《数字图像处理》实验报告

时间: 2021 年 10 月 30 日（第 9 周，星期 六 ）

地点： 测绘工程实验室（B 区）-- B 实验大楼 202

学生姓名： 魏子继 学号： 20194947

一、 实验名称：

大津算法实现图像二值化

二、 实验目的：

该实验的目的是使用大津算法（otsu）实现图像自适应二值化，要求算法能够根据图像灰度值大小，计算出自适应全局阈值，并根据此阈值对图像进行二值化处理；另一方面，能够计算出二值化图像的可分离测度（类间方差），对二值化结果进行评判，从而判断图像二值化结果的优劣。

大津法（OTSU）是一种确定图像二值化分割阈值的算法，由日本学者大津于 1979 年提出。从大津法的原理上来讲，该方法又称作最大类间方差法，因为按照大津法求得的阈值进行图像二值化分割后，前景与背景图像的类间方差最大。

它被认为是图像分割中阈值选取的最佳算法，计算简单，不受图像亮度和对比度的影响，因此在数字图像处理上得到了广泛的应用。它是按图像的灰度特性，将图像分成背景和前景两部分。因方差是灰度分布均匀性的一种度量,背景和前景之间的类

间方差越大,说明构成图像的两部分的差别越大,当部分前景错分为背景或部分背景错分为前景都会导致两部分差别变小。因此,使类间方差最大的分割意味着错分概率最小。

三、 实验步骤:

根据实验的目标和实现方法，该实验由以下三步完成:

1. 设计循环，将 0-255 的每一个灰度值作为阈值，将图像像素分为前景像素和后景像素进行处理。
2. 计算图像的前景像素和后景像素的平均灰度值。
3. 根据推到的数学公式计算类间方差和可分离测度，选择类间方差最大的灰度值作为算法的最优阈值。根据此最优阈值作为二值化阈值，对灰度图像进行二值化处理

四、 实验中的关键点分析（包括关键算法与代码实现）:

大津算法（OTSU.py）

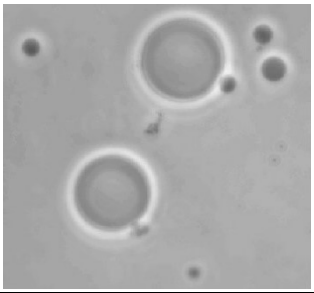
```
"""
Created by Chloe on 2021/10/30
"""

import cv2.cv2 as cv2
import numpy as np
import matplotlib.pyplot as plt

plt.rcParams['font.sans-serif'] = ['SimHei']  # 图像字体汉化


# 大津二值化，返回二值化图像
def otsu(img):
    # 灰度化
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

```

```
    max_gray = 0  # 存储灰度类间方差
    suitable_th = 0  # 存储灰度类间方差对应的阈值
    ave, total = cv2.meanStdDev(gray)  # meanStdDev 计算图像的均值和标准偏差，分别用 ave 和 total 存储

    # 遍历每一个灰度值，此时作为阈值
    for threshold in range(255):
        binary = gray > threshold  # 输出为 true/false，0/1
        binary_inv = gray <= threshold  # 输出为 true/false，0/1
        fore_pix = np.sum(binary)  # 前景像素总数(c1)
        back_pix = np.sum(binary_inv)  # 后景像素总数(c2)
        if fore_pix == 0:
            break
        if back_pix == 0:
            continue

        w0 = float(fore_pix) / gray.size  # gray.size 获得灰度图像总长度，即像素个数
        u0 = float(np.sum(gray * binary)) / fore_pix  # 前景灰度值总和，u0 为 c1 平均灰度
        w1 = float(back_pix) / gray.size
        u1 = float(np.sum(gray * binary_inv)) / back_pix

        class_Var = w0 * w1 * (u0 - u1) * (u0 - u1)  # class_Var 为类间方差
        # 取类间方差最大的对应阈值为最终阈值，并计算可分离测度
        if class_Var > max_gray:
            max_gray = class_Var
            suitable_th = threshold
            yita = float(class_Var / (total ** 2))  # yita 即为可分离测度值
    print("otsu 计算最适阈值为: %d, 可分离测度为: %.20f" % (suitable_th, yita))

    # 生成 otsu 二值化图像
    binary_otsu = np.zeros((height, width), gray.dtype)
    for i in range(height):
        for j in range(width):
            if gray[i, j] >= suitable_th:
                binary_otsu[i, j] = 255
            else:
                binary_otsu[i, j] = 0
    binary_otsu = cv2.cvtColor(binary_otsu, cv2.COLOR_BGR2RGB)
    return binary_otsu
```

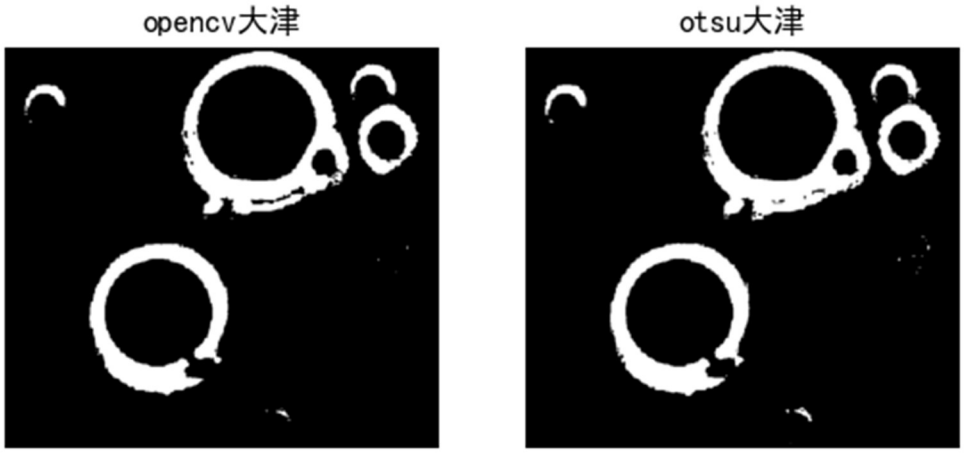
```
55.
56.
57. # 主程序
58. if __name__ == '__main__':
59.     # 读入图片
60.     image = cv2.imread("D:\\pyfiles\\DigitalImageProcessing\\image\\test.
    tif")
61.
62.     # 获取图像尺寸
63.     height, width = image.shape[0:2]
64.
65.     # opencv 大津算法二值化
66.     threshold_num, binary_th = cv2.threshold(cv2.cvtColor(image, cv2.COLO
    R_BGR2GRAY), 0, 255, cv2.THRESH_OTSU)
67.     binary_th = cv2.cvtColor(binary_th, cv2.COLOR_BGR2RGB)
68.     print('opencv 计算阈值 threshold_num:', threshold_num)
69.
70.     # otsu 二值化图像
71.     binary_ot = otsu(image)
72.
73.     # 显示图像
74.     titles = ['opencv 大津', 'otsu 大津']
75.     images = [binary_th, binary_ot]
76.     for i in range(2):
77.         plt.subplot(1, 2, i + 1)
78.         plt.imshow(images[i], cmap='gray')
79.         plt.title(titles[i])
80.         plt.axis('off')
81.     plt.savefig("D:\\pyfiles\\DigitalImageProcessing\\result\\OTSUImage1.
    png", bbox_inches='tight')
82.     plt.show()
```

五、 实验原始数据与实验结果：

源图像：



结果图像：



六、 问题分析与心得体会：

在实验过程中，主要遇到了以下问题并做出总结：

1. 根据 opencv 和自己写的大津算法代码相比较，在细节处还是有些许的差异。打印出 opencv 和自己写的算法阈值大小和可分离测度结果如下：

```
opencv计算阈值threshold_num: 181.0
otsu计算最适阈值为: 181, 可分离测度为: 0.46622919501901138872
```

由此结果能够得出，二值阈值大小相差无几，且自己写的代码的可分离测度和书上的 0.467 相差也不大。这个结果和班内其他同学相比也是一致的，因此认为结果是可信的。

2. 这个算法的最大问题我认为是时间问题。和灰度化一样，甚至比灰度化更加严重，此算法不仅对图像使用了 256 次 for 循环，并且在二值化过程中又使用了一次双 for 循环，无疑对代码时间上的消耗是巨大的。但对于此算法时间上如何减少，我和同学探讨许久，都未得出一个妥善的方法。