R13922136 周洛君

CV homework 7: Report

The resulting image is as follows:

My code consists of the following parts:

1. Binarize and downsample the lena image as in the previous homework
2. Create the marked image, to implement, we use the two operators, each with 2 primitive functions
    I. Yokoi operator
    II. Pair relationship operator
3. Use the connected shrinking operator to decide shrinking or not on a particular pixel (this function is called in the next thinning operator)
4. Use thinning operator to implement thinning on the whole downsampled image (this function contains 2 parts, one is the thinning itself, and one is to do it iteratively)
5. Main function, call the 3 operators:
    I. Yokoi_operator

II. Pair_relationship_operator()

III. Iterative_thinning()

To generate the resulting image, and use cv2.imwrite to write the result into an image file.

The first few parts are almost the same as the previous homework, except that we convert the calculation of Yokoi connectivity number into a callable function.

Next, we'll use the result of conducting the Yokoi function in the pair relationship operator. This operator has 2 primitive functions, and are defined as follows, the definition is just the formula given in the ppt:

```
[209]  ✓ 0.0s
```

```python
def pair_h(a, m):
    if a == m:
        return 1
    else:
        return 0
```

```
[210]  ✓ 0.0s
```

```python
def output(x0, x1, x2, x3, x4, m):
    if pair_h(x0, m) == 1 and pair_h(x1, m) + pair_h(x2, m) + pair_h(x3, m) + pair_h(x4, m) >= 1:
        return 'p'
    else:
        return 'q'

```

Using these 2 primitive functions, the pair relationship operator first initialize a zero array with the same shape as the Yokoi labeled result image, and iterate through each pixel. When we met a pixel that is labeled as 1 (which means its Yokoi connectivity number = 1 = edge), we further inspect its 4-neighbors, using the primitive functions to decide if we're marking the corresponding pixel in the pair_result as 'p' or 'q'. Also, we do not perform this action on the pixels with label different from 1.

```python
def pair_relationship_operator(Yokoi_result, m):
    pair_result = np.zeros(Yokoi_result.shape, dtype = 'U1')
    rows, cols = Yokoi_result.shape
    for i in range(rows):
        for j in range(cols):
            if Yokoi_result[i][j] == 1:
                right = Yokoi_result[i][j+1] if j < cols-1 else 0
                top = Yokoi_result[i-1][j] if i > 0 else 0
                left = Yokoi_result[i][j-1] if j > 0 else 0
                bottom = Yokoi_result[i+1][j] if i < rows-1 else 0
                pair_result[i][j] = output(Yokoi_result[i][j], right, top, left, bottom, m)
            else:
                pair_result[i][j] = ' '

    return pair_result
```

✓ 0.0s

The following connected shrink operator should be used later in the thinning operator, but I'll explain it first. For this operator, we'll inspect its 3x3 neighborhood, and calculate the 4 corner neighborhoods to get a_1,…,a_4. If only one of these a_is has value 1, then this pixel img[i][j] can be shrinked.

```python
def connected_shrink_h(b, c, d, e):
    if b == c and (d != b or e != b):
        return 1
    return 0

def connected_shrink_operator(img, i, j):
    rows, cols = img.shape
    x0 = img[i][j]
    x1 = img[i][j+1] if j < cols-1 else 0
    x2 = img[i-1][j] if i > 0 else 0
    x3 = img[i][j-1] if j > 0 else 0
    x4 = img[i+1][j] if i < rows-1 else 0
    x5 = img[i+1][j+1] if i < rows-1 and j < cols-1 else 0
    x6 = img[i-1][j+1] if i > 0 and j < cols-1 else 0
    x7 = img[i-1][j-1] if i > 0 and j > 0 else 0
    x8 = img[i+1][j-1] if i < rows-1 and j > 0 else 0

    a1 = connected_shrink_h(x0, x1, x6, x2)
    a2 = connected_shrink_h(x0, x2, x7, x3)
    a3 = connected_shrink_h(x0, x3, x8, x4)
    a4 = connected_shrink_h(x0, x4, x5, x1)

    if sum([a1, a2, a3, a4]) == 1:
        return 0
    return 1

```

The thinning operator below will first inspect the downsampled lena, and using the corresponding pair_result pixel to see if a pixel is of label 'p', if so, we use the connected shrink operator to decide if this pixel can be set as 0.

```python
def thinning_operator(img, pair_result):
    rows, cols = img.shape
    for i in range(rows):
        for j in range(cols):
            if pair_result[i][j] == 'p':
                img[i][j] = connected_shrink_operator(img, i, j)
    return img
```

Since thinning should be done iteratively, we have a iterative thinning function, we actually call this function to perform thinning, and this function will call the thinning function inside and determine if we need to do another round. We'll iterate until the thinning result does not change.

```python
def iterative_thinning(downsampled_img):
    previous_result = None
    current_result = downsampled_img.copy()
    iteration = 0

    while True:
        iteration += 1
        previous_result = current_result.copy()

        Yokoi_result = Yokoi_operator(current_result)
        pair_result = pair_relationship_operator(Yokoi_result, 1)
        current_result = thinning_operator(current_result, pair_result)

        if np.array_equal(previous_result, current_result):
            print(f"Converged after {iteration} iterations")
            break

        if iteration > 100:  # Safety limit
            print("Reached maximum iterations")
            break

    return current_result
```

The last part is to call all the operators, and save the resulting image.

```python
Yokoi_result = Yokoi_operator(downsampled_img)
np.savetxt('yokoi_result.txt', Yokoi_result, fmt='%d', delimiter=' ')

pair_result = pair_relationship_operator(Yokoi_result, 1)
np.savetxt('pair_result.txt', pair_result, fmt='%s', delimiter=' ')

thinning_result = iterative_thinning(downsampled_img)
np.savetxt('thinning_result.txt', thinning_result, fmt='%d', delimiter=' ')

display_image = (thinning_result * 255).astype(np.uint8)
cv2.imwrite('thinned_lena.bmp', display_image)
```

✓ 0.1s