

华中科技大学

模式识别课程设计[报告]

花卉识别

院 系 人工智能与自动化学院

专业班级 自动化卓越计划实验班 1601

组 员 夏坤 (U201614457)

组 员 张弛 (U201614455)

组 员 尚颖超 (U201614460)

组 员 曾子恒 (U201614465)

2020 年 3 月 3 日

摘要

花卉识别是一个在日常生活中十分常见的任务，自然界中有数以万计的花卉种类，用机器学习的方式进行花卉识别可以很好地帮助植物学、园艺领域的研究。但是花卉识别同时又是极具挑战性的任务，主要的难点有：1) 不同的花之间可能存在相似的颜色、形状或者纹理；2) 同一种花之间可能因为拍摄角度、光线和背景等因素而存在很大差异；3) 同一只花在生长过程中会随着时间改变而产生不同。在花卉识别任务中，计算机视觉和机器学习通常使用颜色、形状和纹理三种特征的融合去作为图像的特征用于分类，分类主要使用支撑向量机(Support Machine learning, SVM) 和随机森林(Random Forest, RF) 等分类器；随着深度学习兴起，卷积神经网络在图像分类上具有很好的性能和很快的响应受到广泛使用；对于训练样本不足的情况，迁移学习将其他相近领域的知识迁移到花卉识别中指导分类，在训练样本有限的情况下可以达到很高的分类正确率。当然在原始图像提取特征之前，对图像的预处理也必不可少，在花卉识别任务中主要应用分割算法将花朵与背景分离，从而避免背景的干扰。除此之外，花卉识别的另一难点在于自然界中仍然存在或者被培育的当前没有被发现的新品种，对于这种情形，零样本学习(Zero-shot Learning)方法应运而生。最后我们通过在所给的数据集、Oxford 17 花卉数据集和 Oxford 102 花卉数据集上测试得到实验结果，比较各种方法的优劣。

关键词：花卉识别；图像分割；深度学习；迁移学习

Abstract

Flower classification is a very common task in daily life, there are thousands of species of flowers in the nature, which using machine learning approaches to categorize flowers can beneficially foster the research on botany and horticulture. But simultaneously it is an challenging problem, which difficulties is mainly on: 1) different flowers may exist the same features such as color, shape or texture; 2) Flowers of one specie may be different because of shooting angle, light and background; 3) A flower may vary with the growth and time passing. In the flower classification task, computer vision and machine learning usually uses the fusion of three features of color, shape and texture as the features of the image for classification. The classification mainly uses Support Vector Machine and Random Forest as classifier. With the rise of deep learning, convolutional neural networks have excellent performance in image classification and fast response is widely used. For the case of insufficient training samples, transfer learning will transfer knowledge from other similar fields to flower recognition to guide classification, and can achieve high classification accuracy when training samples are limited. Of course, before the original image is extracted, the preprocessing of the iamge is also essential. In the flower recognition task, the segmentation algorithm is mainly used to separate the flower from the background, thereby avoiding background interference. Besides, another difficulty in flower recognition is the new varieties that are still not found in nature or are currently being cultivated. For this situation, Zero-shot Learning has emerged. Finally, we compare the pros and cons of various methods by testing the experimental results on the given dataset, the Oxford 17 flower dataset, and the Oxford 102 flower dataset.

Key Words: Flower classification;Image Segmentation;Deep Learning;Transfer Learning

目 录

摘要	I
Abstract	III
1 课题概述	1
1.1 花卉识别的目的和意义	1
1.2 花卉识别的发展现状	1
1.3 花卉识别的难点	2
1.4 问题描述	2
2 算法分析	3
2.1 需求分析	3
2.1.1 完成目标	3
2.1.2 数据来源	3
2.1.3 软件需求	4
2.2 研究方案设计	4
2.3 图像分割	5
2.3.1 基于 HSV 特征的图像分割方法	5
2.3.2 基于 OTSU 的颜色分割	6
2.4 特征提取	6
2.4.1 基于视觉词汇的特征提取	6
2.4.2 基于融合特征描述子的特征提取	9
2.4.3 基于稀疏编码的 SIFT 特征	12
2.4.4 基于分割的分形纹理分析	13
2.5 机器学习分类算法	16
2.5.1 支持向量机	16
2.5.2 随机森林	17
2.5.3 极限学习机	18
2.6 深度神经网络	21
2.6.1 卷积神经网络概述	22
2.6.2 卷积神经网络的反向传播过程	23

2.6.3 常用的卷积神经网络	24
2.6.4 Snapshot Ensembles	26
3 实验系统设计	28
3.1 总体算法流程图	28
3.2 方案一：基于视觉词汇的花卉识别方法	29
3.2.1 算法流程	29
3.2.2 程序模块说明	30
3.3 方案二：基于融合特征描述子的花卉识别方法	30
3.3.1 算法流程	30
3.3.2 程序模块说明	31
3.4 方案三：基于 Sparse-SIFT 的花卉识别方法	32
3.4.1 算法流程	32
3.4.2 程序模块说明	33
3.5 方案四：基于 SFTA 的花卉识别方法	33
3.5.1 算法流程	33
3.5.2 程序模块说明	34
3.6 方案五：基于卷积神经网络的花卉识别方法	35
3.7 方案六：基于迁移学习的卷积神经网络的花卉识别方法	35
3.8 方案七：基于全卷积神经网络和迁移学习的花卉识别方法	36
4 实验结果分析	38
4.1 图片实验示例	38
4.1.1 图片分割示例	38
4.1.2 图片识别示例	39
4.2 实验结果及其分析	40
4.2.1 基于手动特征提取的花卉识别方法	40
4.2.2 深度学习方法	44
5 总结与展望	47
参考文献	49
附录 A 补充的图像处理知识	52
A.1 SIFT 特征提取算子	52

A.2 LBP 算子	55
A.3 Canny 算子	56
附录 B 神经网络简介	57
B.1 神经网络概述	57
B.2 反向传播算法	57
附录 C 代码	59
C.1 文件夹: test	59
C.2 文件夹: fsvm	68
C.3 文件夹: fSVMoRF	82
C.4 文件夹: fRemote	94

1 课题概述

1.1 花卉识别的目的和意义

在日常生活中，各式各样的花朵随处可见。世界上大约有 250000 种已经命名的花卉。大多数人每天都看见花，但是很少有人可以确定花的种类，他们只有通过向专家咨询或者上网搜索获取花朵的种类信息。但是专家不可能时时刻刻接受人们的询问，最直接的做法就是通过拍摄花的图像来自动识别花的种类^[1]。这种花卉识别系统嵌入手机当中，可以帮助人们了解更多的花卉知识；安装在无人机上，可以通过飞行获取一片区域的更精细的花卉分布情况，用于衡量一个地区的生态环境；嵌入到一些教学软件中，可以帮助学生和研究者更方便快捷地了解一株花朵的相关属性和知识。可以看出，自动的花卉识别系统的用途十分广泛，有十分重要的现实意义。

1.2 花卉识别的发展现状

花卉识别任务是一种经典的图像识别任务。在植物学领域研究花卉识别系统是一个重要的话题。植物分类是植物学研究的基础。自从 18 世纪一种层级植物分类系统被 Carl Linneaus 提出，迄今为止仍在全世界广泛使用。一开始，分类算法只能区分 8000 种植物，到现在已经可以识别 369000 种不同的植物。在花卉识别的传统方法中，植物学家首先观察花的生活习性，然后学习全局特征和形态学结构特征，最终与已记载的植物门类进行比较，确定花的种类。但是这种分类方法需要大量的专业知识，所以，发展自动花卉识别系统对非专业人士十分有帮助^[2]。近些年来，一些机器学习方法被提出来用于花卉识别。Nilsback 和 Zisserman^[3] 提出一种基于视觉词汇的特征提取方式，将颜色、形状和纹理特征融合成为图像的特征，然后用 K 近邻算法完成分类；Saitoh 和 Kaneko^[4] 提出一种针对野生花朵的自动识别系统，他们同时使用花和叶子的图像来识别花卉；Guru 和他的同事^[5] 引入一种使用 K 近邻分类器的自动花卉识别模型；Kanan 和 Cottrell^[6] 使用一种模型，将确定视觉注意序列与稀疏编码结合，可以通过在自然图像上应用无监督学习方法获得基于生物学启发的滤波器；Yoo 等^[7] 提出一种多尺度金字塔池化方法，从而更好地利用从预训练好的卷积神经网络。

1.3 花卉识别的难点

花卉识别的难点在于花卉的种类很多，很多种花很难单纯从一种角度或者特征空间去分辨：

- 不同的花之间的区别在部分情况下不是十分显著，有的时候不同种的花可能十分相似；
- 同一种花也存在差异特别大的情况，有的品种的花颜色各异、有的品种的花因为拍摄角度、光照等原因造成差异；
- 同一株花在不同的生长时期呈现出的差异性仍十分显著，会误导分类器造成正确率下降；
- 我们迄今为止仍没有完全认知所有花的种类，所以很可能会遇到新的没有命名的新品种，这时传统的分类方法无能为力。

1.4 问题描述

数据集中包含 6 种花朵的图片，每个文件夹中存储有对应于文件夹名的花卉图像数据。在训练阶段，对于每类花卉随机算则 40 个样本为训练样本，其余样本为测试样本，计算识别的正确率。如此反复进行 5 轮测试，最终得到 5 轮测试正确率的均值和标准差。

上述简要介绍了花卉识别的一些基本概况和课题的问题描述，第二章将介绍几种花卉识别算法；第三章简述实验系统的设计方案和模块设计；第四章展示实验结果并做简要分析，最后总结全文。附录部分是本次课程设计的实现代码。

2 算法分析

2.1 需求分析

2.1.1 完成目标

通过给定的花卉数据集，训练出一个达到一定性能的花卉分类器，对于一幅给定的花卉图片，通过设计的花卉分类识别系统判断图片中的花卉种类，输出最后的判断结果。

2.1.2 数据来源

本次课程设计用到的数据集主要有三个，说明如下：

- 问题描述中提供的花卉数据集，数据集中只有 6 类数据，每类包含 80 张图片^①；
- Oxford 17 花卉数据集，数据集包含 17 类花卉图片，每类包含 80 张图片。拍摄的花卉是英国一些常见的品种。图像有很大的尺度、姿势和光线变化^②；
- Oxford 102 花卉数据集，包含 8200 张花卉图片，102 种花卉，每一类有 40 到 258 张图像^③。

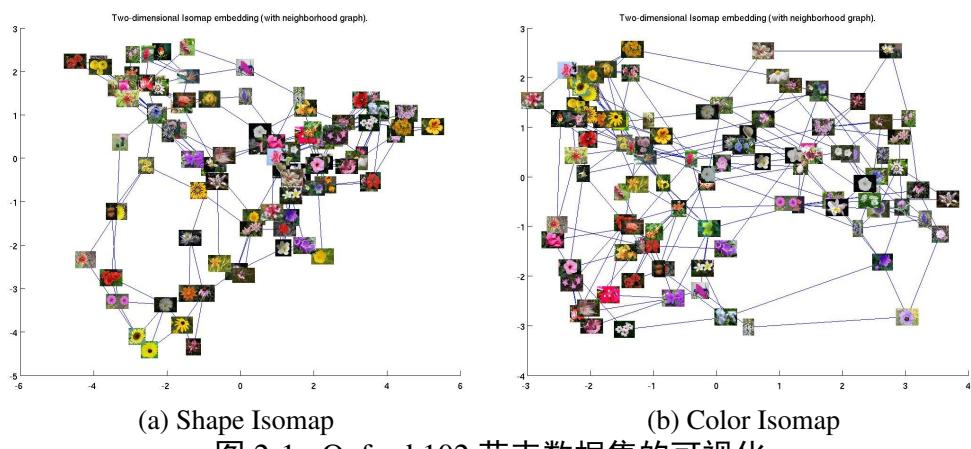


图 2-1 Oxford 102 花卉数据集的可视化

上述图片引用自 <http://www.robots.ox.ac.uk/~vgg/data/flowers/102/index.html>，通过 SIFT 特征和 HSV 颜色空间特征来描述花卉类别的分布。

^①<http://pan.baidu.com/s/1nt6xUVB>

^②<http://www.robots.ox.ac.uk/~vgg/data/flowers/17/index.html>

^③<http://www.robots.ox.ac.uk/~vgg/data/flowers/102/index.html>

2.1.3 软件需求

基于 Python3.7.5 平台编译，需要用的第三方库有：numpy、matplotlib、torch、opencv-python(3.4.2.16)、scipy 等。

2.2 研究方案设计

对于一个普通的图像识别任务，通常由图像分割、特征提取和分类识别三个部分组成。图像分割是指将图像的前景或感兴趣的区域从图像中提取出来，与图像中的其他区域分割开来以屏蔽无关区域或背景对后续操作的干扰；特征提取是指运用图像处理和计算机视觉的知识对图像的特征进行合理描述，并生成可用于分类器使用的具有良好可分性的特征向量；分类识别是指将已经提取好的特征向量运用机器学习的算法对其进行分类，已得到分类结果，完成图像识别任务。

花卉识别是一类典型的图像识别任务，在本次课程设计中我们同样采取上述框架，先对花卉图像进行分割取得花卉图像前景，在通过特征提取的方法获得特征向量，最后用机器学习的分类算法完成分类识别任务，具体过程如下图：



图 2-2 花卉识别的分类框架

图 2-2 中所述的方法是本次课程设计我们所采用的方法，我们采用多种算法进行花卉识别，进行比较和分析来认识不同图像处理算法和机器学习的特性和优缺点，同时也可以通过本次课程设计拓宽知识面，增强编程和阅读文献的技能。通过阅读文献和复现文章代码，可以增进对文献中的算法的理解；通过对传统图像识别算法与深度学习进行比较可以体会到深度学习的强大性能和易用性。后面几节会逐个介绍图 2-2 中所介绍的方法。

2.3 图像分割

2.3.1 基于 HSV 特征的图像分割方法

图像分割的目的是为了将花朵从背景中提取出来。可以观察到颜色的均值和标准差是一个图像的重要指标^[8]。花朵图片通常包含高密度的颜色值。鲜艳的色彩吸引着鸟类和蜜蜂等授粉媒介。这样，花朵的颜色在图像中占据主导地位，平均颜色的和和颜色分布的标准差可以用来当作图像分割的阈值，具体的步骤在算法 1 中呈现。

Algorithm 1 基于 HSV 特征的图像分割

Require:

一幅 RGB 图像

- 1: 将 RGB 图像转为 HSV 图像；
 - 2: 计算 V 通道颜色特征的均值和标准差，将二者相加得到二值化的阈值；
 - 3: 将图像的 V 通道应用所获得的阈值二值化；
 - 4: 获得的二值化图像可能存在不连接的白色区域，选择最大的连通区域作为最后的前景区域；
 - 5: 将二值图像作为掩图应用到原有的 RGB 图像。
 - 6: **return** 分割后的 RGB 图像；
-

需要注意的是，算法使用的 V 通道特征是归一化之后的特征，即将所有 V 通道值变为 $[0, 1]$ 。在我们实际处理花卉图片时，发现如果如上述算法所示第 4 步中只取最大连通区域作为前景区域，会将花的大部分区域也一同分割，我们这里将这一步忽略，虽然牺牲了一定的分割准确性，但是保留了整个花朵前景的区域。图 2-3 是算法的实验效果。



(a) 原始图像

(b) 分割后的图像

图 2-3 基于 HSV 特征的图像分割方法的实验效果

2.3.2 基于 OTSU 的颜色分割

花朵的区域可以通过使用颜色特征来分割出来^[1]，因为它的图像中通常包含一大片绿色的区域和花朵的区域。绿色区域代表花朵周围的叶子，花朵区域是被其颜色所特征表示的区域。

颜色分割可以通过两种颜色的距离或差异来分割。在这个方法中，图像被转换到 Lab 颜色空间。然后， δ_E 表示每一个像素点值与平均 LAB 颜色的距离。在这之后，OTSU 阈值被应用到 L 通道上完成分割。OTSU 算法尝试找到一个最优的分割通过计算一个全局阈值^[9]。

图像分割问题可以看作一个二分类问题，前景是正类，而背景是负类。最佳的阈值 t^* 要能够最小化类间方差，等价于最大化类间方差。

$$t^* = \arg \max_{1 \leq t \leq L} \{w_0(\mu_0 - \mu_T)^2 + w_1(\mu_1 - \mu_T)^2\} \quad (2.1)$$

我们实际操作中，使用 Lab 空间的 L 通道作为分割特征，通过在计算直方图的方式将阈值在 $[0, L_{\max}]$ 找到最佳阈值。图 2-4 是基于 OTSU 的颜色分割方法的实验效果。



(a) 原始图像

(b) 分割后的图像

图 2-4 基于 OTSU 的颜色分割的实验效果

2.4 特征提取

2.4.1 基于视觉词汇的特征提取

就像植物学家一样，我们为了正确区分不同类别的花朵需要能够回答特定的问题。越相似的花朵我们需要回答的问题就越多^[3]。一个花朵通常由花瓣、萼片等组成。花瓣提供了花朵种类的关键信息。一些花的花瓣有极具区分性的形状、一些可能有特殊的颜色，一些拥有辨识性的纹理模式，还有一些通过这些特

性的组合而被识别。这种方法试图构建一个视觉词汇对上述三种特征给予一个正确的表示。

2.4.1.1 颜色词汇

一些花拥有很多种颜色而大多数却只有一种特殊的颜色。所以花朵的颜色特征何以帮助缩小花可能的类别。花卉图像通常是在户外情况下采集的，光照的变化对图片的影响很大。此外，花朵通常或多或少看起来有些透明，特殊强光刺激可能会导致花朵颜色变白，环境的影响因素很大。

一种解决办法是使用一种对光照变化不敏感的颜色空间——HSV 颜色空间。为了能够得到更好的泛化效果，每个像素 HSV 的值被提取出来用 k-means 方法进行聚类。

给定一组聚类中心（视觉单词） $w_i^c, i = 1, 2, \dots, V_c$ ，每幅图像 $I_j, j = 1, 2, \dots, N$ ，然后用一个归一化的频率直方图 $n(w^c | I_j)$ 表示。一幅新的图像可以基于频率直方图特征的最近邻分类器。

$$c^* = \arg \min_j d(n(w^c | I^{\text{test}}), n(w^c | I_j^{\text{train}})) \quad (2.2)$$

其中距离 $d(\cdot)$ 使用 χ^2 度量。实际使用过程中，我们发现对每幅图像的所有像素点进行聚类速度太慢，所以我们实际操作中采用下面的方法加速计算。

对于一幅给定的 HSV 图像 I ，对于每一行和每一列像素，分别计算当前行/列的均值向量 \mathbf{m} 和协方差矩阵 Σ ，将协方差矩阵铺平成一维向量，再与均值向量拼接成一个向量 \mathbf{v}_i 作为一行/列特征，每幅图像以这个计算得的向量做聚类操作，完成上述后续操作。

图 2-5 所示是上述方法的图示解释，实践中我们发现，因为原有特征提取的耗时性主要来源于对每幅图像的每个像素进行 k-means 聚类。举个例子，对于一幅 500*600 大小的图像，就有 300000 个像素点进行聚类，这个对 k-means 是极其耗时且复杂的，运用我们方法，同样也可以表示颜色特征，但数据点样本只有 1100 个，k-means 的速度大大提升，使在我们的硬件平台上可以完成实验并得到结果。

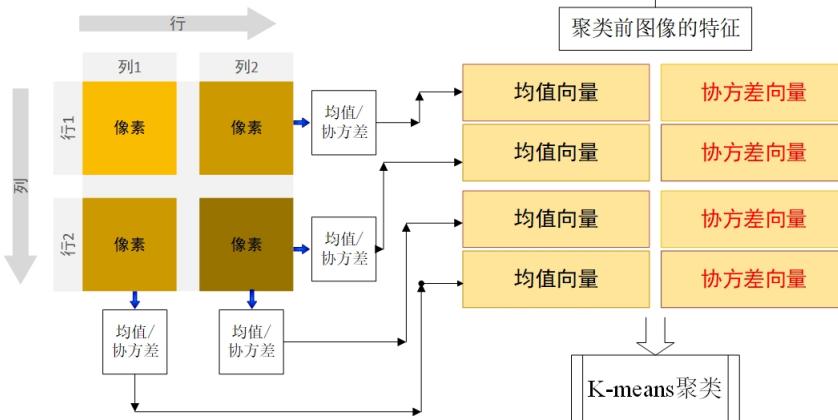


图 2-5 提取颜色词汇特征的框架

2.4.1.2 形状词汇和纹理词汇

每一朵花的花瓣的形状、构造和全局形状可以被用来区分花卉种类。视角和清晰度的变化改变了花卉的形状，所以自然界的自然形变极大提升分类问题的难度。所以我们想到采用 SIFT 特征描述子对每一幅图像提取特征点和相关特征。我们使用与 2.4.1.1 中相同的视觉特征提取方法和分类方法^①获得形状词汇特征 $n(w^s|I)$ ，用于后续分类，分类的方法与 2.4.1.1 所述的一致。

一些花朵的花瓣存在特定的纹理模式，他们更具有区分性，原文^[3]中采用的是 MR8 滤波器组来提取纹理特征，我们实验中发现 MR8 滤波器组提取特征时十分耗时，所以这里我们采用 LBP 算子提取纹理特征，对于 LBP 算子处理过后的灰度图像应用与提取颜色词汇特征同样的步骤，包括求行/列均值和方差简化计算来得到纹理词汇的描述 $n(w^t|I)$ 。分类的方法与之前提到的颜色词汇和形状词汇一样^②。

2.4.1.3 组合词汇

为了将三种特征词汇很好地结合起来，我们引入了一组权重向量 α ，所以我们最终得到的特征直方图如式 (2.3) 所示。

$$n(w|I) = \begin{bmatrix} \alpha_s n(w^s|I) \\ \alpha_c n(w^c|I) \\ \alpha_t n(w^t|I) \end{bmatrix} \quad (2.3)$$

^①这里直接提取 SIFT 特征而不用求行/列均值和方差。

^②关于 SIFT 特征描述子和 LBP 算子的介绍见附录 A。

这种将对于每个方面的分类的结合与组合独立的分类器相似。 α 权重向量给了关于距离向量的线性组合。为了更明白的说明这个问题，在这种情况下， $d(\alpha\mathbf{x}, \alpha\mathbf{y}) = \alpha d(\mathbf{x}, \mathbf{y})$ ，所以最终的分类问题如式（2.4）描述。

$$\begin{aligned} c^* = \arg \min_j \{ & \alpha_s d(n(w^s | I^{\text{test}}), n(w^s | I_j^{\text{train}})) + \\ & \alpha_c d(n(w^c | I^{\text{test}}), n(w^c | I_j^{\text{train}})) + \\ & \alpha_t d(n(w^t | I^{\text{test}}), n(w^t | I_j^{\text{train}})) \} \end{aligned} \quad (2.4)$$

2.4.2 基于融合特征描述子的特征提取

W. Liu 等人在^[10] 中提出了一种基于 DenSIFT 的直方图特征和 LLC 稀疏编码的特征提取方法，图 2-6 展示了该方法的流程。

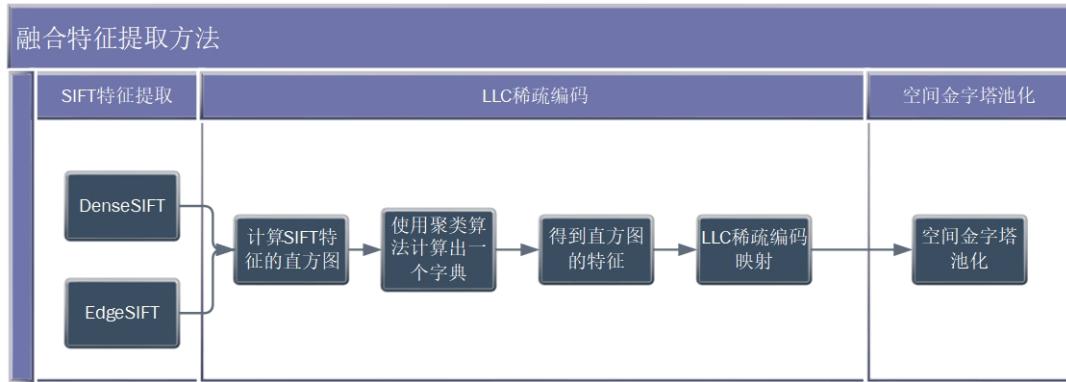


图 2-6 基于融合特征描述子的特征提取流程

颜色和纹理特征是在花卉识别的领域的两个重要且具有区分性的特征。将二者合理地结合可以很大幅度提高分类算法的性能。在这里，我们使用 PHOW-Color、PHOW-Gray 和 edge-SIFT 来提取花卉特征。

PHOW (a Pyramid Histogram Of visual Words) 是一种基于 bag-of-words 方法改进的特征表示方法，通过提取多尺度的 Dense SIFT 来保持特征信息的空间分布。主要有两个不同版本的 PHOW，分别应用于颜色和纹理的特征提取上。PHOW-Color 被用于分别提取图像的 H、S、V 三个通道上的 PHOW 特征，然后将它们拼接成一个完整的特征。PHOW-Gray 是另一种 PHOW 模型用于在灰度空间上提取图像特征。

PHOW 的提取过程主要有散步：Dense-SIFT 特征提取、建立视觉词典和空间金字塔池化。其中建立视觉字典的过程与 2.4.1.1 中建立字典的过程一致，这里不再赘述。

2.4.2.1 Dense SIFT 和 Edge SIFT 特征提取

Dense SIFT 是 SIFT 特征的一种改进型，首先将表达目标的矩形区域分成相同大小的矩形块，计算每一个小块的 SIFT 特征，再对各个小块的稠密 SIFT 特征在中心位置进行采样，建模对目标的表达。在这里，我们在以 16×16 的像素为大小、以 8 个像素为步长的网格划分出的区域上计算 Dense SIFT 特征。这样，我们可以得到一个在颜色和纹理层次上 SIFT 特征描述：

$$S^i = \{(d_1, k_1), (d_2, k_2), \dots, (d_S, k_S)\} \quad i \in (0, 1) \quad (2.5)$$

上式中 S^i 在 $i = 0$ 时表示提取的是颜色特征，而 $i = 1$ 是表示提取的是纹理特征， d_i 表示在其对应的几何坐标系下的 d 维的 SIFT 特征向量， k_i 表示第 i 个描述子的几何坐标系。 S 表示网格区域的个数。

通过统计的方法得到这些描述子的直方图，即

$$H_i = \{h_1, h_2, \dots, h_n\} \quad (2.6)$$

其中 n 是直方图的维度。

Edge SIFT 被用来提取一幅轮廓图像的 SIFT 特征来表示图像的形状特征。在这里，我们通过 Canny 算子^①得到一幅图像的轮廓图。也就是说，对于一幅图像 I ，其轮廓图像 C 为二值化图像，即

$$C(i, j) = \begin{cases} 1 & \text{if } I(i, j) \text{ is contours} \\ 0 & \text{if } I(i, j) \text{ is not contours} \end{cases} \quad (2.7)$$

应用与上述一样的方法得到 Edge SIFT 描述子的直方图。

2.4.2.2 Locality-constrained Linear Coding

这里我们采用 Locality-constrained Linear Coding (LLC) 编码方式来对提取出的直方图字典特征进行编码。这种编码方式更多的考虑局部信息，认为局部信息比稀疏性更重要。LLC 采用了局部约束取代了其他编码方式中采用的稀疏约束^[11]。

假定我们已经得到了每幅图像的特征的统计直方图。令 D 表示每个直方图的维度，K-Means 学习得到的字典 $B = \{b_1, b_2, \dots, b_m\} \in \mathbb{R}^{D \times M} (M \gg D)$ ， M 表示聚类中心的个数。令 $I = \{I_1, I_2, \dots, I_N\} \in \mathbb{R}^{D \times N}$ 代表一个从图像中提取的 D 维的

^①关于 Canny 算子的介绍见附录 A

局部特征。LLC 试图在 B 中通过欧氏距离度量找到 k 个最近邻来构建一个小字典 $\tilde{B} = \{b_{[1]}, b_{[2]}, \dots, b_{[k]}\} \in \mathbb{R}^{D \times k}$, 这里 k 是用户指定的超参数。即

$$\begin{aligned} \min_C \quad & \sum_{i=1}^N \|x_i - Bc_i\|^2 + \lambda \|d_i \odot c_i\|^2 \\ \text{s.t.} \quad & \mathbf{1}^T c_i = 1, \forall i \end{aligned} \quad (2.8)$$

这里 \odot 表示矩阵对应元素相乘, $d_i \in \mathbb{R}^M$ 表示位置适配器, 根据与输入特征 x_i 的相似程度给每个基向量赋予不同的自由度。具体来说,

$$d_i = \exp\left(\frac{\text{dist}(x_i, B)}{\sigma}\right) \quad (2.9)$$

这里 $\text{dist}(x_i, B) = [\text{dist}(x_i, b_1), \dots, \text{dist}(x_i, b_M)]^T$, $\text{dist}(x_i, b_j)$ 表示 x_i 和 b_j 的欧氏距离。 σ 用于调整局部适配器的权重衰减。在计算时要对 d_i 进行归一化。图 2-7 所示是 LLC 编码的过程。

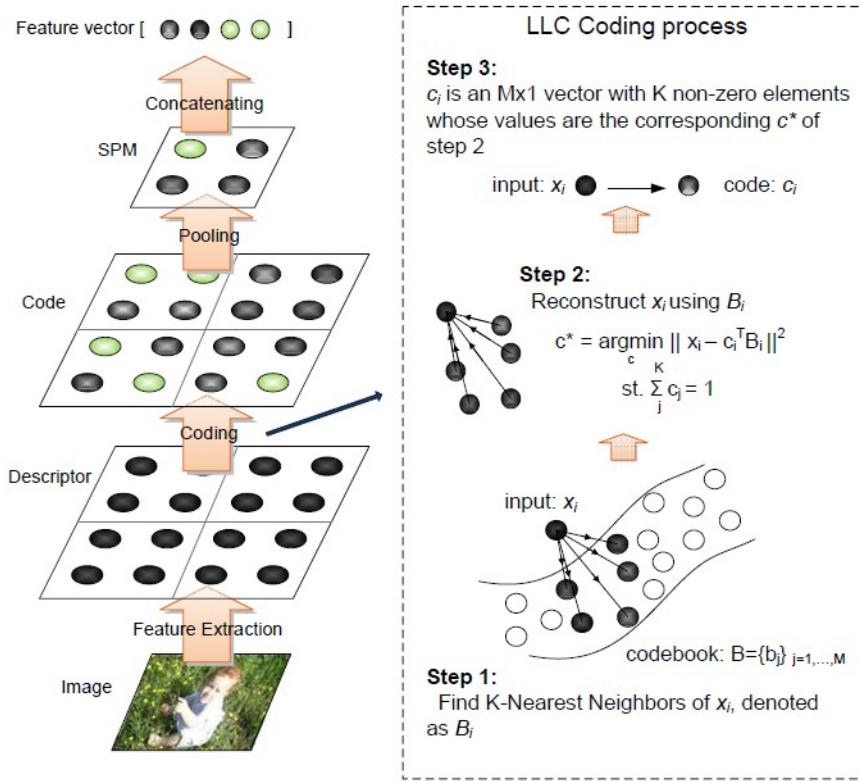


图 2-7 LLC 编码流程

式 (2.8) 所示的优化问题一个很好的性质是拥有解析解, 可以避免迭代求解造成的大量计算资源和时间的消耗。LLC 的解析解形式如式 (2.11) 所示。

$$\tilde{c}_i = (C_i + \lambda \text{diag}(d)) \setminus \mathbf{1} \quad (2.10)$$

$$c_i = \tilde{c}_i / \mathbf{1}^T \tilde{c}_i \quad (2.11)$$

这里 $C_i = (B - \mathbf{1}x_i^T)(B - \mathbf{1}x_i^T)^T$ 是数据的协方差矩阵。

2.4.2.3 空间金字塔池化

空间金字塔表示 (Spatial Pyramid Representation, SPR) 是一种当下十分流行特征编码方式。如图 2-8 中所示，在每一个金字塔等级 l 下，输入图像被分为一系列递增的重叠网格。所有的描述子都从每一个网格中提取特征，然后拼合成最终的图像特征描述。

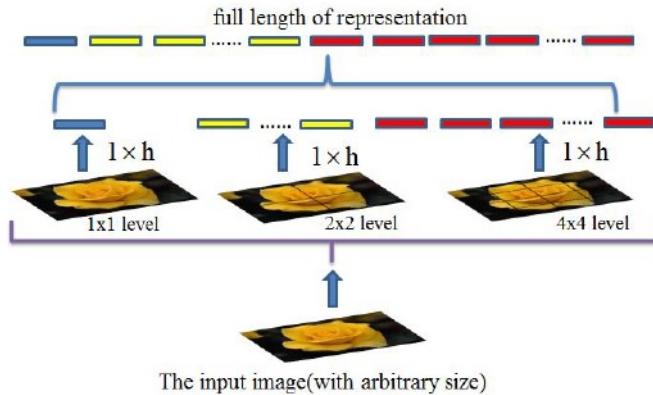


图 2-8 空间金字塔池化示意图

在这里我们采用 LLC (Locality-constrained Linear Coding) 体系来对每个网格提取的特征进行编码，每个局部字典大小设为 60. 我们建立了一个 3 层金字塔，包括 1×1 、 2×2 和 4×4 不同的子区域。令 $h_{C(l)}^j \in \mathbb{R}^d$ 表示从第 j 层的网格 $C(l)$ 上提取的 PHOW 特征编码。所以，最终图像 I_i 的 PHOW 特征编码表示为

$$H_i^k = \{h_1^0, h_1^1, \dots, h_{C(l)}^l\} \quad (2.12)$$

这里，当 $k=1$ 时表示在 HSV 空间上提取的 PHOW 特征， $k=0$ 时表示在灰度空间上提取的 PHOW 特征， $k=2$ 时表示提取的 Edge SIFT 特征。

2.4.3 基于稀疏编码的 SIFT 特征

SIFT (Scale Invariant Feature Transform) 特征描述子是一种被广泛应用于图像和计算机视觉领域来提取特征点，对光照、尺度和旋转具有一定的不变性，可以用来提取花卉分类的特征。Hossam M.Z. 在^[1] 中将 SIFT 特征提取出来的结果采用稀疏编码的方式得到图像的特征。

在我们具体实践的时候，我们采用了 DenseSIFT 描述子来提取图像的特征，

最后在采用式 (2.13) 进行稀疏编码。

$$\min \sum_{i=1}^N \left(\left\| x_i - \sum_{j=1}^M a_i^{(j)} \phi^{(j)} \right\|^2 + L \right) \quad (2.13)$$

$$L = \lambda \sum_{j=1}^M |a_i^{(j)}| \quad (2.14)$$

其中 x_i 是 SIFT 提取的特征, a^j 中大部分元素都是零, ϕ 是稀疏编码的基, λ 是权重向量。我们在实际过程中采用 KSVD 算法求解稀疏编码问题^[12]。

Algorithm 2 K-SVD 稀疏编码

Require: 原始样本、字典、稀疏矩阵

Ensure: 字典、稀疏矩阵

- 1: **初始化:** 从原始样本 $Y \in \mathbb{R}^{m \times n}$ 随机取 K 个列向量或者取它的左奇异矩阵的前 K 个列向量 $\{d_1, d_2, \dots, d_K\}$ 作为初始字典的原子, 得到字典 $D^{(0)} \in \mathbb{R}^{m \times K}$ 。令 $j = 0$, 重复下面的步骤, 直到达到指定的迭代步数, 或收敛到指定的误差;
- 2: **稀疏编码:** 利用字典上一步得到的字典 $D^{(j)}$, 稀疏编码, 得到 $X^{(j)} \in \mathbb{R}^{K \times n}$;
- 3: **字典更新:** 逐列更新字典 $D^{(j)}$, 字典的列 $d_k \in \{d_1, d_2, \dots, d_K\}$;
- 4: 当更新 d_k 时, 计算误差矩阵 $E_k = Y - \sum_{j \neq k} d_j x_T^{(j)}$;
- 5: 取出稀疏矩阵第 k 个行向量 x_T^k 不为 0 的索引的集合 $w_k = \{i | 1 \leq i \leq n, x_T^k(i) \neq 0\}$, $x_T'^k = \{x_T^k(i) | 1 \leq i \leq n, x_T^k(i) \neq 0\}$;
- 6: 从 E_k 取出对应 w_k 不为 0 的列, 得到 E'_k ;
- 7: 对 E'_k 作奇异值分解 $E_k = U \Sigma V^T$, 取 U 的第 1 列更新字典的第 k 列, 即 $d_k = U(\cdot, 1)$, 令 $x_T'^k = \Sigma(1, 1)V(\cdot, 1)^T$, 得到 $x_T'^k$ 后, 将其对应地更新到原 x_T^k ;
- 8: $j = j + 1$

2.4.4 基于分割的分形纹理分析

基于分割的分形纹理分析 (Segmentation-based Fractal Texture Analysis, SFTA) 主要分为两个部分^[13]：1) 把输入的灰度图像分解成一系列二值图像, 这里采用两阈值二值化分解 (Two-Threshold Binary Decomposition, TTBD) 进行二值化; 2) 对于上述得到的二值化图像, 我们计算区域的边界的分形维度, 另外在计算区域的平均灰度值和面积。

2.4.4.1 两阈值二值化分解

两阈值二值化分解 (TTBD) 将一幅灰度图像 $I(x, y)$ 当作输入, 并且返回一系列二值图像。两阈值二值化分解的第一步是计算一组分割阈值 T 。两阈值二值化分解采用一种叫做多等级 OTSU (multi-level OTSU) 算法, 利用输入图像的灰度分布信息来计算阈值。

多等级 OTSU 算法的步骤如下：寻找到阈值能够使输入的图像的类内方差

最小化；然后，以递归的方式不断对每一幅输入图像区域应用 OTSU 算法直到得到 n_t 个阈值，这里 n_t 是一个预先设定的参数。图 2-9 所示是两阈值二值化分解的工作流程。

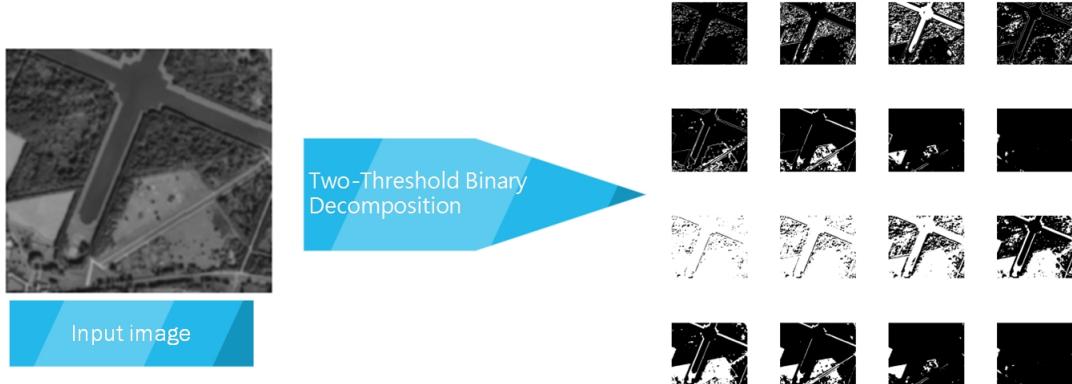


图 2-9 两阈值二值化分解的示意图

接下来，利用得到的阈值将输入灰度图像 $I(x,y)$ 分解成一系列二值图像，即

$$I_b(x,y) = \begin{cases} 1 & \text{if } t_l < I(x,y) \leq t_u \\ 0 & \text{otherwise} \end{cases} \quad (2.15)$$

这里 t_l 和 t_u 是用于分割的上界和下界。

给定一幅输入图像，使用从 $T \cup \{n_l\}$ 得到的所有邻近的阈值对和从 $\{t, n_l\}, t \in T$ 得到的所有邻近的阈值对，通过式 (2.15) 可以得到一系列二值图像， n_l 代表 $I(x,y)$ 的最大可能出现的灰度值。这样一幅输入的灰度图像可以得到 $2n_l$ 个二值图像。

两阈值二值化分解的一个重要性质是，所得的二值图像是通过使用多级 OTSU 算法计算的阈值应用单阈值分割而获得的所有二值图像的超集合。

使用阈值对分割得到二值图像的基本原理是对那些常规阈值分割无法分割出的图像进行分割，尤其是那些目标的灰度区域在输入图像灰度直方图的中部范围。通过使用阈值对可以将那些中等亮度的目标的区域信息从原图像中提取出来。

2.4.4.2 SFTA 特征提取

在对输入灰度图像进行两阈值二值分解之后，将二值图像的面积、平均灰度值和边界分形维度组成最后 SFTA 的特征向量。分形度量被用来表示分割后的目标和结构的边界复杂度。二值图像 $I_b(x,y)$ 的边界组成一幅边界图像记作 $\Delta(x,y)$ ，

有

$$\Delta(x,y) = \begin{cases} 1 & \text{if } \exists(x',y') \in N_8[(x,y)] : \\ & I_b(x',y') = 0 \wedge \\ & I_b(x,y) = 1, \\ 0 & \text{otherwise.} \end{cases} \quad (2.16)$$

这里 $N_8[(x,y)]$ 是该像素的 8-邻域。通过式 (2.16) 得到的边界是一像素宽的。再在每一个得到的边界图像上计算分形维数 D 。

分形几何包括多种定义分形维数的方法，这里我们采用的最为常见的 Hausdorff 维度。考虑一个目标的欧式维度是 E ，它的 Hausdorff 分形维度 D_0 可以由式 (2.17) 计算得到

$$D_0 = \lim_{\epsilon \rightarrow 0} \frac{\log N(\epsilon)}{\log \epsilon^{-1}} \quad (2.17)$$

这里 $N(\epsilon)$ 是 E 维的超立方体的个数， ϵ 表示填充目标的长度。

如果我们考虑一个由二值图像 I_b 表示的目标，可以通过 Box Counting 算法计算 D_0 的近似 D 。不失一般性，下面我们考虑 2D 的情形。起初，图像被分为一个由 $\epsilon \times \epsilon$ 的方格组成的网格。接下来再每一个方格中输出至少包含一个目标像素的 $\bar{N}(\epsilon)$ 。通过改变 ϵ 的值，可以得到一条 $\log \bar{N}(\epsilon)$ - $\log \epsilon^{-1}$ 曲线。最终，通过线拟合方法，用一条直线近似这条曲线，最后分形维数 D 就是这条直线的斜率。

平均灰度值和区域面积用来补充从每张二值图像提取出来的信息，同时不显著增加计算负担。这样，SFTA 特征向量的维度等于两阈值二值化分解得到的图像个数的三倍：分形维度、平均灰度值和区域面积。图 2-10 说明从每幅二值图像提取出来的三个特征。

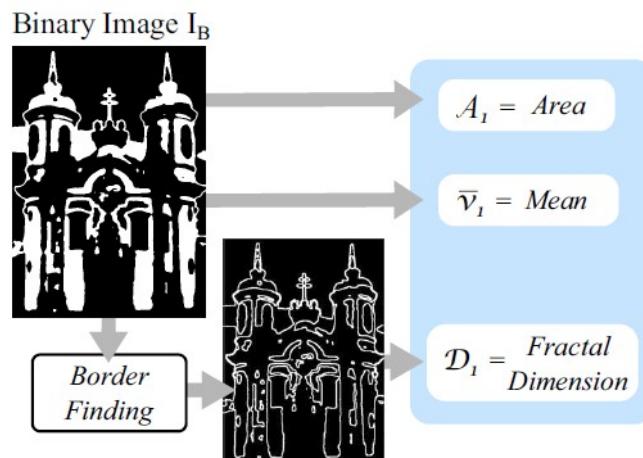


图 2-10 SFTA 特征向量的组成

Algorithm 3 SFTA 特征提取

Require: 灰度图像 I , 阈值的个数 n_t
Ensure: 特征向量 V_{SFTA}

- 1: $T \leftarrow \text{MultiLevelOTSU}(I, n_t)$
- 2: $T_A \leftarrow \{\{t_i, t_{i+1}\} : t_i, t_{i+1} \in T, i \in [1|T|-1]\}$
- 3: $T_B \leftarrow \{\{t_i, n_l\} : t_i \in T, i \in [1|T|]\}$
- 4: $i \leftarrow 0$
- 5: **for** $\{t_l, t_u\} : \{t_l, t_u\} \in T_A \cup T_B$ **do**
- 6: $I_b \leftarrow \text{TwoThresholdSegmentation}(I, t_l, t_u)$
- 7: $\Delta(x, y) \leftarrow \text{FindBorders}(I_b)$
- 8: $V_{\text{SFTA}}[i] \leftarrow \text{BoxCounting}(\Delta)$
- 9: $V_{\text{SFTA}}[i + 1] \leftarrow \text{MeanGrayLevel}(I, I_b)$
- 10: $V_{\text{SFTA}}[i + 2] \leftarrow \text{PixelCount}(I_b)$
- 11: **end for**
- 12: **return** V_{SFTA}

算法 3 说明了 SFTA 特征提取的过程。 V_{SFTA} 表示最后得到的特征向量。第 1 行表示通过多级 OSTU 算法计算一系列的阈值；第 2 行将 T 所有邻近的阈值对加入 T_A ；第 3 行将阈值对 $\{t_i, n_l\}$ 加入到 T_B 中；第 5 行将所有阈值对组合，即 $T_A \cup T_B$ ；第 6 行使用每一个阈值对分割图像；7-10 行计算分型维度、平均灰度值和区域面积。

2.5 机器学习分类算法

2.5.1 支持向量机

支持向量机 (Support Vector Machine, SVM) 是机器学习中最为流行的分类算法之一。以二分类问题为例，给定训练数据 $(\mathbf{x}_i, t_i), i = 1, \dots, N$ ，这里 $\mathbf{x}_i \in \mathbb{R}^d$ 和 $t_i \in \{-1, 1\}$ ，由于一般情况下，数据通常线性不可分，所以通常将原始训练数据 \mathbf{x}_i 通过非线性映射映射到一个新的特征空间 Z : $\phi: \mathbf{x}_i \rightarrow \phi(\mathbf{x}_i)$ 。再新的特征空间 Z 中不同的类之间的距离为 $2/\|\mathbf{w}\|$ 。为了同时最大化类间距离和最小化训练误差 ξ_i ，等价于

$$\begin{aligned} \text{Minimize } L_{\text{SVM}} &= \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \xi_i \\ \text{Subject to } t_i(\mathbf{w}\phi(\mathbf{x}_i) + b) &\geq 1 - \xi_i, \quad i = 1, \dots, N \end{aligned} \tag{2.18}$$

$$\xi_i \geq 0, \quad i = 1, \dots, N$$

C 是一个超参数用于平衡分类间隔和训练误差。

基于 Karush-Kuhn-Tucker (KKT) 定理，训练上述的支持向量机等价于求解

下面的对偶优化问题：

$$\begin{aligned} \text{minimize } L_{D_{\text{SVM}}} &= \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N t_i t_j \alpha_i \alpha_j \phi(\mathbf{x}_i) \phi(\mathbf{x}_j) - \sum_{i=1}^N \alpha_i \\ \text{subject to } \sum_{i=1}^N t_i \alpha_i &= 0 \\ 0 \leq \alpha_i &\leq C, i = 1, \dots, N \end{aligned} \quad (2.19)$$

这里拉格朗日系数 α_i 对应于训练样本 (\mathbf{x}_i, t_i) 。对于向量 \mathbf{x}_i , 若满足 $t_i(\mathbf{w}\phi(\mathbf{x}_i) + b) = 1$ 则 \mathbf{x}_i 为支撑向量。

核函数是支撑向量机的一大技巧。令核函数为 $K(\mathbf{u}, \mathbf{v}) = \phi(\mathbf{u})\phi(\mathbf{v})$, 这样有

$$\begin{aligned} \text{minimize } L_{D_{\text{SVM}}} &= \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N t_i t_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) - \sum_{i=1}^N \alpha_i \\ \text{subject to } \sum_{i=1}^N t_i \alpha_i &= 0 \\ 0 \leq \alpha_i &\leq C, i = 1, \dots, N \end{aligned} \quad (2.20)$$

最后支持向量机的决策函数为

$$f(\mathbf{x}) = \text{sign} \left(\sum_{s=1}^{N_s} \alpha_s t_s K(\mathbf{x}, \mathbf{x}_s) + b \right) \quad (2.21)$$

这里 N_s 是支撑向量的个数。

本次课程设计中，除了采用常用的 RBF 核，我们采用了一种叫直方图交叉核 (Histogram Intersection Kernel, HIK) 的核函数^[14]。令 $\mathbf{h} = (h_1, \dots, h_d) \in \mathbb{R}_+^d$ 为一个直方图， \mathbf{h} 能够代表一幅图像或一个子区域。定义直方图交叉核 \mathcal{K}_{HI} 为

$$\mathcal{K}_{\text{HI}}(\mathbf{h}_1, \mathbf{h}_2) = \sum_{i=1}^d \min(h_{1i}, h_{2i}) \quad (2.22)$$

可以证明 \mathcal{K}_{HI} 是一个有效的正定核函数。这样存在一个映射 ϕ 将任何一个直方图 \mathbf{h} 映射到对应的高维空间 Φ 的向量 $\phi(\mathbf{h})$, 即 $\mathcal{K}_{\text{HI}}(\mathbf{h}_1, \mathbf{h}_2) = \phi(\mathbf{h}_1)\phi(\mathbf{h}_2)$ 。通过非线性映射 ϕ , 直方图的相似性等价于在高维空间 Φ 的内积。

2.5.2 随机森林

随机森林 (Random Forest, RF) 是一系列树结构的分类器的集成。每一棵树依赖于一个独立采样的随机向量和森林中全部的树的分布。它本质上是决策树 (Decision Tree, DT) 的集成。主要的原理是通过将多个弱分类器集成来建立一个强分类器。

随机森林分类器从树的顶部开始输入，指导最后到达一个叶子节点。原始数据被随机采样，但是不断替换成更小的集合。采样样本的类别由随机森林的树决定，这里是个任意数。

令 $(X_1, Y_1), (X_2, Y_2), \dots, (X_n, Y_n)$ 是随机变量对，其中 $X \in \mathbb{R}^d$ 是特征向量， $Y \in \{0, 1\}$ 是由随机变量产生的标签集。 (X, Y) 的联合分布由 X 的边际分布 μ (即, $P\{X \in A\} = \mu(A)$ 对于集合 $A \subset \mathbb{R}^d$)，和一个后验概率 $\eta : \mathbb{R}^d \rightarrow [0, 1]$,

$$\eta(x) = p(Y = 1 | X = x) \quad (2.23)$$

D_n 表示训练数据，为 $(X_1, Y_1), (X_2, Y_2), \dots, (X_n, Y_n)$ 的集合。一个分类器 g_n 是一个关于 X 和 D_n 的二值函数，其错误概率为

$$L(g_n) = P_{(x,y),z}\{g_n(X, D_n) \neq Y\} \quad (2.24)$$

这里 $P_{(x,y)}$ 表示关于 (X, Y) 的概率，也即是由 D_n 的条件概率。随机森林是平均分类器。正式地说，一个有 m 颗树的随机森林是一个基于随机化产生的树分类器 $g_n(x, Z_1), \dots, g_n(x, Z_m)$ 的集合。 Z_1, \dots, Z_m 是同分布的随机向量，在 X, Y, D_n 上条件独立。随机变量通常用于确定在构建树时如何执行连续剪切，例如选择节点和要拆分的坐标以及拆分的位置。

随机森林在以决策树为基学习器构建 Bagging 集成的基础上，进一步在决策树的训练过程中引入了随机属性选择。具体地说，传统决策树在选择划分属性时是在当前结点的属性集合（假定有 d 个属性）中选择一个最优属性；而在随机森林中，对基决策树的每个结点，先从该点的属性集合中随机选择一个包含 k 个属性的子集，然后再从这个子集中选择一个最优属性用于划分。这里的参数 k 控制了随机性的引入程度：若令 $k = d$ ，则基决策树的构建与传统决策树相同；若令 $k = 1$ ，则是随机选择一个属性用于划分；一般情况下，推荐值 $k = \log_2 d$ 。随机森林中基学习器的多样性不仅来自样本扰动，还来自属性扰动，这就使得最终集成的泛化性能可通过个体学习器之间差异度的增加而进一步提升。

2.5.3 极限学习机

极限学习机 (Extreme Learning Machine, ELM) 是单隐层前馈神经网络，随机确定输入权重和可以通过解析解得到输出权重参数^[15]。首先我们要介绍一种 Moore-Penrose 广义逆矩阵和一个一般线性系统 $\mathbf{A}\mathbf{x} = \mathbf{y}$ 的最小范数最小二乘解，其次介绍极限学习机的基本流程。

2.5.3.1 Moore-Penrose 广义逆矩阵

广义线性系统 $\mathbf{A}\mathbf{x} = \mathbf{y}$ 的解, 这里 \mathbf{A} 可能是奇异的, 也可能不是方阵, 这时需要用 Moore-Penrose 广义逆矩阵求解。

定义 2.1. *Moore-Penrose 广义逆矩阵:* 一个矩阵 $\mathbf{G} \in \mathbb{R}^{n \times m}$ 是 $\mathbf{A} \in \mathbb{R}^{m \times n}$ 的 *Moore-Penrose 广义逆矩阵*, 当且仅当

$$\mathbf{AGA} = \mathbf{A}, \mathbf{GAG} = \mathbf{G}, (\mathbf{AG})^T = \mathbf{AG}, (\mathbf{GA})^T = \mathbf{GA} \quad (2.25)$$

为了简化表示, 记矩阵 \mathbf{A} 的 Moore-Penrose 广义逆矩阵为 \mathbf{A}^\dagger 。

2.5.3.2 一般线性系统的最小范数最小二乘解

对于一个一般的线性系统 $\mathbf{AX} = \mathbf{y}$, 我们称 $\hat{\mathbf{x}}$ 是一个最小二乘解, 当且仅当

$$\|\mathbf{A}\hat{\mathbf{x}} - \mathbf{y}\| = \min_x \|\mathbf{Ax} - \mathbf{y}\| \quad (2.26)$$

这里 $\|\cdot\|$ 是欧式空间的范数。

定义 2.2. $\mathbf{x}_0 \in \mathbb{R}^n$ 被称为线性系统 $\mathbf{Ax} = \mathbf{y}$ 最小范数最小二乘解如果对任意 $\mathbf{y} \in \mathbb{R}^m$ 有

$$\|\mathbf{x}_0\|, \forall \mathbf{x} \in \{\mathbf{x} : \|\mathbf{Ax} - \mathbf{y}\| \leq \|\mathbf{Az} - \mathbf{y}\|, \forall \mathbf{z} \in \mathbb{R}^n\} \quad (2.27)$$

也就是说, 一个线性系统 $\mathbf{Ax} = \mathbf{y}$ 的最小范数最小二乘解 \mathbf{x}_0 如果它是所有最小二乘解中范数最小的解。

定理 2.1. 假设存在矩阵 \mathbf{G} 使得 \mathbf{Gy} 是线性系统 $\mathbf{Ax} = \mathbf{y}$ 的最小范数最小二乘解, 当且仅当 $\mathbf{G} = \mathbf{A}^\dagger$ 。

也就是说 \mathbf{G} 是 \mathbf{A} 的 Moore-Penrose 广义逆矩阵。

2.5.3.3 极限学习机的基本原理

对于 N 个可分样本 $(\mathbf{x}_i, \mathbf{t}_i)$, 这里 $\mathbf{x}_i = [x_{i1}, x_{i2}, \dots, x_{in}]^T \in \mathbb{R}^n$, $\mathbf{t}_i = [t_{i1}, t_{i2}, \dots, t_{im}]^T \in \mathbb{R}^m$, 标准的有 \bar{N} 个隐层神经元和激活函数 $g(x)$ 的单隐层前馈神经网络 (Single hidden Layer Feedforward Networks, SLFNs), 即

$$\sum_{i=1}^{\bar{N}} \beta_i g(\mathbf{w}_i \mathbf{x}_j + b_i) = o_j, \quad j = 1, \dots, N \quad (2.28)$$

这里 $\mathbf{w}_i = [w_{i1}, w_{i2}, \dots, w_{in}]^T$ 是连接输入神经元和第 i 个隐层神经元的权重向量, $\beta_i = [\beta_{i1}, \beta_{i2}, \dots, \beta_{im}]^T$ 是连接第 i 个隐层神经元和输出神经元, b_i 是第 i 个隐层神经元的阈值。输出神经元这里是线性的。

那些带有 \bar{N} 个隐层神经元和激活函数 $g(x)$ 的单隐层前馈神经网络拟合 N 个样本, 即存在 $\beta_i, \mathbf{w}_i, b_i$, 使得

$$\sum_{i=1}^{\bar{N}} \beta_i g(\mathbf{w}_i \mathbf{x}_j + b_i) = \mathbf{t}_j, \quad j = 1, \dots, N \quad (2.29)$$

上述等式可以被更紧凑地写成

$$\mathbf{H}\beta = \mathbf{T} \quad (2.30)$$

$$\mathbf{H}(\mathbf{w}_1, \dots, \mathbf{w}_{\bar{N}}, b_1, \dots, b_{\bar{N}}, \mathbf{x}_1, \dots, \mathbf{x}_N) = \begin{bmatrix} g(\mathbf{w}_1 \mathbf{x}_1 + b_1) & \cdots & g(\mathbf{w}_{\bar{N}} \mathbf{x}_1 + b_{\bar{N}}) \\ \vdots & \cdots & \vdots \\ g(\mathbf{w}_1 \mathbf{x}_N + b_1) & \cdots & g(\mathbf{w}_{\bar{N}} \mathbf{x}_N + b_{\bar{N}}) \end{bmatrix}_{N \times \bar{N}} \quad (2.31)$$

$$\beta = \begin{bmatrix} \beta_1^T \\ \vdots \\ \beta_{\bar{N}}^T \end{bmatrix}_{\bar{N} \times m}, \quad \mathbf{T} = \begin{bmatrix} \mathbf{t}_1^T \\ \vdots \\ \mathbf{t}_N^T \end{bmatrix}_{N \times m} \quad (2.32)$$

\mathbf{H} 被称做神经网络地隐层输出矩阵; \mathbf{H} 的第 i 列是输入 \mathbf{X} 的第 i 个隐层神经元的输出向量。

如果隐层神经元的个数等于训练样本数, $\bar{N} = N$, 矩阵 \mathbf{H} 是方阵且可逆的, 单隐层前馈神经网络可以零误差拟合这些训练样本。然而, 大多数情况下隐层神经元个数远小于训练样本个数, $\bar{N} \leq N$, 这里 \mathbf{H} 不是一个方阵, 可能不存在这样的 $\mathbf{w}_i, b_i, \beta_i (i = 1, \dots, \bar{N})$ 使得 $\mathbf{H}\beta = \mathbf{T}$ 。这样, 取而代之的是找到特定的 $\hat{\mathbf{w}}_i, \hat{b}_i, \hat{\beta} (i = 1, \dots, \bar{N})$ 使得

$$\|\mathbf{H}(\hat{\mathbf{w}}_1, \dots, \hat{\mathbf{w}}_{\bar{N}}, \hat{b}_1, \dots, \hat{b}_{\bar{N}})\hat{\beta} - \mathbf{T}\| = \min_{\mathbf{w}_i, b_i, \beta} \|\mathbf{H}(\hat{\mathbf{w}}_1, \dots, \hat{\mathbf{w}}_{\bar{N}}, \hat{b}_1, \dots, \hat{b}_{\bar{N}})\hat{\beta} - \mathbf{T}\| \quad (2.33)$$

上式等价于最小化下面的损失函数:

$$E = \sum_{j=1}^N \left(\sum_{i=1}^{\bar{N}} \beta_i g(\mathbf{w}_i \mathbf{x}_j + b_i) - \mathbf{t}_j \right) \quad (2.34)$$

这里用梯度下降法来寻找到 $\|\mathbf{H}\beta - \mathbf{T}\|$ 的最小值, 定义 \mathbf{W} 是权重 (\mathbf{w}_i, β_i) 和偏置 b_i 的集合。所以 \mathbf{W} 可迭代计算式 (2.34) 得到

$$\mathbf{W}_k = \mathbf{W}_{k-1} - \eta \frac{\partial E(\mathbf{W})}{\partial \mathbf{W}} \quad (2.35)$$

这里 η 是学习率。在前馈神经网络中最流行的学习算法是后向传播, 高效地计

算梯度并从输出向输入向后传播。

在极限学习机中，单隐层前馈神经网络的输入权重和隐层神经元的偏置不需要被调整，可以随机生成。对于已给定的输入权重 \mathbf{w}_i 和隐层偏置 b_i ，训练单隐层前馈神经网络可以简单地等价于找到线性系统 $\mathbf{H}\beta = \mathbf{T}$ 的最小二乘解 $\hat{\beta}$ ，

$$\|\mathbf{H}(\hat{\mathbf{w}}_1, \dots, \hat{\mathbf{w}}_{\bar{N}}, \hat{b}_1, \dots, \hat{b}_{\bar{N}})\hat{\beta} - \mathbf{T}\| = \min_{\mathbf{w}_i, b_i, \beta} \|\mathbf{H}(\hat{\mathbf{w}}_1, \dots, \hat{\mathbf{w}}_{\bar{N}}, \hat{b}_1, \dots, \hat{b}_{\bar{N}})\hat{\beta} - \mathbf{T}\| \quad (2.36)$$

根据定理 2.1，上述线性系统的最小范数最小二乘解为

$$\hat{\beta} = \mathbf{H}^\dagger \mathbf{T} \quad (2.37)$$

算法 4 总结了极限学习机的流程。

Algorithm 4 极限学习机

Require: 给定训练集 $\mathcal{X} = \{(x_i, t_i) | x_i \in \mathbb{R}^n, t_i \in \mathbb{R}^m, i = 1, \dots, N\}$ ，激活函数 $g(x)$ ，隐层神经元个数 \bar{N}

- 1: 给输入权重 \mathbf{w}_i 和偏置 b_i 随机赋值， $i = 1, \dots, \bar{N}$
 - 2: 计算隐层输出矩阵 \mathbf{H}
 - 3: 计算输出权重 β : $\beta = \mathbf{H}^\dagger \mathbf{T}$
 - 4: **return** β
-

很多情况下，我们引入 L2 正则化项^[16]，式 (2.36) 改写为

$$\min_{\beta} \frac{1}{2} \|\beta\|^2 + \frac{C}{2} \|\mathbf{H}\beta - \mathbf{T}\|^2 \quad (2.38)$$

其中 C 为正则化系数，求解该问题等价于岭回归问题，其解析解为

$$\beta = \left(\mathbf{H}^T \mathbf{H} + \frac{1}{C} \right)^{-1} \mathbf{H}^T \mathbf{T} \quad (2.39)$$

对于式 (2.37) 中，若 $\mathbf{H}^T \mathbf{H}$ 是非奇异的，则 $\mathbf{H}^\dagger = (\mathbf{H}^T \mathbf{H})^{-1} \mathbf{H}^T$ ；若 $\mathbf{H} \mathbf{H}^T$ 是非奇异的，则 $\mathbf{H}^\dagger = \mathbf{H}^T (\mathbf{H} \mathbf{H}^T)^{-1}$ 。

2.6 深度神经网络

深度学习是当下最为流形的机器学习分支，其以强大的拟合性能和低门槛赢得人们的喜爱。对于花卉识别这类图像分类问题，深度神经网络同样可以发挥其作用^①。花卉识别的难点在于如何提取便于分类且鲁棒的特征，而深度学习恰恰将这一难题交给神经网络自动解决。实践证明，通过深度学习学习出的特征一般比人工提取的特征的性能要好。但是，并非随便一个神经网络都可以完成这一问题，这里关于图像分类问题通常采用的卷积神经网络。本次课程设计中，我们采用了多种已经存在的卷积神经网络框架，有的结合迁移学习的手段，来

^①关于深度神经网络的基础知识见附录 B。

给花卉进行分类。此外，我们还使用了一种基于深度神经网络的图像分割方法。接下来我们将一一介绍。

2.6.1 卷积神经网络概述

卷积神经网络是一种前馈型神经网络，受生物自然视觉认知机制启发而来的。现在，卷积神经网络（Convolutional Neural Network, CNN）已经成为众多科学领域的研究热点之一，特别是在模式分类领域。由于该网络避免了对图像的复杂前期预处理，可以直接输入原始图像，因而得到广泛应用。

卷积神经网络的核心是核卷积操作。核卷积不仅仅用在卷积神经网络中，它也是很多其他计算机视觉算法的关键元素。假定我们有一个小的数字矩阵（称作卷积核或滤波器），我们将它传递到我们的图像上，然后基于滤波器的数值进行变换。后续的特征图的值要通过下面的公式计算，其中输入图像被记作 f ，卷积核记为 h ，输出图像记作 g ，有

$$g(m, n) = (f * h)(m, n) = \sum_j \sum_k h(j, k) f(m - j, n - k) \quad (2.40)$$

对于图像边界的元素，要进行边界填充（Padding），通常会使用 0 做填充。一般有两种填充方式：Valid 方式表示我们使用的是原始图像，而 Same 方式代表我们在图像周围使用了边界，因此此时输入和输出图像的大小相同。在 Same 方式下扩充的宽度应满足下面方程，其中 p 是填充宽度， f 是卷积核的维度，有

$$p = \frac{f - 1}{2} \quad (2.41)$$

卷积核在图像中的移动也不一定是每次移动一个像素，这里引入步长（Stride）的概念。考虑到扩充 p 和步长 s ，输出图像的维度为

$$n_{\text{out}} = \left\lfloor \frac{n_{\text{in}} + 2p - f}{s} + 1 \right\rfloor \quad (2.42)$$

对于彩色图像，输入图像不再是二维矩阵，而是三维矩阵，需要引入立体卷积的概念。卷积核和想在其上应用卷积核的图像必须拥有相同的通道数。如果我们想在同一张图像上应用多个卷积核，我们会为每一个卷积核独立地计算卷积，然后将计算结果逐个堆叠，最后将他们组合成一个整体。得到的张量（三维矩阵）满足下面的方程，其中 n 是图像的大小， f 是卷积核的大小， n_c 是图像中的通道数， p 是填充宽度， s 是步长， n_f 是卷积核的数量，有

$$[n, n, n_c] * [f, f, n_c] = \left[\left\lfloor \frac{n + 2p - f}{s} + 1 \right\rfloor, \left\lfloor \frac{n + 2p - f}{s} + 1 \right\rfloor, n_f \right] \quad (2.43)$$

卷积层与全连接层的区别在于不使用简单的矩阵相乘，而是使用卷积。前向传播包含两个过程：第一步计算中间结果 Z ，它是由前一层的输入数据与张量 W 的卷积结果，加上偏置项 b 得到的；第二步给中间结果应用非线性的激活函数 g 。其最大的特点是参数共享，可以大大减少卷积核和偏置的参数量。

卷积神经网络另一个特点是池化层。池化是下采样的一种方式，通常将我们的图像分成不同的区域，然后在每一个部分上执行一些运算，比如取最大值的最大池化层和取平均值的平均池化层。同样的，池化区域的滤波器大小和步长定义与卷积层基本一致。

2.6.2 卷积神经网络的反向传播过程

卷积神经网络的反向传播过程与一般的神经网络有一些不同。一般神经网络中每一层输入输出都只是一个向量，而卷积神经网络中均为三维张量。其次，池化层在前向传播过程中，对输入进行了下采样，所以向前反向推导上一层的误差时，需要做上采样处理。下面介绍卷积神经网络中特有的传播过程：

已知池化层的误差，反向推导上一隐藏层的误差：在前向传播时，池化层会使用最大化或平均化对输入进行池化，池化区域大小已知。现在我们反过来，要从池化后的误差，还原前一层较大区域的误差，这个过程叫做上采样。设第 l 层误差的第 k 个子矩阵为 δ_k^l ，池化区域大小为 $a \times a$ ，把 δ_k^l 上下左右各扩展 $a-1$ 行和列进行还原。如果是最大化池化，则将误差放在上一层对应池化区域值最大的地方；如果是平均化池化，则将误差平均分摊到上一层对应的池化区域，即

$$\delta^{l-1} = \text{upsample}(\delta^l) \odot \sigma'(z^{l-1}) \quad (2.44)$$

已知卷积层的误差，反向推到上一隐藏层的误差：

$$\delta^{l-1} = \delta^l \frac{\partial z^l}{\partial z^{l-1}} = \delta^l * \text{rot180}(W^l) \odot \sigma'(z^{l-1}) \quad (2.45)$$

其中 rot180 表示将矩阵上下翻转一次再左右翻转一次。

已知卷积层的误差，推导该层的 W, b 的梯度：

$$\frac{\partial J(W, b)}{\partial W^l} = \frac{\partial J(W, b)}{\partial z^l} \partial z^l \partial W^l = \delta^l * \text{rot180}(a^{l-1}) \quad (2.46)$$

$$\frac{\partial J(W, b)}{\partial b^l} = \sum_{u,v} (\delta^l)_{u,v} \quad (2.47)$$

上面两小节简要介绍了卷积神经网络的基本原理，下面将介绍本次课程设计中我们采用的几种神经网络。

2.6.3 常用的卷积神经网络

2.6.3.1 VGG-16

VGG 是由 Simonyan 和 Zisserman 在^[17] 中提出的卷积神经网络模型，其名称来源于作者所在的牛津大学视觉几何组（Visual Geometry Group）的缩写。

VGG 中根据卷积核大小和卷积层数目的不同，可分为 A, A-LRN, B, C, D, E 共 6 个配置，其中 VGG-16 对应的是 D 配置，其结构如图 (2-11) 所示，

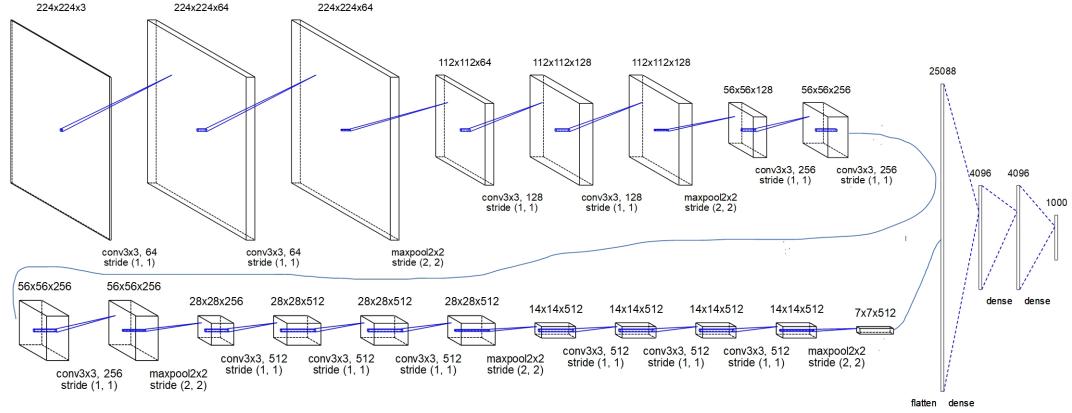


图 2-11 VGG-16 的卷积神经网络模型

VGG16 共包含 13 个卷积层、3 个全连接层和 5 个池化层。VGG-16 的突出特点是简单，其卷积层采用相同的卷积核参数，池化层采用相同的池化和参数。模型是由若干卷积层和池化层堆叠的方式构成，比较容易形成较深的网络结构。

2.6.3.2 Fully-Convolutional Network

卷积层的每层数据是一个 $h * w * d$ 的三维数组，其中 h 和 w 是空间维度， d 是特征或通道维数。第一层是像素尺寸为 $h * w$ 、颜色通道数为 d 的图像。高层中的位置和图像中它们连通的位置相对应，被称为接收域。

卷积网是以平移不变性作为基础的。其基本组成部分（卷积、池化和激活函数）作用在局部输入域，只依赖相对空间坐标。在特定层记 X_{ij} 为在坐标 (i, j) 的数据向量，在接下来的层中有 Y_{ij} ，即

$$y_{ij} = f_{ks}(\{x_{s_i+\delta_i, s_j+\delta_j}\}_{0 \leq \delta_i, \delta_j \leq k}) \quad (2.48)$$

其中 k 为卷积核尺寸， s 是步长或下采样因子， f_{ks} 决定了层的类型：一个卷积的矩阵乘或者平均池化，用于最大池化的最大空间值或者是一个元素级的非线性激活函数，或者其他类型的网络层。

上述过程的函数形式为

$$f_{ks} \circ g_{k's'} = (f \circ g)_{k'+(k-1)s', ss'} \quad (2.49)$$

一个普通的深度神经网络计算一个普通的非线性函数，一个网络只有这种形式的层计算非线性滤波，我们称之为深度滤波或者全卷积网络^[18]。一个全卷积网络可以计算任意尺寸的输入并产生相应的输出。

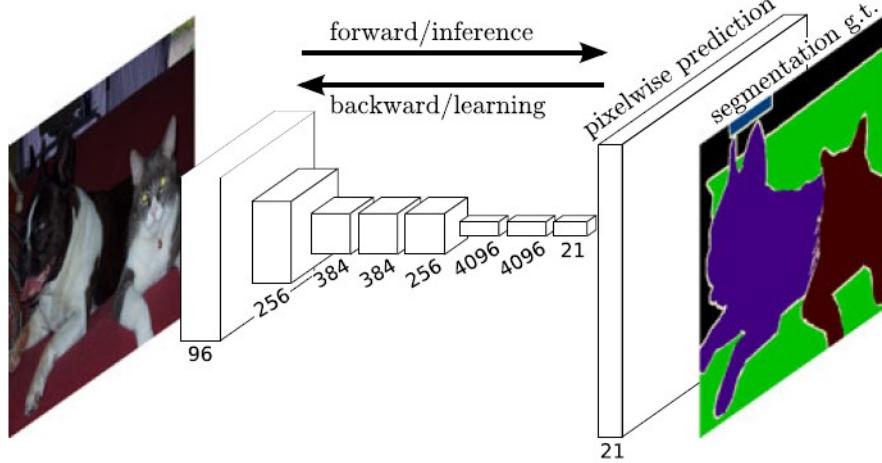


图 2-12 全卷积网络的结构示意图

这里我们使用全卷积神经网络进行图像分割，即我们将图像分割问题转换为二分类问题，0 表示背景，1 表示花朵区域。

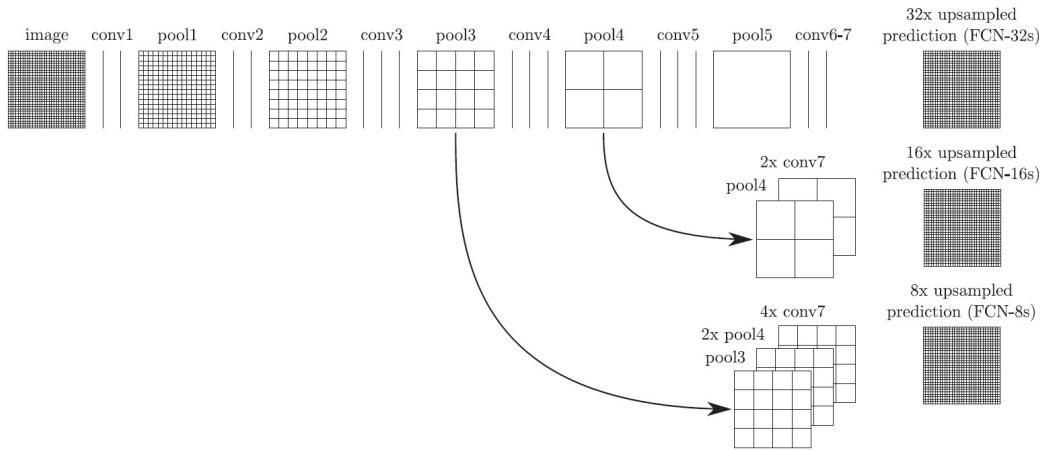


图 2-13 全卷积网络用于分割问题的网络结构图

图 2-13 所示是全卷积神经网络用于分割问题的网络结构图。这里需要将粗糙的、高层信息与精炼的、低层信息结合起来。池化和预测层在图 2-13 表示为揭示表示空间密集程度的网格，同时中间层用两条竖线表示。FCN32s 是在单步上采样步长为 32 的预测值回到像素；FCN16s 是以 16 为步长，结合 pool4 层和最终层的信息，可以得到更精炼的细节信息，同时包含高层次的语义信息；FCN8s

更进一步，将 pool3 层的信息，以步长为 8，与之前信息结合用于提高精度。本次课程设计中我们采用的正是 FCN8s。

2.6.4 Snapshot Ensembles

Snapshot Ensembling^[19] 在一次训练过程中产生一组正确且具有多样性的分类模型，并进行集成。Snapshot Ensembling 的核心是在收敛到最终解之前寻找到多个局部最优。我们利用这些得到的局部最优解模型，在测试时平均它们的预测输出。

集成学习在每一个基模型有低测试误差且它们的错分类样本没有重叠。在大多数优化路径中，一个神经网络的权重赋值往往不等同于低测试误差。实际上，常常可以看到在只有在学习率下降的时候验证误差才显著下降，尤其是在几百个训练回合之后。我们的方法受到训练更好的回合和更早衰减学习率的启发，它们被证实可以减少对最终测试误差的影响。

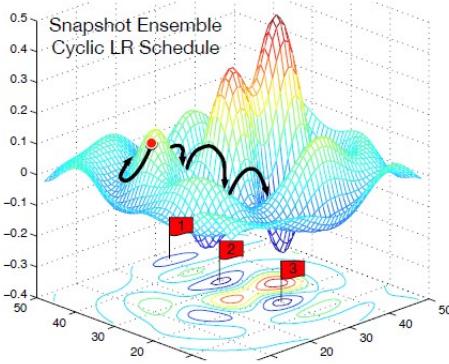
循环余弦退火 (Cyclic Cosine Annealing)：为了收敛到多个局部最优，我们采用循环余弦退火机制。我们以一个很快的速度降低学习率，鼓励模型在 M 个回合之内收敛到它的第一个局部最优。随后，优化过程继续以一个较大的学习率开始，即让模型从当前局部最优中跳出。我们重复几次上诉操作可以得到多个局部最优解。正式地说，对于学习率 α 有

$$\alpha(t) = f(\text{mod}(t - 1, [T/M])) \quad (2.50)$$

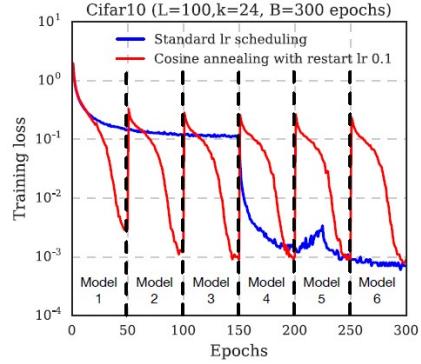
这里 t 是迭代次数， T 训练迭代总次数， f 是一个单调下降的函数。换句话说，我们将训练过程分为 M 循环，每一个循环以一个较大的学习率开始，逐渐退火到一个较小的学习率。这个较大的 $\alpha = f(0)$ 必须能够提供足够的能量让模型从当前局部最优中跳出，同时较小的学习率 $\alpha = f([T/M])$ 让模型收敛到一个较好的局部最优解。在这里，我们设定 f 是移位余弦函数，即

$$\alpha(t) = \frac{\alpha_0}{2} \left(\cos \left(\frac{\pi \text{mod}(t - 1, [T/M])}{[T/M]} \right) + 1 \right) \quad (2.51)$$

这里 α_0 是初始学习率。直觉上讲，这个函数在一个训练循环中从初始学习率退火到接近 0。我们在每一次迭代中就更新学习率，这样可以提高在少回合情况下的快速收敛。



(a) Snapshot ensembles 的收敛路径



(b) Snapshot ensembles 的学习率曲线

图 2-14 Snapshot ensembles

Snapshot Ensembling: 在每一次训练循环的末尾，很显然这时模型已经找到了一个局部最优解。这样，在提升学习率之前，我们将这时的模型保存下来，成为 Snapshot。在 M 个循环之后，我们得到了 M 个模型，即 f_1, \dots, f_M 。更重要的是等到这一系列模型的时间与之前用标准训练方法得到一个模型的时间一样。

在测试环节中，集成输出是将最后 m 个模型的输出进行平均。令 x 是一个测试样本， $h_i(x)$ 是第 i 个模型的 softmax 输出。输出的结果为

$$h_{\text{Ensemble}} = \frac{1}{m} \sum_{i=0}^{m-1} h_{M-i}(x) \quad (2.52)$$

第二章主要介绍了我们这次课程设计所使用的主要原理，下面将介绍本次课程设计的实验系统设计方案。

3 实验系统设计

3.1 总体算法流程图

在本次课程设计中，我们将花卉识别分为3个模块：分割模块、特征提取模块和分类模块，图3-1所示是此次所采用的对应模块所采用的算法和大致流程。

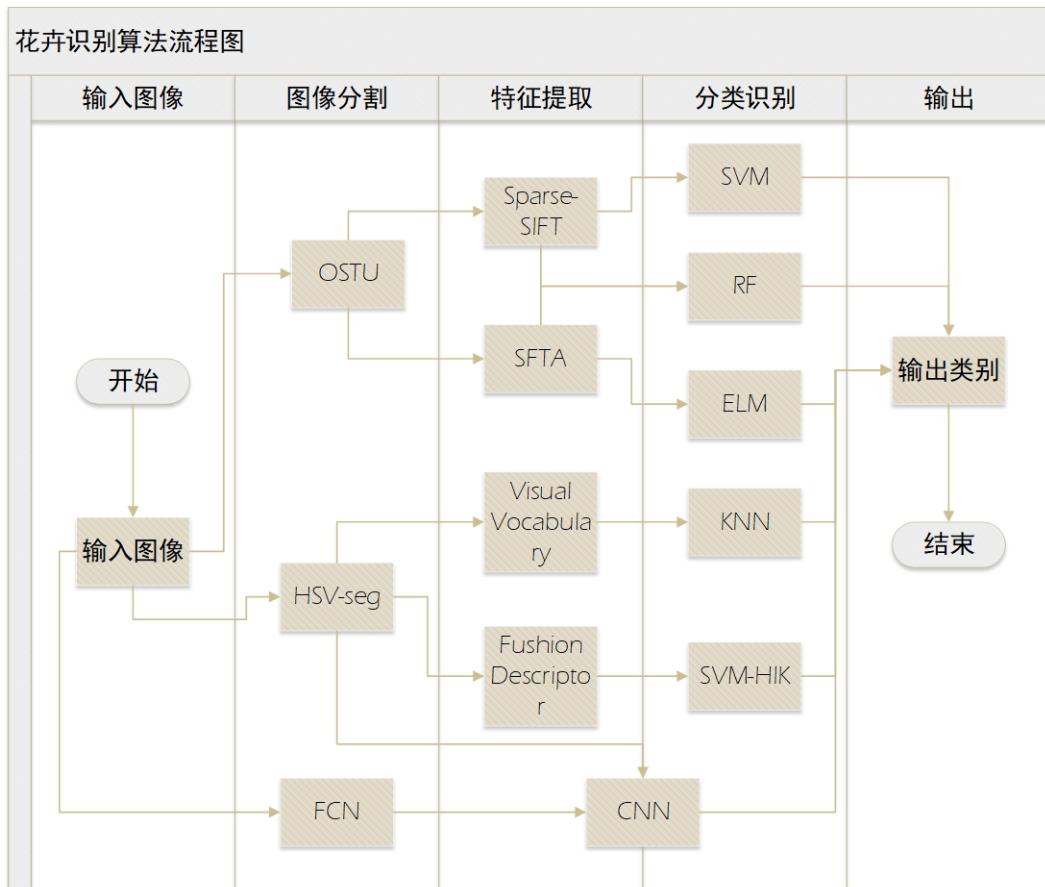


图3-1 总体算法流程图

如图3-1所示，本次课程设计采用了多种花卉分类方法，意图在于比较各种方法的优缺点和差异，即

- 方案一：基于视觉词汇的花卉识别方法^[3]（HSV-seq+Visual Vocabulary+KNN）；
- 方案二：基于融合特征描述子的花卉识别方法^[10]（HSV-seq+Fusion Descriptor+SVM-HIK）；
- 方案三：基于Sparse-SIFT的花卉识别方法^[1]（OSTU+Sparse-SIFT+SVM/RF/ELM）；

- 方案四：基于 SFTA 的花卉识别方法^[1]（OSTU+SFTA+SVM/RF/ELM）；
- 方案五：基于卷积神经网络的花卉识别方法^[3]；
- 方案六：基于迁移学习的卷积神经网络的花卉识别方法^[20]；
- 方案七：基于全卷积神经网络和迁移学习的花卉识别方法^[21]（FCN+CNN）。

下面几个小节将逐一介绍每一种方案的各个模块的具体细节。

3.2 方案一：基于视觉词汇的花卉识别方法

3.2.1 算法流程

基于视觉词汇的花卉识别方法是对一幅花卉图像分别提取其颜色、形状和纹理特征，其中颜色特征是使用 HSV 颜色空间；形状特征使用 SIFT 算子；纹理特征使用 LBP 算子。在应用过程中，对于 HSV 颜色空间和 LBP 算子运算过后的图像，我们不采用逐个像素进行聚类，而对每行和每列分别求一个均值和协方差矩阵，将均值和铺平后的协方差矩阵拼接成一个向量，作为这一行或列的特征表示。如图 3-2 所示是方案一的算法流程图：

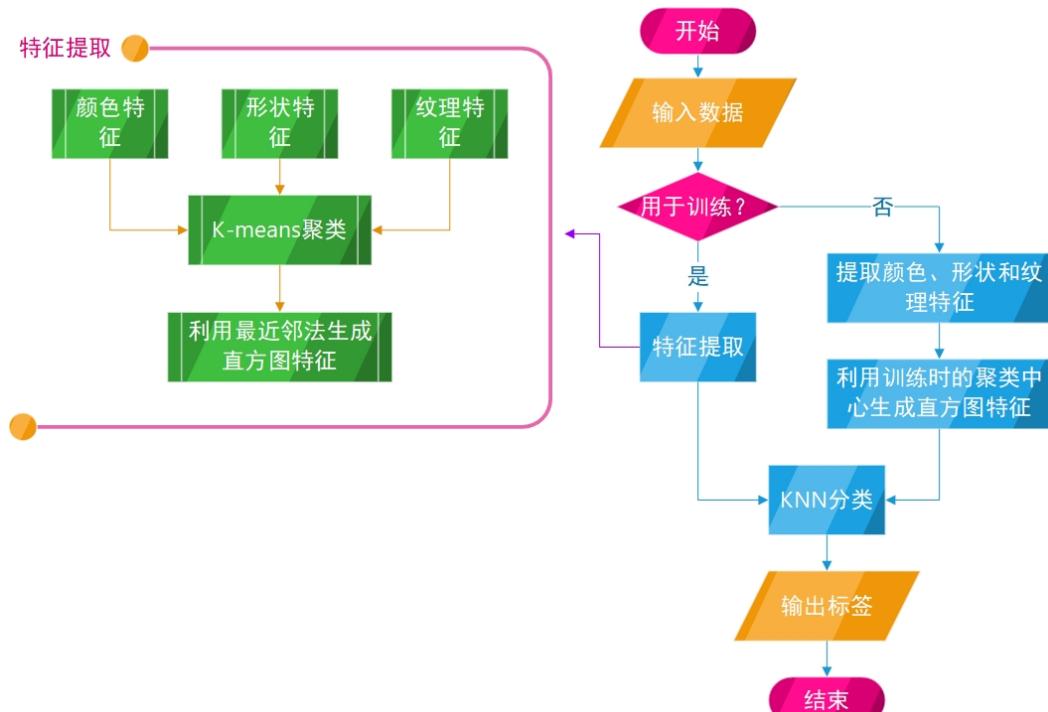


图 3-2 基于视觉词汇的花卉识别算法流程图

3.2.2 程序模块说明

关于方案一的模块说明如表 3-1 所示。

表 3-1 方案一的程序模块说明

程序文件名或函数名	模块说明
main.py	该文件是方案一的主函数模块，包含该方案算法流程的全过程：读取文件、特征提取、分类识别输出和测试正确率，得到最终结果。
randomSample	在花卉数据集中每一个类别文件夹中，随机生成索引选择出指定个数的样本作为训练样本，其余作为测试样本
load	加载数据，其中参数 mode="color" 表示提取颜色特征；mode="shape" 表示提取形状特征；mode="texture" 表示提取纹理特征，最终返回样本数据和每一幅图像提取的特征个数组成的数组。
chi2_distance	计算 χ^2 统计量距离
plot_embedding	用于 t-SNE 的可视化函数
segmentation	分割图片，利用基于 HSV 的分割算法分割花卉图像，并返回分割后的图像
xkNN	基于 χ^2 统计量距离的单一特征的 KNN 分类器
bxkNN	基于 χ^2 统计量距离的联合特征的 KNN 分类器
ekNN	基于欧式距离的 kNN 分类器
featureCOV	用于测试样本提取颜色特征
featureSIFT	用于测试样本提取形状特征
featureLBP	用于测试样本提取纹理特征

程序对应在文件夹 test 中。

3.3 方案二：基于融合特征描述子的花卉识别方法

3.3.1 算法流程

基于融合特征描述子的花卉识别方法是通过提取颜色和纹理特征，同时也考虑形状特征的作用，利用 PHOW 特征生成框架生成对应的特征；提取颜色特征时，主要使用 denseSIFT 算子；提取纹理特征时，主要使用在灰度图上的 denseSIFT 算子；提取形状特征时，则使用基于 edgeSIFT 的方法。在提取特征之后，采用 LLC 编码方式，主要考虑特征样本的局部分布约束，保持局部距离。图

3-3 所示是该方案的算法流程图。

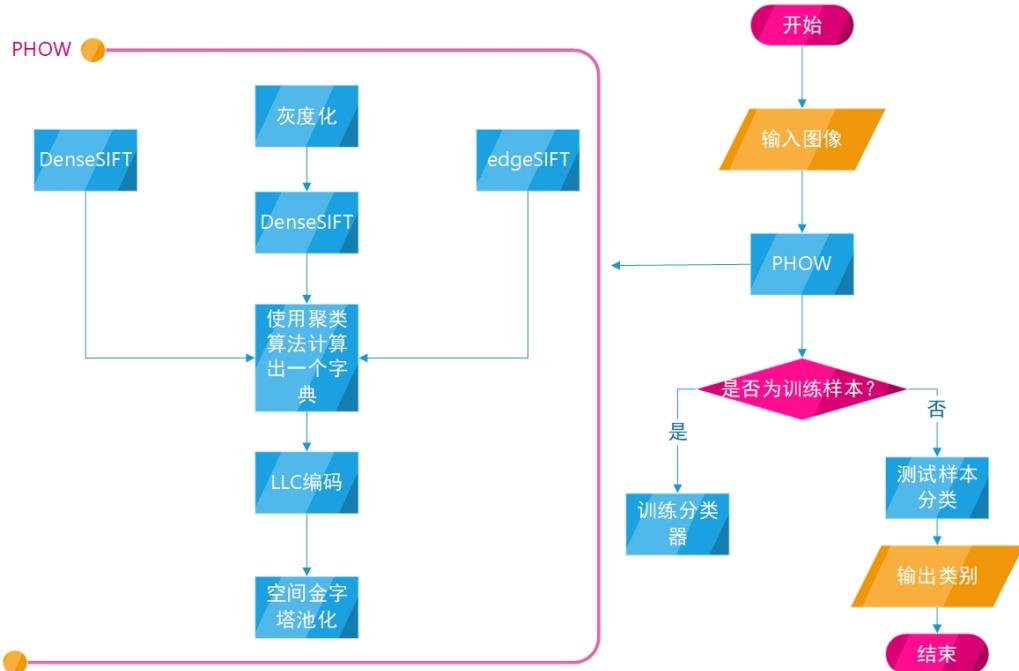


图 3-3 基于融合特征描述子的花卉识别算法流程图

3.3.2 程序模块说明

关于方案二的模块说明如表 3-2 所示。

表 3-2 方案二的程序模块说明

程序文件名或函数名	模块说明
main.py	该文件是方案二的主函数模块，包含该方案算法流程的全过程：读取文件、特征提取、生成特征数据文件、（直接读取生成特征数据文件）、分割训练集和测试集、分类识别输出和测试正确率，得到最终结果。
dSIFT.py	该文件定义了方案二所采用的 denseSIFT 特征提取算子，其余文件可直接调用
matrixPow	矩阵乘方函数，即计算 X^Y
splitImage、splitGEImage	按空间金字塔要求将图像 4 等分
segmentation	图像分割函数，使用 HSV-sep 分割算法
getCodebook	通过聚类获得编码字典
LLCoding	计算 LLC 编码
saveCodebook	保存编码字典

续下页

续表 3-2 方案二的程序模块说明

程序文件名或函数名	模块说明
extractHSVCodebook、 extractGCodebook、 extractECodebook	提取 HSV、Gray 和 edge-SIFT 的编码字典和特征编码
PHOW_HSV、 PHOW_G、PHOW_E	提取 HSV、Gray 和 edge-SIFT 的 PHOW 特征
saveHSVcode、saveG- code、saveECode	保存 HSV、gray 和 edge-SIFT 的特征编码
msvm.py	定义 HIK 核函数，并用于 SVM 分类器

程序对应在文件夹 fsvm 中。

3.4 方案三：基于 Sparse-SIFT 的花卉识别方法

3.4.1 算法流程

基于 Sparse-SIFT 的花卉识别方法是提取图像的 SIFT 特征，再利用 K-SVD 将其进行稀疏编码。在本次课程设计的过程中，我们采用 3.3 节中已经有的 denseSIFT 特征算子提取图像的特征，再进行 K-SVD 稀疏编码。图 3-4 所示为方案三的算法流程图。

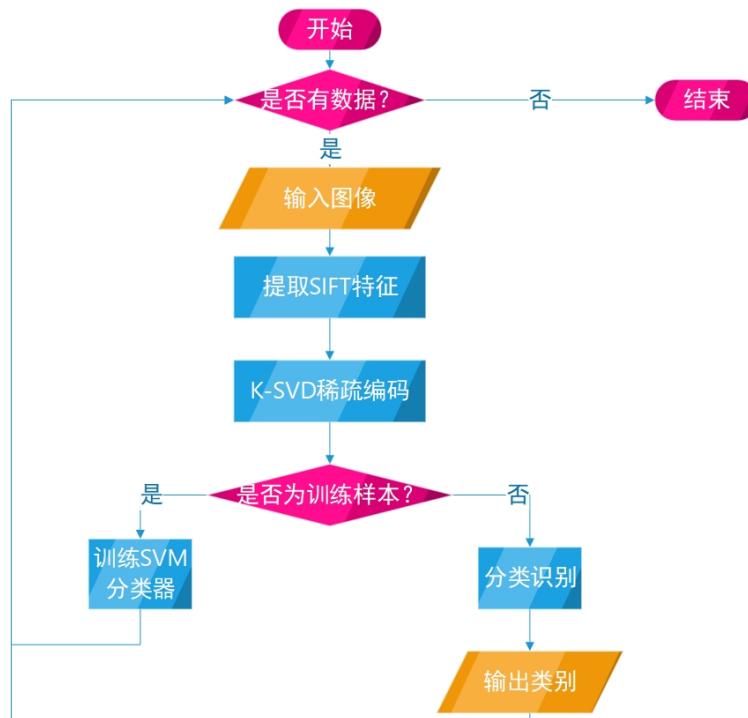


图 3-4 基于 Sparse-SIFT 的花卉识别算法流程图

3.4.2 程序模块说明

关于方案三的模块说明如表 3-3 所示。

表 3-3 方案三的程序模块说明

程序文件名或函数名	模块说明
main.py	该文件是方案三的主函数模块，包含该方案算法流程的全过程：读取文件、特征提取、生成特征数据文件、（直接读取生成特征数据文件）、分割训练集和测试集、分类识别输出和测试正确率，得到最终结果。
plot_embedding	t-SNE 的可视化绘图函数
loadSIFT	用于 SIFT 特征提取的数据加载函数，读入图片并将其转化为灰度图用于后续 SIFT 特征提取
sSIFT.py	该文件定义了方案三所采用的 denseSIFT 特征提取算子，其余文件可直接调用。先用滑动窗口将图像划分为多个重叠的网格，再在每一个小窗口上提取 SIFT 特征，再进行整合计算得到最终结果
ksvd.py	该文件定义了方案三采用的稀疏编码方式 K-SVD
segment	基于 OTSU 方法的图像分割函数，这里为了提高运算速度我们使用了 opencv 自带的 OTSU 阈值分割方法
SIFT	稀疏编码的 SIFT 特征提取函数，先提取 SIFT 特征，再将其用 K-SVD 稀疏编码方式得到 SIFT 特征的稀疏编码并返回稀疏编码特征

程序对应在文件夹 fSVMoRF 中。

3.5 方案四：基于 SFTA 的花卉识别方法

3.5.1 算法流程

基于 SFTA 的花卉识别方法是一种比较新颖的针对花卉的特征提取方法，主要针对纹理特征。主要分为两个部分：1) 把输入的灰度图像分解成一系列二值图像，这里采用两阈值二值化分解进行二值化；2) 对于上述得到的二值化图像，我们计算区域的边界的分形维度，另外在计算区域的平均灰度值和面积。对于多等级 OTSU 算法，寻找到阈值能够使输入的图像的类内方差最小化；然后，以递归的方式不断对每一幅输入图像区域应用 OTSU 算法直到指定个数的阈值；计算分形维度时，可以通过 Box Counting 算法计算分形维度的近似值，和平均灰

度值、区域面积一起拼接成最终的 SFTA 特征。

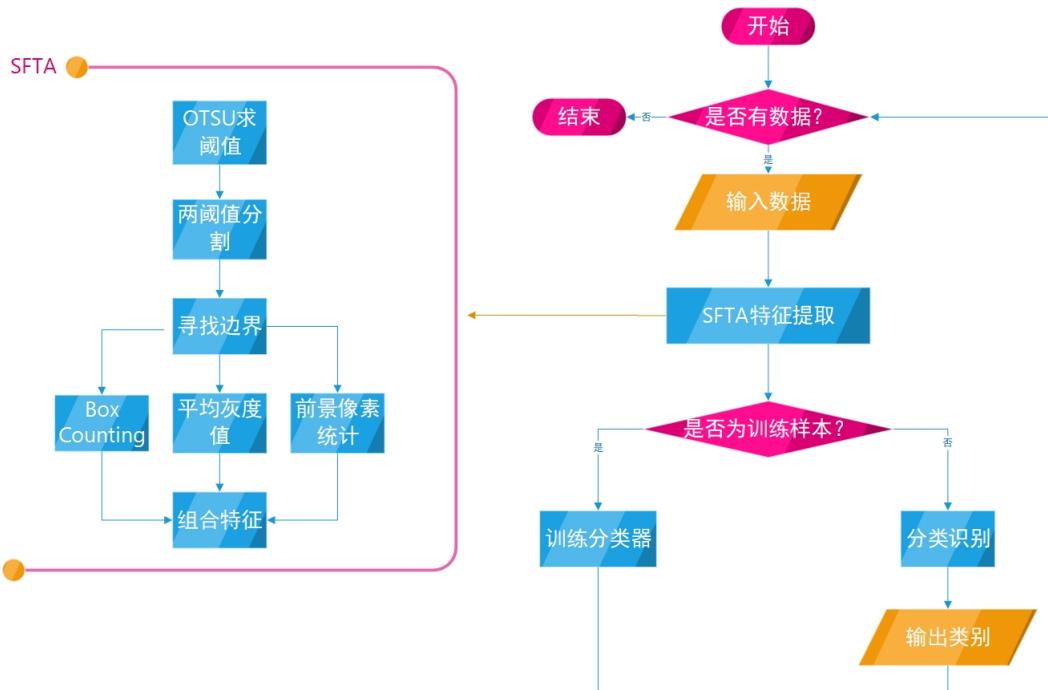


图 3-5 基于 SFTA 的花卉识别算法流程图

3.5.2 程序模块说明

关于方案四的模块说明如表 3-4 所示。

表 3-4 方案四的程序模块说明

程序文件名或函数名	模块说明
main.py	该文件是方案四的主函数模块，包含该方案算法流程的全过程：读取文件、特征提取、生成特征数据文件、（直接读取生成特征数据文件）、分割训练集和测试集、分类识别输出和测试正确率，得到最终结果。
plot_embedding	t-SNE 的可视化绘图函数
load	用于 SFTA 特征提取的数据加载函数，加载文件图片并缩放到 330×250 ，并提取 SFTA 特征
sfta.py	定义提取 sfta 特征的一系列函数
elm.py	极限学习机模块，定义极限学习机模型，并读取已经得到的特征进行分类
pseudoInv.py	用于极限学习机训练的伪逆求解模块，通过迭代方式求解伪逆

程序对应在文件夹 fSVMoRF 中。

3.6 方案五：基于卷积神经网络的花卉识别方法

基于卷积神经网络的花卉识别方法是利用卷积神经网络的方法，通过神经网络自动提取特征和完成分类。实践证明，深度卷积神经网络的分类性能会高于人工提取的特征得到的分类性能。在训练网络过程中，我们对学习率的选择进行了改变，采用了 snapshot ensembles 方式，可以训练一次得到多个模型进行集成得到结果，后面两种方案也同样采取这种方式。图 3-5 所示是本方案所使用的卷积神经网络结构图。

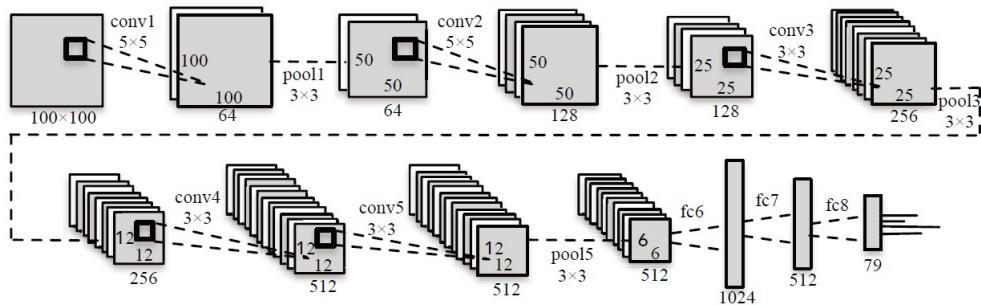


图 3-6 方案五的卷积神经网络结构图

图像通过一组卷积层。第一卷积层有 64 个卷积核，大小为 $5 \times 5 \times 3$ ，输入图像的大小为 $100 \times 100 \times 3$ 。第二、三层卷积层网络分别包含 128、256 个卷积核，其大小分别为 $5 \times 5 \times 64$ 和 $3 \times 3 \times 128$ 连接到前一卷积层的输出，包括归一化和池化。第四卷积层把池化后的第三卷积层的输出作为输入，带有 512 个卷积核，其大小为 $3 \times 3 \times 256$ ，第五卷积层包含 512 个卷积核，其大小为 $3 \times 3 \times 512$ 。卷积核的步长为 1 个像素，对于 5×5 的卷积核补全宽度为 2 个像素，对于 3×3 的卷积核补全宽度为 1 个像素。在一系列卷积层之后接入到全连接层中，前两层分别包含 1024 和 512 个神经元，最后一层的神经元个数与对应分类问题的类别一致。空间池化由 4 个最大池化层完成，池化核大小为 3×3 ，步长为 2 个像素。在所有卷积层和全连接层后接入 ReLU 非线性激活函数。

3.7 方案六：基于迁移学习的卷积神经网络的花卉识别方法

迁移学习是当前机器学习中一个十分重要的分支。迁移学习的最初意图受到人类思维的启发，人们在面对相似但从未见过的问题时可以根据以往的经验对新的问题作出判断，这过程往往具有很高的确信度。ImageNet 是一个庞大的图像数据库，其中包含很多很多种类的图像。显然，花卉识别与 ImageNet 中的

图像识别是相似的任务，即可以用 ImageNet 训练好的神经网络模型用来对花卉进行识别。由于花卉识别与 ImageNet 的识别任务事实上有一定的差别，可以对已经训练好的神经网络模型进行微调，这里我们将 VGG16 的神经网络模型的最后一个全连接层替换成自定义的全连接层，输出神经元个数与我们所需分类的类别数相同。我们固定除自定义层参数之外的其他参数，用 ImageNet 已经训练好的模型参数对应代入，再对最后一层进行训练从而完成对神经网络的微调，从而完成训练过程，实现花卉识别。

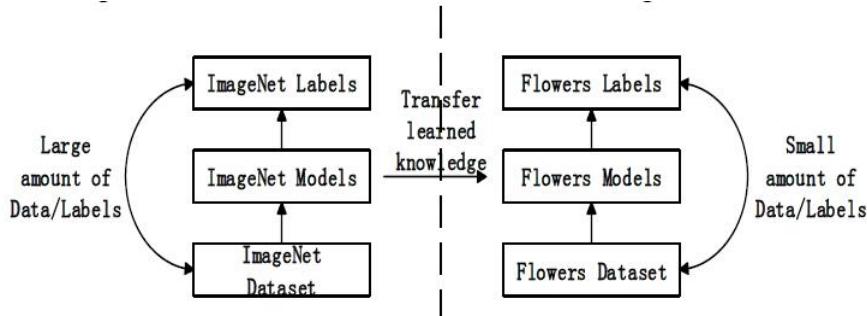


图 3-7 迁移学习示意图

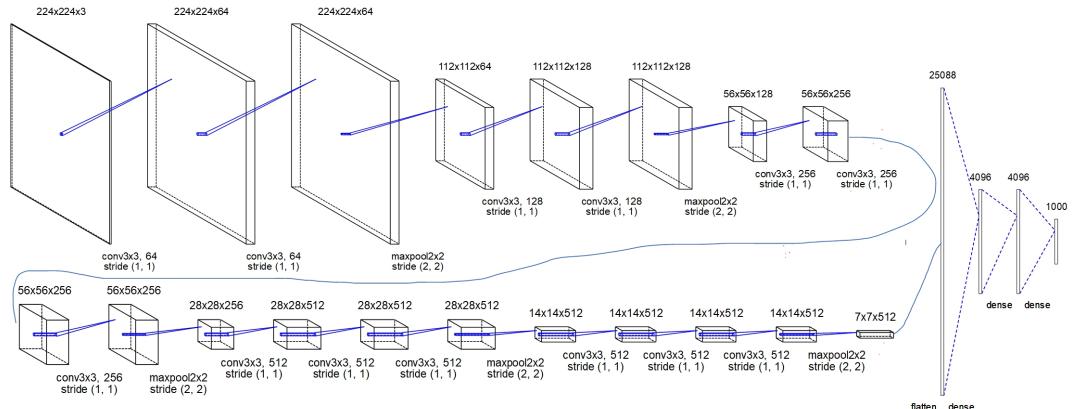


图 3-8 VGG16 神经网络结构图

3.8 方案七：基于全卷积神经网络和迁移学习的花卉识别方法

全卷积神经网络是用于图像分割的有力工具。我们同样借用迁移学习的思想，对全连接神经网络使用预训练好的 VGG16 模型确定全卷积神经网络的卷积层参数，再通过训练后面的解卷积层得到最终的全卷积神经网络模型。后面对于用 FCN 进行图像分割的图片再用方案六所述的基于迁移学习的卷积神经网络进行训练和得到分类结果。图 3-9 所示是该方案的整体流程图，图 3-10 所示是使用预训练模型参数的神经网络参数结构图。

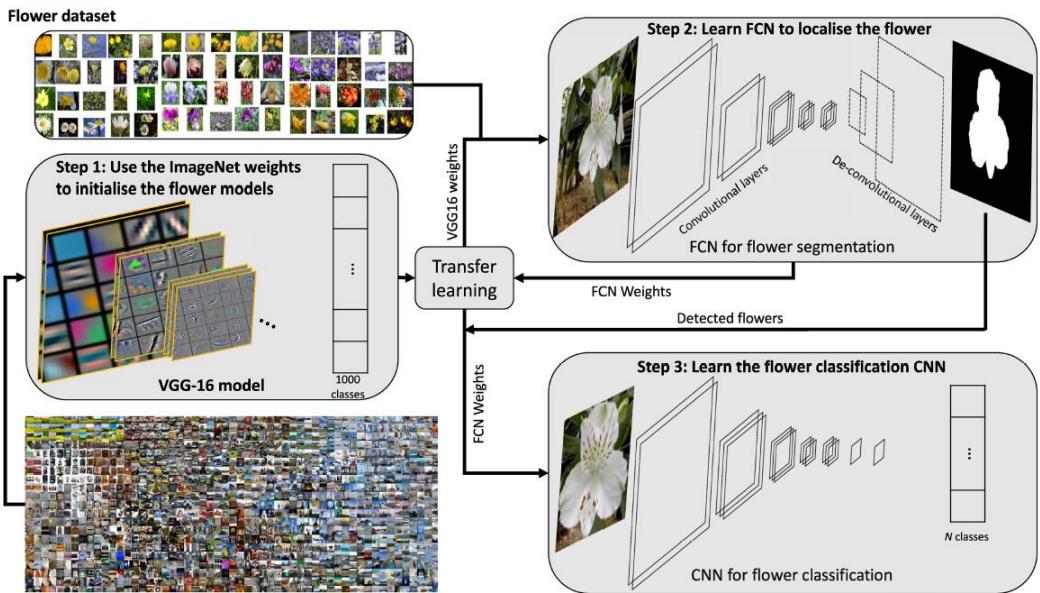


图 3-9 花卉分割和识别框架

Block 1	Block 2	Block 3	Block 4	Block 5	Deconvolution Block
Layers <ul style="list-style-type: none"> Conv1_1 ReLU1_1 Conv1_2 ReLU1_2 MaxPool1 Main parameters <ul style="list-style-type: none"> 3x3x64 feature maps Pool stride=2 	Layers <ul style="list-style-type: none"> Conv2_1 ReLU2_1 Conv2_2 ReLU2_2 MaxPool2 Main parameters <ul style="list-style-type: none"> 3x3x128 feature maps Pool stride=2 	Layers <ul style="list-style-type: none"> Conv3_1 ReLU3_1 Conv3_2 ReLU3_2 Conv3_3 ReLU3_3 MaxPool3 Main parameters <ul style="list-style-type: none"> 3x3x256 feature maps Pool stride=2 	Layers <ul style="list-style-type: none"> Conv4_1 ReLU4_1 Conv4_2 ReLU4_2 Conv4_3 ReLU4_3 MaxPool4 Main parameters <ul style="list-style-type: none"> 3x3x512 feature maps Pool stride=2 	Layers <ul style="list-style-type: none"> Conv5_1 ReLU5_1 Conv5_2 ReLU5_2 Conv5_3 ReLU5_3 MaxPool5 Main parameters <ul style="list-style-type: none"> 3x3x512 feature maps Pool stride=2 	Layers <ul style="list-style-type: none"> Deconv1_1 Deconv1_2 Deconv1_3 Main parameters <ul style="list-style-type: none"> Deconv1_1 stride=2 Deconv1_2 stride=2 Deconv1_3 stride=8
Block 6 Layers <ul style="list-style-type: none"> Fully connected ReLU6 Dropout Main parameters <ul style="list-style-type: none"> # output = 4096 Dropout ration=0.5 	Block 7 Layers <ul style="list-style-type: none"> Fully connected ReLU7 Dropout Main parameters <ul style="list-style-type: none"> # output = 4096 Dropout ration=0.5 	Block 8 Layers <ul style="list-style-type: none"> Fully connected Main parameters <ul style="list-style-type: none"> # output = 1000 			

图 3-10 神经网络参数和结构示意图

本章主要介绍了本次课程设计的实验系统方案设计，主要陈述所采用的几个方案的主要流程和一些程序模块说明。接下来我们将展示本次课程设计的实验结果并进行分析。

4 实验结果分析

4.1 图片实验示例

4.1.1 图片分割示例

这里我们展示图片的分割效果，用于测试的图片是我们实地真实拍摄的花卉图片，用于测试识别系统的泛化能力。



(a) 原始图像 (b) 基于 HSV 的分割 (c) 基于 OTSU 的分割 (d) 基于 FCN 的分割

图 4-1 图片分割示例

可以看出，上述花卉分割的方法是基于花朵的部分颜色比较鲜艳而背景比较暗淡，而我们是在白天拍摄的图片，天空背景等其余目标会极大影响分割的效果。

从图 4-1 中看出，基于 HSV 颜色空间的图像分割算法和基于 OTSU 的颜色分割算法对于天空背景表现不佳，且基于 OTSU 的颜色分割算法对于复杂背景的表现较差，对于非花目标物体没有去除；而基于 FCN 的图像分割算法的结果较前两种算法有有一定的提升，但由于训练时的目标掩图存在噪声会影响最终的 FCN 模型的分割效果，但其对天空背景有一定的分割能力，但其分割效果缺乏解释性，对于一些分割出错的地方缺乏解释性。

4.1.2 图片识别示例

将 4.1.1 中的测试图片放入实验系统中，测试系统的分类能力，表 4-1 所示是每一种算法给出的分类结果，其中图片编号对应图 4-2 中所示的图片。

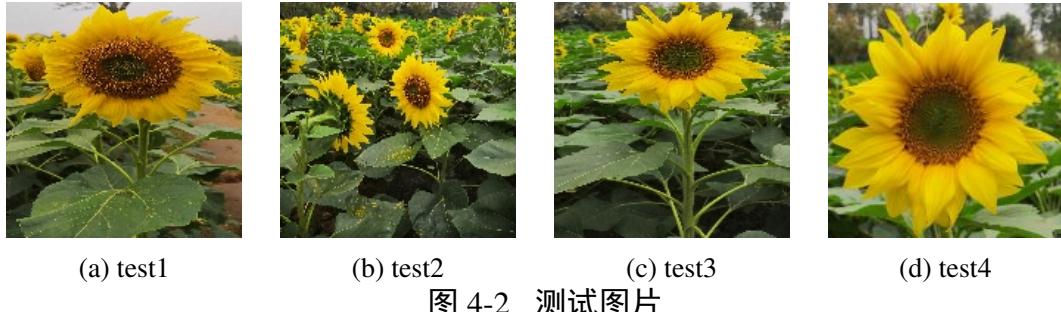


图 4-2 测试图片

第三部分所述的六种方案^①对应的缩写为：VCB、FD、SFTA、CNN、TF-CNN 和 FCN-CNN，本文以后部分均按照这种缩写格式书写，不再另行说明。

表 4-1 图片分类示例

图片编号	1	2	3	4
真实标签	Sunflower	Sunflower	Sunflower	Sunflower
VCB	Sunflower	Buttercup	Sunflower	Sunflower
FD	Sunflower	Lilyvalley	Lilyvalley	Sunflower
SFTA	Lilyvalley	Lilyvalley	Sunflower	Sunflower
TF-CNN-	Sunflower	Iris	Lilyvalley	Sunflower
VGG16				
TF-CNN-	Buttercup	Buttercup	Buttercup	Buttercup
RESNET18				
TF-CNN-	Buttercup	Iris	Buttercup	Sunflower
INCEPTION-				
V3				
TF-CNN-	Buttercup	Buttercup	Buttercup	Windflower
RESNET50				
TF-CNN-	Buttercup	Buttercup	Buttercup	Buttercup
RESNET101				

可以看出实际拍摄的图片与用于训练的图片有较大的差距，我们拍摄的向日葵的图片，观察给定数据集可知，大部分向日葵图片中花朵处于图片中央且仅有少部分包括绿色的叶子和根茎背景；而拍摄的图片中背景嘈杂，且分割算法对

^①方案三基于稀疏编码的 SIFT 特征算法实现时由于论文细节过少，未能达到预期效果，故而不展示其对应的结果。

于天空背景的去除效果较差，且花朵均不处于图片中央，其中还有一张图片有多个向日葵花朵，对分类器具有极大的迷惑性。而且训练集过小，深度神经网络十分容易过拟合，所以对目标图片的泛化能力大幅下降。

对于上述问题，可以对图片作花朵检测，对于框出的花朵区域再进行分类，预期可以获得较好的实验结果。限于时间和能力的限制，包括没有已经标注用于检测的花卉数据集，本次课程设计并没有完成相关工作，后续可以考虑采用迁移学习的方法利用其他的数据集训练的模型迁移到花朵数据集上用于花卉检测。

4.2 实验结果及其分析

按照课程设计要求，我们将数据集每一类的 40 张图片作为训练集，其余图片作为测试集，重复实验 5 次，计算分类正确率的均值和方差。本次课程设计中，我们的分类数据集包括 6 类分类数据集和 102 类分类数据集，但是由于 102 类分类数据集数据量庞大，传统方法计算代价大，所以没有测试。下面分别介绍每一种方案所达到的正确率结果。

4.2.1 基于手动特征提取的花卉识别方法

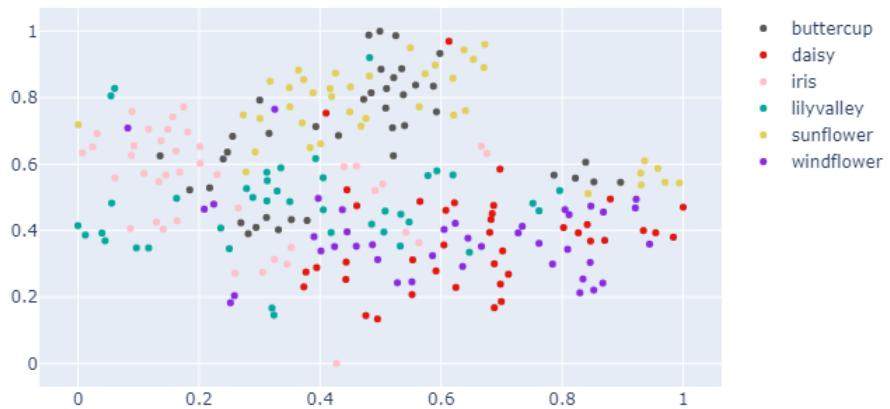
方案一是基于视觉词汇提取特征和 KNN 算法的花卉识别方法，其结果如表 4-2 所示，表中 N-代表没有进行花卉分割，S-代表进行花卉分割；clr-表示使用的是颜色特征，sha-表示使用的是形状特征，tex-表示使用的是纹理特征，all-表示使用的是上述三种特征的混合。

表 4-2 方案一的实验结果

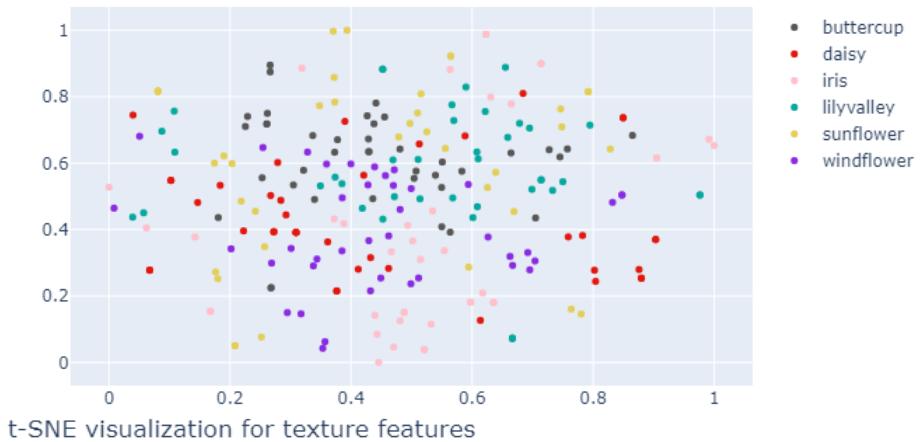
方法	N-clr	N-sha	N-tex	N-all	S-clr	S-sha	S-tex	S-all
均值	31.92	32.42	15.58	29.08	61.5	47.5	30.58	64.25
方差	4.836	1.676	2.306	2.754	2.033	2.571	4.850	3.284

可以看出，基于视觉词汇的花卉识别方法中颜色特征的可靠性最高，而形状和纹理特征提取的具有区分性的信息较少，图 4-3 所示是对提取的颜色、形状和纹理特征的可视化结果，从中可以看出颜色特征的可区分性比形状特征和纹理特征的可区分性高很多。从图 4-3 中看出 Buttercup 和 Sunflower 的颜色特征混杂在一起，因为这两种花的颜色均为黄色，所以仅通过颜色特征无法区分这两种花。

t-SNE visualization for color features



t-SNE visualization for shape features



t-SNE visualization for texture features

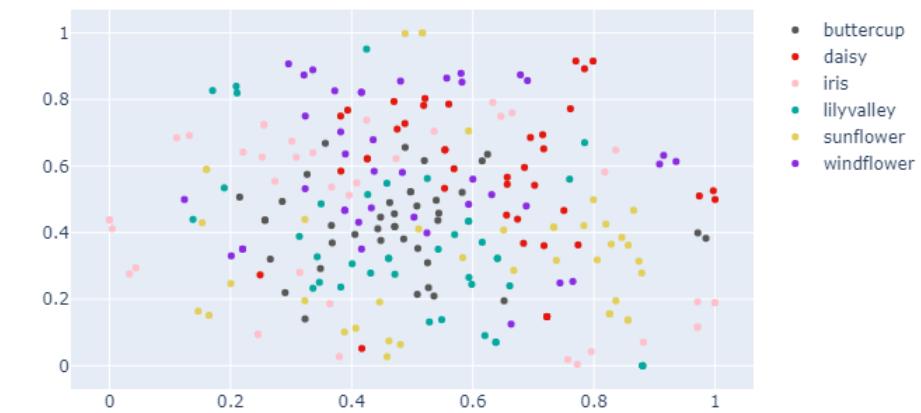


图 4-3 方案一的数据可视化

通过表 4-2 中数据可以看出分割算法具有提高分类正确率的效果，去除了背景的干扰可以提高相应任务的分类正确率。正如前文中提到的一样，原有论文中的方法速度较慢，运算代价大，我们将形状和纹理的提取方法做了一定的改变，从结果中可以看出，导致形状和纹理特征对分类效果没有起到积极的作用。

方案二是基于融合特征的花卉识别方法，图 4-4 所示是融合特征的花卉识别方法的数据可视化，可以看出由于数据维度较高，t-SNE 数据可视化降维到 2 维可视化时不能完全揭示其可分性，所以从图中看出数据的可分性比方案一中的颜色特征的可分性差。

t-SNE visualization for Fusion features

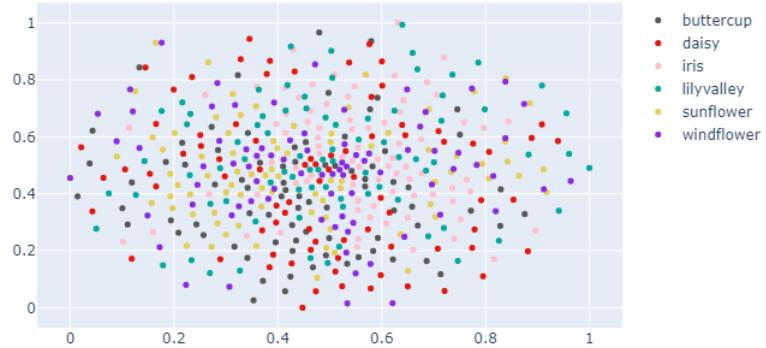


图 4-4 方案二的数据可视化

表 4-3 所示是方案二的分类正确率结果，其中 n- 表示没有进行图像分割，s- 表示进行图像分割； -rbf 表示使用 RBF 核， -hik 表示使用 HIK 核。

表 4-3 方案二的实验结果

方法	n-rbf	n-hik	s-rbf	s-hik
均值	43.50	56.50	41.33	61.17
方差	3.684	8.713	4.395	4.659

其中 s-hik 对应的方法表现的最好，其对应的混淆矩阵如图 4-5 所示，

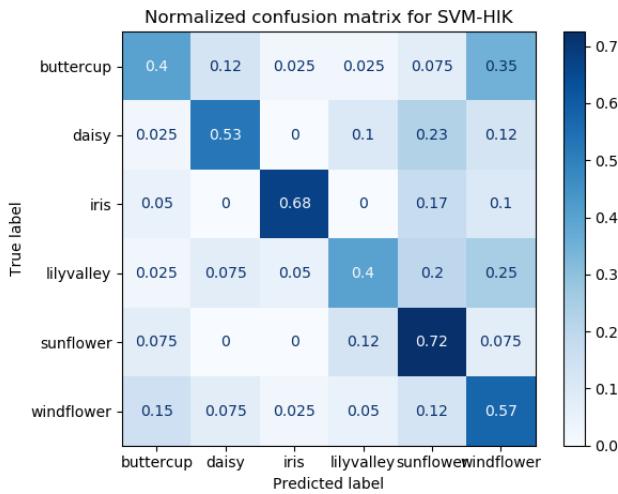


图 4-5 方案二的混淆矩阵 (s-hik)

可以看出，将 Buttercup 错分为 windflower 的可能性较高，Sunflower 正确分

类的概率最高。

方案四是基于 SFTA 特征的花卉识别方法，图 4-6 所示是 SFTA 特征的数据可视化，

t-SNE visualization for SFTA features

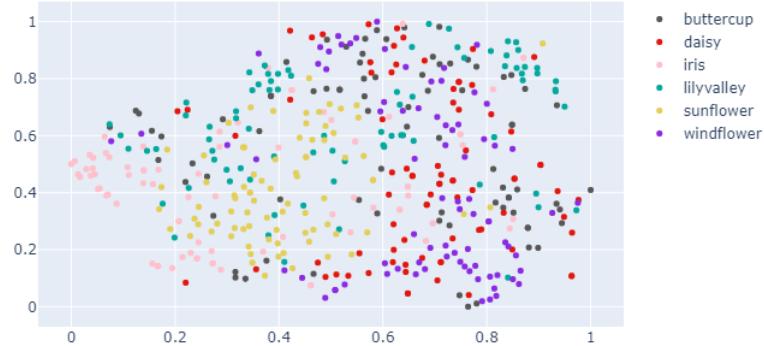


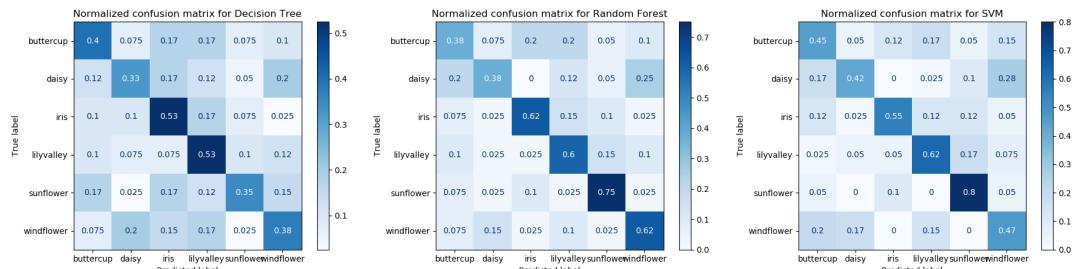
图 4-6 方案四的数据可视化

方案四采用决策树、随机森林和支持向量机三种分类算法，其正确率如表 4-4 所示，其中 DT 表示决策树（Decision Tree），RF 表示随机森林（Random Forest），SVM 表示支持向量机（Support Vector Machine）。

表 4-4 方案四的实验结果

方法	SFTA-DT	SFTA-RF	SFTA-SVM
均值	41.67	55.83	55.42
方差	3.684	2.713	3.395

可以看出基于 SFTA 特征的随机森林方法的分类效果最好，图 4-7 所示分别对应上述三种方法的混淆矩阵，



(a) 决策树的混淆矩阵

(b) 随机森林的混淆矩阵

(c) 支持向量机的混淆矩阵

图 4-7 方案四的分类混淆矩阵

从中可以看出，对于 SFTA 特征，随机森林分类算法的可区分性最高，其与决策树进行比较，可以看出集成学习的作用，通过集成几个弱分类器来提高分类器性能。

4.2.2 深度学习方法

本次课程设计中，除了测试了一种简单的卷积神经网络模型，我们主要采用了基于迁移学习的卷积神经网络模型并取得良好的分类正确率和稳定性，主要包含 VGG16、Inception-v3、RESNET18、RESNET50 和 RESNET101 五种常用的图像识别模型，表 4-5 和表 4-6 分别是上述五种模型在 6 类分类数据集和 Oxford-102 数据集上的结果。

表 4-5 卷积神经网络在 6 分类数据集上的结果

方法	VGG16	INCEPTION-V3	RESNET18	RESNET50	RESNET101
均值	75.58	84.58	90.92	93.00	95.17
方差	2.482	1.264	0.311	0.310	0.206

表 4-6 卷积神经网络在 Oxford 102 数据集上的结果

方法	VGG16	INCEPTION-V3	RESNET18	RESNET50	RESNET101
均值	49.62	64.53	72.48	75.69	76.08
方差	0.780	0.447	0.633	0.567	0.448

为了对比手动提取特征的方法（方案一、二、三和四）和深度学习方法，我们尝试提取网络中的一些数据作为提取的花卉的特征，将数据作为二维图像可视化来进行比较分析。

我们以 Inception-V3 这一模型的输出结果为例来进行进一步的分析。由于靠前的层数的数据不具有直观性且数据庞大，那么我们选择用最后的全连接层的数据 (2048×1) 作为神经网络所提取的特征。相当于每一幅图像都有一个 2048 维度的特征向量。那么为了更加直观的分析，我们采用 t-SNE 的方法，将特征维度降低到 2，可以在二维平面中直观地观察到结果。相对于手动提取的特征，很明显神经网络模型提取的特征的可分性要好得多。从最终的结果也可以看出来，对于六种花卉的识别率可以达到将近百分八十五，相对于之前的算法的特征要好很多。

如图 4-8(a) 所示是 Inception-V3 模型提取特征的可视化，图 4-8(b) 是将得到的特征放入 SVM 分类器中分类得到的混淆矩阵。

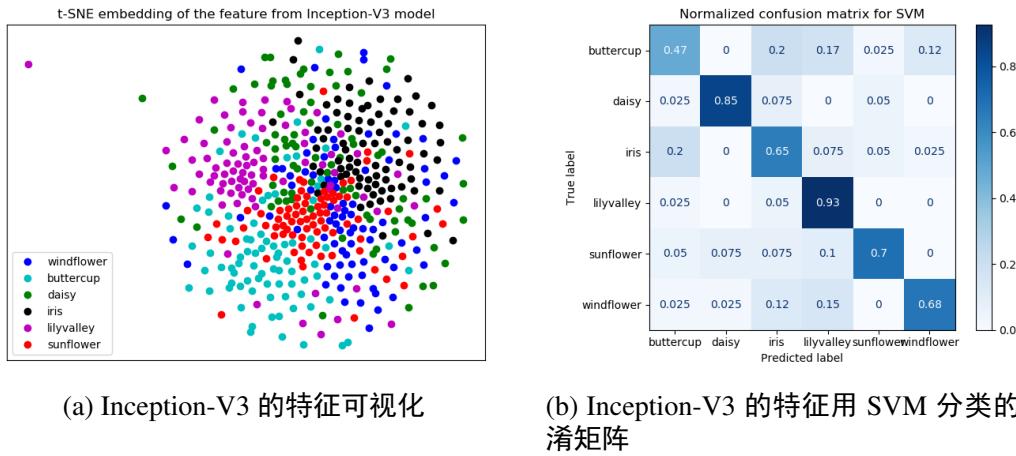


图 4-8 Inception-V3 模型的特征可视化分析

从上图中我们可以很清楚地看出，Livyvalley 和其他花朵相对来说和其他没有什么重叠，得到的准确率比较高。我们分析可能是他的颜色形状和其他的很不一样，相对于其他花朵图片都是一朵大花，Livyvalley 是一串小花。其次是百分之八十五正确率的 Daisy，相对于其他类型花朵的图片，这组图片花朵拍摄清晰，花朵均为正面拍摄，且花朵均为白色，肉眼可见这组图片中的花朵都较为相似，所以得到的结果也会更好。Buttercup 的准确分辨率最低，也就是这样得出的特征这种花是最差的，从可视化图片中也可以看出来相应的颜色小点也是最为分散的一组。我们分析原因是 Buttercup 中花朵的形状有正面也有侧面，花朵在照片中的占比不同，而且各种花朵拍照的光线差距较大，即总体形态相差较大。Buttercup 和 Iris 互相之间分别均有百分之二十的错误率。Iris 之中有各种各样颜色的花朵，其中黄色的花朵和 Buttercup 之中的花朵比较相似，而同为黄色花朵的 Sunflower 正确率较高，是由于其花朵的形态和其他区别较大。而 Iris 这种的花朵的分类结果差很大程度是由于这种花的颜色变化很多，结合之前的实验可以得到一个结果就是花朵的颜色在花朵分类之中占据了很大的比例。Windflower 有一大半是集中的，一小半是掺杂在其他类中，也和结果的百分之六十八比较吻合。分析相对来说其中只有一朵花且花朵较大的图片分类正确几率更高。而 Sunflower 的错误分类在二维特征中很难解释，除开黄色和其他有一定相似，这一种花朵形状完全不一致，按照预期推测可能会有更高的正确率，推测是因为直观观察采用的降维方法引起的信息丢失，可能相对在更高维度来说他们可以被更好的区分开来。有一个比较突出的问题就是对于一张照片中存在多株花朵的情况，这一种的分类效果就会下降很多。分析是由于模型无法分辨有几朵花朵，对花朵进行特征提取的时候就会受到花朵数量的影响，无法正确地提取一朵花。

朵的信息。相应的我们有想法先对所有图片进行单株花朵级别的图像分割，即保证每个输入的图片都是一朵花朵，但最后没有成功，但是这是一个可以进一步研究的方向。

另外我们尝试将未分割的花朵图片直接进行分类。很有趣的一个点是对于所给的六种花朵的分类（每张图片中都只有一朵花），在深度学习的模型中，背景对于花卉的分类效果的影响变得很小。我们思考是在这样的一个模型之中，已经起到了一个对花卉图片的分割的效果。

进一步采用更复杂的模型，如 RESNET101 模型，得到的正确率可以高达百分之九十五（见表 4-5）。我们尝试对其中提取的特征进行进一步的物理等实际意义上的分析，但是都以失败告终，未能得出有效结论。

当我们尝试进行更高更复杂的分类，如 Oxford102 数据集，卷积神经网络也可以达到一个较为良好的效果。正确率最高的为 RESNET101 模型，正确率也在百分之七十六以上。而对于如此复杂的数据集，受限于我们的相关专业知识，以手动提取特征的方法我们已经无能为力了。

5 总结与展望

本次课程设计我们选择的题目是基于图像的花卉识别。花卉识别是一个具有较高挑战性的问题，我们在选题时因其看起来目标清晰明了、数据集不是很庞大，认为该题目比较容易上手所以选定此题。在接触问题的初期，我们调研了相关文献，发现问题的难度超过了我们的预期。花卉识别的难点很多，比如自然界中的花朵种类很多，有些不同种的花朵长相十分相似，资深的植物学家可能都有误判的时候；而且同一种花在不同的生长周期会呈现出不同形态，对分类造成很大干扰。而且一般植物学家判断一朵花的种类，往往不仅仅依赖花朵的外观，包括其生长的地理位置、潮湿程度等等周围环境因素来综合判断该花的种类，所以基于图像的花卉识别理论上也存在一定的误判概率。

通过前期的调研，我们初步确定了 7 种解决方案，主要分为非深度学习方法和深度学习方法两大类。非深度学习方法主要的难点在于如何提取到优质的特征能够特异地表示每一类花卉；而深度学习方法主要的难点集中在如何确定网络结构。在本次课程设计中，我们将深度学习与迁移学习相结合，借助已有网络模型在 ImageNet 上的训练参数，大大减少了我们训练所需花费的时间和缓解了我们数据集过小的困境。

在编写代码的过程中，我们发现将论文中方法转换为实际的程序并能够达到论文宣称的结果是具有难度的。尤其是在非深度学习方法中，例如前文我们采用的方案一中的方法，提取图像的颜色、形状和纹理特征，采用 k-means 聚类对把图像中每一个像素点当作一个数据点进行聚类，通常一幅图像的像素点的数量就是十万数量级，所以程序十分耗时且基本上无法在我们自己的电脑上跑完。于是我们对图像的数据进行压缩，保留基本信息的同时削减数据量，从而加快程序运行的速度。但是我们最终并没有达到论文中的实验效果，可能也跟这里有很大关系。当然，由于论文叙述的大部分是原理，实际操作过程中的一些论文作者自己的细节操作并没有公布出来，可能也是造成我们无法实现论文中的结果的一个很大的因素。

在我们进行论文调研的时候，发现花卉分类还有一个常见的情形：就是在自然界中我们还能发现我们从未发现的新的花卉品种，我们如何通过机器学习方法自动识别出这是一种新的花卉，或者这种并未出现在训练的数据集中，但应用

场景中却经常出现，并且我们掌握这种花的一些性状或者属性描述。这里所提到的就是 Zero-shot Learning。零样本学习或者少样本学习是当前机器学习领域中比较热门的分支，其能解决的场合十分普遍。但是在本次课程设计中，由于缺乏带属性标注的数据集，我们并没有相关方面的实质性的工作。

在花卉识别之前，首先要对花卉图像进行分割处理，本次课程设计中我们采用了几种常见的花卉分割方法。但是在面对一幅图像中有多个花朵，即包含多个目标时，如何将其逐个分割出来就成为新的挑战。通过中期验收与老师的沟通，我们得知可以通过一些特殊的神经网络来完成上述问题，如 mask-RCNN。但是由于缺乏已标注的分割数据集且找不到已训练好的网络模型参数^①，我们同样未能完成上述工作，希望可以在以后的工作学习中可以有相关工作的尝试。

通过本次课程设计的实践，我们了解到一个模式识别的问题应该如何被分析和解决，对于科研工作有了一定的初步了解，从查阅文献到复现代码，通过不断的尝试和老师师兄的指导，一步一步逐渐地完成课程设计的任务，让我们收获颇丰。

^①tensorflow 中包含这种模型，但由于网络原因没有完成下载。

参考文献

- [1] Zawbaa H M, Abbass M, Basha S H, et al. An automatic flower classification approach using machine learning algorithms[C] //2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI). 2014: 895–901.
- [2] Liu Y, Tang F, Zhou D, et al. Flower classification via convolutional neural network[C] //2016 IEEE International Conference on Functional-Structural Plant Growth Modeling, Simulation, Visualization and Applications (FSPMA). 2016: 110–116.
- [3] Nilsback M , Zisserman A. A Visual Vocabulary for Flower Classification[C] //2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’06): Vol 2. 2006: 1447–1454.
- [4] KANEKOT, SAITOHT. Automatic Recognition of Wild Flowers[C/OL] //Pattern Recognition, International Conference on : Vol 2. Los Alamitos, CA, USA : IEEE Computer Society, 2000: 2507.
<https://doi.ieee.org/10.1109/ICPR.2000.906123>.
- [5] Nilsback M, Zisserman A. Automated Flower Classification over a Large Number of Classes[C] //2008 Sixth Indian Conference on Computer Vision, Graphics Image Processing. 2008: 722–729.
- [6] Kanan C, Cottrell G. Robust classification of objects, faces, and flowers using natural image statistics[C] //2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. 2010: 2472–2479.
- [7] Yoo D, Park S, Lee J, et al. Multi-scale pyramid pooling for deep convolutional representation[C]//2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW). 2015: 71–80.
- [8] Lodh A, Parekh R. Flower recognition system based on color and GIST features[C] //2017 Devices for Integrated Circuit (DevIC). 2017: 790–794.

- [9] Najjar A, Zagrouba E. Flower image segmentation based on color analysis and a supervised evaluation[C] //2012 International Conference on Communications and Information Technology (ICCIT). 2012: 397–401.
- [10] Liu W, Rao Y, Fan B, et al. Flower classification using fusion descriptor and SVM[C] //2017 International Smart Cities Conference (ISC2). 2017: 1–4.
- [11] Wang J, Yang J, Yu K, et al. Locality-constrained Linear Coding for image classification[C] //2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. 2010: 3360–3367.
- [12] CHENG K, TAN X. Sparse representations based attribute learning for flower classification[J/OL]. Neurocomputing, 2014, 145: 416 – 426.
<http://www.sciencedirect.com/science/article/pii/S0925231214005827>.
- [13] Costa A F, Humpire-Mamani G, Traina A J M. An Efficient Algorithm for Fractal Analysis of Textures[C] //2012 25th SIBGRAPI Conference on Graphics, Patterns and Images. 2012: 39–46.
- [14] Wu J, Rehg J M. Beyond the Euclidean distance: Creating effective visual codebooks using the Histogram Intersection Kernel[C] //2009 IEEE 12th International Conference on Computer Vision. 2009: 630–637.
- [15] Guang-Bin Huang, Qin-Yu Zhu, Chee-Kheong Siew. Extreme learning machine: a new learning scheme of feedforward neural networks[C] //2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No.04CH37541): Vol 2. 2004: 985–990 vol.2.
- [16] Huang G, Zhou H, Ding X, et al. Extreme Learning Machine for Regression and Multiclass Classification[J]. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 2012, 42(2): 513–529.
- [17] SIMONYAN K, ZISSERMAN A. Very Deep Convolutional Networks for Large-Scale Image Recognition[J/OL]. CoRR, 2014, abs/1409.1556.
<http://arxiv.org/abs/1409.1556>.

- [18] Shelhamer E, Long J, Darrell T. Fully Convolutional Networks for Semantic Segmentation[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2017, 39(4): 640–651.
- [19] HUANG G, LI Y, PLEISS G, et al. Snapshot Ensembles: Train 1, get M for free[J/OL]. CoRR, 2017, abs/1704.00109.
<http://arxiv.org/abs/1704.00109>.
- [20] Wu Y, Qin X, Pan Y, et al. Convolution Neural Network based Transfer Learning for Classification of Flowers[C] //2018 IEEE 3rd International Conference on Signal and Image Processing (ICSIP). 2018: 562–566.
- [21] Hiary H, Saadeh H, Saadeh M, et al. Flower classification using deep convolutional neural networks[J]. IET Computer Vision, 2018, 12(6): 855–862.

附录 A 补充的图像处理知识

A.1 SIFT 特征提取算子

SIFT 特征是 Scale Invariant Feature Transform 的缩写，由加拿大教授 David G.Lowe 提出。SIFT 特征对旋转、尺度缩放、亮度变化等保持不变性，是一种非常稳定的局部特征。

多分辨率图像金字塔：在早期图像的多尺度通常使用图像金字塔表示形式。图像金字塔是同一图像在不同的分辨率下得到的一组结果，其生成过程一般包括两个步骤：

- 对原始图像进行平滑；
- 对处理后的图像进行下采样，下采样后得到一系列尺寸不断缩小的图像。

高斯尺度空间：我们还可以通过图像的模糊程度来模拟人在距离物体由远到近时物体在视网膜上成像过程，距离物体越近其尺寸越大图像也越模糊，这就是高斯尺度空间。使用不同的参数模糊图像（分辨率不变），时尺度空间的另一种表现形式。我们知道图像和高斯函数进行卷积运算能够对图像进行模糊，使用不同的高斯核可得到不同模糊程度的图像。一幅图像其高斯尺度空间可由其和不同的高斯卷积得到：

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \quad (\text{A.1})$$

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{\frac{x^2+y^2}{2\sigma^2}} \quad (\text{A.2})$$

其中， $G(x, y, \sigma)$ 为高斯核函数， σ 为尺度空间因子，它是高斯正态分布的标准差，反映了图像被模糊的程度，其值越大图像越模糊，对应的尺度也就越大。 $L(x, y, \sigma)$ 代表着图像的高斯尺度空间。构建尺度空间的目的是为了检测不同的尺度下都存在的特征点，而检测特征点较好的算子时高斯拉普拉斯算子，

$$\Delta^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \quad (\text{A.3})$$

使用高斯拉普拉斯算子虽然能够较好的检测到图像中的特征点，但是其运算量过大，通常可以使用差分高斯（Difference of Gaussia, DOG）来近似计算。设 k 为相邻两个高斯尺度空间的比例因子，则 DoG 的定义为

$$D(x, y, \sigma) = [G(x, y, k\sigma) - G(x, y, \sigma)] * I(x, y) = L(x, y, k\sigma) - L(x, y, \sigma) \quad (\text{A.4})$$

其中, $L(x,y,\sigma)$ 是图像的高斯尺度空间。从上式可以知道, 将相邻的两个高斯空间的图像相减就得到了 DoG 的响应图像。为了得到 DoG 图像, 先要构建高斯尺度空间, 而高斯的尺度空间可以在图像金字塔降采样的基础上加上高斯滤波得到, 也就是对图像金字塔的每层图像使用不同的参数 σ 进行高斯模糊, 使每层金字塔有多张高斯模糊过的图像。降采样时, 金字塔上边一组图像的第一张是由其下面一组图像倒数第三张采样得到的。

易知, 高斯金字塔有多组, 每组又有多层。一组中的多个层之间的尺度是不一样的, 相邻两层之间的尺度相差一个比例因子 k 。高斯金字塔构建完成后, 将相邻的高斯金字塔相减就得到了 DoG 金字塔。高斯金字塔的组数一般是

$$o = \lceil \log_2 \min(m,n) \rceil - a \quad (\text{A.5})$$

其中 o 表示金字塔的层数, m, n 分别是图像的行和列。减去的系数 a 可以在 $0 \sim \log_2 \min(m,n)$ 之间的任意值。高斯模糊参数 σ 可由下式得到

$$\sigma(o,s) = \sigma_0 2^{\frac{o+s}{S}} \quad (\text{A.6})$$

其中 o 为所在的组, s 为所在的层, σ_0 为初始的尺度, S 为每组的层数。

DoG 空间极值检测: 为了寻找尺度空间的极值点, 每个像素点要和其图像域 (同一尺度空间) 和尺度域 (相邻的尺度空间) 的所有相邻点进行比较, 当其大于 (或者小于) 所有相邻点时, 该点就是极值点。

删除不好的极值点: 通过比较检测得到的 DoG 的局部极值点是在离散空间搜索得到的, 由于离散空间是对连续空间采样得到的结果, 因此在离散空间找到的极值点不一定是真正意义上的极值点, 因此要设法将不满足条件的点剔除掉。可以通过尺度空间 DoG 函数进行曲线拟合寻找极值点, 这一步的本质是去掉 DoG 局部曲率非常不对称的点。要剔除掉的不符合要求的点主要有两种: 低对比度的特征点和不稳定的边缘响应点。

剔除低对比度的特征点: 对于候选特征点 x , 其偏移量定义为 Δx , 其对比度为 $D(x)$ 的绝对值 $|D(x)|$, 对 $D(x)$ 应用泰勒展开式

$$D(x) = D + \frac{\partial D^T}{\partial x} \Delta x + \frac{1}{2} \Delta x^T \frac{\partial^2 D}{\partial x^2} \Delta x \quad (\text{A.7})$$

$$\Delta x = -\frac{\partial^2 D^{-1}}{\partial x^2} \frac{\partial D(x)}{\partial x} \quad (\text{A.8})$$

由于 x 是 $D(x)$ 的极值点, 所以对上式求导并令其为 0, 得到

$$D(\hat{x}) = D + \frac{1}{2} \frac{\partial D^T}{\partial x} \hat{x} \quad (\text{A.9})$$

然后再把求得的 Δx 代入到 $D(x)$ 的泰勒展开式中。设对比度的阈值为 T , 若 $|D(\hat{x})| \geq T$, 则该特征点保留, 否则剔除掉。

剔除不稳定的边缘响应点: 在边缘梯度的方向上主曲率值比较大, 而沿着边缘方向则主曲率值比较小。候选特征点的 DoG 函数 $D(x)$ 的主曲率与 2×2 的 Hessian 矩阵 H 的特征值成正比,

$$H = \begin{bmatrix} D_{xx} & D_{yx} \\ D_{xy} & D_{yy} \end{bmatrix} \quad (\text{A.10})$$

其中 $D_{xx}, D_{xy} = D_{yx}, D_{yy}$ 是候选点邻域对应位置的差分求得的。设 $\alpha = \lambda_{\max}$ 为 H 的最大特征值, $\beta = \lambda_{\min}$ 为其最小特征值, 则

$$\text{Tr}(H) = D_{xx} + D_{yy} = \alpha + \beta \quad (\text{A.11})$$

$$\text{Det}(H) = D_{xx}D_{yy} - D_{xy}^2 = \alpha\beta \quad (\text{A.12})$$

设 $\gamma = \alpha/\beta$ 表示最大特征值和最小特征值的比值, 则

$$\frac{\text{Tr}(H)^2}{\text{Det}(H)} = \frac{(\alpha + \beta)^2}{\alpha\beta} = \frac{(\gamma\beta + \beta)^2}{\gamma\beta^2} = \frac{(\gamma + 1)^2}{\gamma} \quad (\text{A.13})$$

因此, 为了检测主曲率是否在某个阈值 T_γ 下, 只需检测

$$\frac{\text{Tr}(H)^2}{\text{Det}(H)} > \frac{(T_\gamma + 1)^2}{T_\gamma} \quad (\text{A.14})$$

求取特征点的主方向: 经过上面的步骤已经找到了在不同尺度下都存在的特征点, 为了实现图像旋转不变性, 需要给特征点的方向进行赋值。利用特征点邻域像素的梯度分布特性来确定其方向参数, 再利用图像的梯度直方图求取关键局部结构的稳定方向。

找到了特征点, 也就可以得到该特征点的尺度 σ , 也就可以得到特征点所在的尺度图像

$$L(x, y) = G(x, y, \sigma) * I(x, y) \quad (\text{A.15})$$

计算以特征点为中心, 以 $3 \times 1.5\sigma$ 为半径的区域图像的辐角和幅值, 每个点的 $L(x, y)$ 的梯度的模 $m(x, y)$ 以及方向 $\theta(x, y)$ 可通过下式求得

$$m(x, y) = \sqrt{[L(x+1, y) - L(x-1, y)]^2 + [L(x, y+1) - L(x, y-1)]^2} \quad (\text{A.16})$$

$$\theta(x, y) = \arctan \frac{L(x, y+1) - L(x, y-1)}{L(x+1, y) - L(x-1, y)} \quad (\text{A.17})$$

得到特征点的主方向后, 对于每个特征点可以得到三个信息 (x, y, θ) , 即位置、尺度和方向。由此可以确定一个 SIFT 特征区域, 一个 SIFT 特征区域由三个值表示, 中心表示特征点位置, 半径表示关键点的尺度, 箭头表示主方向。具有

多个方向的关键点可以被复制成多份，然后将方向值分别赋给复制后的特征点，一个特征点就产生了多个坐标、尺度相等，但是方向不同的特征点。

生成特征描述：校正旋转主方向，确保旋转不变性；生成描述子，最终形成一个 128 维的特征向量；归一化处理，将特征向量长度进行归一化处理，进一步去除光照的影响。

A.2 LBP 算子

LBP 是 Local Binary Pattern 的缩写，译为局部二值模式，是一种用来描述图像局部特征的算子。LBP 特征具有灰度不变性和旋转不变性等显著优点。它是由 T.Ojala, M.Pietikainen 和 D.Harwood 在 1994 年提出的。由于 LBP 特征计算简单、效果较好，因此 LBP 特征在计算机视觉的许多领域都得到了广泛的应用，LBP 特征比较出名的是应用在人脸识别和目标检测中，在计算机视觉来源库 OpenCV 中有使用 LBP 特征进行人脸识别的接口，也有用 LBP 特征训练目标检测分类器的方法。

原始 LBP 特征定义在像素 3×3 的邻域内，以邻域中心像素为阈值，相邻的 8 个像素的灰度值与邻域中心的像素值进行比较，若周围像素大于中心像素值，则该像素点的位置被标记为 1，否则为 0。上述过程可表示为

$$LBP(x_c, y_c) = \sum_{p=0}^{P-1} 2^p s(i_p - i_c) \quad (\text{A.18})$$

其中 i_c 表示中心像素 (x_c, y_c) 的像素值， i_p 表示其近邻像素的像素值，函数 s 为

$$s(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{else} \end{cases} \quad (\text{A.19})$$

后来将 LBP 从定义在固定邻域上转变为定义在固定半径范围内的小区域，改进后的 LBP 算子允许在半径为 R 的圆形邻域内有任意多个像素点，经过采样得到含有 P 个采样点的 LBP 算子。LBP 的主要步骤如下：

- 将检测窗口划分为固定大小的小区域 (cell)；
- 对于每个 cell 中的每一个像素，将其指定邻域的采样点的像素的灰度值与其进行比较，若采样点像素值大于中心像素值，则该像素点的位置标记为 1，否则为 0；
- 然后计算每个 cell 的直方图，对该直方图进行归一化处理；

- 最后将得到的每一 cell 的统计直方图连接成一个特征向量，也就是整幅图的 LBP 纹理特征向量。

A.3 Canny 算子

Canny 提出一种新的边缘检测方法，它对受白噪声影响的阶跃型边缘是最优的。Canny 检测子的最优性与三个标准有关：第一，检测标准，不失去重要的边缘，不应有虚假的边缘；第二，定位标准，实际边缘与检测到的边缘位置之间的偏差最小；第三，单位应标准，将多个响应降低为单个边缘响应。这一点被第一个标准部分地覆盖了。这第三个标准解决受噪声影响的边缘问题，起抑制非平滑边缘检测算子的作用。

Canny 提出了特征综合方法。首先标记处所有由最小尺度算子得到的突出边缘。而整个 Canny 边缘检测器算法分成如下四步：

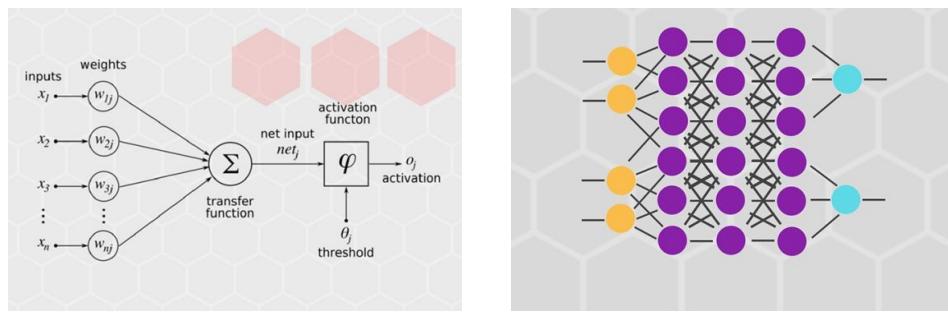
- 噪声去除。因为这个检测器用到了微分算子，所以对于局部的不连续是敏感的，某区域的噪声点很容易造成边缘的模糊；
- 计算图像梯度，得到可能边缘；
- 非最大梯度值抑制。通常灰度变化的地方都比较集中，将局部范围内的梯度方向上，灰度变化最大的保留下来，其它的不留，这样可以剔除掉一大部分的点。将由多个像素宽的边缘变成一个单像素宽的边缘；
- 双阈值筛选。通过给非极大值抑制后，仍然有很多的可能边缘点，进一步的设置一个双阈值，即低阈值和高阈值。灰度变化大于高阈值的，设为强边缘像素；低于低阈值的，剔除。在低阈值和高阈值之间的设置为弱边缘。进一步判断，如果其领域内有强边缘像素，保留，如果没有，剔除。

附录 B 神经网络简介

B.1 神经网络概述

人工神经网络中最小也最重要的单元叫神经元。与生物神经系统类似，这些神经元也互相连接并具有强大的处理能力。一般而言，人工神经网络试图复现真实大脑的行为和过程。

每个神经元都有输入连接和输出连接。这些连接模拟了大脑中突触的行为。与大脑中突触传递信号的方式相同——信号从一个神经元传递到另一个神经元，这些连接也在人造神经元之间传递信息。每一个连接都有权重，这意味着发送到每个连接的值要乘以这个因子。这种模式是从大脑突触得到的启发，权重实际上模拟了生物神经元之间传递的神经递质的数量。所以，如果某个连接重要，那么它将具有比那些不重要的连接更大的权重值。



(a) 单个神经元的模型

(b) 人工神经网络的结构模型

图 B-1 神经网络

由于可能有许多值进入一个神经元，每个神经元便有一个所谓的输入函数。通常，连接的输入值都会被加权求和。然后该值被传递给激活函数，激活函数的作用是计算出是否将一些信号发送到该神经元的输出。

B.2 反向传播算法

根据梯度下降法，我们在每一次迭代过程中都会按照下式进行参数更新，

$$W_{ij}^{(l)} = W^{(l)}_{ij} - \eta \frac{\partial}{\partial W_{ij}^{(l)}} J(W, b) \quad (B.1)$$

$$b_i^{(l)} = b_i^{(l)} - \eta \frac{\partial}{\partial b_i^{(l)}} J(W, b) \quad (B.2)$$

其中 η 是初始学习率。

反向传播算法的思路如下：给定一个样例 (x, y) ，我们首先进行“前向传导”

运算, 计算出网络中所有的激活值, 包括 $h_{W,b}(x)$ 的输出值。之后, 针对第 l 层的每一个节点 i , 我们计算出其残差 $\delta_i^{(l)}$, 该残差表明了该节点对最终输出值的残差产生了多少影响。对于最终的输出节点, 我们可以直接计算出网络产生的激活值与实际值之间的差距, 我们将这个差距定义为 $\delta_i^{(n_l)}$ 为输出层的残差。下面给出关于输出层和隐藏层的残差计算公式:

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2} \|y - h_{W,b}(x)\|^2 = -(y_i - a_i^{(n_l)}) f'(z_i^{(n_l)}) \quad (\text{B.3})$$

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)} \right) f'(z_i^{(l)}) \quad (\text{B.4})$$

那么, 反向传播算法可表示为以下几个步骤:

- 进行前馈传导计算, 利用前向传导公式, 得到 L_2, L_3, \dots, L_{n_l} 的输出值;
- 对输出层 (第 n_l 层), 计算

$$\delta^{(n_l)} = -(y - a^{(n_l)}) f'(z^{(n_l)}) \quad (\text{B.5})$$

- 对于 $l = n_l - 1, n_l - 2, \dots, 2$ 的各层, 计算

$$\delta^{(l)} = ((W^{(l)})^T \delta^{(l+1)}) f'(z^{(l)}) \quad (\text{B.6})$$

- 计算最终需要的偏导数值:

$$\nabla_{W^{(l)}} J(W, b; x, y) = \delta^{(l+1)} (a^{(l)})^T \quad (\text{B.7})$$

$$\nabla_{b^{(l)}} J(W, b; x, y) = \delta^{(l+1)} \quad (\text{B.8})$$

- 计算:

$$\Delta W^{(l)} = \Delta W^{(l)} + \nabla_{W^{(l)}} J(W, b; x, y) \quad (\text{B.9})$$

$$\Delta b^{(l)} = \Delta b^{(l)} + \nabla_{b^{(l)}} J(W, b; x, y) \quad (\text{B.10})$$

- 更新权重参数:

$$W^{(l)} = W^{(l)} - \eta \left[\left(\frac{1}{m} \Delta W^{(l)} \right) + \lambda W^{(l)} \right] \quad (\text{B.11})$$

$$b^{(l)} = b^{(l)} - \eta \left[\frac{1}{m} \Delta b^{(l)} \right] \quad (\text{B.12})$$

附录 C 代码

C.1 文件夹：test

main.py

```

1      """
2      方案一：基于视觉词汇的花卉识别方法
3      """
4
5      import glob
6      import cv2
7      import numpy as np
8      import pandas as pd
9      import matplotlib.pyplot as plt
10     import math
11     import matplotlib.mlab as mlab
12     from sklearn.cluster import KMeans,MiniBatchKMeans
13     import kNN
14     from mpl_toolkits.mplot3d import Axes3D
15     from matplotlib.ticker import NullFormatter
16     from sklearn import manifold,datasets
17     import utils
18     import kNN
19     #import afkmc2.afkmc2 as afk
20     #import MR8
21     from skimage.transform import rotate
22     from skimage.feature import local_binary_pattern
23     from skimage import data,io,data_dir,filters,feature
24     from skimage.color import label2rgb
25     import skimage
26     from PIL import Image
27     import dataLoad
28
29     CHOSEN_TRAIN=40#定义每一类选择的训练样本个数
30     RADIUS=4#定义LBP算子的区域半径
31     N_POINTS=8*RADIUS#定义LBP算子的区域采样点个数
32     ALPHA=6.0#颜色特征的权重
33     BETA=4.5#形状特征的权重
34     GAMMA=3.1#纹理特征的权重
35
36     randomIndex,resIndex=dataLoad.randomSample(80,CHOSEN_TRAIN)#随机选取训练样本和测试样本
37     filenames=[]
38     filenames.append(glob.glob(r'F:/seven/prDesign/fdata/buttercup/*.jpg'))
39     filenames.append(glob.glob(r'F:/seven/prDesign/fdata/daisy/*.jpg'))
40     filenames.append(glob.glob(r'F:/seven/prDesign/fdata/iris/*.jpg'))
41     filenames.append(glob.glob(r'F:/seven/prDesign/fdata/lilyvalley/*.jpg'))
42     filenames.append(glob.glob(r'F:/seven/prDesign/fdata/sunflower/*.jpg'))
43     filenames.append(glob.glob(r'F:/seven/prDesign/fdata/windflower/*.jpg'))
44     #提取原始特征
45     dataX,numX=dataLoad.load(filenames,randomIndex,'color',CHOSEN_TRAIN,N_POINTS,RADIUS)

```

```

45     print('color..completed')
46     dataY,numY=dataLoad.load(filenames,randomIndex,'shape',CHosen_TRAIN,N_POINTS,RADIUS)
47     print('shape..completed')
48     dataZ,numZ=dataLoad.load(filenames,randomIndex,'texture',CHosen_TRAIN,N_POINTS,RADIUS)
49     print('texture..completed')
50     #生成类别标签
51     y=None
52     for i in range(6):
53         tempY=(i+1)*np.ones(CHosen_TRAIN,dtype=int)
54         if y is None:
55             y=tempY
56         else:
57             y=np.concatenate((y,tempY),axis=0)
58
59     #聚类生成视觉词汇
60     kmeansX=MiniBatchKMeans(n_clusters=200)
61     kmeansX.fit(dataX)
62     print('color..k-means..completed')
63
64     kmeansY=MiniBatchKMeans(n_clusters=200)
65     kmeansY.fit(dataY)
66     print('shape..k-means..completed')
67
68     kmeansZ=MiniBatchKMeans(n_clusters=200)
69     kmeansZ.fit(dataZ)
70     print('texture..k-means..completed')
71     print('k-means..complete!')
72     centersX,labelsX=kmeansX.cluster_centers_,kmeansX.labels_
73     centersY,labelsY=kmeansY.cluster_centers_,kmeansY.labels_
74     centersZ,labelsZ=kmeansZ.cluster_centers_,kmeansZ.labels_
75
76     #生成最终的特征表达，即用最近邻法统计直方图
77     histDataX=np.zeros((6*CHosen_TRAIN,200),dtype=int)
78     histDataY=np.zeros((6*CHosen_TRAIN,200),dtype=int)
79     histDataZ=np.zeros((6*CHosen_TRAIN,200),dtype=int)
80     k=0
81     sumXZ=0
82     sumY=0
83     for filename in filenames:
84         for i in range(CHosen_TRAIN):
85             tempImg=cv2.cvtColor(cv2.imread(filename[i],cv2.IMREAD_COLOR),cv2.COLOR_BGR2HSV) #
86             m,n,_=tempImg.shape
87             m,n,_=tempImg.shape
88             for j in range(m+n):
89                 histDataX[k*CHosen_TRAIN+i,labelsX[j+sumXZ]]=histDataX[k*CHosen_TRAIN+i,labelsX[j+sumXZ]]+1
90                 histDataZ[k*CHosen_TRAIN+i,labelsZ[j+sumXZ]]=histDataZ[k*CHosen_TRAIN+i,labelsZ[j+sumXZ]]+1
91             for j in range(numY[k*CHosen_TRAIN+i]):
92                 histDataY[k * CHosen_TRAIN + i, labelsY[j + sumY]] = histDataY[k * CHosen_TRAIN + i,
93                                         labelsY[j + sumY]] + 1

```

```

92     sumXZ=sumXZ+m+n
93     sumY=sumY+numY[k*CHOSEN_TRAIN+i]
94     k=k+1
95
96
97     """
98     #使用t-SNE数据可视化
99     tsne=manifold.TSNE(n_components=2,init='pca',random_state=0)
100    Y=tsne.fit_transform(histData)
101    print('tSNE completed!')
102    fig=utils.plot_embedding(Y,y,'t-SNE embedding of the digits')
103    plt.show()
104    """
105    #测试，并计算正确率
106    """
107    sum=0
108    t=0
109    for filename in filenames:
110        t=t+1
111        for i in range(CHOSEN_TRAIN,80):
112            resultTT=np.zeros(7,dtype=float)
113            testFile=filename[resIndex[i-CHOSEN_TRAIN]]
114            histTestX=kNN.featureCOV(centersX,testFile)
115            histTestY=kNN.featureSIFT(centersY,testFile)
116            histTestZ=kNN.featureLBP(centersZ,testFile,N_POINTS,RADIUS)
117            print('featuren construction completed!')
118            resultC,probC=kNN.xkNN(histDataX,y,histTestX)
119            resultS,probS=kNN.xkNN(histDataY,y,histTestY)
120            resultT,probT=kNN.xkNN(histDataZ,y,histTestZ)
121            resultTT[resultC]=resultTT[resultC]+ALPHA*probC
122            resultTT[resultS]=resultTT[resultS]+BETA*probS
123            resultTT[resultT]=resultTT[resultT]+GAMMA*probT
124            result=np.where(resultTT==np.max(resultTT))
125            result=result[0]
126            print(resultTT,result,t)
127            if result==t:
128                sum=sum+1
129            print(sum*100/240)
130            """
131    #测试函数
132    testFilename='F:/seven/prDesign/test6.jpg'
133    resultTT=np.zeros(7,dtype=float)
134    histTestX=kNN.featureCOV(centersX,testFilename)
135    histTestY=kNN.featureSIFT(centersY,testFilename)
136    histTestZ=kNN.featureLBP(centersZ,testFilename,N_POINTS,RADIUS)
137    print('featuren_construction_completed!')
138    resultC,probC=kNN.xkNN(histDataX,y,histTestX)
139    resultS,probS=kNN.xkNN(histDataY,y,histTestY)
140    resultT,probT=kNN.xkNN(histDataZ,y,histTestZ)
141    resultTT[resultC]=resultTT[resultC]+ALPHA*probC
142    resultTT[resultS]=resultTT[resultS]+BETA*probS

```

```

143 resultTT[resultT]=resultTT[resultT]+GAMMA*probT
144 result=np.where(resultTT==np.max(resultTT))
145 result=result[0]
146 print(resultTT,result)

```

dataLoad.py

```

1      """
2      加载数据模块
3      """
4
5      import cv2
6      import numpy as np
7      from skimage.feature import local_binary_pattern
8      import utils
9
10     #随机采样样本作为训练数据
11     def randomSample(imgNum,chosen_train):
12         index=np.arange(imgNum)
13         randomIndex=np.random.choice(index,size=chosen_train)
14         setIndex=set(index)
15         setRan=set(randomIndex)
16         setresIndex=setIndex-setRan
17         resIndex=np.array(list(setresIndex))
18         return randomIndex,resIndex
19
20     #加载数据并作特征提取
21     def load(filenames,randomIndex,mode,chosen_train,n_points=0,radius=0):
22         data=None
23         num=np.zeros(6*chosen_train,dtype=int)
24         k=0
25         for filename in filenames:
26             for i in range(chosen_train):
27                 tempImg=cv2.imread(filename[randomIndex[i]],cv2.IMREAD_COLOR)
28                 tempImg=utils.segmentation(tempImg)
29                 if mode=='shape':
30                     tempImg=cv2.cvtColor(tempImg,cv2.COLOR_BGR2GRAY)
31                     sift=cv2.xfeatures2d.SIFT_create()
32                     kps,des=sift.detectAndCompute(tempImg,None)
33                     num[k*chosen_train+i]=len(kps)
34                     tData=np.zeros((len(kps),5))
35                     for j in range(len(kps)):
36                         tData[j,:]=np.array([kps[j].angle,kps[j].pt[0],kps[j].pt[1],kps[j].response,kps[j].size])[:]
37             else:
38                 if mode=='color':
39                     tempHSV=cv2.cvtColor(tempImg,cv2.COLOR_BGR2HSV)
40                     m,n,c=tempHSV.shape
41                     tempData = np.array(tempHSV)
42                 elif mode=='texture':
43                     tempImg=cv2.cvtColor(tempImg,cv2.COLOR_BGR2GRAY)
44                     tempLBP=local_binary_pattern(tempImg,n_points,radius)
45                     m,n=tempLBP.shape
46                     c=1

```

```

46         tempData=np.array(tempLBP).reshape((m,n,c))
47         tempData=tempData.swapaxes(1,2)
48         rowData=np.zeros((m,c*c+c))
49         colData=np.zeros((n,c*c+c))
50         for j in range(m):
51             rowData[j,:c*c]=np.cov(tempData[j,:,:]).reshape(-1)
52             rowData[j,c*c:]=np.mean(tempData[j,:,:],axis=1)
53             for j in range(n):
54                 colData[j,:c*c]=np.cov(np.mat(tempData[:, :,j]).T).reshape(-1)
55                 colData[j,c*c:]=np.mean(tempData[:, :,j],axis=0)
56             tData=np.concatenate((rowData,colData),axis=0)
57             print(i)
58             if data is None:
59                 data=tData
60             else:
61                 data=np.concatenate((data,tData),axis=0)
62             k=k+1
63     return data,num

```

kNN.py

```

1      """
2  KNN分类模块
3      """
4  import numpy as np
5  import utils
6  import cv2
7  import MR8
8  from skimage.feature import local_binary_pattern
9
10 #单一特征的KNN分类函数
11 def xkNN(data,y,testX,k=5):
12     m,n=data.shape
13     disMatrix=np.zeros(m,dtype=float)
14     for i in range(m):
15         disMatrix[i]=utils.chi2_distance(data[i],testX)
16     disIndex=np.argsort(disMatrix, axis=0)
17     disIndexR=disIndex[:k]
18     result=y[disIndexR]
19     bresult=np.bincount(result)
20     print(bresult)
21     resultLabel=np.argmax(bresult)
22     sum=np.sum(bresult)
23     prob=float(bresult[resultLabel])/float(sum)
24     return resultLabel,prob
25
26 #联合特征的KNN分类函数
27 def bxNN(dataX,dataY,dataZ,y,testX,testY,testZ,alpha,beta,gamma,k=5):
28     mx,nx=dataX.shape
29     my,ny=dataY.shape
30     mz,nz=dataZ.shape
31     disXMatrix=np.zeros(mx,dtype=float)

```

```

32     disYMatrix=np.zeros(my,dtype=float)
33     disZMatrix=np.zeros(mz,dtype=float)
34     for i in range(mx):
35         disXMatrix[i]=utils.chi2_distance(dataX[i],testX)
36     for i in range(my):
37         disYMatrix[i]=utils.chi2_distance(dataY[i],testY)
38     for i in range(mz):
39         disZMatrix[i]=utils.chi2_distance(dataZ[i],testZ)
40     disMatrix=alpha*disXMatrix+beta*disYMatrix+gamma*disZMatrix
41     disIndex = np.argsort(disMatrix, axis=0)
42     disIndexR = disIndex[:k]
43     result = y[disIndexR]
44     print(np.bincount(result))
45     resultLabel = np.argmax(np.bincount(result))
46     return resultLabel
47
48 #最近邻分类函数
49 def ekNN(data,y,testX,k=1):
50     m,n=data.shape
51     disMatrix=np.zeros(m,dtype=float)
52     for i in range(m):
53         disMatrix[i]=np.linalg.norm(data[i,:]-testX)
54     disIndex=np.argsort(disMatrix, axis=0)
55     disIndexR=disIndex[:k]
56     result=y[disIndexR]
57     return result
58
59 #提取颜色特征
60 def featureCOV(center,filename):
61     img=cv2.imread(filename,cv2.IMREAD_COLOR)
62     img=utils.segmentation(img)
63     hsvImg=cv2.cvtColor(img,cv2.COLOR_BGR2HSV)
64     cluster=len(center)
65     m,n,c=hsvImg.shape
66     data=np.array(hsvImg)
67     data=data.swapaxes(1,2)
68     rowData = np.zeros((m, c * c + c))
69     colData = np.zeros((n, c * c + c))
70     for j in range(m):
71         rowData[j, :c * c] = np.cov(data[j, :, :]).reshape(-1)
72         rowData[j, c * c:] = np.mean(data[j, :, :], axis=1)
73         ttData = np.sort(data[j, :, :], axis=1)
74     for j in range(n):
75         colData[j, :c * c] = np.cov(np.mat(data[:, :, j]).T).reshape(-1)
76         colData[j, c * c:] = np.mean(data[:, :, j], axis=0)
77     tData = np.concatenate((rowData, colData), axis=0)
78     y=np.arange(cluster)+1
79     histFeature=np.zeros(cluster)
80     for i in range(m+n):
81         tempLabel=ekNN(center,y,tData[i])
82         histFeature[tempLabel-1]=histFeature[tempLabel-1]+1

```

```

83     return histFeature
84
85 #提取形状特征
86 def featureSIFT(center,filename):
87     tempImg = cv2.imread(filename, cv2.IMREAD_COLOR)
88     tempImg=utils.segmentation(tempImg)
89     tempImg = cv2.cvtColor(tempImg, cv2.COLOR_BGR2GRAY)
90     data=None
91     cluster=len(center)
92     sift = cv2.xfeatures2d.SIFT_create()
93     kps, des = sift.detectAndCompute(tempImg, None)
94     # print(kps)
95     tData = np.zeros((len(kps), 5))
96     for j in range(len(kps)):
97         tData[j, :] = np.array([kps[j].angle, kps[j].pt[0], kps[j].pt[1], kps[j].response, kps[j].size])[:]
98     if data is None:
99         data = tData
100    else:
101        data = np.concatenate((data, tData), axis=0)
102
103    y = np.arange(cluster) + 1
104    histFeature = np.zeros(cluster)
105    for i in range(len(kps)):
106        tempLabel = ekNN(center, y, tData[i])
107        histFeature[tempLabel - 1] = histFeature[tempLabel - 1] + 1
108    return histFeature
109
110 #提取纹理特征
111 def featureLBP(center,filename,n_points,radius):
112     tempImg = cv2.imread(filename, cv2.IMREAD_COLOR)
113     tempImg=utils.segmentation(tempImg)
114     tempImg = cv2.cvtColor(tempImg, cv2.COLOR_BGR2GRAY)
115     tempLBP=local_binary_pattern(tempImg,n_points,radius)
116     m, n = tempLBP.shape
117     c=1
118     tempData = np.array(tempLBP).reshape((m,n,c))
119     tempData = tempData.swapaxes(1, 2)
120     rowData = np.zeros((m, c * c + c))
121     colData = np.zeros((n, c * c + c))
122     data = None
123     cluster = len(center)
124     for j in range(m):
125         rowData[j, :c * c] = np.cov(tempData[j, :, :]).reshape(-1)
126         rowData[j, c * c:] = np.mean(tempData[j, :, :], axis=1)
127     for j in range(n):
128         colData[j, :c * c] = np.cov(np.mat(tempData[:, :, j]).T).reshape(-1)
129         colData[j, c * c:] = np.mean(tempData[:, :, j], axis=0)
130     tData = np.concatenate((rowData, colData), axis=0)
131     if data is None:
132         data = tData
133     else:

```

```
134     data = np.concatenate((data, tData), axis=0)
135
136     y = np.arange(cluster) + 1
137     histFeature = np.zeros(cluster)
138
139     for i in range(m + n):
140         tempLabel = ekNN(center, y, tData[i])
141         histFeature[tempLabel - 1] = histFeature[tempLabel - 1] + 1
142
143     return histFeature
```

utils.py

```
1      """
2      定义辅助函数
3      """
4
5      import numpy as np
6      import matplotlib.pyplot as plt
7      import cv2
8      import chart_studio.plotly as py
9      import plotly.graph_objs as go
10     import plotly
11
12    #t-SNE可视化绘图函数
13
14    def plot_embedding(data,label,title):
15        x_min,x_max=np.min(data,0),np.max(data,0)
16        data=(data-x_min)/(x_max-x_min)
17
18        fig=plt.figure()
19        ax=plt.subplot(111)
20        color=['b','c','g','k','m','r']
21        colors=['rgba(88,87,86,1.0)', 'rgba(227,23,13,1.0)', 'rgba(255,192,203,1.0)', 'rgba(3,168,158,1.0)', 'rgba(227,207,87,1.0)', 'rgba(138,43,226,1.0)']
22        #colors=['rgba('+str(r)+','+str(g)+','+str(b)+',1.0)' for r in np.linspace(25,168,6) for g in np.linspace(40,80,6) for b in np.linspace(0,255,6)]
23        name=['buttercup','daisy','iris','lilyvalley','sunflower','windflower']
24        Index=0
25        PData=[]
26
27        for co in color:
28            Index=Index+1
29            tempData=[]
30            for i in range(data.shape[0]):
31                if label[i]==Index:
32                    tempData.append(data[i,:])
33            tempData=np.array(tempData)
34            print(tempData.shape)
35            trace=go.Scatter(
36                x=tempData[:,0],
37                y=tempData[:,1],
38                name='{0}'.format(name[Index-1]),
39                mode='markers',
40                marker=dict(
41                    size=5,
42                    color=colors[Index-1]
43                )
44            )
45            py.iplot([trace],filename=title)
46
```

```

40 )
41 PData.append(trace)
42 #plt.plot(data[i,0],data[i,1],color=color[label[i]-1],label="{0}".format(name[label[i]-1]))
43 plt.plot(tempData[:,0],tempData[:,1],'o',color=co,label="{0}").format(name[Index-1]))
44 plt.legend(numpoints=1)
45 plt.xticks([])
46 plt.yticks([])
47 #plt.legend(numpoints=1)
48 plt.title(title)

49
50 layout=go.Layout(
51     title='t-SNE.visualization_for.texture_features',
52     yaxis=dict(zeroline=False),xaxis=dict(zeroline=False)
53 )
54 Pfig = go.Figure(data=PData, layout=layout)
55 plotly.offline.plot(Pfig,filename='VB-TEXTURE-tSNE.html')
56 return fig

57
58 #定义并计算卡方统计量距离
59 def chi2_distance(histA,histB,eps=1e-10):
60     d=0.5*np.sum([(a-b)**2/(a+b+eps) for (a,b) in zip(histA,histB)])
61     return d

62
63 #图像分割函数
64 def segmentation(img):
65     imgHSV=cv2.cvtColor(img,cv2.COLOR_BGR2HSV)
66     _, _, imgV = cv2.split(imgHSV)
67     m,n,c=imgHSV.shape
68     imgV=np.array(imgV)/255
69     imgVF=imgV.flatten()
70     meanV=np.mean(imgVF,axis=0)
71     varV=np.std(imgVF,axis=0)
72     Threshold=meanV+varV
73     fmask=np.zeros((m,n),dtype=np.uint8)
74     mask=np.array(imgV>Threshold,dtype=np.uint8)
75     img_segmentation=cv2.bitwise_and(img,img,mask=mask)
76     return img_segmentation

77
78 #测试分割效果
79 if __name__=="__main__":
80     filename='F:/seven/prDesign/test2.jpg'
81     img=cv2.imread(filename)
82     img_segmentation=segmentation(img)
83     img_segmentation=cv2.resize(img_segmentation,(160,160))
84     cv2.imwrite('F:/seven/prDesign/seg21.jpg', img_segmentation, [int(cv2.IMWRITE_JPEG_QUALITY),
85                                         100])
86     cv2.imshow('ori',img)
87     cv2.imshow('seg',img_segmentation)
88     cv2.waitKey(0)

```

C.2 文件夹: fsvm

main.py

```

1      """
2      方案二的主程序模块
3      """
4
5      import glob
6      import cv2
7      import numpy as np
8      import extractionFeature as exF
9      import msym
10     import sklearn.svm as svm
11     from sklearn.cluster import KMeans
12     from sklearn import manifold
13     import matplotlib.pyplot as plt
14     import chart_studio.plotly as py
15     import plotly.graph_objs as go
16     import plotly
17     from sklearn.metrics import plot_confusion_matrix
18
19     #t-SNE可视化绘图函数
20     def plot_embedding(data,label,title):
21         x_min,x_max=np.min(data,0),np.max(data,0)
22         data=(data-x_min)/(x_max-x_min)
23
24         fig=plt.figure()
25         ax=plt.subplot(111)
26         color=['b','c','g','k','m','r']
27         colors=[rgba(88,87,86,1.0)',rgba(227,23,13,1.0)',rgba(255,192,203,1.0)',rgba(3,168,158,1.0)',rgba(227,207,87,1.0)',rgba(138,43,226,1.0)']
28         #colors=[rgba('+str(r)+','+str(g)+','+str(b)+',1.0)' for r in np.linspace(25,168,6) for g in np.linspace(40,80,6) for b in np.linspace(0,255,6)]
29         name=['buttercup','daisy','iris','lilyvalley','sunflower','windflower']
30         Index=0
31         PData=[]
32         for co in color:
33             Index=Index+1
34             tempData=[]
35             for i in range(data.shape[0]):
36                 if label[i]==Index:
37                     tempData.append(data[i,:])
38             tempData=np.array(tempData)
39             print(tempData.shape)
40             trace=go.Scatter(
41                 x=tempData[:,0],
42                 y=tempData[:,1],
43                 name='{}' .format(name[Index-1]),
44                 mode='markers',
45                 marker=dict(

```

```

46         color=colors[Index-1]
47     )
48 )
49 PData.append(trace)
50 #plt.plot(data[i,0],data[i,1],color=color[label[i]-1],label="{0}".format(name[label[i]-1]))
51 plt.plot(tempData[:,0],tempData[:,1],'o',color=co,label="{0}").format(name[Index-1]))
52 plt.legend(numpoints=1)
53 plt.xticks([])
54 plt.yticks([])
55 #plt.legend(numpoints=1)
56 plt.title(title)
57
58 layout=go.Layout(
59     title='t-SNE.visualization_for_Fusion_features',
60     yaxis=dict(zeroline=False),xaxis=dict(zeroline=False)
61 )
62 Pfig = go.Figure(data=PData, layout=layout)
63 plotly.offline.plot(Pfig,filename='VB-FUSION-tSNE.html')
64 return fig
65
66 #加载数据
67 filenames=[]
68 filenames.append(glob.glob(r'F:/seven/prDesign/fdata/buttercup/*.jpg'))
69 filenames.append(glob.glob(r'F:/seven/prDesign/fdata/daisy/*.jpg'))
70 filenames.append(glob.glob(r'F:/seven/prDesign/fdata/iris/*.jpg'))
71 filenames.append(glob.glob(r'F:/seven/prDesign/fdata/lilyvalley/*.jpg'))
72 filenames.append(glob.glob(r'F:/seven/prDesign/fdata/sunflower/*.jpg'))
73 filenames.append(glob.glob(r'F:/seven/prDesign/fdata/windflower/*.jpg'))
74 #定义存储提取好的特征向量的文件路径
75 root="F:/seven/prDesign/data/"
76 tail=".txt"
77 HCB=[]
78 SCB=[]
79 VCB=[]
80 GCB=[]
81 ECB=[]
82 HC=[]
83 SC=[]
84 VC=[]
85 GC=[]
86 EC=[]
87 for i in range(21):
88     HCB.append(root+"HCB"+str(i+1)+tail)
89     SCB.append(root+"SCB"+str(i+1)+tail)
90     VCB.append(root+"VCB"+str(i+1)+tail)
91     GCB.append(root+"GCB"+str(i+1)+tail)
92     ECB.append(root+"ECB"+str(i+1)+tail)
93     HC.append(root+"HC"+str(i+1)+tail)
94     SC.append(root+"SC"+str(i+1)+tail)
95     VC.append(root+"VC"+str(i+1)+tail)
96     GC.append(root+"GC"+str(i+1)+tail)

```

```

97     EC.append(root+"EC"+str(i+1)+tail)
98
99     """
100    #生成特征并存储
101    for i in range(21):
102        if i==0:
103            level=0
104            split=False
105            split_op=0
106        elif i>0 and i<5:
107            level=1
108            split=True
109            split_op=i-1
110        else:
111            level=2
112            split=True
113            split_op=i-5
114        print('level:split_op:',level,split_op)
115        exF.saveHSVcode(filenames,HCB[i],SCB[i],VCB[i],HC[i],SC[i],VC[i],split,split_op,level)
116        print('Hsv')
117        exF.saveGcode(filenames,GCB[i],GC[i],split,split_op,level)
118        print('Gray')
119        exF.saveEcode(filenames,ECB[i],EC[i],split,split_op,level)
120        print('Edge')
121
122    """"
123    #读取已经生成好的特征并融合
124    FCode=np.zeros((480,21*300))
125    for i in range(21):
126        tempHCode=np.loadtxt(HC[i])
127        #tempHCode=np.nan_to_num(tempHCode)
128        FCode[:,180*i:180*i+60]=tempHCode
129        tempSCode=np.loadtxt(SC[i])
130        #tempSCode=np.nan_to_sum(tempSCode)
131        FCode[:,180*i+60:180*i+120]=tempSCode
132        tempVCode=np.loadtxt(VC[i])
133        FCode[:,180*i+120:180*i+180]=tempVCode
134        tempGCode=np.loadtxt(GC[i])
135        FCode[:,3780+60*i:3840+60*i]=tempGCode
136        tempECode=np.loadtxt(EC[i])
137        FCode[:,5040+60*i:5100+60*i]=tempECode
138        FCode=np.nan_to_num(FCode)
139        print(FCode.shape)
140        print(FCode[0])
141
142    imgNum=80
143    chosen_train=40
144    #定义标签
145    y=None
146    for i in range(6):
147        tempY=(i+1)*np.ones(chosen_train,dtype=int)
148        if y is None:

```

```

148     y=tempY
149   else:
150     y=np.concatenate((y,tempY),axis=0)
151   #生成训练样本和测试样本
152   Ftrain=np.zeros((6*40,6300))
153   Ftest=np.zeros((6*40,6300))
154   ytrain=y[:,]
155   ytest=y[:,]
156
157   yH=np.concatenate((y,y),axis=0)
158   #"""
159   #使用t-SNE数据可视化
160   tsne=manifold.TSNE(n_components=2,init='pca',random_state=0)
161   Y=tsne.fit_transform(FCode)
162   print('tSNE...completed!')
163   fig=plt_embedding(Y,yH,'t-SNE.embedding.of.the.digits')
164   plt.show()
165   #"""
166
167   for i in range(6):
168     print(i)
169     randomIndex,resIndex=msvm.randomSample(imgNum,chosen_train)
170     print(len(randomIndex),len(resIndex))
171     Ftrain[40*i:40*(i+1),:]=FCode[randomIndex+80*i,:]
172     Ftest[40*i:40*(i+1),:]=FCode[resIndex+80*i,:]
173   #分类和测试
174   isvm=msvm.svmClassification(Ftrain,ytrain)
175   #isvm=svm.SVC(kernel='rbf')
176   #isvm.fit(Ftrain,ytrain)
177   FNtest=msvm.toKernel(Ftest,Ftrain)
178   score=isvm.score(FNtest,ytest)
179   print(score)
180
181   #分类测试示例
182   testFile='F:/seven/prDesign/test6.jpg'
183   img=cv2.imread(testFile)
184   img=cv2.resize(img,(300,300))
185   tHCode=np.zeros((21,60))
186   tSCode=np.zeros((21,60))
187   tVCode=np.zeros((21,60))
188   tGCode=np.zeros((21,60))
189   tECode=np.zeros((21,60))
190   for i in range(21):
191     if i==0:
192       level=0
193       split=False
194       split_op=0
195     elif i>0 and i<5:
196       level=1
197       split=True
198       split_op=i-1

```

```

199     else:
200         level=2
201         split=True
202         split_op=i-5
203         print('level:split_op:',level,split_op)
204         tHCode[i],tSCode[i],tVCode[i]=exF.PHOW_HSV(img,HCB[i],SCB[i],VCB[i],split,split_op,level)
205         print('Hsv')
206         tGCode[i]=exF.PHOW_G(img,GCB[i],split,split_op,level)
207         print('Gray')
208         tECode[i]=exF.PHOW_E(img,ECB[i],split,split_op,level)
209         print('Edge')
210         tFCode=np.zeros(6300)
211     for i in range(21):
212         tFCode[180*i:180*i+60]=tHCode[i]
213         tFCode[180*i+60:180*i+120]=tSCode[i]
214         tFCode[180*i+120:180*i+180]=tVCode[i]
215         tFCode[3780+60*i:3840+60*i]=tGCode[i]
216         tFCode[5040+60*i:5100+60*i]=tECode[i]
217         tFCode=np.nan_to_num(tFCode)
218         print(tFCode.shape)
219         FNtest=msvm.toKernel(tFCode.reshape(1,-1),Ftrain)
220         cls=isvm.predict(FNtest)
221         print(cls)
222         np.set_printoptions(precision=2)
223         name=['buttercup','daisy','iris','lilyvalley','sunflower','windflower']
224         # Plot non-normalized confusion matrix
225         titles_options = [("Normalized_confusion_matrix_for_SVM-HIK", 'true')]
226     for title, normalize in titles_options:
227         disp = plot_confusion_matrix(isvm, FNtest, y,
228                                     display_labels=name,
229                                     cmap=plt.cm.Blues,
230                                     normalize=normalize)
231         disp.ax_.set_title(title)
232
233         print(title)
234         print(disp.confusion_matrix)
235
236     plt.show()

```

dSIFT.py

```

1      """
2      denseSIFT特征提取模块
3      """
4
5      import numpy as np
6      import cv2
7
8      #定义denseSIFT特征提取算子
9      class DenseSIFT():
10         def __init__(self):
11             self.sift=cv2.xfeatures2d.SIFT_create()

```

```

12 #检测和提取denseSIFT特征
13 def detectAndCompute(self,image,step_size=8,window_size=(16,16)):
14     if window_size is None:
15         winH,winW=image.shape[2]
16         window_size=(winW//4,winH//4)
17
18     descriptors=np.array([],dtype=np.float32).reshape(0,128)
19     for crop in self._crop_image(image,step_size,window_size):
20         tmp_descriptor=self._detectAndCompute(crop)[1]
21         if tmp_descriptor is None:
22             continue
23         descriptors=np.vstack([descriptors,tmp_descriptor])
24     return descriptors
25
26 def _detect(self,image):
27     return self.sift.detect(image)
28
29 def _compute(self,image,kps,eps=1e-7):
30     kps,descs=self.sift.compute(image,kps)
31
32     if len(kps)==0:
33         return [],None
34
35     descs/==(descs.sum(axis=1,keepdims=True)+eps)
36     descs=np.sqrt(descs)
37     return kps,descs
38
39 def _detectAndCompute(self,image):
40     kps=self._detect(image)
41     return self._compute(image,kps)
42
43 def _sliding_window(self,image,step_size,window_size):
44     for y in range(0,image.shape[0],step_size):
45         for x in range(0,image.shape[1],step_size):
46             yield (x,y,image[y:y+window_size[1],x:x+window_size[0]])
47
48 def _crop_image(self,image,step_size=8,window_size=(16,16)):
49     crops=[]
50     winH,winW=window_size
51     for (x,y,window) in self._sliding_window(image,step_size=step_size,window_size=window_size):
52         if window.shape[0]!=winH or window.shape[1]!=winW:
53             continue
54         crops.append(image[y:y+winH,x:x+winW])
55     return np.array(crops)
56
57 #测试函数
58 if __name__=="__main__":
59     img=cv2.imread("F:/seven/prDesign/fdata/iris/image_0401.jpg")
60     imgHSV=cv2.cvtColor(img,cv2.COLOR_BGR2HSV)
61     Densift=DenseSIFT()
62     des1=Densift.detectAndCompute(imgHSV[:, :, 0])

```

```

63     des2=Densift.detectAndCompute(imgHSV[:, :, 1])
64     des3=Densift.detectAndCompute(imgHSV[:, :, 2])
65     print(des1.shape)
66     print(des2.shape)
67     print(des3.shape)

```

extractionFeature.py

```

1      """
2      特征提取模块
3      """
4
5      import numpy as np
6      import cv2
7      import dSIFT
8      from sklearn.cluster import MiniBatchKMeans
9
10     #矩阵乘方
11     def matrixPow(X, gamma):
12         m, n = X.shape
13         v, Q = np.linalg.eig(X)
14         V = np.diag(v ** (gamma))
15         result = np.dot(Q, V)
16         result = np.dot(result, np.mat(Q).T)
17         return np.real(result)
18
19     #按空间金字塔要求将图像4等分
20     def splitImage(img):
21         m, n = img.shape[0], img.shape[1]
22         centerR = int(m / 2)
23         centerC = int(n / 2)
24         img1 = img[:centerR, :centerC, :]
25         img2 = img[:centerR, centerC:, :]
26         img3 = img[centerR:, :centerC, :]
27         img4 = img[centerR:, centerC:, :]
28         return img1, img2, img3, img4
29
30     #按空间金字塔要求将图像4等分
31     def splitGEImage(img):
32         m, n = img.shape[0], img.shape[1]
33         centerR = int(m / 2)
34         centerC = int(n / 2)
35         img1 = img[:centerR, :centerC]
36         img2 = img[:centerR, centerC:]
37         img3 = img[centerR:, :centerC]
38         img4 = img[centerR:, centerC:]
39         return img1, img2, img3, img4
40
41     #图像分割
42     def segmentation(img):
43         imgHSV = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
44         _, _, imgV = cv2.split(imgHSV)
45         m, n, c = imgHSV.shape
46         imgV = np.array(imgV) / 255
47         imgVF = imgV.flatten()
48         meanV = np.mean(imgVF, axis=0)

```

```

45     varV=np.std(imgVF, axis=0)
46     Threshold=meanV+varV
47     fmask=np.zeros((m,n), dtype=np.uint8)
48     mask=np.array(imgV>Threshold, dtype=np.uint8)
49     img_segmentation=cv2.bitwise_and(img, img, mask=mask)
50     return img_segmentation
51
52 #获得编码字典
53 def getCodebook(H):
54     kmean=MiniBatchKMeans(n_clusters=300)
55     kmean.fit(H)
56     return kmean.cluster_centers_
57
58 #LLC编码函数
59 def LLCoding(codebook,x):
60     disMatrix=np.zeros(len(codebook))
61     for i in range(len(codebook)):
62         disMatrix[i]=np.linalg.norm(codebook[i,:]-x)
63     index=np.argsort(disMatrix, axis=0)
64     Chosenindex=index[:60]
65     B_new=codebook[Chosenindex,:]
66     e1=np.ones((60,1), dtype=float)
67     bm=B_new-np.dot(e1, x.reshape(1,-1))
68     b_hat=np.dot(bm, np.mat(bm).T)
69     c_hat=np.dot(matrixPow(b_hat, -1.), e1).reshape(-1)
70     c=c_hat/np.linalg.norm(c_hat)
71     return c
72
73 #保存编码字典
74 def saveCodebook(pathName, codebook):
75     np.savetxt(pathName, codebook)
76
77 #提取HSV编码字典和特征编码
78 def extractHSVCodebook(filenames, HcodeBookname, ScodeBookname, VcodeBookname, Hcodename,
79                         Scodename, Vcodename, split=False, split_po=0, level=0):
80     n = len(filenames)
81     print(filenames[0])
82     densift = dSIFT.DenseSIFT()
83     H = None
84     lenH=np.zeros(len(filenames), dtype=int)
85     S = None
86     lenS=np.zeros(len(filenames), dtype=int)
87     V = None
88     lenV=np.zeros(len(filenames), dtype=int)
89     for i in range(n):
90         print(i)
91         img = cv2.imread(filenames[i])
92         img=segmentation(img)
93         imgHSV = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
94         if split:
95             if level==1:
96                 simgHSV=splitImage(imgHSV)
97                 imgHSV=simgHSV[split_po]
98             elif level==2:
99                 simgHSV=splitImage(imgHSV)

```

```

95         chose_po=int(split_po/4)
96         which_po=split_po%4
97         imgHSV=splitImage(simgHSV[chose_po])[which_po]
98         desH = densift.detectAndCompute(imgHSV[:, :, 0])
99         lenH[i]=len(desH)
100        desS = densift.detectAndCompute(imgHSV[:, :, 1])
101        lenS[i]=len(desS)
102        desV = densift.detectAndCompute(imgHSV[:, :, 2])
103        lenV[i]=len(desV)
104        if S is None:
105            H = desH
106            S = desS
107            V = desV
108        else:
109            H = np.concatenate((H, desH), axis=0)
110            S = np.concatenate((S, desS), axis=0)
111            V = np.concatenate((V, desV), axis=0)
112            print('dSIFT_complete')
113            Hcodebook = getCodebook(H)
114            Scodebook = getCodebook(S)
115            Vcodebook = getCodebook(V)
116            saveCodebook(HcodeBookname,Hcodebook)
117            saveCodebook(ScodeBookname,Scodebook)
118            saveCodebook(VcodeBookname,Vcodebook)
119
120            sumH=0
121            sumS=0
122            sumV=0
123            HCode=np.zeros((n,60))
124            SCode=np.zeros((n,60))
125            VCode=np.zeros((n,60))
126            for i in range(n):
127                print(i,"th_llc_coding")
128
129                tempH=np.zeros((lenH[i],60))
130                for k in range(lenH[i]):
131                    tempH[k,:]=LLCoding(Hcodebook,H[sumH+k,:])
132                    sumH=sumH+lenH[i]
133                    HCode[i,:]=np.mean(tempH, axis=0)
134
135                tempS=np.zeros((lenS[i],60))
136                for k in range(lenS[i]):
137                    tempS[k,:]=LLCoding(Scodebook,S[sumS+k,:])
138                    sumS=sumS+lenS[i]
139                    SCode[i,:]=np.mean(tempS, axis=0)
140
141                tempV=np.zeros((lenV[i],60))
142                for k in range(lenV[i]):
143                    tempV[k,:]=LLCoding(Vcodebook,V[sumV+k,:])
144                    sumV=sumV+lenV[i]
145                    VCode[i,:]=np.mean(tempV, axis=0)

```

```

146
147
148     np.savetxt(Hcodename,HCode)
149     np.savetxt(Scodename,SCode)
150     np.savetxt(Vcodename,VCode)
151
152 #提取HSV的PHOW特征
153 def PHOW_HSV(img,Hcodebookname,Scodebookname,Vcodebookname,split=False,split_po=0,level=0):
154     Hcodebook=np.loadtxt(Hcodebookname)
155     Scodebook=np.loadtxt(Scodebookname)
156     Vcodebook=np.loadtxt(Vcodebookname)
157
158     img = segmentation(img)
159     imgHSV = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
160     if split:
161         if level == 1:
162             simgHSV = splitImage(imgHSV)
163             imgHSV = simgHSV[split_po]
164         elif level == 2:
165             simgHSV = splitImage(imgHSV)
166             chose_po = int(split_po / 4)
167             which_po = split_po % 4
168             imgHSV = splitImage(simgHSV[chose_po])[which_po]
169     densift=dSIFT.DenseSIFT()
170     desH=densift.detectAndCompute(imgHSV[:, :, 0])
171     desS=densift.detectAndCompute(imgHSV[:, :, 1])
172     desV=densift.detectAndCompute(imgHSV[:, :, 2])
173     HCode=np.zeros((len(desH),60))
174     SCode=np.zeros((len(desS),60))
175     VCode=np.zeros((len(desV),60))
176     for i in range(len(desH)):
177         #print(H[i,:].shape)
178         print('H_coding:',i)
179         HCode[i,:]=LLCoding(Hcodebook,desH[i,:])
180     for i in range(len(desS)):
181         print('S_coding:',i)
182         SCode[i,:]=LLCoding(Scodebook,desS[i,:])
183     for i in range(len(desV)):
184         print('V_coding:',i)
185         VCode[i,:]=LLCoding(Vcodebook,desV[i,:])
186     print('LLC_coding_complete')
187
188     Hfinal=np.mean(HCode[:,],axis=0).reshape(1,-1)
189     Sfinal=np.mean(SCode[:,],axis=0).reshape(1,-1)
190     Vfinal=np.mean(VCode[:,],axis=0).reshape(1,-1)
191
192     return Hfinal,Sfinal,Vfinal
193 #存储HSV的特征编码
194 def saveHSVcode(filenames,Hcodebookname,Scodebookname,Vcodebookname,Hcodename,Scodename,
195     Vcodename,split=False,split_po=0,level=0):
196     Filename=[]

```

```

196     for filename in filenames:
197         for f in filename:
198             Filename.append(f)
199             extractHSVCodebook(Filename,Hcodebookname,Scodebookname,Vcodebookname,Hcodename,
200                               Scodename,Vcodename,split,split_po,level)
201
202     #提取灰度图的编码字典和特征编码
203     def extractGCodebook(filenames,GcodeBookname,Gcodename,split=False,level=0,split_po=0):
204         n = len(filenames)
205         print(filenames[0])
206         densift = dSIFT.DenseSIFT()
207         G = None
208         lenG=np.zeros(n,dtype=int)
209         for i in range(n):
210             print(i)
211             img = cv2.imread(filenames[i])
212             img=segmentation(img)
213             imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
214             if split:
215                 if level==1:
216                     simgHSV=splitGEImage(imgGray)
217                     imgGray=simgHSV[split_po]
218                 elif level==2:
219                     simgHSV=splitGEImage(imgGray)
220                     chose_po=int(split_po/4)
221                     which_po=split_po%4
222                     imgGray=splitGEImage(simgHSV[chose_po])[which_po]
223                     desG = densift.detectAndCompute(imgGray[:, :])
224                     lenG[i]=len(desG)
225                     if G is None:
226                         G = desG
227                     else:
228                         G = np.concatenate((G, desG), axis=0)
229                     print('dSIFT_complete')
230                     Gcodebook = getCodebook(G)
231                     saveCodebook(GcodeBookname,Gcodebook)
232
233                     sumG = 0
234                     GCode = np.zeros((n, 60))
235                     for i in range(n):
236                         print(i, "th_LLC_coding")
237                         tempG = np.zeros((lenG[i], 60))
238                         for k in range(lenG[i]):
239                             tempG[k, :] = LLCoding(Gcodebook, G[sumG + k, :])
240                             sumG = sumG + lenG[i]
241                             GCode[i, :] = np.mean(tempG, axis=0)
242
243                         np.savetxt(Gcodename, GCode)
244                         #提取edge-SIFT的编码字典和特征编码
245                         def extractECodebook(filenames,EcodeBookname,Ecodename,split=False,level=0,split_po=0):
246                             n = len(filenames)

```

```

246 print(filenames[0])
247 densift = dSIFT.DenseSIFT()
248 E = None
249 lenE=np.zeros(n,dtype=int)
250 for i in range(n):
251     print(i)
252     img = cv2.imread(filenames[i])
253     img=segmentation(img)
254     imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
255     if split:
256         if level==1:
257             simgHSV=splitGEImage(imgGray)
258             imgGray=simgHSV[split_po]
259         elif level==2:
260             simgHSV=splitGEImage(imgGray)
261             chose_po=int(split_po/4)
262             which_po=split_po%4
263             imgGray=splitGEImage(simgHSV[chose_po])[which_po]
264             imgEdge=cv2.Canny(imgGray,200,300)
265             desE = densift.detectAndCompute(imgEdge[:, :])
266             lenE[i]=len(desE)
267             if E is None:
268                 E = desE
269             else:
270                 E = np.concatenate((E, desE), axis=0)
271 print('dSIFT complete')
272 Ecodebook = getCodebook(E)
273 saveCodebook(EcodeBookname,Ecodebook)
274
275 sumE = 0
276 ECode = np.zeros((n, 60))
277 for i in range(n):
278     print(i, "th_LLC_coding")
279     if lenE[i]==0:
280         ECode[i,:]=np.zeros(60)
281     else:
282         tempE = np.zeros((lenE[i], 60))
283         for k in range(lenE[i]):
284             tempE[k, :] = LLCoding(Ecodebook, E[sumE + k, :])
285         sumE = sumE + lenE[i]
286         ECode[i, :] = np.mean(tempE, axis=0)
287
288 np.savetxt(Ecodename, ECode)
289 #提取灰度图的PHOW特征
290 def PHOW_G(img,Gcodebookname,split=False,split_po=0,level=0):
291     Gcodebook=np.loadtxt(Gcodebookname)
292     img = segmentation(img)
293     imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
294     if split:
295         if level == 1:
296             simgHSV = splitGEImage(imgGray)

```

```

297     imgGray = simgHSV[split_po]
298     elif level == 2:
299         simgHSV = splitGEImage(imgGray)
300         chose_po = int(split_po / 4)
301         which_po = split_po % 4
302         imgGray = splitGEImage(simgHSV[chose_po])[which_po]
303
304         #imgGray=cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
305         densift=dSIFT.DenseSIFT()
306         desG=densift.detectAndCompute(imgGray[:, :])
307         GCode=np.zeros((len(desG),60))
308         for i in range(len(desG)):
309             #print(H[i, :].shape)
310             GCode[i, :] = LLCoding(Gcodebook, desG[i, :])
311         print('LLC_coding_complete')
312
313         Gfinal=np.mean(GCode[:, :], axis=0).reshape(1, -1)
314
315     return Gfinal
316 #提取edge-SIFT的PHOW特征
317 def PHOW_E(img, Ecodebookname, split=False, split_po=0, level=0):
318     Ecodebook=np.loadtxt(Ecodebookname)
319     img = segmentation(img)
320     imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
321     if split:
322         if level == 1:
323             simgHSV = splitGEImage(imgGray)
324             imgGray = simgHSV[split_po]
325         elif level == 2:
326             simgHSV = splitGEImage(imgGray)
327             chose_po = int(split_po / 4)
328             which_po = split_po % 4
329             imgGray = splitGEImage(simgHSV[chose_po])[which_po]
330             imgEdge = cv2.Canny(imgGray, 200, 300)
331             #imgGray=cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
332             #imgEdge=cv2.Canny(imgGray,200,300)
333             densift=dSIFT.DenseSIFT()
334             desE=densift.detectAndCompute(imgEdge[:, :])
335             ECode=np.zeros((len(desE),60))
336             for i in range(len(desE)):
337                 #print(H[i, :].shape)
338                 ECode[i, :] = LLCoding(Ecodebook, desE[i, :])
339             print('LLC_coding_complete')
340
341         Efinal=np.mean(ECode[:, :], axis=0).reshape(1, -1)
342
343     return Efinal
344 #保存灰度图的特征编码
345 def saveGcode(filenames, Gcodebookname, Gcodename, split=False, split_po=0, level=0):
346     Filename=[]
347     for filename in filenames:

```

```

348     for f in filename:
349         Filename.append(f)
350         extractGCodebook(Filename,Gcodebookname,Gcodename,split,level,split_po)
351 #保存edge-SIFT的特征编码
352 def saveEcode(filenames,Ecodebookname,Ecodename,split=False,split_po=0,level=0):
353     Filename=[]
354     for filename in filenames:
355         for f in filename:
356             Filename.append(f)
357             extractECodebook(Filename,Ecodebookname,Ecodename,split,level,split_po)

```

msvm.py

```

1     """
2     SVM分类函数模块
3     """
4
5     import numpy
6     import sklearn.svm as svm
7     import numpy as np
8
9     #随机分割训练集和测试集
10    def randomSample(imgNum,chosen_train):
11        index=np.arange(imgNum)
12        #print(index)
13        randomIndex=np.random.choice(index,size=chosen_train,replace=False)
14        setIndex=set(index)
15        setRan=set(randomIndex)
16        setresIndex=setIndex-setRan
17        resIndex=np.array(list(setresIndex))
18        print(randomIndex)
19        print(resIndex)
20        return randomIndex,resIndex
21
22    #定义核函数
23    def myKernel(x,y):
24        sum=0
25        for i in range(len(x)):
26            if x[i]<y[i]:
27                sum=sum+x[i]
28            else:
29                sum=sum+y[i]
30        return sum
31
32    #计算核函数
33    def toKernel(X,Y):
34        n=len(X)
35        m=len(Y)
36        result=np.zeros((n,m))
37        for i in range(n):
38            print(i)
39            for j in range(m):
40                result[i][j]=myKernel(X[i],Y[j])
41        return result

```

```

40 #svm分类
41 def svmClassification(X,y):
42     wsvm=svm.SVC(kernel='precomputed')
43     XN=toKernel(X,X)
44     wsvm.fit(XN,y)
45     return wsvm

```

C.3 文件夹：fSVMoRF

main.py

```

1 """
2 方案三和方案四：基于Sparse-SIFT和STFA的花卉识别方法
3 """
4 import numpy as np
5 import glob
6 import load
7 import cv2
8 import featureExtract as fE
9 import matplotlib
10 from sklearn.ensemble import RandomForestClassifier
11 from sklearn.model_selection import cross_val_score
12 from sklearn.ensemble import ExtraTreesClassifier
13 from sklearn.tree import DecisionTreeClassifier
14 import sklearn
15 import matplotlib.pyplot as plt
16 from sklearn import manifold
17 from sklearn import svm, datasets
18 from sklearn.model_selection import train_test_split
19 from sklearn.metrics import plot_confusion_matrix
20 import sfta
21
22 #matplotlib.use('TkAgg')
23 #读取图片文件
24 filenames=[]
25 filenames.append(glob.glob(r'F:/seven/prDesign/fdata/buttercup/*.jpg'))
26 filenames.append(glob.glob(r'F:/seven/prDesign/fdata/daisy/*.jpg'))
27 filenames.append(glob.glob(r'F:/seven/prDesign/fdata/iris/*.jpg'))
28 filenames.append(glob.glob(r'F:/seven/prDesign/fdata/lilyvalley/*.jpg'))
29 filenames.append(glob.glob(r'F:/seven/prDesign/fdata/sunflower/*.jpg'))
30 filenames.append(glob.glob(r'F:/seven/prDesign/fdata/windflower/*.jpg'))
31 nt=10#设定阈值分割的阈值个数
32
33 #加载图片文件并做STFA特征提取，将提取出的特征存入对应文件中
34 #data,label=load.load(filenames,nt)
35 #np.savetxt('F:/seven/prDesign/fSVMoRF/data/data.txt',data)
36 #np.savetxt('F:/seven/prDesign/fSVMoRF/data/label.txt',label)
37
38

```

```

39 #读取已经获得的STFA特征或者Sparse-SIFT特征
40 data=np.loadtxt('data/data10.txt')
41 label=np.loadtxt('data/label10.txt').astype(np.int)
42 print(data.shape)
43 dmin=np.min(data,axis=0)
44 dmax=np.max(data,axis=0)
45 data=(data-dmin)/(dmax-dmin)#归一化
46 #print(data)
47 #print(label)
48
49 #提取Sparse-SIFT特征并存入对应文件
50 #data,label=load.loadSIFT(filenames)
51 #print(data.shape)
52 #sdata=fE.SIFT(data)
53 #np.savetxt('data/sift.txt',sdata)
54 #print(sdata.shape)
55
56 #数据可视化
57 tsne=manifold.TSNE(n_components=2,init='pca',random_state=0)
58 Y=tsne.fit_transform(data)
59 print('tSNE completed!')
60 fig=load.plot_embedding(Y,label,'t-SNE embedding of the SIFT features')
61 plt.show()
62
63 #分割数据集为训练集和测试集
64 train_X,test_X,train_y,test_y=sklearn.model_selection.train_test_split(data,label,test_size=0.5,random_state
       =42,stratify=label)
65 #读取测试图片的特征
66 testFile='F:/seven/prDesign/test6.jpg'
67 tData=cv2.imread(testFile)
68 tData_=fE.segment(tData)
69 #ttData=[]
70 #tData=cv2.cvtColor(tData,cv2.COLOR_BGR2GRAY)
71 tData=cv2.resize(tData,dsize=(330,250))
72 #ttData.append(tData)
73 #ttData=np.array(ttData)
74 tData=sfta.sfta(tData,nt)
75 print('image.sfta.completed!')
76 #tData=fE.SIFT(ttData)
77 tData=(tData-dmin)/(dmax-dmin)#归一化
78 #决策树分类模型
79 clf1=DecisionTreeClassifier(max_depth=None,min_samples_split=2,random_state=0)
80 clf1.fit(train_X,train_y)
81 score1=clf1.score(test_X,test_y)
82 print('Decision.Tree:',score1)
83 print(clf1.predict(tData.reshape(1,-1)))
84 np.set_printoptions(precision=2)
85 name=['buttercup','daisy','iris','lilyvalley','sunflower','windflower']
86 # Plot non-normalized confusion matrix
87 titles_options = [{"Normalized_confusion_matrix_for_Decision_Tree", 'true'}]
88 for title, normalize in titles_options:

```

```

89     disp = plot_confusion_matrix(clf1, test_X, test_y,
90                                   display_labels=name,
91                                   cmap=plt.cm.Blues,
92                                   normalize=normalize)
93     disp.ax_.set_title(title)
94
95     print(title)
96     print(disp.confusion_matrix)
97
98     plt.show()
99 #随机森林分类模型
100 clf2=RandomForestClassifier(n_estimators=10,max_depth=None,min_samples_split=2,random_state=0)
101 clf2.fit(train_X,train_y)
102 score2=clf2.score(test_X,test_y)
103 print('Random.Forest:',score2)
104 print(clf2.predict(tData.reshape(1,-1)))
105 titles_options = [("Normalized.confusion.matrix.for.Random.Forest", 'true')]
106 for title, normalize in titles_options:
107     disp = plot_confusion_matrix(clf2, test_X, test_y,
108                                 display_labels=name,
109                                 cmap=plt.cm.Blues,
110                                 normalize=normalize)
111     disp.ax_.set_title(title)
112
113     print(title)
114     print(disp.confusion_matrix)
115
116     plt.show()
117 #支持向量机模型
118 clf3=sklearn.svm.SVC()
119 clf3.fit(train_X,train_y)
120 score3=clf3.score(test_X,test_y)
121 print('SVM:',score3)
122 print(clf3.predict(tData.reshape(1,-1)))
123 titles_options = [("Normalized.confusion.matrix.for.SVM", 'true')]
124 for title, normalize in titles_options:
125     disp = plot_confusion_matrix(clf3, test_X, test_y,
126                                 display_labels=name,
127                                 cmap=plt.cm.Blues,
128                                 normalize=normalize)
129     disp.ax_.set_title(title)
130
131     print(title)
132     print(disp.confusion_matrix)
133
134     plt.show()

```

load.py

```

1 """
2 数据加载模块
3 """

```

```

4   import cv2
5   import numpy as np
6   import matplotlib.pyplot as plt
7   import sfta
8   import featureExtract as fE
9   import chart_studio.plotly as py
10  import plotly.graph_objs as go
11  import plotly
12  #t-SNE可视化绘图函数
13  def plot_embedding(data,label,title):
14      x_min,x_max=np.min(data,0),np.max(data,0)
15      data=(data-x_min)/(x_max-x_min)
16
17      fig=plt.figure()
18      ax=plt.subplot(111)
19      color=['b','c','g','k','m','r']
20      colors=['rgba(88,87,86,1.0)', 'rgba(227,23,13,1.0)', 'rgba(255,192,203,1.0)', 'rgba(3,168,158,1.0)', 'rgba
21          (227,207,87,1.0)', 'rgba(138,43,226,1.0)']
22      #colors=['rgba('+str(r)+','+str(g)+','+str(b)+',1.0)' for r in np.linspace(25,168,6) for g in np.linspace
23          (40,80,6) for b in np.linspace(0,255,6)]
24      name=['buttercup','daisy','iris','lilyvalley','sunflower','windflower']
25      Index=0
26      PData=[]
27      for co in color:
28          Index=Index+1
29          tempData=[]
30          for i in range(data.shape[0]):
31              if label[i]==Index-1:
32                  tempData.append(data[i,:])
33          tempData=np.array(tempData)
34          print(tempData.shape)
35          trace=go.Scatter(
36              x=tempData[:,0],
37              y=tempData[:,1],
38              name='{0}'.format(name[Index-1]),
39              mode='markers',
40              marker=dict(
41                  size=5,
42                  color=colors[Index-1]
43              )
44          )
45          PData.append(trace)
46      plt.plot(data[i,0],data[i,1],color=color[label[i]-1],label="{0}".format(name[label[i]-1]))
47      plt.plot(tempData[:,0],tempData[:,1],'o',color=co,label="{0}".format(name[Index-1]))
48      plt.legend(numpoints=1)
49      plt.xticks([])
50      plt.yticks([])
51      plt.title(title)
52      layout=go.Layout(

```

```

53     title='t-SNE.visualization.for.SFTA.features',
54     yaxis=dict(zeroline=False),xaxis=dict(zeroline=False)
55   )
56 Pfig = go.Figure(data=PData, layout=layout)
57 plotly.offline.plot(Pfig,filename='SFTA-tSNE.html')
58 return fig
59
60 #用于SFTA特征提取的数据加载函数
61 def load(filenames,nt):
62   data=[]
63   label=[]
64   k=0
65   for filename in filenames:
66     for i in range(len(filename)):
67       tempData=cv2.imread(filename[i])
68       tempData,_=fE.segment(tempData)
69       tempData=cv2.resize(tempData,dsize=(330,250))
70       tempData=sfta.sfta(tempData,nt)
71       print(i,'th.image.sfta.completed!')
72       data.append(tempData)
73       label.append(k)
74     k=k+1
75   return np.array(data,dtype=np.float),np.array(label)
76
77 #用于SIFT特征提取的数据加载函数
78 def loadSIFT(filenames):
79   data=[]
80   label=[]
81   k=0
82   for filename in filenames:
83     for i in range(len(filename)):
84       tempData=cv2.imread(filename[i])
85       tempData=fE.segment(tempData)
86       tempData=cv2.cvtColor(tempData,cv2.COLOR_BGR2GRAY)
87       tempData=cv2.resize(tempData,dsize=(330,250))
88       print(i,'th.image.sift.completed!')
89       data.append(tempData)
90       label.append(k)
91     k=k+1
92   return np.array(data,dtype=np.uint8),np.array(label)

```

ksvd.py

```

1 """
2 K-SVD稀疏编码模块
3 """
4 import numpy as np
5 from sklearn import linear_model
6
7 #定义K-SVD稀疏编码方法
8 class KSVD(object):
9   def __init__(self,n_components,max_iter=10,tol=1e-6):

```

```

10     self.dictionary=None
11     self.sparsecode=None
12     self.max_iter=max_iter
13     self.tol=tol
14     self.n_components=n_components
15
16     def __init__(self,y):
17         u,s,v=np.linalg.svd(y)
18         self.dictionary=u[:,self.n_components]
19
20     def _update_dict(self,y,d,x):
21         for i in range(self.n_components):
22             index=np.nonzero(x[i,:])[0]
23             if len(index)==0:
24                 continue
25
26             d[:,i]=0
27             r=(y-np.dot(d,x))[:,index]
28             u,s,v=np.linalg.svd(r,full_matrices=False)
29             x[i,index]=s[0]*v[0,:]
30
31             return d,x
32
33     def fit(self,y):
34         self.__init__(y)
35         for i in range(self.max_iter):
36             x=linear_model.orthogonal_mp(self.dictionary,y)
37             print('linear_model')
38             e=np.linalg.norm(y-np.dot(self.dictionary,x))
39             if e<self.tol:
40                 self._update_dict(y,self.dictionary,x)
41                 print(i,'th.ksvd completed!')
42
43         self.sparsecode=linear_model.orthogonal_mp(self.dictionary,y)
44         return self.dictionary,self.sparsecode

```

featureExtract.py

```

1     """
2 Sparse-SIFT特征提取模块和图像分割模块
3 """
4 import numpy as np
5 import sSIFT
6 import ksvd
7 import cv2
8 import glob
9
10 #基于OTSU方法的图像分割函数
11 def segment(img):
12     uimg=cv2.cvtColor(img,cv2.COLOR_BGR2Lab)
13     udata=uimg[:, :, 1]
14     Lmax=np.max(udata)
15     muT=np.mean(np.mean(udata, axis=0))

```

```

16 Object=0
17 T=0
18 T,mask=cv2.threshold(udata,0,Lmax,cv2.THRESH_OTSU)
19 img_segmentation=cv2.bitwise_and(img,img,mask=mask)
20 return img_segmentation,mask
21
22 #稀疏编码的SIFT特征提取函数
23 def SIFT(data):
24     n=len(data)
25     sift_feature=np.zeros((n,128))
26     detector=sSIFT.DenseSIFT()
27     for i in range(n):
28         temp_feature=detector.detectAndCompute(data[i,:,:])
29         print(i,temp_feature.shape)
30         skvd=ksvd.KSVD(2720)
31         dictionary,sparse_code=skvd.fit(temp_feature)
32         sift_feature[i]=np.mean(sparse_code, axis=0).astype(float)
33     return sift_feature
34
35 #测试函数
36 if __name__=="__main__":
37     img = cv2.imread("F:/seven/prDesign/test3.jpg")
38     img_re=cv2.resize(img,(160,160))
39     """
40     filenames = sorted(glob.glob(r'F:/seven/prDesign/oxford102/segmim/*.jpg'))
41     root='F:/seven/prDesign/oxford102/mask/'
42     for m in range(8189):
43         img=cv2.imread(filenames[m])
44         print(filenames[m])
45         mask=np.zeros((img.shape[0],img.shape[1]),dtype=np.uint8)
46         for i in range(img.shape[0]):
47             for j in range(img.shape[1]):
48                 if img[i,j,0]==254 and img[i,j,1]==0 and img[i,j,2]==0:
49                     mask[i][j]=0
50                 else:
51                     mask[i][j]=255
52         mask_file=root+str(m+1)+'.jpg'
53         cv2.imwrite(mask_file, mask, [int(cv2.IMWRITE_JPEG_QUALITY), 70])
54     """
55     cv2.imshow('origin',img)
56     img_segmentation,maskk=segment(img)
57     img_segmentation=cv2.resize(img_segmentation,(160,160))
58     cv2.imwrite('F:/seven/prDesign/seg32.jpg', img_segmentation, [int(cv2.IMWRITE_JPEG_QUALITY), 100])
59     cv2.imshow('segmented',img_segmentation)
60     #cv2.imshow('mask',mask)
61     cv2.waitKey(0)

```

sfta.py

```

1 """
2 SFTA特征提取模块
3 """

```

```

4 import numpy as np
5 import cv2
6 import math
7
8 def otsu(counts):
9     """
10     通过OSTU算法获得阈值
11     :param counts: 颜色直方图
12     :return: 最佳阈值
13     """
14     p=counts/sum(counts)#归一化
15     Tbin=0#定义起始值
16     ToBin=len(p)#定义结束值
17     Object=0
18     x=np.arange(ToBin)
19     meanT=0
20     #求直方图中颜色的加权平均
21     for i in range(ToBin):
22         meanT=meanT+p[i]*x[i]
23     #寻找最佳阈值
24     for t in range(ToBin):
25         mean0=0
26         mean1=0
27         #背景的颜色均值
28         for i in range(t):
29             mean0=mean0+p[i]*x[i]
30         #目标的颜色均值
31         for i in range(t,ToBin):
32             mean1=mean1+p[i]*x[i]
33         w0=np.sum(p[:t])
34         w1=np.sum(p[t:])
35         Nobject=w0*(mean0-meanT)**2+w1*(mean1-meanT)**2#最大化类间差异
36         if Nobject>Object:
37             Object=Nobject
38             Tbin=t
39     return Tbin
40
41 def recur_otsu(T,counts,lowerBin,upperBin,tLower,tUpper):
42     """
43     利用递归的方式求解多等级阈值分割的阈值集合
44     :param T: 阈值集合
45     :param counts: 颜色直方图
46     :param lowerBin: 阈值搜寻下界
47     :param upperBin: 阈值搜寻上界
48     :param tLower: 阈值的位置下界
49     :param tUpper: 阈值的位置上界
50     """
51     if tUpper<tLower or lowerBin>=upperBin:
52         return
53     else:
54         thres=otsu(counts[lowerBin:upperBin])+lowerBin

```

```

55     thPos=int((tLower+tUpper)/2)#确定所求得的阈值对应在阈值集合中的位置
56     T[thPos]=thres
57     recur_ostu(T,counts,lowerBin,thres,tLower,thPos-1)
58     recur_ostu(T,counts,thres+1,upperBin,thPos+1,tUpper)
59
60 def multiOTSU(img,nt):
61     """
62     多等级阈值求解
63     :param img: 输入图像
64     :param nt: 阈值的个数
65     :return: 阈值集合
66     """
67     img.astype(np.uint8)
68     counts=cv2.calcHist([img],[0],None,[256],[0,256])
69     T=np.zeros(nt)
70     recur_ostu(T,counts,0,255,0,nt-1)
71     return T
72
73 def hausDim(img):
74     """
75     计算分形维度
76     :param img: 输入图像
77     :return: 分形维度
78     """
79     m,n=img.shape
80     maxDim=np.max([m,n])
81     newDimSize=int(math.pow(2,int(math.log2(maxDim))+1))
82     rowPad=newDimSize-m
83     colPad=newDimSize-n
84     I=np.lib.pad(img,((rowPad,0),(colPad,0)),'symmetric')#补全图像至长、宽均为2的整数次幂
85
86     newm,newn=I.shape
87
88     boxCounts=np.zeros(int(math.log2(maxDim))+1)
89     resolutions=np.zeros(int(math.log2(maxDim))+1)
90
91     m,n=I.shape
92     boxSize=m#初始box的大小
93     boxesPerDim=1#初始box的个数
94     idx=0
95     #不同的box大小统计目标点的个数
96     while boxSize>1:
97         boxCount=0
98         minBox=np.arange(0,(m-boxSize)+1,boxSize).astype(np.int)
99         maxBox=np.arange(boxSize-1,m,boxSize).astype(np.int)
100
101        for boxRow in range(boxesPerDim):
102            for boxCol in range(boxesPerDim):
103                objFound=False
104                for row in range(minBox[boxRow],maxBox[boxRow]):
105                    for col in range(minBox[boxCol],maxBox[boxCol]):
```

```

106     if I[row,col]:
107         boxCount=boxCount+1
108         objFound=True
109         break
110     if objFound:
111         break
112
113     boxCounts[idx]=boxCount+1e-5#防止为0, 对对数求解造成影响
114     resolutions[idx]=float(1/boxSize)+1e-5
115     idx=idx+1
116
117     boxesPerDim=boxesPerDim*2
118     boxSize=int(boxSize)/2
119
120     D=np.polyfit(np.log(resolutions),np.log(boxCounts),1)#线性拟合, 返回斜率
121     return D[0]
122
123 def sfta(img,nt):
124     """
125     SFTA特征提取函数
126     :param img: 输入图像
127     :param nt: 阈值个数
128     :return: SFTA特征向量
129     """
130
131     grayImg=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
132     grayImg.astype(np.uint8)
133     T=multiOTSU(grayImg,nt)
134     dSize=(len(T)*6)-3
135     D=np.zeros(dSize)
136     pos=0
137     #计算单阈值情形
138     for t in range(len(T)):
139         thres=T[t]
140
141         _,Ib=cv2.threshold(grayImg,thres,255,cv2.THRESH_BINARY)
142
143         edge_output=cv2.Canny(Ib,threshold1=50,threshold2=150)
144         Ib=cv2.bitwise_and(Ib,Ib,mask=edge_output)
145
146         IIb=cv2.bitwise_and(grayImg,grayImg,mask=edge_output)
147         Val=IIb.ravel()[np.flatnonzero(IIb)]
148
149         D[pos]=hausDim(Ib)#分形维度
150         pos=pos+1
151
152         D[pos]=np.mean(np.mean(Val)) if len(Val)!=0 else 0#平均灰度
153         pos=pos+1
154
155         D[pos]=len(Val)#目标个数
156         pos=pos+1
157         #两阈值分割的情形

```

```

157     for t in range(len(T)-1):
158         lowerThres=T[t]
159         upperThres=T[t+1]
160
161         #print(lowerThres,upperThres)
162
163         Ibu=np.array(grayImg<upperThres,dtype=np.uint8)
164         Ibl=np.array(grayImg>lowerThres,dtype=np.uint8)
165
166         Ib=Ibu*Ibl*255
167         Ib.astype(np.uint8)
168
169         edge_output = cv2.Canny(Ib, threshold1=50, threshold2=150)
170         Ib = cv2.bitwise_and(Ib, Ib, mask=edge_output)
171
172         IIb = cv2.bitwise_and(grayImg, grayImg, mask=edge_output)
173         Val = IIb.ravel()[np.flatnonzero(IIb)]
174
175         D[pos] = hausDim(Ib)
176         pos = pos + 1
177
178         D[pos] = np.mean(np.mean(Val)) if len(Val)!=0 else 0
179         pos = pos + 1
180
181         D[pos] = len(Val)
182         pos = pos + 1
183
184     return D
185
186 #测试函数
187 if __name__=="__main__":
188     img = cv2.imread("F:/seven/prDesign/fdata/buttercup/image_0021.jpg")
189     grayImg=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
190     T=multiOTSU(grayImg,10)
191     D=sfta(img,10)
192     print(D)

```

sSIFT.py

```

1     """
2 Dense-SIFT模块
3     """
4     import numpy as np
5     import cv2
6
7     #定义DenseSIFT特征提取算子
8     class DenseSIFT():
9         def __init__(self):
10             self.sift=cv2.xfeatures2d.SIFT_create()
11
12         def detectAndCompute(self,image,step_size=8,window_size=(16,16)):
13             if window_size is None:

```

```

14     winH,winW=image.shape[:2]
15     window_size=(winW//4,winH//4)
16
17     descriptors=np.array([],dtype=np.float32).reshape(0,128)
18     for crop in self._crop_image(image,step_size,window_size):
19         tmp_descriptor=self._detectAndCompute(crop)[1]
20         if tmp_descriptor is None:
21             continue
22         descriptors=np.vstack([descriptors,tmp_descriptor])
23     return descriptors
24
25 def _detect(self,image):
26     return self.sift.detect(image)
27
28 def _compute(self,image,kps,eps=1e-7):
29     kps,descs=self.sift.compute(image,kps)
30
31     if len(kps)==0:
32         return [],None
33
34     descs/==(descs.sum(axis=1,keepdims=True)+eps)
35     descs=np.sqrt(descs)
36     return kps,descs
37
38 def _detectAndCompute(self,image):
39     kps=self._detect(image)
40     return self._compute(image,kps)
41
42 def _sliding_window(self,image,step_size,window_size):
43     for y in range(0,image.shape[0],step_size):
44         for x in range(0,image.shape[1],step_size):
45             yield (x,y,image[y:y+window_size[1],x:x+window_size[0]])
46
47 def _crop_image(self,image,step_size=8,window_size=(16,16)):
48     crops=[]
49     winH,winW=window_size
50     for (x,y,window) in self._sliding_window(image,step_size=step_size,window_size=window_size):
51         if window.shape[0]!=winH or window.shape[1]!=winW:
52             continue
53         crops.append(image[y:y+winH,x:x+winW])
54     return np.array(crops)
55
56 #测试函数
57 if __name__=="__main__":
58     img=cv2.imread("F:/seven/prDesign/fdata/iris/image_0401.jpg")
59     imgHSV=cv2.cvtColor(img,cv2.COLOR_BGR2HSV)
60     #imgGray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
61     Densift=DenseSIFT()
62     des1=Densift.detectAndCompute(imgHSV[:, :, 0])
63     des2=Densift.detectAndCompute(imgHSV[:, :, 1])
64     des3=Densift.detectAndCompute(imgHSV[:, :, 2])

```

```

65     print(des1.shape)
66     print(des2.shape)
67     print(des3.shape)

```

C.4 文件夹：fRemote

main.py

```

1     """
2 神经网络模块——方案五、六和七
3     """
4 import torch
5 import torch.utils.data as Data
6 import torch.nn as nn
7 from torchvision import datasets, models, transforms
8 from math import pi
9 from math import cos
10 import torch.backends.cudnn as cudnn
11 import torch.optim as optim
12 from torchvision import transforms, utils
13 import torchvision
14 from torch.autograd import Variable
15 from tensorboardX import SummaryWriter
16 import numpy as np
17 import glob
18 import load
19 from sklearn.model_selection import train_test_split
20 import scipy.io as scio
21 import cv2
22 import snapensemble
23 #import FCN
24 import os
25
26 filenames=[]
27 filenames=sorted(glob.glob(r'oxford102/segmim/*.jpg'))
28 testFile='test6.jpg'
29 print(filenames)
30 """
31 filenames.append(glob.glob(r'fdata/buttercup/*.jpg'))
32 filenames.append(glob.glob(r'fdata/daisy/*.jpg'))
33 filenames.append(glob.glob(r'fdata/iris/*.jpg'))
34 filenames.append(glob.glob(r'fdata/lilyvalley/*.jpg'))
35 filenames.append(glob.glob(r'fdata/sunflower/*.jpg'))
36 filenames.append(glob.glob(r'fdata/windflower/*.jpg'))
37 """
38 #data,label=load.load(filenames)
39 #label=tf.one_hot(label,6)
40 #print(data.shape)
41 #"""

```

```

42 data=[]
43 for filename in filenames:
44     print(filename)
45     tempData=cv2.imread(filename,cv2.IMREAD_COLOR)
46     tempData=cv2.cvtColor(tempData,cv2.COLOR_BGR2HSV)
47     tempData=cv2.resize(tempData,(299,299))
48     tempData=np.swapaxes(tempData,0,2)
49     tempData=tempData/255.
50     data.append(tempData)
51 data=np.array(data,dtype=float)
52 #"""
53 tData=[]
54 testData=cv2.imread(testFile,cv2.IMREAD_COLOR)
55 testData=load.segmentation(testData)
56 testData=cv2.cvtColor(testData,cv2.COLOR_BGR2HSV)
57 testData=cv2.resize(testData,(299,299))
58 testData=np.swapaxes(testData,0,2)
59 testData=testData/255.
60 tData.append(testData)
61 tData=np.array(tData)
62
63 y=scio.loadmat('oxford102/imagelabels.mat')
64 label=np.array(y['labels']).reshape(-1)-1
65 print(y)
66 #"""
67 writer=SummaryWriter()
68
69 BATCH_SIZE=40
70 CLASS_NUM=6
71 EPOCH=50
72 #LR=0.0001 #for 6 classes
73 LR=0.000005
74
75 X_train,X_test,y_train,y_test=train_test_split(data,label,test_size=0.2,random_state=42,stratify=label)
76 X_train=torch.from_numpy(X_train).double()
77 X_test=torch.from_numpy(X_test).double()
78 y_train=torch.from_numpy(y_train).int()
79 y_test=torch.from_numpy(y_test).int()
80
81 TESTX=torch.from_numpy(tData).double()
82
83 train_dataset=Data.TensorDataset(X_train,y_train)
84 train_loader=Data.DataLoader(dataset=train_dataset,batch_size=BATCH_SIZE,shuffle=True)
85 test_dataset=Data.TensorDataset(X_test,y_test)
86 test_loader=Data.DataLoader(dataset=test_dataset,batch_size=BATCH_SIZE,shuffle=False)
87
88 #自定义卷积神经网络
89 class CNN(nn.Module):
90     def __init__(self):
91         super(CNN,self).__init__()
92         self.conv1=nn.Sequential(

```

```

93     nn.Conv2d(in_channels=3,out_channels=64,kernel_size=5,stride=1,padding=2),
94     nn.ReLU(),
95     nn.MaxPool2d(kernel_size=3)
96   )
97   self.conv2=nn.Sequential(
98     nn.Conv2d(in_channels=64,out_channels=128,kernel_size=5,stride=1,padding=2),
99     nn.ReLU(),
100    nn.MaxPool2d(kernel_size=3)
101  )
102  self.conv3=nn.Sequential(
103    nn.Conv2d(in_channels=128,out_channels=256,kernel_size=3,stride=1,padding=1),
104    nn.ReLU(),
105    nn.MaxPool2d(kernel_size=3)
106  )
107  self.conv4=nn.Sequential(
108    nn.Conv2d(in_channels=256,out_channels=512,kernel_size=3,stride=1,padding=1),
109    nn.ReLU(),
110    nn.Conv2d(in_channels=512,out_channels=512,kernel_size=3,stride=1,padding=1),
111    nn.ReLU(),
112    nn.MaxPool2d(kernel_size=3)
113  )
114  self.fc1=nn.Sequential(
115    nn.Linear(512,1024),
116    nn.ReLU()
117  )
118  self.fc2=nn.Sequential(
119    nn.Linear(1024,512),
120    nn.ReLU()
121  )
122  self.out=nn.Linear(512,102)

123
124  # 迭代循环初始化参数
125  for m in self.modules():
126    if isinstance(m, nn.Linear):
127      nn.init.xavier_normal_(m.weight)
128      nn.init.constant_(m.bias,1.)
129    # 也可以判断是否为conv2d, 使用相应的初始化方式
130    elif isinstance(m, nn.Conv2d):
131      nn.init.kaiming_normal_(m.weight, mode='fan_out', nonlinearity='relu')

132
133  def forward(self,x):
134    x=self.conv1(x)
135    x=self.conv2(x)
136    x=self.conv3(x)
137    x=self.conv4(x)
138    x=x.view(x.size(0),-1)
139    x=self.fc1(x)
140    #x=torch.nn.functional.dropout(x,p=0.5)
141    x=self.fc2(x)
142    #x=torch.nn.functional.dropout(x,p=0.5)
143    output=self.out(x)

```

```

144     output=torch.nn.functional.relu(output)
145     return output
146
147 #使用自定义卷积神经网络模型
148 #cnn=CNN().to("cuda")
149 #simInput=torch.randn(40,3,100,100).to('cuda')
150
151 #optimizer=torch.optim.Adam(cnn.parameters(),lr=LR)
152 #optimizer=torch.optim.SGD(cnn.parameters(),lr=LR,momentum=0.9)
153 #loss_function=nn.CrossEntropyLoss()
154 totalStep=0
155 #fcn_model=torch.load('fcn_model_1000').cuda()
156 #使用预训练模型
157 model_conv=torchvision.models.inception_v3(pretrained=True)
158 #model_conv=torchvision.models.resnet101(pretrained=True)
159 #model_conv=torchvision.models.vgg16(pretrained=True)
160 #固定预训练模型的参数（除最后一层），不参与训练
161 for param in model_conv.parameters():
162     param.requires_grad=False
163
164 print('have_executed')
165 #重新定义最后一层网络结构，并用于训练
166 num_ftrs=model_conv.fc.in_features
167 model_conv.fc=nn.Linear(num_ftrs,102)
168 #model_conv.classifier._modules['6']=nn.Linear(4096,102)
169 model_conv.aux_logits=False
170
171 #params = [{ 'params': md.parameters()} for md in model_conv.children()]
172 #if md in [model_conv.classifier]
173
174 model_conv=model_conv.to('cuda')
175 simInput=torch.randn(20,3,299,299).to('cuda')
176
177 loss_function=nn.CrossEntropyLoss()
178 #optimizer_conv=optim.SGD(model_conv.parameters(),lr=0.01,momentum=0.9)
179 optimizer_conv=optim.SGD(model_conv.parameters(),lr=0.01,momentum=0.9)
180
181 with SummaryWriter(comment='Net1') as w:
182     w.add_graph(model_conv,[simInput,])
183
184 """
185 #常规训练过程训练
186 for epoch in range(EPOCH):
187     for step,(x,y) in enumerate(train_loader):
188         b_x=torch.tensor(x,dtype=torch.float32).to('cuda')
189         b_y=Variable(y).type(torch.LongTensor).to('cuda')
190         optimizer_conv.zero_grad()
191
192         output=model_conv(b_x)
193         #print(output,b_y)
194         loss=loss_function(output,b_y)

```

```

195 loss.backward()
196 optimizer_conv.step()
197
198 if step%100==0:
199     #accuracy=0
200     sum=0
201     for tdata in test_loader:
202         x,y=tdata
203         t_x=torch.tensor(x,dtype=torch.float32).to('cuda')
204         t_y=torch.tensor(y,dtype=torch.int).to('cuda')
205         test_output=model_conv(t_x)
206         pred_y=torch.max(test_output,1)[1].data.squeeze()
207         for k in range(t_y.size(0)):
208             if pred_y[k]==t_y[k]:
209                 sum=sum+1
210         accuracy=sum/y_test.size(0)
211         print('Epoch:',epoch,'|Step:',step,'|train.loss: %.4f' % loss.item(),'|test.accuracy: %.4f' % accuracy)
212         writer.add_scalar('Test',accuracy,totalStep)
213         writer.add_scalar('Train',loss,totalStep)
214         totalStep=totalStep+1
215
216 #预测输出结果示
217 TestT=torch.tensor(TESTX,dtype=torch.float32).to('cuda')
218 test_output=model_conv(TestT)
219 pred_y=torch.max(test_output,1)[1].data.squeeze()
220 print(pred_y,'prediction_number')
221 print(y_test[:10],'real_number')
222
223 #snapshot ensemble训练过程
224 #models=snapensemble.train(cnn,300,6,0.01,train_loader,writer)
225 #snapensemble.test(CNN,models,5,test_loader)
226
227 writer.close()

```

load.py

```

1 import cv2
2 import numpy as np
3 import glob
4
5 M=np.array([
6     [0.412453,0.357580,0.180423],
7     [0.212671,0.715160,0.072169],
8     [0.019334,0.119193,0.950227]
9 ])
10
11 def f(im_channel):
12     return np.power(im_channel,1/3) if im_channel>0.008856 else 7.787*im_channel+0.137931
13
14 def __rgb2xyz__(pixel):
15     b,g,r=pixel[2],pixel[1],pixel[0]
16     rgb=np.array([r,g,b])

```

```

17     XYZ=np.dot(M,rgb.T)
18     XYZ=XYZ/255.0
19     return (XYZ[0]/0.95047,XYZ[1]/1.0,XYZ[2]/1.0883)
20
21 def __xyz2lab__(xyz):
22     F_XYZ=[f(x) for x in xyz]
23     L=116*F_XYZ[1]-16 if xyz[1]>0.008856 else 903.3*xyz[1]
24     a=500*(F_XYZ[0]-F_XYZ[1])
25     b=200*(F_XYZ[0]-F_XYZ[2])
26     return L,a,b
27
28 def RGB2Lab(pixel):
29     xyz=__rgb2xyz__(pixel)
30     L,a,b=__xyz2lab__(xyz)
31     return L,a,b
32
33 def HisSegmentation(img):
34     uImg=cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
35     lImg=cv2.cvtColor(uImg,cv2.COLOR_RGB2Lab)
36     Hiscenter=np.array([11,33,55,77,99,121,143,165,187,209,231,249])
37     valueThre=22
38     Hisc=np.zeros((12*12*12,3))
39     LHisc=np.zeros((12*12*12,3))
40     for i in range(12):
41         for j in range(12):
42             for k in range(12):
43                 Hisc[i*12*12+j*12+k,0]=Hiscenter[i]
44                 Hisc[i*12*12+j*12+k,1]=Hiscenter[j]
45                 Hisc[i*12*12+j*12+k,2]=Hiscenter[k]
46                 LHisc[i*12*12+j*12+k,0],LHisc[i*12*12+j*12+k,1],LHisc[i*12*12+j*12+k,2]=RGB2Lab(Hisc[i
47                     *12*12+j*12+k,:])
48     LHisc=LHisc.reshape(12*12*12,3)
49     Histo=np.zeros(12*12*12)
50     m,n,_=uImg.shape
51     for i in range(m):
52         for j in range(n):
53             index=int(uImg[i,j,0]/22)*12*12+int(uImg[i,j,1]/22)*12+int(uImg[i,j,2]/22)
54             Histo[index]=Histo[index]+1
55             #print(Histo)
56             sortIndex=np.argsort(-Histo, axis=0)
57             Histo=-np.sort(-Histo, axis=0)
58             print(Histo)
59             endIndex=0
60             sumPixels=0
61             thres=m*n*0.95
62             #print(len(index))
63             for i in range(12*12*12):
64                 sumPixels=sumPixels+Histo[i]
65                 if sumPixels>=thres:
66                     endIndex=i
67                     break

```

```

67     index=sortIndex[:endIndex]
68     print(len(index))
69     NLhisc=np.zeros((len(index),3))
70     for i in range(len(index)):
71         NLhisc[i]=LHisc[index[i]]
72     SLhisc=np.zeros(len(index))
73     for i in range(len(index)):
74         sum=0.
75         c_l=NLhisc[i,:]
76         for j in range(len(index)):
77             disc=np.linalg.norm(c_l-NLhisc[j,:])
78             sum+=float(Histo[j])/(m*n)*disc
79         SLhisc[i]=sum
80     mm=int(len(index)/4)
81     result=np.zeros((m,n))
82     for i in range(m):
83         for j in range(n):
84             tempData=uImg[i,j,:]
85             inD=int(tempData[0]/22)*12*12+int(tempData[1]/22)*12+int(tempData[2]/22)
86             if inD in index:
87                 tinD=np.where(index==inD)
88                 result[i][j]=SLhisc[tinD]
89             else:
90                 tempLData=lImg[i,j,:]
91                 dist=np.zeros(len(index))
92                 for k in range(len(index)):
93                     dist[k]=np.linalg.norm(tempLData-NLhisc[k,:])
94                 LIndex=np.argsort(dist, axis=0)
95                 LIndex=LIndex[:mm]
96                 mdist=np.sort(dist, axis=0)[:mm]
97                 t=np.sum(mdist)
98                 sum=0
99                 for k in LIndex:
100                     sum+=t-dist[k]*SLhisc[k]
101                 result[i][j]=sum/((mm-1)*t)
102     max=np.max(np.max(result, axis=0), axis=0)
103     min=np.min(np.min(result, axis=0), axis=0)
104     print(max,min)
105     print(result[1][2])
106     result=255*(result-min)/(max-min)
107     print(result[1][2])
108     return result
109
110 def segmentation(img):
111     imgHSV=cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
112     _, _, imgV = cv2.split(imgHSV)
113     m,n,c=imgHSV.shape
114     imgV=np.array(imgV)/255
115     imgVF=imgV.flatten()
116     meanV=np.mean(imgVF, axis=0)
117     varV=np.std(imgVF, axis=0)

```

```

118     Threshold=meanV+varV
119     fmask=np.zeros((m,n),dtype=np.uint8)
120     mask=np.array(imgV>Threshold,dtype=np.uint8)
121     img_segmentation=cv2.bitwise_and(img,img,mask=mask)
122     return img_segmentation
123
124 def load(filenames):
125     data=[]
126     label=[]
127     k=0
128     for filename in filenames:
129         for i in range(len(filename)):
130             tempData=cv2.imread(filename[i],cv2.IMREAD_COLOR)
131             tempData=segmentation(tempData)
132             tempData=cv2.cvtColor(tempData,cv2.COLOR_BGR2HSV)
133             tempData=cv2.resize(tempData,(224,224))
134             tempData=np.swapaxes(tempData,1,2)
135             tempData=np.swapaxes(tempData,0,1)
136             tempData=tempData/255.
137             data.append(tempData)
138             label.append(k)
139             k=k+1
140     return np.array(data,dtype=float),np.array(label)
141
142 if __name__=="__main__":
143     """
144     img=cv2.imread('F:/seven/prDesign/fdata/buttercup/image_0009.jpg')
145     imgS=HisSegmentation(img)
146     imgS=imgS.astype(np.uint8)
147     #ret, imgS = cv2.threshold(imgS, 100, 255, cv2.THRESH_BINARY)
148     print(imgS)
149     cv2.imshow('seg',imgS)
150     cv2.waitKey(0)
151     """
152     # img = cv2.imread("F:/seven/prDesign/oxford102/segmim/segmim_00036.jpg")
153     filenames = sorted(glob.glob(r'oxford102/segmim/*.jpg'))
154     root = 'oxford102/mask/'
155     for m in range(8189):
156         img = cv2.imread(filenames[m])
157         print(filenames[m])
158         mask = np.zeros((img.shape[0], img.shape[1]), dtype=np.uint8)
159         for i in range(img.shape[0]):
160             for j in range(img.shape[1]):
161                 if img[i, j, 0] == 254 and img[i, j, 1] == 0 and img[i, j, 2] == 0:
162                     mask[i][j] = 0
163                 else:
164                     mask[i][j] = 255
165         mask_file = root + str(m + 1) + '.jpg'
166         cv2.imwrite(mask_file, mask, [int(cv2.IMWRITE_JPEG_QUALITY), 70])

```

snapensembly.py

```

1     """
2     snapshot ensembling模块
3     """
4
5     import torch
6     import torch.nn as nn
7     import torch.nn.functional as F
8     from torch.autograd import Variable
9     import torch.optim as optim
10    from torch.utils.data import Dataset,DataLoader
11    from math import pi
12    from math import cos
13    import multiprocessing as mp
14    import numpy as np
15
16    cuda=torch.cuda.is_available()
17
18    #计算学习率
19    def snapEnsemble_lr(initial_lr,iteration,epoch_per_cycle):
20        return initial_lr*(cos(pi*iteration/epoch_per_cycle)+1)/2
21
22    #训练过程
23    def train(model,epochs,cycles,initial_lr,train_loader,writer):
24        snapshots=[]
25        _lr_list,_loss_list=[], []
26        count=0
27        epochs_per_cycle=epochs//cycles
28        optimizer=optim.SGD(model.parameters(),lr=initial_lr)
29        totalStep=0
30        loss_func = torch.nn.CrossEntropyLoss()
31
32        for i in range(cycles):
33            for j in range(epochs_per_cycle):
34                _epoch_loss=0
35                lr=snapEnsemble_lr(initial_lr,j,epochs_per_cycle)
36
37                optimizer.state_dict()["param_groups"][0]["lr"]=lr
38                writer.add_scalar('Learning',lr,i*epochs_per_cycle+j)
39                for batch_idx,(data,target) in enumerate(train_loader):
40                    data=torch.tensor(data,dtype=torch.float32).to('cuda')
41                    target=Variable(target).type(torch.LongTensor).to('cuda')
42
43                    optimizer.zero_grad()
44                    output=model(data)
45                    loss=loss_func(output,target)
46                    _epoch_loss+=loss.item()/len(train_loader)
47                    loss.backward()
48                    optimizer.step()
49                    writer.add_scalar('Train',loss,totalStep)
50                    totalStep+=1
51                    print(i,j,batch_idx,loss.item())

```

```

52     _lr_list.append(lr)
53     _loss_list.append(_epoch_loss)
54     count+=1
55     snapshots.append(model.state_dict())
56
57
58 #测试函数
59 def test(Model,weights,use_model_num,test_loader):
60     index=len(weights)-use_model_num
61     weights=weights[index:]
62     model_list=[Model() for _ in weights]
63
64     for model,weight in zip(model_list,weights):
65         model.load_state_dict(weight)
66         model.eval()
67         model.cuda()
68
69     test_loss=0
70     correct=0
71     for data,target in test_loader:
72         data=torch.tensor(data,dtype=torch.float32).to('cuda')
73         target=torch.tensor(target,dtype=torch.int).to('cuda')
74         targetL=Variable(target).type(torch.LongTensor).to('cuda')
75         output_list=[model(data).unsqueeze(0) for model in model_list]
76         output=torch.mean(torch.cat(output_list),0).squeeze()
77         test_loss+=F.nll_loss(output,targetL).item()
78         pred=output.data.max(1)[1]
79         correct+=pred.eq(target.data).cpu().sum()
80
81     test_loss/=len(test_loader)
82     print('\nTest.set.:Average.loss: {:.4f}, Accuracy: {} / {} ({:.0f}%)'.format(
83         test_loss,correct,len(test_loader.dataset),100*correct/len(test_loader.dataset)
84     ))

```

FCN.py

```

1 import cv2
2 import glob
3 import torch
4 import torch.nn as nn
5 import torch.optim as optim
6 from torch.utils.data import Dataset,DataLoader
7 from torchvision import models
8 from torchvision.models.vgg import VGG
9 from torchvision import transforms
10 import matplotlib.pyplot as plt
11 import matplotlib
12 import pdb
13 import numpy as np
14 import os
15 os.environ["CUDA_VISIBLE_DEVICES"] = "0"
16 #matplotlib.use('Qt5Agg')

```

```

17
18     def onehot(data,n):
19         buf=np.zeros(data.shape+(n,))
20         nmsk=np.arange(data.size)*n+data.ravel()
21         buf.ravel()[nmsk-1]=1
22         return buf
23
24     class BagDataset(Dataset):
25         def __init__(self,filenames,msk_filenames):
26             self.filenames=filenames
27             self.msk_filenames=msk_filenames
28         def __len__(self):
29             return len(self.filenames)
30         def __getitem__(self,idx):
31             img_name=self.filenames[idx]
32             msk_img_name=self.msk_filenames[idx]
33             img=cv2.imread(img_name)
34             img=cv2.resize(img,(160,160))
35             img_msk=cv2.imread(msk_img_name)
36             #img_msk = cv2.cvtColor(img_msk, cv2.COLOR_BGR2GRAY)
37             img_msk=cv2.resize(img_msk,(160,160))
38             img_msk=img_msk/255
39             img_msk=img_msk.astype('uint8')
40             img_msk=onehot(img_msk,2)
41             img=img.swapaxes(0,2).swapaxes(1,2)
42             img_msk=img_msk.swapaxes(0,2).swapaxes(1,2)
43             img=torch.FloatTensor(img)
44             img_msk=torch.FloatTensor(img_msk)
45             item={'ori':img,'msk':img_msk}
46             return item
47
48
49     class FCN(nn.Module):
50         def __init__(self,pretrained_net,n_class):
51             super().__init__()
52             self.n_class=n_class
53             self.pretrained_net=pretrained_net
54             self.relu=nn.ReLU(inplace=True)
55             self.deconv1=nn.ConvTranspose2d(512,512,kernel_size=3,stride=2,padding=1,dilation=1,
56                                         output_padding=1)
57             self.bn1=nn.BatchNorm2d(512)
58             self.deconv2=nn.ConvTranspose2d(512,256,kernel_size=3,stride=2,padding=1,dilation=1,
59                                         output_padding=1)
60             self.bn2=nn.BatchNorm2d(256)
61             self.deconv3=nn.ConvTranspose2d(256,128,kernel_size=3,stride=2,padding=1,dilation=1,
62                                         output_padding=1)
63             self.bn3=nn.BatchNorm2d(128)
64             self.deconv4=nn.ConvTranspose2d(128,64,kernel_size=3,stride=2,padding=1,dilation=1,
65                                         output_padding=1)
66             self.bn4=nn.BatchNorm2d(64)
67             self.deconv5=nn.ConvTranspose2d(64,32,kernel_size=3,stride=2,padding=1,dilation=1,output_padding

```

```

    =1)
64   self.bn5=nn.BatchNorm2d(32)
65   self.classifier=nn.Conv2d(32,n_class,kernel_size=1)
66
67   def forward(self,x):
68       output=self.pretrained_net(x)
69       x5=output['x5']
70       x4=output['x4']
71       x3=output['x3']
72
73       score=self.relu(self.deconv1(x5))
74       score=self.bn1(score+x4)
75       score=self.relu(self.deconv2(score))
76       score=self.bn2(score+x3)
77       score=self.bn3(self.relu(self.deconv3(score)))
78       score=self.bn4(self.relu(self.deconv4(score)))
79       score=self.bn5(self.relu(self.deconv5(score)))
80       score=self.classifier(score)
81
82   return score
83
84   ranges={
85       'vgg11':((0,3),(3,6),(6,11),(11,16),(16,21)),
86       'vgg13':((0,5),(5,10),(10,15),(15,20),(20,25)),
87       'vgg16':((0,5),(5,10),(10,17),(17,24),(24,31)),
88       'vgg19':((0,5),(5,10),(10,19),(19,28),(28,37))
89   }
90
91   cfg={
92       'vgg11':[64,'M',128,'M',256,256,'M',512,512,'M',512,512,'M'],
93       'vgg13':[64,64,'M',128,128,'M',256,256,'M',512,512,'M',512,512,'M'],
94       'vgg16':[64,64,'M',128,128,'M',256,256,256,'M',512,512,512,'M',512,512,512,'M'],
95       'vgg19':[64,64,'M',128,128,'M',256,256,256,256,'M',512,512,512,512,'M',512,512,512,512,'M']
96   }
97
98   def make_layers(cfg,batch_norm=False):
99       layers=[]
100      in_channels=3
101      for v in cfg:
102          if v=='M':
103              layers+=[nn.MaxPool2d(kernel_size=2,stride=2)]
104          else:
105              conv2d=nn.Conv2d(in_channels,v,kernel_size=3,padding=1)
106              if batch_norm:
107                  layers+=[conv2d,nn.BatchNorm2d(v),nn.ReLU(inplace=True)]
108              else:
109                  layers+=[conv2d,nn.ReLU(inplace=True)]
110              in_channels=v
111
112      return nn.Sequential(*layers)
113
class VGGNet(VGG):

```

```

114     def __init__(self, pretrained=True, model='vgg16', requires_grad=False, remove_fc=True):
115         super().__init__(make_layers(cfg[model]))
116         self.ranges=ranges[model]
117
118     if pretrained:
119         exec("self.load_state_dict(models.%s(pretrained=True).state_dict()'%smodel")
120
121     if not requires_grad:
122         for param in super().parameters():
123             param.requires_grad=False
124
125     if remove_fc:
126         del self.classifier
127
128     def forward(self, x):
129         output={}
130         for idx in range(len(self.ranges)):
131             for layer in range(self.ranges[idx][0], self.ranges[idx][1]):
132                 x=self.features[layer](x)
133                 output["x%d"% (idx+1)]=x
134
135         return output
136
137     if __name__=="__main__":
138         #filenames = sorted(glob.glob(r'oxford102/jpg/*.jpg'))
139         filenames=sorted(glob.glob(r'F:/seven/prDesign/test*.jpg'))
140         #msk_filenames=sorted(glob.glob(r'oxford102/mask/*.jpg'))
141         BATCH_SIZE=40
142         EPOCHS=1000
143
144         #bag=BagDataset(filenames, msk_filenames)
145         #dataLoader=DataLoader(bag, batch_size=BATCH_SIZE, shuffle=True, num_workers=4)
146
147         #vgg_model=VGGNet(requires_grad=True)
148         #fcn_model=FCN(pretrained_net=vgg_model, n_class=2)
149         #fcn_model=fcn_model.cuda()
150         fcn_model=torch.load('fcn_model_1000').cuda()
151         #loss_fn=nn.BCELoss().cuda()
152         #optimizer=optim.SGD(fcn_model.parameters(), lr=0.01, momentum=0.9)
153
154         saving_index=0
155         """
156         for epoch in range(EPOCHS):
157             saving_index+=1
158             index=0
159             epoch_loss=0
160             for item in dataLoader:
161                 index+=1
162                 x=item['ori']
163                 y=item['msk']

```

```

165     x=torch.autograd.Variable(x).to('cuda')
166     y=torch.autograd.Variable(y).to('cuda')
167
168     optimizer.zero_grad()
169     output=fcn_model(x)
170     output=nn.functional.sigmoid(output)
171     loss=loss_fn(output,y)
172
173     loss.backward()
174     iter_loss=loss.item()
175     epoch_loss+=iter_loss
176     optimizer.step()
177
178     #output_np=output.cpu().data.numpy().copy()
179     #output_np=np.argmin(output_np, axis=1)
180     #y_np=y.cpu().data.numpy().copy()
181     #y_np=np.argmin(y_np, axis=1)
182
183     #plt.subplot(1,2,1)
184     #plt.imshow(np.squeeze(y_np[0,:,:]),'gray')
185     #plt.subplot(1,2,2)
186     #plt.imshow(np.squeeze(output_np[0,:,:]),'gray')
187     #plt.savefig(str(index+1)+'.png')
188     #plt.show()
189     #plt.pause(0.5)
190
191     print('epoch loss = %f %(epoch_loss/len(dataLoader)))'
192
193     if np.mod(saving_index,5)==0:
194         torch.save(fcn_model,'checkpoint/fcn_model_{ }'.format(epoch+1))
195         print('saving checkpoints/fcn_model_{ }.pt'.format(epoch+1))
196     """
197     for i in range(len(filenames)):
198         img=cv2.imread(filenames[i])
199         img=cv2.resize(img,(160,160)).astype(np.uint8)
200         img_e=img.reshape(1,3,160,160)
201         img_ten=torch.tensor(img_e,dtype=torch.float32).to('cuda')
202         img_cu = torch.autograd.Variable(img_ten).to('cuda')
203         img_mask=fcn_model(img_cu)
204         print(i)
205         img_mask_np=img_mask.cpu().data.numpy().copy()
206         img_mask_np = np.argmin(img_mask_np, axis=1)
207         img_mask_np=(img_mask_np+1/255).astype(np.uint8).reshape(160,160)
208         #print(img_mask_np)
209         print(img.shape,img_mask_np.shape)
210         img_seg=cv2.bitwise_and(img,img,mask=img_mask_np)
211         cv2.imwrite('seg'+str(i+1)+'.jpg',img_seg,[int(cv2.IMWRITE_JPEG_QUALITY), 100])
212         #cv2.imshow('seg',img_seg)
213         #cv2.waitKey(0)

```