Computer Vision HW7: Thinning

R10741015 鄭傑鴻

Oct. 28, 2022

Since some of the operations (i.e., thresholding, down-sampling and Yokoi) are identical to the code I designed in HW6, I’ll use them directly.

Step 1: Thresholding, Down Sampling and Generate Yokoi Connectivity Number

- Description:
  Call the functions implemented in HW6. Please kindly refer to the report of HW6 if you are interested.
- Result:

  Down sampled Yokoi Connectivity Images (Value 0 replaced by ‘_’ to better visualize)**:**

```
11111111________12111111111122322221_____111111111111___________
15555551_________115555555511_2_11__11___1155555555511__________
15555551________1_2115555112__21112221____155555555551______21__
15555551________1_2_155112_22221511_______1555555555511_____1___
15555551_________22_2112_22____121________15555555555511________
15555551_________1__2__21_2_____1___1_____15555555555551________
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_____221555555555555555555555555112_____1155551
____1551_______111___22_15555555555555555555555551_1_____1555551
____1551_______1511__1_125112111121115555555555111______11555551
____1551_______15521__1_121_1_11__1__15555555111________15555551
____1551_______1151__132_2__________1155555111_________115555551
____1551________151____322_________115555111__121______155555551
____1551________1221___2___________1555551___131______1155555551
____1551_________2_____1__________115555511___1_______1155555551
____1551_________2_______________1155555551__________1_155555551
____1551_________2______________11555555551__________21155555551
____1551_________1_____________115555555551__________15555555551
____1551__________1___________11511115555521__1_____115555555551
____1551________1_1__________11111__1155511___2_____155555555551
____1551_______131___________111_____15111____2_____155555555551
____1551______121__________1121___1__111__1___2____1155555555551
____1551______11___________111_1__221_11__1___2____1555555555551
____1551____12_______1_____21_121__11_1111____2____1555555555551
____1551_____1______12____22__151111111551____2___11555555555551
____1551___1______________2___1555551115511___1___15555555555551
____1551___2_____________22__12555551_15551____1__15555555555551
____1551___1_____________1____1555511_11511____2_115555555555551
____1551_______________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_________1_12__________151____2_1______15555555111555551
____1551_______________________1111__121_______155555551_1555551
____1551________________________11111111_______155555551_1555551
____1551_________________________115551________155555551_1555511
____1551__________________________15551________211111111_155511_
____11521______1___12__________122155511_______2_____11_115511__
1____151_______1____1____________155555111_____2111_____15511___
22___1511__________1_____________15555555111___155111___1511____
_22__1511__________1_____________15555555551___155551__1151_____
__2__151______________1________11155555555511__155511__1511_____
__2__1521_____________1________155555555555511_15551_12151______
__2__151___________121_________155555555555551_155511_1551______
__2__1511______________________155555555555551_115551_1511______
__21_1511____________11________155555555555551__111111151_______
__11_151______________________11555555555555511____111511_______
__11_151______________________15555555555555551______151________
__11_151_____________________115555555555555551______211________
__11_151_____________________1155555555555555511_____1__________
__11_151______________________155555555555555551________________
__11_111_____________________1211111111111111111________________
```

Step 2: Implement the Pair Relationship Operator

- Description

  Starting from the Yokoi result in step 1, find the edge pixels (those with Yokoi connectivity number quals 1), check its 4-connected neighbors to decide whether they have edge neighbors. If yes, label it as ‘p’: interesting. For the other scenarios, assign ‘q’: not interesting.

- Algorithm

  Scan through all the pixels:

  If the pixel is an edge:

  If it has edge neighbors (function h, f): assign ‘p’

  Else: assign ‘q’

  If not an edge:

  Assign ‘q’

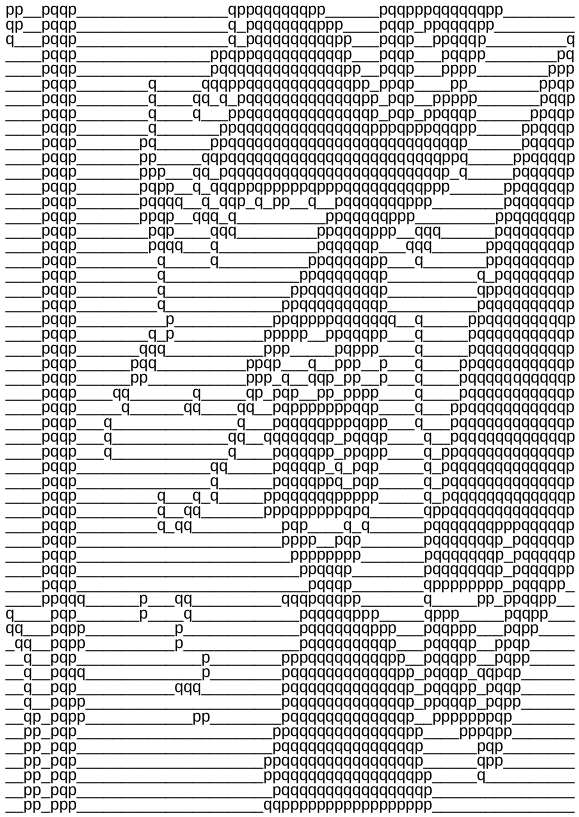
- Code

```python
def pairRelationship(self, array):
    # Input: Yokoi (shape 64,64, integer)
    # Output: Pair Relationship result (shape 64,64, integer)
    def funH(a, m):
        if a == m:
            return 1
        else:
            return 0

    def funF(a1, a2, a3, a4):
        if (a1 + a2 + a3 + a4) < 1:
            return 'q'
        else:
            return 'p'

    ret_array = np.zeros((64,64)).astype(str)
    for i in range(64):
        for j in range(64):
            if array[i][j] == 1:
                x0 = array[i][j]
                x1 = 0 if not self.inRange(i + 0, j + 1) else array[i + 0][j + 1]
                x2 = 0 if not self.inRange(i - 1, j + 0) else array[i - 1][j + 0]
                x3 = 0 if not self.inRange(i + 0, j - 1) else array[i + 0][j - 1]
                x4 = 0 if not self.inRange(i + 1, j + 0) else array[i + 1][j + 0]

                a1 = funH(x0, x1)
                a2 = funH(x0, x2)
                a3 = funH(x0, x3)
                a4 = funH(x0, x4)

                ret_array[i][j] = funF(a1, a2, a3, a4)
            elif array[i][j] == 0:
                ret_array[i][j] = '0'
            else:
                ret_array[i][j] = 'q'
    return ret_array
```

- Result

  Take the result from the first iteration as an example:

```
pppppppp________qqppppppppppqqqqqqqq_____pppppppppppp___________
pqqqqqqp_________ppqqqqqqqqpp_q_pp__pp___ppqqqqqqqqqpp__________
pqqqqqqp________p_qppqqqqppq__qpppqqqp____pqqqqqqqqqqp______qq__
pqqqqqqp________p_q_pqqppq_qqqqpqpp_______pqqqqqqqqqqpp_____q___
pqqqqqqp_________qq_qppq_qq____pqp________pqqqqqqqqqqqpp________
pqqqqqqp_________q__q__qq_q_____q___q_____pqqqqqqqqqqqqp________
pqqqqqqp___________pq_q__pqpppp____qqqq___pqqqqqqqqqqqqpp_______
pqpppqqp___________pqqq_ppqqqqpppp________pqqqqqqqqqqqqqp_______
ppp_pqqp____________q__qqpqqqqqqqpp_______pqqqqqqqqqqqqpp_______
```



Step 3: Implement the Connected Shrink Operator

- Description:

  By using the original binary image and the result from Pair Relationship operator, for those pixels who are marked as interesting (‘p’) by the pair relationship operator, check its neighbors in the original binary image to decide whether it can be eliminated without disconnecting the region.

  Since Connected Shrink is a recursive operator, we must update the result in real time.

- Algorithm:

  Scan through all the pixels:

   If the pixel is marked as interesting (‘p’):

    If it can be eliminated (by using function f and h):

     Update the pixel to background

    Else: Remain unchanged

   Else: Remain unchanged

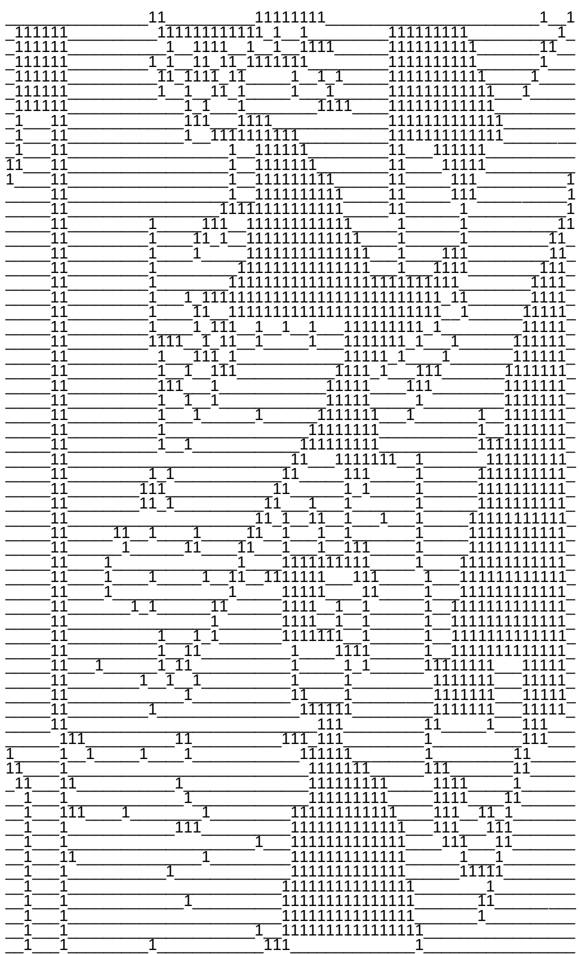
- Code:

```
def connectedShrink(self, pr_array, binds_array):
    # Input: Pair Relationship(shape (64,64), str): '0':background, 'p': interesting, 'q':not interestin
    #        Binary Down-sample array(shape (64,64), int): 0:background, 1:foreground
    # Output: Connected Shrink result(shape (64,64), int): 0:background, 1:foreground
    def funH(b,c,d,e):
        if b == c and (d!=b or e!=b): return 1
        else: return 0
    def funF(a1,a2,a3,a4):
```

```
        if (a1+a2+a3+a4) == 1: # make it background
            self.cnt += 1
            return 0
        else:
            return 1
    ret_array = np.copy(binds_array)
    for i in range(64):
        for j in range(64):
            if pr_array[i][j] == 'p':
                x0 = ret_array[i][j]
                x1 = 0 if not self.inRange(i + 0, j + 1) else ret_array[i + 0][j + 1]
                x2 = 0 if not self.inRange(i - 1, j + 0) else ret_array[i - 1][j + 0]
                x3 = 0 if not self.inRange(i + 0, j - 1) else ret_array[i + 0][j - 1]
                x4 = 0 if not self.inRange(i + 1, j + 0) else ret_array[i + 1][j + 0]
                x5 = 0 if not self.inRange(i + 1, j + 1) else ret_array[i + 1][j + 1]
                x6 = 0 if not self.inRange(i - 1, j + 1) else ret_array[i - 1][j + 1]
                x7 = 0 if not self.inRange(i - 1, j - 1) else ret_array[i - 1][j - 1]
                x8 = 0 if not self.inRange(i + 1, j - 1) else ret_array[i + 1][j - 1]

                # Upper right: [0,0], [0, 1], [-1, 1], [-1, 0]
                a1 = funH(x0, x1, x6, x2)
                # Upper left: [0,0], [-1, 0], [-1, -1], [0, -1]
                a2 = funH(x0, x2, x7, x3)
                # Bottom left: [0,0], [0, -1], [1, -1], [1, 0]
                a3 = funH(x0, x3, x8, x4)
                # Bottom right: [0,0], [1, 0], [1, 1], [0, 1]
                a4 = funH(x0, x4, x5, x1)
                ret_array[i][j] = funF(a1, a2, a3, a4)
    return ret_array
```

- Result:

  Take the result from the first iteration as an example:



Step 4: Repeat the above procedure until no change

- Description:

  Repeat the Yokoi->Pair Relationship->Connected Shrink procedure, until it can't be further skeletonized (7 times later).

- Algorithm:

  Set up a variable 'cnt' to keep track of whether there are changes

  While cnt != 0:

      Repeat the Yokoi->Pair Relationship->Connected Shrink procedure

  Show the final result

- Code:

```python
def thinning(self):

    ans = np.copy(self.binds)
    while(self.cnt != 0):
        print('Iteration {}'.format(self.iter))
        # Initialize
        self.cnt = 0

        # Execute
        yk = self.yokoi(ans)
        self.printImg(yk)
        pr = self.pairRelationship(yk)
        self.printImg(pr)
        sh = self.connectedShrink(pr, ans)
        self.printImg(sh)

        # Update
        ans = sh
        self.iter += 1
    self.showImg(ans, save=True)
```

- Result:

  After looping for 7 times:

(The result with shape (64, 64) in .bmp)

```
________________11__________11111111________________________1__1
_________________11111__11111_1__1____________________________1_
__________________1__1111__1__1__1111_______________________11__
________________1_1___1_11_1111111__________________________1___
```