922 U0610 電腦視覺
Computer Vision

Homework 6

授課教師： 傅楸善 教授

學生系級： 資工所一年級

學生姓名： 姚嘉昇

學生學號： R06922002

I. INTRODUCTION

1.1. Descriptions of Problem

This homework is to do Yokoi connectivity number with following rules:

A. Please binarize leba.bmp with threshold 128.

B. Please down sampling binary.bmp from 512x512 to 64x64, using 8x8 blocks as unit and take the topmost-left pixel as the down sampling data.

C. Print Yokoi connectivity number to text file.

1.2. Programming Tools

1.2.1. Programming Language: Python3

1.2.2. Programming IDE: Visual Studio Code

II. METHOD

2.1. Algorithms

2.1.1. Yokoi h function for 4-connectivity

$$h(b,c,d,e) = \begin{cases} q & if\ b = c\ and\ (d \neq b \lor e \neq b) \\ r & if\ b = c\ and\ (d = b \land e = b) \\ s & if\ b \neq c\ and\ (d = b \land e = b) \end{cases}$$

2.1.2. Yokoi f function for 4-connectivity

$$f(a_1, a_2, a_3, a_4) = \begin{cases} 5, if\ a_1 = a_2 = a_3 = a_4 = r \\ n,\ where\ \ n =\ \ numberof\{\#a_k | a_k = q\},\ otherwise \end{cases}$$

2.2. Code Fragments

2.2.1. Code fragments of this homework

```python
if __name__ == '__main__':
    from PIL import Image
    import numpy as np

    # Load image from file.
    originalImage = Image.open('lena.bmp')
    # Get binary image.
    binaryImage = getBinaryImage(originalImage, 128)
    # Save binary image fo file.
    binaryImage.save('binary.bmp')

    # Get downsampling image.
    downsamplingImage = downsampling(binaryImage, 8)
    # Save downsampling image fo file.
    downsamplingImage.save('downsampling.bmp')

    # Get Yokoi Connectivity Number.
    YokoiConnectivityNumber = YokoiConnectivityNumber(downsamplingImage)
    # Save Yokoi Connectivity Number to file.
    np.savetxt('YokoiConnectivityNumber.txt',
        YokoiConnectivityNumber.T,
        delimiter='', fmt='%s')
```

Figure 2.2.1.1. Code of main function.

```python
def getBinaryImage(originalImage, threshold):
    """
    :type originalImage: Image (from PIL)
    :type threshold: int
    :return type: Image (from PIL)
    """
    from PIL import Image
    # New image with the same size and 'binary' format.
    binaryImage = Image.new('1', originalImage.size)
    # Scan each column in original image.
    for c in range(originalImage.size[0]):
        # Scan each row in original image.
        for r in range(originalImage.size[1]):
            # Get pixel value in original image at (c, r).
            originalPixel = originalImage.getpixel((c, r))
            if (originalPixel >= threshold):
                # Put pixel value '1'  to binary image.
                binaryImage.putpixel((c, r), 1)
            else:
                # Put pixel value '0'  to binary image.
                binaryImage.putpixel((c, r), 0)
    # Return binary image.
    return binaryImage
```

Figure 2.2.1.2. Code of binarize.

```python
def downsampling(originalImage, sampleFactor):
    """
    :type originalImage: Image (from PIL)
    :type sampleFactor: int
    :return type: Image (from PIL)
    """
    from PIL import Image
    # Calculate the width and height of downsampling image.
    downsamplingWidth = int(originalImage.size[0] / sampleFactor)
    downsamplingHeight = int(originalImage.size[1] / sampleFactor)
    # New image with the downsampling size and 'binary' format.
    downsamplingImage = Image.new('1', (downsamplingWidth, downsamplingHeight))
    # Scan each column in downsampling image.
    for c in range(downsamplingImage.size[0]):
        # Scan each row in downsampling image.
        for r in range(downsamplingImage.size[1]):
            # Get pixel value in original image at (c * sampleFactor, r * sampleFactor).
            originalPixel = originalImage.getpixel((c * sampleFactor, r * sampleFactor))
            # Put pixel to downsampling image.
            downsamplingImage.putpixel((c, r), originalPixel)
    # Return downsampling image.
    return downsamplingImage
```

Figure 2.2.1.3. Code of down sampling.

```python
def getNeighborhoodPixels(originalImage, position):
    """
    :type originalImage: Image (from PIL)
    :type position: tuple
    :return type: numpy array
    """
    # Allocate memory space of neighborhoodPixels.
    neighborhoodPixels = np.zeros(9)
    # Get x and y of position.
    x, y = position
    # Scan dx from -1 to 1.
    for dx in range(3):
        # Scan dy from -1 to 1.
        for dy in range(3):
            # Calculate destination x, y position.
            destX = x + (dx - 1)
            destY = y + (dy - 1)
            # Avoid out of image range.
            if ((0 <= destX < originalImage.size[0]) and \
                (0 <= destY < originalImage.size[1])):
                # Get neighborhood pixel values.
                neighborhoodPixels[3 * dy + dx] = originalImage.getpixel((destX, destY))
            # It is out of image range.
            else:
                # Padding zeros when it is out of image range.
                neighborhoodPixels[3 * dy + dx] = 0
    # Original order:   [[x0, x1, x2], [x3, x4, x5], [x6, x7, x8]]
    # Sort pixels in    [[x7, x2, x6], [x3, x0, x1], [x8, x4, x5]] order.
    neighborhoodPixels = [
        neighborhoodPixels[4], neighborhoodPixels[5], neighborhoodPixels[1],
        neighborhoodPixels[3], neighborhoodPixels[7], neighborhoodPixels[8],
        neighborhoodPixels[2], neighborhoodPixels[0], neighborhoodPixels[6]]
    # Return Neighborhood Pixels.
    return neighborhoodPixels
```

Figure 2.2.1.4. Code of getting neighborhood pixels.

```python
def hFunctionYokoi(b, c, d, e):
    """
    :type b: int
    :type c: int
    :type d: int
    :type e: int
    :return type: str
    """
    if ((b == c) and (b != d or b != e)):
        return 'q'
    if ((b == c) and (b == d and b == e)):
        return 'r'
    if (b != c):
        return 's'
```

Figure 2.2.1.5. Code of Yokoi h function.

```python
def fFunctionYokoi(a1, a2, a3, a4):
    """
    :type a1: str
    :type a2: str
    :type a3: str
    :type a4: str
    :return type: str
    """
    # a1 == a2 == a3 == a4 == r
    if ([a1, a2, a3, a4].count('r') == 4):
        # Return label 5 (interior).
        return 5
    else:
        # Return count of 'q'.
        # 0: Isolated, 1: Edge, 2: Connecting, 3: Branching, 4: Crossing.
        return [a1, a2, a3, a4].count('q')
```

Figure 2.2.1.6. Code of Yokoi f function.

III. RESULTS

3.1. Original Image

Figure 3.1. Original lena.bmp.

3.2. Results of binary and down sampling

Figure 3.2.1. binary.bmp. Figure 3.2.2. downsampling.bmp.

3.3. Results of Yokoi connectivity number

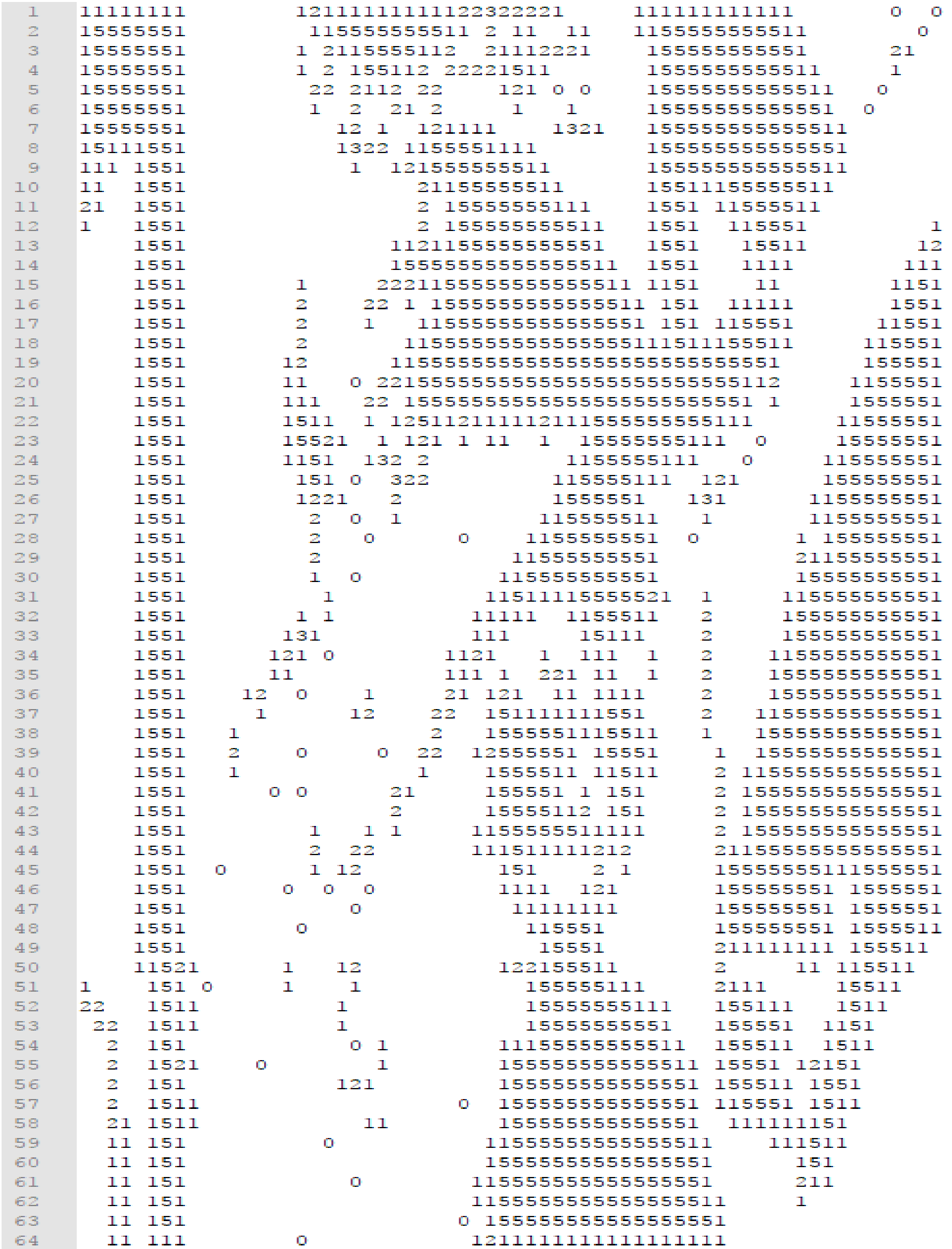

Figure 3.3.1. Yokoi connectivity number.