Computer Vision Homework 6

B03902042 宋子維

Description

In this homework, we are asked to generate Yokoi connectivity number of a binary image in *Figure 1-1*. First downsample the image from 512 x 512 to 64 x 64 by using 8 x 8 block as a unit, and taking the topmost-left pixel as downsample data. Then generate Yokoi connectivity number with respect to the downsampling image.

Figure 1-1: Binary image.

The result is shown in the last page or you can find it in **yokoi.txt**.

Programming

I use python to implement the algorithms. There is one python program, namely, **Yokoi.py**, where I use **pillow** to process basic image I/O. In the program, there are some basic functions:

1. `PIL.Image.open(img)`: load the image `img` and return a pillow **Image** object.
2. `pix = Image.load()`: return the **PixelAccess** object of **Image** object to `pix`, which offers us to use `pix[x, y]` to access the pixel value at position (x, y).
3. `Image.size`: pair (width, height) of **Image** object.
4. `sumTuple((a, b), (c, d))`: return the tuple (a+c, b+d).

5. `product(range(a), range(b))`: return the list of cartesian product of set $\{0, 1, \cdots, a-1\}$ and set $\{0, 1, \cdots, b-1\}$.

The usage of **Yokoi.py** is

```
python3 Yokoi.py IMG_IN FILE_OUT
```

The program will generate Yokoi connectivity number with respect to downsampling image of **IMG_IN** and write the result into **FILE_OUT**.

Algorithm

First, we need to downsample the image. Since we take 8 x 8 block as a unit and take the topmost-left pixel as downsample data, it is clear that the value at (x, y) after downsampling is exactly the the value at $(8x, 8y)$ before downsampling. The following function returns a dictionary object, where key is the position and value is the downsample data corresponding to the key, and the size after downsampling.

```
def downsample(pix, size, BLOCK_LEN):
    # BLOCK_LEN is 8 and size is a 2-tuple (512, 512) in this assignment
    width, height = int(size[0] / BLOCK_LEN), int(size[1] / BLOCK_LEN)

    calPos = lambda x: (x[0]*BLOCK_LEN, x[1]*BLOCK_LEN)
    M = {_: pix[calPos(_)] for _ in product(range(width), range(height))}

    return M, (width, height)
```

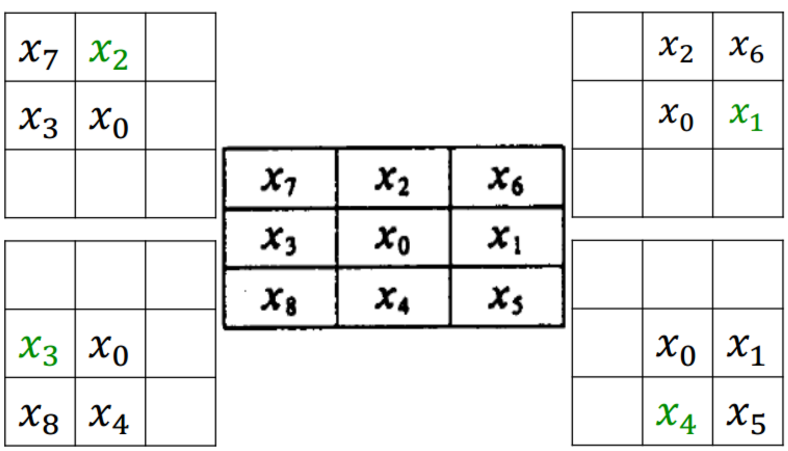

Figure 2-1: 3x3 block and 4 corners of it.

Now take a look at the algorithm of Yokoi connectivity number. To do this, first iterate all pixel (x, y) in the downsample data. If (x, y) is white, and then find the translation at (x, y) with respect to 3×3 block. After it, apply function `h()` to each corner and apply function `f()` on the return value of `h()` of 4 corners where `f()` and `h()` are given as:

$$h(b, c, d, e) = \begin{cases} q, & \text{if } b = c \text{ and } (d \neq b \text{ or } e \neq b) \\ r, & \text{if } b = c \text{ and } (d = b \text{ and } e = b) \\ s, & \text{if } b \neq c \end{cases}$$

$$f(a_1, a_2, a_3, a_4) = \begin{cases} 5, & \text{if } a_1 = a_2 = a_3 = a_4 = r \\ \#(a_k | a_k = q), & \text{otherwise} \end{cases}$$

The we can get the following code. Function `translation()` returns the translation at `cur` with respect to 3×3 block. Function `f()` and `h()` exactly do the same thing with the definition above, but take different arguments. Function `Yokoi()` takes the downsample data and its size as arguments. Then iterate over each data in it. For each pixel, First do the translation at it, then calculate the Yokoi connectivity number on it. The detailed descriptions of parameters are in the comments of code.

```
def translation(M, cur):
    ret = {}

    for _ in product(range(-1, 2), repeat=2):
        pos = sumTuple(cur, _)
        try:
            ret[_] = M[pos]
        except:
            # If pos out of the boundary of M, assign BLACK to the translation.
            ret[_] = BLACK

    return ret

def h(M, B):
    # M is the 3x3 block and B is one of the corner block in Figure 2-1
    # We can simply regard M[B[0]] as b, M[B[1]] as c, M[B[2]] as d, and M[B[3]] as e
    if M[B[0]] != M[B[1]]:
        return s
    elif M[B[0]] == M[B[1]] == M[B[2]] == M[B[3]]:
        return r
    else:
        return q

def f(a):
    # a is a list [a_1, a_2, a_3, a_4]
    if a[0] == a[1] == a[2] == a[3] == r:
        return 5
    else:
        return a.count(q)

def Yokoi(M, size):
    ret = {}
    width, height = size
    for cur in product(range(width), range(height)):
        if M[cur] == WHITE:
            T = translation(M, cur)
            # BLOCK contains 4 corner block in Figure 2-1
            ret[cur] = f([h(T, B) for B in BLOCK])
        else:
            ret[cur] = ' '
    return ret
```

The result is shown as follows or in **yokoi.txt**.

```
11111111        12111111111122322221     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  151111111551    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
```