University of Illinois at Urbana Champaign

ECE 418 Image and Video Signal Processing

Final Project Report

Steganography
A DCT Approach
With Image Hidden Inside Image

Author	*Net ID*
Alvin Sun	yixiaos3
Bowen Jiang	bowenj2
Maohao Shen	maohaos2

May 8, 2019

1 Introduction

Our project mainly focus on the digital steganography performed on the frequency domain of colored images, which is also commonly known as the watermarking. Motivated by the increasing demands of security in massive data transmissions and digital communications, we designed our project to be able to hide digital images as the watermark inside larger images as the message to be transmitted, whose histogram, spatial domain, and frequency domain are all indistinguishable from the original. To make the following explanation clearer, we would use the naming convention for the 2 images as below.

- **Carrier Image.** The larger base image that will eventually be transmitted or stored to disk.
- **Watermark Image.** The smaller image that is embedded inside the Carrier Image. It should not be visible during transmission.

2 Proposed Approach

Most of the images have their frequency components concentrated in the low frequency ranges, while details expressed as edges also exist in the very high freqeuncy range. Therefore, we decided to hide the *watermark image* into the mid-high frequency regions of the *carrier image*, which has the least significant impact on the visual effect of the image.

2.1 General Idea

Discrete Cosine Transform (2D-DCT) is chosen to obtain the frequency domain of the *carrier image*, other than the Discrete Fourier Transform (2D-DFT), in order to avoid the complex number involved in the calculation. Meanwhile, RGB color space is selected, instead of LAB color space, so that the troublesome nonlinear transformation of color spaces can be avoided. Any nonlinearity could potentially introduce floating point truncation errors that could badly hurt the quality of the hidden *watermark image*.

2.2 Encoding Stage

In the encoding stage, the *watermark image* is read directly from an image file on the disk, which stays with its compressed format (e.g. JPEG, PNG, etc.) The *carrier image* is also read from the disk, but will be decompressed into raw RGB pixel format. And then the image will go through normalization such that all pixel values falls within $[0, 1]$. Then, we iterate through a predetermined rectangle range in the mid-high frequency region of the frequency domain of the *carrier image*. The iteration also scans through the 3 RGB channels as it progresses. Every iteration, when the magnitude of a frequency component is less than a pre-determined threshold, we encode one data byte from the *watermark image* into the magnitude of that freqeuncy value, which means its sign is preserved. The encoding is not a direct assignment as the normalization cause the average magnitude of the freqeuncy component to be quite small. Instead, we multiple the byte value with a pre-defined scale and then assign this scaled value to the magnitude of the frequency component. The smaller the scale it is, the smaller the visual impact this freqeuncy domain operation could cause, but also the less significant it has on the spatial domain pixel values. As a result, we need a 16-bit quantization instead of 8-bit to preserve the fine details we modified inside the freqeuncy domain. In addtion to encoding the image data itself, we also needs to know the total length of the *watermark image* byte stream for the decoding stage, so we encode this length information as a 4-byte unsigned integer

using the same scheme as encoding the data, but at a pre-determined location within the DCT image. Figure 1 shows a general work flow of the encoding stage.

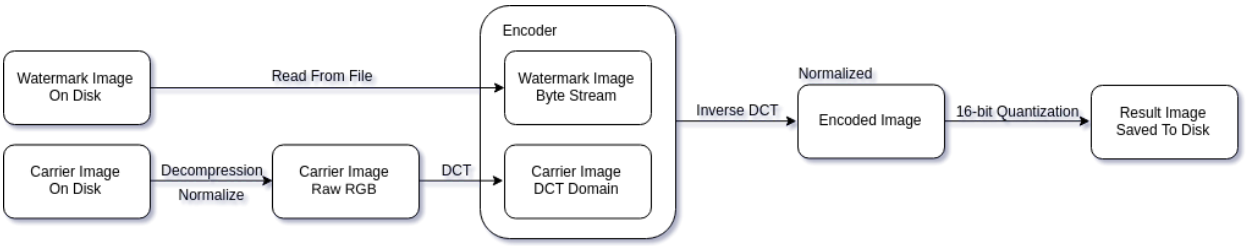


Figure 1: Encoder Diagram

2.3 Decoding Stage

In the decoding stage, the whole process is not very different but just the reverse of the encoding stage. As we iterate through the rectangle region and three color channels, if the current frequency magnitude is lower than the predetermined threshold, we can divide the magnitude by the predetermined scale and then round it to the neareast integer value to extract the original *watermark image* bytes, until the number of bytes retrieved matches the length we extract at the very beginning from a pre-determined location. After the complete byte array is obtained, the data will be transformed back to the colored image through image decompression based on its original format. Figure 2 shows a general work flow of the decoding stage.

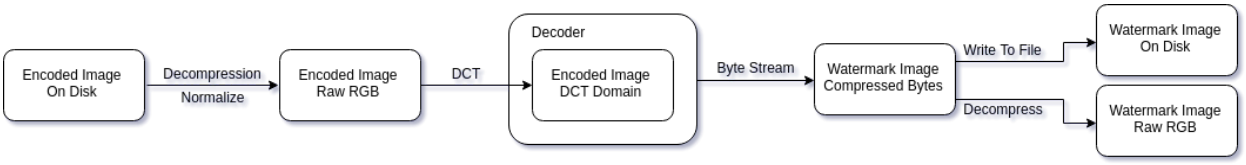


Figure 2: Decoder Diagram

3 Experimental Design And Results

3.1 Carrier Image Choice

We implemented and experimented with a carrier image shown in Figure 3, which has many high frequency components. This will be especially challenging for us to test the robustneses of our steganography algorithm as we are potentially modifying quite an amount of mid-high freqeuncy component of the original *carrier image*. This also helps us visualize the actual impact on the visual impression of the *carrier image* when embedding the *watermark image* inside.

Figure 3: Carrier Image

3.2 DCT of Carrier Image

Shown in Figure 4 is the DCT domain
of the 3 RGB channels of the original *carrier image*. The values are truncated to enhance the contrast for the purpose of visulization. As we can see from the images, most information is concentrated in the low frequency region, while high freqeuncy components also occupy a decent portion of the total energy.

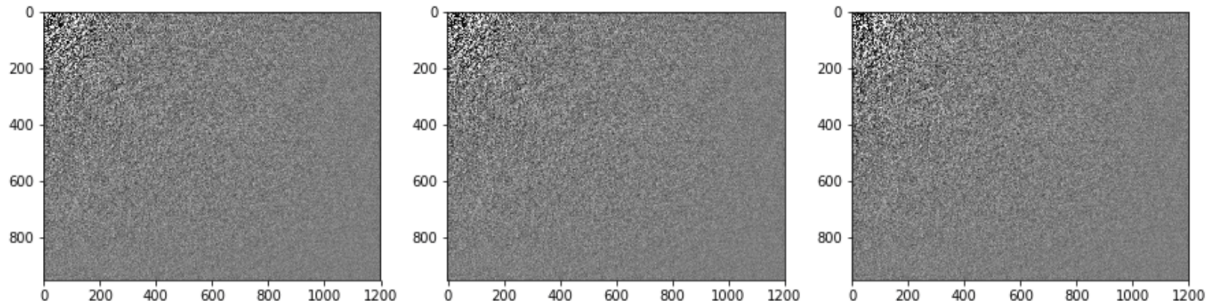


Figure 4: DCT of The Original Carrier Image

3.3 Watermark Image Choice

Because the maximum length of data bytes that can be encoded is bounded by the size of the rectangle region we pick in the frequency domain, we cannot hide very large sized image within the *carrier image*. Therefore, we picked the resized picture shown in Figure 5 as our *watermark image* to experiment with.

Figure 5: Watermark Image

3.4 Encoding / Decoding Results

After the encoding stage, the inverse DCT gives us the following result as shown in Figure 6a, which looks indistinguishable from the original image before the encryption. Then we put this encoded image to our decoder, and we extracted back the *watermark image* with absolutely no attenuation. In other words, the recoverage of hidden information is perfect. Figure 6b is the extracted hidden *watermark image* from the encoded *carrier image*. Again, we can see that it is pixel-wise identical to the original *watermark image*.

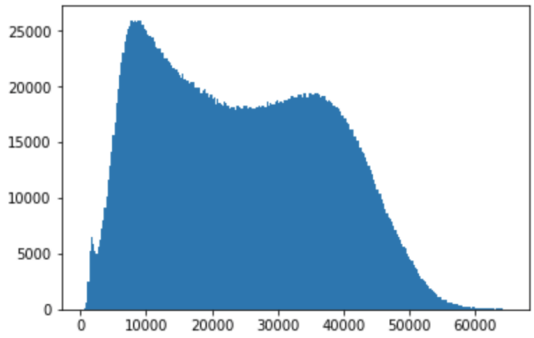
(a) Encoded Carrier Image

(b) Decoded Watermark Image

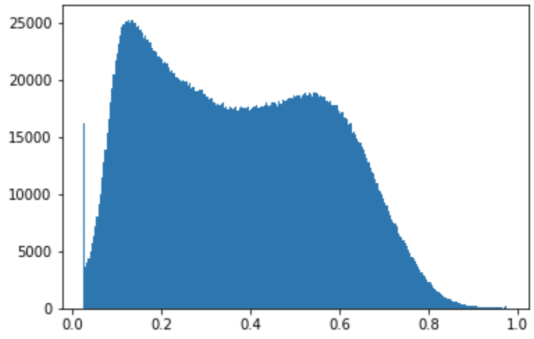
Figure 6: Encoder / Decoder Results

3.5 Histogram Analysis

We have also compare the histogram of the image before and after watermarking, and the result shows that it is unobvious to discover that the *carrier image* has been watermarked. Figure 7 shows the the 2 histograms.



(a) Histogram of Original Image



(b) Histogram of Watermarked Image

Figure 7: Encoder / Decoder Results

4 Discussion And Future Work

In summary, the result of this project reached our expectation. At the beginning, we decided to implement steganography for text message, but we soon tried to improve it to hide small image into large image, which is more robust and add an additional level of indirection to the message we would like to hide.

Our steganography can hide any type of digital information inside large colored images in the form of high frequency components within DCT domain. Meanwhile, the whole encryption process can not be discovered from the spatial domain image, frequency domain image, or the pixel statistics such as histograming. This ensures the security of the hidden message or image.

That being said, there still exist some shortcomings as well. For example, the watermark image is of limited size and definition. Also, the encoder and decoder needs to agree on quite a large number of pre-determined constants to have a perfect reconstruction of the hiddle information. We still have some ideas to improve this project. Considering the security level, we can encode the message into random locations of large image, instead of in a fixed rectangle region. To implement this, we need a pseudo-random function with a seed, which generates pseudo-random locations for message encoding, while using the same seed in decoding process. This improvement will be able to make this project more secure and reliable, since no one could find the difference in frequency domain even though someone else may try to find regularity based on large amounts of encoded images. Another relatively tricky improvement is real time steganography. we may try to encode message or small images in video such as video chat. This process may require processing a lot of data and we also need consider the temporal domain effect.

Appendix

Our code is open source and is developed in Python 3.6, which is available HERE. To use the encoder / decoder program, simply run

```
0  python3 encode.py <carrier_img> <watermark_img> # for encoding
1  python3 decode.py <encoded_img> # for decoding
```

Note. To run the program, you may need to look into the source code and get the prerequisite package installed beforehand.