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 **result.pdf** (A4 page) or in **yokoi.txt** (text file).

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)
```

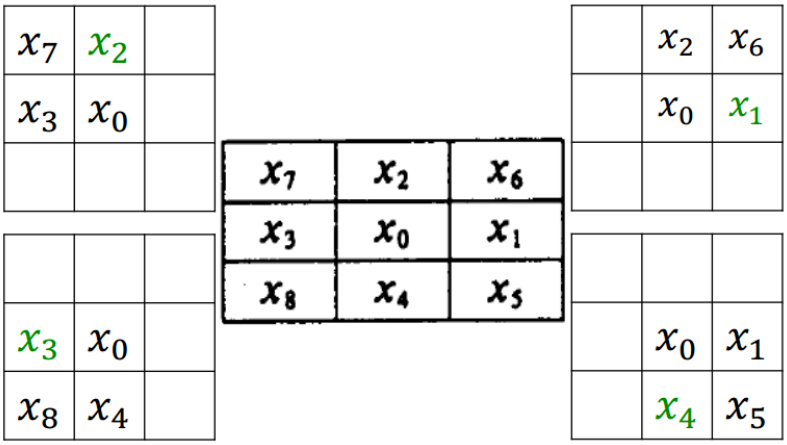

Figure 2-1: 3x3 block and 4 corners of it.

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 in **result.pdf** (A4 page) or in **yokoi.txt** (text file).