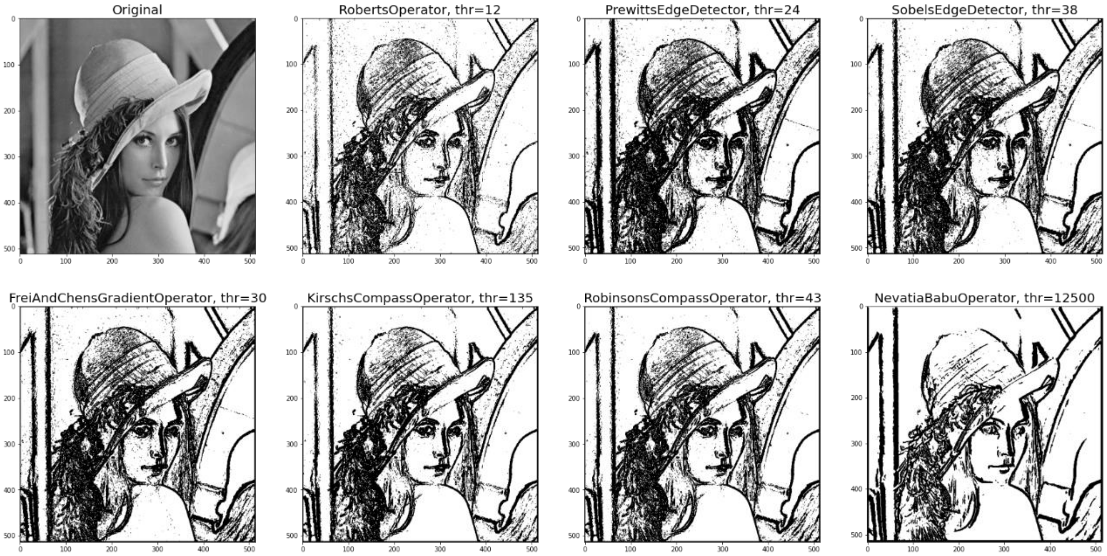
Computer Vision HW9

R08922a27 資工系 人工智慧碩士班 李吉昌

I use python 3.7 to implement all image processing requirements. Reading .bmp file by **PIL**, and then processing through **NumPy** array.

1. Results


Original
RobertsOperator, thr=12
PrewittsEdgeDetector, thr=24
SobelsEdgeDetector, thr=38
FreiAndChensGradientOperator, thr=30
KirschsCompassOperator, thr=135
RobinsonsCompassOperator, thr=43
NevatiaBabuOperator, thr=12500
0
100
200
300
400
500

2. Code fragment

```
def binarize(img, thr):
    img_bin = np.zeros(img.shape)
    img_bin[img <= thr] = 255
    return img_bin


def magnitude(Gx, Gy):
    return np.sqrt(Gx ** 2 + Gy ** 2)


def roberts_operator(img):
    k1 = np.array([
        [1, 0],
        [0, -1]
    ])
    k2 = np.array([
        [0, 1],
        [-1, 0]
    ])
    return magnitude(signal.convolve2d(img, k1), signal.convolve2d(img,
k2))


def prewitts_edge_detector(img):
    k1 = np.array([
        [-1, 0, 1],
        [-1, 0, 1],
        [-1, 0, 1]
    ])
    k2 = np.array([
        [-1, -1, -1],
        [0, 0, 0],
        [1, 1, 1]
    ])
    return magnitude(signal.convolve2d(img, k1), signal.convolve2d(img,
k2))


def sobels_edge_detector(img):
    k1 = np.array([
        [-1, 0, 1],
        [-2, 0, 2],
        [-1, 0, 1]
    ])
    k2 = np.array([
        [-1, -2, -1],
        [0, 0, 0],
        [1, 2, 1]
    ])
    return magnitude(signal.convolve2d(img, k1), signal.convolve2d(img,
k2))


def frei_and_chens_gradient_operator(img):
    k1 = np.array([
        [-1, -np.sqrt(2), -1],
        [0, 0, 0],
        [1, np.sqrt(2), 1]
    ])
    k2 = np.array([
        [-1, 0, 1],
        [-np.sqrt(2), 0, np.sqrt(2)],
        [-1, 0, 1]
    ])
    return magnitude(signal.convolve2d(img, k1), signal.convolve2d(img,
k2))


def kirschs_compass_operator(img):
    k0 = np.array([
        [-3, -3, 5],
        [-3, 0, 5],
        [-3, -3, 5]
    ])
    k1 = np.array([
        [-3, 5, 5],
        [-3, 0, 5],
        [-3, -3, -3]
    ])
    k2 = np.array([
        [5, 5, 5],
        [-3, 0, -3],
        [-3, -3, -3]
    ])
    k3 = np.array([
        [5, 5, -3],
        [5, 0, -3],
        [-3, -3, -3]
    ])
    k4 = np.array([
        [5, -3, -3],
        [5, 0, -3],
        [5, -3, -3]
    ])
    k5 = np.array([
        [-3, -3, -3],
        [5, 0, -3],
        [5, 5, -3]
    ])
    k6 = np.array([
        [-3, -3, -3],
        [-3, 0, -3],
        [5, 5, 5]
    ])
    k7 = np.array([
        [-3, -3, -3],
        [-3, 0, 5],
        [-3, 5, 5]
    ])
    return np.max(np.array(
        [signal.convolve2d(img, k0),
         signal.convolve2d(img, k1),
         signal.convolve2d(img, k2),
         signal.convolve2d(img, k3),
         signal.convolve2d(img, k4),
         signal.convolve2d(img, k5),
         signal.convolve2d(img, k6),
         signal.convolve2d(img, k7)]), axis=0)


def robinsons_compass_operator(img):
    k0 = np.array([
        [-1, 0, 1],
        [-2, 0, 2],
        [-1, 0, 1]
    ])
    k1 = np.array([
        [0, 1, 2],
        [-1, 0, 1],
        [-2, -1, 0]
    ])
    k2 = np.array([
        [1, 2, 1],
        [0, 0, 0],
        [-1, -2, -1]
    ])
    k3 = np.array([
        [2, 1, 0],
        [1, 0, -1],
        [0, -1, -2]
    ])
    k4 = np.array([
        [1, 0, -1],
        [2, 0, -2],
        [1, 0, -1]
    ])
    k5 = np.array([
        [0, -1, -2],
        [1, 0, -1],
        [2, 1, 0]
    ])
    k6 = np.array([
        [-1, -2, -1],
        [0, 0, 0],
        [1, 2, 1]
    ])
    k7 = np.array([
        [-2, -1, 0],
        [-1, 0, 1],
        [0, 1, 2]
    ])
    return np.max(np.array(
        [signal.convolve2d(img, k0),
         signal.convolve2d(img, k1),
         signal.convolve2d(img, k2),
         signal.convolve2d(img, k3),
         signal.convolve2d(img, k4),
         signal.convolve2d(img, k5),
         signal.convolve2d(img, k6),
         signal.convolve2d(img, k7)]), axis=0)


def nevatia_babu_5x5_operator(img):
    k0 = -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],
    ])
    k1 = -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]
    ])
    k2 = -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]
    ])
    k3 = 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]
    ])
    k4 = -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]
    ])
    k5 = -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]
    ])
    return np.max(np.array(
        [signal.convolve2d(img, k0),
         signal.convolve2d(img, k1),
         signal.convolve2d(img, k2),
         signal.convolve2d(img, k3),
         signal.convolve2d(img, k4),
         signal.convolve2d(img, k5)]), axis=0)
```

- ## 3. Brief Description

All the assigned operators' implementation details follow the course's lecture slides. The binarize and magnitude functions are shared for all the assigned operators. The 2d convolution operation is called from the **scipy.signal** package.