机器视觉第七次作业

1、实现一个具有尺度不变特性的关键点检测器（编写伪代码）。

具有尺度不变特性的关键点检测器是一种能够在不同尺度下（即不同的图像分辨率或不同的模糊程度）识别并定位图像中显著特征的算法。这类检测器的核心目标是确保所检测到的关键点在图像的尺度变换（如缩放、模糊等）下依然能够被识别和描述，因此它们对于图像的旋转、尺度变化以及部分变形具有鲁棒性。

具体而言，检测器首先会通过构建尺度空间来对图像进行多尺度的分析。尺度空间是通过对图像进行多次不同程度的高斯模糊，然后对每个模糊后的图像进行下采样，生成多个不同尺度的图像版本。然后，在这个尺度空间中，检测器通过在每个尺度的图像上寻找局部极值点（即比周围像素更显著的点），这些极值点就是潜在的关键点。

为了进一步提高检测的准确性，检测器还会比较不同尺度的图像特征，避免在图像的边缘或噪声区域误检测到无关的点。当一个点在不同尺度上表现出稳定的显著性时，就被认为是一个有效的关键点。最终，这些关键点可以被用来进行图像匹配、物体识别、图像拼接等任务。这种方法的优势在于即使图像发生了缩放、旋转等变化，关键点仍能保持其特征，从而使得检测到的关键点可以用于各种图像处理任务中，比如物体跟踪、图像拼接或 3D 重建等。检测器伪代码如下：

```
function ScaleInvariant_KeyPoint_Detection(I, num_octaves, num_scales_per_octave, sigma, threshold):
    1. 初始化：
        // 将图像 I 转换为尺度空间特征：构建尺度空间
        scale_space = BuildScaleSpace(I, num_octaves, num_scales_per_octave, sigma)
        N = total number of pixels in I   // 图像的总像素数量
        keypoints = []   // 存储最终检测到的关键点
    2. 构建尺度空间：
        function BuildScaleSpace(I, num_octaves, num_scales_per_octave, sigma):
            scale_space = []
            for octave = 1 to num_octaves:
                octave_images = []
```

```
            for scale = 1 to num_scales_per_octave:
                blurred_image = ApplyGaussianBlur(I, sigma * (2 ^ (scale /
num_scales_per_octave)))
                octave_images.append(blurred_image)
            scale_space.append(octave_images)
            I = Downsample(octave_images[-1])   // 下采样图像，构建下一个金字
塔层
        return scale_space
```

3. 对每个像素点进行迭代检测：

```
for octave in scale_space:
    for scale_idx = 2 to len(octave) - 2:   // 避免边界
        for row = 2 to image_height - 2:
            for col = 2 to image_width - 2:
                // 提取当前像素点在尺度空间中的特征 M_i
                M_i = GetFeature(octave, scale_idx, row, col)
                // 检查当前点是否为极值点
                if IsExtrema(octave, scale_idx, row, col, threshold):
                    keypoints.append((row, col, scale_idx))
```

4. 判断是否为极值点：

```
function IsExtrema(octave, scale_idx, row, col, threshold):
    current_pixel = octave[scale_idx][row, col]
    neighbors = []
    // 在相邻尺度和空间位置的邻域内查找邻近像素
    for dscale in [-1, 0, 1]:
        for dx in [-1, 0, 1]:
            for dy in [-1, 0, 1]:
                if dscale == 0 and dx == 0 and dy == 0:
                    continue
                neighbor_row = row + dy
                neighbor_col = col + dx
```

```
                                neighbor_scale_idx = scale_idx + dscale
                                if Valid(neighbor_row, neighbor_col, neighbor_scale_idx,
octave):
    then    neighbors.append(octave[neighbor_scale_idx][neighbor_row, neighbor_col])
                    // 判断当前点是否为极值
    shift = CalculateShift(current_pixel, neighbors)
    return shift < threshold
        5. 聚类与标签分配：
            // 根据检测到的关键点，聚类相近的关键点并分配唯一标签
            clustered_keypoints = ClusterKeyPoints(keypoints)
        6. 映射标签到图像空间：
            // 将标签映射回原始图像像素空间
            labeled_image = MapKeyPointsToImage(clustered_keypoints, I)
        7. 返回分割后的关键点图像标签：
            return labeled_image
    end function
    // 其他辅助函数：
    function ApplyGaussianBlur(I, sigma):
        // 对图像应用高斯模糊，返回模糊后的图像
        return blurred_image
    function Downsample(I):
        // 对图像进行下采样
        return downsampled_image
    function GetFeature(octave, scale_idx, row, col):
        // 提取特定位置和尺度的特征
        return feature
    function CalculateShift(current_pixel, neighbors):
        // 计算当前点与邻近点之间的偏移量
        return shift
```

function Valid(row, col, scale_idx, octave):

// 检查当前坐标是否合法

return row >= 0 and col >= 0 and scale_idx >= 0

function ClusterKeyPoints(keypoints):

// 聚类相近的关键点

return clustered_keypoints

function MapKeyPointsToImage(keypoints, I):

// 将关键点标签映射回原始图像

return labeled_image

根据伪代码，我也编写了具体实现的代码进行测试，代码如图 1 所示，

```python
import numpy as np
import cv2
from scipy.ndimage import gaussian_filter

def build_scale_space(image, num_octaves, num_scales_per_octave, sigma=1.6):
    """
    构建尺度空间
    :param image: 输入图像
    :param num_octaves: 尺度空间的金字塔层数
    :param num_scales_per_octave: 每个金字塔层的尺度数
    :param sigma: 初始高斯模糊的标准差
    :return: 尺度空间
    """
    scale_space = []
    for octave in range(num_octaves):
        octave_images = []
        for scale in range(num_scales_per_octave):
            sigma_k = sigma * (2 ** (scale / num_scales_per_octave))
            blurred_image = gaussian_filter(image, sigma_k)
            octave_images.append(blurred_image)
        scale_space.append(octave_images)
        # 下采样图像，构建下一个金字塔层
        image = downsample(octave_images[-1])
    return scale_space

def downsample(image):
    """
    对图像进行下采样
    :param image: 输入图像
    :return: 下采样后的图像
    """
    return image[::2, ::2]  # 下采样：每隔一行一列取样

def detect_keypoints(scale_space, threshold=0.03):
    """
    检测尺度空间中的关键点
    :param scale_space: 尺度空间
    :param threshold: 极值点的阈值
    :return: 关键点坐标列表
    """
    keypoints = []
    for octave in scale_space:
        for scale_idx in range(1, len(octave) - 1):  # 避免处理边界
            for row in range(1, octave[scale_idx].shape[0] - 1):
                for col in range(1, octave[scale_idx].shape[1] - 1):
                    if is_extrema(octave, scale_idx, row, col, threshold):
                        keypoints.append((col, row, scale_idx))
    return keypoints
```

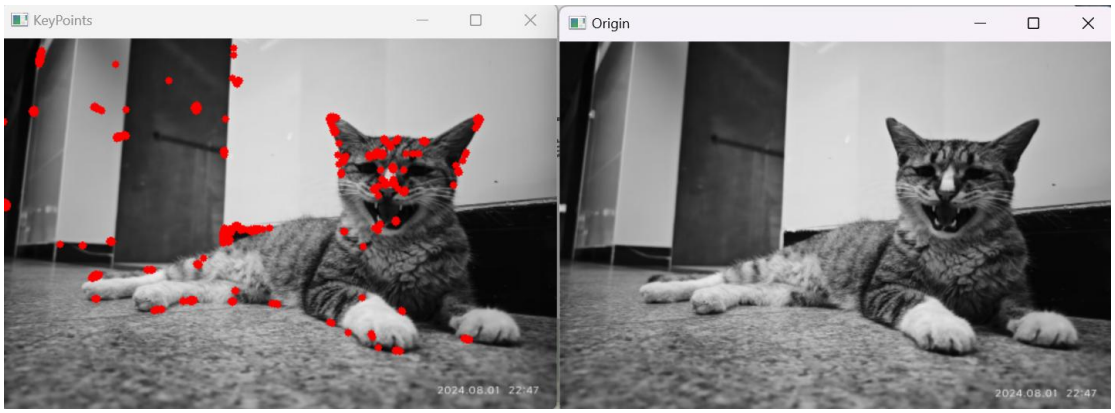
```python
def is_extrema(octave, scale_idx, row, col, threshold):
    """
    判断某个点是否为极值点
    :param octave: 当前金字塔层
    :param scale_idx: 当前尺度的索引
    :param row: 当前像素的行坐标
    :param col: 当前像素的列坐标
    :param threshold: 极值点阈值
    :return: 是否为极值点
    """
    current_pixel = octave[scale_idx][row, col]
    neighbors = []

    # 计算当前点与邻域的差异，包括相邻尺度和空间位置的邻域
    for dscale in [-1, 0, 1]:
        for dx in [-1, 0, 1]:
            for dy in [-1, 0, 1]:
                if dscale == 0 and dx == 0 and dy == 0:
                    continue
                neighbor_row = row + dy
                neighbor_col = col + dx
                neighbor_scale_idx = scale_idx + dscale
                if 0 <= neighbor_row < octave[scale_idx].shape[0] and 0 <= neighbor_col < octave[scale_idx].shape[1]:
                    if 0 <= neighbor_scale_idx < len(octave):
                        neighbors.append(octave[neighbor_scale_idx][neighbor_row, neighbor_col])

    # 判断当前点是否为极值
    shift = np.abs(current_pixel - np.mean(neighbors))
    return shift > threshold
```

图 1 关键点检测器

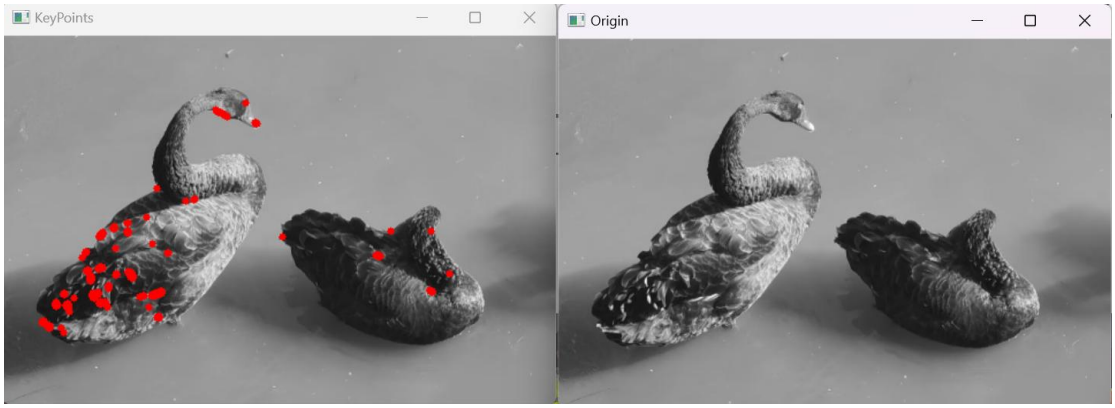
在多张图片的测试中，我发现当阈值设置小导致图像中识别到较多关键点时，尺度不变特性的关键点检测器的计算开销显著增加。原因在于，图像中的边缘点通常具有较强的灰度变化，在尺度空间中容易被误判为潜在的关键点，因此需要进行更多的计算来检查这些点是否为局部极值导致了计算量的增加，特别是在每个像素及其邻域进行多尺度比较时，边缘点的计算量会成倍增加。如图 2(a)和(b)，二者相互比较显然图 2(b)的关键点相对更少，所以所需要的迭代开销更少。

此外，图像尺寸越大，计算开销也会显著增加。较大的图像意味着更多的像素需要处理，尺度空间中的每一层和每个像素点的计算量都会增加，从而导致搜

索时间延长。因此，随着图像尺寸的增大，关键点检测所需的时间和内存消耗也会增加。



(a)



(b)

图 2 两张图片测试结果

2、阅读论文 David G. Lowe, Distinctive Image Features from Scale-Invariant Keypoints, IJCV 2004.

论文提出了一种从图像中提取独特不变特征的方法，这些特征可以用于在不同视角下进行可靠的匹配。这些特征对图像的尺度和旋转不变，并且能够在相当范围的仿射变形、三维视角变化、噪声添加和光照变化下提供鲁棒的匹配。这些特征具有高度的辨识度，即单个特征可以在来自多个图像的大型特征数据库中以高概率正确匹配。论文还描述了一种利用这些特征进行物体识别的方法。识别过程通过将单个特征与已知物体的特征数据库进行匹配，使用快速的最近邻算法，然后通过霍夫变换识别属于同一物体的特征簇，最后通过最小二乘解法验证一致的姿态参数。这种识别方法能够在杂乱和遮挡的情况下鲁棒地识别物体，并且实现近实时的性能。

该特征提取方式主要通过以下四步进行：

尺度空间极值检测：首先，通过对不同尺度和图像位置进行搜索，使用差分高斯函数（DoG）来检测潜在的兴趣点，这些点对尺度和方向具有不变性。

关键点定位：在每个候选位置上，通过拟合详细的模型来确定其精确位置和尺度。关键点通过衡量其稳定性来进行选择，以确保其可靠性。

方向分配：为每个关键点位置分配一个或多个方向，这些方向基于局部图像的梯度方向。之后，所有的操作都在变换后的图像数据上进行，这些数据相对于每个特征的分配方向、尺度和位置进行了变换，从而确保了对这些变换的鲁棒性。

关键点描述符：在选定的尺度下，测量关键点周围区域的局部图像梯度，并将其转化为一种描述符表示。这些描述符具有一定的抗光照变化和形状失真的能力，能够稳健地表达局部图像特征，便于匹配和识别。

其中，上述四个步骤的流程概括如图 1 所示：

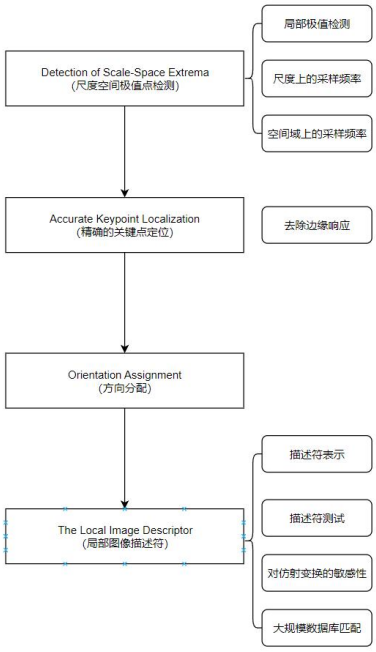


图 1 算法流程图

算法首先通过在不同尺度和图像位置上搜索，SIFT 检测图像中的局部极值点，这些点在图像的尺度变化下保持稳定。尺度空间的构建通过对图像进行多次高斯模糊来实现，每个尺度通过差分高斯（DoG）函数来检测潜在的兴趣点，从而找到在不同尺度下稳定的特征位置，确保对尺度不变性。

对于检测到的潜在特征点，SIFT 通过精确定位的方式进一步筛除对比度低或边缘响应强的点，确保只保留具有稳定性的关键点。这一过程通过最小二乘法对每个关键点的精确位置和尺度进行优化，从而提高定位精度。随后，SIFT 通过计算每个关键点的局部图像梯度方向来为其分配方向，使得这些关键点具有旋转不变性。通过这种方法，SIFT 能够处理图像中的旋转变化，保证特征点的稳定性。

在此基础上，SIFT 为每个关键点生成描述符。描述符通过计算关键点周围区域的梯度方向直方图来表示该区域的局部结构。这些描述符具备高度的辨识度，能够确保在大规模图像数据库中准确匹配到相同的特征。在匹配过程中，算法通过计算描述符之间的欧几里得距离找出最匹配的特征点，从而完成图像之间的匹配。并且为了提高匹配效率，SIFT 使用了最近邻搜索算法，同时对误匹配进行过滤，确保识别出的特征点具有较高的准确性。最后，SIFT 在匹配完成后，进

一步利用 RANSAC 等算法去除错误匹配，并通过霍夫变换等方法识别物体的位置。

该算法在实现的过程中主要分为构建尺度空间、检测空间极值点、关键点定位、尖酸关键点方向、计算描述符以及匹配特征，通过查找资料发现 opencv 中已经将这个算法集成到库函数中，所以使用了 opencv 对这个算法进行简单的复现，如图 2 所示。

```python
import cv2
import matplotlib.pyplot as plt
# 加载图像
image = cv2.imread('./test1.jpg', cv2.IMREAD_GRAYSCALE)
# 创建SIFT对象
sift = cv2.SIFT_create()
# 检测关键点和计算描述符
keypoints, descriptors = sift.detectAndCompute(image, None)
# 绘制关键点
image_with_keypoints = cv2.drawKeypoints(image, keypoints, None)
# 显示结果
plt.imshow(image_with_keypoints)
plt.title('SIFT Keypoints')
plt.show()
```

图 2 使用 opencv 库函数复现

首先通过 cv2.imread()加入图像并其转换为灰度图，因为 SIFT 算法通常在灰度图像上进行处理。随后通过 cv2.SIFT_create()创建 SIFT 对象，并使用 detectAndCompute()方法，这个方法不仅检测图像中的关键点，还会计算每个关键点的描述符。这一过程包括了尺度空间的构建和局部极值点的检测，也会自动去除低对比度或者边缘的关键点，并为每个关键点分配一个方向，从而保证旋转不变性。随后通过 sift.detectAndCompute(image, None)函数，代码在图像中检测到了这些关键点并计算出了相应的描述符，描述符是每个关键点局部区域的特征表示，通常用于图像匹配或物体识别。最后，使用 cv2.drawKeypoints 将检测到的关键点绘制到原图上，并通过 Matplotlib 显示出来，从而可以直观地看到图像中的关键点位置。在后续测试中输入了两张图片，如图 3 所示，通过 SIFT 算法将两张图片的关键点提取并标志了出来。

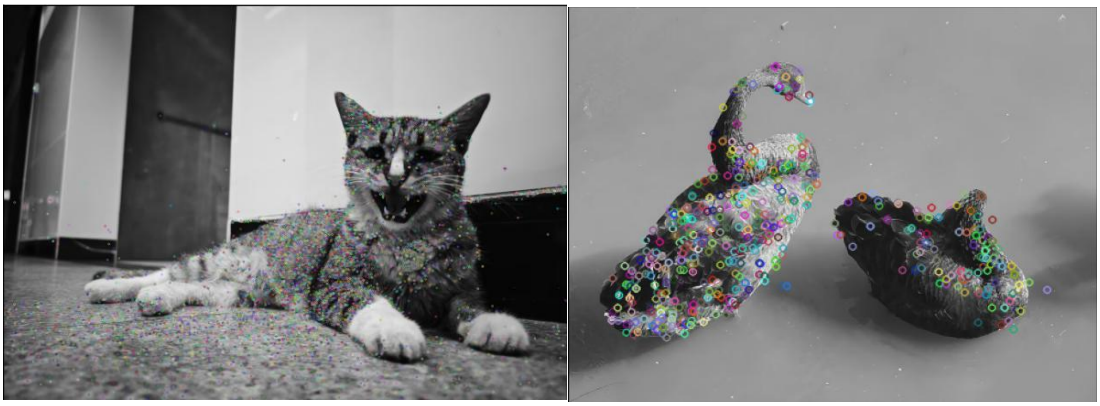


图 3 测试算法效果