Computer Vision HW7 Report
蕭恩慈 / B07902095

The task is to write a program that does thinning on a downsampled image. For this task, we are using the original image (lena.bmp). To complete the task, I use Python as the program language with cv2, numpy, and copy library. Note that I used to use PIL to do the tasks, but I realize using cv2 is also allowed, which is relatively easier and simpler for coding, which makes the program lines shorter so it is easier to debug. But I will only use the cv2 library for the basic function like uploading or saving the image.

What needs to be done first to downsample the original image from 512x512 to 64x64. To do this, we need to use the binary image of the original image (using hw2 as reference, binarize lena.bmp with threshold at 128), and the 4-connected neighborhood detection (Yokoi connectivity) as taught in class (as well as referencing last hw6), which is using 8x8 blocks as unit and take the topmost-left pixel as the downsampled data. Then, we implement the interior/border operator and pair-relationship operator with the Yokoi connectivity computation for the thinning process. Lastly, for the thinning process, repeat the whole process until it is unchanged by making the result image of each thinning process as the "original" image for the next process.

Code snippets for each operators:

- Interior/Border operator

```
def findInteriorBorder(ori):
    def h(c, d):
        if c == d:
            return c
        return "b"
    interior_border = np.zeros(ori.shape, np.int)
    for i in range(ori.shape[0]):
        for j in range(ori.shape[1]):
            #background px
            if ori[i][j] > 0:
                x1, x2, x3, x4 = 0, 0, 0, 0
                if i == 0:
                    if j == 0:
                        x1, x4 = ori[i][j + 1], ori[i + 1][j]

                    elif j == (ori.shape[1]) - 1:
                        x3, x4 = ori[i][j - 1], ori[i + 1][j]

                    else:
                        x1, x3, x4 = ori[i][j + 1], ori[i][j - 1], ori[i + 1][j]

                elif i == (ori.shape[0]) - 1:
                    if j == 0:
                        x1, x2 = ori[i][j + 1], ori[i - 1][j]

                    elif j == (ori.shape[1]) - 1:
                        x2, x3 = ori[i - 1][j], ori[i][j - 1]

                    else:
                        x1, x2, x3 = ori[i][j + 1], ori[i - 1][j], ori[i][j - 1]
                else:
                    if j == 0:
                        x1, x2, x4 = ori[i][j + 1], ori[i - 1][j], ori[i + 1][j]

                    elif j == (ori.shape[1]) - 1:
                        x2, x3, x4 = ori[i - 1][j], ori[i][j - 1], ori[i + 1][j]

                    else:
                        x1, x2, x3, x4 = ori[i][j + 1], ori[i - 1][j], ori[i][j - 1],
 ori[i + 1][j]

                x1 = x1 / 255
                x2 /= 255
                x3 /= 255
                x4 /= 255
                a1 = h(1, x1)
                a2 = h(a1, x2)
                a3 = h(a2, x3)
                a4 = h(a3, x4)

                if a4 == "b":
                    interior_border[i][j] = 2 #border px

                else:
                    interior_border[i][j] = 1 #interior px

    return interior_border
```

- Pair relationship operator

```
def findPairRelationship(interior_border):
   def h(a, m):
      if a == m:
         return 1
      return 0

   pair_relationship = np.zeros(interior_border.shape, np.int)
   for i in range(interior_border.shape[0]):
      for j in range(interior_border.shape[1]):
         if interior_border[i][j] > 0:
            x1, x2, x3, x4 = 0, 0, 0, 0
            if i == 0:
               if j == 0:
                  x1, x4 = interior_border[i][j + 1], interior_border[i + 1][j]

               elif j == (interior_border.shape[1]) - 1:
                  x3, x4 = interior_border[i][j - 1], interior_border[i + 1][j]

```

```
                   else:
                       x1, x3, x4 = interior_border[i][j + 1], interior_border[i][j -
1], interior_border[i + 1][j]

               elif i == (interior_border.shape[0]) - 1:
                   if j == 0:
                       x1, x2 = interior_border[i][j + 1], interior_border[i - 1][j]

                   elif j == (interior_border.shape[1]) - 1:
                       x2, x3 = interior_border[i - 1][j], interior_border[i][j - 1]

                   else:
                       x1, x2, x3 = interior_border[i][j + 1], interior_border[i -
1][j], interior_border[i][j - 1]

               else:
                   if j == 0:
                       x1, x2, x4 = interior_border[i][j + 1], interior_border[i -
1][j], interior_border[i + 1][j]

                   elif j == (interior_border.shape[1]) - 1:
                       x2, x3, x4 = interior_border[i - 1][j], interior_border[i][j -
1], interior_border[i + 1][j]

                   else:
                       x1, x2, x3, x4 = interior_border[i][j + 1], interior_border[i
- 1][j], interior_border[i][j - 1], interior_border[i + 1][j]

               if ((h(x1, 1) + h(x2, 1) + h(x3, 1) + h(x4, 1)) >= 1) and
(interior_border[i][j] == 2):
                   pair_relationship[i][j] = 1

               else:
                   pair_relationship[i][j] = 2

   return pair_relationship
```

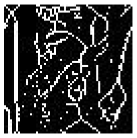
Code snippet for thinning process:

```
   #thinning
   thin = down
   while True:
       cp_thin = copy.deepcopy(thin)
       interior_border = findInteriorBorder(thin)
       pair_relationship = findPairRelationship(interior_border)

       yokoi_map = yokoi(thin)
       rm_map = (yokoi_map == 1) * 1
```

```
    for c in range(pair_relationship.shape[0]):
        for r in range(pair_relationship.shape[1]):
            if (rm_map[c][r] == 1) and (pair_relationship[c][r] == 1): #p
                thin[c][r] = 0

    if (np.sum(thin == cp_thin)) == (thin.shape[0] * thin.shape[1]):
        break
```

Result:

Original image (*lena.bmp*)


Final thinning results (*thin-lena.bmp*)