

DIP-HW1

黃川懿 520030910268

2022 年 11 月 7 日

摘要

通过实现实现至少二种阈值分割，实现 Robert、Prewitt、Sobel 梯度算子、Laplacian 算子、高斯 Laplacian 算子、Canny 算子，实现区域生长算法，实现或测试区域分裂、合并算法，对于阈值分割、边缘检测、区域分割算法中产生的现象进行思考，从而进一步了解阈值分割、边缘检测、区域分割算法在图像处理中的作用。

目录

1 阈值分割	1
1.1 简介	1
1.2 实验结果	1
1.2.1 大津法实验结果	1
1.2.2 自适应阈值法实验结果	3
1.3 总结	4
1.3.1 两种方法的异同	4
1.3.2 大津法对于简单, 复杂, 噪声图像处理的效果	5
1.3.3 自适应算法对于简单, 复杂, 噪声图像处理的效果	5
2 边缘检测	5
2.1 简介	5
2.2 实验结果	6
2.2.1 Robert 算子实验结果 (以梯度阈值 40 进行二值化)	6
2.2.2 Prewitt 算子实验结果以梯度阈值 40 进行二值化	7
2.2.3 Sobel 算子实验结果 (以梯度阈值 40 进行二值化)	8
2.2.4 梯度算子 (Robert,Prewitt,Sobel) 实验结果分析	8
2.2.5 Laplacian 算子实验结果 (以梯度阈值 10 进行二值化)	9
2.2.6 Laplacian 算子实验结果分析	10
2.2.7 LoG 算子实验结果 (默认卷积模版 5x5)	11
2.2.8 LoG 算子实验结果分析	12
2.2.9 Canny 算子实验结果	13
2.2.10 Canny 算子实验结果分析	15
3 区域生长	17
3.1 简介	17
3.2 种子选取	17
3.3 自回归的理解	17
3.4 实验结果	17
3.5 总结	18
4 区域合并	19
4.1 简介	19
4.2 自回归的理解即实现	19
4.3 实验结果	21
4.4 总结	22
5 总结	22

1 阈值分割

1.1 简介

阈值分割可以看作一种门限处理，而门限处理可以表示成涉及如下形式函数 T 的操作 $T = T[x, y, p(x, y), f(x, y)]$, f, p 分别表示灰度级，这个点的局部性质，所以根据 T 取决于 f 还是 p 或则 (x, y) 阈值分割分为全局，局部，自适应三种形式。常见的阈值分割方法有直方图双峰法，大津法，自适应阈值等，这里采取全局方法大津法和自适应算法自适应阈值两种算法对于给定的简单复杂，和有噪声的图像进行处理

1.2 实验结果

1.2.1 大津法实验结果

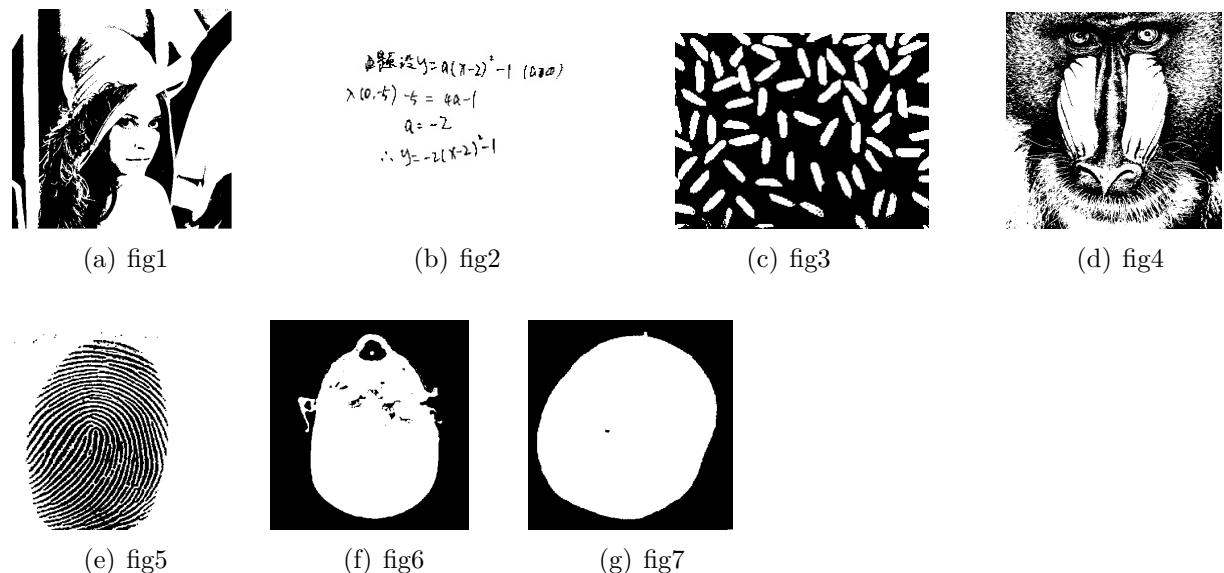


图 1: 大津法处理的简单图像



图 2: 大津法处理的噪声图像



(a) fig1



(b) fig2



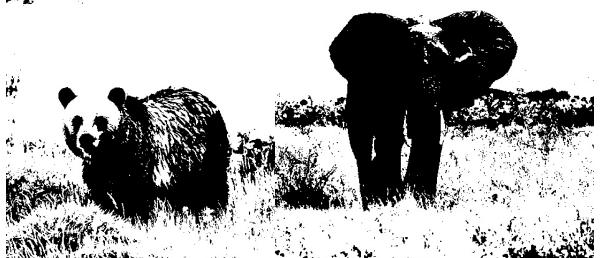
(c) fig3



(d) fig4



(e) fig5



(f) fig6

(g) fig7



(h) fig8



(i) fig9



(j) fig10

图 3: 大津法处理的复杂图像

1.2.2 自适应阈值法实验结果

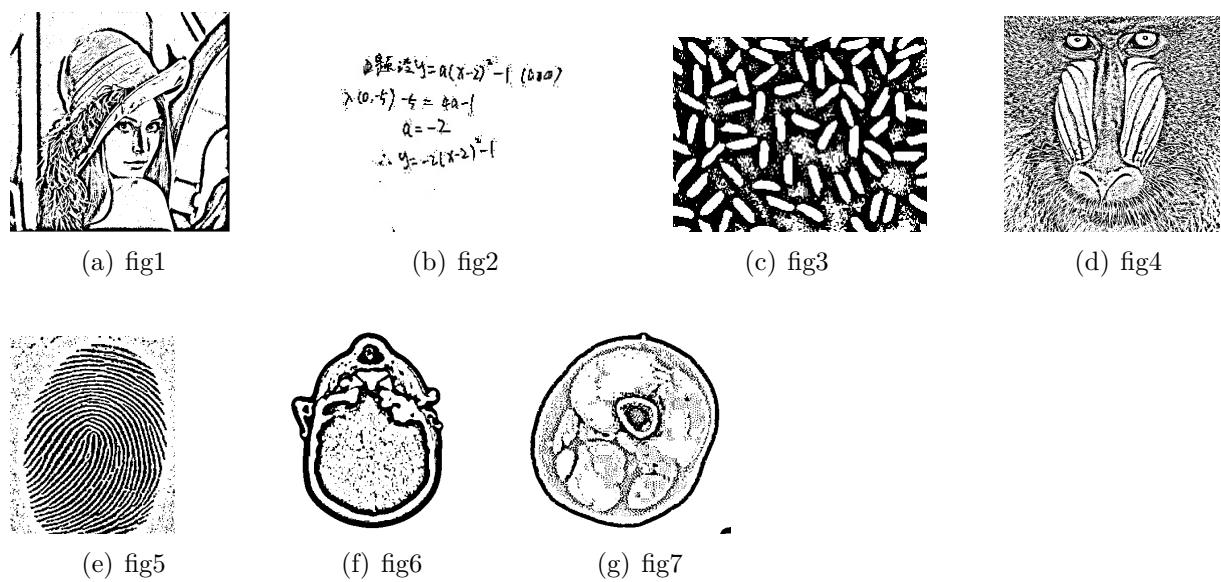


图 4: 自适应阈值法处理的简单图像



图 5: 大津法处理的噪声图像

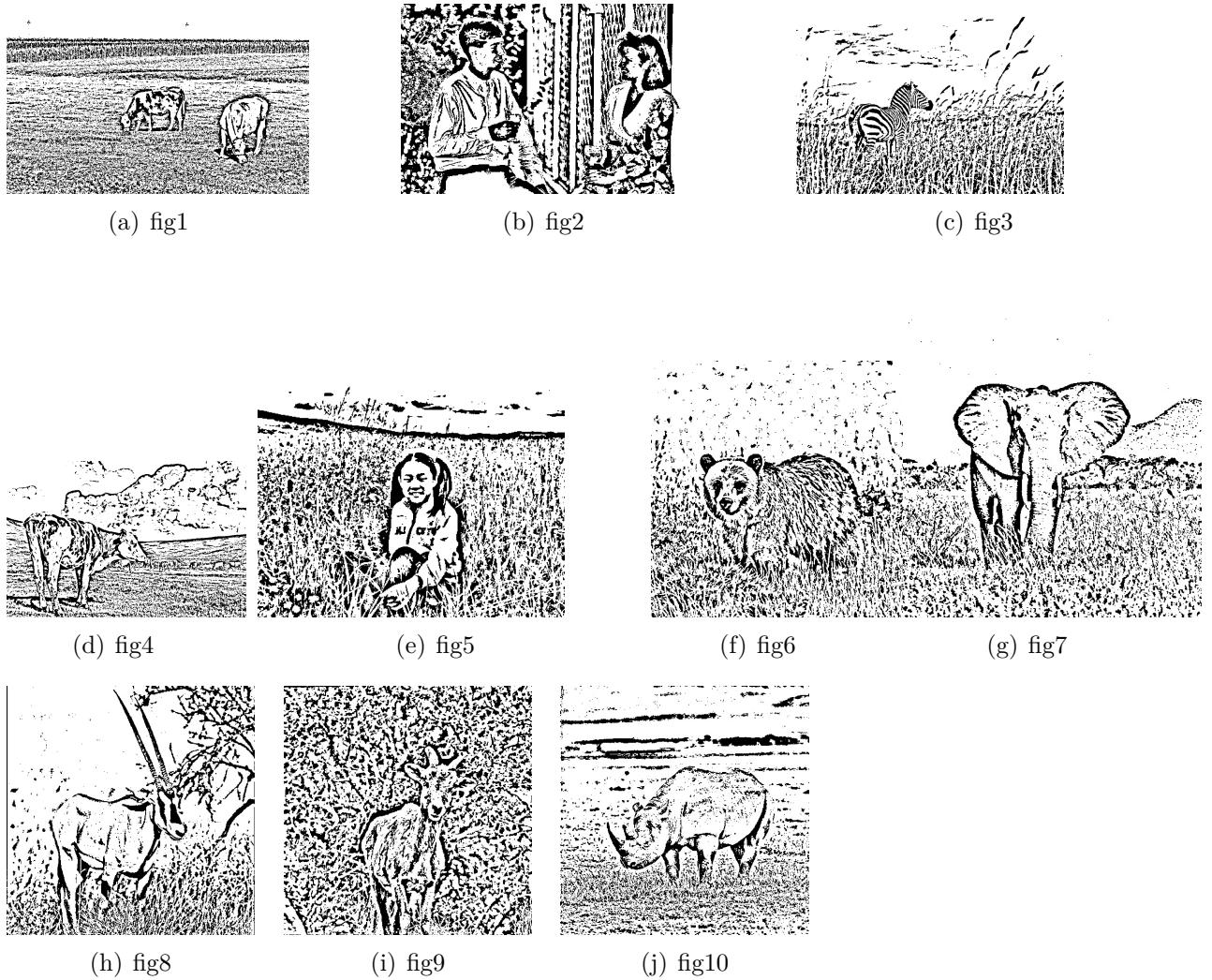


图 6: 自适应阈值法处理的复杂图像

1.3 总结

1.3.1 两种方法的异同

相同的地方:1. 两者在面对简单图像 (可以理解为图像中有明显的前景和背景, 也可以说物体与背景有较强对比的图像) 都能有效的将图像进行分割。(比如图 1 fig2, fig5 和图 4 fig2,fig5)

2. 同时在面对噪声图像时, 均不能有效消除噪声带来的影响
3. 两者的计算量, 性能相比其他的分割方法都具有优势。

不同的地方: 1. 抗干扰性不同: 大津法只是借助了图像的像素灰度信息, 并未考虑像素间的空间相关信息 (如邻域信息), 所以当外部干扰使得灰度直方图的波峰和波谷并不一定明显时, 大津法的分离效果不是很理想, 这点可以在复杂图像中体现, 而自适应阈值法

由于是分区域分离图像，考虑到了像素空间关系，当区间选取适当时，有一定的抗干扰效果

2. 分离细节不同：大津法本质是全局的二值分割，对于较复杂的图像往往不能较好的分离出一些细节，只能大体分离出感兴趣物体，其轮廓完整性欠佳。而自适应算法是基于局部二值分割，当局部大小取得较小时，可以依次获取感兴趣物体的区间部分，所以能较好的提取出感兴趣物体，但是同样也会引入一些不必要的噪声。

1.3.2 大津法对于简单，复杂，噪声图像处理的效果

对简单图像：具有较好的分离效果，可以较为清晰的分离前景和背景，前景的边缘完整性较好

对复杂图像：分离效果不是很理想，主要存在两个问题，一是前景的边缘完整性不佳（如图三 fig1,fig3,fig4,fig8,fig9），二是会存在无法有效分离出感兴趣的物体（如 fig2, fig5, fig9），甚至会出现完全无法分离的现象（fig10）

1.3.3 自适应算法对于简单，复杂，噪声图像处理的效果

对简单图像：具有较好的分离效果，可以较为清晰的分离前景和背景，前景的边缘完整性较好

对复杂图像：分离效果有好有坏，好的方面是分离出的物品都有相对完整的边界，不好的地方是，由于引入了过多的细节，当感兴趣物品的纹理和背景（树叶，草等）的纹理有一定相似时，较难分辨物品和背景（如图 6 fig2,fig3,fig4,fig6,fig9）

2 边缘检测

2.1 简介

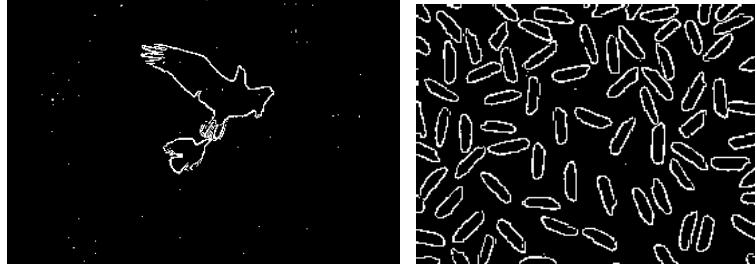
边缘检测本质上就是一种滤波算法，区别在于滤波器的选择，滤波的规则是完全一致的。

而边缘检测算法其目标是找到一个最优的边缘，而最优边缘的定义是：1. 算法能够尽可能多地标示出图像中的实际边缘 2. 标识出的边缘要与实际图像中的实际边缘尽可能接近 3. 最小响应 - 图像中的边缘只能标识一次，并且可能存在的图像噪声不应该标识为边缘

基于不同的准则，有以下常见的滤波器，Robert 算子（斜向偏差分的梯度），Prewitt, Sobel 算子（一阶梯度），Laplacian 算子（二阶梯度），LoG，Canny 算子（基于边缘检测算法），下面给出不同算子对于简单，复杂，噪声图像处理的结果

2.2 实验结果

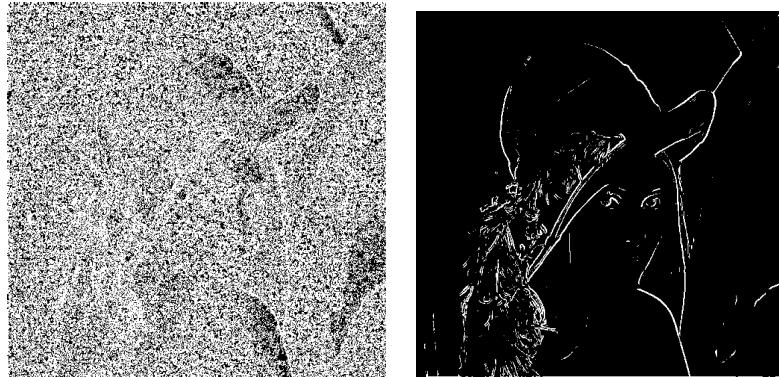
2.2.1 Robert 算子实验结果 (以梯度阈值 40 进行二值化)



(a) fig1

(b) fig2

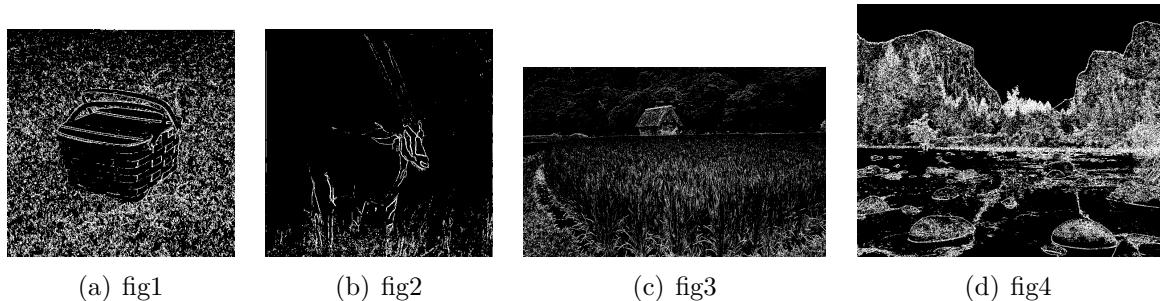
图 7: Robert 算子处理的简单图像



(a) fig1

(b) fig2

图 8: Robert 算子处理的噪声图像



(a) fig1

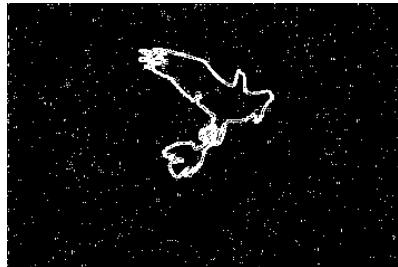
(b) fig2

(c) fig3

(d) fig4

图 9: Robert 算子处理的复杂图像

2.2.2 Prewitt 算子实验结果以梯度阈值 40 进行二值化

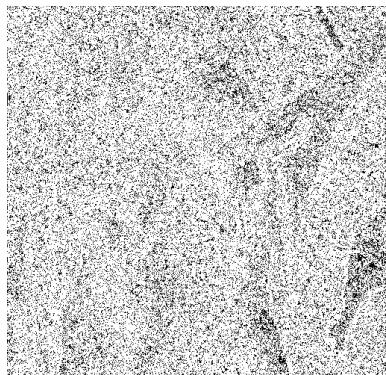


(a) fig1



(b) fig2

图 10: Prewitt 算子处理的简单图像

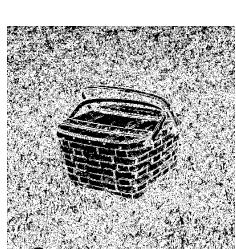


(a) fig1



(b) fig2

图 11: Prewitt 算子处理的噪声图像



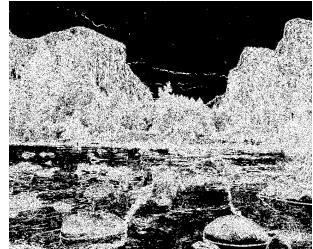
(a) fig1



(b) fig2



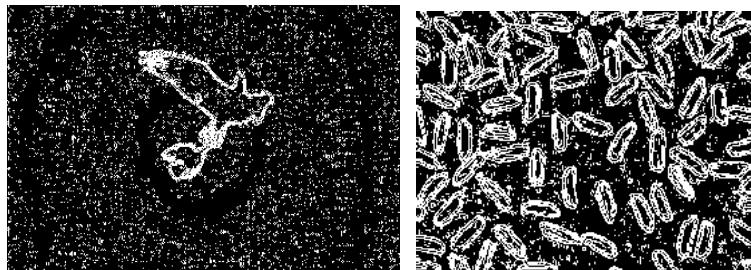
(c) fig3



(d) fig4

图 12: Prewitt 算子处理的复杂图像

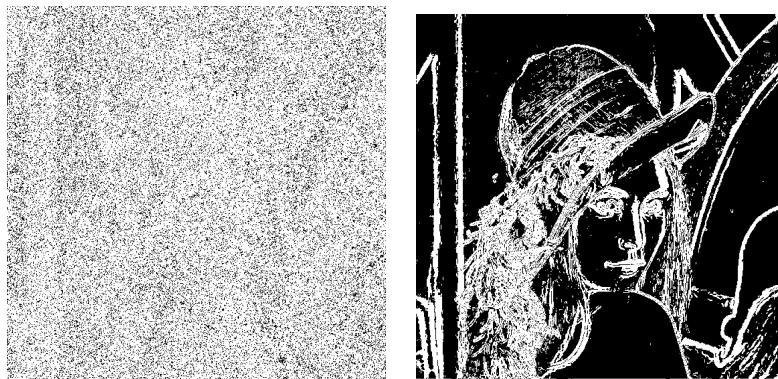
2.2.3 Sobel 算子实验结果 (以梯度阈值 40 进行二值化)



(a) fig1

(b) fig2

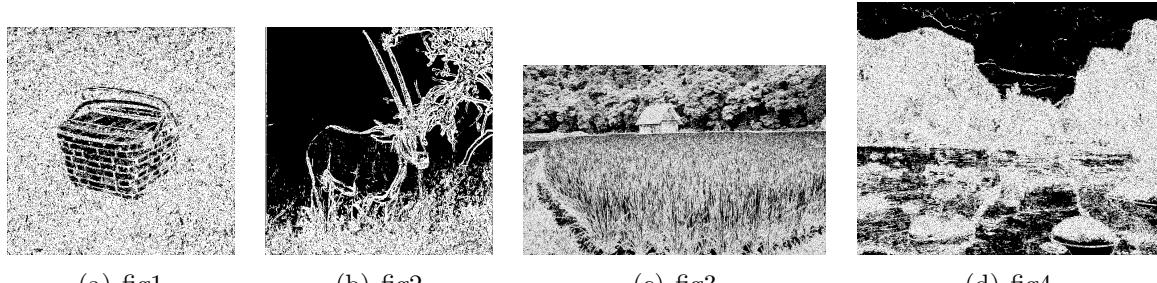
图 13: Sobel 算子处理的简单图像



(a) fig1

(b) fig2

图 14: Sobel 算子处理的噪声图像



(a) fig1

(b) fig2

(c) fig3

(d) fig4

图 15: Sobel 算子处理的复杂图像

2.2.4 梯度算子 (Robert,Prewitt,Sobel) 实验结果分析

一. 现象分析:

1. 厚边界现象: 可以看到不管 Robert, Prewitt 还是 Sobel 算子对边缘定位都不是很准确, 图像的边缘不止一个像素。其中 Prewitt 和 Sobel 算子边界比较“厚”, Robert 算

子边界相对理想一点。这是因为梯度算子对于边界的响应比较宽，而设置梯度的时候会将一些实际不是边界的点视作边界，导致图像的边缘不止一个像素。

2. 对噪声的敏感性分析：梯度算子对于噪声都比较敏感，分别对比图 8 fig1, fig2；图 11 fig1, fig2；图 14 fig1, fig2。都可以看出，相比没有噪声的图像，梯度算子处理有噪声的图像后已经无法识别感兴趣对象的边界。整体来说，Prewitt 和 Sobel 对与噪声的抑制相对于 Robert 算子强一点

3. 不同阈值下检测结果（由于图像较多，没有将其他阈值处理的图像体现在报告中，但是在附件中）：可以发现，随着阈值的增加，处理后的图像中厚边界的现象有一定的缓解，但是相应的会损失一定的细节，甚至导致边界的断裂

4. 边界连接性以及封闭性：可以发现梯度算子对于简单图像处理后边界连接性以及封闭性较好，但是对于复杂图像（如图 9，图 12，图 15 fig2）以及有噪声干扰的图像（图 8，图 11，图 14 fig 1）处理后边界有一定程度上的断裂，封闭性不是很好

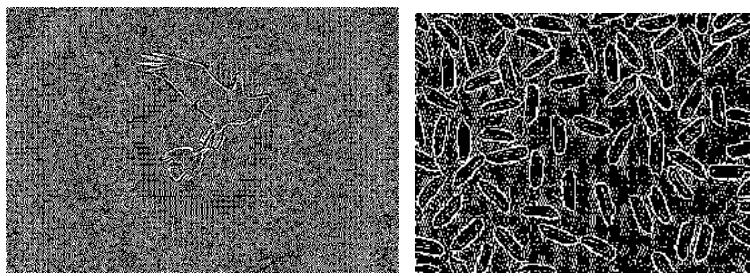
二. 同一幅图像在不同梯度检测算子下的检测结果的差异：对比 Robert,Prewitt 和 Sobel 的实验结果，发现在选择相同的阈值情况下：

1.Robert 对于边界的响应宽度相当于 Prewitt, Sobel 算子较小，但是响应的，对于边界的响应信息也相对较少。Sobel 对于边界的响应最宽。

2.Robert 的复杂图像边界信息丢失相对于 Sobel,Prewitt 算子更多，这是因为 Robert 采用对角线差分，边缘定位相对准一些，但是对噪声敏感，所以边界更容易收到噪声的影响

3. 噪声图像 Sobel 处理相对最好，Prewitt 处理次之，Robert 最差。这是因为算子中引入了类似局部平均的运算，因此对噪声具有平滑作用，能很好的消除噪声的影响。Sobel 算子对于象素的位置的影响做了加权，与 Prewitt 算子、Roberts 算子相比因此效果更好

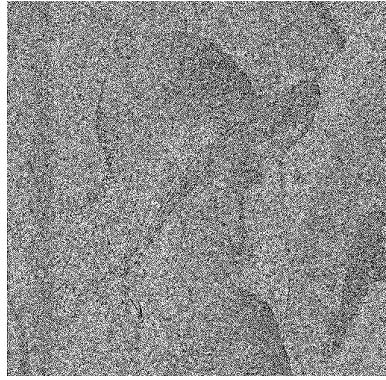
2.2.5 Laplacian 算子实验结果（以梯度阈值 10 进行二值化）



(a) fig1

(b) fig2

图 16: Laplacian 算子处理的简单图像



(a) fig1



(b) fig2

图 17: Laplacian 算子处理的噪声图像



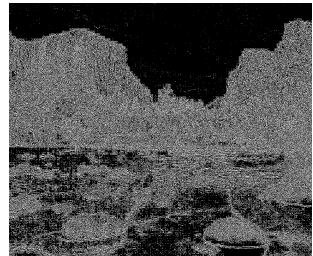
(a) fig1



(b) fig2



(c) fig3



(d) fig4

图 18: Laplacian 算子处理的复杂图像

2.2.6 Laplacian 算子实验结果分析

1. 观察实验图像，发现 Laplacian 算子对任何灰度变化都存在响应，包括细节与非边界处，这是因为任何灰度变化都会导致零交叉，有零交叉就会有正部分和负部分，就存在双边界。
2. 观察实验图像，发现有很多噪点，但是灰度变化的地方和边界也很有明显的线条。尤其是带椒盐噪声的图（图 17 fig1）噪声点已经完全将图像覆盖，说明 Laplacian 算子对于噪声，有灰度变化的地方，以及边界都十分敏感

2.2.7 LoG 算子实验结果 (默认卷积模版 5x5)



(a) fig1



(b) fig2

图 19: LoG 算子处理的简单图像

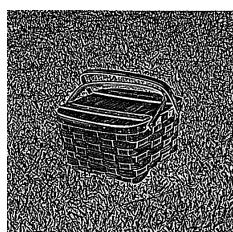


(a) fig1



(b) fig2

图 20: LoG 算子处理的噪声图像



(a) fig1



(b) fig2

图 21: LoG 算子处理的复杂图像

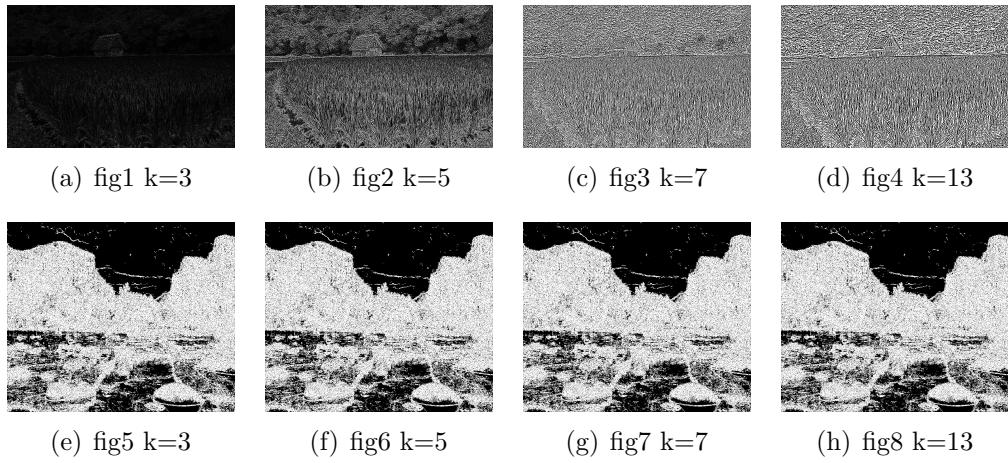


图 22: 不同大小高斯卷积模板 LoG 算子处理的图像

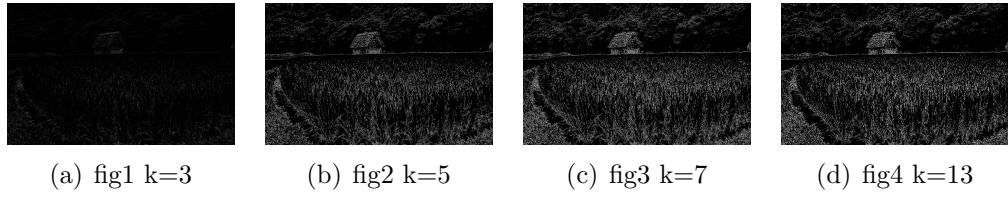


图 23: 不同大小高斯卷积模板 LoG 算子经过高梯度阈值约束后处理的图像

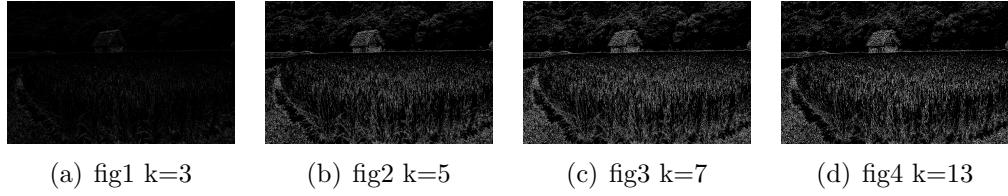


图 24: 不同大小高斯卷积模板 LoG 算子经过高梯度阈值约束后并化双像素宽为单像素宽处理后的图像

2.2.8 LoG 算子实验结果分析

- 观察图 22 知不同大小的高斯卷积模板对图像平滑后，降低了噪声，孤立的点噪声和较小的组织结构被滤除。降低了噪声对于 Laplacian 的影响
- 虽然高斯卷积模板对图像平滑后降低了噪声对于 Laplacian 的影响，但是图像的平滑也导致了边界的延伸（即产生伪边界），可哟对比图 22 中 fig1-fg4，可以发现尺度越大的边界位置的偏移量越大，尺度越大的小灰度变化或细节的响应越宽，宽度也越宽，越不利于边界的定位。
- 由图 23 知，加入高梯度阈值这一额外的约束后能有效消除伪边界。其具体实现如下：

¹ # LoG 高梯度阈值

```

2 def High_Grad_Threshold_LoG( filename ,k ,T=40):
3     img=cv2 . imread( filename ,0)
4     img_grad_x= cv2 . Sobel( img ,cv2 . CV_64F,1 ,0)
5     img_grad_y= cv2 . Sobel( img ,cv2 . CV_64F,0 ,1)
6     img_grad=sqrt( img_grad_x**2+img_grad_y**2)
7     mask=np . where( img_grad>T,1 ,0)
8     img_gaussian = cv2 . GaussianBlur( img , (k ,k) , 1)
9     LoG = cv2 . Laplacian( img_gaussian , cv2 . CV_64F,  ksize=k)
10    LoG=LoG*mask
11    return  LoG
12 # LoG 单像素宽
13 def single_edge_LoG( filename ,k ,T=40):
14     LoG=High_Grad_Threshold_LoG( filename ,k ,T)
15     return  np . where( LoG>0,LoG ,0)

```

4. 上文代码中也引出了得到单像素宽的边界的思路，即双像素宽边界包含了正部分，以及截断后的负部分，那么可以消除负部分从而得到单像素宽的边界，如图 24 所示

2.2.9 Canny 算子实验结果

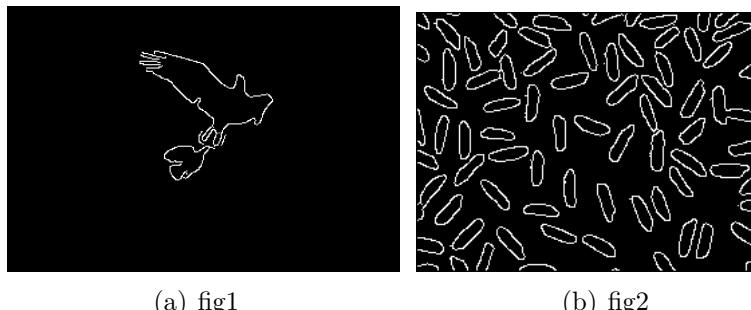
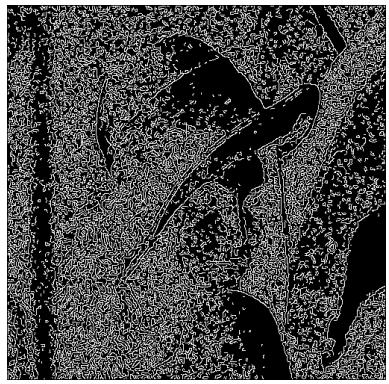


图 25: Canny 算子处理的简单图像

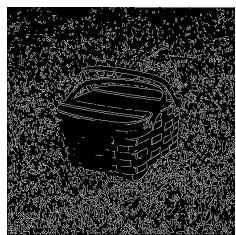


(a) fig1



(b) fig2

图 26: Canny 算子处理的噪声图像



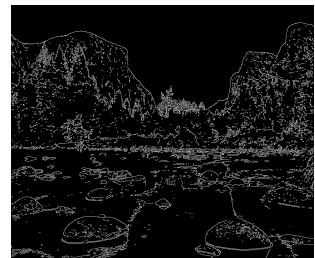
(a) fig1



(b) fig2

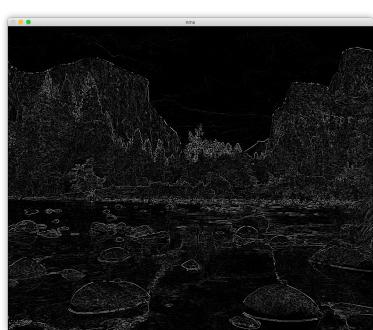


(c) fig3

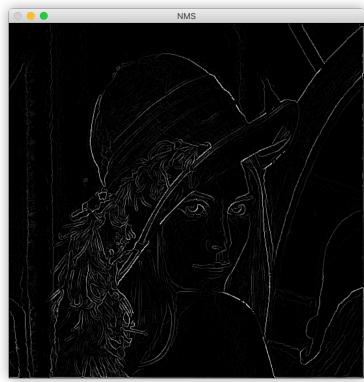


(d) fig4

图 27: Canny 算子处理的复杂图像



(a) fig1



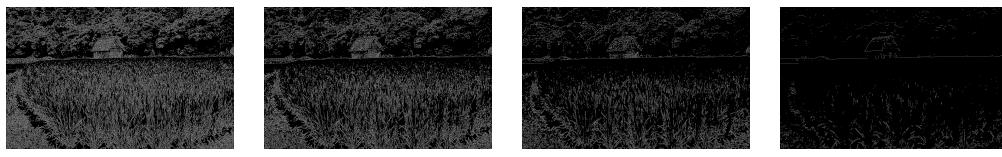
(b) fig2

图 28: NMS 细化边界的图像



(a) fig1 High=120 (b) fig2 Low=80 only
only (Strong Edge) (Weak Edge) (c) fig3 (d)
low=80,high=120 low=10,high=40 fig4

图 29: 滞后双阈值的效果及双阈值大、小不同阈值下的检测图像



(a) fig1 k=3 (b) fig2 k=5 (c) fig3 k=7 (d) fig4 k=13

图 30: 不同大小高斯卷积模板 Canny 算子结果

2.2.10 Canny 算子实验结果分析

1. 由图 28 知, 非极大抑制的实现及细化边界接近理想边界, NMS 的实现如下:

```

1
2     def NMS():
3         for i in range(1, len(gra)-1):
4             for j in range(1, len(gra[0])-1):
5                 if gra[i, j]==0:
6                     continue
7                 else:
8                     dTmp1 = 0
9                     dTmp2 = 0
10                    if theta[i, j] >= 0 and theta[i, j]
11                        < 45:
12                            g1, g2, g3, g4 = gra[i+1, j-1],
13                                gra[i+1, j], gra[i-1, j+1],
14                                gra[i-1, j]
15                            W = abs(np.tan(theta[i, j]*np.
16                                pi/180))
17                            dTmp1 = W * g1 + (1-W) * g2
18                            dTmp2 = W * g3 + (1-W) * g4

```

```

15     elif theta[i,j] >= 45 and theta[i,
16         j] < 90:
17             g1, g2, g3, g4 = gra[i+1,j-1],
18                 gra[i, j-1], gra[i-1, j
19                     +1], gra[i, j+1]
20             W = abs(np.tan((theta[i,j]-90)
21                             *np.pi/180))
22             dTmp1 = W * g1 + (1-W) * g2
23             dTmp2 = W * g3 + (1-W) * g4
24             elif theta[i,j] >= -90 and theta[i,
25                 j] < -45:
26                 g1, g2, g3, g4 = gra[i-1, j
27                     -1], gra[i, j-1], gra[i+1,
28                         j+1], gra[i, j+1]
29                 W = abs(np.tan((theta[i,j]-90)
30                               *np.pi/180))
31                 dTmp1 = W * g1 + (1-W) * g2
32                 dTmp2 = W * g3 + (1-W) * g4
33                 elif theta[i,j]>=-45 and theta[i,j
34                     ]<0:
35                     g1, g2, g3, g4 = gra[i+1, j
36                         +1], gra[i+1, j], gra[i-1,
37                             j-1], gra[i-1, j]
38                     W = abs(np.tan(theta[i,j] * np
39                         .pi / 180))
40                     dTmp1 = W * g1 + (1-W) * g2
41                     dTmp2 = W * g3 + (1-W) * g4
42                     if dTmp1 < gra[i,j] and dTmp2 <
43                         gra[i,j]:
44                         results[i,j]=gra[i,j]
45
46     return results

```

2. 滞后双阈值的效果考察单一的大、小不同阈值下的检测效果：由图 29fig1-3 知 Canny 最后结果为强边界加其连接的弱边界

3. 由图 29 fig3-4 知阈值选择越高，能连接的弱边界越少，边界信息越少，连续性受一定的影响

4. 由图 30 知卷积的尺度越大，所包含的边界信息越小，这是图像平滑导致梯度稀释所导致的

3 区域生长

3.1 简介

区域生长算法的基本思想是将有相似性质的像素点合并到一起。对每一个区域要先指定一个种子点作为生长的起点，然后将种子点周围领域的像素点和种子点进行对比，将具有相似性质的点合并起来继续向外生长，直到没有满足条件的像素被包括进来为止。

3.2 种子选取

这里有一个核心问题，如何选取种子，我的做法是通过设定判断峰值的条件(peak_distance)选出图像直方图中的峰值对应灰度值的所有像素作为种子，加入递归的队列中。

3.3 自回归的理解

区域生长的核心是递归，当种子向其 8 领域发散时，如果其领域中的一点满足阈值条件，则这一点作为新的种子向外发散，直到没有点满足条件为止，其形成的联通域可视为一个区域。实际实现是循环加维护一个队列实现。

3.4 实验结果

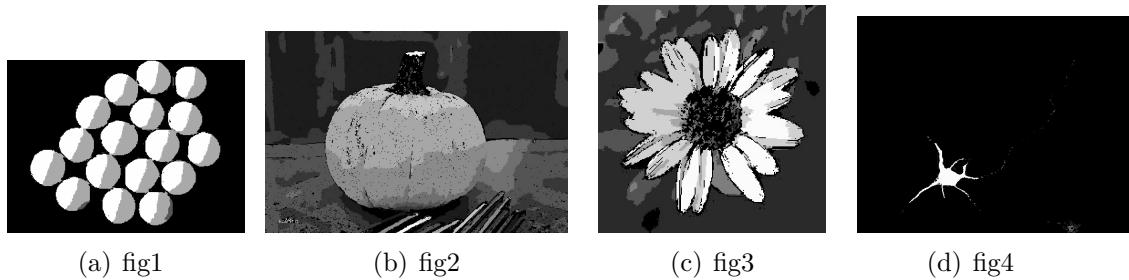


图 31: 区域生长处理的简单图像

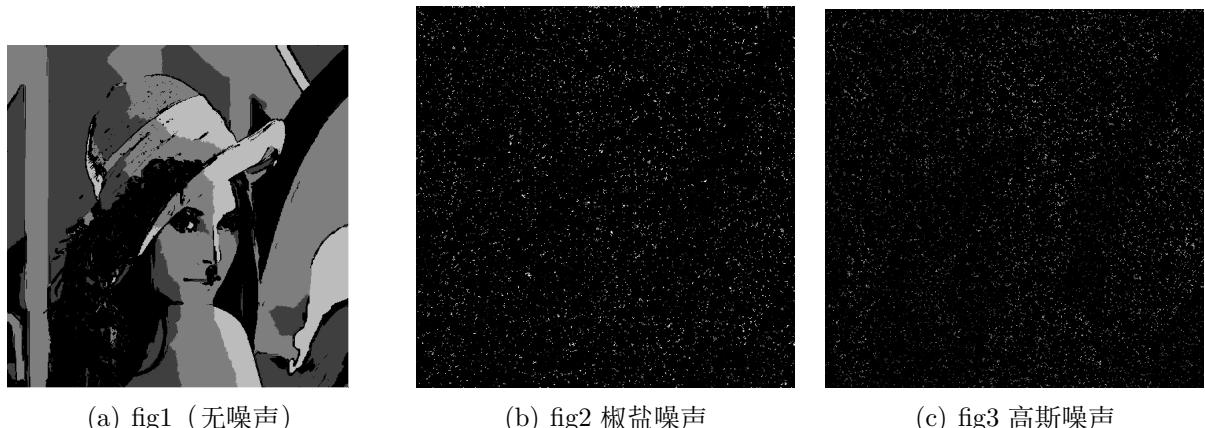


图 32: 区域生长处理的噪声图像

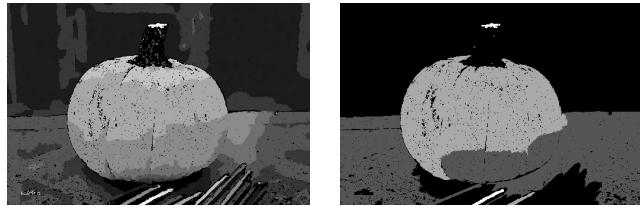


(a) fig1

(b) fig2

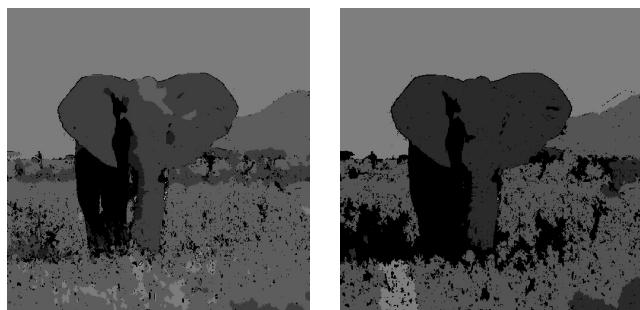
(c) fig3

图 33: 区域生长处理的复杂图像



(a) fig1 peak_distance=20

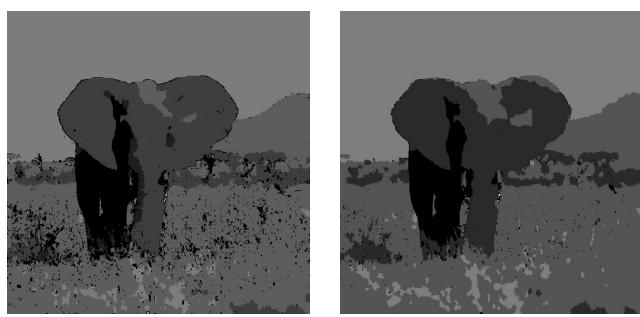
(b) fig2 peak_distance=60



(c) fig3 peak_distance=20

(d) fig4 peak_distance=60

图 34: 区域生长处理选取不同的种子（即峰值灰度）的复杂图像



(a) fig1 t=4

(b) fig2 t=8

图 35: 区域生长选择不同的阈值 t 的结果

3.5 总结

- 从图 34 可以得知，种子（峰值）的选取不同，其分离出的区域数量不同

2. 从图 35 可以得知, T (阈值) 的选取不同, 其划分的精细程度不同, T 越大, 小区域及噪声被单独划为一个区域的可能性越低

3. 区域生长对于简单的图像有较好的分割效果, 如图 31 所示, 基本可以将将物品和背景分离开。但是对于复杂图像的效果会受一些因素的影响, 比如: 物体会因为自身的阴影和纹理被划分为几部分 (图 34 fig1, 3), 当物体上一部分区域和环境比较接近时, 会被化为环境区域 (图 33 fig2)

4. 区域生长对于噪声的抵抗性不佳, 从图 32 可以看出, 区域生长在面对噪声时完全失效。

4 区域合并

4.1 简介

开始时将图像分割成一系列任意不相关的区域, 然后将它们合并或者拆分以满足限制条件, 这就是区域分裂与合并。

4.2 自回归的理解即实现

一种区域分裂方法是首先将图像等分为 4 个区域, 然后反复将分割得到的子图像再次分为 4 个区域, 直到对任意 R_i , $P(R_i)=\text{TRUE}$, 表示区域 R_i 已经满足相似性准则 (比如该区域内灰度值相等或相似), 此时不再进行分裂操作, 同时比较相邻区域, 如果满足相似性准则, 则合并。这个过程叫做自回归, 可以用四叉树形式表示。其实现如下:

```
1 def construct(self, img, loc):
2     root = Node(None, False, None, None, None, None,
3                 loc)
4     if img.shape[0] < 4:
5         root.isLeaf = True
6         root.mean_val = img.mean()
7         self.paint(loc, root.mean_val)
8     elif self.stop_split(img): # 判断是否继续分割
9         root.isLeaf = True
10        root.mean_val = img.mean()
11        self.paint(loc, root.mean_val)
12    else:
13        height = img.shape[0]
14        width = img.shape[1]
15        halfheight = height // 2
16        halfwidth = width // 2
```

```

16     root.isLeaf = False # 如果网格中有值不相等，这个节点就不是叶子节点
17
18     # 自回归
19     base_start_x = loc[0].x
20     base_start_y = loc[0].y
21     base_end_x = loc[1].x
22     base_end_y = loc[1].y
23     root.topLeft = self.construct(img[:halfheight,
24                                     :halfwidth], [Point(base_start_x,
25                                     base_start_y), Point(base_start_x +
26                                     halfheight, base_start_y + halfwidth)])
27     root.topRight = self.construct(img[:halfheight,
28                                     , halfwidth:], [Point(base_start_x,
29                                     base_start_y + halfwidth), Point(
30                                     base_start_x + halfheight, base_end_y)])
31     root.bottomLeft = self.construct(img[
32                                     halfheight:, :halfwidth], [Point(
33                                     base_start_x + halfheight, base_start_y),
34                                     Point(base_end_x, base_start_y + halfwidth)
35                                     ])
36     root.bottomRight = self.construct(img[
37                                     halfheight:, halfwidth:], [Point(
38                                     base_start_x + halfheight, base_start_y +
39                                     halfwidth), Point(base_end_x, base_end_y)])
40
41     if (root.topLeft.isLeaf and root.topRight.
42         isLeaf and np.abs(root.topRight.mean_val-
43         root.topLeft.mean_val)<3):
44         self.merge(root.topLeft, root.topRight)
45     if (root.topLeft.isLeaf and root.bottomLeft.
46         isLeaf and np.abs(root.bottomLeft.mean_val-
47         root.topLeft.mean_val)<3):
48         self.merge(root.topLeft, root.bottomLeft)
49     if (root.bottomRight.isLeaf and root.
50         bottomLeft.isLeaf and np.abs(root.
51         bottomLeft.mean_val-root.bottomRight.
52         mean_val)<3):
53         self.merge(root.bottomRight, root.
54         bottomLeft)

```

```

32         if (root.bottomRight.isLeaf and root.topRight.
33             isLeaf and np.abs(root.topRight.mean_val-
34             root.bottomRight.mean_val)<3):
            self.merge(root.bottomRight,root.topRight)
        return root

```

4.3 实验结果

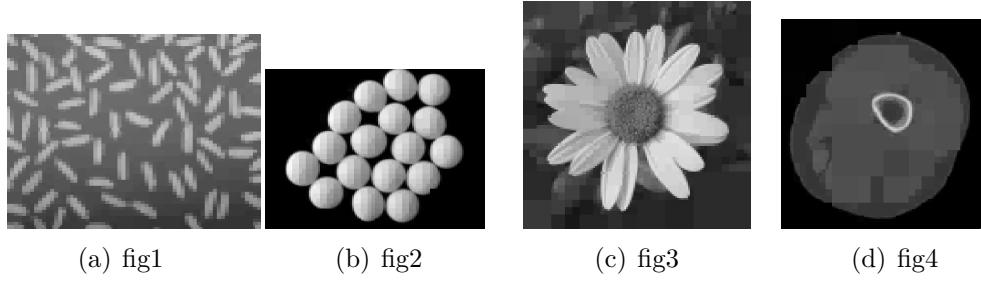


图 36: 区域合并处理的简单图像

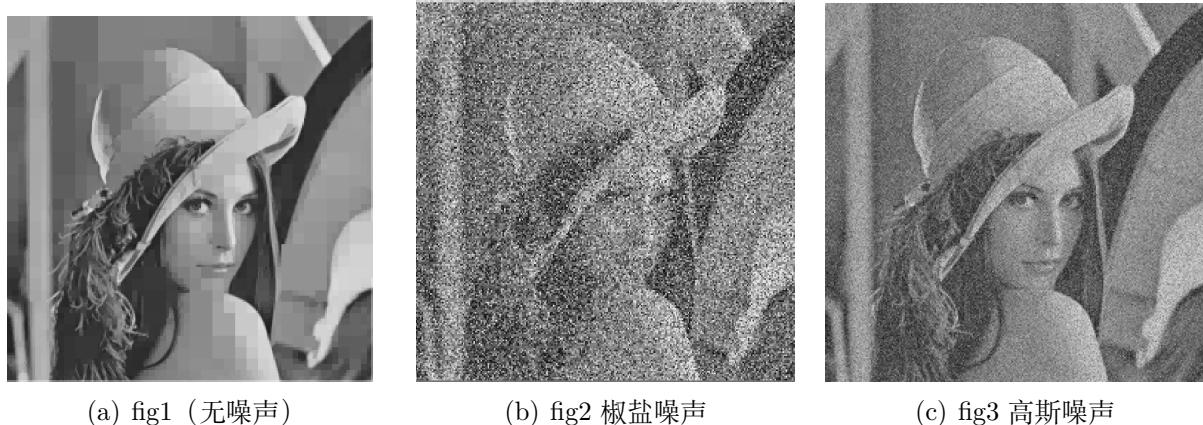


图 37: 区域合并处理的噪声图像



(a) fig1



(b) fig2



(c) fig3



(d) fig4



(e) fig4

图 38: 区域合并处理的复杂图像

4.4 总结

- 对于简单的图像，由于较大区域内的像素均符合相似性准则，所以物体有明显的“抹平”现象，即一部分区域转化成相同灰度的正方形。
- 对于复杂图像，“抹平”现象对于变化比较小的背景比较明显，而对于变化比较复杂的区域和物体则不明显，几乎看不出合并的现象，这是因为区域内像素不相似导致分裂一直持续，直到分裂为单个像素，这时和原图像没有区别。
- 对于噪声图像，由于有合并的存在，所以对于噪声有一定的抑制效果，其中对于高斯噪声的效果较好，对于椒盐噪声的效果不是那么好

5 总结

通过这次作业，自己实现了很多图像分离的方法，其中遇到了不少问题，但是最后通过努力搞清楚后，对于之前学习的知识点真的是一次巩固，理清和纠正了之前不少错误的思路，更深刻地理解了阈值分割、边缘检测、区域分割算法在图像处理中的作用。感谢这次作业给我带来的锻炼。