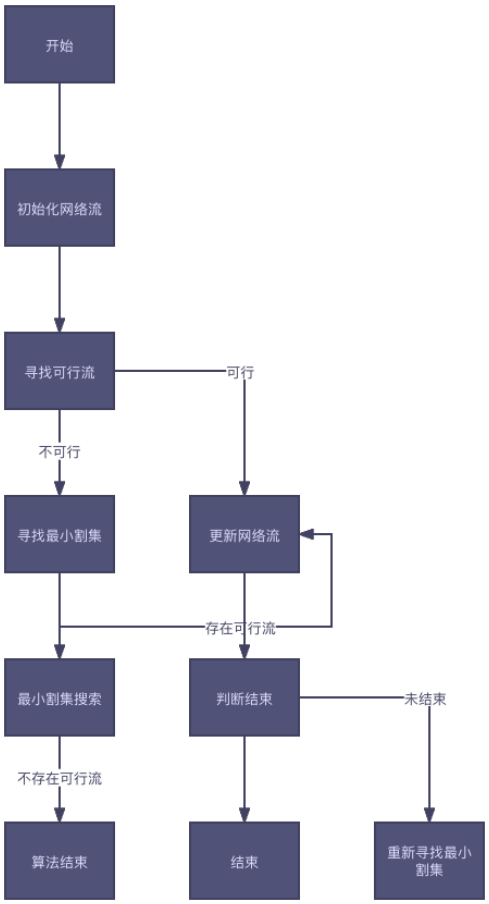
算法设计与分析 网络流实验作业

林宇浩 21311274

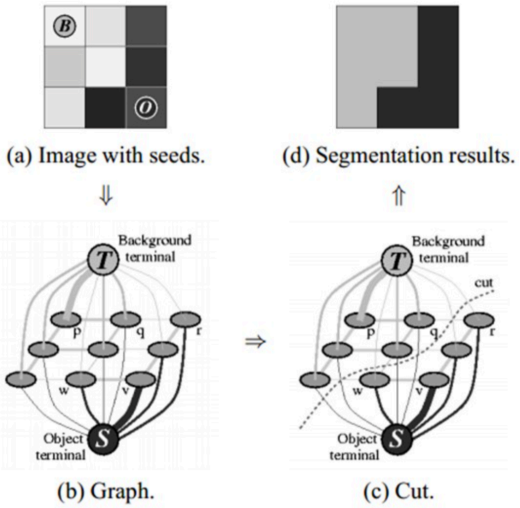
一、算法实现思想

最大流算法流程图

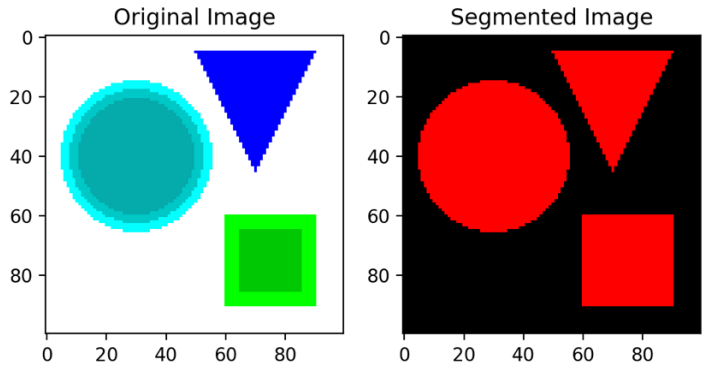


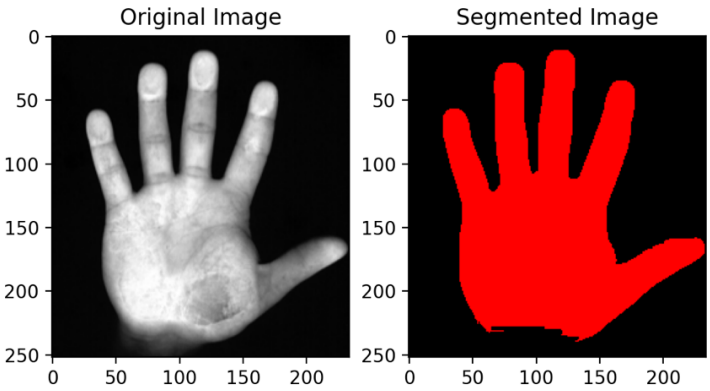
使用最大流算法找到最小割的基本步骤如下：
1、建立流网络： 将原始问题转换为网络流问题，创建一个有向图表示流网络。图中包括源节点（表示流的起点）、汇点（表示流的终点）、以及连接它们的有向边，每条边上有一个容量值表示最大流量。
2、初始化流： 将初始流设置为零。
3、寻找增广路径： 使用增广路径算法寻找从源节点到汇点的一条路径，该路径上的边具有剩余容量。
4、更新流： 将沿着增广路径找到的最小剩余容量，添加到路径上的每一条边上，然后更新图中的流。
5、重复步骤 3 和 4： 重复以上步骤，直到无法找到增广路径为止。当无法再找到增广路径时，最小割被找到了。最小割由将图中的节点划分为两个不相交的集合所确定的，一个集合包含可以到达的节点，另一个集合包含不能到达的节点。

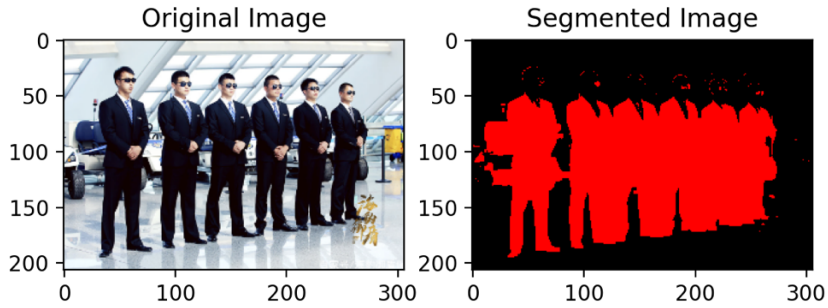
关于利用最大流最小割集思想对图像进行分割的原理，举例而言，图像分割问题的核心是通过评估像素点之间的相似程度来确定不同的区域，从而实现图像的分割。我们可以将相邻像素点之间的相似程度定义为边，其中相似程度越大，边的权值越大。显而易见的是，不同区域之间存在很大的差异，因此相邻区域之间的连通边的权值通常较小，因为不同区域之间的相似程度较小，因此相邻边的权值较小。从而找到权值之和相对较小的边集，可能相当于找到了区域的边界，如下图所示。最早提出此方法的论文为 Interactive Graph Cuts for Optimal Boundary & Region Segmentation of Objects in N-D Images。



二、输出截图



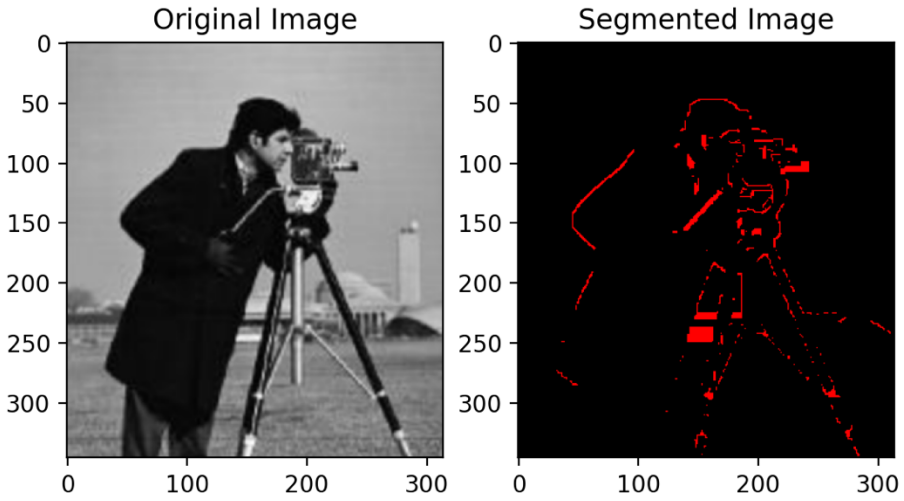


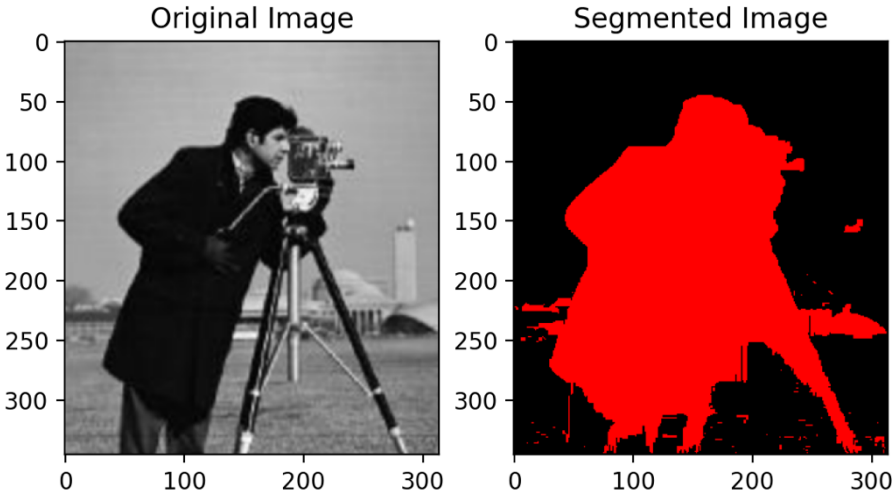


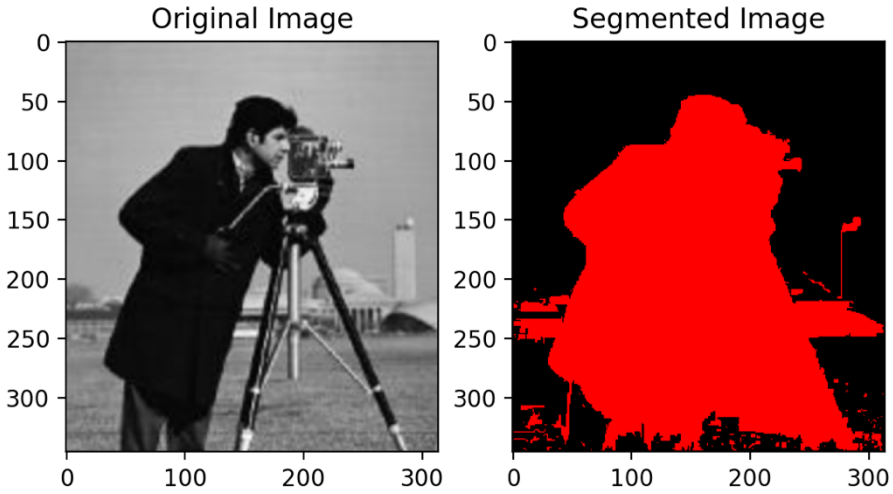
实验中发现，最大流最小割集算法分割图像的性能受像素差异衡量函数的影响较大，在测试了十几种函数后，目前发现反比例函数的效果最好。具体公式如下所示：

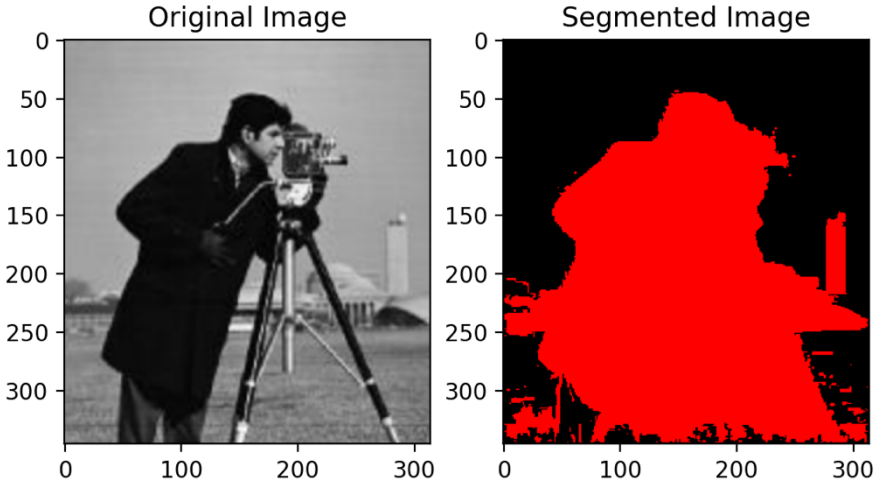
```
dif = ((abs(img.getpixel((x1, y1)) - img.getpixel((x2, y2)))+0.01)**(-a))*math.sqrt(img.size[0]*img.size[1])
```

公式中加了 0.01 是为了防止除 0 错误，乘以根号下 img.size[0]*img.size[1] 是为了排除图片尺寸对分割造成的影响，反比例函数系数的值 a 可以根据想要的分割效果进行赋值，效果差异如下，图片 a 值依次从小到大：









三、源代码

```
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
from PIL import Image

def draw_graph(G):
    pos = nx.spring_layout(G)
    nx.draw(G, pos, with_labels=True, font_weight='bold',
node_color='skyblue', node_size=700, edge_color='gray')
    plt.title('Graph Structure')
    plt.show()


def image_segmentation(img):

    # 创建图像的图
    G = create_image_graph(img)
    # draw_graph(G)
```

```
    # 使用最大流最小割算法
    cut_value, partition = nx.minimum_cut(G, 'source', 'sink')

    # 获取分割结果
    segmented_image = get_segmented_image(img, partition[0])

    # 显示原图和分割结果
    plt.subplot(1, 2, 1)
    plt.title('Original Image')
    plt.imshow(img)

    plt.subplot(1, 2, 2)
    plt.title('Segmented Image')
    plt.imshow(segmented_image)

    plt.show()


def create_image_graph(img):
    # 创建图
    img = img.convert('L')  # 转为灰度图

    G = nx.Graph()

    # 图像的宽和高
    width, height = img.size

    # 添加源节点和汇点
    G.add_edge('source', f'{int(width/2)}-{int(height/2)}',
capacity=999999999999)
    # G.add_edge('source', f'{int(width/4)}-{int(height/4)}',
capacity=999999999999)
    # G.add_edge('source', f'{int(width*3/4)}-{int(height/5)}',
capacity=999999999999)
    # G.add_edge('source', f'{int(width*3/4)}-{int(height*3/4)}',
capacity=999999999999)

    for i in range(width):
        G.add_edge(f'{i}-{0}', 'sink', capacity=999999999999)
        G.add_edge(f'{i}-{height-1}', 'sink', capacity=999999999999)
    for i in range(height):
        G.add_edge(f'{0}-{i}', 'sink', capacity=999999999999)
        G.add_edge(f'{width-1}-{i}', 'sink', capacity=999999999999)

    # 添加节点和边
    for i in range(width):
        for j in range(height):
            node_id = f'{i}-{j}'

            # 添加节点之间的边
            if i < width - 1:
                G.add_edge(node_id, f'{i+1}-{j}',
capacity=get_pixel_difference(img, i, j, i+1, j))
            if j < height - 1:
                G.add_edge(node_id, f'{i}-{j+1}',
capacity=get_pixel_difference(img, i, j, i, j+1))

    return G
```

```python
import math
def get_pixel_difference(img, x1, y1, x2, y2):
    # 获取两个像素之间的差异
    dif = ((abs(img.getpixel((x1, y1)) - img.getpixel((x2, y2)))+0.01)**(-
4))*math.sqrt(img.size[0]*img.size[1])
    print(dif,end=' ')

    return int(dif)


def get_segmented_image(img, partition):
    # 创建分割后的图像
    segmented_image = Image.new('RGB', img.size)

    # 根据分割结果设置像素值
    for node_id in partition:
        if node_id != 'source' and 'sink':
            x, y = map(int, node_id.split('-'))
            # segmented_image.putpixel((x, y), img.getpixel((x, y)))
            segmented_image.putpixel((x, y), (255, 0, 0))

    return segmented_image

# if __name__ == "__main__":
#     image_path = "circles.png"  # 替换为你的图像路径
#     image_segmentation(image_path)


def matrix_to_image(matrix):
    # 将矩阵转换为 NumPy 数组
    array = np.array(matrix, dtype=np.uint8)

    # 创建图像对象
    img = Image.fromarray(array)

    # 显示图像
    # img.show()

    return img


# # 用一个示例矩阵表示图像
# example_matrix = [
#     [0,  0, 0],
#     [0, 255, 0],
#     [0,  255, 0],
#     [0, 0, 0]
# ]

# # 显示图像
# img = matrix_to_image(example_matrix)

image_path = "camera.png"  # 替换为你的图像路径
img = Image.open(image_path)

image_segmentation(img)
```