《计算机视觉》实验报告

姓名：严昕宇 学号：20121802

实验 8

一. 任务 1

a) 核心代码：

实验8 图像检索

采用SIFT特征实现图像检索功能，即输入一张图片，在数据集中检索出相似的图片，数据集自选。

以下2问选做加分
(1) 基于词袋模型实现
(2) 检索结果按照相似度进行排序

总体思路

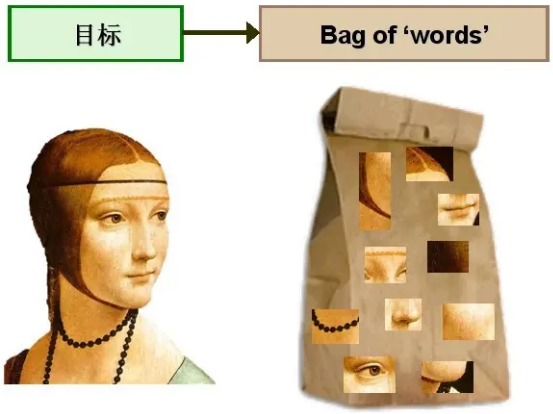
- 输入路径
- 预处理样本库、提取特征值（SIFT）
- 读取待检索图片、提取特征值
- 计算TF/IDF词频（Term Frequency/Inverse Document Frequency）
- 遍历样本库、计算相似度
- 排序、取最相似的前Top-N个结果
- 重排序（Re-ranking）
- 扩展查询（Query Expansion）
- 输出结果

原理分析

Bag of Words 词袋模型

Bag of Words 词袋模型

- 概念很容易理解，BoW模型即对于训练集中的每一幅图像提取其局部特征，然后将所有图像所提取出来的局部特征点作为一个整体进行聚类
- BoW的关键思路在于提取一张图片的特征作为视觉单词/视觉特征（Words/Feature），然后一个个单词聚类/编码成视觉词典/码本（Vocabulary/Codebook）

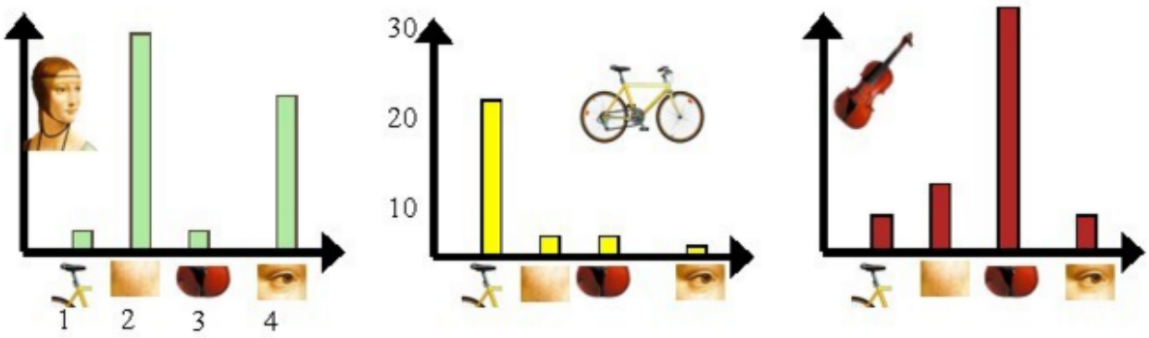


K-means 聚类

- 那么提取到多张图片的多组单词之后怎么办呢？接下来就得开始进行聚类
- Kmeans本质是一种迭代算法，即对于 n 个待处理的特征点，先根据给定的簇的个数 k，初始化 k 个簇中心点，然后对于其他的每一个点，计算其到各个中心点的距离，并将其归入距离自己最近的中心点所在的簇中，如此重复这样一个过程

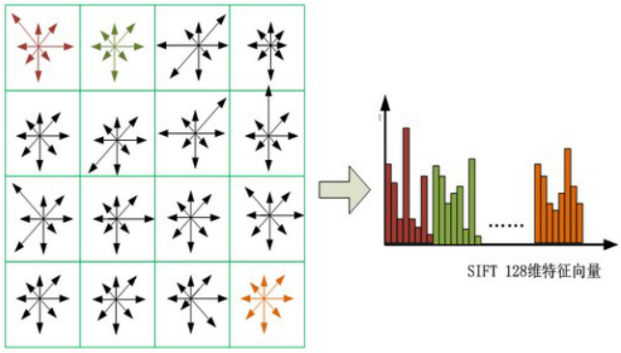
```
选择K个点作为初始质心
repeat
    将每个点指派到最近的质心，形成K个簇
    重新计算每个簇的质心
until 簇不发生变化或达到最大迭代次数
```

- 最后我们用到的是一个K维直方图，即统计单词表中每个单词在图像中出现的次数，从而将图像表示成为一个K维数值向量（这里的K是自己取的）



SIFT 纹理特征提取

- HOG算法得出的是一个大的多维特征向量，这是计算相似度时所乐意得到的。与HOG不同的是，SIFT提取的则是一张图片中的大量特征点，而不同的图片提取出的特征点数量是不同的，所以有效的聚类显得很重要，这也是为什么SIFT往往和K-means一起出没



- SIFT算法的原理更为复杂，但可以直接调用OpenCV库提供的SIFT方法 `detectAndCompute()` 实现

倒排索引&词频 TF-IDF

- TF即Term Frequency，IDF即Inverse Document Frequency；

$$TF = \frac{\text{图像中某个 } word \text{ 出现的次数}}{\text{图像 } word \text{ 的总的个数}}$$

$$IDF = \log\left(\frac{\text{图像的总个数}}{\text{含有该 } word \text{ 的图像数 } +1}\right)$$

- 在文本检索中，TF-IDF技术常用以评估词语对于一个文件数据库中的其中一份文件的重要程度。词语的重要性随着它在文件中出现的频率成正比增加，但同时会随着它在文件数据库中出现的频率成反比下降。
- 引入这两个概念是为了用IDF对BoW模型得出的结果进行加权处理，减少其他不重要的视觉单词所占的权重，其中h为得到的一张图片的直方图向量，IDF公式即：

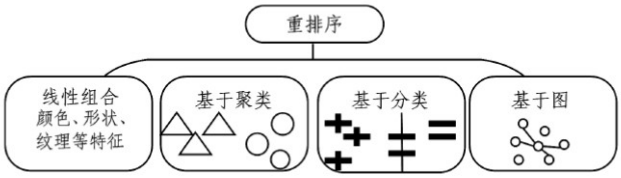
$$h_j = (h_j / \sum_i h_i) \log(\frac{N}{f_n})$$

相似度计算

- 一般方法有两种，一种是直接计算欧式距离，一种是计算余弦相似度；

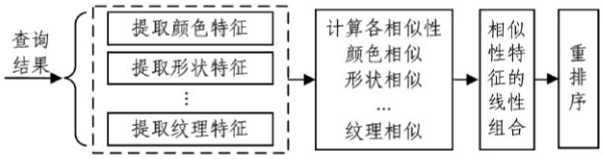
Re-rank 重排序

- 将初始检索得到的图像进行分析和管理，即第二次排序；

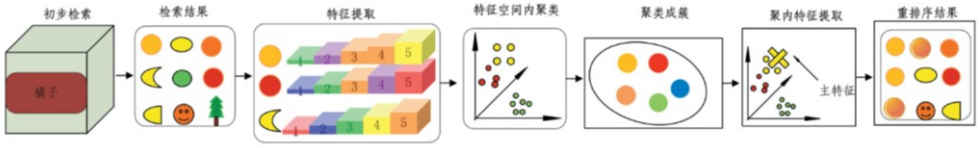


- 常用的手段有两种，一种是基于内容的重排序（如上图），一种是基于文本的重排序，前者准确率较高但效率较低，后者则相反；

- 我这里用了两种最简单的重排序方法，分别是：
- 基于线性组合：先利用某一个特征得到排序效果，对多次排序效果进行**线性融合**,然后按照降序排列得到最终的排序结果。



- 基于聚类：对检索的**图像结果**进行聚类，首先对图像集群进行排序，然后在集群内部进行排序，为了得到最准确的相关聚类，将图像检索结果中最相关的集群图像聚在一起。



Query Expasion 拓展查询

- 对返回的前Top-N个结果，包括查询样本本身，对它们的特征求和取平均，再做一次查询，此过程称为拓展查询。很好理解也很容易实现。
- 通过Query Expansion拓展查询，以达到提高检索召回率的目的。

核心代码（完整代码请见附页）

SIFT特征提取

```
def get_SIFT_feature(img_src, resize_width = img_width, resize_height = img_height):
  img = cv2.imread(img_src)
  gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
  gray = cv2.resize(img, (img_width, img_height))

  # 实例化函数
  sift = cv2.SIFT_create()
  # 找出图像中的关键点
  kp = sift.detect(gray, None)
  # 绘制关键点
  ret = cv2.drawKeypoints(gray, kp, img)
  cv2.imshow('ret', ret)
  cv2.waitKey(0)
  cv2.destroyAllWindows()
  # 计算关键点对应的sift特征
  kp, des = sift.compute(gray, kp)
  # 这里用了cv的库一步到位，返回Descriptor
  return des
```

K-means聚类

```
def kmeans_open(dataSet,k):
    # 随机取k个聚类中心
    # 对每个点求最近k个聚类中心，
    # 更新每个特征点到聚类中心的距离，
    # 求每个聚类中心的特征点集的平均值

    m = np.shape(dataSet)[0]  #行的数目
    # 第一列存样本属于哪一簇
    # 第二列存样本的到簇的中心点的误差
    clusterAssment = np.mat(np.zeros((m,2)))
    clusterChange = True
    # 第1步 初始化centroids
    centroids = randCent(dataSet,k)
    while clusterChange:
        clusterChange = False
        # 遍历所有的样本（行数）
        num = 0
        for i in range(m):
            minDist = 100000.0
            minIndex = -1
            # 遍历所有的质心
            #第2步 找出最近的质心
            for j in range(k):
                # 计算该样本到质心的欧式距离
                distance = distEclud(centroids[j,:],dataSet[i,:])
                if distance < minDist:
                    minDist = distance
                    minIndex = j
            # 第 3 步：更新每一行样本所属的簇
            if clusterAssment[i,0] != minIndex:
                num = num + 1
                clusterChange = True
                clusterAssment[i,:] = minIndex,minDist**2
        #第 4 步：更新质心
        for j in range(k):
            # 获取簇类所有的点 .A:matrix->array
            pointsInCluster = dataSet[np.nonzero(clusterAssment[:,0].A == j)[0]]
            # 对矩阵的列求均值，最后得到一个质心坐标
            centroids[j,:] = np.mean(pointsInCluster,axis=0)
        print(num)
    return clusterAssment.A[:,0], centroids
```

相似度计算

```
# 欧式距离
def EuclideanDist(A, B, length, vecNum):
    result = 0
    for i in range(vecNum):
        sum = 0
        for index in range(length):
            sum += np.power((A[i][index]-B[i][index]),2)
        sum = np.power(sum, 1/2)
        result += sum / vecNum
    return result
```

```
# 余弦相似度
def CosinDist(A, B)
    A = np.mat(A)
    B = np.mat(B)
    num = float(A * B.T)
    denom = np.linalg.norm(A) * np.linalg.norm(B)
    cos = num / denom
    sim = 0.5 + 0.5 * cos
    return sim
```

查询框架

```
# 首次查询
queryRetrieval(datasetCombine, topN)

# 检索评价实现
[recall, precision] = retrievalEvaluation(datasetCombine)
print('查询成功！')
print('Percision：%f' %precision)
print('Recall：%f' %recall)

# 扩展查询
avgFeat = averageFeat(datasetCombine, topN)
datasetCombine = similarityImg(datasetImg, avgFeat)
showImg(datasetCombine, topN)
[recallEQ, precisionEQ] = retrievalEvaluation(datasetCombine)
print('扩展查询成功！')
print('PercisionEQ：%f' %precisionEQ)
print('RecallEQ：%f' %recallEQ)
```

b) 实验结果截图

此处使用的 Caltech 101 数据集是 101 类别的对象的图片组成的图片数据集，主要用于目标识别和图像分类。其中每类约 40 至 800 张图片，大多数类别有大约 50 张图片，每幅图像的大小约为 300x200 像素。且该数据集发布者标注出了其中的目标以供使用。

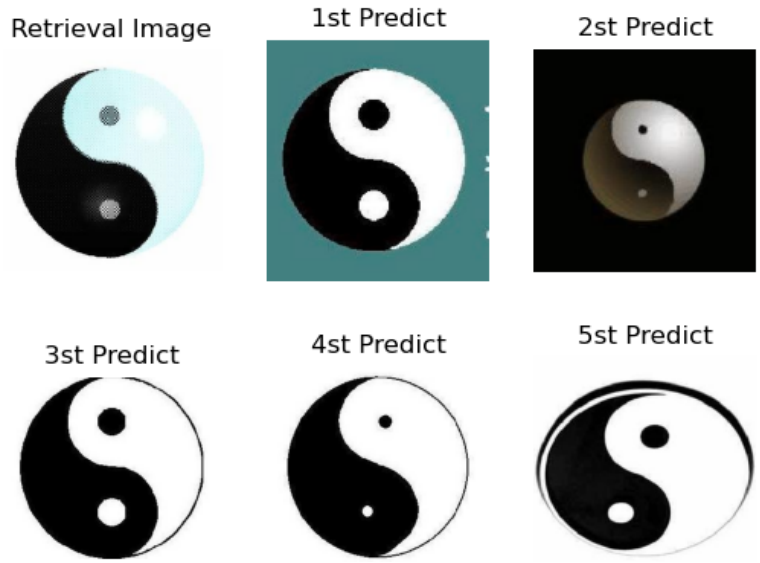


图 1 实验结果 1

图 2 实验结果 2

同时我也选用一些自选图片作为数据集，测试模型效果，效果如下：

首次查询成功！
Percision: 0.800000
Recall: 0.039216

图 3 实验结果 3

图 4 实验结果 4

c) 实验小结

SIFT 特征与 BoW 模型下的图像检索系统作为视觉方向的经典项目，其中牵扯到的原理知识并不是太复杂。但真正上手写代码的时候，还是容易出问题，特别是如何处理从图片中提取到的特征、如何进行匹配，是这个项目中最有意思、也是最具挑战性最关键的一环。

实验结果可以发现，该模型的效果较好，能较为准确的匹配到相似的图片，并按照相似度进行排序。

附页 A：代码

train.py

```
# TF Term Frequency
# IDF Inverse Document Frequency

import cv2
import numpy as np
import re
import matplotlib.pyplot as plt

import ImgInfo  # 引入一个图片信息类
import file  # 多文件读取操作
import hog  # HoG 特征提取
import sift  # SIFT 特征提取
import similarity  # 近似度计算

queryImgAddr = './test/1.jpg'
datasetAddr = './image/'

# feature paraments 图像参数
featType = 'SIFT'  # SIFT, HOG, LBP, Haar-like

# embeding paraments 特征参数
clusterAlgo = 'k-means'
clusterNum = 3

# retrieval paraments
topN = 5  # 返回照片数量

queryImg = cv2.imread(queryImgAddr)

# 读取数据集库中图片的地址，放入数组
datasetImg = file.readImgs(datasetAddr);

print('数据集图片数：' + str(len(datasetImg)) + ' 张')

################ 计算特征值 ###############
datasetImgFeatsHoG = []
datasetImgFeatsSIFT = []

# 提取待检索图片特征值
print('待检索图片特征提取中...')
queryImgFeatHoG = hog.get_HoG_feature(queryImgAddr)
```

```
queryImgFeatSIFT = sift.get_SIFT_feature(queryImgAddr)
print('待检索图片特征提取完毕！')


################ 计算特征值 ###############

################ 计算相似度 ###############
def similarityImg(datasetImg, queryImgFeatHoG, queryImgFeatSIFT=-1):
    simArrayHoG = []
    simArraySIFT = []

    temp = 1

    print('数据集图片特征提取中...')
    print('图片相似度计算中...')
    for datasetImgAddr in datasetImg:

        # print('正在处理:', str(temp) + '/' + str(len(datasetImg)))

        # 遍历数据集图片，提取特征值
        # 一般来说这一步可以后台进行预处理，即提前保存为 txt 文件

        # 提取 HoG 特征
        datasetImgFeatHoG = hog.get_HoG_feature(datasetImgAddr)
        datasetImgFeatsHoG.append(datasetImgFeatHoG)
        # 计算 HoG 相似度（欧氏距离）
        simHoG = similarity.EuclideanDist(queryImgFeatHoG, datasetImgFeatHoG,
len(queryImgFeatHoG[0]),
                                          len(queryImgFeatHoG))
        # print('- HoG 相似度为: %f' %(simHoG))
        simArrayHoG.append(simHoG)

        # 提取 SIFT 特征
        datasetImgFeatSIFT = sift.get_SIFT_feature(datasetImgAddr)
        datasetImgFeatsSIFT.append(datasetImgFeatSIFT)
        # 计算 SIFT 相似度
        if (queryImgFeatSIFT != -1):
            simSIFT = similarity.KNN(queryImgFeatSIFT, datasetImgFeatSIFT)
        else:
            simSIFT = 0
        # print('- SIFT 相似度为: %f' %(simSIFT))
        simArraySIFT.append(simSIFT)
        temp += 1
```

```
        # print (simArrayHoG)
        # print (simArraySIFT)
    print('数据集图片特征提取完毕！')
    print('图片相似度计算完毕！')

    # 以类的形式存储图片的路径和特征值
    datasetCombine = []

    # 重排序：对两种相似性特征结果进行线性组合
    for index in range(len(datasetImg)):
        datasetCombine.append(
            ImgInfo.ImgInfo(datasetImg[index], simArrayHoG[index],
simArraySIFT[index], datasetImgFeatsHoG[index]))

    # 遍历数据对象集合，并以特征值升序排序
    datasetCombine = sorted(datasetCombine)

    return datasetCombine


################ 计算相似度 ###############

################ 查询结果 ###############
# def showImg(datasetCombine, topN):
#     for index in range(topN):
#         title = 'Retrieval Image ' + '#' + str(index + 1)
#         img = cv2.imread(datasetCombine[index].img_src)
#         cv2.imshow(title, img)
#     cv2.waitKey(0)
#     cv2.destroyAllWindows()
def showImg(datasetCombine, topN):
    # 设定整个画布的尺寸
    fig = plt.figure()
    # 目标图
    ax = fig.add_subplot(2, 3, 1)
    target_img = cv2.cvtColor(queryImg, cv2.COLOR_BGR2RGB)
    ax.imshow(target_img)
    plt.axis('off')
    # 标题
    plt.title('Retrieval Image')
    # 第一幅图的下标从 2 开始，设置 5 子图
    for index in range(topN):
        # 图片
        img = cv2.imread(datasetCombine[index].img_src)
```

```
        img_RGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        # 往画布上添加子图：按二行三列，添加到下标为plt_index的位置
        ax = fig.add_subplot(2, 3, index + 2)
        # 绘制对应的子图
        ax.imshow(img)
        # 子图标题
        ax.set_title(f'{str(index + 1)}st Predict')
        plt.axis('off')
        pass

    # 关闭坐标轴
    plt.axis('off')
    plt.xticks(alpha=0)
    plt.tick_params(axis='x', width=0)
    # 显示画布
    plt.show()


################ 查询查询 ###############

################ 首次普通查询 ###############
def queryRetrieval(datasetCombine, topN):
    showImg(datasetCombine, topN)


################ 首次普通查询 ###############

################ 扩展查询 Query Expasion ###############
# 这里对HoG返回的前TopN个结果+查询样本求和取平均，然后再进行一次查询
def averageFeat(datasetCombine, topN):
    row = len(datasetCombine[0].HoG_feat)
    col = len(datasetCombine[0].HoG_feat[0])

    # 取平均
    sum = datasetCombine[0].HoG_feat.tolist()
    for index in range(topN):
        for i in range(row):
            for j in range(col):
                if (index == 0):
                    sum[i][j] /= topN
                else:
                    sum[i][j] += datasetCombine[index].HoG_feat[i][j] / topN
    return np.array(sum)
```

```
'''
def queryExpasion(datasetCombine, topN):
  avgFeat = averageFeat(datasetCombine, topN)
  datasetCombine = similarityImg(datasetImg, avgFeat)
  showImg(datasetCombine, topN)
'''


################ 扩展查询 Query Expasion ###############

################ 计算召回率和精确度 ###############
def getPlateNum(filePath):
    # 利用正则表达式匹配车牌号(图片文件名)
    re_year = re.compile(r'_2015')  # 模式对象匹配
    index = re.search(re_year, filePath).span()[0] - 6
    plateNum = filePath[index:index + 6]
    return plateNum

def retrievalEvaluation(datasetRank):
    targetPlateNum = getPlateNum(datasetRank[0].img_src)

    # 计算召回率和精确度相关的变量
    correctNum = 1  # 检索正确的数量
    allNum = len(datasetRank)

    # 遍历检索结果
    for index in range(1, topN):
        tmpNum = getPlateNum(datasetRank[index].img_src)
        if tmpNum == targetPlateNum:
            # 如果车牌号正确匹配，即检索成功
            correctNum += 1

    # 召回率：正确结果占总体数据
    recall = correctNum / allNum
    # 精确度：正确结果占返回结果
    precision = correctNum / topN

    return [recall, precision]


################ 计算召回率和精确度 ###############
datasetCombine = similarityImg(datasetImg, queryImgFeatHoG, queryImgFeatSIFT)
```

```
# 首次查询
queryRetrieval(datasetCombine, topN)

# 检索评价实现
[recall, precision] = retrievalEvaluation(datasetCombine)

print('首次查询成功! ')
print('Percision: %f' % precision)
print('Recall: %f' % recall)

# 扩展查询
print('扩展查询开始! ')

avgFeat = averageFeat(datasetCombine, topN)
datasetCombine = similarityImg(datasetImg, avgFeat)
showImg(datasetCombine, topN)

[recallEQ, precisionEQ] = retrievalEvaluation(datasetCombine)

print('扩展查询成功! ')
print('PercisionEQ: %f' % precisionEQ)
print('RecallEQ: %f' % recallEQ)
```

ImgInfo.py

```
from numpy import *
import operator

# 图像信息类

# 线性融合参数 手动调节
alpha_HoG = 0.7
alpha_SIFT = 1 - alpha_HoG


class ImgInfo(object):
    similarity = -1

    # 构造函数
    def __init__(self, img_src, HoG_similarity, SIFT_similarity, HoG_feat):
        self.img_src = img_src
        self.HoG_similarity = HoG_similarity
        self.HoG_feat = HoG_feat
        self.similarity = ImgInfo.combine_similarity_linearly(HoG_similarity,
SIFT_similarity)
```

```
    # 线性组合特征
    def combine_similarity_linearly(HoG_similarity, SIFT_similarity):
        similarity = alpha_HoG * (HoG_similarity + 0.8) + alpha_SIFT *
SIFT_similarity / 100
        return similarity

    # 相等操作符
    def __eq__(self, other):
        return self.img_src == other.img_src and self.HoG_similarity ==
other.HoG_similarity

    # 大于操作符
    def __gt__(self, other):
        return self.similarity > other.similarity

    # 小于操作符
    def __lt__(self, other):
        return self.similarity < other.similarity
```

file.py

```
# 处理多文件读取操作，读取多个文件的地址放在一个数组中返回
import os
import re

# 存储结果的数组
resultArr = [];

# 读取数据集文件夹里面的图片，返回所有图片的地址列表
def readImgs(addr):
    # 因为目前所有的图片都是.jpg 形式，所以正则判断时只用了.jpg
    reg = re.compile(r'.jpg');
    for filename in os.listdir(addr):
        filePath = addr + filename;
        if re.search(reg, filePath):  # 如果是图片的话直接加到列表
            resultArr.append(filePath);
        else:
            # 每个文件夹存在一个看不到但是会被 python 识别的文件，做特殊处理（就是不处理）
            if filename != 'Thumbs.db':  # 这个情况下，如果条件达成，意味着是文件夹
                readImgs(filePath + '/');  # 是文件夹的话递归就好
    return resultArr;
```

hog.py

```
import cv2
import numpy as np
import math
import matplotlib.pyplot as plt


img_src = 'd:/test/1.jpg'  # 测试用图片路径
cell_size = 8  # 将 cell 设为 8*8 个像素
bin_size = 8  # 8 个 bin 的直方图

# 图片大小格式化
img_height = 500
img_width = 500


def cell_gradient(cell_magnitude, cell_angle):
    angle_unit = 360 / bin_size
    orientation_centers = [0] * bin_size
    for k in range(cell_magnitude.shape[0]):
        for l in range(cell_magnitude.shape[1]):
            gradient_strength = cell_magnitude[k][l]
            gradient_angle = cell_angle[k][l]
            min_angle = int(gradient_angle / angle_unit) % 8
            max_angle = (min_angle + 1) % bin_size
            mod = gradient_angle % angle_unit
            orientation_centers[min_angle] += (gradient_strength * (1 - (mod /
angle_unit)))
            orientation_centers[max_angle] += (gradient_strength * (mod /
angle_unit))
    return orientation_centers


def get_HoG_feature(img_src, resize_width=img_width,
resize_height=img_height):
    # 读取图片，以灰度图的形式
    img = cv2.imread(img_src, cv2.IMREAD_GRAYSCALE)
    img = cv2.resize(img, (img_width, img_height))
    # 采用 Gamma 校正对图像的颜色空间归一化
    img = np.sqrt(img / float(np.max(img)))

    # 计算每个像素的梯度值
    # 计算图像横坐标和纵坐标方向的梯度，并据此计算每个像素位置的梯度方向值；
    # 求导操作不仅能够捕获轮廓，人影和一些纹理信息，还能进一步弱化光照的影响。
```

```
    # 在求出输入图像中像素点（x,y）处的水平方向梯度、垂直方向梯度和像素值，从而求出梯度幅值和方向。
    height, width = img.shape
    gradient_values_x = cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=5)
    gradient_values_y = cv2.Sobel(img, cv2.CV_64F, 0, 1, ksize=5)
    gradient_magnitude = cv2.addWeighted(gradient_values_x, 0.5, gradient_values_y, 0.5, 0)
    gradient_angle = cv2.phase(gradient_values_x, gradient_values_y, angleInDegrees=True)
    # print ('gradient_magnitude.shape, gradient_angle.shape: ')
    # print (gradient_magnitude.shape, gradient_angle.shape)

    # 为每个细胞单元构建梯度方向直方图
    angle_unit = 360 / bin_size  # 每个梯度方向块的角度
    gradient_magnitude = abs(gradient_magnitude)
    # print ('cell_gradient_vector:')
    cell_gradient_vector = np.zeros((height // cell_size, width // cell_size, bin_size))
    # print (cell_gradient_vector.shape)

    for i in range(cell_gradient_vector.shape[0]):
        for j in range(cell_gradient_vector.shape[1]):
            cell_magnitude = gradient_magnitude[i * cell_size:(i + 1) * cell_size,
                             j * cell_size:(j + 1) * cell_size]
            cell_angle = gradient_angle[i * cell_size:(i + 1) * cell_size,
                         j * cell_size:(j + 1) * cell_size]
            cell_gradient_vector[i][j] = cell_gradient(cell_magnitude, cell_angle)

    # 可视化 Cell 梯度直方图
    hog_image = np.zeros([height, width])
    cell_width = cell_size / 2
    max_mag = np.array(cell_gradient_vector).max()
    for x in range(cell_gradient_vector.shape[0]):
        for y in range(cell_gradient_vector.shape[1]):
            cell_grad = cell_gradient_vector[x][y]
            cell_grad /= max_mag
            angle = 0
            angle_gap = angle_unit
            for magnitude in cell_grad:
                angle_radian = math.radians(angle)
                x1 = int(x * cell_size + magnitude * cell_width * math.cos(angle_radian))
                y1 = int(y * cell_size + magnitude * cell_width *
```

```
math.sin(angle_radian))
                x2 = int(x * cell_size - magnitude * cell_width *
math.cos(angle_radian))
                y2 = int(y * cell_size - magnitude * cell_width *
math.sin(angle_radian))
                cv2.line(hog_image, (y1, x1), (y2, x2), int(255 *
math.sqrt(magnitude)))
                angle += angle_gap

    # plt.imshow(hog_image, cmap=plt.cm.gray)
    # plt.show()

    # 统计 Block 的梯度信息
    hog_vector = []
    for i in range(cell_gradient_vector.shape[0] - 1):
        for j in range(cell_gradient_vector.shape[1] - 1):
            block_vector = []
            block_vector.extend(cell_gradient_vector[i][j])
            block_vector.extend(cell_gradient_vector[i][j + 1])
            block_vector.extend(cell_gradient_vector[i + 1][j])
            block_vector.extend(cell_gradient_vector[i + 1][j + 1])
            mag = lambda vector: math.sqrt(sum(i ** 2 for i in vector))
            magnitude = mag(block_vector)
            if magnitude != 0:
                normalize = lambda block_vector, magnitude: [element / magnitude
for element in block_vector]
                block_vector = normalize(block_vector, magnitude)
            hog_vector.append(block_vector)
    # print ('np.array(hog_vector).shape')
    # print (np.array(hog_vector).shape)
    return np.array(hog_vector)


# print('以下目标图片的特征值: ')
# print(get_HoG_feature(img_src))
# print(get_HoG_feature(img_src).shape)
# print('特征值计算成功! ')

cv2.waitKey(0)
```

sift.py

```
import os
import cv2
import numpy as np
import math
import matplotlib.pyplot as plt

from typing import Tuple, Dict
from PIL import Image
from pylab import *

# 格式化统一大小
img_height = 500
img_width = 500

# 注：由于 SIFT 库在 OpenCV3.4 之后被移除，而 2020 年 3 月 7 日后，SIFT 的专利将到期，因此应免费使用。
# opencv 将其移出非自由文件夹将使所有人都能使用默认标志编译 opencv 并仍然获得 SIFT。
# 参见 https://github.com/opencv/opencv/issues/16736

sift_det: cv2.Feature2D = cv2.SIFT_create()

'''
def get_SIFT_feature(img_src, resize_width = img_width, resize_height =
img_height):
  img = cv2.imread(img_src)
  gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
  gray = cv2.resize(img, (img_width, img_height))

  # 实例化函数
  sift = cv2.SIFT_create()
  # 找出图像中的关键点
  kp = sift.detect(gray, None)

  # 绘制关键点
  # ret = cv2.drawKeypoints(gray, kp, img)
  # cv2.imshow('ret', ret)
  # cv2.waitKey(0)
  # cv2.destroyAllWindows()

  # 计算关键点对应的 sift 特征
  kp, des = sift.compute(gray, kp)
  # 这里用了 cv 的库一步到位，返回 Descriptor
  return des
```

```
  '''


def get_SIFT_feature(img_src, resize_width=img_width,
resize_height=img_height):
    img = cv2.imread(img_src)
    img = cv2.resize(img, (img_width, img_height))
    return sift_det.detectAndCompute(img, None)
```

kmeans.py

```
from PIL import Image
import numpy as np
import os
import matplotlib.pyplot as plt
import pdb
import cv2
import math
import pylab

# 针对 SURF/SIFT 结果

# Perform Tf-Idf vectorization
nbr_occurences = np.sum((im_features > 0) * 1, axis=0)
idf = np.array(np.log((1.0 * len(image_paths) + 1) / (1.0 * nbr_occurences + 1)),
'float32')


# 为给定数据集构建一个包含 K 个随机质心的集合
def randCent(dataSet, k):
    m, n = dataSet.shape
    centroids = np.zeros((k, n))
    for i in range(k):
        index = int(np.random.uniform(0, m))  #
        centroids[i, :] = dataSet[index, :]
    return centroids


def kmeans_open(dataSet, k):
    # 随机取 k 个聚类中心
    # 对每个点求最近 k 个聚类中心，
    # 更新每个特征点到聚类中心的距离，
    # 求每个聚类中心的特征点集的平均值

    m = np.shape(dataSet)[0]  # 行的数目
```

```
    # 第一列存样本属于哪一簇
    # 第二列存样本的到簇的中心点的误差
    clusterAssment = np.mat(np.zeros((m, 2)))
    clusterChange = True

    # 第 1 步 初始化 centroids
    centroids = randCent(dataSet, k)
    while clusterChange:
        clusterChange = False

        # 遍历所有的样本（行数）
        num = 0
        for i in range(m):
            minDist = 100000.0
            minIndex = -1

            # 遍历所有的质心
            # 第 2 步 找出最近的质心
            for j in range(k):
                # 计算该样本到质心的欧式距离
                distance = distEclud(centroids[j, :], dataSet[i, :])
                if distance < minDist:
                    minDist = distance
                    minIndex = j
            # 第 3 步：更新每一行样本所属的簇
            if clusterAssment[i, 0] != minIndex:
                num = num + 1
                clusterChange = True
                clusterAssment[i, :] = minIndex, minDist ** 2
        # 第 4 步：更新质心
        for j in range(k):
            pointsInCluster = dataSet[np.nonzero(clusterAssment[:, 0].A == j)[0]]
# 获取簇类所有的点 .A:matrix->array
            centroids[j, :] = np.mean(pointsInCluster, axis=0)  # 对矩阵的列求均值，
最后得到一个质心坐标
        print(num)

    return clusterAssment.A[:, 0], centroids
```

similarity.py

```
import cv2
import numpy as np
from numpy import *


# import kmeans

# 欧式距离
def EuclideanDist(A, B, length, vecNum):
    result = 0
    for i in range(vecNum):
        sum = 0
        for index in range(length):
            sum += np.power((A[i][index] - B[i][index]), 2)
        sum = np.power(sum, 1 / 2)
        result += sum / vecNum
    return result


'''
# 余弦相似度
def CosinDist(A, B)
    A = np.mat(A)
    B = np.mat(B)
    num = float(A * B.T)
    denom = np.linalg.norm(A) * np.linalg.norm(B)
    cos = num / denom
    sim = 0.5 + 0.5 * cos
    return sim
'''

'''
# 不太地道
def getHogSim(A, B, length, vecNum):
    return EuclideanDist(A, B, length, vecNum)
'''


def KNN(datasetSiftFeat, querySiftFeat):
    kp1, des1 = datasetSiftFeat
    kp2, des2 = querySiftFeat
    # 创建暴力匹配对象
    bf = cv2.BFMatcher()
```

```
    # 使用 Matcher.knnMatch 获得两幅图像的最佳匹配
    matches = bf.knnMatch(des1, des2, k=2)

    good = []
    for m, n in matches:
        if m.distance < 0.75 * n.distance:
            good.append([m])

    # 对 good 中的匹配点的 distance 进行加权平均
    sum = 0
    for index in range(len(good)):
        sum += good[index][0].distance

    return sum / len(good)
```