Lab2-SIFT尺度不变特征变换

1. 实验概览

1.1 实验目的

本实验旨在理解并掌握 SIFT（尺度不变特征变换）算法的原理与实现方法，通过自编程实现多尺度 Harris 角点检测结合自定义 SIFT 描述子的方法，并与 OpenCV 自带 SIFT 算法进行匹配效果对比，从而加深对特征提取、描述及匹配机制的理解。

1.2 实验原理

1. SIFT 算法原理 SIFT（Scale-Invariant Feature Transform）由 David Lowe 提出，是一种在尺度空间中提取稳定关键点并计算其描述子的算法。SIFT 特征具有旋转、尺度及亮度变化不变性。其核心步骤包括：(1) 检测尺度空间极值点（通常通过高斯差分 DoG 实现）；(2) 精确定位关键点并剔除低对比度点；(3) 为关键点分配主方向以获得旋转不变性；(4) 计算关键点邻域内的梯度方向直方图以形成描述子。
2. Harris 角点检测 Harris 角点检测通过计算图像局部灰度变化矩阵的特征值，检测强度变化显著的像素点作为角点。在本实验中，为实现多尺度鲁棒性，使用图像金字塔在多个缩放层次上提取角点。

2. 算法设计与实现

整体流程如下：

1. `multi_scale_harris`：基于`goodFeaturesToTrack`的多尺度 Harris 检测，构建高斯金字塔，使用多尺度 Harris 算法提取关键点。

   · 在构建高斯金字塔时，首先确定金字塔的层数`levels`和缩放比例`scale_factor`，然后使用`cv2.resize()`函数进行图像的缩放（这里仅对图像做了下缩放），然后将图像存入金字塔中，得到高斯金字塔。其中`cv2.resize`函数中插值方法使用了`cv2.INTER_LANCZOS4`的Lanczos插值方法以做到高质量的图像缩放：

```
def multi_scale_harris(img, levels=8, scale_factor=0.75,
                       max_corners=10000, quality_level=0.00001, min_distance=2,
k=0.04):
    pyramid = [img]
    keypoints = []

    # 构建图像金字塔
    for _ in range(1, levels):
        h, w = pyramid[-1].shape[:2]
        new_img = cv2.resize(pyramid[-1], (int(w * scale_factor), int(h *
scale_factor)), interpolation=cv2.INTER_LINEAR)
        pyramid.append(new_img)
```

· 对高斯金字塔中每一层的图像转换成灰色图像，然后使用`cv2.goodFeaturesToTrack()`获取Harris角点，得到每一层的角点坐标。

```
    # 遍历每一层
    for level, im in enumerate(pyramid):
        gray = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
        gray = np.float32(gray)

        # 使用 goodFeaturesToTrack 获取 Harris 角点
        corners = cv2.goodFeaturesToTrack(
            gray,
            maxCorners=max_corners,
            qualityLevel=quality_level,
            minDistance=min_distance,
            useHarrisDetector=True,
            k=k
        )
```

·将在不同金字塔层上检测到的 Harris 角点坐标映射回原图坐标系，并生成对应的`cv2.KeyPoint`对象，保存到总关键点列表中。

```
        scale = 1 / (scale_factor ** level)  # 金字塔缩放回原图大小

        if corners is not None:
            for pt in corners:
                x, y = pt.ravel()
                keypoints.append(cv2.KeyPoint(x * scale, y * scale, 3 * scale))
```

2. `compute_gradients`计算图像梯度与方向。其中高斯模糊`cv2.GaussianBlur`是前置预处理，减少噪声造成的“假角点”；使用Sobel算子，对关键点周围的16x16的区域进行梯度计算，得到梯度幅度`mag`和方向`angle`，并使用`np.rad2deg()`将方向转换到0-360度之间

```python
def compute_gradients(gray):
    gray = cv2.GaussianBlur(gray, (3, 3), 0.5)
    Ix = cv2.Sobel(gray, cv2.CV_32F, 1, 0, ksize=3)
    Iy = cv2.Sobel(gray, cv2.CV_32F, 0, 1, ksize=3)
    mag = np.sqrt(Ix ** 2 + Iy ** 2)
    angle = np.rad2deg(np.arctan2(Iy, Ix)) % 360
    return mag, angle
```

3. `assign_orientation` 在关键点 kp 的邻域内统计梯度方向直方图，找出该关键点的主方向（主方向用于把描述子旋转到物体坐标系，从而获得旋转不变性）。
    - 其中 `weight` 为**高斯空间权重**，靠近关键点的像素权重更大。这里 `sigma = 0.5 * radius`，所以权重是 `exp(-(r^2)/(2*sigma^2))`。选择 `sigma` 的目的是限制贡献主要来自关键点邻域中心。
    - `np.argmax(histogram)` 返回直方图最大值索引，`* bin_width` 转换为角度。
    - 返回 **关键点的主方向**。

```python
def assign_orientation(mag, angle, kp, radius=18, num_bins=36):
    x, y = kp.pt

    sigma = 0.5 * radius
    weight = np.exp(-(i ** 2 + j ** 2) / (2 * sigma ** 2))

    main_angle = np.argmax(histogram) * bin_width
    return main_angle
```

4. `compute_sift_descriptor` 计算 SIFT 描述子（旋转对齐 + 双线性插值 + 高斯加权 + 阈值截断 + 双归一化）。

```
def compute_sift_descriptor(mag, angle, kp, patch_size=16, num_bins=8,
clip_val=0.25):

  main_angle = assign_orientation(mag, angle, kp)
  cos_t, sin_t = np.cos(np.radians(main_angle)), np.sin(np.radians(main_angle))

  for i in range(-half, half):
      for j in range(-half, half):
          # 旋转对齐
          rot_x = j * cos_t - i * sin_t
          rot_y = j * sin_t + i * cos_t
          xx, yy = int(x + rot_x), int(y + rot_y)

          if 0 <= xx < mag.shape[1] and 0 <= yy < mag.shape[0]:
              m = mag[yy, xx]
              a = (angle[yy, xx] - main_angle) % 360

              # 高斯加权
              weight = np.exp(-(rot_x**2 + rot_y**2) / (2 * sigma**2))
              m_weighted = m * weight
```

把当前像素分配到 4×4 的空间网格，每个像素不只影响一个 cell，而是根据距离分布到相邻 4 个 cell（双线性插值）。

```
cell_size = patch_size / 4.0
bx_f = (rot_x + half) / cell_size
by_f = (rot_y + half) / cell_size
bx0 = int(np.floor(bx_f))
by0 = int(np.floor(by_f))
dx = bx_f - bx0
dy = by_f - by0
# 空间权重四个（双线性）
w = [(1-dx)*(1-dy), dx*(1-dy), (1-dx)*dy, dx*dy]
bx_idx = [bx0, bx0+1, bx0, bx0+1]
by_idx = [by0, by0, by0+1, by0+1]
```

对每个像素：

根据空间插值权重 w[ii]，方向插值权重 wb，高斯权重 weight，梯度幅值 m； → 综合累加到对应的直方图 bin。

结果：每个 cell（4×4）都有一个方向直方图（8 维）。

```
for ii in range(4):
    bx_i, by_i = bx_idx[ii], by_idx[ii]
    if 0 <= bx_i < 4 and 0 <= by_i < 4:
    for bb, wb in zip(bin_idx, w_dir):
    desc[by_i, bx_i, bb] += m_weighted * w[ii] * wb
```

5. match_features `对两张图像的 SIFT 描述子集合` desc1 `和` desc2` 进行匹配，
   返回通过 **Lowe 比率检测 (Lowe's ratio test)** 筛选后的匹配点索引对。

   - `BFMatcher` = **Brute Force Matcher（暴力匹配器），工作机制：对 `desc1` 中的每个特征，依次计算它与 `desc2` 中所有特征的距离，使用欧氏距离（`NORM_L2`）比较两个描述子的相似程度，由于 SIFT 描述子已做归一化，因此欧氏距离可以直接反映特征相似度。

```
def match_features(desc1, desc2, ratio=0.75):
    bf = cv2.BFMatcher(cv2.NORM_L2)
    matches = bf.knnMatch(desc1.astype(np.float32), desc2.astype(np.float32), k=2)
```

   - David Lowe 在原始 SIFT 论文中提出：

     > "如果第一匹配的距离明显小于第二匹配的距离，则该匹配可信。"

     换句话说：
     若一个特征在另一图像中找到了两个候选匹配点，
     只有当最优匹配 **明显优于** 第二优匹配时，
     才认为它是"稳定匹配"。

```
for m, n in matches:
    if m.distance < ratio * n.distance:
        good.append((m.trainIdx, m.queryIdx))
```

6. `custom_sift_match`

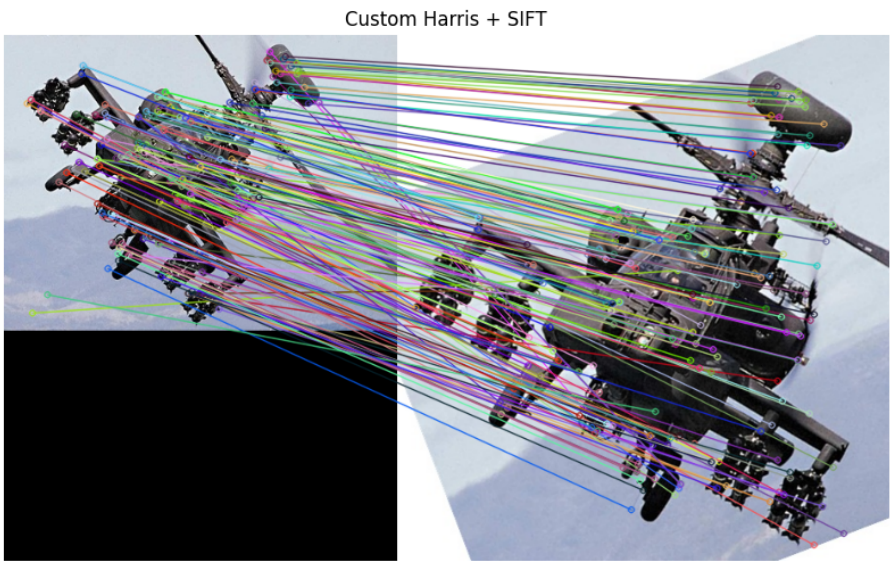
   - 读入目标图（target），提取 keypoints + 描述子（descs_t）。
   - 遍历数据集里的每张图：
     - 提取 keypoints + 描述子（descs）。
     - 用 `match_features` 对 `descs_t` 与 `descs` 做匹配并计数。
     - 选择匹配数最多的一张作为 `best_img`。
   - 把 `best_img`（左）和 `target`（右）及匹配线画出并保存。

7. `opencv_sift_match` 使用OpenCV自带的SIFT算法实现上述步骤。

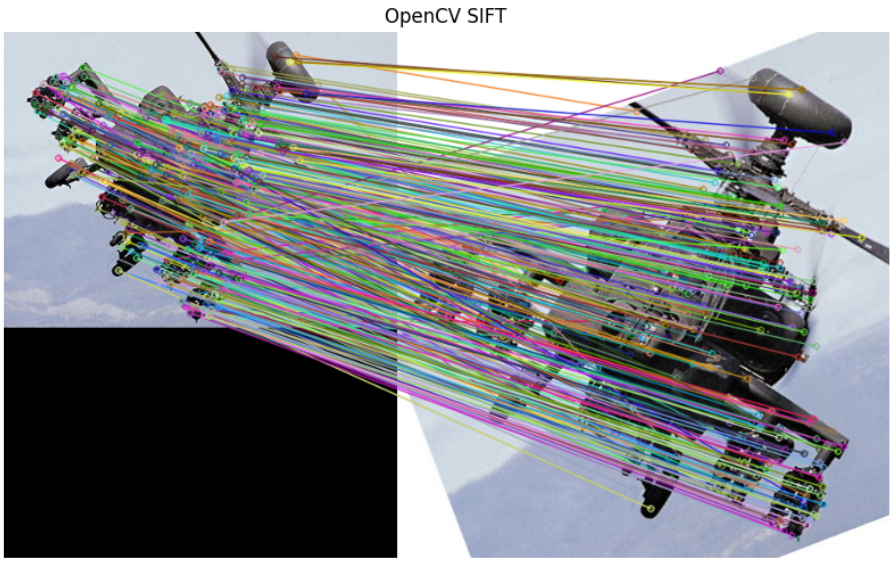
3. 运行结果

```
开始自实现SIFT匹配...
1.jpg: 匹配数量 15
2.jpg: 匹配数量 27
3.jpg: 匹配数量 191
4.jpg: 匹配数量 47
5.jpg: 匹配数量 17
匹配结果已保存!
开始OpenCV SIFT匹配...
1.jpg: OpenCV匹配数量 55
2.jpg: OpenCV匹配数量 53
3.jpg: OpenCV匹配数量 421
4.jpg: OpenCV匹配数量 72
5.jpg: OpenCV匹配数量 39
OpenCV SIFT匹配结果已保存
所有匹配结果已保存至 output 文件夹
```

- 使用自己实现的SIFT算法进行特征点匹配，得到的最佳匹配结果如下图所示，匹配上的关键点对数为191：



- 使用OpenCV中的SIFT算法进行特征点匹配，得到的最佳匹配结果如下图所示，匹配上的关键点对数为421:



4. 实验感想

4.1. 实验收获

通过本次实验，我对SIFT算法有了更深入的了解，掌握了SIFT算法的基本原理和实现方法，学会了如何使用Python和OpenCV实现SIFT算法，并与OpenCV中的SIFT算法进行对比，从中学到了很多知识和技巧，收获颇丰。

4.2. 实验困难

在实验过程中，我遇到了一些困难，主要包括：

- SIFT算法的实现比较复杂，需要对图像金字塔、Harris角点检测、SIFT描述子等多个方面有深入的了解，因此在实现过程中花费了大量的时间和精力来调整算法和参数，最终得到的匹配效果也不是很理想。
- 在特征点匹配的过程中，匹配算法的选择和参数的设置对匹配效果有很大的影响，我尝试了OpenCV中的`BFMatcher()`暴力匹配器进行knn匹配，并进行比值测试，但是最终结果并不理想，代码运行时间较长。
- 在图像金字塔上，不同层的特征点实际覆盖区域不同，如果 patch_size 为固定值会出现在边缘采样不足的情况，得到的匹配点数异常的少。后来尝试让keypoint.size随层缩放，desc = compute_sift_descriptor(mag, ang, kp, **patch_size=int(4 * kp.size)**)，可以使每个特征点的描述子窗口大小与尺度匹配。

4.3. 实验改进

如果可以对自己实现的SIFT算法进行改进，我觉得可以从以下几个方面进行改进：

- 对SIFT算法的实现进行优化，可以使用更加复杂的高斯金字塔和其他的关键点检测方法，以及更加复杂的SIFT描述子计算方法，提高匹配效果。
- 在特征点匹配的过程中，可以使用更加高效的匹配算法，如FLANN匹配器，来提高匹配效果，同时缩短匹配所需要的时间。
- 进一步优化参数的设置，找到最佳参数设置来提高匹配效果。

总的来说，本次实验是一次很好的实践机会，通过实现SIFT算法并与OpenCV中的SIFT算法进行对比，我对SIFT算法有了更深入的了解，并学会了如何使用Python和OpenCV实现SIFT算法。

5. 附录：代码

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


# ================================
# 基于 goodFeaturesToTrack 的多尺度 Harris 角点检测
# ================================
def multi_scale_harris(img, levels=6, scale_factor=0.75,
                       max_corners=9000, quality_level=0.00001,
                       min_distance=3, k=0.04, border_size=16):
    pyramid = [img]
    keypoints = []

    # ===== 构建图像金字塔 =====
    for _ in range(1, levels):
        h, w = pyramid[-1].shape[:2]
        new_img = cv2.resize(
            pyramid[-1],
            (int(w * scale_factor), int(h * scale_factor)),
            interpolation=cv2.INTER_LINEAR
        )
        pyramid.append(new_img)

    h0, w0 = img.shape[:2]  # 原图尺寸，用于边界判断

    # ===== 遍历每一层，提取角点 =====
    for level, im in enumerate(pyramid):
        gray = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
        gray = np.float32(gray)

        corners = cv2.goodFeaturesToTrack(
            gray,
            maxCorners=max_corners,
            qualityLevel=quality_level,
            minDistance=min_distance,
            useHarrisDetector=True,
            k=k
        )

        scale = 1 / (scale_factor ** level)  # 还原到原图比例

        if corners is not None:
            for pt in corners:
                x, y = pt.ravel()
                x_scaled, y_scaled = x * scale, y * scale

                # 边界过滤：保证计算 descriptor 时不会越界
                if (border_size <= x_scaled < w0 - border_size and
```

```
                    border_size <= y_scaled < h0 - border_size):
                    keypoints.append(cv2.KeyPoint(x_scaled, y_scaled, 3 * scale))

    return keypoints



# ================================
# 梯度与方向图
# ================================
def compute_gradients(gray):
    gray = cv2.GaussianBlur(gray, (3, 3), 0.5)
    Ix = cv2.Sobel(gray, cv2.CV_32F, 1, 0, ksize=3)
    Iy = cv2.Sobel(gray, cv2.CV_32F, 0, 1, ksize=3)
    mag = np.sqrt(Ix ** 2 + Iy ** 2)
    angle = np.rad2deg(np.arctan2(Iy, Ix)) % 360
    return mag, angle


# ================================
# 为关键点分配主方向
# ================================
def assign_orientation(mag, angle, kp, radius=18, num_bins=36):
    x, y = kp.pt
    h, w = mag.shape
    histogram = np.zeros(num_bins, dtype=np.float32)
    bin_width = 360 / num_bins
    for i in range(-radius, radius + 1):
        for j in range(-radius, radius + 1):
            xx, yy = int(x + j), int(y + i)
            if 0 <= xx < w and 0 <= yy < h:
                m = mag[yy, xx]
                a = angle[yy, xx]
                sigma = 0.5 * radius
                weight = np.exp(-(i ** 2 + j ** 2) / (2 * sigma ** 2))
                bin_idx = int(a // bin_width) % num_bins
                histogram[bin_idx] += m * weight
    main_angle = np.argmax(histogram) * bin_width
    return main_angle


# ================================
# 描述子计算（旋转对齐 + 双线性插值 + 高斯加权 + 阈值截断 + 双归一化）
# ================================
def compute_sift_descriptor(mag, angle, kp, patch_size=16, num_bins=8, clip_val=0.2):
    x, y = kp.pt
    main_angle = assign_orientation(mag, angle, kp)
    cos_t, sin_t = np.cos(np.radians(main_angle)), np.sin(np.radians(main_angle))
    half = patch_size // 2
    desc = np.zeros((4, 4, num_bins), dtype=np.float32)
    bin_width = 360.0 / num_bins
```

```
    sigma = 0.5 * patch_size

    for i in range(-half, half):
        for j in range(-half, half):
            # 旋转对齐（注意 i->y, j->x）
            rot_x = j * cos_t - i * sin_t
            rot_y = j * sin_t + i * cos_t
            xx_f = x + rot_x
            yy_f = y + rot_y

            # 边界检查（注意使用 floor/ceil 时保持安全）
            if not (0 <= xx_f < mag.shape[1] and 0 <= yy_f < mag.shape[0]):
                continue

            # 推荐：对 mag 做双线性插值会更好。这里先用 nearest
            xx, yy = int(xx_f), int(yy_f)
            m = mag[yy, xx]
            a = (angle[yy, xx] - main_angle) % 360

            # 高斯权重
            weight = np.exp(-(rot_x**2 + rot_y**2) / (2 * sigma**2))
            m_weighted = m * weight

            # 浮点小块坐标（修正：不要先 int）
            cell_size = patch_size / 4.0
            bx_f = (rot_x + half) / cell_size
            by_f = (rot_y + half) / cell_size

            bx0 = int(np.floor(bx_f))
            by0 = int(np.floor(by_f))
            dx = bx_f - bx0
            dy = by_f - by0

            # 空间权重四个（双线性）
            w = [(1-dx)*(1-dy), dx*(1-dy), (1-dx)*dy, dx*dy]
            bx_idx = [bx0, bx0+1, bx0, bx0+1]
            by_idx = [by0, by0, by0+1, by0+1]

            # 方向插值（线性，带环绕）
            bin_f = a / bin_width
            b0 = int(np.floor(bin_f)) % num_bins
            b1 = (b0 + 1) % num_bins
            wb1 = bin_f - np.floor(bin_f)
            wb0 = 1.0 - wb1
            bin_idx = [b0, b1]
            w_dir = [wb0, wb1]

            # 累加
            for ii in range(4):
                bx_i, by_i = bx_idx[ii], by_idx[ii]
                if 0 <= bx_i < 4 and 0 <= by_i < 4:
```

```
                    for bb, wb in zip(bin_idx, w_dir):
                        desc[by_i, bx_i, bb] += m_weighted * w[ii] * wb

    # 展平 + 截断 + 双归一化
    desc = desc.flatten()
    norm = np.linalg.norm(desc) + 1e-7
    desc = desc / norm
    desc = np.clip(desc, 0, clip_val)
    norm2 = np.linalg.norm(desc) + 1e-7
    desc = desc / norm2
    return desc



# ================================
# 特征匹配（BFMatcher）
# ================================
def match_features(desc1, desc2, ratio=0.75):
    bf = cv2.BFMatcher(cv2.NORM_L2)
    matches = bf.knnMatch(desc1.astype(np.float32), desc2.astype(np.float32), k=2)
    good = []
    for m, n in matches:
        if m.distance < ratio * n.distance:
            good.append((m.trainIdx, m.queryIdx))
    return good


# ================================
# 自实现 Harris + SIFT 描述子匹配
# ================================
def custom_sift_match(target_path, dataset_dir, output_dir):
    os.makedirs(output_dir, exist_ok=True)

    target = cv2.imread(target_path)
    gray_t = cv2.cvtColor(target, cv2.COLOR_BGR2GRAY)
    kps_t = multi_scale_harris(target)
    mag_t, ang_t = compute_gradients(gray_t)
    descs_t = np.array([compute_sift_descriptor(mag_t, ang_t, kp, patch_size=int(4 *
kp.size)) for kp in kps_t])

    best_img, best_score, best_kps, best_matches = None, 0, None, None

    for name in os.listdir(dataset_dir):
        img_path = os.path.join(dataset_dir, name)
        img = cv2.imread(img_path)
        if img is None:
            continue
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        kps = multi_scale_harris(img)
        mag, ang = compute_gradients(gray)
        descs = np.array([compute_sift_descriptor(mag, ang, kp, patch_size=int(4 *
```

```
kp.size)) for kp in kps])

        matches = match_features(descs_t, descs)
        print(f"{name}: 匹配数量 {len(matches)}")

        if len(matches) > best_score:
            best_img, best_score, best_kps, best_matches = img, len(matches), kps,
matches

    if best_img is not None:
        matched_img = cv2.drawMatches(
            best_img, best_kps,
            target, kps_t,
            [cv2.DMatch(_queryIdx=i, _trainIdx=j, _imgIdx=0, _distance=0) for i, j in
best_matches],
            None, flags=2
        )
        # 统一使用 Matplotlib 保存为 RGB
        plt.figure(figsize=(12, 6))
        plt.imshow(cv2.cvtColor(matched_img, cv2.COLOR_BGR2RGB))
        plt.title("Custom Harris + SIFT")
        plt.axis('off')
        plt.savefig(os.path.join(output_dir, "custom_harris_sift.png"))
        plt.close()
        print("匹配结果已保存！")


# ================================
# OpenCV SIFT匹配（保持原样）
# ================================
def opencv_sift_match(target_path, dataset_dir, output_dir):
    target = cv2.imread(target_path)
    gray_target = cv2.cvtColor(target, cv2.COLOR_BGR2GRAY)
    sift = cv2.SIFT_create()
    kp_t, desc_t = sift.detectAndCompute(gray_target, None)

    best_img, best_score, best_matches, best_kp = None, 0, None, None
    bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck=False)

    for name in os.listdir(dataset_dir):
        img_path = os.path.join(dataset_dir, name)
        img = cv2.imread(img_path)
        if img is None:
            continue
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        kp, desc = sift.detectAndCompute(gray, None)
        matches = bf.knnMatch(desc_t, desc, k=2)
        good = [m for m, n in matches if m.distance < 0.75 * n.distance]
        print(f"{name}: OpenCV匹配数量 {len(good)}")
        if len(good) > best_score:
            best_img, best_score, best_matches, best_kp = img, len(good), good, kp
```

```
    if best_img is not None:
        reversed_matches = [
            cv2.DMatch(_queryIdx=m.trainIdx, _trainIdx=m.queryIdx, _imgIdx=m.imgIdx,
_distance=m.distance) for m in
            best_matches]
        result = cv2.drawMatches(best_img, best_kp, target, kp_t, reversed_matches,
None, flags=2)
        plt.figure(figsize=(12, 6))
        plt.imshow(cv2.cvtColor(result, cv2.COLOR_BGR2RGB))
        plt.title("OpenCV SIFT")
        plt.axis('off')
        plt.savefig(os.path.join(output_dir, "opencv_sift_result.png"))
        plt.close()
        print("OpenCV SIFT匹配结果已保存")


# ==============================
# 主函数
# ==============================
if __name__ == "__main__":
    cwd = os.path.dirname(__file__)
    dataset_dir = os.path.join(cwd, "dataset")
    target_path = os.path.join(cwd, "target.jpg")
    output_dir = os.path.join(cwd, "output")
    os.makedirs(output_dir, exist_ok=True)
    print("开始自实现SIFT匹配...")
    custom_sift_match(target_path, dataset_dir, output_dir)
    print("开始OpenCV SIFT匹配...")
    opencv_sift_match(target_path, dataset_dir, output_dir)
    print("所有匹配结果已保存至 output 文件夹")
```