太原科技大學
TAIYUAN UNIVERSITY OF SCIENCE &TECHNOLOGY

学士学位论文

基于多种特征和标签相关度量的图像完备标注方法

设计人：　李先淼

指导教师：张素兰

所属系部：计算机科学与技术学院

专业班级：计算机科学与技术 182002 班

学号：　201820010223

2022 年 6 月

太原科技大学毕业设计（论文）任务书

学院: 计算机科学与技术学院

学生姓名	李先淼	学号	201820010223
专业班级	计算机 182002 班	同组人	无
任务下发时间	2022 年 3 月 7 日	任务完成时间	2022 年 6 月 5 日
设计（论文）题目	基于多种特征和标签相关度量的图像完备标注方法		
设计目的要求	随着网络图像数据资源的增加，图像标签完备标注成为目前图像自动标注的一项重要研究内容。图像有效的视觉特征提取和标签相关分析是提高图像完备标注精度的重要手段。研究并实现一种基于多种特征和标签相关度量的图像完备标注方法，要求能够实现基于待标注图像多种特征和初始语义的近邻检索、视觉内容和语义相关度量以及标签完备标注等功能。翻译一篇相关的英文资料；书写毕业论文；培养基于科学原理并采用科学方法对复杂计算机工程问题进行分析和研究的能力。		
设计主要内容	1.学习 SIFT 特征提取算法、学习 CNN 特征提取算法； 2.学习 VLAD 聚合算法； 3.学习 PCA 降维算法； 4.原型系统体系结构设计； 5.系统总体设计； 6.系统详细设计与测试； 7.相关英文资料的翻译。		
设计提交资料	1.毕业论文 1 份； 2.源码 1 份。		
学生签名		指导教师签名	
系主任签名		主管院长签名	

说明：一式两份，一份装订入学生毕业设计（论文）内，一份交学院（直属系）

摘要

随着互联网普及程度的增加，网络中存在着越来越多的图像。许多应用软件在检索图像时会根据给定的标签词进行查找，然而由于用户标注图像的随意性和主观性等因素使得图像本身所带有的标签并不完备，许多图像往往只有较少的标签词，不足以完整地描述图像内容，从而影响图像检索效果。图像标签完备标注主要依据待完备图像本身与其近邻图像的相似程度，实现自动补全待完备图像的缺失标签，该工作因网络图像数量的不断增加正受到越来越多学者的关注。

本文主要研究并实现了一种基于多种特征和标签相关度量的图像完备标注方法。图像完备标注的一般方法是先通过比较待完备图像和候选图像的相似度找出待完备图像的近邻图像，再通过比较待完备图像和候选标签的相关度选择出缺失标签补全待完备图像。因此，本文首先设计了一种通过综合考虑图像的多种视觉特征和初始标签语义来查找近邻图像的近邻检索方法，其中在计算语义相似度时不仅考虑了标签之间的直接相关性，并且通过构建标签的间接相关集来度量标签之间的间接相关性；其次，以近邻图像的标签作为候选标签，给出了一种从视觉相关度和语义相关度两方面度量待完备图像和候选标签相关性的完备标注方法，实现从候选标签中选择出待完备图像的缺失标签，进而实现图像的完备标注。

基于上述方法，利用 Python 语言和 QT Designer 界面设计工具实现了一个图像标签完备系统，并详细展示了系统的体系结构及其各模块功能。

关键词：图像完备标注；SIFT；CNN；标签相关度量

A method for complete image annotation based on multiple features and label correlation metrics

author: Li Xianmiao tutor: Zhang Sulan

Abstract

With the increasing popularity of the Internet, there are more and more images on the Web. Many applications retrieve images based on the given tag words, however, due to the arbitrary and subjective nature of the user labeling images, the images themselves are not fully labeled, and many images often have only a small number of tag words, which is not enough to fully describe the image content, thus affecting the image retrieval effect. The tag-complete labeling of images is mainly based on the similarity between the image to be labeled and its near-neighboring images to achieve automatic completion of the missing tags of the image to be completed, and this work is receiving more and more attention from scholars due to the increasing number of images on the web.

In this paper, we study and implement an image completion annotation method based on multiple features and label correlation metrics. The general method of image completion annotation is to first find the nearest neighbor image of the image to be completed by comparing the similarity of the image to be completed and the candidate image, and then select the missing label to complete the image to be completed by comparing the correlation of the image to be completed and the candidate label. In this paper, we firstly design a nearest neighbor retrieval method to find the nearest neighbor image by considering various visual features of the image and the semantics of the initial labels, in which not only the direct correlation between the labels is considered when calculating the semantic similarity, but also the indirect correlation between the labels is measured by constructing the indirect correlation set of the labels; secondly, we use the labels of the nearest neighbor image as the candidate labels, and give a method to find the nearest neighbor image from both visual correlation and semantic correlation. Secondly, this paper uses the labels of the nearest neighboring images as candidate labels, and presents a complete annotation method to measure the correlation between the image to be completed and

the candidate labels in terms of both visual correlation and semantic correlation, so as to select the missing labels of the image to be completed from the candidate labels and realize the complete annotation of the image.

Based on the above algorithm, this paper implements an image tagging completion system using Python language and QT Designer interface design tool, and shows the architecture of the system and the functions of its modules in detail.

Key Words: Complete image annotation; SIFT; CNN; Label correlation metrics

目录

第一章 绪论

1.1 课题研究背景及意义

随着互联网技术的发展，网络中存在着越来越多的图像；与此同时，如今的很多应用软件都有检索图像的需求，搜寻存在于个人电脑或者共享网络上的图像。应用程序在搜索图像的时候往往是根据指定的图像标签来检索，由相关的搜索算法经过处理，把带有指定标签的图像返回给用户。因此，图像本身所带有标签的完备性、准确性则在一定程度上决定了检索效果。然而，由于用户在存储及共享图像时对图像的标注带有一定主观性和随意性，使得个人电脑和网络中的大量图像标签不完备，并不足以完整、准确的描述图像内容，因而这些标签不完备的图像在被动地参与检索时能够匹配到的关键词更少，导致图像检索效果不好。因此，研究一种有效的标签完备方法用于尽可能准确地补全待完备图像缺失标签具有重要的现实意义。

标签补全的一般方法通常是先找到待补全图像的近邻图像集合，然后从所有近邻图像的标签中选择与待补全图像高度相关的标签来补全待完备图像。显然，标签完备的效果在很大程度上受到近邻图像质量的影响。而当前的大多数标签完备方法在查找近邻图像的时候往往只根据待完备图像单一的视觉特征来计算与近邻图像的相似度，不可避免地受到待完备图像噪声的影响；另外，待完备图像的初始标签在一定程度上可以描述图像的部分内容，而这些标签完备方法并没有考虑待完备图像的初始标签，也造成了一定的精度损失。因此，研究一种利用图像多种视觉特征并结合待完备图像初始标签语义来查找近邻图像的方法是有价值的。

标签完备标注方法通常由三步组成：特征提取、近邻检索和完备标注。特征提取主要提取图片的视觉特征；近邻检索则是查找待完备图像的近邻图像；完备标注则是从候选标签中选择出与待完备图像相关度高的标签作为缺失标签补全待完备图像。在近邻检索和完备标注过程中若考虑待完备图像的初始标签，则需要进行标签与标签之间相关度的度量。标签之间的相关性常用两个标签的共现频率来度量，但共现频率仅可以在一定程度上衡量两个标签的直接相关性，而忽略了标签之间的间接相关性，不足以充分表征标签之间的相关性。因此，研究一种更全面的标签相关性度量方法更加充分地量化标签

之间相关性具有一定的研究价值。

总之，通过利用待完备图像的多种视觉特征和初始标签语义查找近邻图像，并在考虑标签相关性时引入更全面的标签相关性度量方法提高待完备图像的标注精度，使图像标签趋于完备，从而提高图像检索效果的方法具有重要的研究意义和价值。

1.2 国内外研究现状

近年来随着图像检索需求的普遍化以及网络上图像数量不断增多，标签完备方法成为越来越多的学者研究的重点内容。迄今为止，已有很多人在这个领域进行了研究，许多种方法和模型被提出用于解决图像的标签完备标注问题[1,2]，这些模型及方法所考虑的角度各不相同，但其主要由两大模块组成：图像特征提取和候选标签获取，通过两模块的结合进行候选标签生成和缺失标签选择[3]。这些方法具体讲又可以划分为五种类型[4]，分别是通过标签矩阵完备、线性子空间重构、子空间聚类、低秩矩阵分解和近邻图像检索来补全缺失标签。

其中，通过标签矩阵完备的标注方法的主要思想对图片集的标签矩阵进行补全处理使其达到完备，Wu L 等人提出了一种标签矩阵完备模型，即 TMC 模型[5]，该模型通过调整标签的相似性和基于视觉内容的相似性之间的差异，使其达到最小来更新和优化标签矩阵，进而实现图像的标签完备。

通过线性空间重构标注方法的主要思想是把图像的视觉特征和标签分别视作一个子空间，然后对子空间进行重建以实现标签完备。Lin 等人提出了一种图像特定和标签特定线性稀疏重建模型，即 LSR 模型[6]，该模型基于图像视觉和语义特征的相似性重建图像视觉特征的子空间、基于标签共现重建标签的子空间，最后通过多种重建空间的融合实现对图像缺失标签的补全。

通过子空间聚类实现完备标注的标注方法的典型代表是 Hou 等人提出的子空间聚类和矩阵完成模型，即 SCMC 模型[7]。该模型首先在子空间聚类框架中进行标记完备，然后利用矩阵完备方法来完备标签矩阵。

通过低秩矩阵分解的标注方法更多的关注噪声带来的影响。具体的例子是 Li 等人提出的局部敏感低秩重建模型，即 LSLR 模型[8]，该模型利用低秩稀疏来处理噪声标签，为获取更全面的相关性，加入了一些局部线性模型来近似一个非线性模型，并以减少噪声标签，保证标签传播准确进行。

通过近邻图像检索来补全缺失标签的方法的典型例子是 Guillaumin M 等人提出的利用近邻图像的标签进行完备的算法[9]，该方法首先提取图像的视觉特征，然后根据视觉特征来计算查找近邻图像，再从近邻图像的标签中选择出和待完备图像有相关度的标签补全给待完备图像。

1.3 论文主要研究内容

本文研究了一种基于多种特征和图像相关度量的图像完备标注方法。首先，提取图像的多种视觉特征向量，再分别利用特征向量求图像之间的视觉相似度，然后将多种视觉特征相似度加权求和作为综合视觉相似度的度量[10]；其次，利用两幅图片所含标签的平均语义相关度作为图片之间语义相似度的度量，其中从直接相关和间接相关两个角度考虑标签之间的相关性。最后，综合考虑两幅图片之间的视觉相似度和语义相似度，并以此来查找待完备图像的近邻图像。

接着，对近邻图像进行处理。首先把近邻图像所带有的标签作为候选标签，然后计算候选标签与待完备图像的相关性，选择相关性高的标签补全待完备图像。在度量候选标签与待完备图像相关度时，需要从标签与标签、图像与图像两个方面对标签与图像的关系进行模态转换，其中在计算标签与标签相关度时，从直接相关和间接相关两个角度比较相关性。最后综合考虑两个方面的相关度得到候选标签与待完备图像的最终相关度并基于上述算法在 Corel5k 数据集上进行实验，验证了算法的有效性。

此外，本文还利用 Python 语言和 Qt Designer 设计工具实现了一个基于图像多种特征和标签相关度量的图像完备标注系统，该系统由一个主线程和三个子线程构成，由主线程控制整个系统的启动、关闭以及其它底层功能，而三个子线程分别控制特征提取、近邻检索和完备标注功能。

1.4 论文组织安排

本文研究了一种基于多种特征和标签相关度量的图像完备标注方法。首先提取图像的多种视觉特征并计算基于多种视觉特征下的视觉相似度，此外还结合待完备图像的初始标签，即语义相似度，以此查找近邻图像；然后，将近邻图像的标签作为待完备图像的候选标签，并从中选出与待完备图像具有高相关性的标签补全图像。本文结构如下：

第一章：本章简要叙述了标签完备的研究背景和研究意义，介绍了标签完备的国内

外研究现状，并阐述了本文研究的主要内容。

第二章：本章介绍了研究过程中涉及到的相关算法理论，包括特征提取、视觉相似度度量、语义相似度度量等，以及实现算法所采用的技术工具。

第三章：本章先介绍了对数据集预处理的方法，然后介绍了提取图像视觉特征的具体过程和检索待完备图像的近邻图像的方法，并分析了相关参数的设定。

第四章：在第三章的基础上，本章先介绍了候选标签的提取方法，然后介绍了完备标注的具体过程，并分析了相关参数的设定。

第五章：在第三章和第四章所提出的图像完备标注方法的基础上，利用 Python 语言及 QT Designer 界面设计工具实现了一个可视化的图像标签完备系统，并展示了系统的体系结构和各个模块的功能。

结束语，基于上述工作，总结了本文所提出的基于多种特征和标签相关度量的图像完备标注方法，分析了其中可能存在的问题。

第二章 相关理论和技术工具

2.1 相关理论

2.1.1 SIFT 特征

尺度不变特征变换即 SIFT(Scale-invariant feature-transform)是一种处理计算机视觉问题的算法。它用来侦测与描述影像中的局部性特征，通过在空间尺度中寻找极值点，提取出其位置、尺度和旋转不变量，此算法由 David Lowe 在 1999 年所发表[11]，2004 年完善总结。SIFT 特征主要是采集图像的一些区域兴趣点，与图像本身的尺寸、旋转等无关，受到光照等其它的噪声所带来的影响也非常小。SIFT 算法的具体过程为：

（1）构建高斯金字塔和高斯差分金字塔

图像高斯金字塔实际上是一种图像的尺度空间，如图 2.1 所示。尺度的概念是用来模拟观察者所看到物体的远近程度和模糊程度，其中高斯金字塔通过采样法实现远近程度的模拟，例如对一幅图像每隔一个像素点采样一次，那么最终得到的图像行和列各位原图像的一半，这便是下采样的一种；此外高斯金字塔通过高斯核对图像进行平滑处理，David Lowe 已经证实了高斯卷积核是实现尺度变换的唯一线性核。

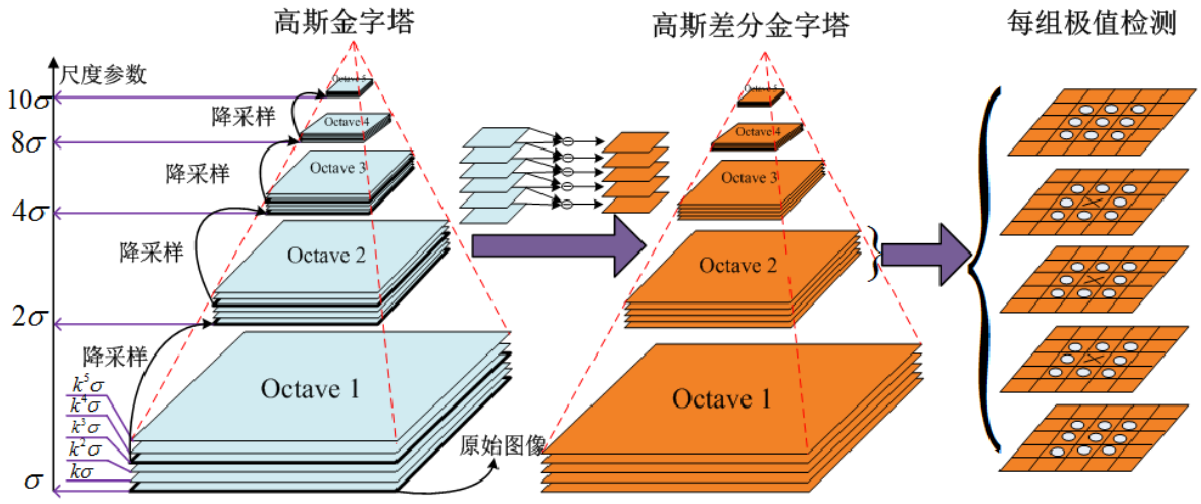

图 2.1 高斯金字塔及高斯差分金字塔

高斯金字塔含有多组，每组含有多层，层与层之间以及组与组之间均是通过下采样得到。高斯金字塔组数的计算方法如公式(2.1)所示。

$$O = \log_2\left[\min(m,n)\right] - 3 \tag{2.1}$$

式(2.1)中，m、n 分别是原始图像像素点的行高和列宽。另外，高斯金字塔每组的层数计算方法如公式(2.2)所示。

$$S = n + 3 \tag{2.2}$$

式(2.2)中，n 为在高斯差分金字塔每组中须提取特征的图像数，S 则为高斯金字塔每组的层数。在利用高斯核实现对图像模糊的过程中，高斯模糊系数的计算方法如公式(2.3)所示。

$$\sigma(h,r) = \sigma_0 \cdot 2^{h+\frac{r}{n}} \tag{2.3}$$

式(2.3)中，h 为组索引号，其中 $h \in \{0,1,\cdots,O\}$，r 为每组的层索引，n 为在高斯差分金字塔每组中需要提取特征的图像数，σ_0 为高斯模糊初始值，一般取 1.6 为恰当。由高斯模糊系数的计算公式可以得出，每一组内相邻层之间的高斯模糊系数相差 $2^{1/n}$；第 0 组第 0 层、第 1 组第 0 层、第 2 组第 0 层、…的高斯模糊系数应为 $\sigma_0, 2\sigma_0, 4\sigma_0, \cdots$；在得到高斯模糊系数以及组数和层数之后，高斯函数的计算方法如公式(2.4)所示。

$$G(x,y,\sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.4}$$

式(2.4)中，x、y 为图像各像素点的位置，σ 为高斯模糊系数。于是，高斯金字塔中的每幅图像的表示方式如公式(2.5)所示。

$$L(x,y,\sigma) = G(x,y,\sigma) \otimes I(x,y) \tag{2.5}$$

式(2.5)中，$I(x,y)$ 表示图像，$G(x,y,\sigma)$ 表示高斯函数，则每幅图像则可以表示为二者的卷积运算。

在创建好高斯金字塔后，对各组内的相邻层之间求下层和上层的差值即可得到高斯差分金字塔。Mikolajczyk 通过实验研究，得出高斯拉普拉斯函数在尺度归一化之后所计算出的极大值和极小值比对应的提取函数提的取结果产生的图像特征更具有稳定性。然而，Lindeberg 在此之前已经通过实验研究得出这一结果与高斯差分函数非常相近，因此，在 SIFT 算法中，使用高斯差分金字塔即 DOG 尺度空间进行后续的极值点检测、定位、赋值等可以生成更加稳定的图像特征。高斯差分金字塔的建立过程如图 2.2 所示。

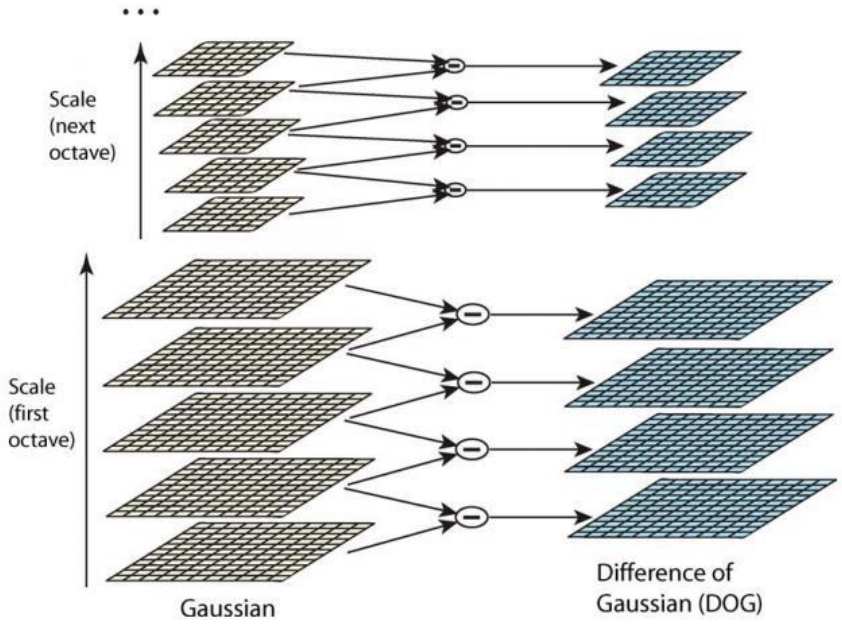

图 2.2 建立高斯差分金字塔

（2）关键点搜索和定位

在高斯差分金字塔中寻找极值点，除了要考虑二维平面上 x、y 方向的点，还要考虑垂直方向上即σ方向上的点。为此，判断一个像素点是否为极值点，需要和其上下左右共 26 个点进行比较，看其是否比它的同层或上下层的像素点大或者小。具体检测过程中需要检测的点如图 2.3 所示。

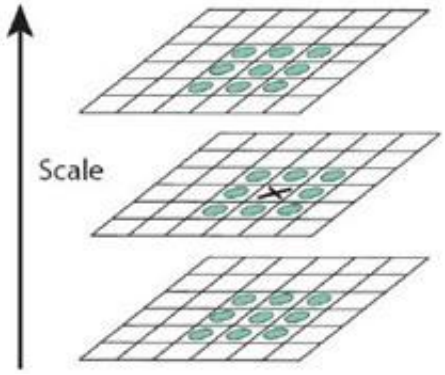

图 2.3 极值点检测

由于像素是离散的，尺度空间也是不连续的，因此有可能找到的极值点是真正极值点旁边的点，如图 2.4 所示。因此为了找到更高亚像素位置精度的极值点，需要利用曲线拟合方法来处理尺度空间高斯差分函数使得结果达到预期，即利用高斯差分函数在尺度空间的泰勒展开式。高斯差分函数的具体泰勒展开式如公式(2.6)所示。

$$D(X) = D + \frac{\partial D^T}{\partial X} X + \frac{1}{2} X^T \frac{\partial^2 D}{\partial X^2} X \tag{2.6}$$

式(2.6)中，$X = (x, y, \sigma)^T$，D 为高斯差分函数。通过将高斯差分函数在尺度空间做泰勒展开并用展开式拟合可以在一定程度上提高极值点检测的准确度。此外，还需要舍去一些低对比度的点和由边缘效应所带来的不稳定的点。

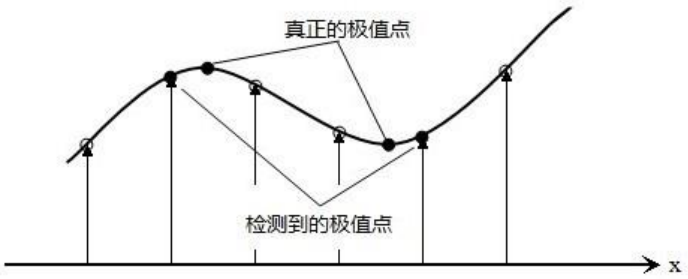

图 2.4 不准确的检测示例

（3）方向赋值

要实现旋转不变性，最有效的方法是根据图像的区域特征，为每个特征点选择并分配一个参考方向。 在此过程中，利用图像梯度法可以得到局部结构的稳定方向。

针对在高斯差分金字塔中已经找到的特征点，计算特征点所在高斯金字塔图像 3σ 领域窗口内像素分布特点。然后使用直方图统计这一区域内各个像素的梯度以及方向，梯度直方图从 0 度到 360 度按照步长为 10 度依次划分为 36 个段长，其中直方图的最高点方向即代表了特征点的主方向。

（4）关键点描述子的生成

在上述过程，已经找到了关键点并确定了其方向，后需要使用数学方法对特征点进行数学层面的特征描述，也就是构建特征点的描述子。

首先将特征点周围的空间划分为 4×4 个子区域，在每个子区域的 8 个方向上分别计算直方图以便获得每个方向的梯度幅值，这样处理的结果是总共可以组成 128 维描述向量，则得到了该特征点的 SIFT 描述子；对每个关键点重复上述操作，即可生成各个关键点的描述子。

2.1.2 CNN 特征

卷积神经网络即 CNN(Convolutional Neural Network) 是一种主流的、特征提取效果

非常好的前馈神经网络，它所构建和训练出的人工神经元可以对一部分自身范围内的周围神经元做出响应，在图像处理方面具有非常优异的性能。卷积神经网络的特点主要有参数共享、局部感知以及多核处理，其中参数共享能够使得 CNN 最大程度的减少运算量，局部感知能够使得提取到的特征对局部描述更精细，另外多个卷积核对图像进行卷积，能够从多个角度提取图像特征。

卷积神经网络在结构上主要由输入层、卷积层、激励层、池化层和全连接层等构成。CNN 主要结构如图 2.5 所示。

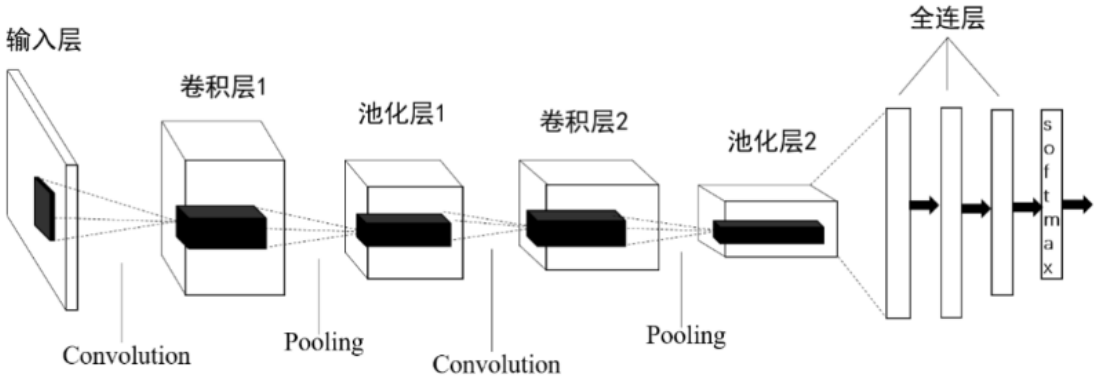

图 2.5 卷积神经网络基本结构

其中输入层用于接收输入数据；卷积层利用卷积核对输入数据进行卷积，通过该操作可以实现局部的特征提取和特征映射；而在激励层，由于卷积是一个线性操作，所以激励层在卷积输出上添加了一个非线性映射以增强网络的表达能力和抗干扰能力；在池化层主要对卷积结果进行降采样，即稀疏处理 Feature Map 以减少数据运算量加快特征提取的过程；全连接层位于神经网络的末端，主要起分类作用，可以将多层卷积过程中提取的 Feature Map 映射到一个新的空间，实现分类。在利用 CNN 提取图像视觉特征时，可以将 CNN 视为一个特征提取器，CNN 通过多层卷积层和池化操作，提取出可以描述图像全局特征的特征图。CNN 特征提取的具体过程如下：

（1）卷积层使用卷积核与原始图像进行卷积

卷积层利用训练好的卷积核权重，与图像矩阵进行卷积运算，生成特征图(Feature Map)，提取出图像的视觉特征。卷积公式如(2.7)所示。

$$Map_{i,j} = f\left(\sum_{m=0}^{M}\sum_{n=0}^{N}\omega_{m,n} \bullet I_{i+m,j+n} + w_b\right) \tag{2.7}$$

式(2.7)中，M、N 是卷积核大小，$\omega_{m,n}$ 是卷积核(m,n)位置处的权重，$I_{i+m,j+n}$ 是图像矩阵对应位置的数值，w_b 为偏置项，f 为激活函数，$Map_{i,j}$ 为特征图对应位置的数值。

（2）池化层对卷积层输出的特征图稀疏处理

池化层(Pooling)使用下采样的方法对特征图进行稀疏处理，通过损失部分特征信息来达到和计算性能的妥协，减少运算量加快计算速度。例如一个 128×128 的特征图使用 2×2 的过滤器池化后，特征图的尺寸减小为 64×64，如图 2.6 所示，极大减少了运算量。

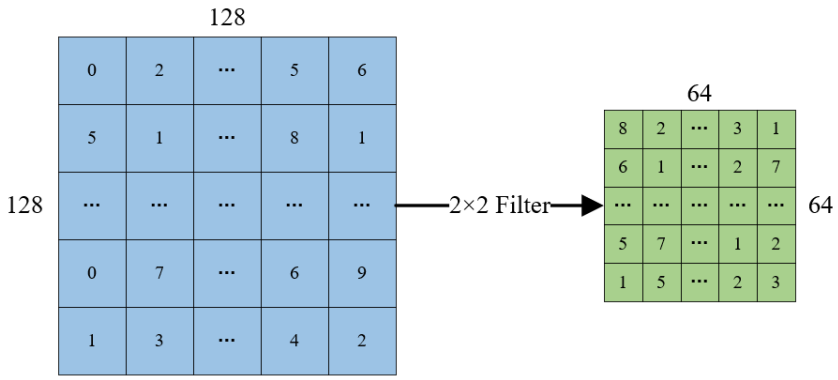

图 2.6 池化操作

（3）多层卷积和池化

单层卷积操作提取了图像的局部特征，多层卷积使提取的特征更加全局化。在进入全连接层之前，已经经过了多层的卷积和池化，此时网络末端的特征图是通过反复卷积和池化操作提取的，能够描述一幅图像的特征。

2.1.3 欧氏距离和 Google 距离

（1）欧式距离

欧几里得距离(Euclidean Distance)也称欧式距离，是一种思想简洁、计算简单、应用广泛且被普遍接受的距离度量方法，指在 n 维空间中两个点之间的直接距离，或者两个向量差值的长度。在计算机视觉领域，欧氏距离也常常被用来精确的量化两幅图像之间的相似度，例如，两个等长的特征向量其之间的欧式距离可以抽象表示两向量差值的模长，如公式(2.8)所示。

$$Distance = \| X - Y \| \tag{2.8}$$

式(2.8)中，X 和 Y 为两个等长的一维向量，$\| X - Y \|$ 为两向量差值的模长，Distance 为二者的欧式距离，是一个确定的数值。

（2）Google 距离

Google 距离是一种语义相似性度量方法，由计算机自然语言处理专家 Rudi L.Cilibrasi 和 Paul M.B.Vitanyi 提出[10]。这个方法最初适用于搜索引擎，是由谷歌搜索引擎对一组给定的关键词返回的点击数得出的，在自然语言意义上所表达的含义越相近的关键词 Google 距离越小，而含义相悖或者联系较小的关键词则往往距离较大。例如，两个搜索词 x 和 y 的 Google 距离计算方法如公式(2.9)所示。

$$\mathrm{NGD}(x,y)=\frac{\max\{\log_{10} f(x),\log_{10} f(y)\}-\log_{10} f(x,y)}{\log M-\min\{\log_{10} f(x),\log_{10} f(y)\}} \tag{2.9}$$

式(2.9)中，f(x)和 f(y)分别是出现搜索词 x 和 y 的网页数量，f(x,y)是同时出现搜索词 x 和 y 的网页数量，M 是搜索到的网页总数。

2.1.4 标签相关性

两个标签词之间的相关性存在直接相关性和间接相关性，直接相关性是指两个标签直接存在直接的语义联系，具有直接相关性的两个标签往往以较高的概率伴随出现；间接相关性是指两个标签之间不存在直接的关联，但这两个标签都与另外的一个标签存在直接相关性，于是称这两个标签存在间接相关性。具有间接相关性的两个标签虽然不常常伴随出现，但二者仍然具有重要的联系[13,14]。

对于两幅图像之间的语义相关性，由于每幅图像可能带有多个标签，因此可以取两幅图像之间各个标签两两相关性的最大值或平均值作为图像之间的标签相关性。

2.2 技术工具

本文在研究和实现图像标签完备方法的过程中使用 Python 语言作为计算工具，主要用到了 OpenCV、Tensorflow、Sklearn、Numpy 和 PyQt5 等库函数；在设计系统可视化界面时使用 QT Designer 设计工具搭建 UI 界面。

OpenCV 是一个基于 BSD 许可发行的跨平台开源计算机视觉库，可以在 Windows、Linux、和 MacOS 等系统运行，它用 C 语言和 C++ 语言编写并进行了深度优化，从而可以享受多线程处理的优势。OpenCV 的一个目标是提供易于使用的计算机视觉接口，从而帮助人们快速建立精巧的视觉应用。它轻量级而且高效，实现了图像处理和计算机视觉方面的很多通用算法，其中也包括 SIFT 算法和聚类算法等，本文的在提取 SIFT 描述子以及 VLAD 算法中实现聚类操作时都使用了 OpenCV 提供的接口。

TensorFlow 是一个开源软件库，用于各种感知和语言理解任务的机器学习，TensorFlow 允许将深度神经网络的计算部署到任意数量的 CPU、GPU 的服务器、PC 或移动设备上，且只利用一个 TensorFlow API，此外它还拥有包括 TensorFlow Hub、TensorFlow Lite 和 TensorFlow Research Cloud 等在内的多个项目以及各类应用程序接口。在 TensorFlow 中提供了一些熟知的卷积神经网络预训练模型，其中包括 VGG16 模型。本文在利用 VGG16 网络提取 CNN 特征的过程中使用了 TensorFlow 提供的函数接口。

Sklearn 是基于 Python 语言的机器学习工具，全称是 Scikit-Learn，它建立在 NumPy 和 SciPy 之上，提供了大量用于数据挖掘和分析的工具以及支持多种算法的一系列接口，集成了分类、回归、聚类、降维、模型选择和预处理六大任务模块，其中降维方面包括了 PCA 降维算法。本文在实现主成分分析算法时使用了 Sklearn 提供的 PCA 接口。

NumPy 是 Python 语言中用于科学计算的第三方库，它提供多维数组对象和各种派生对象，以及用于数组快速操作的各种 API，有包括数学、逻辑、形状操作、排序、选择、输入输出、离散傅立叶变换、基本线性代数、基本统计运算和随机模拟等等各类接口。本文在图像标签完备标注方法研究与实现的过程中多处使用了 Numpy 函数接口。

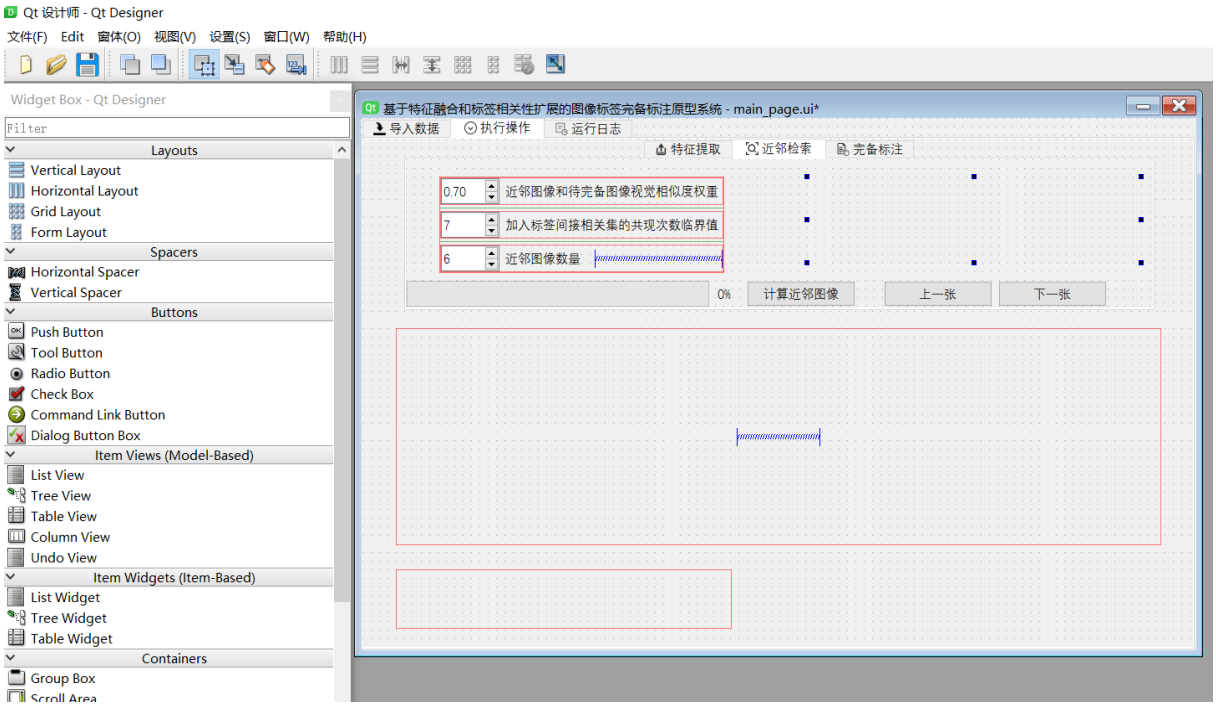

图 2.7 QT Designer 设计工具使用界面

在系统可视化过程中，本文使用 QT Designer 作为图形用户界面的实现工具，具体使用界面如图 2.7 所示。它是一款使用 Qt 小部件设计和构建图形用户界面的 Qt 界面设

计工具，可以按所见即所得的方式快速搭建图形界面，实现了视图和逻辑的分离，从而加快了开发速度。

PyQt5 是一套 Python 绑定 Digia QT5 应用的框架，其作为 Python 的一个第三方库，它有 620 多个类以及 6000 多个函数或方法；它是一个跨平台的工具包，可以运行在 UNIX，Windows，MacOS 等操作系统之上。本文在将利用 QT Designer 设计的 UI 界面与底层算法实现绑定的过程中使用了 PyQt5 函数库。

2.3 本章小结

本章主要介绍了本文研究过程中所涉及的相关理论，以及系统实现过程中所使用技术工具。其中详细介绍了 SIFT 特征和 CNN 特征的提取方式及其特点，介绍了欧式距离和 Google 距离这两种重要的距离度量方法，阐述了标签之间的相关性并说明了标签之间的间接相关性的挖掘意义，最后介绍了本文在方法研究和系统实现过程中所采用的编程语言和使用的技术工具。总体来讲，本章为下文具体阐述研究过程和图像标签完备标注方法提供了理论来源和技术支持。

第三章 图像多种特征提取和近邻检索

本章主要介绍图像完备标注过程中的特征提取和近邻检索模块，其中首先在第一节介绍本章要解决的主要问题和所采用的方法，然后从第二节开始依次介绍对数据集预处理的方法、图像 SIFT 特征和 CNN 特征的提取过程、图像间的相似度度量方法以及近邻图像的生成过程。

3.1 问题描述与方法介绍

标签完备算法在补全待完备图像缺失标签的过程中，首要环节是生成待完备图像的候选标签。本文选择以待完备图像的近邻图像标签作为候选标签，为此首先需要进行近邻检索查找出待完备图像的近邻图像。由于待完备图像的近邻图像标签直接作为候选标签，成为后续选择缺失标签的重要数据来源，所以近邻检索算法的优劣在一定程度上将直接决定图像完备标注的效果。本章图像多种特征提取和近邻检索方法流程如图 3.1 所示。

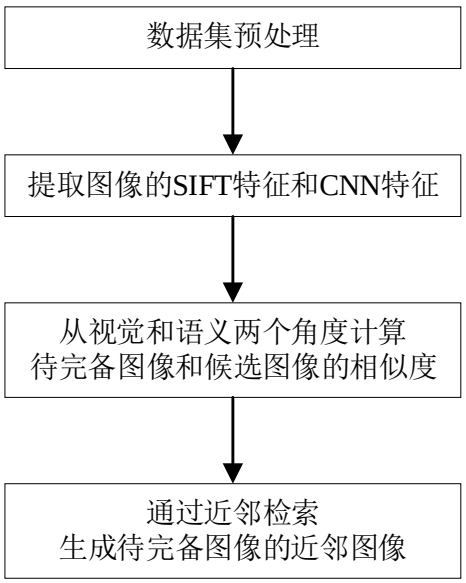

图 3.1 图像的多种特征提取和近邻检索方法流程

3.2 数据集预处理

本文的算法研究及系统测试等均基于 Corel5k 数据集进行。Corel5k 数据集是由 Corel 公司收集整理的 5000 幅图像，其涵盖多个主题，例如风景、老虎、飞机等等，共包括 260 个标注词，数据集中每幅图片大小均为 128×192 像素或 192×128 像素。

将数据集中的 5000 幅图像随机划分为两部分，第一部包含 4500 幅图片用于近邻图像查找，第二部分包含 500 幅图像，对这 500 幅图像随机去掉一些标签后用作待完备图像。

经过上述处理，第一部分中每幅图像包含 1~7 个标注词，第二部分中每幅图像包含 1~2 个标注词。然后将两部分中每幅图像的标注情况分别构建一个标注矩阵，两个矩阵形状分别是 4500×260 和 500×260，每行表示一幅图像的标注情况，每列表示一个标注词，以 0 表示未标注，1 表示标注。第一部分图像的标签矩阵如图 3.2 所示，第二部分图像的标签矩阵如图 3.3 所示。

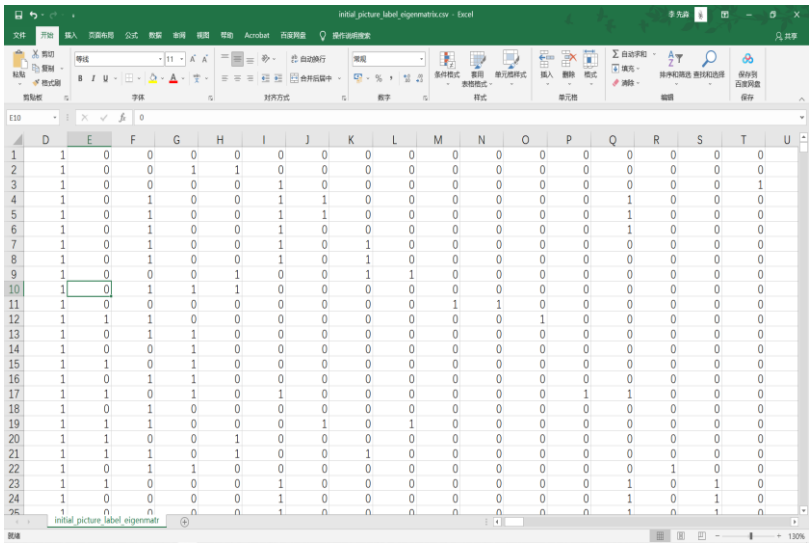

图 3.2 第一部分图像标签矩阵

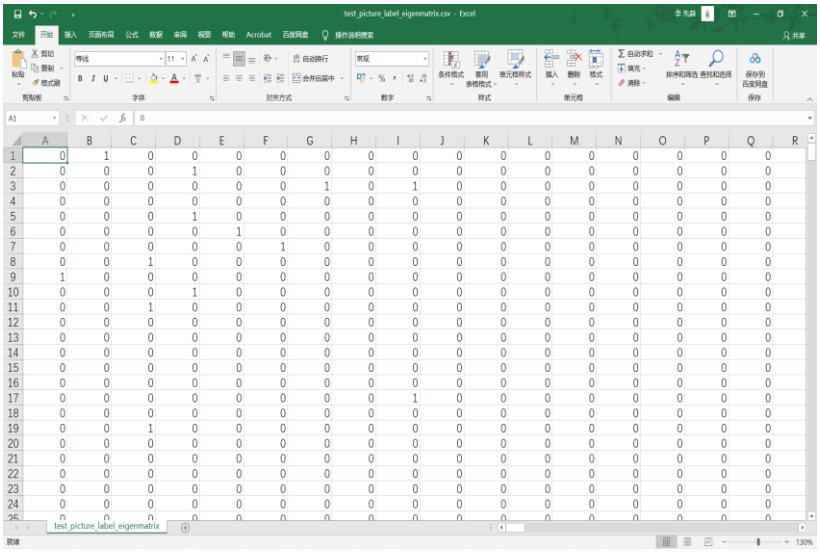

图 3.3 第二部分图像标签矩阵

3.3 视觉特征提取

3.3.1 SIFT 特征向量生成

SIFT 特征优势在于尺度变换、平移变换和旋转变换的不变性，对视角变化、仿射变换、噪声也保持一定程度的稳定性[11]。

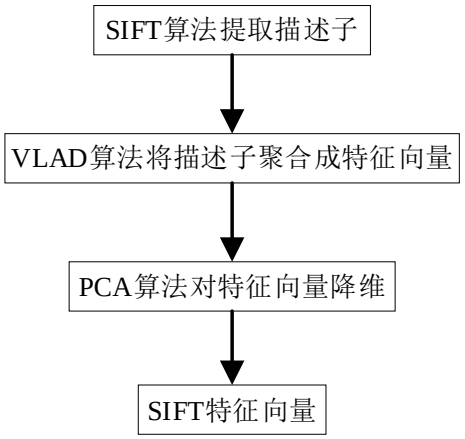

图 3.4 SIFT 特征向量生成步骤

在本文中，生成一幅图像 SIFT 特征向量的方法如图 3.4 所示，具体的步骤和说明如下：

（1）SIFT 算法提取描述子。

在 SIFT 算法中通过构建尺度空间、关键点搜索和定位、方向赋值以及描述子生成等步骤，提取出图像的 SIFT 描述子。对一幅图像而言，SIFT 算法提取出的描述子是一个 n×128 的矩阵，其中 n 的值取决于该幅图像特征点的个数，SIFT 算法本身对每个特征点采用 128 维的向量描述。

OpenCV 是一个基于 BSD 许可发行的跨平台计算机视觉库，它轻量级而且高效，实现了图像处理和计算机视觉方面的很多通用算法，其中也包括 SIFT 算法，本文的标签完备算法在提取 SIFT 描述子以及后续提到的 k-means 聚类算法时使用了 OpenCV 提供的接口。

（2）VLAD 算法将 SIFT 描述子聚合成特征向量。

VLAD 算法的思路是用聚类方法训练一个编码本，即由聚类中心组成的矩阵，为每幅图像中的特征找到最近的编码本聚类中心，然后累积所有特征和聚类中心之间的差异，得到一个新的矩阵，将矩阵扁平化并通过 L2 范数化归一化，得到的向量就是聚合的 SIFT

特征向量。L2 范数归一化计算方法如公式(3.1)所示。

$$X = X \Big/ \sqrt{x_1^{\,2} + x_2^{\,2} + \cdots + x_n^{\,2}} \tag{3.1}$$

式(3.1)中，X 为展平后得到的行向量，x_n为行向量 X 的各个分量。VLAD 算法的具体步骤见表 3-1 所示。

表 3-1 VLAD 算法步骤

操作内容
Step1：提取所有训练集图像的 SIFT 描述子矩阵 des_i，得到 N 个 des_i
Step2：以每个 des_i 中的一行为一个样本，将 N 个 des_i 中所有样本一起执行 k-means 聚将生成的 k 个聚类中心组成的矩阵作为码本 $[c_1, c_2, \cdots, c_k]$
Step3：对每幅图像的 des_i 的各个样本 des_{ij}，计算 des_{ij} 和距离其最近的聚类中心 c_m 的残差 v_m；当有多个 des_{ij} 落入同一个聚类中心 c_m 时，v_m 则为各个残差的累加和
Step4：统计所属 k 个聚类中心的残差累加和 $[v_1, v_2, \cdots, v_k]$，如聚类中心 c_m 没有 des_{ij} 落入则 v_m 表示为零向量，最终得到聚合矩阵 $V_{k,128}$
Step5：将矩阵 $V_{k,128}$ 展平并做 L2 范数归一化处理后即为 SIFT 特征向量 X

（3）PCA 算法对特征向量降维。

在 VLAD 算法聚合后生成的 SIFT 特征向量是一个超长的行向量，特征向量长度较长为后续计算带来更多的时间开销，降低了算法性能；为此，使用主成分分析法对特征向量降维，在损失小部分原始特征的条件下尽可能缩短向量长度。

PCA（Principal Component Analysis）即主成分分析方法，是一种使用最广泛的数据降维算法。PCA 的主要思想是将 n 维特征通过数学运算转换成 k 维，这 k 维是全新的正交特征也被称为主成分，是在原有 n 维特征的基础上重新构造出来的 k 维特征。

PCA 的具体操作是从原始数据空间中依次找出一组相互正交的轴，新轴的选择与数据本身密切相关。其中，第一条新轴被选为原始数据中方差最大的方向，第二条新轴被选为在原始数据与第一条轴正交的平面中方差最大的轴，第三条轴是在原始数据与第一条和第二条轴正交的平面中方差最大的轴。通过类比可以得到 n 个这样的轴。在以这种方式得到的新轴中，大部分方差包含在前 k 个轴中而后几个轴的方差几乎为零，因此，只保留包含大部分方差的前 k 个轴。这等价于保留了包含大部分方差的维度特征而忽略

了包含几乎为零方差的维度特征，实现了数据特征的降维。

Sklearn （全称 Scikit-Learn）是基于 Python 语言的机器学习工具，它集成了分类、回归、聚类、降维、模型选择和预处理六大任务模块，其中也包括 PCA 降维算法。在本文中使用了 Sklearn 提供的 PCA 算法接口，将数据集划分的第一部分作为 PCA 算法的训练集，训练好降维模型并保存，用于系统使用过程中对待完备图像降维。

3.3.2 CNN 特征向量生成

卷积特征对图像的局部特征感知更精细，通过多层卷积和池化可以提取到更全局化、更精细的特征。当数据集图片数量较少时训练出的 CNN 模型识别度较低，因此本文选用预训练的 VGG16 模型作为特征提取器。

VGG16 模型由 5 层卷积层、3 层全连接层构成，如图 3.5 所示，层与层之间使用池化层减少运算量，所有隐层的激活单元都采用 ReLU 函数。VGG16 舍弃了传统拥有较大卷积核的卷积层，改为使用多个较小卷积核（3×3）的卷积层，这样做一方面可以减少训练过程中参数的数量以降低模型复杂度和训练时间，此外还相当于进行了更多的非线性映射，可以增加网络的表达能力。除此之外，VGG16 模型还拥有小池化核、通道数多、层数更深等特点。

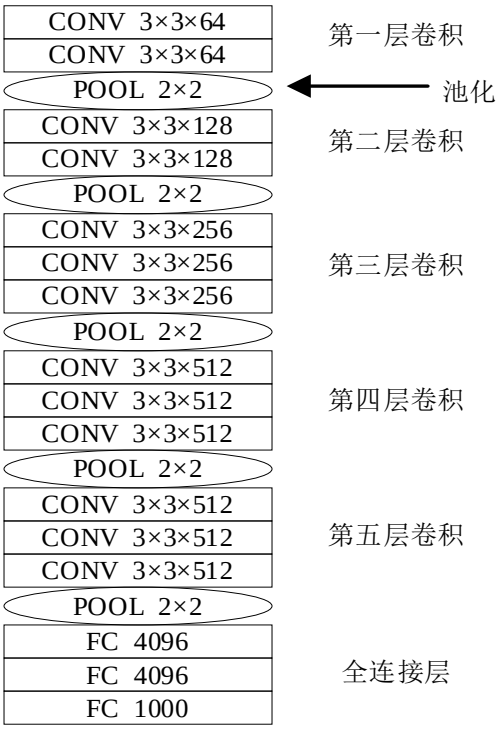

图 3.5 VGG16 模型的网络结构

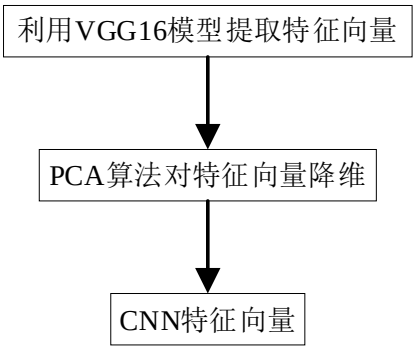

图 3.6 CNN 特征向量生成步骤

在本文中，生成一幅图像 CNN 特征向量的方法如图 3.6 所示，主要分为提取特征向量和对特征向量降维两步，具体步骤如下：

（1）利用 VGG16 模型对特征向量降维。

导入预训练的卷积核权重后，VGG16 网络对输入的图片进行多层的卷积和池化操作，不断更新 Feature Map 使得提取到的特征趋于全局化，同时由于较小尺寸卷积核的控制，对局部特征的感知也更加精细。将最后一次池化操作后且进入全连接层之前的 Feature Map 提取出来，展平后做 L2 范数归一化处理即得到 CNN 特征向量。

TensorFlow 是一个开源软件库，用于各种感知和语言理解任务的机器学习，拥有包括 TensorFlow Hub、TensorFlow Lite、TensorFlow Research Cloud 在内的多个项目以及各类应用程序接口。在 TensorFlow 中提供了一些熟知的卷积神经网络模型，其中包括 VGG16 模型。本文利用 VGG16 网络提取 CNN 特征的过程中使用了 TensorFlow 提供的函数接口。

（2）PCA 算法对特征向量降维。

利用 VGG16 模型提取出的 CNN 特征向量同样是一个超长的行向量，在对其进行 PCA 降维以缩短向量长度后可以极大的加快计算效率，具体的降维过程以及 PCA 降维理论见本章上一小节 SIFT 特征向量生成所述的 PCA 降维，不再赘述。

3.4 图像间的相似度度量

3.4.1 图像间视觉相似度度量

在生成图像特征向量之后，本文利用欧式距离度量两幅图像的相似性；由于欧氏距离和图像的相似度成反比，即距离越大图像相似度小，因此对欧式距离进行负指数化处理，使得数值与图像相似度呈正相关，同时将其映射到 0-1 区间，以此作为两图像的视

觉相似度。欧氏距离计算方法如公式(3.2)所示。

$$\text{Euclidean\_D}(X,Y)=\sqrt{(x_1-y_1)^2+(x_2-y_2)^2+\cdots+(x_n-y_n)^2} \tag{3.2}$$

式(3.2)中，$X=[x_1,x_2,\cdots,x_n]$和$Y=[y_1,y_2,\cdots,y_n]$分别为两个同类型的特征向量，n 是向量长度。接着，对欧式距离进行负指数[10]处理得到视觉相似度。计算如公式(3.3)所示。

$$\text{visual\_sim}(I_X,I_Y)=e^{-\text{Euclidean\_D}(X,Y)} \tag{3.3}$$

式(3.3)中，X 和 Y 分别是两幅图像I_X和I_Y的同类型特征向量，$\text{Euclidean_D}(X,Y)$是两向量的欧式距离。在本文中所设计的视觉相似度为 SIFT 特征和 CNN 特征的综合相似度，为此，首先根据两幅图像的 SIFT 特征向量和 CNN 特征向量分别计算相应的 SIFT 视觉相似度和 CNN 视觉相似度，然后加权求和作为二者的综合视觉相似度。本文定义综合视觉相似度的计算方法如公式(3.4)所示。

$$\text{mixed\_visual}(I_X,I_Y)=\text{visual\_sim}(X_{sift},Y_{sift})\times\beta_1+\text{visual\_sim}(X_{cnn},Y_{cnn})\times(1-\beta_1) \tag{3.4}$$

式(3.4)中，X_{sift}和Y_{sift}分别为图像I_X和图像I_Y的 SIFT 特征向量，同理，X_{cnn}和Y_{cnn}分别为图像I_X和图像I_Y的 CNN 特征向量，$\text{visual_sim}(X_{sift},Y_{sift})$是图像$I_X$和图像$I_Y$的 SIFT 特征视觉相似度，$\text{visual_sim}(X_{cnn},Y_{cnn})$是图像$I_X$和图像$I_Y$的 CNN 特征相似度，$\beta_1$是 SIFT 相似度和 CNN 相似度的融合权重。

3.4.2 标签的直接相关性和间接相关性

标签之间的相关度包括直接相关度和间接相关度。本文中标签之间的直接相关度通过 Google 距离来度量。Google 距离是一种语义相似性度量方法，由计算机自然语言处理专家 Rudi L.Cilibrasi 和 Paul M.B.Vitanyi 提出[12]，在自然语言意义上有相同或类似含义的标签词往往 Google 距离较小，而有不同含义或者相关度较低的标签词则往往距离较大。Google 距离的计算方法如公式(3.5)所示。

$$\text{Google\_D}(L_X,L_Y)=\frac{\max(\log_{10}\text{count}(L_X),\log_{10}\text{count}(L_Y))-\log_{10}\text{count}(L_X,L_Y)}{\log_{10}N-\min(\log_{10}\text{count}(L_X),\log_{10}\text{count}(L_Y))} \tag{3.5}$$

式(3.5)中，L_X和L_Y分别代表两个标签词，$\text{count}(L_X)$和$\text{count}(L_Y)$分别表示两个标签词在标签矩阵中的出现次数，$\text{count}(L_X,L_Y)$表示两个标签词L_X和L_Y同时出现的次数，

N 表示标签矩阵中所记录的图像总数。由于距离越大，两标签的语义相关度越小，为此对其进行负指数化处理[10]，使得数值与语义相关度呈正相关，同时将其映射到 0-1 区间，以此作为两标签的直接相关度。直接相关度公式如下。

$$\text{label\_rel}\left(L_X, L_Y\right) = e^{-\text{Google\_D}\left(L_X, L_Y\right)} \tag{3.6}$$

式(3.6)中，L_X 和 L_Y 分别代表两个标签词，$\text{Google\_D}\left(L_X, L_Y\right)$ 代表两个标签词的 Google 距离。此外，标签之间除了具有直接相关性，还需要考虑标签之间间接相关性，为此，本文提出了通过构建标签的间接相关集来度量标签的间接相关性，构建标签的间接相关集的主要思想是把满足与目标标签的共现频率大于给定参数的标签加入间接相关集。其中，标签L_i对标签L_y的共现频率计算公式如下所示。

$$\text{co\_frequency}\left(L_i, L_y\right) = \text{count}\left(L_i, L_y\right) / \text{count}\left(L_y\right) \tag{3.7}$$

式(3.7)中，$\text{count}\left(L_i, L_y\right)$ 是标签 L_i 和标签 L_y 在标签矩阵中共同出现的次数，$\text{count}\left(L_y\right)$ 是标签 L_y 在标签矩阵中出现的总次数，$\text{co\_frequency}\left(L_i, L_y\right)$ 是标签 L_i 和标签 L_y 的共现频率。构建标签的间接相关集的具体步骤见表 3-2。

表 3-2 构造标签的间接相关集

标签L_x关于标签L_y的间接相关集构造方法
Step1：初始化一个空集合 S
Step2：在标签矩阵中统计除L_x以外其余所有标签L_i对标签L_y的共现频率，若标签L_i对标签L_y共现频率大于 α 时，将L_i加入集合 S
Step3：返回集合 S，即为标签L_x关于标签L_y的间接相关集

在构建了标签间接相关集的基础上，本文将标签的间接相关度定义为两标签各自与其对应的间接相关集中的各标签直接相关度最大值的均值，具体定义方法如公式(3.8)所示。

$$\text{label\_indirect\_rel}\left(L_X, L_Y\right) = \frac{\max\left\{\text{label\_rel}\left(S_{Yi}, L_Y\right)\right\} + \max\left\{\text{label\_rel}\left(S_{Xi}, L_X\right)\right\}}{2} \tag{3.8}$$

式(3.8)中，S_Y 是标签 L_Y 关于标签 L_X 的间接相关集，S_X 是标签 L_X 关于标签 L_Y 的间接相关集，$\max\left\{\text{label\_rel}\left(S_{yi}, L_y\right)\right\}$ 是 L_Y 和间接相关集 S_Y 中的各标签直接相关度的最大

值，同理，$\max\{\text{label\_rel}(S_{xi},L_x)\}$ 是 L_X 和间接相关集 S_X 中的各标签直接相关度的最大值。

3.4.3 图像间的语义相似度度量

对于两个标签，二者的相关性应综合同时考虑二者直接相关性和间接相关性，为此，本文对两标签相关度的计算方法定义如公式(3.9)所示。

$$\text{label\_sim}(L_X,L_Y)=\left(\text{label\_indirect\_rel}(L_X,L_Y)+\text{label\_rel}(L_X,L_Y)\right)/2 \quad (3.9)$$

式(3.9)中，$\text{label\_indirect\_rel}(L_X,L_Y)$ 是标签 L_X 和 L_Y 的间接相关度，$\text{label\_rel}(L_X,L_Y)$ 是标签 L_X 和 L_Y 的直接相关度。然而对于数据集中的图像，每幅图像往往带有一个或者多个标签，在度量两幅图像的语义相似度时需要综合考虑各个标签。本文对两幅图像之间的语义相似度度量方法如公式(3.10)所示。

$$\text{semantic\_sim}(I_X,I_Y)=\frac{\sum_{i=1}^{m}\sum_{j=1}^{n}\text{label\_sim}(L_i,L_j)}{m\cdot n} \quad (3.10)$$

式(3.10)中，m、n 分别是两幅图像 I_X 和 I_Y 所带有的标签个数，L_i、L_j 分别是图像 I_X 和 I_Y 的标签，$\text{label\_sim}(L_i,L_j)$ 是两标签的综合相关度，由公式(3.9)得到。

3.5 近邻检索

3.5.1 近邻图像的生成

本文在检索待完备图像的近邻图像过程中，首先提取图像的 SIFT 特征和 CNN 特征并计算综合视觉相似度，然后计算待完备图像和其它图像的语义相似度，最后综合考虑融合视觉相似度和语义相似度来查找近邻图像。综合相似度的计算方法如公式(3.11)所示。

$$\text{similarity}(I_X,I_Y)=\text{mixed\_visual}(I_X,I_Y)\times\beta_2+\text{semantic\_sim}(I_X,I_Y)\times(1-\beta_2) \quad (3.11)$$

式(3.11)中，I_X 和 I_Y 是待比较的两幅图像；$\text{mixed\_visual}(I_X,I_Y)$ 是图像 I_X 和 I_Y 的综合视觉相似度，由公式(3.4)得出；$\text{semantic\_sim}(I_X,I_Y)$ 是图像 I_X 和 I_Y 的语义相似度，由公式(3.10)得出；β_2 是视觉相似度的权重占比。

根据待完备图像和所有候选图像综合相似度的计算结果，选择综合相关度高的前 γ 张图像作为待完备图像的近邻图像，近邻检索的流程如图 3.7 所示。

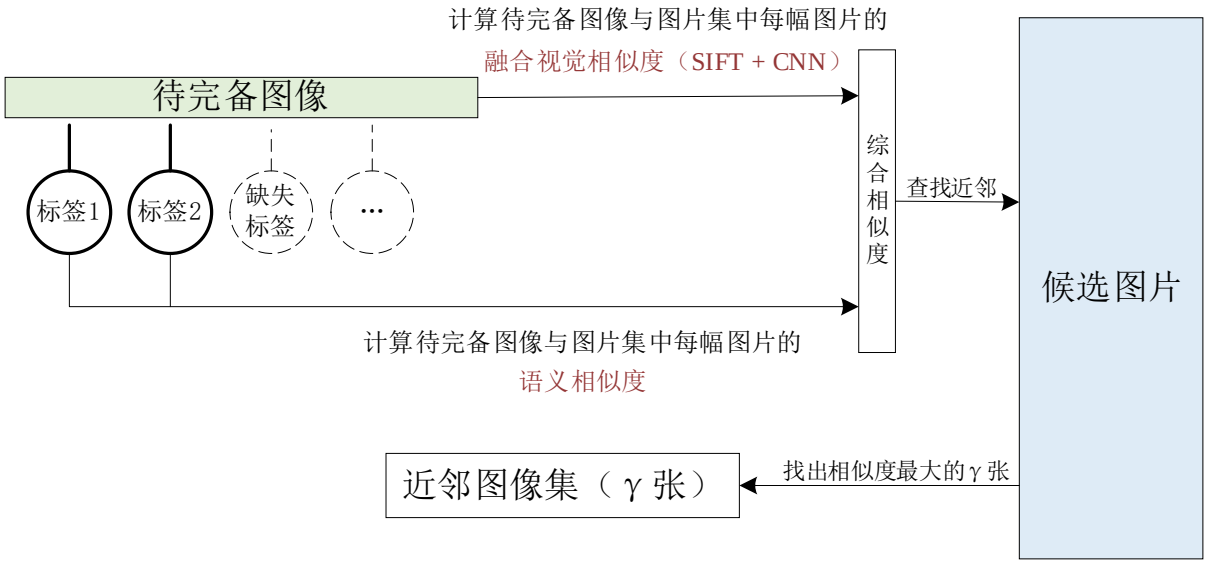

图 3.7 近邻检索流程

3.5.2 参数分析与设定

在检索待完备图像的近邻图像过程中，共提供了三个参数，分别是 SIFT 特征和 CNN 特征的融合权重β_1、加入标签间接集的最低共现频率α以及近邻检索的视觉相似度权重占比β_2。本文选择以平均准确率(average precision, AP)指标来分析近邻检索算法性能。在近邻检索中，平均准确率是指每幅待完备图像准确找出的近邻图像数量和找出的所有近邻图像数量的比值，如公式(3.12)所示。

$$AP = \frac{1}{k}\sum_{i=0}^{k}\frac{PN(I_i)}{AN(I_i)} \tag{3.12}$$

式(3.12)中，k 是指测试的待完备图像数量，$PN(I_i)$是指待完备图像I_i的近邻图像中正确的图像个数，$AN(I_i)$指图像I_i的所有近邻图像个数。基于 Corel5k 数据集，利用平均准确率对各个参数的分析如下：

（1）参数β_1：作为 SIFT 特征和 CNN 特征的融合权重，因此β_1在 0 到 1 之间取值。β_1的大小决定了 SIFT 特征在综合相似度计算时的占比，β_1越大 SIFT 特征所占比重越多，β_1为 0 时相当于不考虑 SIFT 特征，为 1 时相当于只考虑 SIFT 特征。β_1取不同数值时近邻检索平均准确率的变化情况如图 3.8 所示。可以看到，当β_1取 0.4 时为近邻检索的平均准确率可以达到最优。

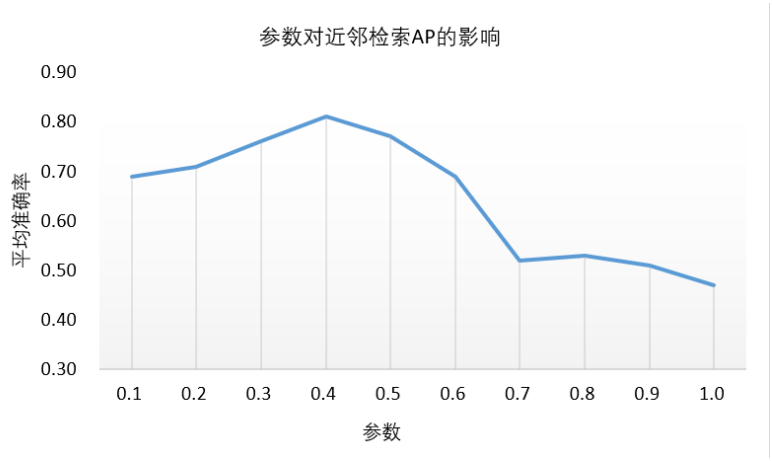

图 3.8 参数β_1对近邻检索效果的影响

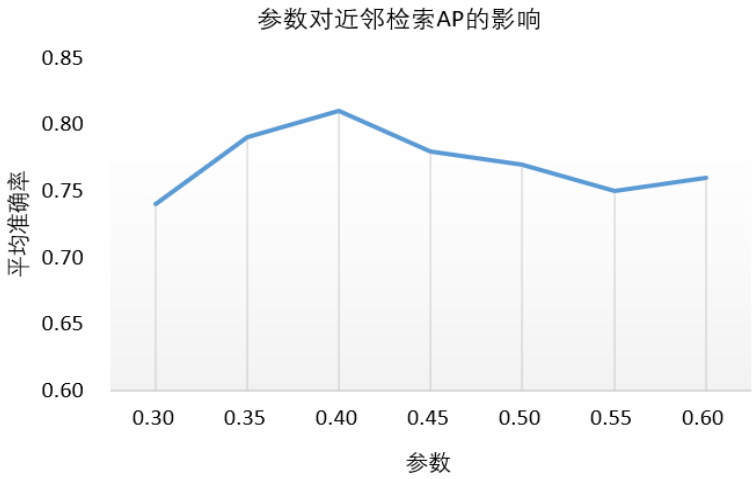

图 3.9 参数α对近邻检索效果的影响

（2）参数α：作为加入标签间接集的最低共现频率，α的取值范围同样是在 0 到 1 之间。当α取值较小时，最直接的影响是使得标签间接相关集中的标签数量激增，进而使得语义相似度的计算量空前增大，同时α取值过小或者过大也意味着标签加入间接相关集的两种极端，不符合间接相关集的设计思想，因此本文综合考虑规定α的范围在 0.3 到 0.6 的闭区间内。α取不同数值时近邻检索平均准确率的变化情况如图 3.9 所示。可以看到，当α取 0.4 时为近邻检索的平均准确率可以达到最优。

（3）参数β_2：作为近邻检索的视觉相似度权重占比，因此β_2在 0 到 1 之间取值。β_2的大小决定了视觉特征在查找近邻图像时的占比，β_2越大视觉特征所占比重越多，β_2为 0 时相当于不考虑视觉特征，为 1 时相当于只考虑视觉特征。β_2取不同数值时近邻检索平均准确率的变化情况如图 3.10 所示。可以看到，当β_2取 0.8 时为近邻检索的平

均准确率可以达到最优。

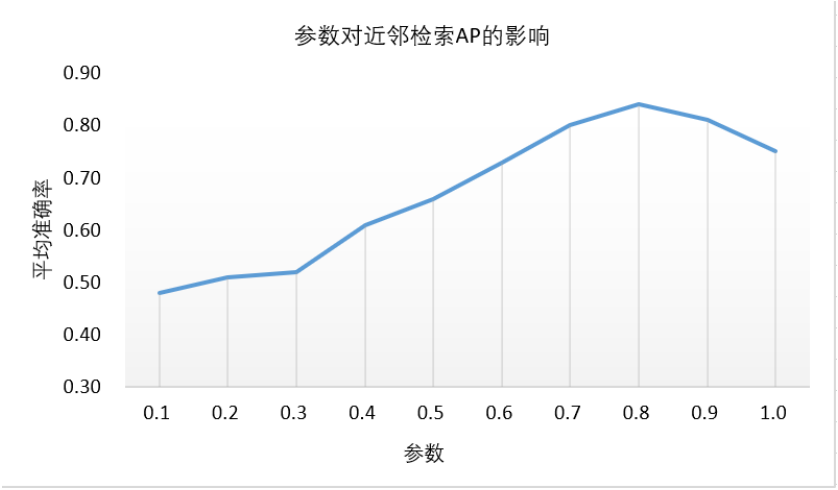

图 3.10 参数 β_2 对近邻检索效果的影响

3.6 本章小结

本章设计了一种基于多种特征和图像初始标签语义的近邻检索方法。其中利用欧式距离给出了对 SIFT 特征和 CNN 特征的综合视觉相似度计算方法；利用 Google 距离和标签间接相关集给出了计算标签综合相关度的方法；最后在 3.5 节中阐述了近邻检索的具体步骤以及本文所构建的近邻检索的方法，并分析了近邻检索过程中相关参数的设定。

第四章 标签与图像的相关度度量和完备标注

本章主要介绍图像完备标注过程中的完备标注模块，其中首先在第一节介绍本章要解决的主要问题和所采用的方法，然后从第二节开始依次介绍待完备图像候选标签的生成方法、标签与图像相关度度量方法以及缺失标签的生成过程。

4.1 问题描述和方法介绍

在第三章的基础上，通过近邻检索方法计算出了待完备图像的近邻图像，而本章要实现的是补全待完备图像的缺失标签，即完备标注功能。本文将近邻图像所带有的标签作为待完备图像的候选标签，从候选标签中选择出属于待完备图像的缺失标签补全。因此，在选择缺失标签的过程中，需要度量候选标签和待完备图像的相关度。然而标签和图片属于两个不同的模态，因此需要对其进行模态转换，从标签与标签、图像与图像两个角度进行考虑标签与图像的关系，即从度量标签与图像的视觉相关度和语义相关度两个方面综合考虑各个候选标签和与待完备图像的相关度，进而实现完备标注。本章标签与图像的相关度量和完备标注方法流程如图 4.1 所示。

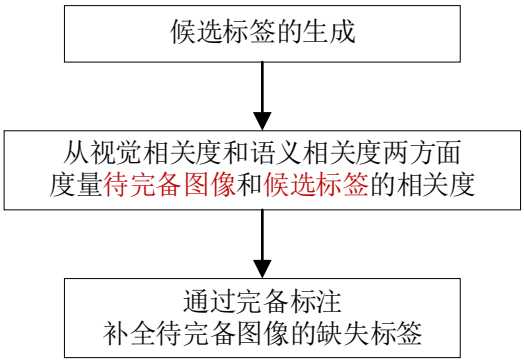

图 4.1 标签与图像的相关度量和完备标注方法流程

4.2 候选标签的生成

通过近邻检索，图像标签完备算法得到了待完备图像的近邻图像。在完备标注模块，图像标签完备算法将会选择出待完备图像的缺失标签并补全。本文将待完备图像的近邻图像所带有的标签作为候选标签。候选标签的提取步骤见表 4-1 所示。

在选择出待完备图像缺失标签的过程中，需要度量候选标签与待完备图像的相关度，而由于标签与图像属于两个不同的模态，无法直接度量，为此，本文选择将度量标签与

图像的相关度转换为度量图像与图像、标签与标签的相关度。

表 4-1 候选标签的提取步骤

操作
Step1：构建一个空集合 Candidate_S
Step2：依次遍历每幅近邻图像所带有的标签 L_i，若 L_i 不在 Candidate_S 中，则加入
Step3：从集合 Candidate_S 中去除掉待完备图像的初始标签
Step4：返回集合 Candidate_S，集合中的标签即为待完备图像的候选标签

4.3 标签与图像的相关度度量

4.3.1 候选标签的图片依赖集构建

为度量候选标签与待完备图像的相关度，首先从图像与图像角度进行模态转换。为此，本文对每个候选标签构建了候选标签的图片依赖集，即从所有图片中选择出含有最多“候选标签”、“待完备图像初始标签”以及“初始标签关于候选标签的间接相关集中的标签”的图片组成一个集合，具体构建方法如下：

（1）首先计算初始标签关于候选标签的间接相关集，间接相关集的构建方法由表 3-2 得出；然后将候选标签 L_c、待完备图像初始标签 L_PS_i 以及初始标签关于候选标签的间接相关集中的标签 $L_c\_indirectS_i$ 组成一个无重复标签的标签组 L_S。

（2）在图片集的标签矩阵中查找同时含有 L_S 中各个标签的图片，将找到的图片组成一个集合 $Dependent\_S$。

（3）若图片集合 $Dependent\_S$ 为空，则随机去掉 $L_c\_indirectS$ 中的一个标签，返回步骤（1）；直到 $L_c\_indirectS_i$ 为空时，若 $Dependent\_S$ 仍然为空，则随机去掉 L_PS 中的一个标签，返回步骤（1）；直到 $Dependent\_S$ 不为空。

（4）返回 $Dependent\_S$ 即为候选标签 L_c 的图片依赖集。

下面以一个具体例子来说明图片依赖集的建立过程。假设候选标签 L_c 为 sky，待完备图像的初始标签为{water}，两个初始标签关于候选标签 L_c 的间接相关集的并集为{clouds，grass}，构造候选标签 L_c 的图片依赖集算法流程见表 4-2 所示。

表 4-2 构造候选标签的图片依赖集实例

构造流程实例
Step1：初始的标签组 L_S 是{sky，water，clouds，grass}，在图片集标签矩阵中查找同时含有这几个标签的图像组成集合 $Dependent\_S$，假设此时 $Dependent\_S$ 为空
Step2：随机去掉 $L_c\_indirectS$ 中的一个标签，假设去掉 grass，然后重新组成的标签组 L_S 是{sky，water，clouds }，查找同时含有这几个标签的图像组成集合 $Dependent\_S$，仍为空
Step3：再次随机去掉 $L_c\_indirectS$ 中的一个标签，即 clouds，然后再重新组成的标签组 L_S 是{sky，water}，查找同时含有这几个标签的图像组成集合 $Dependent\_S$，不为空
Step4：返回 $Dependent\_S$

4.3.2 标签与图像的视觉相关度度量

在构建了候选标签图片依赖集的基础上，本文对候选标签与待完备图像之间的视觉相关度定义为待完备图像和某候选标签图片依赖集中各图像的综合视觉相似度的均值，即将候选标签的所表达的视觉内容映射到该候选标签对应的图片依赖集中，具体定义如公式(4.1)所示。

$$visual\_rel(L_c, I) = \frac{\sum_{i=1}^{m} mixed\_visual(I, Dependent\_S_i)}{m} \tag{4.1}$$

式(4.1)中，L_c 是候选标签，I 是待完备图像，$Dependent\_S$ 是候选标签 L_c 的图片依赖集，m 是 $Dependent\_S$ 中所包含的图片数量，$mixed\_visual(I, Dependent\_S_i)$ 是待完备图像和图片依赖集中每幅图片的综合视觉相似度，由公式(3.4)得到。

4.3.3 标签与图像的语义相关度度量

在度量候选标签和待完备图像相关度的过程中，除了从图像与图像的角度考虑，还应从标签与标签的角度考虑，即计算候选标签和待完备图像的语义相关度。本文对候选标签和待完备图像的语义相关度定义如公式(4.2)所示。

$$semantic\_rel(L_c, I) = \max\{label\_sim(L_c, L\_PS_i)\} \tag{4.2}$$

式(4.2)中，L_c 是候选标签，I 是待完备图像，L_PS 是待完备图像 I 的初始标签集合，$label\_sim(L_c, L\_PS_i)$ 是候选标签和待完备图像的各个初始标签的综合相关度。

4.4 完备标注

4.4.1 缺失标签的生成

为了选择出待完备图像的缺失标签，需要度量候选标签与待完备图像的相关度，然后选择出与待完备图像相关度高的前 μ 个标签作为缺失标签补全待完备图像。完备标注的流程如图 4.2 所示。

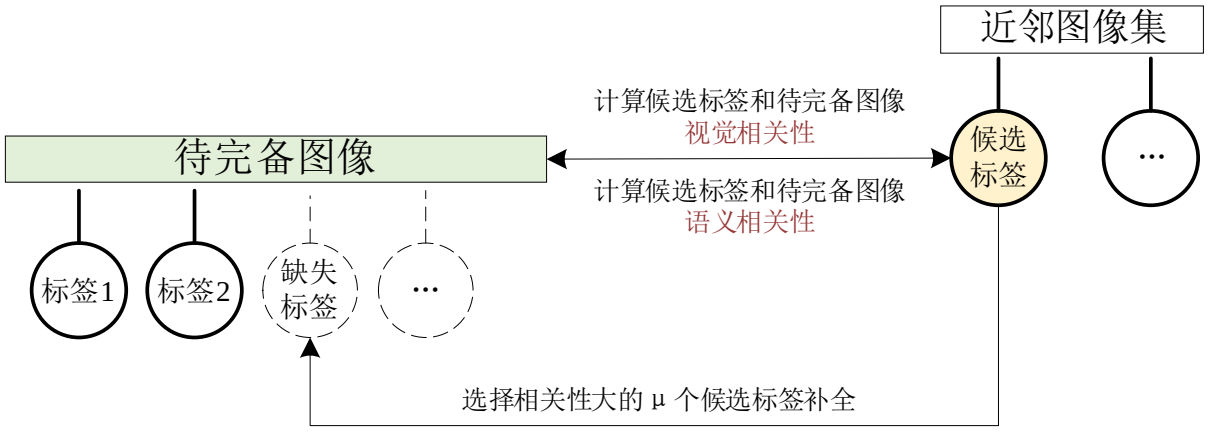

图 4.2 完备标注流程

在度量标签与图像相关度的过程中，从图像与图像、标签与标签两个角度进行了模态转换，即从标签与图像的视觉相关度和语义相关度两个方面进行构造和量化，最后综合考虑两者的结果作为候选标签与待完备图像的综合相关度。本文对标签与图像的综合相关度定义如公式(4.3)所示。

$$\mathrm{Relevancy}(L_c, I) = \mathrm{visual\_rel}(L_c, I) \times \beta_3 + \mathrm{semantic\_rel}(L_c, I) \times (1-\beta_3) \qquad (4.3)$$

式(4.3)中，L_c 是候选标签，I 是待完备图像，$\mathrm{visual\_rel}(L_c, I)$ 和 $\mathrm{semantic\_rel}(L_c, I)$ 分别是候选标签和待完备图像的视觉相关度和语义相关度，$\mathrm{Relevancy}(L_c, I)$ 是候选标签 L_c 和待完备图像 I 的相关度，β_3 是视觉相关度的权重占比。

4.4.2 参数分析与设定

在从候选标签中选择待完备图像缺失标签的过程中，提供了一个完备标注的视觉相关度权重占比参数 β_3。本文选择以平均召回率(average recall, AR)指标来衡量完备标注算法的性能。在完备标注中，平均召回率是指每幅图像准确找回的缺失标签数和待完备图像实际缺失标签数的比值，如公式(4.4)所示。

$$AN = \frac{1}{k}\sum_{i=0}^{k}\frac{PC(I_i)}{AC(I_i)} \tag{4.4}$$

式(4.4)中，k 是指测试的待完备图像数量，$PC(I_i)$是指待完备图像I_i的找回的缺失标签

个数，$AC(I_i)$指图像I_i的实际缺失标签个数。基于 Corel5k 数据集，利用平均召回率对参数β_3的分析如下：

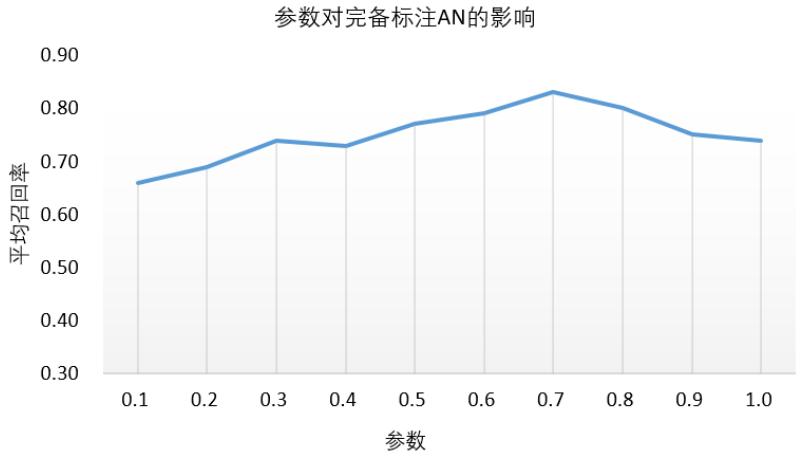

图 4.3 参数β_3对完备标注效果的影响

β_3作为完备标注的视觉相关度权重占比参数，β_3的大小决定了在从候选标签中选择缺失标签时视觉相关度的占比，β_3越大视觉相关度所占比重越多，β_3为 0 时相当于不考虑视觉相关度，为 1 时相当于只考虑视觉相关度。β_3取不同数值时完备标注的平均召回率的变化情况如图 4.3 所示，可以看到，当β_3取 0.7 时为近邻检索的平均召回率可以达到最优。

4.5 本章小结

本章提出了一种基于视觉相关性和语义相关性度量候选标签和待完备图像相关性的完备标注方法。其中，通过构建候选标签的图片依赖集实现了视觉相关性的度量；然后通过考虑标签之间的直接相关性和间接相关性给出了计算标签与图像语义相关性的方法；最后综合考虑视觉相关性和语义相关性，阐述了从候选标签中选择缺失标签的完备方法并分析了相关参数的设定。

第五章 图像完备标注原型系统

在前面章节研究的基础上，本章设计并实现了一个图像完备标注原型系统，其中第一节主要介绍系统的功能、操作流程以及系统结构，在第二节中详细展示了系统各个功能模块的测试情况和最终标注效果。

5.1 系统概述

5.1.1 软件功能和操作流程

在第三章和第四章的方法基础上，利用 Python 语言设计并实现了基于多种特征和标签相关度量的图像完备标注原型系统，同时利用 PyQt5 图形库实现了本系统的可视化图形界面；本系统共分为“数据导入”、“执行操作”、“运行日志”三个部分，其中“执行操作”部分又可以分为特征提取、近邻检索和完备标注三个模块，软件功能模块如图 5.1 所示。

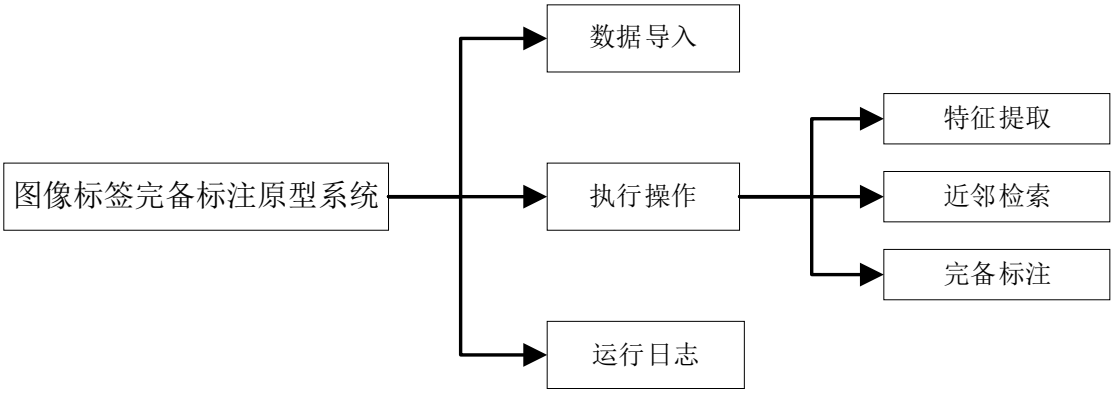

图 5.1 软件功能模块

在“数据导入”部分，需要导入系统运行所需要的图片数据集及相关的标签文件，此外，还需要训练生成或者直接导入训练好的图片集特征矩阵及相关算法模型等。在“执行操作”部分，特征提取模块实现提取待完备图像的视觉特征；近邻检索模块实现了根据视觉相似度和语义相似度查找待完备图像的近邻图像；完备标注模块实现了从候选标签中选择出与符合待完备图像的缺失标签。在“运行日志”部分，可以查看整个运行过程中生成的日志数据。

使用该系统时需根据指定的操作次序按步骤进行。从导入数据开始，把所有的数据文件导入完整后即可开始执行操作；选择好待完备图像后，按照特征提取、近邻检索、

完备标注的次序有序进行，具体操作步骤如下：

（1）导入图片集的相关文件。包括图片集所在的文件夹、图片名称、图片集标签矩阵和标签词。

（2）设置参数训练数据生成指定文件或者直接导入训练结果。训练结果的文件包括 SIFT 特征矩阵、CNN 特征矩阵、SIFT 描述子聚类码本、SIFT 降维模型和 CNN 降维模型，点击“开始训练”或“确认导入”。

（3）导入完成后自动进入特征提取页面。点击“选择待完备图像”选择一张待完备图像，选择完成后在界面左下方显示出已经选择的待完备图像及其所带有的初始标签。

（4）设置参数后点击“提取视觉特征”提取图片的 SIFT 特征和 CNN 特征，并读取特征矩阵。

（5）特征提取完成后，进入近邻检索页面。设置参数后点击“计算近邻图像”开始查找待完备图像的近邻图像，查找完成后在界面右下方显示出近邻图像及其所带有的标签；点击“上一张”或者“下一张”进行切换查看近邻图像。

（6）近邻图像查找完成后，进入完备标注页面。设置参数后点击“计算缺失标签”进行完备标注，计算完成后在待完备图像初始标签下方以红色字体显示找到的缺失标签。

（7）进入运行日志页面可以查看整个过程中生成的日志数据，在完备标注页面点击“清空”后自动进入特征提取页面，即可开始测试下一张待完备图像

（8）重复步骤（3）。

5.1.2 系统结构

该标签完备系统在导入数据文件之后，根据用户给定的参数对待完备图像进行视觉特征提取，生成特征向量用于视觉相似度计算；近邻检索模块从视觉相似度和语义相似度两个方面综合计算图像之间的相似度，根据计算结果查找出待完备图像的近邻图像，把近邻图像所带有的标签作为候选标签；完备标注模块通过计算从候选标签中选择出与待完备图像相关度高的标签作为缺失标签补全待完备图像。该系统的体系结构如图 5.2 所示。

总的来看，该系统中主要由四个线程组成，其中包括 1 个主线程和 3 个子线程。主线程除了控制整个系统的运行和关闭等最底层操作，还负责数据导入和 UI 界面的控制操作，而提取图像视觉特征、计算近邻图像和选择缺失标签这一类操作往往需要耗费大

量的时间和空间，通常将这一类计算密集型作业封装成独立的子线程，进而避免了程序运行过程中出现界面卡顿、闪退等现象。

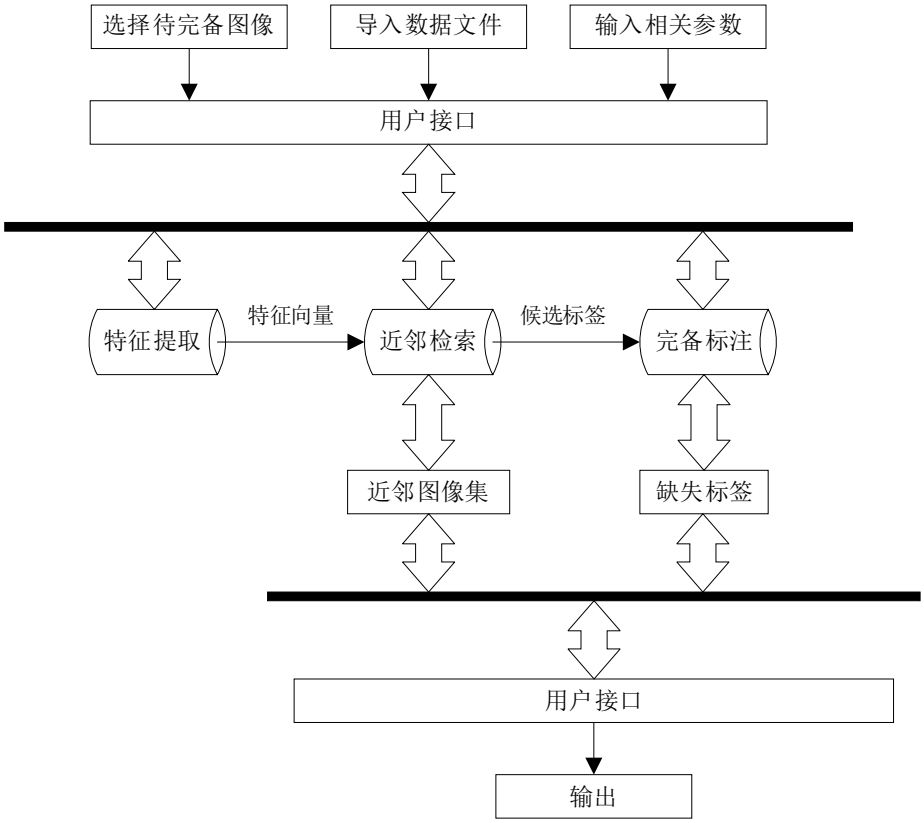

图 5.2 图像标签完备标注原型系统的体系结构

该原型系统的主体代码结构如图 5.3 所示。图中表示出了系统各模块的类名或函数名及各自的调用关系，其中“蓝色框”代表类名，“绿色框”代表函数名。

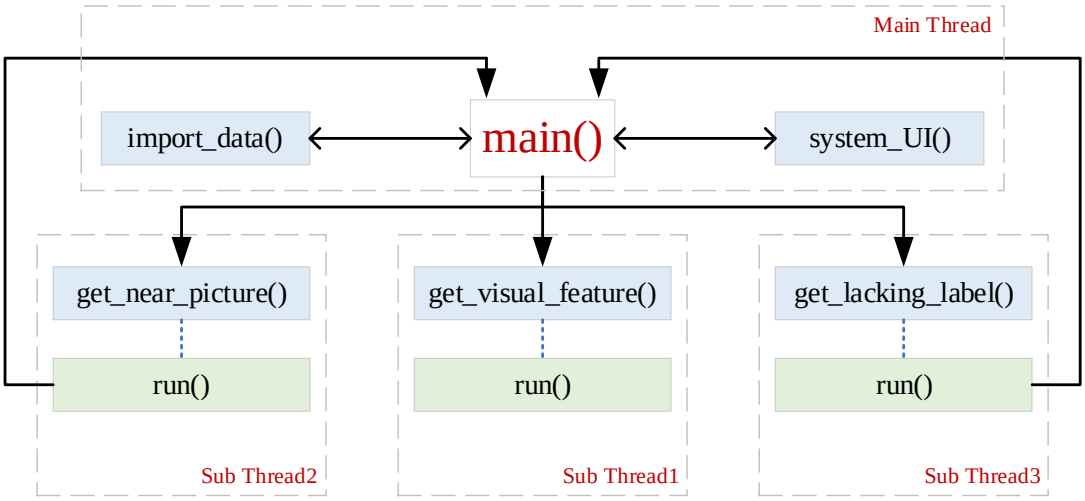

图 5.3 系统主体的函数结构

在子线程 1 中实现了对待完备图像的视觉特征提取。本文在程序代码中将特征提取操作封装在 get_visual_feature 类中，如图 5.4 所示。该类继承了 Python 语言的第三方库函数 PyQt5 的 QThread 父类并覆写了父类的 run 方法，run 方法由父类的构造函数自动调用。在 run 方法中主要执行了三个操作，分别是 read_files、get_sift_one_vector 和 get_cnn_one_vector,其中通过 read_files 函数读取已经导入的特征矩阵并转换为数组，通过 get_sift_one_vector 函数提取待完备图像的 SIFT 特征并生成 SIFT 特征向量，通过 get_cnn_one_vector 函数提取待完备图像的 CNN 特征并生成 CNN 特征向量，其余计算操作均在 run 方法中执行。

子线程 1 提取完成后由 run 方法发射对应信号，主线程收到子线程 1 发射的信号后做出对应的响应。

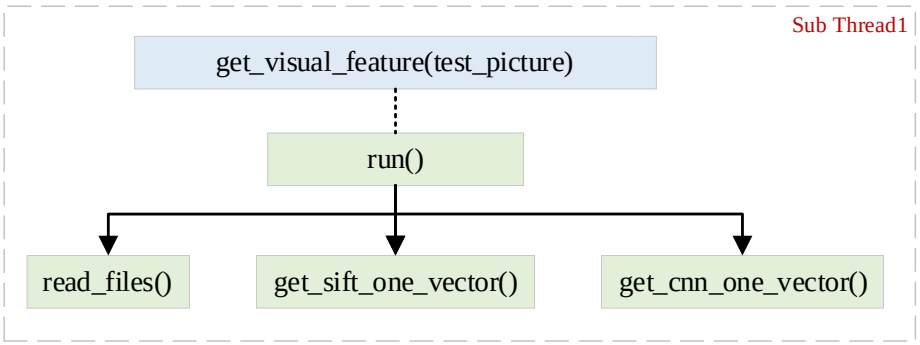

图 5.4 子线程 1 的函数结构

在子线程 2 中实现了查找待完备图像的近邻图像功能。本文在程序代码中将近邻检索操作封装在 get_near_picture 类中，如图 5.5 所示。该类也继承了 QThread 父类并覆写了父类的 run 方法。在 run 方法中通过调用 get_mixed_visual_similarity 函数来计算待完备图像和其它图像的综合视觉相似度，其中该函数又两次调用 get_visual_similarity 函数来计算 SIFT 视觉相似度和 CNN 视觉相似度，get_visual_similarity 使用 get_euclideanDistance 函数来计算两个向量的欧式距离；此外 run 方法还通过调用 get_label_similarity 函数来计算待完备图像和其它图像的语义相似度，该函数又调用 get_both_dierct 函数和 get_both_indirect 函数来计算两个标签的直接相关度和间接相关度，其中 get_both_indirect 函数通过调用 get_expanded_labelSet 来构建一个标签的间接相关集合，然后使用 get_both_googleDistance 来计算两标签的 Google 距离，其余计算操作均在 run 方法中执行。

子线程 2 近邻检索完成后由 run 方法发射对应信号，主线程收到子线程 2 发射的信号后做出相应的响应。

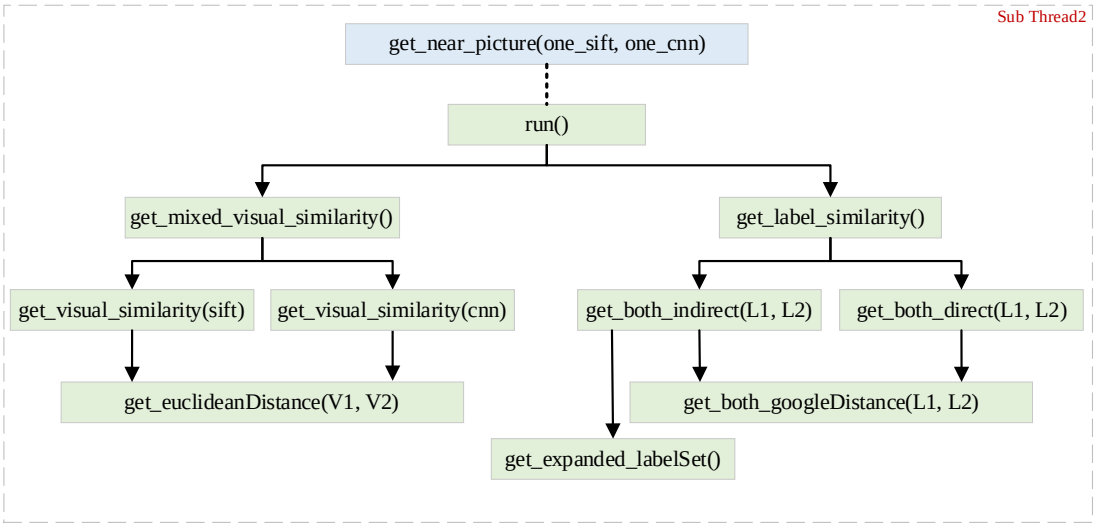

图 5.5 子线程 2 的函数结构

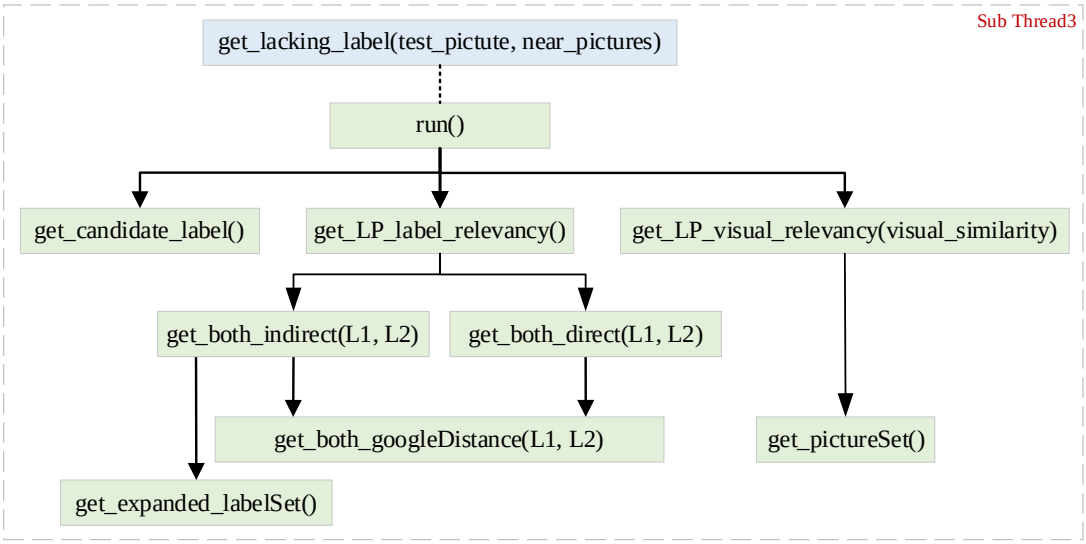

图 5.6 子线程 3 的函数结构

在子线程 3 中实现了选择待完备图像缺失标签的功能。本文在程序代码中将完备标注操作封装在 get_lacking_label 类中，如图 5.6 所示。与上述几个类一样，该类继承了 QThread 父类并覆写了父类的 run 方法。在 run 方法中首先通过 get_candidate_label 函数从近邻图像的标签中取得候选标签；然后通过 get_LP_label_relevancy 函数从语义角度计算标签与图像的相关度，其中 get_LP_label_relevancy 所引出的调用结构同近邻检索中

get_label_similarity 函数的调用结构；接着通过 get_LP_visual_relevancy 函数从视觉角度计算标签与图像的相关度，其中参数 visual_similarity 是近邻检索中计算出的综合视觉相似度，此外该函数在构建候选标签的图片依赖集时调用了 get_pictureSet 函数其余计算操作均在 run 方法中执行。

子线程 3 完备标注完成后由 run 方法发射对应信号，主线程收到子线程 3 发射的信号后做出相应的响应。

5.2 功能演示

5.2.1 数据导入

首先要导入的图片集的数据文件，一共包括四个，分别是图片集所在文件夹、图片名称、图片集的标签矩阵和标签词，如图 5.7 和 5.8 所示。

导入图片集的数据文件后，可以选择训练数据或者直接导入训练好的结果，训练结束后会自动在指定目录下生成相应的五个文件，分别是 SIFT 特征矩阵、SIFT 描述子聚类码本、SIFT 降维模型、CNN 特征矩阵和 CNN 降维模型。以上所有文件中，除图片集所在路径应选择对应文件夹和降维模型应选择.m 文件类型外，其余文件类型均为.csv 文件。设置参数进行训练和直接导入训练结果的界面分别如图 5.6 和图 5.4 所示，由于训练过程较长，通常选择直接导入训练结果。

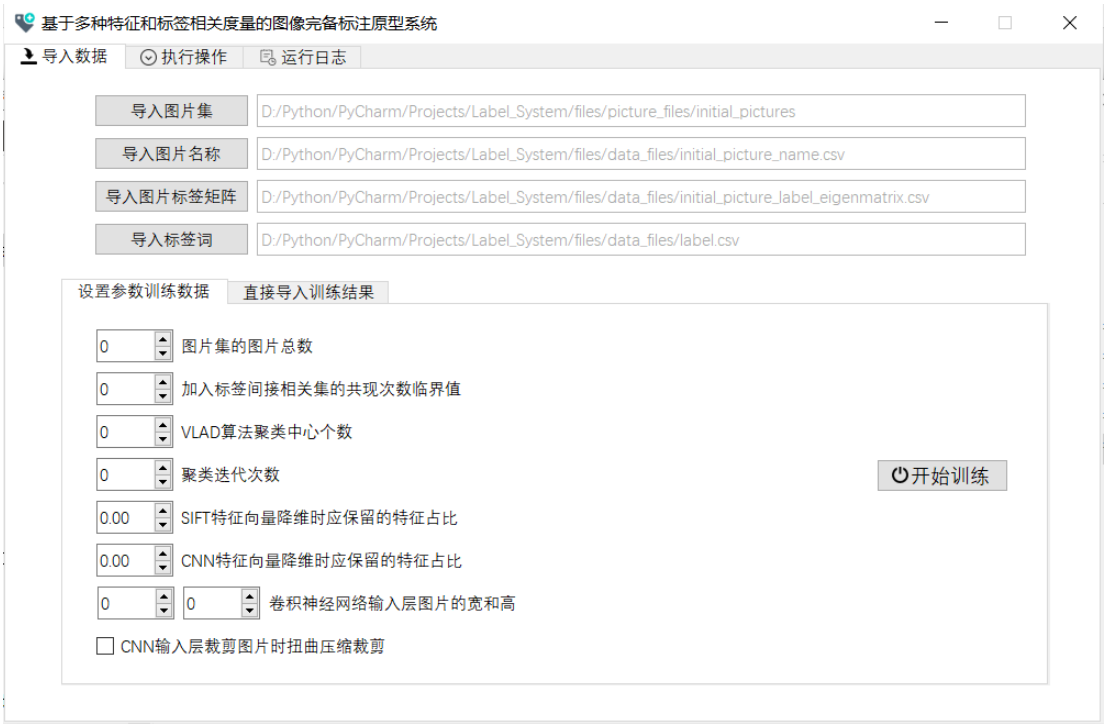

图 5.7 设置训练参数

图 5.8 导入训练结果

5.2.2 图像特征提取

在特征提取页面先选择一张待完备图像，页面左下方将会显示待完备图像以及待完备图像的初始标签，然后点击“提取视觉特征”完成提取，如图 5.9 所示。

图 5.9 选择待完备图片和特征提取

第一行的参数设定在提取 CNN 特征时卷积网络输入层对图片大小的限制，本系统默认大小为长和宽均为 128 像素；第二行参数设定在对不满足 CNN 输入层设定的图片大小的图片进行裁剪时，是否压缩图片以尽可能较多的保留图片特征，避免失真过于严重；第三行参数设定 SIFT 特征和 CNN 特征融合时的权重占比，权重为 0 表示只考虑 CNN 特征，权重为 1 表示只考虑 SIFT 特征。

5.2.3 查找近邻图像

在近邻检索页面点击“计算近邻图像”即可开始查找，在此过程中设定了本次近邻检索应返回的近邻图像数量为 9，该待完备的所有近邻图像及近邻图像所带有的标签见表 5-1 所示。

查找完成后在页面右下方显示近邻图像和其对应的标签，如图 5.10 所示；点击“上一张”或“下一张”切换查看近邻图像。

第一行参数设定在查找近邻图像时所依据的视觉相似度和语义相似度的权重占比，权重为 0 时表示只考虑语义相似度，权重为 1 时表示只考虑视觉相似度；第二行参数设定了加入标签间接相关集的最低共现次数，共现次数最低值设置得越小，计算语义相似度所需的时间越长；第三行参数设定了近邻检索完成后应返回的近邻图像数量，近邻图像数量至少为 1。

图 5.10 近邻检索

表 5-1 近邻图像及所带标签

序号	近邻图像	图像标签
1		city, sky, water, buildings, island, river
2		water, horizon, shore, restaurant, tables
3		water, sea, boats, buildings, shore, harbor, wall
4		water, boats, ship, road, dock
5		pots
6		city, water, buildings, river, skyline, night
7		city, sky, buildings, sunset, skyline
8		city, water, bridge, sky
9		city, water, hills, buildings, shore

5.2.4 完备标注

在完备标注页面，点击“计算缺失标签”开始补全，计算完成后在待完备图像的下方以红色字体显示出补全的缺失标签，如图 5.11 所示。

图 5.11 计算缺失标签

第一行参数设定从候选标签中选择缺失标签时所依据的候选标签和待完备图像之间视觉相关度和语义相关度权重占比，权重为 0 时表示只考虑语义相关度，权重为 1 时表示只考虑视觉相关度；第二行参数表示完备标注最终应补全的缺失标签个数。

可以看到待完备所具有的初始标签是 buildings，经过该系统标签完备，补全的缺失标签是 water、city、shore、sky、skyline、river，均符合待完备图像的视觉内容。至此使用本系统对一张图像进行标签完备结束。

查看整个标签完备过程中日志文件数据如图 5.12 所示。在日志文件中可以看到所选择的待完备图像路径和初始标签、近邻检索得到的各个近邻图像标签、候选标签以及各个候选标签与待完备图像的相关度排序结果。

在完备标注模块，该系统分别计算待完备图像和 22 个候选标签 island、shore、road、boats、horizon、skyline、restaurant、pots、bridge、hills、water、night、sunset、harbor、tables、dock、sea、sky、river、ships、wall、city 的相关度，并将相关度结果排序，根据

缺失标签个数返回相关度高的 6 个缺失标签 water、city、shore、sky、skyline、river 补全待完备图像，这 6 个缺失标签的相关度依次是 0.243803912575508、0.167959688853618、0.114437989264422、0.109764194990877、0.081306990876860、0.068776021817618。

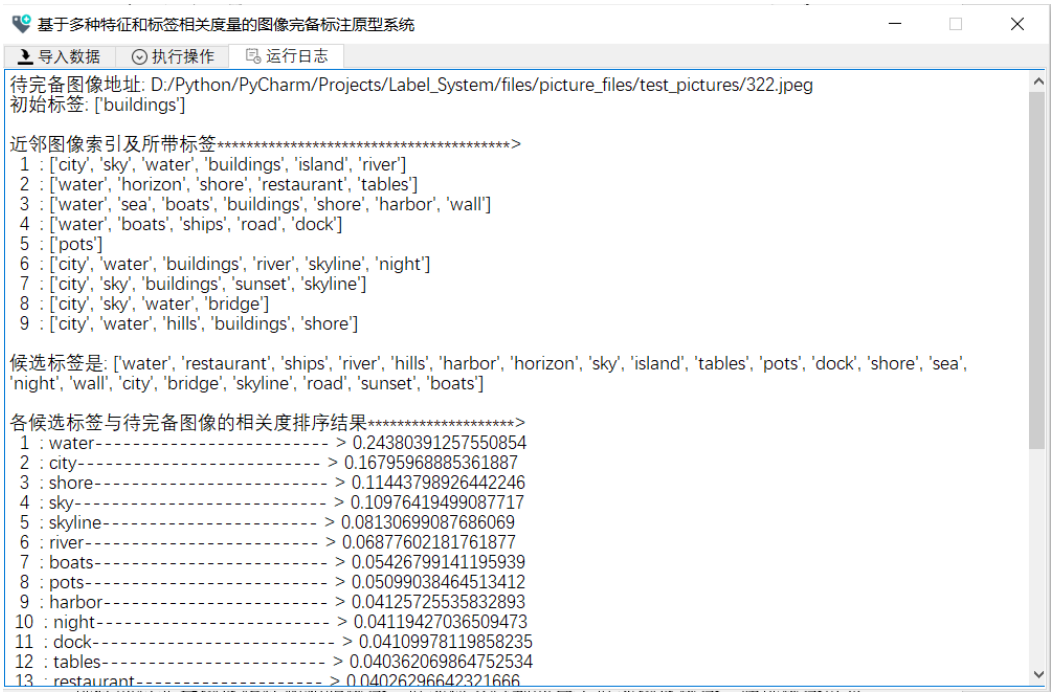

图 5.12 查看日志数据

5.3 本章小结

本章利用 Python 语言实现了第二章及第三章中提出的基于多种特征和标签相关度量的图像标签完备标注原型系统，并详细地展示了系统功能和体系结构等，最后基于 Corel5k 数据集演示了使用该系统补全待完备图像缺失标签的具体过程和运行结果。

结束语

随着网络中存在的图片数量不断增多以及图像检索需求的增加，自动完备图像标签以提高图像检索性能日益成为研究的重点内容，完备标注系统应用而生。本文提出了一种基于图像多种特征和标签相关度量的图像完备标注方法，首先利用图像的 SIFT 视觉特征和 CNN 视觉特征计算综合视觉相似度，并结合待完备图像的初始标签语义计算语义相似度，根据视觉相似度和语义相似度来查找待完备图像的近邻图像；然后将近邻图像所的标签作为候选标签，从候选标签与待完备图像的视觉相关度和语义相关度两个方面度量标签与图像相关度，选择出相关度高的候选标签作为待完备图像的缺失标签补全图像。本文的主要任务体现在以下几个方面：

（1）提取图像的 SIFT 视觉特征和 CNN 视觉特征，用欧氏距离计算图像的视觉相似度。在提取 SIFT 特征向量的过程中，使用 VLAD 算法将 SIFT 算法提取出的描述子矩阵聚合成一个特征向量，并使用 PCA 算法对该特征向量进行降维处理；在提取 CNN 特征向量的过程中，使用预先训练的 VGG16 模型提取 CNN 特征向量，并使用 PCA 算法对该特征向量进行降维处理。

（2）给出了在考虑标签间接相关性时度量图像之间语义相似度的方法。在计算两幅图像的语义相似度时，分别求两幅图像带有的各个标签之间的相关性，取均值作为两幅图像的语义相似度；在度量标签之间的相关性时，利用 Google 距离度量标签之间的直接相关性，通过构建标签的间接相关集来度量标签之间的间接相关性。

（3）在上述算法的基础上，以 Python 语言作为开发工具实现了基于多种特征和标签相关度量的图像标签完备标注原型系统，并实现了可视化的图形界面，对软件的功能模块和系统结构等进行了详细的描述。

本文实现的图像标签完备标注系统在补全待完备图像缺失标签方面取得了一定的效果，但也存在一些局限性供有待研究，如：

（1）要求待完备图像至少包含一个初始标签；

（2）在待完备图像的初始标签数量较多时，图像之间的语义相似度计算量会变得非常大等。

致谢

转眼间，在大学里已经度过了四个年头。在这四年里，感谢学校的培养让我系统地学习了计算机科学方面的相关知识，也感谢四年里遇到的各位老师对我的辛勤教导。

在毕业论文即将完成之际，首先感谢我的指导老师张素兰老师，在整个毕业论文的撰写过程中，从论文选题，到撰写开题报告，再到中期答辩，最后到论文定稿，整个过程中都不厌其烦地给予我重要指导，给予我及时的帮助，使我最后顺利完成论文的撰写工作，再次由衷地感谢她。

此外，感谢四年里的和我一同在这所校园里学习生活的各位同窗好友，感谢各位的陪伴给我的大学生活留下了美好的回忆。

最后，我要特别感谢我的父母，在我漫长的求学生涯中，他们的无私关爱、鼓励和支持，是我不断前进的力量源泉，至此十几年的寒窗苦读即将结束之时，谨向他们致以衷心的感谢。

参考文献

[1] 张素兰,郭平,张继福,等.图像语义自动标注及其粒度分析方法[J].自动化学报,2012,38(5):688-697.

[2] 黎健成,袁春,宋友.基于卷积神经网络的多标签图像自动标注[J].计算机科学,2016,43(7):41-45.

[3] 李雯莉,张素兰,张继福,等.基于卷积神经网络和概念格的图像语义完备标注[J].小型微型计算机系统,2020,41(9):1979-1986.

[4] 孟磊,张素兰,胡立华,等.基于低秩稀疏分解优化的图像标签完备[J].计算机辅助设计与图形学学报,2020,32(1):36-44.

[5] Wu L,Jin R,Jain A K. Tag completion for image retrieval[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence,2013,35(3): 716-727.

[6] Lin Z,Ding G,Hu M,et al. Image tag completion via image-specific and tag-specific linear sparse reconstructions[C]. IEEE Conference on Computer Vision and Pattern Recognition. Los Alamitos:IEEE Computer Society Press,2013: 1618-1625.

[7] Hou Y,Lin Z. Image tag completion and refinement by subspace clustering and matrix completion[C].Visual Communications and Image Processing.Los Alamitos:IEEE Computer Society Press,2015:1-4.

[8] Li X,Shen B,Liu B D,et al. A locality sensitive low-rank model for image tag completion[J]. IEEE Transactions on Multimedia,2016,18(3):474-483.

[9] Guillaumin M,Mensink T,Verbeek J,Schmid C.Tagprop:discriminative metric learning in nearest neighbor models for image auto-annotation[C].2009 IEEE 12th International Conference on Computer Vision,IEEE,2009:309-316.

[10] 杨凯婷.融合视觉特征和语义学习的图像标签完备标注[D].山西:太原科技大学,2021.

[11] Lowe D G.Distinctive image features from scale-invariant keypoints[J].International Journal of Computer Vision,2004,60(2):91-110.

[12] Cilibrasi R L,Vitanyi P M B. The Google similarity distance[J].IEEE Transactions on Knowledge and Data Engineering,2007,19(3):370-383.

[13] 田枫,沈旭昆.基于标签集相关性学习的大规模网络图像在线标注[J].自动化学报,2014,40(8): 1635-1643.

[14] 崔超然,马军.一种结合相关性和多样性的图像标签推荐方法[J].计算机学报,2013,36(3):654-663.

附录 I

英文原文

Distinctive Image Features from Scale-Invariant Keypoints

David G. Lowe

Abstract

This paper presents a method for extracting distinctive invariant features from images that can be used to perform reliable matching between different views of an object or scene. The features are invariant to image scale and rotation, and are shown to provide robust matching across a Jinn substantial range of affine distortion, change in 3D viewpoint, addition of noise, and change in illumination. The features are highly distinctive, in the sense that a single feature can be correctly matched with high probability against a large database of features from many images. This paper also describes an approach to using these features for object recognition. The recognition proceeds by matching individual features to a database of features from known objects using a fast nearest-neighbor algorithm, followed by a Hough transform to identify clusters belonging to a single object, and finally performing verification through least-squares solution for consistent pose parameters. This approach to recognition can robustly identify objects among clutter and occlusion while achieving near real-time performance.

Introduction

Image matching is a fundamental aspect of many problems in computer vision, including object or scene recognition, solving for 3D structure from multiple images, stereo correspondence, and motion tracking. This paper describes image features that have many properties that make them suitable for matching differing images of an object or scene. The features are invariant to image scaling and rotation, and partially invariant to change in illumination and 3D camera viewpoint. They are well localized in both the spatial and frequency domains, reducing the probability of disruption by occlusion, clutter, or noise. Large numbers of features can be extracted from typical images with efficient algorithms. In addition, the features are highly distinctive, which allows a single feature to be correctly matched with high probability against a large database of features, providing a basis for object and scene recognition.

The cost of extracting these features is minimized by taking a cascade filtering approach, in which the more expensive operations are applied only at locations that pass an initial test. Following are the major stages of computation used to generate the set of image features:

1. **Scale-space extrema detection**: The first stage of computation searches over all scales and image locations. It is implemented efficiently by using a difference-of-Gaussian function to identify potential interest points that are invariant to scale and orientation.
2. **Keypoint localization**: At each candidate location, a detailed model is fit to determine location and scale. Keypoints are selected based on measures of their stability.
3. **Orientation assignment**: One or more orientations are assigned to each keypoint location based on local image gradient directions. All future operations are performed on image data that has been transformed relative to the assigned orientation, scale, and location for each feature, thereby providing invariance to these transformations.

4. **Keypoint descriptor**: The local image gradients are measured at the selected scale in the region around each keypoint. These are transformed into a representation that allows for significant levels of local shape distortion and change in illumination.

This approach has been named the Scale Invariant Feature Transform (SIFT), as it transforms image data into scale-invariant coordinates relative to local features.

An important aspect of this approach is that it generates large numbers of features that densely cover the image over the full range of scales and locations. A typical image of size 500x500 pixels will give rise to about 2000 stable features (although this number depends on both image content and choices for various parameters). The quantity of features is particularly important for object recognition, where the ability to detect small objects in cluttered backgrounds requires that at least 3 features be correctly matched from each object for reliable identification.

For image matching and recognition, SIFT features are first extracted from a set of reference images and stored in a database. A new image is matched by individually comparing each feature from the new image to this previous database and finding candidate matching features based on Euclidean distance of their feature vectors. This paper will discuss fast nearest-neighbor algorithms that can perform this computation rapidly against large databases.

The keypoint descriptors are highly distinctive, which allows a single feature to find its correct match with good probability in a large database of features. However, in a cluttered image, many features from the background will not have any correct match in the database, giving rise to many false matches in addition to the correct ones. The correct matches can be filtered from the full set of matches by identifying subsets of keypoints that agree on the object and its location, scale, and orientation in the new image. The probability that several features will agree on these parameters by chance is much lower than the probability that any individual feature match will be in error. The determination of these consistent clusters can be performed rapidly by using an efficient hash table implementation of the generalized Hough transform.

Each cluster of 3 or more features that agree on an object and its pose is then subject to further detailed verification. First, a least-squared estimate is made for an affine approximation to the object pose. Any other image features consistent with this pose are identified, and outliers are discarded. Finally, a detailed computation is made of the probability that a particular set of features indicates the presence of an object, given the accuracy of fit and number of probable false matches. Object matches that pass all these tests can be identified as correct with high confidence.

Related research

The development of image matching by using a set of local interest points can be traced back to the work of Moravec (1981) on stereo matching using a corner detector. The Moravec detector was improved by Harris and Stephens (1988) to make it more repeatable under small image variations and near edges. Harris also showed its value for efficient motion tracking and 3D structure from motion recovery (Harris, 1992), and the Harris corner detector has since been widely used for many other image matching tasks. While these feature detectors are usually called corner detectors, they are not selecting just corners, but rather any image location that has large gradients in all directions at a predetermined scale.

The initial applications were to stereo and short-range motion tracking, but the approach was later extended to more difficult problems. Zhang et al. (1995) showed that it was possible to match Harris corners over a large image range by using a correlation window around each corner to select likely matches. Outliers

were then removed by solving for a fundamental matrix describing the geometric constraints between the two views of rigid scene and removing matches that did not agree with the majority solution. At the same time, a similar approach was developed by Torr (1995) for long-range motion matching, in which geometric constraints were used to remove outliers for rigid objects moving within an image.

The ground-breaking work of Schmid and Mohr (1997) showed that invariant local feature matching could be extended to general image recognition problems in which a feature was matched against a large database of images. They also used Harris corners to select interest points, but rather than matching with a correlation window, they used a rotationally invariant descriptor of the local image region. This allowed features to be matched under arbitrary orientation change between the two images. Furthermore, they demonstrated that multiple feature matches could accomplish general recognition under occlusion and clutter by identifying consistent clusters of matched features.

The Harris corner detector is very sensitive to changes in image scale, so it does not provide a good basis for matching images of different sizes. Earlier work by the author (Lowe, 1999) extended the local feature approach to achieve scale invariance. This work also described a new local descriptor that provided more distinctive features while being less sensitive to local image distortions such as 3D viewpoint change. This current paper provides a more in-depth development and analysis of this earlier work, while also presenting a number of improvements in stability and feature invariance.

There is a considerable body of previous research on identifying representations that are stable under scale change. Some of the first work in this area was by Crowley and Parker (1984), who developed a representation that identified peaks and ridges in scale space and linked these into a tree structure. The tree structure could then be matched between images with arbitrary scale change. More recent work on graph-based matching by Shokoufandeh, Marsic and Dickinson (1999) provides more distinctive feature descriptors using wavelet coefficients. The problem of identifying an appropriate and consistent scale for feature detection has been studied in depth by Lindeberg (1993, 1994). He describes this as a problem of scale selection, and we make use of his results below.

Recently, there has been an impressive body of work on extending local features to be invariant to full affine transformations (Baumberg, 2000; Tuytelaars and Van Gool, 2000; Mikolajczyk and Schmid, 2002; Schaffalitzky and Zisserman, 2002; Brown and Lowe, 2002). This allows for invariant matching to features on a planar surface under changes in orthographic 3D projection, in most cases by resampling the image in a local affine frame. However, none of these approaches are yet fully affine invariant, as they start with initial feature scales and locations selected in a non-affine-invariant manner due to the prohibitive cost of exploring the full affine space. The affine frames are Tutelars also more sensitive to noise than those of the scale-invariant features, so in practice the affine features have lower repeatability than the scale-invariant features unless the affine distortion is greater than about a 40 degree tilt of a planar surface (Mikolajczyk, 2002). Wider affine invariance may not be important for many applications, as training views are best taken at least every 30 degrees rotation in viewpoint (meaning that recognition is within 15 degrees of the closest training view) in order to capture non-planar changes and occlusion effects for 3D objects.

While the method to be presented in this paper is not fully affine invariant, a different approach is used in which the local descriptor allows relative feature positions to shift significantly with only small changes in the descriptor. This approach not only allows the descriptors to be reliably matched across a considerable range of affine distortion, but it also makes the features more robust against changes in 3D viewpoint for non-planar surfaces. Other advantages include much more efficient feature extraction and the ability to

identify larger numbers of features. On the other hand, affine invariance is a valuable property for matching planar surfaces under very large view changes, and further research should be performed on the best ways to combine this with non-planar 3D viewpoint invariance in an efficient and stable manner.

Many other feature types have been proposed for use in recognition, some of which could be used in addition to the features described in this paper to provide further matches under differing circumstances. One class of features are those that make use of image contours or region boundaries, which should make them less likely to be disrupted by cluttered backgrounds near object boundaries. Matas et al., (2002) have shown that their maximally-stable extremal regions can produce large numbers of matching features with good stability. Mikolajczyk et al., (2003) have developed a new descriptor that uses local edges while ignoring unrelated nearby edges, providing the ability to find stable features even near the boundaries of narrow shapes superimposed on background clutter. Nelson and Selinger (1998) have shown good results with local features based on groupings of image contours. Similarly, Pope and Lowe (2000) used features based on the hierarchical grouping of image contours, which are particularly useful for objects lacking detailed texture.

The history of research on visual recognition contains work on a diverse set of other image properties that can be used as feature measurements. Carneiro and Jepson (2002) describe phase-based local features that represent the phase rather than the magnitude of local spatial frequencies, which is likely to provide improved invariance to illumination. Schiele and Crowley (2000) have proposed the use of multidimensional histograms summarizing the distribution of measurements within image regions. This type of feature may be particularly useful for recognition of textured objects with deformable shapes. Basri and Jacobs (1997) have demonstrated the value of extracting local region boundaries for recognition. Other useful properties to incorporate include color, motion, figure-ground discrimination, region shape descriptors, and stereo depth cues. The local feature approach can easily incorporate novel feature types because extra features contribute to robustness when they provide correct matches, but otherwise do little harm other than their cost of computation. Therefore, future systems are likely to combine many feature types.

…

The local image descriptor

The previous operations have assigned an image location, scale, and orientation to each keypoint. These parameters impose a repeatable local 2D coordinate system in which to describe the local image region, and therefore provide invariance to these parameters. The next step is to compute a descriptor for the local image region that is highly distinctive yet is as invariant as possible to remaining variations, such as change in illumination or 3D viewpoint.

One obvious approach would be to sample the local image intensities around the keypoint at the appropriate scale, and to match these using a normalized correlation measure. However, simple correlation of image patches is highly sensitive to changes that cause misregistration of samples, such as affine or 3D viewpoint change or non-rigid deformations. A better approach has been demonstrated by Edelman, Intrator, and Poggio (1997). Their proposed representation was based upon a model of biological vision, in particular of complex neurons in primary visual cortex. These complex neurons respond to a gradient at a particular orientation and spatial frequency, but the location of the gradient on the retina is allowed to shift over a small receptive field rather than being precisely localized. Edelman et al. hypothesized that the function of these complex neurons was to allow for matching and recognition of 3D objects from a range of viewpoints. They have performed detailed experiments using 3D computer models of object and animal shapes which show

that matching gradients while allowing for shifts in their position results in much better classification under 3D rotation. For example, recognition accuracy for 3D objects rotated in depth by 20 degrees increased from 35% for correlation of gradients to 94% using the complex cell model. Our implementation described below was inspired by this idea, but allows for positional shift using a different computational mechanism.

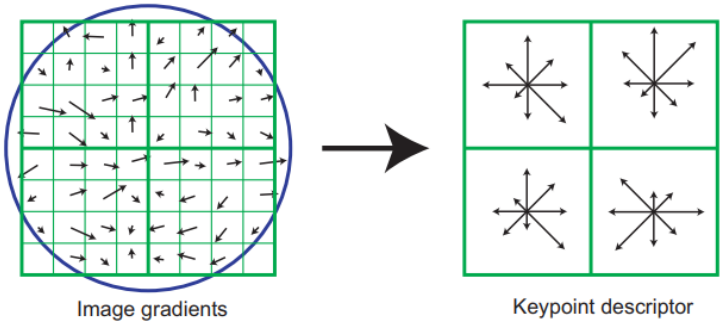

Figure 7

Figure 7 illustrates the computation of the keypoint descriptor. First the image gradient magnitudes and orientations are sampled around the keypoint location, using the scale of the keypoint to select the level of Gaussian blur for the image. In order to achieve orientation invariance, the coordinates of the descriptor and the gradient orientations are rotated relative to the keypoint orientation. For efficiency, the gradients are precomputed for all levels of the pyramid as described in Section 5. These are illustrated with small arrows at each sample location on the left side of Figure 7.

A Gaussian weighting function with σ equal to one half the width of the descriptor window is used to assign a weight to the magnitude of each sample point. This is illustrated with a circular window on the left side of Figure 7, although, of course, the weight falls off smoothly. The purpose of this Gaussian window is to avoid sudden changes in the descriptor with small changes in the position of the window, and to give less emphasis to gradients that are far from the center of the descriptor, as these are most affected by misregistration errors.

The keypoint descriptor is shown on the right side of Figure 7. It allows for significant shift in gradient positions by creating orientation histograms over 4×4 sample regions. The figure shows eight directions for each orientation histogram, with the length of each arrow corresponding to the magnitude of that histogram entry. A gradient sample on the left can shift up to 4 sample positions while still contributing to the same histogram on the right, thereby achieving the objective of allowing for larger local positional shifts.

It is important to avoid all boundary affects in which the descriptor abruptly changes as a sample shifts smoothly from being within one histogram to another or from one orientation to another. Therefore, trilinear interpolation is used to distribute the value of each gradient sample into adjacent histogram bins. In other words, each entry into a bin is multiplied by a weight of 1 − d for each dimension, where d is the distance of the sample from the central value of the bin as measured in units of the histogram bin spacing.

The descriptor is formed from a vector containing the values of all the orientation histogram entries, corresponding to the lengths of the arrows on the right side of Figure 7. The figure shows a 2×2 array of orientation histograms, whereas our experiments below show that the best results are achieved with a 4×4 array of histograms with 8 orientation bins in each. Therefore, the experiments in this paper use a 4×4×8 = 128 element feature vector for each keypoint.

Finally, the feature vector is modified to reduce the effects of illumination change. First, the vector is normalized to unit length. A change in image contrast in which each pixel value is multiplied by a constant

will multiply gradients by the same constant, so this contrast change will be canceled by vector normalization. A brightness change in which a constant is added to each image pixel will not affect the gradient values, as they are computed from pixel differences. Therefore, the descriptor is invariant to affine changes in illumination. However, non-linear illumination changes can also occur due to camera saturation or due to illumination changes that affect 3D surfaces with differing orientations by different amounts. These effects can cause a large change in relative magnitudes for some gradients, but are less likely to affect the gradient orientations. Therefore, we reduce the influence of large gradient magnitudes by thresholding the values in the unit feature vector to each be no larger than 0.2, and then renormalizing to unit length. This means that matching the magnitudes for large gradients is no longer as important, and that the distribution of orientations has greater emphasis. The value of 0.2 was determined experimentally using images containing differing illuminations for the same 3D objects.

…

中文翻译

来自尺度不变量关键点的独特的图像特征

戴维·罗威

摘要

本文提出了一种从图像中提取独特的不变特征的方法，该方法可用于在物体或场景的不同视图之间进行可靠的匹配。这些特征对图像的比例和旋转是不变的，并显示出在相当大的范围内，在仿生扭曲、三维视角的变化、噪声的增加和光照的变化中提供了稳健的匹配。这些特征具有高度的独特性，即单一特征可以与来自许多图像的大型特征数据库进行高概率的匹配。本文还描述了一种使用这些特征进行物体识别的方法。识别过程中，使用快速近邻算法将单个特征与已知物体的特征数据库进行匹配，然后使用 Hough 变换来识别属于单个物体的集群，最后通过最小二乘法解决一致的姿势参数来进行验证。这种识别方法可以在杂波和遮挡中稳健地识别物体，同时实现接近实时的性能。

介绍

图像匹配是计算机视觉中许多问题的一个基本方面，包括物体或场景识别、从多个图像中解决三维结构、立体对应和运动跟踪。本文描述了具有许多特性的图像特征，这些特性使它们适合于匹配一个物体或场景的不同图像。这些特征对图像的缩放和旋转是不变的，对光照和三维摄像机视角的变化也有部分不变性。它们在空间和频率领域都有很好的定位，减少了被遮挡、杂波或噪声破坏的可能性。大量的特征可以通过有效的算法从典型图像中提取出来。此外，这些特征具有高度的独特性，这使得单个特征可以与大型特征数据库进行高概率的匹配，为物体和场景识别提供了基础。

通过采取级联过滤的方法，提取这些特征的成本降到最低，其中较昂贵的操作只应用于通过初始测试的位置。以下是用于生成图像特征集的主要计算阶段：

1.尺度空间极值检测：第一阶段的计算在所有尺度和图像位置上进行搜索。它通过使用高斯差函数来识别对比例和方向不变的潜在兴趣点，从而有效地实现。

2.关键点的定位：在每一个候选地点，都要进行详细的模型拟合以确定位置和规模。

关键点的选择是基于对其稳定性的衡量。

3.方向分配：根据当地的图像梯度方向，为每个关键点位置分配一个或多个方向。所有未来的操作都是在图像数据上进行的，这些数据已经相对于为每个特征分配的方向、比例和位置进行了转换，从而提供了对这些转换的不变性。

4.关键点描述符：在每个关键点周围的区域内，以选定的比例测量局部图像梯度。这些都被转化为一种表示方法，允许局部形状失真和光照变化的显著水平。

这种方法被命名为尺度不变特征转换（SIFT），因为它将图像数据转换为相对于局部特征的尺度不变的坐标。

这种方法的一个重要方面是，它产生了大量的特征，这些特征密集地覆盖了图像的全部比例和位置。一个典型的 500x500 像素的图像将产生大约 2000 个稳定的特征（尽管这个数字取决于图像内容和各种参数的选择）。特征的数量对于物体识别尤其重要，在杂乱的背景中检测小物体的能力要求每个物体至少有 3 个特征被正确匹配，以实现可靠的识别。

对于图像匹配和识别，首先从一组参考图像中提取 SIFT 特征并存储在数据库中。通过将新图像中的每个特征与之前的数据库进行单独比较，并根据其特征向量的欧几里得距离找到候选的匹配特征，来匹配新的图像。本文将讨论快速的近邻算法，该算法可以针对大型数据库快速进行这种计算。

关键点描述符具有高度的独特性，这使得单个特征能够在一个大的特征数据库中以良好的概率找到其正确匹配。然而，在一个杂乱的图像中，许多来自背景的特征在数据库中没有任何正确的匹配，除了正确的匹配外，还产生了许多错误的匹配。正确的匹配可以通过识别与物体及其在新图像中的位置、比例和方向一致的关键点子集从全部匹配中过滤出来。几个特征在这些参数上偶然一致的概率远远低于任何单个特征匹配出错的概率。通过使用广义 Hough 变换的有效哈希表实现，可以快速确定这些一致的集群。

每个由 3 个或更多的特征组成的群组，如果与一个物体及其姿势一致，就会接受进一步的详细验证。首先，对物体姿势的仿生近似进行最小平方估计。任何与该姿势一致的其他图像特征都被识别出来，并将异常值丢弃。最后，考虑到拟合的准确性和可能的错误匹配的数量，对某一特定特征集表示物体存在的概率进行详细计算。通过所有这些测试的物体匹配可以被确定为正确的，具有很高的可信度。

相关研究成果

通过使用一组局部兴趣点进行图像匹配的发展可以追溯到 Moravec（1981）使用角落检测器进行立体匹配的工作。Harris 和 Stephens（1988）对 Moravec 检测器进行了改进，使其在小的图像变化和靠近边缘的情况下更具有可重复性。Harris 还展示了其在高效运动跟踪和运动恢复的三维结构方面的价值（Harris, 1992），此后 Harris 角检测器被广泛用于许多其他图像匹配任务。虽然这些特征检测器通常被称为角落检测器，但它们并不是只选择角落，而是选择任何在预定比例的各个方向上有大梯度的图像位置。

最初的应用是立体和短距离运动跟踪，但该方法后来被扩展到更困难的问题。后来被扩展到更困难的问题。Zhang 等人（1995）表明，通过使用每个角周围的相关窗口来选择可能的匹配，有可能在很大的图像范围内匹配 Harris 的角。每个角的相关窗口来选择可能的匹配。然后，通过解决描述刚性场景的两个视图之间的几何约束的基本矩阵来去除异常值，并去除与之不一致的匹配。删除与大多数解决方案不一致的匹配。同时，一个类似的 Torr(1995)为长距离运动匹配开发了一个类似的方法，在这个方法中，几何

约束被用来去除异常值。在该方法中，几何约束被用来去除图像中移动的刚性物体的异常值。

Schmid 和 Mohr(1997)的开创性工作表明，不变的局部特征匹配可以扩展到一般的图像识别问题中，在这些问题中，一个特征与一个大型的图像数据库相匹配。他们也使用 Harris 角来选择兴趣点，但不是用相关窗口来匹配，而是使用局部图像区域的旋转不变的描述符。这使得特征可以在两幅图像之间的任意方向变化下被匹配。此外，他们证明了在闭塞和杂乱的情况下，多个特征匹配可以通过识别匹配特征的一致集群来完成一般的识别。

Harris 角落检测器对图像比例的变化非常敏感，所以它不能为不同尺寸的图像的匹配提供良好的基础。作者早期的工作(Lowe, 1999)扩展了局部特征方法以实现尺度不变性。这项工作还描述了一种新的局部描述符，它提供了更独特的特征，同时对局部图像失真（如三维视角变化）不那么敏感。本文对这一早期工作进行了更深入的发展和分析，同时也提出了一些稳定性和特征不变性方面的改进。

以前有相当多的研究是关于识别在尺度变化下稳定的表征的。Crowley 和 Parker(1984)在这一领域最早开展了一些工作，他们开发了一种表征，可以识别比例空间中的山峰和山脊，并将它们连接成一个树状结构。然后，该树状结构可以在具有任意比例变化的图像之间进行匹配。Shokoufandeh、Marsic 和 Dickinson(1999)在基于图的匹配方面的最新工作提供了使用小波系数的更有特色的特征描述符。Lindeberg(1993, 1994)深入研究了为特征检测确定一个合适的、一致的尺度的问题。他将此描述为尺度选择问题，我们在下文中利用了他的成果。

最近，在扩展局部特征以使其不受全仿生变换的影响方面有大量的工作（Baumberg, 2000; Tuytelaars 和 Van Gool, 2000; Mikolajczyk 和 Schmid, 2002; Schaffalitzky 和 Zisserman, 2002; Brown 和 Lowe,2002）。这允许在正交三维投影的变化下对平面上的特征进行不变的匹配，在大多数情况下是通过在局部仿生框架中重新取样图像。然而，这些方法都不是完全的仿生不变的，因为由于探索整个仿生空间的成本过高，它们从以非仿生不变的方式选择的初始特征尺度和位置开始。仿生框架对噪声也比标度不变的特征更敏感，所以在实践中，仿生特征的重复性比标度不变的特征低，除非仿生变形大于平面的 40 度倾斜(Mikolajczyk,2002)。更广泛的仿生不变性对许多应用来说可能并不重要，因为训练视图最好至少每 30 度旋转一次视角（意味着识别在最接近的训练视图的 15 度之内），以便捕捉非平面变化和三维物体的遮挡效应。

虽然本文要介绍的方法不完全是仿生不变的，但采用了一种不同的方法，其中局部描述符允许相对特征位置大幅移动，而描述符只有很小的变化。这种方法不仅允许描述符在相当大的仿生变形范围内被可靠地匹配，而且还使特征对非平面表面的三维视点的变化更加坚固。其他优点包括更有效的特征提取和识别更多特征的能力。另一方面，仿生不变性对于在非常大的视图变化下匹配平面表面是一个有价值的属性，应该进一步研究如何以高效和稳定的方式将其与非平面三维视点不变性结合起来的最佳方法。

许多其他的特征类型已经被提议用于识别，其中一些可以在本文描述的特征之外使用，以在不同的情况下提供进一步的匹配。一类特征是那些利用图像轮廓或区域边界的特征，这应该使它们不太可能被物体边界附近杂乱的背景所干扰。Matas 等人已经表明，他们的最大稳定极值区域可以产生大量具有良好稳定性的匹配特征。Mikolajczyk 等人开发了一种新的描述符，它使用局部边缘而忽略附近不相关的边缘，提供了即使在叠加在背景杂波上的狭窄形状的边界附近也能找到稳定特征的能力。Nelson 和 Selinger 用基

于图像轮廓分组的局部特征显示了良好的效果。同样，Pope 和 Lowe 使用了基于图像轮廓的分层分组的特征，这对缺乏详细纹理的物体特别有用。

视觉识别的研究历史包含了一系列可作为特征测量的其他图像属性的工作。Carneiro 和 Jepson 描述了基于相位的局部特征，它代表了局部空间频率的相位而不是幅度，这可能会提供更好的光照不变性。Schiele 和 Crowley 提出使用多维直方图来总结图像区域内的测量分布。这种类型的特征可能对识别具有可变形形状的纹理物体特别有用。Basri 和 Jacobs 证明了提取局部区域边界用于识别的价值。其他可纳入的有用属性包括颜色、运动、人物-地面辨别、区域形状描述符和立体深度线索。局部特征方法可以很容易地纳入新的特征类型，因为额外的特征在提供正确的匹配时有助于提高鲁棒性，但除计算成本外几乎没有其他危害。因此，未来的系统可能会结合许多特征类型。

……

局部图像描述符

前面的操作已经为每个关键点分配了一个图像位置、比例和方向。这些参数规定了一个可重复的本地二维坐标系统，用来描述本地图像区域，因此提供了这些参数的不变性。下一步是为本地图像区域计算一个描述符，该描述符具有高度的独特性，同时尽可能不受其余变化的影响，例如光照或三维视角的变化。

一个明显的方法是以适当的比例对关键点周围的局部图像强度进行采样，并使用归一化的相关度来匹配这些图像。然而，图像斑块的简单关联对导致样本错误注册的变化非常敏感，例如仿生或三维视点变化或非刚性变形。Edelman, Intrator, and Poggio (1997) 已经证明了一种更好的方法。他们提出的表示方法是基于生物视觉的模型，特别是初级视觉皮层的复杂神经元。这些复杂的神经元对特定方向和空间频率的梯度作出反应，但梯度在视网膜上的位置被允许在一个小的接受区域内移动，而不是被精确定位。埃德尔曼等人假设，这些复杂的神经元的功能是允许从一系列视点对三维物体进行匹配和识别。他们使用物体和动物形状的三维计算机模型进行了详细的实验，这些实验表明，在允许其位置移动的同时，匹配梯度的结果是在三维旋转下有更好的分类。例如，深度旋转 20 度的三维物体的识别精度从梯度相关的 35%提高到使用复杂细胞模型的 94%。我们下面描述的实施方案是受这一想法的启发，但允许使用不同的计算机制进行位置移动。

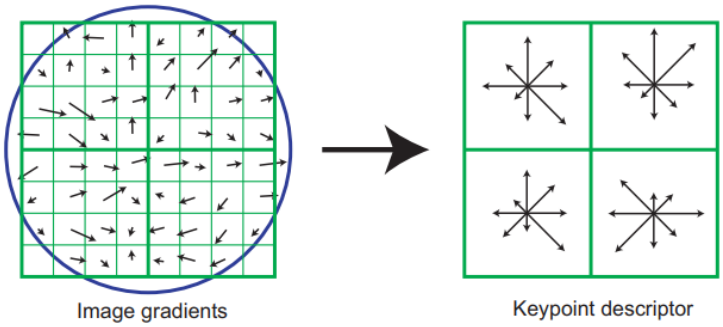

图 7

图 7 说明了关键点描述符的计算过程。首先，图像梯度大小和方向在关键点位置周围被取样，利用关键点的比例来选择图像的高斯模糊程度。为了实现方向不变性，描述符的坐标和梯度方向相对于关键点方向进行旋转。为了提高效率，如第 5 节所述，对金字塔的所有层次的梯度进行了预计算。在图 7 左侧的每个样本位置上都有小箭头说明了这些。

一个 σ 等于描述符窗口宽度一半的高斯加权函数被用来给每个样本点的幅度分配一个权重。图 7 左边的圆形窗口说明了这一点，当然，权重是平滑下降的。这个高斯窗口的目的是为了避免描述符因窗口位置的微小变化而发生突然变化，并对远离描述符中心的梯度给予较少的重视，因为这些梯度受错位误差的影响最大。

关键点描述符显示在图 7 的右侧。它通过在 4×4 的样本区域上创建方向直方图来实现梯度位置的显著转移。图中显示了每个方向直方图的八个方向，每个箭头的长度对应于该直方图条目的大小。左边的一个梯度样本最多可以移动 4 个样本位置，同时仍对右边的同一直方图有贡献，从而达到允许较大的局部位置移动的目的。

重要的是要避免所有的边界影响，即当一个样本从一个直方图内平滑地转移到另一个直方图内或从一个方向转移到另一个方向时，描述符会突然改变。因此，三线插值被用来将每个梯度样本的值分配到相邻的直方图槽中。换句话说，进入一个仓的每个条目都乘以每个维度的权重 1-d，其中 d 是样本与仓的中心值的距离，以直方图仓间距的单位衡量。

描述符是由一个包含所有方向直方图条目值的向量形成的，对应于图 7 右边的箭头的长度。图中显示的是一个 2×2 的方向直方图阵列，而我们下面的实验表明，最好的结果是用一个 4×4 的直方图阵列，每个直方图中有 8 个方向仓。因此，本文的实验对每个关键点使用 4×4×8=128 个元素的特征向量。

最后，特征向量被修改以减少光照变化的影响。首先，该向量被归一化为单位长度。图像对比度的变化，即每个像素值乘以一个常数，将使梯度乘以相同的常数，所以这种对比度的变化将被矢量归一化所抵消。亮度的变化，即每个图像像素被加上一个常数，不会影响梯度值，因为它们是由像素差异计算出来的。因此，该描述符对照度的仿生变化是不变的。然而，由于相机的饱和度或由于影响不同方向的三维表面的光照变化，也会发生非线性光照变化。这些影响会导致一些梯度的相对大小发生较大的变化，但影响梯度方向的可能性较小。因此，我们通过对单位特征向量中的数值进行阈值处理，使其不大于 0.2，然后重新归一化为单位长度，来减少大梯度的影响。这意味着对大梯度的匹配不再那么重要，而对方向的分布则更加重视。0.2 的值是通过实验确定的，使用的图像包含相同三维物体的不同光照。

……

附录 II

程序代码

本文使用 Python 语言进行数据处理，Python 版本号为 3.7，所有程序运行过程均在 PyCharm 集成开发环境中进行，程序的主体代码主要包含 8 个模块，分别是：

1.SIFT 特征提取模块 (sift_process.py)
2.CNN 特征提取模块 (cnn_process.py)
3.图像与图像的视觉相似度模块 (visual_similarity.py)
4.图像与图像的语义相似度模块 (label_similarity.py)
5.特征提取和近邻检索模块 (near_picture.py)
6.标签与图像的语义相关度模块 (label_to_label.py)
7.标签与图像的视觉相关度模块 (label_to_picture.py)
8.完备标注模块 (aim_label.py)

1. SIFT 特征提取模块代码 (sift_process.py)

```
# -*- coding:utf-8- -*-
import cv2
import numpy as np
from do_sift import my_io
from sklearn.decomposition import PCA
from sklearn.externals import joblib

# 相关参数
initial_picture_nums = 4500  # 已完备的图片数量
clusters_nums = 260  # 聚类中心个数
epochs = 23  # k-means 聚类迭代次数
pca_feature_sift = 0.95  # pca 降维后保留的特征百分比

# sift 算法提取描述子
def sift_fun(path, show=False):
    img = cv2.imread(path)  # 读取图片
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)  # 获取灰度图像
    sift = cv2.xfeatures2d.SIFT_create()  # sift 特征提取
    keyPoint, descriptor = sift.detectAndCompute(gray, None)  # 计算特征点和描述子（numpy.ndarray 类型）
    marked_img = cv2.drawKeypoints(gray, keyPoint, img)  # 在灰度图像上标注特征点
    if show:  # 是否显示标注后的图像
        cv2.imshow("drawKeyPoints", marked_img)
        cv2.waitKey(0)
        cv2.destroyAllWindows()
    return descriptor  # 返回描述子

# 计算并保存所有已完备图片特征向量
def get_sift_all_vector():
    # 生成由所有已完备图片的 sift 描述子组成的超大矩阵
    initial_name_path = "D:/Python/PyCharm/Projects/Label_System/files/data_files/initial_picture_name.csv"
    all_name = my_io.readCsv(initial_name_path, all=True).squeeze(axis=1)
    all_des = np.empty(shape=[0, 128])
    img_des_len = []  # 记录每张图片的描述子数量
    for name in all_name:
        # 提取一张图片的描述子
        des = sift_fun("D:/Python/PyCharm/Projects/Label_System/files/picture_files/initial_pictures/{}.jpeg".form
```

```
at(name))
            all_des = np.concatenate([all_des, des])
            img_des_len.append(len(des))
        print("1.描述子矩阵已经生成...")
        # 聚类，训练出整个已完备图片的码本，即聚类中心组成的数组
        criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 0.01)  # 定义终止条件
        des_set = np.float32(all_des)
        # cv2.k-means 返回三个参数，分别是：
        # compactness     集聚性，每个特征到相应聚类中心的平方和
        # labels          结果标识，每个特征所属分组的序号,np.array
        # centers         聚类中心，由聚类中心组成的数组,np.array
        compactness, labels, centers = cv2.kmeans(des_set, clusters_nums, None, criteria, epochs, cv2.KMEANS_RANDOM_CENTERS)

        my_io.saveCsv("D:/Python/PyCharm/Projects/Label_System/code/do_sift/codeBook.csv", centers)  # 保存码本
        print("2.聚类结束，码本已保存...")
        # 计算所有已完备图片的各个聚合特征向量组成的 sift 特征矩阵
        cursor = 0  # 记录切片时的截断点
        all_vlad_vector = []  # 记录所有聚合特征向量
        codeBook = centers  # 码本
        # 计算所有图片的聚合特征向量（未降维）
        for eachImage in img_des_len:
            # 截取一张图片的描述子矩阵
            descrips = all_des[cursor: cursor + eachImage]
            centriods_id = labels[cursor: cursor + eachImage]
            centriods = codeBook[centriods_id]
            # clusters_nums*128 的全 0 矩阵
            vlad = np.zeros(shape=[clusters_nums, 128])
            # 计算聚合特征向量
            for eachDes in range(eachImage):
                vlad[centriods_id[eachDes]] = vlad[centriods_id[eachDes]] + descrips[eachDes] - centriods[eachDes]
            cursor += eachImage  # 更新截断点
            # 展平
            vlad = vlad.reshape(1, -1).squeeze(axis=0)
            # 归一化处理
            vlad_norm = vlad.copy()
            cv2.normalize(vlad, vlad_norm, 1.0, 0.0, cv2.NORM_L2)
            all_vlad_vector.append(vlad_norm)  # 加入记录，4500*(clusters_nums*128)
        print("3.VLAD 聚合结果已经生成...矩阵形状为{}".format(np.shape(all_vlad_vector)))  # 4500*33280
        # 主成分分析，降维
        pca = PCA(n_components=pca_feature_sift)  # 使得降维后保留 95%的特征
        pca_all_vector = pca.fit_transform(np.array(all_vlad_vector))  # 训练模型，返回降维结果
        joblib.dump(pca, 'sift_pca_model.m')  # 保存模型
        print("4.PCA 降维完成，降维模型已保存...")
        # 再次归一化
        pca_all_vector_norm = []
        for pca_one_vector in pca_all_vector:
            # 逐条归一化
            pca_one_vector_norm = np.array(pca_one_vector).copy()
            cv2.normalize(pca_one_vector, pca_one_vector_norm, 1.0, 0.0, cv2.NORM_L2)
            # 加入列表

            pca_all_vector_norm.append(pca_one_vector_norm)
        pca_all_vector_norm =
```

```
np.array(pca_all_vector_norm)
    print("5.正在保存 sift 特征矩阵...")
    # 保存 sift 特征矩阵

my_io.saveCsv("D:/Python/PyCharm/Projects/Label_System/code/do_sift/pca_all_vector_sift.csv", pca_all_vector_norm)
    return pca_all_vector_norm

# 获取待完备图片的特征向量(1*3598)
def get_sift_one_vector(test_picture_path):
    # 计算待完备图像的 sift 描述子
    one_des = sift_fun(test_picture_path)
    # 读取码本
    codeBook = my_io.readCsv("D:/Python/PyCharm/Projects/Label_System/code/do_sift/codeBook.csv", all=True)
    codeBook = codeBook.astype('float64')
    # 计算待完备图像的特征向量
    vlad = np.zeros(shape=[clusters_nums, 128])
    for eachDes in range(np.shape(one_des)[0]):
        # 找出最近的类中心
        des = one_des[eachDes]
        min_dist = 1000000000.0
        ind = 0
        for i in range(clusters_nums):
            dist = np.linalg.norm(des - codeBook[i])  # 各元素平方和开根号
            if dist < min_dist:  # 找出最近的类中心
                min_dist = dist  # 更新最近距离
                ind = i  # 记录类中心
        # 记录残差
        vlad[ind] = vlad[ind] + des - codeBook[ind]
    # 展平
    not_pca_one_vector = vlad.reshape(1, -1).squeeze(axis=0)
    # 归一化
    not_pca_one_vector_norm = not_pca_one_vector.copy()
    cv2.normalize(not_pca_one_vector, not_pca_one_vector_norm, 1.0, 0.0, cv2.NORM_L2)
    # 主成分分析，降维
    vector = np.array(not_pca_one_vector).reshape(1, -1)  # 添加一个维度使之成为二维才能进行降维
    pca = joblib.load("D:/Python/PyCharm/Projects/Label_System/code/do_sift/sift_pca_model.m")  # 读取已经训练好的模型
    pca_vector = pca.transform(vector)  # 降维
    pca_vector = np.array(pca_vector).squeeze(axis=0)
    # 再次归一化
    one_pca_vector = np.array(pca_vector).copy()
    cv2.normalize(pca_vector, one_pca_vector, 1.0, 0.0, cv2.NORM_L2)
    return one_pca_vector
```

2. CNN 特征提取模块代码 (cnn_process.py)

```
import numpy as np
# from keras.applications.vgg16 import VGG16
# from keras.applications.vgg16 import preprocess_input
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input
from sklearn.decomposition import PCA
from sklearn.externals import joblib
from do_cnn import my_io
import cv2
import imutils
```

```
# 相关参数
input_picture_size = (128, 128)  # 设置 CNN 输入层图片大小
pca_feature_cnn = 0.95

# 自定义裁剪图片
def resize_crop_image(image, target_width, target_height):
    (h, w) = image.shape[:2]
    dH = 0
    dW = 0
    if w < h:
        image = imutils.resize(image, width=target_width, inter=cv2.INTER_AREA)
        dH = int((image.shape[0] - target_height) / 2.0)
    else:
        image = imutils.resize(image, height=target_height, inter=cv2.INTER_AREA)
        dW = int((image.shape[1] - target_width) / 2.0)
    (h, w) = image.shape[:2]
    image = image[dH:h - dH, dW:w - dW]
    return cv2.resize(image, (target_width, target_height), interpolation=cv2.INTER_AREA)

# 获取整个图片库的 cnn 特征矩阵(4500*2204)
def get_cnn_all_vector():
    # 读取模型权重
    path = "D:/Python/PyCharm/Projects/Label_System/code/do_cnn/vgg16_model_weights.h5"
    model = VGG16(weights=path, include_top=False)
    # 提取特征并保存
    initial_name_path = "D:/Python/PyCharm/Projects/Label_System/files/data_files/initial_picture_name.csv"
    all_name = my_io.readCsv(initial_name_path, all=True)
    all_name = all_name.squeeze(axis=1)
    all_vector = []  # 记录所有特征向量（未降维）
    for eachPicture in all_name:
        img_path = "D:/Python/PyCharm/Projects/Label_System/files/picture_files/initial_pictures" \

"/{}.jpeg".format(eachPicture)
        image = cv2.imread(img_path)
        img_data = resize_crop_image(image, input_picture_size[0], input_picture_size[1])  # 自定义裁剪,使得保留最大特征
        img_data = np.expand_dims(img_data, axis=0)
        img_data = preprocess_input(img_data)
        features = model.predict(img_data)
        vector = np.array(features).reshape(1, -1).squeeze(axis=0)  # 展平
        # 归一化
        vector_norm = vector.copy()
        cv2.normalize(vector, vector_norm, 1.0, 0.0, cv2.NORM_L2)
        all_vector.append(vector_norm)
    all_vector = np.array(all_vector).astype('float64')  # 列表转为数组
    # 主成分分析，降维
    pca = PCA(n_components=pca_feature_cnn)  # 使得降维后可以保留 95%的特征
    all_vector_pca = pca.fit_transform(all_vector)  # 训练模型，并返回降维后的矩阵
    joblib.dump(pca, 'cnn_pca_model.m')  # 保存模型
    # 再次归一化
    all_vector_pca_norm = np.array(all_vector_pca).copy()
    cv2.normalize(all_vector_pca, all_vector_pca_norm, 1.0, 0.0, cv2.NORM_L2)
    # 保存 cnn 训练矩阵

my_io.saveCsv("D:/Python/PyCharm/Projects/Label_System/files/running_files/all_picture_pca_vlad_vector"
```

```
                "/all_vector_cnn.csv",
all_vector_pca_norm)
    return all_vector_pca_norm

# 获取单张图片的 cnn 特征向量(1*2204)
def get_cnn_one_vector(one_picture_path):
    # 读取模型权重
    path =
"D:/Python/PyCharm/Projects/Label_System/code/
do_cnn/vgg16_model_weights.h5"
    model = VGG16(weights=path,
include_top=False)
    # 提取特征
    image = cv2.imread(one_picture_path)
    img_data = resize_crop_image(image,
input_picture_size[0], input_picture_size[1])  #
自定义裁剪,使得保留最大特征
    img_data = np.expand_dims(img_data,
axis=0)
    img_data = preprocess_input(img_data)
    features = model.predict(img_data)
    vector = np.array(features).reshape(1, -
1).squeeze(axis=0)  # 展平(1*8192)
    # 归一化
    vector_norm = vector.copy()
    cv2.normalize(vector, vector_norm, 1.0, 0.0,
cv2.NORM_L2)
    # 主成分分析，降维
    vector_norm = vector_norm.reshape(1, -1)  #
(1*1*8192)
    pca =
joblib.load("D:/Python/PyCharm/Projects/Label_S
ystem/code/do_cnn/cnn_pca_model.m")  # 读取
已经训练好的 pca 模型
    pca_vector = pca.transform(vector_norm)  #
降维
    pca_vector =
np.array(pca_vector).squeeze(axis=0)
    # 再次归一化
    pca_vector_norm = pca_vector.copy()
    cv2.normalize(pca_vector, pca_vector_norm,
1.0, 0.0, cv2.NORM_L2)
    return pca_vector_norm
```

3. 图像与图像的视觉相似度计算模块 (visual_similarity.py)

```
from compare_label_visual import my_io
import numpy as np
import math

# 计算两个向量的欧氏距离
def calculate_euclidean(aim_picture, one_demo):
    dist = np.sqrt(np.sum(np.square(aim_picture -
one_demo)))
    return dist

# 计算待完备图像与 4500 张图片的视觉相似度
def get_visual_similarity(all_vector, test_vector):
    # 处理待完备图像已经生成的特征向量
    test_vector = np.array(test_vector)
    test_vector = test_vector.astype('float64')
    # 处理整个图片库已经生成的特征矩阵
    all_vector = np.array(all_vector)
    all_vector = all_vector.astype('float64')
    # 计算待完备图像和 4500 张图片的视觉相
似度
    visual_similarity = []  # 1*4500
    for i in range(0, len(all_vector)):
        dist = calculate_euclidean(test_vector,
all_vector[i])
        visual_similarity.append(math.exp(-
dist))
    # 数据规范化
    visual_similarity =
(np.array(visual_similarity)-
min(visual_similarity))/(max(visual_similarity)-
min(visual_similarity))
    return np.array(visual_similarity)
```

4. 图像与图像的语义相似度计算模块 (label_similarity.py)

```
import numpy as np
from compare_label_visual import my_io

all_picture_nums = 4500
entrance_ex_set = 0.7

# 计算两个标签的共现频率
def get_both_label_co(aim_id, one_id,
all_initial_label):
    # 读取标签矩阵
    all_initial_label =
np.transpose(all_initial_label)
    temp_vector = all_initial_label[aim_id] +
all_initial_label[one_id]
    up = list(temp_vector).count(2)
    down_1 =
list(all_initial_label[aim_id]).count(1)
    down_2 =
list(all_initial_label[one_id]).count(1)
    return (up / down_1 + up / down_2) / 2

# 构建标签间接相关集
def get_expanded_labelSet(label_id,
all_initial_vector):
    all_initial_vector =
np.transpose(all_initial_vector)
    label_vector = all_initial_vector[label_id]  #
label_id 所在向量
    label_id_count = list(label_vector).count(1)
    expanded_labelSet = []  # 记录扩展的相关
标签
    for i in range(0,
np.shape(all_initial_vector)[0]):
        if i != label_id:
            temp = list(label_vector +
all_initial_vector[i])
            if temp.count(2)/label_id_count >
entrance_ex_set:  # 某共现频率大于
entrance_ex_set
                expanded_labelSet.append(i)
# 记录标签 id
    return expanded_labelSet  # 记录了间接相
关度的标签 id

# 求两个标签的谷歌距离
def get_two_label_google_dist(the_label_id,
another_label_id, all_label_vector):
    all_label_vector =
np.transpose(all_label_vector)
    the_label_id_count =
np.sum(all_label_vector[the_label_id])
    if the_label_id_count == 0:
        the_label_id_count = 0.000000001
    another_label_id_count =
np.sum(all_label_vector[another_label_id])
    if another_label_id_count == 0:
        another_label_id_count = 0.000000001
    both_label_id_count = 0
    for e in
(np.array(all_label_vector[the_label_id])+np.array(
all_label_vector[another_label_id])):
        if e == 2:
            both_label_id_count += 1
    if both_label_id_count == 0:
        both_label_id_count = 0.000000001
    up = max(np.log10(the_label_id_count),
np.log10(another_label_id_count) -
np.log10(both_label_id_count))
    down = np.log10(all_picture_nums) -
min(np.log10(the_label_id_count),
np.log10(another_label_id_count))
    dist = up / down  # 得到两标签的 google
距离
    return dist

# 计算两个标签的直接相关度
```

```
def get_two_label_direct_sim(the_label_id,
another_label_id, all_label_vector):
    dist =
get_two_label_google_dist(the_label_id,
another_label_id, all_label_vector)
    return np.exp(-dist)

# 计算两个标签的间接相关度
def get_two_label_indirect_sim(the_label_id,
another_label_id, all_label_vector):
    # 1.分别构建两个标签的扩展标签集
    ex_labet_set_1 =
get_expanded_labelSet(the_label_id,
all_label_vector)
    ex_labet_set_2 =
get_expanded_labelSet(another_label_id,
all_label_vector)
    # 2.返回与扩展标签集中标签直接相关度的
最大值
    indirect_sim_list = [0, 0]
    if ex_labet_set_1:  # 扩展标签集 1 不为空
        sim = []
        for indirect_id in ex_labet_set_1:

sim.append(get_two_label_direct_sim(indirect_id,
another_label_id, all_label_vector))
        indirect_sim_list[0] = max(sim)
    if ex_labet_set_2:  # 扩展标签集 2 不为空
        sim = []
        for indirect_id in ex_labet_set_2:

sim.append(get_two_label_direct_sim(the_label_id
, indirect_id, all_label_vector))
        indirect_sim_list[1] = max(sim)
    # 3.求平均
    indirect_sim = np.sum(indirect_sim_list)/2
# 两标签的间接相关度
    return indirect_sim

# 计算两标签的最终语义相关度
def get_two_label_final_sim(the_id, another_id,
all_label_vector):
    direct_sim =
get_two_label_direct_sim(the_id, another_id,
all_label_vector)
    # indirect_sim =
get_two_label_indirect_sim(the_id, another_id,
all_label_vector)
    # return (direct_sim + indirect_sim)/2
    return direct_sim

# 计算 sigmoid 权重
def get_sigmoid_weight(aim_id, all_label_vector):
    all_label_vector =
np.transpose(all_label_vector)
    count =
list(all_label_vector[aim_id]).count(1)
    sigmoid_weight = 1 / (1 + (count /
all_picture_nums))
    return sigmoid_weight

# 返回待完备图像和某一张图的语义相关度
（初始标列号，某一张图所带标签列号，初始
标签矩阵）
def
get_bothPicture_label_similarity(aim_picture_col_
id, one_demo_col_id, initial_all_label):
    up = 0
    for id_1 in aim_picture_col_id:
        # 加入初始标签的 sigmoid 权重
        sigmoid_weight =
get_sigmoid_weight(id_1, initial_all_label)
        for id_2 in one_demo_col_id:
            up +=
get_two_label_final_sim(id_1, id_2,
initial_all_label) * sigmoid_weight
    down = np.size(aim_picture_col_id) *
np.size(one_demo_col_id)
    label_similarity = up/down
    return label_similarity

# 返回待完备图像和所有图像的语义相关度
def get_label_similarity(test_path):
    # 解析路径，取出待完备图片名称
```

```
        temp = str(test_path).split('/')
        test_name = str(str(temp[np.size(temp)-
1]).split('.')[0])
        # 获取待完备图像的标签向量
        test_all_name = my_io.readCsv(

"D:/Python/PyCharm/Projects/Label_System/files/
data_files/test_picture_name.csv", all=True)
        test_all_name =
test_all_name.squeeze(axis=1)
        aim_picture_id =
list(test_all_name).index(test_name)
        test_all_label = my_io.readCsv(

"D:/Python/PyCharm/Projects/Label_System/files/
data_files/test_picture_label_eigenmatrix.csv",
all=True)
        aim_label_vector =
test_all_label[aim_picture_id]
        aim_picture_col_id = []
        for i in range(np.size(aim_label_vector)):   #
找出待完备图片所对应标签的列索引
            if aim_label_vector[i] == '1':
                aim_picture_col_id.append(i)
        # 处理待完备图像和训练集图片的关系
        initial_all_label = my_io.readCsv(
"D:/Python/PyCharm/Projects/Label_System/files/
data_files/initial_picture_label_eigenmatrix.csv",
all=True)
        initial_all_label = initial_all_label.astype('int')
        label_similarity = []   # 1*4500
        for count in range(4500):
            # 获取训练集中某图片的标签
            one_demo_vector =
initial_all_label[count]
            one_demo_col_id = []
            for i in
range(np.size(one_demo_vector)):   # 找出训练
集中某图片所对应标签的列索引
                if one_demo_vector[i] == 1:
                    one_demo_col_id.append(i)

label_similarity.append(get_bothPicture_label_sim
ilarity(aim_picture_col_id, one_demo_col_id,
initial_all_label))
        # 规范化处理
        label_similarity = (np.array(label_similarity)-
min(label_similarity))/(max(label_similarity)-
min(label_similarity))
        return label_similarity
```

5. 特征提取和近邻检索模块（near_picture.py)

```
import numpy as np
from PyQt5.QtCore import QThread, pyqtSignal
from compare_label_visual import my_io,
visual_similarity, label_similarity
from do_sift import sift_process
from do_cnn import cnn_process
from picture_label import label
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

# 近邻检索
class get_near_picture(QThread):
        signal_result = pyqtSignal(list, list)
        signal_str = pyqtSignal(str)

        def __init__(self, test_path,
img_img_visual_weight, num, sift_cnn_weight,
all_sift, all_cnn, test_sift, test_cnn):
            super(get_near_picture,
self).__init__(parent=None)
            self.test_path = test_path
            self.img_img_visual_weight =
img_img_visual_weight
            self.num = num
```

```
            self.sift_cnn_weight = sift_cnn_weight
            self.all_vector_sift = all_sift
            self.all_vector_cnn = all_cnn
            self.test_vector_sift = test_sift
            self.test_vector_cnn = test_cnn
            self.context = ""

        def run(self):
            self.signal_str.emit("正在计算视觉相似度")
            visualSimilarity_sift = visual_similarity.get_visual_similarity(self.all_vector_sift, self.test_vector_sift)
            visualSimilarity_cnn = visual_similarity.get_visual_similarity(self.all_vector_cnn, self.test_vector_cnn)
            visualSimilarity = visualSimilarity_sift * self.sift_cnn_weight + visualSimilarity_cnn * (1-self.sift_cnn_weight)
            visualSimilarity = (visualSimilarity-min(visualSimilarity))/(max(visualSimilarity)-min(visualSimilarity))
            self.signal_str.emit("正在计算语义相似度")
            labelSimilarity = label_similarity.get_label_similarity(self.test_path)
            self.signal_str.emit("正在计算综合相似度")
            similarity = np.array(labelSimilarity) * (1 - self.img_img_visual_weight) + np.array(visualSimilarity) * self.img_img_visual_weight
            similarity = (similarity - min(similarity)) / (max(similarity) - min(similarity))
            self.signal_str.emit("正在查找近邻图像")
            picture_index = list(np.arange(0, 4500, 1))
            temp = dict(zip(picture_index, list(similarity)))  # 生成键值对
            # 找出近邻图像的 id
            near_picture_id = []
            for i in range(self.num):
                max_id = max(temp, key=temp.get)
                near_picture_id.append(max_id)
                del temp[max_id]  # 根据 key 删除键值对
            # 找出近邻图像名称
            near_picture_names = []
            all_name = my_io.readCsv("D:/Python/PyCharm/Projects/Label_System/files/data_files"
                                     "/initial_picture_name.csv", all=True)
            all_name = all_name.squeeze(axis=1)
            for p_id in near_picture_id:
                near_picture_names.append(all_name[p_id])
            context = "\n 近邻图像索引及所带标签**********>\n"
            for i in range(0, np.size(near_picture_names)):
                context = context + "{0:>3}  : {1}\n".format(i+1, label.get_theInitialPicture_all_label(near_picture_names[i]))
            context += "\n 候选标签是: {}\n".format(self.get_label(near_picture_names))
            self.context = context
            self.signal_str.emit("近邻检索完成")

            self.signal_result.emit(list(near_picture_names), list(visualSimilarity))
            return

        def get_label(self, all_near_names):
            # 提取候选标签
            candidate_labels = set()  # 创建空集合，记录候选标签
            all_near_picture_label = []  # 记录每个近邻图像所带有的标签
            for name in all_near_names:
                labs =
```

```
label.get_theInitialPicture_all_label(name)  # 取得一张待完备图像的候选标签
                all_near_picture_label.append(labs)
                for one_label in labs:

candidate_labels.add(one_label)  # 将标签加入集合，重复的标签自动忽略
            # 从候选标签中去掉初始标签，得到真正的候选标签集合
            for e in label.get_theTestPicture_all_label(self.test_path):
                candidate_labels.discard(e)  # 从去掉初始标签，不存在不报错
            candidate_labels = list(candidate_labels)
            return candidate_labels

        def stop(self):
            self.terminate()

# 特征提取
class get_visual_feature(QThread):
        signal_str = pyqtSignal(str)
        signal_result = pyqtSignal(list, list, list, list)

        def __init__(self, test_path, cnn_input_imgSize, cnn_cut, sift_cnn_weight):
            super(get_visual_feature, self).__init__(parent=None)
            self.test_path = test_path
            self.cnn_input_imgSize = cnn_input_imgSize  # 元组
            self.cnn_cut = cnn_cut # bool
            self.sift_cnn_weight = sift_cnn_weight # double

        def run(self):
            self.signal_str.emit("正在读取 SIFT 特征矩阵")
            all_vector_sift = \
my_io.readCsv("D:/Python/PyCharm/Projects/Label_System/code/do_sift/pca_all_vector_sift.csv", all=True)

            self.signal_str.emit("正在读取 CNN 特征矩阵")
            all_vector_cnn = \
my_io.readCsv("D:/Python/PyCharm/Projects/Label_System/code/do_cnn/pca_all_vector_cnn.csv", all=True)

            self.signal_str.emit("正在提取 SIFT 特征")
            test_vector_sift = sift_process.get_sift_one_vector(self.test_path)

            self.signal_str.emit("正在提取 CNN 特征")
            test_vector_cnn = cnn_process.get_cnn_one_vector(self.test_path)
            # 发射提取结果
            self.signal_str.emit("特征提取完成")

self.signal_result.emit(list(all_vector_sift), list(all_vector_cnn),
list(test_vector_sift), list(test_vector_cnn))

        def stop(self):
            self.terminate()
```

6. 标签的图像的语义相似度模块 (label_to_label.py)

```
# 从（候选标签-初始标签）角度量化
from find_aim_label import my_io
from compare_label_visual.label_similarity import *
```

```
# 计算（候选标签-待完备图像）的语义相关度
def get_LP_label_relevancy(test_picture_label, candidate_labels):
    # 读取相关文件
    all_labels = my_io.readCsv("D:/Python/PyCharm/Projects/Label_System"
"/files/data_files/label.csv").squeeze(axis=1)  # 获取标签词
    all_initial_label_vector = my_io.readCsv("D:/Python/PyCharm/Projects/Label_System"
"/files/data_files/initial_picture_label_eigenmatrix.csv")
    all_initial_label_vector = all_initial_label_vector.astype('int')  # 这里不需要提前转置
    # 找出初始标签 id
    test_id = []  # 记录初始标签 id
    for one_test_label in test_picture_label:
        test_id.append(list(all_labels).index(one_test_label))
    # 计算初始标签与各个候选标签的语义相关度
    LP_label_relevancy = []
    for one_candidate_label in candidate_labels:
        candidate_id = list(all_labels).index(one_candidate_label)  # 找出候选标签 id
        all_test_label_rel = []  # 为了找出最大的
        for one_test_id in test_id:
            rel = (get_two_label_direct_sim(candidate_id, one_test_id, all_initial_label_vector)
                   + get_two_label_indirect_sim(candidate_id, one_test_id, all_initial_label_vector))/2
            all_test_label_rel.append(rel)
    LP_label_relevancy.append(np.max(all_test_label_rel))  # 候选标签与初始标签相关度最大的作为结果
    return LP_label_relevancy
```

7. 标签与图像的视觉相似度模块 (label_to_picture.py)

```
# 从（候选标签-待完备图像）角度量化
from find_aim_label import my_io
from compare_label_visual.label_similarity import *

# 计算（候选标签-待完备图像）的视觉相关度
def get_LP_visual_relevancy(test_picture_label, candidate_labels, similarity):
    # 读取相关文件
    all_labels = my_io.readCsv("D:/Python/PyCharm/Projects/Label_System"
"/files/data_files/label.csv").squeeze(axis=1)  # 获取标签词
    all_initial_label_vector = my_io.readCsv("D:/Python/PyCharm/Projects/Label_System"
"/files/data_files/initial_picture_label_eigenmatrix.csv")  # 训练集标签
    all_initial_label_vector = all_initial_label_vector.astype('int')  # 这里也不需要提前转置
    # 找出初始标签 id
    test_id = []  # 记录初始标签 id
    for one_test_label in test_picture_label:
```

```
test_id.append(list(all_labels).index(one_test_label
))
    # 计算各个候选标签和待完备图像的视觉
相似度
    LP_visual_relevancy = []
    all_vector =
np.transpose(all_initial_label_vector)  # 标签矩
阵转置，便于获取向量
    for one_candidate_label in candidate_labels:
        # 构建含有候选标签和最多初始标签
的集合 I
        one_id =
list(all_labels).index(one_candidate_label)  # 获
取候选标签 id
        I = get_pictureSet(one_id, test_id,
all_vector)
        up = 0
        for one_I_id in I:  # 对集合 I 中各图
片和待完备图片的相似度求和
            up = up + similarity[one_I_id]
        LP_visual_relevancy.append(up /
np.size(I))  # 得到该候选标签与待完备图像的
视觉相似度，加入记录
    return LP_visual_relevancy

# 构建含有最多{候选标签,初始标签，初始标
签的间接相关集内的标签}的图片集合
def get_pictureSet(one_id, test_id, all_vector):
    all_vector = np.transpose(all_vector)
    ## 计算初始标签的间接相关集内的所有
标签
    # ex_label_ids = []
    # for test_id in test_ids:
    #     ex_label_ids +=
get_expanded_labelSet(test_id, all_label_vector)
    # ex_label_ids = set(ex_label_ids)  # 一定
不会存在 test_ids，不需要对其去重
    # ex_label_ids.discard(one_id)  # 去除候选
标签，不存在不报错
    # ex_label_ids = list(ex_label_ids)
    # all_ids = [one_id] + test_ids + ex_label_ids
# 存储所有标签
    picture_set = []  # 存放图片 id
    if np.size(test_id) == 2:  # 两个初始标签
        count = list(all_vector[one_id] +
all_vector[test_id[0]] +
all_vector[test_id[1]]).count(3)  # 3 出现的个数
        if count == 0:
            count_0 = list(all_vector[one_id] +
all_vector[test_id[0]]).count(2)  # 看候选标签和
第一个初始标签
            if count_0 == 0:
                count_1 =
list(all_vector[one_id] +
all_vector[test_id[1]]).count(2)  # 看候选标签和
第二个初始标签
                if count_1 == 0:  # 只剩候选
标签所在向量
                    vector =
all_vector[one_id]
                    for num in range(0,
np.size(vector)):
                        if vector[num] == 1:

picture_set.append(num)  # 添加记录
                else:  # 候选标签和第二个
初始标签
                    vector =
all_vector[one_id] + all_vector[test_id[1]]
                    for num in range(0,
np.size(vector)):
                        if vector[num] == 2:

picture_set.append(num)  # 添加记录
            else:  # 候选标签和第一个初始
标签
                vector = all_vector[one_id] +
all_vector[test_id[0]]
                for num in range(0,
np.size(vector)):
                    if vector[num] == 2:

picture_set.append(num)  # 添加记录
        else:  # 候选标签和两个初始标签
```

```
                vector = all_vector[one_id] +
all_vector[test_id[0]] + all_vector[test_id[1]]
                for num in range(0,
np.size(vector)):
                    if vector[num] == 3:
                        picture_set.append(num)
# 添加记录
    else:  # 一个初始标签
        count = list(all_vector[one_id] +
all_vector[test_id[0]]).count(2)
        if count == 0:  # 只剩候选标签所在向
量
            vector = all_vector[one_id]
            for num in range(0,
np.size(vector)):
                if vector[num] == 1:
                    picture_set.append(num)
# 添加记录
        else:
            vector = all_vector[one_id] +
all_vector[test_id[0]]
            for num in range(0,
np.size(vector)):
                if vector[num] == 2:
                    picture_set.append(num)
# 添加记录
    return picture_set
```

8. 完备标注模块（aim_label.py)

```
# 补全待完备图像
import numpy as np
from PyQt5.QtCore import QThread, pyqtSignal
from find_aim_label import my_io, label_to_label,
label_to_picture
from picture_label import label
import time

# 完备标注
class get_lacking_label(QThread):
    signal_result = pyqtSignal(list)
    signal_str = pyqtSignal(str)

    def __init__(self, test_picture_label,
all_near_names, similarity, num, weight):
        super(get_lacking_label,
self).__init__(parent=None)
        self.test_picture_label =
test_picture_label
        self.all_near_names = all_near_names
        self.similarity = similarity
        self.num = num
        self.weight = weight
        self.context = ""

    def run(self):
        # 提取候选标签
        candidate_labels = set()  # 创建空集
合，记录候选标签
        all_near_picture_label = []  # 记录每
个近邻图像所带有的标签
        for name in self.all_near_names:
            labs =
label.get_theInitialPicture_all_label(name)  # 取
得一张待完备图像的候选标签
            all_near_picture_label.append(labs)
            for one_label in labs:

candidate_labels.add(one_label)  # 将标签加入
集合，重复的标签自动忽略
        # 从候选标签中去掉初始标签，得到
真正的候选标签集合
        for e in self.test_picture_label:
            candidate_labels.discard(e)  # 从
去掉初始标签，不存在不报错
        candidate_labels = list(candidate_labels)
        show_candidate_labels =
label.get_string_label(candidate_labels)  # 得到
```

```
候选标签字符串

            # 计算语义相关度
            self.signal_str.emit("正在计算候选标签和待完备图像的语义相关度")
            LP_label_relevancy = label_to_label.get_LP_label_relevancy(self.test_picture_label, candidate_labels)
            # 计算视觉相关度
            self.signal_str.emit("正在计算候选标签和待完备图像的视觉相关度")
            time.sleep(1)
            LP_visual_relevancy = label_to_picture.get_LP_visual_relevancy(self.test_picture_label, candidate_labels, self.similarity)
            # 提取缺失标签
            self.signal_str.emit("正在计算缺失标签")
            time.sleep(1)
            # 计算近邻集中含有某标签的图片个数占比
            near_weight = []
            for one in candidate_labels:
                count = 0
                for one_set in all_near_picture_label:
                    if one in one_set:
                        count += 1

                near_weight.append(count/np.size(self.all_near_names))
            # 计算最终的相似度
            LP_relevancy = np.array(LP_visual_relevancy) * self.weight + np.array(LP_label_relevancy) * (1 - self.weight)
            for i in range(0, np.size(near_weight)):
                LP_relevancy[i] = LP_relevancy[i] * near_weight[i]
            # 选择出 value 最大的几个标签返回
            LP_relevancy = dict(zip(candidate_labels, LP_relevancy))  # 合并为键值对，key=label_name，value=LP_relevancy
            lacking_label = []
            lacking_labels_relevancy = []
            context = "\n 各候选标签与待完备图像的相关度排序结果********************>\n"
            for i in range(0, len(LP_relevancy)):
                max_label = max(LP_relevancy, key=LP_relevancy.get)  # 找出值最大的标签返回
                context += "{0:>3}  : {1:-<30} > {2}\n".format(i+1, max_label, LP_relevancy[max_label])
                lacking_label.append(max_label)  # 记录标签

                lacking_labels_relevancy.append(LP_relevancy[max_label])  # 记录标签的相关度
                del LP_relevancy[max_label]  # 根据 key 删除键值对
            self.context = context
            lacking_label = lacking_label[0:min(self.num, np.size(lacking_label))]
            # lacking_labels_relevancy = lacking_labels_relevancy[0:min(self.num, len(lacking_labels_relevancy))]
            self.signal_str.emit("缺失标签补全完成")
            time.sleep(0.4)
            self.signal_result.emit(lacking_label)

    def stop(self):
        self.terminate()
```