

测试文档

课程	计算机视觉
姓名	詹宗沅
学号	15331386
时间	2018.05.22

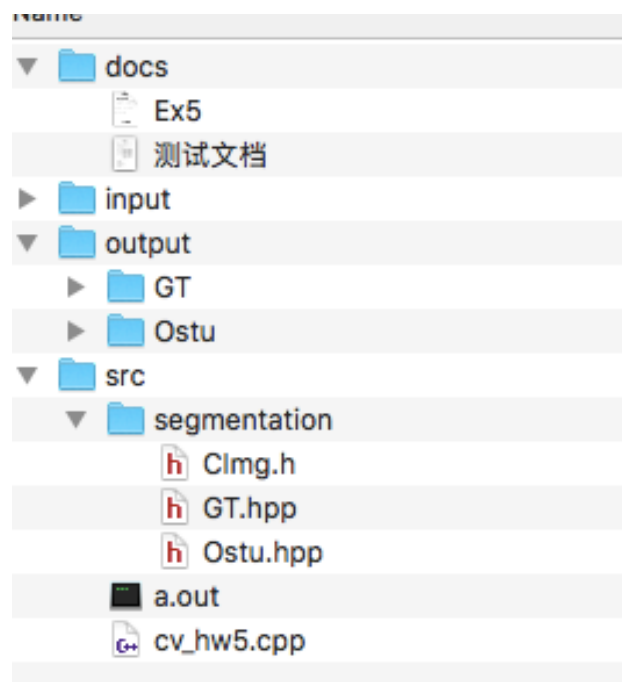
一、理解“迭代法”和“OSTU”

	迭代法	OSTU
算法过程	<ol style="list-style-type: none">1. 随机选择一个阈值T2. 对这个阈值分割的两个像素集合分别计算平均灰度T1, T23. 用平均灰度T1, T2的均值计算下一个中间阈值T'4. 比较T'和T的差, 如果足够小, 说明算法收敛, T'为最终分割阈值。如果不够小, 则将T用T'替换, 进入第二步	<ol style="list-style-type: none">1. 阈值T依次遍历各个灰度级2. 在每个灰度级记录T分割的两个区域的平均灰度和整个图像的平均灰度m_1, m_2, m_G, 和区域中像素占总像素的比例P_1, P_23. 由于为了达到类间方差最大, 可以通过2中数值计算类间方差$\sigma_B^2 = P_1(m_1 - m_G)^2 + P_2(m_2 - m_G)^2$, 最后得到类间方差最大的那个阈值, 即为最终的分割阈值
区别	<ol style="list-style-type: none">1. 迭代法的算法结束标志是达到收敛, 所以T不用遍历所有的阈值情况2. 由于T不用遍历所有阈值情况, 计算复杂度会比较小, 为$O(\log(k)k + n)$, k为灰度等级数, n为像素个数	<ol style="list-style-type: none">1. OSTU算法结束是固定的, 遍历计算所有的灰度作为中间阈值2. 由于T需要遍历所有的阈值情况, 计算复杂度较大, 为$O(k^2 + n)$, 其中k是灰度等级数, n为图像像素点个数

二、项目代码说明

2.1 项目结构

1. docs: 项目相关文档 (测试文档, 实验要求)
2. input: 项目图片数据集, 包含任务一中的100张图片
3. output: 程序运行结果截图, 分成GT和Ostu两个结果
4. src: 项目代码
 - 4.1 segmentation: 实现分割算法的主要代码, 主要实现两个类: GT和Ostu
 - 5.3 cv_hw5.cpp: 项目测试入口文件
6. Makefile: 构建c++项目的makefile文件



2.2 命令行构建运行

1. 项目构建运行 (编译时间较久): 在bash终端(Mac OSX)中进入项目当前文件, 输入:

```
make complie
```

2. 运行可执行文件: 在bash终端(Mac OSX)进入src文件, 输入命令:

```
./a.out [inputPath] [GT outputPath] [Ostu outputPath] [picture number]
```

example: ./a.out ../input ../output/GT ../output/Ostu 100

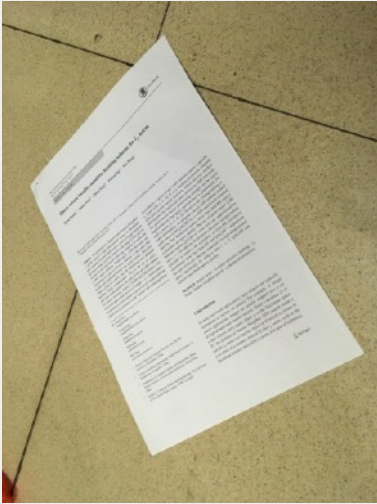
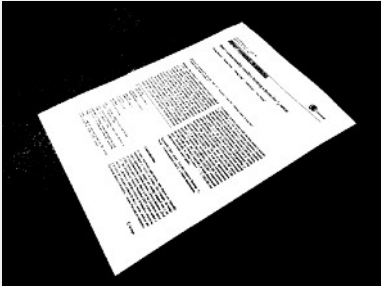
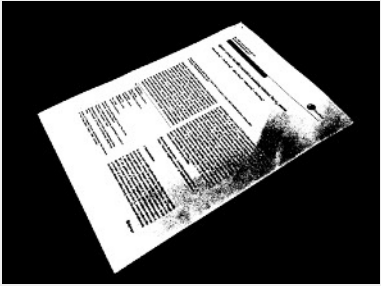
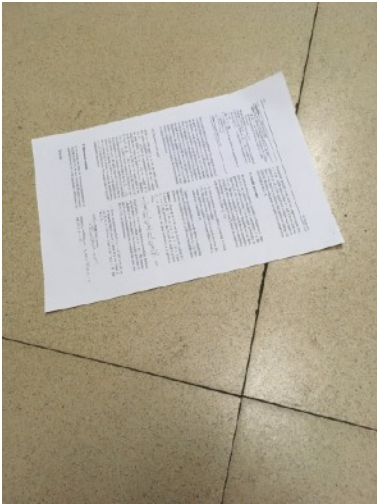
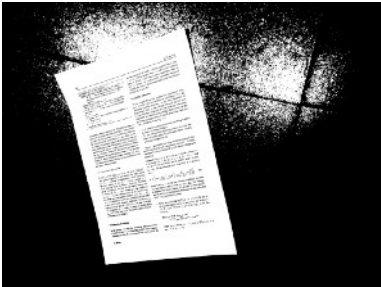

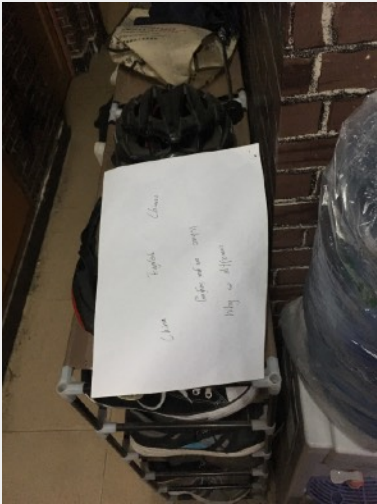
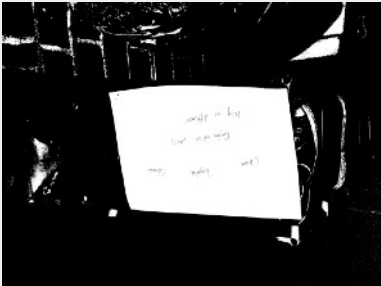
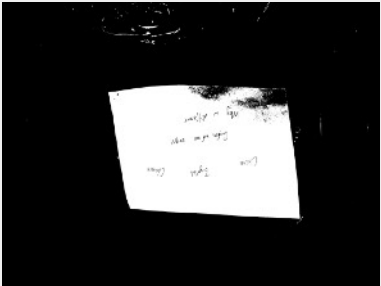
三、测试环境要求

测试系统	macOS High Sierra
编译器	clang++
所需静态链接库	X11、libjpeg

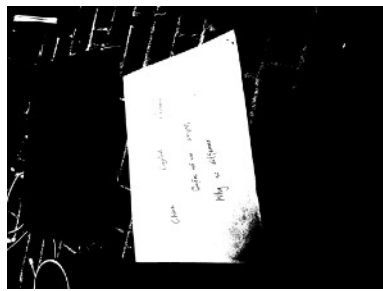
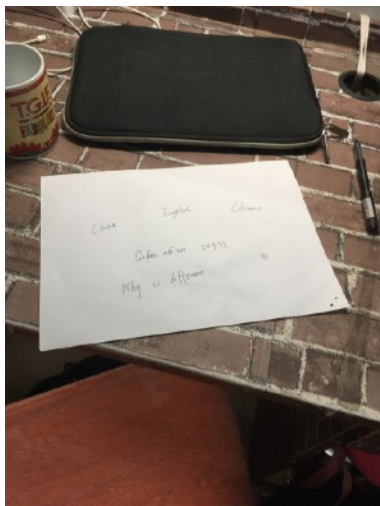
四、测试数据及结果

下面最好最坏仅凭人眼判断A4纸是否完整区分

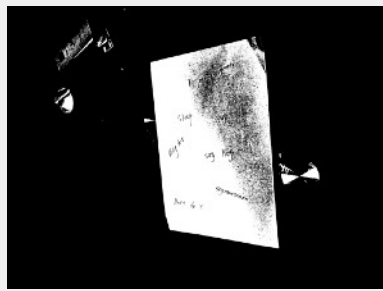
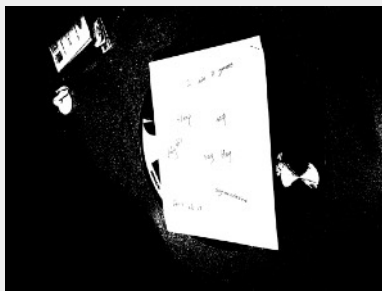
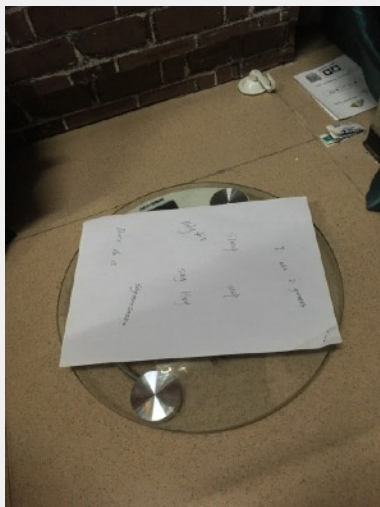
4.1 10个最好结果

编号		Global Thresholding	Ostu
1			
15			
19			

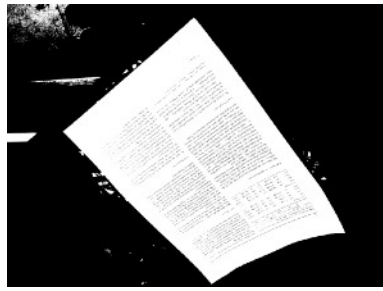
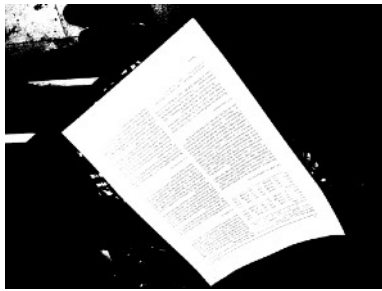
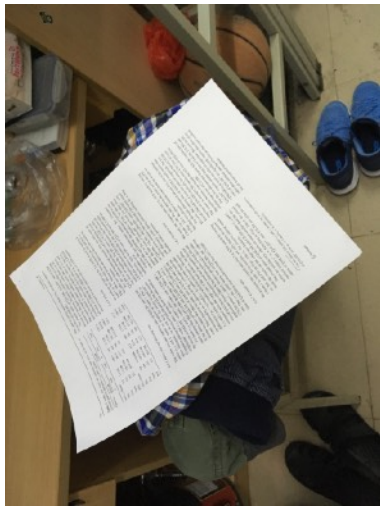
22



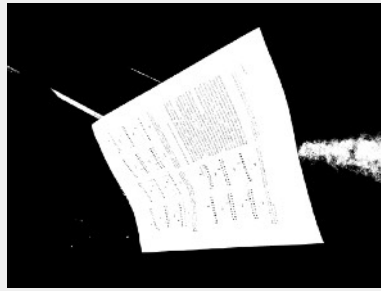
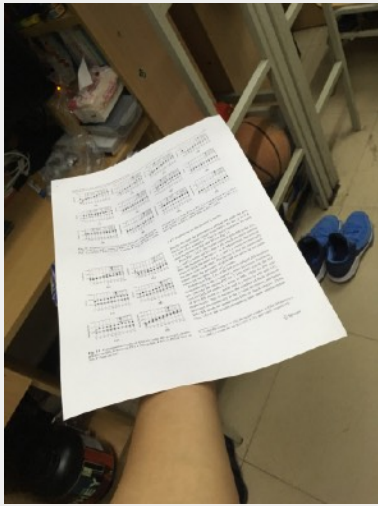
23



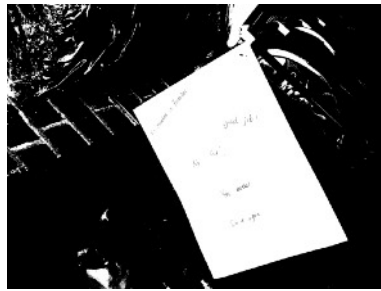
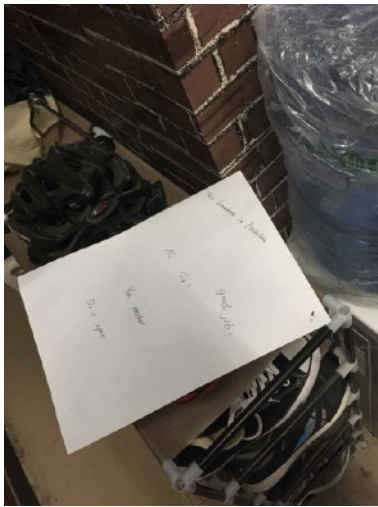
27



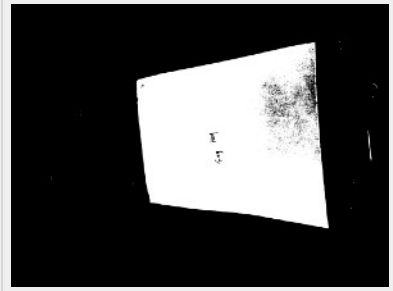
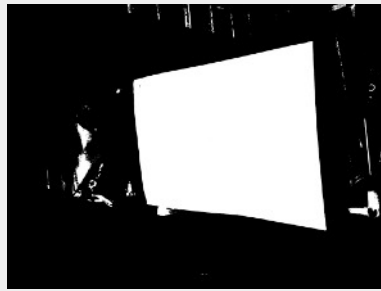
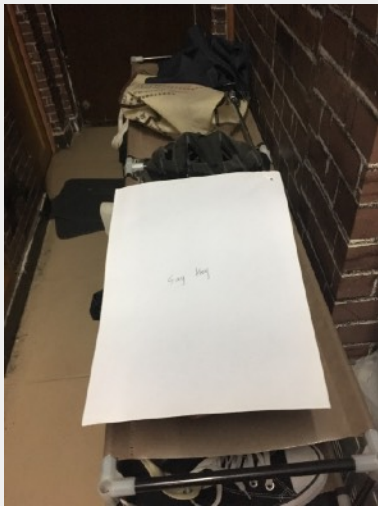
30



31

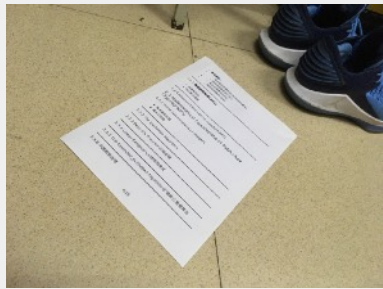
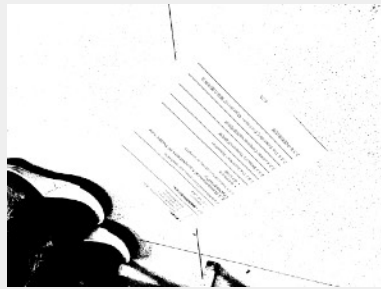
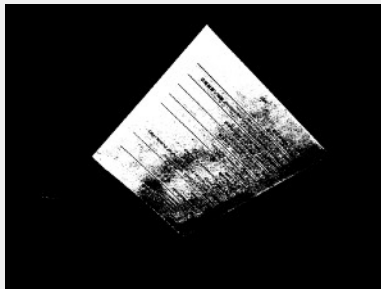
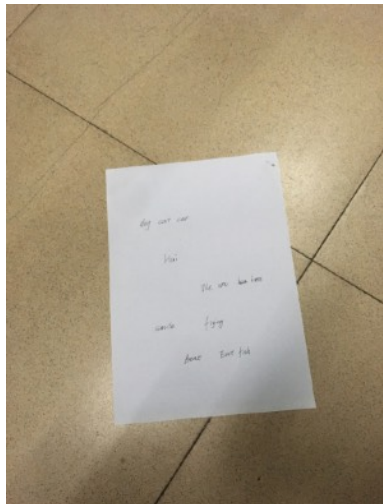
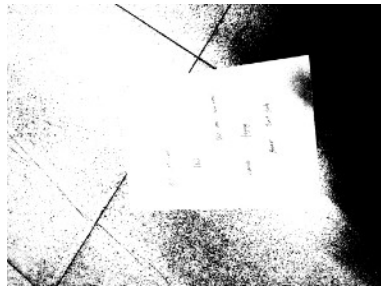
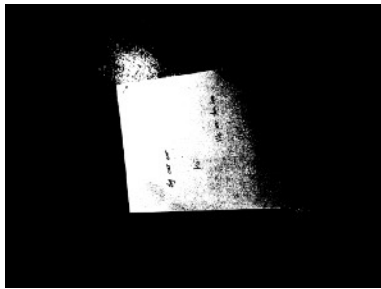


67

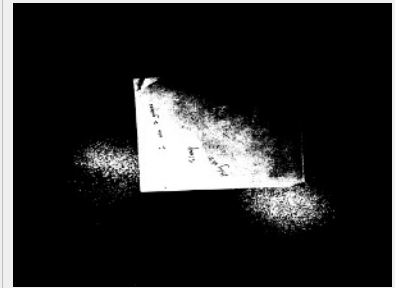
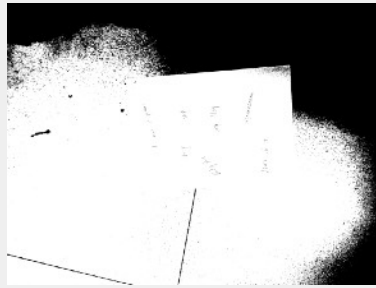
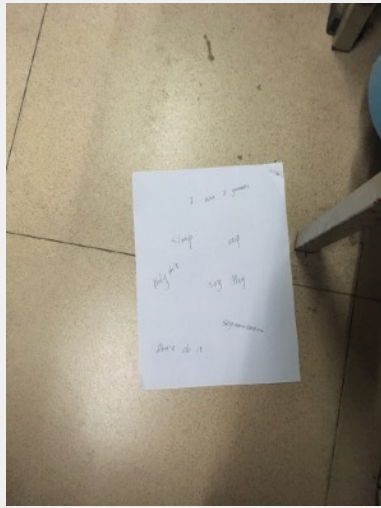


78			
----	---	--	---

4.2 10个最差结果

编号	原图	Global Thresholding	Ostu
43			
45			

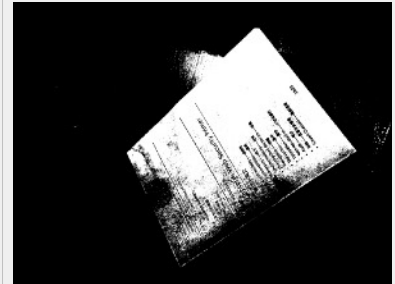
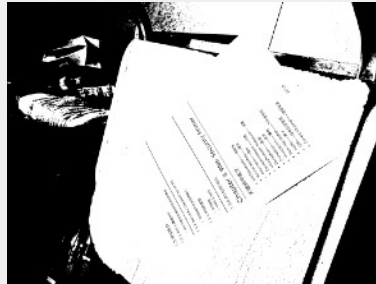
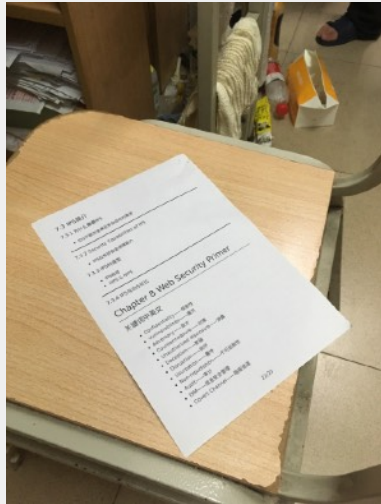
47



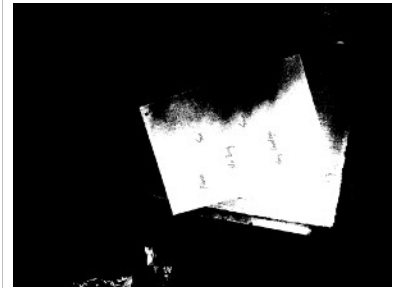
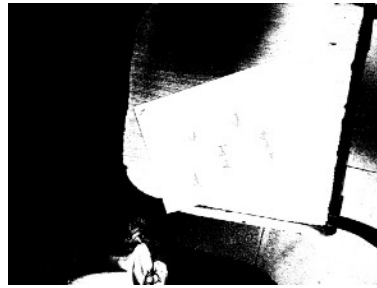
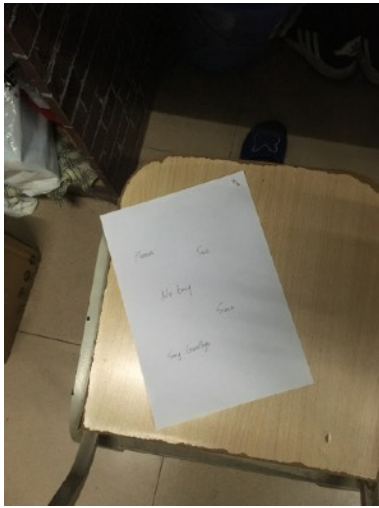
52



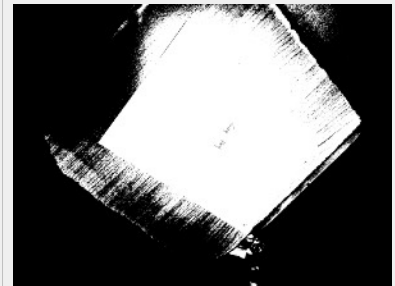
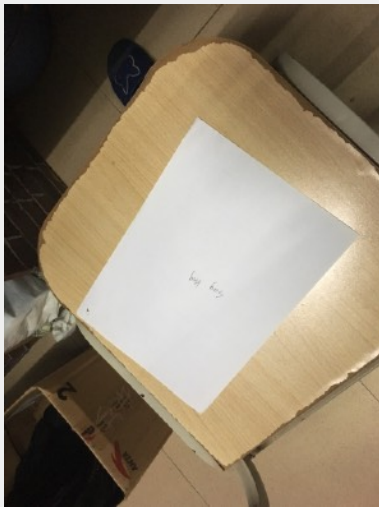
61



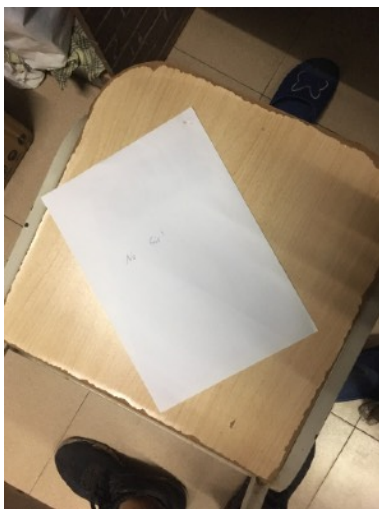
63



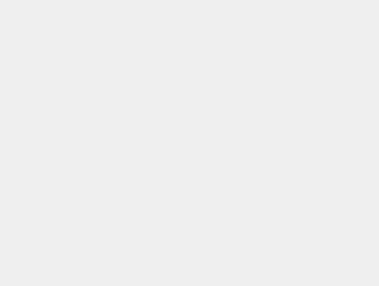
64

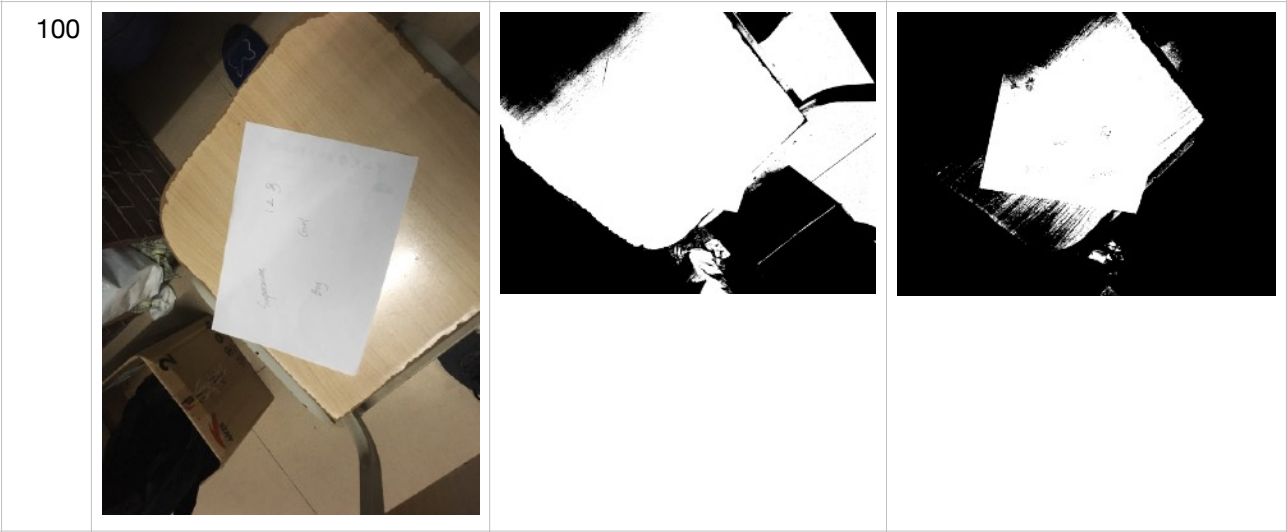


75



90





五、测试分析及感想

5.1 性能分析

1. 时间复杂度分析

GT	Ostu
<pre>Global Threshold algorithm average costs(on 100 jpg image(s)): [read]: 0.249092 s [process]: 0.362527 s [write]: 0.195625 s [total]: 0.807244 s</pre>	<pre>Ostu algorithm average costs(on 100 jpg image(s)): [read]: 0.244438 s [process]: 0.371877 s [write]: 0.191431 s [total]: 0.807746 s</pre>

两个算法的时间复杂度上，差别不大，GT的实际复杂度会比Ostu小一些。
同时两个算法从读入到处理，再到写出所花费时间基本达到每张图片1s以内。

2. 准确性分析


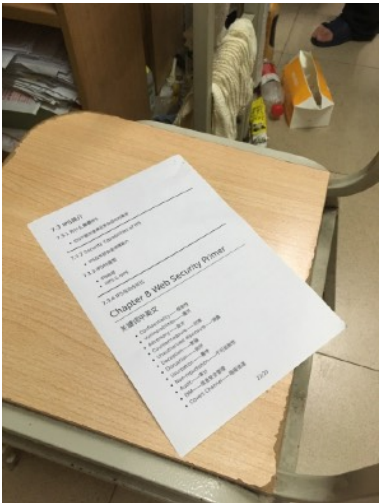
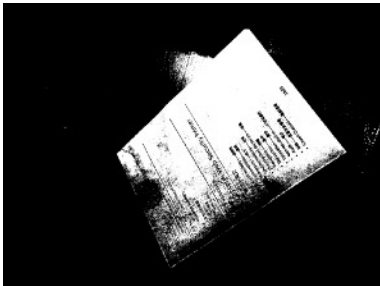
由于没有算法可以量化图片的准确性，所以这里我通过人眼判断，分割的效果，以图片边缘完整分割出来表示正确分割（部分分割图会根据我的理解判断是否正确分割）。

	GT	Ostu
总张数	100	100
正确分割数	45	60
正确率	0.45	0.60

5.2 Global Thresholding效果分析

原图	分割图	分析
		<ol style="list-style-type: none">1. 由于cimg对jpeg的读写的内在机制，导致最后输出的分割图都是横向的。2. 图像中把凳子错分割成了前景的一部分，这是由于图像背景不单一。显然在一个正确以A4纸为目标分割的算法中，这是一个必然要克服的问题
		<ol style="list-style-type: none">1. 将A4张中的一小角分割成背景，这是因为原图中出现了部分阴影，光照不均匀导致的。

5.3 Ostu效果分析

原图	分割图	分析
		1. 可以明显看到A4纸的边缘，但是在原图中出现阴影的部分，被误分割成背景，而由于背景中的凳子出现光源反射的情况，导致亮度较高，被当作A4纸的一部分
		1. 可以明显看到A4纸分割出来的轮廓，背景也基本正确，但是A4纸内部出现很大一块当作背景的部分。

5.4 感想

1. Ostu算法分割效果明显比Global Thresholding的要好，正确率也明显较高。在背景情况复杂的情况下，GT算法容易发生误分割，把背景分割成前景。在出现阴影在纸面的情况下，Ostu算法分割出的纸张会出现残缺。
2. 面临的困难比较多，主要出现在：
a. 背景内容复杂多变的情况下，容易将背景中灰度值高的部分分割成前景。
b. 纸张光照不均衡，导致纸张分割残缺