**实验报告—Graph cut算法**

小组成员：陈海龙 戴熹 杨琛璟 熊慧

1. 算法描述
2. 算法简介：

Graph cut是一种十分有用和流行的能量优化算法，在计算机视觉领域普遍应用于前背景分割（Image segmentation）、立体视觉（stereo vision）、抠图（Image matting）等。Graph cut算法是一种直接基于图切算法的图像分割技术。它仅需要在前景和背景处各画几笔作为输入，算法将建立各个像素点与前景背景相似度的赋权图，并通过求解最小切割区分前景和背景。

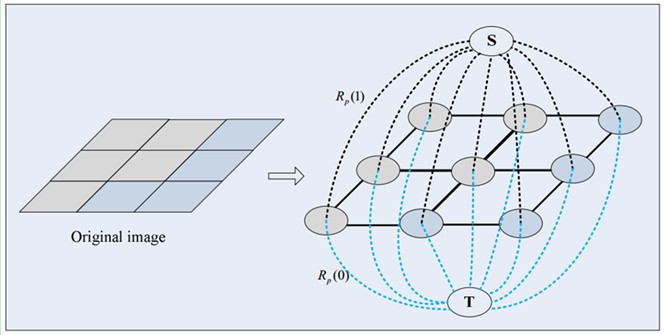
GrabCut是对其的改进版，是迭代的Graph Cut。该算法利用了图像中的纹理（颜色）信息和边界（反差）信息，只要少量的用户交互操作即可得到比较好的分割结果。

1. Graph cut算法原理
2. 此类方法把图像分割问题与图的最小割（min cut）问题相关联。

首先用一个无向图G=<V，E>表示要分割的图像，V和E分别是顶点（vertex）和边（edge）的集合。此处的Graph和普通的Graph稍有不同。普通的图由顶点和边构成，如果边的有方向的，这样的图被则称为有向图，否则为无向图，且边是有权值的，不同的边可以有不同的权值，分别代表不同的物理意义。而Graph Cuts图是在普通图的基础上多了2个顶点，这2个顶点分别用符号”S”和”T”表示，统称为终端顶点。其它所有的顶点都必须和这2个顶点相连形成边集合中的一部分。所以Graph Cuts中有两种顶点，也有两种边。

第一种顶点和边是：第一种普通顶点对应于图像中的每个像素。每两个邻域顶点（对应于图像中每两个邻域像素）的连接就是一条边。这种边也叫n-links。

第二种顶点和边是：除图像像素外，还有另外两个终端顶点，叫S（source：源点，取源头之意）和T（sink：汇点，取汇聚之意）。每个普通顶点和这2个终端顶点之间都有连接，组成第二种边。这种边也叫t-links。



上图就是一个图像对应的s-t图，每个像素对应图中的一个相应顶点，另外还有s和t两个顶点。上图有两种边，实线的边表示每两个邻域普通顶点连接的边n-links，虚线的边表示每个普通顶点与s和t连接的边t-links。在前后景分割中，s一般表示前景目标，t一般表示背景。

图中每条边都有一个非负的权值we，也可以理解为cost（代价或者费用）。一个cut（割）就是图中边集合E的一个子集C，那这个割的cost（表示为|C|）就是边子集C的所有边的权值的总和。

Graph Cuts中的Cuts是指这样一个边的集合，很显然这些边集合包括了上面2种边，该集合中所有边的断开会导致残留”S”和”T”图的分开，所以就称为“割”。如果一个割，它的边的所有权值之和最小，那么这个就称为最小割，也就是图割的结果。而福特-富克森定理表明，网路的最大流max flow与最小割min cut相等。所以由Boykov和Kolmogorov发明的max-flow/min-cut算法就可以用来获得s-t图的最小割。这个最小割把图的顶点划分为两个不相交的子集S和T，其中s ∈S，t∈ T和S∪T=V 。这两个子集就对应于图像的前景像素集和背景像素集，那就相当于完成了图像分割。

1. 图像分割可以看成pixel labeling（像素标记）问题。

目标（s-node）的label设为1，背景（t-node）的label设为0，这个过程可以通过最小化图割来最小化能量函数得到。那很明显，发生在目标和背景的边界处的cut就是我们想要的（相当于把图像中背景和目标连接的地方割开，那就相当于把其分割了）。同时，这时候能量也应该是最小的。假设整幅图像的标签label（每个像素的label）为L= {l1,l2,,,, lp }，其中li为0（背景）或者1（目标）。那假设图像的分割为L时，图像的能量可以表示为：

E(L)=aR(L)+B(L)

其中，R(L)为区域项（regional term），B(L)为边界项（boundary term），而a就是区域项和边界项之间的重要因子，决定它们对能量的影响大小。如果a为0，那么就只考虑边界因素，不考虑区域因素。E(L)表示的是权值，即损失函数，也叫能量函数，图割的目标就是优化能量函数使其值达到最小。

a)区域项：

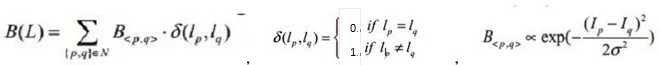
http://img.my.csdn.net/uploads/201301/23/1358872516_2332.jpg

其中Rp(lp)表示为像素p分配标签lp的惩罚，Rp(lp)能量项的权值可以通过比较像素p的灰度和给定的目标和前景的灰度直方图来获得，换句话说就是像素p属于标签lp的概率，我希望像素p分配为其概率最大的标签lp，这时候我们希望能量最小，所以一般取概率的负对数值，故t-link的权值如下：

Rp(1) = -ln Pr(Ip|’obj’)； Rp(0) = -ln Pr(Ip|’bkg’)

由上面两个公式可以看到，当像素p的灰度值属于目标的概率Pr(Ip|’obj’)大于背景Pr(Ip|’bkg’)，那么Rp(1)就小于Rp(0)，也就是说当像素p更有可能属于目标时，将p归类为目标就会使能量R(L)小。那么，如果全部的像素都被正确划分为目标或者背景，那么这时候能量就是最小的。

b)边界项：



其中，p和q为邻域像素，边界平滑项主要体现分割L的边界属性，B<p,q>可以解析为像素p和q之间不连续的惩罚，一般来说如果p和q越相似（例如它们的灰度），那么B<p,q>越大，如果他们非常不同，那么B<p,q>就接近于0。换句话说，如果两邻域像素差别很小，那么它属于同一个目标或者同一背景的可能性就很大，如果他们的差别很大，那说明这两个像素很有可能处于目标和背景的边缘部分，则被分割开的可能性比较大，所以当两邻域像素差别越大，B<p,q>越小，即能量越小。

1. 我们目标是将一幅图像分为目标和背景两个不相交的部分，我们运用图分割技术来实现。

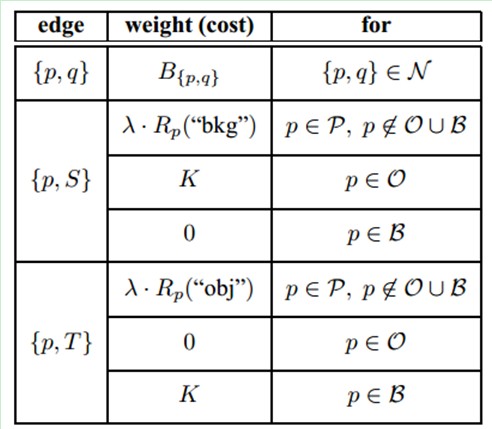
首先，图由顶点和边来组成，边有权值。那我们需要构建一个图，这个图有两类顶点，两类边和两类权值。普通顶点由图像每个像素组成，然后每两个邻域像素之间存在一条边，它的权值由上面说的“边界平滑能量项”来决定。还有两个终端顶点s（目标）和t（背景），每个普通顶点和s都存在连接，也就是边，边的权值由“区域能量项”Rp(1)来决定，每个普通顶点和t连接的边的权值由“区域能量项”Rp(0)来决定。这样所有边的权值就可以确定了，也就是图就确定了。这时候，就可以通过min cut算法来找到最小的割，这个min cut就是权值和最小的边的集合，这些边的断开恰好可以使目标和背景被分割开，也就是min cut对应于能量的最小化。而min cut和图的max flow是等效的，故可以通过max flow算法来找到s-t图的min cut。目前的算法主要有：

1) Goldberg-Tarjan

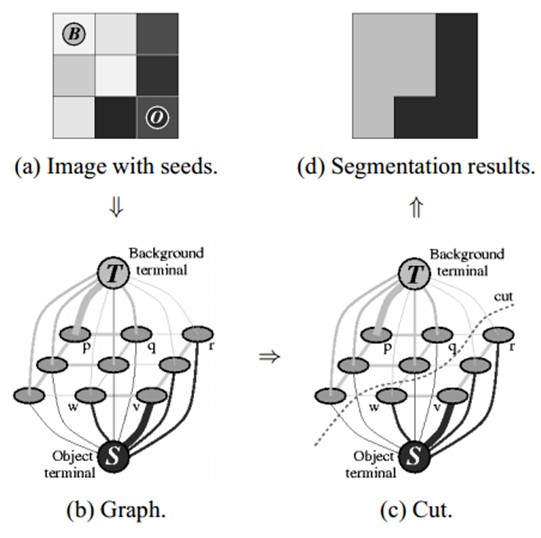
2) Ford-Fulkerson

3) 上诉两种方法的改进算法

1. 权值：



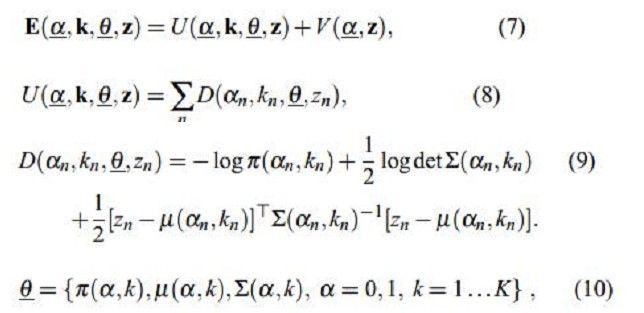
1. Graph cut的3x3图像分割示意图：我们取两个种子点（就是人为的指定分别属于目标和背景的两个像素点），然后我们建立一个图，图中边的粗细表示对应权值的大小，然后找到权值和最小的边的组合，也就是（c）中的cut，即完成了图像分割的功能。



1. Grab cut算法原理
2. 颜色模型

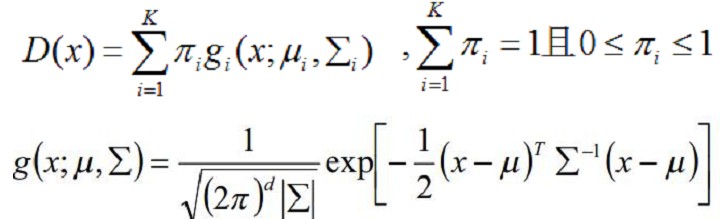
我们采用RGB颜色空间，分别用一个K个高斯分量（一取般K=5）的全协方差GMM（混合高斯模型）来对目标和背景进行建模。于是就存在一个额外的向量**k** = {k1, . . ., kn, . . ., kN}，其中kn就是第n个像素对应于哪个高斯分量，kn∈ {1, . . . K}。对于每个像素，要不来自于目标GMM的某个高斯分量，要不就来自于背景GMM的某个高斯分量。

用于整个图像的Gibbs能量为(7)

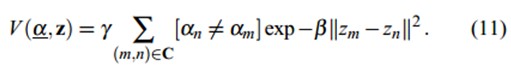


U就是区域项，表示一个像素被归类为目标或者背景，也就是某个像素属于目标或者背景的概率的负对数。

混合高斯密度模型：



所以取负对数之后就变成式（9）那样的形式了，其中GMM的参数**θ**就有三个：每一个高斯分量的权重π、每个高斯分量的均值向量**u**（因为有RGB三个通道，故为三个元素向量）和协方差矩阵**∑**（因为有RGB三个通道，故为3x3矩阵）。如式（10）。一旦确定了这三个参数，那么我们知道一个像素的RGB颜色值之后，就可以代入目标的GMM和背景的GMM，就可以得到该像素分别属于目标和背景的概率了，也就是Gibbs能量的区域能量项就可以确定了，即图的t-link的权值我们就可以求出。



边界项和之前说的Graph Cut的差不多，体现邻域像素m和n之间不连续的惩罚，如果两邻域像素差别很小，那么它属于同一个目标或者同一背景的可能性就很大，如果他们的差别很大，那说明这两个像素很有可能处于目标和背景的边缘部分，则被分割开的可能性比较大，所以当两邻域像素差别越大，能量越小。而在RGB空间中，衡量两像素的相似性，我们采用欧式距离（二范数）。这里面的参数β由图像的对比度决定，如果图像的对比度较低，也就是说本身有差别的像素m和n，它们的差||zm-zn||还是比较低，那么我们需要乘以一个比较大的β来放大这种差别，而对于对比度高的图像，那么也许本身属于同一目标的像素m和n的差||zm-zn||还是比较高，那么我们就需要乘以一个比较小的β来缩小这种差别，使得V项能在对比度高或者低的情况下都可以正常工作。常数γ为50（经过作者用15张图像训练得到的比较好的值）。OK，那这时候，n-link的权值就可以通过式（11）来确定了，这时候我们想要的图就可以得到了，我们就可以对其进行分割了。

1. 迭代能量最小化分割算法

Graph Cut的算法是一次性最小化的，而Grab Cut是迭代最小的，每次迭代过程都使得对目标和背景建模的GMM的参数更优，使得图像分割更优。我们直接通过算法来说明：

1. 初始化
2. 用户通过直接框选目标来得到一个初始的trimap T，即方框外的像素全部作为背景像素TB，而方框内TU的像素全部作为“可能是目标”的像素。
3. 对TB内的每一像素n，初始化像素n的标签αn=0，即为背景像素；而对TU内的每个像素n，初始化像素n的标签αn=1，即作为“可能是目标”的像素。
4. 经过上面两个步骤，可以分别得到属于目标（αn=1）的一些像素，剩下的为属于背景（αn=0）的像素，这时候，可以通过这个像素来估计目标和背景的GMM了。可以通过k-mean算法分别把属于目标和背景的像素聚类为K类，即GMM中的K个高斯模型，这时候GMM中每个高斯模型就具有了一些像素样本集，它的参数均值和协方差就可以通过他们的RGB值估计得到，而该高斯分量的权值可以通过属于该高斯分量的像素个数与总的像素个数的比值来确定。
5. 迭代最小化
6. 对每个像素分配GMM中的高斯分量（例如像素n是目标像素，那么把像素n的RGB值代入目标GMM中的每一个高斯分量中，概率最大的那个就是最有可能生成n的，也即像素n的第kn个高斯分量）：

IMG_256

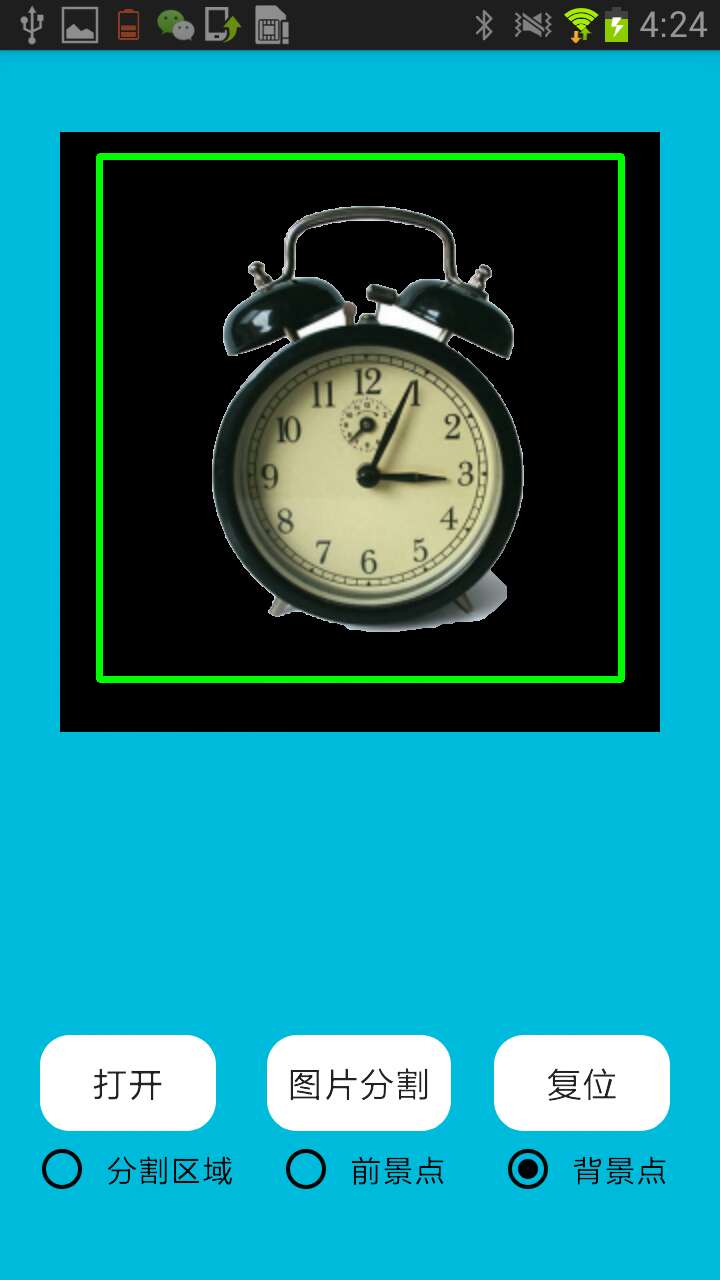
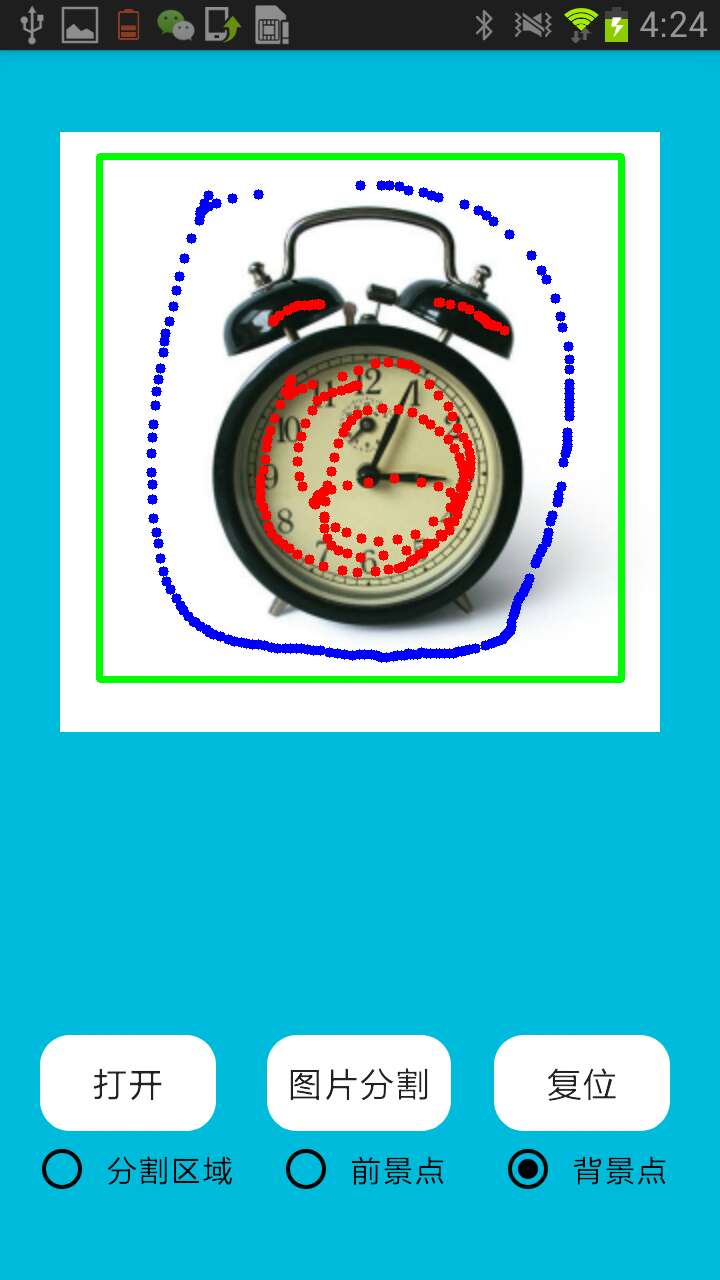
1. 对于给定的图像数据Z，学习优化GMM的参数（因为在步骤（a）中我们已经为每个像素归为哪个高斯分量做了归类，那么每个高斯模型就具有了一些像素样本集，这时候它的参数均值和协方差就可以通过这些像素样本的RGB值估计得到，而该高斯分量的权值可以通过属于该高斯分量的像素个数与总的像素个数的比值来确定。）：

IMG_256

1. 分割估计（通过1中分析的Gibbs能量项，建立一个图，并求出权值t-link和n-link，然后通过max flow/min cut算法来进行分割）：

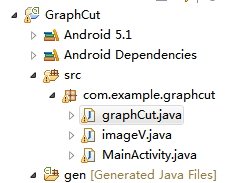
IMG_256

1. 重复步骤（a）到（c），直到收敛。经过（c）的分割后，每个像素属于目标GMM还是背景GMM就变了，所以每个像素的kn就变了，故GMM也变了，所以每次的迭代会交互地优化GMM模型和分割结果。另外，因为步骤（a）到（c）的过程都是能量递减的过程，所以可以保证迭代过程会收敛。
2. 采用border matting对分割的边界进行平滑等等后期处理。
3. 用户编辑（交互）
4. 编辑：人为地固定一些像素是目标或者背景像素，然后再执行一次中步骤（c）；
5. 重操作：重复整个迭代算法。（可选，实际上这里是程序或者软件抠图的撤销作用）
6. 缺点
7. 如果背景比较复杂或者背景和目标相似度很大，分割效果不太好
8. 速度略慢
9. **软件操作流程**
10. 整体界面如下图，包括一个图片显示区域、三个按钮与一个单选按钮组，单选按钮组包括三个单选按钮
11. 在单选按钮组中选择功能后，在图片上点击即可执行相应的操作，其中分割区域可以框定分割的区域，前景点用来标记前景，背景点用来标记背景
12. 点击打开按钮，可以在本地选择图片并打开
13. 点击图片分割按钮，可以在框定的分割区域中进行图片分割
14. 点击复位按钮，可以复位到图片初始状态



1. 关键代码
2. 外部库使用

使用了android中opencv库





1. 声明类的变量

**package** com.example.graphcut;

**import** java.util.Vector;

**import** org.opencv.android.Utils;

**import** org.opencv.core.Core;

**import** org.opencv.core.CvType;

**import** org.opencv.core.Mat;

**import** org.opencv.core.Point;

**import** org.opencv.core.Rect;

**import** org.opencv.core.Scalar;

**import** org.opencv.imgproc.Imgproc;

**import** android.graphics.Bitmap;

**import** android.util.Log;

**import** android.widget.ImageView;

**class** GCApplication

{

**public** **enum** State{ ***NOT\_SET*** , ***IN\_PROCESS*** , ***SET*** };

**public** **static** **final** **int** ***radius*** = 5;

**public** **static** **final** **int** ***thickness*** = -1;

**final** Scalar RED = **new** Scalar(255,0,0);

**final** Scalar PINK = **new** Scalar(230,130,255);

**final** Scalar BLUE = **new** Scalar(0,0,255);

**final** Scalar LIGHTBLUE = **new** Scalar(160,255,255);

**final** Scalar GREEN = **new** Scalar(0,255,0);

Mat image;

Mat mask;

Mat bgdModel = **null**, fgdModel = **null**;

Mat temp, res, binMask;

Bitmap showBitmap;

State rectState, lblsState, prLblsState;

**public** **boolean** isInitialized;

Rect rect = **new** Rect();

Vector<Point> fgdPxls = **new** Vector<Point>();

Vector<Point> bgdPxls = **new** Vector<Point>();

Vector<Point> prFgdPxls = **new** Vector<Point>();

Vector<Point> prBgdPxls = **new** Vector<Point>();

**int** iterCount;

1. 载入图片，创建图片的Mat对象，初始化各成员变量

**public** **void** setImage(Bitmap bitmap)

{

**if** (bitmap == **null**)

**return**;

Log.*e*("bit",bitmap.getWidth()+" "+bitmap.getHeight());

Mat tempimage = **new** Mat();

Utils.*bitmapToMat*(bitmap, tempimage);

Mat image = **new** Mat();

Imgproc.*cvtColor*(tempimage, image, Imgproc.***COLOR\_RGBA2RGB***);

**this**.image = image;

mask = **new** Mat(image.size(), CvType.***CV\_8UC1***);

mask.setTo(Scalar.*all*(Imgproc.***GC\_BGD***));

temp = **new** Mat(image.size(), CvType.***CV\_8UC1***);

temp.setTo(Scalar.*all*(1));

bgdModel = **new** Mat(); fgdModel = **new** Mat();

showBitmap = Bitmap.*createBitmap*(bitmap);

binMask = **new** Mat(image.size(), CvType.***CV\_8UC1***);

reset();

}

1. 复位函数

**public** **void** reset(){

**if**( !mask.empty() )

mask.setTo(Scalar.*all*(Imgproc.***GC\_BGD***));

bgdPxls.clear(); fgdPxls.clear();

prBgdPxls.clear(); prFgdPxls.clear();

isInitialized = **false**;

rectState = State.***NOT\_SET***;

lblsState = State.***NOT\_SET***;

prLblsState = State.***NOT\_SET***;

iterCount = 0;

}

1. 获取触屏输入，根据输入设置分割区域、前景点、背景点，并将数据标记到mask中。

**public** **void** ontouch( **int** event, **int** x, **int** y ,**int** flag)

{

**switch**( event )

{

**case** 0: // Touch Down

{

**if**( flag == 0 && rectState == State.***NOT\_SET*** )

{

rectState = State.***IN\_PROCESS***;

rect = **new** Rect( x, y, 1, 1 );

}

**if** ( (flag == 1 || flag == 2) && rectState == State.***SET*** )

lblsState = State.***IN\_PROCESS***;

**if** ( (flag == 3 || flag == 4) && rectState == State.***SET*** )

prLblsState = State.***IN\_PROCESS***;

}

**break**;

**case** 1: //Touch Up

**if**( rectState == State.***IN\_PROCESS*** )

{

rect = **new** Rect( **new** Point(rect.x, rect.y), **new** Point(x,y) );

rectState = State.***SET***;

setRectInMask();

**assert**( bgdPxls.isEmpty() && fgdPxls.isEmpty() && prBgdPxls.isEmpty() && prFgdPxls.isEmpty() );

}

**if**( lblsState == State.***IN\_PROCESS*** )

{

setLblsInMask(flag, **new** Point(x,y), **false**);

lblsState = State.***SET***;

}

**if**( prLblsState == State.***IN\_PROCESS*** )

{

setLblsInMask(flag, **new** Point(x,y), **true**);

prLblsState = State.***SET***;

}

**break**;

**case** 2: //Touch Move

**if**( rectState == State.***IN\_PROCESS*** )

{

rect = **new** Rect( **new** Point(rect.x, rect.y), **new** Point(x,y) );

**assert**( bgdPxls.isEmpty() && fgdPxls.isEmpty() && prBgdPxls.isEmpty() && prFgdPxls.isEmpty() );

}

**else** **if**( lblsState == State.***IN\_PROCESS*** )

{

setLblsInMask(flag, **new** Point(x,y), **false**);

}

**else** **if**( prLblsState == State.***IN\_PROCESS*** )

{

setLblsInMask(flag, **new** Point(x,y), **true**);

}

**break**;

}

}

**private** **void** setRectInMask(){

**assert**( !mask.empty() );

mask.setTo( **new** Scalar(Imgproc.***GC\_BGD***));

rect.x = Math.*max*(0, rect.x);

rect.y = Math.*max*(0, rect.y);

rect.width = Math.*min*(rect.width, image.cols()-rect.x);

rect.height = Math.*min*(rect.height, image.rows()-rect.y);

(mask.submat(rect)).setTo( **new** Scalar(Imgproc.***GC\_PR\_FGD***) );

}

**private** **void** setLblsInMask( **int** flag, Point p, **boolean** isPr ){

Vector<Point> bpxls, fpxls;

**int** bvalue, fvalue;

**if**( !isPr )

{

bpxls = bgdPxls;

fpxls = fgdPxls;

bvalue = Imgproc.***GC\_BGD***;

fvalue = Imgproc.***GC\_FGD***;

}

**else**

{

bpxls = prBgdPxls;

fpxls = prFgdPxls;

bvalue = Imgproc.***GC\_PR\_BGD***;

fvalue = Imgproc.***GC\_PR\_FGD***;

}

**if**( flag == 1 || flag == 3)

{

fpxls.add(p);

Core.*circle*( mask, p, ***radius***, **new** Scalar(fvalue), ***thickness*** );

}

**if**( flag == 2 || flag == 4 )

{

bpxls.add(p);

Core.*circle*( mask, p, ***radius***, **new** Scalar(bvalue), ***thickness*** );

}

}

1. 输入参数image、mask、rect，调用opencv的grabcut函数进行图片分割，将结果记录到mask中。

**public** **int** nextIter()

{

**if**( isInitialized )

Imgproc.*grabCut*( image, mask, rect, bgdModel, fgdModel, 1 );

**else**

{

**if**( rectState != State.***SET*** )

**return** iterCount;

**if**( lblsState == State.***SET*** || prLblsState == State.***SET*** )

Imgproc.*grabCut*( image, mask, rect, bgdModel, fgdModel, 1, Imgproc.***GC\_INIT\_WITH\_MASK*** );

**else**

Imgproc. *grabCut*( image, mask, rect, bgdModel, fgdModel, 1, Imgproc.***GC\_INIT\_WITH\_RECT*** );

isInitialized = **true**;

}

iterCount++;

bgdPxls.clear(); fgdPxls.clear();

prBgdPxls.clear(); prFgdPxls.clear();

**return** iterCount;

}

1. 根据mask计算出实际显示的图片掩码binMask，根据binMask确定要显示的点并复制到res中，返回图片并进行显示

**public** Bitmap showImage()

{

**if**( image.empty() )

**return** **null**;

res = **new** Mat();

**if**( !isInitialized )

image.copyTo( res );

**else**

{

getBinMask( mask, binMask );

image.copyTo( res, binMask);

}

**for**( Point it : bgdPxls)

Core.*circle*( res, it, ***radius***, BLUE, ***thickness*** );

**for**( Point it : fgdPxls)

Core.*circle*( res, it, ***radius***, RED, ***thickness*** );

**for**( Point it : prBgdPxls)

Core.*circle*( res, it, ***radius***, LIGHTBLUE, ***thickness*** );

**for**( Point it : prFgdPxls)

Core.*circle*( res, it, ***radius***, PINK, ***thickness*** );

**if**( rectState == State.***IN\_PROCESS*** || rectState == State.***SET*** )

Core.*rectangle*( res, **new** Point( rect.x, rect.y ), **new** Point(rect.x + rect.width, rect.y + rect.height ), GREEN, 5);

Utils.*matToBitmap*(res, showBitmap);

**return** showBitmap;

}

**private** **final** **void** getBinMask( Mat comMask, Mat binMask )

{

**if**( comMask.empty() || comMask.type()!=CvType.***CV\_8UC1*** )

Log.*e*("error","comMask is empty or has incorrect type (not CV\_8UC1)");

**if**( binMask.empty() || binMask.rows() != comMask.rows() || binMask.cols() != comMask.cols() )

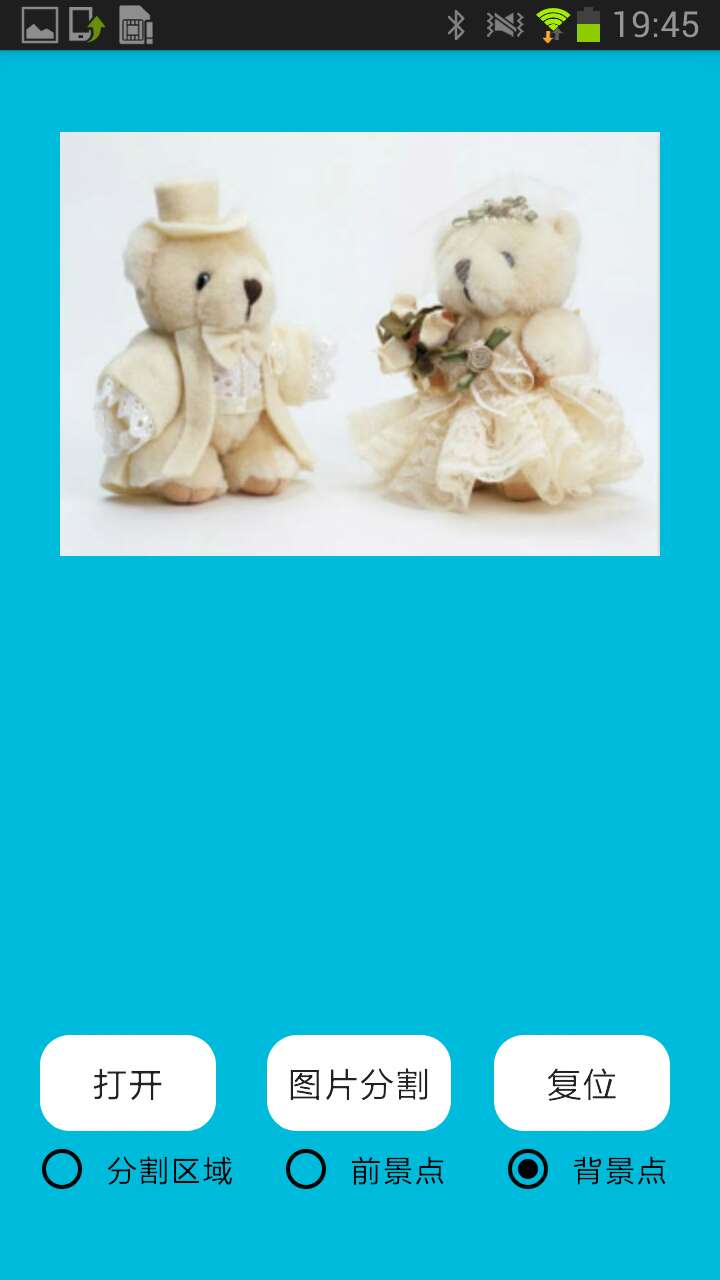
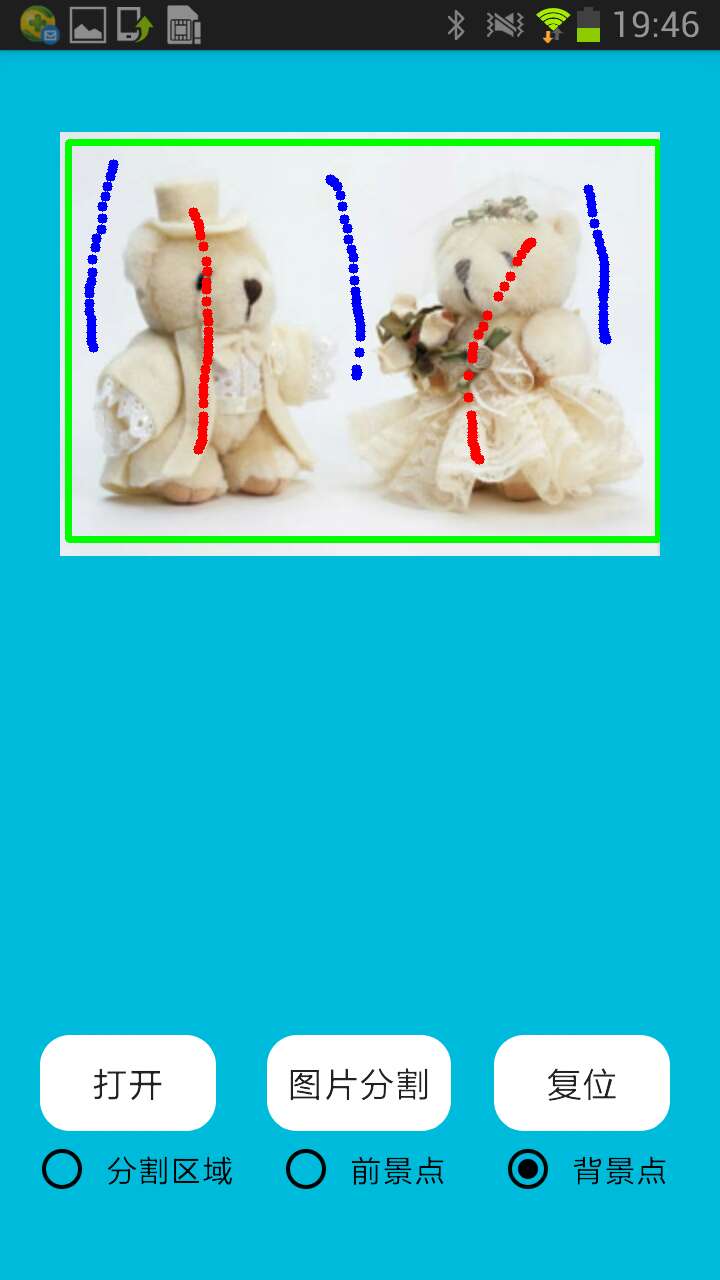
binMask.create( comMask.size(), CvType.***CV\_8UC1*** );

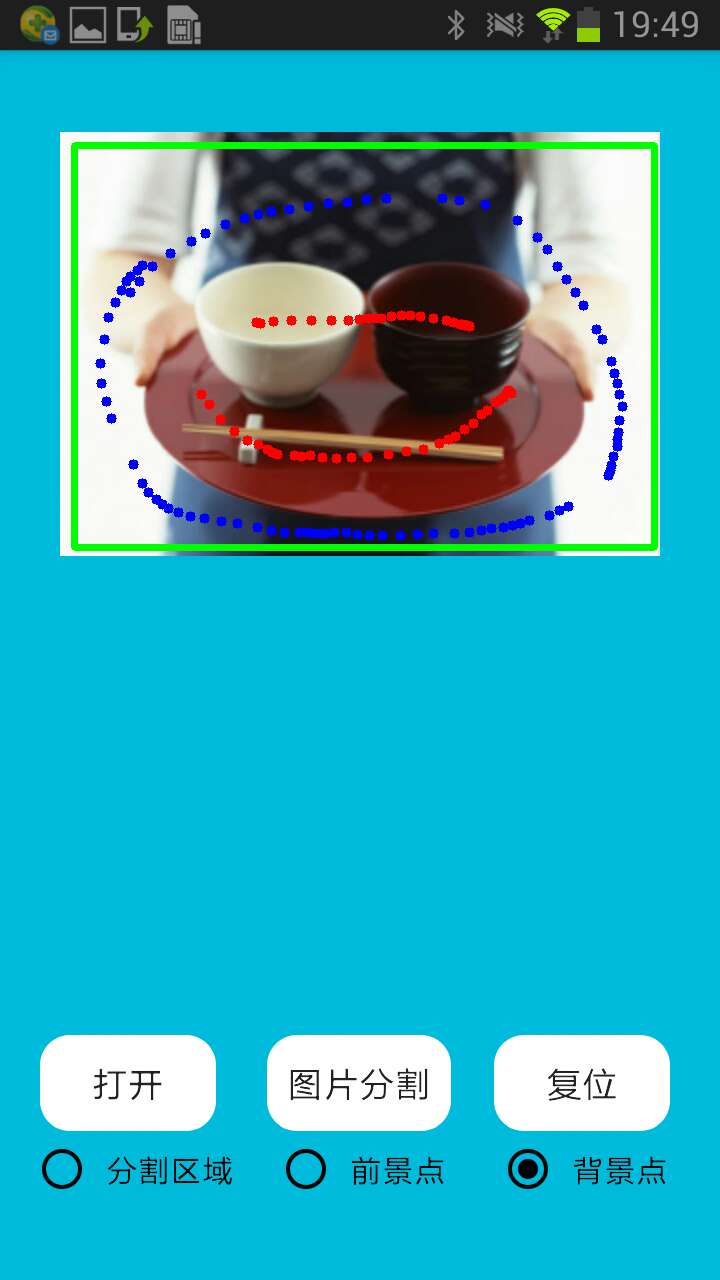
Core.*bitwise\_and*(comMask, temp, binMask);

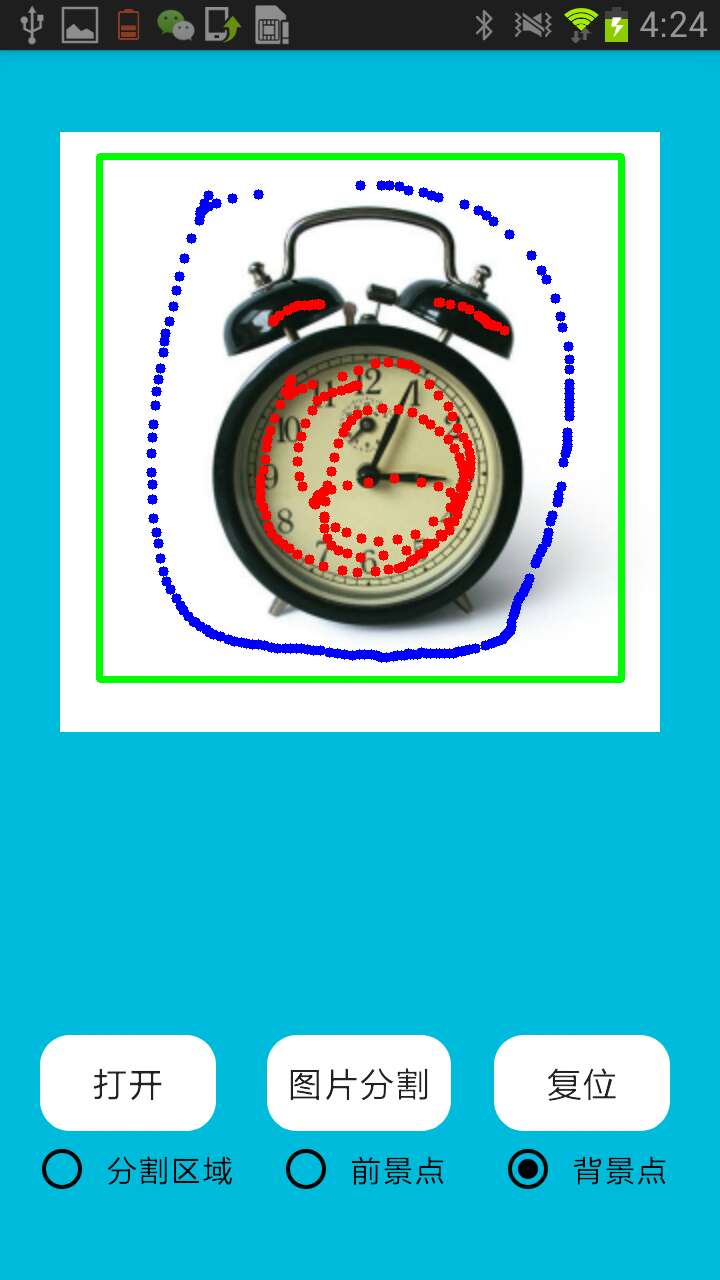
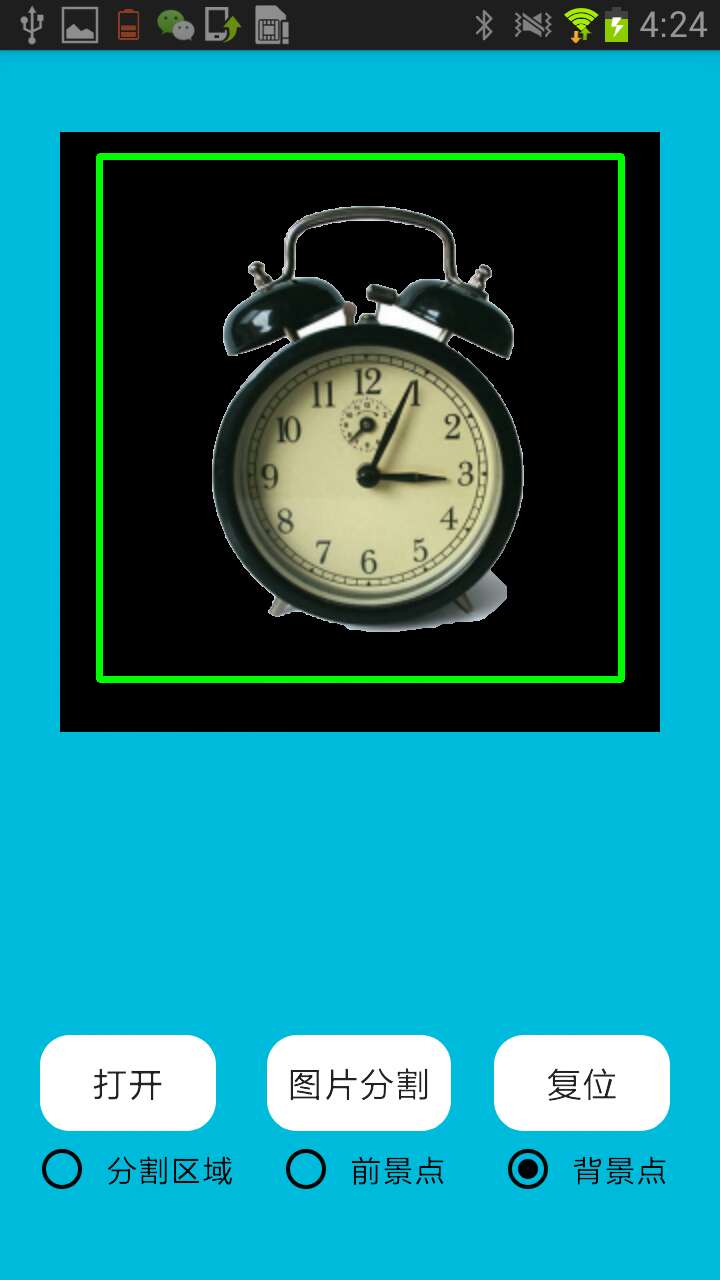
}

}

1. 实验结果
2. 截图







1. 两个算法的对比

（1）Graph Cut的目标和背景的模型是灰度直方图，Grab Cut取代为RGB三通道的混合高斯模型GMM；

（2）Graph Cut的能量最小化（分割）是一次达到的，而Grab Cut取代为一个不断进行分割估计和模型参数学习的交互迭代过程；

（3）Graph Cut需要用户指定目标和背景的一些种子点，但是Grab Cut只需要提供背景区域的像素集就可以了。也就是说你只需要框选目标，那么在方框外的像素全部当成背景，这时候就可以对GMM进行建模和完成良好的分割了。即Grab Cut允许不完全的标注（incomplete labelling）。