1. 大津算法介绍

阈值处理可视为一种统计决策理论问题，其目的是在把像素分配给两个或多个组的过程中引入的平均误差最小。将图像背景和前景分成黑白两类很好理解，但是如何确定背景和前景的二值化界限（阈值）呢？大津算法（OTSU）对此进行了全局最佳阈值处理。该方法在类间方差最大的情况下是最佳的。

2. 大津算法的分析与设计

大津算法包含以下几个阶段：

1）灰度化

灰度计算公式：（有多种算法，这里选取下式所示的一组 RGB 系数）

$$Gray[i, j] = 0.299 * R[i, j] + 0.587 * G[i, j] + 0.114 * B[i, j]$$

2）遍历处理每个灰度[0,255]

① 将当前遍历到的灰度值作为当前阈值 t

h：图像的宽度

w：图像的高度（h*w 得到图像的像素数量）

t ：灰度阈值（灰度大于这个值的像素将灰度设置为 255，小于的设置为 0）

n0：小于阈值的像素，前景

n1：大于等于阈值的像素，背景

n0 + n1 == h * w

② 分别统计阈值 t 下的前景、背景像素比例

w0：前景像素数量占总像素数量的比例

w0 = n0 / (h * w)

w1：背景像素数量占总像素数量的比例

w1 = n1 / (h * w)

w0 + w1 == 1

③ 计算前景、背景像素的平均灰度

u0：前景平均灰度

u0 = n0 灰度累加和 / n0

u1：背景平均灰度

u1 = n1 灰度累加和 / n1

④ 计算全局灰度均值

u：全局灰度均值

u = (n0 灰度累加和 + n1 灰度累加和) / (h * w) 根据上面的关系

u = w0 * u0 + w1 * u1

⑤ 计算阈值 t 下的类间方差

g：类间方差

g = w0 * (u0 - u)^2 + w1 * (u1 - u)^2

根据上面的关系，进一步推出等价公式：

g = w0 * w1 * (u0 - u1) ^ 2

⑥ 保留最佳全局阈值

当上一步中的类间方差最大时，此时的阈值 t 就是 OTUS 算法要找的最佳全局阈值。

3）阈值处理

使用 OTSU 算法得到的全局最佳阈值对图像进行处理：对于图像中大于该阈值的像素灰度值置为 255，对于图像中小于该阈值的像素灰度值置为 0，最终得到一副黑白两色的图像。

3. 测试示例

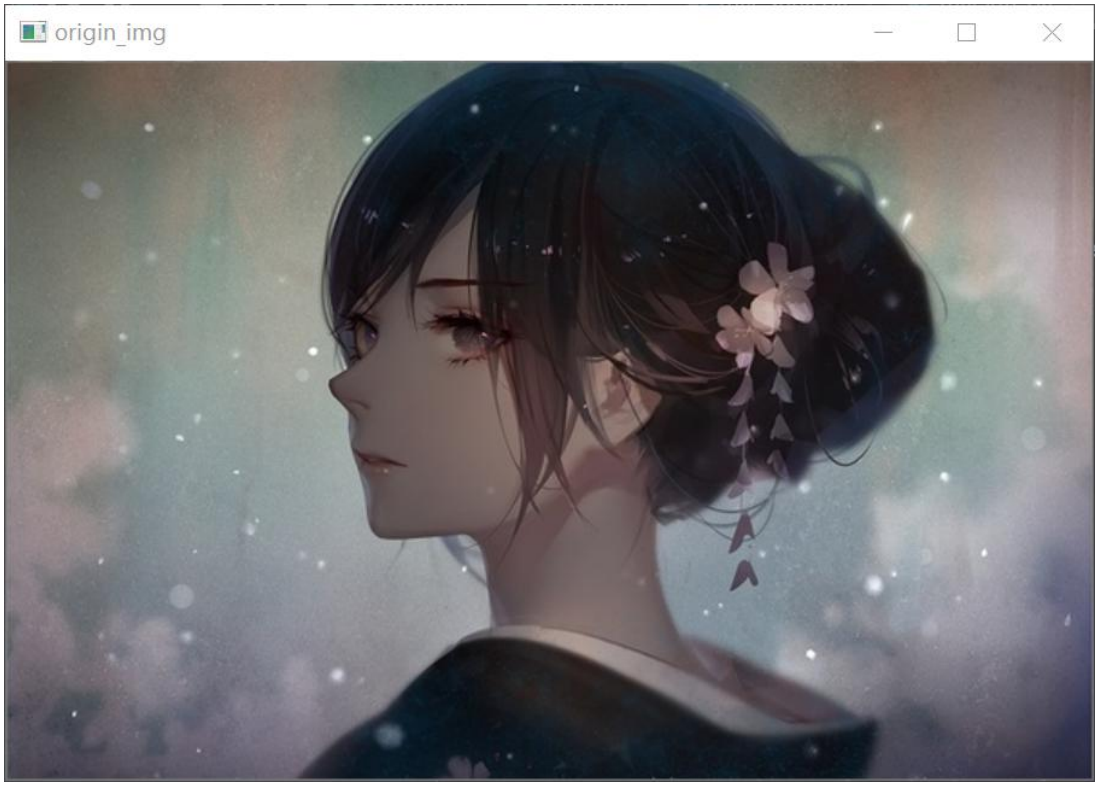

图 1 原图

图 2　自实现的 OTSU 算法处理结果

图 3　opencv 的 OTSU 算法处理结果

4. 对比分析

对比图 2 和图 3 的测试结果，可以发现自实现的 OTSU 算法和 opencv 自带的 OTSU 算法处理后得到的图像基本上效果一致。

我认为，自实现的 OTSU 算法对图像的处理效果能够到达这种不错效果，很大程度上是因为 OTSU 算法本身较为简单，算法的核心就是计算和比较类间方差，所以基本上只要按照类间方差最大找出阈值，就预期算法的效果都可以基本上达到 opencv 自带的 OTSU 算法的水平。

虽然，该 OTSU 算法的效果整体上还算不错，但是在一些细节方面的处理效果还可以进一步改善，比如：测试图像中的人物耳朵这一块。我认为 OTSU 算法按照类间方差最大的标准选取的阈值是一种很优秀的做法，但是再好的阈值也不可能将图像中的无数细节都全面考虑到（除非是很简单、几乎没有细节的图形），所以算法处理结果的误差是可以理解的。

5. 源代码

```
import cv2
import numpy as np


# 函数：灰度化
def make_gray(image):
    """
    RGB 转灰度计算公式：
        Gray(i,j) = 0.299 * R(i,j) + 0.587 * G(i,j) + 0.114 * B(i,j)
    """
    b = image[:, :, 0].copy()
    g = image[:, :, 1].copy()
    r = image[:, :, 2].copy()
    # RGB 转灰度
    gray_image = 0.299 * r + 0.587 * g + 0.114 * b
    gray_image = gray_image.astype(np.uint8)
    # 返回灰度图像
    return gray_image


# 函数：大津算法
def otsu(gray_image):
    """
        h：图像的宽度
        w：图像的高度
        （h*w 得到图像的像素数量）
        threshold_t ：灰度阈值（我们要求的值，大于这个值的像素我们将它的灰度设置
```

为 255，小于的设置为 0）

n0：小于阈值的像素数量，前景

n1：大于等于阈值的像素数量，背景

n0 + n1 == h * w

w0：前景像素数量占总像素数量的比例

w0 = n0 / (h * w)

w1：背景像素数量占总像素数量的比例

w1 = n1 / (h * w)

w0 + w1 == 1

u0：前景平均灰度

u0 = 前景灰度累加和 / n0

u1：背景平均灰度

u1 = 背景灰度累加和 / n1

u：平均灰度

u = (前景灰度累加和 + 背景灰度累加和) / (h * w)

u = w0 * u0 + w1 * u1

g：类间方差（那个灰度的 g 最大，哪个灰度就是需要的阈值 threshold_t）

g = w0 * (u0 - u)^2 + w1 * (u1 - u)^2

根据上面的关系，可以推出：

g = w0 * w1 * (u0 - u1) ^ 2

"""

```
    h = gray_image.shape[0]
    w = gray_image.shape[1]
    threshold_t = 0
    temp_t = 0

    # 遍历每一个灰度值
    for t in range(255):
        # 使用 numpy 直接对数组进行运算（注：其中包含了一些直方图的操作）
        n0 = gray_image[np.where(gray_image < t)]     # np.where：利用条件筛选前景和背景像素
        n1 = gray_image[np.where(gray_image >= t)]
        w0 = len(n0) / (h * w)   # len：返回数组长度，即：像素数量
        w1 = len(n1) / (h * w)
        u0 = np.mean(n0) if len(n0) > 0 else 0.   # np.mean：求平均值
        u1 = np.mean(n1) if len(n0) > 0 else 0.
        g = w0 * w1 * (u0 - u1) * (u0 - u1)   # 求类间方差
        # 判断此次遍历得到的方差是否足够大
        if g > temp_t:
            temp_t = g
            threshold_t = t
    print('类间方差最大时对应阈值：', threshold_t)
    # 返回 otus 的结果阈值
    return threshold_t
```

```
origin_img = cv2.imread('otusTest1.jpg')
cv2.imshow('origin_img', origin_img)

# 1. 将数据转换成 float32
img = origin_img.astype(np.float32)
# 2. 灰度化
gray_image = make_gray(image=img)
# 3. 执行 outs 算法并进行阈值处理
best_threshold_t = otsu(gray_image=gray_image)
# 4. 使用 outs 的结果进行阈值处理
otsu_img = gray_image
otsu_img[otsu_img < best_threshold_t] = 0    # 较小
otsu_img[otsu_img >= best_threshold_t] = 255    # 较大
cv2.imshow('otsu_img ', otsu_img)

cv2.waitKey(0)
cv2.destroyAllWindows()
```