Homework 7

R13525009 羅筠笙

a. Yokoi Connectivity Number

Description: This program processes a binarized Lena image to perform iterative thinning. It begins by downsampling the image to a 64x64 grid using the top-left pixel from each 8x8 block. Then, it calculates the Yokoi connectivity number for each pixel to determine its structural importance and marks candidate pixels for removal. The thinning process removes marked pixels iteratively while preserving the overall shape. Once the thinning process stabilizes, the final result is upsampled back to the original 512x512 resolution for clearer visualization.

```
from PIL import Image as im
import numpy as np

# Load the image
img = im.open('lena.bmp')
img_arr = np.asarray(img)
img_arr_binary = (img_arr >= 128).astype(np.uint8) * 255

# Downsample the image
def downsample(arr):
    m, n = arr.shape
    result = np.zeros((m // 8, n // 8), dtype=arr.dtype)
    for i in range(m // 8):
```

```
            for j in range(n // 8):
                result[i, j] = arr[i * 8, j * 8]
    return result

# Upsample the image back to 512x512
def upsample(arr):
    m, n = arr.shape
    result = np.zeros((m * 8, n * 8), dtype=arr.dtype)
    for i in range(m * 8):
        for j in range(n * 8):
            result[i, j] = arr[i // 8, j // 8]
    return result

# Expand the image array with a border of zeros for boundary conditions
def expand_with_zeros(arr):
    m, n = arr.shape
    result = np.zeros((m + 2, n + 2), dtype=arr.dtype)
    for i in range(m):
        for j in range(n):
            result[i + 1, j + 1] = arr[i, j]  # Put the original pixel to the center
    return result

# Function for computing Yokoi connectivity number
def h(b, c, d, e):
    if b != c:
        return 's'
    else:
        if d == b and e == b:
            return 'r'
        else:
            return 'q'

def f(a1, a2, a3, a4):
    if a1 == 'r' and a2 == 'r' and a3 == 'r' and a4 == 'r':
        return 5
    count = sum(1 for a in [a1, a2, a3, a4] if a == 'q')
    return count

def yokoi(arr):
```

```
    m, n = arr.shape
    result = [[0] * n for _ in range(m)]
    arr = expand_with_zeros(arr)

    for i in range(m):
        for j in range(n):
            a1 = h(arr[i+1][j+1], arr[i+1][j+2], arr[i][j+2], arr[i][j+1])
            a2 = h(arr[i+1][j+1], arr[i][j+1], arr[i][j], arr[i+1][j])
            a3 = h(arr[i+1][j+1], arr[i+1][j], arr[i+2][j], arr[i+2][j+1])
            a4 = h(arr[i+1][j+1], arr[i+2][j+1], arr[i+2][j+2], arr[i+1][j+2])

            tmp = f(a1, a2, a3, a4)
            if tmp and arr[i+1][j+1]:
                result[i][j] = tmp
    return np.array(result)

# Mark pairs in the Yokoi array for thinning
def mark_pairs(yokoi_arr):
    m, n = len(yokoi_arr), len(yokoi_arr[0])
    result = [[0] * n for _ in range(m)]
    yokoi_arr = expand_with_zeros(yokoi_arr)

    for i in range(m):
        for j in range(n):
            if yokoi_arr[i+1][j+1] == 1:
                count = sum(1 for x, y in [(i, j+1), (i+1, j), (i+2, j+1), (i+1,
j+2)] if yokoi_arr[x][y] == 1)
                if count >= 1:
                    result[i][j] = 1
    return np.array(result)

# Helper function for thinning
def h2(b, c, d, e):
    return 1 if b == c and (b != d or b != e) else 0

def f2(a1, a2, a3, a4, x):
    return 0 if a1 + a2 + a3 + a4 == 1 else x

# Perform the thinning process on the image
def thinning_process(original, marked_arr):
    m, n = original.shape
```

```
    original = expand_with_zeros(original)
    modified = 0   # Track if changes occur

    for i in range(m):
        for j in range(n):
            if marked_arr[i][j]:
                a1 = h2(original[i+1][j+1], original[i+1][j+2], original[i][j+2], original[i][j+1])
                a2 = h2(original[i+1][j+1], original[i][j+1], original[i][j], original[i+1][j])
                a3 = h2(original[i+1][j+1], original[i+1][j], original[i+2][j], original[i+2][j+1])
                a4 = h2(original[i+1][j+1], original[i+2][j+1], original[i+2][j+2], original[i+1][j+2])

                temp = f2(a1, a2, a3, a4, original[i+1][j+1])
                if temp != original[i+1][j+1]:
                    modified = 1
                    original[i+1][j+1] = temp

    # Remove the zero-padded border
    thinned_result = [[original[i+1][j+1] for j in range(n)] for i in range(m)]
    return np.array(thinned_result), modified

# Downsample, apply thinning, and upsample for visualization
downsampled = downsample(img_arr_binary)
change = 1

# Update the image until the last output never changed.
while change:
    yokoi_arr = yokoi(downsampled)
    marked_arr = mark_pairs(yokoi_arr)
    downsampled, change = thinning_process(downsampled, marked_arr)

# # Upsample for clearer visualization
final_result = upsample(downsampled)
```