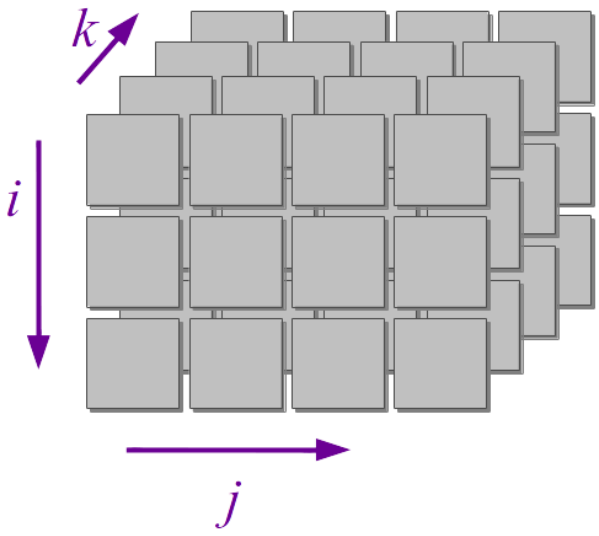
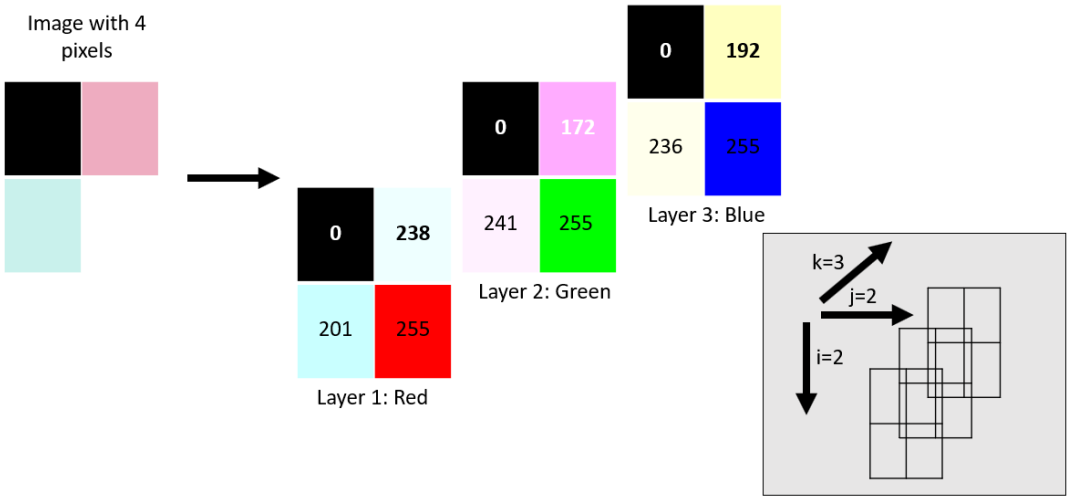
LSB implementation in RGB images

An image in a computer is saved as a Matrix in which every position saves the information of a single Pixel. The matrix is usually three-dimensional (I x j x k) and its respective sizes depend on the height, length and color model of the image.



In this case we will talk about an LSB implementation for images with the color model RGB. Therefore, k will be equal 3. This means, that the image can be seen as three layers of I x j matrices and each layer will save the information of each pixel for the weight of the colors: red, green and blue.

Here is an illustrative example of a matrix that represents an RGB image:

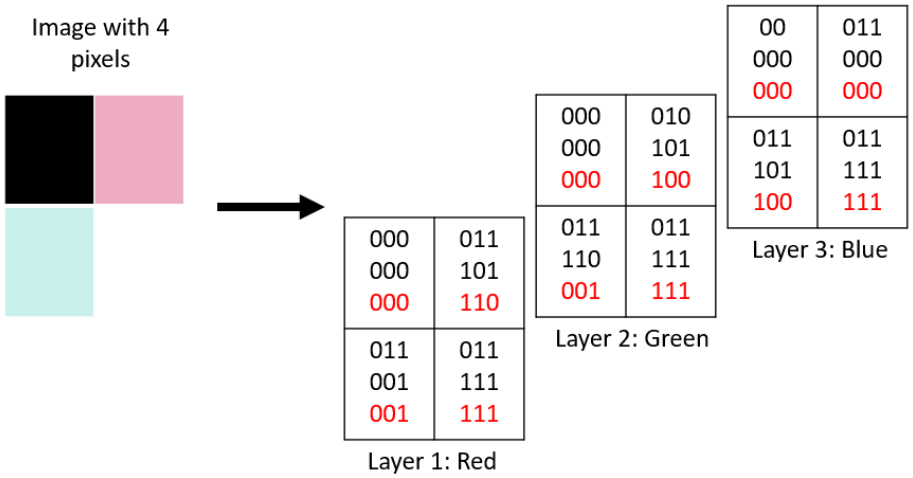


<u>The embedding algorithm (Intuition)</u>

Now that we understand the computational representation of an image it is time to introduce the concept of LSB. The idea of hiding the information in the least significant bit is to hide each character of the message in the last bits of an A_{ijk} of the matrix.

In order to do that is important to mark that the numbers saved in the matrix are written in binary code and since they can go from 0 to 255 the binary representation has a range 00000000-11111111. Additionally, the ASCII code also assigns values that have no longer than 8 digits and this is allows the hidden message to fit perfectly in the least significant bits.

Let´s explain the LSB algorithm intuitively with an example based on the image 2. In this case the numbers saved on each will be shown on their binary representation as follows:



Notice that the numbers were separated in blocks of three and we added a 0 before each number. The last block for each number are the least significant bits, because those digits are the ones that allow us to change its values without changing the number itself the least as possible. For example, in the position A_{121} the number is 11101110 and with the additional 0 and the block separation we have 011 101 110 meaning that 110 are the least significant bits.

Now we are going to explain the process of message embedding. The number of characters we can hide in this image is 4 because we have 4 pixels. In this case we want to hide the word “love”.

Character	ASCII	Binary Representation
l	108	001 101 100
o	111	001 101 111
v	118	001 110 110
e	101	001 100 101

Using the binary representation of each character we only have to save each block of a character in the same position but in the 3 different layers. For example “l” will be stored in A_{111} , A_{112} and A_{113}. Meaning that those positions will be changed as follows:

Position	Before	After
A_{111}	000 000 000	000 000 001
A_{112}	000 000 000	000 000 101
A_{113}	000 000 000	000 000 100

Applying those changes to every position of the matrix we obtain the following:

000 000 001	011 101 001
011 001 001	011 111 001

Layer 1: Red

000 000 101	010 101 101
011 110 110	011 111 100

Layer 2: Green

00 000 100	011 000 111
011 101 110	011 111 101

Layer 3: Blue

1	233
201	249

Layer 1: Red

5	173
246	252

Layer 2: Green

4	199
238	253

Layer 3: Blue

The final image :

Before

After