**电工导实验报告9**

**一、实验目的**

1. 基于LSH索引的快速图像检索

**二、实验内容**

1. **了解为什么使用LSH**
2. **通过Hash函数提高效率**
3. **LSH检索**

**三、实验环境**

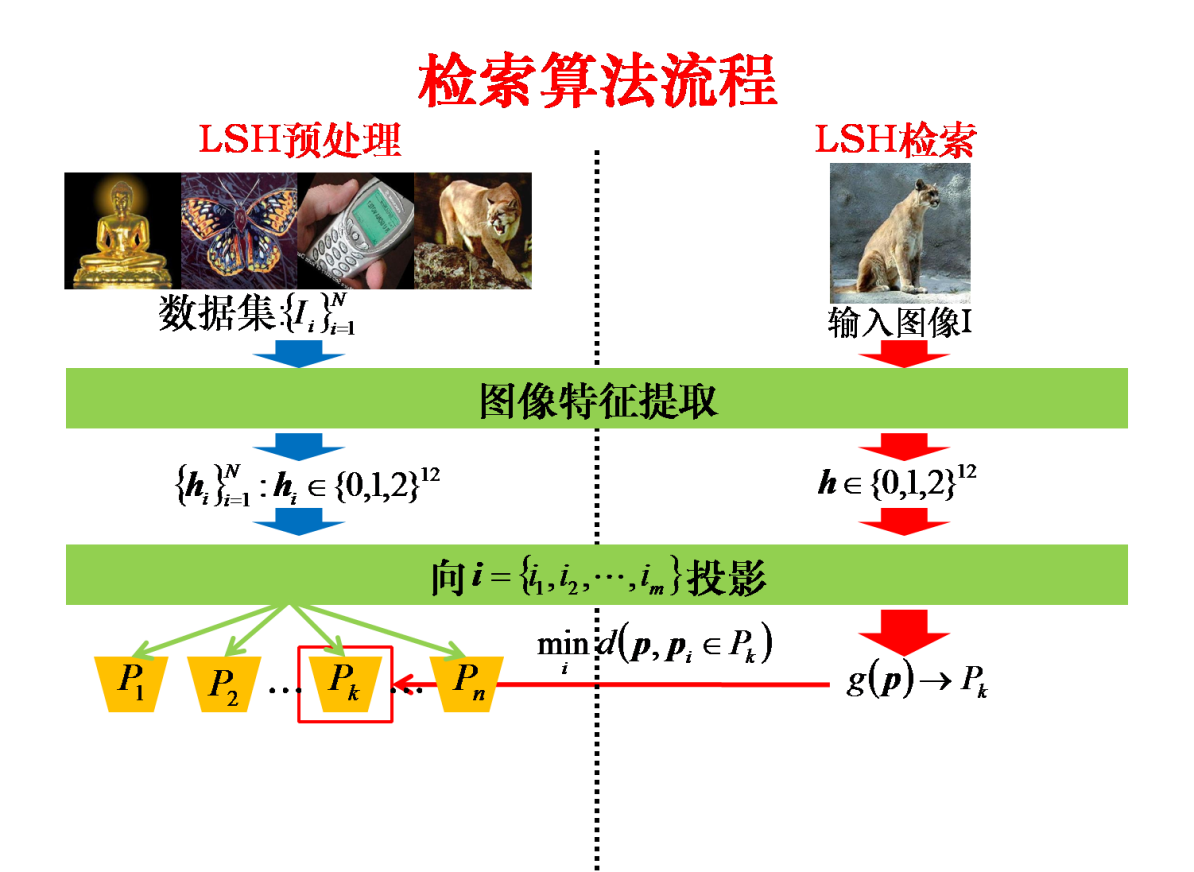
1. **Python 2.7 + opencv +numpy**

**四、实验原理**

用Nearest neighbor (NN) 或k-nearest neighbor (KNN)在数据库中检索和输入数据距离最近的1个或k个数据，一般情况下算法复杂度为O(n)（例如暴力搜索），优化情况下可达到O(log n)（例如二叉树搜索），其中n为数据库中的数据量。当数据库很大（即N 很大时），搜索速度很慢。

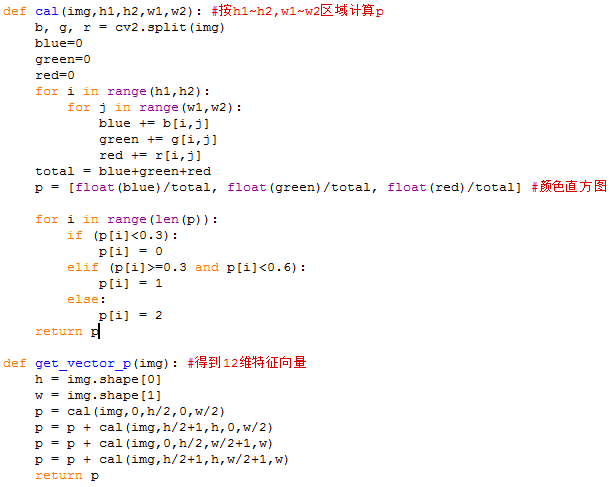
Hashing的基本思想是按照某种规则（Hash函数）把数据库中的数据分类，对于输入数据，先按照该规则找到相对应的类别，然后在其中进行搜索。由于某类别中的数据量相比全体数据少得多，因此搜索速度大大加快。

一幅图像可以分成四个部分，计算每个部分的RGB的比例，映射到0，1，2上，因此一幅图会得到一个由元素｛0，1，2｝组成的一个特征向量。通过哈希函数，将12维的特征向量投影到一个集合I上，相当于每一幅图都会得到一个投影的字符串，而一个字符串可能对应多幅颜色分布相近的图片。在很多图片中寻找最相似的图片后，通过哈希函数将需要比对的范围缩小到更小的范围，以提高匹配的效率。

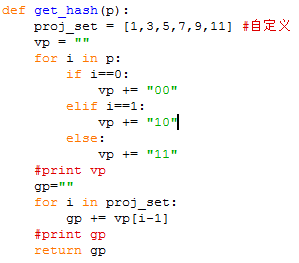


**五、算法实现及流程简介**

对于一幅图，首先要分四个区域计算其颜色直方图，颜色直方图的计算之前已经非常熟练。然后将RGB三原色各自的比例投影到0，1，2，以此来量化特征向量。将得到的四组三维向量组合到一起，得到一个由元素｛0，1，2｝构成的12维特征向量。



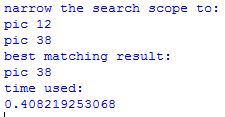
之后是构造哈希函数，通过将12维的向量投影到I上，得到一个len(I)的字符串。

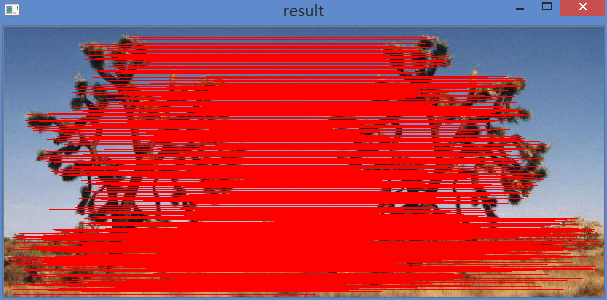


在main函数中，建立一个dictionary,将所有得到的映射后的字符串作为key,以此来对图片进行分类。当要寻找一幅图的时候，只要计算其经过hash后的字符串，到dictionary中寻找对应的一个小分类即可。如果出现找不到的情况，那就与所有的key进行比较，寻找相似度最高的key，并与该key对应的图片进行比较，比较过程就通过上一次的sift。由于hash函数的分类作用，能大大缩短匹配的时间，相当于把一些不太可能的图片直接滤去了，大幅度提升了效率和程序运行的速度。

**六、运行效果**

运行效果如下：

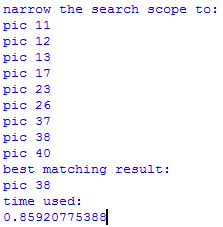




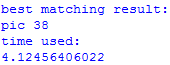
可见，通过hash分类以后，匹配的时候只需要和两幅相近的图片进行进一步的sift匹配，所需的时间也非常少。（注：时间计算的时候没有将建立hash的时间计算在内，计算的只是匹配的时间）。

上述结果我设定 proj\_set = [1,3,5,7,9,11] 时的，集合的元素越多，相当于分类的类别越细，搜索的时候时间也就越短（可能建立hash时会多花一点点时间）；反之，如果集合元素的个数很少，如果只有一个元素，那只能分成两类，虽然比不分类快，但相比别的集合来说速度就慢了很多。

例如三个元素的时候，分类不是很细，需要匹配的图片也就相应增多。



如果不通过hash来减少匹配的次数，每一幅图都匹配一次，那是时间会大大增加。对比运行结果如下：



结果是相同的，但是时间相差了7倍，可见hash函数的强大作用。

**七、实验总结**

LSH，局部敏感哈希算法通过给图片分类，使得在很多图像中匹配一幅图片的时间大幅缩短，大大节省了时间，增加了效率。dataset中只有40幅图效率就相差很多，在成百上千幅图中匹配的话运用hash的优势将会更加被放大。之前网络爬虫的时候也用过hash函数，进而判断该url是否已经爬取过。这一次接触hash,不仅更加熟悉和了解，也感叹算法对程序的优化效果如此惊人。

F1403023 5140309534 韩坤言

附源码：

*# -\*- coding: utf-8 -\*-*

*import cv2*

*import numpy as np*

*import time*

*def cal(img,h1,h2,w1,w2): #按h1~h2,w1~w2区域计算p*

*b, g, r = cv2.split(img)*

*blue=0*

*green=0*

*red=0*

*for i in range(h1,h2):*

*for j in range(w1,w2):*

*blue += b[i,j]*

*green += g[i,j]*

*red += r[i,j]*

*total = blue+green+red*

*p = [float(blue)/total, float(green)/total, float(red)/total] #颜色直方图*

*for i in range(len(p)):*

*if (p[i]<0.3):*

*p[i] = 0*

*elif (p[i]>=0.3 and p[i]<0.6):*

*p[i] = 1*

*else:*

*p[i] = 2*

*return p*

*def get\_vector\_p(img): #得到12维特征向量*

*h = img.shape[0]*

*w = img.shape[1]*

*p = cal(img,0,h/2,0,w/2)*

*p = p + cal(img,h/2+1,h,0,w/2)*

*p = p + cal(img,0,h/2,w/2+1,w)*

*p = p + cal(img,h/2+1,h,w/2+1,w)*

*return p*

*def get\_hash(p):*

*proj\_set = [1,3,5,7,9,11] #自定义*

*vp = ""*

*for i in p:*

*if i==0:*

*vp += "00"*

*elif i==1:*

*vp += "10"*

*else:*

*vp += "11"*

*#print vp*

*gp=""*

*for i in proj\_set:*

*gp += vp[i-1]*

*#print gp*

*return gp*

*def sift\_match(target,imgs): #sift以及匹配*

*sift = cv2.SIFT()*

*kp\_targ, des\_targ = sift.detectAndCompute(target,None)*

*FLANN\_INDEX\_KDTREE = 0*

*index\_params = dict(algorithm = FLANN\_INDEX\_KDTREE, trees = 5)*

*search\_params = dict(checks=50)*

*flann = cv2.FlannBasedMatcher(index\_params,search\_params)*

*max\_num = 0*

*for img in imgs:*

*kp, des = sift.detectAndCompute(img[1],None)*

*matched = flann.knnMatch(des,des\_targ,k=2) #和target比较*

*#print matched*

*accept = []*

*for m,n in matched:*

*if m.distance < 0.08\*n.distance: #限定来找相似点 0.08*

*accept.append(m)*

*print len(accept)*

*if len(accept)>max\_num: #img匹配的数量更多 刷新*

*max\_num = len(accept)*

*best\_img = img*

*best\_kp = kp*

*best = accept*

*print "best matching result:"*

*print "pic", best\_img[0]*

*#将两幅图放在一幅图中*

*h1, w1 = target.shape[:2]*

*h2, w2 = best\_img[1].shape[:2]*

*result = np.zeros((max(h1, h2), w1 + w2, 3), np.uint8)*

*result[:h1, :w1] = target[:,:]*

*result[:h2, w1:] = best\_img[1][:,:]*

*#将匹配到的关键点连起来*

*for m in best:*

*color = (0,0,255)*

*cv2.line(result, \*

*(int(best\_kp[m.queryIdx].pt[0]+w1),int(best\_kp[m.queryIdx].pt[1])) , \*

*(int(kp\_targ[m.trainIdx].pt[0]), int(kp\_targ[m.trainIdx].pt[1])), color)*

*return result*

*def main():*

*imgs = []*

*for i in range(1,41):*

*img = cv2.imread(str(i)+".jpg")*

*imgs.append([i,img]) #把图片的名字一并记下*

*hash\_dic = {} #hash函数值作为key 用来分类*

*for i in range(len(imgs)):*

*p = get\_vector\_p(imgs[i][1])*

*gp = get\_hash(p)*

*if hash\_dic.has\_key(gp):*

*hash\_dic[gp].append(imgs[i])*

*else:*

*hash\_dic[gp] = [imgs[i]]*

*target = cv2.imread("target.jpg")*

*p = get\_vector\_p(target)*

*gp = get\_hash(p)*

*t1 = time.clock()*

*#通过哈希缩小需要匹配的范围*

*if hash\_dic.has\_key(gp):*

*pos\_imgs = hash\_dic[gp]*

*else: #如果没有找到 寻找hash值最接近的*

*pos\_imgs =[]*

*keys = hash\_dic.keys()*

*match = []*

*for item in keys:*

*cnt = 0*

*for i in range(len(gp)):*

*if (gp[i]==item[i]):*

*cnt = cnt + 1;*

*match.append(cnt)*

*maxi = max(match)*

*for i in range(len(match)):*

*if (match[i]==maxi):*

*pos\_imgs = pos\_imgs + hash\_dic[keys[i]]*

*print "narrow the search scope to:"*

*for i in pos\_imgs:*

*print "pic", i[0]*

*result = sift\_match(target,pos\_imgs)*

*t2 = time.clock()*

*print "time used:"*

*print t2-t1*

*cv2.imshow('result',result)*

*cv2.waitKey()*

*cv2.destroyAllWindows()*

*if \_\_name\_\_ == '\_\_main\_\_':*

*main()*