机器视觉第二次作业

1、了解概念：二值图像连通区域。编写“种子填充法”标记二值图像的连通区域的伪代码，并讨论其算法复杂度。

（1）二值图像连通区域是指在二值图像中，图像只由 0 和 1 组成，所以连通区域是图像中具有相同像素值且为指向零的前景像素点组成的图像区域，如图 1 所示，假设黑色点表示 0，白色像素点表示 1，则若 A 像素点为黑色，则黑色联通区域为 A 周围所有的 0（红框区域）。

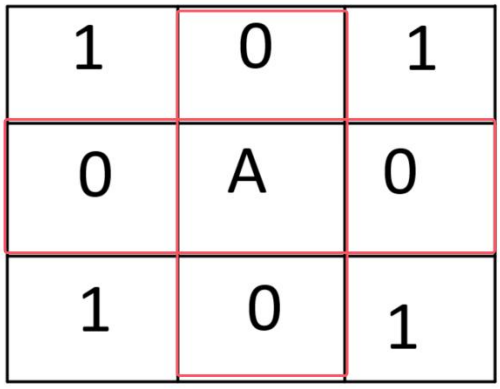


图 1 连通区域示例

（2）“种子填充法”首先选择一个未标记的“种子”像素，通过四邻域或八邻域或其它的规则将其扩展到连通的像素，直到整个连通区域都被标记，下面采用四邻域方法书写伪代码：

```
SeedFill(image, visited, i, j, label): //赋予每个像素标签
    // 判断当前坐标是否合法
    if i < 0 or i >= len(image) or j < 0 or j >= len(image[0]):
        return  // 超出图像边界返回
    // 如果该像素已经被访问过，或者该像素值为背景（0），则返回
    if visited[i][j] == true or image[i][j] == 0:
        return
    // 标记当前像素为已访问，并赋予标签
    visited[i][j] = true
    image[i][j] = label
    // 递归处理四邻域的像素（上下左右）
```

```
    SeedFill(image, visited, i + 1, j, label)  // 下
    SeedFill(image, visited, i - 1, j, label)  // 上
    SeedFill(image, visited, i, j + 1, label)  // 右
    SeedFill(image, visited, i, j - 1, label)  // 左
//主函数
LabelConnectedRegions(image):
    n, m = len(image), len(image[0])  // 获取图像的尺寸
    visited = [[false] * m for _ in range(n)]  // 初始化访问标记数组
    label = 0  // 连通区域的标签
    // 遍历整个图像
    for i from 0 to n-1:
        for j from 0 to m-1:
            // 如果当前像素是前景（1）并且尚未访问，则开始标记
            if image[i][j] == 1 and visited[i][j] == false:
                label = label + 1  // 增加标签
                SeedFill(image, visited, i, j, label)  // 填充该连通区域
```

2、编程实现二值图像连通区域标记，并计算其面积等统计量。

二值图像连通区域标记主要在以下几个步骤进行设计：

读取图像——种子填充法找连通区域——将不同连通区域使用不同颜色标注。为了方便展示，代码中使用了一个自定义的小型二值图像，从而更方便地展示在连通区域标注后的结果，原二值图像如图 2 所示，它的 npy 数组格式展示为，1 表示前景，0 表示背景：

```
[0, 1, 0, 0, 0, 1],
[1, 1, 0, 1, 0, 0],
[0, 0, 0, 0, 1, 0],
[0, 1, 1, 0, 0, 0],
 [1, 0, 1, 0, 0, 0]
```

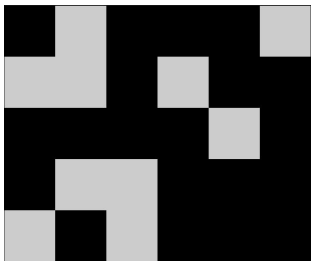


图 2 二值图像

随后在此基础上，我使用种子填充法，以从左到右、从上到下的顺序遍历整个图像，并采用四邻域的方式进行连通区域判定，当检测到值为 1 且未访问时，认为找到了一个新的连通区域，并通过栈模拟的深度优先搜索（DFS）进行区域扩展，判断四邻域是否属于同一区域。在扩展过程中，不断更新区域面积、边界信息以及对应的标签值。通过这种方式，对所有连通区域实现了标记和统计。

```python
def seed_fill(image, visited, i, j, label):
    '''
    进行种子填充，应用四邻域的规则
    '''
    # 获取图像的行列数
    n, m = len(image), len(image[0])
    # 定义四邻域方向（上下左右）
    directions = [(1, 0), (-1, 0), (0, 1), (0, -1)]
    # 使用栈进行迭代填充（避免递归栈溢出）
    stack = deque([(i, j)])
    area = 0  # 计算连通区域的面积
    boundary = [i, i, j, j]  # 保存边界框
    while stack:
        x, y = stack.pop()
        # 检查边界和是否已访问
        if x < 0 or x >= n or y < 0 or y >= m or visited[x][y] or image[x][y] == 0:
            continue
        visited[x][y] = True
        image[x][y] = label  # 给连通区域标记一个标签
        area += 1  # 统计连通区域的面积，找到一个就加一
        # 更新连通区域的边界框
        boundary[0] = min(boundary[0], x)  # 上
        boundary[1] = max(boundary[1], x)  # 下
        boundary[2] = min(boundary[2], y)  # 左
        boundary[3] = max(boundary[3], y)  # 右
        # 添加四个邻域像素到栈中
        for dx, dy in directions:
            stack.append((x + dx, y + dy))
    return area, boundary
```

图 3 种子填充法代码图

程序在标记连通区域的基础上，统计了每个区域的面积、边界（上下左右像素索引）、以及区域的中心点，并输出这些信息。此外，使用不同颜色标签图像中不同的连通区域从而能够直观显示标记结果，如图 5 所示，所分出来的连通区域与原二值图像对比能够完全匹配。

```python
def label_connected_regions(image):
    '''
    找连通区域
    '''
    n, m = len(image), len(image[0])
    visited = np.zeros_like(image, dtype=bool)  # 用于标记访问过的像素
    label = 0  # 连通区域标签
    regions = []  # 存储每个连通区域的统计信息

    # 遍历整个图像
    for i in range(n):
        for j in range(m):
            if image[i][j] == 1 and not visited[i][j]:
                label += 1  # 找到一个新的连通区域，给它分配一个新的标签
                area, boundary = seed_fill(image, visited, i, j, label)


                # 保存该连通区域的统计量
                regions.append({
                    'label': label,
                    'area': area,
                    'boundary': boundary,
                })

    return image, regions
```

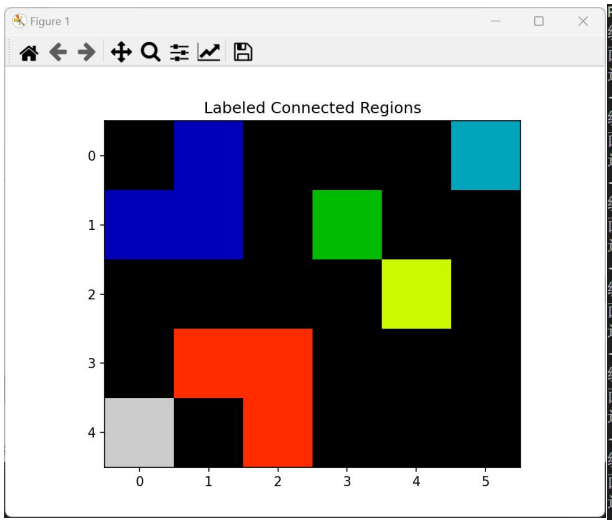
```python
# 1是前景，0是背景
image = np.array([
    [0, 1, 0, 0, 0, 1],
    [1, 1, 0, 1, 0, 0],
    [0, 0, 0, 0, 1, 0],
    [0, 1, 1, 0, 0, 0],
    [1, 0, 1, 0, 0, 0]
])

# 标记连通区域并计算统计量
labeled_image, regions = label_connected_regions(image.copy())

# 输出每个连通区域的统计量
for region in regions:
    print(f"编号: {region['label']-1}")#连通区域的编号
    print(f"面积: {region['area']}")#连通区域的面积，即有多少个像素点
    print(f"边界: 上: {region['boundary'][0]}, 下: {region['boundary'][1]}, 左: {region['boundary'][2]}, 右: {region['boundary'][3]}")
    print("------------------------")

plt.imshow(labeled_image, cmap='nipy_spectral', interpolation='nearest')
plt.title("Labeled Connected Regions")
plt.gca().set_facecolor('black')  # 设置背景为黑色
plt.show()
```

图 4 选择连通区域及计算统计量代码图




```
PS W:\Codes\VsCode\VsCode_Storage\CV\homework\hw3
编号: 0
面积: 3
边界: 上: 0, 下: 1, 左: 0, 右: 1
------------------------
编号: 1
面积: 1
边界: 上: 0, 下: 0, 左: 5, 右: 5
------------------------
编号: 2
面积: 1
边界: 上: 1, 下: 1, 左: 3, 右: 3
------------------------
编号: 3
面积: 1
边界: 上: 2, 下: 2, 左: 4, 右: 4
------------------------
编号: 4
面积: 3
边界: 上: 3, 下: 4, 左: 1, 右: 2
------------------------
编号: 5
面积: 1
边界: 上: 4, 下: 4, 左: 0, 右: 0
```

图 5 标记结果及统计量