

南昌大学

软件学院大作业

课程名称： 信息隐藏技术

题 目： 数字指纹算法的设计与测试

专 业： 信息安全

班 级： 信息安全 193 班

学 号： 8003119100

学生姓名： 丁俊

完成人数： 1

起讫日期： 2022.5.13 至 2022.6.12

任课教师： 刘凌锋 职称： 副教授

系主任： 邹春华

完成时间： 2022.06.12

得 分： _____

自适应数字图像水印算法的设计与评估

专 业： 信息安全

学 号： 8003111900

学生姓名： 丁俊

指导教师： 刘凌锋

摘要

近年来，信息技术的迅猛发展以及以信息技术为基础的电子商务的广泛应用使各类文字、图片、音乐、影视等作品通过网络传播范围空前扩大，为创作者和发行商带来了新的机遇，但同时，以数字形式存在的产品很容易被非法拷贝和分发，如何对数字化产品进行版权保护已经成为信息时代产权保护的核心问题之一。

数字水印技术是近几年发展起来的新型数字版权保护技术，发行商通过在其所要发售的拷贝中嵌入与购买者有关的水印密码信息可以对盗版行为进行跟踪。而水印技术能被有效地应用于版权保护的关键在于：嵌入的水印兼备不可见性和鲁棒性。对于图像水印算法，使用自适应的方案让图像自主选择嵌入区域，是影响水印的不可见性和鲁棒性的关键因素。自适应调参模型的实质是嵌入参数（用于控制水印嵌入的强度），或者是根据某些图像的属性（均值、方差、熵等）来选择嵌入的位置和大小等，可以降低水印嵌入的不确定性带来的弱鲁棒性，提高水印的抗攻击能力。本文首先介绍根据图像熵值高低嵌入水印的自适应算法的原理，给出嵌入及提取算法步骤和流程，最后对水印图像进行不同种类的攻击测试，评估其水印算法性能，如鲁棒性、不可见性等。

关键词： 自适应；嵌入算法；鲁棒性；攻击测试

目录

摘要	2
第一章 自适应数字图像水印算法	4
1.1 原理介绍	4
1.2 数字图像水印算法	4
第三章 对数字图像水印算法的攻击测试	11
3.1 抗攻击测试	11
3.2 鲁棒性攻击与评估	17
3.3 水印的不可见性	19
四、总结分析	19
参考文献	20

第一章 自适应数字图像水印算法

1.1 原理介绍

算法是直接将秘密水印图像嵌入载体图像的一种最简单的方法，指直接用水印图像像素值的高 4 位 bit 去替换载体图像像素值的低 4bit。我们这里使用的秘密水印图像是二值或灰度图像。但无论如何选择图像 RGB 三层中的哪层嵌入，都会在不同程度上对原始图像造成破坏。将水印图像隐藏在某一层中，容易使得该像素点的色彩的突出分量突出。所以，我们并不能笼统的认为图像隐藏在某层比较好而隐藏在某层不好，这是因为对于具体的某个像素点其哪个颜色分量突出是不确定的。但是，我们可以通过改进算法来限制这种颜色沿相应坐标的绝对偏移。

首先，我们引入一个相似度的概念。所谓相似度，是指两图像块中同一坐标下的像素中第 4bit 相同的像素数量占一块图像全部像素的比例，表示为 $\mu = s/64$ 。其中 s 为第 4bit 相同的像素数量为 $8*8$ 块中的总像素数。我们知道图像像素点的最低有效位不会影响人们肉眼的观察，所以这里采取了将水印图像像素点的高 4 位嵌入图像像素点的低 4 位，可能会造成图像的部分失真，由此想到使用采用图像降级的方式降低这种影响，还将图像进行分块，每块 $8*8$ 的像素，进行二维到一维的空域变换。此外，根据图像的熵值进入水印信息的嵌入，依据载体图像和水印图像的熵值降序，自适应依据熵值的高低进行嵌入的位置选择，即将水印图像中熵值高的图像块隐藏在载体图像中熵值高的图像块中。

1.2 数字图像水印算法

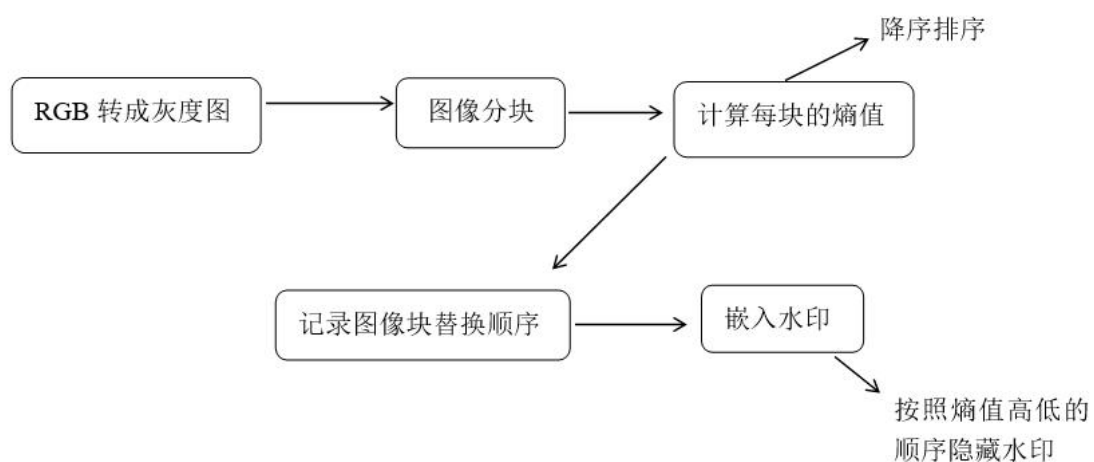
图像的熵表示为图像灰度级集合的比特平均数，描述了图像信源的平均信息量，熵指的是体系的混乱程度，表示图像灰度分布的聚集特征。本文将对原图像和水印图像进行分块，每块的大小为 $8*8$ ，并计算每个块的熵值，随后对所有熵值按照从大到小的顺序排序。将熵值保存到一个二维数组 e 中，其中 $e(i,j)$ 表示的是第 $i+1$ 行 $j+1$ 列对应的图像块的熵值，这里我们将数组变成一维数组，假如图像的大小为 $400*400$ 像素，那么就被分成了 50 行*50 列的图像 $8*8$ 图像块，从一维熵值数组转变成二维熵值数组只需要对 50 取余就行了。

把一维熵值数组进行降维，然后降序排序，分别获得原图像和水印图像的从

大到小的熵值。利用图像降级的方法，将水印图像中熵值高的图像块隐藏在载体图像中熵值高的图像块中。实现自适应嵌入水印的位置，同时，需要保存熵值排序后的索引值，便于水印的提取过程。

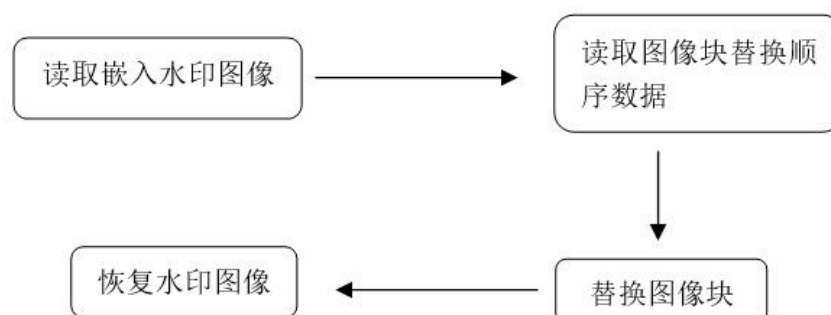
水印嵌入过程：

- 1、将 RGB 图像转化为灰度图，计算长度和宽度；
- 2、对图像进行分块(8*8 一小块)，计算每块的熵值；
- 3、对熵值矩阵进行降维，降序排列；
- 4、将水印图像中熵值高的图像块隐藏在原图像熵值高的图像块中；
- 5、保存图像块对应的替换顺序；



水印提取过程：

- 1、读取嵌入水印图像；
- 2、读取图像块替换顺序数据
- 3、替换图像块；
- 4、恢复水印图像；



嵌入水印算法流程:

- 1、将 RGB 图像转化为灰度图，计算图像的长度和宽度
获取图像的大小：长度和宽度

```
clear;
img_hidden=imread('dingjun.jpg');
img_hidden=rgb2gray(img_hidden);
imwrite(img_hidden,'dingjun.bmp');
img=imread('source.jpg');
img=rgb2gray(img);% 将 RGB 图像转化为灰度图
imwrite(img,'source.bmp');% 将灰度图保存
[row_h,col_h]=size(img_hidden); % 水印图像的大小
w_h = row_h/8; % 一行方块的数量
h_h = col_h/8; % 一列方块的数量
[row,col]=size(img); % 原图像的大小
w = row/8;
h = col/8;
```

- 2、对图像进行分块，并计算每一块的熵值

```
for i=1:row_h/8
    for j=1:col_h/8
        block=img((i-1)*8+1:i*8,(j-1)*8+1:j*8,:);
        block_h=img_hidden((i-1)*8+1:i*8,(j-1)*8+1:j*8,:);
        e(i,j)=entropy(block);
        e_h(i,j)=entropy(block_h);
    end
end
```

将图像划分成 8×8 的小块，那么图像总共被分成了 $row/8 \times col/8$ 个小块，遍历每个小块，将小块中的像素值保存到 block 和 block_h 数组中，并使用 entropy 函数算出每个 8×8 小块的熵值，保存至 e 和 e_h 数组中。

- 1、对熵值矩阵进行降维，然后降序排序

```
e=e(:);
e_h=e_h(:);
[e,I]=sort(e,'descend');
[e_h,I_h]=sort(e_h,'descend');
```

```
>> length(e)          I =
ans =                  722
                2022
                723
                838
                1270
```

先将熵值矩阵 e 和 e_h 转成一维数组，由于我们这里采用的图片是 400×400 像素，

所以被分成了50行50列的小块，转成一维矩阵后就有2500行*1列。然后使用sort：
‘descend’ 将整个矩阵降序排列。再将排序前各行对应的索引存储到I数组中。
如图，表示的是第722个小块的熵值最高，其对应的是原图中第14行22列的小块。

2、利用图像降级的方法，将水印图像中熵值高的图像块隐藏在原图像熵值高的图像块中。

```
for i=1:length(I)
    r=mod(I(i),h); % 求原图像块的列坐标
    if r==0
        r=h;
    end
    r_h=mod(I_h(i),h_h); % 求水印图像块的列坐标
    if r_h==0
        r_h=h_h;
    end
    c=floor(I(i)/h)+1; % 求原图像块的行坐标
    if c==h+1
        c=h;
    end
    c_h=floor(I_h(i)/h_h)+1; % 求水印图像块的行坐标
    if c_h==h_h+1
        c_h=h_h;
    end
    block=img((r-1)*8+1:r*8,(c-1)*8+1:c*8,:);
    block_h=img_hidden((r_h-1)*8+1:r_h*8,(c_h-1)*8+1:c_h*8,:);
    [x,y]=size(block);
    img_t=block;
    for a=1:x
        for b=1:y
            tmp=dec2bin(block(a,b),8);
            tmp_h=dec2bin(block_h(a,b),8);
            tmp(5:8)=tmp_h(1:4);
            img_t(a,b)=bin2dec(tmp);
        end
    end
    img_f((r-1)*8+1:r*8,(c-1)*8+1:c*8)=img_t; % 覆盖图像块
end
```

由于I和I_h数组中对应的原图像和水印图像中每个块的熵值从大到小降序的排列对应的一维索引值，我们直接遍历I和I_h数组，将一维数组除以每一行的块的数量就可以求出块的横纵坐标，通过`block=img((r-1)*8+1:r*8,(c-1)*8+1:c*8,:)`可以

锁定该位置的8*8小方块像素格子，在这里每一行的块数为col_h/8=h_h(在这里为400/8=50)，对h_h取余即得到方块的列号，除数加1即得放开的行号。

将水印图像按照熵值从大到小的顺序嵌入到原图像熵值对应高的分块中。在把水印嵌入到对应的分块的时候，遍历分块 block 的每一个像素值，先将 10 进制转成 8 位的 2 进制，再把水印图像像素的前 4 位替换原图像的后 4 位，然后覆盖原图像该块，直到 I 数组遍历完毕完成所有水印的嵌入。这样我们就实现了根据图像和嵌入的水印图片按照熵值从高到低自主选择嵌入位置，实现了该数字图像水印算法的自适应特点。

3、保存图像块替换的顺序

由于我们在前面把图像块替换的顺序存储到了 I 和 I_h 数组，我们需要把他们存放到文件中，便于提取水印的时候按照顺序提取信息。

```
imwrite(img_f,'result.bmp');  
%保存图像块替换的顺序  
save('I.mat','I');  
save('I_h.mat','I_h');
```

4、显示结果

```
figure;  
subplot(1,3,1);imshow('source.bmp');title('原始图像');  
subplot(1,3,2);imshow('dingjun.bmp');title('水印图像');  
subplot(1,3,3);imshow('result.bmp');title('嵌入水印图像');
```



提取水印算法流程:

首先读取嵌入水印的原始图像和图像块替换的索引值数据，按照索引顺序数据对图像进行分块，根据原始图像和水印图像替换的对应关系，将后 4 位作为提取图像的高 4 位，后面补 0，最后生成提取图像。

1、读取图像块替换顺序数据

```
clear;
%读取图像
img=imread('result_yasuo0.8.bmp');
[row,col]=size(img);
w=row/8; %分成了多少行
h=col/8; % 分成多少列
img_f=img;
I=load('I.mat','T');
I=I.I;
I_h=load('I_h.mat','I_h');
I_h=I_h.I_h;
```

2、遍历I和I_h数组，将原图像与水印图像替换对应的块进行逆变换

读取嵌入水印图像的图像块 block，遍历 block 中每一个像素值，将其转成 8 位二进制，将后 4 位作为提取图像的高 4 位，后面补 0，替换图像块，最后生成提取图像。相当于嵌入水印的逆过程。

```
for i=1:length(I)
    r=mod(I(i),h); % 求原图像块的行
    if r==0
        r=h;
    end
    r_h=mod(I_h(i),h); % 求水印图像块的行
    if r_h==0
        r_h=h;
    end
    c=floor(I(i)/h)+1; % 求原图像块的列
    if c==h+1
        c=h;
    end
    c_h=floor(I_h(i)/h)+1; % 求水印图像块的列
    if c_h==h+1
        c_h=h;
    end
    block=img((r-1)*8+1:r*8,(c-1)*8+1:c*8,:); % 取图像块    [x,y]=size(block);
```

```

img_t=block;
[x,y]=size(block);
for a=1:x
    for b=1:y
        tmp=dec2bin(block(a,b),8);
        tmp(1:4)=tmp(5:8);
        tmp(5:8)=[ '0','0','0','0']; % 补0
        img_t(a,b)=bin2dec(tmp);
    end
end
img_f((r_h-1)*8+1:r_h*8,(c_h-1)*8+1:c_h*8)=img_t;% 替换图像块
end

```

3、生成图像

```

imwrite(img_f,'extract.bmp')
figure;
subplot(1,3,1);imshow('result.bmp');title('嵌入水印图像');
subplot(1,3,2);imshow('dingjun.bmp');title('水印图像');
subplot(1,3,3);imshow('extract.bmp');title('提取水印图像');

```



第三章 对数字图像水印算法的攻击测试

3.1 抗攻击测试

1、旋转攻击

在本程序中，对嵌入水印的载体图像旋转不同的度数，然后再使用水印提取程序对嵌入水印的载体图像进行提取和比较结果和影响。

```
P1=imread('result.bmp');
P1=imrotate(P1,90);% 逆时针旋转90度
imwrite(P1,'result_xuanzhuan.bmp');
figure;
subplot(1,2,1);imshow('result.bmp');title('原始嵌入水印的图像');
subplot(1,2,2);imshow('result_xuanzhuan.bmp');title('旋转后嵌入水印的图像');
```



经过多次旋转攻击测试，得出下列表格：

旋转攻击参数 (逆时针)	提取水印	旋转攻击参数 (逆时针)	提取水印
5 度	<div>提取水印图像</div>	90 度	<div>提取水印图像</div>

30 度		180 度	
------	---	-------	---

从结果表格可以看出，该水印算法对压缩攻击几乎没有抵抗能力，原因可能是由于图像旋转后影响了图片的分块以及信息位的分布，导致读取和替换图像块的时候发生混乱，故造成提取水印信息出现无规则的乱象。

2、压缩攻击

在本程序中，对嵌入水印的载体图像进行压缩，压缩为原来的 0.8 倍，然后运行水印提取算法程序进行数字图像水印的提取，进行结果对比。



但是我们仔细考虑会发现，由于初始时记录原图像和水印图像对应替换的分块是 50×50 (就算这里图片分辨率变了也会产生这种错误) 的，而压缩攻击相当于把整个图像等比例缩放，但是我们记录的是在原图比例下的 8×8 小方块，一旦进行压缩，也会与原先的分块函数操作产生冲突且报错。同时，图片压缩后在空间域上也会造成一些信息的流失。而且这样做没有意义，我

们要求对应替换块的行和列号，就必须得对 40 求余(压缩 0.8 倍后图像变成 320*320)，这样还是不能对应上原来的替换块，就使得对应分块的替换混乱。说明此替换块的数字图像水印算法不能抵抗压缩攻击。

```
>> fir
>> fir2
位置 2 处的索引超出数组边界(不能超出 320)。

出错 fir2 (line 30)
    block=img((r-1)*8+1:r*8,(c-1)*8+1:c*8,:);% 取图像块
    [x,y]=size(block);
```

而且由于索引矩阵 I 和 I_h 是对应于原图像的大小降维后的一维矩阵，此时若继续读取会超出索引范围，故该算法不能抵抗压缩攻击。

3、剪切攻击

在本程序中，对嵌入水印的载体图像进行不同位置的剪切，将剪切的位置变为纯白，相当于直接在水印载体图像上抠去一些图片的部分或角落，然后运行水印提取程序进行结果和分析。这里我们用灰度图像进行测试，可以减少其他格式对剪切的干扰。

% 剪切攻击

```
P3=imread('result.bmp');
P3(1:128,1:128)=256;% 剪切
imwrite(P3,'result_jianqie.bmp');
figure;
subplot(1,2,1);imshow('result.bmp');title('原载体图像');
subplot(1,2,2);imshow('result_jianqie.bmp');title('剪切后的载体图像');
对水印载体图像进行不同位置的剪切攻击，然后进行水印提取的结果如下：
```

剪切攻击水印载体图像	提取水印	剪切攻击水印载体图像	提取水印
			
			
			

从结果可以看出，该水印算法对剪切攻击具有较强的抵抗能力，说明我们在本实验中根据图像熵的大小进行水印的嵌入信息还是比较均匀的，当最后剪切掉大部分图片的时候，这时候信息位大量丢失，仍然可以看出大致的水印图片轮廓。从结果表也可看出，该水印的信息大量集中在载体图像的中下部分。




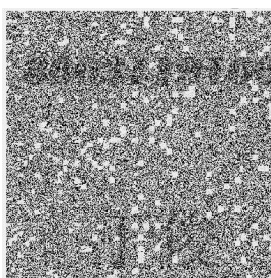

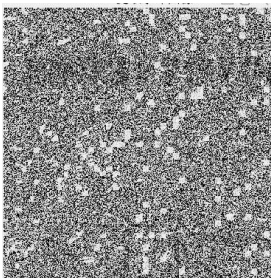

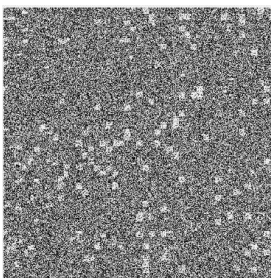
4、高斯噪声攻击

在本程序中，对嵌入水印的载体图像进行不同参数的高斯噪声攻击，然后运行水印提取程序进行水印的提取和影响的对比。(m 均值，var 方差)

```
P5=imread('result.bmp');
P5=imnoise(P5,'gaussian',0.0001,0.0001);% 给图像加高斯噪声
imwrite(P5,'noise0.0001.bmp');
figure;
subplot(1,2,1);imshow('result.bmp');title('原载体图像');
subplot(1,2,2);imshow('noise0.0001.bmp');title('加入噪声后的载体图像');
```

经过多次实验发现，该算法不能抵抗高斯噪声攻击，在 m 和 var 分别取万分

之一的时候提取的水印才勉强不受影响。因为噪声攻击主要攻击的是图像的最低位，而我们恰好是把水印图像的信息隐藏在了载体图像的后 4 位，受到的影响较大，只要高斯攻击的均值和方差稍微调大一点，水印信息就遭到严重破坏。但是当高斯造成攻击取得 m 个 var 不大时大致上还是能看出水印信息的轮廓。

高斯噪声攻击水印图像	提取水印信息	高斯噪声攻击水印图像	提取水印信息
 m=var=0.0001	 8003119100 丁俊	 m=var=0.00005	 8003119100 丁俊
 m=var=0.0001	 8003119100 丁俊	 m=var=0.001	 8003119100 丁俊

5、椒盐噪声攻击

在本程序中，对嵌入水印的载体图像进行椒盐噪声攻击，攻击的椒盐噪声密度分别取 0.1、0.3、0.6 进行攻击测试，然后再提取出相应的水印图像信息。

```
P = imread('result.bmp');  
P1 = imnoise(P, 'salt & pepper', 0.1);  
P2 = imnoise(P, 'salt & pepper', 0.3);  
P3 = imnoise(P, 'salt & pepper', 0.8);
```

```
imwrite(P1, 'jiaoyan1.bmp');
imwrite(P2, 'jiaoyan2.bmp');
imwrite(P3, 'jiaoyan3.bmp');
```

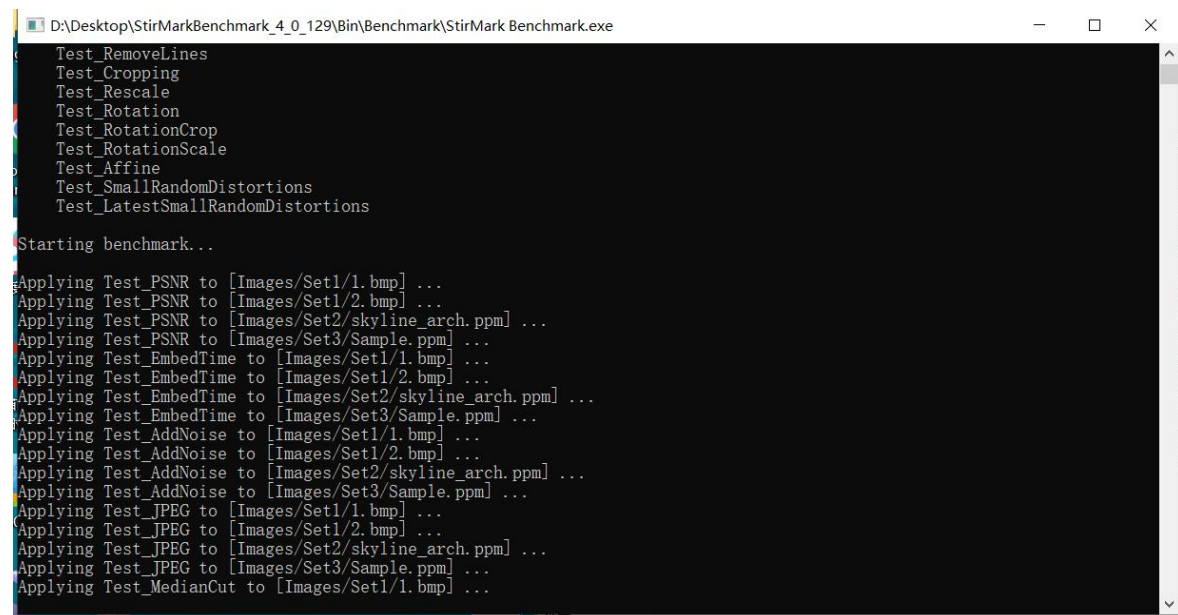
椒盐噪声攻击水印图像			
提取水印信息	<div>提取水印图像</div> 	<div>提取水印图像</div> 	<div>提取水印图像</div> 

从以上几种不同噪声密度的椒盐噪声可以看出，该水印算法可以抵抗椒盐噪声攻击。椒盐噪声是数字图像的一个常见噪声，所谓椒盐，椒就是黑，盐就算白，椒盐噪声就是在图像上随机出现黑色白色的像素。由于我们的水印图像信息是隐藏在低 4 位的，当椒盐攻击的噪声密度取较小时，还不足以对这些信息位产生影响，所以依然能够很清晰地分辨出水印图像；当噪声密度取到了 0.3 及以上，信息 bit 位就受到了严重影响，而在这种情况下依然是可以分辨出水印内容的轮廓，可见该水印算法对椒盐噪声攻击还是有一定抵抗能力的。

3.2 鲁棒性攻击与评估

上面我们对水印算法的抗攻击能力进行了简要攻击测试和分析。下面我们继续对水印的鲁棒性进行评估，使用 **stir mark** 软件进行鲁棒攻击测试。继续选择部分以上攻击方式：旋转、剪切、高斯噪声攻击、椒盐噪声攻击、扩展攻击。

启动 **StirMark** 应用程序攻击：

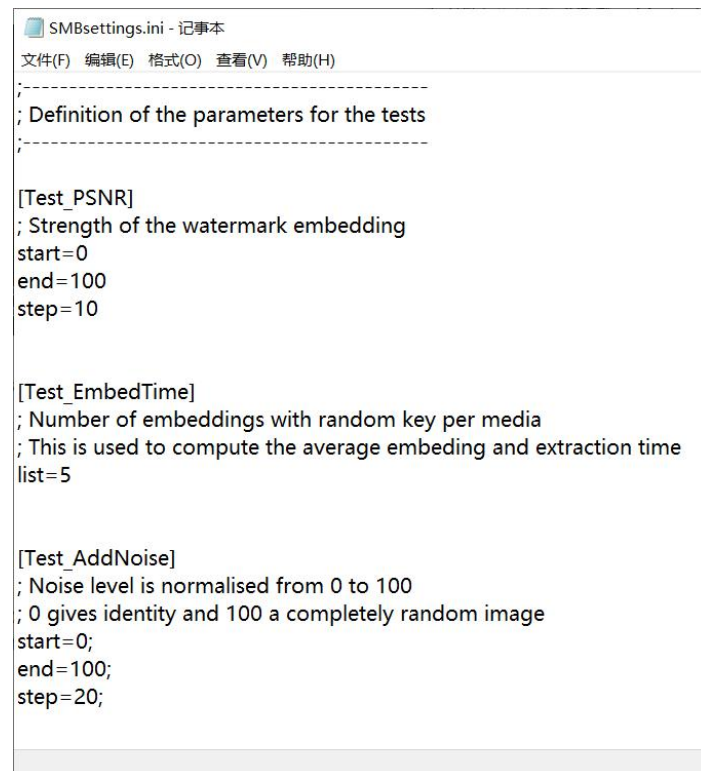


```
Test_RemoveLines
Test_Cropping
Test_Rescale
Test_Rotation
Test_RotationCrop
Test_RotationScale
Test_Affine
Test_SmallRandomDistortions
Test_LatestSmallRandomDistortions

Starting benchmark...

Applying Test_PSNR to [Images/Set1/1.bmp] ...
Applying Test_PSNR to [Images/Set1/2.bmp] ...
Applying Test_PSNR to [Images/Set2/skyline_arch.ppm] ...
Applying Test_PSNR to [Images/Set3/Sample.ppm] ...
Applying Test_EmbedTime to [Images/Set1/1.bmp] ...
Applying Test_EmbedTime to [Images/Set1/2.bmp] ...
Applying Test_EmbedTime to [Images/Set2/skyline_arch.ppm] ...
Applying Test_EmbedTime to [Images/Set3/Sample.ppm] ...
Applying Test_AddNoise to [Images/Set1/1.bmp] ...
Applying Test_AddNoise to [Images/Set1/2.bmp] ...
Applying Test_AddNoise to [Images/Set2/skyline_arch.ppm] ...
Applying Test_AddNoise to [Images/Set3/Sample.ppm] ...
Applying Test_JPEG to [Images/Set1/1.bmp] ...
Applying Test_JPEG to [Images/Set1/2.bmp] ...
Applying Test_JPEG to [Images/Set2/skyline_arch.ppm] ...
Applying Test_JPEG to [Images/Set3/Sample.ppm] ...
Applying Test_MedianCut to [Images/Set1/1.bmp] ...
```

部分攻击参数：



```
SMBsettings.ini - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

;-----
; Definition of the parameters for the tests
;-----

[Test_PSNR]
; Strength of the watermark embedding
start=0
end=100
step=10

[Test_EmbedTime]
; Number of embeddings with random key per media
; This is used to compute the average embedding and extraction time
list=5

[Test_AddNoise]
; Noise level is normalised from 0 to 100
; 0 gives identity and 100 a completely random image
start=0;
end=100;
step=20;
```

攻击类型	攻击后的图像	提取水印信息	相似性 (NC)
旋 转 攻 击 (ROT)	 P1_ROT30.bmp		0.2125
剪切攻击	 P1_jianqie1.bmp		0.7713
高斯噪声攻击 (noise)	 noise0.0001.bmp		0.4145
椒盐噪声攻击 (noiseSNR)	 P1_SNR0.3.bmp		0.5644
扩 展 攻 击 (CROP)	 CROP.bmp		0.1145

从以上可以看出，该水印算法的鲁棒性还是较好的，对于剪切、高斯噪声和椒盐噪声攻击下水印依然可以提取出大概的轮廓，而对于扩展攻击和旋转攻击则完全不能分辨出原有水印的内容。

3.3 水印的不可见性

对于该水印嵌入算法，我们通过对比原始图像与嵌入水印后的图像发现，嵌入水印图像相比模糊了一些，可能是由于我们直接是把水印替换和嵌入到原始图像的信息位，造成了原始图像的部分失真，说明该水印算法的不可见不是很高，肉眼是可以察觉出来存在水印的，这也是一个需要改进的缺点。



四、总结分析

本次实验，使用基于图像熵自适应水印算法来进行水印嵌入，首先将图像根据像素大小分成 8×8 的分块，计算每个分块熵值的大小并降序排列，按照熵值从高到低，将水印图像熵值高的分块中的信息嵌入到载体图像熵值高的分块中，替换信息位和图像块，将图片纹理特征作为自适应算法的判定依据。并对水印图像进行了多种攻击测试，评估了水印算法的鲁棒性、不可见性和抗攻击能力。最终得出该水印算法的整体鲁棒性良好，抗攻击能力一般，但是不可见性较差，很容易被察觉出被嵌入了水印，这也是接下来需要改进的地方。通过本次大作业让我对数字水印算法的鲁棒性和抵抗攻击的能力的评估方法了解得更深入。

参考文献

- [1] 宫海梅. 图像隐藏算法的 MATLAB 实现 [J]. 智能计算机与应用, 2016, 6(04): 60-62.
- [2] 龙香玉. 基于图像纹理的数字水印算法研究[D]. 江西理工大学, 2019.
- [3] 夏嵬, 赵小厦. 一种参数自适应的鲁棒性数字图像水印算法[J]. 信息与电脑(理论版), 2021, 33(24): 70-72.
- [4] 李玥. 图像块自适应均衡水印算法[D]. 辽宁工程技术大学, 2018.