信息隐藏——LSB隐写分析

原创

HizT_1999 于 2020-06-24 21:37:59 发布 4709 收藏 28

分类专栏： 信息隐藏 文章标签： 信息安全 matlab

本文链接：https://blog.csdn.net/HizT_1999/article/details/106951364

LSB隐写分析

【实验目的】：

了解并实现常见的LSB隐写分析法。

【实验内容】：

■实现针对LSB隐写的卡方隐写分析算法，并分析其性能。

■实现针对LSB隐写的RS隐写分析算法，并分析其性能。

1.卡方隐写分析算法

主要针对图像所有像素点的LSB全嵌入情况；利用数理统计假设检验中的卡方检验模型来分析。

设图像中灰度值为j的象素数为hj，其中0≤j≤255。如果载体图像未经隐写，h2i和h2i+1的值会相差很大。秘密信息在嵌入之前往往经过加密，可以看作是0、1 随机分布的比特流，而且值为0与1的可能性都是1/2。如果秘密信息完全替代载体图像的最低位，那么h2i和h2i+1的值会比较接近，可以根据这个性质判断图像是否经过隐写。

嵌入信息会改变直方图的分布，由差别很大变得近似相等，但是却不会改变h2i+h2i+1的值，因为样值要么不改变，要么就在h2i和h2i+1之间改变。

$$h_{2i}^{*} = \frac{h_{2i} + h_{2i+1}}{2} \quad q = \frac{h_{2i} - h_{2i+1}}{2}$$

令 ；

当2h2i*较大时，根据中心极限定理，下式成立：

$$\frac{h_{2i} - h_{2i+1}}{\sqrt{2h_{2i}^{*}}} = \sqrt{2} \cdot \frac{h_{2i} - h_{2i}^{*}}{\sqrt{h_{2i}^{*}}} \to N(0,1)$$

所以，

$$r = \sum_{i=1}^{k} \frac{(h_{2i} - h_{2i}^{*})^2}{h_{2i}^{*}}$$

服从卡方分布。

结合卡方分布的密度计算函数计算载体被隐写的可能性为：

$$p = 1 - \frac{1}{2^{\frac{k-1}{2}}\Gamma(\frac{k-1}{2})} \int_0^r \exp(-\frac{t}{2}) t^{\frac{k-1}{2}-1} dt$$

如果p接近于1，则说明载体图像中含有秘密信息。

2.RS隐写分析算法

对于一个M*N像素的图片，设各个像素的值取自集合P，例如一个8bit的灰度图像，P={0,1,2,…,255}。将这些像素分为有着n个相邻像素的子集，例如n可以取值为4，记为G=（x1，x2，x3，x4）。进一步利用如下函数表示图像块的空间相关性

$$f(x_1, x_2, \cdots x_n) = \sum_{i=1}^{n-1} |x_{i+1} - x_i|$$

定义集合P上的3个函数：
交换函数F1：

$$2n \leftrightarrow 2n+1 \; 0 \leftrightarrow 1, 2 \leftrightarrow 3, \cdots 254 \leftrightarrow 255$$

，

偏移函数F-1：

$$2n \leftrightarrow 2n-1 \; -1 \leftrightarrow 0, 1 \leftrightarrow 2, \cdots 255 \leftrightarrow 256$$

，

恒等变换F0：

$$F_0(x) = x, x \in P$$

Regular G：$G \in R \Leftrightarrow f(F(G)) > f(G)$ 也就是说对G中的元素进行变换之后增大了元素之间的差别程度，R表示正则组。
Singular G：$G \in S \Leftrightarrow f(F(G)) < f(G)$ 也就是说对G正的元素进行变换之后减小了元素之间的差别程度，S表示奇异组。
Unusable G：$G \in U \Leftrightarrow f(F(G)) = f(G)$ ，也就是对G中的元素进行变换之后元素之间的差别程度几乎不变，G表示无用组。

引入伪装M，它的取值范围是-1，0，1。对应着
F-1，F0，F1三个变换函数。
当在载体中嵌入了秘密信息的话，就会有下面的式子成立：

$$R_{-M} - S_{-M} > R_M - S_M$$

结合如下方程：

$$2(d_1+d_0)x^2 + d_{-0} - d_{-1} - d_1 - 3d_0)x + d_0 - d_{-0} = 0$$

其中：

$$d_0 = R_M(p/2) - S_M(p/2), d_1 = R_M(1-p/2) - S_M(1-p/2)$$
$$d_{-0} = R_{-M}(p/2) - S_{-M}(p/2), d_{-0} = R_{-M}(1-p/2) - S_{-M}(1-p/2)$$

解上述方程，取绝对值较小的x，计算嵌入率p为：$p = x/(x-1/2)$。

【实验分析】：

1.卡方分析

使用matlab来实现卡方分析算法。
首先，将信息隐藏进一张灰度图中，注意信息长度要足够，否则会导致嵌入率过低引发的检测失败。

```
        end
        p = p + 1;
    end
行窗口
>> Embed_sub('inputgray.bmp','msg.txt','hide.bmp');
>>
```


```
>> kafang('hide.bmp');
    0.9879

图像是信息隐藏之后的
```

在这里稍微分析一下kafang.m这个文件。
与算法的结构几乎一模一样，

```
r=0; %记录卡方统计量
K=0;
p_num = p_num-1;
for i=1:p_num
    if(count(2*i-1)+count(2*i))~=0
        r=r+(count(2*i-1)-count(2*i))^2/(2*(count(2*i-1)+count(2*i)));
        K=K+1;
    end
end
    %r为卡方统计量，K-1为自由度
```

上图为计算卡方统计量和自由度的循环。

```
flag = chi2cdf(r,K-1);
P=1-flag;
```

这里则是计算

$$p = 1 - \frac{1}{2^{\frac{k-1}{2}}\Gamma(\frac{k-1}{2})}\int_0^r \exp(-\frac{t}{2})t^{\frac{k-1}{2}-1}dt$$

的部分代码。
最后，根据p的值决定其是否为隐藏信息后的图片。
当信息的长度不足时，或者设定的判断值p0不合理时，很容易出现判断出错。总的来说，不论是从我的代码还是从算法本身，卡方分析我认为仅仅适用于那种接近满嵌的图片，他不适用于实际使用，但可以作为一个评判标准。

2.RS分析

老规矩先来讨论一下RS分析的实现。

```
>> Embed_sub('inputgray.bmp','msg.txt','RShide.bmp');
>>
```

我们使用的是跟上一个实验相同的嵌入函数，在这里我觉得这并不影响我们的结果。

```
>> rstest('RShide.bmp');
是隐藏之后的
    0.2122

fx >>
```

这里是我自己写的一个rstest.m，其中包括获取图片，利用RS算法思想计算其嵌入率以及是否是嵌入信息的图片。

```
>> rstest('RShide.bmp');
是隐藏之后的
    0.2122

>> rstest('inputgray.bmp');
是隐藏之前的
   -0.0722

fx >>
```

我们不妨对照一下完全没有嵌入的图片的结果，看得出来还是比较可观的。
之所以结果是-0.07是因为，毕竟只是一个估算值，应该允许其误差。结果与0已经足够接近了。

```
8 -        d0 = rp-sp;
9 -        d1 = rp1-sp1;
0 -        d_0 = rn-sn;
1 -        d_1 = rn1-sn1;
2
3 -        delta = (d_0 - d_1 - d1 - 3d0)^2 - 4*2*(d1+d0)*(d0-d_0);
4 -        x1 = (-(d_0 - d_1 - d1 - 3d0)+sqrt(delta))/(2*2*(d1+d0));
5 -        x2 = (-(d_0 - d_1 - d1 - 3d0)-sqrt(delta))/(2*2*(d1+d0));
6
```

这部分是计算嵌入率的，与算法中的思想完全一致，甚至参数我都没有更换他们的名字。

```
case 1                  %交换函数F1
    cp = cp + 1;
    for i = 1:64
        x = col(i);
        if(mod(x, 2)==0)
            col(i) = x+1;
        else
            col(i) = x-1;
        end
    end
    pca = getPixelCorrelation(col);
    if pca > pcb
        rp = rp + 1;
    elseif pca < pcb
        sp = sp + 1;
    end
case 2                  %偏移函数F-1
    cn = cn + 1;
    for i = 1:64
        x = col(i);
        if(mod(x, 2)==0)
            col(i) = x-1;
        else
            col(i) = x+1;
        end
    end
    pca = getPixelCorrelation(col);
    if pca > pcb
```

```
        rn = rn + 1;
    elseif pca < pcb
        sn = sn + 1;
    end
case 3                      %恒等变换F0
```

这部分对应的是RS分析的三个函数，我都分别做了注释，其中，代码中的getPixelCorrelation是一个外部函数：

```
function [sum] = getPixelCorrelation(col)  % 获取像素相互关系
    len = size(col,1);
    sum = 0;
    for z = 2:len
        sum = sum + abs(col(z-1) - col(z));
    end
end
```

其实很简单，就是下图的步骤：

$$f(x_1, x_2, \cdots x_n) = \sum_{i=1}^{n-1} \left| x_{i+1} - x_i \right|$$

在循环中，我们可以看到在不断计算更新rn,rp,sn,sp的值，最后将作为rs.m的返回值返回给rstest.m。

```
53 -        rp = rp / cp;
54 -        sp = sp / cp;
55 -        rn = rn / cn;
56 -        sn = sn / cn;
```

```
5 -     [rp, sp, rn, sn]=rs(img);
6 -     [rp1, sp1, rn1, sn1]=rs(img_1);
```

当然需要解释一下的是，img和img_1分别对应着原图像矩阵与LSB位取反的图像矩阵。
性能分析：
RS分析算法相比于卡方分析，从实现的角度来说难了不少，必然有其存在的意义，他不仅能检测是否是嵌入信息的图片，还能够估算出嵌入率。

```
>> rstest('RShide.bmp');
是隐藏之后的
    0.2122

>> rstest('inputgray.bmp');
是隐藏之前的
   -0.0722

fx >>
```

但我觉得我个人的实现代码比较粗糙，从性能的角度来说，算法本身的性能必然是非常快速的，因为它不存在什么迭代和嵌套循环。除此之外，由于本次实验时间有限，我都是进行的灰度图像的检测，RGB图像的检测在之后的学习中有机会再进一步完善。

【实验代码】

information_hidding