Université de Genève

Chosen Chapter
14x060

TP 1 : Steganography

Author: Deniz Sungurtekin

E-mail: Deniz.Sungurtekin@etu.unige.ch

Mars 2020

1 Introduction

In this report we will implement the LSB extraction and re-insertion by embedding a message into it. First, we will see how we can split an image into bitplanes and analyse their impact on the original image. Then we will discuss the robustness of this method to classical processing and the differences we can see in the stego image.

2 LSB extraction and re-insertion

2.1 Task1

The goal here is to put our uint8 image in grayscale and to extract the 8-bit planes of the image. Slicing the image at different planes plays an important role in image processing for example in data compression.

Here is our original RBG image:

Figure 1: Original Image VS Grayscale

Now, for each pixel we will simply convert the value in binary and store each 8 bits in each 8 bits planes. For example, the first plane will be made of the most significant bit of each pixel so we will have a matrix in double format composed of 1 and 0. In particular, we can visualize the 1th, 4th and 8th bitplane:

Figure 2: Some bitplanes

We can see that the more the bitplanes are made of significant bits the more we distinguish the original image which is logical because he carries more information on the pixel. With those eight bitplanes we can totally reconstruct the original image by simply doing the inverse operation:

Figure 3: Original Vs Reconstruction

More importantly, we can almost reconstruct the exact image by not taking in account the less important bitplanes. For example, if we set at 0 one of the 8 bitplanes we can have a good result depending of which one we remove. Here, an example with the 8th, 4th and 1th:

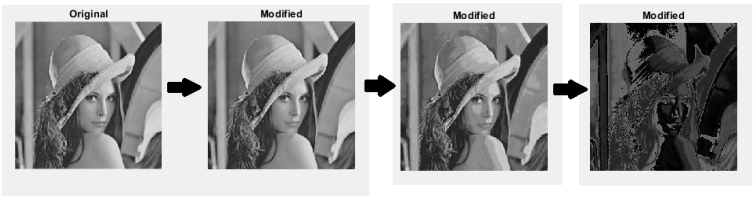

Figure 4: Reconstruction with one bitplane set to 0

Of course, it might not be very clever to ignore the most significant bitplane because he carry most of the information. However we can clearly see that when we remove the 8th or the 4th, the image is still in a good shape.

3 Embedding a message into LSB

3.1 Task2

Now that we have see that the LSB have less impact on the image, we can deduce that it's possible to hide some secret inside. For example, let's hide 100 times the message "Help me " in our image. We simply need to convert our message in binary and store it in the LSB bitplane. In general, if we have a string message of lenght M we will need 8 * M = L bits to encode our message with ASCII encoding. Then, we just need to choose L random position in the plane to store our message. Finally, we simply reconstruct our image by regrouping all planes:

Figure 5: Reconstruction with secret in LSB

We can observe that the differences between the original and secret image can't be found by human eyes. To find the secret without any errors the receiver only need a key indicating at which position the secret is in the LSB plane.

Now, we will compare both image with some measure to see their differences:

distortion =	PSNR =	StructuralSimilarity =
2817	61.7977	0.9997

Figure 6: Comparison

To compute the distortion, we first compute the pixel-wise difference matrix which indicate where the value are different. In our case, those locations are set to 1 because we only modified the LSB. Then we simple count the number of 1 in this matrix to obtain our distortion. The distortion is not equal to the number of bits we modified because sometimes the original value is the same than the inserted value. So we inserted 5600 bits in the image and we had 2817 different pixels which is almost 50%.

In regards to the PSNR which use the MSE between two images, we clearly see that this method is easy to detect because we have only a PSNR of 61. However, for the SSIM, a perceptual metric that quantifies image quality degradation, we have good result because as we can observe its very hard to distinguish the differences between the two images.

4 Embedding of binary image into LSB

4.1 Task3

This time we will hide an entire grayscale image inside a RGB image. We will extract the three colors channels and hide the message in the LSB of a channel (for example the green one). The method is the same as the previous one, we will separate our green channel in 8 bitplanes, insert the secret inside the 8th plane, reconstruct the modified green channel and merge the RGB color channel. Here are our original images:

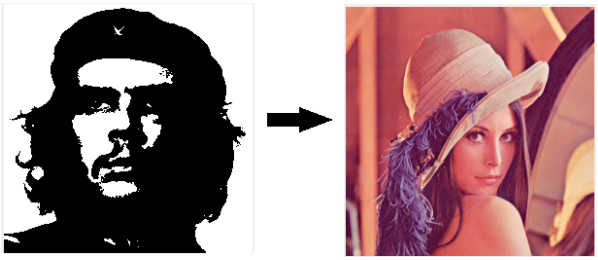

Figure 7: Secret VS Original

In this case, our secret image has the same dimension than our green channel so to reconstruct our secret without any error we only need to extract the LSB of the green channel.

Here are the stego image and the reconstructed secret:

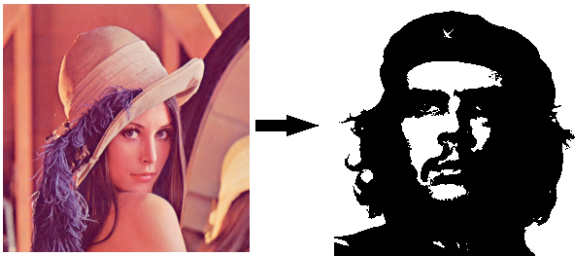

Figure 8: Stego VS Reconstructed secret

Now, we will analyse the impact of the chosen channel on the method by computing the pixel-wise difference, the distortion, the PSNR, the SSIM and the color difference for each channel:

distortion = 32880	PSNR = 51.1263	StructuralSimilarity = 0.9984	ColorDifference = 19565
distortion = 32927	PSNR = 51.1201	StructuralSimilarity = 0.9980	ColorDifference = 19616
distortion = 32852	PSNR = 51.1300	StructuralSimilarity = 0.9981	ColorDifference = 19579

Figure 9: Green/Red/Blue channel

As we can see, the chosen channel have not a big impact on the result. The more the chosen secret have a similar distribution than the LSB the more its hard to detect it. In general, we can easily detect this kind of steganography by computing the PSNR or the color difference (In my case the sum of all element in the color difference matrix). Moreover, the differences only appear in the corresponding modified channel so i only put the measure for the corresponding color.

Furthermore, we can also modify the algorithm such that the secret image is scrambled according to a key which indicate the position and the order in which the secret is stored in the LSB. Finally, it's possible to store a smaller secret image by simply having a key indicating which pixels doesn't contain any information on the secret.

4.2 Task4

In this section, we will discuss the effects of various processing on LSB stego images. Unfortunately, this method isn't robost to typical image processing. For example, cropping an image crop also our secret because its information is stored in the corresponding pixel of the cover image:

Figure 10: Cropping a cover

With the same reasoning adding some noise produce the same effect:

Figure 11: Peper Noise

However, the biggest impact is shown by a simple double compression:

Figure 12: Double compression

This process change completely the LSB of our image so there is absolutely no clue of the secret anymore. If we wanted robustness to Jpeg compression we needed to store our secret inside the LSB in DCT domain.

Finaly, histogram equalization leave also a trace in the secret because we will change some pixels value and therefore their LSB (50% of chance):

Figure 13: Histogram equalization

4.3 Task5

As we discuss in the previous chapter, the LSB embedding is easy to detect. Moreover this method leaves traces into image histogram, so we will analyse it to see which effect it produce. We will compute the pixel histogram of the original green channel and of the stego green channel:

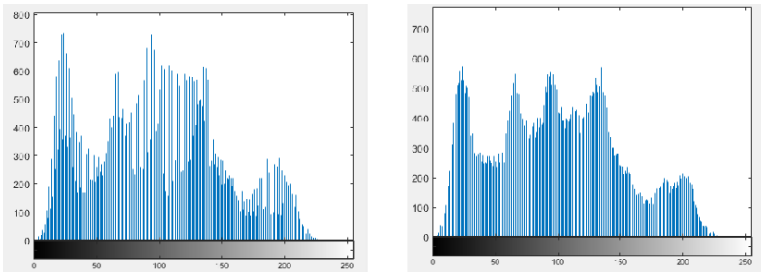

Figure 14: Stego/Original

We can clearly see that there is much more occurrences for some pixels value and more empty bins in the stego histogram.