

南开大学

信息隐藏技术 课程实验报告

二值图像隐藏法



学院 网络空间安全学院

专业 信息安全

姓名 齐明杰

学号 2113997

2024 年 4 月 23 日

目 录

1	实验目的	3
2	实验原理	3
2.1.1	二值图像隐藏法	3
2.1.2	简易算法概述	4
3	实验过程	4
3.1	实验准备	4
3.2	隐藏文本信息	5
3.3	提取文本信息	9
3.4	运行结果	10
4	实验心得	11

图表

图 2.1.1:	黑像素分布	4
图 3.1.2:	Lena 图像	5
图 3.4.3:	嵌入信息	11

1 实验目的

二值图像隐藏法

隐藏：利用二值图像隐藏法将秘密信息（可以是图像、文字等信息）嵌入到位图中；

提取：将秘密信息提取出来。

2 实验原理

本实验旨在实现基于二值图像的信息隐藏与提取技术。通过该方法，可以将文本或其他形式的秘密信息嵌入到二值图像中，并从修改后的图像中提取出原始信息。

2.1.1 二值图像隐藏法

二值图像隐藏法利用二值图像（只包含黑色和白色像素的图像）的像素分布来嵌入和提取信息。具体方法如下：

二值图像隐藏法

嵌入过程

1. 图像预处理：将原始图像划分为 1×4 的矩阵像素块，每个区域包含四个连续的像素点。这些像素点只能为黑色或白色。
2. 像素点分类：基于每个区域的黑色像素数量，像素块可以分为五类：全白（0 个黑点）、1 个黑点、2 个黑点、3 个黑点、全黑（4 个黑点）。
3. 信息编码：
 - 当需要隐藏的秘密信息为 ‘0’ 时，目标是将当前区域的黑像素点数量调整为 3 个。
 - 如果当前区域的黑像素点数量为 1、2 或 4，进行必要的像素点调整。
 - 如果当前区域为全黑（4 个黑点）或全白（0 个黑点），则跳过，避免在视觉上的显著变化。
 - 当需要隐藏的秘密信息为 ‘1’ 时，目标是将当前区域的黑像素点数量调整为 1 个。
 - 如果当前区域的黑像素点数量为 0、2 或 3，进行必要的像素点调整。
 - 如果当前区域为全黑（4 个黑点），则跳过，避免在视觉上的显著变化。

提取过程

1. 图像遍历：遍历原图的每个 1×4 的矩形区域。
2. 信息解码：
 - 只关注黑像素点的数量为 1 或 3 的区域。
 - 如果区域中的黑像素点数量为 1，提取的信息为 ‘1’。
 - 如果区域中的黑像素点数量为 3，提取的信息为 ‘0’。
 - 如果区域中的黑像素点数量为 0 或 4，该区域不包含有效的秘密信息。

通过上述方法，我们可以有效地利用二值图像进行信息的隐写与提取，同时确保图像的视觉变化最小化，增强隐藏信息的安全性。

2.1.2 简易算法概述



将原图划分为 1×4 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

图 2.1.1: 黑像素分布

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

嵌入过程：遍历原图的每个 1×4 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。
提取过程：遍历原图的每个 1×4 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。	

3 实验过程

3.1 实验准备

本次实验我仍然选用 Lena 图像作为我的底图：



图 3.1.2: Lena 图像

这个图像是灰度的，并不是二值图像，因此在代码需要转化为二值图像：

```
1 img = imbinarize(img);
```

[matlab](#)

本次我采用简单的**文本信息**进行嵌入: 学号: 2113997

先编写一个公共函数，便于隐藏信息和提取信息使用：

```
1 function blackCount = CalculateBlack(pixelBlock, startIdx)
2     blackCount = 0;
3     for idx = startIdx : startIdx + 3
4         if pixelBlock(idx) == 0
5             blackCount = blackCount + 1;
6         end
7     end
8 end
```

[matlab](#)

该函数接受两个参数：pixelBlock 和 startIdx。其中，pixelBlock 是包含图像数据的一维数组，而 startIdx 是数组中要检查的四像素序列的起始索引。函数的主体是一个循环，它遍历从 startIdx 到 startIdx + 3 的四个连续像素，检查每个像素点是否为黑色（在二值图像中用 0 表示）。

每当发现一个黑色像素，一个名为 blackCount 的计数器就会递增。这个计数器最初被设置为 0，并在检测到黑色像素时增加，最终结果反映了四像素块中黑色像素的总数。通过这种方法，我们能够准确地计算出图像中任何给定四像素块内的黑色像素数量，这对于后续的图像处理和信息编码步骤是必需的。

3.2 隐藏文本信息

编写如下代码，实现文本信息的隐藏：

```
1 %% encode
```

[matlab](#)

```
2   clc;
3   clear;
4   inputImage = imread('./lady.bmp');
5   binaryImage = imbinarize(inputImage);
6   imwrite(binaryImage, 'processedImage.bmp', 'bmp');
7   subplot(1, 2, 1);
8   imshow(binaryImage, []);
9   title('Original Image');
10  hiddenMessage = 2113997;
11
12  for bitIndex = 1:24
13      messageBits(bitIndex) = bitget(hiddenMessage, bitIndex);
14  end
15
16  pixelIndex = 1;
17  currentBit = 1;
18
19  while currentBit < 24
20
21      if messageBits(currentBit) == 0
22
23          blackPixels = CalculateBlack(binaryImage, pixelIndex);
24
25          switch blackPixels
26              case 0
27                  currentBit = currentBit - 1;
28                  pixelIndex = pixelIndex + 4;
29              case 1
30                  count = 1;
31                  adjustIndex = pixelIndex;
32
33                  while count < 3
34                      if binaryImage(adjustIndex) == 1
35                          binaryImage(adjustIndex) = 0;
36                          count = count + 1;
37                          adjustIndex = adjustIndex + 1;
38                      end
39                  end
40
41                  pixelIndex = pixelIndex + 4;
42              case 2
43                  count = 2;
44                  adjustIndex = pixelIndex;
45
```

```
46         while count < 3
47             if binaryImage(adjustIndex) == 1
48                 binaryImage(adjustIndex) = 0;
49                 count = count + 1;
50                 adjustIndex = adjustIndex + 1;
51             end
52         end
53
54         pixelIndex = pixelIndex + 4;
55         case 3
56             pixelIndex = pixelIndex + 4;
57         case 4
58             count = 4;
59             adjustIndex = pixelIndex;
60
61             while count > 3
62                 if binaryImage(adjustIndex) == 0
63                     binaryImage(adjustIndex) = 1;
64                     count = count - 1;
65                     adjustIndex = adjustIndex + 1;
66                 end
67             end
68
69             pixelIndex = pixelIndex + 4;
70         end
71
72     else
73         blackCount = CalculateBlack(binaryImage, pixelIndex);
74
75         switch blackCount
76             case 0
77                 count = 4;
78                 adjustIndex = pixelIndex;
79
80                 while count > 3
81                     if binaryImage(adjustIndex) == 1
82                         binaryImage(adjustIndex) = 0;
83                         count = count - 1;
84                         adjustIndex = adjustIndex + 1;
85                     end
86                 end
87
88                 pixelIndex = pixelIndex + 4;
89             case 1
```

```
90         pixelIndex = pixelIndex + 4;
91         case 2
92             count = 2;
93             adjustIndex = pixelIndex;
94
95             while count < 3
96                 if binaryImage(adjustIndex) == 0
97                     binaryImage(adjustIndex) = 1;
98                     count = count + 1;
99                     adjustIndex = adjustIndex + 1;
100             end
101         end
102
103         pixelIndex = pixelIndex + 4;
104         case 3
105             count = 1;
106             adjustIndex = pixelIndex;
107
108             while count < 3
109                 if binaryImage(adjustIndex) == 0
110                     binaryImage(adjustIndex) = 1;
111                     count = count + 1;
112                     adjustIndex = adjustIndex + 1;
113             end
114         end
115
116         pixelIndex = pixelIndex + 4;
117         case 4
118             currentBit = currentBit - 1;
119             pixelIndex = pixelIndex + 4;
120         end
121     end
122
123     currentBit = currentBit + 1;
124 end
125
126 imwrite(binaryImage, 'encodedImage.bmp', 'bmp');
127 subplot(1, 2, 2);
128 imshow(binaryImage, []);
129 title('Image with Watermark');
```


解析

首先,脚本通过读取一个图像文件并将其转换为二值图像(即图像中的像素值仅包含0和1,代表黑白两色),为后续的信息嵌入做准备。接着,脚本定义了一个隐藏信息 `hiddenMessage`, 将其转换为24位的二进制形式,每位存储在数组 `messageBits` 中。随后,脚本通过遍历这个二值图像的像素,根据 `messageBits` 中的位来调整相应像素块的黑色像素数量,以此实现信息的隐藏。

具体操作中,脚本通过 `CalculateBlack` 函数计算每个像素块(由四个像素组成)的黑色像素数量,并根据需要隐藏的信息位是0还是1,对黑色像素的数量进行相应的调整。例如,如果需要隐藏的位是0,则尽可能将该块的黑色像素数量调整为3;如果是1,则调整为1个黑色像素。这种调整是通过将特定像素从白变黑或从黑变白来实现的,具体操作依赖于像素块当前的黑色像素计数。

这种方法不仅提供了一种相对隐蔽的信息隐藏手段,而且还保持了图像的视觉一致性,因为大多数的调整都是在小范围内进行,不易被肉眼察觉。通过这种技术,可以有效地在图像中隐藏重要信息,例如数字签名或其他敏感数据,而不影响图像的基本视觉展示。

3.3 提取文本信息

```
1 %% decode
2 clc;
3 clear;
4 encodedImage = imread('encodedImage.bmp');
5
6 % Initialize the bits array
7 decodedBits = zeros(1, 24);
8
9 bitIndex = 1;
10 blockIndex = 1;
11
12 while bitIndex < 24
13     blackPixels = CalculateBlack(encodedImage, blockIndex);
14     switch blackPixels
15         case 0
16             blockIndex = blockIndex + 4;
17         case 1
18             decodedBits(bitIndex) = 1;
19             bitIndex = bitIndex + 1;
20             blockIndex = blockIndex + 4;
21         case 3
22             decodedBits(bitIndex) = 0;
```

matlab

```
23         bitIndex = bitIndex + 1;
24         blockIndex = blockIndex + 4;
25     case 4
26         blockIndex = blockIndex + 4;
27     end
28 end
29
30 % Calculate the secret message from binary bits
31 decodedMessage = 0;
32
33 for idx = 1:24
34     decodedMessage = decodedMessage + decodedBits(idx) * 2^(idx - 1);
35 end
36
37 fprintf("The hidden message is: %d\n", decodedMessage);
```

解析

脚本首先从存储有编码后图像的文件中读取图像数据，该图像包含了以特定方式修改过的像素块。接着，脚本初始化一个用于存储解码位的数组 `decodedBits`，该数组将包含从图像中提取的二进制信息。通过一个循环，脚本检查图像中每个连续的四像素块（即一个像素块），利用 `CalculateBlack` 函数计算每个像素块中的黑色像素数量。根据黑色像素的数量（1 或 3），脚本将相应的位设置为 1 或 0，存入 `decodedBits` 数组。这一过程持续进行，直到收集了足够的位来构成原始的隐藏信息。

在完成所有像素块的检查后，脚本通过对 `decodedBits` 数组中的位进行加权求和（每个位乘以其对应的二进制权重），计算出最终的隐藏信息。这个数值是通过将每个二进制位转换成其对应的十进制数得到的，反映了最初隐藏在图像中的消息。此方法不仅展示了二进制数据与图像像素之间的直接关系，还体现了数字图像处理在信息安全领域的实际应用。

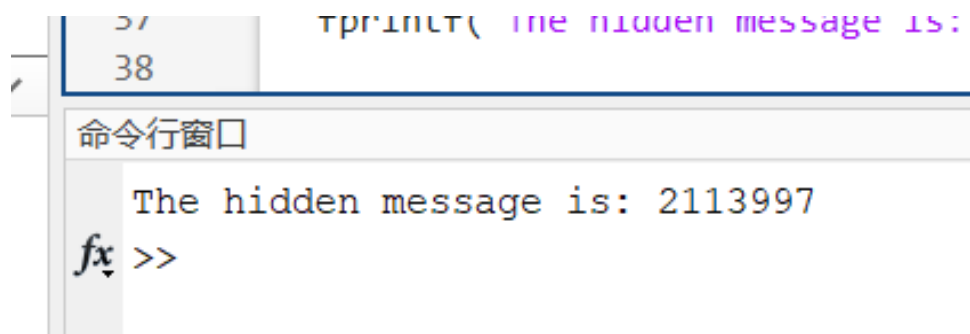
3.4 运行结果

首先运行 `encode.m`，将会生成 `encodedImage.bmp` 和 `processedImage.bmp`，分别是嵌入信息后的图片，和初始图片二值化的结果：



图 3.4.3: 嵌入信息

然后运行 decode.m，尝试解密出学号信息：



提取信息成功！

4 实验心得

在本次实验中，通过将二进制信息编码和解码嵌入到二值图像中，我深刻体会到了数字图像处理技术在信息隐藏领域的应用潜力。通过实际操作，我不仅加深了对像素级操作的理解，还学习了如何利用图像的视觉特性来安全地隐藏和提取数据。实验过程中遇到的挑战，如确保信息准确无误地嵌入及提取，以及优化算法以减少对图像质量的影响，都极大地锻炼了我的问题解决能力和编程技能。这次学习经历证明了理论知识与实践操作相结合的重要性，为我未来在更复杂的图像处理和数据安全领域的探索奠定了坚实的基础。