PATTERN RECOGNITION

Project 1 of class CS480

Master of Science in Computer Science

By

***Miao Tian*(19977)**

Prepared Under the Direction of *Dr. Henry Chang*

ABSTRACT

In this Pattern Recognition project, I analyzed a digital image file, and then found objects and respective shapes in the image. Following are the steps of this project.

Step 1: Get a *digital image* from a file, and then store it into a 2-D array *pixels*

Step 2: Analyze the *pixels* and get its histogram, store it into a 1-D array *hist*

Step 3: Analyze the *hist* and find the threshold of it (the cut point of the image)

Step 4: Convert the *pixels* into binary (background and foreground)

Step 5: Label the *pixels* by objects

Step 6: Find how many objects are in the object

ACKNOWLEDGMENTS

I would like to thank Dr. Henry Chang for his great guidance and support on this project.
I would also thank my classmates Rui Ding and Bo Dang for exchanging ideas during the project.

CONTENTS

ILLUSTRATIONS

Section 1: INTRODUCTION

Pattern recognition finds its application across various sectors in contemporary tech industries, including the identification of written characters in scanned documents, understanding human speech, and evaluating biometric fingerprints. Acquiring foundational knowledge in pattern recognition is essential.

In this project, I read a digital image from a text file and stored it into a 2-D array, then analysis the array to recognize the objects in the image and determine the shape of the objects. The thesis will include in the following sections[1] *(Dr. Henry Chang, November, 2023)*.

Section 1: INTRODUCTION, overview the key points of the project.

Section 2: PROGRAM DESIGN, describe the major steps of the project design, which include following steps. *(Exercises for 2D Array, Dr. Henry Chang)*

Step 1: read the digital image from a file, and store into an array for future operations.

1	3	5	7	9	3	4	5	5	6
1	20	25	24	3	5	6	4	2	4
1	22	35	24	3	5	6	4	5	7
1	20	28	34	2	5	6	4	8	9
1	3	5	7	9	3	4	4	5	6
1	3	5	7	9	3	27	4	5	6
1	3	5	7	9	28	24	24	5	6
1	3	5	7	9	29	28	24	5	6
1	3	5	7	9	3	24	4	5	6
1	3	5	7	9	3	4	4	5	6

Step 2: Analysis the array and find the gray-level distribution.

number |	Histogram	number |	Histogram
1 |	10	20 |	2
2 |	2	22 |	1
3 |	14	24 |	6
4 |	12	25 |	1
5 |	18	27 |	1
6 |	10	28 |	3
7 |	8	29 |	1
8 |	1	34 |	1
9 |	8	35 |	1

Step 3: Analyze the gray-level distribution and find the cut point of the image.

For example, 14

Step 4: Convert the array to binary with only background and foreground.

0	0	0	0	0	0	0	0	0	0
0	**1**	**1**	**1**	0	0	0	0	0	0
0	**1**	**1**	**1**	0	0	0	0	0	0
0	**1**	**1**	**1**	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	**1**	0	0	0
0	0	0	0	0	**1**	**1**	**1**	0	0
0	0	0	0	0	**1**	**1**	**1**	0	0
0	0	0	0	0	0	**1**	0	0	0
0	0	0	0	0	0	0	0	0	0

Step 5: Label the objects in the image.

0	0	0	0	0	0	0	0	0	0
0	**1**	**1**	**1**	0	0	0	0	0	0
0	**1**	**1**	**1**	0	0	0	0	0	0
0	**1**	**1**	**1**	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	**2**	0	0	0
0	0	0	0	0	**2**	**2**	**2**	0	0
0	0	0	0	0	**2**	**2**	**2**	0	0
0	0	0	0	0	0	**2**	0	0	0
0	0	0	0	0	0	0	0	0	0

Step 6: Find the number and shape of the objects in the image.

In the above image, we can determine one square and one circle are in the image.

Section 3: PROGRAM IMPLEMENTATION, describe the structure of the above steps.

Section 4: ENHANCEMENT IDEAS, describe how this project can be improved.

Section 5: CONCLUSION, briefly summarize the project.

Section 2: Program Design

In the section, I will introduce the detailed design of this project.

As I introduced before, there are total 6 steps in this project.

Step 1: read the digital image from a file, and store into an array for future operations.

Input: a file of digital image.

Output: a 2D array which contains the gray-level of the image.

Algorithm: read the file character by character, and determine whether it is a data line. If yes, determine it is beginning of the number, in the number, or end of a number, then, add it to the corresponding element in the array.

Step 2: Analysis the array and find the gray-level distribution.

Input: a 2D array which contains the gray-level of the image.

Output: a histogram which indicates the gray-level distribution of the image.

Algorithm: first, initialize the histogram to 0, then scan the image array one by one, and increase the corresponding histogram element.

Step 3: Analyze the gray-level distribution and find the cut point of the image.

Input: histogram of the image distribution.

Output: the threshold which separate background and foreground.

Algorithm: scan the histogram from the beginning to the end, and then determine the variance gap between left and right side. Use the point with biggest gap of variance as threshold[2]. *(Otsu, 1979)*

Step 4: Convert the array to binary with only background and foreground.

Input: image array and the threshold.

Output: binary image with only background and foreground.

Algorithm: scan the image from beginning to the end, and set all the pixels higher than the threshold to 1, and the ones lower than the threshold to 0.

Step 5: Label the objects in the image.

Input: binary image.

Output: labeled image, each number indicate an object or background.

Algorithm: scan the image from beginning to the end, if the pixel is connected to the left or upper one, set the respective value to the pixel, else assigned a new group number to it. Then check if the left and upper are connected and with different labels, if yes, group them. Then merge the groups[3]. *(Shapiro, L., and Stockman, G. ,2002)*

Step 6: Find the number and shape of the objects in the image.

Input: labeled image.

Output: the objects and the respective shapes.

Algorithm: scan the image from beginning to the end, get the area and perimeter of the objects, then base on the ratio, we can figure out the shape of the objects.

Section 3: PROGRAM IMPLEMENTATION

In this section, I will introduce the detailed implementation of the project.

3.1 Transfer Data from File to 2D Array

In this step, I will read the digital image into a 2D Array. It's the RecogImage.java file that implements this part.

1) Class Definition: The class RecogImage is defined with a single member variable, path, which will hold the file path of the image to be processed.
2) Constructor: The RecogImage constructor takes a file path as a parameter and initializes the path member variable with it.
3) readBmp() Method: This method is responsible for reading and processing the image.
    a) A BufferedImage object and a File object are declared.
    b) The try block attempts to read the image from the given path using ImageIO.read(f). If an exception occurs during this I/O operation, it is caught and printed out.
    c) The dimensions of the image are obtained using the getWidth() and getHeight() methods.
    d) A 2-dimensional integer array arr is created with dimensions corresponding to the image's width and height.
    e) A nested for loop iterates over every pixel in the image.
        i. For each pixel, its ARGB (alpha, red, green, blue) values are extracted using bitwise operations.
        ii. The average of the R, G, and B values is calculated to convert the pixel to grayscale.
        iii. This average value is then stored in the arr array at the corresponding position.
        iv. The pixel's RGB value in the image is replaced with the grayscale average, effectively converting the colored image into grayscale.

Test result

Figure 1 Original File and the image array

3.2 Make Histogram

In this step, I will find out the frequency of each color of all the pixels, i.e. histogram.

1. Initialization of Histogram Array:

int[] hist = new int[256];

This line initializes an array hist of size 256. This size is chosen assuming the image is a grayscale image where each pixel's intensity value ranges from 0 to 255. Each element of this array will eventually hold the count of how many pixels in the image have the intensity value corresponding to the index of the array.

2. Nested Loops to Traverse the Image:

```
for (int i = 0; i < image.length; i++) {
    for (int j = 0; j < image[i].length; j++) {
        ...
    }
}
```

These nested for loops iterate over the 2-dimensional array image. The outer loop iterates over each row (i), and the inner loop iterates over each column (j) in that row. Together, they visit every pixel in the image.

3. Incrementing Histogram Values:

```
hist[image[i][j]]++;
```

Within the inner loop, this line is executed for every pixel in the image. image[i][j] accesses the intensity value of the pixel at row i and column j. This value is used as an index to access the corresponding element in the hist array. The ++ operator increments the value at this index by 1. This incrementing operation effectively counts the number of times each intensity value appears in the image.

4. Returning the Histogram:

Finally, the method returns the hist array, which now contains the histogram of the image. Each index of this array represents an intensity value (from 0 to 255), and the value at each index represents the count of pixels in the image that have that intensity.

Test result

Figure 2 Histogram

```
pixels = {
{ 1, 3, 5, 7, 9, 3, 4, 4, 5, 6 },
{ 1, 3, 5, 7, 9, 3, 4, 4, 5, 6 },
{ 1, 3, 5, 7, 9, 3, 4, 4, 5, 6 },
{ 1, 3, 5, 20, 25, 24, 33, 5, 6, 4 },
{ 1, 3, 5, 22, 35, 24, 32, 5, 6, 4 },
{ 1, 3, 5, 20, 28, 34, 23, 5, 6, 4 },
{ 1, 3, 5, 21, 25, 27, 23, 5, 6, 4 },
{ 1, 3, 5, 7, 9, 3, 4, 4, 5, 6 },
{ 1, 3, 5, 7, 9, 3, 4, 4, 5, 6 },
{ 1, 3, 5, 7, 9, 3, 4, 4, 5, 6 }
```

gray-level	#-of-pixels
1	10
3	16
4	16
5	20
6	10
7	6
9	6
20	2
21	1
22	1
23	2
24	2
25	2
27	1
28	1
32	1
33	1
34	1
35	1

3.3 Find the Threshold

In this step, I will find out the threshold of the image, i.e. the cut point of background and fore ground. The first algorithm is created by me but it not suitable for images with noises. The second algorithm is the famous Otsu Algorithm.

3.3.1 Biggest Gap Algorithm

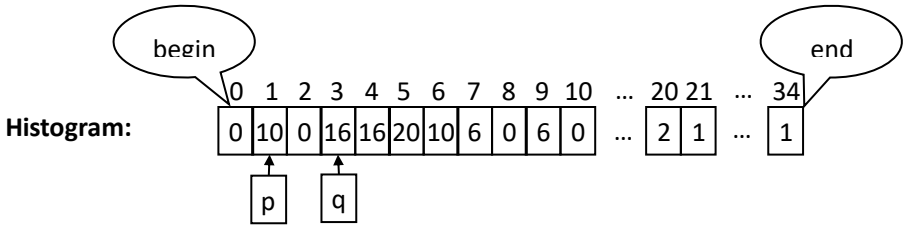
1) define two pointers **begin**, **end** to point to the first and last element of the array
   define another two pointers **p**, **q** to control the algorithm
   define **gap** to make sure the biggest gap
2) Initialize **threshold** to 0; // if 0 is returned, no successful value returned.
3) for **gap**=maximum_number to 1
   Let **p** point to the first non-zero element
   Let **q** point to the second non-zero element
   while **q**<**end**
   check if **q**-**p** is **gap**
   if ture, **threshold** = **q**-**begin**; and return **threshold**;
   else **p** move to **q** and **q** move to next none zero element
   next **gap**

Test result

Figure 3 Biggest Gap result

```
Threshold is: 20
```

Figure 4 Biggest Gap Algorithm



This algorithm will not perform well if there is noise. Then *Dr. Henry Chang* introduced Bimodal and the Otsu Algorithm fit better for this problem.

3.3.2 Bimodal Method (Otsu) Algorithm

Otsu's method is an algorithm of image thresholding named after Nobuyuki Otsu. This algorithm is only suitable for bimodal.

1) use a function **FindVariance** to get the variance of an **Array** of size **size**
   first, get the **mean** of the **Array**
   for **i**=0 to **size**
   sum += **Array**[**i**]
   next **i**
   mean = **sum** / **size**
   second, get the **variance**
   for **i**=0 to **size**
   sum += (**Array**[**i**] - **mean**) * (**Array**[**i**] - **mean**)
   next **i**
   variance = **sum** / **size**
2) set a pointer **p** to the second number of histogram **hist**[]
   use **varianceGap** to catch the biggest **variance** gap, initialize it to 0
   use **varianceGapTemp** to store each **variance** gap temporarily.
3) scan **p** from the second number to the **end** of the histogram, and calculate the gap of the variance between the left side and right side of the pointer

varianceGapTemp = **FindVariance** (**hist**, **p** - **hist**) - (**p**, **end** - **p**)

if **varianceGapTemp** > **varianceGap**

varianceGap = **varianceGapTemp**

threshold = **p** - **hist**

p++

4) return **threshold**

Test result

Figure 5 Ostu Algorithm result

```
Threshold is: 14
```

3.4 Convert to Binary

This step converts the image to binary, i.e. with only background and foreground.

```
for i=0 to row
    for j=0 to col
        if imageTable[i][j] < threshold
            imageTable[i][j]=0
        else
            imageTable[i][j]=1
        next j
    next i
```

Test result

Figure 6 Original image data

```
int[][] pixel1 = {
    {1, 3, 5, 7, 9, 3, 4, 4, 5, 6},
    {1, 3, 5, 7, 9, 3, 4, 4, 5, 6},
    {1, 3, 5, 7, 9, 3, 4, 4, 5, 6},
    {1, 3, 5, 20, 25, 24, 33, 5, 6, 4},
    {1, 3, 5, 22, 35, 24, 32, 5, 6, 4},
    {1, 3, 5, 20, 28, 34, 23, 5, 6, 4},
    {1, 3, 5, 21, 25, 27, 23, 5, 6, 4},
    {1, 3, 5, 7, 9, 3, 4, 4, 5, 6},
    {1, 3, 5, 7, 9, 3, 4, 4, 5, 6},
    {1, 3, 5, 7, 9, 3, 4, 4, 5, 6}
};
```

Threshhold and the binary image

```
Thresholded Image 1:

Target Threshold: 20
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 0 0 0
0 0 0 1 1 1 1 0 0 0
0 0 0 1 1 1 1 0 0 0
0 0 0 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```

3.5 Connectivity Analysis, Convert to labeled areas

This step labels the image by area.

1) create a dynamic 2D array **DynamicImage** to store the binary image temporarily.
   create a **labelSize** to check how many different areas we have (including duplicated ones)
   create a dynamic array **groupCheck[groupSize]** to mark whether it is duplicated or not
2) check the first row and the first column to check if there is an object
3) scan the **imageTable** from top to bottom, left to right. (row by row, column by colum)
   if connected to the left or upper one(same **DynamicImage** value), assign the corresponding **imageTable** value to it
   else
      use new **labelSize** to update the pixel
      increase **labelSize**
      update **groupCheck** [**labelSize**]
   then group check the left and upper pixel

if they are a group with different label, update **groupCheck**

4) free **DynamicImage**
5) merge the same group to one

   use a function **groupMerge** to recursively find the smallest index of the same group and then update the **groupCheck**

   for i=0 to **labelSize**

   if **groupCheck**[i] is not i

   groupMerge (**groupCheck**, i)

   next i
6) merge the groups of the image

   if **groupCheck**[**imageTable**[i][j]] is not equal to **imageTable**[i][j]

   imageTable[i][j] = **groupCheck**[**imageTable**[i][j]]
7) free **groupCheck**

Test result

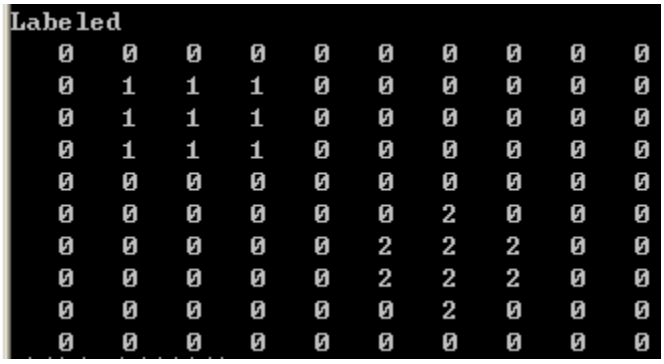
Figure 7 Labeled Image

Before group merge:

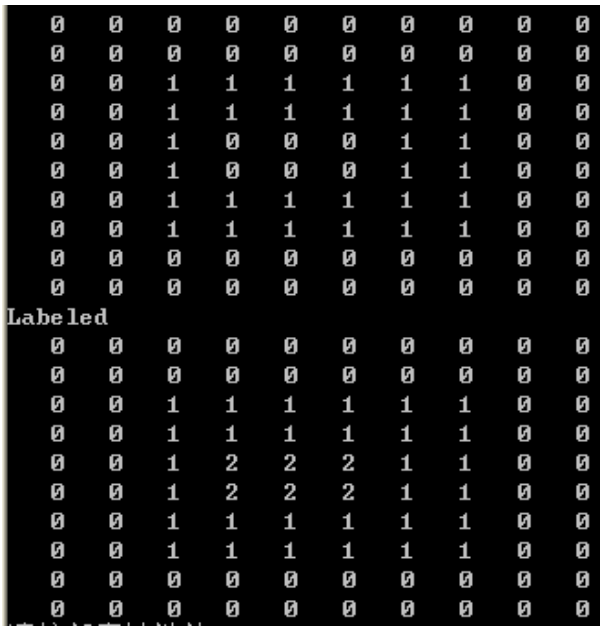
```
  0   0   0   0   0   0   0   0   0   0
  0   1   1   1   0   0   0   0   0   0
  0   1   1   1   0   0   0   0   0   0
  0   1   1   1   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   2   0   0   0
  0   0   0   0   0   3   3   3   0   0
  0   0   0   0   0   3   3   3   0   0
  0   0   0   0   0   0   3   4   4   4
  0   0   0   0   0   0   0   0   0   0
```

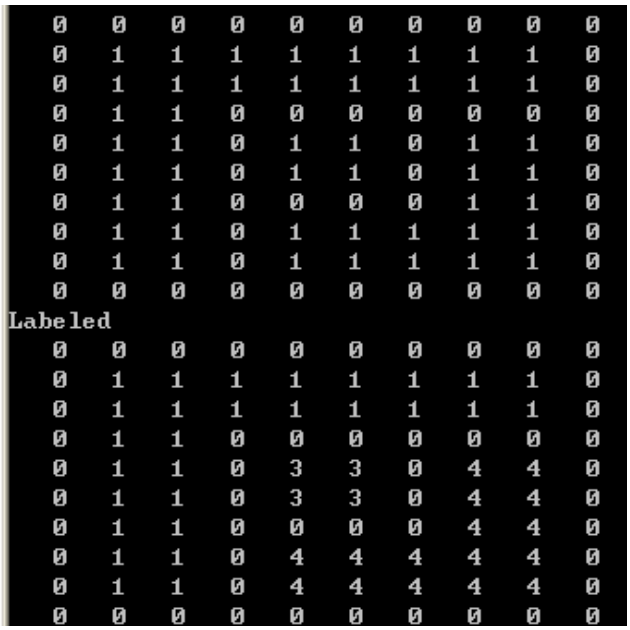
```
groupCheck:
GROUP  ¦ LABEL
    0 ¦     0
    1 ¦     1
    2 ¦     2
    3 ¦     2
    4 ¦     0
```

After group merge:



More data tested:





3.6 Compute attributes and recognize objects

This step detect what object we have in the image. We assume there is only circle or square.

1) create two dynamic array **Area** and **Perimeter** to store the area and perimeter of the objects
2) initialize all the elements of **Area** and **Perimeter** to 0
3) scan the **imageTable** from top to bottom, left to right
   if there is a pixel of an object
      increase respective **Area**
      if the pixel is edge pixels

increase respective **Perimeter**

4) adjust **Area** since it is 1 bigger than what it should be
5) scan each object

   ratio = 16.0 * **Area**[i] / (**Perimeter**[i]***Perimeter**[i])

   if **ratio** < 1.1

      it is a square (around 1)

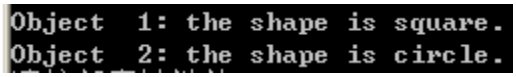
   else

      it is a circle (around 1.27)

6) free **Area** and **Perimeter**

Test result

Figure 8 Objects recognition



Section 4: ENHANCEMENT IDEAS

This project only works for digital file image with low resolution and fixed object shapes. We can improve this project from following ways.

- First, we can improve it closer to real-world product by inserting a piece of code which can open and read real image file, such as bmp file, jpeg file, etc.
- Second, we can improve the ability of this project to handle higher resolution images.
- Third, this project can only figure out bi-model images, we can improve the ability of this project to handle more complex images.
- Last but not least, we can improve this project to make it available to detect more objects, not just squares and circles.

Section 5: CONCLUSION

During the design and implementation of this Pattern Recognition project, I enjoyed the happiness of design my own project, and the time shared with classmates who are also enthusiastic and passionate for doing this project. I learned a lot from them.

In conclusion, in this project, we read the data from a digital image file, analyzed it and determine the objects and corresponding shapes of them. Following is a brief summary of the project.

Step 1: Get a *digital image* from a file, and then store it into a 2-D array *pixels*

Step 2: Analyze the *pixels* and get its histogram, store it into a 1-D array *hist*

Step 3: Analyze the *hist* and find the threshold of it (the cut point of the image)

Step 4: Convert the *pixels* into binary (background and foreground)

Step 5: Label the *pixels* by objects

Step 6: Find how many objects are in the object

t

BIBLIOGRAPHY

1. Dr. Henry Chang. Exercises for 2D Array. November, 2023
   https://hc.labnet.sfbu.edu/~henry/sfbu/course/image/pattern_recog/slide/exercise_pattern_recog.html
2. Otsu, Nobuyuki. A Threshold Selection Method from Gray-Level Histograms. January, 1979
   http://en.wikipedia.org/wiki/Otsu's_method
3. Shapiro, L., and Stockman, G.. Computer Vision. 2002
   http://en.wikipedia.org/wiki/Connected-component_labeling

APPENDIX: Source Code