



南開大學  
Nankai University

南 开 大 学

网络空间安全学院

信息隐藏技术实验报告

---

## 实验 8：二值图像隐藏法

---

姓名：2113203 付政烨

年级：2021 级

专业：信息安全、法学

指导教师：李朝晖

2024 年 4 月 23 日

# 目录

<b>一、 实验内容</b>	<b>1</b>
(一) 实验目的 . . . . .	1
(二) 实验环境 . . . . .	1
(三) 实验要求 . . . . .	1
<b>二、 实验原理</b>	<b>1</b>
(一) 二值图像隐藏法 . . . . .	1
(二) 简易算法概述 . . . . .	1
(三) 嵌入过程：遍历原图的每个 1x4 的矩形区域 . . . . .	2
(四) 提取过程：遍历原图的每个 1x4 的矩形区域 . . . . .	2
<b>三、 实验步骤</b>	<b>2</b>
(一) 黑色像素数量计算 . . . . .	2
(二) 嵌入秘密信息 . . . . .	3
(三) 提取秘密信息 . . . . .	6
<b>四、 扩展实验——多层水印嵌入与提取</b>	<b>7</b>
(一) 原理介绍 . . . . .	7
(二) 代码实现 . . . . .	9
(三) 实验结果 . . . . .	10
<b>五、 实验心得体会</b>	<b>11</b>

## 一、实验内容

### (一) 实验目的

1. **隐藏**：利用二值图像隐藏法，实现将秘密信息（可以是图像、文字等信息）嵌入到位图中；
2. **提取**：将秘密信息提取出来。

### (二) 实验环境

- 运行系统：Windows11
- 实验工具：Matlab2022a
- 数据：PNG 格式图像

### (三) 实验要求

- 在 MATLAB 中调试完成
- 编写实验代码和报告，并给出截图
- QQ 群提交作业

## 二、实验原理

### (一) 二值图像隐藏法

- **二值图像**：黑白两种像素组成的图像，1bit 表示存储像素颜色 (0 或 1)。
- **通常方法**：利用图像区域中黑色像素个数相对于区域中全部像素个数的百分比来对秘密信息进行编码。

### (二) 简易算法概述

将原图划分为  $1 \times 4$  的矩阵像素块，每个区域有连续四个像素点。像素点取值情况共有 5 类：全白，1 个黑像素点，2 个黑像素点，3 个黑像素点和全黑。

黑像素个数	0	1	2	3	4
像素分布	全白	1 黑 3 白	两黑两白	3 黑 1 白	全黑
含义	无效块	隐藏“1”	不能出现	隐藏“0”	无效块

当隐藏文本文档中的字符串时需要注意：

1. 嵌入信息的长度不可以过大，不能超过图像大小能负担的度量
2. 为了简化过程，可规定接收者已知秘密信息的长度

### (三) 嵌入过程：遍历原图的每个 1x4 的矩形区域

当秘密信息为 0，需要将当前区域黑像素点数量调整为 3 个。当黑像素点为 3 时，不需修改。当黑像素点为 1、2、4 时，需要进行修改。对原始的黑像素直接利用，位置不做修改，在嵌入秘密信息时减少对图片的修改。当黑像素点为 0 时，舍弃该预取不做修改，否则在直观视觉上可能被感受到。

当秘密信息为 1，需要将当前区域黑像素点数量调整为 1 个。当黑像素点为 1 时，不需修改。当黑像素点为 0、2、3 时，需要进行修改。对原始的黑像素直接利用，多余的翻转为白像素，在嵌入秘密信息时减少对图片的修改。当黑像素点为 4 时，舍弃该预取不做修改，否则在直观视觉上可能被感受到。

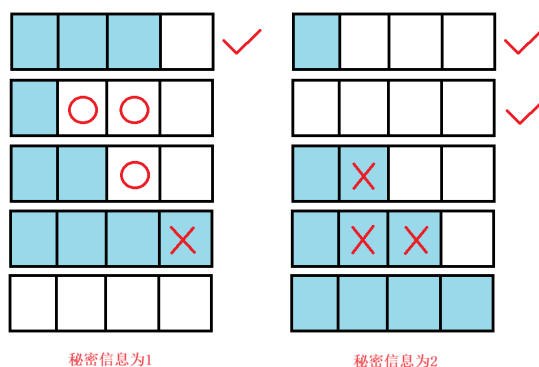


图 1

### (四) 提取过程：遍历原图的每个 1x4 的矩形区域

- 嵌入信息的图像的每个区域的黑色像素只有 4 个取值：0, 1, 3, 4
- 黑像素个数为 1 或 3 时，提取信息 1 或 0。黑像素个数为 0 或 4 时，不提取信息

## 三、实验步骤

### (一) 黑色像素数量计算

设  $x$  为二值图像数组，其中  $x[a]$  的值为 0 表示像素是黑色，为 1 表示像素是白色。函数 ‘CalculateBlack’ 的目的是计算从位置  $i$  开始的连续四个像素中黑色像素的数量。数学表达如下：

$$\text{out} = \sum_{a=i}^{i+3} (1 - x[a])$$

这里， $1 - x[a]$  的操作用于将白色像素（值为 1）转换为 0，黑色像素（值为 0）转换为 1，从而使得当  $x[a]$  为黑色（0）时，结果为 1，为白色（1）时，结果为 0。随后，通过求和来计算黑色像素的总数。

本次实验代码中的 CalculateBlack 函数的功能便是计算给定图像  $x$  中从位置  $i$  开始连续四个像素中黑色像素的数量。函数接收两个参数： $x$ ，表示图像数据，通常是一个二值化后的数组，其中像素值为 0 代表黑色，像素值为 1 代表白色； $i$  是起始检测位置的索引。

```
1 function out = CalculateBlack(x, i)
2 out = 0;
```

```
3 for a = i : i + 3
4     if x(a) == 0
5         out = out + 1;
6     end
7 end
8 end
```

函数开始时，输出变量 out 被初始化为 0。这个变量用于累计从位置 i 开始的四个像素中黑色像素的数量。使用 for 循环从位置 i 遍历到 i + 3，即连续四个像素。在循环内部，使用 if 语句检查每个像素值。如果像素值为 0（表示黑色），则将 out 的值增加 1。循环结束后，out 变量包含了检查区域内黑色像素的总数，这个值随后被返回。

## （二） 嵌入秘密信息

### （1） 图像读取和二值化

首先使用 MATLAB 的 imread 函数读取位图文件 (.bmp)，然后通过 imbinarize 函数将图像转换为二值图像，即图像中的像素点只有黑白两种可能的状态。最后，使用 imwrite 函数保存转换后的二值图像。

```
1 clc;
2 clear;
3 d = imread('pic.bmp');
4 d = imbinarize(d);
5 imwrite(d, 'black1.bmp', 'bmp')
6 subplot (1, 2, 1); imshow (d, []); title ('原始图片');
```

### （2） 秘密信息的比特提取

代码定义了一个名为 secret 的变量，它是一个整数，代表需要隐藏在图像中的秘密信息。然后通过一个循环和 bitget 函数提取 secret 变量的每一位，将这些比特存储在数组 s 中。

主循环遍历 secret 的每个比特。在每次循环中，代码根据当前比特的值（0 或 1）以及由 CalculateBlack 函数返回的结果来决定如何修改图像。CalculateBlack 函数似乎用于计算图像中某个区域的黑色像素数量。根据黑色像素的数量，代码在相应区域进行修改，以隐藏比特信息。

之所以选择循环 24 次，是因为这与如何表示和处理秘密信息的方式紧密相关。整数 secret 在二进制形式下可能不需要 24 位来完整表示，**选择固定使用 24 位，以确保无论 secret 的实际大小如何，处理都是一致和完整的。**这种做法确保图像足够大以容纳 24 个不同的比特，这样可以在不同区域隐藏更多信息而不明显影响图像质量。此外，固定循环 24 次也简化了编程和处理逻辑，使得每个比特的处理可以标准化，从而提高代码的通用性和可重用性。

```
1 secret = 2113203;
2 for t = 1:24
3     s(t) = bitget(secret, t);
4 end
5 num = 1;
6 t = 1;
7 while t < 24
8     if s(t) == 0
9         % 依据 CalculateBlack 函数的结果，修改图像...
```

```
10     else
11         a = CalculateBlack(d, num)
12         % 依据 CalculateBlack 函数的结果, 修改图像...
13     end
14     t = t + 1;
15 end
```

### (3) 像素调整

switch 语句被用来根据不同的黑色像素数量来决定如何修改图像中的像素。switch 语句根据 CalculateBlack 函数的返回值来执行不同的代码块。每个 case 代表 CalculateBlack 返回的黑色像素的数量, 而每个 case 中的操作则根据当前嵌入的比特值来调整像素, 具体来说:

```
1 % 当比特为 0 时的处理逻辑
2 switch (CalculateBlack(d, num))
3     case 0
4         t = t - 1;
5         num = num + 4;
6     case 1
7         temp = 1;
8         startnum = num;
9         while temp < 3
10             if d(startnum) == 1
11                 d(startnum) = 0;
12                 temp = temp + 1;
13                 startnum = startnum + 1;
14             end
15         end
16         num = num + 4;
17
18     case 2
19         temp = 2;
20         startnum = num;
21         while temp < 3
22             if d(startnum) == 1
23                 d(startnum) = 0;
24                 temp = temp + 1;
25                 startnum = startnum + 1;
26             end
27         end
28         num = num + 4;
29
30     case 3
31         num = num + 4;
32
33     case 4
34         temp = 4;
35         startnum = num;
36         while temp > 3
```

```

37         if d(startnum) == 0
38             d(startnum) = 1;
39             temp = temp - 1;
40             startnum = startnum + 1;
41         end
42     end
43     num = num + 4;
44 end

```

#### 当比特为 0 时的处理逻辑:

- case 0: 如果当前区域中没有黑色像素，并且当前比特是 0，如果连续处理失败（连续没有黑色像素），将指针向后移动四个单位，然后重新检测。
- case 1 和 case 2: 当区域中有 1 个或 2 个黑色像素时，代码将尝试减少黑色像素的数量以匹配当前的比特值（期望更多的白色像素）。具体做法是将一些黑色像素（值为 1）转换为白色（值为 0），直到黑色像素的数量减少到适当的水平（小于 3 个）。
- case 3: 如果黑色像素正好是 3 个，这种情况不需要做任何修改，直接向后移动四个单位。
- case 4: 当区域有 4 个黑色像素时，需要将一个黑色像素（值为 1）改为白色（值为 0）以减少黑色像素的数量，因为目标是使得黑色像素数量少于 3。

#### 当比特为 1 时的处理逻辑

- case 0: 当没有黑色像素时，为了将当前比特表示为 1，需要增加黑色像素的数量。通过将一些白色像素（值为 0）转变为黑色（值为 1），直到黑色像素数量增加到 2 个。
- case 1: 当黑色像素数量为 1 时，不需要做修改，因为这已经与表示比特 1 的状态足够接近。
- case 2: 当有 2 个黑色像素时，继续保持状态不变，因为它符合表示比特 1 的要求。
- case 3 和 case 4: 如果黑色像素数量超过 2 个，需要通过增加白色像素来减少黑色像素数量，直到黑色像素数量降到 2 个以下（详见源码）。

### (4) 水印图片生成

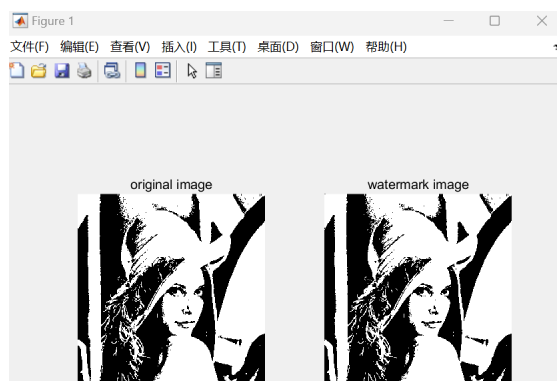


图 2: 信息隐藏结果

循环继续执行直到处理完 secret 中的所有比特。修改后的图像通过 imwrite 再次保存，并通过 subplot 和 imshow 显示处理后的图像，即添加了水印的图像。

整个过程结束后，最终的图像显示出来（如图2所示），左侧为原始二值化图像，右侧为加入了隐藏信息后的图像。这展示了如何通过对二值图像的像素修改来隐藏秘密信息，形成了一种基本的数字水印技术。

```
1 imwrite(d, 'black2.bmp', 'bmp')
2 subplot (1, 2, 2); imshow (d, []); title (' Watermark ');
```

### （三） 提取秘密信息

#### （1） 图像载入与初始化比特数组

使用 imread 函数读取图像。这是一个已经嵌入了秘密信息的二值化图像。接下来，代码为存储解码后的信息初始化一个长度为 24 的数组 s。数组的每个元素都被初始化为 0，准备存放从图像中解码的比特值。

```
1 clc;
2 clear;
3 d = imread('black2.bmp');
4 for t = 1:24
5     s(t) = bitget(0, t);
6 end
```

#### （2） 图像遍历和解码

代码设置一个 while 循环，循环 24 次，与之前的嵌入过程相对应。在循环中，每次通过调用 CalculateBlack 函数计算从当前位置 num 开始的四个像素中黑色像素的数量。

```
1 t = 1;
2 num = 1;
3
4 while t < 24
5     a = CalculateBlack(d, num);
6     ...
7     t = t + 1;
8 end
```

#### （3） 根据黑色像素数解码比特

使用 switch 语句根据 CalculateBlack 函数的结果来确定每个比特的值：

```
1 switch a
2     case 0
3         num = num + 4;
4     case 1
5         s(t) = 1;
6         t = t + 1;
7         num = num + 4;
8     case 3
9         s(t) = 0;
```



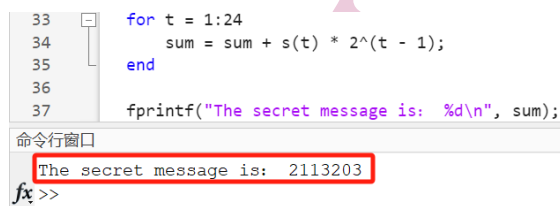
```

10     t = t + 1;
11     num = num + 4;
12     case 4
13         num = num + 4;
14 end

```

- **case 0 和 case 4:** 当检测到 0 个或 4 个黑色像素时，不修改比特数组，只是将索引向前移动 4 个单位，因为这两种情况在信息编码时没有用于存储有效的比特信息。
- **case 1:** 如果有 1 个黑色像素，这意味着在该位置编码的比特为 1。设置对应的比特值为 1，并将计数器 t 和索引 num 同时增加。
- **case 3:** 如果有 3 个黑色像素，这表明在该位置编码的比特为 0。设置对应的比特值为 0，并适当增加计数器和索引。

#### (4) 计算显示解码后的整数



```

33 for t = 1:24
34     sum = sum + s(t) * 2^(t - 1);
35 end
36
37 fprintf("The secret message is: %d\n", sum);

```

命令行窗口

```

The secret message is: 2113203
fx >>

```

图 3: 提取秘密信息结果

在完成所有 24 次循环后，代码通过一个新的循环将二进制比特数组 s 转换为一个整数 sum。这是通过遍历 s 数组，并将每个比特值乘以其相应的 2 的幂次求和来实现的。最后，使用 fprintf 函数输出解码得到的整数 sum，这个值代表从图像中提取的秘密信息（如图3所示）。

```

1 sum = 0;
2 for t = 1:24
3     sum = sum + s(t) * 2^(t - 1);
4 end
5 fprintf("秘密信息是: %d\n", sum);

```

## 四、 扩展实验——多层水印嵌入与提取

### (一) 原理介绍

在二值图像中嵌入多层水印能够在单一图像中存储多种不同的信息，使得每一层的信息能够独立于其他层被嵌入和提取。多层水印技术在需要同时传递多种信息的场景中特别有用，例如在版权保护中，可能需要同时嵌入版权者的信息、版权开始日期以及其他相关的法律信息。每一层的信息可以根据需要独立访问和验证，这增加了处理的灵活性和应用的广泛性。

再具体实现上，首先图像被读取并转换成二值化格式，其中黑色像素代表 0，白色像素代表 1。这种格式简化了后续的水印嵌入处理，因为每个像素只有两种状态。每层水印信息（如版权信息、时间戳）首先被转换成二进制格式。可以通过 bitget 函数实现，该函数从数字中提取指定位的状态。

每一层水印选择不同的起始点和区域进行信息嵌入，这避免了不同水印层之间的直接干扰。对于每个二进制位，根据其值（0 或 1），相应地修改图像中的像素。例如，如果需要嵌入的位是 1，代码尝试将选定的像素设置为白色（代表 1）。如果是 0，则将其设置为黑色（代表 0）。水印的处理是独立进行的，虽然它们可能共用同一张图像，但通过选择不同的嵌入区域和策略，每层可以独立存储信息。

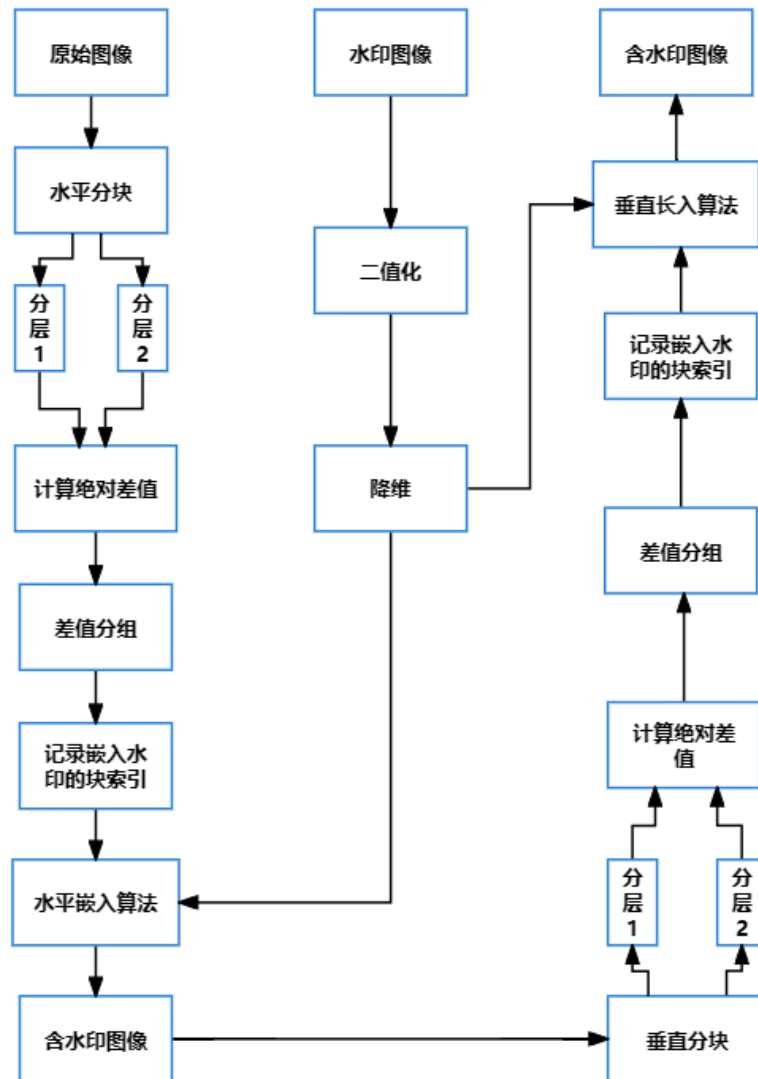


图 4: 多层水印嵌入流程图

### 与单层水印嵌入的不同之处

1. **区域分隔**: 在多层水印中，不同信息层通常会在图像中占据不同的区域或使用不同的嵌入模式，这样做可以减少层间的干扰和数据损失。
2. **数据容量**: 多层水印嵌入可以显著增加图像的数据容量，因为它允许在同一图像中存储多个独立的信息流。
3. **复杂性和安全性**: 多层水印由于其结构的复杂性，提供了更高的安全性和更强的防篡改能力。如果不了解所有层的详细嵌入策略，恢复原始信息会更加困难。

多层水印提取的原理基于对二值图像中特定位置像素的检查, 通过计算这些位置的黑色像素数量来确定在该位置隐藏的信息是 0 还是 1。这种方法允许从同一图像中提取多层不同的秘密信息。该过程首先通过读取一个已嵌入水印的图像文件开始。对于每层水印, 设定一个起始像素位置, 并逐个分析指定像素区域, 使用 CalculateBlack 函数计算该区域内黑色像素的数量。根据黑色像素的数量, 通过预设的逻辑(如 3 个黑色像素代表 0, 其他情况代表 1) 解码出二进制比特值。这些二进制比特值随后被转换为十进制数, 以显示嵌入的秘密信息。整个过程允许从同一图像文件中独立提取多种不同类型的信息, 如版权数据、时间戳等。

## (二) 代码实现

### (1) 第一层水印：版权信息的嵌入

```
1 secret1 = 123;           % 版权信息的数字表示
2 s1 = bitget(secret1, 1:24); % 提取版权信息的24个比特
3 num = 1;                 % 开始位置
4 for t = 1:24
5     if s1(t) == 0
6         % 嵌入逻辑, 调整像素来匹配0
7     else
8         % 嵌入逻辑, 调整像素来匹配1
9     end
10    num = num + 4; % 移动到下一个嵌入区域
11 end
```

此部分展示如何将数字表示的版权信息嵌入到图像中。首先, 定义版权信息的数字表示 secret1, 然后使用 bitget 函数从数字中提取出 24 个比特。接着, 定义一个变量 num 作为起始位置, 循环 24 次, 根据每个比特的值 (0 或 1), 调整图像的像素来匹配这些比特值。每次循环后, num 变量以 4 的步长移动, 指向下一个嵌入区域。

### (2) 第二层水印：时间戳的嵌入

```
1 secret2 = 456;           % 时间戳的数字表示
2 s2 = bitget(secret2, 1:24);
3 num = 1;                 % 可以选择不同的开始位置
4 for t = 1:24
5     if s2(t) == 0
6         % 嵌入逻辑, 调整像素来匹配0
7     else
8         % 嵌入逻辑, 调整像素来匹配1
9     end
10    num = num + 4; % 移动到下一个嵌入区域
11 end
```

与版权信息嵌入类似, 这一部分展示如何将数字表示的时间戳嵌入到图像中。过程与版权信息嵌入相同, 但可以选择不同的起始位置 num, 以避免两个水印之间的相互干扰。

### (3) 第一, 二层水印的解码

```

1 secret2 = 456;      % 时间戳的数字表示
2 s2 = bitget(secret2, 1:24);
3 num = 1;           % 可以选择不同的开始位置
4 for t = 1:24
5     if s2(t) == 0
6         % 嵌入逻辑, 调整像素来匹配0
7     else
8         % 嵌入逻辑, 调整像素来匹配1
9     end
10    num = num + 4; % 移动到下一个嵌入区域
11 end

```

此代码块用于解码图像中嵌入的版权信息。它使用一个循环, 根据预定义的逻辑 (例如假设 3 个黑色像素代表比特 0, 其他情况代表比特 1), 从图像中检索和解码嵌入的比特。使用 CalculateBlack 函数 (自定义) 来检测特定区域内的黑色像素数量, 然后将这些比特转换回十进制形式。时间戳的解码过程与版权信息解码相同, 但是从不同的位置开始解码, 这里不再赘述。

### (三) 实验结果

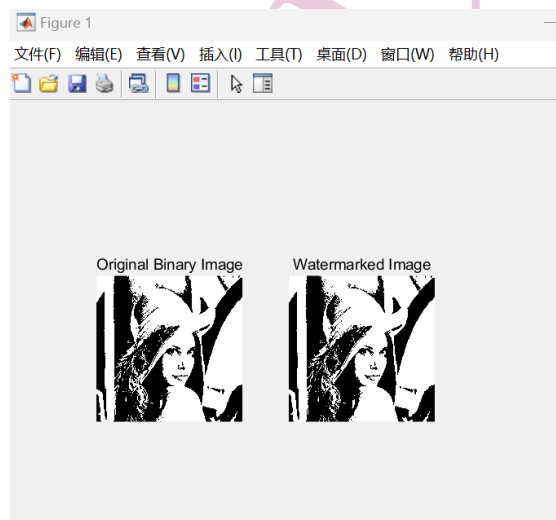


图 5: 多层水印嵌入

```

33 %% 显示解码的信息
34 fprintf("Decoded Copyright Information: %d\n", sum1);
35 fprintf("Decoded Timestamp Information: %d\n", sum2);
36
37 %% 自定义二进制到十进制转换函数
38 function decimal = oDecimal(binaryVector)
39     decimal = 0;

```

命令行窗口

```

Decoded Copyright Information: 1234
Decoded Timestamp Information: 4567

```

图 6: 多层水印提取

## 五、 实验心得体会

在这次的信息隐藏技术实验中，我深入学习并实践了二值图像隐藏法，尤其是对多层水印嵌入与提取技术的实现过程，给我留下了深刻的印象。实验不仅增强了我对信息隐藏基本原理的理解，也锻炼了我使用 MATLAB 进行图像处理技能。多层水印技术的探索尤为吸引我，因为它展示了在同一图像中嵌入和提取多种信息的可能性。通过将不同信息编码到图像的不同区域，实现了数据的层次性和复杂性管理。例如，版权信息和时间戳的独立嵌入和提取，不仅提升了信息安全，也增加了操作的灵活性。这种技术在现实中有广泛的应用，如在数字媒体版权管理、机密通讯等领域。

在实验过程中，我也体会到了精确控制像素修改对于保持图像质量的重要性。过度的或不恰当的像素调整可能会导致图像质量明显下降，这在实际应用中是不可接受的。因此，如何在不影响图像可观感的前提下有效隐藏信息，是设计高效信息隐藏算法的关键。此外，通过实验，我学到了如何使用 MATLAB 中的函数来处理和分析图像，这些技能对我未来在图像处理或数据分析的研究将大有帮助。实验不仅提升了我的编程能力，也激发了我对信息安全技术更深层次的探索兴趣。

MINI