R13922136 周洛君

Computer Vision HW6: Report

The snapshot of my code result is as the following image:

```
11111151        22111111111122211111     155555111111       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      1555112 151      2 155555555555551
    1551         1   1 1     1155555511111     2 155555555555551
    1551         2  22       111511111212      21155555555555551
    1551  0      1 12          151    2 1      1555555511555551
    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
```

The main code structure of the .ipynb file consists of the following parts (each part is a code cell, with detailed description in the preceding markdown cell):

1. Import the libraries
2. The 2 primitive functions that we used to count the Yokoi connectivity number
   a. h
   b. f
3. Read in the image by cv2.imread() as usual
4. downsampling
5. Calculate the Yokoi connectivity number by first obtaining the value of each index in the neighborhood, then plug in to the 2 primitive functions we have defined in step 2, print after each row is processed.

A really detailed explanation for each step is covered in the markdown cells, so I'll just paste the screenshots and give brief explanation here.

For the primitive function h, our aim is to determine whether a three-pixel corner neighborhood is connected in a particular way, so we use three symbols 'q', 'r', 's' to represent the 3 situations, and check to return which symbol by definition:

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

✓ 0.0s
```
[1]

[1] The definition written in mathematical equations, and examples of each returning value is contained in the corresponding markdown cell.

For the primitive function f, we just implement it as definition:

primitive function f

f counts the number of arguments having a paricular value.

The symbol (label) 5 stands for interior pixels, so we output 5 if all the arguments are r.

For other cases, the connectivity number is the number of times a 4-connected neighbor has the same value but the corresponding 3-pixel corner neighborhood does not, if we represent this in a formula, it would be:

$$f(a_1, a_2, a_3, a_4) = \#\{a_k | a_k = q\}$$

```python
def f(a_1, a_2, a_3, a_4):
    if a_1 == a_2 == a_3 == a_4 == 'r':
        return 5
    else:
        connectivity_number = 0
        for a_i in [a_1, a_2, a_3, a_4]:
            if a_i == 'q':
                connectivity_number += 1
        return connectivity_number

```

[27] ✓ 0.0s Python

After the 2 primitive functions are defined, we do the downsampling part:

Downsampling lena from 512x512 to 64x64

1. binarize lena image as HW2
2. use 8x8 blocks as a unit
3. take top-most left pixel as downsampled data

Note: Result is a 64x64 matrix

We start by binarizing the lena image as usual:

Step 1: Binarize the image

```python
binarized_img = np.zeros(img.shape, np.int8)
for i in range(img.shape[0]):
    for j in range(img.shape[1]):
        if img[i][j] >= 128:
            binarized_img[i][j] = 1
```

[29] ✓ 0.2s

Then the downsample process is just assigning each downsampled pixel by the corresponding topmost left pixel of the 8x8 block in the original binarized image:

```python
downsampled_img  = np.zeros((64, 64), np.int8)
for i in range(0, 64):
    for j in range(0, 64):
        downsampled_img[i][j] = binarized_img[i*8][j*8]
```

[30] ✓ 0.0s

Finally we calculate the Yokoi connectivity number, to print out the results, we print after each row is proceeded, so we create an empty string after each iteration of i:

```
for i in range(downsampled_img.shape[0]):
    line = ''
```

Then we get the values of each pixel in the neighborhood, with the indexing like below:

The indexing for pixels in a 3x3 neighborhood is:

$$\begin{bmatrix} x_7 & x_2 & x_6 \\ x_3 & x_0 & x_1 \\ x_8 & x_4 & x_5 \end{bmatrix}$$

To obtain these values, cases that may contain indexes out of the image should be taken into consideration:

Here we have to define different cases, since for pixels on the boundary, some of their neighbors do not exist. We classify the situation into 9 cases:

- top
    1. top-left
    2. top-right
    3. top
- bottom
    4. bottom-left
    5. bottom-right
    6. bottom

7. left
8. right
9. center

For foreground pixels, we define the values by cases differently, and we set the nonexist pixels' values to zero (due to the report length, here I only present the first 3 cases that represents the top row):

```
# top
if i == 0:
    # case 1: top-left
    if j == 0:
        x_7, x_2, x_6 = 0, 0, 0
        x_3, x_0, x_1 = 0, downsampled_img[i][j], downsampled_img[i][j+1]
        x_8, x_4, x_5 = 0, downsampled_img[i+1][j], downsampled_img[i+1][j+1]

    # case 2: top-right
    elif j == downsampled_img.shape[1] - 1:
        x_7, x_2, x_6 = 0, 0, 0
        x_3, x_0, x_1 = downsampled_img[i][j-1], downsampled_img[i][j], 0
        x_8, x_4, x_5 = downsampled_img[i+1][j-1], downsampled_img[i+1][j], 0

    # case 3: top
    else:
        x_7, x_2, x_6 = 0, 0, 0
        x_3, x_0, x_1 = downsampled_img[i][j-1], downsampled_img[i][j], downsampled_img[i][j+1]
        x_8, x_4, x_5 = downsampled_img[i+1][j-1], downsampled_img[i+1][j], downsampled_img[i+1][j+1]
```

```
    a_1 = h(x_0, x_1, x_6, x_2)
    a_2 = h(x_0, x_2, x_7, x_3)
    a_3 = h(x_0, x_3, x_8, x_4)
    a_4 = h(x_0, x_4, x_5, x_1)
    connectivity_number = f(a_1, a_2, a_3, a_4)
    line += str(connectivity_number)
else:
    line += ' '
```

The last part is to call the functions and derive the connectivity number by definition, and concatenate the result to the string of the current row.