SVD-Based Watermarking Schemes

Ian McGovern, Adam Rohde, Annie Zhang

March 17, 2020

Abstract

This paper surveys techniques for digital image watermarking that employ singular value decomposition as a method for embedding watermarks imperceptibly in images. Multiple desirable qualities of digital watermarks are explained and their balance discussed. Three foundational papers are discussed in detail, implemented in Python, and experiments on these methods are conducted. We find that simpler methods tend to have better performance but at the cost of higher computational intensity. We discuss extensions to these schemes as well as practical applications in areas from medicine to copyright protection.

1 Introduction

As economic activity, medical records, media, and other aspects of modern life become increasingly digital, the need to protect ownership and verify legitimacy grows immensely. Digital watermarking plays a key role in facilitating this transition. Watermarking has numerous applications, from securing the intellectual property of images to creating verifiable medical records. This paper focuses on watermarking methods that involve singular value decomposition (SVD for short). SVD is a method for decomposing a matrix into simpler matrices that contain information about the fundamental properties of the matrix. This decomposition allows for watermarks to be strategically embedded into images.

Measures of Performance

The performance of SVD-based watermarking schemes is typically judged along four dimensions: imperceptibility, robustness, security, and capacity.

1. Imperceptibility: Imperceptibility describes the perceptual difference between the original image and the watermarked image. We prefer watermarks with little to no perceptual difference. That is, a watermarking scheme should not be obvious upon viewing the image. The image should not be noticeably destroyed or changed upon the addition of a watermark.

2. Security: Security describes how well the secret key is encrypted. A secure watermarking scheme should not allow for counterfeit watermarks and should provide sufficient evidence of the original owner who possesses both the original image and the watermarked image. It should not be possible for an individual to be able to produce a counterfeit original image and original watermark that can be proved to be authentic.

3. Robustness: Robustness refers to the scheme's ability to be resistant to changes. A watermarking scheme should not fail upon common transformations of images. This includes, but is not limited to, cropping, rescaling, compression, blurring, and rotation. If these transformations occur, the watermark scheme should still be able to produce the rightful owner of the image. Attack operators such as image shifting test the robustness of a watermark by altering rows or columns of pixels in the watermarked image.

4. Capacity: Capacity describes the volume of information that can be embedded into the original image. Watermarks with high capacity are capable of embedding a large amount of information while maintaining high imperceptibility. This dimension is useful in practical applications where capacity constraints make digital watermarking a difficult task.

Digital Images and SVD Review

Digital images are stored as arrays of pixel intensities. In particular grey-scale images are represented digitally by two dimensional arrays of values from 0 to 255, indicating black to white. Thus, we can view these as non-negative matrices and are able to perform linear algebra operations on them to understand the matrices underlying structure as well as to manipulate the matrices to alter the image. As such, we will refer to the original image and the original matrix somewhat interchangeably. Simple manipulations like multiplying such a matrix by a positive constant will alter the intensity of the image making all pixels lighter or darker. Carefully chosen individual elements or sets of elements of the matrix (pixels) can also be altered to make small changes to the image that allow for imperceptible embedding of watermarks. More complex operations can also be carried out. Of particular here is the singular value decomposition of a matrix. As mentioned above, this is a matrix factorization that produces matrices that capture important structure in the original matrix. This operation can be performed on any rectangular matrix. This is a generalization of the eigendecomposition of square matrices. SVD decomposes a matrix A into the factorization UDV^T, where U and V are orthogonal matrices with the left and right singular vectors and D is a diagonal matrix with the singular values along the diagonal. The singular values are the square roots of the eigenvalues of the covariance matrix, when the original matrix is centered. The singular vectors are related to the eigenvectors of the covariance matrix. This provides some insight into why this decomposition is of interest here. SVD is related to PCA and to covariance and, therefore, allows us to identify portions of the image that have larger variance in terms of pixel intensity and to make subtle alterations to the original image.

Overview

In this paper we review, implement, and run experiments on three foundational papers proposing SVD-based image watermarking techniques. We also explore more recent extensions and applications. The papers covered in detail and implemented are

1. **SVD Only** (Liu & Tan, 2002)
2. **SVD + DWT** (Lai & Tsai, 2010)
3. **SVD + Blocking + Randomness** (Chang, Tsai, & Lin, 2005)

These papers all allow for semantically meaningful watermarks but build in complexity starting from quite simple to relatively complex. However, we find the complexity of these methods does not necessarily lead to better performance. We also discuss the following papers and elucidate digital watermarking techniques in the context of attacks and practical applications.

1. **SVD + Two Note Scheme** (Chung, Yang, Huang, Wu, & Hsu, 2007)
2. **Watermarking Attack Operators** (Tao, Chongmin, Zain, & Abdalla, 2014)
3. **Practical Applications** (Zear, Singh, & Kumar, 2018) (Tao et al., 2014)

The papers and topics discussed here serve only as an introduction to the vast array of application areas and digital watermarking techniques, but they provide insight into the goals, advances, and difficulties present in modern digital watermarking.

2 SVD Only: Liu and Tan (2002)

2.1 Procedure

A basic SVD-based watermarking scheme can be performed on an original image A, first by performing a singular value decomposition on A to calculate $A = USV^T$. This decomposition is performed such that U and V are orthonormal matrices, and S is a diagonal matrix with the singular values (or the square root of the eigenvalues of AA^T) on the diagonal, arranged by size. Consider a watermark, represented as the matrix W, which is the same size as A. Another matrix S' is calculated as $S' = S + aW$, with a being a variable controlling the "strength" of the watermark, W. Another singular value decomposition is performed on $S' = U_w S_w V_w^T$, and then the watermarked image is represented as $A_w = US_wV^T$. This watermarked image will look, to the human eye, essentially identical to the original image, depending on the control parameter a. The process is, therefore, as follows:

$$\begin{aligned} A &\to USV^T \\ S + aW &\to S' \\ S' &\to U_w S_w V_w^T \\ US_wV^T &\to A_w \end{aligned} \tag{1}$$

For watermark extraction, the watermark embedding process is performed in reverse. Given S, U_w, a, and V_w from the embedding procedure, and a watermarked (and potentially distorted image) A_w^*, perform an SVD decomposition to get $A_w^* = U^* S_w^* V^{*T}$. Then calculate $U_w S_w^* V_w^T = D^*$, and extract the (potentially distorted) watermark, $W^* = \frac{1}{a}(D^* - S)$. The similarity between $W*$ and W can be calculated and use to prove the ownership of the image. The process is therefore as follows:

$$\begin{aligned} A_w^* &\to U^* S_w^* V^{*T} \\ U_w S_w^* V_w^T &\to D^* \\ \frac{1}{a}(D^* - S) &\to W^* \end{aligned} \tag{2}$$

2.2 Justification

Suppose an individual attempts to claim that they have the original image and watermark (denoted as A_F and W_F). In order for this counterfeit watermark to be verified, it would need to satisfy both (1) and (2). Let $E(A, W)$ correspond to a watermarking embedding procedure, for an original image, A, and watermark, W. Liu and Tan proved the following theorem:

Theorem 1: Using the watermarking scheme given in (1):

$$A_w = E(A, W) \iff S + aW = U_w S_w V_w^T$$

$$A_w = E(A_F, W_F) \iff S_F + aW_F = U_{F,w} S_w V_{F,w}^T$$

such that $U_{F,w}$ and $V_{F,w}$ are orthonormal matrices. Liu and Tan further argued that in order for this procedure to be secure, the following to equations cannot be satisfied at the same time:

$$S + aW = U_w S_w V_w^T \tag{3}$$

$$S_F + aW_F = U_{F,w} S_w V_{F,w}^T \tag{4}$$

Proposition 1: With regards to (3), the following conditions hold:
a) Given S_w, it's impossible to find S and W due to the non-uniqueness of SVD decompositions.
b) Given S_w and S, W is impossible to obtain due to the number of constraints within the problem.
c) Given S_w and W, it is computationally difficult to calculate S.

These conditions allow for security with regards to the watermarking scheme, since knowing S_w and potentially additional information is actually not useful in calculating the complete watermarking scheme. For (4), the first two parts of Proposition 1 also hold for the corresponding matrices in (4). Critically, the following proposition can be shown:

Proposition 2: Given S_w and S_F, it is difficult to find W_F if restrictions are put on the watermarks used in this scheme.

Clearly, a solution for W_F can be found by

$$W_F = \frac{1}{a}(U_{F,w} S_w V_{F,w}^T - S_F).$$

However, if it is required that the watermarked image satisfy a constraint, this solution is not necessarily viable, and therefore a fake watermark verification process can not be done easily. An example of a constraint would be that the watermark must be a recognizable image, for instance.

3 SVD + DWT: Lai and Tsai (2010)

Discrete Wavelet Transformation (or DWT) is a process in which an image is decomposed into subbands. In this case, a one-level DWT is performed to decompose the image into subbands labeled as LL, LH, HL, and HH. The LL subband corresponds to the low-frequency part of the image, which the human eye is more sensitive to. Therefore, the watermarking process can be done on the LH and HL subbands, which are less noticeable to the human eye. In addition, since preforming and SVD is a computationally intense process, since the LH and HL subbands are smaller than the original image, computation can be done more efficiently. The process proposed by Lai and Tsai is essentially the process proposed by Liu and Tan applied to the LH and HL subbands. Figure 2 shows the original image, along with the LH and HL subbands after DWT.

Procedure for DWT Expansion

The watermarking procedure is as follows. Given an original image, A, and watermark, W:

1. Decompose A using Haar DWT into the subbands LL,LH, HL, and HH.
2. Perform SVD on the LH and HL subbands to get $A^k = U^k S^k V^{kT}$ for $k = 1, 2$.
3. Split the watermark into W^1 and W^2 such that $W = W^1 + W^2$.
4. Take the matrix of singular values for each subbands, S^k, and add a multiple of the watermark, to get $S'^k = S^k + aW^k$, where a is some scaling factor.
5. Perform SVD on S'^k to get $S'^k = U_W^k S_W^k V_W^{kT}$.
6. Calculate the watermarked subband as $A_W^k = U^k W_W^k V^{kT}$.
7. Perform a reverse DWT transformation, using the original LL and HH subbands, and replacing the LH and HL subbands with the calculated A_W^k subbands to get the watermarked image, A_W.

Now, for the watermark extraction, the procedure is as follows:

1. Decompose A_W using Haar DWT into the subbands LL, LH, HL and HH.
2. Apply SVD to the decomposed LH and HL subbands, denoted as A_W^k for $k = 1, 2$ to get $A_W^{*k} = U^{*k} S_W^{*k} V^{*kT}$.
3. Calculate $D^{*k} = U_W^k S_W^{*k} V_W^{kt}$, using U_W^k and V_W^{kt} from the watermark embedding procedure.
4. Extract $W^{*k} = \frac{1}{a}(D^{k*} - S^k)$.
5. Calculate $W^* = W^{*1} + W^{*2}$.

4 SVD + Blocking + Randomness: Chang et al. (2005)

Earlier investigations (e.g., Liu and Tan (2002)) showed the power of singular value decomposition as a method for image watermarking; however Chang et al. (2005) sought to improve upon these by synthesizing and extending existing methods. Their proposal forgoes some of the simplicity of methods like Liu and Tan (2002) in order to introduce pseudo-randomness in an block-oriented watermark embedding procedure that has the advantage of altering the original image very little while being more difficult to remove and robust to alterations. The watermark embedding and extraction procedures of Chang et al. (2005) proceed as follows.

4.1 Embedding Procedure

1. The original image is divided into square blocks. Choose block size to be small relative to size of original image. (For example, in our implementation discussed further below, we split the 1100 x 1100 original image into 10 x 10 blocks.)
2. Each of the blocks are decomposed via singular value decomposition into matrices U, S, and $V^\top$.
3. The complexity of each block, defined as the number of non-zero singular values (diagonal elements of S), is determined.

   (Steps 1 - 3 are focused on identifying the portions of the image (blocks) that have the largest variances across pixel intensities. The idea being to embed the watermark in the more heterogeneous portions of the image so as to not noticeably disrupt homogeneous or smooth portions of the original image. Singular value decomposition provides a way for us to evaluate this, since the singular values are generalizations of eigenvalues and are related to the variance across pixel intensities. So selecting blocks with more non-zero singular values is akin to selecting the less smooth blocks.)
4. A random sample (without replacement) of blocks with the maximum possible complexity (defined above) are chosen. The size of the random sample is equal to the number of pixels in the watermark image. These are the blocks in which the watermark will be embedded. Note that the watermark should be a binary image and should be small relative to the original image.

   (Here randomness is introduced to increase the security of the watermark.)

5. We associate a single pixel of the watermark with each of the selected blocks by examining two coefficients in the first column of U for the block.
    (a) Chang et al. (2005) chose to compare the second and third elements of the first column on U for each of the selected blocks.
    (b) The idea is to associate a single pixel from the watermark to the absolute value difference in these coefficients (choose this difference to be $d = |U_{2,1}| - |U_{3,1}|$).
    (c) If we are trying to associate the block with a watermark pixel with value 1, we want to ensure that the difference in these coefficients is positive.
    (d) If we are trying to associate the block with a watermark pixel with value 0, we want to ensure that the difference in these coefficients is negative.
    (e) So we make small adjustments to these coefficients to ensure that the absolute value difference is of the correct sign for the watermark pixel that we're embedding with that block. Note that we also add a slight buffer (that is, make already positive differences slightly more positive, already negative differences slightly more negative, and switch the sign of the differences when appropriate).
    (f) These adjustments to the coefficients are done such that the correct sign difference is achieved but without noticeable altering the pixel values of the original image.
6. Then the watermarked blocks are reconstructed by undoing the singular value decomposition.
7. Finally, the watermarked blocks are reassembled into the full watermarked image.

4.2 Extracting Procedure

1. The watermarked image is divided into the same sized blocks as in the embedding procedure.
2. The blocks are decomposed via singular value decomposition into matrices U, S, and $V^\top$.
3. The relevant coefficients in the first column of U of the chosen blocks (this information and the watermark are retained by the owner of the image) are then examined.
4. A recovered watermark is constructed based on the absolute differences in the relevant coefficients. If there is a positive difference, then we use a pixel value of 1. If it is negative then we use 0.
5. Once all relevant blocks have been examined, we assemble the recovered watermark.

5 Implementations and Experimental Results

Figures 1, 2, and 3 show the original images, the watermarks, the watermarked images, and the absolute difference images for the three watermarking schemes discussed above. These figures clearly show that all three of the methods discussed so far perform well on imperceptibility, since it's not possible to easily tell the difference between the original and the watermarked images. We've also seen that the three methods perform well on security in that they incorporate semantically meaningful watermarks as well as improvements like blocking, randomization, and DWT. In what follows, we will test the robustness of these watermarking schemes by manipulating the watermarked images and then extracting the potentially corrupted watermark. The correlation between the corrupted watermark and the original watermark is calculated for each method and distortion. If this correlation is sufficiently above zero, it means that the true watermark is identifiable despite the manipulation to the watermarked image. When the watermark is robust to such manipulations, individuals cannot get rid of the watermark easily through common image manipulations, which is preferable in a watermarking scheme.

For our implementations we used an original image size 1100×1100 and two different watermarks: one sized size 1100×1100 and one size 32×32.[1] The watermark used depends on the watermarking method. Figures 4, 5, and 6 show the altered images, along with the watermarks that were extracted from each altered image, and graphs that show the correlation of the extracted watermarks with random noise images, with the last point the correlation between the extracted watermark and the original watermark. The accuracy of the extraction procedure on the unaltered watermarked image is shown in the first row of the figures. Here we see that we were able to exactly recover the watermark using the extraction procedures. The correlation between the watermark and the extracted

[1] *Image Of Dog 16* (n.d.), *Retro Mushroom Super 3 Icon* (n.d.), *Snow to hit parts of Colorado this weekend* (n.d.)

watermark is 1. To test the robustness of the procedures, we alter the watermarked image by a variety of methods and then test how well we are able to recover the watermark from the altered image. The alterations that we apply are adding Gaussian noise, blurring, compression, rotation, and cropping.

5.1 SVD Only

Liu and Tan implemented their watermarking scheme and tested to insure that their watermarks were still identifiable under image distortions to test the robustness of their scheme. We also implemented their watermarking scheme ourselves and tested the robustness. See Figure 4. Adding Gaussian noise, as seen in the second row, only slightly decreased correlation. The same can be said for blurring and compressing the image. More dramatic transformations, however, did affect the correlation between the watermark and extracted watermarks. Rotating the image decreased correlation to below .4, and cropping the image reduced correlation to approximately .1. However, both of these still have a correlation that is noticeably different from a random watermark.

5.2 SVD + DWT

The DWT-based watermarking scheme was also tested for robustness using the same tests as Liu and Tan. As seen in Figure 5, the results were slightly different in comparison to Liu and Tan's watermarking scheme. While compression had a similarly high correlation, the cropped image also have a high correlation, unlike in Liu and Tan's implementation, where cropping the image had the lowest correlation. Blurring did not preform very well either compared to Liu and Tan's implementation, having a correlation coefficient below .4. Rotation, however, similarly had a low correlation coefficient of slightly over .2. Finally, adding Gaussian noise preformed well, but not as well as it did under Liu and Tan's implementation.

5.3 SVD + Blocking + Randomness

We also implemented and tested the methodology of Chang Tsai and Lin (2005). See Figure 6. Visual inspection of the original image and the watermarked image suggests that the procedure achieves good levels of imperceptibility. However, the procedure offers a tuning parameter for when higher levels of fidelity to the original image are required or if lower levels are acceptable. There is an inherent trade off between the level of security and the level of imperceptibility. We see that, while visually many of the recovered watermarks are not very close to the original watermark, the recovered watermarks tend to be easily identifiable relative to noise in terms of correlation with the original watermark. The main exception is rotation. This makes sense, as the extraction procedure looks at the blocks that were watermarked in the embedding procedure; but when the image has been rotated, these have changed location. However, since rotation can be easily undone, this isn't necessarily a fatal flaw in this methodology. In summary, this procedure does well in resisting deletion in our experiments but it does not seem to preform as well as the simpler methods the authors sought to improve upon.

6 Extension: SVD + Two Note Scheme: Chung et al. (2007)

Previously, researchers proved that modifying the coefficients in one row of n x n matrix U will distort n pixels of matrix A. In this paper the authors propose a slight variation by modifying two notes. Like before, we perform SVD; for the U component, modifying the coefficients in column vector will cause less visible distortion than modifying the coefficients in row vector. The authors of this paper propose that modifying the V^T component will improve robustness without creating additional visual distortion. In this paper, Chung et al. describe that modifying coefficients in a column vector will disperse the distortion to n x $(n-1)$ pixels of A. This will decrease visibility of the distortion compared to the row technique alone. By the same token, modifying coefficients in a row vector of V^T is advantageous relative to modifying its column vectors. The authors then test the performance of implementing the column technique for matrix U compared to modifying both matrix U and V. The authors describe the following embedding and extraction procedures.
Embedding Procedure:

1. Partition the host image into blocks.
2. Perform SVD transformation.
3. Let Z = D(1,1) mod Q. Extract the largest coefficient D(1, 1) from each D component.

4. For bit values of 0, modify D(1,1) if Z $< \frac{3}{4}$Q. For bit values of 1, modify D(1,1) if Z $< \frac{1}{4}$Q.
5. Perform Inverse SVD (ISVD) to obtain the reconstructed watermark.

Extracting Procedure:

1. Partition the watermarked image into blocks.
2. Perform SVD transformation.
3. Extract the largest coefficient $D^{'}(1,1)$.
4. Let Z = D(1,1) mod Q. If Z $< \frac{1}{2}$Q, the extracted watermark has bit value 0 and 1 otherwise.

The performance was measured using Peak Signal to Noise Ratio (PSNR) and robustness, which are inversely related. For the image with embedding the watermark into U only, the PSNR was 43.52 compared to PSNR of 42.69 when watermarked into both U and V (see Figure 7). The image watermarked into both components has higher robustness.

The experiment results show that modifying the column vectors of U improve imperceptibility. Furthermore, embedding the watermark in both U and V increases robustness and capacity. This result shows that the Two Note scheme achieves better imperceptibility, robustness, and capacity compared to techniques that modify only the U component, such as ordinary SVD or SVD with blocking and randomness.

7 Watermarking Attack Operators: Tao et al. (2014)

In their paper Robust Image Watermarking Theories and Techniques: A Review, Tao et al. provide a cohesive look at digital watermarking applications and the utility of watermarking in the context of other existing techniques. This paper explains different types of attacks on watermarked images and when to use them. The authors provide perspective on different attack operators in digital watermarking. There are five categories of attacks: Removal, Geometric, Cryptographic, Protocol, and Image Shifting.

For removal attacks, the intent is to remove watermarks from the host image. This classification includes lossy compression, image denoising, demodulation, quantization, averaging, and collusion attacks. Attacks involving denoising, also understood as filtering, typically use maximum likelihood, a minimax criterion or a minimum mean square error. Remodulation uses the same filtering and adds stationary Gaussian noise. Averaging is a common technique in collusion attacks. Numerous examples of the same data are used; we take a small sample of each data set and reconstruct a new attack data set. For geometric attacks, the intent is to distort watermark through temporal or spatial transformations. Cryptographic attacks intend to discover secret information using exhaustive searches. To prevent this type of attack, we need secret keys of a safe length that are not able to be easily guessed. Protocol attacks are common in copyright protection and test the noninvertibility of watermarks. Since watermarks are non-invertible, this type of attack attempts to detect a watermark in non-watermarked images. Finally, image shifting attacks change rows or columns of pixels in the watermarked image. These attacks test the robustness of the watermark.

8 Practical Applications

8.1 Medical Watermarking Applications: Zear et al. (2018)

In recent years, digital watermarking schemes have gained traction in the medical field. Popular schemes include Discrete Wavelet Transforms (DWT), Discrete Cosine Transforms (DCT), and Singular Value Decomposition (SVD). In their paper A proposed secure multiple watermarking technique based on DWT, DCT, and SVD for application in medicine, Zear et al. discuss ways to increase robustness and security of watermarking schemes for identity authentication. They propose a technique to include a doctor signature, doctor identification code, and patient diagnosis into a watermark.

Furthermore, the authors note methodical differences between image watermarks like Lump images and text watermarks such as signatures and patient symptoms. Watermarks containing important information and requiring more robustness are embedded in higher level DWT subbands. Text watermarks, such as a doctor signature and symptom information, are embedded in LH1, LH2, and LL3 subbands. In comparison, image watermarks like a Lump image are embedded on the low high frequency band of the first level. The image watermark embedding algorithm is below.

1. $A_c = U_c S_c V_c^T$
2. $A_w = U_w S_W V_W^T$
3. $S_{wat} = S_c + k * S_w$
4. $A_{wat} = U_c * S_{wat} * V_c^T$

In healthcare, digital watermarks are used to protect the confidentiality of patient data by keeping it from being accessed by unauthorized users. This application, known as Medical Image Watermarking (MIW), is an integral part of preventing costly security problems such as medical identity theft. The authors introduce a technique combining previously implemented watermarking schemes DWT, DCT, and SVD with Back Propagation Neural Network (BPNN). Zear et al. explain that SVD based watermarking schemes is their imperceptibility in the watermarked image. However, SVD suffers from high false positives, when a watermark is detected when in fact the image does not contain a watermark. The authors suggest Shuffled SVD (SSVD) as an alternative to SVD to alleviate high false positive rate.

The procedure for embedding medical watermarks is straightforward. First the authors apply DWT transformation to the original cover image, decomposing it into subbands. The authors apply DCT to the chosen subband and SVD to each transformed DCT coefficient, giving us U, S, and V matrices in step 1. The watermark is encrypted and the procedure is repeated for the watermark, which the authors use interchangeably with the Lump image. The transformed Lump image is shown in step 2. In the next step, the singular values of selected subbands are modified for both the cover image and the Lump image. Finally, they apply Inverse SVD (IVSD) to obtain DCT coefficients in step 4. The DCT coefficients are used to obtain the watermarked image using Inverse DCT and Inverse DWT procedures. The Lump example is shown in Figure 8. The authors then test attacks on the watermarked image to assess robustness. The medical extraction algorithm for image watermarks is set up below.

1. $A_c = U_c S_c V_c^T$
2. $A_w = U_w S_w V_w^T$
3. $A_{wat} = U_{wat} S_{wat} V_{wat}^T$
4. $S_w^* = (S_{wat} - S_c)/k$
5. $A_{ew} = U_w * S_w^* * V_w^T$

Like the embedding algorithm, we begin by applying the DWT transform to the cover image and DCT to selected subbands. The SVD decomposition of DCT coefficients gives us the equation in step 1. This procedure is repeated for the watermark and watermarked image (steps 2 and 3). We obtain singular values of the watermark by applying the decomposition equation on step 4. Finally, we apply ISVD and IDCT to obtain the extracted watermark (step 5). BPNN is optionally applied to increase robustness.

The authors additionally propose a way to evaluate quality of both watermarked images and watermark extractions. They devise a method to quantify visual quality, a subjective measure for the watermarked image as well as extracted watermark. Using the proposed method, six participants vote for a level of gain factor assigned to a numerical value which measures the visual quality of the image. After they test the algorithm, the authors measure performance in the following dimensions: imperceptibility, robustness, and capacity.

Zear et al. conclude that the combination of DWT, DCT, and SVD outperforms performance of each technique individually. The authors also show that BPNN training can improve the quality of the watermark image, as illustrated in Figure 9. The first example does not use BPNN while the second image extracts the watermark with BPNN training. We observe higher resolution and clarity in the image on the right which uses BPNN. They also note that embedding multiple watermarks in one image decreases the storage and bandwidth requirements, both important considerations in the medical field. Finally, they caution that while using the combined algorithm improves robustness, imperceptibility, capacity, and security, the advantages come at the cost of simplicity by increasing the complexity of applying watermarks.

8.2 Further Applications: Tao et al. (2014)

The paper Robust Image Watermarking Theories and Techniques: A Review, Tao et al. provide a detailed background that researchers who are new to digital watermarking will find useful. The authors describe the context of digital watermarking and practical applications of the technique, including copyright protection, fingerprinting, and copy control.

With the rise of distributed computer networks in recent years, the use of digital watermarking has increased exponentially. Tao et al. explain the difference between digital and traditional watermarks, a helpful clarification for researchers who are unfamiliar with watermarking. We found that much of existing watermarking literature is written from a technical perspective; this paper clarifies the technical nuances, explaining how watermarking frameworks work and what criteria scientists use to assess quality of watermarks. As we have seen in watermarking literature, this paper describes the four evaluation dimensions: imperceptibility, robustness, capacity, and security. The authors define each dimension clearly and the rationale for using each one. Tao et al. also explain the false positive problem which occurs when a watermark is detected for images that do not actually contain a watermark.

Tao et al. additionally offer some intuition on techniques for decomposing images. As we have seen before, Singular Value Decomposition (SVD) provides a robust method for watermarking schemes. Tao et al. compare SVD techniques with Discrete Wavelet Transform (DWT), noting that DWT is advantageous in its irregular transformation distribution and lower computational cost with O(n) complexity. We set up the procedures for encoding and decoding watermarks below. The formula provides intuition regarding the process by which digital watermarks are applied to images.

$$CW = E(CO, W, K)$$

Here we have the watermarked image (CW) on the left and encoding algorithm (E), original image (CO), watermark (W), security key (K). The watermarked image is generated as a function of the image, watermark, and key. The authors explain that a robust watermark can be considered as a CW that is able to undergo severe distortions and remain unchanged. Next, we set up the paper's algorithm for decoding. In this formula we have the detected watermark (W') on the left as a function of the potentially watermarked image (CW) and the key (K). The decoder e(.) loads CW and extracts the watermark hidden in the image.

$$W' = e(CW, K, ...)$$

These algorithms provide useful intuition on how digital watermarking works. One area the paper falls short is it oversimplifies the process and lacks extensions beyond initial algorithm setups. We recommend that future work be done to weave the basic intuition with more complex watermarking methods used by researchers.

A strength of this paper is its review of practical applications of digital watermarks. In this section, the authors begin with digital watermarking for copyright protection. One of the earliest uses of watermarking, copyright protection refers to watermarked digital content which include information about the copyright owner. This technique is used by popular software programs like Adobe Photoshop and commercial software such as the Digimarc ImageBridge Solution. The watermark carries information about the owner whom the user can contact to request permission to use the copyrighted image. Next the authors discuss watermarking applications in fingerprinting. For fingerprinting, watermarks must be particularly resistant to attacks to prevent identity theft. In this case, the watermark contains an embedded ID number. The authors emphasize that embedding only one ID number in the host file enables the watermarked image to be resistant to collusion attack. Finally, watermarks are often used for copy control. This application uses a technique to detect when a music or video clip contains a fragile watermark. If detected, the media player will deny the user from playing the file. This method also prevents unauthorized users from illegally making copies of copyrighted media.

9 Discussion

In conclusion, digital watermarking is an essential tool that allows people in academia, technology, entertainment and numerous other fields to verify the original owner of a trademarked image. From this paper we found that SVD provides an excellent basis for digital watermarking. This technique allows for a protected image to be represented as a matrix, a property that gives us the ability to embed watermarks without obvious visual distortion. We implemented three foundational papers and found that simpler methods perform better. However, these schemes are also computationally expensive. We found that later improvements, such as the Two Note technique, improve traditional schemes and create watermarks that are more robust. Attack operators give us a method to test robustness and noninvertibility of the embedded watermark. Finally, digital watermarking has many practical applications. They are particularly useful in healthcare and copyright protection, where embedding large quantities of data in a watermark increases storage capacity. All in all, SVD-based watermarking schemes have many desirable properties and give us a system to embed watermarks that have high imperceptibility, security, robustness, and capacity.

References

Chang, C., Tsai, P., & Lin, C. (2005). Svd-based digital image watermarking scheme. *Pattern Recognition Letters*, *26*, 1577-1586.

Chung, K.-L., Yang, W.-N., Huang, Y.-H., Wu, S.-T., & Hsu, Y.-C. (2007, 05). On svd-based watermarking algorithm. *Applied Mathematics and Computation*, *188*, 54-57. doi: 10.1016/j.amc.2006.09.117

Image of dog 16. (n.d.). Retrieved 2020-03-16, from `https://webcomicms.net/clipart-9402765-image-dog`

Lai, C., & Tsai, C. (2010). Digital image watermarking using discrete wavelet transform and singular value decomposition. *IEEE Transactions on Instrumentation and Measurement*, *59*(11), 3060-3063.

Liu, R., & Tan, T. (2002). An svd-based watermarking scheme for protecting rightful ownership. *IEEE Transactions on Multimedia*, *4*(1), 121-128.

Retro mushroom super 3 icon. (n.d.). Retrieved 2020-03-16, from `http://www.iconarchive.com/show/super-mario-icons-by-ph03nyx/Retro-Mushroom-Super-3-icon.html`

Snow to hit parts of colorado this weekend. (n.d.). Retrieved 2020-03-16, from `https://www.outtherecolorado.com/snow-to-hit-parts-of-colorado-this-weekend/`

Tao, H., Chongmin, L., Zain, J. M., & Abdalla, A. N. (2014). Robust image watermarking theories and techniques: A review. *Journal of Applied Research and Technology*, *12*(1), 122 - 138. doi: https://doi.org/10.1016/S1665-6423(14)71612-8

Zear, A., Singh, A., & Kumar, P. (2018). A proposed secure multiple watermarking technique based on dwt, dct and svd for application in medicine. *Multimed Tools Appl*, *77*, 4863–4882. doi: https://doi.org/10.1007/s11042-016-3862-8

Figures

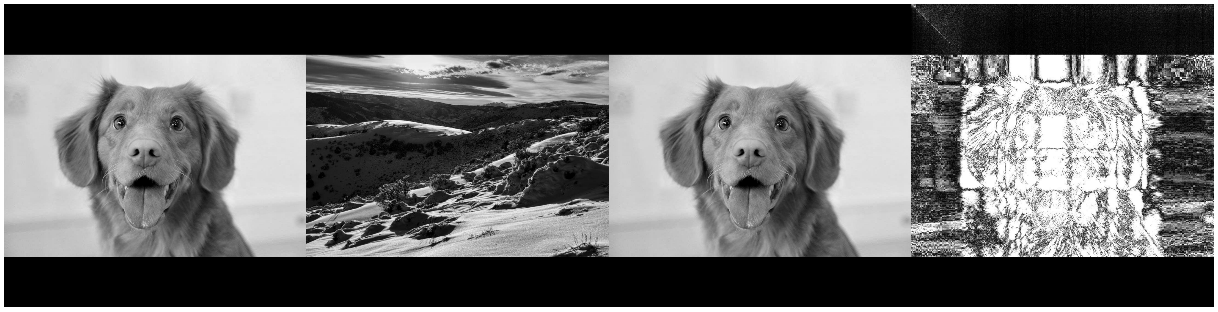
Figure 1: SVD Only Watermarking Process. From left to right: Original Image (1100 x 1100); Watermark (1100 x 1100); Watermarked Image; Absolute Difference Image (intensified 100x)

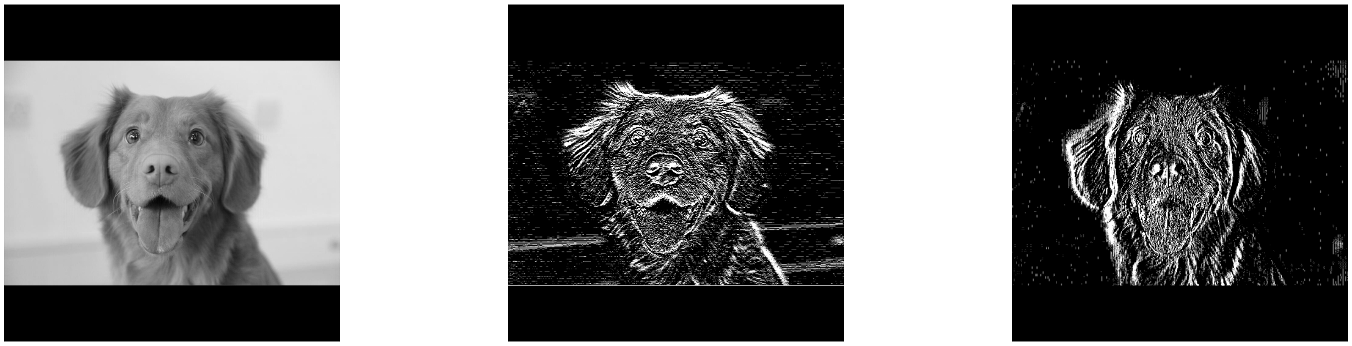
Figure 2: SVD + Discrete Wavelet Transformation. From left to right: Original Image, LH Subband, and HL Subband.

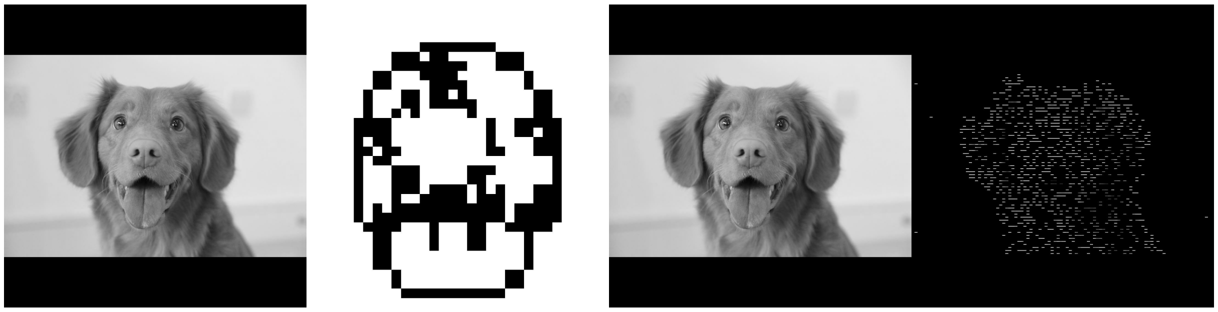
Figure 3: SVD + Blocking + Randomness Watermarking Process. From left to right: Original Image (1100 x 1100); Watermark (32 x 32); Watermarked Image; Absolute Difference Image (intensified 100x).

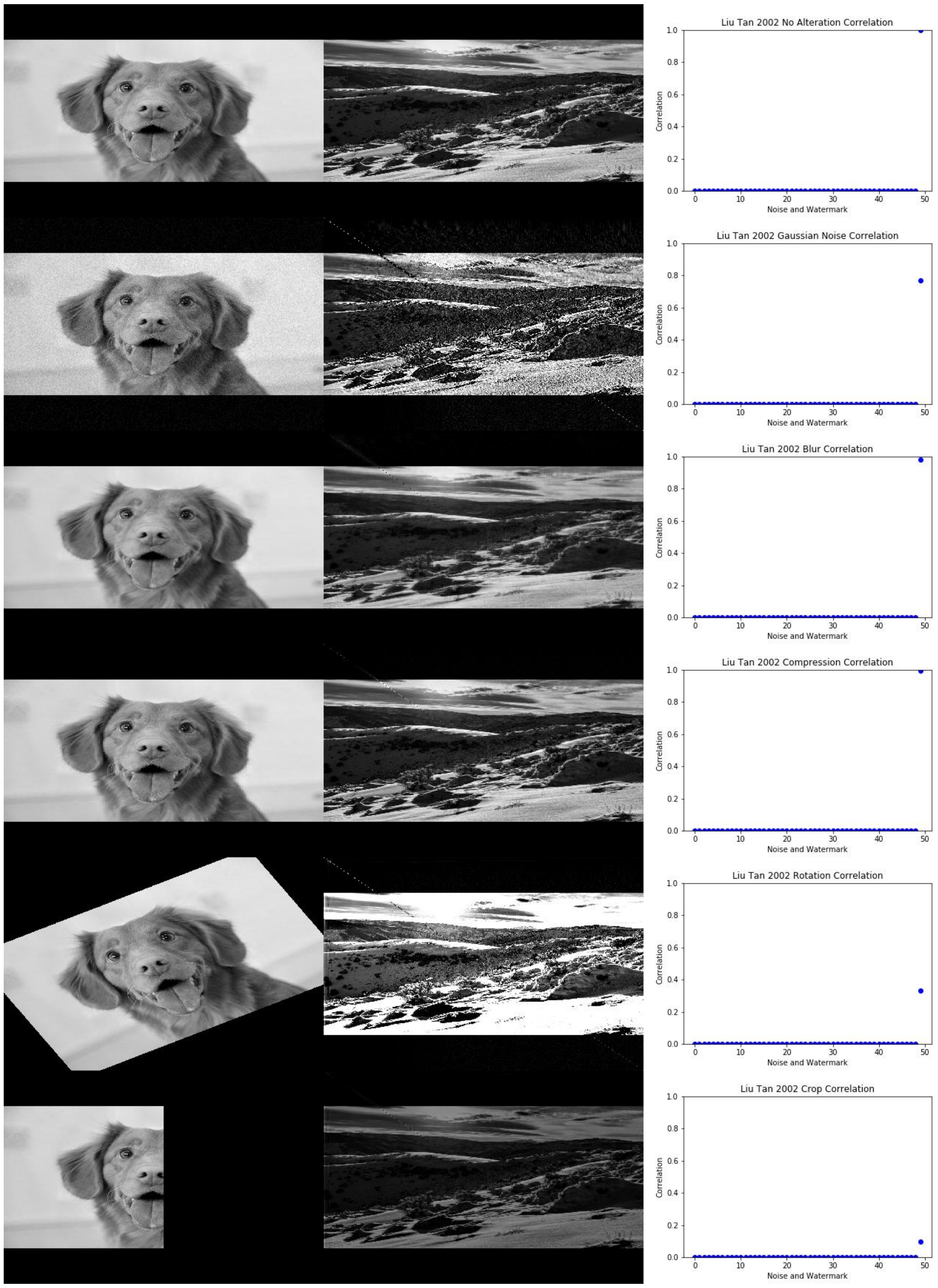

Figure 4: SVD Only Experiments. First column is altered watermarked images, second column is recovered watermarks, and third column is correlation between recovered watermark and noise / original watermark (correlation between recovered and original watermarks is last point in charts). Alterations to watermarked image are [1] No Alteration, [2] Gaussian Noise, [3] Blurring, [4] Compression, [5] Rotation, and [6] Cropping.

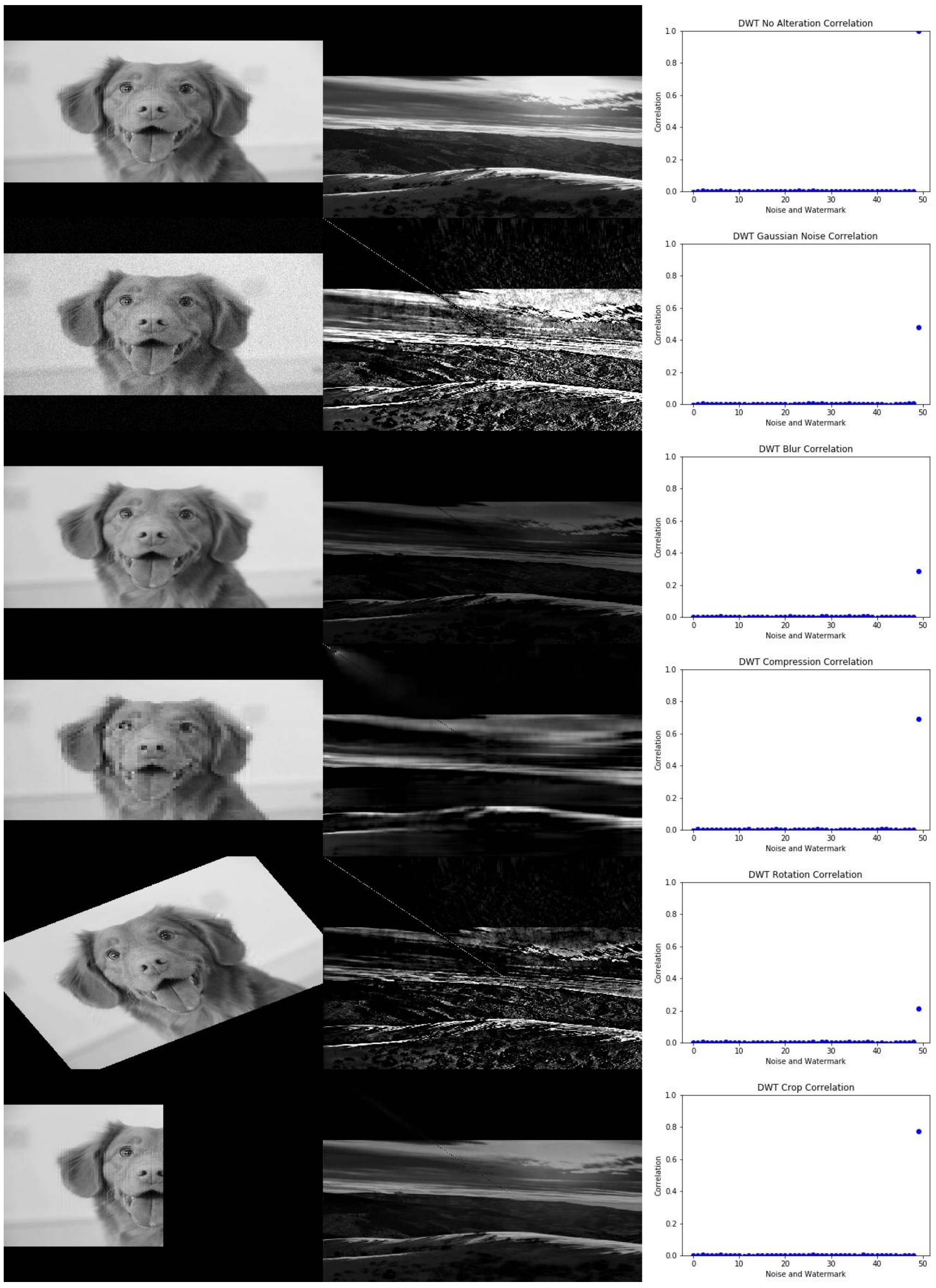

Figure 5: SVD + DWT Experiments. First column is altered watermarked images, second column is recovered watermarks, and third column is correlation between recovered watermark and noise / original watermark (correlation between recovered and original watermarks is last point in charts). Alterations to watermarked image are [1] No Alteration, [2] Gaussian Noise, [3] Blurring, [4] Compression, [5] Rotation, and [6] Cropping.

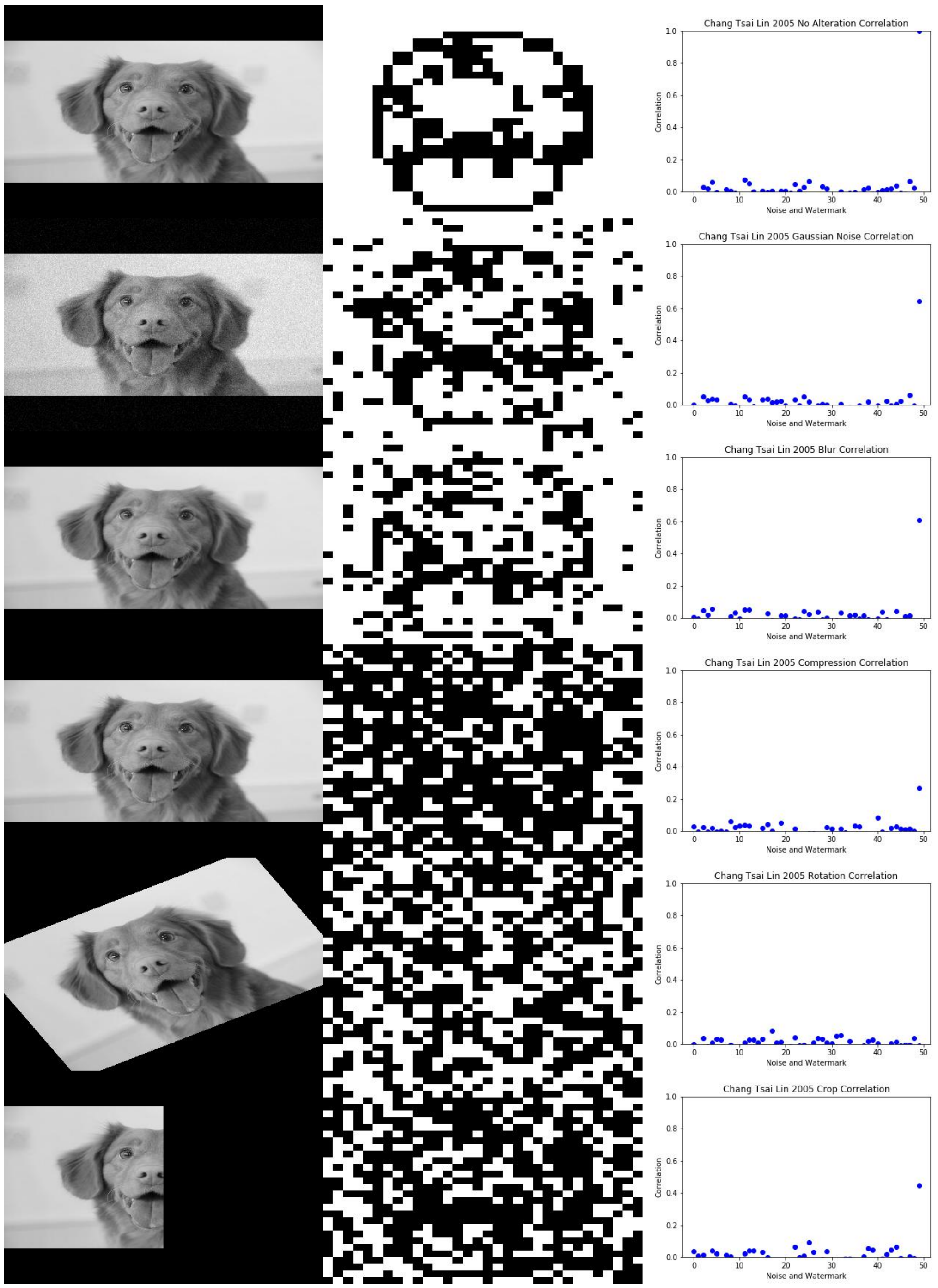

Figure 6: SVD + Blocking + Randomness Experiments. First column is altered watermarked images, second column is recovered watermarks, and third column is correlation between recovered watermark and noise / original watermark (correlation between recovered and original watermarks is last point in charts). Alterations to watermarked image are [1] No Alteration, [2] Gaussian Noise, [3] Blurring, [4] Compression, [5] Rotation, and [6] Cropping.

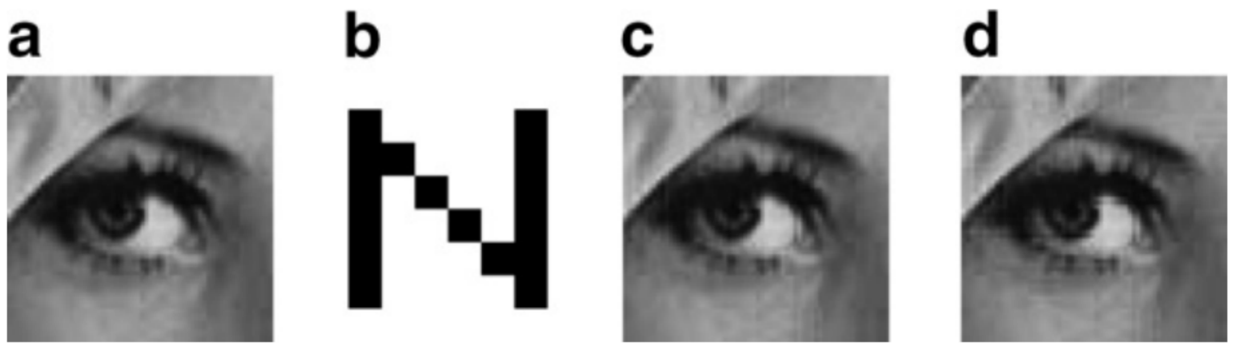

Figure 7: *Two Note Example. [a] Cover Image [b] Watermark [c] Embedding [b] into U component only [d] Embedding (b) into both U and V^T.*

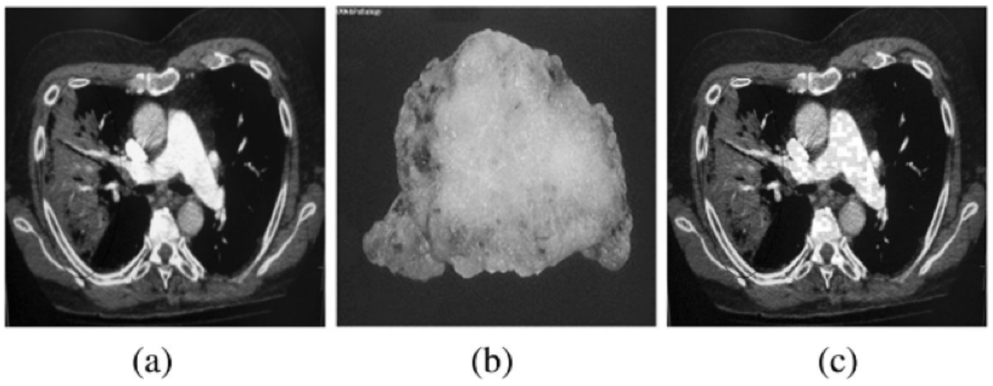

Figure 8: *Lump Watermark Example. (a) Cover Image (b) Lump Watermark (c) Watermarked Image*

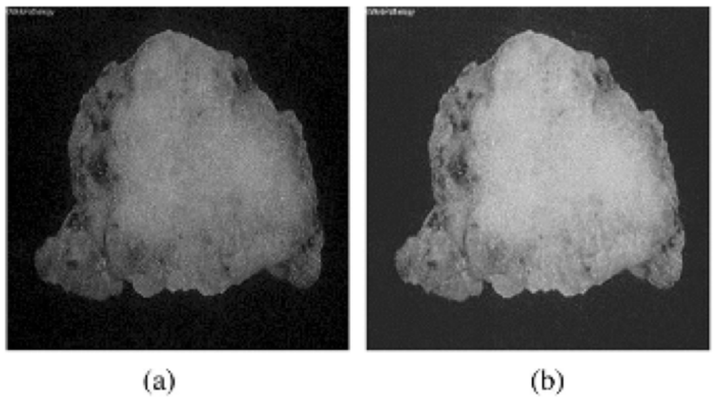

Figure 9: *BPNN Example. (a) Without BPNN (b) With BPNN*

```
'''
Liu, R., & Tan, T. (2002). An svd-based watermarking scheme for protecting
rightful ownership.IEEE Transactionson Multimedia,4(1), 121-128.
'''

import matplotlib.pyplot as plt
from PIL import Image, ImageFilter
import numpy as np
import numpy.linalg as la


###############################
###############################
###############################
## Functions


###############################
## create square images and convert to matrix
def MakeSquareMatrix(img):
    width, height = img.size
    if width == height:
        if len(np.array(img).shape)==3:
            return np.array(img)[:,:,0]
        else:
            return np.array(img)
    elif width > height:
        result = Image.new(img.mode, (width, width))
        result.paste(img, (0, (width - height) // 2))
        if len(np.array(result).shape)==3:
            return np.array(result)[:,:,0]
        else:
            return np.array(result)
    else:
        result = Image.new(img.mode, (height, height))
        result.paste(img, ((height - width) // 2, 0))
        if len(np.array(result).shape)==3:
            return np.array(result)[:,:,0]
        else:
            return np.array(result)

###############################
## Embed Watermark

def EmbedWatermark(A,W,a):
    #[1] Decompose A into U S Vh
    U, S, Vh = la.svd(A,full_matrices=False)
    S = np.diag(S)
    print(U.shape, S.shape, Vh.shape)

    #[2] Add watermark to S (scaled by constant factor a)
    #a = 0.1
    S_aW = S + a*W

    #[3] Decompose (S + aW) into Uw Sw Vwh
    Uw, Sw, Vwh = la.svd(S_aW,full_matrices=False)
    Sw = np.diag(Sw)
    #[4]Generate watermarked original image as U Sw Vh
    Aw = U.dot(Sw.dot(Vh))
    return(Aw,Uw,Vwh,S)

################################
## Extract Watermark

def ExtractWatermark(Aw,Uw,Vwh,S,a):
    #[1] Decompose Aw into U Sw Vh
    U2, Sw2, Vh2 = la.svd(Aw,full_matrices=False)
    Sw2 = np.diag(Sw2)

    #[2] Create D as Uw Sw Vwh
    #(note you need Uw and Vwh from the embedding process to do this)
    D = Uw.dot(Sw2.dot(Vwh))

    #[3] Recover watermark as W = (1/a)(D - S)
    #(note you need a and S from the embedding process to do this)
    W_rec = (1/a)*(D - S)
    return(W_rec)

################################
## Compare Two Arrays

def CompareArrays(a,b,display):
    #convert to 1D vectors
    a_1D = np.concatenate(a)
    b_1D = np.concatenate(b)

    #calc correlation
    corr = np.corrcoef(a_1D,b_1D)

    #display absolute difference image (scaled up by factor 1)
    if display==1:
        dif = np.abs(a - b)
        dif_image = Image.fromarray(dif*1)
        dif_image.show()
        return(corr,dif_image)
    else:
        return(corr)

################################
## Create rectangular watermarked image

def crop_rectangle(Aw, img_Orig):
    img_W = Image.fromarray(Aw)
    w1, h1 = img_Orig.size
    w2, h2 = img_W.size
    img_W = img_W.crop(((w2 - w1) // 2,(h2 - h1) // 2,(w2 + w1) // 2,(h2 + h1) // 2))
    Aw_crop = np.array(img_W)
    return(img_W,Aw_crop)


################################
################################
################################
```

```
## Run functions


################################
# bring in images and embed watermark

#read original image and watermark and convert to grey scale
watermark = Image.open('..\images\Mountains.jpg').convert('LA')
original = Image.open('..\images\golden-retriever-puppy.jpg').convert('LA')

watermark.save(".\\results\\LiuTan2002_watermark.png", "png")
original.save(".\\results\\LiuTan2002_original.png", "png")


#create square images and convert to matrix
W = MakeSquareMatrix(watermark)
A = MakeSquareMatrix(original)

#embed watermark
a = 0.1
Aw,Uw,Vwh,S = EmbedWatermark(A,W,a)

# display watermarked image
original_w = Image.fromarray(Aw)
original_w.convert('LA').save(".\\results\\LiuTan2002_original_w.png", "png")


################################
# checks

#absolute difference image
Aw_dif_image = Image.fromarray(np.abs(A - Aw)*100)
Aw_dif_image.convert('LA').save(".\\results\\LiuTan2002_Aw_dif_image.png", "png")

#check that we get the watermark back
W_rec = ExtractWatermark(Aw,Uw,Vwh,S,a)
watermark_rec_rect,W_rec_rect = crop_rectangle(W_rec,watermark)
watermark_rec_rect.convert('LA').save(".\\results\\LiuTan2002_watermark_rec_rect.png",
"png")


################################
################################
################################
## Experimental Results


################################
# create noise as the "comparison" watermarks
noise = np.zeros((A.shape[0]*A.shape[1],50))
for i in range(49):
    noise[:,i] = np.random.normal(loc=0, scale=(i+1)/2,size=(A.shape[0]*A.shape[1]))
noise[:,49] = np.concatenate(W)


################################
#[0] no_alteration
```

```
#alter image
Aw_no_alteration = Aw
original_w_no_alteration = Image.fromarray(Aw_no_alteration)

#display altered image
original_w_no_alteration.convert('LA').save(".\\results\
\LiuTan2002_original_w_no_alteration.png", "png")

#extract watermark from altered image
W_rec_no_alteration = ExtractWatermark(Aw_no_alteration,Uw,Vwh,S,a)

#plot correlations for different "comparison" watermarks
W_check = np.concatenate(W_rec_no_alteration)
corr = np.zeros(50)
for i in range(50):
    corr[i] = np.corrcoef(W_check,noise[:,i])[0,1]
plt.plot(corr, 'bo')
plt.ylim(0,1)
plt.title('Liu Tan 2002 No Alteration Correlation')
plt.xlabel('Noise and Watermark')
plt.ylabel('Correlation')
plt.savefig('.\\results\\LiuTan2002_Correlation_no_alteration.png')
plt.clf()

#display extracted watermark from altered image
extract_no_alteration = Image.fromarray(W_rec_no_alteration)
extract_no_alteration.convert('LA').save(".\\results\
\LiuTan2002_extract_no_alteration.png", "png")


###############################
#[1] add gaussian noise to watermarked image

#alter image
Aw_GaussNoise = Aw + np.random.normal(loc=0, scale=10,size=(Aw.shape[0],Aw.shape[1]))
original_w_GaussNoise = Image.fromarray(Aw_GaussNoise)

#display altered image
original_w_GaussNoise.convert('LA').save(".\\results\
\LiuTan2002_original_w_GaussNoise.png", "png")

#extract watermark from altered image
W_rec_GaussNoise = ExtractWatermark(Aw_GaussNoise,Uw,Vwh,S,a)

#plot correlations for different "comparison" watermarks
W_check = np.concatenate(W_rec_GaussNoise)
corr = np.zeros(50)
for i in range(50):
    corr[i] = np.corrcoef(W_check,noise[:,i])[0,1]
plt.plot(corr, 'bo')
plt.ylim(0,1)
plt.title('Liu Tan 2002 Gaussian Noise Correlation')
plt.xlabel('Noise and Watermark')
plt.ylabel('Correlation')
plt.savefig('.\\results\\LiuTan2002_Correlation_GaussNoise.png')
plt.clf()
```

```
#display extracted watermark from altered image
extract_GaussNoise = Image.fromarray(W_rec_GaussNoise)
extract_GaussNoise.convert('LA').save(".\\results\\LiuTan2002_extract_GaussNoise.png",
"png")


################################
##[2] blur filter of watermarked image

#alter image
original_w_blur = Image.fromarray(Aw).convert('LA').filter(ImageFilter.GaussianBlur(1))

#display altered image
original_w_blur.convert('LA').save(".\\results\\LiuTan2002_original_w_blur.png", "png")

#create matrix version
Aw_blur = np.array(original_w_blur)[:,:,0]

#extract watermark from altered image
W_rec_blur = ExtractWatermark(Aw_blur,Uw,Vwh,S,a)

#plot correlations for different "comparison" watermarks
W_check = np.concatenate(W_rec_blur)
corr = np.zeros(50)
for i in range(50):
    corr[i] = np.corrcoef(W_check,noise[:,i])[0,1]
plt.plot(corr, 'bo')
plt.ylim(0,1)
plt.title('Liu Tan 2002 Blur Correlation')
plt.xlabel('Noise and Watermark')
plt.ylabel('Correlation')
plt.savefig('.\\results\\LiuTan2002_Correlation_Blur.png')
plt.clf()


#display extracted watermark from altered image
extract_blur = Image.fromarray(W_rec_blur)
extract_blur.convert('LA').save(".\\results\\LiuTan2002_extract_blur.png", "png")


###############################
#[3] image compression

#alter image
original_w_compress = Image.fromarray(Aw).convert('LA').resize((A.shape[0]//2,
                                    A.shape[1]//2)).resize((A.shape[0],A.shape[1]))

#display altered image
original_w_compress.convert('LA').save(".\\results\
\LiuTan2002_original_w_compress.png", "png")

#create matrix version
Aw_compress = np.array(original_w_compress)[:,:,0]

#extract watermark from altered image
W_rec_compress = ExtractWatermark(Aw_compress,Uw,Vwh,S,a)
```

```
#plot correlations for different "comparison" watermarks
W_check = np.concatenate(W_rec_compress)
corr = np.zeros(50)
for i in range(50):
    corr[i] = np.corrcoef(W_check,noise[:,i])[0,1]
plt.plot(corr, 'bo')
plt.ylim(0,1)
plt.title('Liu Tan 2002 Compression Correlation')
plt.xlabel('Noise and Watermark')
plt.ylabel('Correlation')
plt.savefig('.\\results\\LiuTan2002_Correlation_Compression.png')
plt.clf()

#display extracted watermark from altered image
extract_compress = Image.fromarray(W_rec_compress)
extract_compress.convert('LA').save(".\\results\\LiuTan2002_extract_compress.png",
"png")


################################
#[4] rotate image

#alter image
original_w_rotated = Image.fromarray(Aw).convert('LA').rotate(30)

#display altered image
original_w_rotated.convert('LA').save(".\\results\\LiuTan2002_original_w_rotated.png",
"png")

#create matrix version
Aw_rotated = np.array(original_w_rotated)[:,:,0]

#extract watermark from altered image
W_rec_rotated = ExtractWatermark(Aw_rotated,Uw,Vwh,S,a)

#plot correlations for different "comparison" watermarks
W_check = np.concatenate(W_rec_rotated)
corr = np.zeros(50)
for i in range(50):
    corr[i] = np.corrcoef(W_check,noise[:,i])[0,1]
plt.plot(corr, 'bo')
plt.ylim(0,1)
plt.title('Liu Tan 2002 Rotation Correlation')
plt.xlabel('Noise and Watermark')
plt.ylabel('Correlation')
plt.savefig('.\\results\\LiuTan2002_Correlation_Rotation.png')
plt.clf()

#display extracted watermark from altered image
extract_rotated = Image.fromarray(W_rec_rotated)
extract_rotated.convert('LA').save(".\\results\\LiuTan2002_extract_rotated.png", "png")


################################
#[5] crop image
```

```
#alter image
original_w_crop = Image.fromarray(Aw).convert('LA').crop((0, 0,
                                  Image.fromarray(Aw).convert('LA').size[0]//2,
                                  Image.fromarray(Aw).convert('LA').size[1]))
Aw_crop = np.array(original_w_crop)[:,:,0]
blackspace = np.zeros((Aw_crop.shape[0],Aw_crop.shape[1]))
Aw_crop = np.concatenate((Aw_crop,blackspace),axis=1)
original_w_crop = Image.fromarray(Aw_crop)

#display altered image
original_w_crop.convert('LA').save(".\\results\\LiuTan2002_original_w_crop.png", "png")

#extract watermark from altered image
W_rec_crop = ExtractWatermark(Aw_crop,Uw,Vwh,S,a)

#plot correlations for different "comparison" watermarks
W_check = np.concatenate(W_rec_crop)
corr = np.zeros(50)
for i in range(50):
    corr[i] = np.corrcoef(W_check,noise[:,i])[0,1]
plt.plot(corr, 'bo')
plt.ylim(0,1)
plt.title('Liu Tan 2002 Crop Correlation')
plt.xlabel('Noise and Watermark')
plt.ylabel('Correlation')
plt.savefig('.\\results\\LiuTan2002_Correlation_Crop.png')
plt.clf()


#display extracted watermark from altered image
extract_crop = Image.fromarray(W_rec_crop)
extract_crop.convert('LA').save(".\\results\\LiuTan2002_extract_crop.png", "png")
```

```
'''
Lai, C., & Tsai, C.  (2010).  Digital image watermarking using discrete wavelet
transform and singular value decom-position.IEEE Transactions on Instrumentation and
Measurement,59(11), 3060-3063.

Note that this code draws from the Liu and Tan 2002 code
'''

#Source: Gregory R. Lee, Ralf Gommers, Filip Wasilewski, Kai Wohlfahrt, Aaron O'Leary
(2019).
#PyWavelets: A Python package for wavelet analysis. Journal of Open Source Software,
4(36), 1237, https://doi.org/10.21105/joss.01237.
import pywt

#Preform DWT
coeffs2 = pywt.dwt2(A, 'haar')
LL, (LH, HL, HH) = coeffs2

#Make Watermark same size as the LH and HL matrices
W_sized = W[range(0,(int(W.shape[0]/2))),:]
W_sized = W_sized[:,range(int(W.shape[0]/2))]
halfW = .5*W_sized

#preform watermarking procedure, with watermark halfW, on LH and HL
Aw1,Uw1,Vwh1,S1 = EmbedWatermark(LH,halfW,0.1)
Aw2,Uw2,Vwh2,S2 = EmbedWatermark(HL,halfW,0.1)

#Preform reverse DWT procedure to get Aw
coefs = LL, (Aw1, Aw2, HH)
Aw = pywt.idwt2(coefs, 'haar')
Aw_img = Image.fromarray(Aw)
Aw_img.show()


def ExtractWatermarkDWT(Aw,Uw1,Vwh1,S1,Uw2,Vwh2,S2,a):
    #Break watermarked image using DWT into LL, LH,HL, HH
    coefs_Aw = pywt.dwt2(Aw, 'haar')
    LLw, (LHw, HLw, HHw)= coefs_Aw
    #extract half watermark from LHw and HLw, add together and display
    W_1 = ExtractWatermark(LHw,Uw1,Vwh1,S1,0.1)
    W_2 = ExtractWatermark(HLw,Uw2,Vwh2,S2,0.1)
    W_DWT = W_1 + W_2
    return W_DWT

W_DWT = ExtractWatermarkDWT(Aw,Uw1,Vwh1,S1,Uw2,Vwh2,S2,0.1)

#create random watermarks for half image
noise  = np.zeros((W_sized.shape[0],W_sized.shape[1],50))
for i in range(49):
    noise[:,:,i] = np.random.normal(loc=0, scale=(i+1)/
2,size=(W_sized.shape[0],W_sized.shape[1]))
noise[:,:,49] = W_DWT


###############################
#[0] no_alteration
a = .01
```

```
#alter image
Aw_no_alteration = Aw
original_w_no_alteration = Image.fromarray(Aw_no_alteration)

#display altered image
original_w_no_alteration.convert('LA').save(".\\DWT_original_w_no_alteration.png",
"png")

#extract watermark from altered image
W_rec_no_alteration = ExtractWatermarkDWT(Aw_no_alteration,Uw1,Vwh1,S1,Uw2,Vwh2,S2,a)

#plot correlations for different "comparison" watermarks
W_check = np.concatenate(W_rec_no_alteration)
corr = np.zeros(50)
for i in range(50):
    corr[i] = np.corrcoef(W_check,np.concatenate(noise[:,:,i]))[0,1]
plt.plot(corr, 'bo')
plt.ylim(0,1)
plt.title('DWT No Alteration Correlation')
plt.xlabel('Noise and Watermark')
plt.ylabel('Correlation')
plt.savefig('.\\DWT_Correlation_no_alteration.png')
plt.clf()

#display extracted watermark from altered image
extract_no_alteration = Image.fromarray(W_rec_no_alteration)
extract_no_alteration.convert('LA').save(".\\DWT_extract_no_alteration.png", "png")


################################
#[1] add gaussian noise to watermarked image

#alter image
Aw_GaussNoise = Aw + np.random.normal(loc=0, scale=10,size=(Aw.shape[0],Aw.shape[1]))
original_w_GaussNoise = Image.fromarray(Aw_GaussNoise)

#display altered image
original_w_GaussNoise.convert('LA').save(".\\DWT_original_w_GaussNoise.png", "png")

#extract watermark from altered image
W_rec_GaussNoise = ExtractWatermarkDWT(Aw_GaussNoise,Uw1,Vwh1,S1,Uw2,Vwh2,S2,a)

#plot correlations for different "comparison" watermarks
W_check = np.concatenate(W_rec_GaussNoise)
corr = np.zeros(50)
for i in range(50):
    corr[i] = np.corrcoef(W_check,np.concatenate(noise[:,:,i]))[0,1]
plt.plot(corr, 'bo')
plt.ylim(0,1)
plt.title('DWT Gaussian Noise Correlation')
plt.xlabel('Noise and Watermark')
plt.ylabel('Correlation')
plt.savefig('.\\DWT_Correlation_GaussNoise.png')
plt.clf()

#display extracted watermark from altered image
extract_GaussNoise = Image.fromarray(W_rec_GaussNoise)
```

```
extract_GaussNoise.convert('LA').save(".\\DWT_extract_GaussNoise.png", "png")


##################################
##[2] blur filter of watermarked image

#alter image
original_w_blur = Image.fromarray(Aw).convert('LA').filter(ImageFilter.GaussianBlur(1))

#display altered image
original_w_blur.convert('LA').save(".\\DWT_original_w_blur.png", "png")

#create matrix version
Aw_blur = np.array(original_w_blur)[:,:,0]

#extract watermark from altered image
W_rec_blur = ExtractWatermarkDWT(Aw_blur,Uw1,Vwh1,S1,Uw2,Vwh2,S2,a)

#plot correlations for different "comparison" watermarks
W_check = np.concatenate(W_rec_blur)
corr = np.zeros(50)
for i in range(50):
    corr[i] = np.corrcoef(W_check,np.concatenate(noise[:,:,i]))[0,1]
plt.plot(corr, 'bo')
plt.ylim(0,1)
plt.title('DWT Blur Correlation')
plt.xlabel('Noise and Watermark')
plt.ylabel('Correlation')
plt.savefig('.\\DWT_Correlation_Blur.png')
plt.clf()

#display extracted watermark from altered image
extract_blur = Image.fromarray(W_rec_blur)
extract_blur.convert('LA').save(".\\DWT_extract_blur.png", "png")


#################################
#[3] image compression

#alter image
x = Image.fromarray(Aw).resize((100, 100))
Aw_compress = np.array(x.resize((Aw.shape[0],Aw.shape[1])))
original_w_compress = Image.fromarray(Aw_compress)

#display altered image
original_w_compress.convert('LA').save(".\\DWT_original_w_compress.png", "png")

#extract watermark from altered image
W_rec_compress = ExtractWatermarkDWT(Aw_compress,Uw1,Vwh1,S1,Uw2,Vwh2,S2,a)

#plot correlations for different "comparison" watermarks
W_check = np.concatenate(W_rec_compress)
corr = np.zeros(50)
for i in range(50):
    corr[i] = np.corrcoef(W_check,np.concatenate(noise[:,:,i]))[0,1]
plt.plot(corr, 'bo')
plt.ylim(0,1)
plt.title('DWT Compression Correlation')
plt.xlabel('Noise and Watermark')
plt.ylabel('Correlation')
plt.savefig('.\\DWT_Correlation_Compression.png')
plt.clf()

#display extracted watermark from altered image
extract_compress = Image.fromarray(W_rec_compress)
extract_compress.convert('LA').save(".\\DWT_extract_compress.png", "png")


################################
#[4] rotate image

#alter image
original_w_rotated = Image.fromarray(Aw).convert('LA').rotate(30)

#display altered image
original_w_rotated.convert('LA').save(".\\DWT_original_w_rotated.png", "png")

#create matrix version
Aw_rotated = np.array(original_w_rotated)[:,:,0]

#extract watermark from altered image
W_rec_rotated = ExtractWatermarkDWT(Aw_rotated,Uw1,Vwh1,S1,Uw2,Vwh2,S2,a)

#plot correlations for different "comparison" watermarks
W_check = np.concatenate(W_rec_rotated)
corr = np.zeros(50)
for i in range(50):
    corr[i] = np.corrcoef(W_check,np.concatenate(noise[:,:,i]))[0,1]
plt.plot(corr, 'bo')
plt.ylim(0,1)
plt.title('DWT Rotation Correlation')
plt.xlabel('Noise and Watermark')
plt.ylabel('Correlation')
plt.savefig('.\\DWT_Correlation_Rotation.png')
plt.clf()

#display extracted watermark from altered image
extract_rotated = Image.fromarray(W_rec_rotated)
extract_rotated.convert('LA').save(".\\DWT_extract_rotated.png", "png")


################################
#[5] crop image

#alter image
original_w_crop = Image.fromarray(Aw).convert('LA').crop((0, 0,
                                  Image.fromarray(Aw).convert('LA').size[0]//2,
                                  Image.fromarray(Aw).convert('LA').size[1]))
Aw_crop = np.array(original_w_crop)[:,:,0]
blackspace = np.zeros((Aw_crop.shape[0],Aw_crop.shape[1]))
Aw_crop = np.concatenate((Aw_crop,blackspace),axis=1)
original_w_crop = Image.fromarray(Aw_crop)

#display altered image
```

```
original_w_crop.convert('LA').save(".\\DWT_original_w_crop.png", "png")

#extract watermark from altered image
W_rec_crop = ExtractWatermarkDWT(Aw_crop,Uw1,Vwh1,S1,Uw2,Vwh2,S2,a)

#plot correlations for different "comparison" watermarks
W_check = np.concatenate(W_rec_crop)
corr = np.zeros(50)
for i in range(50):
    corr[i] = np.corrcoef(W_check,np.concatenate(noise[:,:,i]))[0,1]
plt.plot(corr, 'bo')
plt.ylim(0,1)
plt.title('DWT Crop Correlation')
plt.xlabel('Noise and Watermark')
plt.ylabel('Correlation')
plt.savefig('.\\DWT_Correlation_Crop.png')
plt.clf()


#display extracted watermark from altered image
extract_crop = Image.fromarray(W_rec_crop)
extract_crop.convert('LA').save(".\\DWT_extract_crop.png", "png")
```

```
'''
Chang, C., Tsai, P., & Lin, C.  (2005).  Svd-based digital image watermarking
scheme.Pattern Recognition Letters,26, 1577-1586.
'''

import matplotlib.pyplot as plt
from PIL import Image, ImageFilter
import numpy as np
import numpy.linalg as la


################################
################################
################################
## Functions


################################
## create square images and convert to matrix
def MakeSquareMatrix(img):
    width, height = img.size
    if width == height:
        if len(np.array(img).shape)==3:
            return np.array(img)[:,:,0]
        else:
            return np.array(img)
    elif width > height:
        result = Image.new(img.mode, (width, width))
        result.paste(img, (0, (width - height) // 2))
        if len(np.array(result).shape)==3:
            return np.array(result)[:,:,0]
        else:
            return np.array(result)
    else:
        result = Image.new(img.mode, (height, height))
        result.paste(img, ((height - width) // 2, 0))
        if len(np.array(result).shape)==3:
            return np.array(result)[:,:,0]
        else:
            return np.array(result)


################################
## Embed Watermark

def EmbedWatermark(A,W,t,nb):

    np.random.seed(seed=100)

    #[1] divide original image into blocks and turn watermark into 1D vector
    bsize = int(A.shape[0]/nb)
    A_Blocks = np.zeros((bsize,bsize,int(A.shape[0]/bsize)*int(A.shape[0]/bsize)))
    k = 0
    for i in range(int(A.shape[0]/bsize)):
        for j in range(int(A.shape[1]/bsize)):
            A_Blocks[:,:,k] = A[(i)*bsize:min((i)*bsize+bsize,A.shape[0]),
(j)*bsize:min((j)*bsize+bsize,A.shape[1])]
```

```
            k = k + 1
    W_long = W.reshape(int(W.shape[0]*W.shape[0]))

    #[2] SVD Decompose each block into U S Vh
    USV = np.zeros((A_Blocks.shape[0],A_Blocks.shape[1],3,A_Blocks.shape[2]))
    for i in range(A_Blocks.shape[2]):
        U, S, Vh = la.svd(A_Blocks[:,:,i],full_matrices=False)
        S = np.diag(S)
        USV[:,:,0,i],USV[:,:,1,i],USV[:,:,2,i] = U, S, Vh

    #[3] Determine block importance/complexity
    #D-Feature = # non zero components in D (here D is our S matrix)
    #the greater number of non-zero coefficients would indicate greater complexity
    #For a block-based watermarking scheme, a more complex block was favored for
embedding a watermark with perceptibility
    #Applying the feature of the D component prevents the smooth blocks from being
selected
    #and benefits the perceptibility of the watermarked image
    DFeature = np.zeros((A_Blocks.shape[2],2))
    for i in range(A_Blocks.shape[2]):
        DFeature[i,0] = i
        DFeature[i,1] = sum(np.round(np.diag(USV[:,:,1,i]),10)!=0)

    #[4] Select random sample of (W_long.shape[0]) blocks with max D-Feature
    #we sample the same number of blocks as we have pixels in the watermark
    #Using the pseudo random number generator (PRNG) increases the watermarking
security.
    BlocksIDs = np.random.choice(DFeature[DFeature[:,1]==np.max(DFeature[:,1]),
0],size=int(W_long.shape[0]),replace=False)

    #[5] For selected blocks, find U-feature
    #U-Feature = magnitude difference between the neighboring elements in first column
of U
    #we only compare element (2,1) and (3,1) for (row, col)

    for i, b in enumerate(BlocksIDs):
        b=int(b)

        #coefficient reassignment
        #goal is to have a positive difference when the watermark pixel is 1
        #                a negative difference when the watermark pixel is 0
        #so when we already see the correct difference, we make it larger and
        #when we see the opposite difference, we switch it to the correct one

        d = (np.abs(USV[1,0,0,b])-np.abs(USV[2,0,0,b]))
        #positive difference
        if d>0:
            #positive difference, match
            if W_long[i]==True:
                USV[1,0,0,b] = -np.abs( np.abs(USV[1,0,0,b]) + (d+t)/2 )
                USV[2,0,0,b] = -np.abs( np.abs(USV[2,0,0,b]) - (d+t)/2 )
            #positive difference, no match
            if W_long[i]==False:
                USV[1,0,0,b] = -np.abs( np.abs(USV[1,0,0,b]) - (d+t)/2 )
                USV[2,0,0,b] = -np.abs( np.abs(USV[2,0,0,b]) + (d+t)/2 )

        #negaitve difference
        if d<=0:
            #negaitve difference, match
            if W_long[i]==False:
                USV[1,0,0,b] = -np.abs( np.abs(USV[1,0,0,b]) + (d-t)/2 )
                USV[2,0,0,b] = -np.abs( np.abs(USV[2,0,0,b]) - (d-t)/2 )
            #negaitve difference, no match
            if W_long[i]==True:
                USV[1,0,0,b] = -np.abs( np.abs(USV[1,0,0,b]) - (d-t)/2 )
                USV[2,0,0,b] = -np.abs( np.abs(USV[2,0,0,b]) + (d-t)/2 )

    #[6] inverse SVD (U Sw Vh) to get watermarked blocks
    A_Blocks_Water = np.zeros((A_Blocks.shape[0],A_Blocks.shape[1],A_Blocks.shape[2]))
    for i in range(USV.shape[3]):
        A_Blocks_Water[:,:,i] = USV[:,:,0,i].dot(USV[:,:,1,i].dot(USV[:,:,2,i]))

    #[7] Generate watermarked original image by composing blocks
    Aw = np.zeros((A.shape[0],A.shape[1]))
    k = 0
    for i in range(int(nb)):
        for j in range(int(nb)):
            Aw[(i)*bsize:min((i)*bsize+bsize,A.shape[0]),(j)*bsize:min((j)*bsize
+bsize,A.shape[1])] = A_Blocks_Water[:,:,k]
            k = k + 1

    return(Aw,BlocksIDs)


###############################
## Extract Watermark

def ExtractWatermark(Aw,BlocksIDs,nb):

    np.random.seed(seed=100)

    #[1] divide watermarked image into blocks and create empty watermark to be
populated
    bsize = int(Aw.shape[0]/nb)
    A_Blocks_Water2 = np.zeros((bsize,bsize,int(Aw.shape[0]/bsize)*int(Aw.shape[0]/
bsize)))
    k = 0
    for i in range(int(Aw.shape[0]/bsize)):
        for j in range(int(Aw.shape[1]/bsize)):
            A_Blocks_Water2[:,:,k] = Aw[(i)*bsize:min((i)*bsize+bsize,Aw.shape[0]),
(j)*bsize:min((j)*bsize+bsize,Aw.shape[1])]
            k = k + 1
    W_rec_long = np.ones(BlocksIDs.shape[0], dtype=bool)

    #[2] SVD Decompose each block into U S Vh
    USV_w = np.zeros((A_Blocks_Water2.shape[0],A_Blocks_Water2.shape[1],
3,A_Blocks_Water2.shape[2]))
    for i in range(A_Blocks_Water2.shape[2]):
        U, S, Vh = la.svd(A_Blocks_Water2[:,:,i],full_matrices=False)
        S = np.diag(S)
        USV_w[:,:,0,i],USV_w[:,:,1,i],USV_w[:,:,2,i] = U, S, Vh

    #[3] Generate recovered watermark from U values of watermarked blocks
    for i, b in enumerate(BlocksIDs):
        b=int(b)

        d = (np.abs(USV_w[1,0,0,b])-np.abs(USV_w[2,0,0,b]))
        #positive relationship
        if d>0:
            W_rec_long[i] = True

        #negative relationship
        if d<=0:
            W_rec_long[i] = False

    #[4] Form recovered watermark
    W_rec =
W_rec_long.reshape(int(np.sqrt(BlocksIDs.shape[0])),int(np.sqrt(BlocksIDs.shape[0])))

    return(W_rec)


################################
## Compare Two Arrays

def CompareArrays(a,b,display):
    #convert to 1D vectors
    a_1D = 1*(np.concatenate(a))
    b_1D = 1*(np.concatenate(b))

    #calc correlation
    corr = np.corrcoef(a_1D,b_1D)

    #display absolute difference image (scaled up by factor 1)
    if display==1:
        dif = np.abs(1*a - 1*b)
        dif_image = Image.fromarray(dif*1)
        dif_image.show()
        return(corr,dif_image)
    else:
        return(corr)

################################
## Create rectangular watermarked image

def crop_rectangle(Aw, img_Orig):
    img_W = Image.fromarray(Aw).convert('LA')
    w1, h1 = img_Orig.size
    w2, h2 = img_W.size
    img_W = img_W.crop(((w2 - w1) // 2,(h2 - h1) // 2,(w2 + w1) // 2,(h2 + h1) // 2))
    Aw_crop = np.array(img_W)
    return(img_W,Aw_crop)


################################
################################
################################
## Run functions

################################
# bring in images and embed watermark
```

```
#read original image and convert to grey scale
original = Image.open('..\images\golden-retriever-puppy.jpg').convert('LA')

#read watermark image and convert to black and white
watermark = Image.open('..\images\mario1.png').convert('1')

#resize watermark
watermark = watermark.resize((32, 32))

watermark.save(".\\results\\ChangTsaiLin2005_watermark.png", "png")
original.save(".\\results\\ChangTsaiLin2005_original.png", "png")

#create square images and convert to matrix
W = MakeSquareMatrix(watermark)
A = MakeSquareMatrix(original)

#embed watermark
t=0.01
nb = 100
Aw,BlocksIDs = EmbedWatermark(A,W,t,nb)

# display watermarked image
original_w = Image.fromarray(Aw)
original_w.convert('LA').save(".\\results\\ChangTsaiLin2005_original_w.png", "png")


################################
# checks

#absolute difference image
Aw_dif_image = Image.fromarray(np.abs(A - Aw)*100)
Aw_dif_image.convert('LA').save(".\\results\\ChangTsaiLin2005_Aw_dif_image.png", "png")

#check that we get the watermark back
W_rec = ExtractWatermark(Aw,BlocksIDs,nb)
watermark_rec_rect,W_rec_rect = crop_rectangle(W_rec,watermark)
watermark_rec_rect.convert('LA').save(".\\results\
\ChangTsaiLin2005_watermark_rec_rect.png", "png")



################################
################################
################################
## Experimental Results

################################
# create noise as the "comparison" watermarks
noise = np.zeros((W.shape[0]*W.shape[1],50))
for i in range(49):
    noise[:,i] = np.random.binomial(p=0.5,n=1,size=(W.shape[0]*W.shape[1]))
noise[:,49] = np.concatenate(W)


################################
#[0] no_alteration
```

```
#alter image
Aw_no_alteration = Aw
original_w_no_alteration = Image.fromarray(Aw_no_alteration)

#display altered image
original_w_no_alteration.convert('LA').save(".\\results\
\ChangTsaiLin2005_original_w_no_alteration.png", "png")

#extract watermark from altered image
W_rec_no_alteration = ExtractWatermark(Aw_no_alteration,BlocksIDs,nb)

#plot correlations for different "comparison" watermarks
W_check = np.concatenate(W_rec_no_alteration)
corr = np.zeros(50)
for i in range(50):
    corr[i] = np.corrcoef(W_check,noise[:,i])[0,1]
plt.plot(corr, 'bo')
plt.ylim(0,1)
plt.title('Chang Tsai Lin 2005 No Alteration Correlation')
plt.xlabel('Noise and Watermark')
plt.ylabel('Correlation')
plt.savefig('.\\results\\ChangTsaiLin2005_Correlation_no_alteration.png')
plt.clf()

#display extracted watermark from altered image
extract_no_alteration = Image.fromarray(W_rec_no_alteration)
extract_no_alteration.convert('LA').save(".\\results\
\ChangTsaiLin2005_extract_no_alteration.png", "png")


################################
#[1] add gaussian noise to watermarked image

#alter image
Aw_GaussNoise = Aw + np.random.normal(loc=0, scale=10,size=(Aw.shape[0],Aw.shape[1]))
original_w_GaussNoise = Image.fromarray(Aw_GaussNoise)

#display altered image
original_w_GaussNoise.convert('LA').save(".\\results\
\ChangTsaiLin2005_original_w_GaussNoise.png", "png")

#extract watermark from altered image
W_rec_GaussNoise = ExtractWatermark(Aw_GaussNoise,BlocksIDs,nb)

#plot correlations for different "comparison" watermarks
W_check = np.concatenate(W_rec_GaussNoise)
corr = np.zeros(50)
for i in range(50):
    corr[i] = np.corrcoef(W_check,noise[:,i])[0,1]
plt.plot(corr, 'bo')
plt.ylim(0,1)
plt.title('Chang Tsai Lin 2005 Gaussian Noise Correlation')
plt.xlabel('Noise and Watermark')
plt.ylabel('Correlation')
plt.savefig('.\\results\\ChangTsaiLin2005_Correlation_GaussNoise.png')
plt.clf()
```

```
#display extracted watermark from altered image
extract_GaussNoise = Image.fromarray(W_rec_GaussNoise)
extract_GaussNoise.convert('LA').save(".\\results\
\ChangTsaiLin2005_extract_GaussNoise.png", "png")


################################
##[2] blur filter of watermarked image

#alter image
original_w_blur = Image.fromarray(Aw).convert('LA').filter(ImageFilter.GaussianBlur(1))

#display altered image
original_w_blur.convert('LA').save(".\\results\\ChangTsaiLin2005_original_w_blur.png",
"png")

#create matrix version
Aw_blur = np.array(original_w_blur)[:,:,0]

#extract watermark from altered image
W_rec_blur = ExtractWatermark(Aw_blur,BlocksIDs,nb)

#plot correlations for different "comparison" watermarks
W_check = np.concatenate(W_rec_blur)
corr = np.zeros(50)
for i in range(50):
    corr[i] = np.corrcoef(W_check,noise[:,i])[0,1]
plt.plot(corr, 'bo')
plt.ylim(0,1)
plt.title('Chang Tsai Lin 2005 Blur Correlation')
plt.xlabel('Noise and Watermark')
plt.ylabel('Correlation')
plt.savefig('.\\results\\ChangTsaiLin2005_Correlation_Blur.png')
plt.clf()

#display extracted watermark from altered image
extract_blur = Image.fromarray(W_rec_blur)
extract_blur.convert('LA').save(".\\results\\ChangTsaiLin2005_extract_blur.png", "png")


###############################
#[3] image compression

#alter image
original_w_compress = Image.fromarray(Aw).convert('LA').resize((A.shape[0]//
2,A.shape[1]//2)).resize((A.shape[0],A.shape[1]))

#display altered image
original_w_compress.convert('LA').save(".\\results\
\ChangTsaiLin2005_original_w_compress.png", "png")

#create matrix version
Aw_compress = np.array(original_w_compress)[:,:,0]

#extract watermark from altered image
W_rec_compress = ExtractWatermark(Aw_compress,BlocksIDs,nb)
```

```
#plot correlations for different "comparison" watermarks
W_check = np.concatenate(W_rec_compress)
corr = np.zeros(50)
for i in range(50):
    corr[i] = np.corrcoef(W_check,noise[:,i])[0,1]
plt.plot(corr, 'bo')
plt.ylim(0,1)
plt.title('Chang Tsai Lin 2005 Compression Correlation')
plt.xlabel('Noise and Watermark')
plt.ylabel('Correlation')
plt.savefig('.\\results\\ChangTsaiLin2005_Correlation_Compression.png')
plt.clf()

#display extracted watermark from altered image
extract_compress = Image.fromarray(W_rec_compress)
extract_compress.convert('LA').save(".\\results\
\ChangTsaiLin2005_extract_compress.png", "png")


################################
#[4] rotate image

#alter image
original_w_rotated = Image.fromarray(Aw).convert('LA').rotate(30)

#display altered image
original_w_rotated.convert('LA').save(".\\results\
\ChangTsaiLin2005_original_w_rotated.png", "png")

#create matrix version
Aw_rotated = np.array(original_w_rotated)[:,:,0]

#extract watermark from altered image
W_rec_rotated = ExtractWatermark(Aw_rotated,BlocksIDs,nb)

#plot correlations for different "comparison" watermarks
W_check = np.concatenate(W_rec_rotated)
corr = np.zeros(50)
for i in range(50):
    corr[i] = np.corrcoef(W_check,noise[:,i])[0,1]
plt.plot(corr, 'bo')
plt.ylim(0,1)
plt.title('Chang Tsai Lin 2005 Rotation Correlation')
plt.xlabel('Noise and Watermark')
plt.ylabel('Correlation')
plt.savefig('.\\results\\ChangTsaiLin2005_Correlation_Rotation.png')
plt.clf()

#display extracted watermark from altered image
extract_rotated = Image.fromarray(W_rec_rotated)
extract_rotated.convert('LA').save(".\\results\\ChangTsaiLin2005_extract_rotated.png",
"png")


################################
#[5] crop image
```

```
#alter image
original_w_crop = Image.fromarray(Aw).convert('LA').crop((0, 0,
Image.fromarray(Aw).convert('LA').size[0]//2,
Image.fromarray(Aw).convert('LA').size[1]))
Aw_crop = np.array(original_w_crop)[:,:,0]
blackspace = np.zeros((Aw_crop.shape[0],Aw_crop.shape[1]))
Aw_crop = np.concatenate((Aw_crop,blackspace),axis=1)
original_w_crop = Image.fromarray(Aw_crop)

#display altered image
original_w_crop.convert('LA').save(".\\results\\ChangTsaiLin2005_original_w_crop.png",
"png")

#extract watermark from altered image
W_rec_crop = ExtractWatermark(Aw_crop,BlocksIDs,nb)

#plot correlations for different "comparison" watermarks
W_check = np.concatenate(W_rec_crop)
corr = np.zeros(50)
for i in range(50):
    corr[i] = np.corrcoef(W_check,noise[:,i])[0,1]
plt.plot(corr, 'bo')
plt.ylim(0,1)
plt.title('Chang Tsai Lin 2005 Crop Correlation')
plt.xlabel('Noise and Watermark')
plt.ylabel('Correlation')
plt.savefig('.\\results\\ChangTsaiLin2005_Correlation_Crop.png')
plt.clf()

#display extracted watermark from altered image
extract_crop = Image.fromarray(W_rec_crop)
extract_crop.convert('LA').save(".\\results\\ChangTsaiLin2005_extract_crop.png", "png")
```