数字图像处理第五次实验图像分割报告（文末附代码）

1 问题

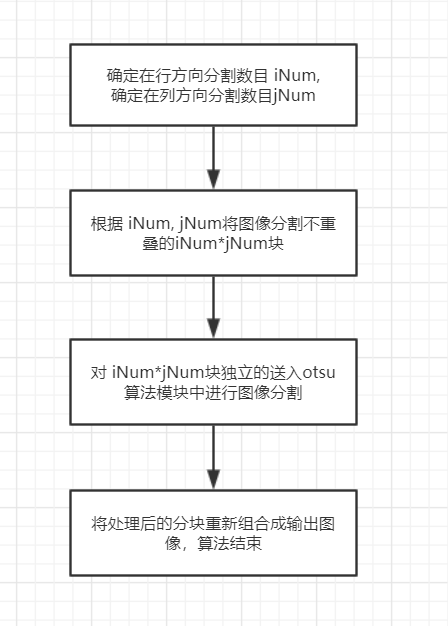
设计图像分割算法完成 SegImg.bmp 图像的分割，分割结果“米粒”像素标 记为 255，“背景”像素标记为 0。

2 问题分析

本次实验进行图像分割的实验，观察整幅图像我发现该图像仅有米粒和背景，而且每个米粒的颜色相近（也就是灰度值相近），因此可以采用单阈值分割的方法来实现图像分割。在所有单阈值分割方法中，我选用已被证明的利用最大化类间方差的otsu算法计算阈值。

同时再观察整幅图像，我发现该图像在垂直方向上受光照的影响明显，图像的灰度图存在被光照污染的可能，为了解决这个问题我采用了分块的方案，通过分块我让受光照影响相近的区域为一个整体做图像分割，不同分块之间不重叠，通过这种方案我很好地解决了光照对图像分割的影响。

综上，本次图像分割实验采用的分割方案如下：



其中 iNum表示在行方向 iNum等分，jNum表示在列方向 jNum等分

在接下来的实验中，我将展示 直接使用 otsu和 分块使用otsu的区别，并对实验结果进行分析。

3 实验结果：

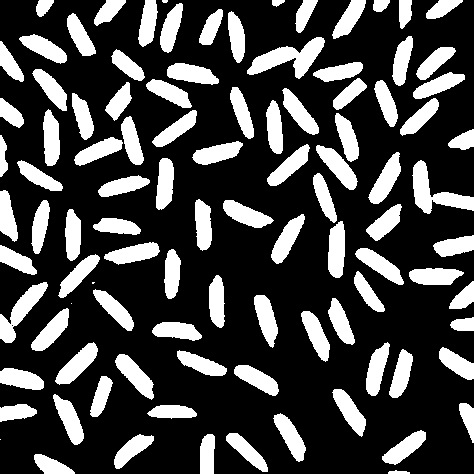
原图像： （命名为 test.bmp）



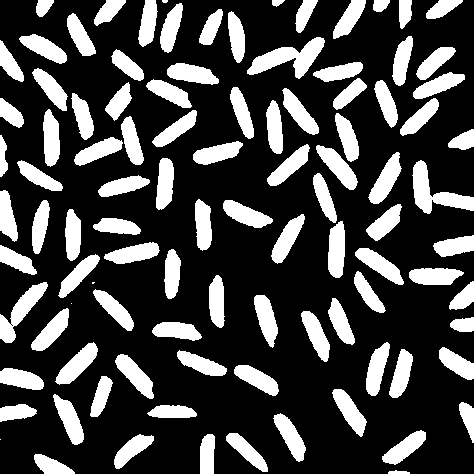
iNum = 1, jNum=1时的结果（不进行分块）



iNum = 8, jNum =1:



iNum =8, jNum=8:



4 实验结果

分析实验结果，我们可以看到如果不分块直接应用 otsu算法，图片底部的分割效果很不好。观察原图，我们可以看到原图的底部受光照影响最大。光照导致底部的直方图和中上部的直方图混合，导致整幅图像的直方图的波谷不是特别明显，所以底部的分割效果不尽如人意，同时导致中上部分出现了噪声（小白点）。

从原图上来看，原图主要在垂直方向光照变化明显，因此我设置iNum=8, jNum =1只在垂直方向进行分块，让光照均匀的区域单独进行otsu分割。从实验结果上看极大地改善了分割效果。对比原图可以说很好地完成了分割任务。这是因为当我们分块之后，不同光照强度区域不会相互影响，从而提升了分割效果

而当我们对列也进行分块的时候（设置jNum=8）我们看到分割效果和上一步相近。这是因为原图在列方向（横向）的光照基本均匀，所以得到的效果几乎相同。

除上述，我们还看到分块之后的图像分割结果仍然存在噪声，这是因为原图本来就存在噪声的原因，这可以通过先对图像进行滤波降噪处理，再进行图像分割来解决。

（所有的实验结果在 result下，根据 参数iNum,jNum的顺序对图片进行了命名）

5代码

**import** os  
**import** numpy **as** np  
**import** cv2 **as** cv  
  
**def** otsu(input):  
 *'''* **:param** *input: 输入图片是一个numpy数组，深度为1* **:return***: output: 分割之后的结果  
 '''* hist,edges = np.histogram(input,256,[0,256],density=**True** ) *#hist是概率密度分布,计算概率分布，这一步自己实现也很简单只需要遍历input统计即可* cdf = hist.cumsum() *# cdf将概率密度求和, cdf[k] 就是 i 从 0 到 k (包括k ) pi的求和* mean\_global = np.sum(np.arange(256)\*hist ) *#计算全局均值* var\_k\_array = np.zeros(256)  
 **for** k **in** range(256):  
 mean\_k = np.sum(np.arange(k+1)\*hist[0:k+1] ) *#计算 m\_k ;m\_k 为 i 从 0 到 k (包括k) i\*pi的求和* **if** cdf[k] ==1 **or** cdf[k] ==0:  
 **continue** temp\_var = np.power((mean\_global\*cdf[k]-mean\_k),2)/(cdf[k]\*(1-cdf[k]))  
 var\_k\_array[k] = temp\_var  
 max\_var\_betweet\_class = np.max(var\_k\_array) *# 求出最大的类间方差* threshold = np.average(np.where(var\_k\_array==max\_var\_betweet\_class)) *# 将所有等于类间最大方差的k值 求平均 计算出分割阈值* mask = (input > threshold) *#输入的图片 所有灰度值 > 阈值的设置为 true* output = np.zeros(input.shape, dtype = np.uint8) *#初始化0矩阵* output[mask]=255 *# 将 =1的值设置为 255* **return** output  
  
**def** devide\_block\_ostu( input, i\_divided\_num, j\_divided\_num ):  
 *'''  
 将图片分块用 ostu算法处理 返回分割后的图像* **:param** *input: 输入灰度图，numpy数组* **:param** *i\_devided\_num: i 轴等分数* **:param** *j\_divided\_num: j 轴等分数* **:return***:  
 '''* output = np.zeros(input.shape,dtype = np.uint8)  
 block\_height = input.shape[0]// i\_divided\_num  
 block\_width = input.shape[1]//j\_divided\_num  
 *#分块执行 ostu算法* **for** i **in** range(i\_divided\_num):  
 i\_begin = i \* block\_height  
 i\_end = i\_begin + block\_height **if** i < i\_divided\_num - 1 **else** input.shape[0] *# 最后一次取全图 防止遗漏* **for** j **in** range(j\_divided\_num):  
 j\_begin = j \* block\_width  
 j\_end = j\_begin + block\_width **if** j < j\_divided\_num - 1 **else** input.shape[1] *# 最后一次取全图 防止遗漏* output[i\_begin:i\_end,j\_begin:j\_end] = otsu( input[i\_begin:i\_end,j\_begin:j\_end] )  
 **return** output  
  
**if** \_\_name\_\_ ==**'\_\_main\_\_'**:  
 BASE\_PATH = os.path.dirname(os.path.abspath(\_\_file\_\_))  
 RESULT\_PATH = os.path.join(BASE\_PATH, **"result"**)  
 **if not** os.path.exists(RESULT\_PATH):  
 os.mkdir(RESULT\_PATH)  
 **import** argparse  
  
 parser = argparse.ArgumentParser()  
 parser.add\_argument(**'--img'**, help=**"the path of the input img"**)  
 parser.add\_argument(**'--iNum'**, help=**'the num of in'**)  
 parser.add\_argument(**'--jNum'**, help=**'the num of jn'**)  
 *# parser.add\_argument('--kernal\_size', help='the size of the kernal')* args = parser.parse\_args()  
 *################# 图片读取模块* img\_name = os.path.split(args.img)[-1]  
 img\_name, ext = os.path.splitext(img\_name)  
 input = cv.imread(args.img, flags=cv.IMREAD\_GRAYSCALE) *# 读取灰度图片  
 # input = cv.imread(args.img, flags=cv.IMREAD\_COLOR) #读取rgb图片  
 ########################## 处理模块* output = devide\_block\_ostu(input,int(args.iNum),int(args.jNum))  
 *#########图片输出模块* img\_name = img\_name + **"\_segmentation\_"** + str(args.iNum)+ **"\_"** + str(args.jNum)  
 img\_name = img\_name + ext  
 img\_name = os.path.join(RESULT\_PATH, img\_name)  
 cv.imwrite(img\_name, output)