

基于内容的图像检索系统*

严子涵 3180103551¹

¹ 计算机学院工业设计专业/竺院工高班

摘要

基于内容的图像检索主要分为特征提取和近邻查询两步。本文尝试通过三种方法进行特征提取，然后通过 Ball-Tree 算法 (KNN 的加速实现) 进行近邻查询。

本文先通过两种传统方法实现特征提取。在方法一中，本文从底层 (不依赖复杂的第三方库，仅用 Numpy 和 Pandas) 实现基于 SIFT 的特征提取和基于 VLAD 的特征聚合。在方法二中，本文尝试通过 Pytorch 建立 SIFTNET，对一张图片只提取一个 SIFT 描述符，即总共 128 个特征。也因此无需再使用 VLAD (K-means) 进行特征聚合，从而实现了一定程度上的对传统方法的创新。在方法三中，本文引入了深度学习方法 DenseNet 进行特征提取。

对比分析三种方法的在 6 折交叉验证测试下的准确率结果，发现 SIFT+VLAD 的方法准确率为 57%，而用 SIFTNET 提取单一 SIFT 描述子的方法准确率为 47%。DenseNet 的准确率可达到 63%。

1. 研究背景

基于内容的图像检索 (CBIR, Content Based Image Retrieval) 是一个基于图片特征的检索，主要包括了特征提取、索引构建、近邻搜索等技术。通常情况下，我们先用某一种特征提取方法，建立图像的特征矢量描述并存入图像特征库。当用户需要查询某一张图片的时候，我们首先用相同的特征提取方法得到该图像的查询向量，然后通过某种相似性

度量准则计算该图像的查询向量与图像特征库中各个特征矢量的形似性关系，最后再按相似性大小进行排序并顺序输出对应的图片，其整体流程如图一所示。而我认为其中较为重要的两部是特征提取和近邻查找，特征提取可以提高特征表达能力、降低特征存储空间和降低特征向量距离计算复杂度，近邻查找可以减少搜索范围和快速距离计算。

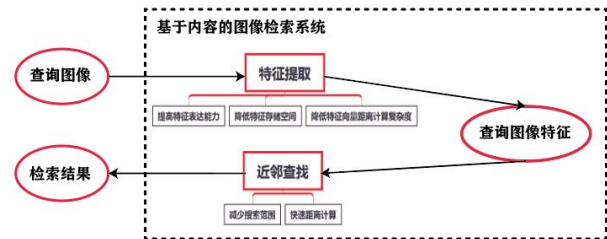


图 1: CBIR 系统

关于图像特征提取技术，在深度特征方法出现之前，即卷积神经网络出现之前，研究人员较为普遍的使用以 Harris Cornor、Scale-Invariant Feature Transform、GIST 和 HOG 为代表的手工特征，在整理中，我将这类特征归到传统特征。无论是传统特征还是深度特征，从表征内容上都可以划分为局部特征和全局特征。比如传统特征中的 SIFT 和深度特征中卷积层输出 feature map 就是局部特征，而传统特征中 GIST 和深度特征中 FC 的输出就可以理解为是全局特征。

在特征提取中，值得一提的一个环节是特征聚合 (Embedding)，这不仅是因为在使用局部特征表征图像时，需要将局部特征聚合成能够表征图像整体信息的全局特征；还因为特征聚合还可以将不

*本文为机器视觉和机器学习的期末课程作业

同数量的局部特征编码到同一长度，比如不同图像的 SIFT 特征个数是不同，使用聚合方法可以使得每张图像的特征表示长度相等。在对于 SIFT 特征的处理中，Michal Perdoch 等人实现了单个 SIFT 描述子的提取。该方法用卷积选择了一个最重要的描述点。

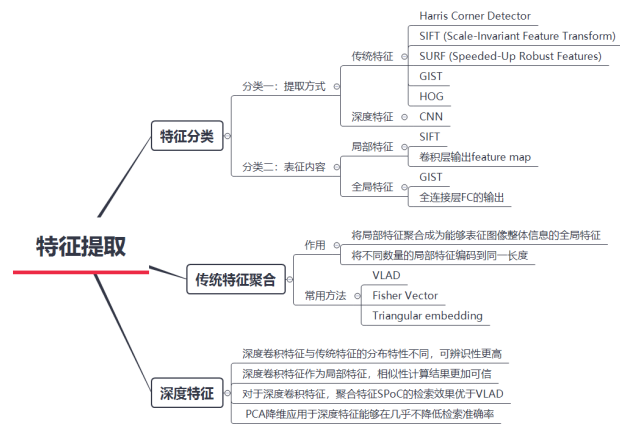


图 2: 特征提取

在传统图像的特征聚合中，BOW、VLAD 和 FV 比较典型。BOW 是把局部特征的个数累加到聚类中心上，VLAD 是把局部特征相对于聚类中心的偏差 (有正负) 累加到聚类中心上，而且是对最相邻的 k 个聚类中心都进行累加，FV 则采用混合高斯模型 (GMM) 构建码本，其实就是对高斯分布的变量求偏导。

图像经特征提取后的特征矢量是高维向量，在大规模高维向量数据集的检索需要的数据，对查找性能有一定的要求。在 CBIR 任务中，检索任务的最终目标是返回与查询图片最相似的图片数据集，其方法通常分为最近邻查找 (NN) 和近似最近邻 (ANN) 查找。在 NN 查找中，穷尽查找法的复杂度过高，所以相继产生了 KD-Tree 和 Ball-Tree 等优化方案。相比 NN，ANN 通过减小搜索空间的方式，提高查找效率，其代表算法有 hash function 仿射变换、Hamming Embedding 和 Inverted Multi-Index 等。

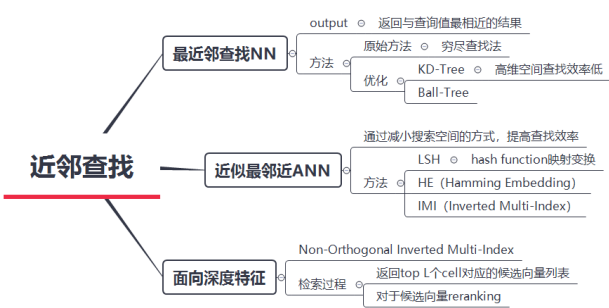


图 3: 近邻查找

2. 研究介绍

在本文中，将选取两种传统方法进行 CBIR 的实现，同时对比其效果。第一种方法较为常见，用 SIFT 做特征提取，用 VLAD (K-means) 做特征聚合，用 Ball-Tree 做近邻查找，这是一个比较经典的传统做法。本文将在不直接使用此三步的库函数，从原理上复现这三步。第二种方法，是一种尝试，考虑到 SIFT+VLAD 是为了得到相同维度的图像特征，因而选择提取单个 SIFT 描述子。因为使用了两种方法的特征提取，因而在结果分析中，将给出两种方法的对比。

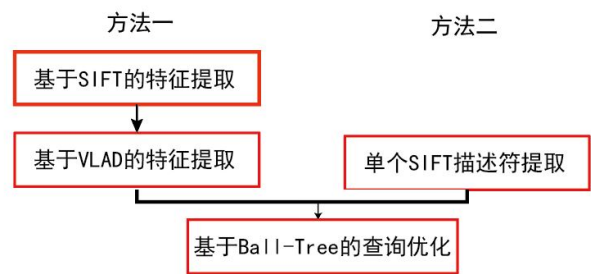


图 4: 本文使用的两种传统方法

随之，本文引入了深度学习算法——DenseNet。该算法架构如下图所示。

3. 特征提取方法一

3.1. 基于 SIFT 的特征提取

如图所示，SIFT 算法主要包括尺度空间极值检测、关键点的定位、指定方向和提取特征点描述符

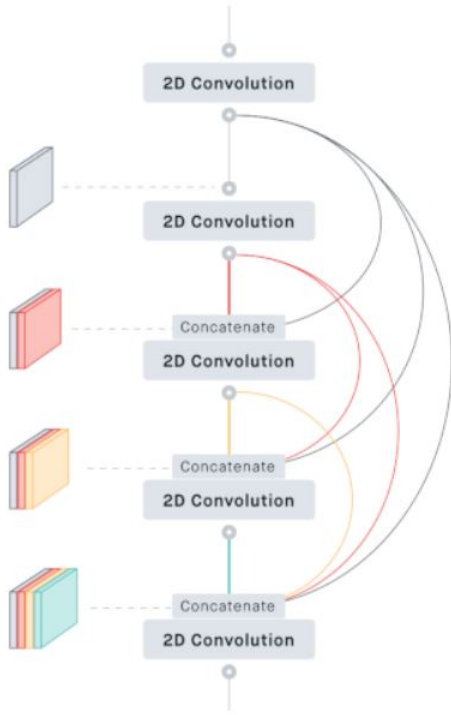


图 5: DenseNet 架构



图 6: SIFT 算法

四个关键步骤，其中关键点的定位又主要包括插值附近数据以获得准确的位置、丢弃低对比度的关键点和消除边缘相应三个环节。

在尺寸空间极值检测中，先用不同尺度的高斯滤波器对图像进行卷积，得到连续高斯模糊图像的差值。然后将关键点作为多个尺度上出现的高斯函数 (DoG) 的差值的最大值/最小值。值得一提的是，一个 DoG 是这样被给出的：

$$D(x, y, \sigma) = L(x, y, k_i \sigma) - L(x, y, k_j \sigma)$$

其中 $L(x, y, k\sigma)$ 是原始图像 $I(x, y)$ 与高斯模糊 $G(x, y, k\sigma)$ 在尺度 $k\sigma$ 处的卷积。即：

$$L(x, y, k\sigma) = G(x, y, k\sigma) * I(x, y)$$

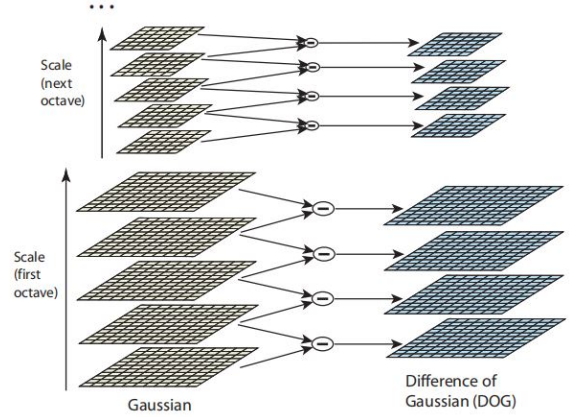


图 7: DOG

对于 SIFT 算法中的尺度空间极值检测，首先对图像进行不同尺度的高斯模糊卷积。关键点的检测则通过检测尺度空间极值尺度归一化的拉普拉斯算子进行。

在关键点定位时，首先，对于每个候选关键点，使用附近数据的插值来准确确定其位置。该插值是利用高斯尺度空间差分函数的二次泰勒展开式完成的，将 $D(x, y, \sigma)$ 作为原点。则泰勒展开式为：

$$D(x) = D + \frac{\partial D}{\partial x} x + \frac{1}{2} x^T \frac{\partial^2 D}{\partial x^2} x$$

为了丢弃对比度较低的关键点，在偏移量 \hat{X} 处计算二阶泰勒展开的值，当该值小于 0.03 时，丢弃候选关键点。其中偏移量的计算方法为：

$$\hat{X} = -\frac{\partial^2 D^{-1}}{\partial X^2} \frac{\partial D}{\partial X}$$

DoG 对边缘有更高的反应，所以边缘也需要被移除。在边缘消除中使用 2×2 Hessian 矩阵 (H) 来计算主曲率。具体矩阵如图所示：

经过以上步骤，就有了合法的关键点。然后就可以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。利用邻域像素的梯度方向的分布，为每个关键点赋予方向信息。因为使用的是梯度信息，使得后面的 local descriptor 是相对于这个方向生成的，所以有旋转不变性。对于

- Reject flats:
 - $|D(\hat{x})| < 0.03$
- Reject edges:

$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

Let α be the eigenvalue with larger magnitude and β the smaller.

$$\text{Tr}(\mathbf{H}) = D_{xx} + D_{yy} = \alpha + \beta,$$

$$\text{Det}(\mathbf{H}) = D_{xx}D_{yy} - (D_{xy})^2 = \alpha\beta.$$

Let $r = \alpha/\beta$.
So $\alpha = r\beta$

$$\frac{\text{Tr}(\mathbf{H})^2}{\text{Det}(\mathbf{H})} = \frac{(\alpha + \beta)^2}{\alpha\beta} = \frac{(r\beta + \beta)^2}{r\beta^2} = \frac{(r+1)^2}{r},$$

$(r+1)^2/r$ is at a min when the 2 eigenvalues are equal.

$\square \quad r < 10$

图 8: 边缘消除

每个已检测的关键点, 考虑该关键点 16×16 的邻域, 分别计算 gradient 以及 magnitude: $m(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2}$
 $\theta(x, y) = \tan^{-1}((L(x, y+1) - L(x, y-1)) / ((L(x+1, y) - L(x-1, y)))$

之后创建一个包含 36 个容器、覆盖 360 度的方向直方图。利用高斯加权对方向直方图进行平滑, 增加稳定性。填充直方图后, 将最高值对应的方向和与最高值 80% 以内的局部峰值对应的方向分配给关键点。在分配多个方向的情况下, 会创建一个额外的关键点, 对于每个额外的方向, 关键点的位置和比例与原始关键点相同。

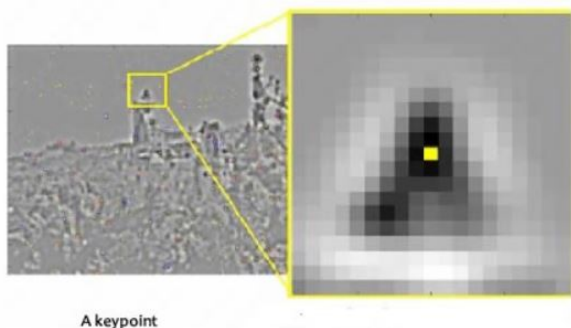


图 9: 在关键点位置附近取邻域

最后一步是基于每个特征点生成 local descriptor, 主要思路为基于每个特征点, 考虑特征点的邻域, 比如 16×16 的梯度信息, 将其分块, 计算梯度直方图, 生成具有独特性的 feature。

我们可以看到通过 SIFT 在本文所使用的数据集上的提取描述点的效果 (以数据集中使用的桥的

照片为例):

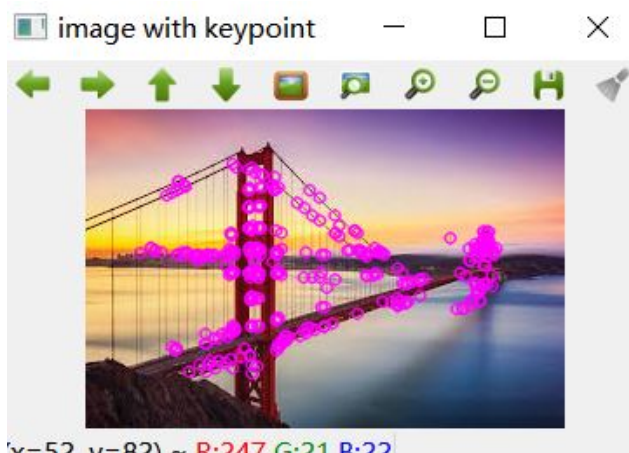


图 10: SIFT 描述点提取——原图

因为 SIFT 有较好的缩放不变性, 而且在方法二中将所有图像都拉成了正方形, 因而, 在这里也测试该图被拉伸成正方形后的 SIFT 描述点。

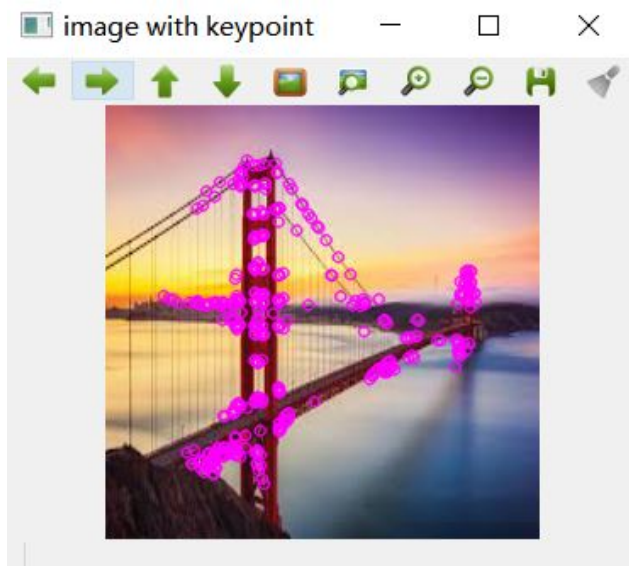


图 11: SIFT 描述点提取——缩放图

又因为 SIFT 有旋转缩放性, 所以旋转 45° 后再次测试, 结果如图所示。

3.2. 基于 VLAD 的特征聚合

在 VLAD 的实现中, 首先基于学习数据中的所有局部特征向量 X (d 维), 使用 K-means 聚类,

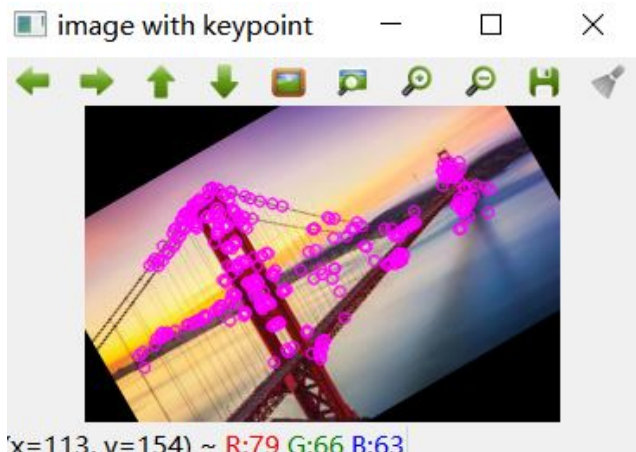


图 12: SIFT 描述点提取——旋转图

得到 k 个质心 $\{c_1, \dots, c_k\}$; 图像的 VLAD 表示是一个 $k \times d$ 维的向量, 向量元素 $v_{i,j}, i \in (0, k]$ 为质心索引, $j \in (0, d]$ 为局部特征向量中每个元素的索引, 对每个输入向量 x , 计算距离它最近的质心向量 c_i , 则 $v_{i,j}$ 为所有距离 c_i 最近的特征向量与 c_i 之间的差异在对应向量位置 j 上的累积 $v_{i,j} = \sum_{x \text{ such that } NN(x)=c_i} x_j - c_{i,j}$ 。最后, 对得到的 $v_{i,j}$ 向量使用 L2 范式进行归一化。质心数 k 通常取 16 256 即可得到较好的效果。

4. 特征提取方法二

4.1. 提取单个 SIFT 描述子

在介绍标准的 SIFT 做法时, 本文已详尽地给出了算法描述。其中在指定局部方向这一步时, 利用了关键点的尺度选择距离最近的高斯平滑图像 L , 使所有计算都以尺度不变的方式进行。现在用 3×1 和 1×3 的卷积核分别计算 $L(x+1, y) - L(x-1, y)$ 和 $L(x, y+1) - L(x, y-1)$, 分别记作 $G_x(x, y), G_y(x, y)$ 。

再计算每个像元的方向 $\theta(x, y)$ (如图 13b) 和梯度大小 $m(x, y)$ (如图 13c):

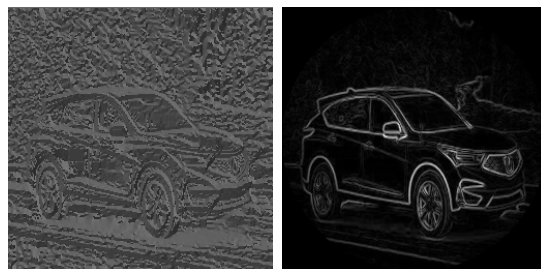
$$m(x, y) = \sqrt{G_x(x, y)^2 + G_y(x, y)^2}$$

$$\theta(x, y) = \arctan \frac{G_y(x, y)}{G_x(x, y)}$$

接下来的部分不同于传统的 SIFT 做法, 而是参考 Michal Perdoch 的实现, 将像元的梯度大小通

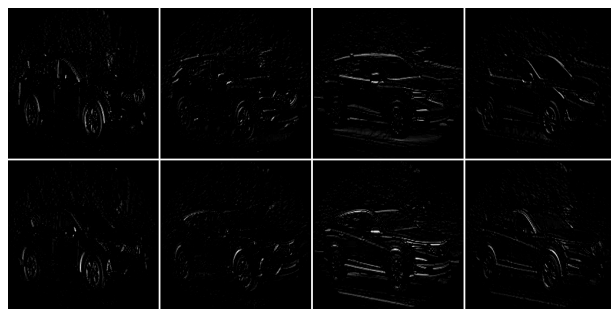


(a) 原始测试图像



(b) $\theta(x, y)$

(c) $m(x, y)$



(d) 朝向八个方向的梯度大小

图 13: 测试图像中每个像元的方向 ($\theta(x, y)$)、梯度大小 ($m(x, y)$) 和八个方向上的梯度大小

过一个圆形裁剪过的高斯窗口 ($\sigma^2 = 0.45R^2$, R 为图像宽/高度的一半, 如图 14) 进行加权, 避免边缘的无关内容对描述子产生影响。

由此分别获得朝向八个方向的梯度大小 (如图 13d), 并在整个图像按照 4×4 划分出的 16 个子区域内, 用 A 加权, 得到子区域内各方向的梯度大

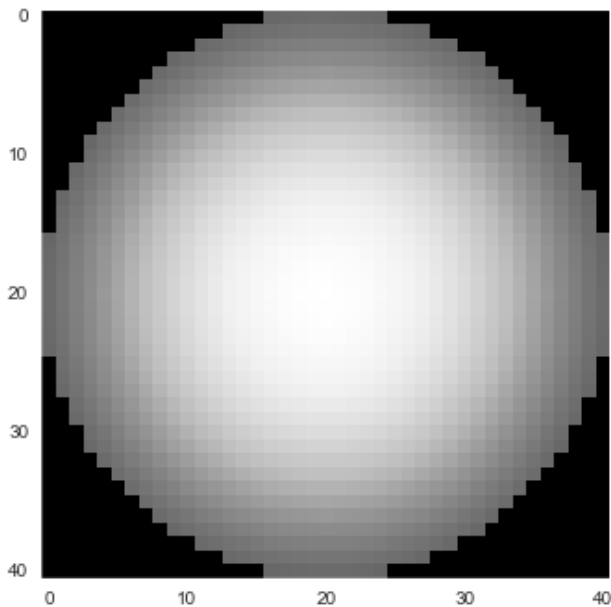


图 14: Michal Perdoch 的实现中的掩膜

小。其中, A 的形式为:

$$X = \begin{cases} \left[\underbrace{\frac{1}{n} \quad \frac{3}{n} \quad \frac{5}{n} \quad \dots \quad \frac{n-1}{n} \quad \frac{n-1}{n} \quad \dots \quad \frac{5}{n} \quad \frac{3}{n} \quad \frac{1}{n}}_{n \text{ elements}} \right], & \text{if } n \text{ is even} \\ \left[\underbrace{\frac{1}{n} \quad \frac{3}{n} \quad \frac{5}{n} \quad \dots \quad \frac{n}{n} \quad \dots \quad \frac{5}{n} \quad \frac{3}{n} \quad \frac{1}{n}}_{n \text{ elements}} \right], & \text{if } n \text{ is odd} \end{cases}$$

$$A = X^T X$$

将这 $8 \times 4 \times 4$ 个子区域的大小所组成的张量展平, 得到 128 维向量, 用二范数归一化, 将超过 0.2 的值设定为 0.2, 并再次用二范数归一化, 输出得到的 128 维向量作为描述子。

5. 特征提取方法三

5.1. 基于 DenseNet 的特征提取

因为卷积神经网络 (CNNs) 已经成为视觉对象识别的主要机器学习方法。所以在探讨完传统方法后, 本文也将探讨深度学习的方法。其中 VGG feature 19、Highway Networks 和 Residual Networks 都

是较为常用的 CNN 网络, 在此, 本文选择 DenseNet 进行特征提取。

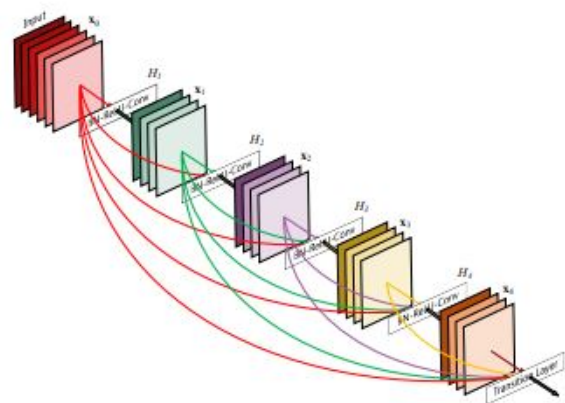


图 15: 一个 5-layer dense block

DenseNet 模型的基本思路与 ResNet 一致, 但是它建立的是前面所有层与后面层的密集连接 (dense connection), 其一大特色是通过特征在 channel 上的连接来实现特征重用 (feature reuse)。这些特点让 DenseNet 在参数和计算成本更少的情形下实现比 ResNet 更优的性能

如果用公式表示的话, 传统的网络在 l 层的输出为:

$$x_l = H_l(x_{l-1})$$

而对于 ResNet, 增加了来自上一层输入的 identity 函数:

$$x_l = H_l(x_{l-1}) + x_{l-1}$$

在 DenseNet 中, 会连接前面所有层作为输入:

$$x_l = H_l([x_0, x_1, \dots, x_{l-1}])$$

其中, 公式中的 $H_l(\cdot)$ 代表是非线性转化函数 (non-linear transformation), 它是一个组合操作, 其可能包括一系列的 BN (Batch Normalization), ReLU, Pooling 及 Conv 操作。当然这里 l 层与 $l-1$ 层之间可能包含多个卷积层。

DenseNet 的前向过程如图所示, 可以更直观地理解其密集连接方式。

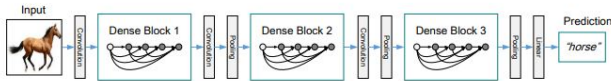


图 16: 由三个密集区块组成的 DenseNet

6. 基于 Ball-Tree 的近邻查找

Ball-Tree 在本质上是一种二叉树，其中每个节点都定义了一个 d 维超球，超球中包含了要搜索的点的子集。尽管球本身会相交，但每个点会根据到球中心的距离分配给分区中的一个或另一个球。对于任何一个给定测试点 t 和 t 所属于的根 $Root$ ，有：

$$D^B(t) = \begin{cases} \max(|t - B.pivot| - B.radius, D^{B.parent}), & \text{if } B \neq Root \\ \max(|t - B.pivot| - B.radius, 0), & \text{if } B = Root \end{cases}$$

其中， $D^B(t)$ 是球 B 中任意一点到 t 点的最小可能距离。简而言之，就是与 Kd-tree 基于欧氏距离的特性 $\|x - y\| \geq \|x_i - y_i\|$ 不同，balltree 基于更一般的距离特性 $\|x - y\| + \|y - z\| \geq \|x - z\|$ 基于球树的最邻近算法从根节点开始，按照深度优先顺序检查节点。在搜索中，会通过堆保存遇到该节点前的 k 个最近点的最大优先级队列 Q ，在每个节点 B 上，它可能会执行三种操作中的一种：

- 如果测试点 t 到当前节点 B 的距离大于 Q 中最远的点，则忽略 B ，返回 Q
- 如果 B 是一个叶节点，扫描 B 中枚举的每个点，并更新最近的 n
- 如果 B 是一个内部节点，递归调用 B 的两个子节点算法，搜先搜索中心更靠近 t 的子节点。在每次调用后更新队列，然后返回。

若以下图中搜索点 q 的半径为 r 内的最近邻来进行举例说明，我们可以认为在搜索过程中，我们从根节点 q 开始从上至下递归遍历每个可能包含最终近邻的子空间 p_i 。在目标空间 (q, r) 中，我们会遍历搜索所有被该超球体截断的子超球体内的子空间。最终，由于子超球体 a 与 b 被 q 所截，而对于 a 与 b 内的子空间， d, h, f 又被 q 所截，所以接下

来就会在 d, h, f 内进行线性搜索。在该过程中 c, e, g 等距离相对较远的子空间会被舍去，遍历结束后，会得出 $[x_4, x_7]$ 为最近邻。

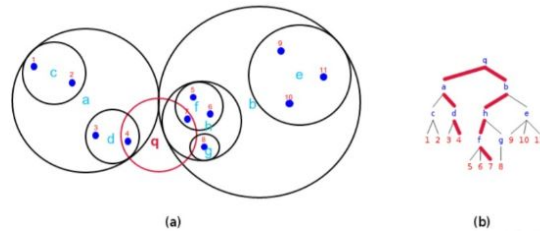


图 17: (a)Ball-tree partitions; (b)Corresponding search tree

7. 三种方法的结果比较

本文使用 6 折交叉验证的方法计算模型的准确率。也就是，先将数据集随机打乱顺序。然后，将数据集分为 6 组数量相等的数据集切片。在一次迭代中，5 组切片的数据进行训练，剩余的进行测试。这一过程被重复执行 6 次，而 6 次迭代后，6 组受试者均测试完毕。鉴于测试数据应独立于训练数据，这种交叉验证相当必要。

为了将 CBIR 搜索到的图片转化为准确率，选取了搜索到的相似程度最高的三张图片，统计这三张图片标签，选择出现频率最高的标签作为请求图片的标签预测结果。如果三张图片标签互不相同，选择与请求的图片相似程度最高的图片的标签作为请求图片的标签预测结果。再将预测结果与真实标签比较，判断预测正确与否。

使用 SIFT+VLAD 时，准确率最高达到了 57% ($\frac{17}{30}$)

单个 SIFT 描述子时，准确率最高达到了 47% ($\frac{14}{30}$)

使用 DenseNet 时，准确率最高达到 63% ($\frac{19}{30}$)

对于两种传统算法，本文尝试通过调整 k 折交叉验证的 k 来测试参数对结果的影响，可以制作两种描述子在不同超参数中的分类准确率图表：

从图表中可以发现，若 K 过小，可能影响准确率，但若 K 较大，对准确率的影响并不是非常大。

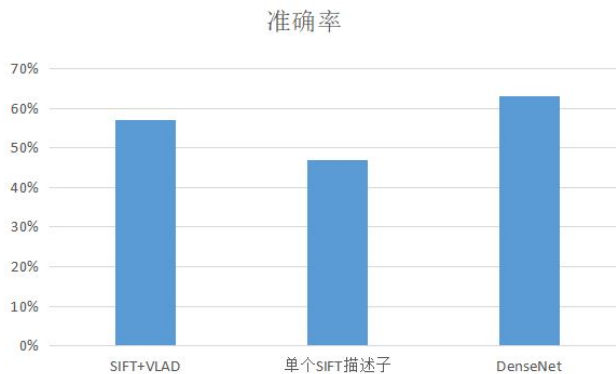


图 18: 三种方法的准确率对比

K-fold中不同K对应的算法准确率					
方法	K=5	K=6	K=10	K=15	K=30
SIFT+VLAD	40%	60%	50%	57%	57%
单个SIFT描述子	37%	47%	43%	47%	43%

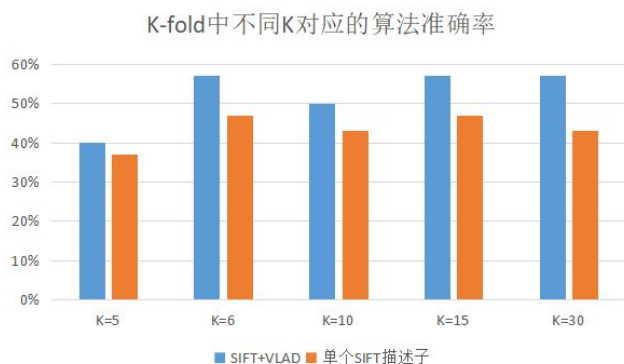


图 19: 图——K-fold 中不同 K 对应的算法准确率

而使用 SIFT+VLAD 的方法，准确率会明显高于使用单一 SIFT 描述子。

8. 代码和数据集说明

本文所使用的传统方法的数据集和代码都在 CODE 文件夹中

8.1. CODE 文件夹下文件组织

1. code

1. method1 (方法一的代码)
2. method2 (方法二的代码)
3. method3 (方法三的代码)

4. test (测试单个 py 文件的代码)

2. data (数据集)

1. bread (5 张面包图片)
2. bridge (5 张桥的图片)
3. car (5 张车的图片)
4. cat (5 张猫的图片)
5. dog (5 张狗的图片)
6. tree (5 张树的图片)

3. result

1. sift+vlad-result.txt (方法一的控制台直接输出结果)
2. sift-descriptor-result.txt (方法二的控制台直接输出结果)
3. densenet-result.txt (方法三的控制台直接输出结果)

8.2. Method1(SIFT+VLAD) 代码说明

1. 涉及文件

1. balltree.py
2. dataset.py
3. kmeans.py
4. main.py
5. vlad.py(使用基于 SIFT 的 VLAD 的 CBIS 实现)
6. pysift.py

2. 在本模块中，除非特殊标记，文件用途和名字相同

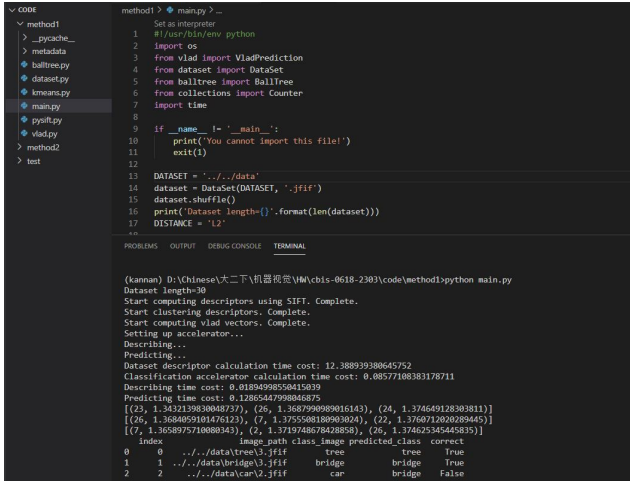
3. 测试会涉及的两个参数

1. 测试用的是 k 折 (k=6) 交叉验证，其中 k 在 main.py 的第 22 行可以修改。
2. 计算预测结果使用的 knn 的 k 在 23 行可以修改

4. 运行

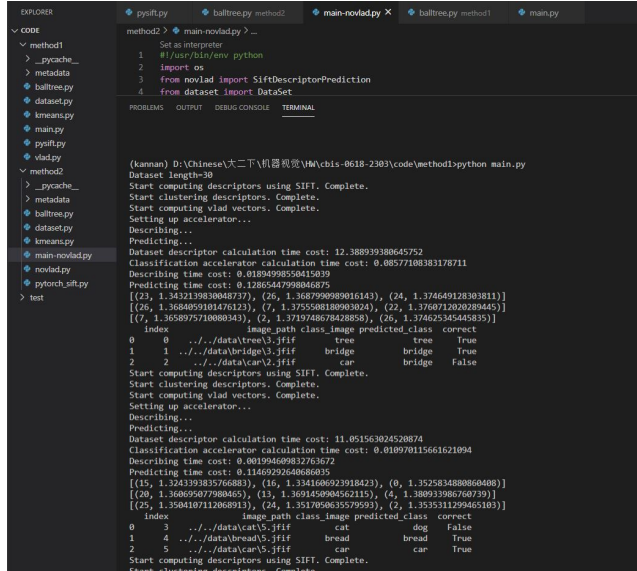
1. 直接运行主程序 main.py 即可，数据集的调用在程序中硬编码。

- 运行程序所需的 python 环境为: numpy; pandas; matplotlib; opencv



```
method1 > main.py > ...
1 #!/usr/bin/env python
2 import os
3 from vlad import VladPrediction
4 from dataset import Dataset
5 from balltree import Balltree
6 from collections import Counter
7 import time
8
9 if __name__ == '__main__':
10     print('You cannot import this file!')
11     exit(1)
12
13 DATASET = '../data'
14 dataset = Dataset(DATASET, '.jfif')
15 dataset.shuffle()
16 print('dataset length={}'.format(len(dataset)))
17 DISTANCE = 'L2'
18
19 PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
20
21 (kannan) D:\(Chinese\大二下\机器视觉\VM\cbis-0618-2303\code\method1\python main.py
22 Dataset length=30
23 Start computing descriptors using SIFT. Complete.
24 Start clustering descriptors. Complete.
25 Start computing vlad vectors. Complete.
26 Setting up accelerator...
27 Describing...
28 Predicting...
29 Dataset descriptor calculation time cost: 12.388939380645752
30 Classification accelerator calculation time cost: 0.08577108383178711
31 Describing time cost: 0.01894989550415839
32 Predicting time cost: 0.12865447998046875
33 [(23, 1.3432139830048737), (26, 1.3687990989016143), (24, 1.3746491283038111)]
34 [(26, 1.3686659101476123), (7, 1.375508180903024), (22, 1.3760712003039445)]
35 [(7, 1.3658975710808343), (2, 1.3719748678428858), (26, 1.374625345445835)]
36 index image_path class_image predicted_class correct
37 0 0 ../data/tree3.jfif tree tree True
38 1 1 ../data/bridge3.jfif bridge bridge True
39 2 2 ../data/car2.jfif car bridge False
```

图 20: 方法一运行截图



```
method2 > main-novlad.py > ...
1 Set as interpreter
2 #!/usr/bin/env python
3 import os
4 from vlad import SiftDescriptorPrediction
5 from dataset import DataSet
6
7 PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
8
9 (kannan) D:\(Chinese\大二下\机器视觉\VM\cbis-0618-2303\code\method2\python main.py
10 Dataset length=30
11 Start computing descriptors using SIFT. Complete.
12 Start clustering descriptors. Complete.
13 Start computing vlad vectors. Complete.
14 Setting up accelerator...
15 Describing...
16 Predicting...
17 Dataset descriptor calculation time cost: 12.388939380645752
18 Classification accelerator calculation time cost: 0.08577108383178711
19 Describing time cost: 0.01894989550415839
20 Predicting time cost: 0.12865447998046875
21 [(23, 1.3432139830048737), (26, 1.3687990989016143), (24, 1.3746491283038111)]
22 [(26, 1.3686659101476123), (7, 1.375508180903024), (22, 1.3760712003039445)]
23 [(7, 1.3658975710808343), (2, 1.3719748678428858), (26, 1.374625345445835)]
24 index image_path class_image predicted_class correct
25 0 0 ../data/tree3.jfif tree tree True
26 1 1 ../data/bridge3.jfif bridge bridge True
27 2 2 ../data/car2.jfif car bridge False
28 Start computing descriptors using SIFT. Complete.
29 Start clustering descriptors. Complete.
30 Start computing vlad vectors. Complete.
31 Setting up accelerator...
32 Describing...
33 Predicting...
34 Dataset descriptor calculation time cost: 11.051563024520874
35 Classification accelerator calculation time cost: 0.018910115661621094
36 Describing time cost: 0.001994609832763672
37 Predicting time cost: 0.11469292640868635
38 [(15, 1.3413938357656883), (16, 1.3341686923918423), (0, 1.3525034808060408)]
39 [(20, 1.36065077980465), (13, 1.369145090452115), (4, 1.380933986760739)]
40 [(25, 1.3584107112068913), (24, 1.3517050635579593), (2, 1.3535311299465103)]
41 index image_path class_image predicted_class correct
42 0 3 ../data/cat5.jfif cat dog False
43 1 4 ../data/bread5.jfif bread bread True
44 2 5 ../data/car5.jfif car car True
45 Start computing descriptors using SIFT. Complete.
46 Start clustering descriptors using SIFT. Complete.
```

图 21: 方法二运行截图

8.3. Method2(单个 SIFT 描述符) 代码说明

1. 涉及文件

- balltree.py
- dataset.py
- kmeans.py
- main-novlad.py
- novlad.py (使用单个 SIFT 描述子的 CBIS 实现)
- pytorch_sift.py (SIFT 描述子计算)

2. 测试

- 本部分说明同 Method1

3. 运行

- 直接运行主程序 main-novlad.py 即可, 数据集的调用在程序中硬编码。
- 运行程序所需的 python 环境为: numpy; pandas; matplotlib; opencv; pytorch (pytorch_sift.py 中使用)

8.4. Method3(DenseNet-161 预训练模型) 代码说明

1. 涉及文件

- balltree.py
- dataset.py
- kmeans.py
- main-densenet.py
- densenet.py (使用预训练的 DenseNet-161 模型作为描述子的 CBIS 实现)

2. 测试

- 本部分说明同 Method1

3. 运行

- 直接运行主程序 main-densenet.py 即可, 数据集的调用在程序中硬编码。
- 运行程序所需的 python 环境为: numpy; pandas; matplotlib; opencv; pytorch; torchvision

8.5. KMeans 的 scikit-learn 实现和手动实现的切换

若需要将 scikit-learn 的 KMeans 切换成手动实现的 KMeans, 需要将 vlad.py 和 novlad.py 的第五行和第六行下面的代码:

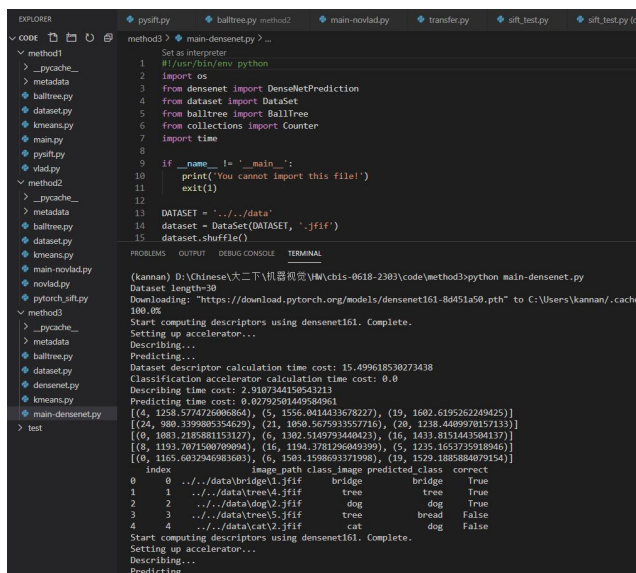


图 22: 方法三运行截图

```
#from kmeans import KMeans
from sklearn.cluster import KMeans
```

改成:

```
from kmeans import KMeans
```

8.6. 数据集说明

本次实验所使用的图片来自 Google 图片，一共为六类，其中 cat 和 dog 特征比较相近，其他图片 car, bread, tree 和 bridge 差异度较高。这里值得一提的是，在使用方法二的时候，在将数据图片送进 SIFTNET 前，需要将所有数据集图片转成同样大小的正方形格式。

9. 总结与展望

其一，在前面的工作中，我将较多的精力放在了特征提取上，但在图像检索的评价指标上考虑有所欠缺。后经查阅发现图像检索较多的使用 mAP 作为评价指标。mAP，即 AP 的平均值，需要先计算 AP，然后再对其进行平均。下图可以比较好的解释这个过程。

如果以后还有机会再次接触 CBIR，可能用 mAP 作为测评方式会是一个更好的选择。



本地磁盘 (D:) > Chinese > 大二下 > 机器视觉 > HW > cbis-0618-2303 > data > cat



本地磁盘 (D:) > Chinese > 大二下 > 机器视觉 > HW > cbis-0618-2303 > data > bread



本地磁盘 (D:) > Chinese > 大二下 > 机器视觉 > HW > cbis-0618-2303 > data > dog

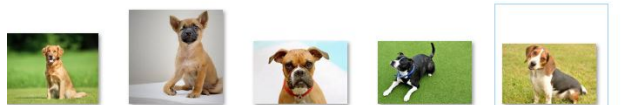


图 23: 部分数据集

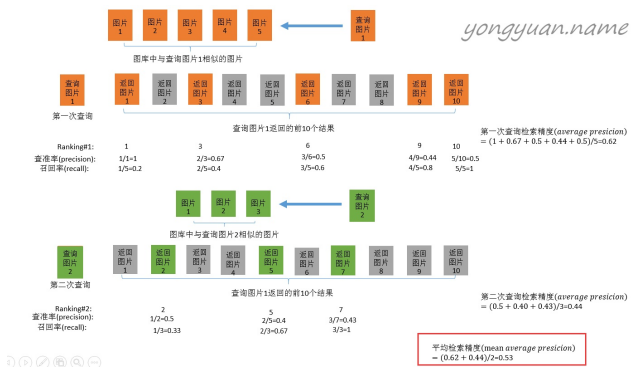


图 24: mAP

其二，深度模型的效果在本次测试中并不是特别的突出，这可能和返回特征的维数太高或是使用了 KNN 查询相关。针对这个问题，我也希望能在日后的学习中继续深入。

参考文献

[1]G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely Connected Convolutional Networks,"

arXiv:1608.06993 [cs], Jan. 2018, Accessed: Jun. 20, 2020.
[Online]. Available: <http://arxiv.org/abs/1608.06993>.

[2]L. David G., “Distinctive Image Features from Scale-Invariant Keypoints,” *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, Nov. 2004, doi: 10.1023/B:VISI.0000029664.99615.94.

[3]M. Perd’ och, O. Chum, and J. Matas, “Efficient Representation of Local Geometry for Large Scale Object Retrieval,” p. 8.

[4]D. Mishkin, F. Radenovic, and J. Matas, “Repeatability Is Not Enough: Learning Affine Regions via Discriminability,” arXiv:1711.06704 [cs], Aug. 2018, Accessed: Jun. 13, 2020. [Online]. Available: <http://arxiv.org/abs/1711.06704>.

[5]R. Arandjelović and A. Zisserman, “Three things everyone should know to improve object retrieval,” p. 39.

[6]H. Jégou, F. Perronnin, M. Douze, J. Sánchez, P. Pérez, and C. Schmid, “Aggregating Local Image Descriptors into Compact Codes,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 9, pp. 1704–1716, Sep. 2012, doi: 10.1109/TPAMI.2011.235.