# #Assignment VI

R08521609 王澤庠

First of all, I converted the lena.bmp to binary format by implementing *ConvertToBinary function* with threshold equal to 128, and this function is defined as followed:

```
def ConvertToBinary(originImg):
    binaryImage = Image.new('1', originImg.size)

    for c in range(originImg.size[0]):
        for r in range(originImg.size[1]):
            originalPixel = originImg.getpixel((c, r))
            if (originalPixel >= 128):
                binaryImage.putpixel((c, r), 1)
            else:
                binaryImage.putpixel((c, r), 0)
    return binaryImage
```

Afterwards, a *downsampling function* is used to down sample the binary image to size = (64,64), by using 8x8 blocks as a unit, and take the topmost-left pixel as the downsampled data. The detailed of this function is defined as follow:

```
def downsampling(originImg, factor):

    downsamplingImage = Image.new('1', (64,64))
    for c in range(0, originImg.size[0], factor):
        for r in range(0, originImg.size[1], factor):
            # take the topmost-left pixel as the downsampled data
            downsamplingImage.putpixel((int(c/factor), int(r/factor)), orig
inImg.getpixel((c, r)))

    return downsamplingImage
```

Then after the preprocessing of binary image, I defined a *neighborhoodPixels function* to get set of neighborhood pixels which correspond to each position in the downsampling image:

```
def neighborhoodPixels(originImg, curPos):
    #define the neighborhood array size
    nbPixs = np.zeros(9)
    #current position in the image
    x, y = curPos

    for dx in range(3):
        for dy in range(3):
            #calculating the position x,y in the image
            posX = x + (dx - 1)
            posY = y + (dy - 1)
            # Check if the pixel is out of boundary
            if ((0 <= posX < originImg.size[0]) and (0 <= posY < originImg.
size[1])):
                # store the position in neighborhood array
                nbPixs[3 * dy + dx] = originImg.getpixel((posX, posY))
            else:
                nbPixs[3 * dy + dx] = 0
    return nbPixs
```

Then I defined *hFunc function* to classify every 2x2 region in 3x3 neighborhood pixels into types “q”, “r” or “s”:

```
def hFunc(b, c, d, e):

    if ((b == c) and (b != d or b != e)):
        return 'q'
    if ((b == c) and (b == d and b == e)):
        return 'r'
    if (b != c):
```

```
        return 's'
```

and *fFunc* is used to determine whether this 3x3 neighborhood pixels is 5,4,3,2,1 or 0.

```
def fFunc(a1, a2, a3, a4):

    if ([a1, a2, a3, a4].count('r') == 4):
        # Return label 5 (interior)
        return 5
    else:
        # Return count of 'q'
        return [a1, a2, a3, a4].count('q')
```

Below is the implementation of Yokoi connectivity number, which covers the whole process that I addressed above, and then it saves the result as a 2D array *Yokoi_arr*:

```
def YokoiConnectivityNumber(originImg):

    Yokoi_init_list = [[ " " for x in range(originImg.size[0])] for y in ra
nge(originImg.size[1])]
    Yokoi_arr = np.array(Yokoi_init_list) #defined a 2d array to store the
result

    for c in range(originImg.size[0]):
        for r in range(originImg.size[1]):
            if (originImg.getpixel((c, r)) != 0):
                # Get neighborhood pixel values.
                nbPixs = neighborhoodPixels(originImg, (c, r))
                Yokoi_arr[r, c] = fFunc(
                    hFunc(nbPixs[4], nbPixs[5], nbPixs[2], nbPixs[1]),
                    hFunc(nbPixs[4], nbPixs[1], nbPixs[0], nbPixs[3]),
                    hFunc(nbPixs[4], nbPixs[3], nbPixs[6], nbPixs[7]),
                    hFunc(nbPixs[4], nbPixs[7], nbPixs[8], nbPixs[5]))
            else:
                Yokoi_arr[r, c] = ' '

    return Yokoi_arr
```

Implementation of main:

```
if __name__ == '__main__':

    originImg = Image.open('lena.bmp')
    # Get binary image.
    binaryImg = ConvertToBinary(originImg)
```

```python
    binaryImg.save('binary.bmp')

    # Get downsampling image.
    dsaImg = downsampling(binaryImg, 8)
    dsaImg.save('downsampling.bmp')

    # Get Yokoi Connectivity Number
    Yokoi_arr = YokoiConnectivityNumber(dsaImg)

    with open('Yokoi.txt', "w") as txt_file:
        for line in Yokoi_arr:
            txt_file.write("".join(line) + "\n") # works with any number of elements in a line
```

Result in Yokoi.txt: (shown in next page)

```
11111111        1211111111111122322221   111111111111       0  0
15555551         115555555511 2 11  11   1155555555511        0
15555551        1 2115555112  21112221    155555555551      21
15555551        1 2 155112 22221511       1555555555511     1
15555551         22 2112 22    121 0 0    15555555555511   0
15555551         1  2  21 2     1   1     15555555555551  0
15555551           12 1  121111    1321   155555555555511
15111551           1322 1155551111        155555555555551
111 1551            1  121555555511       155555555555511
11  1551                 21155555511      15511155555511
21  1551                 2 15555555111    1551 11555511
1   1551                 2 155555555511   1551  115551         1
    1551               1121155555555551   1551   15511        12
    1551               15555555555555511  1551   1111        111
    1551        1     2221155555555555511 1151    11        1151
    1551        2    22 1 1555555555555511 151  11111       1551
    1551        2    1   11555555555555551 151 115551      11551
    1551        2       11555555555555555111511155511     115551
    1551       12      11555555555555555555555555551      155551
    1551       11   0 221555555555555555555555555112     1155551
    1551       111   22 15555555555555555555555551 1     1555551
    1551       1511  1 125112111112111555555555111      11555551
    1551       15521  1 121 1 11  1  15555555111  0     15555551
    1551       1151  132 2          1155555111   0     115555551
    1551        151 0  322         115555111  121      155555551
    1551        1221   2           1555551   131      1155555551
    1551         2  0  1          115555511   1       1155555551
    1551         2   0      0    1155555551  0       1 155555551
    1551         2              11555555551          21155555551
    1551         1  0          115555555551          15555555551
    1551          1           11511115555521  1     115555555551
    1551        1 1          11111  1155511   2     155555555551
    1551       131           111     15111    2     155555555551
    1551      121 0        1121   1 111   1   2    1155555555551
    1551      11           111 1  221 11  1   2    1555555555551
    1551    12  0    1     21 121  11 1111    2    1555555555551
    1551     1      12    22  15111111551     2   11555555555551
    1551   1              2   1555551115511   1   15555555555551
    1551   2    0     0  22  12555551 15551    1  15555555555551
    1551   1             1    1555511 11511    2 115555555555551
    1551      0 0      21     155551 1 151     2 155555555555551
    1551               2      15555112 151     2 155555555555551
    1551         1   1 1     1155555511111     2 155555555555551
    1551         2  22       111511111212      21155555555555551
    1551  0      1 12          151    2 1      15555555111555551
    1551       0  0  0         1111  121       155555551 1555551
    1551            0           11111111       155555551 1555551
    1551        0                115551        155555551 1555511
    1551                          15551        211111111 155511
    11521      1   12          122155511       2     11 115511
1    151 0     1    1            155555111     2111     15511
22   1511          1             15555555111   155111   1511
 22  1511          1             15555555551   155551  1151
  2  151            0 1        11155555555511  155511  1511
  2  1521    0        1        155555555555511 15551 12151
  2  151           121         155555555555551 155511 1551
  2  1511                   0  155555555555551 115551 1511
  21 1511            11        155555555555551  111111151
  11 151          0           11555555555555511    111511
  11 151                      15555555555555551      151
  11 151            0        115555555555555551      211
  11 151                     1155555555555555511     1
  11 151                    0 155555555555555551
  11 111        0            1211111111111111111
```