基于图的图像分割实验报告

一、 实验目的

读入一幅图像，用基于图的方法对图像进行分割，并将分割前后图像进行对比显示，要求分割有较好的效果。

二、 实验环境

编程语言：Python3.6

操作系统：Ubuntu18.04

代码编辑器：vscode

调用 Python 库：sys、numpy、matplotlib

三、 算法原理

图像分割的过程就是将图片中像素点不断分类的过程，因此这里采用了连通图的思想进行图像的分割。具体做法是先为所有相邻像素点生成边，这些边具有所连接的像素点位置信息和权重信息，其中的权重指的是相连像素点的差异程度，这里采用了颜色空间的欧氏距离来衡量，即$w = \mathrm{sqrt}\big(\mathrm{sum}((\mathrm{R}_1 - \mathrm{R}_2)^2 + (\mathrm{G}_1 - \mathrm{G}_2)^2 + (\mathrm{B}_1 - \mathrm{B}_2)^2)\big)$。例子中 R、G、B 分别表示三维颜色空间的红、绿、蓝三通道。除了用于衡量像素点之间的相似度的权重，还需要用一定的方式衡量已经聚好类的像素点簇的内聚性，以及能够用算法评估两个簇之间的相似性，簇与边权重之间的相似性。为此定义了簇的内聚性$c = \frac{k}{size}$，其中 k 表示一个常量（通常对不同类型的图片有不同取值），size 表示簇的大小。同时为了表示簇和类的权重之间的关系用$\mathrm{w} + c$表示簇的权重。接着将所有边按权重大小排好序，并对所有边依次进行如下操作：

1、 判断该边连接的两个像素点的权重是否都小于他们所在簇的权重，如果不是，跳过下面步骤直接分析下一条边，否则按如下方式将两个像素点所在簇合并为一个：

2、 为了方便操作用并查集来表示所有像素点所属的簇，当需要合并两个并查集的时候，只需要调用相应的 merge 函数，同时为了能够查到某个簇的大小以及权重，在并查集的基础上加了 sizes 和 values 两个变量。

完成所有像素点的聚类操作后，接着需要将大小较小的簇进一步合并（这里通过 min_pixels 变量来控制最小的簇大小）。通过合并能够很好的减少噪声对结果的影响。以上就完成了分割的过程，接下来需要从并查集中生成分割好的图片，具体做法如下：

从并查集中提取出所有的像素点集合，将集合的均值作为该簇的像素值，并还原到相应的图片位置上。

为了能够更好的过滤掉噪点对分割效果的影响，对图像进行了一定的预处理过程，主要体现在实现了高斯模糊函数，并通过调整高斯模糊的半径大小和高斯函数sigma的取值来改变模糊的程度。

四、 代码流程

基于图的图像分割实现流程：

1、 首先对图像进行高斯模糊处理。

```
def guassian_blur(img, radius, sigma):
    '''高斯模糊函数

    Args:
        radius: 高斯核的半径
```

```
    sigma: 高斯分布的 sigma
    img: 待模糊图像
  '''
  height, width, size = img.shape[0], img.shape[1], radius * 2 + 1
  if len(img.shape) < 3:
    filter, chanel = np.empty((size, size)), 1
  else:
    filter, chanel = np.empty((size, size, img.shape[2])), img.shape[2]
  for i in range(size):
    for j in range(size):
      x, y = i - radius, j - radius
      filter[i, j] = (1 / np.sqrt(2 * np.pi * sigma * sigma) *
      np.exp(-(x * x + y * y) / (2 * sigma * sigma)))
  filter = filter / filter.sum() * chanel
  output = np.empty(img.shape)

  for i in range(radius, height - radius):
    for j in range(radius, width - radius):
      output[i, j] = (filter * img[i - radius: i + radius + 1,
      j - radius: j + radius + 1]).sum(axis=(0, 1))
  return output
```

2、 然后将图片中的相邻像素点转化成边

```
def gen_edges(img, diff):
  '''将图片中相邻的像素点用边连接，得到一个联通图

  Args:
    img: 待处理图片
    diff: 用于衡量像素点之间的差别

  Returns:
    所有边的列表
  '''
  edges = []
  height, width = img.shape[: 2]
  get_id = lambda x, y: x * width + y
  for i in range(height):
    for j in range(width):
      if i > 0:
        weight = diff(img, (i, j), (i - 1, j))
        edges.append(Edge(get_id(i - 1, j), get_id(i, j), weight))
      if j > 0:
        weight = diff(img, (i, j), (i, j - 1))
        edges.append(Edge(get_id(i, j - 1), get_id(i, j), weight))
      # other case
```

```
    return edges
```

3、 用图的算法将所有像素点按边权从小到大聚类，为了加快速率采用了并查集作为数据结构

```
def seg_img(img, k, min_pixels, diff=lambda img, xy, xxyy:
    np.sqrt(np.sum((0. + img[xy] - img[xxyy]) ** 2)),
    threshold=lambda k, size: k / size):
    '''图像分割的主调函数

    Args:
        img: 待分割的图像
        k: k 值，配合内聚函数使用
        min_pixels: 最小的像素点聚类个数
        diff: 用于衡量像素点之间的差异程度
        threshold: 衡量像素点聚类的内聚度函数

    Returns:
        返回分割好的图像
    '''
    # 先进行高斯模糊处理
    img = utl.guassian_blur(img, 3, 1)
    # 得到图像的所有边集
    edges = gen_edges(img, diff)
    height, width = img.shape[: 2]
    # 生成并查集
    ds = utl.DisjointSet(height * width, threshold(k, 1))
    # 将所有的边排好序
    edges.sort(key=lambda edge: edge.w)
    # 开始基于图的分割过程
    for edge in edges:
        pa, pb = ds.find(edge.a), ds.find(edge.b)
        if edge.w <= ds[pa] and edge.w <= ds[pb] and pa != pb:
            ds.merge(pa, pb)
            ds[pa] = edge.w + threshold(k, ds.get_size(pa))
    # 对小区域的聚类进行合并
    for edge in edges:
        pa, pb = ds.find(edge.a), ds.find(edge.b)
        if pa != pb and ds.get_size(pa) < min_pixels and ds.get_size(pb) < min_pixels:
            ds.merge(pa, pb)
    return gen_img(ds, img)
```

4、 最后根据并查集的像素点集合还原出分割后的图像

```
def gen_img(ds, img):
    '''由像素点组成的并查集 ds 来还原分割后的图片

    Args:
```

```
        ds: 像素点组成的并查集
        img: 原图片

    Returns:
        返回还原好的图片
    '''
    image = np.empty(img.shape)
    height, width = img.shape[: 2]
    colors, counts = {}, {}
    for i in range(height):
        for j in range(width):
            parent = ds.find(i * width + j)
            if parent not in colors:
                colors[parent] = img[i, j]
                counts[parent] = 1
            else:
                colors[parent] = colors[parent] + img[i, j]
                counts[parent] += 1
    for parent in colors:
        colors[parent] = colors[parent] / counts[parent]
    for i in range(height):
        for j in range(width):
            parent = ds.find(i * width + j)
            image[i, j] = colors[parent]
    return image
```

5、 并查集的主要功能实现如下

```
class DisjointSet:
    '''并查集

    Attributes:
        self.array: 并查集的关键数组
        self.sizes: 每个元素所属的集合元素个数
        self.values: 每个元素所属的集合的取值
    '''
    def __init__(self, n, value=None):
        self.array = [i for i in range(n)]
        self.sizes = [1 for i in range(n)]
        self.values = [value for i in range(n)]

    def find(self, a):
        '''从并查集中寻找元素所属的集合'''
        parent = self.array[a]
        while a != parent:
            self.array[a] = a = self.array[parent]
```

```
            parent = self.array[a]
        return a

    def merge(self, a, b, value=None):
        '''合并两个元素，并可选择指定合并之后的集合取值

        Args:
            a: 元素 a
            b: 元素 b
            value: 合并后他们所属集合的取值
        '''
        parent_a, parent_b = self.find(a), self.find(b)
        if parent_a != parent_b:
            self.array[parent_b] = parent_a
            self.sizes[parent_a] += self.sizes[parent_b]
            self.values[parent_a] = value

    def get_size(self, a):
        '''得到元素 a 所属集合的大小'''
        parent = self.find(a)
        return self.sizes[parent]

    def __setitem__(self, key, value):
        '''设定元素 key 所属集合的取值'''
        parent = self.find(key)
        self.values[parent] = value

    def __getitem__(self, key):
        '''得到元素 key 所属集合的取值'''
        parent = self.find(key)
        return self.values[parent]
```

五、 实验结果

不同超参数的选择对最后图像分割的效果影响较大，下面展示几个不同图像和参数选取时的分割效果。

Figure 1 图像分割效果对比

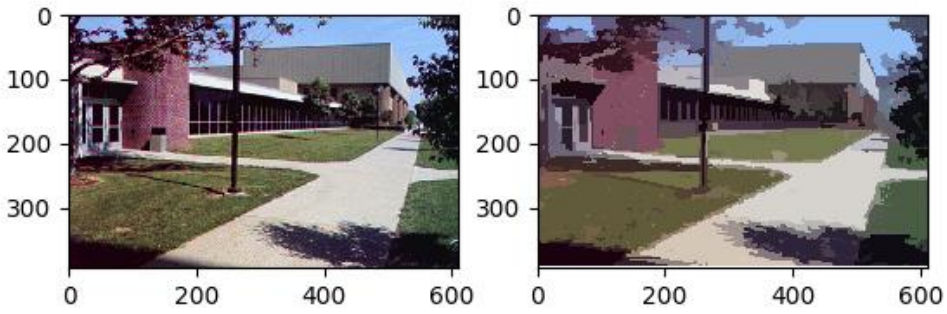
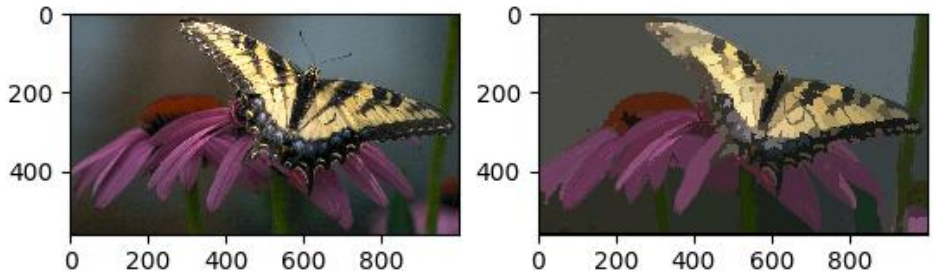


Figure 2 图像分割效果对比



六、 总结分析

基于图的图像分割是一种简单且基础的图像分割方式，其中用到了并查集和图论的基本方法，为了能够更好的适应图像分割的像素点聚簇，在传统并查集的基础上添加了查看并查集簇大小和取值的函数。为了能够忽略像素中的噪点影响，用高斯模糊对图像进行了预处理。该实验中不同超参数的取值对实验结果影响较大，而且不同图像的分割往往需要不同的参数，因此程序可以进一步在参数选取方面进行优化。