

# 华北水利水电大学

North China University of Water Resources and Electric Power

## 《数字图像处理》 实验报告(4)

院 系： 信息工程学院  
专 业： 人工智能  
班 级： 2020185  
姓 名： 高树林  
学 号： 202018526  
指 导 教 师： 韩光辉

2022-2023 学年 第一学期

## 实验四 图像分割与特征提取

### 一、实验目的

- (1) 理解图像分割的目的与思想；
- (2) 理解并掌握灰度图像分割的几种经典方法；
- (3) 理解影响灰度图像分割的因素及相关解决方法。

### 二、实验内容和要求

实验报告内容应格式统一(注意同一个大纲级别下的字体、字号、行间距等应统一，英文用 Times New Roman 字体)，整体美观整洁。实验代码可读性强，包含适量注释，说明求解思路和过程。

#### 1. 基本全局阈值法

阅读并完善程序，要求采用基本全局阈值法(迭代算法)获得阈值，并进行图像阈值分割。将完善后的程序代码置于实验过程的代码部分。

提示：OpenCV 工具包相关函数

```
retval,dst = cv2.threshold(src,thresh,maxval,type,[,dst])
```

函数功能：对灰度图像进行阈值分割，产生二值图像。使用方法自行查阅官方文档。

代码补全：

```
# 添加导入所需工具包代码
```

```
# 迭代法（基本全局阈值法）求取阈值
```

```
# 输入的图像要求为灰度图像
```

```
img = cv2.imread(r'..\img\paopao.jpg',0) #使用此图像，据实修改路径
```

```
# 定义矩阵分别用来存放被阈值 T1 分开的两部分
```

```
G1 = np.zeros(img.shape, np.uint8)
```

```
G2 = np.zeros(img.shape, np.uint8)
```

```
T1 = np.mean(img) # 用图像均值做初始阈值
```

```
diff = 255
```

```
T0 = 0.01 #设置的最大阈值差
```

```
while (diff > T0):
```

```
    # THRESH_TOZERO 超过 thresh 的像素不变, 其他设为 0
```

```
    # THRESH_TOZERO_INV 与 THRESH_TOZERO 相反
```

```
    _, G1 = cv2.threshold(img, T1, 255, cv2.THRESH_TOZERO_INV)
```

```
    _, G2 = cv2.threshold(img, T1, 255, cv2.THRESH_TOZERO)
```

```
    loc1 = np.where(G1 > 0.001) # 可以对二维数组操作，获得 G1 部分非 0 像素的坐标
```

```

loc2 = _____
# g1 = list(filter(lambda a: a > 0, G1.flatten()))#只能对一维列表筛选,得到的
的是一个筛选对象
# g2 = list(filter(lambda a: a > 0, G2.flatten()))
ave1 = np.mean(G1[loc1]) #获得 G1 部分非 0 像素的均值
ave2 = _____
T2 = _____ #更新阈值
diff = _____ #计算前后两次迭代产生的阈值之间的差
T1 = T2
_,img_result = cv2.threshold(_____)
# 添加绘制结果展示代码

```

## 2. 含噪图像的分割

噪声会影响图像分割效果,一个简单的处理方法是对含噪声的图像进行平滑滤波,之后再行分割。

编程先给输入图像添加高斯噪声,然后对图像进行平滑后,再用 OTSU 算法对图像进行阈值分割,比较加入平滑步骤和没有平滑步骤的图像分割效果,结果展示部分应含有带噪声的图像、**噪声图像直方图**、噪声图像的 OTSU 分割、高斯平滑的图像、**平滑图像直方图**、平滑图像的 OTSU 分割。

自行编写函数实现直方图计算。

提示: 使用 cv2.threshold 函数可实现阈值分割。

## 3. 图像特征提取

图像特征提取是后续分类、预测等任务的基础,请使用第三方工具包中的特征提取函数,分别提取图像的 LBP 特征、HOG 特征,并将特征图与原图一起可视化输出。

提示: scikit-image 工具包含有提取 LBP 和 HOG 特征的函数,自行查找文档学习使用方法。

## 三、实验环境(学生填写)

(较详细地说明实验运行环境,包括操作系统、编程 IDE、编程语言及版本号、依赖的第三方类库及版本号,及其它可能影响重复实验的因素)

**硬件方面:** 本实验室基于处理器为 Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz 2.11 GHz, 基带 RAM 为 16.0 GB (15.8 GB 可用), 系统类型为 64 位操作系统, 基于 x64 的处理器 MagicBook 商务本上运行并得出相应结果和数据的。

**软件方面:** 本实验基于 windows10 家庭中文版, 版本号为 21H2, 操作系统

内部版本为 19044.2130 的 MagicBook 商务本。编程语言选择 python 语言，使用运行时版本号为 11.0.14.1+1-b2043.45 amd64 的 PyCharm 2022.1.1 (Professional Edition)集成开发环境编写代码。其中代码的 python 解释器版本为 Python 3.8.13 (default, Mar 28 2022, 06:59:08) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32。引入 1.0.0.14 版本的 OpenCV-python 库，其中集成的 OpenCV 版本为 4.6.0。引入 Matplotlib3.5.3 版本的库。

## 四、实验过程(学生填写)

### 1. 基本全局阈值法

#### (1)实验方案（设计思路说明）

使用全局阈值法（迭代法）来对图像进行分割，需要先设置一个初始值，之后按该初始值获得 G1,G2 的值，分别代表大于该灰度的像素点数目和小于改回度的像素点数目。之后计算平均灰度值，并将该值设置为新的初始值。之后判断这个值是否符合误差要求，若符合就选择这个值，因此需要写一个循环或者递归的函数。最后使用 cv2.threshold 函数可实现阈值分割，得到符合题目要求的二值图。

#### (2)代码编写

```
1. import numpy as np
2. import cv2
3. from matplotlib import pyplot as plt
4.
5. img = cv2.imread(r'../figures/img/paopao.jpg', 0)
6. eps = 0.01 # 精度
7. iry = np.array(img)
8. r, c = img.shape
9. avg = 0
10.
11. T = np.mean(img) # 用图像均值做初始阈值
12. dis = 255
13. while dis >= eps:
14.     G1, G2, cnt1, cnt2 = 0, 0, 0, 0 # G1 用来记录像素大于阈值的点的灰度和，G2 用来记录
        像素小于阈值的点的灰度和
15.     for i in range(r): # cnt1 用来记录大于阈值的点数，cnt2 用来记录小于阈值点数
16.         for j in range(c):
17.             if iry[i][j] >= T:
18.                 G1 += iry[i][j]; cnt1 += 1
19.             else:
20.                 G2 += iry[i][j]; cnt2 += 1
21.     T2 = (int(G1 / cnt1) + int(G2 / cnt2)) / 2
22.     dis = abs(T2 - T)
23.     T = T2
24. new_img = np.zeros((r, c), np.uint8) # 初始化 new_img
```

```
25. _, img_result = cv2.threshold(img, T, 255, cv2.THRESH_BINARY)
26.
27. plt.subplot(221), plt.imshow(img, 'gray')
28. plt.subplot(222), plt.imshow(img_result, 'gray')
29. # 绘制原图直方图并显示最佳阈值
30. plt.subplot(223)
31. plt.hist(img.ravel(), 256)
32. plt.title('hist')
33. plt.axvline(T) # 绘制最佳阈值分割线
34. plt.text(25, 6100, "Best Threshold: {}".format(T), size=15, alpha=0.8)
35.
36. plt.subplot(224)
37. plt.hist(img_result.ravel(), 256)
38. plt.title('hist')
39. plt.show()
40. # 添加绘制结果展示代码
```

(3)调试（编码实现过程中遇到的错误，及调试解决等内容）

**问题 1：**在调试过程中，我按照实验报告给出的代码运行得到的结果是全黑的图像，如下图 1 所示。

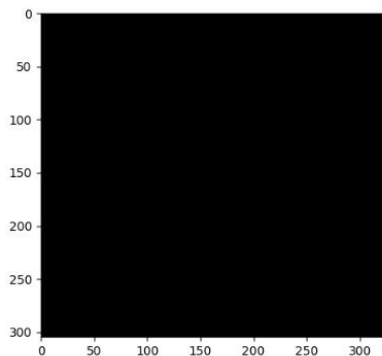


图 1 问题 1 所述错误图像

**解决 1：**出现这个问题的原因就是代码问题，在给出代码中，`loc1 = np.where(G1 > 0.001)`这个语录返回的值是空的，并不会返回预期中的大于 0 像素点的坐标。因此我没有采纳上述代码，自己按照全局阈值化的思想修改了代码，结果成功运行出了结果。

(4)实验结果（实验运行结果截图等体现实验结果的内容）

**结果 1：**使用 matplotlib 工具包中的 `plt.hist` 函数可以画出源图像的直方图。其结果如下图 2 所示。

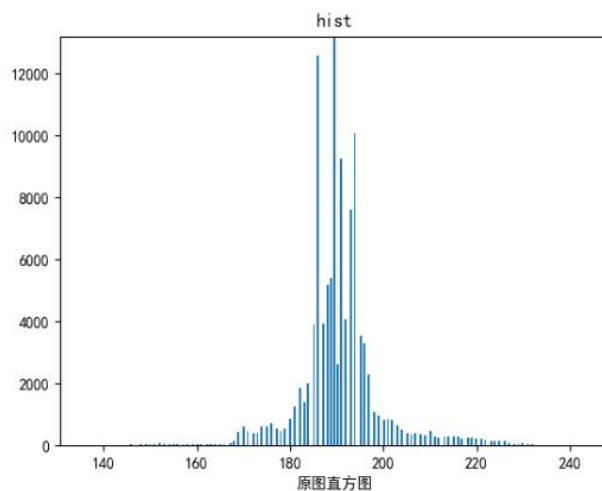


图2 原图像的直方图

**结果2：**使用全局阈值法迭代最终的阈值像素维189.5，以该灰度值做二值图得到的结果如下图3所示。

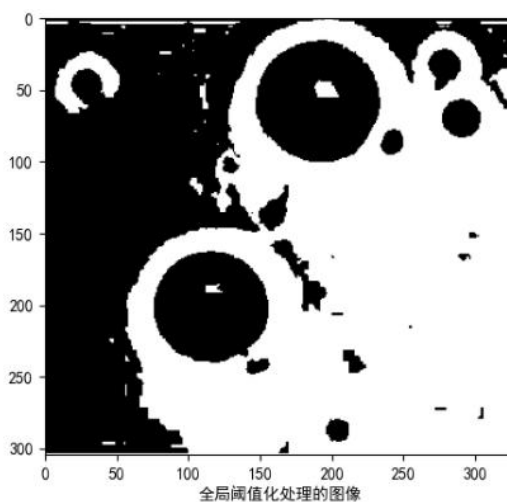


图3 全局阈值化处理得到的图像

**结果3：**对结果2中经过全局阈值化处理得到的图像进行处理的到其对应的直方图如下图4所示。

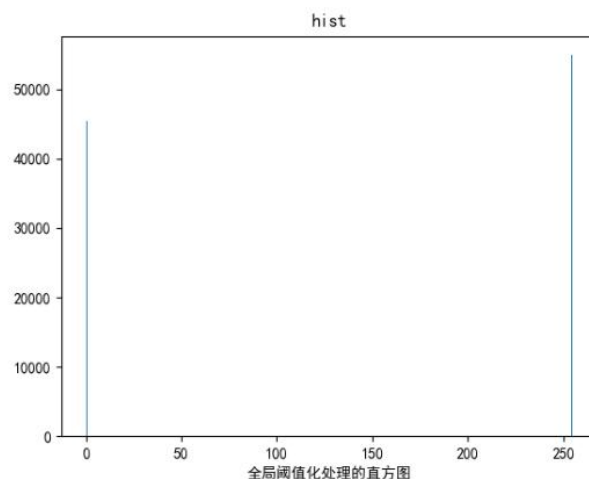


图 4 全局阈值化处理的直方图

**结果 4:** 将原图和上述结果放在一起作比较更能直观看出其变换的细节，其结果如下图 5 所示。

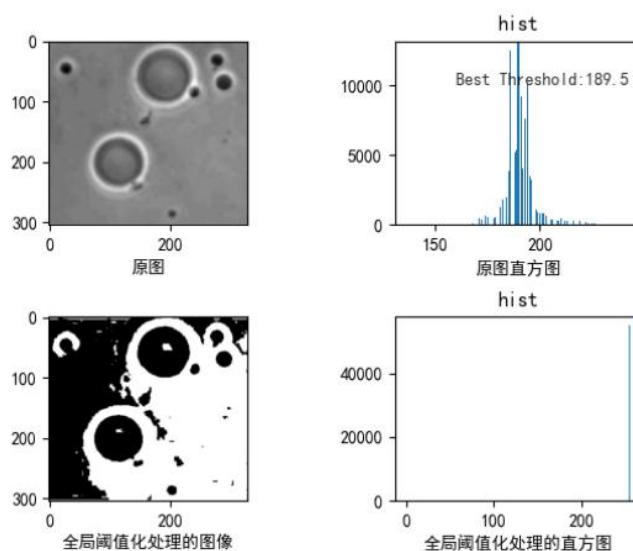


图 5 原图、二值图与其对应的直方图

#### (5)分析与总结（与本次实验相关的分析与总结）

通过本次实验，我掌握了如何利用全局阈值法进行图像分割，并总结了以下几步：

- (1)为全局值选择一个初始值  $T$ ;
- (2)用阈值  $T$  对图像进行分割，将图像分为 2 组像素:  $G1$  由灰度值大于  $T$  的所有像素组成， $G2$  由所有小于等于  $T$  的像素组成。
- (3)对  $G1$  和  $G2$  分别计算平均灰度值  $m1$  和  $m2$
- (4)计算一个新的阈值:  $T=(m1+m2)/2$
- (5)重复步骤 (2) 到 (4)，直到相邻 2 次的  $T$  值之差小于预定的参数  $\epsilon$  为止。

通过上述步骤就能够得到正确的实验结果。在此次实验，我很少采用已给代

码,但是我了解了二值化原理,通过自己对每个像素点的遍历该值,同样达到了完成实验的结果。因此在数字图像处理的道路上也并非是一成不变的,正所谓条条大路通罗马。但是无论怎么改变,它的实质激怒是对像素点的修改和找到适合二值化的像素点的值。

## 2. 含噪图像的分割

### (1)实验方案(设计思路说明)

本实验分为两块,每一块分为三步。第一步:添加高斯噪声,通过 `skimage` 库可以直接实现该操作;第二步:做出噪声图的直方图,在上一个实验中我们提到,可以用 `plt.hist()` 函数做出直方图;第三步,做出噪声图的 OSTU 单域阈值分割后的图像。第二块首先要对含噪图像处理得到平滑后的图像,之后按照第一块中的 2、3 步就能完成对应操作。

### (2)代码编写

```
1. def ww(path):
2.     image = cv2.imread(path)
3.     gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
4.     image = skimage.util.random_noise(gray, mode='gaussian', var=0.01,
        mean=0) # 添加高斯噪声
5.     smooth = cv2.GaussianBlur(image, (3, 3), 1.3) # 高斯滤波(平滑)
6.
7.
8.     plt.subplot(131)
9.     plt.imshow(image, "gray")
10.    plt.title("source image")
11.    plt.xlabel('噪声图')
12.    plt.subplot(132)
13.    plt.hist(image.ravel(), 256)
14.    plt.title("Histogram")
15.    plt.xlabel('噪声直方图')
16.    ret1, th1 = cv2.threshold(gray, 0, 255, cv2.THRESH_OTSU) # 方法选
        择为 THRESH_OTSU
17.    plt.subplot(133)
18.    plt.imshow(th1, "gray")
19.    plt.title("OTSU,threshold is " + str(ret1))
20.    plt.xticks([])
21.    plt.yticks([])
22.    plt.xlabel('OTSU 分割图')
23.
24.
25.    plt.subplot(234)
26.    plt.imshow(smooth,'gray')
27.    plt.title('高斯平滑处理')
28.    plt.subplot(235)
```



```

29.     plt.hist(smooth.ravel(), 256)
30.     plt.title("Histogram")
31.     plt.xlabel('滤波后直方图')
32.     smooth = (255 * smooth).astype(np.uint8)
33.     ret1, th1 = cv2.threshold(smooth, 0, 255, cv2.THRESH_OTSU) # 方法
        选择为 THRESH_OTSU
34.     plt.subplot(236)
35.     plt.imshow(th1, "gray")
36.     plt.title("OTSU,threshold is " + str(ret1))
37.     plt.xticks([])
38.     plt.yticks([])
39.     plt.xlabel('OTSU 分割图')
40.
41.
42. if __name__ == '__main__':
43.     path = r'../figures/img/paopao.jpg'
44.     ww(path)
45.     delect(path)
46.     plt.show()
    
```

(3)调试（编码实现过程中遇到的错误，及调试解决等内容）

**问题 1：**在我使用高斯滤波的时候，出现了如图 6 所时的报错。

```

ret1, th1 = cv2.threshold(smooth, 0, 255, cv2.THRESH_OTSU) # 方
cv2.error: OpenCV(4.4.0) C:\Users\appveyor\AppData\Local\Temp\1\pip-
> THRESH_OTSU mode:
>     'src_type == CV_8UC1 || src_type == CV_16UC1'
> where
>     'src_type' is 6 (CV_64FC1)
    
```

图 6 高斯滤波报错

**解析：**这是因为经过高斯低通滤波后，得到的数组类型是 float64 类型，该类型的数组不能够被 cv2.threshold()处理，因为 cv2.threshold()函数只接受 unit8 格式的数组，因此需要将 float64 的数组转化为 unit8 类型就能解决问题。解决的截图如下图 7 所示。

```

plt.xlabel('滤波后直方图')
smooth = (255 * smooth).astype(np.uint8) 这一行是修改数组类型
ret1, th1 = cv2.threshold(smooth, 0, 255, cv2.THRESH_OTSU) # 方法选择为THRESH_OTSU
plt.subplot(236) 只接受unit8类型的数据
    
```

图 7 问题 1 的解决方案

(4)实验结果（实验运行结果截图等体现实验结果的内容）

**结果 1：**向原图添加高斯噪声的结果如下图 8 所示。

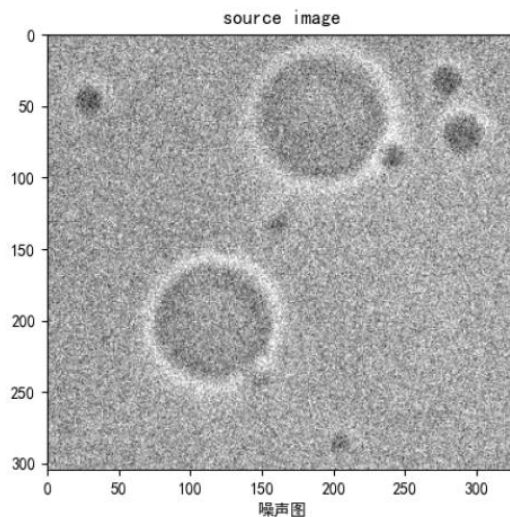


图 8 添加高斯噪声的结果

**结果 2:** 添加高斯噪声后的直方图如下如 9 所示。

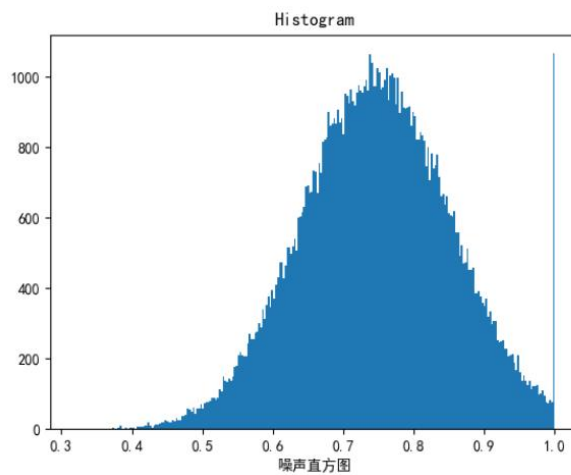


图 9 添加高斯噪声后的直方图

**结果 3:** 添加高斯噪声后利用 OTSU 算法分割图像，得到的二值图的结果如下图 10 所示。

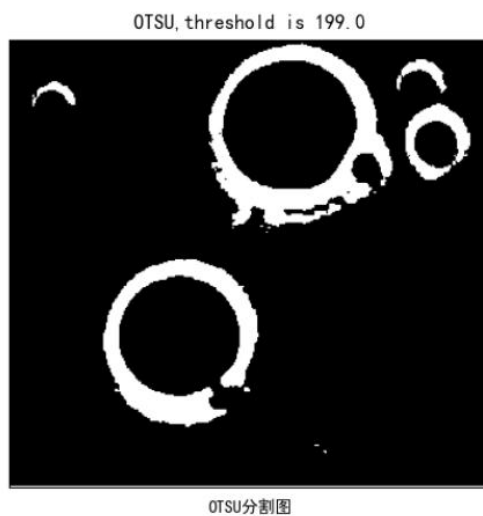


图 10 OTSU 算法分割图像的结果

**结果 4:** 利用高斯平滑处理，对高斯噪声的图像进行滤波，得到的结果如下图 11 所示。

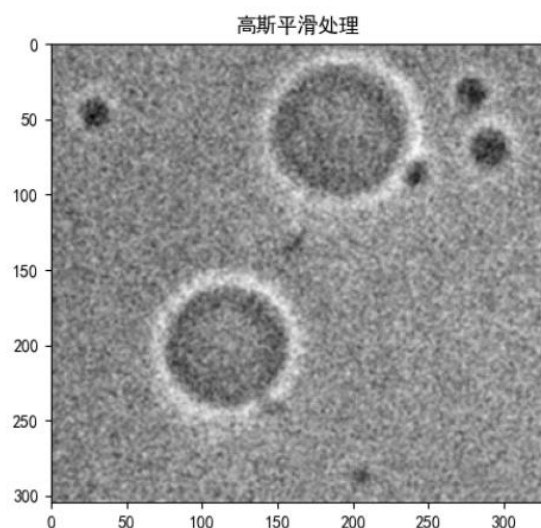


图 11 高斯平滑处理得到的图像

**结果 5:** 利用高斯滤波后对滤波后的图像取直方图得到的直方图图像如下图 12 所示。

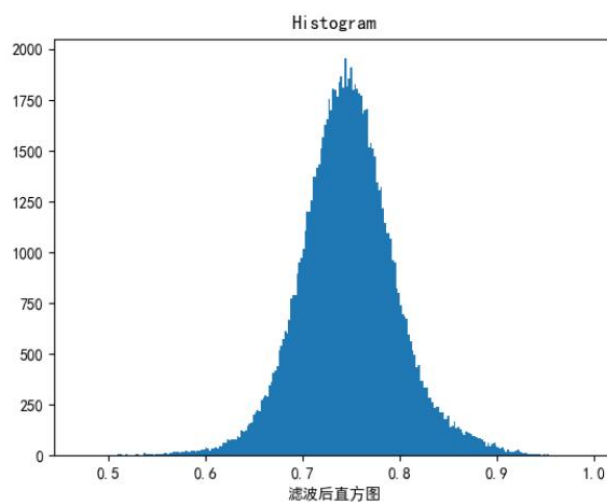


图 12 滤波后的直方图

**结果 6:** 使用高斯平滑后利用 OTSU 算法分割图像，得到的二值图的结果如下图 13 所示。

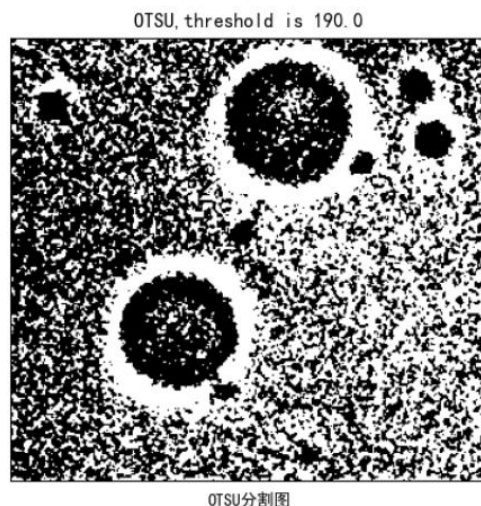


图 13 平滑处理后的二值图

**结果 7:** 为了方便比较，将实验 2 的图像整理到同一张图上。得到的结果如下图所示。

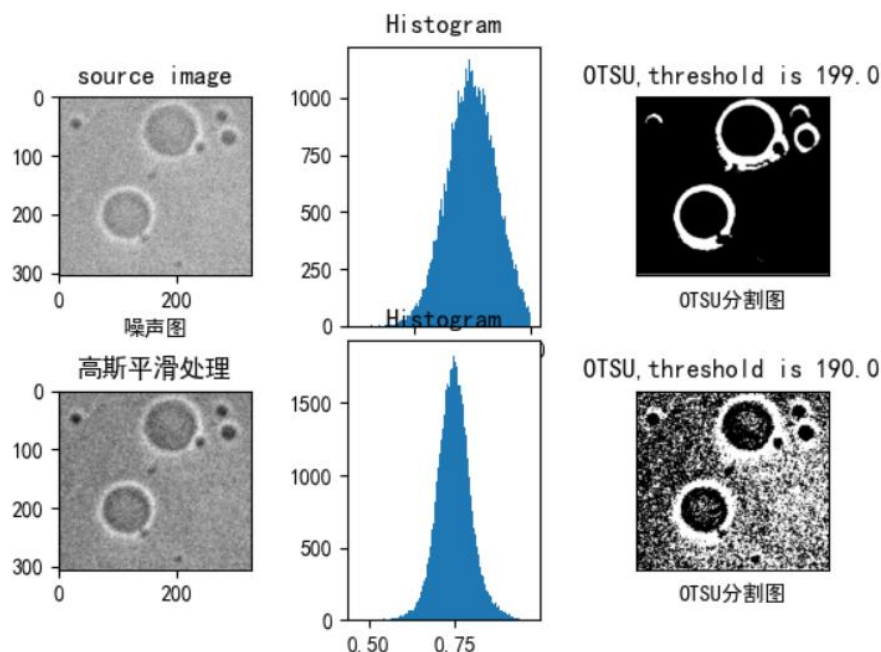


图 14 实验 2 图像

#### (5)分析与总结（与本次实验相关的分析与总结）

通过本次实验，我学会了怎么给图像加噪声和去除噪声。此外，可以通过 `plt.hist()` 函数可以做出对应图像的直方图。另外，我对 OSTU 和全局阈值法对图像进行分割的方法有了新的认识。下面谈一下看法：

根据前两个图像直方图，可以看到 Otsu 方法的分割明显要优于基本的全局阈值处理，Otsu 方法成功地将细胞边界分割了出来，实现了图像分割的预定目标。在没有明显的波谷时，采用基本的全局阈值处理并不能得到预想的图像分割效果，这是因为阈值  $T$  不能通过迭代很好的确定导致的，这时我们就要采用 Otsu 方法进行图像分割。

### 3. 图像特征提取

#### (1)实验方案（设计思路说明）

##### HOG 特征提取：

将一个 image（你要检测的目标或者扫描窗口）：

- 1) 灰度化（将图像看做一个  $x,y,z$ （灰度）的三维图像）；
- 2) 采用 Gamma 校正法对输入图像进行颜色空间的标准化（归一化）；目的是调节图像的对比度，降低图像局部的阴影和光照变化所造成的影响，同时可以抑制噪音的干扰；
- 3) 计算图像每个像素的梯度（包括大小和方向）；主要是为了捕获轮廓信息，同时进一步弱化光照的干扰。
- 4) 将图像划分成小 cells（例如  $6*6$  像素/cell）；
- 5) 统计每个 cell 的梯度直方图（不同梯度的个数），即可形成每个 cell 的 descriptor；
- 6) 将每几个 cell 组成一个 block（例如  $3*3$  个 cell/block），一个 block 内所有 cell 的特征 descriptor 串联起来便得到该 block 的 HOG 特征 descriptor。
- 7) 将图像 image 内的所有 block 的 HOG 特征 descriptor 串联起来就可以得到该 image（你要检测的目标）的 HOG 特征 descriptor 了。这个就是最终的可供分类使用的特征向量了。

##### LBP 特征：

- （1）首先将检测窗口划分为  $16 \times 16$  的小区域（cell）；
- （2）对于每个 cell 中的一个像素，将相邻的 8 个像素的灰度值与其进行比较，若周围像素值大于中心像素值，则该像素点的位置被标记为 1，否则为 0。这样， $3*3$  邻域内的 8 个点经比较可产生 8 位二进制数，即得到该窗口中心像素点的 LBP 值；
- （3）然后计算每个 cell 的直方图，即每个数字（假定是十进制数 LBP 值）出现的频率；然后对该直方图进行归一化处理。
- （4）最后将得到的每个 cell 的统计直方图进行连接成为一个特征向量，也就是整幅图的 LBP 纹理特征向量；

#### (2)代码编写

##### HOG 提取特征的代码如下：

```
1. class Hog_descriptor():
2.     def __init__(self, img, cell_size=16, bin_size=8):
3.         self.img = img
4.         self.img = np.sqrt(img / np.max(img))
5.         self.img = img * 255
6.         self.cell_size = cell_size
7.         self.bin_size = bin_size
8.         self.angle_unit = 360 / self.bin_size
9.
10.    def extract(self):
11.        height, width = self.img.shape
12.        # 计算图像的梯度大小和方向
13.        gradient_magnitude, gradient_angle = self.global_gradient()
14.        gradient_magnitude = abs(gradient_magnitude)
```

```

15.         cell_gradient_vector = np.zeros((int(height / self.cell_size),
        int(width / self.cell_size), self.bin_size))
16.         for i in range(cell_gradient_vector.shape[0]):
17.             for j in range(cell_gradient_vector.shape[1]):
18.                 # cell 内的梯度大小
19.                 cell_magnitude = gradient_magnitude[i * self.cell_size:
        (i + 1) * self.cell_size,
20.                 j * self.cell_size:(j + 1) * self.cell_size]
21.                 # cell 内的梯度方向
22.                 cell_angle = gradient_angle[i * self.cell_size:(i + 1)
        * self.cell_size,
23.                 j * self.cell_size:(j + 1) * self.cell_size]
24.                 # 转化为梯度直方图格式
25.                 cell_gradient_vector[i][j] = self.cell_gradient(cell_
        magnitude, cell_angle)
26.
27.                 # 绘制梯度直方图
28.                 hog_image = self.render_gradient(np.zeros([height, width]), c
        ell_gradient_vector)
29.
30.                 # block 组合、归一化
31.                 hog_vector = []
32.                 for i in range(cell_gradient_vector.shape[0] - 1):
33.                     for j in range(cell_gradient_vector.shape[1] - 1):
34.                         block_vector = []
35.                         block_vector.extend(cell_gradient_vector[i][j])
36.                         block_vector.extend(cell_gradient_vector[i][j + 1])
37.                         block_vector.extend(cell_gradient_vector[i + 1][j])
38.                         block_vector.extend(cell_gradient_vector[i + 1][j + 1]
        )
39.                         mag = lambda vector: math.sqrt(sum(i ** 2 for i in ve
        ctor))
40.                         magnitude = mag(block_vector)
41.                         if magnitude != 0:
42.                             normalize = lambda block_vector, magnitude: [elem
        ent / magnitude for element in block_vector]
43.                             block_vector = normalize(block_vector, magnitude)
44.                         hog_vector.append(block_vector)
45.                 return hog_vector, hog_image
46.
47.     def global_gradient(self):

```



```

48.         gradient_values_x = cv2.Sobel(self.img, cv2.CV_64F, 1, 0, ksize=5)
49.         gradient_values_y = cv2.Sobel(self.img, cv2.CV_64F, 0, 1, ksize=5)
50.         gradient_magnitude = cv2.addWeighted(gradient_values_x, 0.5,
gradient_values_y, 0.5, 0)
51.         gradient_angle = cv2.phase(gradient_values_x, gradient_values
_y, angleInDegrees=True)
52.         return gradient_magnitude, gradient_angle
53.
54.     def cell_gradient(self, cell_magnitude, cell_angle):
55.         orientation_centers = [0] * self.bin_size
56.         for i in range(cell_magnitude.shape[0]):
57.             for j in range(cell_magnitude.shape[1]):
58.                 gradient_strength = cell_magnitude[i][j]
59.                 gradient_angle = cell_angle[i][j]
60.                 min_angle, max_angle, mod = self.get_closest_bins(gra
dient_angle)
61.                 orientation_centers[min_angle] += (gradient_strength
* (1 - (mod / self.angle_unit)))
62.                 orientation_centers[max_angle] += (gradient_strength
* (mod / self.angle_unit))
63.         return orientation_centers
64.
65.     def get_closest_bins(self, gradient_angle):
66.         idx = int(gradient_angle / self.angle_unit)
67.         mod = gradient_angle % self.angle_unit
68.         return idx, (idx + 1) % self.bin_size, mod
69.
70.     def render_gradient(self, image, cell_gradient):
71.         cell_width = self.cell_size / 2
72.         max_mag = np.array(cell_gradient).max()
73.         for x in range(cell_gradient.shape[0]):
74.             for y in range(cell_gradient.shape[1]):
75.                 cell_grad = cell_gradient[x][y]
76.                 cell_grad /= max_mag
77.                 angle = 0
78.                 angle_gap = self.angle_unit
79.                 for magnitude in cell_grad:
80.                     angle_radian = math.radians(angle)
81.                     x1 = int(x * self.cell_size + magnitude * cell_wi
dth * math.cos(angle_radian))
82.                     y1 = int(y * self.cell_size + magnitude * cell_wi
dth * math.sin(angle_radian))

```

```

83.             x2 = int(x * self.cell_size - magnitude * cell_wi
                    dth * math.cos(angle_radian))
84.             y2 = int(y * self.cell_size - magnitude * cell_wi
                    dth * math.sin(angle_radian))
85.             cv2.line(image, (y1, x1), (y2, x2), int(255 * mat
                    h.sqrt(magnitude)))
86.             angle += angle_gap
87.         return image
    
```

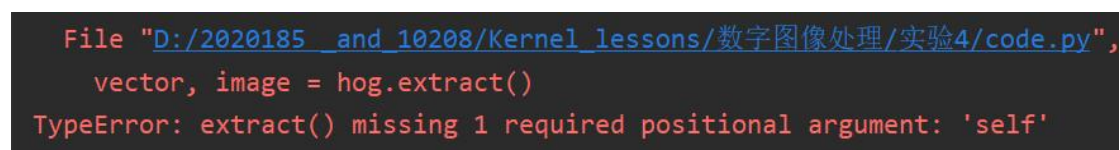
**LBP 提取特征的代码如下：**

```

1. def LBP_feature(path):
2.     radius = 3
3.     n_points = 8 * radius
4.
5.     image = cv2.imread(path)
6.     image1 = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
7.     plt.subplot(131)
8.     plt.imshow(image1, cmap='gray')
9.
10.    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
11.    # print(image.shape)
12.    plt.subplot(132)
13.    plt.imshow(image, cmap='gray')
14.    lbp = local_binary_pattern(image, n_points, radius, method='unifo
        rm')
15.    plt.subplot(133)
16.    plt.imshow(lbp, 'gray')
17.
18.    n_bins = int(lbp.max() + 1)
19.    lbp_hist, _ = np.histogram(lbp, density=True, bins=n_bins,
20.                               range=(0, n_bins)) # 绘制直方图, 百分比
21.    plt.plot(lbp_hist)
    
```

(3)调试（编码实现过程中遇到的错误，及调试解决等内容）

**问题 1：**在调试过程中我遇到了如下图 15 所示的报错。



```

File "D:/2020185_and_10208/Kernel_lessons/数字图像处理/实验4/code.py",
    vector, image = hog.extract()
TypeError: extract() missing 1 required positional argument: 'self'
    
```

图 15 问题 1 报错截图

**解决 1：**出现这个报错的原因是在定义类的时候，没有加括号，不加括号就是不经过实例化，结果是将这个类直接赋值给了变量，其实质是将变量的指针指向类的地址，但这个并不是实例化，要给这个类赋新值才是实例化，如果原类中不需要参数，括号里面可以为空，但是括号不能省略。修改之后运行正确。修改后的截图如下图 16 所示。



```
img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
hog = Hog_descriptor()
vector, image = hog.extract()
# 输出图像的特征向量shape
```

实例化

图 16 问题 1 解决方法

(4)实验结果（实验运行结果截图等体现实验结果的内容）

**结果 1：**提取的 HOG 特征如下图 17 所示。

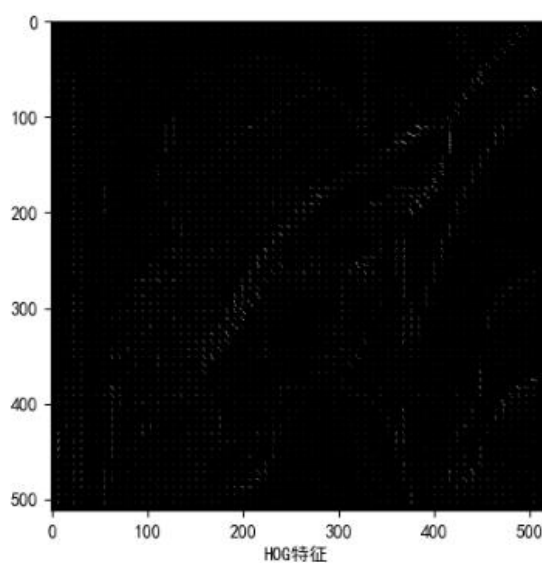


图 17 HOG 特征图

**结果 2：**提取的 LBP 特征如下图 18 所示。

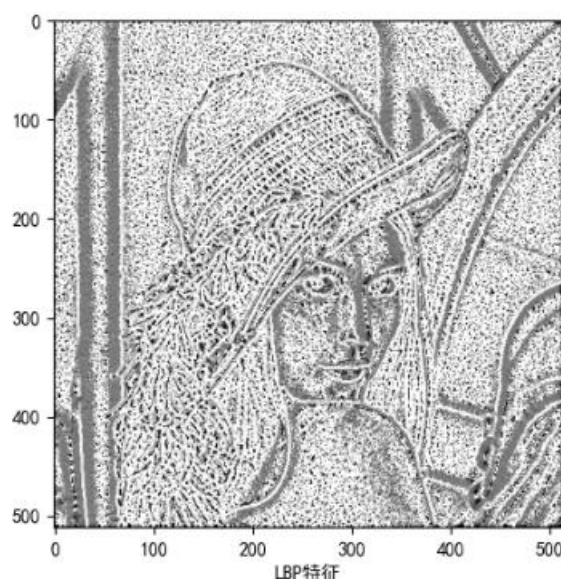


图 18 LBP 特征图

(5)分析与总结（与本次实验相关的分析与总结）

通过本次实验,我学会使用 HOG 算子和 LBP 算子对图像进行特征提取。HOG 的优点是对图像几何和光学形变都保持良好的不变性, 以及对于刚性物体的特征提取具有良好的特性。但是他的缺点也很明显, 主要体现在特征维度大、计算量大、无法处理遮挡的情况。它主要运用在轮廓信息捕获和行人检测上。LBP 的优点是具有旋转不变性和灰度不变性, 此外 LBP 还具有运算速度快的优点。但是它对方向信息敏感, 很微小的偏差就能造成很大的影响。其主要的应用场景为人脸识别和照片分类。