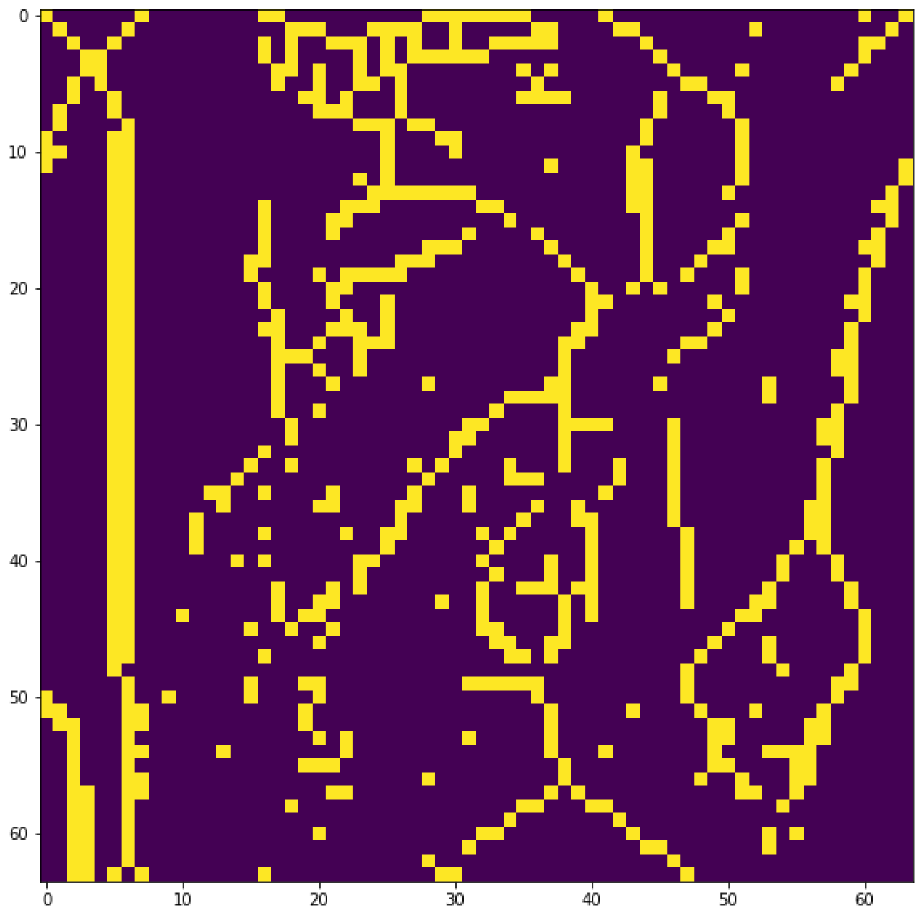
Computer Vision HW7

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



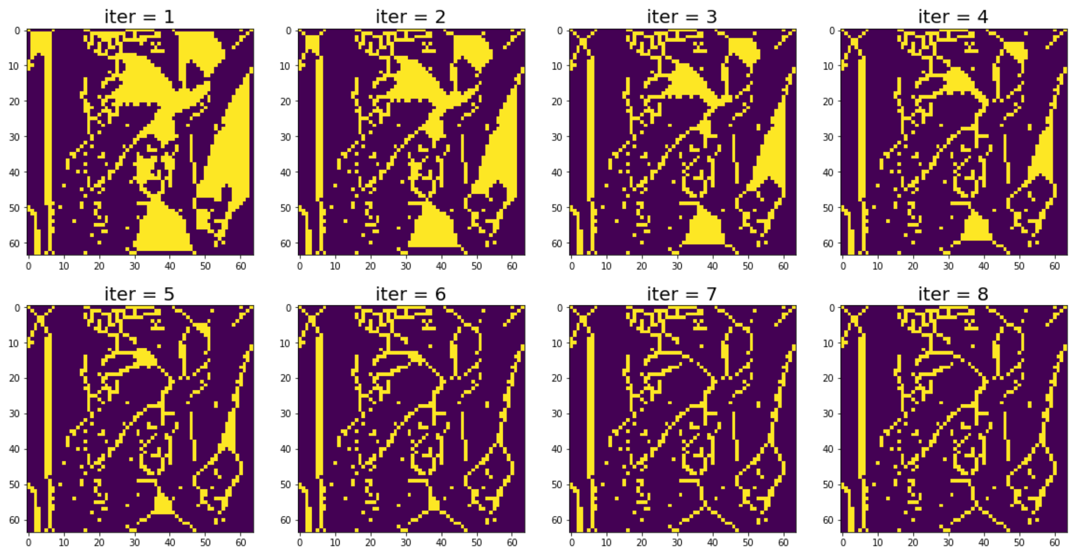
• 2. Code fragment

```
def MarkInteriorBorderPixel(bin_img):
    # for marking interior/border
    # pixel
    def h(c, d):
        if c == d:
            return c
        return 'b'
    def InteriorBorderPixel(bin_img, i, j):
        x1, x2, x3, x4 = 0, 0, 0, 0
        if i == 0:
            if j == 0:
                x1, x4 = bin_img[i, j + 1], bin_img[i + 1, j]
            elif j == bin_img.shape[1] - 1:
                x3, x4 = bin_img[i, j - 1], bin_img[i + 1, j]
            else:
                x1, x3, x4 = bin_img[i, j + 1], bin_img[i,
                                                        j - 1], bin_img[i +
1, j]
        elif i == bin_img.shape[0] - 1:
            if j == 0:
                x1, x2 = bin_img[i, j + 1], bin_img[i - 1, j]
            elif j == bin_img.shape[1] - 1:
                x2, x3 = bin_img[i - 1, j], bin_img[i, j - 1]
            else:
                x1, x2, x3 = bin_img[i, j +
                                     1], bin_img[i - 1, j], bin_img[i, j -
1]
        else:
            if j == 0:
                x1, x2, x4 = bin_img[i, j +
                                     1], bin_img[i - 1, j], bin_img[i + 1,
j]
            elif j == bin_img.shape[1] - 1:
                x2, x3, x4 = bin_img[i - 1, j], bin_img[i,
                                                        j - 1], bin_img[i +
1, j]
            else:
                x1, x2, x3, x4 = bin_img[i, j + 1], bin_img[i -
                                                            1, j],
bin_img[i, j - 1], bin_img[i + 1, j]
        x1 /= 255
        x2 /= 255
        x3 /= 255
        x4 /= 255
        a1 = h(1, x1)
        a2 = h(a1, x2)
        a3 = h(a2, x3)
        a4 = h(a3, x4)
        return 2 if a4 == 'b' else 1

    output = np.zeros(bin_img.shape)
    # 0: background pixel
    # 1: interior pixel
    # 2: border pixel
    for i in range(bin_img.shape[0]):
        for j in range(bin_img.shape[1]):
            if bin_img[i, j] > 0:
                output[i, j] = InteriorBorderPixel(bin_img, i, j)

    return output
```

```
def MarkPairRelationship(bin_img):
    # for marking pair relationship
    def h(a, m):
        if a == m:
            return 1
        return 0

    def PairRelationship(bin_img, i, j):
        x1, x2, x3, x4 = 0, 0, 0, 0
        if i == 0:
            if j == 0:
                x1, x4 = bin_img[i, j + 1], bin_img[i + 1, j]
            elif j == bin_img.shape[1] - 1:
                x3, x4 = bin_img[i, j - 1], bin_img[i + 1, j]
            else:
                x1, x3, x4 = bin_img[i, j + 1], bin_img[i,
                                                        j - 1], bin_img[i +
1, j]
        elif i == bin_img.shape[0] - 1:
            if j == 0:
                x1, x2 = bin_img[i, j + 1], bin_img[i - 1, j]
            elif j == bin_img.shape[1] - 1:
                x2, x3 = bin_img[i - 1, j], bin_img[i, j - 1]
            else:
                x1, x2, x3 = bin_img[i, j +
                                     1], bin_img[i - 1, j], bin_img[i, j -
1]
        else:
            if j == 0:
                x1, x2, x4 = bin_img[i, j +
                                     1], bin_img[i - 1, j], bin_img[i + 1,
j]
            elif j == bin_img.shape[1] - 1:
                x2, x3, x4 = bin_img[i - 1, j], bin_img[i,
                                                        j - 1], bin_img[i +
1, j]
            else:
                x1, x2, x3, x4 = bin_img[i, j + 1], bin_img[i -
                                                            1, j],
bin_img[i, j - 1], bin_img[i + 1, j]

        return 1 if h(x1, 1) + h(x2, 1) + h(x3, 1) + h(x4, 1) >= 1 and
img_ib[i, j] == 2 else 2

    output = np.zeros(bin_img.shape)
    # background pixel: 0
    # p: 1
    # q: 2
    for i in range(bin_img.shape[0]):
        for j in range(bin_img.shape[1]):
            if bin_img[i, j] > 0:
                output[i, j] = PairRelationship(bin_img, i, j)

    return output
```

```
def YokoiConnectivityNumberTransform(bin_img):
    def h(b, c, d, e):
        if b == c and (d != b or e != b):
            return 'q'
        if b == c and (d == b and e == b):
            return 'r'
        return 's'

    def YokoiConnectivityNumber(bin_img, i, j):
        if i == 0:
            if j == 0:
                # top-left
                x7, x2, x6 = 0, 0, 0
                x3, x0, x1 = 0, bin_img[i][j], bin_img[i][j + 1]
                x8, x4, x5 = 0, bin_img[i + 1][j], bin_img[i + 1][j + 1]
            elif j == bin_img.shape[1] - 1:
                # top-right
                x7, x2, x6 = 0, 0, 0
                x3, x0, x1 = bin_img[i][j - 1], bin_img[i][j], 0
                x8, x4, x5 = bin_img[i + 1][j - 1], bin_img[i + 1][j], 0
            else:
                # top-row
                x7, x2, x6 = 0, 0, 0
                x3, x0, x1 = bin_img[i][j -
                                        1], bin_img[i][j], bin_img[i][j +
1]
                x8, x4, x5 = bin_img[i + 1][j -
                                            1], bin_img[i + 1][j],
bin_img[i + 1][j + 1]
        elif i == bin_img.shape[0] - 1:
            if j == 0:
                # bottom-left
                x7, x2, x6 = 0, bin_img[i - 1][j], bin_img[i - 1][j + 1]
                x3, x0, x1 = 0, bin_img[i][j], bin_img[i][j + 1]
                x8, x4, x5 = 0, 0, 0
            elif j == bin_img.shape[1] - 1:
                # bottom-right
                x7, x2, x6 = bin_img[i - 1][j - 1], bin_img[i - 1][j], 0
                x3, x0, x1 = bin_img[i][j - 1], bin_img[i][j], 0
                x8, x4, x5 = 0, 0, 0
            else:
                # bottom-row
                x7, x2, x6 = bin_img[i - 1][j -
                                            1], bin_img[i - 1][j],
bin_img[i - 1][j + 1]
                x3, x0, x1 = bin_img[i][j -
                                        1], bin_img[i][j], bin_img[i][j +
1]
                x8, x4, x5 = 0, 0, 0
        else:
            if j == 0:
                x7, x2, x6 = 0, bin_img[i - 1][j], bin_img[i - 1][j + 1]
                x3, x0, x1 = 0, bin_img[i][j], bin_img[i][j + 1]
                x8, x4, x5 = 0, bin_img[i + 1][j], bin_img[i + 1][j + 1]
            elif j == bin_img.shape[1] - 1:
                x7, x2, x6 = bin_img[i - 1][j - 1], bin_img[i - 1][j], 0
                x3, x0, x1 = bin_img[i][j - 1], bin_img[i][j], 0
                x8, x4, x5 = bin_img[i + 1][j - 1], bin_img[i + 1][j], 0
            else:
                x7, x2, x6 = bin_img[i - 1][j -
                                            1], bin_img[i - 1][j],
bin_img[i - 1][j + 1]
                x3, x0, x1 = bin_img[i][j -
                                        1], bin_img[i][j], bin_img[i][j +
1]
                x8, x4, x5 = bin_img[i + 1][j -
                                            1], bin_img[i + 1][j],
bin_img[i + 1][j + 1]

        a1 = h(x0, x1, x6, x2)
        a2 = h(x0, x2, x7, x3)
        a3 = h(x0, x3, x8, x4)
        a4 = h(x0, x4, x5, x1)

        if a1 == 'r' and a2 == 'r' and a3 == 'r' and a4 == 'r':
            return 5
        else:
            return sum(np.array([a1, a2, a3, a4]) == 'q')

    output = np.zeros(bin_img.shape)

    # compute and output Yokoi Connectivity Number ...
    for i in range(bin_img.shape[0]):
        for j in range(bin_img.shape[1]):
            if bin_img[i, j] > 0:
                output[i, j] = YokoiConnectivityNumber(bin_img, i, j)

    return output
```

- ## 3. Brief Description



All the assigned operators ' implementation details follow the course's lecture slides. Each transformation used 4-connected, and their h function has been shown in the code fragment part.

Firstly, the preprocessing processes, downsampling image from 512x512 to 64x64, and the binarization at the threshold 128 were conducted. Then, the image would be processed by the three-step operations iteratively.