Task5 图像分割和二值化

图像分割就是把图像分成若干个特定的、具有独特性质的区域并提出感兴趣目标的技术和过程。

现有的图像分割方法主要有以下几类：

- 基于阈值的分割方法
- 基于区域的分割方法
- 基于边缘的分割方法
- 基于特定理论的分割方法

该部分的学习内容是对经典的阈值分割算法进行回顾，图像阈值化分割是一种传统的最常用的图像分割方法，因其实现简单、计算量小、性能较稳定而成为图像分割中最基本和应用最广泛的分割技术。它特别适用于目标和背景占据不同灰度级范围的图像。它不仅可以极大的压缩数据量，而且也大大简化了分析和处理步骤，因此在很多情况下，是进行图像分析、特征提取与模式识别之前的必要的图像预处理过程。图像阈值化的目的是要按照灰度级，对像素集合进行一个划分，得到的每个子集形成一个与现实景物相对应的区域，各个区域内部具有一致的属性，而相邻区域不具有这种一致属性。这样的划分可以通过从灰度级出发选取一个或多个阈值来实现。

一、最大类间方差法（大津阈值法）

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。

应用： 是求图像全局阈值的最佳方法，应用不言而喻，适用于大部分需要求图像全局阈值的场合。

优点： 计算简单快速，不受图像亮度和对比度的影响。

缺点： 对图像噪声敏感；只能针对单一目标分割；当目标和背景大小比例悬殊、类间方差函数可能呈现双峰或者多峰，这个时候效果不好。

求类间方差：

OTSU算法的假设是存在阈值TH将图像所有像素分为两类C1(小于TH)和C2(大于TH)，则这两类像素各自的均值就为m1、m2，图像全局均值为mG。同时像素被分为C1和C2类的概率分别为p1、p2。因此就有：

$$p1*m1+p2*m2=mG \quad (1)$$

$$p1+p2=1 \quad (2)$$

根据方差的概念，类间方差表达式为：

$$\sigma^2 = p1(m1-mG)^2 + p2(m2-mG)^2 \quad (3)$$

我们把上式化简，将式（1）代入式（3），可得：

$$\sigma^2 = p1p2(m1-m2)^2 \quad (4)$$

其实求能使得上式最大化的灰度级 k 就是OTSU阈值了，很多博客也是这样做的。

其中：

$$p1 = \sum_{i=0}^{k} p_i \quad (5)$$

$$m1 = 1/p1 * \sum_{i=0}^{k} ip_i \quad (6)$$

$$m2 = 1/p2 * \sum_{i=k+1}^{L-1} ip_i \quad (7)$$

照着公式，遍历0~255个灰度级，求出使式(4)最大的 k 就ok了。

根据原文，式（4）还可以进一步变形：

首先**灰度级K的累加均值m**和**图像全局均值mG**分别为：

$$m = \sum_{i=0}^{k} ip_i \quad (8)$$

$$mG = \sum_{i=0}^{L-1} ip_i \quad (9)$$

再瞅瞅式(6)，m1、m2就可变为：

$$m1 = 1/p1 * m \quad (10)$$

$$m2 = 1/p2 * (mG - m) \quad (11)$$

式(10)、(11)代入式（4），我们可得原文最终的类间方差公式：

$$\sigma^2 = \frac{(mG * p1 - m)^2}{p1(1-p1)} \quad (12)$$

根据公式（5）、（8）、（9）**求能使得上式(12)最大化的灰度级 k 就是OTSU阈值。**

分割：

这个分割就是二值化，OpenCV给了以下几种方式，很简单，可以参考：

- **THRESH_BINARY**

$$\mathtt{dst}(x,y) = \begin{cases} \mathtt{maxval} & \text{if } \mathtt{src}(x,y) > \mathtt{thresh} \\ 0 & \text{otherwise} \end{cases}$$

- **THRESH_BINARY_INV**

$$\mathtt{dst}(x,y) = \begin{cases} 0 & \text{if } \mathtt{src}(x,y) > \mathtt{thresh} \\ \mathtt{maxval} & \text{otherwise} \end{cases}$$

- **THRESH_TRUNC**

$$\mathtt{dst}(x,y) = \begin{cases} \mathtt{threshold} & \text{if } \mathtt{src}(x,y) > \mathtt{thresh} \\ \mathtt{src}(x,y) & \text{otherwise} \end{cases}$$

- **THRESH_TOZERO**

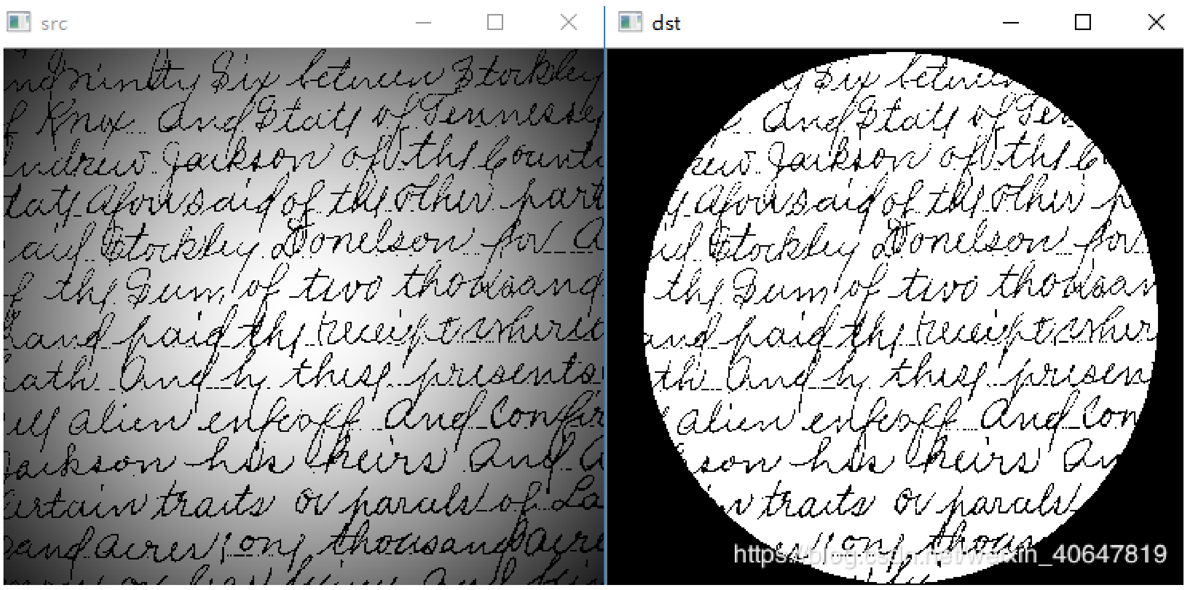
$$\mathtt{dst}(x,y) = \begin{cases} \mathtt{src}(x,y) & \text{if } \mathtt{src}(x,y) > \mathtt{thresh} \\ 0 & \text{otherwise} \end{cases}$$

- **THRESH_TOZERO_INV**

$$\mathtt{dst}(x,y) = \begin{cases} 0 & \text{if } \mathtt{src}(x,y) > \mathtt{thresh} \\ \mathtt{src}(x,y) & \text{otherwise} \end{cases}$$

二、自适应阈值

OTSU算法属于全局阈值法，对于某些光照不均的图像，这种全局阈值分割的方法会显得苍白无力。如图：



显然，这样的阈值处理结果不是我们想要的，那么就需要一种方法来应对这样的情况。

这种办法就是自适应阈值法(adaptiveThreshold)，它的思想不是计算全局图像的阈值，而是根据图像不同区域亮度分布，计算其局部阈值，所以对于图像不同区域，能够自适应计算不同的阈值，因此被称为自适应阈值法。(其实就是局部阈值法)

如何确定局部阈值呢？可以计算某个邻域(局部)的均值、中值、高斯加权平均(高斯滤波)来确定阈值。值得说明的是：如果用局部的均值作为局部的阈值，就是常说的移动平均法。

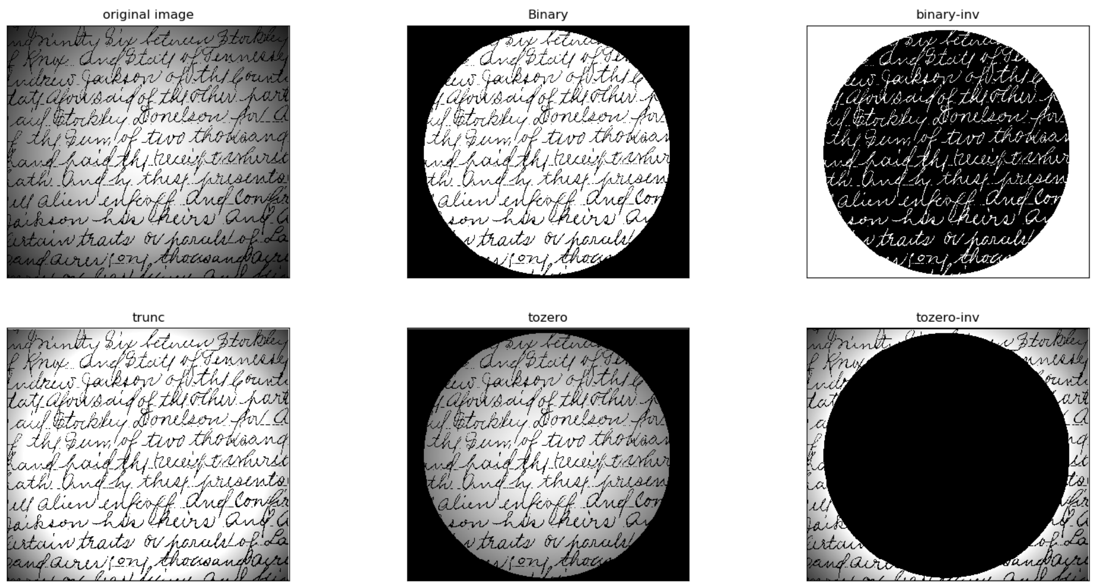
三、代码实现

用python实现

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

# 一、简单阈值法
img = cv2.imread('E:/PythonProgram/opencv_study/fig_transaction/light.PNG',0)
ret , thresh1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
ret , thresh2 = cv2.threshold(img,127,255,cv2.THRESH_BINARY_INV)
ret , thresh3 = cv2.threshold(img,127,255,cv2.THRESH_TRUNC)
ret , thresh4 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO)
ret , thresh5 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO_INV)

titles = ['original image','Binary','binary-inv','trunc','tozero','tozero-inv']
images = [img,thresh1,thresh2,thresh3,thresh4,thresh5]

for i in range(6):
    plt.subplot(2,3,i+1),plt.imshow(images[i],'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])

plt.show()
# 二、大津法
# ret , thresh1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
# b, g, r = cv2.split(img)
# img = cv2.merge([r, g, b])
# gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
ret1, th1 = cv2.threshold(img,0,255,cv2.THRESH_OTSU)
ret2, th2 = cv2.threshold(img,255,255,cv2.THRESH_OTSU)
imgs = np.hstack([img, th1, th2])
cv2.imshow('OTSU', imgs)
cv2.waitKey(0)
cv2.destroyAllWindows()
# plt.subplot(121), plt.imshow(th1)
# plt.subplot(122), plt.imshow(th2)
# plt.xticks([]),plt.yticks([])
# plt.show()
# cv2.imshow('OTSU',th1)
# cv2.waitKey(0)
# cv2.destroyAllWindows()
# cv2.imshow('OTSU',th2)
# cv2.waitKey(0)
# cv2.destroyAllWindows()

# 三、自适应阈值法
#中值滤波![5](E:\Github\GithubProject\ComputerVisionStudy\5\图像\5.PNG)
# img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# img = cv2.medianBlur(img,5)
ret , th1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
# 11为block size，2为C值
th2 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_MEAN_C , cv2.THRESH_BINARY,11,2)
th3 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C , cv2.THRESH_BINARY,11,2)
```

```python
imgs = np.hstack([img, th1, th2, th3])
cv2.imshow('OTSU', imgs)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

结果:


original image
Binary
binary-inv
trunc
tozero
tozero-inv