Python Implementation of Image Steganography

Mason W. Edgar
University of Arizona
ECE 529 – Fall 2020
Tempe, Arizona

***Abstract*—Steganography is a useful, yet lesser known counterpart to traditional cryptography. There are many approaches to steganographic obfuscation, and this paper seeks to explore the realm of image steganography. A Python3 implementation of a modified JSteg algorithm is discussed and evaluated.**

Keywords—DSP, DCT, IDCT, Steganography, JSteg, Cryptography, Data Hiding

I. Introduction

Communication is a fundamental aspect of all sentient life. The ability to communicate is critical for any species to survive and evolve. There are various types of communication in modern society, and with the proliferation of the internet, it has tied the entire world together into a global society. For as long as people have been able to meaningfully communicate with each other, there has been an innate desire to prevent certain information from being readily available to unauthorized recipients. This desire for obfuscation has created two fields of interest: cryptography and steganography. Cryptography seeks to systematically obscure information into a format that is completely unusable for anyone the information isn't intended for. Being able to send cryptographically secure messages is a crucial part of any nation's defense strategy. There have been several cryptography methods employed by nations during the various wars across the planet over time. One particularly famous historical example of cryptography in action was the United States "Navajo Code Talker" program during the Second World War, which gave the United States a nearly unbreakable cryptographic code to transmit critical wartime information [1]. While cryptography is a lucrative and popular field of study today, there exists a second form of obfuscation: steganography. Cryptography and steganography are quite similar in many regards, but they have some key differences. While cryptography seeks only to render the secret message unreadable, steganography seeks to render the message invisible by hiding data in plain sight. Steganography has many branches within the overarching field, but the branch of interest to this report is image steganography. The aspect of a "World-Wide Web" is deeply engrained into modern culture, and many can't imagine a world before it was available. Digital pictures are a particularly prevalent class of data being exchanged over the internet, so they make for an excellent steganography medium. The goal of this paper is to explore and implement a slightly modified version of the rather popular image steganography algorithm known as "JSteg".

II. Background

The JSteg algorithm is a modified version of the standardized JPEG encoding algorithm and invented by Derek Upham [2]. The main idea behind JSteg is to manipulate the Forward Discrete Cosine Transform (DCT) portion of the standard JPEG encoding process in order to embed secret data. The JSteg algorithm embeds the secret data in to the least-significant bit (LSB) of the AC coefficients that are not equal to -1, 0, or +1 [2]. There are many versions of the Discrete Cosine Transform process, but the version of interest is the orthonormal Forward Discrete Cosine Transform – Type II and orthonormal Inverse Discrete Cosine Transform – Type II (IDCT). The equations for both the Forward and Inverse DCT-II are shown below in Figure 1 [2]:

$$F(u,v) = \left(\frac{C(u)}{2} \frac{C(v)}{2} \sum_{x=0}^{7} \sum_{y=0}^{7} f(x,y) \cos \frac{(2x+1)u\pi}{16} \cos \frac{(2y+1)v\pi}{16} \right)$$

where,

$$C(u) = \begin{cases} \frac{1}{\sqrt{2}} & \text{for } u = 0 \\ 1 & \text{for } u > 0 \end{cases} \quad and \quad C(v) = \begin{cases} \frac{1}{\sqrt{2}} & \text{for } v = 0 \\ 1 & \text{for } v > 0 \end{cases}$$

Figure 1: Equations for the Forward and Inverse Orthonormal Discrete Cosine Transforms

Before any transformations can be implemented, the host data (carrier image) must first be broken down into 8x8 non-overlapping data (pixel) blocks. These blocks are fed sequentially into the DCT, starting from the top left corner of the image, and working towards the right and then down to the bottom right corner. An example block [3] is shown in Figure 2 that details a high-level view of the DCT process:

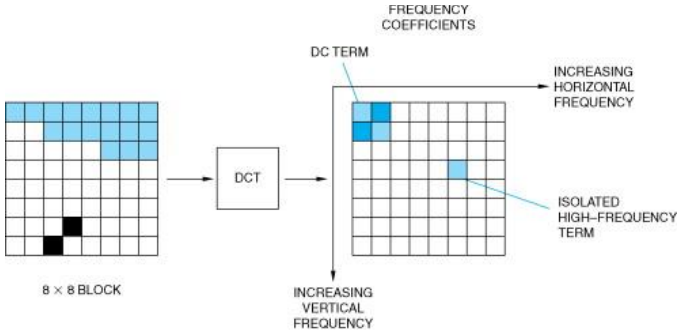

Figure 2: Example of 8x8 Pixel Block Undergoing DCT

Once the 8x8 block has been transformed, "each DCT coefficient indicates the amount of a particular horizontal or vertical frequency within the block. DCT coefficient (0, 0) is the DC coefficient, or average sample value. Since natural images tend to vary only slightly from sample to sample, low frequency coefficients are typically larger values and high frequency coefficients are typically smaller values" [3]. Since the DC coefficient involves taking an average over the entire block, modifications to this coefficient cause the most distortion in the resultant image. As such, the JSteg algorithm avoids these coefficients and higher frequency (AC) coefficients that have a value close to zero or one. In order to determine which coefficients correspond to AC coefficients, the transformed block is processed in what's called the 'zig-zag' pattern. An example of this process is shown below in Figure 3 [3]:

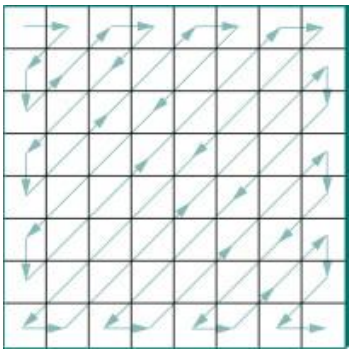

Figure 3: Zig-Zag Algorithm Traversing 8x8 Frequency Block

Once the DCT coefficients have been properly ordered by frequency content, the resultant matrix is then quantized in order to get integer values that are more reliable for data insertion. This is a lossy process, and causes some degradation of the image. For purposes of steganography, the lossy nature of quantization is a desirable one as it helps to further obfuscate the changes being applied in the frequency content. The JPEG encoding standard defines a standard quantization matrix for adjusting the luminance content of an image. This quantization matrix is applied to the DCT block element-wise, and the result of each division is rounded to the nearest integer. The standard quantization matrix is shown in Figure 4 [2]:

$$Q(u,v) = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

Figure 4: Standard Quantization Table for Luminance as per JPEG

Once the host data has been fully quantized into the frequency domain, the secret data can be embedded into the AC coefficients. After the embedding process, the host data is inversely transformed with the IDCT to bring the data back into the spatial domain. At this point, the new image is referred to as a *stego-image* and is ready to transport the embedded information.

III. Algorithm Implementation

The modified JSteg algorithm was implemented in the Python3 programming language. As is customary with most Python scripts, the implementation makes use of several external (not in the standard distribution) libraries, which are listed in Table 1:

Table 1: Required External Python Dependencies

Dependency Name	Dependency Homepage
OpenCV for Python	https://opencv.org/
Bitstring	https://bitstring.readthedocs.io/
Numpy	https://numpy.org/
Zig-Zag Implementation	https://github.com/.../simple-JPEG compression/.../zigzag.py

The main driver in the code is the OpenCV library, which handles all of the image processing and performs the actual DCT portions of the algorithm. The first step is to read in a cover image and ensure the image dimensions are an integer multiple of 8 length-wise and height-wise. If they aren't natively compliant, the image is resized so that it can be evenly split into 8x8 pixel blocks. After any necessary adjustments are made to the raw dimensions, the next step is to convert the color space from RGB to YCbCr, so the luminance (Y) layer is exposed. This is the layer intended for the data insertion as it is harder to notice the slight variations in luminosity [4]. The user chooses a character string to embed into the cover image, which is then packed into a binary representation and stored for embedding later. Once the cover image has been properly segmented, it is transformed block-wise with the Forward DCT-II mentioned previously, and quantized by the JPEG quantization matrix. Once the frequency coefficients have been quantized, the embedding process can begin. It is at this point where this particular implementation of the JSteg algorithm differs slightly from the original. The original JSteg algorithm ignores the DC

coefficient, and any AC coefficient with the values -1, 0, or +1. The modified algorithm take the restriction a step further and limits the valid coefficients to only those who are positive and have a value greater than one. This is to avoid editing negative value coefficients, which are more susceptible to corruption and also cause more distortion in the image. Once the data has been embedded into the frequency content, the image is inversely transformed back into the spatial domain and stitched together into the original dimensions. At this point the image is ready for transport and subsequent decoding. The decoding process has also been implemented, and is simply the reverse operation that was just stated.

IV. Performance Evaluation

Post-processing was performed in MATLAB in order to evaluate the relative performance of the modified algorithm. Figure 5 below shows an RGB histogram analysis of the data and features three subplots: the top plot is the raw host image that has only been resized to ensure 8x8 conformity. The middle plot shows the image after going through the transform/quantization procedure but with no embedded payload. Finally, the bottom image is the complete stego-image with the payload embedded into the frequency content.

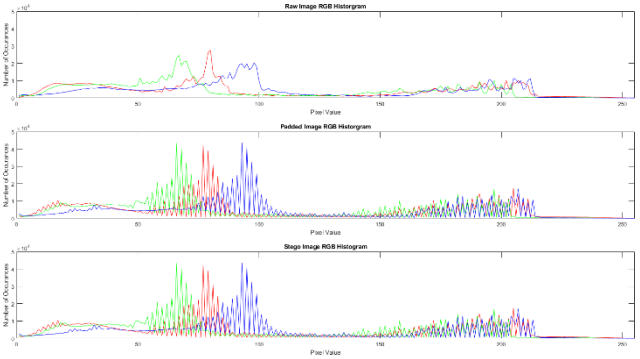

Figure 5: RGB Histogram Analysis of Raw Image (Resized), Padded Image (Transformed with no payload) and Stego Image with payload

The PNG images in Figure 6, Figure 7, and Figure 8 show the spatial-domain representations of each image mentioned in previously in Figure 5. Some standard performance metrics such as Peak Signal to Noise Ratio (PSNR), Mean Squared Error (MSE) and Structural Similarity Index (SSIM) are given below in Table 2:

Table 2: Standard Steganography Performance Metrics

	Stego Image vs. Raw Image	Stego Image vs. Padded Image
PSNR	37.0093	64.1749
MSE	12.9463	0.0249
SSIM	0.9778	0.9999634

Figure 6: Raw cover image with only 8x8 pixel resizing performed

Figure 7: Cover image after algorithm implementation but empty payload

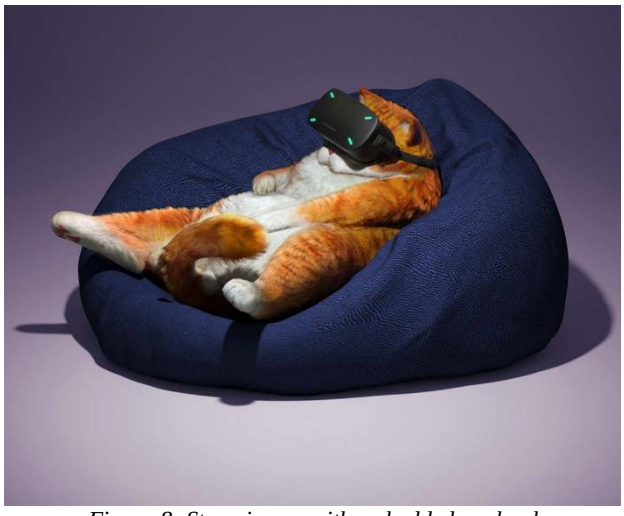

Figure 8: Stego image with embedded payload

V. Conclusion

The Python implementation of the modified JSteg algorithm was successful and a user can embed a message of their choice into a .PNG (or another lossless format) cover image and have it received by their intended recipient. JPEG images are not used in the algorithm, as they inherently perform an additional lossy DCT / quantization process which can destroy the embedded data. When looking at the post-analysis, the modified algorithm appears to perform within accepted steganography standards [2]. In future revisions of the Python implementation, one could incorporate an additional compression scheme to embed larger amounts of payload into the host image. While steganography may not be quite as well-known as traditional cryptography, it remains a useful tool for certain applications like digital watermarking and intellectual property (IP) management.

References

[1] Unknown, Author. 2016. "Navajo Code Talkers and the Unbreakable Code." Central Intelligence Agency. Central Intelligence Agency. https://www.cia.gov/news-information/featured-story-archive/2008-featured-story-archive/navajo-code-talkers/.

[2] Amiruzzaman, Md. 2011. "Steganographic Covert Communication Channels and Their Detection." Thesis. Kent State University

[3] Peterson, Larry L, Bruce S Davie, and Keith Jack. 2012. "Frequency Coefficient." Frequency Coefficient - an Overview | ScienceDirect Topics https://www.sciencedirect.com/topics/computer-science/frequency-coefficient

[4] Almohammad, Adel, Gheorghita Ghinea, and Robert M. Hierons. 2009. "JPEG Steganography: A Performance Evaluation of Quantization Tables." 2009 International Conference on Advanced Information Networking and Applications. doi:10.1109/aina.2009.67.

[5] Walia, Ekta & Jain, Payal & Navdeep,. (2010). An analysis of LSB & DCT based steganography. Global Journal of Computer Science and Technology. 10.

[6] Unknown, Author. 2020. "JPEG." *Wikipedia*. Wikimedia Foundation. https://en.wikipedia.org/wiki/JPEG.