《数字图像处理》课程项目

ICCB 项目报告

组号：4. 小组成员：

刘洋 10848316 leoyonn@gmail.com

刘守君 10848315 liushoujun@icst.pku.edu.cn

钟原 10848362 zhongyuan@icst.pku.edu.cn

2009 年 5 月 14 日

目录

1. 【系统分析与设计】 2
2. 【工作阶段划分与项目分工】 3
3. 【详细设计及实现】 4
3.1 特征提取 4
3.1.1 边缘直方图（EDH） 5
3.1.2 颜色直方图（CCH） 6
3.1.3 Gabor纹理特征 7
3.1.4 Hu不变矩及扩展 8
3.1.5 灰度共生矩阵GLCM及扩展 11
3.1.6 Sift和PCASift及其变种 15
3.2 分类器的设计与实现 19
3.2.1 支持向量机（SVM） 19
3.2.2 K最近邻（KNN） 20
3.2.3 特征融合 20
【界面设计及实现】 21
3.3 界面设计 21
3.4 界面实现 21
4. 系统整合及各模块之间的联系 24
5. 【分类结果演示】 24
5.1 分类结果分析 24
5.2 多种结果方式查看 25
5.2.1 分类结果文本： 25
5.2.2 主界面分类类别列表与缩略图列表 25
5.2.3 Confusion Matrix方式查看 26
6. 【项目总结】 27
【附录 1】. 项目环境配置 28
【附录 2】. 项目文件层级结构 29

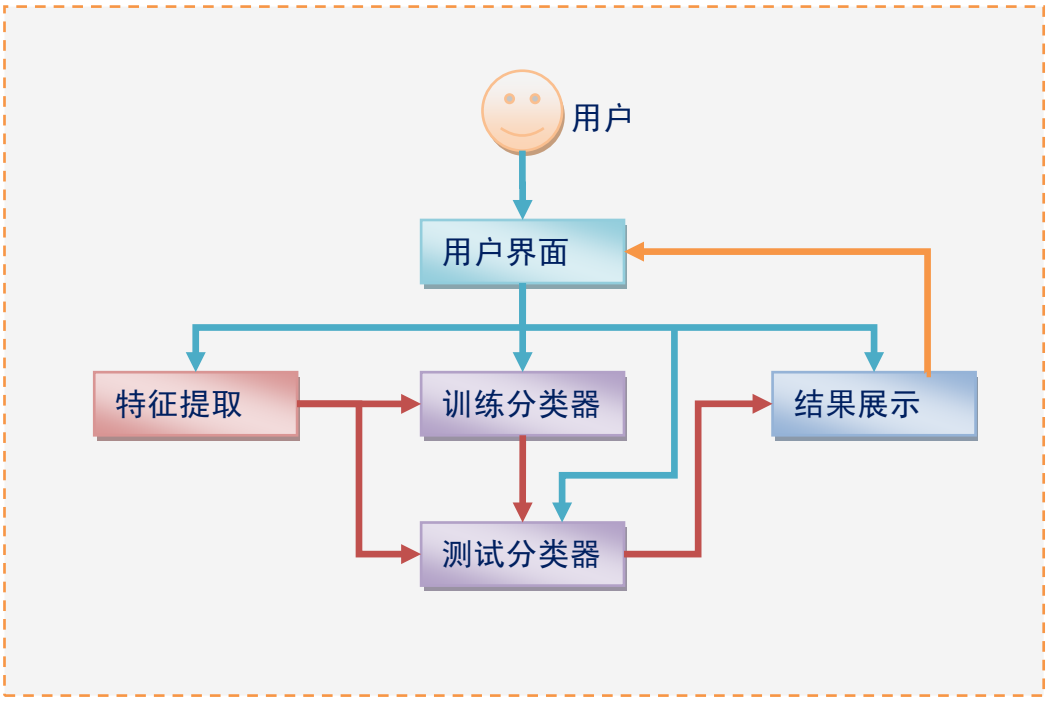
1. 【系统分析与设计】

本项目是实现一个基于内容的图像分类/检索系统。基于内容的图像检索 CBIR 是图像分析的一个研究领域，其目的是在给定查询图像的前提下，依据内容信息或指定查询标准，在图像数据库中搜索并查找出符合查询条件的相应图片。CBIR 具有着很广泛的应用前景，目前各大搜索引擎都推出相应的图片搜索功能，但是这种搜索主要是基于图片的文件名或者网页上相关的文字信息建立索引来实现查询功能，这样检索到的图片也许根本不是用户想要的。为了能让搜索与图片中所体现的内容紧密关联起来，就系要基于内容的图像检索，即查询条件本身就是一个图像，或者是对于图像内容的描述，它建立索引的方式是通过提取底层特征，然后通过计算比较这些特征和查询条件之间的距离，来决定两个图片的相似程度，以达到最终的检索以及分类的目的。

作为这样一项技术的系统实现，我们需要综合考虑到的就是课程要求、可能用户需求等多个方面，对我们的设计与实现过程提出如下要求：

1. 从界面上要一目了然、易于使用，且要同时保证操作上的简单以及视觉上的美观；
2. 从功能上要保证系统所设计的功能的齐全，运行流程完备；
3. 从系统角度来说就要尽力保证程序的健壮性、高效性等特点。

结合这些方面，我们的所设计的系统结构图如下：



【图】 1　　系统设计架构图

从【图】1. 系统设计架构图中可以看出，整个系统大致分为五个模块：

1. 用户界面模块：

- ✓ 负责用户与程序之间的交互；
- ✓ 负责结果的输出展示
- ✓ 负责及其他各模块之间的协调；

2. 特征提取模块：

- ➢ 负责对图像训练集与测试集分别提取特征，这一模块又可以根据图像特征的不同分为如下几个小的模块：
- ➢ 边缘直方图特征提取模块（EDH）；
- ➢ 颜色直方图特征提取模块（CCH）；
- ➢ Gabor 纹理特征提取模块；
- ➢ 灰度共生矩阵特征提取模块（GLCM）；
- ➢ Hu 不变矩特征提取模块（IM）；
- ➢ SIFT 特征提取模块
- ➢ 其他……

3. 分类器训练模块：

对图像训练集特征提取的结果进行分类器的训练；

4. 分类器测试模块：

结合测试集图像的特征提取结果，以及分类器训练的结果，对图像测试集进行测试；

5. 结果展示模块：

将图像分类分类结果通过各种不同的方式反馈给用户，例如平均精确率、Confusion Matrix 等，以及为用户界面所显示的结果提供数据与信息。

2. 【工作阶段划分与项目分工】

根据对系统的分析与初步设计，我们把任务分为如下四个阶段：

- ➢ 第一阶段，组内各成员进行相关的资料搜集与调研、学习；
- ➢ 第二阶段，根据调研结果进行代码的初步编写，这一步主要是做实验；
- ➢ 第三阶段，项目功能整合，将特征提取与分类器的训练、测试结合起来，反复进行实验与验证；
- ➢ 第四阶段，代码整合、界面设计实现以及文档的编写整理。

项目实际分工如下表所示：

任务名称		参与组员		
系统分析设计	系统分析	刘洋	刘守君	钟原
	系统初步设计	刘洋	刘守君	
特征提取	边缘直方图（EDH）			钟原
	颜色直方图（CCH）			钟原
	Gabor 纹理特征			钟原
	Hough 变换	刘洋		
	灰度共生矩阵及扩展	刘洋		
	Hu 不变矩及扩展	刘洋		
	Sift 和 PCASift 及其变种	刘洋	刘守君	
分类器设计实现	支持向量机 SVM		刘守君	
	K 最近邻 KNN		刘守君	
	特征融合		刘守君	
界面设计实现	界面设计	刘洋		
	界面编码实现	刘洋		
系统整合	结果计算及分析	刘洋		
	系统整合	刘洋		
文档撰写	程序说明文档	刘洋	刘守君	
	项目报告文档	刘洋	刘守君	钟原
	文档整合排版	刘洋		

说明：文档由小组各成员分别编写自己所涉及的部分，系统设计、结果评测、附录等部分的撰写以及文档的排版与整合由刘洋完成。

3. 【详细设计及实现】

3.1 特征提取

比较常用的图像特征有颜色特征、纹理特征、形状特征、空间关系特征等，接下来对这些特征做一简短的介绍：

- 颜色特征：

颜色特征是一种全局特征,描述了图像或图像区域所对应的景物的表面性质。由于颜色对图像或图像区域的方向、大小等变化不敏感，所以颜色特征不能很好地捕捉图像中对象的局部特征。其中颜色直方图是最常用的表达颜色特征的方法，

其优点是不受图像旋转和平移变化的影响，进一步借助归一化还可不受图像尺度变化的影响，但是没有表达出颜色空间分布的信息。颜色特征常用的提取与匹配方法包括直方图相交法、距离法、中心距法、累加颜色直方图法等。此外，还有颜色集、颜色矩、颜色聚合向量、颜色相关图等特征提取方法。

- 纹理特征：

纹理特征也是一种全局特征，描述了图像或图像区域所对应景物的表面性质。但纹理特征不是基于像素点的特征，它需要在包含多个像素点的区域中进行统计计算。作为一种统计特征，纹理特征常具有旋转不变性，并且对于噪声有较强的抵抗能力。纹理特征的一个很明显的缺点是当图像的分辨率变化的时候，所计算出来的纹理可能会有较大偏差。在检索具有粗细、疏密等方面较大差别的纹理图像时，利用纹理特征是一种有效的方法。常用的纹理特征的提取主要有：灰度共生矩阵、Tamura 纹理特征、自回归纹理模型、小波变换等。

- 形状特征：

各种基于形状特征的检索方法都可以比较有效地利用图像中感兴趣的目标来进行检索，但目前基于形状的检索方法还缺乏比较完善的数学模型,而且许多形状特征仅描述了目标局部的性质，要全面描述目标常对计算时间和存储量有较高的要求。形状特征有两类表示方法，一类是轮廓特征，另一类是区域特征。常用的提取方法有边界特征法、傅里叶形状描述符法、几何参数法形状、形状不变矩法、小波描述符等。

- 空间关系特征

空间关系是指图像中分割出来的多个目标之间的相互的空间位置或相对方向关系，包括连接关系、重叠关系和包含关系等。空间关系特征的使用可加强对图像内容的描述区分能力，但空间关系特征常对图像或目标的旋转、反转、尺度变化等比较敏感。一般情况下，空间关系特征需要其它特征来配合使用。

由于时间和精力有限，我们不能把各种特征都提取作实验，只能选取如下几种比较常用的特征以及能达到较好的检索效果的特征。

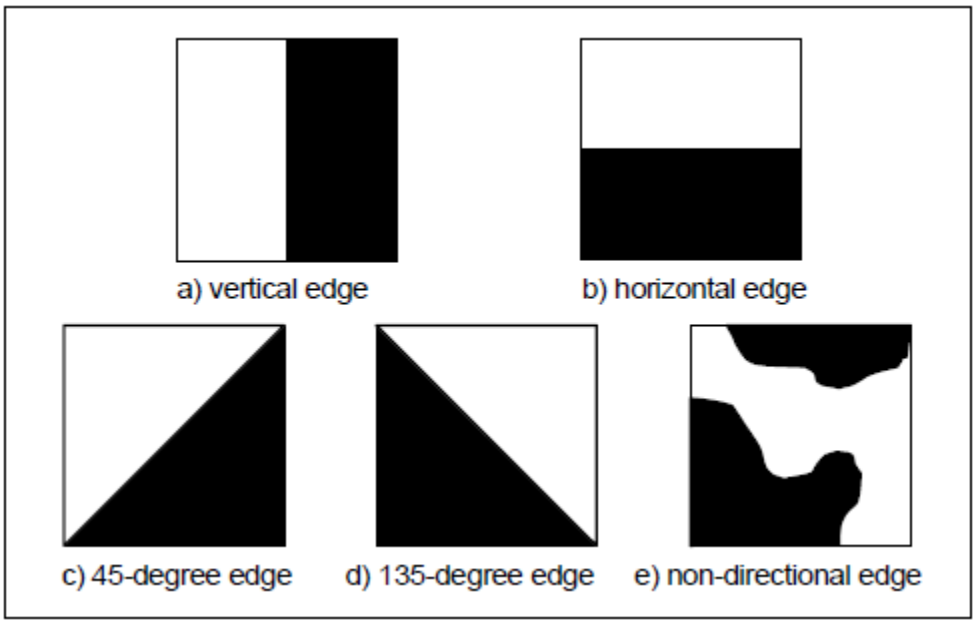
3.1.1 边缘直方图（EDH）

边缘直方图的基本思路是对于一幅灰度图像的每个像素，计算出在这像素上的边缘方向值，并量化到指定的边缘方向，最后统计出各个方向像素数目，得到直方图。具体到我的工作中，可以分为以下几个步骤：

1. 决定边缘方向数目。在这里，我选择了 5 个方向，即水平、垂直、45 度、135 度和无边缘方向，如下页图中所示。
2. 将彩色图像转为灰度图。这个很容易，使用 Cximage 读取像素的 RGB 分量，取其平均数作为灰度即可。
3. 求得每个像素处的边缘方向。我在程序中采用了 sobel 算子，经过对邻近像素的计算得到了边缘的梯度值；之后对梯度值进行反正弦变换，即可粗略得到边缘方向。
4. 将求得的角度量化到指定的 5 个边缘方向中。
5. 统计直方图数据。在这里，如果对于整幅图像进行统计，那么只能得到

5 个特征，过于稀少而不具体有足够的区分度。因此，我将一幅图像分成了 4*4 共 16 个亚图像块，分别对这 16 个块进行单独的直方图统计，这样，原始的一幅图像能得到 5*16 共 80 个特征。

至此，依据图像的边缘直方图，就可以得出一个 80 维的特征向量了。



测试结果：根据 EHD 得到的特征向量，在识别中有 41%的正确率。

3.1.2 颜色直方图（CCH）

这个特征提取相对来说比较简单。将 HSV 颜色空间划分为若干个区域，统计图像中落在每个区域里的像素数即可。我的工作中一些细节如下：

1. 划分区域：将 H 分量划分成 16 个区域，S 和 V 分量各划分成 4 个区域。这样，在 HSV 颜色空间中，一共划分出了 16*4*4 共 256 个区域。
2. 将原图像转化到 HSV 图像空间中。使用 Cximage 读取出每个像素的 RGB 值，再根据 RGB 转化到 HSV 的公式计算出 H、S 和 V 的值。
3. 量化并进行统计，得出落在 256 个区域中的像素数，形成 256 个特征。
4. 进行平滑。将本来连续的空间强行割裂进行统计，导致了一个弊病：在肉眼看来很接近的颜色，很可能被划分到完全不同的区域中；这样，本来相似的像素点，被划分到不同区域中后，相似性就完全失去了。为了减轻这一问题带来的后果，我对得到的直方图进行平滑，即每个区域中的数据，对附近的两个区域再带来 20%的贡献，这样可以在一定程度上弥补失去的相似性。

经过实验，平滑之后的效果要比之前好。

测试结果：在识别中达到了 43.80% 的识别率

3.1.3 Gabor 纹理特征

Gabor 变换属于加窗傅立叶变换，Gabor 函数可以在频域不同尺度、不同方向上提取相关的特征。另外 Gabor 函数与人眼的生物作用相仿，所以经常用作纹理识别上，并取得了较好的效果。二维 Gabor 函数可以表示为：

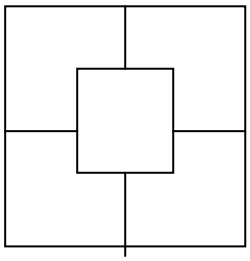
$$\psi_{\mu,\nu}(z)=\frac{\|k_{\mu,\nu}\|^2}{\sigma^2}e^{-\frac{\|k_{\mu,\nu}\|^2\|z\|^2}{2\sigma^2}}\left[e^{ik_{\mu,\nu}z}-e^{-\frac{\sigma^2}{2}}\right]$$

其中，$z=(x,y)$是图像中的点，(x, y)是点的坐标。这里 v 的取值决定了 Gabor 滤波核函数的波长，而 u 的取值则表示核函数的方向。

可以看到，波长数与方向数的乘积，即为变换中所有核函数的数目。在这里，我设置了 4 个波长和 8 个方向，故对一个图像来说，这个变换能生成 32 个和原图像大小一致的变换结果。

特征提取的过程：

1. 将彩色图像变为灰度图：这和 EHD 部分的方法一致，不再重述；
2. 将得到的灰度图进行 Gabor Wavelet 变换：在这一部分，我采用了从网络上下载来的用 C 语言写成的源代码，该源代码的下载地址为：http://www.pudn.com/downloads158/sourcecode/graph/texture_mapping/detail702702.html
3. 如上所述，得到了 32 个和原图像大小一致的变换。对于每一个变换，进行计算；
4. 分块：把每个图像变换划分成如下图所示的 5 块，求出每一块中数据的平均值与方差。



5．取得特征值：这样，对于一个变换，我们能得到 10 个特征；而对于 32 个变换，共能得到 320 个特征。

测试结果：在测试中达到了 31%的正确率。

3.1.4 Hu 不变矩及扩展

3.1.4.1 Hu 的七个不变矩

首先，形状无关矩（Moment Invariants）是基于区域的物体形状表示方法。假设 R 是用二值图像表示的物体，则 R 形状的第 p+q 阶中心矩为：

$$\mu_{p,q} = \sum_{(x,y)\in R} (x-x_c)^p (y-y_c)^q$$

其中（x_c, y_c）是物体的中心。为获得缩放无关的性质，可以对该中心矩进行标准化操作：

$$\eta_{p,q} = \frac{\mu_{p,q}}{\mu_{0,0}^{\gamma}}, \quad \gamma = \frac{p+q+2}{2}$$

基于这些矩，Hu[1]提出了一系列分别具有变换、旋转和缩放无关性的7个矩：

$$\left.\begin{aligned}
\phi_1 &= \mu_{2,0} + \mu_{0,2} \\
\phi_2 &= (\mu_{2,0} - \mu_{0,2})^2 + 4\mu_{1,1}^2 \\
\phi_3 &= (\mu_{3,0} - 3\mu_{1,2})^2 + (\mu_{0,3} - 3\mu_{2,1})^2 \\
\phi_4 &= (\mu_{3,0} + \mu_{1,2})^2 + (\mu_{0,3} + \mu_{2,1})^2 \\
\phi_5 &= (\mu_{3,0} - 3\mu_{1,2})(\mu_{3,0} + \mu_{1,2})\left[(\mu_{3,0} + \mu_{1,2})^2 - 3(\mu_{0,3} + \mu_{2,1})^2\right] \\
&\quad + (\mu_{0,3} - 3\mu_{2,1})(\mu_{0,3} + \mu_{2,1})\left[(\mu_{0,3} + \mu_{2,1})^2 - 3(\mu_{3,0} + \mu_{1,2})^2\right] \\
\phi_6 &= (\mu_{2,0} - \mu_{0,2})[(\mu_{3,0} + \mu_{1,2})^2 - (\mu_{0,3} + \mu_{2,1})^2] + 4\mu_{1,1}(\mu_{3,0} + \mu_{1,2})(\mu_{0,3} + \mu_{2,1}) \\
\phi_7 &= (3\mu_{2,1} - \mu_{0,3})(\mu_{3,0} + \mu_{1,2})\left[(\mu_{3,0} + \mu_{1,2})^2 - 3(\mu_{0,3} + \mu_{2,1})^2\right] \\
&\quad + (\mu_{3,0} - 3\mu_{2,1})(\mu_{0,3} + \mu_{2,1})\left[(\mu_{0,3} + \mu_{2,1})^2 - 3(\mu_{3,0} + \mu_{1,2})^2\right]
\end{aligned}\right\}$$

矩特征主要表征了图像区域的几何特征,又称为几何矩， 由于其具有旋转、平移、尺度等特性的不变特征，所以又称其为不变矩。在图像处理中，几何不变矩可以作为一个重要的特征来表示物体，可以据此特征来对图像进行分类等操作。矩在统计学中被用来反映随机变量的分布情况，推广到力学中，它被用作刻画空间物体的质量分布。同样的道理，如果我们将图像的灰度值看作是一个二维或三维的密度分布函数，那么矩方法即可用于图像分析领域并用作图像特征的提取。

[1] M. K. Hu, “Visual pattern recognition by moment invariants,” in J. K. Aggarwal, R. O. Duda, and A. Rosenfeld, Computer Methods in Image analysis, IEEE computer Society, Los Angeles, CA, 1977]

3.1.4.2 Hu 的 7 个不变矩的实现

根据 Hu 的 7 个不变矩，很快就可以编码实现。
首先给出计算中心坐标的函数：

```
for (i = 0; i < m_iImgHeight; i ++)
    for (j = 0; j < m_iImgWidth; j ++)
    {
        if(m_pImageData[i][j] == 255)
            m += m_pImageData[i][j]*pow(j,p)*pow(i,q);
    }
```

然后计算出点（0，0）的μ值：

```
m_mx = (int) (ComputeM(1,0) / ComputeM(0,0));
m_my = (int) (ComputeM(0,1) / ComputeM(0,0));
m_lMui00 = (LONG) Miu(0, 0);
```

接着计算每一个η值：

```
YiTa=Miu(p , q) / pow(m_lMui00, r);
```

最后一步就是求解这 7 个不变矩：

```
//计算不变矩1-7
m_dIM[0] = yita20+yita02;
m_dIM[1]=pow((yita20-yita02),2)+4*pow(yita11,2);
m_dIM[2]=pow((yita30-3*yita12),2)+pow((3*yita21-yita03),2);
m_dIM[3]=pow((yita30+yita12),2)+pow((yita21+yita03),2);
m_dIM[4]=(yita30-3*yita12)*(yita30+yita12)*(pow((yita30+yita12),2)
        -3*pow((yita21+yita03),2))+(3*yita21-yita03)*(yita21+yita03)
         *(3*pow(yita30+yita12,2)-pow((yita21+yita03),2));
m_dIM[5]=(yita20-yita02)*(pow((yita30+yita12),2)
        -pow((yita21+yita03),2))+4*yita11*(yita30+yita12)*(yita21+yita03);
m_dIM[6]=(3*yita21-yita03)*(yita30+yita12)*(pow((yita30+yita12),2)
            -3*pow((yita21+yita03),2))+(3*yita12-yita30)*(yita21+yita03)
             *(pow((3*yita30+yita12),2)-pow((yita21+yita03),2));
```

但是经过 libsvm 测试，使用这组特征对图片进行分类结果并不是很好，还不到 20%，（我还因此向老师和助教师兄发信询问有关数据的情况，以确定算法的正误），在排除算法上的错误以后，认为这组特征效果并不佳，一方面的原因是数据维度太小。于是我查了一些文章，找到篇对 Hu 不变矩进行扩展的方法，在下一小节详细介绍。

3.1.4.3 对 Hu 不变矩的扩展及实现

前面说到 Hu 的 7 个不变矩产生的结果并不理想，于是找到一片文章，将 Hu 不变矩 M1~M7 扩展为 R1~R10 共十个新的特征值，而且经过证明这十个特征值也具有结构平移、缩放和旋转不变性等特征。R1 ~ R10 的计算公式如下所示：

$$R_1 = \frac{\sqrt{M_2}}{M_1};\ R_2 = \frac{M_1 + \sqrt{M_2}}{M_1 - \sqrt{M_2}};$$

$$R_3 = \frac{\sqrt{M_3}}{\sqrt{M_4}};\ R_4 = \frac{\sqrt{M_3}}{\sqrt{|M_5|}};$$

$$R_5 = \frac{\sqrt{M_4}}{\sqrt{|M_5|}};\ R_6 = \frac{|M_6|}{M_1 \times M_3};$$

$$R_7 = \frac{|M_6|}{M_1 \times \sqrt{|M_5|}};\ R_8 = \frac{|M_6|}{M_3 \times \sqrt{|M_2|}}$$

$$R_9 = \frac{|M_6|}{\sqrt{M_2 \times |M_5|}};\ R_{10} = \frac{|M_5|}{M_3 \times M_4}。$$

根据公式很容易编码实现得到新的特征值：

```
//计算不变矩扩展变种（10个）
m_dExIM[0] = sqrt(fabs(m_dIM[1])) / m_dIM[0];
m_dExIM[1] = (m_dIM[0] + sqrt(fabs(m_dIM[1])))
             / (m_dIM[0] - sqrt(fabs(m_dIM[1]))) ;
m_dExIM[2] = sqrt(fabs(m_dIM[2])) / sqrt(fabs(m_dIM[3]));
m_dExIM[3] = sqrt(fabs(m_dIM[2])) / sqrt(fabs(m_dIM[4]));
m_dExIM[4] = sqrt(fabs(m_dIM[3])) / sqrt(fabs(m_dIM[4]));
m_dExIM[5] = m_dIM[5] / (m_dIM[0] * m_dIM[2]);
m_dExIM[6] = sqrt(fabs(m_dIM[5])) / (m_dIM[0] * sqrt(fabs(m_dIM[4])));
m_dExIM[7] = sqrt(fabs(m_dIM[5])) / (m_dIM[2] * sqrt(fabs(m_dIM[1])));
m_dExIM[8] = sqrt(fabs(m_dIM[5]))
             / sqrt(fabs(m_dIM[1]) * (fabs(m_dIM[4])));
m_dExIM[9] = fabs(m_dIM[4]) / m_dIM[2] * m_dIM[3];
```

产生新的数据后经过实验，分类正确率结果刚过 20%，仍然是一个比较低的数值，于是又查阅了一些文章和资料，发现较多的研究工作中在使用形状不变矩时进行了边缘检测、二值化等操作。于是我为这组特征的提取添加了 Canny 边缘检测的过程，结果反而更糟糕。经过仔细推敲对这组特征没发现什么更好的解决办法，初步估计是因为这组图片分辨率太小，不适合使用 Hu 不变矩或者其扩展作为特征进行基于内容的图像检索，因此先暂停了对这组特征的提取，把精力转而放在如灰度共生矩阵及 Sift/PCASift 等更加重要的特征上。

3.1.5 灰度共生矩阵 GLCM 及扩展

3.1.5.1 灰度共生矩阵及其特征

灰度共生矩阵（Gray Level CoOccurrence Matrix）是目前公认的一种重要的纹理分析方法，它用两个位置的象素的联合概率密度来定义，它不仅反映亮度的分布特性，也反映具有同样亮度或接近亮度的象素之间的位置分布特性，是有关图像亮度变化的二阶统计特征。它是定义一组纹理特征的基础。一幅图像的灰度共生矩阵能反映出图像灰度关于方向、相邻间隔、变化幅度的综合信息，它是分析图象的局部模式和它们排列规则的基础。

具体来说，灰度共生矩阵是统计空间上具有某种位置关系的一对像元灰度对出现的概率，可以理解为象素对或灰度级对的直方图。这里所说的象素对和灰度级对是有特定含义的，一是象素对的距离不变，二是象素灰度差不变。灰度共生矩阵是反映图像区域微观纹理的有力工具，它按一定的空间关系描述像素点对之间的灰度相关性。灰度共生矩阵描述从图像(x，y)灰度为 i 的像素出发，统计与距离为 δ、灰度为 j 的像素(x+Δx，y+Δy)同时出现的概率 P(i，j，δ，θ)。

当 θ 和 δ 选定时，P(i，j，δ，θ)也可简记为 Pi，j。显然灰度共生矩阵是一个对称矩阵，其阶数由图像中的灰度级数决定，即若灰度级数为 N，则灰度共生矩阵为 N×N 的方阵。通常情况下，选取 θ 为 0 度，45 度，90 度，135 度 4 个方向来计算灰度共生矩阵。δ 的选取与图像有关，一般根据实验确定。在给定方向和距离时，实际常通过计算共现灰度 i 和 j 的像元对数来表示 P(i，j，δ，θ)。

由于灰度共生矩阵的计算量很大，为简便起见，一般采用几个比较常用的特征来提取图像的纹理特征。我查阅了大量的文章和资料，总结了所有涉及到的特征（共找到了 12 个）将其 公式拼在一起如下所示：（这个过程中还有一个小插曲，就是个个文章对同一个特征的名称还不同，我只能通过公式来进行区别）：

1.能量（角二阶矩）：$E = \sum_i \sum_j [P(i,j)]^2$

2.熵：$H = -\{\sum_i \sum_j P(i,j)\log P(i,j)\}$

3.灰度熵：$H_1 = -\{[\sum_i \sum_j P(i,j)]\log[\sum_j P(i,j)]\}$

4.梯度熵：$H_g = -\{\sum[\sum_j P(i,j)]\log[\sum_j P(i,j)]\}$

5.逆差矩（同质度）：$L = \sum_i \sum_j \frac{1}{1+(i-j)^2} P(i,j)$

6.惯性矩（对比度）：$I = \sum_i \sum_j (i-j)^2 P(i,j)$

7.梯度分布不均匀性：$U_g = \{\sum_i [\sum_j F(i,j)]^2\} / [\sum_i \sum_j F(i,j)]$

8.小梯度优势：$T_{min} = [\sum_i \sum_j \frac{F(i,j)}{j^2}] / [\sum_i \sum_j F(i,j)]$

9.大梯度优势：$T_{max} = [\sum_i \sum_j j^2 F(i,j)] / [\sum_i \sum_j F(i,j)]$

10.方差: $$VAR = \sum_{i=1}^{N_g}\sum_{j=1}^{N_g}(i-m)^2 p(i,j,d,\theta)$$

11.均值和: $$SOA = \sum_{k=2}^{2N_g} k \sum_{i=1}^{N_g}\sum_{j=1}^{N_g} p(i,j,d,\theta)$$

12.相关性: $$C = [\sum_i \sum_j (ij\hat{P}(i,j) - \mu_x\mu_y)]/\sigma_x^2\delta_y^2$$

其中在各篇文章中，使用次数较多的几个特征简介如下：

- **能量：**

能量是灰度共生矩阵元素值的平方和，所以也称能量，反映了图像灰度分布均匀程度和纹理粗细度。如果共生矩阵的所有值均相等，则能量值小；相反， 如果其中一些值大而其它值小，则能量值大。当共生矩阵中元素集中分布时，此时能量值大。ASM 能量大表明一种较均一和规则变化的纹理模式。

- **熵：**

熵是图像所具有的信息量的度量，纹理信息也属于图像的信息，是一个随机性的度量，当共生矩阵中所有元素有最大的随机性、空间共生矩阵中所有值几乎相等时，共生矩阵中元素分散分布时，熵较大。它表示了图像中纹理的非均匀程度或复杂程度。

- **相关：**

相关度量空间灰度共生矩阵元素在行或列方向上的相似程度，因此，相关值大小反映了图像中局部灰度相关性。当矩阵元素值均匀相等时，相关值就大； 相反，如果矩阵像元值相差很大则相关值小。如果图像中有水平方向纹理，则水平方向矩阵的相关大于其余矩阵的相关值。

- **惯性矩（对比度）：**

反映了图像的清晰度和纹理沟纹深浅的程度。纹理沟纹越深，其对比度越大，视觉效果越清晰；反之，对比度小，则沟纹浅，效果模糊。灰度差即对比度大的象素对越多，这个值越大。灰度公生矩阵中远离对角线的元素值越大，惯性矩越大。

- **逆差距：**

反映图像纹理的同质性，度量图像纹理局部变化的多少。其值大则说明图像纹理的不同区域间缺少变化，局部非常均匀。

接下来介绍对灰度共生矩阵的实现以及这些特征的提取过程。

3.1.5.2 灰度共生矩阵特征提取的实现

根据前面的描述，灰度共生矩阵可以由转成灰度以后的图片数据来建立。对于 0°、45°、90°、135°四个方向分别建立灰度共生矩阵的实现如下所示：

```
//计算0度的灰度共现阵
for(i=0; i<iImageHeight; i++)
{
    for(j=0; j< iImageWidth - dist; j++)
    {
        m_iMatrix01[(unsigned int)grayImage[i][j]][(unsigned int)
                grayImage[i][j + dist]] += 1;
        m_iMatrix01[(unsigned int)grayImage[i][j + dist]][(unsigned int)
                grayImage[i][j]] += 1;
    }
}
//计算45、90、135度的灰度共现阵与之形式相同
```

然后对于每一个方向，分别计算其 12 个特征值（iDir 表示方向）：

```
for(i = 0; i < iDim; i ++)
{
    for(j = 0; j < iDim; j ++)
    {
        dSumIJ += pdMatrix[i][j];
        m_dFeatures[0][iDir] += pdMatrix[i][j]*pdMatrix[i][j];
        if(pdMatrix[i][j]>1e-12)
        {
            m_dFeatures[1][iDir] -= pdMatrix[i][j]*log(pdMatrix[i][j]);
        }
        m_dFeatures[2][iDir] += (double)(i-ux) * (double)(j-uy) * pdMatrix[i][j];
        m_dFeatures[3][iDir] += pdMatrix[i][j]/(1+(double)(i-j)*(double)(i-j));
        m_dFeatures[4][iDir] += (double)(i-j)*(double)(i-j)*pdMatrix[i][j];
        dTmpFeature5 += pdMatrix[i][j];
        dTmpFeature6 += pdMatrix[j][i];
        if(j > 0)
            m_dFeatures[8][iDir] += pdMatrix[i][j] / (j * j);
        m_dFeatures[9][iDir] += j * j * pdMatrix[i][j];
        m_dFeatures[10][iDir] += (i - dMean) * (i - dMean) * pdMatrix[i][j];

    }
    //省去若干代码
}
```

如下的代码是在调用了 4 个方向的特征计算之后，进行均值和方差的求解：

```
ComputeFeature(nFeature, m_iMatrix01, 0);    //分别是4个方向
ComputeFeature(nFeature, m_iMatrix10, 1);
ComputeFeature(nFeature, m_iMatrix11, 2);
ComputeFeature(nFeature, m_iMatrix1_1, 3);
for(j = 0; j < nFeature; j ++)
{
      m_duFeatures[j] = 0;
      for(i = 0; i < 4; i ++)
            m_duFeatures[j] += m_dFeatures[j][i];
      m_duFeatures[j] /= 4;
}

for(j = 0; j < nFeature; j ++)
{
      m_dsFeatures[j] = 0;
      for(i = 0; i < 4; i ++)
            m_dsFeatures[j] += (m_dFeatures[j][i] - m_duFeatures[j]) *
                               (m_dFeatures[j][i] - m_duFeatures[j]);
      m_dsFeatures[j] /= 4;
      m_dsFeatures[j] = sqrt(m_dsFeatures[j]);
}
```

这样得到的特征是 12 * 4 / 4 * 2 = 24 维。之前我做过一个只有五个特征值的实验，当时的分类准确率在 30 以上，但是这组 24 维的特征试验结果居然不到 20%，我当时直接断定，肯定有些特征求值过程有问题，或者特征本身就有问题，这就需要删减一些特征。

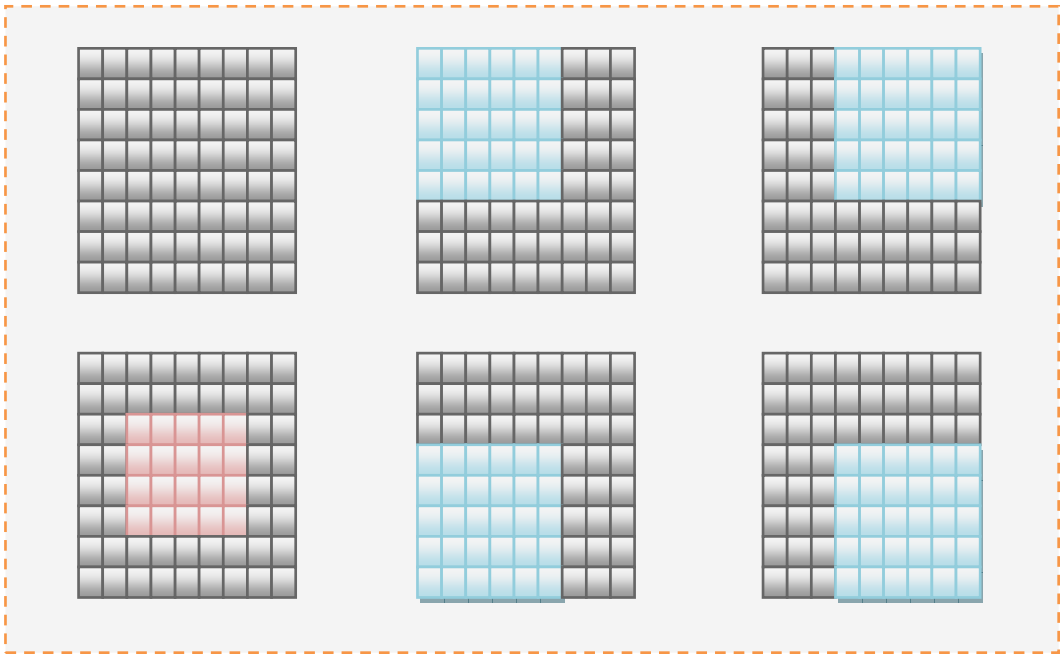
3.1.5.3 灰度共生矩阵的优化及扩展

前面说到对 24 维的灰度共生矩阵提取特征时结果出乎意料的低，我核对了一遍公式和代码并未发现特征值 11、12 的公式有问题，于是直接把这两个维度删除，留下其余的 20 个维度，经过测试，分类的准确率超过了 40%，说明之前的判断是正确的。

然后我又做了进一步的深入查看，发现特征提取的结果文档中前面 15 个维度中，在同一类中的数值都比较相似，而后面的 5 个维度结果数值则跳跃性比较大。这样一来，我直接在程序中手动删除，分类结果又提升了 2 个百分点。

最后考虑对灰度共生矩阵的**分块实现**。由于图片较小， 分块不宜过多；经过调研相关的文献，发现“重合”分块能得到更好的结果，于是我得出这样的一个分块方法：

首先将图片的宽、高都 8 等分，然后如下图的方法划分 5 块，这样的 5 块之间都有交叉：



【图】 2 GLCM 分块方法

最后做测试发现这样分块结果提升了 5 个百分点，达到 47%。这个分类结果也是我们所做的单个特征中正确率最高的。(后面的 PCASift 由于有重复点以及聚类过程较慢等原因，导致分类结果不如这个理想)

3.1.6 Sift 和 PCASift 及其变种

3.1.6.1 Sift 特征提取算法及实现

SIFT 算法是 David G.Lowe 在 2004 年总结了现有的基于不变量技术的特征检测方法的基础上提出的一种基于尺度空间的、对图像缩放、旋转甚至仿射变换保持不变性的图像局部特征描述算子，其全称是 Scale Invariant Feature Transform，即尺度不变特征变换。

SIFT 算法首先在尺度空间进行特征检测，并确定关键点的位置和关键点所处的尺度，然后使用关键点邻域梯度的主方向作为该点的方向特征，以实现算子对尺度和方向的无关性。它选择高斯残差在尺度空间上的极值点为特征点，并计算特征点局部邻域内的梯度方向直方图为描述子。这种算法将图像金字塔结构引入尺度空间以减少计算量，同时针对 128 维的特征向量空间加快搜索过程，取得了较好的效果。SIFT 算法已经在图像匹配、全景拼接和视觉导航等领域得到成功应用。

SIFT 特征匹配算法是目前国内外特征点匹配研究领域的热点与难点，其匹配能力较强，可以处理两幅图像之间发生平移、旋转、仿射变换情况下的匹配问题，甚至在某种程度上对任意角度拍摄的图像也具备较为稳定的特征匹配能力。

SIFT 算法提取的 SIFT 特征向量具有如下特性：

- SIFT 特征是图像的局部特征，其对旋转、尺度缩放、亮度变化保持不变性，对视角变化、仿射变换、噪声也保持一定程度的稳定性；
- 区分度好，信息量丰富，适用于在海量特征数据库中进行快速、准确的匹配；
- 多量性，即使少数的几个物体也可以产生大量 SIFT 特征向量；
- 高速性，经优化的 SIFT 匹配算法甚至可以达到实时的要求；
- 可扩展性，可以很方便的与其他形式的特征向量进行联合；

SIFT 特征匹配算法包括两个阶段：第一阶段是 SIFT 特征的生成，即从多幅待匹配图像中提取出对尺度缩放、旋转、亮度变化无关的特征向量；第二阶段是 SIFT 特征向量的匹配。而具体来说，一幅图像 SIFT 特征向量的生成算法可以偶那个如下四个步骤来概括：

- 1. 尺度空间极值检测，以初步确定关键点位置和所在尺度。
  在检测尺度空间极值时，图中标记为叉号的像素需要跟包括同一尺度的周围邻域 8 个像素和相邻尺度对应位置的周围邻域 9×2 个像素总共 26 个像素进行比较，以确保在尺度空间和二维图像空间都检测到局部极值；
- 2. 通过拟和三维二次函数以精确确定关键点的位置和尺度，同时去除低对比度的关键点和不稳定的边缘响应点，以增强匹配稳定性、提高抗噪声能力。
- 3. 利用关键点邻域像素的梯度方向分布特性为每个关键点指定方向参数，使算子具备旋转不变性。
  如下式为(x,y)处梯度的模值和方向公式，其中 L 所用的尺度为每个关键点各自所在的尺度；

$$m(x,y)=\sqrt{(L(x+1,y)-L(x-1,y))^2+(L(x,y+1)-L(x,y-1))^2}$$

$$\theta(x,y)=a\tan 2((L(x,y+1)-L(x,y-1))/(L(x+1,y)-L(x-1,y)))$$

- （4）生成 SIFT 特征向量。

实现方面由于 sift 代码编写的复杂性，我从网上找到了代码经过修改而成我们所需要的。代码下载链接如下：

SIFT下载地址：http://www.pudn.com/detail.asp?id=391723

下面介绍一下主要的流程及函数：

```
void ComputeImgSift(char *szImgDir, int iImgNum,
                    FILE *foutTrain, FILE *foutTest);
```

在 ComputeImgSift 中，调用 sift 特征提取函数，并按照一定的格式将结果输出到参数中指定的文件里。调用的 sift 提取函数如下：

```
int _sift_features( IplImage* img, struct feature** feat, int intvls,
                    double sigma, double contr_thr, int curv_thr,
                    int img_dbl, int descr_width, int descr_hist_bins )
```

其中 feat 是最终存储特征向量的数组指针，sigma 是高斯平滑的量。

在_sift_features 函数中又依次调用如下函数，首先由图像数据简历高斯尺度空间金字塔；然后建立一个不同的高斯尺度空间（DoG 尺度空间），从这个空间提取特征；接下来就是特征尺度、朝向的计算以及特征描述符的计算，最终将 sfit 特征向量建立：

```
init_img = create_init_img( img, img_dbl, sigma );
octvs = log( MIN( init_img->width, init_img->height ) ) / log(2) - 2;
gauss_pyr = build_gauss_pyr( init_img, octvs, intvls, sigma );
dog_pyr = build_dog_pyr( gauss_pyr, octvs, intvls );
storage = cvCreateMemStorage( 0 ); //64K
features = scale_space_extrema( dog_pyr, octvs, intvls, contr_thr,
                                curv_thr, storage );
calc_feature_scales( features, sigma, intvls );
if( img_dbl )
    adjust_for_img_dbl( features );
calc_feature_oris( features, gauss_pyr );
compute_descriptors( features, gauss_pyr, descr_width, descr_hist_bins );
```

从很多文章中可以发现使用 SIFT 特征能够很好的完成图像的检索匹配与分类，但是经过这个过程提取的结果并不理想，在 30%左右（注：由于不同图像特征点的个数不同，无法使用 svm 支持向量机进行分类训练，我们后面还会加一步 k-means 聚类的过程）。经过查看，发现 SIFT 特征提取的结果里有一些重复的特征，尚未查清这到底是否算法上的问题。但是特征的重复给 k-means 聚类带来困难，导致聚类收敛过程的困难，这一点可以通过去除重复点解决。

3.1.6.2 PCA-Sift 特征提取算法及实现

由于 sift 特征维度较高，计算复杂性较高，且是不完全的仿射不变，后来的文章对其有几种扩展，其中较为著名的一个就是 PCA-SIFT。PCA-SIFT 与 SIFT 有相同的亚像素位置（sub-pixel）、尺度（scale）和主方向（dominant orientations），但在第 4 步计算描述子的时候，它用特征点周围的 41×41 的像斑计算它的主元，并用 PCA-SIFT 将原来的 2×39×39 维的向量降成 20 维，以达到更精确的表示方式。

创建 PCA-SIFT 描述子的步骤为：首先计算或者载入投影矩阵；然后 检测关键点；接着 通过与投影矩阵相乘投影关键点周围的像斑。PCA-SFIT 建立描述子的步骤如下：

- 输入： 在尺度空间关键点的位置和方向；
- 在关键点周围提取一个 41×41 的像斑于给定的尺度，旋转到它的主方向；
- 计算 39×39 水平和垂直的梯度，形成一个大小为 3042 的矢量；
- 用预先计算好的投影矩阵 n×3042 与此矢量相乘；
- 这样生成一个大小为 n 的 PCA-SIFT 描述子；

PCA-SIFT 的优点就是在保留不变性的同时降低维度，大大减少了计算时间。

在实现上，我们的做法是使用 SIFT 的结果作为 PCA-SIFT 的输入，对特征向量进行重新计算。参考的 PCASIFT 代码下载地址：

http://www.pudn.com/downloads79/sourcecode/graph/texture_mapping/detail303262.html

```
void KeypointDetector::RecalcKeys(Image * im, vector <Keypoint *> & keys)
 {
     vector<vector<Image *> > GOctaves = BuildGaussianOctaves(image);
     RecalcKeyIndices(keys);
     ComputeLocalDescr(keys, GOctaves);
}
```

3.1.6.3 特征维数的统一化

前面说到 SIFT 和 PCA-SIFT 的结果对不同的图片提取的特征点数不一样，导致分类器的输入维度不同，无法正常使用 SVM 进行分类训练及测试。例如实验图片组第一类（非洲人民）特征点在 80~100 个左右，而室内台灯一组很多图片提取的特征点数还不到 10 个。我们想到的一个办法就是用 K-means 进行聚类：

K-means 算法接受输入量 k ；然后将 n 个数据对象划分为 k 个聚类以便使得所获得的聚类满足：同一聚类中的对象相似度较高；而不同聚类中的对象相似度较小。聚类相似度是利用各聚类中对象的均值所获得一个“中心对象”（引力中心）来进行计算的。

K-means 算法的工作过程说明如下：首先从 n 个数据对象任意选择 k 个对象作为初始聚类中心；而对于所剩下其它对象，则根据它们与这些聚类中心的相似度（距离），分别将它们分配给与其最相似的（聚类中心所代表的）聚类；然后再计算每个所获新聚类的聚类中心（该聚类中所有对象的均值）；不断重复这一过程直到标准测度函数开始收敛为止。一般都采用均方差作为标准测度函数. k 个聚类具有以下特点：各聚类本身尽可能的紧凑，而各聚类之间尽可能的分开。用伪代码的方式可以描述如下：

【K-MEANS 算法-输入】

聚类个数 k，以及包含 n 个数据对象的数据库。

【K-MEANS 算法-输出】

满足方差最小标准的 k 个聚类。

【K-MEANS 算法-处理流程】

1) 从 n 个数据对象任意选择 k 个对象作为初始聚类中心；
2) 循环（3）到（4）直到每个聚类不再发生变化为止
3) 根据每个聚类对象的均值（中心对象），计算每个对象与这些中心对象的距离；并根据最小距离重新对相应对象进行划分；
4) 重新计算每个（有变化）聚类的均值（中心对象）。

3.2 分类器的设计与实现

训练：

在图像的分类中，我们使用了两种分类器：svm 分类器和 knn 分类器，下面就这两种分类器进行描述。

3.2.1 支持向量机（SVM）

Svm 分类器是当前图像分类的重要工具，其分类的效果在各种分类器中占有优势。我们在这次图像分类的作业中，就尝试了使用这种分类器进行分类。

Svm 全称 Support Vector Machine，这种分类器就是将数据的特征向量向更高的维度投影，然后使用超平面进行分类。在分类之中，只有少部分的点会决定分类结果，这些点就被称为支撑向量。

我们组实际使用的svm是作业要求中给出的svm分类器Libsvm，下载地址是http://www.csie.ntu.edu.tw/~cjlin/libsvm/，这个分类器是由Chih-Chung Chang和 Chih-Jen Lin开发的Svm库。作者在对Libsvm的说明中，给出了说明文档 http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf，在这个文档中，Libsvm的发布者对svm的实际使用给出了一些建议，并且利用实验展示了建议所带来的好处，在阅读文档之后，对libsvm的使用会有一个更好的认识。

根据作者的建议，svm 的核函数最好使用 rbf，然后在一个范围内，搜索最好的 rbf 核函数的参数 C 和 Gamma，作者提供了一个工具，grid.py，这个工具在 python 环境下使用 rbf 核函数对训练集进行交叉验证，得到 C 和 Gamma 的最优结果。然后根据这个最优的结果去训练训练集的特征数据，将得到的 kernel 用来预测，得到 test 的结果。以上的步骤总结起来就是根据作者所给工具，进行操作。总结起来，在单一特征的 svm 中，我们进行如下的操作：

以 hsv 颜色直方图为例，我们需要用到作者提供的 grid.py，svm-train.exe 和 svm-predict.exe

1. 提取图像特征向量，存储格式是：label index:value index:value…index:value\n，存储的文件名为 hsv_train 和 hsv_test。
2. 得到特征之后，运行 python 程序 grid.py，调用的命令如下：F:\tools\grid.py -log2c -15,15,1 -log2g -15,15,1 -v 5 -svmtrain svm-train.exe -gnuplot gnu hsv_train。这里面-log2c 设定 C 的范围和步长，-log2g 设定 Gamma 的范围和步长。-svmtrain 设定 svm-train 的路径，-v 设定交叉检验的分组。-gnuplot 设定 gnu 的路径，这里用 gnuroot 代替,hsv_train 是训练集的路径。这里 svm-train 和 hsv_train 和 grid.py 在一个文件夹中。
3. 根据 grid.py 的结果得到交叉检验识别率最高的参数 C、Gamma，然后用 svm-train.exe 进行训练。命令行参数是 svm-trian –c 1 –v 0.00003 hsv_train hsv_model。这里 hsv_model 是生成的 model 的文件名。
4. 之后根据训练的 model 进行识别，使用程序 svm-predict.exe 进行识别。命令行参数是 svm-train hsv_model hsv_test hsv_out。这里 hsv_model 就是 3 中的 model，hsv_test 是测试集的特征数据，hsv_out 是输出结果。

3.2.2 K 最近邻（KNN）

Knn 分类器，全称 K nearest neighborhood，算法的思想利用训练样本，计算每个预测样本的最近的 K 的点，然后根据这 K 个点的归属进行归类。这个算法中比较重要的部分便是 K 和距离函数的选取。在我们的作业中，我们选取 K = 8，距离用欧式距离进行计算，利用 knn 的方法实现了分类，但是从效果上来看，knn 的效果没有 svm 的效果好。具体的实验结果如下：

特征	Knn 识别准确率
EDH	33.8%
HSV	43.7%
GM	29.2%
IM	16.2%
PCASIFT	20.4%

从实验结果中可以看到 knn 的效果一般要比 svm 差，虽然在一些特征上表现和 svm 相差无几（HSV），但是在总体特征上，其表现不如 svm 的表现。

3.2.3 特征融合

在得到单个特征的结果之后，我们尝试了进行特征之间的融合。现在融合的办法主要分两种，一种办法是在训练之前，将不同的特征进行合并，得到新的特征进行分类；另一种是根据分类的结果进行重新分类。我们采用的是后一种办法，结合 svm 训练的结果进行重新的分类。

在 Libsvm 中，其多分类器的实现是由二分类器组合而成的，比如一个 3 分类器，就两两生成一个二分类器，一共生成 3 个分类器，然后这些分类器进行投票，将图片分到得票最多的那一类上。就 3 分类器而言，一共生成 3 个二分类器，分别是 12，13 和 23 类的分类器。然后新的图片输入之后三个分类器进行识别，然后把结果加总，比如上面的分类结果分别是 1、1 和 2，那么最后分类的呃结果是 1，因为第一类得到的投票最多。再融合多个分类器的过程中，我们也采用了同样的策略，每种特征训练出一个 svm 的 model 进行分类，然后将所有的分类器的结果融合，得到投票的结果，将得票最多的类别作为判定的类别。

实验中，我们采用了四种特征里面表现最好的特征，这四种特征分别是：边缘直方图、hsv 颜色直方图、灰度共生矩阵和 pcasift。下页表中是实验结果。

特征	准确率
EDH	42.3%
HSV	43.8%
GM	47.1%
Pcasift	44.5
EDH+HSV+GM	64.3%
EDH+GM+PCASIFT	62.1%
EDH+HSV+SCASIFT	62.3%
EDH+HSV+GM	63.1%
EDH+HSV+GM+PCASIFT	67.7%

这些特征，彼此较为独立，因此可以猜想融合可以达到互补的效果，从而提高实验结果。从结果来看，我们可以看到特征融合的确起到了提升的效果，而且，这里的每个特征都会对总的特征有所贡献。当然实验中也使用了形状特征 hu 不变矩，但是由于其初始的识别效果不好，和其他特征融合之后结果反而变差。

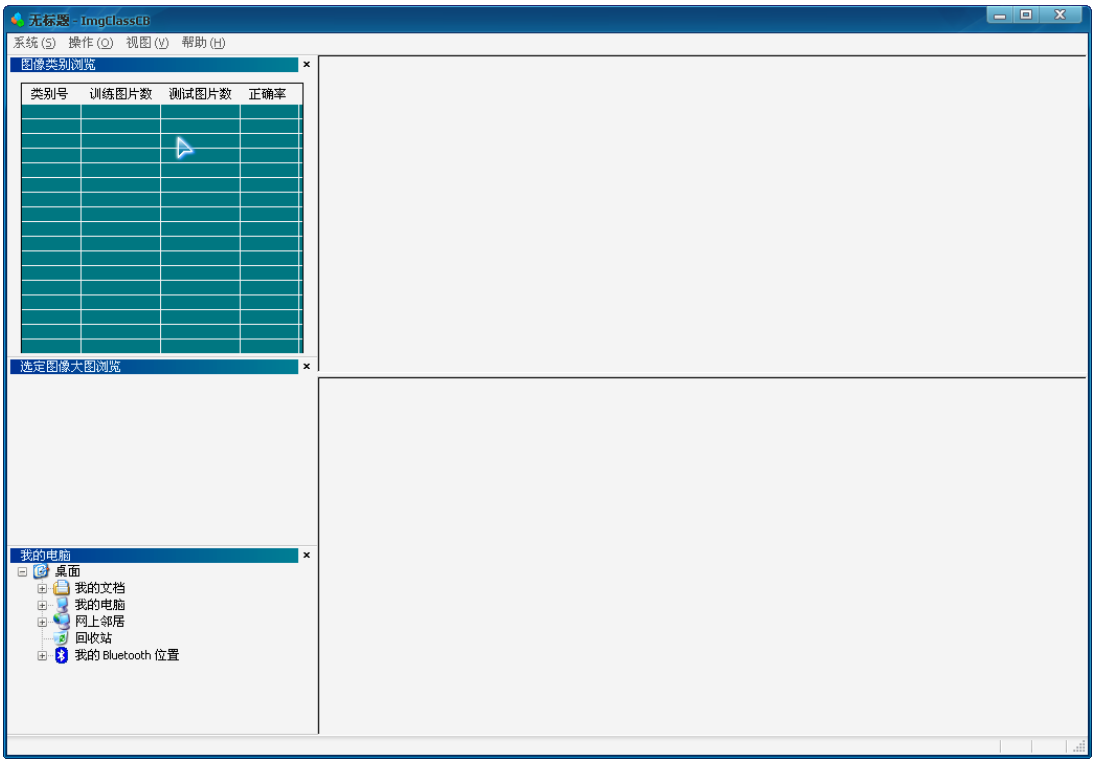
【界面设计及实现】

3.3 界面设计

在界面的设计上我们力求简洁实用又不失美观大方，从用户的角度来考虑界面的结构。作为一个图片分类系统，界面上应该有：图片列表（缩略图）浏览区域、图片大图显示区域、类别选择区域等与用户交互的部分；另外还需要图片路径输入接口、结果输出接口、特征提取设置及控制面板、分类器训练设置及控制面板、分类器测试设置及控制面板、结果输出设置及控制面板等多个部分。

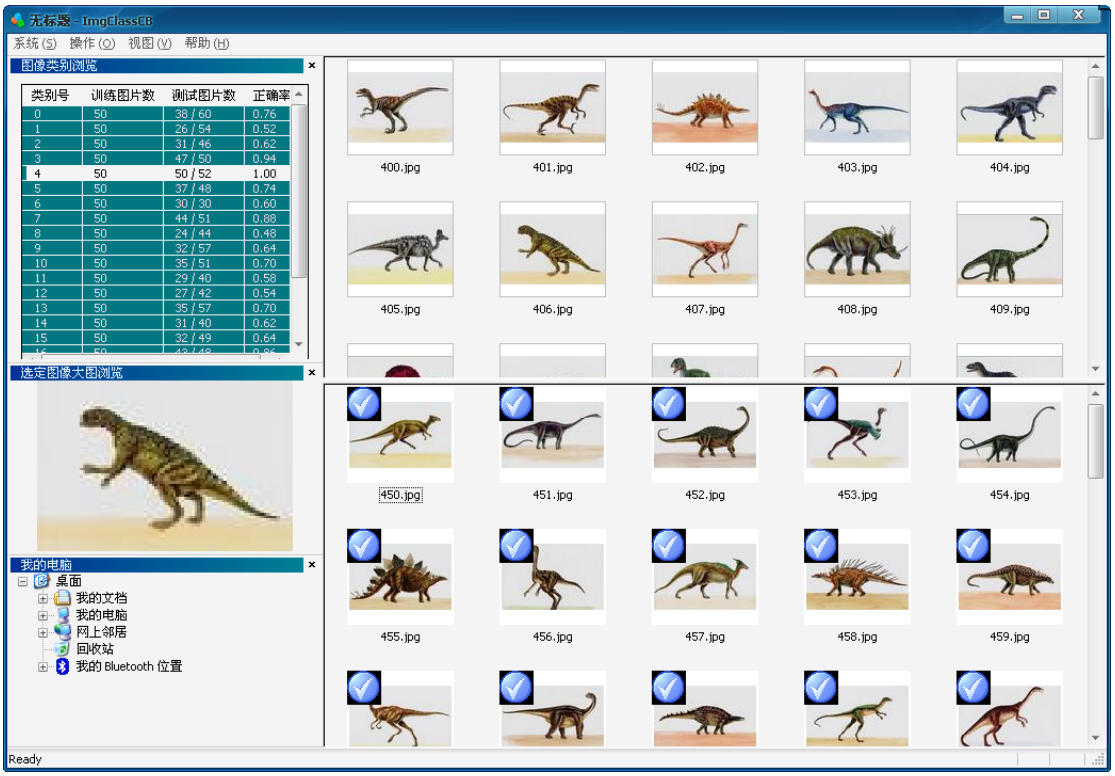
3.4 界面实现

按照设计，首先是如下图所示的主窗口：



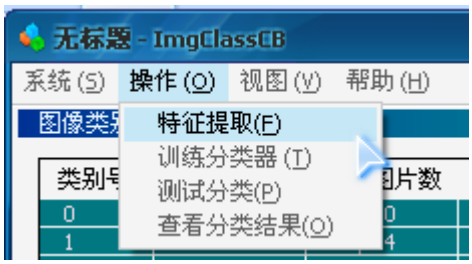
界面上大致可分为 2 个部分，左边是操作控制面板，右半部分是缩略图显示区域。而显示区域又分为上下两部分，分别是训练集和测试集图片的显示区域；控制面板中主要部分是分类结果列表，还有两个辅助部分为预览区域和“我的电脑查看区域”，在这两个部分用户可以选择性的查看大图以及电脑中的图片。

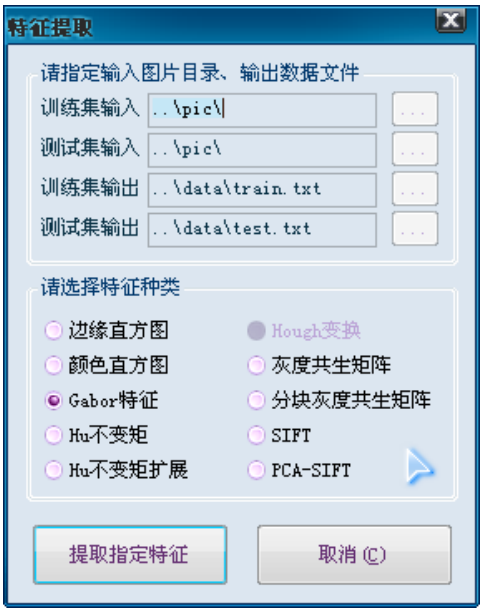
下图显示导入分类结果数据后的截图。从图中可以看到类别列表、预览图、测试集与训练集图片缩略图等都一目了然：

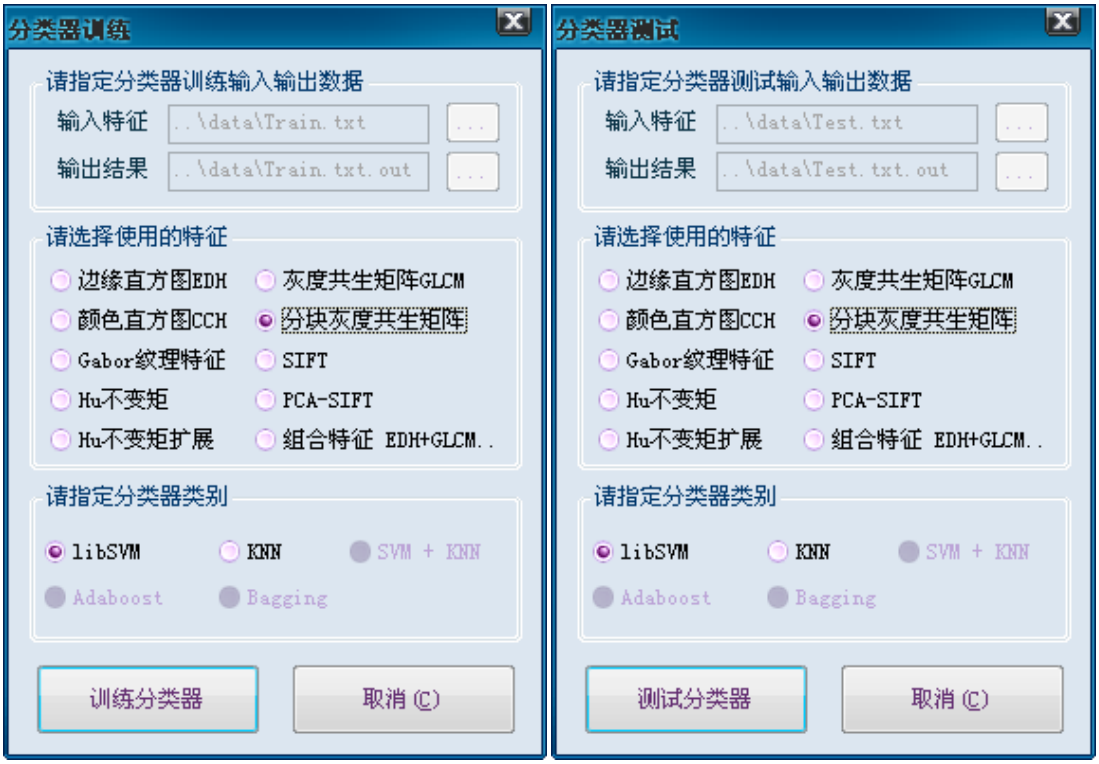


类列表的设计，包括类别号、训练图片数、测试图片数、正确率等结果展示，最后一行为平均正确率；在缩略图列表中，对测试集分类结果图片添加了标注功能：正确的结果左上角贴上一个对号的图片，错误的结果贴上错误标号。

接下来是系统操作菜单以及对应的控制面板（根据程序运行的流程步骤，在【操作】菜单中有 4 个子菜单项，分别是【特征提取】、【训练分类器】、【测试分类器】、【查看结果】，而每一个子菜单项都对应了一个操作控制面板，用户可以指定输入输出路径以及选择特征和分类器的种类。如下面图形所示：







在【查看分类结果】控制面板上，为用户提供了多种查看结果的方式，如直接查看文本结果，在主界面上查看分类结果图片缩略图表，查看 Confusion Matrix 文本、查看 Confusion Matrix 的立体柱状图等。下图显示的是 Confusion Matrix 的图片演示效果：

4. 系统整合及各模块之间的联系

由于本系统是一个较为复杂的系统，并且有些步骤如 sfit 特征提取、gabor 特征提取、svm 支持向量机训练过程等都比较耗费时间，我们采取了进程调用的策略，而不是函数调用，这样能保证各个子模块之间的并行执行，比如在提取特征的同时也可以在主界面上进行其他的操作。

这样做就需要把其他各模块（包括特征提取的各个子模块、各分类器模块）做成与之独立的程序，编译产生的 release 版本供主界面（界面模块）调用；这样一来就与第一节中所说的界面模块负责各模块的联系对应上了：界面不仅要提供用户操作的接口，而且需要将各模块的操作结果读入并通过各种方式显示给用户。具体各程序模块以及其之间的接口关系在【附录 2】中详述。

5. 【分类结果演示】

5.1 分类结果分析

如下两个表格分别表示 SVM 与 KNN 对各种特征的分类**平均准确率**：

特征	SVM 准确率
EDH	42.3%
HSV	43.8%
Gabor	29.8%
IM	20.6%
GM	47.6%
Pcasift	44.5%
EDH+HSV+GM	64.3%
EDH+GM+PCASIFT	62.1%
EDH+HSV+PCASIFT	62.3%
EDH+HSV+GM	63.1%
EDH+HSV+GM+PCASIFT	67.7%

特征	Knn 识别准确率
EDH	33.8%
HSV	43.7%
GM	29.2%
IM	16.2%
PCASIFT	20.4%

对分类程序结果的评价一个比较好的方法是 Precision-Recall 图的方法，但是由于我们的图片库中没有多余的“无关“图片”，PR 图其实并不能更精确的反映分类结果的情况。我们采用了多种结果方式查看的方法，从各种途径让结果一目了然。

5.2 多种结果方式查看

5.2.1 分类结果文本：

分类结果文本中每一个测试集图片对应一行，指出图片真实类别与分类结果：

```
50      1    1
51      1    1
52      1    2
53      1    1        ……
```

5.2.2 主界面分类类别列表与缩略图列表

类别列表中的列包括：类别号、训练图片数、测试图片数、正确率等结果展示，而且在最后一行为平均正确率，使结果最直接方便地显示出来。

缩略图片列表与类别列表关联使用，使用方法就是点击图片类别列表中的某一行，然后这一行所对应的类别的所有训练集图片与测试集图片则显示在缩略图列表区域。

为了更突出显示结果，我把测试图片的缩略图显示增加了一个新的功能，就是根据图片分类结果为这个图形左上标记。例如下面右图显示的是对恐龙类别的分类结果，其中有两个滑雪场景的人物照片被错误的分到了这一类：

5.2.3 Confusion Matrix 方式查看

对于 Confusion Matrix 提供了两种方式查看，即图片和文本的方式。文本的 Confusion Matrix 可以由系统自动生成，但图片需要手动在 Excel 中生成（这一步本想使用 GNUplot 等在程序运行中自动生成，但做了很久的实验没有成功，只好放弃）。如下表示 Confusion Matrix 的文本以及图片显示的结果样例（67.7%的平均准确率）：

0.76 0.00 0.04 0.00 0.00 0.06 0.00 0.00 0.02 0.02 0.02 0.02 0.00 0.00 0.00 0.00 0.00 0.00 0.06 0.00
0.02 0.52 0.08 0.00 0.00 0.00 0.00 0.00 0.02 0.04 0.00 0.02 0.00 0.02 0.02 0.02 0.00 0.00 0.04 0.20
0.08 0.04 0.62 0.00 0.00 0.00 0.00 0.00 0.04 0.00 0.02 0.02 0.02 0.00 0.00 0.04 0.00 0.00 0.08 0.04
0.02 0.00 0.00 0.94 0.00 0.00 0.00 0.00 0.00 0.02 0.00 0.00 0.00 0.00 0.00 0.02 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.04 0.00 0.00 0.00 0.00 0.74 0.00 0.00 0.04 0.00 0.00 0.04 0.00 0.00 0.00 0.00 0.00 0.02 0.06 0.06
0.02 0.00 0.00 0.00 0.00 0.00 0.60 0.04 0.00 0.10 0.04 0.04 0.02 0.04 0.00 0.02 0.00 0.00 0.00 0.08
0.00 0.00 0.00 0.00 0.00 0.10 0.00 0.88 0.02 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.02 0.08 0.04 0.02 0.00 0.00 0.00 0.00 0.48 0.00 0.04 0.00 0.00 0.04 0.02 0.10 0.00 0.02 0.02 0.12
0.12 0.02 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.64 0.00 0.02 0.10 0.00 0.02 0.00 0.00 0.02 0.06 0.00
0.08 0.00 0.00 0.00 0.00 0.02 0.00 0.06 0.00 0.02 0.70 0.02 0.02 0.00 0.00 0.02 0.00 0.00 0.04 0.02
0.00 0.00 0.08 0.00 0.00 0.00 0.00 0.02 0.00 0.10 0.06 0.58 0.00 0.00 0.02 0.00 0.00 0.00 0.02 0.12
0.00 0.02 0.00 0.00 0.00 0.04 0.00 0.00 0.00 0.04 0.06 0.00 0.54 0.06 0.00 0.04 0.00 0.02 0.06 0.12
0.00 0.10 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.02 0.00 0.02 0.70 0.02 0.04 0.00 0.00 0.02 0.08
0.00 0.04 0.02 0.02 0.00 0.00 0.00 0.00 0.02 0.04 0.02 0.00 0.04 0.18 0.62 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.02 0.00 0.00 0.00 0.00 0.02 0.08 0.02 0.04 0.02 0.00 0.00 0.02 0.64 0.02 0.02 0.02 0.08
0.00 0.02 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.86 0.00 0.02 0.10
0.00 0.06 0.02 0.02 0.00 0.00 0.00 0.00 0.04 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.04 0.76 0.04 0.02
0.02 0.04 0.00 0.00 0.04 0.00 0.00 0.00 0.06 0.02 0.00 0.00 0.00 0.00 0.00 0.00 0.02 0.02 0.64 0.14
0.02 0.14 0.00 0.00 0.00 0.00 0.00 0.00 0.06 0.08 0.00 0.02 0.08 0.10 0.06 0.04 0.02 0.02 0.04 0.32

6. 【项目总结】

本项目从开始分析讨论到最终程序的整合打包，确实工作量比较大，而且存在一些分工不均衡等问题，不过总算按时完成了整个项目的各项工作。

我们作为总结，我想说一下我们的亮点：

- ✧ 1. 优秀的用户界面，提供用户方便的操作、美观的视觉感受以及多种途径的结果查看方式；
- ✧ 2. 程序结构清晰明了，各部分模块之间的耦合度很小，有利于程序的调试等工作；最终打包的程序全部为优化后的 Release 版本，同时做到了缩小程序体积以及在其他机器上的正常运行；
- ✧ 3. 最终分类结果达到 67.7%，对于小规模的训练集来说这个结果已经能够接受了^_^；
- ✧ 4. 出色的文档，项目文档、程序说明文档等都用了很好的图文排版方式。

但由于时间和精力有限，有一些问题并没有很好的解决，比如：

- ✧ Hu 不变矩形状特征的结果并不是很好；
- ✧ SIFT 特征提取的结果又重复特征，导致 K-means 聚类过程出现问题，SIFT 及 PCA-SIFT 特征提取的结果并不乐观；
- ✧ Adaboost 等分类器以及分类器的融合等工作没有实验出较好的结果

等等。
但是我们这段时间在项目上已尽心尽力，努力做到能做到的最好的。

刘洋 2009-5-14

【附录1】. 项目环境配置

a) 【总体环境配置】

- **运行环境：** Windows Xp Sp1~ Sp3
- **编程环境：** Visual Studio 6.0 + Sp6
- **如无特别说明，每个工程直接打开*.dsw 编译运行即可。**

b) 【SVM 分类器配置及用法】

- libsvm 中的工具 grid.py 和训练与预测程序 svm-train.exe 和 svm-predict.exe
- 在作业中，先调用 grid.py，利用交叉检验得到 rbf kernel 里效果最好 c 和 gamma 的参数，然后利用这个参数在 svm-train 里训练，然后使用 svm-predict 进行预测得到输出结果。
- 下面以 edh 特征为例，给出操作：
  - 在系统中安装 python 环境；
  - 运行 grid.py，命令行：F:\tools\grid.py -log2c -15,15,1 -log2g -15,15,1 -v 5 -svmtrain svm-train.exe -gnuplot F:\tools\gnuplot\bin\pgnuplot.exe edh_train
- 这是调用 grid 的命令，-log2c 设置 c 的取值范围 2 的-15 到 15 此方，步长是 1，-log2g 设置 g 的取值，v 设置了交叉检验的分组，gnuplot 设置 gunplot 的路径，可以展示参数好坏，最后输入训练集。
  - 运行 svm-train，命令行：svm-train -c 2 -g 0.0000305 edh_train edh_model
- c 和 g 的参数是有第二步得到的最佳参数，edh_train 是训练集，edh_model 是输出的 model
  - 运行 svm-predict,命令行：svm-predic edh_test edh_model edh_out
- edh_test 是测试集的特征，model 是 3 的，edh_out 是分类结果。

每一个程序中命令行的格式以及如何使用，在各自文件夹里的 readme 中。

c) 【SIFT 特征提取环境配置】

- **编程环境：** 需要安装 OpenCV

【附录2】. 项目文件层级结构

项目文件夹里共有 5 个文件夹：

```
/doc/
        ICCB_程序说明文档.pdf
        ICCB_项目报告.pdf

/bin/
        ImgClassCB.exe          //主界面的可执行程序，运行此程序
        FE_CCH.exe              //以下几个为特征提取程序，
        FE_EDH.exe              //通过主程序 ImgClassCB.exe 调用
        FE….
        Kmeans.exe              //Sift 特征聚类程序（内部自动调用）

        svm-train.exe           //SVM 特征训练程序（自动调用）
        svm-predict.exe         //svm 测试程序（自动调用）
        knn.exe                 //KNN 分类程序（自动调用）
        conbine.exe             //特征组合分类程序（自动调用）

/code/
        /SIFT/                  //根据文件夹名可知为哪类程序的源码
        /…..

/data/                          //data 文件夹中存放各个阶段的结果
        *_train.txt             //训练集特征结果，*表示特征类型
        *_test.txt              //测试集特征结果，*表示特征类型
        *_train.mdl             //训练集的 SVM 结果

        resultraw.rrw           //分类初始结果数据
        result.txt              //结果数据[ 50    1    1]格式的文本
        ConMat.txt              //Confusion Matrix 文本
        ConMat.jpg              //Confusion Matrix 图片
        ……

/pic/                           //注：请将图片数据解压到这个文件夹
```