《计算机视觉（1）》实验报告

实验六 使用区域生长的分割

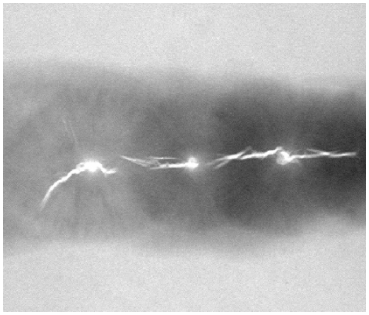
实验小组成员 （学号+班级+姓名）	分工及主要完成任务	成绩
201800810253+数据科学+王荣	图像的处理，算法的代码实现	
201800820149+数据科学+徐潇涵	实验报告	

山东大学

2021 年 3 月

完成《数字图像处理》P494页例10.23的编程实验，编程语言可以选择Matlab，C，C++，OpenCV，Python等。设计方案可参照教科书中的分析，也可以自行设计新的方案。

参照例10.23中“使用区域生长的分割”算法，对如下的有缺陷焊缝的X射线图像得到图10.51中(b)~(i)中的实验结果。



原始图像的电子版图像在 Images 文件夹中。实验报告写在如下空白处，页数不限。

实验报告

一、 实验内容

本实验的主要内容在于重现图 10.51 中(b)~(i)中的结果。重点编程内容如下：

1) 基于形态学腐蚀的种子图像的生成

首先在高百分比处（99.5%）设置阈值（254）对原始图像进行阈值处理，接着对图中的连通分量进行检测，对每个连通分量用形态学腐蚀为一个单点，即为最终的种子图像。

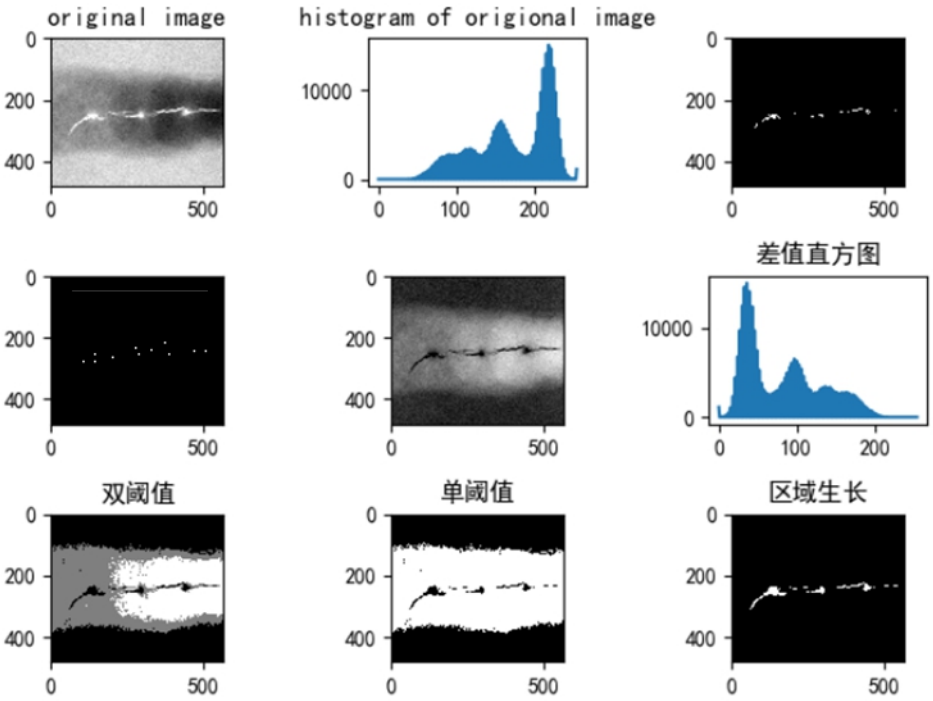
2) 区域生长时各像素点属性的确定

作出 255-原图之差的绝对值图像，用于计算每个位置(x, y)处的属性所需要的差值。再对其使用最小双阈值（仅使用 T1=68）进行阈值处理，其中像素为 0 的点在区域生长时属性为“真”，其他点是属性为“假”的像素。

3) 基于种子图像的区域生长算法

从先前的种子图像入手，根据预先定义的生长准则，不断判断种子 8 邻域内是否有符合条件的新种子，并将其添加至种子图像，最终获得生长区域。

下图是重现的结果：

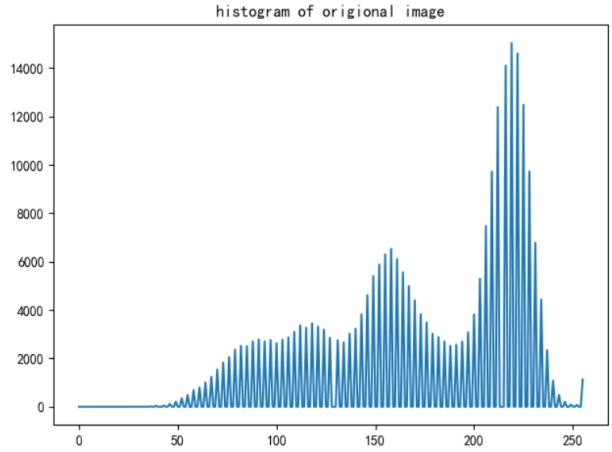


以下将从(b)-(i)图的绘制及其实现原理的角度进行步骤展示。

二、 实验步骤

1. 原图直方图的绘制

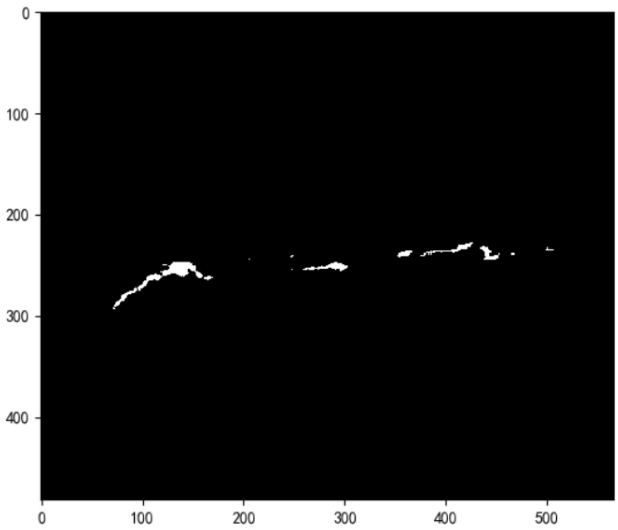
```
img = cv2.imread('pic.tif', 0)
hist1 = cv2.calcHist([img], [0], None, [256], [0, 256])
plt.figure(), plt.plot(hist1)
plt.title('histogram of origional image')
```



2. 初始种子图像的生成

在高百分比处（99.5%）设置阈值（254）对原始图像进行阈值处理，得到初始种子图像。

```
# 初始种子图像
_, speed = cv2.threshold(img, 254, 255, cv2.THRESH_BINARY)
plt.figure(), plt.imshow(speed, cmap='gray')
```



3. 最终种子图像的生成

对初始种子图像（speed）中的连通分量进行检测，并在各连通分量中任选一点，由这些单点形成的图像即为最终种子图像。

以下展示了二值图连通分量检测的代码，每次检测到一个新的连通分量时，记录下该连通分量的中的初始像素点，保存下来以生成最终的种子图像。具体的实现原理如下：

1. 创建和 speed 相同大小但是元素都为 0 的图像 B，并复制 speed 到 speed_copy 中；

2. 在 speed_copy 中任选一像素值为 255 的点，保存下像素点位置，接着使用连通分量算法，即在 B 中对应像素点进行膨胀，再和 speed 执行 and 操作（取交集），当算法收敛时，则检测出一个连通分量；

3. 将检测出来的连通分量复制到 B 中，将 speed_copy 中对应位置的值设为 0；

4. 重复三、四步，直到 speed_copy 中所有的像素值为 0，检测出所有连通分量。

```
ret, speed = cv2.threshold(gray_A, 254, 255, cv2.THRESH_BINARY)  # 灰度图转换成二值图像

speed_copy = speed.copy()  # 复制speed到speed_copy
B = np.zeros(speed.shape, np.uint8)  # B大小与speed相同，像素值为0

kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3,3))  # 3×3结构元

count = []  # 为了记录连通分量中的像素个数
point = []
```

```
while speed_copy.any():

    Xa_copy, Ya_copy = np.where(speed_copy > 0)  # speed_copy中值为255的像素的坐标
    B[Xa_copy[0]][Ya_copy[0]] = 255  # 选取第一个点，并将B中对应像素值改为255

    position = (Xa_copy[0], Ya_copy[0])
    print(position)
    point.append(position)

    for i in range(200):
        dilation_B = cv2.dilate(B, kernel, iterations=1)
        B = cv2.bitwise_and(speed, dilation_B)

    # 取B值为255的像素坐标，并将speed_copy中对应坐标像素值变为0
    Xb, Yb = np.where(B > 0)
    speed_copy[Xb, Yb] = 0
```

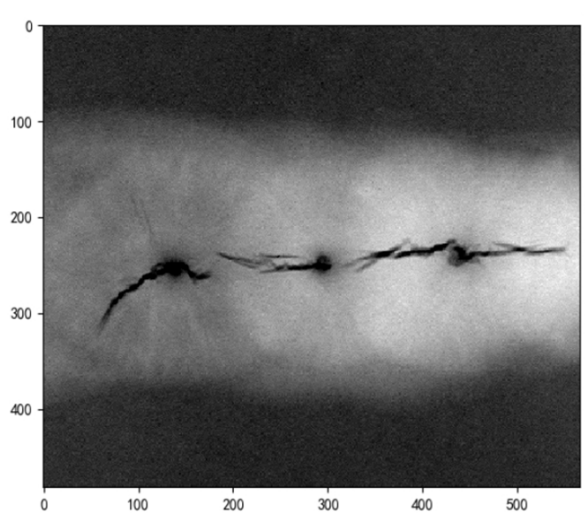
最后可以打印出检测到的连通分量个数以及单点的集合：

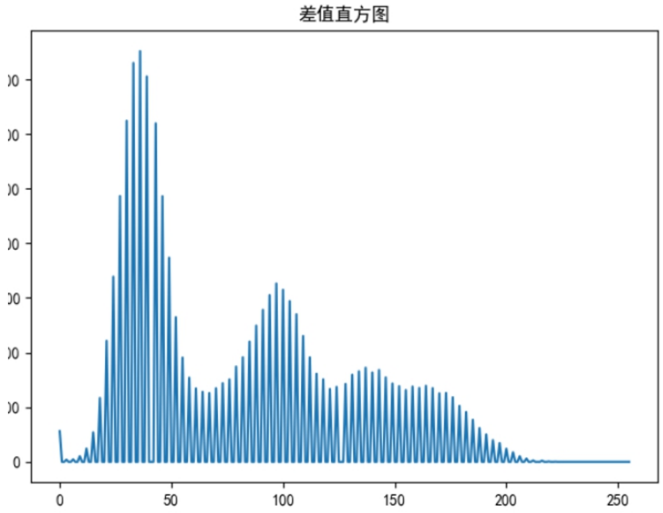
```
单点位置的集合为：
(228, 427)(229, 424)(232, 436)(233, 500)(236, 366)(236, 531)(239, 378)(239, 452)(239, 465)(241, 248)(244, 203)(245,
 291)(248, 131)(248, 290)(250, 121)(255, 248)(262, 165)(273, 93)(292, 72)连通分量有19个
```

由结果可知，图中共有 19 个连通分量。

4. 差值图像的生成及其直方图绘制

```
attribute = abs(255 - img)
plt.figure(), plt.imshow(attribute, cmap='gray')

hist2 = cv2.calcHist([attribute], [0], None, [256], [0, 256])
plt.figure(), plt.plot(hist2)
plt.title('差值直方图')
```

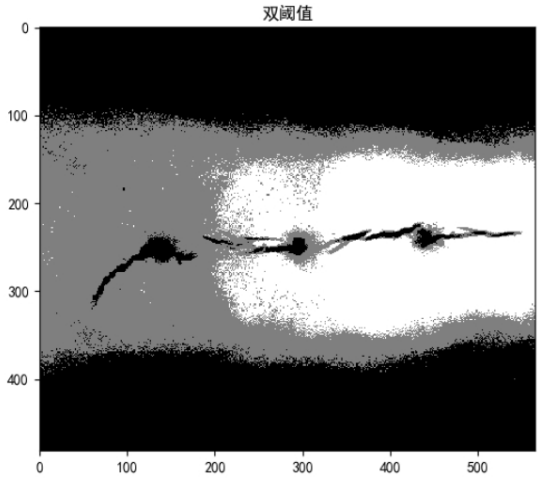




5. 差值图像的双阈值处理

如下是自定义编写的双阈值处理函数，它接收两个阈值和图像矩阵，当像素值位于三个不同区间时，分别被赋予不同的新像素值。根据直方图，我们选取双阈值为 68, 126。

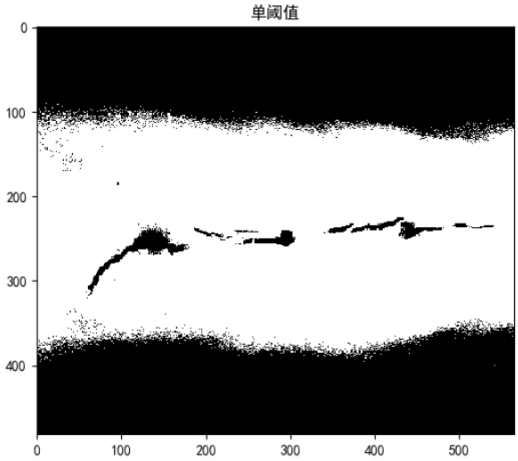
```
# 双阈值
def double(t1, t2, attribute):
    double_pic = np.zeros(attribute.shape, np.uint8)
    for i in range(0, attribute.shape[0]):
        for j in range(0, attribute.shape[1]):
            if attribute[i, j] < t1:
                double_pic[i, j] = 0
            elif attribute[i, j] > t1 and attribute[i, j] < t2:
                    double_pic[i, j] = 127
            else:
                double_pic[i, j] = 255
    plt.figure(), plt.imshow(double_pic, cmap='gray')
    plt.title('双阈值')


double(68, 126, attribute)
```



6. 差值图像的单阈值处理

只对差值图像进行单阈值处理（68）。得到的新图中像素为 0 的点在区域生长时属性为“真”，其他点是属性为“假”的像素。

```
# 单阈值
_, single = cv2.threshold(attribute, 68, 255, cv2.THRESH_BINARY)
plt.figure(), plt.imshow(single, cmap='gray'), plt.title('单阈值')
```



7. 区域生长

程序思路：生成一个同样大小空白矩阵，设置种子点的位置为 1，通过计算判断各种子点 8 邻域内是否有符合条件新种子，有则加入，直到算法停止。

```
origin = cv2.imread("new.png", 0)
im_array = np.array(origin)
m, n = origin.shape
a = np.zeros((m, n))  # 建立等大小空矩阵
a[228, 427] = 1; a[229, 424] = 1; a[232, 436] = 1
a[233, 500] = 1; a[236, 366] = 1; a[236, 531] = 1
a[239, 378] = 1; a[239, 452] = 1; a[239, 465] = 1
a[241, 248] = 1; a[244, 203] = 1; a[245, 291] = 1
a[248, 131] = 1; a[248, 290] = 1; a[250, 121] = 1
a[255, 248] = 1; a[262, 165] = 1; a[273, 93] = 1
a[292, 72] = 1
```

```
flag = 1
while flag == 1:
    flag = 0
    for i in range(2, m):
        for j in range(2, n):
            if a[i, j] == 1:
                for x in range(-1, 2):
                    for y in range(-1, 2):
                        if a[i+x, j+y] == 0:
                            value = single[i+x, j+y]
                            if value == 0:
                                flag = 1
                                a[i+x, j+y] = 1
```

最终得到区域生长的效果图如下：