《计算机视觉》（本科，2024）作业3

丁晟元　　杜凌霄

211250097 211250066

1　Q1

要求说明

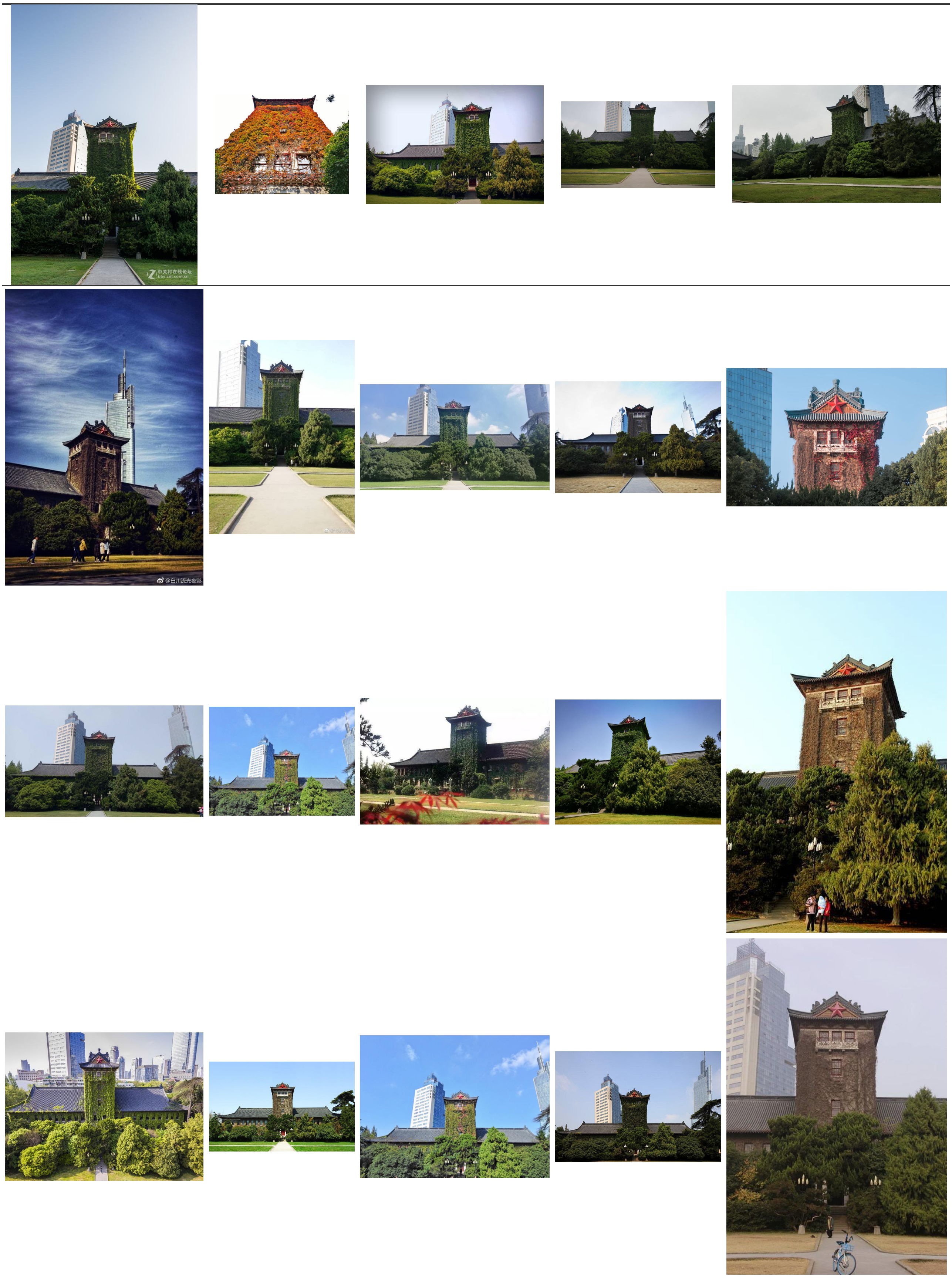
在搜索引擎上按照某一关键词，搜索50张不同的图像，从中选出5张作为检索请求，另45张作为被检索图像。

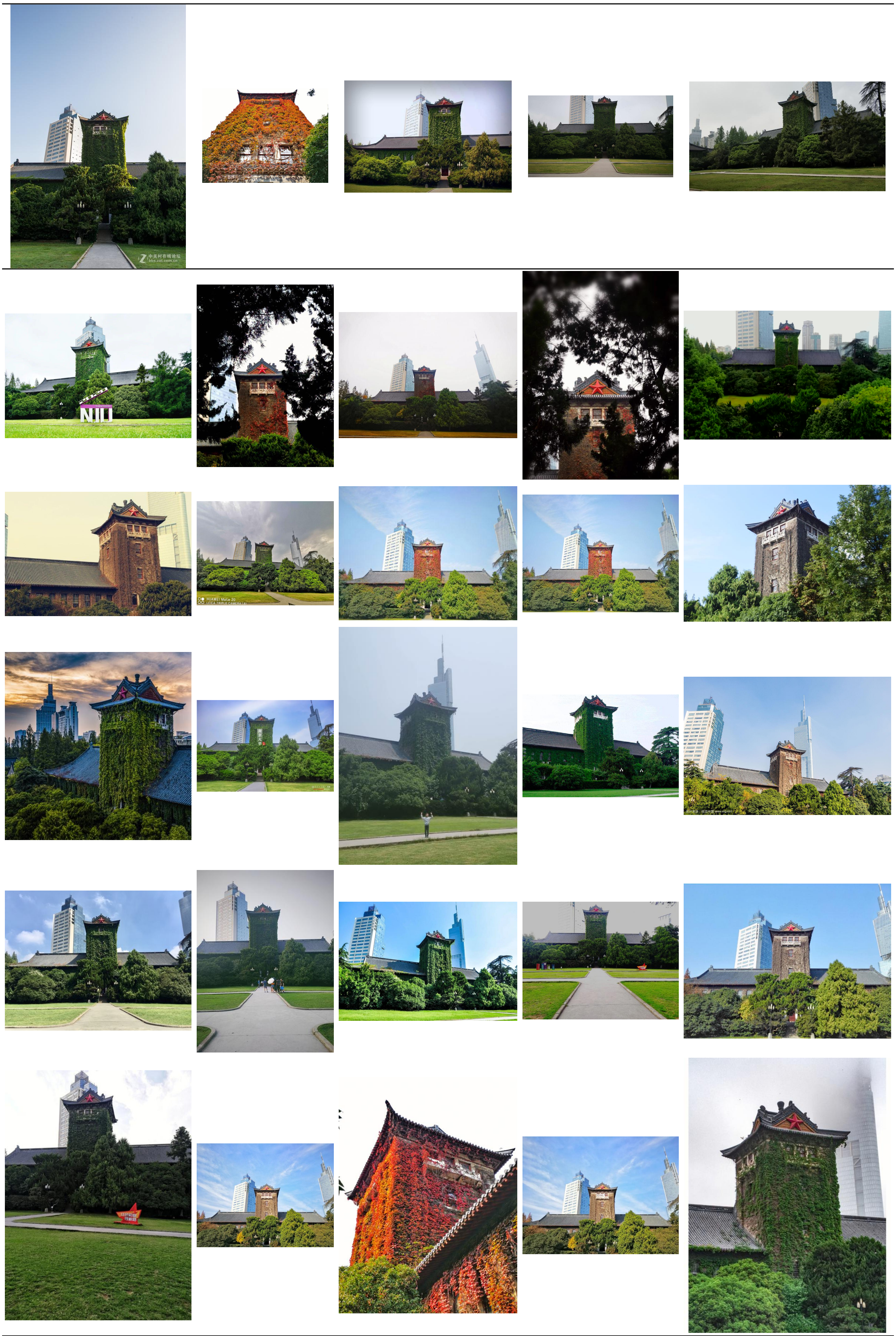
使用爬虫，在**百度图片**进行搜索，关键词为**“南京大学鼓楼校区北大楼”**，先爬取了90张图片（以防有不相关的，因此先多爬取了点，例如结果中有爬到了紫峰大厦），经过筛选后留下的5+45如下所示：

这5张图像虽然都是北大楼，但是差异较为明显，也算是选了比较清晰的不同时间和视角的北大楼

1.1　5张检索请求图像

1.2 45张被检索图像




NJU

2 Q2

要求说明

以全局RGB颜色直方图（每通道bin的数量为8）作为特征，进行图像检索。展示每个检索请求及对应前3个结果。

2.1 特征抽取代码（全局RGB颜色直方图）

```python
def extract_rgb_histogram(image_path, bins=8):
    # 读取图像
    image = cv2.imread(image_path)
    if image is None:
        raise ValueError(f"无法读取图像文件: {image_path}")
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    # 计算直方图
    histogram = [cv2.calcHist([image], [i], None, [bins], [0, 256]) for i in range(3)]
    # ravel()将(3, 8)的数组变成(24,)的数组
    histogram = np.concatenate(histogram).ravel()
    # 比如q1是679*500, 那么histogram.sum()=679*500*3
    histogram = histogram / histogram.sum()  # 归一化
    return histogram


# 处理文件夹中的所有图片
def process_images(folder_path):
    histograms = {}
    for filename in os.listdir(folder_path):
        file_path = os.path.join(folder_path, filename)
        if os.path.isfile(file_path) and file_path.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp', '.tiff')):
            try:
                histograms[filename] = extract_rgb_histogram(file_path)
            except ValueError as e:
                print(e)
    return histograms


# 完成图片RGB直方图(24个bin总共)的提取
query_histograms = process_images('./images/query/')
database_histograms = process_images('./images/database/')
```

2.2 检索（特征匹配）代码

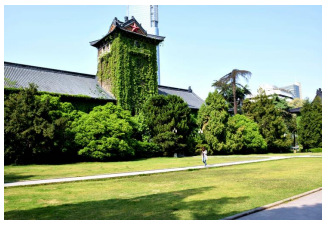
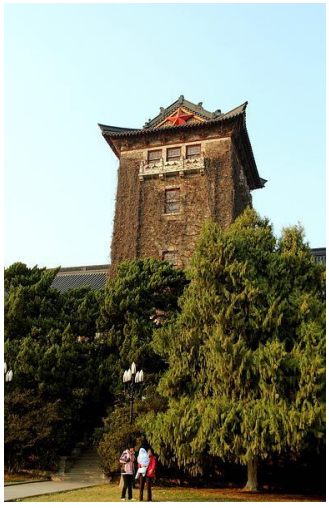
```python
# 检索
# 计算相似度
# 每个query存储相似度最高的三个
results = {}
for query_name, query_histogram in query_histograms.items():
    similarities = []
    for database_name, database_histogram in database_histograms.items():
        # 相似度计算使用欧式距离
        similarity = np.linalg.norm(query_histogram - database_histogram)
        similarities.append((database_name, similarity))
    # 按相似度排序并取前三个
    similarities.sort(key=lambda x: x[1])
```

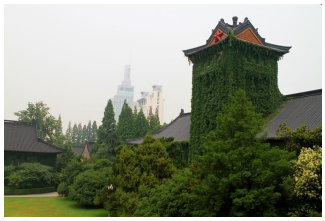
```
    results[query_name] = similarities[:3]
    # 后三个保存下相似度最小的三个，主要是对比下数值
    similarities.sort(key=lambda x: x[1], reverse=True)
    results[query_name] += similarities[:3]

# 打印结果
for query_name, top_matches in results.items():
    print(f"Query Image: {query_name}")
    # 前三个是相似度最高（也就是欧式距离最小）的
    print("Top 3 Matches:")
    for i, (match_name, similarity) in enumerate(top_matches[:3], start=1):
        print(f"  {i}. {match_name} - Similarity: {similarity:.2f}")
    # 后三个是相似度最低（也就是欧式距离最大）的
    print("Bottom 3 Matches:")
    for i, (match_name, similarity) in enumerate(top_matches[3:], start=1):
        print(f"  {i}. {match_name} - Similarity: {similarity:.2f}")
```

2.3 结果

程序运行结果如下

```
Query Image: q_1.jpg
Top 3 Matches:
  1. s_32.jpg - Similarity: 0.12
  2. s_10.jpg - Similarity: 0.16
  3. s_40.jpg - Similarity: 0.16
Bottom 3 Matches:
  1. s_24.jpg - Similarity: 0.39
  2. s_22.jpg - Similarity: 0.38
  3. s_25.jpg - Similarity: 0.34
Query Image: q_2.jpg
Top 3 Matches:
  1. s_14.jpg - Similarity: 0.10
  2. s_16.jpg - Similarity: 0.12
  3. s_8.jpg - Similarity: 0.12
Bottom 3 Matches:
  1. s_23.jpg - Similarity: 0.32
  2. s_22.jpg - Similarity: 0.30
  3. s_24.jpg - Similarity: 0.30
Query Image: q_3.jpg
Top 3 Matches:
  1. s_30.jpg - Similarity: 0.08
  2. s_15.jpg - Similarity: 0.10
  3. s_3.jpg - Similarity: 0.11
Bottom 3 Matches:
  1. s_24.jpg - Similarity: 0.30
  2. s_18.jpg - Similarity: 0.27
  3. s_22.jpg - Similarity: 0.27
Query Image: q_4.jpg
Top 3 Matches:
  1. s_32.jpg - Similarity: 0.09
  2. s_40.jpg - Similarity: 0.10
  3. s_36.jpg - Similarity: 0.11
Bottom 3 Matches:
  1. s_24.jpg - Similarity: 0.34
  2. s_22.jpg - Similarity: 0.33
  3. s_23.jpg - Similarity: 0.30
Query Image: q_5.jpg
Top 3 Matches:
  1. s_43.jpg - Similarity: 0.09
  2. s_13.jpg - Similarity: 0.10
  3. s_9.jpg - Similarity: 0.10
Bottom 3 Matches:
  1. s_18.jpg - Similarity: 0.33
  2. s_12.jpg - Similarity: 0.33
  3. s_39.jpg - Similarity: 0.32

Process finished with exit code 0
```

query	top1	top2	top3
q1	32	10	40
			
q2	14	16	8
			
q3	30	15	3
			
q4	32	40	36
			
q5	43	13	9

query	top1	top2	top3
			

3 Q3

要求说明

选择SIFT特征，重复问题2。

3.1 SIFT特征抽取代码的来源及说明

参考了OpenCV官方文档和基于GPT，使用了`cv2.SIFT_create()`相关的api，代码如下

```
def extract_sift_features(image_path):
    # 读取图像
    image = cv2.imread(image_path)
    if image is None:
        raise ValueError(f"无法读取图像文件: {image_path}")
    # 需要转换为灰度图像
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # 初始化SIFT检测器
    sift = cv2.SIFT_create()
    # 检测关键点并计算描述子
    keypoints, descriptors = sift.detectAndCompute(gray, None)
    if descriptors is None:
        descriptors = np.array([])
    return descriptors


# 处理文件夹中的所有图片
def process_images_sift(folder_path):
    sift_features = {}
    for filename in os.listdir(folder_path):
        file_path = os.path.join(folder_path, filename)
        if os.path.isfile(file_path) and file_path.lower().endswith(('.png', '.jpg',
'.jpeg', '.bmp', '.tiff')):
            try:
                sift_features[filename] = extract_sift_features(file_path)
            except ValueError as e:
                print(e)
    return sift_features


```

```
# 完成图片SIFT特征的提取
query_sift_features = process_images_sift('./images/query/')
database_sift_features = process_images_sift('./images/database/')
```

3.2 SIFT特征匹配代码的来源及说明

这部分计算一开始直接沿用了之前的相似度计算发现是不行的，因为不同图像的关键点数量可能不同，因此每张图像的描述符集合的大小（关键点个数）是不一样的，因此采用了暴力匹配（Brute-Force Matcher, BFMatcher）方法，将查询图像的描述符与数据库图像的描述符进行匹配。BFMatcher可以计算两个特征向量之间的距离，并找到最佳匹配点对。

1. **bf.knnMatch**为每个图像的每个query_descriptor对于一张database中的图像，查找两个最近的database_descriptor

2. 然后进行**比值测试**，目的是为了判断当前最佳匹配项 `m` 是否足够优秀，是否与次佳匹配项 `n` 有足够大的差距。**这种比较的目的是为了过滤掉那些可能是噪声或错误匹配的次优匹配，就是最近的那个其实也有可能并不匹配，只是“矮子里面拔高个”，比较勉强，因此需要进行比较看下是否真正足够匹配。但其实也并非能完全保证，只是减少一些误判。**在SIFT特征匹配中，由于场景复杂性和视角变化，可能会出现一些误匹配的情况，即使最佳匹配也不是绝对可靠的。因此，我们使用比值测试的方法来判断当前最佳匹配是否足够可靠，只有当最佳匹配的距离远远小于次佳匹配的距离时，我们才认为它是一个可靠的匹配。这个比值测试的阈值通常设置在0.5到0.8之间，这里设置为0.75。通过比值测试的 `m` 可以加入到good_matches中

3. 最终相似度的比较是按照**good_matches的数量**进行排序，越多则认为匹配度越高

```
# 检索
# 计算相似度
# 每个query存储相似度最高的三个
results = {}

# 使用暴力匹配器
bf = cv2.BFMatcher()

for query_name, query_descriptors in query_sift_features.items():
    similarities = []
    for database_name, database_descriptors in database_sift_features.items():
        if len(query_descriptors) == 0 or len(database_descriptors) == 0:
            continue
        # 使用KNN进行匹配，每个描述符找两个最近的邻居
        matches = bf.knnMatch(query_descriptors, database_descriptors, k=2)
        # 应用比值测试
        good_matches = []
        for m, n in matches:
            if m.distance < 0.75 * n.distance:
                good_matches.append(m)
        # 以好的匹配数量作为相似度度量
        similarity = len(good_matches)
        similarities.append((database_name, similarity))
    # 按相似度排序并取前三个
    similarities.sort(key=lambda x: x[1], reverse=True)
    results[query_name] = similarities[:3]
    # 后三个保存下相似度最小的三个，主要是对比下数值
    similarities.sort(key=lambda x: x[1])
    results[query_name] += similarities[:3]

# 打印结果
```

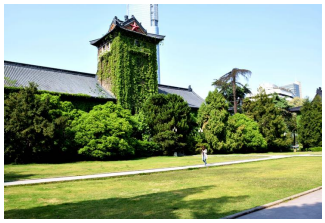
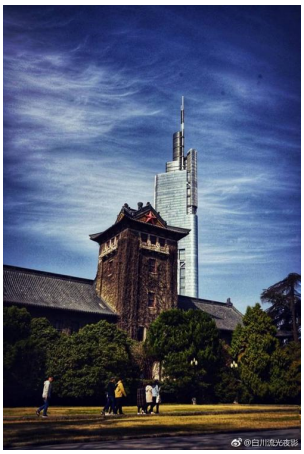
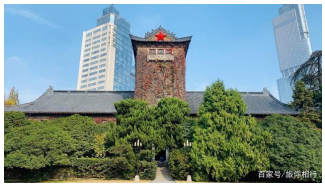
```python
for query_name, top_matches in results.items():
    print(f"Query Image: {query_name}")
    # 前三个是相似度最高（也就是匹配点最多）的
    print("Top 3 Matches:")
    for i, (match_name, similarity) in enumerate(top_matches[:3], start=1):
        print(f"  {i}. {match_name} - Similarity: {similarity}")
    # 后三个是相似度最低（也就是匹配点最少）的
    print("Bottom 3 Matches:")
    for i, (match_name, similarity) in enumerate(top_matches[3:], start=1):
        print(f"  {i}. {match_name} - Similarity: {similarity}")
```

3.3 结果

程序执行结果如下

```
Query Image: q_1.jpg
Top 3 Matches:
  1. s_29.jpg - Similarity: 170
  2. s_28.jpg - Similarity: 158
  3. s_37.jpg - Similarity: 81
Bottom 3 Matches:
  1. s_2.jpg - Similarity: 8
  2. s_26.jpg - Similarity: 9
  3. s_43.jpg - Similarity: 10
Query Image: q_2.jpg
Top 3 Matches:
  1. s_20.jpg - Similarity: 39
  2. s_29.jpg - Similarity: 24
  3. s_12.jpg - Similarity: 23
Bottom 3 Matches:
  1. s_16.jpg - Similarity: 3
  2. s_39.jpg - Similarity: 4
  3. s_43.jpg - Similarity: 4
Query Image: q_3.jpg
Top 3 Matches:
  1. s_6.jpg - Similarity: 75
  2. s_37.jpg - Similarity: 56
  3. s_24.jpg - Similarity: 53
Bottom 3 Matches:
  1. s_2.jpg - Similarity: 9
  2. s_22.jpg - Similarity: 16
  3. s_43.jpg - Similarity: 16
Query Image: q_4.jpg
Top 3 Matches:
  1. s_29.jpg - Similarity: 59
  2. s_44.jpg - Similarity: 52
  3. s_42.jpg - Similarity: 46
Bottom 3 Matches:
  1. s_30.jpg - Similarity: 2
  2. s_2.jpg - Similarity: 5
  3. s_34.jpg - Similarity: 6
Query Image: q_5.jpg
Top 3 Matches:
  1. s_5.jpg - Similarity: 44
  2. s_33.jpg - Similarity: 38
  3. s_4.jpg - Similarity: 34
Bottom 3 Matches:
  1. s_35.jpg - Similarity: 8
  2. s_38.jpg - Similarity: 9
  3. s_22.jpg - Similarity: 10

Process finished with exit code 0
```

注，topk列如 29(170)代表s_29.jpg，与相对应query相似度计算是170

query	top1	top2	top3
q1	29(170)	28(158)	37(81)
			
q2	20(39)	29(24)	12(23)
			
q3	6(75)	37(56)	24(53)
			
q4	29(59)	44(52)	42(46)
			
q5	5(44)	33(38)	4(34)

query	top1	top2	top3
			

4 Q4

要求说明

将问题2和问题3的结果进行比较和分析。

4.1 方法介绍与结果对比

4.1.1 全局RGB颜色直方图

全局RGB颜色直方图是一种简单的图像特征表示方法，通过计算图像中每个颜色通道的颜色分布来实现。

从Q2的结果来看，我们可以观察到一个特点是——query和top3的背景颜色（在此场景下是”天空”）往往是相似的，这可以推测是因为像天空，草地这样的背景占据了画面的大部分，所占的比重也比较大，因此计算欧氏距离的时候比如均为蓝天的和均为白天的这种相似度就会比较高，导致忽略了对于画面主体内容的匹配，因而会把一些看起来风格迥异的“北大楼”给匹配上。

4.1.2 SIFT特征匹配

SIFT特征匹配则更加复杂和精细，它不仅关注颜色信息，还包括图像中关键点的位置和局部特征描述。这种方法更适用于处理视角变化较大或图像中有遮挡的情况。从Q3的结果来看，SIFT特征匹配在一定程度上减少了背景颜色的干扰，并能更准确地匹配与查询图像相似的图像。

可以看到应用SIFT对于q4的检索效果是非常不错的，其他的几个在matches数较高时即使是不同角度拍摄的但是主体内容相同的图片也能匹配上。

4.2 分析与讨论

- **受干扰的情况：** RGB直方图对背景色高度敏感，例如在图像中有大片统一颜色的背景时，这些背景色会在直方图中占主导地位，从而影响匹配的准确性。相比之下，SIFT通过关键点和局部特征描述子，能更好地捕捉图像内容的本质特征，从而提高匹配的准确度和鲁棒性。
- **适用场景：** RGB直方图可能更适合于颜色分布均匀且无复杂背景干扰的图像。而SIFT特征则更适合复杂场景下的图像检索，特别是当图像中包含多变的视角和动态变化时。
- **性能考虑：** 尽管SIFT提供了更高的匹配准确性，但其**计算复杂度**也相对较高。在实际运行过程中，虽然没有对时间进行一个量化，但能明显感受到在使用SIFT计算的时候运行时间明显长于RGB直方图计算（以至于以为是不是哪里代码写错了）

4.3 总结

在选择合适的图像检索技术时，必须考虑到实际应用中的具体场景和需求。RGB直方图提供了一种计算上非常高效的方法，但在处理复杂或多变的图像内容时可能不足够。而SIFT特征则提供了一种高精度的替代方案，尤其适合在要求高准确性的场合使用。未来的研究可以考虑如何融合这两种技术，以达到既快速又准确的图像检索效果。

比如说不定照片主体并不是主要的检索目标，或者说大抵相同，关键的反而是背景的话，RGB直方图其实也可以用的，并不是就没法使用了，还得看检索需求。