

基于内容的图像检索系统设计与实现

**实训报告**

|  |  |
| --- | --- |
| **姓 名：** | **董怡蕙** |
| **班 级：** | **计科1403** |
| **学 号：** | **2014317200323** |
| **指导教师：** | **翟瑞芳 彭辉 周雄辉 高俊祥** |

中国·武汉

二〇一七 年 七 月 九 日

**目 录**

[一、 系统设计 3](#_Toc17793)

[二、 所采用](#_Toc1184)**[算法思](#_Toc1184)**[想 3](#_Toc1184)

[2.1颜色特征相似度识别——直方图 3](#_Toc11224)

[2.2纹理特征相似度识别——LBP 4](#_Toc2594)

[三、 详细实现过程 5](#_Toc11870)

[3.1 图像检索系统工作流程 5](#_Toc5004)

[3.2 图像检索系统框架 5](#_Toc205)

[3.3 数据库构建与实现 7](#_Toc27484)

[3.4核心代码及注释 8](#_Toc15020)

[四、 实验结果分析 17](#_Toc28146)

[五、 实训总结和心得 18](#_Toc17901)

[六、 参考文献 19](#_Toc10312)

1. 系统设计

基于内容的图像检索系统（Content Based Image Retrieval, 以下简称CBIR），允许用户在界面提交一张需要比对的图像，在图像数据库（或本地机、或网络）中查找图片，后台通过执行相似度比较算法查找具有相同或相似内容的其它图像，在界面中按照相似度从大到小，显示6张图像及其相似度数值。

本系统的基本功能是实现基于视觉特征的图像检索。具体包括：（1）通过颜色直方图实现基于颜色信息的图像检索（2）通过实现基于纹理特征的图像检索。

1. 所采用**算法思**想

## 2.1颜色特征相似度识别——直方图

首先对源图像与要筛选的图像进行直方图数据采集，对采集的各自图像直方图进行归一化再使用巴氏系数（Bhattacharyya coefficient）算法对直方图数据进行计算，最终得出图像相似度值，其值范围在[0, 1]之间0表示极其不同，1表示极其相似（相同）。

灰度直方图是灰度级的函数，描述图像中该灰度级的像素个数（或该灰度级像素出现的频率）：其横坐标是灰度级，纵坐标表示图像中该灰度级出现的个数（频率）。一维直方图结构如下：

 (1)

相似度比较：巴氏系数（Bhattacharyyacoefficient）算法

 (2)

其中P, P’分别代表源与候选的图像直方图数据，对每个相同i的数据点乘积开平方以后相加得出的结果即为图像相似度值（巴氏系数因子值），范围为0到1之间。

## 2.2纹理特征相似度识别——LBP

LBP（Local Binary Pattern，局部二值模式）是一种用来描述图像局部纹理特征的算子，常用于如纹理分类、人脸分析等；它具有旋转不变性和灰度不变性等显著的优点。它是首先由T. Ojala, M.Pietikäinen, 和 D. Harwood 在1994年提出，用于纹理特征提取。而且，提取的特征是图像的局部的纹理特征；

原始的LBP算子定义为在3\*3的窗口内，以窗口中心像素为阈值，将相邻的8个像素的灰度值与其进行比较，若周围像素值大于中心像素值，则该像素点的位置被标记为1，否则为0。这样，3\*3邻域内的8个点经比较可产生8位二进制数（通常转换为十进制数即LBP码，共256种），即得到该窗口中心像素点的LBP值，并用这个值来反映该区域的纹理信息。如图1所示：

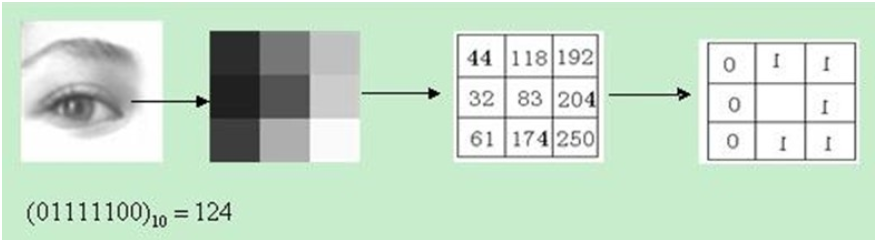


图1：LBP算子示意图

LBP旋转不变模式。从 LBP 的定义可以看出，LBP 算子是灰度不变的，但却不是旋转不变的。图像的旋转就会得到不同的 LBP值。Maenpaa等人又将 LBP算子进行了扩展，提出了具有旋转不变性的 LBP 算子，即不断旋转圆形邻域得到一系列初始定义的 LBP值，取其最小值作为该邻域的 LBP 值。

对原始的LBP模式进行降维，使得数据量减少的情况下能最好的代表图像的信息。为了解决二进制模式过多的问题，提高统计性，Ojala提出了采用一种“等价模式”（Uniform Pattern）来对LBP算子的模式种类进行降维。Ojala等认为，在实际图像中，绝大多数LBP模式最多只包含两次从1到0或从0到1的跳变。因此，Ojala将“等价模式”定义为：当某个LBP所对应的循环二进制数从0到1或从1到0最多有两次跳变时，该LBP所对应的二进制就称为一个等价模式类。

1. 详细实现过程

图像搜索现实的一般过程。第一步，提取图像特征值。第二步，对特征值进行处理。第三步，匹配特征值。

## 3.1 图像检索系统工作流程

基于内容的图像检索技术是对输入的图像进行分析并分类统一建模，提取其颜色、形状、纹理、轮廓和空间位置等特征，建立特征索引, 存储于特征数据库中。检索时，用户提交查询的源图像，通过用户接口设置查询条件，可以采用一种或几种的特征组合来表示，然后在图像数据库中提取出查询到的所需关联图像，按照相似度从大到小的顺序，反馈给用户。

## 3.2 图像检索系统框架

基于内容的图像检索系统框架如图2所示。系统的核心是图像特征数据库。图像特征既可以从图像本身提取得到, 又可以通过用户交互获得, 并用于计算图像之间的相似度计算。系统框架应主要包含以下几个基本功能模块：检索方法设置、检索结果浏览、数据库管理维护等。其逻辑结构如图3所示。

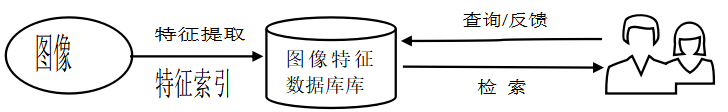


图2：基于内容的图像检索框架

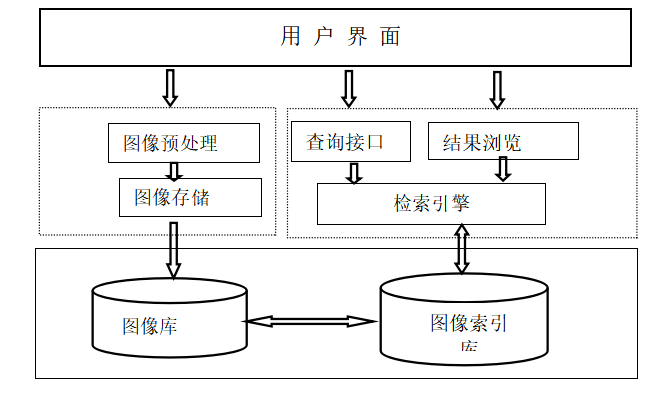


图3：基于内容的图像检索系统逻辑模型

图像检索CBIR系统界面如图4 所示。该界面主要包括：数据库预处理（向数据库中上传图片信息）选择目录提交区，选择提交待检索图像区，检索出的6幅图像显示区域等。

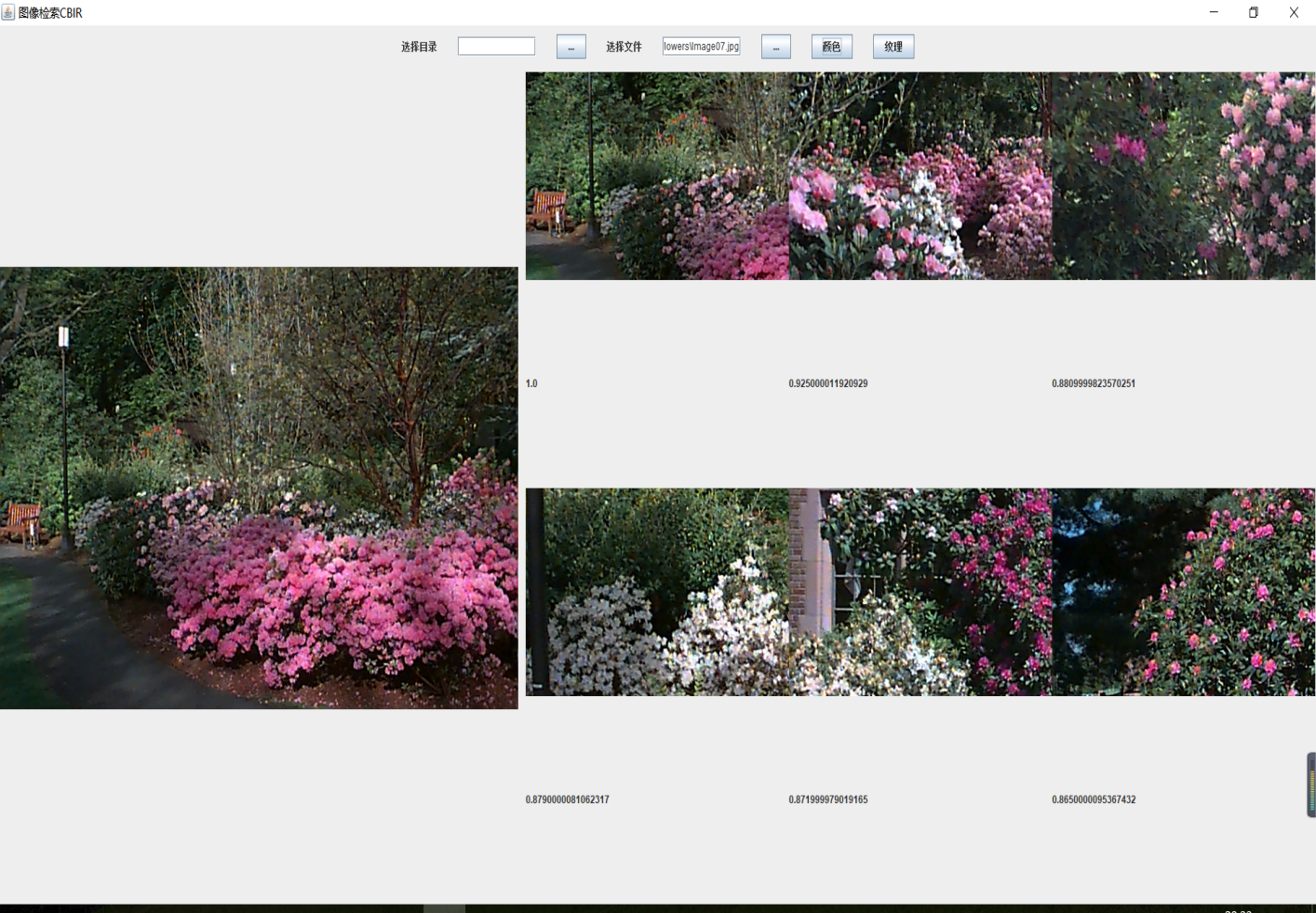


图4：图像检索CBIR系统界面

## 3.3 数据库构建与实现

图像是图像元灰度值的纪录, 以行列数据矩阵表示, 一般信息量比较大。直接读取图像的信息存入数据库中, 不但增加了数据库的容量, 而且增加了计算机的负担。若直接根据图像的路径名称建立图像地址库, 可有效提高计算机的计算效率。当需要提取图像时, 再根据图像的地址, 对图像进行检索, 平时, 只需对图像的基本信息进行管理。数据库的基本功能包括: 增加删除图像、图像统计、图像的显示等。数据库存储的基本信息部分示意图如图5所示：

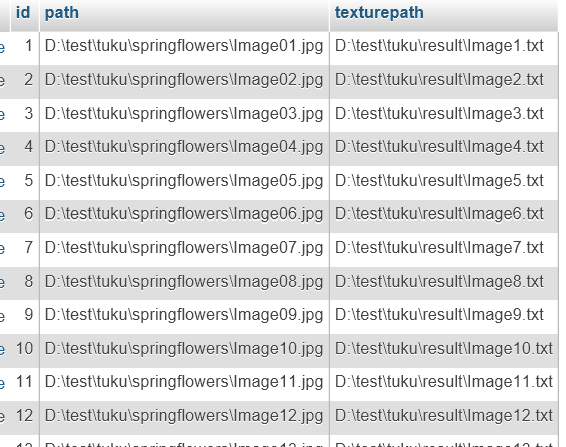


图5：数据库存储数据示意

由于图库中有大量的图像，每一张图像中有许多像素点，每个像素点又有许多特征值，如果都将其放入内存中，是不可能实现的。因此将提取出的特征值存放到txt文件中，每打开应用系统，就可以直接从外存中读取文件，构建图像特征索引库。

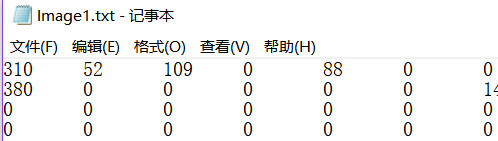


图6：纹理特征值txt文件内容局部示意

## 3.4核心代码及注释

**3.4.1界面**

涉及的文件：



图7：界面实现所涉及的文件

界面代码中主要是文件选择器和按钮响应动作的实现,两个选择器分别向系统提交，一是待上传数据库的文件夹目录路径，二是用户提交的需要比较相似度的文件路径。

（1）文件选择器关键代码：

chooser.setCurrentDirectory(**new** File("d:/test/tuku"));//设置文件选择器的初始目录

//在“...”按钮的触发相应动作中写如下代码：

chooser.setFileSelectionMode(1);//1设定只能选择到文件夹，0能选择到文件

**int** state=chooser.showOpenDialog(**null**);//此句是打开文件选择器界面的触发语句

（2）按钮动作代码：

//颜色特征比较按钮触发事件

button2.addActionListener(**new** ActionListener(){

**public** **void** actionPerformed(ActionEvent e){

**if**(path!=**null**){

**try** {

Element[] el = SearchImage.*searchImage*(1,path);//若是特征按钮则此处*searchImage（）*方法的第一个参数为2

**for**(**int** i=0;i<6;i++){

resimage[i].setIcon(**new** ImageIcon(el[i].epath));

res[i].setText(Double.*toString*(el[i].similar));

}

} **catch** (SQLException | IOException e1) {

// **TODO** Auto-generated catch block

e1.printStackTrace();

}

}

**else**{

System.***out***.println("未读取到图片");

}

}

});



图8：界面功能——文件选择器及按钮展示

1. 整体布局：

frame.setLayout(**new** BorderLayout(10,5));

frame.add(panel2,BorderLayout.***WEST***);//待检索图片显示区

frame.add(panel1,BorderLayout.***NORTH***);//用户操作区

frame.add(panel3,BorderLayout.***CENTER***);//检索结果显示区

其中检索结果显示区为4行3列表格布局，分别按相似度大小对应显示检索结果图片及其相似度，值得注意的是为了能够使其按照一行是图片，一行是对应文字来显示，需要设置如下的循环：

**for**(**int** i=0;i<2;i++){

panel3.add(resimage[i\*3]);

panel3.add(resimage[i\*3+1]);

panel3.add(resimage[i\*3+2]);

panel3.add(res[i\*3]);

panel3.add(res[i\*3+1]);

panel3.add(res[i\*3+2]);

}

**3.4.2向数据库中批量存储图片信息**

主要涉及的文件：

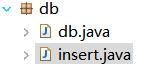


图9：向数据库插入数据涉及的文件

当用户在界面点击“选择目录”文本框后面的“...”按钮选择路径后，提交文件夹目录路径后，后台代码开始执行如下操作：（在工程文件夹右键点击buid path选择项，添加jdbc包后）通过代码连接数据库，主要通过以下的循环来完成插入(insert)语句：

**for**(**int** i=10;i<=48;i++){

temppath=fpath+"\\Image"+i+".jpg";

temppath=temppath.replaceAll("\\\\","\\\\\\\\");

ttpath=tpath+i+".txt";

ttpath=ttpath.replaceAll("\\\\","\\\\\\\\");

sql = "insert into test(id,path,texturepath)VALUES('"+i+"','"+temppath+"','"+ttpath+"')";

statement.executeUpdate(sql);

}

上述代码中，值得注意的有两个地方，分别是：1.当在数据库中存储路径是不能单纯的用\，否则存入数据库中的地址将不会显示’\’而显示空格，需要调用语句temppath=temppath.replaceAll("\\\\","\\\\\\\\"); 方能正确存储文件路径。2.需要特别注意需要插入变量名的sql的insert语句中，变量名需要用’”+i+”’。

**3.4.3 图像颜色特征相似度比较**

主要涉及的文件：

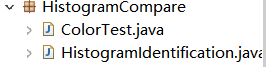


图10：颜色特征比较涉及的主要文件

当用户在界面提交需要比较的图片后，获取该图片地址，当用户再次点击“颜色”按钮时执行searchImage()函数，在searchImage()函数中调用Colortest()方法，在colortest()方法中调用HistogramIdentification.java中的方法来完成计算并提取颜色特征值，通过颜色特征值完成相似度比较。

降低灰度的级数，用一维直方图表示，如将256级的灰度降至16级，可用12位的int表示灰度值，前4位表示red,中间4们表示green,后面4位表示blue。RGB像素的单一直方图SH表示稍微复杂点，每个颜色的值范围为0 ~ 255之间的，假设可以分为一定范围等份，当8等份时，每个等份的值范围为32， 16等份时，每个等份值范围为16，当4等份时候，每个等份值的范围为64，假设RGB值为(14, 68, 221), 16等份之后，它对应直方图索引值(index)分别为: (0, 4, 13), 根据计算索引值公式:index = R + G\*16 + B\*16\*16对应的直方图index = 0 + 4\*16 + 13 \* 16 \* 16， SH[3392] += 1。

关键代码：

/\*\*

\* 求一维的灰度直方图

\* **@param** img

\* **@return**

\*/

**public** **double**[] getHistogram(BufferedImage img) {

**int** w = img.getWidth();

**int** h = img.getHeight();

**int** series = (**int**) Math.*pow*(2, ***GRAYBIT***); //GRAYBIT=4;用12位的int表示灰度值，前4位表示red,中间4们表示green,后面4位表示blue

**int** greyScope = 256/series;

**double**[] hist = **new** **double**[series\*series\*series];

**int** r, g, b, index;

**int** pix[] = **new** **int**[w\*h];

pix = img.getRGB(0, 0, w, h, pix, 0, w);

**for**(**int** i=0; i<w\*h; i++) {

r = pix[i]>>16 & 0xff;

r = r/greyScope;

g = pix[i]>>8 & 0xff;

g = g/greyScope;

b = pix[i] & 0xff;

b = b/greyScope;

index = r<<(2\****GRAYBIT***) | g<<***GRAYBIT*** | b;

hist[index]++;

}

**for**(**int** i=0; i<hist.length; i++) {

hist[i] = hist[i]/(w\*h);

//hist[i] = (float)Math.round(hist[i]\*1000)/1000;

}

**return** hist;

}

/\*\*

\* 基于一维灰度直方图特征的图像匹配（巴氏系数）

\* **@param** histR

\* **@param** histD

\* **@return**

\*/

**public** **static** **double** identification(**double**[] histR, **double**[] histD) {

**double** p = (**double**) 0.0;

**for**(**int** i=0; i<histR.length; i++) {

p += Math.*sqrt*(histR[i]\*histD[i]);

}

p = (**double**)Math.*round*(p\*1000)/1000;

**return** p;

}

**public** **double**[] getCharacteristic(String srcPath) **throws** IOException {

//System.out.println("hhh"+srcPath);

BufferedImage img

= ImageIO.*read*(**new** File(srcPath));

**return** getHistogram(img);

}

3.4.4图像纹理特征比较

主要涉及的文件：

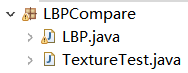


图11：纹理特征比较涉及的主要

当用户在界面提交需要比较的图片后，获取该图片地址，当用户再次点击“颜色”按钮时执行searchImage()函数，在searchImage()函数中调用TextureCompare()方法，在TextureCompare()方法中调用LBP.java中的方法来完成计算并提取纹理特征值，通过纹理特征值完成相似度比较。

关键代码：

(1)图像灰度化的代码和颜色特征比较基本相同此处不再重复列出。

(2)首先介绍什么是uniform，它是指均匀环形结构内包含非常少的空间转换。我们定义U(pattern),用来记录空间转换的数量，即0-1变化的次数。

0-1变化少于等于两次的二进制码形成的LBP看成是一个 uniform LBP ,比如说11000011，01变化次数是两次。就是一个uniform LBP。

/\*\*

\* 计算跳变次数

\* **@param** i

\* **@return**

\*/

**private** **static** **int** getHopCount(**int** i)

{

**int**[] a = **new** **int**[8];

**int** cnt = 0;

**int** k = 7;

**while**(i > 0)

{

a[k] = i & 1;

i = i >> 1;

--k;

}

**for**(k = 0; k < 7; k++)

{

**if**(a[k] != a[k+1])

{

++cnt;

}

}

**if**(a[0] != a[7])

{

++cnt;

}

**return** cnt;

}

/\*\*

\* 对256个特征值进行分组

\* groupNums的下标为特征值，value为组号

\* **@return**

\*/

**private** **static** **int**[] groupFeatureValue(){

**int**[] groupNums = **new** **int**[256];

**int** num = 1;

**for**(**int** i = 0; i <= 255; i++){

**if**(*getHopCount*(i) <= 2){

groupNums[i] = num;

num++;

}

}

**return** groupNums;

}

（3）为了降低LBP的编码模式，对同一编码模式经旋转(循环位移，按位旋转)后产生的编码结果编码为同一值，即这些旋转结果中的最小值。

/\*\*

\* 旋转不变性

\* 此处的feature的二进制位数固定为8

\* **@param** feature

\* **@return**

\*/

**private** **static** **int** getMinFeature(**int** feature){

**int** minFeature = feature;

**for**(**int** i = 0; i < 7; i++){

// 循环右移一位

feature = (feature>>1 | feature<<7) & 0xff;

**if**(feature < minFeature) minFeature = feature;

}

**return** minFeature;

}

（4）通过计算LBP算子，得到纹理特征值。原始的LBP算子定义为在3\*3的窗口内，以窗口中心像素为阈值，将相邻的8个像素的灰度值与其进行比较，若周围像素值大于中心像素值，则该像素点的位置被标记为1，否则为0。这样，3\*3领域内的8个点可产生8bit的无符号数，即得到该窗口的LBP值，并用这个值来反映该区域的纹理信息。

**public** **static** **int**[] getFeatureVector(String imagePath) **throws** IOException{

// 获取灰度矩阵

**int**[][] grayMatrix = *getGrayPixel*(imagePath,100,100);//756\*504

// 为特征值(0-255)分组(降维)

**int**[] groupNums = *groupFeatureValue*();

// 遍历像素点，计算其特征值，并确定其分组，并进行分组统计

**int**[] vector = **new** **int**[59];

**for**(**int** i = 1; i < grayMatrix.length - 1; i++){

**for**(**int** j = 1; j < grayMatrix[0].length - 1; j++){

**int** center = grayMatrix[i][j];

**int** feature = 0;

feature = grayMatrix[i - 1][j - 1]>=center ? (feature<<1)+1 : (feature<<1);

feature = grayMatrix[i][j - 1]>=center ? (feature<<1)+1 : (feature<<1);

feature = grayMatrix[i + 1][j - 1]>=center ? (feature<<1)+1 : (feature<<1);

feature = grayMatrix[i + 1][j]>=center ? (feature<<1)+1 : (feature<<1);

feature = grayMatrix[i + 1][j + 1]>=center ? (feature<<1)+1 : (feature<<1);

feature = grayMatrix[i][j + 1]>=center ? (feature<<1)+1 : (feature<<1);

feature = grayMatrix[i - 1][j + 1]>=center ? (feature<<1)+1 : (feature<<1);

feature = grayMatrix[i - 1][j]>=center ? (feature<<1)+1 : (feature<<1);

feature = *getMinFeature*(feature);

vector[groupNums[feature]]++;

}

}

**return** vector;

}

//计算相似度

 (4)

**public** **static** **double** calculateSimilarity(**int**[] vector, **int**[] vector1) {

**double** len = 0, len1 = 0, numerator = 0;

**for** (**int** i = 0; i < vector.length; i++) {

len += Math.*pow*(vector[i], 2);

len1 += Math.*pow*(vector1[i], 2);

numerator += vector[i] \* vector1[i];

}

len = Math.*sqrt*(len);

len1 = Math.*sqrt*(len1);

//System.out.println(len1);

**return** numerator / (len \* len1);

}

1. 实验结果分析

对于色彩鲜艳的图片，通过分析颜色特征比较相似度，搜索到的图片的色彩分布基本相近，而LBP基于局部纹理特征值比较相似度，结果并不十分令人满意，如图12和图13。LBP更适合于人脸识别。

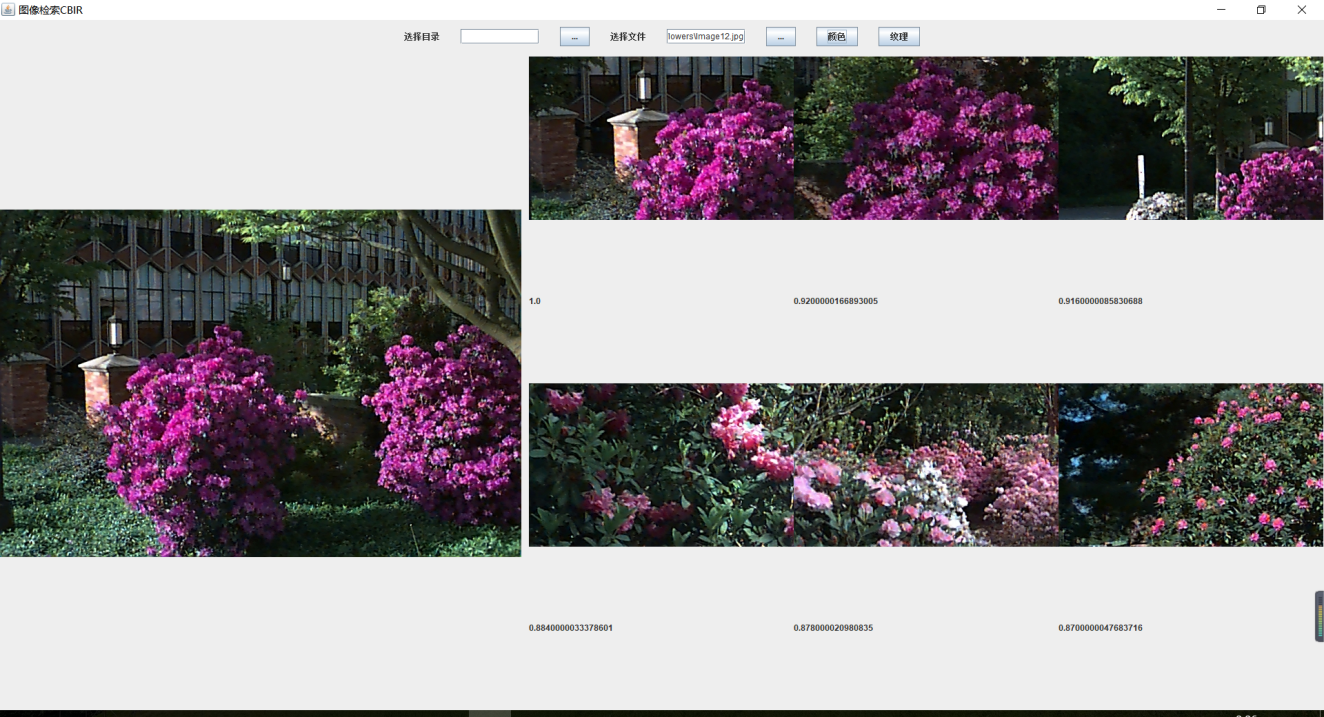


图12：颜色分析结果一

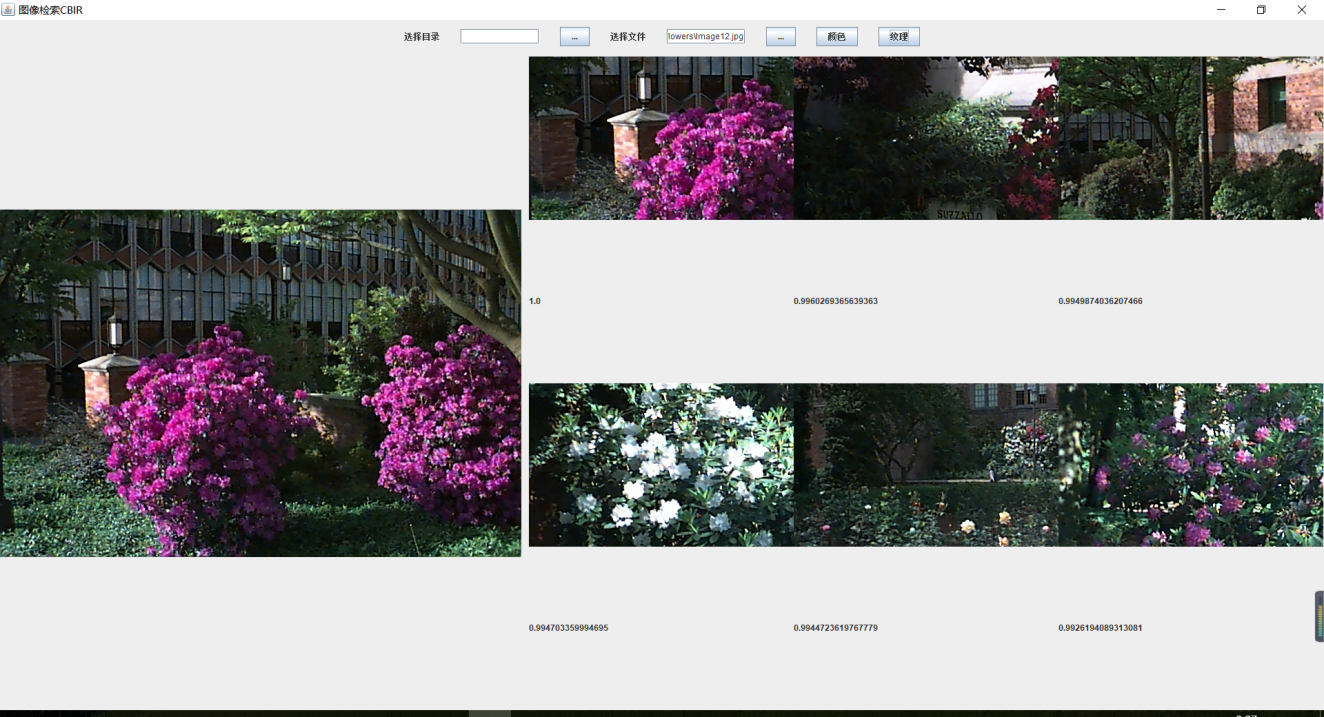


图13：纹理分析结果一

1. 实训总结和心得

1.通过本次实训，加强了java编程的动手能力。在编程过程中，许多细节值得注意。比如使用sql的insert语句插入变量时，eclipse提示数据库错误，把sql语句放在mysql中也能正确执行，通过上网查阅资料后发现，在java中调用要用'"+i+"'将其包裹起来。

2.本次项目涉及到路径的读取存储等，需要注意在java中路径中的分隔符需要使用/或者\\，而当读取到路径将其存入数据库时需要temppath.replaceAll("\\\\","\\\\\\\\"); 使得路径能够正确的插入数据库的表中。

3.在本次项目中，频繁使用到了函数getRGB(int startX, int startY, int w, int h, int rgbArray, int offset, int scansize) 容易且高效地提取图像的像素值。这个方法可以将图像中矩形区域的像素数据传输到一个整数数组中。startX, startY 是要提取的区域左上角图像的坐标；w, h 是要提取的区域的宽度和高度；rgbArray 是接收像素值的整数数组；offset 是数组中接收第一个像素值的位置的索引。

4.还有一个经常出现的问题是“空指针异常问题”，因为涉及到文件调用每次出错的地方都涉及几个文件，所以排除“空指针异常”出错位置，比较方便的是打印值，通过打印可以清楚的看到究竟是值出了问题。再有就是“数组越界”问题，需要我们在设置循环遍历数组时严格把控其边界。

5.在Javaproject中组织文件尽量不要用默认包，应使用MVC模式组织文件。在完成这次项目的过程中，中途因为初期所有文件都放在默认包里，没有建立其他包分类，在调用方法时也不利于条理清楚的组织自己的逻辑，在意识到这个问题后进行建包分类，一些java文件里的方法要按照逻辑拆分浪费了很多时间。所以在建立project时应该有一个良好的意识和习惯。

6.通过本次实训，接触到了数字图像处理这一学科，通过动手实践对图像特征相似度算法有了一定的了解，巩固了关于java文件读写，sql数据库查询语言的知识，以及独立完成前台界面，后台运算逻辑，数据库等使自己对小项目的开发有了更加清楚的实施思路。同样也发现自己还有很多不足，如对java语言编写界面还不是很熟悉，做出的界面只是实现了基本的功能儿没有达到美观的效果。对图像特征值的算法了解不够深入，仅是皮毛。实现的图像检索系统并没有考虑其检索效率。希望在日后自己能够加强学习，尽力弥补这些不足。

1. 参考文献

[1] gloomyfish.图像处理之相似图片识别（直方图应用篇）[EB/OL].

<http://blog.csdn.net/jia20003/article/details/7771651#comments>，2012-07-22/2017-07-06

[2] zouxy09.目标检测的图像特征提取之（二）LBP特征[EB/OL]. <http://blog.csdn.net/zouxy09/article/details/7929531>，2012-08-31/2017-07-06

[3]djh512.LBP纹理特征[EB/OL].

<http://blog.csdn.net/djh512/article/details/9001518>，2013-05-31/2017-07-06