机器视觉第九次作业

1、编写伪代码实现一个基于尺度不变特征 SIFT 的图像检索系统以及实现系统性能度量的评价。

SIFT 是一种用于检测和描述图像局部特征的算法，具有尺度不变性和旋转不变性，它的主要思想是先构建高斯金字塔，随后由高斯金字塔构建高斯差分金字塔。在差分金字塔中进行每层每组与其上下两组进行对比，若该像素点是极值则保留，如果不是则丢弃。

SIFT 图像检索：

```
function detectAndCompute(image):
    # 构建高斯金字塔，生成不同尺度的图像
    gaussian_pyramid = build_gaussian_pyramid(image)
    # 构建高斯差分金字塔（DoG），通过相邻尺度的高斯图像相减得到
    dog_pyramid = build_dog_pyramid(gaussian_pyramid)
    # 初始化关键点列表
    keypoints = []
    # 遍历 DoG 金字塔中的每一层
    for scale in dog_pyramid:
        # 遍历当前层中的每个像素
        for x, y in scale:
            # 检查当前像素是否为局部极值点
            if is_local_extremum(dog_pyramid, x, y):
                # 如果是极值点，将其添加到关键点列表中
                keypoints.append((x, y, scale))
    # 初始化精炼后的关键点列表
    refined_keypoints = []
    # 遍历所有初步检测到的关键点
    for kp in keypoints:
        # 对关键点进行精确定位，通过泰勒展开插值
        refined_kp = refine_keypoint(kp, dog_pyramid)
```

```
            # 检查关键点是否稳定（剔除低对比度和边缘响应的点）
            if is_valid_keypoint(refined_kp):
                # 如果是稳定的关键点，将其添加到精炼后的列表中
                refined_keypoints.append(refined_kp)
        # 初始化带方向的关键点列表
        oriented_keypoints = []
        # 遍历所有精炼后的关键点
        for kp in refined_keypoints:
            # 为关键点分配主方向，使其具有旋转不变性
            orientation = assign_orientation(kp, image)
            # 将关键点及其方向添加到列表中
            oriented_keypoints.append((kp, orientation))
        # 初始化描述符列表
        descriptors = []
        # 遍历所有带方向的关键点
        for kp, orientation in oriented_keypoints:
            # 计算关键点的描述符，通常是一个 128 维的向量
            descriptor = compute_descriptor(kp, orientation, image)
            # 将描述符添加到列表中
            descriptors.append(descriptor)
        # 返回最终的关键点和描述符
        return oriented_keypoints, descriptors
    # 提取图像的 SIFT 特征（关键点和描述符）
    function extractSIFTFeatures(image):
        # 使用 SIFT 算法检测图像中的关键点并计算描述符
        # SIFT 关键点是图像中的显著特征，描述符是对这些特征的数学描述
        # detectAndCompute 方法返回两个结果：
        #   - keypoints：检测到的关键点列表，包含位置、尺度和方向等信息
```

```
    #   - descriptors: 关键点的描述符，通常是一个 128 维的向量
    keypoints, descriptors = SIFT.detectAndCompute(image)
    return keypoints, descriptors
# 匹配两个图像的特征描述符
function matchFeatures(descriptor1, descriptor2):
    # 使用匹配器（可以使用暴力匹配器或 FLANN）对两组描述符进行匹配
    # 暴力匹配器会计算 descriptor1 和 descriptor2 中每对描述符的距离
    # FLANN 是一种近似最近邻搜索算法，速度更快，适合大规模数据
    # 返回的 matches 是一个列表，包含匹配的特征点对
    matches = Matcher.match(descriptor1, descriptor2)
    return matches
# 计算匹配的距离
function computeDistance(match):
    # 计算一对匹配的特征描述符之间的距离
    # 距离越小，表示两个特征点越相似
    return match.distance
# 基于 SIFT 特征进行图像检索
function retrieveSimilarImages(queryImage, imageDatabase):
    # 提取查询图像的 SIFT 特征
    queryKeypoints, queryDescriptors = extractSIFTFeatures(queryImage)
    # 用于存储每张图像与查询图像的匹配得分
    scores = []
    # 遍历图像数据库中的每一张图像
    for image in imageDatabase:
        # 提取当前数据库图像的 SIFT 特征
        keypoints, descriptors = extractSIFTFeatures(image)
                # 匹配查询图像与当前数据库图像的特征描述符
        matches = matchFeatures(queryDescriptors, descriptors)
                # 计算匹配结果的总得分
```

```
        totalScore = 0
        for match in matches:
            # 累加每个匹配的距离
            totalScore += computeDistance(match)
                # 将当前图像和其得分加入到分数列表中
        scores.append((image, totalScore))
        # 按照得分从低到高排序，得分越低代表图像越相似
    scores.sort(key=lambda x: x[1])
        # 返回前 5 张最相似的图像
    return scores[:5]
# 计算精确度 (Precision)
function computePrecision(retrieved, relevant):
    # 精确度是检索结果中相关图像的比例
    # retrieved: 检索到的图像列表
    # relevant: 所有相关图像的列表
    # retrieved ∩ relevant: 检索到的相关图像
    return len(retrieved ∩ relevant) / len(retrieved)
# 计算召回率 (Recall)
function computeRecall(retrieved, relevant):
    # 召回率是检索到的相关图像占所有相关图像的比例
    # retrieved: 检索到的图像列表
    # relevant: 所有相关图像的列表
    # retrieved ∩ relevant: 检索到的相关图像
    return len(retrieved ∩ relevant) / len(relevant)
# 计算 F1 分数
function computeF1Score(precision, recall):
    # F1 分数是精确度和召回率的调和平均数
    # 用于综合衡量检索结果的精确性和覆盖率
```

```
    if precision + recall == 0:
        return 0
    return 2 * (precision * recall) / (precision + recall)
# 计算单次查询的平均精度 (Average Precision, AP)
function computeAveragePrecision(retrieved, relevant):
    # 计算检索结果的平均精度
    # retrieved: 检索到的图像列表 (按相似度排序)
    # relevant: 所有相关图像的列表
    precision_at_k = []  # 存储每个位置 k 的精确度
        # 遍历检索结果，计算每个位置 k 的精确度
    for k = 1 to len(retrieved):
        # 计算前 k 个检索结果的精确度
        precision_at_k.append(computePrecision(retrieved[:k], relevant))
        # 返回平均精度，即所有位置精确度的平均值
    return average(precision_at_k)
# 计算均值平均精度 (Mean Average Precision, MAP)
function computeMAP(queryResults, relevantImages):
    # 计算所有查询的平均精度均值 (MAP)
    # queryResults: 每个查询的检索结果列表
    # relevantImages: 每个查询对应的相关图像列表
    avgPrecision = 0  # 用于累加所有查询的平均精度
        # 遍历所有查询结果和对应的相关图像
    for query, relevant in zip(queryResults, relevantImages):
        # 计算当前查询的平均精度并累加
        avgPrecision += computeAveragePrecision(query, relevant)
      # 返回 MAP，即所有查询的平均精度的均值
    return avgPrecision / len(queryResults)
```

2、阅读论文：Josef Sivic and Andrew Zisserman, Video Google: A Text Retrieval Approach to Object Matching in Videos. ICCV 2003.

这篇文章是由 Josef Sivic 和 Andrew Zisserman 在 ICCV 2003 发表的一篇开创性论文。在当时的背景下，随着数字视频内容的爆炸式增长，如何在海量视频数据中快速、准确地检索特定内容成为一个亟待解决的问题。

传统的基于全局特征的检索方法计算复杂度高，且对视角变化和部分遮挡等情况较为敏感，所以这篇论文提出了一种创新的视频检索方法，通过将文本检索中的成熟技术巧妙地迁移到视频检索领域，为解决大规模视频数据的快速检索问题提供了全新的思路。

论文的核心思想是将视频中的视觉特征类比为文本文档中的单词，采用文本检索中广泛使用的"词袋"模型来表示视频帧中的视觉内容。这种类比非常富有创意，就像文档是由词语组成的，图像也可以看作是由视觉特征组成的。通过这种方式，可以将复杂的视觉匹配问题转化为相对简单的文本检索问题。具体来说，首先使用 SIFT（尺度不变特征变换）等局部特征检测器在视频关键帧中提取特征点，并计算这些点的特征描述符。这些局部特征具有对旋转、尺度变化和光照变化的不变性，能够稳定地描述图像的局部结构。然后，通过对所有特征描述符进行聚类，将聚类中心作为"视觉词汇"，构建视觉词汇表。这个过程相当于建立了一个视觉的"字典"，每个视觉词代表了一类特定的局部图像模式。

这个系统的核心思想是将视频检索问题转化为文本检索问题：首先使用 SIFT 算法提取视频帧的局部特征，然后通过 K-means 聚类构建"视觉词汇表"，将特征量化为离散的视觉词，接着借鉴文本检索的思想构建倒排索引并使用 tf-idf 权重，最终实现基于内容的视频帧快速检索。在进行复现的过程中由于使用了 opencv 库函数，所以初步的提取特征直接可以使用库函数进行提取，空间验证和搜索相似帧代码如图 1 所示。在测试过程中，发现代码在 kmeans 聚类步骤所耗费时间最长。

```python
def spatial_verification(self, query_keypoints, candidate_keypoints):
    """
    空间验证
    """
    if len(query_keypoints) < 4 or len(candidate_keypoints) < 4:
        return False
    # RANSAC找到单应性矩阵
    H, mask = cv2.findHomography(query_keypoints, candidate_keypoints,
                                 cv2.RANSAC, 5.0)
    # 计算内点比例
    inlier_ratio = np.sum(mask) / len(mask)
    return inlier_ratio > 0.5

def search(self, query_image, top_k=10):
    """
    搜索最相似的帧
    """
    # 提取查询图像特征
    query_keypoints, query_descriptors = self.extract_sift_features(query_image)
    query_word_freq = self.quantize_features(query_descriptors)
    query_weights = self.compute_tf_idf(query_word_freq)

    # 计算相似度得分
    scores = defaultdict(float)
    for word_id, weight in query_weights.items():
        for frame_id, freq in self.inverted_index[word_id]:
            frame_weight = freq * self.idf[word_id]
            scores[frame_id] += weight * frame_weight

    # 归一化得分
    if scores:
        max_score = max(scores.values())
        scores = {k: v/max_score for k, v in scores.items()}

    # 获取top-k结果
    ranked_results = sorted(scores.items(), key=lambda x: x[1], reverse=True)
    return ranked_results[:top_k]
```

图 1 部分代码图

```
总共加载了 76 帧
开始构建视觉词汇...
特征提取进度: 10/76
特征提取进度: 20/76
特征提取进度: 30/76
特征提取进度: 40/76
特征提取进度: 50/76
特征提取进度: 60/76
特征提取进度: 70/76
开始K-means聚类, 特征点总数: 13980
视觉词汇表构建完成, 包含 10000 个视觉词
构建倒排索引...
帧 9: 相似度得分 1.0000
帧 5: 相似度得分 0.9954
帧 10: 相似度得分 0.8341
帧 6: 相似度得分 0.7726
帧 3: 相似度得分 0.7251
帧 4: 相似度得分 0.7099
帧 7: 相似度得分 0.6272
帧 29: 相似度得分 0.5617
帧 62: 相似度得分 0.5237
帧 28: 相似度得分 0.4914
```

图 2 代码测试相似度结果

论文在多部电影片段上进行了广泛的实验，包括不同场景、不同光照条件和不同视角下的物体检索任务。从实验结果中可以看出，该方法不仅能够准确地在视频中检索特定物体，而且检索速度远超传统方法。在一个包含数小时视频内容的数据集上，系统能够在几秒钟内完成检索。这种高效性主要得益于视觉词袋模型和倒排索引的使用，它们显著减少了特征匹配的计算量。此外，通过引入空间一致性验证，系统还能有效地过滤掉误匹配，提高检索精度。该方法可以应用于多个重要领域。例如，在电影制作过程中，后期制作团队常常需要在海量素材中找到特定道具或场景出现的所有片段，使用这种方法可以将原本需要数小时的人工检索工作缩短到几秒钟。

然而，这种方法也存在一些值得关注的局限性：检索效果对视觉特征的选择较为敏感，不同的特征提取方法可能导致显著不同的结果；在处理大角度视角变化时表现不够理想（因为局部特征的描述符可能发生显著变化）；视觉词汇量的选择也是一个需要仔细权衡的问题：词汇量太小会导致区分性不足，而词汇量太大则会增加计算复杂度并可能引入噪声。这些问题在当今深度学习时代都有了新的解决方案，例如使用卷积神经网络提取更鲁棒的特征，或使用学习得到的特征编码方法。但论文提出的基本框架思想仍然具有重要的参考价值。

从个人角度来看，这篇论文最大的贡献在于开创性地将文本检索的思想引入视觉领域，为后续的视觉检索研究提供了新的范式。这种跨领域的思维方式非常值得学习：当面对一个看似复杂的问题时，如果能够找到合适的类比，将其转化为一个已经有成熟解决方案的问题，往往能够事半功倍。虽然现在已经有了更先进的深度学习方法，但论文提出的将复杂视觉问题简化为文本检索问题的思路，以及设计简单而有效的解决方案的方法，仍然值得我们深入学习和借鉴。