procedure with (a) but with a different threshold and masks.



[Code]

```
masks = [
    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],
        ]
    ),
    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],
        ]
    ),
    np.array(
        [
            [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],
        ]
    ),
    np.array(
        [
            [-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],
        ]
    ),
    np.array(
        [
            [-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],
        ]
    ),
    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],
        ]
    ),
]
edge_detector2(img, masks, 12500)
```