922 U0610 電腦視覺
Computer Vision

Homework 2

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

Part 1 of this homework is to binarize lena.bmp with threshold 128 (0-127, 128-255).

Part 2 of this homework is to draw the histogram of lena.bmp.

Part 3 of this homework is to find connected components with following rules:

A. Draw bounding box of regions.

B. Draw cross at centroid of regions.

C. Omit regions that have a pixel count less than 500.

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. Binarize lena.bmp with threshold 128 (0-127, 128-255)

Step 0. Define threshold of binary image.
Step 1. Load image from file.
Step 2. Get width and height of image.
Step 3. New image with the same size and ‘binary’ format.
Step 4. Process image pixel by pixel. (r: row, c: column)
Step 4.1. Get pixel value from lena.bmp at (c, r).
Step 4.2. If value bigger than or equal to threshold, set pixel as 1 to target at (c, r).
Step 4.3. If value smaller than threshold, set pixel as 0 to target at (c, r).
Step 5. Save image.

2.1.2. Draw the histogram of lena.bmp

Step 1. Load image from file.
Step 2. Get width and height of image.
Step 3. Create histogram array with zeros.
Step 4. Process image pixel by pixel. (r: row, c: column)
Step 4.1. Get pixel from lena.bmp at (width - 1 - c, r).
Step 4.2. Record count in histogram array.
Step 5. Save histogram to csv file.
Step 6. Plot, save and show the histogram.

2.1.3. Find connected components

Step 0. Define threshold of region pixels.
Step 1. Load image from file.
Step 2. Get width and height of image.
Step 3. Assign a unique ID to each region with 8-connected neighborhood detection.
Step 4. Only deal with region which has at least 500 pixels.
Step 4.1. Push rectangle's information to stack.
Step 5. New image with the same size and 'RGB' format.
Step 6. Draw rectangles and crosses on image.
Step 7. Save image.

2.2. Code Fragments

2.2.1. Part 1 of this homework

HW2.1.py binary.bmp

```python
from PIL import Image

# Define threshold of binary image.
threshold = 128

# Load image from file.
originalImage = Image.open('lena.bmp')

# Get width and height of image.
width, height = originalImage.size
# print ('width = %d, height = %d' %(width, height))

# New image with the same size and 'binary' format.
binaryImage = Image.new('1', originalImage.size)

# Process image pixel by pixel.
for c in range(width):
    for r in range(height):
        # Get pixel from original image.
        value = originalImage.getpixel((c, r))
        if (value >= threshold):
            value = 1
        else:
            value = 0
        # Put pixel to binary image.
        binaryImage.putpixel((c, r), value)

# Save image.
binaryImage.save('binary.bmp')
```

Figure 2.2.1. Code of part 1 of this homework.

2.2.2. Part 2 of this homework

HW2.2.py

```python
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np
import csv

# Load image from file.
originalImage = Image.open('lena.bmp')

# Get width and height of image.
width, height = originalImage.size
# print ('width = %d, height = %d' %(width, height))

# Create histogram array with zeros.
histogram = np.zeros(256)

# Process image pixel by pixel.
for c in range(width):
    for r in range(height):
        # Get pixel from original image.
        pixelValue = originalImage.getpixel((c, r))
        # Record count in histogram array.
        histogram[pixelValue] += 1

# Save histogram to csv file.
csvFile = open('histogram.csv', 'w')
writer = csv.writer(csvFile)
writer.writerow(histogram)

# Plot histogram.
plt.bar(range(len(histogram)), histogram)
# Save histogram to image file.
plt.savefig('histogram.png')
# Show plot.
plt.show()
```

Figure 2.2.2. Code of part 2 of this homework.

2.2.3. Part 3 of this homework

```python
from PIL import Image, ImageDraw
import numpy as np

class Stack:
    "A container with a last-in-first-out (LIFO) queuing policy."
    def __init__(self):
        self.list = []

    def push(self,item):
        "Push 'item' onto the stack"
        self.list.append(item)

    def pop(self):
        "Pop the most recently pushed item from the stack"
        return self.list.pop()

    def isEmpty(self):
        "Returns true if the stack is empty"
        return len(self.list) == 0

# Define threshold of region pixels.
thresholdRegionPixels = 500

# Load image from file.
originalImage = Image.open('lena.bmp')
binaryImage = Image.open('binary.bmp')

# Get width and height of image.
width, height = originalImage.size

# Record is this location visited or not.
visited = np.zeros((width, height))
# Image array with region label.
labeledImageArray = np.zeros((width, height))
# Count for region ID.
idCount = 1
# Record how many pixels in each region.
numberLabel = np.zeros(width * height)
```

Figure 2.2.3.1. Code of part 3 of this homework.

```python
# Process image pixel by pixel.
for c in range(width):
    for r in range(height):
        # If this location is 0, mark as visited.
        if binaryImage.getpixel((c, r)) == 0:
            visited[c, r] = 1
        # If this location is 1 and we haven't visited yet.
        elif visited[c, r] == 0:
            # Create a stack.
            stack = Stack()
            # Push this location to stack.
            stack.push((c, r))
            # While stack is not empty.
            while not stack.isEmpty():
                # Pop col and row from stack.
                col, row = stack.pop()

                # If we have visited this location, then continue.
                if visited[col, row] == 1:
                    continue
                # Mark this location as visited.
                visited[col, row] = 1
                # Assign a unique ID.
                labeledImageArray[col, row] = idCount

                # Count how many pixels in this label.
                numberLabel[idCount] = numberLabel[idCount] + 1

                # Look at 8 neighbouring locations.
                for x in [col - 1, col, col + 1]:
                    for y in [row - 1, row, row + 1]:
                        # If x, y is in range of image.
                        if (0 <= x < width) and (0 <= y < height):
                            # If this location isn't 0 and we haven't visited yet.
                            if (binaryImage.getpixel((x, y)) != 0) and (visited[x, y] == 0):
                                stack.push((x, y))
            idCount += 1
```

Figure 2.2.3.2. Code of part 3 of this homework.

```
78  # Use stack to store rectangle's information.
79  rectangles = Stack()
80
81  # Look through each label.
82  # regionID: ID of region which we want to bound.
83  # n: numberLabel[regionID]
84  for regionID, n in enumerate(numberLabel):
85      # Only deal with region which has at least 500 pixels.
86      if (n >= thresholdRegionPixels):
87          # left position of rectangle.
88          rectLeft = width
89          # right position of rectangle.
90          rectRight = 0
91          # top position of rectangle.
92          rectTop = height
93          # bottom position of rectangle.
94          rectBottom = 0
95          # Process image pixel by pixel.
96          for x in range(width):
97              for y in range(height):
98                  # Search label in this region.
99                  if (labeledImageArray[x, y] == regionID):
100                     # Update rectLeft with smaller x.
101                     if (x < rectLeft):
102                         rectLeft = x
103                     # Update rectRight with bigger x.
104                     if (x > rectRight):
105                         rectRight = x
106                     # Update rectTop with smaller y.
107                     if (y < rectTop):
108                         rectTop = y
109                     # Update rectBottom with bigger y.
110                     if (y > rectBottom):
111                         rectBottom = y
112         # Push rectangle's information to stack.
113         rectangles.push((rectLeft, rectRight, rectTop, rectBottom))
```

Figure 2.2.3.3. Code of part 3 of this homework.

```
115  # New image with the same size and 'RGB' format.
116  connectedImage = Image.new('RGB', originalImage.size)
117  connectedImageArray = connectedImage.load()
118
119  # Process image pixel by pixel.
120  for c in range(width):
121      for r in range(height):
122          # Convert binary image to 'RGB' format.
123          if (binaryImage.getpixel((c, r)) == 0):
124              connectedImageArray[c, r] = (0, 0, 0)
125          else:
126              connectedImageArray[c, r] = (255, 255, 255)
127
128  # Draw rectangles and crosses on image.
129  while not rectangles.isEmpty():
130      # Get rectangle's information.
131      rectLeft, rectRight, rectTop, rectBottom = rectangles.pop()
132      # Object to draw image.
133      draw = ImageDraw.Draw(connectedImage)
134      # Draw rectangle with red pen.
135      draw.rectangle(((rectLeft, rectTop), (rectRight, rectBottom)), outline = 'red')
136      # Center of rectangle.
137      rectCenterX = (rectLeft + rectRight) / 2
138      rectCenterY = (rectTop + rectBottom) / 2
139      # Draw horizontal line of cross.
140      draw.line(((rectCenterX - 10, rectCenterY), (rectCenterX + 10, rectCenterY)), \
141      fill = 'red', width = 5)
142      # Draw vertical line of cross.
143      draw.line(((rectCenterX, rectCenterY - 10), (rectCenterX, rectCenterY + 10)), \
144      fill = 'red', width = 5)
145
146  # Save image.
147  connectedImage.save('connectedImage.bmp')
```

Figure 2.2.3.4. Code of part 3 of this homework.

III. RESULTS

3.1. Original Image

Figure 3.1. Original lena.bmp.

3.2. Results of part 1 of this homework

Figure 3.2.1. Original lena.bmp.

Figure 3.2.2. Binary.bmp.

3.3. Results of part 2 of this homework

Figure 3.3.1. Original lena.bmp.

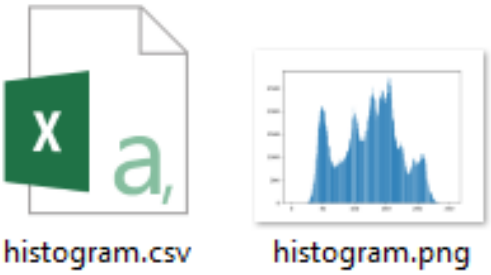

Figure 3.3.2. Output files of part 2.

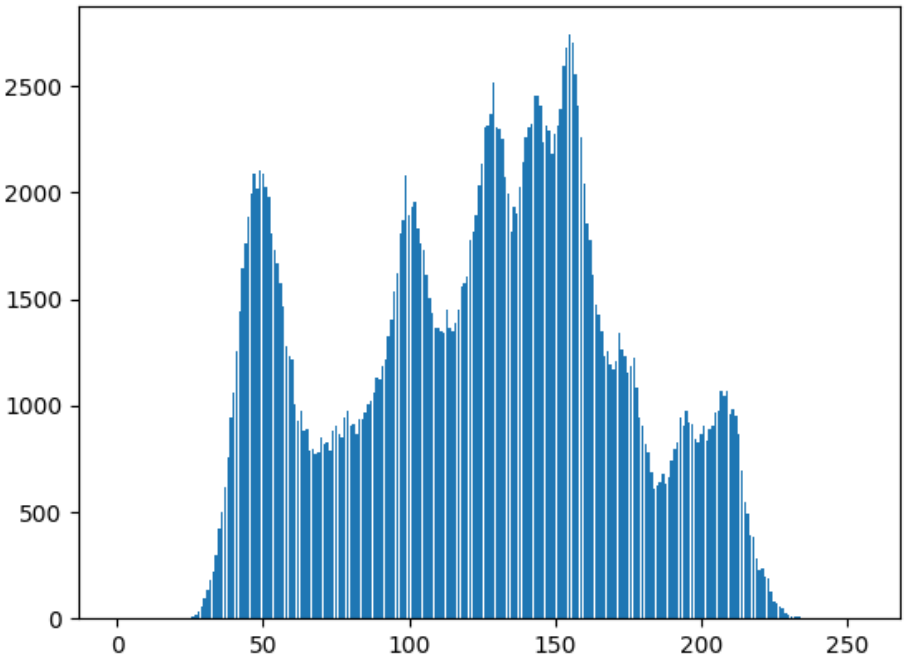

Figure 3.3.3. Histogram of lena.bmp.

3.4. Results of part 3 of this homework

Figure 3.4.1. Original lena.bmp.

Figure 3.4.2. Binary.bmp..

Figure 3.4.3. connectedImage.bmp.