An Implementation of SVD-Based Image Watermarking

Alper Tutum
Computer Engineering
Istanbul Technical University
Istanbul, Turkey
tutum21@itu.edu.tr

I. Introduction

Digital image watermarking plays an important role in protecting original images from illegal attacks. A watermarking scheme enables the holder of the watermark to prove that a digital image is theirs legally. The modern way of embedding a watermark to an image is to make it invisible, ciphered inside the image. This way the original image stays mostly the same and the watermark can't be seen by attackers directly. However a trade off between the robustness of the watermark and the quality of the original image is existent.

The digital watermarking scheme proposed by Chang et al. in 2005 aims to soften the loss of image quality by a singular value decomposition (SVD) based watermarking procedure [1]. This project is a Python implementation of the same algorithm with minor differences. The code includes 3 libraries:

- numpy as np
- matplotlib.pyplot as plt
- random

II. Singular Value Decomposition

Before going any further into the watermark embedding and extracting procedures, the details of the SVD implementation are explained step by step:

A. First Step

First, a symmetric matrix needs to be obtained in order to perform SVD. This is achieved by multiplying the transpose of the matrix with itself.

$$STS = np.matmul(ST, S)$$

B. Second Step

The eigenvalues and eigenvectors of STS are to be found and stored for later. In this implementation the QR method is used for approximating the eigenvalues and eigenvectors. The QR method simply performs QR decomposition on the given matrix and re-establishes the matrix from its decomposed form over and over again. After a set amount of iterations the diagonal of the final matrix gives the eigenvalues. The eigenvectors are calculated by multiplying the Q matrix from the decomposition for every iteration. With number of iterations set to 100, $theqr\_method()$ defined in the code achieves an almost identical result as the $np.linalg.eigh()$ method. In this case STS is given as an argument to $qr\_method()$ and the eigenvalues and eigenvectors of it are stored.

C. Third Step

The D and V^T matrices from the SVD equation $A = UDV^T$ are obtained in this step. The D matrix holds the singular values on its diagonals, which are the square root of the eigenvalues of STS. The right singular matrix V^T stores the eigenvectors of STS on its rows so it will be the transpose of the eigenvectors matrix obtained from $qr\_method(STS)$.

D. Fourth Step

Finally the left singular matrix U is to be found. The equation for each column of U is:

$$u_n = (A * v_n)/\sigma_n$$

v_n being the nth row of V^T and u_n being the nth column of U.

E. Inverse SVD

Inverse SVD is fairly simple to implement. The matrix multiplication of U, D and V^T gives the original matrix back. A function for this is defined in the code with the name $inverse\_svd()$ for later use.

III. The Watermarking Scheme Algorithm

The watermarking scheme proposed by Chang et al. [1] consists of two procedures. First an algorithm for embedding the watermark into the host image and a second algorithm for extracting the watermark from the watermarked image.

A. Watermark Embedding

The original image which is 512x512 pixels is partitioned into 8 by 8 blocks. Then SVD is performed on each block and the blocks are sorted in order of how many non-zero elements their singular values consist of. This is done by looking at the D matrix of each block. 1024 blocks, each corresponding to one pixel of the 32x32 watermark, are selected randomly from the most complex blocks.

The next step is to pick every selected block and look at the first column of their U matrix. The difference between the absolute values of the second and third element in this column will be observed and the selected U matrix will be changed accordingly. Let's call the difference:

$$r = u_{2,1} - u_{3,1}$$

<u>Case 0:</u>

If the corresponding watermark bit has value 0, r should be negative. If not the unmatched case will play out, else the matched case.

<u>Case 1:</u>

If the corresponding watermark bit has value 1, r should be positive. If not the unmatched case will play out, else the matched case.

Before that, the concept of having a threshold should be explained. In this particular watermarking scheme, it is desired for the relation to have a greater magnitude than a preset threshold. For this instance the threshold is set as 0.002.

<u>Unmatched Case:</u>

$$u_{2,1} = -||u_{2,1}| + (threshold - r)/2|$$
$$u_{3,1} = -||u_{3,1}| - (threshold - r)/2|$$

<u>Matched Case:</u>

$$u_{2,1} = -||u_{2,1}| - (threshold + r)/2|$$
$$u_{3,1} = -||u_{3,1}| + (threshold + r)/2|$$

After performing this embedding operation on all of the selected U matrices, perform $inverse\_svd()$ on all the decomposed blocks. Then reassemble the partitioned blocks of the host image and obtain the watermarked image.

B. Watermark Extraction

The owner of the watermark holds the seed of the randomiser which was used for selecting the blocks out of the most complex ones. The blocks at the same indices will be examined in the watermarked image. Just as in the previous algorithm, SVD will be performed on the necessary blocks and the sign of r will decide what the corresponding bit value of the watermark is.

If r is negative the corresponding pixel of the extracted watermark will be 1, else it will be 0. This procedure will be applied to every necessary block and the extracted watermark will be constructed from scratch.

C. Results

A grayscale baboon png with dimensions 512,512 was selected for testing. The image was partitioned into blocks of size 8 as mentioned before. Then the watermark was embedded into the image.

The result of the watermark embedding process was mostly successful. The watermarked image was resized to have dimensions of 512,512 by using a method of PIL library in python. This was required in order for the Peak Signal-to-Noise Ratio (PSNR) to work properly. PNSR was calculated as 43.5 dB which is very close to the result Chang et al. achieved [1]. The calculation was done with the method $peak\_signal\_noise\_ratio()$ from the skimage library

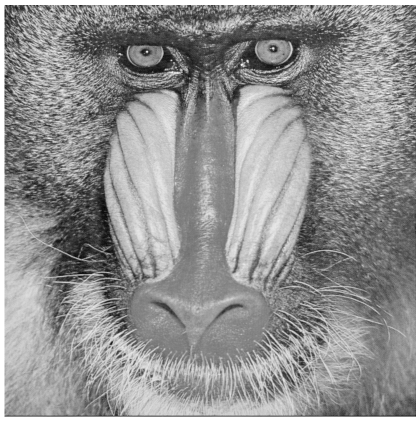
Fig. 1. The original host image [1]

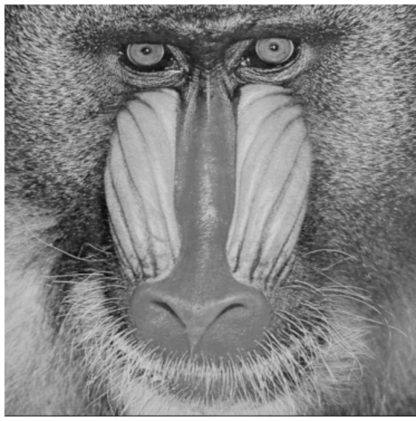
Fig. 2. The watermarked image

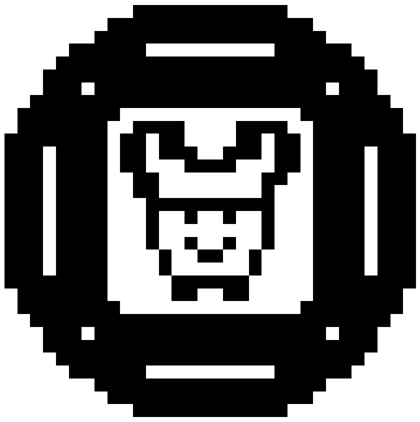
Fig. 3. Original watermark

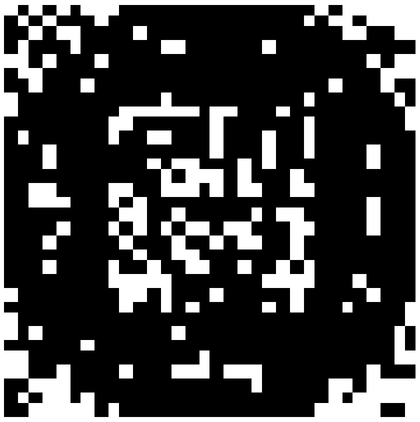

Fig. 4. Extracted watermark [1]

in python. The host image and watermarked image are shown in Figure 1 and Figure 2 respectively.

Then the watermark was extracted from the newly obtained image with the explained procedure. The original watermark is shown in Figure 3, and the extracted watermark is shown in Figure 4. The change in the watermark seems more dramatic than the change in the host image. The extracted watermark is barely recognizable but still resembles the original.

IV. Conclusion

The SVD-based digital image watermarking scheme proposed by Chang et al. was implemented and tested with an original watermark [1]. The results were reasonable and thus the efficiency of the scheme was proven.

References

[1] Chin-Chen Chang, Piyu Tsai, Chia-Chen Lin, “SVD-based digital image watermarking scheme” Pattern Recognition Letters, vol. 26, pp. 1577-1586, July 2005.