- 应用最大流最小割集思想对图像进行分割
- 数据输入：一幅图像（起码3个实例）
- 结果输出：分割后图像

1. 实现思想

算法采用图割（Graph Cut）思想，通过用户提供的前景和背景种子点，**运用最小割算法将图像划分为前景和背景两个部分。**
在图割问题中，**图的节点表示图像像素，** 图的边表示像素之间的关系，而通过引入源点和汇点，将前景和背景的种子点连接至网络，形成网络流结构。**边的权重根据像素之间的相似性计算，用户提供的种子点则被赋予特殊权重，加强它们之间的关系。** 通过最小割算法，寻找使前景和背景之间的权重最小的切割，最终获得图像的分割结果。

最大流最小割算法:

Ford-Fulkerson算法是解决网络流问题的经典算法，其基本思想是通过不断寻找增广路径来增加流量，直到无法找到增广路径为止。算法的主要步骤包括初始化，寻找增广路径，更新流量，重复寻找增广路径和更新流量直至无法找到增广路径，最终计算得到图中的最大流。在每次找到增广路径时，通过在残余图上进行DFS或BFS来确定路径，并更新路径上边的流量，以逐步增加整个图的流量。

将图片转化为图：

因为本次实验重点在于最大流最小割的理解，将图片转化为图的质量好坏很大程度就能决定最终实现的效果好坏。**顶点的创建**：通过遍历图像中的每个像素，创建对应的节点，并根据其属于前景还是背景，设置相应的容量。**边的创建与权重赋值**：对图像中的每个像素，计算其与右侧像素和下方像素的边的容量，分别表示水平方向和垂直方向上的边。边的容量计算通常基于像素之间的颜色相似度，采用归一化的颜色差的倒数。与此同时，也为用户提供的前景和背景的种子点，进行了特殊处理。

算法步骤

- 步骤 1：初始化：读取输入图像。设置默认边权重、最大容量、前景和背景的种子点列表，以及前景选择状态。
- 步骤 2：选择种子点：用户通过鼠标点击在图像上选择前景和背景的种子点，按'S'键切换选择状态，按回车键结束选择。
- 步骤3：计算默认权重：根据用户选择的前景和背景种子点，初始化边权重矩阵。
- 步骤 4：创建图的节点和边：构建图的节点和边，表示图像中的像素点和它们之间的关系。
- 步骤 5：解决最大流问题：利用`maxflow`库创建图，添加节点和边。使用最大流算法求解最大流问题，得到图中节点的分割状态。
- 步骤 6：提取分割结果：根据最大流问题的解，提取图像的分割结果。利用OpenCV的`connectedComponents`函数找到分割结果中的连通组件。
- 步骤 7：**展示结果：展示分割结果图和相邻组件之间的连接线**，通过可视化展示算法的分割效果。

算法实现了一种基于用户交互的图像分割方法，用户通过选择前景和背景的种子点，算法根据图的最大流进行图像分割，并通过可视化展示结果。

在本次的实验中，使用最大流最小割算法将像素点分为两个集合时，选择绘制展示两个集合交接的线，展示分割结果。**本次算法在最大流最小割的基础上，自行将图片转化为顶点与边的图，并赋予权重**（个人认为此处仍有很大的改进空间，因为**权重的好坏决定了最终的划分结果**，查阅网上有许多复杂科学的相关证明。因为时间原因，本次实验采用的权重赋值方法仍是较为基础的，**且受参数影响较大**），因为掌握了最大流最小割算法原理，所以本次算法在这一步骤直接调用了maxflow库。在图像分割领域，有许多成熟的库函数以及方法，本次实验只是突出了最大流最小割算法的实践效果。

2. 输出截图

按照题目要求在resource文件夹中加入三张测试的原图像，运行代码将分割后的结果图片也放置在该文件夹中，分割前后图片如下所示：

image1分割前后如下图所示：

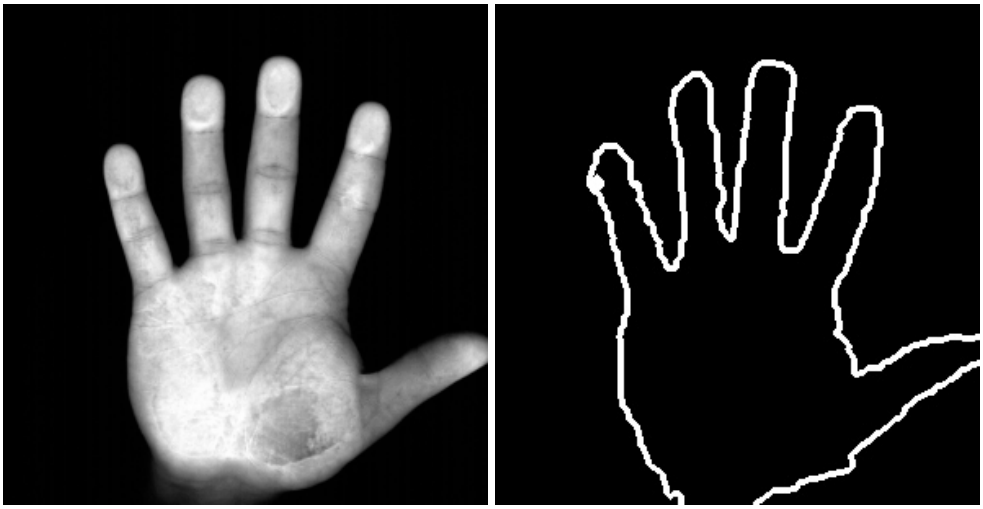

image2分割前后如下图所示：

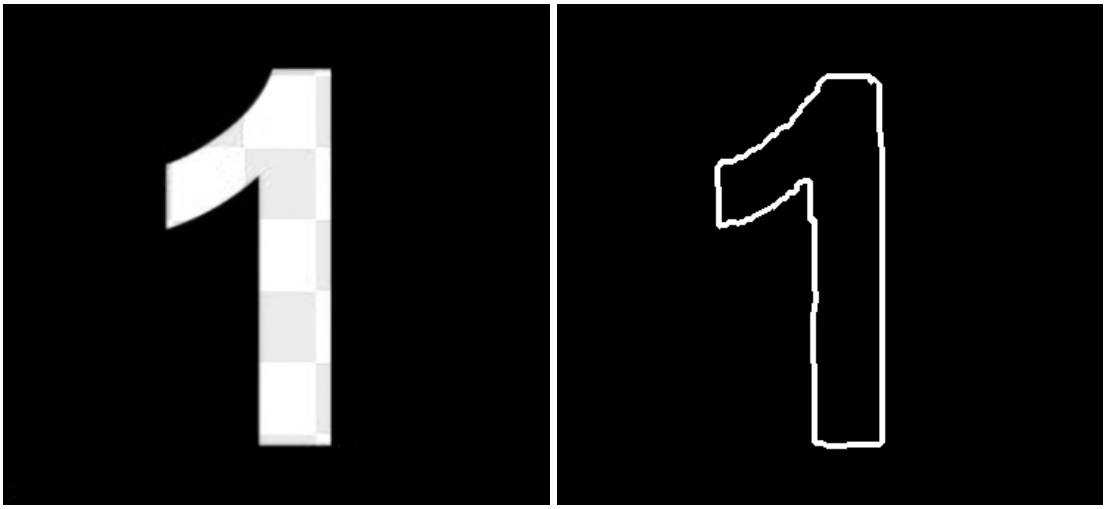

image3分割前后如下图所示：

在本次的实验中，使用最大流最小割算法将像素点分为两个集合时，选择绘制展示两个集合交接的线，展示分割结果。上文中的结果较好的展示了算法效果。

```
#分割图像类函数，属性值包括图像处理，如前景后景集合等
class InteractiveImageSegmentation:
    def __init__(self, image_path):
        self.image = cv2.imread(image_path)
        self.default_weight = 0.5
        self.MAX_CAPACITY = 100000
        self.background_seeds = []
        self.foreground_seeds = []
        self.is_foreground_selected = True
```

```
    # 鼠标回调函数，用于在图像上选择前景和背景种子点
    def mouse_callback(self, event, x, y, flags, param):
        if event == cv2.EVENT_LBUTTONDOWN or (event == cv2.EVENT_MOUSEMOVE and
flags & cv2.EVENT_FLAG_LBUTTON):
            if self.is_foreground_selected:
                self.foreground_seeds.append((x, y))
                cv2.circle(self.image, (x, y), 2, (255, 0, 0), -1)
            else:
                self.background_seeds.append((x, y))
                cv2.circle(self.image, (x, y), 2, (0, 0, 255), -1)
            cv2.imshow("Image", self.image)

    # 切换前景和背景种子点的选择状态
    def switch_seed_selection(self):
    self.is_foreground_selected = not self.is_foreground_selected
    if self.is_foreground_selected:
        print("选择前景种子点")
    else:
        print("选择背景种子点")

    # 交互式选择前景和背景种子点
    def select_seeds_interactively(self):
        print("选择前景种子点")
        cv2.imshow("Image", self.image)
```

```
        cv2.setMouseCallback("Image", self.mouse_callback)

        while True:
            key = cv2.waitKey(0)
            if key == ord('s'):
                self.switch_seed_selection()
            elif key == 13:  # 回车键
                break
        cv2.destroyAllWindows()
```

```
# 计算默认权重，即边的权重
    def calculate_default_weights(self):
        self.edge_weights = np.zeros((self.image.shape[0], self.image.shape[1]))
        self.edge_weights.fill(self.default_weight)

        for coordinate in self.background_seeds:
            self.edge_weights[coordinate[1], coordinate[0]] = 0

        for coordinate in self.foreground_seeds:
            self.edge_weights[coordinate[1], coordinate[0]] = 1

# 创建图的顶点与边权重
    def create_graph_nodes(self):
        self.graph_nodes = []  # 存储图的节点
        self.graph_edges = []  # 存储图的边

        for (y, x), weight in np.ndenumerate(self.edge_weights):
            if weight == 0.0:  # 背景像素，容量为(最大容量，0)
                self.graph_nodes.append((self.get_node_num(x, y,
self.image.shape), self.MAX_CAPACITY, 0))
            elif weight == 1.0:  # 前景像素，容量为(0，最大容量)
                self.graph_nodes.append((self.get_node_num(x, y,
self.image.shape), 0, self.MAX_CAPACITY))
            else:  # 未分类像素，容量为(0，0)
                self.graph_nodes.append((self.get_node_num(x, y,
self.image.shape), 0, 0))

        for (y, x), weight in np.ndenumerate(self.edge_weights):
            if y == self.edge_weights.shape[0] - 1 or x ==
self.edge_weights.shape[1] - 1:
                continue

            my_index = self.get_node_num(x, y, self.image.shape)

            # 计算与右边像素的边的容量
            neighbor_index = self.get_node_num(x + 1, y, self.image.shape)
            edge_capacity = 1 / (1 + np.sum(np.power(self.image[y, x] -
self.image[y, x + 1], 2)))
            self.graph_edges.append((my_index, neighbor_index, edge_capacity))

            # 计算与下方像素的边的容量
```

```
            neighbor_index = self.get_node_num(x, y + 1, self.image.shape)
            edge_capacity = 1 / (1 + np.sum(np.power(self.image[y, x] -
self.image[y + 1, x], 2)))
            self.graph_edges.append((my_index, neighbor_index, edge_capacity))
```

```
# 解决最大流问题
    def solve_maxflow(self):
        g = maxflow.Graph[float](len(self.graph_nodes), len(self.graph_edges))
        node_list = g.add_nodes(len(self.graph_nodes))

        for node in self.graph_nodes:
            g.add_tedge(node_list[node[0]], node[1], node[2])

        for edge in self.graph_edges:
            g.add_edge(edge[0], edge[1], edge[2], edge[2])

        g.maxflow()

        return g
```

```
# 提取分割结果
    def extract_segmentation_results(self, maxflow_graph):
        segment_overlay = np.zeros_like(self.image)
        mask = np.zeros((self.image.shape[0], self.image.shape[1]),
dtype=np.uint8)

        for index in range(len(self.graph_nodes)):
            if maxflow_graph.get_segment(index) == 1:
                xy = self.get_xy(index, self.image.shape)
                segment_overlay[xy[1], xy[0]] = (255, 0, 255)
                mask[xy[1], xy[0]] = 255

        return mask, segment_overlay

# 可视化连接的组件
    def visualize_connected_components(self, segmentation_mask):
        num_labels, labels = cv2.connectedComponents(segmentation_mask)
        lines_image = np.zeros_like(self.image)

        for (y, x), value in np.ndenumerate(labels):
            if x < self.image.shape[1] - 1 and labels[y, x] != labels[y, x + 1]:
                cv2.line(lines_image, (x, y), (x + 1, y), (255, 255, 255), 2)
            if y < self.image.shape[0] - 1 and labels[y, x] != labels[y + 1, x]:
                cv2.line(lines_image, (x, y), (x, y + 1), (255, 255, 255), 2)

        cv2.imshow("seg_image", lines_image)
        cv2.imwrite("seg_image3.jpg", lines_image)
        cv2.waitKey(0)
        cv2.destroyAllWindows()
```

```
    # 获取节点编号
    def get_node_num(self, x, y, shape):
        return y * shape[1] + x

    # 根据节点编号获取坐标
    def get_xy(self, index, shape):
        y = index // shape[1]
        x = index % shape[1]
        return x, y

    # 运行交互式图像分割
    def run_segmentation(self):
        self.select_seeds_interactively()
        self.calculate_default_weights()
        self.create_graph_nodes()
        maxflow_graph = self.solve_maxflow()
        segmentation_mask, segmentation_overlay =
self.extract_segmentation_results(maxflow_graph)
        self.visualize_connected_components(segmentation_mask)
```

```
# 用法:
image_path = "image.jpg"
segmentation = InteractiveImageSegmentation(image_path)
segmentation.run_segmentation()
```