
Practical Content-based Image Retrieval Handbook

基于内容的图像检索 实用手册



一本以实践经验为基础的图像检索指南书

整理: 袁勇
整理时间: October 9, 2017
Email: yongyuanstu@gmail.com

Version: 2.00

目 录



1 简介	1
1.1 基于内容的图像检索技术	2
1.1.1 相同物体图像检索	2
1.1.2 相同类别图像检索	3
1.1.3 大规模图像检索特点	4
1.2 计算机视觉通用工具包	6
1.2.1 OpenCV	6
1.3 图像特征表达工具包	6
1.3.1 VLFeat	6
1.3.2 Yael	6
1.3.3 Caffe	7
1.4 索引构建工具包	7
1.4.1 Falconn	7
1.4.2 Nmslib	7
1.4.3 Faiss	7
2 工业常用算法	8
2.1 PCA	8
2.2 KMeans	9
3 图像特征表达	11
3.1 局部特征	11
3.1.1 SIFT 特征	11
3.1.2 HOG 特征	11
3.2 全局特征	12
3.2.1 CNN 特征	12
4 载入、显示及保存	13
5 图像基础	16

6 特征编码	17
6.1 BoF 词袋模型	17
6.2 VLAD 局部聚合向量	17
6.2.1 VLAD 归一化	20
6.3 Fisher Vector	20
6.3.1 Fisher 核	20
6.3.2 图像的 Fisher 向量	21
6.3.3 Fisher Vector 实验	22
7 ANN 搜索	28
7.1 矩阵相乘	28
7.2 基于树的方法	28
7.3 哈希方法	29
7.4 矢量量化方法	29
7.4.1 PQ 乘积量化	30
7.4.2 倒排乘积量化	31
7.5 拓展查询	32
参考文献	33





第1章 简介



在 Web2.0 时代，尤其是随着 Flickr、Facebook 等社交网站的流行，图像、视频、音频、文本等异构数据每天都在以惊人的速度增长。例如，Facebook 注册用户超过 10 亿，每月上传超过 10 亿的图片；Flickr 图片社交网站 2015 年用户上传图片数目达 7.28 亿，平均每天用户上传约 200 万的图片；中国最大的电子商务系统淘宝网的后端系统上保存着 286 亿多张图片。针对这些包含丰富视觉信息的海量图片，如何在这些浩瀚的图像库中方便、快速、准确地查询并检索到用户所需的或感兴趣的图像，成为多媒体信息检索领域研究的热点。基于内容的图像检索方法充分发挥了计算机长于处理重复任务的优势，将人们从需要耗费大量人力、物力和财力的人工标注中解放出来。经过十来来的发展，基于内容的图像检索技术已广泛应用于搜索引擎、电子商务、医学、纺织业、皮革业等生活的方方面面。

图像检索按描述图像内容方式的不同可以分为两类，一类是基于文本的图像检索 (TBIR, Text Based Image Retrieval)，另一类是基于内容的图像检索 (CBIR, Content Based Image Retrieval)。

基于文本的图像检索方法始于上世纪 70 年代，它利用文本标注的方式对图像中的内容进行描述，从而为每幅图像形成描述这幅图像内容的关键词，比如图像中的物体、场景等，这种方式可以是人工标注方式，也可以通过图像识别技术进行半自动标注。在进行检索时，用户可以根据自己的兴趣提供查询关键字，检索系统根据用户提供的查询关键字找出那些标注有该查询关键字对应的图片，最后将查询的结果返回给用户。这种基于文本描述的图像检索方式由于易于实现，且在标注时有人工介入，所以其查准率也相对较高。在今天的一些中小规模图像搜索 Web 应用上仍有使用，但是这种基于文本描述的方式所带来的缺陷也是非常明显的：首先这种基于文本描述的方式需要人工介入标注过程，使得它只适用于小规模的图像数据，在大规模图像数据上要完成这一过程需要耗费大量的人力与财力，而且随时不断外来的图像在入库时离不开人工的干预；其次，“一图胜千言”，对于需要精确的查询，用户有时很难用简短的关键字来描述出自己真正想要获取的图像；再次，人工标注过程不可避免的会受到标注者的认知水平、言语使用以及主观判断等的影响，因此会造成文字描述图片的差异。

随着图像数据快速增长，针对基于文本的图像检索方法日益凸现的问题，在 1992 年美国国家科学基金会就图像数据库管理系统新发展方向达成一致共识，即表示索引图像信息的最有效方式应该是基于图像内容自身的。自此，基于内容的图像检索技术便逐步建立起来，并在近十多年里得到了迅速的发展。典型的基于内容的图像检索基本框架如上图 1.1 所示，它利用计算机对图像进行分析，建立图像特征矢量描述并存入图像特征库，当用户输入一张查询图像时，用相同的特征提取方法提取查询图像的

特征得到查询向量，然后在某种相似性度量准则下计算查询向量到特征库中各个特征的相似性大小，最后按相似性大小进行排序并顺序输出对应的图片。基于内容的图像检索技术将图像内容的表达和相似性度量交给计算机进行自动的处理，克服了采用文本进行图像检索所面临的缺陷，并且充分发挥了计算机长于计算的优势，大大提高了检索的效率，从而为海量图像库的检索开启了新的大门。不过，其缺点也是存在的，主要表现为特征描述与高层语义之间存在着难以填补的语义鸿沟，并且这种语义鸿沟是不可消除的。

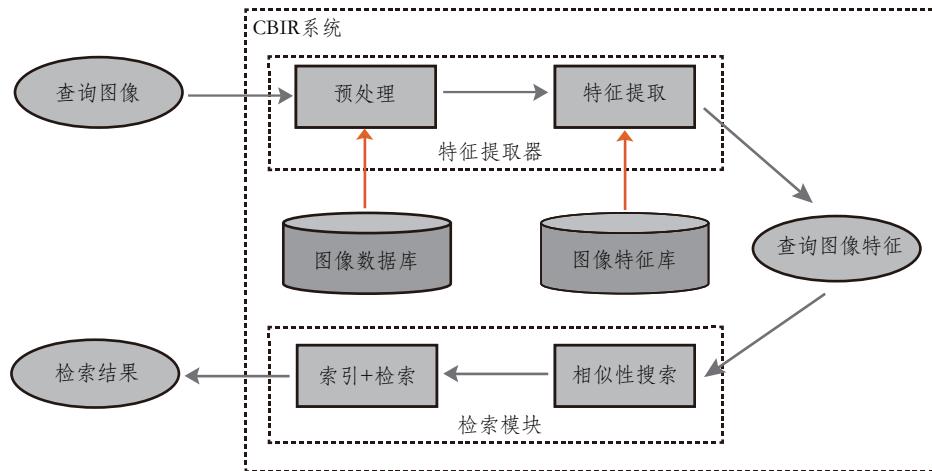


图 1.1: PQ 量化索引示意过程

基于内容的图像检索技术在电子商务、皮革布料、版权保护、医疗诊断、公共安全、街景地图等工业领域具有广阔的应用前景。在电子商务方面，谷歌的 Goggles、阿里巴巴的拍立淘等闪拍购物应用允许用户抓拍上传至服务器端，在服务器端运行图片检索应用从而为用户找到相同或相似的衣服并提供购买店铺的链接；在皮革纺织工业中，皮革布料生产商可以将样板拍成图片，当衣服制造商需要某种纹理的皮革布料时，可以检索库中是否存在相同或相似的皮革布料，使得皮革布料样本的管理更加便捷；在版权保护方面，提供版权保护的服务商可以应用图像检索技术进行商标是否已经注册了的认证管理；在医疗诊断方面，医生通过检索医学影像库找到多个病人的相似部位，从而可以协助医生做病情的诊断……基于内容的图像检索技术已经深入到了许许多多的领域，为人们的生活生产提供了极大的便利。

1.1 基于内容的图像检索技术

1.1.1 相同物体图像检索

相同物体图像检索是指对查询图像中的某一物体，从图像库中找出包含有该物体的图像。这里用户感兴趣的是图像中包含的特定物体或目标，并且检索到的图片应该是包含有该物体的那些图片。如 1.3 图所示，给定一幅“蒙娜丽莎”的画像，相同物体检



索的目标就是要从图像库中检索出那些包含有”蒙娜丽莎”人物的图片，在经过相似性度量排序后这些包含有”蒙娜丽莎”人物的图片尽可能的排在检索结果的前面。相似物体检索在英文文献中一般称为物体检索 (Object Retrieval)，近似样本搜索或检测 (Duplicate Search or Detection) 也可以归类于相同物体的检索，并且相同物体检索方法可以直接应用到近似样本搜索或检测上。相同物体检索不论是在研究还是在商业图像搜索产业中都具有重大的价值，比如购物应用中搜索衣服鞋子、人脸检索等。



图 1.2: PQ 量化索引示意过程

对于相同物体图像检索，在检索相同的物体或目标时，易受拍摄环境的影响，比如光照变化、尺度变化、视角变化、遮挡以及背景的杂乱等都会对检索结果造成较大的影响，图 1.3 左图给出了这几种变化的例子，此外，对于非刚性的物体，在进行检索时，物体的形变也会对检索结果造成很大的影响。

由于受环境干扰比较大，因而对于相同物体图像检索，在选取特征的时候，往往会选择那些抗干扰性比较好的不变性局部特征，比如 SIFT1、SURF2、ORB3 等，并以此为基础通过不同的编码方式构建图像的全局描述，具有代表性的工作有词袋模型 4(BoW, Bag of Words)、局部特征聚合描述符 5(VLAD, Vector of Locally Aggregated Descriptors) 以及 Fisher 向量 6(FV, Fisher Vector)，这一类以类 SIFT 为基础的图像检索方法，由于结合了类 SIFT 不变性的特性，并且采用了由局部到全局的特征表达方式，并且在实际应用时在提取 SIFT 的时候还可以使用 siftGPU 加速 SIFT 提取，因而从整体上来说能够获得比较好的检索效果，但这一类方法通常其特征维度往往是非常高的，如图 1.2 所示，在牛津建筑物图像数据库上采用词袋模型进行检索，为了获得较高的检索精度，在聚类时聚类数目一般都设置到了几十万，因而其最终表示的特征其维度高达几十万维，因此为它们设计高效的索引方式显得十分必要。

1.1.2 相同类别图像检索

对给定的查询图片，相似图像检索的目标是从图像库中查找出那些与给定查询图像属于同一类别的图像。这里用户感兴趣的是物体、场景的类别，即用户想要获取的是那些具有相同类别属性的物体或场景的图片。为了更好的区分相同物体检索和相同类别检索这两种检索方式区，仍以图 1.3 左图所举的”蒙娜丽莎”为例，用户如果感兴趣



的就是”蒙娜丽莎”这幅画，那么检索系统此时工作的方式应该是以相同物体检索的方式进行检索，但如果用户感兴趣的并不是”蒙娜丽莎”这幅画本身，而是”画像”这一类图片，也就是说，用户所感兴趣的已经是对这幅具体的画进行了类别概念的抽象，那么此时检索系统应该以相同类别检索的方式进行检索。相同类别图像检索目前已广泛应用于图像搜索引擎，医学影像检索等领域。

对于相同类别图像检索，面临的主要问题是属于同一类别的图像类内变化巨大，而不同类的图像类间差异小。如图 1.3 右图所示，对于”湖泊”这一类图像，属于该类别的图像在表现形式上存在很大的差异，而对于图????? 右图下面所示的”dog”类和”woman”类两张图像，虽然它们属于不同的类，但如果采用低层的特征去描述，比如颜色、纹理以及形状等特征，其类间差异非常小，直接采用这些特征是很难将这两者分开的，因此相同类别图像检索在特征描述上存在着较大的类内变化和较小的类间差异等挑战。近年来，以深度学习 (DL, Deep Learning) 为主流的自动特征在应用到相同类别图像检索上时，能够极大的提高检索的精度，使得面向相同物体的检索在特征表达方面得到了较好的解决。目前，以卷积神经网络 (CNN, Convolutional Neural Network) 为主导的特征表达方式也开始在相同物体图像检索上进行展开，并已有了一些相应的工作⁷，但由于相同物体在构造类样本训练数据时并不像相同类别图像检索那样那么方便，因而相同物体图像检索在 CNN 模型训练以及抽取自动特征等方面还有待深入。不管是相同物体图像检索还是相同类别图像检索，在使用 CNN 模型提取自动特征的时候，最终得到的维度一般是 4096 维的特征，其维度还是比较高的，直接使用 PCA 等降维的手段，虽然能达到特征维度约减的目的，但在保持必要的检索精度前提下，能够降低的维度还是有限的，因而对于这一类图像检索，同样有必要为它构建够高效合理的快速检索机制，使其适应大规模或海量图像的检索。

1.1.3 大规模图像检索特点

无论是对于相同物体图像检索还是相同类别图像检索，在大规模图像数据集上，它们具有三个典型的主要特征：图像数据量大、特征维度高以及要求相应时间短。下面对这三个主要特征逐一展开说明：

(1) 图像数据量大。得益于多媒体信息捕获、传输、存储的发展以及计算机运算速度的提升，基于内容的图像检索技术经过十几年的发展，其需要适用的图像规模范围也从原来的小型图像库扩大到大规模图像库甚至是海量图像数据集，比如在上世纪九十年代图像检索技术发展的早期阶段，研究者们在验证图像检索算法性能的时候，用得比较多是 corel1k，该图像库共 1000 张图片，与今天同样可以用于图像检索的最流行的图像分类库 imageNet 数据集相比，其量级已经有了成千上万倍的增长，因而图像检索应满足大数据时代的要求，在大规模图像数据集上应该具备伸缩性。

(2) 特征维度高。图像特征作为直接描述图像视觉内容的基石，其特征表达的好坏直接决定了在检索过程中可能达到的最高检索精度。如果前置特征未表达好，在构建后置检索模型的时候，不但会复杂化模型的构建，增加检索查询的响应时间，而且



能够提升的检索精度也是极其有限的。所以在特征提取之初，应该有意识的选取那些比较高层特征。如果将局部特征表达方式也作为“高维”的一种，那么特征的描述能力跟特征的维度高低具有较大的关联，因而在特征描述方面大规模图像检索具有明显的特征维度高的特性，比如词袋模型 BoW、VLAD、Fisher 向量以及 CNN 特征。为了对这些高维的特征有一个维度量级的定量认识，本文以词袋模型构建的特征向量为例，在牛津大学建筑物图像数据集上试验了特征维度（在数值上跟聚类单词数目大小相等）对检索精度的影响，从图 1.2 中可以看到，词袋模型的特征维度是非常高的。因此，面向大规模图像数据集检索的另一个典型特点是图像特征描述向量维度高。

(3) 要求响应速度快。对于用户的查询，图像检索系统应该具备迅速响应用户查询的能力，同时由于大规模图像数据量大、特征维度高，直接采用暴力搜索 (Brute Search) 索引策略（也称为线性扫描）难以满足系统实时性的要求，图 1.2 右图所示的是在牛津大学建筑物图像数据集上平均每次查询所耗费的时间，可以看到在图像数量仅有 4063 张的牛津大学建筑物图像集，其查询时间在单词数目为 100 万且重排深度为 1000 的条件下就需要耗费 1 秒左右的时间，并且整个程序还是运行在一台高配的服务器上，因此，大规模图像检索需要解决系统实时响应的问题。

基于哈希的图像检索技术其具体框架如图 1.4 所示，按步骤可以分为特征提取、哈希编码、汉明距离排序以及重排四个步骤：

(1) 特征提取。对图像数据库中的图像逐一进行特征提取，并将其以图像文件名和图像特征一一对应的方式添加到特征库中；

(2) 哈希编码。哈希编码可以拆分成为两个子阶段，在对特征进行编码之前需要有哈希函数集，而哈希函数集则通过哈希函数学习阶段而得到，因此这两个子阶段分别为哈希函数学习阶段和正式的哈希编码阶段。在哈希函数学习阶段，将特征库划分成训练集和测试集，在训练库上对构造的哈希函数集 $H(x)=h_1(x), h_2(x), \dots, h_K(x)$ 进行训练学习；正式的哈希编码阶段时，分别将原来的特征 $x_i (i=1, 2, \dots, N)$ 代入到学习得到的哈希函数集 $H(x)$ 中，从而得到相应的哈希编码。值得注意的是，如果设计的哈希算法已经经过实验验证有效，那么在实际的应用系统中，在划分数据集的时候，可以将整个图像库既作为训练集也作为图像数据库，从而使得在大规模图像上学到的哈希函数具备较好的适应性；

(3) 汉明距离排序。在汉明距离排序阶段，对于给定的查询图像，逐一计算查询图像对应的哈希编码到其他各个哈希编码之间的汉明距离，然后按从小到大的顺序进行相似性排序，从而得到检索结果；

(4) 重排。针对步骤 (3) 汉明排序后的结果，可以选择前 $M(M < N)$ 个结果或者对汉明距离小于某一设置的汉明距离 d_{min} 的结果进行重排。一般地，在重排的时候采用欧式距离作为相似性度量得到重排后的结果。因此，从这里可以看到，哈希过程可以看作是筛选候选样本或是粗排序的过程。在采用哈希方法进行大规模图像检索的应用系统中，通常会有重排这一步，但是在设计哈希算法的时候，对性能进行指标评价直接采用的是汉明距离，也就是在评价哈希算法性能的时候，不需要重排这一



步。

随着视觉数据的快速增长，面向大规模视觉数据的基于内容的图像检索技术不论是在商业应用还是计算机视觉社区都受到了极大的关注。传统的暴力(brute-force)搜索方法(又称线性扫描)通过逐个与数据库中的每个点进行相似性计算然后进行排序，这种简单粗暴的方式虽然很容易实现，但是会随着数据库的大小以及特征维度的增加其搜索代价也会逐步的增加，从而使得暴力搜索仅适用于数据量小的小规模图像数据库，在大规模图像库上这种暴力搜索的方式不仅消耗巨大的计算资源，而且单次查询的响应时间会随着数据样本的增加以及特征维度的增加而增加，为了降低搜索的空间的空间复杂度与时间复杂度，在过去的十几年里研究者们找到了一种可供替代的方案—近似最近邻(ANN, Approximate Nearest Neighbor)搜索方法，并提出了很多高效的检索技术，其中最成功的方法包括基于树结构的图像检索方法、基于哈希的图像检索方法和基于向量量化的图像检索方法。

1.2 计算机视觉通用工具包

1.2.1 OpenCV

如果说 NumPy 的主要目标是用于有效地表示大型多维数组，那么，OpenCV 的主要目标便是实时处理图像。该库从 1999 年发布以来，已随处可见，直到在 2009 年发布的第 2 版中我们才看到了它对 NumPy 的支持。OpenCV 这个库本身是用 C/C++ 写的，但在运行该安装包时它提供了 Python 的绑定。OpenCV 是我手下最喜欢的计算机视觉库，在本书中，我们会经常使用它。

OpenCV 的安装是常常会变化的。由于这个库是用 C/C++ 写的，所以在编译的时候需要特别的注意，并要确保预先要安装的东西都已安装。由于 OpenCV 的最新安装说明是经常会改变的，所以在安装 OpenCV 时要确保自己去查看 OpenCV 的网站<http://opencv.org/>。

1.3 图像特征表达工具包

1.3.1 VLFeat

1.3.2 Yael

Yael 是一个用 C 写的针对大规模图像检索而开发的工具包，它实现了图像检索中常用的诸如 BoVW、VLAD、Fisher 向量、聚类、倒排等算法，此外它还提供了某些函数对应的 Python 接口，使得研究人员和开发者能够很方便快捷地实现一些相应的对比算法。本书在相关的实现上会比较多的借助该库中的一些函数。总之，该库不论是从实用性来看，还是纯粹从学习的角度来看，都是非常值得推荐的一个工具包。



这里对 Yael 的安装过程做一个简单的介绍。首先下载 Yael v438，解压后执行下面命令即可完成 Yael 的安装：

```
1 ./configure.sh --msse4 --enable-numpy  
2 make  
3 python setup.py install
```

1.3.3 Caffe

1.4 索引构建工具包

1.4.1 Falconn

1.4.2 Nmslib

1.4.3 Faiss

你可以在这里找到他的说明：<http://www.guifreitas.com/installing-opencv-2-4-2-on-mac-osx-mountain-lionwith-python-support>。



第2章 工业常用算法



2.1 PCA

```
1
2 /* Train PCA using OpenCV
3 Input:
4     data:           n*d
5     num_reduced_dim: reduced number
6     filename:       path of saved
7 */
8 void trainPCA(cv::Mat &data, int num_reduced_dim, std::string &filename){
9     if(data.cols <= 0 || data.rows <= 0){
10         std::cout << "data is empty, training is failed" << std::endl;
11     }
12     //PCA pca(data, cv::Mat(), CV_PCA_DATA_AS_ROW, 1.0);
13     std::cout << "start traing PCA model" << std::endl;
14     cv::PCA pca(data, cv::Mat(), CV_PCA_DATA_AS_ROW, num_reduced_dim);
15     std::cout << "traing PCA model is finished" << std::endl;
16     std::cout << "start writing PCA model" << std::endl;
17     cv::FileStorage fs(filename, cv::FileStorage::WRITE);
18     pca.write(fs);
19     std::cout << "writing PCA model is finished" << std::endl;
20 }
21
22
23 /* Load PCA model
24 Input:
25     filename:       path of saved
26 */
27 cv::PCA loadPCA(std::string &filename){
28     cv::PCA pca;
29     cv::FileStorage fs(filename, cv::FileStorage::READ);
30     pca.read(fs.root());
31     return pca;
32 }
```

2.2 KMeans

```
1
2 #!/usr/bin/env python
3 # -*- coding: utf-8 -*-
4
5 import os
6 import numpy as np
7 from yael import ynumpy
8 import time
9
10
11 txt_path = '/home/yuanyong/py/fv_retrieval/oxford.txt'
12 sift_dir = '/home/yuanyong/py/fv_retrieval/oxford_hesaff_sift'
13
14 with open(txt_path, 'r') as f:
15     content = f.readlines()
16     content = [x.strip() for x in content]
17
18
19 all_desc = []
20 for i, line in enumerate(content):
21     print "%d(%d): %s" %(i+1, len(content), line)
22     hesaff_path = os.path.join(sift_dir, os.path.splitext(os.path.
23         basename(line))[0] + '.hesaff.sift')
24     hesaff_info = np.loadtxt(hesaff_path)
25     if hesaff_info.shape[0] == 0:
26         continue
27     elif hesaff_info.shape[0] > 0 and len(hesaff_info.shape) == 1:
28         desc = hesaff_info[5:]
29         all_desc.append(desc)
30     elif hesaff_info.shape[0] > 0 and len(hesaff_info.shape) > 1:
31         desc = hesaff_info[:, 5:]
32         all_desc.append(desc)
33
34 # make a big matrix with all image descriptors, rootsift
35 all_desc = np.sqrt(np.vstack(all_desc))
36
37 n_sample = 256 * 1000
38
39 # choose n_sample descriptors at random
40 np.random.seed(1024)
41 sample_indices = np.random.choice(all_desc.shape[0], n_sample)
42 sample = all_desc[sample_indices]
```



```
42
43 # until now sample was in uint8. Convert to float32
44 sample = sample.astype('float32') # yael likes floats better than doubles
45
46 # compute mean and covariance matrix for the PCA
47 mean = sample.mean(axis = 0)
48 sample = sample - mean
49 cov = np.dot(sample.T, sample)
50
51 # compute PCA matrix and keep only 64 dimensions
52 eigvals, eigvecs = np.linalg.eig(cov)
53 perm = eigvals.argsort()                      # sort by increasing
54                                     eigenvalue
55 pca_transform = eigvecs[:, perm[64:128]]      # eigenvectors for the 64 last
56                                     eigenvalues
57
58 # transform sample with PCA (note that numpy imposes line-vectors,
59 # so we right-multiply the vectors)
60 sample = np.dot(sample, pca_transform)
61
62 # train Kmeans
63 print "start_kmeans....."
64
65 k = 128           # number of cluster to create
66 d = sample.shape[1] # dimensionality of the vectors
67 n = sample.shape[0] # number of vectors
68 nt = 20          # number of threads to use
69 niter = 0         # number of iterations (0 for convergence)
70 redo = 1          # number of redo
71
72 t0 = time.time()
73 (centroids, qerr, dis, assign, nassign) = ynumpy.kmeans(sample, k, nt =
74             nt, niter = niter, redo = redo, output = 'full')
75 t1 = time.time()
76 print "kmeans performed in %.3f s" % (t1 - t0)
77
78 np.save("./models/centroids_128_data", centroids)
79 np.save("./models/pca_data", pca_transform)
80 np.save("./models/mean_data", mean)
```



第3章 图像特征表达



要探索计算机视觉世界，首先我们需要安装一些安装包。作为探索计算机视觉之旅的起始，要安装这些安装包特别是 OpenCV 是非常冗余乏味的，它依赖于你所使用的操作系统。我已经试着将这些安装说明简化成了一份简短的使用指南，不过，正如你所了解的，随着项目、网站的改变，这些安装说明是会变化的！如果你在安装中遇到了问题，确保查看安装包的网站以获得最新安装说明。

我强力推荐你要么用 easy_install 或 pip 来管理你那些安装包。它会使你的生活变得更容易！

最后，如果你不想自己逐一安装这些安装包的话，我已经预先安装好后所有需要的安装包，并把它们放在一个 Ubuntu 的虚拟机里。使用该虚拟机可以使你跳过那些繁琐的安装包的安装，直接进入本书的实例中，而不必理会这些安装包的管理、安装说明以及编译错误。

要想获得更多这个预先配置好的虚拟机，可以前往<http://www.pyimagesearch.com/practical-python-opencv/>查看详细信息。

现在，让我们来安装这些安装包！

3.1 局部特征

NumPy 是一个用于 Python 编程语言的库，它对大型多维数组提供支持。为什么 NumPy 很重要呢？原因是使用 NumPy，我们可以将图像表示为一个多维数组。将图像表示为 NumPy 数组，不仅易于计算和资源得以有效利用，而且一些其他的图像处理操作和机器学习库也是用 NumPy 数组来表示的。此外，使用 NumPy 内置的高水准数学函数，我们能在一幅图像上做快速数值分析。

与 NumPy 形影不离的是 SciPy。SciPy 为科学和技术计算提供了更多的支持。

3.1.1 SIFT 特征

到目前为止，在你的 Windows 系统上安装 NumPy 和 SciPy 最简单的方式是<http://www.scipy.org/install.html>下载并安装二进制分发包。

3.1.2 HOG 特征

如果你运行的是 OSX 10.7.0(Lion) 或更高版本，NumPy 和 SciPy 已经预先安装好了。

不过，我喜欢安装 ScipySuperpack。它包含了最新版的 NumPy、SciPy、Matplotlib 和其他非常有用的安装包，比如 ipython、pandas 和 scikit-learn。所有的这些安装包是值得安装的，并且如果你在www.PyImageSearch.com上阅读我的博客，你会发现这些安装包我用得非常的频繁。

3.2 全局特征

用一句简洁的话概括，matplotlib 是一个绘图库。如果你以前用过 MATLAB，在 matplotlib 环境里你很可能会觉得非常的舒适。在分析图像的时候，我们会用到它。不管是画图像直方图还是只是简单地查看图像本身，matplotlib 是一个非常有用的工具，你值得将它纳入到你的工具箱中。

3.2.1 CNN 特征

Matplotlib 可以在<http://matplotlib.org/> 上下载。如果你已经安装了 ScipySuperpack，那么 Matplotlib 就已经在你的计算机上安装了。你也可以通过 easy_install 或 pip 来安装它。

此外，在 matplotlib 官网上还提供了 Windows 的二进制安装文件。



第 4 章 载入、显示及保存



这本书的初衷是成为一本实用手册，用于指引怎样用 Python 和 OpenCV 来开始计算机视觉编程。本着这样一条信念，那就让我们不要浪费任何时间了。拿起你的键盘开始写些简单的代码，用于从磁盘载入一幅图像，在屏幕上显示它，并将它用不同的格式写入一个文件。在执行该代码的时候，我们的 Python 脚本应该可以在屏幕上显示我们载入的图像，正如图 3.1 所示。

首先，我们创建一个名为 load_display_save.py 来包含我们的代码，现在我们可以开始写代码了：

列表 3.1: load_display_save.py

```
1 import argparse
2 import cv2
3
4 ap = argparse.ArgumentParser()
5 ap.add_argument("-i", "--image", required = True,
6                 help = "Path to the image")
7 args = vars(ap.parse_args())
```

我们要做的第一件事是导入在这个例子中我们需要的包。我们用了 argparse 来解析我们的命令行参数。然后，导入 cv2 - 这里 cv2 就是我们的 OpenCV 库，它包含了我们需要的图像处理函数。

上面代码第 4-7 行用于解析命令行参数。这里我们需要的唯一一个参数是 -image：磁盘上图像所在的路径。最后，我们解析这些参数，并在它们保存在一个字典中。

列表 3.2: load_display_save.py

```
1 image = cv2.imread(args["image"])
2 print "width: %d pixels" % (image.shape[1])
3 print "height: %d pixels" % (image.shape[0])
4 print "channels: %d" % (image.shape[2])
5
6 cv2.imshow("Image", image)
7 cv2.waitKey(0)
```

既然我们有了图像所在的路径，在第 8 行我们就可以用 cv2.imread 函数从磁盘载入该图像。cv2.imread 函数返回一个 NumPy 数组，用于表示一幅图像。

第 9-11 行检查图像的维数。由于图像是用 NumPy 数组来表示的，我们可以用 shape 属性来检查该图像的宽、高，以及颜色通道数。

最后，第 13 行和第 14 行用于在屏幕上显示实际载入的图像。cv2.imshow 的第一

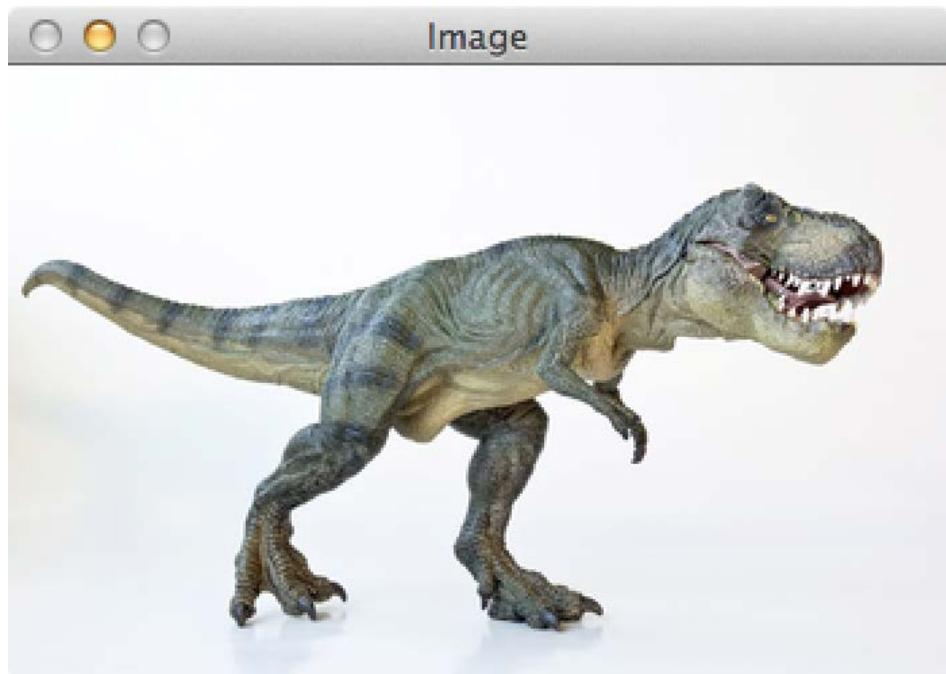


图 4.1: 载入并在屏幕上显示一幅霸王龙图像的例子

一个参数是一个字符串，是窗口的“名字”；第二个参数是我们在第 8 行从磁盘载入的图像。最后调用了 cv2.WaitKey 函数来暂停运行的脚本，直到我们在键盘上按下一个键。用参数“0”表示按下任意一个键就会释放暂停。

最后，我们将该幅图像以 JPG 的格式写入文件中：

列表 3.3: load_display_save.py

```
1 cv2.imwrite("newimage.jpg", image)
```

要运行上面的脚本并显示出图片，我们只需要打开终端窗口，并执行下面的命令：

列表 3.4: load_display_save.py

```
1 $ python load_display_save.py --image ../images/trex.png
```

如果不错什么问题的话，你应该可以在你的屏幕上看到图 3.1 中的霸王龙。要终止脚本的执行的话，只需要在图像窗口上单击，并按下任意键。

检查上面脚本的输出结果，你应该可以看到图片的一些基本信息。你会看到对于那张霸王龙图片，它的宽是 350 个像素，高是 228 个像素，以及 3 个颜色通道 (R、G、B 三个颜色通道)。用一个 NumPy 数组表示的话，该数组大小为 (350,228,3)。

当我们在写矩阵时，通常我们会将它们写成这种形式：# 行 × # 列——但对于 NumPy，便不是这样的了。NumPy 通常会先给出列数，然后才是行数。这点，你要牢牢记住。

最后，注意文件夹里的内容。你会看到有一个新文件在那儿：newimage.jpg。OpenCV 会自动地为我们将 PNG 格式的图片转换为 JPG 格式图片！所以在图像格式间进行转换时我们并不需要费什么劲。



下一章，我们将探索怎样在一幅图像上获取像素值并对像素值进行操作。



第 5 章 图像基础



在这一章中，我们将审查图像的积木——像素。我们将详细的讨论什么是像素，以及像素是怎样形成一幅图像的，并且怎么在 OpenCV 里对这些像素进行操作。

每幅图像由像素集组成。像素是一幅图像的原材料，积木。它是图像的最小单元。

一般情况下，像素通常采用两种方式进行表示：灰度和彩色。在灰度图像中，每一个像素的像素值在 0 到 255 之间，“0”代表“黑色”，“1”表示“白色”。

第6章 特征编码



局部特征作为一种非常鲁棒性的特征，其与全局特征构成了计算机领域图像内容描述的基础。相比于全局特征，局部特征在刻画图像局部区域的形状、纹理、梯度等方面，往往能够更细的粒度，因而对图像内容的表征也更丰富，但同时也引发了新的问题，即特征处理效率低、存储大等方面的问题。因而需要将局部特征经过某种编码方式，最终表示成一种紧凑的全局特征表示。

6.1 BoF 词袋模型

基于 SIFT 局部特征的 BOF 模型非常适合于做 Object retrieval, 下面是自己在 oxford building 数据库 (5063 张图片) 上进行的一些实验。表格中单词数目为聚类时设定的聚类数目，以及是否采用 SIFT 或者 rootSIFT，rootSIFT 怎么计算的可以阅读 Object retrieval with large vocabularies and fast spatial matching 这篇文章，空间校正即在重排的时候，对错配的 SIFT 点对进行剔除，剔除的方法可以采用 RANSAC 或者类 RANSAC 方法，详细介绍可以阅读 SIFT(ASIFT) Matching with RANSAC，检索精度采用平均检索精度 (mean Average Precision, mAP)，其计算过程可以阅读信息检索评价指标这篇文章。下面需要注意的是查询时间单次查询的结果，并没有进行多次查询进行平均，此外查询时间是查询和计算 mAP 时间的总和。

6.2 VLAD 局部聚合向量

在进行理论分析之前，先来看看 VLAD 长个什么样子，这里本小子分步展开 VLAD 是怎么得来的。

提取 SIFT 特征。对于一个样本数为 N 的数据库，先对图像库中的所有图像提取 SIFT 描述子，假设提取到了所有 SIFT 描述子数目为 n , 用 X 来表示的话， X 就是一个 $n \times 128$ 的矩阵。聚类生成词汇向量。假设要生成 K 个单词，对 X 直接用 Kmeans 聚成 K 类，类中心即为单词 (也叫码字)。生成 VLAD 向量。这一步其实如果对 BOW 的生成过程清楚的话，这一步理解起来就非常简单了。BoW 统计的是描述子落入最近单词里的数目，而 VLAD 统计的则是这些落入最近单词里与该单词的累积残差。根据 Aggregating local image descriptors into compact codes 的描述：

BOW 做的是描述子的 0 阶统计分布，而 FV 则是扩展了的 BOW 的高阶统计。这里引出来的 FV 是什么呢？VLAD 是 FV 的特例，这里我们先不关注 FV，我们只要借

此推得 VLAD 是 BOW 的高阶统计就行。

这里我们可以直接调用 INRIA 开发的 Yael 工具包，该工具包提供了 BoW、VALD 以及 FV 的接口。为了更好的理解 VALD 编码的过程，可以打开 Yael 的 vlad.c 文件，其中有 *vlad_compute* 方法：

```

1 void vlad_compute( int k, int d, const float *centroids,
2                     int n, const float *v, float *desc)
3 {
4     int i, j;
5     int *assign = ivec_new(n);
6     nn(n, k, d, centroids, v, assign);
7     fvec_0(desc, k * d);
8     for (i = 0; i < n; i++) {
9         for (j = 0; j < d; j++)
10            desc[assign[i]*d+j] += v[i*d+j] - centroids[assign[i]*d+j];
11    }
12    free(assign);
13 }
```

上面清楚的显示了得到的 desc(即 VLAD 特征表示) 为距离类中心最近的局部特征的累积和，其中方法 nn 是做最近邻查找。有了这个接口后，我们要做的就是提取局部特征，比如 SIFT，然后使用 Yael 里提供的 KMeans 接口做聚类，得到聚类中心，然后调用该函数，即可得到 VLAD 特征表示。

```

1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3 # Author: yongyuan.name
4
5 import os, h5py
6 import numpy as np
7 from yael import ynumpy
8
9 # save feature to h5py
10 h5f = h5py.File('./models/vlad_128_8192.h5', 'w')
11 # load PCA and KMeans model
12 centroids = np.load('./models/centroids_data.npy')
13 pca_transform = np.load('./models/pca_data.npy')
14 mean = np.load('./models/mean_data.npy')
15
16 #
17 txt_path = '/home/yuanyong/py/fv_retrieval/oxford.txt'
18 with open(txt_path, 'r') as f:
19     content = f.readlines()
20     content = [x.strip() for x in content]
```



```
22 # compute VLAD feature
23 features = []
24 image_names = []
25 for i, line in enumerate(content):
26     img_name = os.path.basename(line)
27     print "%d(%d): %s" %(i+1, len(content), img_name)
28     hesaff_path = os.path.join('/home/yuanyong/py/fv_retrieval/',
29                               'oxford_hesaff_sift', os.path.splitext(os.path.basename(line))[0] +
30                               '.hesaff.sift')
31     hesaff_info = np.loadtxt(hesaff_path)
32     if hesaff_info.shape[0] == 0:
33         hesaff_info = np.zeros((1, 133), dtype = 'float32')
34     elif hesaff_info.shape[0] > 0 and len(hesaff_info.shape) == 1:
35         hesaff_info = hesaff_info.reshape([1, 133])
36
37     image_desc = np.sqrt(hesaff_info[:, 5:])
38
39     # apply the PCA to the image descriptor
40     image_desc = np.dot(image_desc - mean, pca_transform)
41     image_desc = image_desc.astype(np.float32)
42
43     # compute VLAD
44     vlad = numpy.vlad(centroids, image_desc) # num_centroids * d
45     # L2 normalization
46     norms_vlad = np.sqrt(np.sum(vlad ** 2, 1))
47     norms_vlad[norms_vlad < 1e-12] = 1e-12
48     vlad /= norms_vlad.reshape(-1, 1)
49     vlad = vlad.flatten()
50     features.append(vlad)
51     image_names.append(img_name)
52
53     # make one matrix with all FVs
54     features = np.vstack(features)
55
56     # power-normalization, square-rooting normalization
57     features = np.sign(features) * np.abs(features) ** 0.5
58
59     # L2 normalize
60     norms = np.sqrt(np.sum(features ** 2, 1))
61     norms[norms < 1e-12] = 1e-12
62     features /= norms.reshape(-1, 1)
63
64     # save feats
65     print "number samples: %d, dimension: %d, number names: %d" %(features.
66                     shape[0], features.shape[1], len(image_names))
```



```

64 h5f[ 'feats' ] = features
65 h5f[ 'names' ] = image_names
66 h5f.close()

```

经过上面三个步骤后，一幅图像可以用一个 $1(K128)$ 维的向量表示。为了初步验证上面的过程是否正确，来看看上面那篇论文中 VLAD 的维数是否如这里所理解的是一个 $1(K128)$ 维的向量，直接看实验表：

6.2.1 VLAD 归一化

6.3 Fisher Vector

6.3.1 Fisher 核

在空间 χ 中，某样本 X 存在 T 个观测量，记为 $X = \{x_t, t = 1 \dots T\}$ 。对应到图像上，样本 X 为图像 I 提取到的 T 个 D 维的局部描述子，比如 SIFT。设 μ_λ 为概率密度函数，该函数包含有 M 个参数，即 $\lambda = [\lambda_1, \dots, \lambda_M]$ 。根据生成式原理，空间 χ 中的元素 X 可以由概率密度函数进行建模。在统计学上，分数函数 (score function) 可以由对数似然函数的梯度给出，即：

$$G_\lambda^X = \nabla \log u_\lambda(X) \quad (6.1)$$

上式(6.1)对数函数的梯度，在数学形式上为对数似然函数的一阶偏导，它描述了每一个参数 λ_i 对该生成式过程的贡献度，换言之，该分数函数 G_λ^X 描述了生成式模型 μ_λ 为了更好的拟合数据 X ，该模型中的参数需要做怎样的调整。又因为 $G_\lambda^X \in R^M$ 是一个维度为 M 维的向量，所以该分数函数的维度仅依赖于 λ 中参数的数目 M ，而与观测样本的数目 T 无关。此外，一般情况下，该分数函数的期望 $E[G_\lambda^X] = 0$ ，这一点对于下面讲到的 Fisher 信息矩物理意义的得到非常重要。

根据信息几何理论，含参分布 $\Gamma = \{\mu_\lambda, \lambda \in \Lambda\}$ 可以视为一个黎曼流形 M_Λ ，其局部度量方式可以由 Fisher 信息矩 (Fisher Information Matrix, FIM) $F_\lambda \in R^{M \times M}$ 。

$$F_\lambda = E_{x \sim u_\lambda}[G_\lambda^X (G_\lambda^X)^T] \quad (6.2)$$

从上式可以看到，Fisher 信息矩是分数函数的二阶矩。在一般条件下很容易证明（注意 $E[G_\lambda^X] = 0$ ）：

$$\begin{aligned}
F_\lambda &= E_{x \sim u_\lambda}[G_\lambda^X (G_\lambda^X)^T] \\
&= E[(G_\lambda^X)^2] \\
&= E[(G_\lambda^X)^2] - E[G_\lambda^X]^2 \\
&= Var[G_\lambda^X]
\end{aligned} \quad (6.3)$$



从上式(6.3)可以看到, Fisher 信息矩是用来估计最大似然估计 (Maximum Likelihood Estimate, MLE) 的方程的方差。它直观的表述就是, 在独立性假设的条件下, 随着收集的观测数据越来越多, 这个方差由于是一个相加的形式, 因而 Fisher 信息矩也就变的越来越大, 也就表明得到的信息越来越多。【注: 此处引用了<https://www.zhihu.com/question/26561604>】

对于两组不同的观察样本 X 和 Y , Jaakkola 和 Haussler 提出了使用 Fisher 核来度量它们之间的相似性, 其数学表达形式为:

$$K_F K(X, Y) = (G_\lambda^X)^T F_\lambda^{-1} G_\lambda^Y \quad (6.4)$$

又因为 F_λ 是半正定的, 所以其逆矩阵是存在的。使用 cholesky 分解可以得到 $F_\lambda^{-1} = (L_\lambda)^T L_\lambda$, 式(6.4)可以写成内积的表示形式:

$$K_F K(X, Y) = (\phi_\lambda^X)^T \phi_\lambda^Y \quad (6.5)$$

其中,

$$\phi_\lambda^X = L_\lambda G_\lambda^X = L_\lambda \nabla_\lambda \log u_\lambda(X) \quad (6.6)$$

上式(6.6)是 L_λ 对 G_λ^X 的归一化, 我们将 ϕ_λ^X 称为 Fisher 向量, 该 Fisher 向量 ϕ_λ^X 等于梯度向量 G_λ^X 的维度, 又由于 G_λ^X 的维度仅与概率密度函数的参数数目 M 有关, 所以空间 χ 中任意样本 X 的 T 个观测量最终都可以表示成一固定维度的向量。通过使用 ϕ_λ^X 算子, 使得非线性核相似性度量问题转化为线性问题。这种变换带来的一个明显的优势是, 在分类的时候可以采用更高效的线性分类器。

6.3.2 图像的 Fisher 向量

设 $X = \{x_t, t = 1 \dots T\}$ 是从一幅图像上提取到的 T 个局部描述子, 比如 SIFT。假设这 T 个局部描述子是相互独立的 (注意此假设提前), 则公式(6.6)可以写成如下形式:

$$\phi_\lambda^X = \sum_{n=1}^T L_\lambda \nabla_\lambda \log u_\lambda(x_t) \quad (6.7)$$

从式(6.7)可知, 在独立性假设的前提下, Fisher 向量是图像中每一个局部描述子归一化梯度统计 $L_\lambda \nabla_\lambda \log u_\lambda(x_t)$ 的求和。我们可以定义如下操作:

$$x_t \longrightarrow \varphi_{FK}(x_t) = L_\lambda \nabla_\lambda \log u_\lambda(x_t) \quad (6.8)$$

该操作将任意一个局部描述子 x_t 嵌入到高维空间中, 使得其更易于使用线性分类器进行分类。在得到式(6.6)的时候, 有一个基本的前提假设, 即局部描述子之间是相互独立的, 也就是在提取图像的局部描述子的时候每个小区块是相互独立的, 显然这样一种假设对于图像而言是不成立的, 特别是当每个小区快中间有重叠的时候。对于这个问题的讨论, 我们放在后面展开。



由于高斯混合模型 (Gaussian Mixture Model, GMM) 可以以任意精度去逼近任意连续分布，所以对概率密度函数 u_λ ，选用高斯混合模型。值得一提的是，在计算机视觉文献中，将用来对图像局部特征进行建模的高斯混合模型称为概率视觉词汇 (Probabilistic Visual Vocabulary)。对于有 K 个组分的高斯混合模型，其包含的参数为 $\lambda = \{w_k, \mu_k, \Sigma_k, k = 1, \dots, K\}$ ，其中 w_k 、 μ 、 Σ_k 分别是第 k 个高斯的混合权重，均值向量和协方差矩阵。因此，概率密度函数：

$$u_\lambda(x) = \sum_{n=1}^K w_k u_k(x) \quad (6.9)$$

其中，

$$u_k(x) = \frac{1}{(2\pi)^{D/2} |\Sigma_k|^{1/2}} \exp\left\{-\frac{1}{2}(x - \mu_k)^T \Sigma_k^{-1} (x - \mu_k)\right\} \quad (6.10)$$

为了保证 $u_\lambda(x)$ 是一个有意义的概率密度函数，会对其做如下约束：

$$\begin{aligned} \sum_{n=1}^K w_k &= 1 \\ w_k &\geq 0 (k = 1, \dots, K) \end{aligned} \quad (6.11)$$

6.3.3 Fisher Vector 实验

提取 SIFT 特征，这里我们采用 Hesaff SIFT，为了提取 Hesaff SIFT，首先需要编译得到二进制文件，其后，使用下面的 Python 脚本可完成 Hesaff SIFT 特征的提取，当然，也可以直接使用 OpenCV 的 SIFT 特征。

```

1 #!/usr/bin/env python
2 # encoding: utf-8
3 # Author: yongyuan.name
4
5 import os
6 import multiprocessing
7 from multiprocessing import Process, freeze_support, Pool
8
9
10 def split_list(alist, wanted_parts=1):
11     length = len(alist)
12     return [alist[i * length // wanted_parts: (i + 1) * length // wanted_parts]
13             for i in range(wanted_parts)]
14
15 def cpu_task(img_names, bbin, save_dir):
16     for i, line in enumerate(img_names):
17         img_path = os.path.join('/home/yuanyong/datasets/oxford', line)
18         cmd = bbin + ' ' + img_path + ' ' + save_dir

```



```

19         os.system(cmd) # returns the exit status
20         print "%d(%d), %s" %(i+1, len(img_names), line)
21
22 if __name__ == '__main__':
23
24     multiprocessing.freeze_support()
25     pool = multiprocessing.Pool()
26
27     parts = 50
28     bbin = '/home/yuanyong/cpp/hesaff/hesaff'
29     txt_path = '/raid/yuanyong/yael_fv/oxford.txt'
30     save_dir = '/raid/yuanyong/yael_fv/oxford_hesaff_sift/'
31
32     with open(txt_path, 'r') as f:
33         content = f.readlines()
34         content = [x.strip() for x in content]
35     blocks = split_list(content, wanted_parts = parts)
36
37     for i in xrange(0, parts):
38         pool.apply_async(cpu_task, args=(blocks[i], bbin, save_dir,))
39     pool.close()
40     pool.join()

```

在提取到 SIFT 特征后，便可以使用 GMM 模型进行聚类，得到 GMM 的参数了：

```

1 #!/usr/bin/env python
2 # encoding: utf-8
3 # Author: yongyuan.name
4
5 import os
6 import numpy as np
7 from yael import ynumpy
8
9 txt_path = '/home/yuanyong/py/fv_retrieval/oxford.txt'
10 sift_dir = '/home/yuanyong/py/fv_retrieval/oxford_hesaff_sift'
11
12 with open(txt_path, 'r') as f:
13     content = f.readlines()
14     content = [x.strip() for x in content]
15
16
17 all_desc = []
18 for i, line in enumerate(content):
19     print "%d(%d): %s" %(i+1, len(content), line)
20     hesaff_path = os.path.join(sift_dir, os.path.splitext(os.path.
21         basename(line))[0] + '.hesaff.sift')

```



```
21     hesaff_info = np.loadtxt(hesaff_path, skiprows=2)
22     if hesaff_info.shape[0] == 0:
23         continue
24     elif hesaff_info.shape[0] > 0 and len(hesaff_info.shape) == 1:
25         desc = hesaff_info[5:]
26         all_desc.append(desc)
27     elif hesaff_info.shape[0] > 0 and len(hesaff_info.shape) > 1:
28         desc = hesaff_info[:, 5:]
29         all_desc.append(desc)
30
31
32 # make a big matrix with all image descriptors
33 all_desc = np.sqrt(np.vstack(all_desc))
34 #n_sifts = all_desc.shape[0]
35 #for i in range(n_sifts):
36 #    if np.linalg.norm(all_desc[i], ord=2) == 0.0:
37 #        continue
38 #    all_desc[i] = all_desc[i]/np.linalg.norm(all_desc[i], ord=2)
39
40 # sift: root-sift
41 #n_sifts = all_desc.shape[0]
42 #for i in range(n_sifts):
43 #    if np.linalg.norm(all_desc[i], ord=1) == 0.0:
44 #        continue
45 #    all_desc[i] = np.sqrt(all_desc[i]/np.linalg.norm(all_desc[i], ord=1))
46
47 # sift: sign(x)log(1 + |x|)
48 #n_sifts = all_desc.shape[0]
49 #for i in range(n_sifts):
50 #    all_desc[i] = np.sign(all_desc[i]) * np.log(1.0 + np.abs(all_desc[i]))
51
52
53 k = 128
54 n_sample = 256 * 1000
55
56 # choose n_sample descriptors at random
57 np.random.seed(1024)
58 sample_indices = np.random.choice(all_desc.shape[0], n_sample)
59 sample = all_desc[sample_indices]
60
61 # until now sample was in uint8. Convert to float32
62 sample = sample.astype('float32')
63
```



```

64 # compute mean and covariance matrix for the PCA
65 mean = sample.mean(axis = 0)
66 sample = sample - mean
67 cov = np.dot(sample.T, sample)
68
69 # compute PCA matrix and keep only 64 dimensions
70 eigvals, eigvecs = np.linalg.eig(cov)
71 perm = eigvals.argsort()                      # sort by increasing
    eigenvalue
72 pca_transform = eigvecs[:, perm[96:128]]      # eigenvectors for the 64 last
    eigenvalues
73
74 # transform sample with PCA (note that numpy imposes line-vectors,
75 # so we right-multiply the vectors)
76 sample = np.dot(sample, pca_transform)
77
78 # train GMM
79 print "start_train_GMM....."
80 gmm = ynumpy.gmm_learn(sample, k, nt = 400, niter = 2000, seed = 0, redo
    = 1, use_weights = True)
81
82 np.save("./oxford_gmm_root_32/w.gmm", gmm[0])
83 np.save("./oxford_gmm_root_32/mu.gmm", gmm[1])
84 np.save("./oxford_gmm_root_32/sigma.gmm", gmm[2])
85 np.save("./oxford_gmm_root_32/mean.gmm", mean)
86 np.save("./oxford_gmm_root_32/pca_transform.gmm", pca_transform)

```

```

1 #!/usr/bin/env python
2 # encoding: utf-8
3 # Author: yongyuan.name
4
5 import os, h5py
6 import numpy as np
7 from yael import ynumpy
8
9
10 h5f = h5py.File('./oxford_gmm_root_32/fisher_8192.h5', 'w')
11 weights = np.load('./oxford_gmm_root_32/w.gmm.npy')
12 mu = np.load('./oxford_gmm_root_32/mu.gmm.npy')
13 sigma = np.load('./oxford_gmm_root_32/sigma.gmm.npy')
14 mean = np.load('./oxford_gmm_root_32/mean.gmm.npy')
15 pca_transform = np.load('./oxford_gmm_root_32/pca_transform.gmm.npy')
16
17 gmm = [weights, mu, sigma]
18

```



```
19 #
20 txt_path = '/home/yuanyong/py/fv_retrieval/oxford.txt'
21 with open(txt_path, 'r') as f:
22     content = f.readlines()
23     content = [x.strip() for x in content]
24
25 #
26 features = []
27 image_names = []
28 for i, line in enumerate(content):
29     img_name = os.path.basename(line)
30     print "%d(%d): %s" %(i+1, len(content), img_name)
31     hesaff_path = os.path.join('/home/yuanyong/py/fv_retrieval/',
32                               'oxford_hesaff_sift', os.path.splitext(os.path.basename(line))[0] +
33                               '.hesaff.sift')
34     hesaff_info = np.loadtxt(hesaff_path)
35     if hesaff_info.shape[0] == 0:
36         hesaff_info = np.zeros((1, 133), dtype = 'float32')
37     elif hesaff_info.shape[0] > 0 and len(hesaff_info.shape) == 1:
38         hesaff_info = hesaff_info.reshape([1, 133])
39
40     image_desc = np.sqrt(hesaff_info[:, 5:])
41     #n_sifts = image_desc.shape[0]
42     #for i in range(n_sifts):
43     #    if np.linalg.norm(image_desc[i], ord=2) == 0.0:
44     #        continue
45     #    image_desc[i] = image_desc[i]/np.linalg.norm(image_desc[i], ord=2)
46
47     # root-sift
48     #n_sifts = image_desc.shape[0]
49     #for i in range(n_sifts):
50     #    if np.linalg.norm(image_desc[i], ord=1) == 0.0:
51     #        continue
52     #    image_desc[i] = np.sqrt(image_desc[i]/np.linalg.norm(image_desc[
53     #        i], ord=1))
54
55     #n_sifts = image_desc.shape[0]
56     #for i in range(n_sifts):
57     #    image_desc[i] = np.sign(image_desc[i]) * np.log(1.0 + np.abs(
58     #        image_desc[i]))
59
60     # apply the PCA to the image descriptor
61     image_desc = np.dot(image_desc - mean, pca_transform)
62     image_desc = image_desc.astype(np.float32)
```



```
59
60     # compute the Fisher vector , using only the derivative w.r.t mu
61     fv = numpy.fisher(gmm, image_desc, include = [ 'mu' , 'sigma' ])
62     features.append(fv)
63     image_names.append(img_name)
64
65 # make one matrix with all FVs
66 features = np.vstack(features)
67
68 # normalizations are done on all descriptors at once
69
70 # power-normalization
71 features = np.sign(features) * np.abs(features) ** 0.5
72
73 # L2 normalize
74 #norms = np.sqrt(np.sum(image_fvs ** 2, 1))
75 #image_fvs /= norms.reshape(-1, 1)
76
77 # save feats
78 print "number_of_samples: %d, dimension: %d, number_of_names: %d" %(features.
    shape[0], features.shape[1], len(image_names))
79 h5f['feats'] = features
80 h5f['names'] = image_names
81 h5f.close()
```

每幅图像由像素集组成。像素是一幅图像的**原材料，积木**。它是图像的最小单元。

一般情况下，像素通常采用两种方式进行表示：灰度和彩色。在灰度图像中，每一个像素的像素值在0到255之间，“0”代表“黑色”，“1”表示“白色”。



第7章 ANN 搜索



7.1 矩阵相乘

7.2 基于树的方法

几乎所有的 ANN 方法都是对全空间的划分，所以基于树的方法也不例外。基于树的方法采用树这种数据结构的方法来表达对全空间的划分，其中又以 KD 树最为经典。下面左图是 KD 树对全空间的划分过程，以及用树这种数据结构来表达的一个过程。

对 KD 树选择从哪一维度进行开始划分的标准，采用的是求每一个维度的方差，然后选择方差最大的那个维度开始划分。这里有一个比较有意思的问题是：为何要选择方差作为维度划分选取的标准？我们都应该知道，方差的大小可以反映数据的波动性。方差大表示数据波动性越大，选择方差最大的开始划分空间，可以使得所需的划分面数目最小，反映到树数据结构上，可以使得我们构建的 KD 树的树深度尽可能的小。为了更进一步加深对这一点的认识，可以以一个简单的示例图说明：

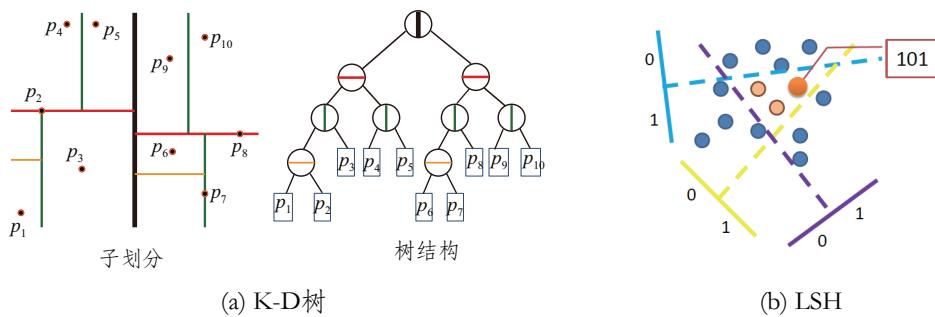


图 7.1: PQ 量化索引示意过程

假设不以方差最大的 x 轴为划分面 (

$$x_{var} = 16.25$$

), 而是以 y 轴 (

$$y_{var} = 0.0$$

) 轴为划分面，如图中虚线所示，可以看到，该划分使得图中的四个点都落入在同一个子空间中，从而使得该划分成为一个无效的划分，体现在以树结构上，就是多了一层无用的树深度。而以 x 轴为初始划分则不同 (图像实线所示)，以 x 轴为初始划分可以得到数据能够比较均匀的散布在左右两个子空间中，从而使得整体的查找时间能够

最短。注意，在实际的 kd 树划分的时候，并不是图中虚线所示，而是选取中值最近的点。上面示意图构建的具体 kd 树如下所示：

一般而言，在空间维度比较低的时候，KD 树是比较高效的，当空间维度较高时，可以采用下面的哈希方法或者矢量量化方法。

7.3 哈希方法

哈希，顾名思义，就是将连续的实值散列化为 0、1 的离散值。在散列化的过程中，对散列化函数（也就是哈希函数）有一定的要求。根据学习的策略，可以将哈希方法分为无监督、有监督和半监督三种类型。在评估某种哈希方法用于图像检索的检索精度时，可以使用 knn 得到的近邻作为 ground truth，也可以使用样本自身的类别作为 ground truth。所以在实际评估准确度的时候，根据 ground truth 的定义，这里面是有一点小的 trick 的。通常对于无监督的哈希图像检索方法，由于我们使用的都是未标记的数据样本，所以我们会很自然的采用 knn 得到的近邻作为 ground truth，但是对于图像检索的这一任务时，在对哈希函数的构造过程中，通常会有“相似的样本经编码后距离尽可能的近，不相似的样本编码后则尽可能的远”这一基本要求，这里讲到的相似，指语义的相似，因而你会发现，编码的基本要求放在无监督哈希方法里，似乎与采用 knn 得到的近邻作为 ground truth 的评价方式有些南辕北辙。对无监督哈希方法的 ground truth 一点小的疑惑在小白菜读书的时候就心存这样的困惑，一直悬而未解。当然，在做无监督的图像哈希方法，采用样本自身的类别作为 ground truth 是毋庸置疑的。

小白菜读书那会儿，研究了很多的哈希图像检索方法（见 hashing-baseline-for-image-retrieval），有时候总会给一些工程实践上的错觉（在今天看来是这样的），即新论文里的方法远远碾压经典的方法，那是不是在实际中这些方法就很 work 很好使。实践的经历告诉小白菜，还是经典的东西更靠谱，不是因为新的方法不好，而是新的事物需要经过时间的沉淀与优化。

所以，这里不会对近两年的哈希方法做铺陈，而是聊一聊工程中在要使用到哈希方法的场景下一般都会选用的局部敏感哈希（Local Sensitive Hashing, LSH）。

7.4 矢量量化方法

矢量量化方法，即 vector quantization，其具体定义为：将一个向量空间中的点用其中的一个有限子集来进行编码的过程。在矢量量化编码中，关键是码本的建立和码字搜索算法。比如常见的聚类算法，就是一种矢量量化方法。而在 ANN 近似最近邻搜索中，向量量化方法又以乘积量化 (PQ, Product Quantization) 最为典型。在之前的博文基于内容的图像检索技术的最后，对 PQ 乘积量化的方法做过简单的概要。在这一小节里，小白菜结合自己阅读的论文和代码对 PQ 乘积量化、倒排乘积量化 (IVFPQ) 做



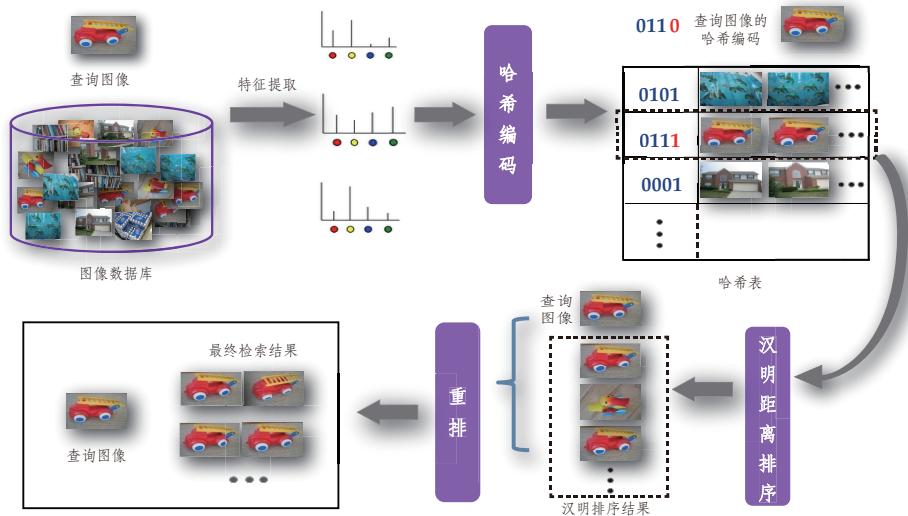


图 7.2: PQ 量化索引示意过程

一种更加直观的解释。

7.4.1 PQ 乘积量化

PQ 乘积量化的核心思想还是聚类，或者说具体应用到 ANN 近似最近邻搜索上，K-Means 是 PQ 乘积量化子空间数目为 1 的特例。PQ 乘积量化生成码本和量化的过程可以用如下图示来说明：

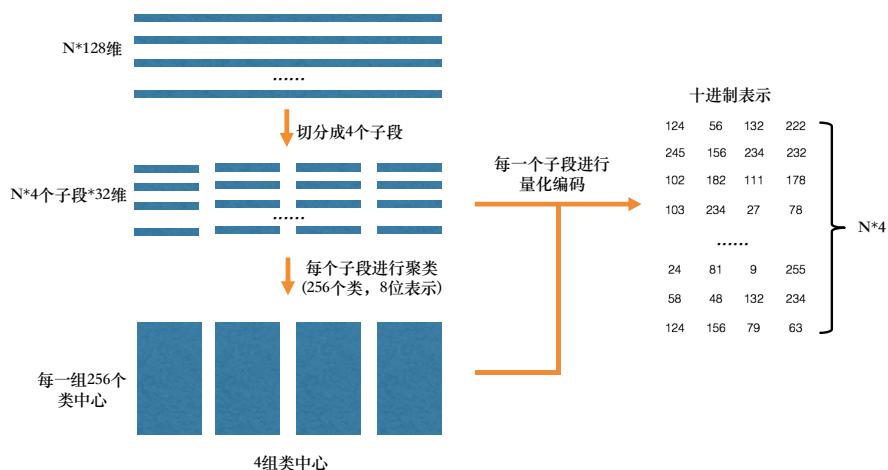


图 7.3: PQ 量化索引示意过程

在训练阶段，针对 N 个训练样本，假设样本维度为 128 维，我们将其切分为 4 个子空间，则每一个子空间的维度为 32 维，然后我们在每一个子空间中，对子向量采用 K-Means 对其进行聚类（图中示意聚成 256 类），这样每一个子空间都能得到一个码本。这样训练样本的每个子段，都可以用子空间的聚类中心来近似，对应的编码即为类中心的 ID。如图所示，通过这样一种编码方式，训练样本仅使用的很短的一个编



码得以表示，从而达到量化的目的。对于待编码的样本，将它进行相同的切分，然后在各个子空间里逐一找到距离它们最近的类中心，然后用类中心的 id 来表示它们，即完成了待编码样本的编码。

正如前面所说的，在矢量量化编码中，关键是码本的建立和码字的搜索算法，在上面，我们得到了建立的码本以及量化编码的方式。剩下的重点就是查询样本与 dataset 中的样本距离如何计算的问题了。

在查询阶段，PQ 同样在计算查询样本与 dataset 中各个样本的距离，只不过这种距离的计算转化为间接近似的方法而获得。PQ 乘积量化方法在计算距离的时候，有两种距离计算方式，一种是对称距离，另外一种是非对称距离。非对称距离的损失小（也就是更接近真实距离），实际中也经常采用这种距离计算方式。下面过程示意的是查询样本来到时，以非对称距离的方式（红框标识出来的部分）计算到 dataset 样本间的计算示意：

具体地，查询向量来到时，按训练样本生成码本的过程，将其同样分成相同的子段，然后在每个子空间中，计算子段到该子空间中所有聚类中心得距离，如图中所示，可以得到 4×256 个距离，这里为便于后面的理说明，小白菜就把这些算好的距离称作距离池。在计算库中某个样本到查询向量的距离时，比如编码为 (124, 56, 132, 222) 这个样本到查询向量的距离时，我们分别到距离池中取各个子段对应的距离即可，比如编码为 124 这个子段，在第 1 个算出的 256 个距离里面把编号为 124 的那个距离取出来就可，所有子段对应的距离取出来后，将这些子段的距离求和相加，即得到该样本到查询样本间的非对称距离。所有距离算好后，排序后即得到我们最终想要的结果。

从上面这个过程可以很清楚地看出 PQ 乘积量化能够加速索引的原理：即将全样本的距离计算，转化为到子空间类中心的距离计算。比如上面所举的例子，原本 brute-force search 的方式计算距离的次数随样本数目 N 成线性增长，但是经过 PQ 编码后，对于耗时的距离计算，只要计算 4×256 次，几乎可以忽略此时间的消耗。另外，从上图也可以看出，对特征进行编码后，可以用一个相对比较短的编码来表示样本，自然对于内存的消耗要大大小于 brute-force search 的方式。

在某些特殊的场合，我们总是希望获得精确的距离，而不是近似的距离，并且我们总是喜欢获取向量间的余弦相似度（余弦相似度距离范围在 [-1,1] 之间，便于设置固定的阈值），针对这种场景，可以针对 PQ 乘积量化得到的前 top@K 做一个 brute-force search 的排序。

7.4.2 倒排乘积量化

倒排 PQ 乘积量化 (IVFPQ) 是 PQ 乘积量化的更进一步加速版。其加速的本质逃不开小白菜在最前面强调的是加速原理：brute-force 搜索的方式是在全空间进行搜索，为了加快查找的速度，几乎所有的 ANN 方法都是通过对全空间分割，将其分割成很多小的子空间，在搜索的时候，通过某种方式，快速锁定在某一（几）子空间，然后在



该（几个）子空间里做遍历。在上一小节可以看出，PQ乘积量化计算距离的时候，距离虽然已经预先算好了，但是对于每个样本到查询样本的距离，还是得老老实实挨个去求和相加计算距离。但是，实际上我们感兴趣的是那些跟查询样本相近的样本（小白菜称这样的区域为感兴趣区域），也就是说老老实实挨个相加其实做了很多的无用功，如果能够通过某种手段快速将全局遍历锁定为感兴趣区域，则可以舍去不必要的全局计算以及排序。倒排 PQ 乘积量化的“倒排”，正是这样一种思想的体现，在具体实施手段上，采用的是通过聚类的方式实现感兴趣区域的快速定位，在倒排 PQ 乘积量化中，聚类可以说应用得淋漓尽致。

倒排 PQ 乘积量化整个过程如下图所示：

在 PQ 乘积量化之前，增加了一个粗量化过程。具体地，先对 N 个训练样本采用 K-Means 进行聚类，这里聚类的数目一般设置得不应过大，一般设置为 1024 差不多，这种可以以比较快的速度完成聚类过程。得到了聚类中心后，针对每一个样本 x_i ，找到其距离最近的类中心 c_i 后，两者相减得到样本 x_i 的残差向量 $(x_i - c_i)$ ，后面剩下的过程，就是针对 $(x_i - c_i)$ 的 PQ 乘积量化过程，此过程不再赘述。

在查询的时候，通过相同的粗量化，可以快速定位到查询向量属于哪个 c_i （即在哪一个感兴趣区域），然后在该感兴趣区域按上面所述的 PQ 乘积量化距离计算方式计算距离。

7.5 拓展查询

拓展查询 (QE, Query Expansion): 指对返回的前 top@K 个结果，包括查询样本本身，对它们的特征求和取平均，再做一次查询，此过程称为拓展查询。

从上面的定义可以看出，拓展查询属于重排的一种方式。通过 Query Expansion，以达到提高检索召回率的目的。前面的博文 RANSAC 算法做直线拟合曾介绍过 RANSAC 的基本思想，放在词袋模型里（相应博文见图像检索：BoW 图像检索原理与实战），我们可以使用 RANSAC 方法或 Weak Geometry Consistency 方法做几何校正，进行重排以提高检索的精度。在这篇博文中，小白菜暂时抛开其他的重排方法，重点分析 Query Expansion 对图像检索精度的提升。

根据小白菜读图像检索论文获得的对 Query Expansion 的感知，做完 Query Expansion 能够获得百分之几的精度提升。为了证实 Query Expansion 对检索精度的改善，在过去一段时间里，小白菜在 Oxford Building 数据库上对其做了验证。下面是小白菜对 Query Expansion 的实验整理和总结。



参考文献

