南开大学
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 5：图像的位平面实验

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 5 日

目录

一、 实验内容

图像的位平面实验内容:

1. 实现对 1-8 任意位平面的提取并显示;
2. 实现对 1-n 低位平面的图像显示和 8-(n+1) 高位平面的图像显示;
3. 实现去掉 1-n 位平面后的图像的显示;
4. 实现在低位平面上的图像隐藏

二、 实验过程

(一) 对 1-8 任意位平面的提取并显示

1. 原理介绍

在数字图像处理中，灰度图像的每个像素通常用 8 位二进制数（1 字节）表示，其数值范围为 0 到 255。这意味着每个像素的亮度级别可以在 256 种可能的灰度值中选择。在这种表示中，每个二进制位（比特）对于确定像素的最终灰度值扮演了关键角色。如果我们将这 8 位比特分别视为一个独立的层次或” 平面”，那么每一位比特都可以视为一种二进制图像或所谓的” 比特平面”。

将灰度值转换为二进制格式，每个像素值将由 8 位比特串表示。例如，灰度值 127 在二进制中表示为 01111111。如果我们将这些比特值“立起来”，可以想象成一个由 8 层组成的三维结构，每层代表一位比特。在这种表示中，最低位（右端）是第 1 位，代表的值最小（1），而最高位（左端）是第 8 位，代表的值最大（128）。这样，从最低位到最高位的每一层，我们可以看到不同的二值图像，每一层揭示了图像中特定亮度级别的分布。

- **最低位平面（第 1 位）**对图像的影响相对较小，改变这些位通常只会引起图像中非常细微的亮度变化。在视觉上，这些改变几乎是不可察觉的。
- **最高位平面（第 8 位）**对图像的总体亮度和对比度影响最大。这一层几乎决定了图像的主要轮廓和结构。在这一层中，一个像素要么是完全黑色（0），要么是完全白色（1），这对应于灰度值中的最大跳跃，即从 0 跳至 128 或反之。

通过分别观察和分析这 8 个比特平面，我们可以获得对图像质量、内容和结构的独特见解。例如，在进行图像压缩或加密时，理解哪些比特平面携带了最重要的视觉信息，可以帮助我们做出更加有效的决策。

2. 代码实现

```
img = imread("Lena.bmp"); % Read the image file
[m, n] = size(img); % Get the image dimensions
figure('Position', [100, 100, 1200, 800]); % Create a new large image window
for k = 1:8 % Loop to display images of eight bit planes
    c = zeros(m, n); % Create a zero matrix the same size as the image
    % Iterate over each pixel of the image
    for i = 1:m
        for j = 1:n
```

```
            c(i, j) = bitget(img(i, j), k); % Extract pixel information from the kth bit
                plane
        end
    end
    % Place images of eight bit planes in a subplot and adjust the size of the subplot
    subplot(2, 4, k);
    pos = get(gca, 'Position');
    pos(3:4) = [0.2 0.2]; % Adjust the size of the subplot
    set(gca, 'Position', pos);
    imshow(c, []);
    title(['Bit Plane ', num2str(k)]); % Set the title of the subplot
end
saveas(gcf, 'Images_of_Eight_Bit_Planes.png');
```

上述代码展示了如何从一个图像中提取并显示其八个比特平面的过程。首先，它通过 imread 函数读取一个图像文件（如"Lena.bmp"），并利用 size 函数获取这个图像的尺寸。然后，代码创建了一个新的图像窗口，为接下来展示比特平面图像做准备。在接下来的循环中，对每一个比特位（从 1 到 8），代码执行以下步骤：创建一个与原图像同尺寸的零矩阵，遍历每个像素，并使用 bitget 函数提取对应的比特位值填入矩阵。每提取一个比特平面，就通过 subplot 和 imshow 函数在图像窗口中显示出来，并设置相应的标题来指示当前显示的是哪个比特平面。循环结束后将包含八个比特平面图像的窗口保存为图片，不仅揭示了每个比特位对图像视觉效果的贡献，也提供了一种直观的方式来分析和理解图像数据的二进制表示。

3. 结果展示

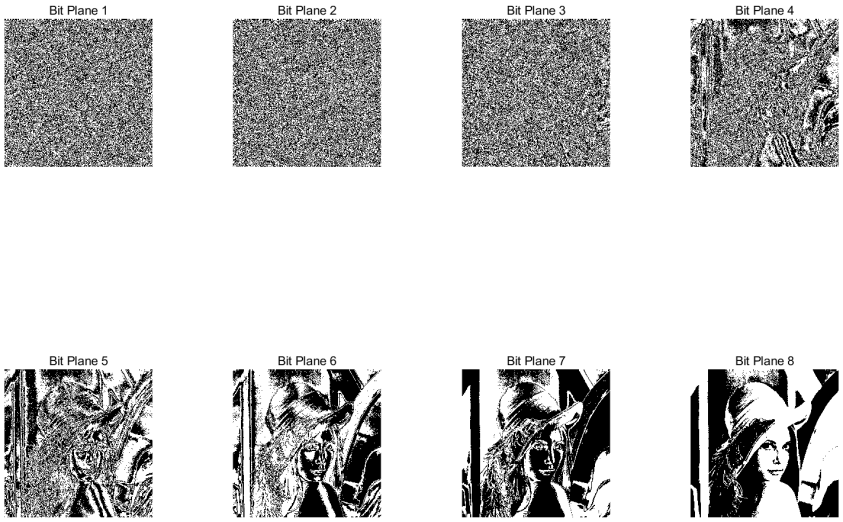

图 1: 每层位平面图像

通过观察图像的位平面（如图1），我们可以观察到不同位平面呈现出的规则性与图像特征之间的关联。较低位的位平面图呈现出较为不规则的像素分布，而高位的位平面图则更能够反映原始图像的特征。特别是最低位的位平面图，其呈现出类似于随机像素分布的特征。这一现象可归因于在二进制表示中，较低位的比特更容易受到影响而发生改变，而较高位的比特则更不易改变。从神经网络的角度来看，高位平面所保存的信息更具有整体性和高维特征，因而更能够反映出图像的整体特征。

(二) 对 1-n 低位平面的图像显示和 8-(n+1) 高位平面的图像显示

1. 原理介绍

位平面（Bit plane）是指将一个数字图像的每个像素分解成单独的位，并将这些位按其重要性（通常是按照其权重，从最低位到最高位）分组表示的平面。例如，对于一个 8 位的灰度图像，每个像素值可以从 0 变化到 255，这意味着每个像素可以用一个 8 位的二进制数来表示。在这种情况下，图像可以分解成 8 个不同的位平面，第 1 个位平面包含所有像素的最低位（LSB），而第 8 个位平面包含所有像素的最高位（MSB）。

- **低位平面**：低位平面包含了像素值中的较低位。这些位平面通常反映了图像中的细节信息，如细微的纹理变化、边缘和噪声。由于这些信息对于图像的整体视觉感知贡献相对较小，因此在某些应用中，如图像压缩，可以选择舍弃一些最低位平面以减少数据量，而对视觉质量的影响却很小。然而，对于包含大量细节信息或高频信息的图像，低位平面的重要性可能会增加。
- **高位平面**：高位平面包含了像素值中的较高位。这些位平面携带了图像中的大部分亮度信息，反映了图像的主要结构和形状。随着位的升高，每个位平面对图像的视觉质量和可辨识度的贡献也随之增加。在大多数情况下，高位平面的信息对于人眼识别图像内容至关重要，因此在进行图像处理时，这些位平面通常被保留。

2. 代码实现

```
img = imread("Lena.bmp");
k = input("Please enter the value of n: ");
[a, b] = size(img);
y = zeros(a, b);
z = zeros(a, b);

for n = 1:k
    for i = 1:a
        for j = 1:b
            x(i, j) = bitget(img(i, j), n);
        end
    end
    for i = 1:a
        for j = 1:b
            y(i, j) = bitset(y(i, j), n, x(i, j));
        end
    end
end
```

```
19
20 for n = k+1:8
21     for i = 1:a
22         for j = 1:b
23             x(i, j) = bitget(img(i, j), n);
24         end
25     end
26     for i = 1:a
27         for j = 1:b
28             z(i, j) = bitset(z(i, j), n, x(i, j));
29         end
30     end
31 end
32
33 figure;
34 subplot(1, 2, 1); % 1 row, 2 columns, 1st subplot
35 imshow(y, []);
36 title(['Bit planes 1-', num2str(k)]);
37 subplot(1, 2, 2); % 1 row, 2 columns, 2nd subplot
38 imshow(z, []);
39 title(['Bit planes ', num2str(k+1), '-8']);
40 exportgraphics(gcf, 'bit_planes_decomposition.png');
```

首先，通过读取一个图像文件来获取原始图像数据，并初始化两个空矩阵 y 和 z，分别用于存储重构后的低位平面和高位平面图像。对于每个像素，程序首先处理从第 1 位到第 n 位的低位平面，使用 bitget 函数提取每位的值，并通过 bitset 函数在新图像 y 中重建这一部分的图像信息。随后，程序处理第 n+1 位到第 8 位的高位平面，同样使用 bitget 提取位值并用 bitset 在新图像 z 中重建图像信息。这种方法使得能够分别观察图像的低位平面和高位平面包含的信息，有助于理解不同的位平面对于图像整体视觉质量的贡献。

3. 结果展示

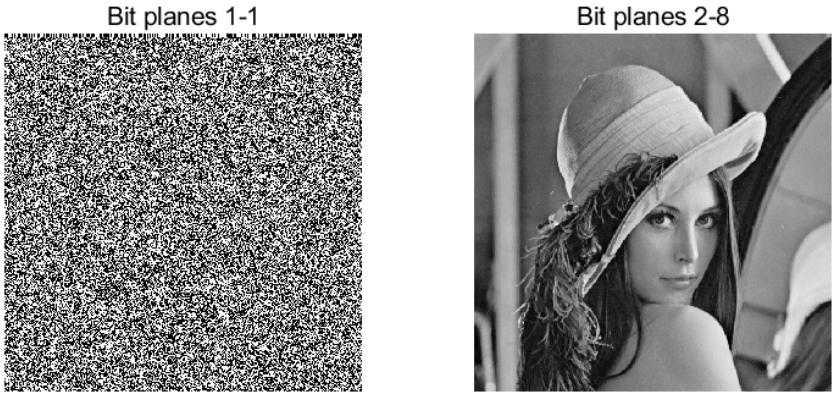

图 2: 第 1 层与第 2-8 层

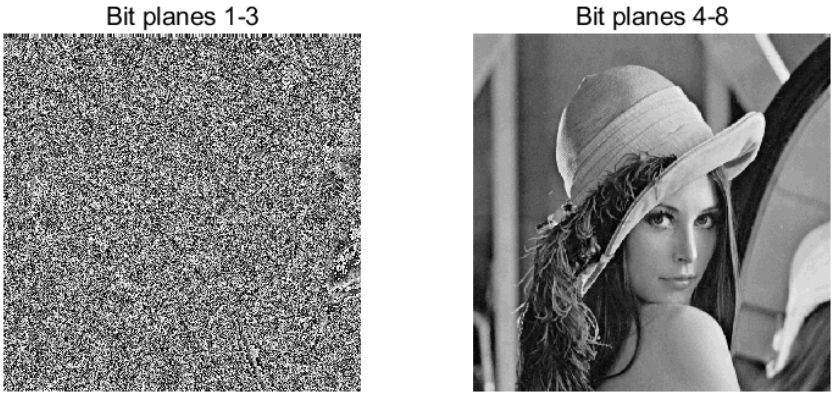

图 3: 第 1-3 层与第 4-8 层

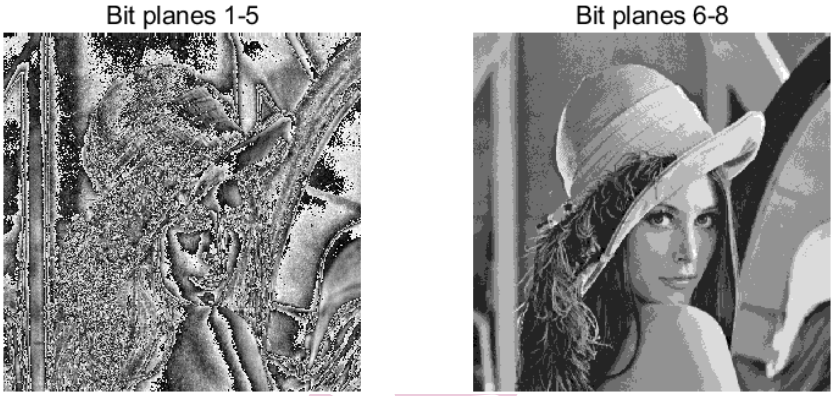

图 4: 第 1-5 层与第 6-8 层

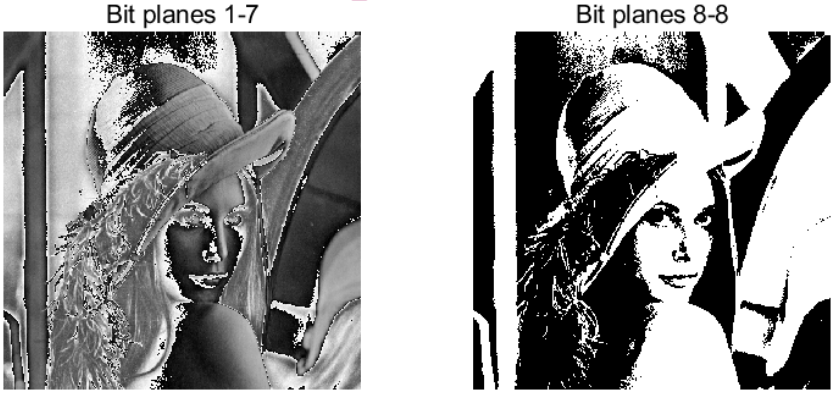

图 5: 第 1-层与第 8 层

实验效果图 (由于篇幅原因仅展示部分图片) 如图所示，通过观察实验效果图中的图像序列 3456，我们观察到在去除图像的较低层信息时，并未观察到对原始图像的显著视觉影响。然而，当我们去除了图像的第 1 至第 5 层信息后，才能够在原始图像中观察到较为明显的模糊感。这一观察结果再次验证了先前关于层次特征在图像感知中的重要性的假设。具体而言，较低层次的特征在感知上似乎不如较高层次的特征显著，这表明了图像感知在一定程度上依赖于抽象的高层次特征。

(三) 去掉 1-n 位平面后的图像的显示

1. 代码实现

```
img = imread("Lena.bmp"); % Load the image
[a, b] = size(img); % Get the size of the image
figure; % Create a figure for all subplots
for n = 1:8 % Automatically iterate through the first 8 bit planes
    modifiedImg = img; % Copy the original image for modification
    for i = 1:a
        for j = 1:b
            modifiedImg(i, j) = bitset(modifiedImg(i, j), n, 0); % Set the nth bit to 0
        end
    end
    subplot(2, 4, n); % Arrange the subplots in 2 rows and 4 columns
    imshow(modifiedImg, []); % Display the modified image
    title(['Remove ', num2str(n), ' bit']); % Use English for title
end
saveas(gcf, 'modified_image_subplot_2x4.png');
```

在这一部分实验中，我们将进行去除低层位平面图的实验。程序首先通过 imread 函数加载指定的图像文件，然后使用 size 函数获取其尺寸。接着，程序进入一个从 1 到 8 的循环，每次循环都会复制原始图像，然后通过两层嵌套的 for 循环遍历图像的每个像素，使用 bitset 函数将当前像素的第 n 位设置为 0，这样就模拟了去除了第 n 个位平面的效果。去除操作后，使用 subplot 函数和 imshow 函数在一个 2 行 4 列的图形窗口中展示修改后的图像，并为每个子图添加标题，指明移除了哪一位平面。

2. 结果展示

图 6: 去除低层位平面效果图

实验结果验证了先前观察到的规律，即高位平面图相较于低位平面图，保存了更多的图像特征。因此，实验中观察到移除低位位平面图对图像整体视觉效果的影响较小。这一现象从信息隐藏技术的角度具有重要意义。它表明，将秘密信息嵌入到图像的高位平面中，不仅能够有效保障信息的安全性，同时也减少了对图像质量的损害。基于这一发现，可以得出结论，通过在高位平面嵌入信息的方式，能够在尽可能减少对原始图像视觉质量影响的同时，保证信息隐藏的安全性。

（四） 在低位平面上的图像隐藏

1. 原理简介

隐藏图像（图像隐写术）和提取隐藏图像的过程基于一种简单而强大的原理：利用人类视觉系统对细微变化的不敏感性，通过在图像的最低有效位（Least Significant Bit, LSB）进行微小修改来隐藏信息。这种技术可以在不显著影响图像质量的前提下，嵌入大量的数据。下面详细介绍这两个过程的原理。

（1）隐藏图像的原理

数字图像是由像素（图像的基本单元）组成的，每个像素在灰度图像中通常由 8 位（1 字节）表示，这 8 位表示了 256 个可能的亮度值（0-255）。彩色图像通常包含三个颜色通道（红、绿、蓝），每个通道的每个像素也是由 8 位表示。在 8 位的像素表示中，最右边的位（即第一位）是最低有效位，它对像素的颜色或亮度贡献最小。改变这一位的值，通常不会对人眼可见的图像质量造成明显的影响，使其成为隐写术的理想选择。

隐藏信息时，首先将要隐藏的图像转换成二进制形式（如果是灰度或彩色图像，可能需要先转换成黑白形式，即每个像素只有两种状态，表示为 0 或 1）。然后，将这些二进制位逐一嵌入到另一个图像（载体图像）的像素的 LSB 中。这是通过替换载体图像的每个像素的 LSB 为要隐藏图像的相应像素值来实现的。

（2）提取隐藏图像的原理

提取隐藏信息时，需要访问并读取含有隐藏信息的图像（即经过修改后的载体图像）的每个像素的 LSB。通过收集这些 LSB 值，可以重建出原始隐藏的信息或图像。由于在隐藏过程中，原始信息是直接嵌入到 LSB 中的，提取过程基本上是隐藏过程的逆操作。通常，提取出的信息将以二值图像的形式显示，其中每个像素只包含 0 或 1。如果隐藏的是一幅图像，那么这个二值图像将直接表示那幅被隐藏的图像。如果隐藏的是其他形式的数据（例如文本），则需要进一步处理才能恢复原始数据格式。

2. 代码实现

（1）隐藏图像

```
clc;
clear all;
close all;
image = imread('Lena.bmp');
message = imread('lion.bmp');
[m, n] = size(image);
for i = 1:m
    for j = 1:n
```

```
        image(i, j) = bitset(image(i, j), 1, message(i, j));
    end
end
figure;
imshow(image, []);
title('Image After Hiding');
imwrite(image, 'hidden.png', 'png');
```

代码实现了在一个图像（Lena.bmp）的 LSB 中隐藏另一个图像（lion.bmp）的过程。使用 imread 函数读取两个图像。通过循环遍历每个像素，使用 bitset 函数修改 Lena.bmp 图像的每个像素的 LSB，将其设置为 lion.bmp 图像相应像素的值。这里假设 lion.bmp 图像已经被处理为只包含 0 和 1 的二值图像，即每个像素只代表信息位的状态（隐藏信息）。修改后的图像显示并保存为 hidden.png。由于改变只发生在 LSB，所以这种改变肉眼几乎察觉不到，图像看起来仍然与原始的 Lena.bmp 非常相似。

（2）提取隐藏图像

```
clc;
clear all;
close all;
image = imread('hidden.png');
[m, n] = size(image);
x = zeros(m, n);
for i = 1:m
    for j = 1:n
        x(i, j) = bitget(image(i, j), 1);
    end
end
figure;
imshow(x, []);
title('Decrypted Image');
saveas(gcf, 'decrypted.png');
```


代码实现了从修改过的图像（hidden.png）中提取隐藏信息的过程。使用 imread 函数读取含有隐藏信息的图像。初始化一个与原图相同大小的矩阵 x，用于存储提取出的信息。通过循环遍历每个像素，使用 bitget 函数获取每个像素的 LSB，并将这个值存储到矩阵 x 中。由于隐藏信息是通过修改 LSB 来嵌入的，所以这一步实际上是读取隐藏的图像信息。显示并保存提取出的图像（decrypted.png）。这个图像应该与最初用于隐藏的 lion.bmp 图像相似，尽管可能因为位深度的差异而在视觉上有所不同。

3. 结果展示

在完成图像隐藏与提取的过程后，我们得到了两个关键的输出：隐藏后的图像和解密后的图像。这两个图像分别展现了隐藏信息后的效果和提取隐藏信息后的结果。

图 7: 隐藏后的图像

如图所示，这是在原图像中嵌入另一图像信息之后的结果。尽管图像的每个像素的最低有效位（LSB）被修改以包含另一图像的信息，肉眼几乎无法区分出与原始图像之间的差异。这说明了 LSB 隐写术的效力：能够在不显著改变视觉感知的情况下，隐藏大量的数据。

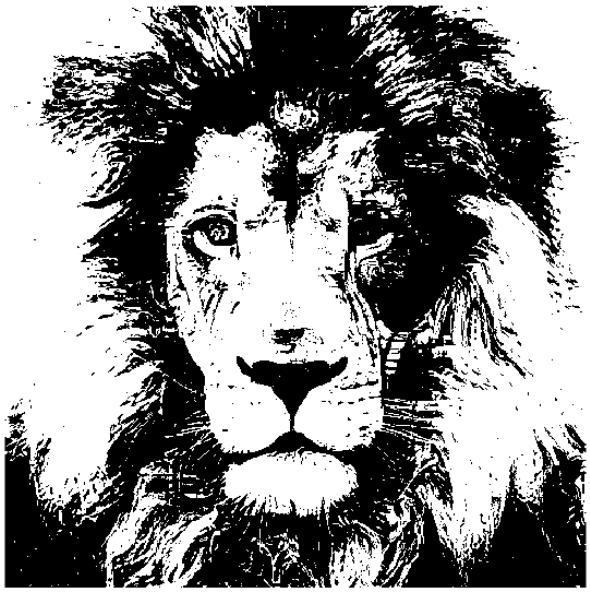

图 8: 解密后的图像

解密后的图像显示的是从隐藏信息图像中提取出的信息，即我们最初尝试隐藏的图像。这个过程通过提取每个像素的 LSB 并将其重构成一个新的图像来完成。由于这些 LSB 包含了隐藏图像的信息，因此重构出的图像应该与最初隐藏的图像相似。然而，这里需要注意的是，由于在隐藏过程中只使用了二值信息（即 0 和 1），所以解密后的图像可能在色彩深度和细节上与原始

隐藏图像有所不同，尤其是如果原始图像是彩色的。

通过这两个结果，我们可以看到数字图像隐写技术的强大能力，它允许我们在一个图像中秘密地嵌入另一个图像，而不会引起观察者的注意。

三、 扩展实验（多位隐写实验）

从最低有效位（LSB）扩展到多位隐写实验大幅**提升了图像隐写术的数据隐藏容量，同时为我们提供了评估图像质量与隐蔽性之间权衡的机会**。通过在载体图像的更多位上嵌入信息，我们不仅可以隐藏更大的数据量或更复杂的图像，还可以探索新的应用场景。

(一) 原理简介

在传统的 LSB 隐写术中，信息是通过替换图像的最低有效位来隐藏的。例如，如果你想在一个 8 位的灰度图像中隐藏文本，可能会将文本转换为二进制形式，然后将这些二进制位逐个嵌入到载体图像的像素值的 LSB 中。这种方法的一个优点是对图像的影响最小，因为 LSB 的变化对图像的整体视觉效果影响不大。在多位隐写中，我们不仅使用最低位，而是利用像素值的最低两位、三位或更多位来隐藏信息。这意味着每个像素现在可以存储更多的信息（例如，使用两位可以存储 4 个可能的值，使用三位可以存储 8 个可能的值）。

使用更多的位来隐藏信息意味着对原始图像的影响更大。LSB 改变对图像质量的影响可能是微不足道的，但是更改较高的位可能会使图像出现可见的失真。因此，需要权衡隐藏信息的需要与保持图像质量之间的平衡。通过在每个像素中隐藏更多的位，可以显著增加隐藏数据的总量。这对于需要传输大量隐秘信息的应用场景非常有用。

实现多位隐写的一个方法是通过修改载体图像中像素值的最低两位或三位来嵌入信息。例如，如果使用两位，可以将每个像素的最低两位替换为二进制数据流的两个连续位。这将要求原始隐藏信息在大小上进行适当的调整，以确保它适合于修改后的容量。

(二) 代码实现

下述代码段实现了一种多位隐写技术，具体是通过修改载体图像的最低两个有效位（2 LSBs）来隐藏信息图像。该方法允许在图像中嵌入更多的信息，同时尽可能保持图像的视觉质量。

```
clc;
clear all;
close all;

% 读取载体图像和信息图像
carrier_image = imread('Lena.bmp'); % 载体图像
message_image = imread('lion.bmp'); % 信息图像，假设已经预处理为0-3之间的值

[m, n] = size(carrier_image);

% 确保信息图像与载体图像大小相同并且是uint8类型
message_image_resized = imresize(message_image, [m, n], 'nearest');
message_image_resized = uint8(message_image_resized); % 转换为uint8类型

for i = 1:m
    for j = 1:n
```

```
        % 清除载体图像当前像素的最低两位
        carrier_image(i, j) = bitand(carrier_image(i, j), uint8(252)); % 252 = 11111100, 确保操作数为uint8类型
        % 将信息图像的两位信息添加到载体图像的最低两位
        carrier_image(i, j) = bitor(carrier_image(i, j), message_image_resized(i, j));
    end
end

figure;
imshow(carrier_image, []);
title('Image After Hiding with 2 LSBs');
imwrite(carrier_image, 'hidden_2LSB.png', 'png');
```

第一段代码中，首先读取了载体图像（Lena.bmp）和预处理后的信息图像（lion.bmp），后者假设已经调整为只包含 0 到 3 之间的值，以便通过载体图像的最低两位来表示。代码接着确保信息图像被调整为与载体图像相同的尺寸，并且是 uint8 类型。通过遍历每个像素，代码首先使用 bitand 操作和值 252（二进制的 11111100）来清除载体图像像素的最低两位。然后，利用 bitor 操作将信息图像的相应像素值嵌入到这两位中。这样处理后的载体图像包含了隐藏的信息，且视觉上与原始图像相似，最终将其显示并保存为 hidden_2LSB.png。

```
clc;
clear all;
close all;

% 读取含有隐藏信息的图像
hidden_image = imread('hidden_2LSB.png');
[m, n] = size(hidden_image);

% 初始化一个矩阵来存储提取出的信息，确保其类型为double或与要进行位操作的数据类型一致
extracted_message = zeros(m, n, 'uint8');

for i = 1:m
    for j = 1:n
        % 提取每个像素的最低两位，保证操作数为相同类型
        extracted_message(i, j) = bitand(hidden_image(i, j), uint8(3)); % 3 = 00000011
    end
end

figure;
imshow(extracted_message, []);
colormap(gray(4)); % 调整颜色映射以显示0-3之间的值
title('Decrypted Image with 2 LSBs');
imwrite(extracted_message, 'decrypted_2LSB.png', 'png');
```

在第二段代码中，实现了从处理过的图像（hidden_2LSB.png）中提取隐藏信息的过程。这一过程同样涉及遍历图像的每一个像素，使用 bitand 操作和值 3（二进制的 00000011）提取每个像素的最低两位。提取出的信息被存储在一个新的矩阵中，并使用 imshow 函数以特定的颜色映射（gray(4)）显示，使得 0 到 3 之间的值可视化，最后将提取的信息图像保存为 decrypted_2LSB.png。

(三) 结果展示

在本扩展实验中，我们采用了多位隐写技术，在载体图像的最低两位中隐藏了另一幅图像的信息。此方法旨在探索使用多位隐写时数据容量增加与图像视觉质量之间的关系。下面展示了实验的关键结果，包括隐藏信息后的图像和从该图像中提取的隐藏信息图像。

图 9: 隐藏后的图像

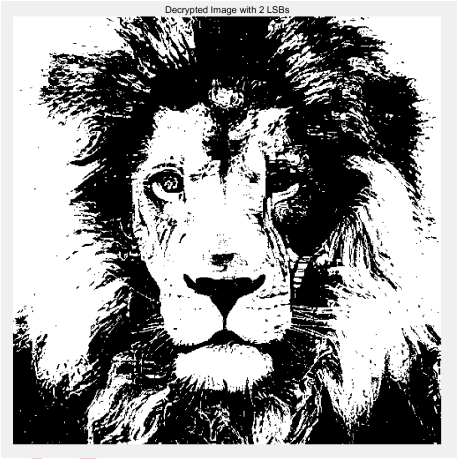

图 10: 解密后的图像

左侧图像显示了在将隐藏信息嵌入后的载体图像。通过将信息隐藏在图像的最低两位中，我们能够在不显著影响图像视觉质量的情况下，增加隐藏信息的量。尽管如此，细心的观察者可能会注意到相比原始图像，隐藏后的图像在某些细节上略有差异，特别是在颜色过渡较为平滑的区域。

右侧图像展示了从修改后的载体图像中提取出的隐藏信息。尽管在多位隐写过程中，图像信息经过了较为复杂的嵌入与提取过程，提取出的图像仍然保持了较高的清晰度和可识别度。这证明了即使在使用多位进行隐写时，我们也能有效地恢复隐藏的信息。

四、 实验心得体会

本次通过参与这一系列的图像处理实验，包括位平面分析、多位隐写实验等，我获得了宝贵的学习经验，深化了对图像处理技术和隐写术原理的理解。实验不仅增强了我的实际操作能力，也让我对数字图像的本质有了更深入的认识。

在对 1-8 位平面提取并显示的实验中，我明白了每个位平面在图像中所扮演的角色，尤其是在图像的视觉质量和信息含量方面。通过这个实验，我学会了如何分析和解构图像，了解到最高位平面对图像的轮廓和结构影响最大，而最低位平面则主要影响图像的细节信息。这一发现对于理解图像压缩和加密技术非常有帮助。进一步地，通过多位隐写实验，我探索了在载体图像的最低两位或三位上隐藏信息的技术。这不仅让我认识到隐写技术在增加隐藏信息容量方面的潜力，也让我意识到在隐写术的应用中需要权衡信息容量和图像质量之间的关系。特别是，在实践中操作和观察到，即使是微小的改动（比如修改最低的两位），也可能对图像的视觉质量产生影响，尽管这种影响不总是肉眼可见的。

这一系列的实验加深了我对于隐写技术不仅仅是隐藏信息那么简单的认识。它还包括了对载体图像的理解、对隐写信息提取的精确度、以及隐写技术的安全性等多个方面。这些实验让我意识到，作为一个有效的隐写工具，需要在技术的隐蔽性、容量、鲁棒性和安全性之间找到平衡。