**#Assignment VII**

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))

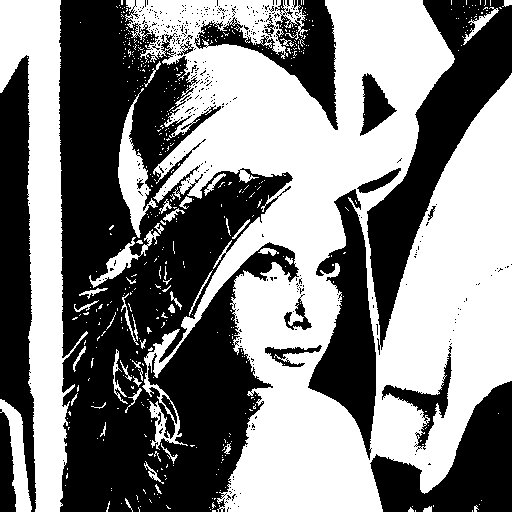
            if (originalPixel >= 128):

                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)), originImg.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 range(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)

    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)

