BROWN UNIVERSITY
SUMMER@ICERM
Providence (RI) USA
Project

Randomized Singular Value Decomposition and its Applications

Authors:

Katherine Keegan

David Melendez

Jennifer Zheng

Primary Advisor:
Minah Oh

Secondary Advisor:
Akil Narayan

Teaching Assistants:

Justin Dong
Alexandru Mihai

Summer 2020
Report

Abstract

The singular value decomposition, or the SVD, was first discovered independently by Eugenio Beltrami and Camille Jordan in the 1870s as they were tackling problems related to bilinear forms in linear algebra. Since then, it has become one of the most useful tools in linear algebra, seeing applications in widely disparate fields. In this report, we introduce applications of the SVD in image, video, and audio processing, data analysis, and digital ownership protection. We provide several examples illustrating applications and properties of the SVD, including image and audio compression, image and audio processing, video background extraction, analysis of data from the SARS-CoV-2 pandemic, and watermarking for digital ownership protection. Finally, we propose a modified version of a watermarking scheme introduced in [10, 13] which offers improved robustness and imperceptibility properties.

Contents

1 Introduction **4**

2 Background **5**
- 2.1 Singular Value Decomposition 5
- 2.2 Eckart-Young Theorem 5
- 2.3 Randomized SVD Algorithms 6
  - 2.3.1 Randomized SVD 6
  - 2.3.2 Compressed SVD 7
- 2.4 Principal Component Analysis 7

3 Methodology **9**
- 3.1 Image and Video Processing 9
  - 3.1.1 Matrix Representation for Images and Videos 9
  - 3.1.2 Image Compression 10
  - 3.1.3 Modifying Singular Values 11
- 3.2 Audio Processing 12
  - 3.2.1 Matrix Representation for Audio 12
  - 3.2.2 Singular Value Modification for Audio 12
- 3.3 Data Analysis: COVID-19 13
  - 3.3.1 Scree Plots 13
  - 3.3.2 Scatter Plots of Principal Components 14
  - 3.3.3 Rank 1 Outer Product Plots 14
- 3.4 Watermarking 15
  - 3.4.1 Liu & Tan Watermarking Scheme 15
  - 3.4.2 Jain et al. Watermarking Scheme 16
  - 3.4.3 Modification to the Jain et al. Watermarking Scheme . 17
  - 3.4.4 Evaluating Watermarking Schemes 18

4 Applications **18**
- 4.1 Image and Video Processing 18
  - 4.1.1 Image Compression 18
  - 4.1.2 Video Background Extraction 19
  - 4.1.3 Modifying Singular Values for Images 20
- 4.2 Audio Processing 25
  - 4.2.1 Audio Compression 25
  - 4.2.2 Modifying Singular Values for Audio 26

4.3 Data Analysis: COVID-19 28
4.3.1 State Clustering 28
4.3.2 States 30
4.3.3 Counties in Florida 39
4.4 Watermarking and Steganography 46
4.4.1 Liu & Tan Watermarking Scheme 46
4.4.2 Jain et al. Watermarking Scheme 57
4.4.3 Modified Jain et al. Watermarking Scheme 65
4.4.4 Comparing Jain et al. and Modified Jain et al. Watermarking Schemes 74
4.4.5 Audio 78

5 Concluding Remarks **80**

1 Introduction

The singular value decomposition(SVD) was discovered in the 1870s by Eugenio Beltrami and Camille Jordan independently while they were studying problems related to bilinear forms in linear algebra. Since then, the SVD has become one of the most important tools in linear algebra, seeing applications in widely disparate problem domains including media processing, data analysis, and digital ownership protection.

Low-rank approximations using the SVD preserve the most dominant features of the media and effectively compresses other information. By modifying the singular values of the media matrix, we can also modify certain characteristics of the media. The method is also proved to be the closest low-rank approximation by the Eckart-Young Theorem.

SVD-based analysis is also effective in determining trends or important features in data as we apply it onto the COVID-19 dataset. Each singular vector corresponds to a certain pattern of the dataset and can group observations into clusters base on the different trends as we plot the singular vectors.

In the application of steganography, and in particular digital ownership protection, SVD allows us to embed and extract one media into another with low perceptibility. For image watermarking, we conduct numerical experiments testing the performance of two different algorithms from Liu & Tan[13] and Jain et al.[10]. As one is robust but not secure and the other one is secure with low imperceptibility, we make modifications to the Jain et al. scheme and present a secure algorithm with better robustness.

The rest of our report will present our work as follows. In Section 2, we review the SVD and its computation. In Section 3, we introduce our methodology in each of the three applications we work on. In Section 4, we present our results and the corresponding numerical experiments for the specific applications. In Section 5, we conclude our work with a short discussion of the future directions.

2 Background

2.1 Singular Value Decomposition

The SVD is closely related to the eigenvalue decomposition (EVD): given an $n \times n$ matrix X, there exists an orthogonal matrix P and a diagonal matrix D such that $X = PDP^{-1}$. In EVD, the columns of P are eigenvectors of X that form an orthonormal basis for $\mathbb{R}^n$, and the diagonal entries of D contain the corresponding eigenvalues[11]. However, EVD can only be performed on diagonalizable symmetric square matrices, thus to generalize the EVD, we have the SVD.

According to the definition of SVD, any $m \times n$ matrix A with rank r can be decomposed into

$$A_{m\times m} = U_{m\times m}\Sigma_{m\times n}V^{\top}_{n\times n},$$

where U and V are orthogonal matrices, and Σ is a diagonal matrix with ordered positive diagonal entries $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_r$, called the singular values of A. The rest of the diagonal entries of A are zero. The columns of U and V are called the left and right singular vectors of A respectively.

Computing the SVD consists of finding the eigenvalues and eigenvectors of $AA^{\top}$ and $A^{\top}A$. Since $AA^{\top}$ is a square matrix, we can perform EVD and let $A^{\top}A = PDP^{\top}$. The eigenvectors of $A^{\top}A$, which are the columns of P would then make up the columns of V, the diagonal entries of D consists of the squared singular value of A, and similarly the eigenvectors of $AA^{\top}$ would make up the columns of U.

2.2 Eckart-Young Theorem

An important application of the SVD is in approximating matrices. For an $m \times n$ matrix A with rank r, we take each left and right singular vector and the corresponding singular value to factor A as

$$A = U\Sigma V^{\top} = \sigma_1 u_1 v_1^{\top} + \sigma_2 u_2 v_2^{\top} + \sigma_3 u_3 v_3^{\top} + ... + \sigma_r u_r v_r^{\top}$$

using the SVD. Let A_k be the sum of the first k terms in the previous factor form for $k \leq r$. According to the Eckart-Young Theorem, for any B with rank k, we have

$$\|A - B\|_F \geq \|A - A_k\|_F .$$

This also holds true for

$$\|A - B\|_{L2} \geq \|A - A_k\|_{L2}.$$

In other words, A_k is the closest rank k approximation to A.

2.3 Randomized SVD Algorithms

2.3.1 Randomized SVD

For very large matrices, directly computing the deterministic SVD of a matrix becomes computationally expensive. To combat this challenge, the Randomized SVD algorithm can be used to significantly speed up computation.

Let A be a real $m \times n$ matrix where $m >> n$. We can construct a random projection matrix P of size $n \times r$, where r is the target rank. For intrinsically low rank matrices, we can use a smaller value of r [8]. which will allow us to sample the column space of A. By multiplying A by this random projection matrix, we produce AP = Z, giving us a smaller matrix Z that has a high probability of retaining the same column space as our original matrix X.

We can then compute a QR factorization on Z, giving us an orthonormal basis for the column space of Z and thus an approximate orthonormal basis for the column space of A.

Afterwards, we use our original matrix A again by computing $\mathrm{Y} = \mathrm{Q}^{\mathrm{T}}\mathrm{A}$. Finally, we directly compute the SVD on this much smaller matrix Y. The singular values and right singular vectors of Y are highly likely to approximate those of X, giving us

$$\mathrm{Y} = \mathrm{U_Y}\Sigma_{\mathrm{Y}}\mathrm{V_Y^T} = \mathrm{U_Y}\Sigma\mathrm{V}^{\mathrm{T}}.$$

As our last step in this algorithm, we compute $\mathrm{U_A} = \mathrm{QU_Y}$ to obtain the left singular values for A. This gives us a low rank r SVD approximation for A:

$$\mathrm{A} = \mathrm{U_A}\Sigma\mathrm{V}^{\mathrm{T}}$$

In order to increase the accuracy, we can specify a small oversampling parameter p. Using this parameter, we add p extra columns to our random projection matrix P. Even with very small values of p, including some oversampling greatly improves this algorithm. This topic is further explained in [8].

2.3.2 Compressed SVD

The compressed SVD, proposed in [9], is another randomized SVD algorithm that combines the idea of the randomized SVD (Section 2.3.1) with ideas from compressed sensing, providing us with an algorithm that performs very well in image processing applications.

Suppose we want to compute a rank k approximation of an $m \times n$ matrix A. The first step, similar to the randomized SVD, is to sample the column space of A. We choose an oversampling parameter p, let $l = k + p$, generate a (sparse) random $l \times m$ matrix Φ, and put $Y = \Phi A$. Since the columns of Y are vectors randomly sampled from the column space of A, we have in high likelihood that the column space of Y is a decent low-rank approximation for the column space of A. Thus, to approximate the right singular vectors V of A, we compute the SVD of Y.

Compute the SVD of Y to obtain $Y = \tilde{U}\tilde{\Sigma}\tilde{V}^\top$., where we truncate to include only the first k singular vectors and their singular values. Compute the SVD of $A\tilde{V}$ as $A\tilde{V} = U\Sigma Q^\top$, and let $V = \tilde{V}Q$. The rank k approximation of A is then given by $A \approx U\Sigma V^\top$.

For more details, see [9].

2.4 Principal Component Analysis

One of the most useful applications of the SVD is principal component analysis, or PCA. Principal component analysis is a dimension reduction technique which selects the directions in which the data has the most variance. In this sense, PCA is optimal in how much information it captures for any given dimension. The next paragraph describes the procedure for performing PCA on a data matrix using an eigenanalysis.

Let X be an $m \times n$ data matrix representing n observations, each of which contain m variables. Assume X is centered by column means, so that the mean of each variable across all observations is 0. If X is not column centered, we may replace $X = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}$ with $\begin{bmatrix} x_1 - \mu_1 & x_2 - \mu_2 & \cdots x_n - \mu_n \end{bmatrix}$, where μ_i is the mean of x_1. Then, we are interested in the covariance matrix $XX^\top$. Note that $XX^\top$ is a symmetric, positive-semidefinite matrix, and so there exists an orthogonal eigendecomposition $XX^\top = PDP^\top$, where $P = \begin{bmatrix} u_1 & u_2 & \cdots & u_m \end{bmatrix}$ is orthogonal and $D = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_m)$ is diagonal. Assume that the entries in D are arranged such that $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_r$, and note that all of these are nonnegative. Then the principal components

are $u_1, \ldots, u_m$, and the variances captured by these principal components are proportional to their corresponding eigenvalues.

Interestingly, it is possible to perform PCA on a matrix without computing the covariance matrix. Let X be our matrix as above, and let $U\Sigma V^\top$ be its singular value decomposition. Then the covariance matrix of X is

$$\begin{aligned} XX^\top &= (U\Sigma V^\top)(U\Sigma V^\top)^\top \\ &= U\Sigma V^\top V\Sigma^\top U^\top \\ &= U\Sigma\Sigma^\top U^\top \\ &= U\Sigma^2 U^\top, \end{aligned}$$

since Σ is a diagonal, self-adjoint matrix. This is exactly the eigendecomposition of $XX^\top$—the principal components are the left singular vectors u_i, and the eigenvalues of the covariance matrix are the squares of the singular values $\lambda_i = \sigma_i^2$. Thus, one can perform a principal component analysis on a matrix by computing the singular value decomposition of the matrix and taking the left singular vectors.

In some situations, we may not know whether it is more useful to think of the "observations" in our data matrix as being in the rows or in the columns. In these situations, we may perform two separate PCAs on the matrix, one with row-mean centering and column-mean centering, and see which one offers the best interpretability in the context of the data. This is a consideration we make when using SVD techniques to analyze COVID-19 data in Section 4.3.

Zhang et al. provide an in-depth analysis of how using four different mean centering methods on the data matrix provide varying levels of complexity, performance interpretability, etc. in [16]. Our analyses in Section 4.3 utilize either the simple SVD with no mean centering (SSVD) or centering around the column means (CSVD). Other centering methods include row centering (RSVD) and double centering (DSVD), the latter of which involves centering using both the row and column means.

As an example, we use a data matrix in Section 4.3.2 in which the columns are days since January 22 and the rows are states, and the entries are new cases. For this specific data matrix, row mean centering would involve centering each day's new case count by the average new case count for all states on that particular day, while column mean centering would center each state's daily new case count around the average new case count for that particular state.

3 Methodology

In order to perform matrix computations on data D, it is necessary to represent the data as a matrix. If $\mathcal{M}$ is a transformation mapping D to its matrix form D, then we can perform a matrix operation f on D to produce the new data $f(\mathsf{D})$ as follows:

$$\mathsf{D} \xrightarrow{\mathcal{M}} D \xrightarrow{f} f(D) \xrightarrow{\mathcal{M}^{-1}} f(\mathsf{D}) \tag{1}$$

This is the general framework we follow when applying matrix computations to image, video, and audio media. In the following sections, we describe these transformations $\mathcal{M}$ for images, audio, and video, and thereafter make implicit use of (1) when appropriate. In addition, we describe the procedures involved in various applications of the SVD to image and video processing (3.1), audio processing (3.2), data analysis (3.3), and watermarking (3.4).

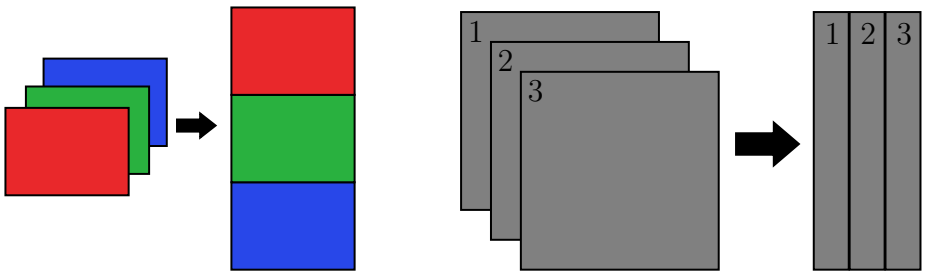
3.1 Image and Video Processing

3.1.1 Matrix Representation for Images and Videos

A grayscale image of size $m \times n$ is represented by an $m \times n$ array of luminosity values and can be operated on directly. A color image is represented by a $m \times n \times c$ array, where c is 3 or 4 which can be thought of as an $m \times n$ array of c-tuples representing the red, green, and blue, and potentially alpha components of each pixel. Such a color image is represented by a $m \times 3n$ matrix, produced by "stacking" the color channels.

A video can be thought of as an array of k images, each of which is represented by an $m \times n \times c$ array, where c is the number of color channels. Such a video can be represented as a $cmn \times k$ matrix, where each of the k columns is a "flattened" image.

Both of these procedures are visualized in Figure 1



(a) Color video (b) Video

Figure 1: Matrix representation of images and videos

3.1.2 Image Compression

The Eckart-Young Theorem provides us with a remarkably simple image compression algorithm: Simply use a low-rank approximation of your image, computed using randomized algorithms if more speed is necessary. The only complication here is that one must choose the rank of the approximation, and the desirable choice varies with the complexity of the image. Since the error in a rank k approximation A_k of a matrix A depends on the singular value distribution of $A - A_k$, or equivalently the singular values $\sigma_k + 1, \sigma_k + 2, \cdots, \sigma_r$ of A, one may use the singular value distribution of A to choose an appropriate approximation rank.

One method for choosing an approximation rank is using a "ratio" $0 < \rho < 1$ of the rank of A. For example, if $\rho = 1/2$ and $\operatorname{rank} A = 200$, then we would use a rank 100 approximation of A. One advantage to this method is that it selects higher ranks for images of higher ranks. A significant disadvantage, however, is that it it blind to the singular value distribution of A, leading to undesirable accuracy if A has high intrinsic rank or an unnecessarily high selected rank if A has low intrinsic rank.

A more sophisticated method is to select the smallest rank k such that $\|A_k\| / \|A\|$ surpasses some minimum value ρ, using your desired norm. In the nuclear norm, this is the minimum k such that $\frac{\sigma_1+\cdots+\sigma_k}{\sigma_1+\cdots+\sigma_r} \geq \rho$. The advantage to this approach is that with fixed α, the approximation error will be somewhat consistent regardless of the image used and its singular value spectrum. The disadvantage is that this requires computing at least the first k singular values of A, where k is unknown, making the use of randomized algorithms for efficiency improvements difficult.

3.1.3 Modifying Singular Values

Modifying the singular values of a matrix by executing some mapping, such as a logarithmic transformation or multiplication by a scalar, preserves some general characteristics within the data while distorting others.

One modification we use simply involves scaling the singular values by some scalar. We also add or subtract a scalar to each of the singular values, raise each singular value to some exponent, and compute the mapping $\log(\sigma + 1)^p$ on each singular value.

When modifying image matrices, computing the above logarithmic mapping appears to have the effect of making specific parts of the image more vibrant while making other less important parts of the image white. We can experiment with this by assigning each of the non-white pixels some scaling value s and assigning all white pixels a value of 1. We then multiply the pixels in the original image based on each pixel's corresponding value in the modified image. In effect, this allows us to accentuate the specific parts of the image highlighted in the modified image while keeping everything else the same, with the intention of highlighting important parts of the image.

As a brief technical description, this process is accomplished using Python code by reshaping our original image of the cat as one long column vector as well as the modified image. If a pixel in the reshaped modified image is equal to 0, we set it to 1, while if it is equal We then multiply each value in the original photo column vector by its corresponding value (1 or s) in the scaling matrix (i.e. similarly shaped as a long column). Finally, we reconstruct an image from this matrix. This produces an interesting effect on our reconstructed image, as is displayed and discussed in (4.1.3).

For matrices representing video files, the effects of singular value modifications are slightly different. Since we are essentially stacking all of the frames in our matrix instead of only the color channels of a single image, modifying the singular values of a video matrix doesn't have the same effect as computing an identical mapping on an individual frame from the video.

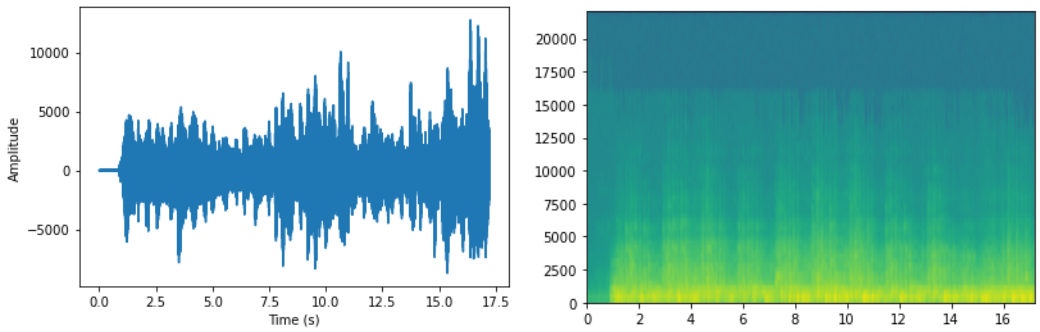
We experiment with various such mappings for audio and image matrices in Sections 4.2.2 and 4.1.3.

3.2 Audio Processing

3.2.1 Matrix Representation for Audio

Mono audio signals, in their raw form, come in as an array of N points sampled with frequency f_s (typically 44 100 Hz). To convert audio signals into matrices, we use the Short Time Fourier Transform (STFT), which is a time-frequency analysis technique. The output is a rectangular matrix where, roughly speaking, the entry in position (f, t) is a complex number which describes the amplitude and phase of frequency f at time t. For visualization, a *spectrogram* can be obtained from the STFT of a signal by plotting the magnitude squared of the entries of the STFT.

Figure 2 displays an example audio signal and its associated spectrogram.



(a) Audio signal (b) Spectrogram

Figure 2: An audio signal and its spectrogram

In particular, we use a Short Time Fourier Transform (STFT) with window size 256 and shift 128 to convert audio signals into matrices. For more information on the STFT, see [12].

3.2.2 Singular Value Modification for Audio

Modifying the singular values of the matrix representation for audio results in interesting changes in certain aspects of the audio, specifically amplitude and noise levels. We perform SVD on an audio matrix, and modify the diagonal entries of Σ.

Our experiment consists of two methods. The first is linear scaling by some scalar p. We take each singular value and multiply them by a positive

number p with p ranging from 0.5 to 4. The second method is exponential scaling by some scalar e. We raise each singular value to the e-th power. After the modifications on Σ, we reform the matrix by multiplying $U\Sigma^* V^\top$.

As U and V remain the same, the most dominant features of the audio are still preserved in the left and right singular vectors. The output audio will thus have the same melody line with modified amplitude and/or noise levels.

3.3 Data Analysis: COVID-19

3.3.1 Scree Plots

Before performing SVD on our datasets, we first need to determine the principal components to use, which is evaluated through scree plots.

A scree plot is a visual aid to determine the appropriate centering and the number of components [16] and shows the residual proportion of each principal component. As singular values compute variance explained by each singular vectors, one way to plot the scree plot is by showing the percentage of the singular value over the sum of all singular values, which produces the bar graph below.

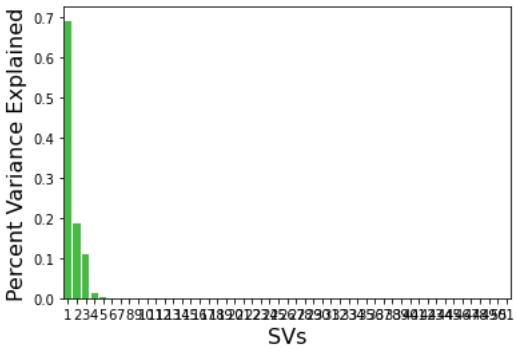


Figure 3: Scree Plot of Cumulative Deaths Nationwide

In the example scree plot, more than 99% of the variances are explained by the first three singular vectors, so we can perform analysis on those singular vectors in later steps.

3.3.2 Scatter Plots of Principal Components

In our data matrix, each row represents an observation, and each column represents an index of the observation. Plotting the most dominant left singular vectors and projecting the data matrix onto the most dominant right singular vectors are both very useful in identifying the outliers among the row observations.

By performing SVD on the data matrix, we have matrix U containing the left singular vectors, Σ containing the singular values, and V containing the right singular vectors. According to the scree plot of the data matrix, the first three singular vectors cover most of the variances, so we produce a scatter plot with the three axis being the first three columns of U. As each point represents one state, the 3-D scatterplot puts all states into clusters by their most dominant features in the dataset, and similarly a 2-D scatterplot projected onto the first two columns of U produces a similar plot with one less feature.

As the left singular vectors contain the dominant features of each row observation, plotting the right singular vectors will give us the general trend.

It is also useful to add a grouping variable onto the scatterplot. By color coding the different rwo observations, we can see the correlation between the coded variable and the general observed trend.

3.3.3 Rank 1 Outer Product Plots

Recall that the SVD of a matrix can be written as a sum of scaled rank one outer products of the left and right singular vectors: $X = U\Sigma V^\top = \sigma_1 u_1 v_1^\top + \sigma_2 u_2 v_2^\top + \cdots + \sigma_r u_r v_r^\top$, where $r = \operatorname{rank} X$. Another useful way to analyze a set of data is to look at the individual rank one components $u_i v_i^\top$. The idea is that the first rank one component $u_1 v_1^\top$ should represent the most dominant feature or trend in the data, the second component $u_2 v_2^\top$ should show up the second most dominant feature or trend, and so on.

In Section 4.3.2, we use this method to analyze COVID-19 cases and deaths across all 50 states and Washington, DC. Similarly, in Section 4.3.3, we use this technique the analyze the trend of COVID-19 cases and deaths in Florida in response to actions taken by the government to close down and open up facilities.

In Section 4.3.2, we also analyze surface plots representing these rank 1 matrices and attempt to explain what patterns these matrices are displaying

in the data. This is done by comparing notable changes in the plots to rankings and timelines provided by [5] and [4].

3.4 Watermarking

Another use for the SVD is in digital ownership protection. When digital property like videos, photos, or music is being sent and shared through the internet, it is often important to the owners of this property to be able to embed some kind of mark into their media. This allows them to prove ownership of the original media should the need arise.

To do this, various watermarking schemes have been developed which utilize the SVD matrices and "hide" some watermark within the original data. We explore how some of these schemes address specific concerns, including perceptibility (how different our watermarked media appears when compared to the original media), security (whether our scheme can be manipulated by a third party to fake ownership), and robustness against distortions, attacks, or data compression. We also briefly discuss extraction error and how various scaling factors used while embedding a watermark have a small effect on accuracy during the extraction process.

In general, a watermarking scheme works as follows: after choosing some real number $\alpha > 0$ which determines the "intensity" with which the watermark is embedded, we use a watermarking embedding function E to embed a watermark matrix W into a matrix A, giving us $E(A, W, \alpha) = (A_W, K)$—the watermarked matrix and key, respectively. Given this key and a possibly distorted watermarked matrix A_W^*, we can extract the possibly distorted watermark W^* using the extraction function $E^{-1}(A_W^*, K, \alpha) = W^*$.

3.4.1 Liu & Tan Watermarking Scheme

The Liu & Tan watermarking scheme, described in great detail in [13], is a widely-cited and foundational scheme in SVD-based watermarking techniques. This method involves embedding a watermark into a matrix's singular values and then computing this new matrix's SVD as a means of embedding information, as is outlined in the following steps:

1. $A \to USV^\top$
2. $S + \alpha W \to U_W S_W V_W^\top$

3. $A_W \leftarrow US_W V^\top$

4. $K \leftarrow (S, U_W, V_W, \alpha)$

5. Return (A_W, K)

This scheme relies on replacing the singular values of A with the singular values of $S + \alpha W$ in order to reconstruct a watermarked piece of media.

In Section 4.1.3, we visually see how when the singular values are slightly modified in some way, the major characteristics of an image are still preserved. The final step of the Liu & Tan embedding process is essentially taking advantage of this useful aspect of the SVD.

The Liu & Tan scheme requires knowledge of U_W, S, V_W, and α, which can be thought of as our keys to prove ownership.

To attempt to extract our watermark from a (possibly distorted) watermarked image A_W^*, we compute the following steps:

1. $A_W^* \rightarrow U^* S_W^* V^\top$

2. $D^* \leftarrow U_W S_W^* V_W^\top$

3. $W^* \leftarrow \frac{1}{\alpha}(D^* - S)$

While the Liu & Tan method is robust against certain distortions, there are some security issues for watermark extraction using this scheme that are improved upon in other schemes.

3.4.2 Jain et al. Watermarking Scheme

One issue with the Liu & Tan watermarking scheme, as we demonstrate in Section 4.4.1, is that a good approximation of a watermark can be extracted from an image even if it wasn't embedded. This makes the watermarking scheme insecure in that it is difficult to use to prove ownership.

In [10], Jain et al. propose a modification to the Liu & Tan watermarking scheme, where instead of embedding the entire watermark in the singular value matrix of an image, we only embed the principal components of the watermark. If we want to embed a watermark W into a matrix A to obtained the watermarked matrix A_J, we use the following scheme:

1. $A \rightarrow USV^\top$

2. $W \to U_W S_W V_W^\top$

3. $S_1 \leftarrow S + \alpha U_W S_W$

4. $A_W \leftarrow U S_1 V^\top$

5. $K \leftarrow (A, V_W, \alpha)$

6. Return (A_W, K)

For an alternative perspective, note that

$$\begin{aligned} A_W &= U S_1 V^\top \\ &= U(S + \alpha U_W S_W) V^\top \\ &= USV^\top + \alpha U U_W S_W V^\top \\ &= A + \alpha U U_W S_W V^\top. \end{aligned}$$

Given the (possibly distorted) watermarked image A_W^*, the original image A, the right singular vectors V_W of the watermark, and the scaling factor α, we can extract the (possibly distorted) watermark W^* using

$$W^* \leftarrow \alpha^{-1} U^\top (A_W^* - A) V V_W^\top.$$

This alternative formulation of the Jain et al. watermarking scheme will be useful to us.

3.4.3 Modification to the Jain et al. Watermarking Scheme

Here, we propose a modified watermarking scheme that preserves the desirable properties (security and robustness to distortions) of the Jain watermarking scheme while also increasing imperceptibility of the mebedded watermark for a given scaling factor α. With notation as in Section 3.4.2, the proposed watermarking scheme is as follows:

Watermark embedding is given by the following:

1. $A \to USV^\top$

2. $W \to U_W S_W V_W^\top$

3. $A_W \leftarrow A + \alpha U_W S_W V^\top$

4. $K \leftarrow (A, V, \alpha)$

5. Return (A_W, K)

Watermark extraction is given by the following:

$$W^* \leftarrow \frac{(A_W^* - A)VV_W^\top}{\alpha}$$

Notice that in the Jain et al. watermarking scheme, we add a scalar multiple of the matrix $UU_W S_W V^\top$ to the original matrix A. In contrast, in the modified Jain watermarking scheme, we add a scalar multiple of the matrix $U_W S_W V^\top$ to A.

3.4.4 Evaluating Watermarking Schemes

In order for a watermarking scheme to be employable in proof of ownership, it is vital that given a watermarked image A_W, an adversary cannot produce a phony original image A_p and watermark W_p such that the watermark W can be extracted from A. A basic test of security, then, is given by the following procedure, using an image A, two watermarks W_1, W_2, a watermark embedding function E, and a watermark extraction function E^{-1}:

1. $W_1, K_1 \leftarrow E(A, W_1, \alpha)$

2. $W_2, K_2 \leftarrow E(A, W_2, \alpha)$

3. $W_2^* \leftarrow E^{-1}(A_1, K_2)$

4. Compare W_2^* to W_2

In Section 4.4, we use this procedure to evaluate the basic security properties of the watermarking schemes described in sections 3.4.1, 3.4.2, and 3.4.3.

4 Applications

4.1 Image and Video Processing

4.1.1 Image Compression

Although images often have hundreds of thousands of degrees of freedom in their matrix representations, a reasonable approximation of the image can

typically be obtained using much less information. In the language of linear algebra, matrices representing images usually have low intrinsic rank. As an example, Figure (4), displays various images along with their ranks and singular value spectra.

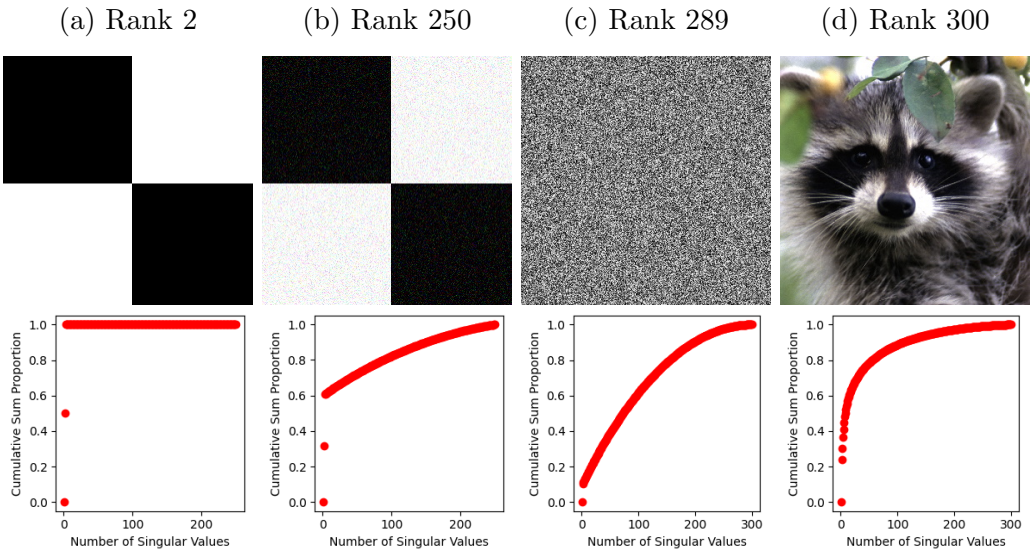


Figure 4: Various images their ranks, and their singular value cumulative sum (nuclear norm) plots

Notice that while adding noise to the checkerboard image (Figure 4b) does increase its rank, the singular value spectrum shows us that most of the variation in the image is still accounted for by the first two principal components.

4.1.2 Video Background Extraction

A low-rank approximation can be used to extract the background from a still video with moving subjects. Using the method described in Section 3.1, we use a rank one approximation to extract the background from security camera footage of a school. The result is shown in Figure 5.

(a) Frame from school video

(b) Background extracted using rank 1 approximation

Figure 5: Extracting the background from security camera footage

With this technique, the best results are achieved if the camera is still and the background is static. In the case of the school video, the camera is still and the only moving figures are the students.

4.1.3 Modifying Singular Values for Images

.

In this section, we explore the effect that various mappings on our singular values have on our reconstructed image. We use the public domain chelsea.png cat image found in the Imageio standard images library [1].

Scaling the singular values by some factor c results in interesting modifications to our final image. Multiplying them by a positive number seems to increase the brightness of the final image, while scaling them by a decimal value between 0 and 1 darkens the final image. This is illustrated in Figure 6, where each of the images are full-rank with only modifications to the singular values.

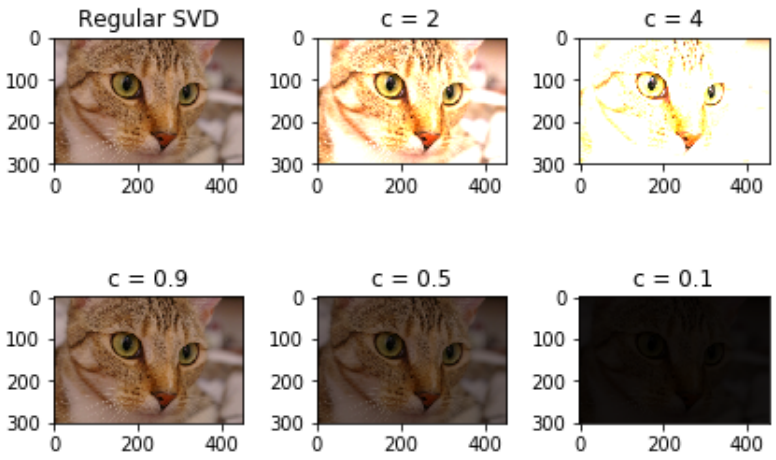


Figure 6: $c\sigma$ Mapping

Similarly, we see in Figure 7 that adding or subtracting to the singular values by large numbers results in significant changes in the final image color.

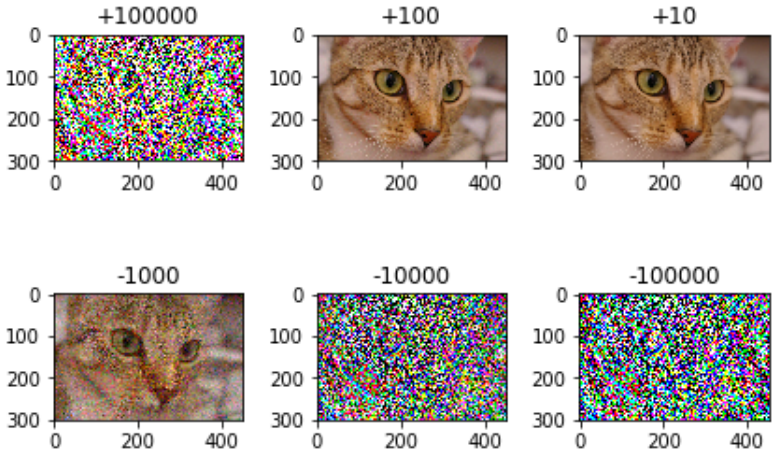


Figure 7: Adding to σ Mapping

Raising the singular values to some exponent n also has an effect. Values greater than $n \approx 1.2$ seem to result in singular values too far out of range to yield anything other than a white block, but fine changes between $n \approx 1.05$ and $n \approx 1.1$ appear to modify the softness of the image. This is shown in Figure 8.

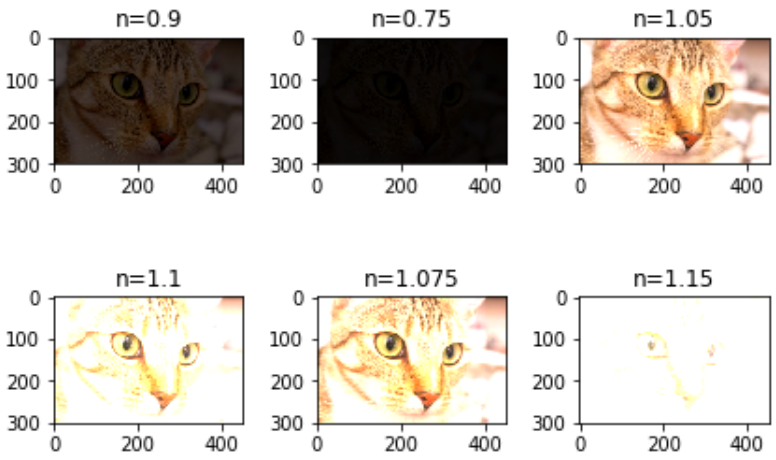


Figure 8: σ^n Mapping

To compare this mapping against other images, we include examples of 3 other photos with distinct subjects in Figure 9.

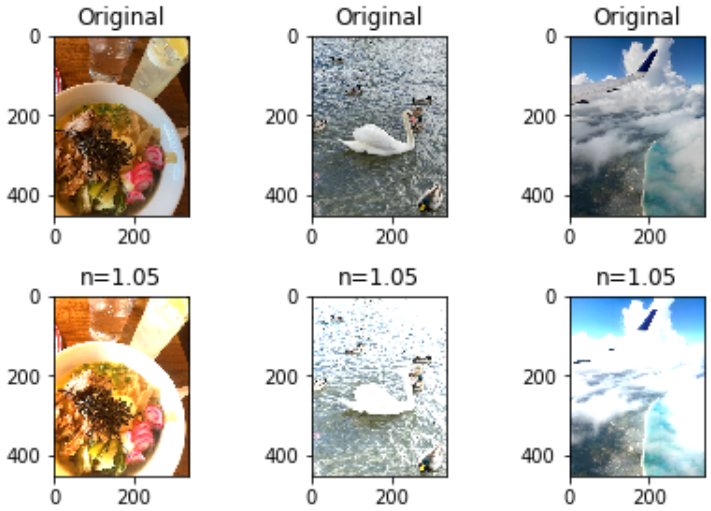


Figure 9: σ^n Mapping: Additional Examples

Another mapping function that can be used on the singular values is $\log(\sigma + 1)^p$. While this does generate some noise, some of the final images appear to be more vibrant, particularly for $4 \leq p \leq 5$. This is shown in Figure 10.

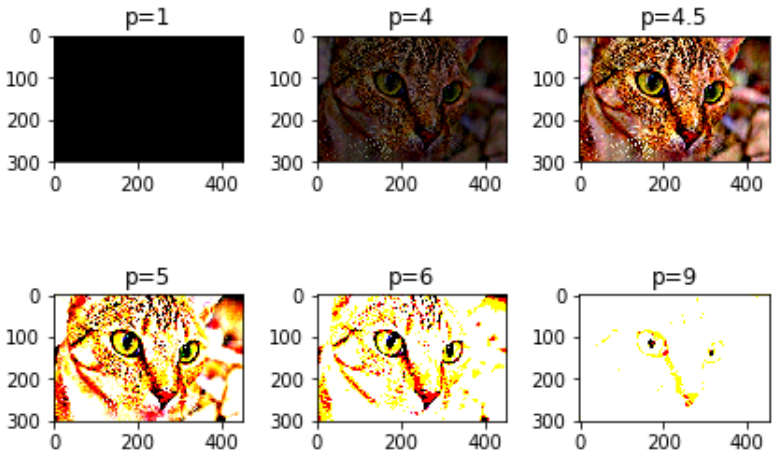


Figure 10: $\log(\sigma + 1)^p$ Mapping

Extending this further, we can compute the method described in Section 3.1.3 on our cat image, where we leave white pixels unchanged and assign non-white pixels some scaling value to increase their effect when combined with the original image. For example, for $p = 9$, the majority of the entries in the matrix representing the image would have a value of 1. Various examples of this process are seen in Figure 11. In the last two, the rank of the scaling matrix was truncated at ranks 5 and 20, respectively.

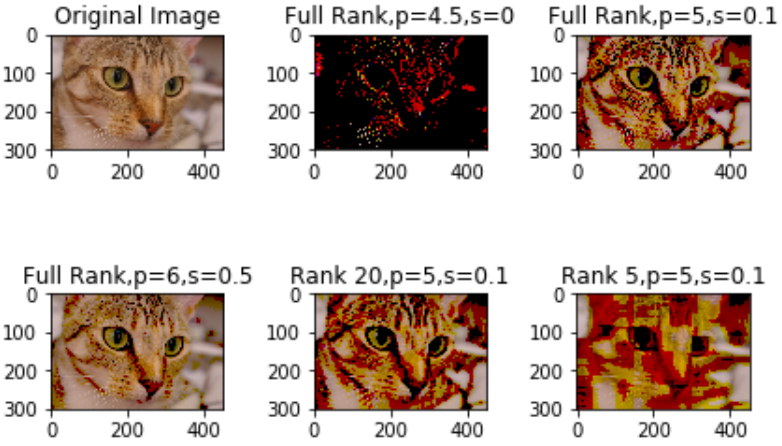


Figure 11: Combined Mapping and Scaling Method

Note that taking the $p = 6$ singular value mapping and assigning each of the non-255 elements a scaling value of $s = 0.5$ seems to accentuate certain features of the image reconstruction.

Attempting this same mapping and modification with $p = 6, s = 0.5$ on three additional personal images has different effects, as is shown in Figure 12. Note that while these effects are less pronounced for the left and center images, applying this singular value modification on the right image causes certain blocks to look unnaturally colored. Variation in the background and subject matter appear to be significant in how the singular values should be modified to sharpen/accentuate features.

Figure 12: Combined Mapping and Scaling Method: Additional Examples

This approach to making use of the modified singular value image is not perfect. For example, when reshaping the modified image as a column vector and reassign pixels values of 1 or s, we do not acknowledge how this is technically only reassigning the red, green, or blue component of a specific pixel.

Say we have a pixel in the original, unstacked image, and that its corresponding pixel in the unstacked version of its logarithmic modified image has positive values for both red and green, but a value of 0 for blue. Similarly, imagine that another pixel on the original image has a corresponding modified pixel with only a positive value for red. When our combined mapping and scaling method is applied, the first pixel will be scaled by $2s$, while the second will be scaled by s, even though they are both non-white and that was the only parameter that we were initially trying to modify our image

with. While this captures some variation between colors, it does so purely based on color variation between red, green, and blue, and not on a spectrum of intensity that would be more useful to us for image modification.

Similarly, when assigning each non-zero pixel in the stacked, modified image a value of s, we completely discard any information about the variation within the non-white regions. For example, consider the $p = 9$ example in Figure 10. When completing our algorithm, we ignore how the non-white region is darker for the pupil of the cat's eye but less so for the surrounding parts of the face. Even so, our algorithm essentially discards these differences in intensity.

In the future, we can perhaps investigate how to capture the variation and refine our algorithm by using a heat map or some other method to visualize and develop a better . This can allow us to still use the information captured by a reconstructed image from a logarithmic singular value mapping, while better preserving the variation within image characteristics that this mapping provides.

Adjusting and fine-tuning these singular value modification methods may be useful for various means of photo editing, such as sharpening, softening, or saturating.

4.2 Audio Processing

4.2.1 Audio Compression

Audio files are preserved in a different way compared to images, but they can still be compressed using low rank approximation after being transformed into a matrix by using STFT.

(a) Original (b) Rank 25 (c) Rank 5

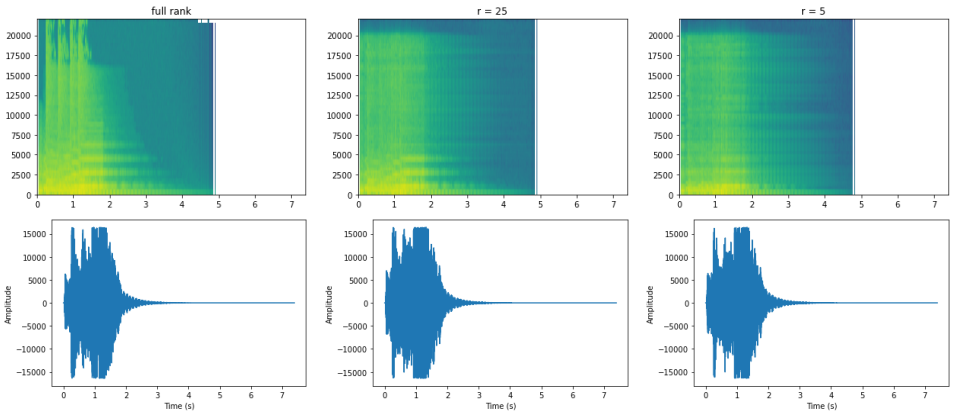


Figure 13: Low rank approximation of a spectrogram representing an audio signals

We perform low-rank approximations on a news opening audio clip [2]. As observed in the spectrograms, the audio gets noisier when the rank is 5 while the rank 25 approximation gives a good approximation to the original sound. The amplitude increases as rank decreases in the audio signal graph, but changes in volumes are not much perceptible and covered by the noise resulting from the approximation.

4.2.2 Modifying Singular Values for Audio

Using the method introduced in section 3.2.2, we modify singular values by some scalar of the rank 25 approximation of the audio output from the previous section.

As depicted by the diagrams below, when we multiply each singular value by a positive integer p, the amplitude of the audio increases as the scalar p increases. We also tried splitting the singular values into two parts and perform different scaling on the first half and the second half. Result shows that there is no perceptible difference on modifications performed on the lower singular values.

(a) p = 4 (b) p = 2 (c) p = 0.5 (d) p = 0.5 + 10

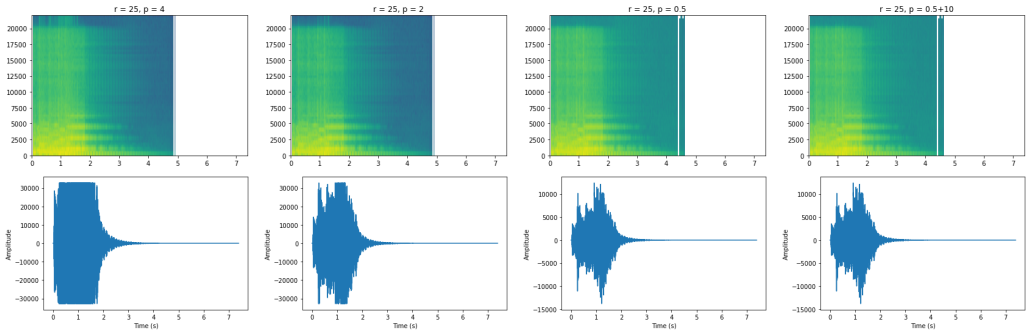


Figure 14: Modification of singular values by a linear scalar for audio represented by their spectrograms and audio signals

Differences are more significant when we modify the singular values exponentially. When we raise each singular value to its eth power, both noise level and overall amplitude increases as e increases. The output can be disturbing for e greater than 2 and can be too soft for e smaller than 0.5.

(a) e = 2 (b) e = 1.5 (c) e = 0.7

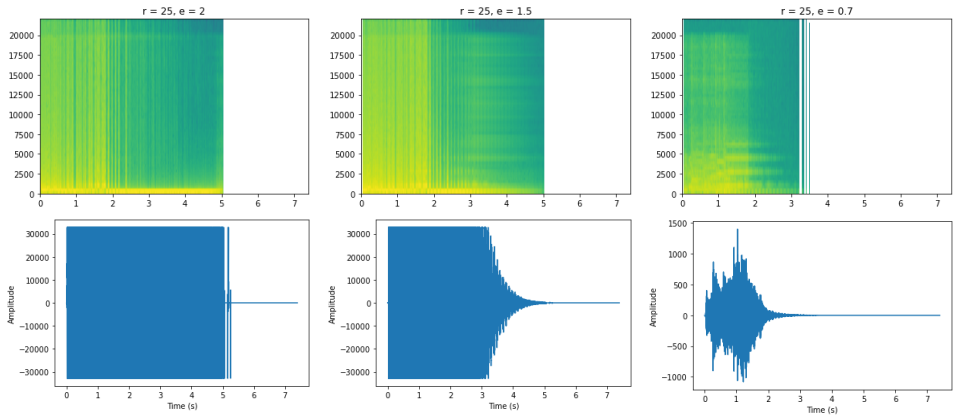


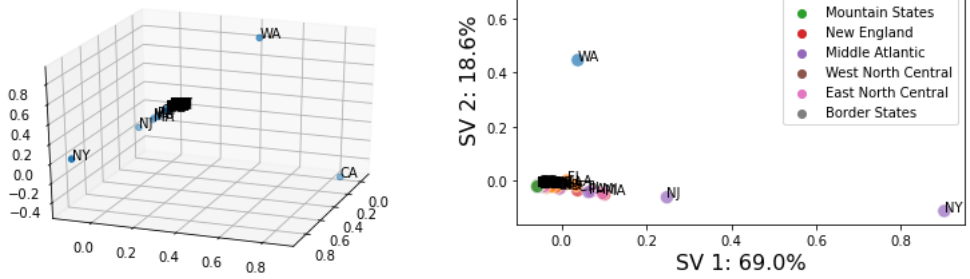
Figure 15: Modification of singular values by an exponential scalar for audio represented by their spectrograms and audio signals

4.3 Data Analysis: COVID-19

We perform our data collection on the COVID-19 dataset acquired from USA facts[3]. The data are available starting from January 22, 2020 and are updated daily. The ending date of the collected date is subject to change across different sections of our data analysis based on the download date. Our data analysis are performed on two data matrix from the website. Both have county as row variable, date as column variable, and have the corresponding State data available in a separate column. Each entries contain the cumulative death number and the cumulative case number for each county on the date in the two data matrix respectively.

4.3.1 State Clustering

Cumulative Death Number for States To look at the clustering among different states, we first sum up the entries of each county by the state they belong to. As we treat Washington D.C. as an individual observation, we now have 51 total row observations together with the 50 states. Using the method introduced in Section 3.3.3, we plot the first three left singular vectors of the data matrix.



As observed in the scatter-plot above, the four notable outliers are NY, NJ, CA, and WA. Each of the outlying states having significant death rates and different patterns in the outbreak timeline that group them away from the cluster of all other states in three different directions.

WA is where the first outbreak of the virus happens in the United States and has an outbreak curve a few weeks ahead of all other states. NY and NJ are the main epicenters following WA that still had the greatest death number till the time this dataset was collected. CA has an consistent increase

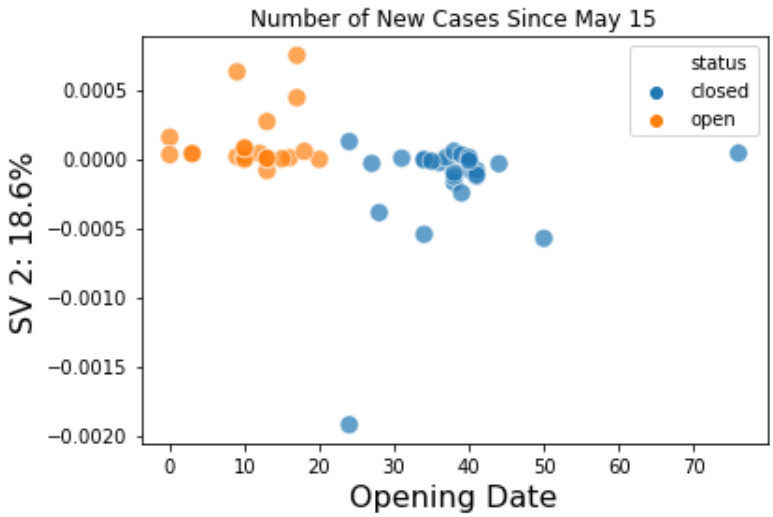
in death number at an increase rate of 1% throughout the outbreak and is now the newest state with the most number of cases. With these three regions being the epicenter of the United States at different times of the outbreak, it makes sense for them to be away from the cluster in three different directions.

Note that NY and NJ have similar patterns in the outbreak but differ in severity. This characteristic results in the two states being away from the clustering at the same direction but in different distance in the graph above.

Daily New Cases Using similar techniques as stated in the previous section, we produce a data matrix with the reported cases for each state on each date. As the raw data contains cumulative case number in each entry, we take the difference between consecutive columns and produce a data matrix with the newly reported cases in each entry. One thing to note is that the data relies on daily government-reported data, meaning that cumulative numbers fluctuate, and that certain states can have have negative new death number on some days.

By projecting the data matrix onto its second right singular vector, we observe the general trend across the selected time period. As we are trying to investigate in the influence of statewide reopening plans, we select the columns from May 15, 2020 to July 5, 2020.

We set the grouping color variable as whether the state was open in the beginning of May, and leave a 14-day grace period for the virus to be tested. By comparing the reopening plan across different states, we observe that the phase with 50% retail rate is stratified into two groups, either at the end of April and at the beginning of May or at the end of May towards early June. For certain states that do not have detailed reopening plans, we chose the date of the most aggressive reopening phase as its reference for our grouping variable. The x-axis is labeled as the date duration between April 21, 2020 and the reopening date of each state decided as previously mentioned. The date April 21 is chosen as it is the earliest reopening date across all 50 states despite the one state that did not issue a stay-at-home order. The color-coded clustering graph is resulted as below.



We observe for the states that have 50% retail industry open at the beginning of May to be located on the top half of the graph, and for the ones that remain closed to be located on the bottom graph. We here can interpret the second singular vector as closely related to the curve flattening of each state.

4.3.2 States

Daily New Deaths for States As of July 9, 2020, USA Facts only provides raw data which includes cumulative information on COVID-19 deaths in all United States counties. This data can be manipulated to produce a new data frame for daily new deaths in each state.

To visualize this data set, we construct a series of surface plots inspired by the exploratory data analysis research on network traffic data provided in [16]. This plots are displayed in Figure 16. The first plot describes the original data about new deaths in each state, where the x-axis records days since January 22 and the y-axis corresponds to each state in alphabetical order (there are a total of 51 entries since Washington, DC is included).

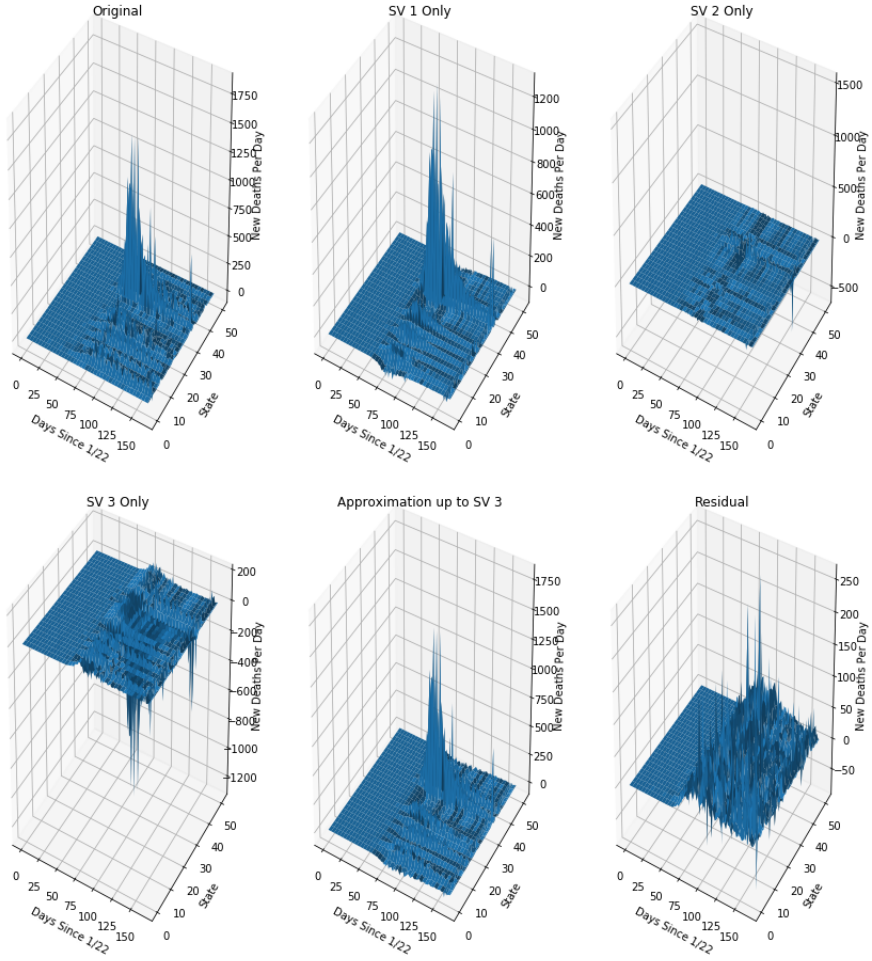


Figure 16: New Deaths Nationwide (CSVD)

As a comparison, the plots in Figure 17 are in the exact same format as Figure 16, but use the cumulative deaths data.

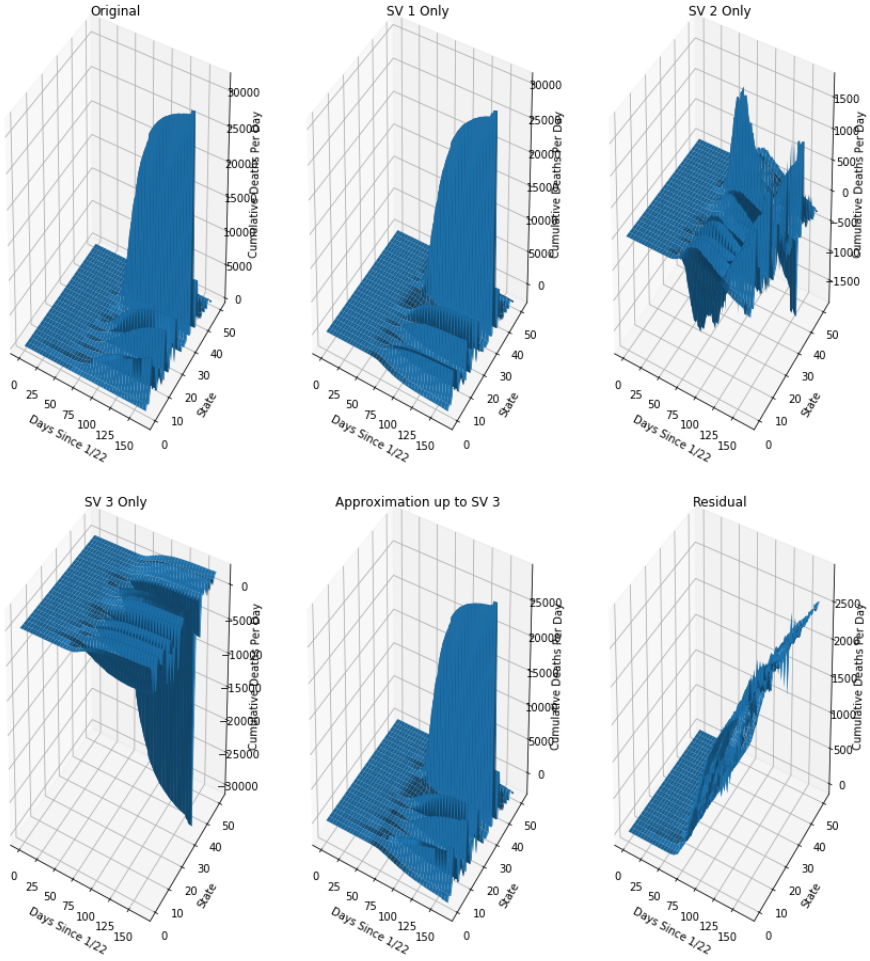


Figure 17: Cumulative Deaths Nationwide (CSVD)

The above plots were constructed by centering the data around the column mean. Scree plots like the one provided in section 6.3 can be used to find an optimal mean-centering method. To experimentally compare how different means may illustrate different aspects of the data, we provide some plots using no centering (SSVD), row centering (RSVD), and double centering (DSVD) in Figure 18.

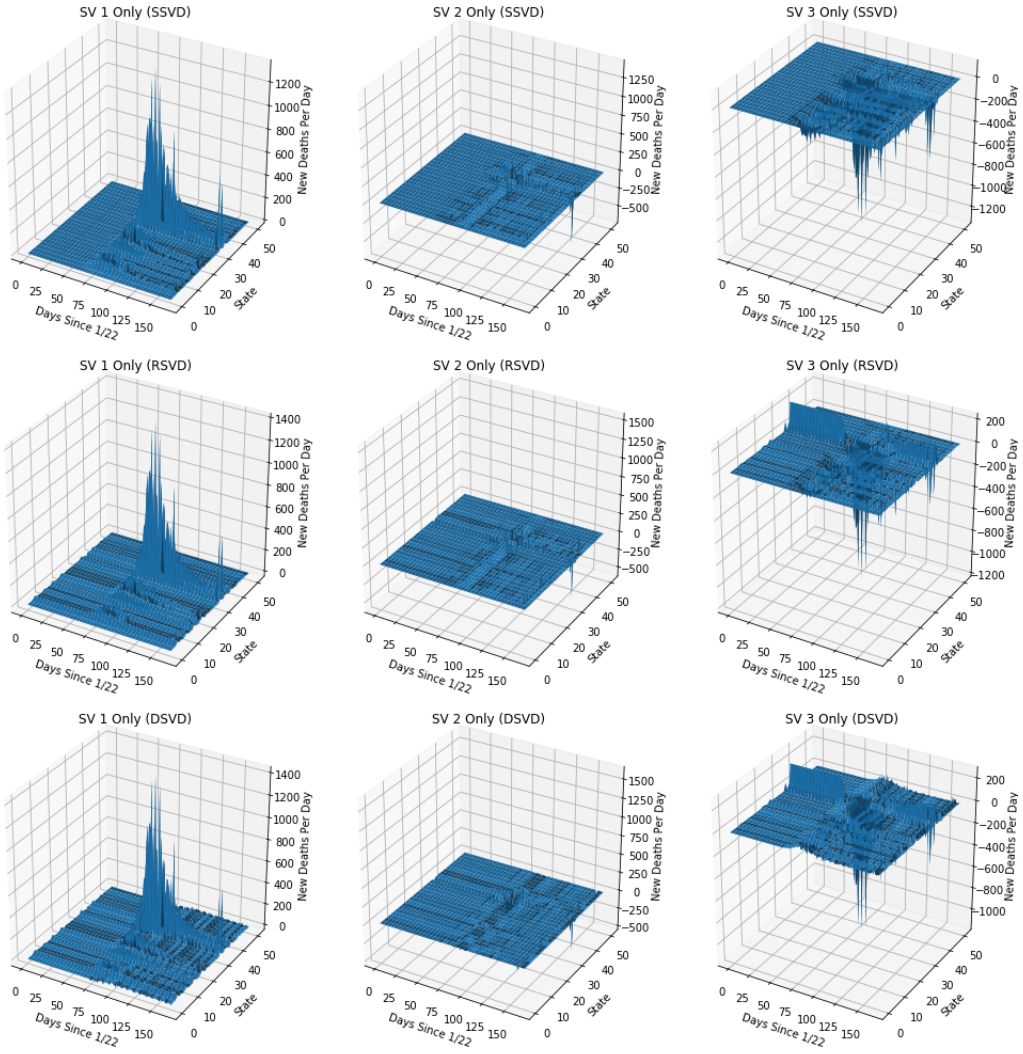


Figure 18: New Deaths Using Different Centerings

While analyzing the plots illustrating the original data for new COVID-19 deaths, it was noticed that some of the new death counts were negative. Similarly, there was an abnormally large spike for New Jersey (1887 new deaths) at the end of June, well after New Jersey experienced its largest wave in COVID-19 deaths. In their page on sources and methodology for their data collection, USA Facts acknowledges this and explains that due to the fluctuation of reported deaths in goverment-provided information, some

cumulative death counts may decrease on a daily basis.

New Cases Per Day Due to the lag between initial diagnosis and deaths that may arise from a positive case, it is hard to glean information from SVD approximations from the above plots. In order to better identify patterns in the ongoing situation regarding COVID-19 in the United States, we compute the same plots as above but specifically for new cases diagnosed per day in Figure 19. This was also obtained through the data provided by USA Facts, but this time as downloaded July 13, 2020. Thus, the data included ranges from January We again use the column mean centering for this plot.

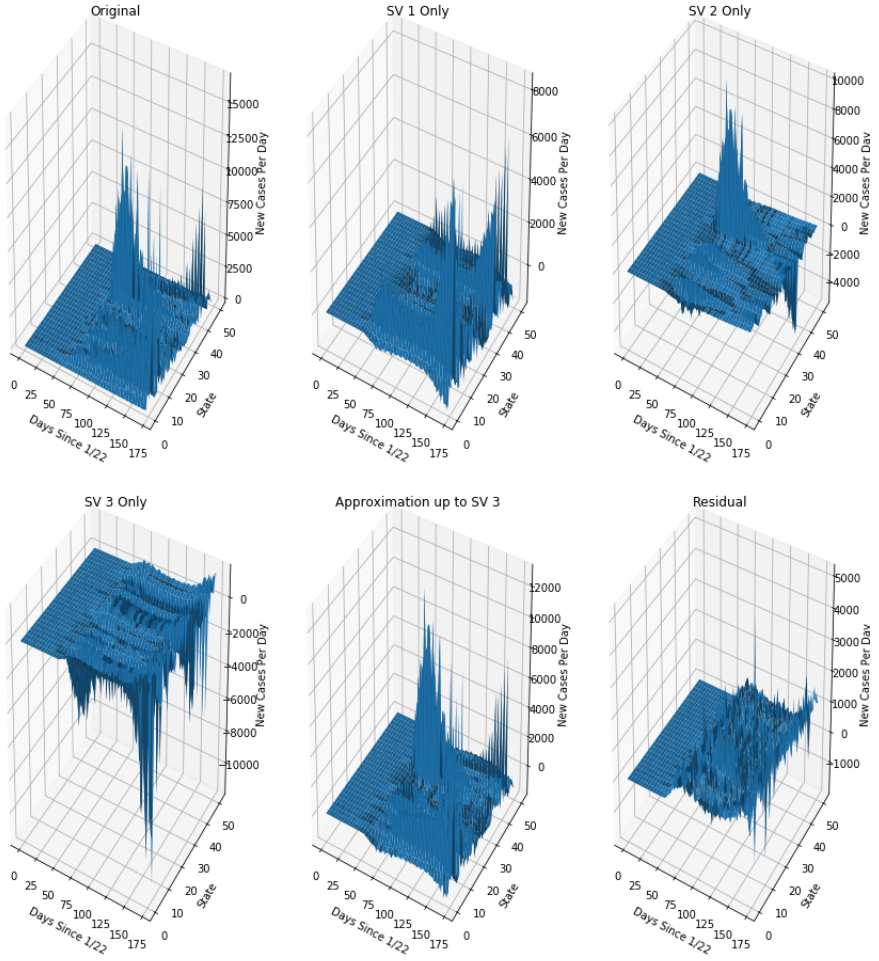
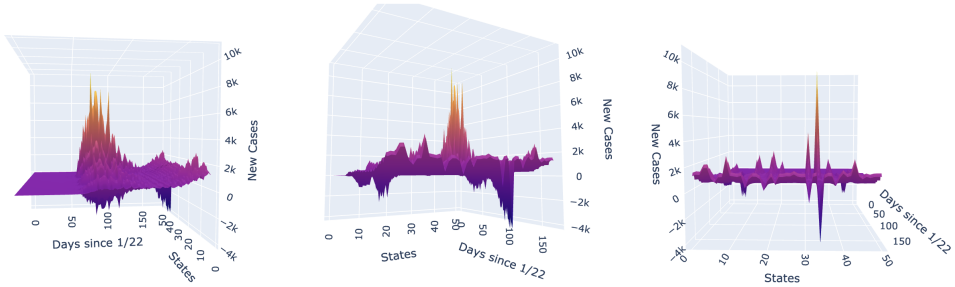


Figure 19: New Cases Nationwide (CSVD)

The SV 1 plot clearly provides a general approximation of the most dominant trends in the data. However, a more in-depth analysis of the SV 2 and SV 3 plots reveal interesting patterns in the data. To further understand these plots, interactive plots were constructed in Jupyter Notebook using Plotly to determine which peaks correspond to each day and state. Three viewpoints of interest from the interactive plots for SV 2 are provided in Figure 20.



(a) (b) (c) Snapshot 3

Figure 20

In Figure 20 (a), there are obvious dramatic peaks for most states around March and April, particularly in New York and New Jersey. However, there also seem to be some states that are negative around this same time frame, as is better seen in Figure 20 (b). It is important to note that a state showing negative behavior in this plot doesn't actually mean that there were negative new cases. Rather, a negative value indicates that whatever pattern this plot displays is being experienced in the opposite direction at that particular moment in time.

Upon further inspection, just three states in this plot were negative while the remainder were positive or plateauing: California (4), Florida (9), and Minnesota(43). Researching more about cumulative case data shows that most of these states experienced relative plateaus in cumulative cases around this time frame when compared to other states.

Figure 20 (b) displays the more recent behavior for states in the second singular value term, or the most recent and second-most-dominant traits of the new cases data as of July 12. From left to right, there are six states with a noticeable downward slope from this view: Illinois (14), Massachusetts (18), Minnesota (23), New Jersey (31), New York (34), and Pennsylvania (38). These are all states that experienced intense peaks earlier in the pandemic and have since experienced a decrease.

Figure 20 (a) also shows a relative lull in dramatic spikes around the 130-150 day range. This phase is marked by an interesting change for New York (34), which goes from having the tallest peaks of all states to dipping into the negative range. This transition from positive to negative for New York corresponds to days 138 and 139, or June 8-9. Interestingly, June 8 marks the 100th day since the first COVID-19 case in the state, as well as

the beginning of the first phase of reopening in New York City.

New Jersey's progression in SV 2 is similar to New York's, but on a smaller scale. For New Jersey (31), the transition from positive to negative occurs from June 15 to June 16. This exactly corresponds to the first day of Stage 2 reopening in NJ, which includes outdoor dining and special events, nonessential retail stores, and child daycare centers, to name a few. While this does not align exactly with the policy change that took place on June 8 in New York, it is interesting to consider how these transitions tend to correspond to significant policy changes in each respective state.

We can explore this hypothesis further by looking at other states that didn't transition to positive to negative, but instead transitioned from periods of relative plateau to sharp increases. For most states, the plateau changes to an unusually high increase around days 140-150, or June 10-20. As an example, we look at Indiana (13), which plateaued for a long time in the 200-300 range before starting to rapidly climb into the 600+ range around June 20. Indeed, June 12 through July 3 is the time frame for Indiana's Stage 4 reopening plan, which does correspond to a relatively significant (although not dramatic) increase in cases in the original data.

As an interesting outlier, consider Massachusetts (19). While Massachusetts also experienced a spike earlier in the pandemic, its plateau stays consistent past day 150, even while almost all other states experience some kind of stark change. While Massachusetts has also been executing a gradual reopening plan, it has also been among the more aggressive states in combating coronavirus.

This inspires us to look at aggressiveness as another metric beyond specific reopening dates. WalletHub provides a thorough analysis of this concept, computing a ranking of all 50 states according to 51 relevant and carefully weighted parameters in [5]. New York ranks number one at the top of the list as of July 14, with New Jersey ranked 5 and Massachusetts ranked 8. While the WalletHub ranking we viewed was published on April 7, 2020, the information it provided helped us develop an idea of what pattern the SV 2 plot is depicting, and the ranking is likely to still be relevant even a few months later into the pandemic.

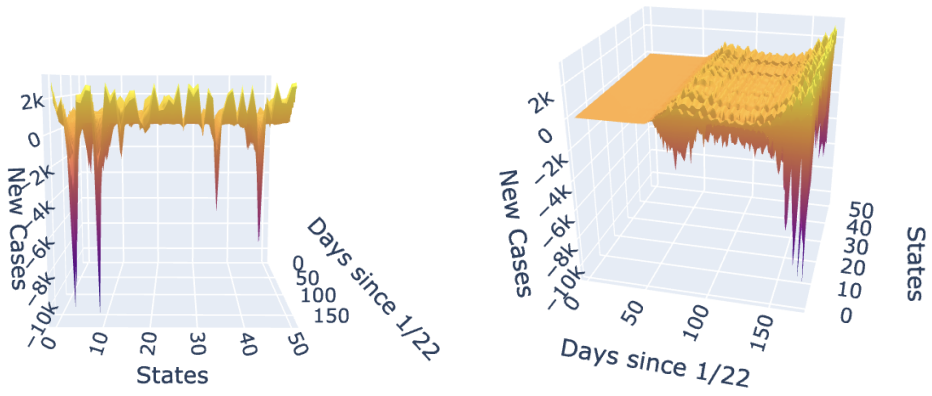
Comparing peaks/plateaus with this ranking (among other coronavirus news sources) seems to indicate that the SV2 plot has something to do with the aggressiveness of state lockdowns or safe reopening policy (or effects of aggressive policy). States which are dealing with the negative repercussions of a relatively lax COVID-19 containment policy in June and July, such as

Alabama or Florida, are reflected as such in the SV2 plot. On the other hand, the downward trends following a transition from positive to negative numbers seem to indicate a "safe" reopening in New Jersey and New York in terms of new cases diagnosed.

It should be understood that this interesting behavior around days 140-150 doesn't precisely align with policy changes in each state. It is also obvious that most states wouldn't initiate reopening plans in the first place unless there was evidence (i.e. a plateau or downward trend in recent new cases) to support such a policy change. Likewise, access to testing is highly variable even within states and reopening policy is highly subjective and different depending on cases experienced. However, the behavior of the SV 2 plot during and following this period provides some interesting insight into the effects of aggressive policy in affecting counts of new cases.

Similarly, we could also view our SV 2 plot as representing some measure of flattening the curve of new infections, since states would have only made policy changes following evidence that the curve had flattened somewhat and it was safe to begin a phased reopening. This hypothesis would align with our above observations and connections.

Looking at the SV 3 plot is also of interest. Specific viewing angles of the SV 3 plot are provided in Figure 21.



(a) (b)

Figure 21: Selected Views of SV 3 Only Plot

In Figure 21 (a), we see four huge negative spikes: California (4), Florida (9), New York (34), and Texas (43). These are precisely the four states with

the highest overall new cases tolls as of July 14.

Meanwhile, in Figure 21 (b), we see that almost all but the 4 aforementioned states are relatively flat for most of the plot. However, they almost all start to increase towards the end.

Observing the values in the z-axis for y = 170 (the last day in the data frame we downloaded, or July 12), the highest values are z = 2,932.275 for Vermont (46) and z = 2,922.471 for Hawai'i (11). Hawai'i and Vermont are the two lowest states in the country in terms of cumulative cases as of July 14. While other interacting factors would contribute to the total case count of a state, it appears that the SV3 plot has something to do with cumulative positive cases in states.

4.3.3 Counties in Florida

Cumulative Deaths in Florida Here, we focus our attention on the state of Florida. Before beginning our analysis, we look at the generalized scree plot for the daily cumulative COVID-19 deaths in Florida. Note that in the data matrix, the rows correspond to counties and the columns correspond to dates. Thus, row mean centering centers each day's cumulative deaths by the average cumulative deaths for all counties that day, while column mean centering centers each county's daily cumulative deaths by the average cumulative deaths per day for that county.

Regardless of centering, we can see in Figure 23 that most of the variation in the data is captured in the first three SVD components, to which we now limit our analysis. In Figure 24, we plot the daily cumulative COVID-19 deaths in Florida over time, as well as the SVD components using each type of centering.

First, notice that the first principal component, which captures the most of the variation in the data, accounts for the general trend of deaths, as can be seen in the SV1 component plots. This holds true for all states.

For Florida, we now focus on Pinellas County and Broward County. We can see in Figure 22 that the death rate in Broward County decreses as time goes on, while the death rate in Pinellas County saw a sharp increase after day 150. These differences in trajectory seem to be captured in the SV2 component, where Pinellas increases and Broward decreases towards the end. The sharp change in trajectory towards the end of the time period for Pinellas county, although not completely apparent in the original plot, seems to coincide with an extreme (in comparison to other states) deviation

of the rank three approximation from the original data. This makes sense since trajectory is captured in the first principal component, and deviations in this feature account for a large proportion of the residual.

Figure 22: Daily cumulative COVID-19 deaths for each Florida county

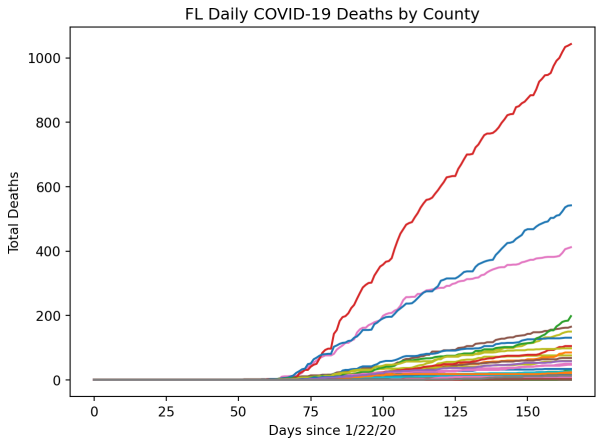
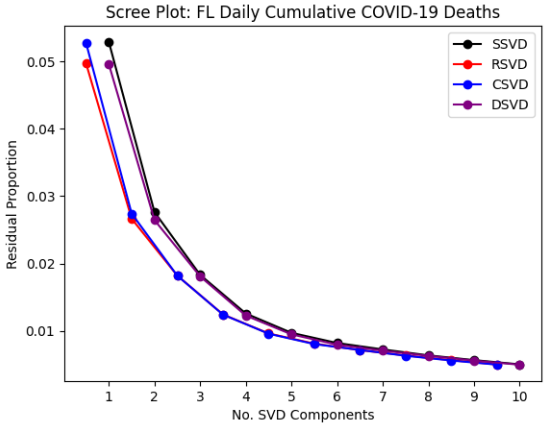
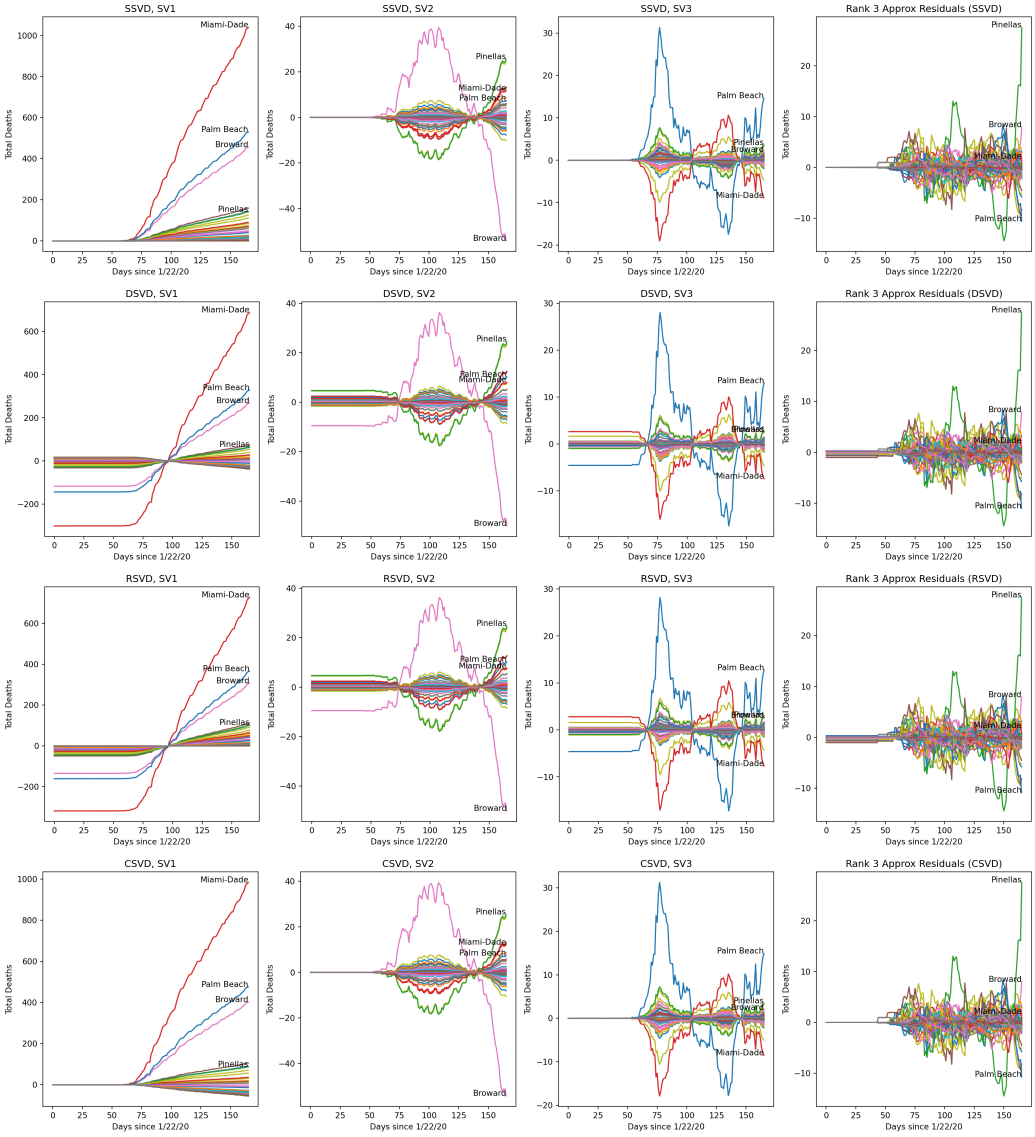


Figure 23: Generalized scree plot for daily cumulative COVID-19 deaths in Florida counties



Cases in Florida We now turn our attention to the number of known COVID-19 cases over time in Florida, organized by county and day, from data provided by USA Facts. In our analysis we also use information on the

Figure 24: SVD components and rank 3 approximation residuals for daily cumulative COVID-19 deaths in Florida counties



dates of opening and closing policy decisions in Florida, provided by Johns Hopkins University. The scree plots for daily new cases and total cases by day are shown in Figure 25.

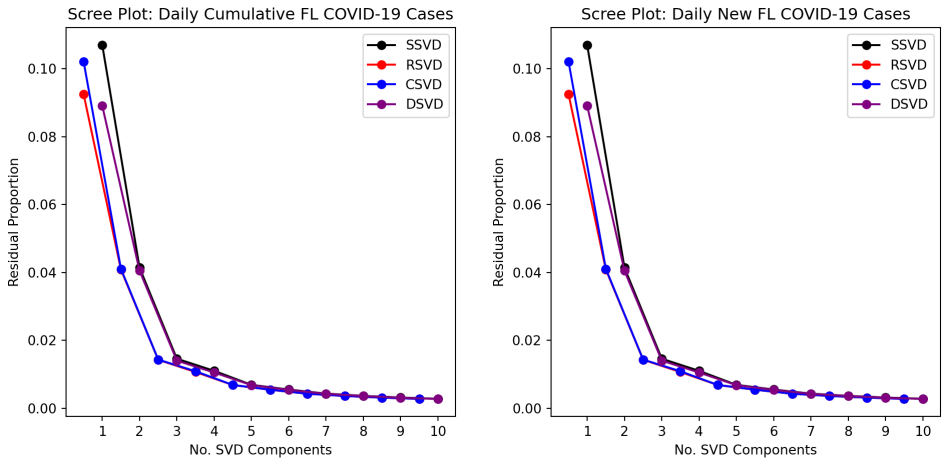


Figure 25: Scree Plots for Florida Cases

These scree plots show that most of the variation in the data is captured in the first three principal components and that mean centering has little effect. Thus, we focus out analysis on the first three principal components using the uncentered data.

In Figures 26 and 27, we plot the known cumulative and new cases over time and the SVD components. Red vertical lines represent policy decisions towards the closure of facilities, while blue vertical lines represent policy decisions towards the opening of facilities. There is typically a fourteen-day lag between infection and reporting of a new case, so we shift the case line plots back by fourteen days to align policy decisions and the time at which we should see any of their affects on case numbers.

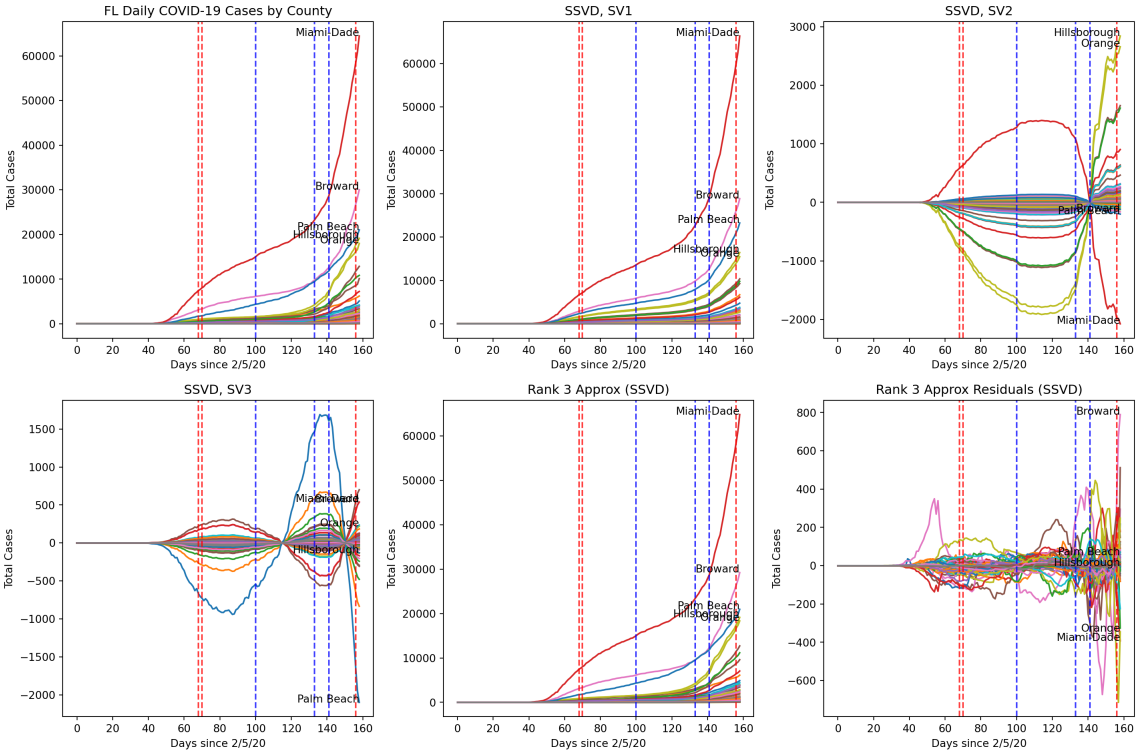


Figure 26: SVD Plots for Daily Cumulative COVID-19 Cases in Florida, with points shifted to the left by fourteen days, with red and blue dotted lines to represent closing and reopening of facilities.

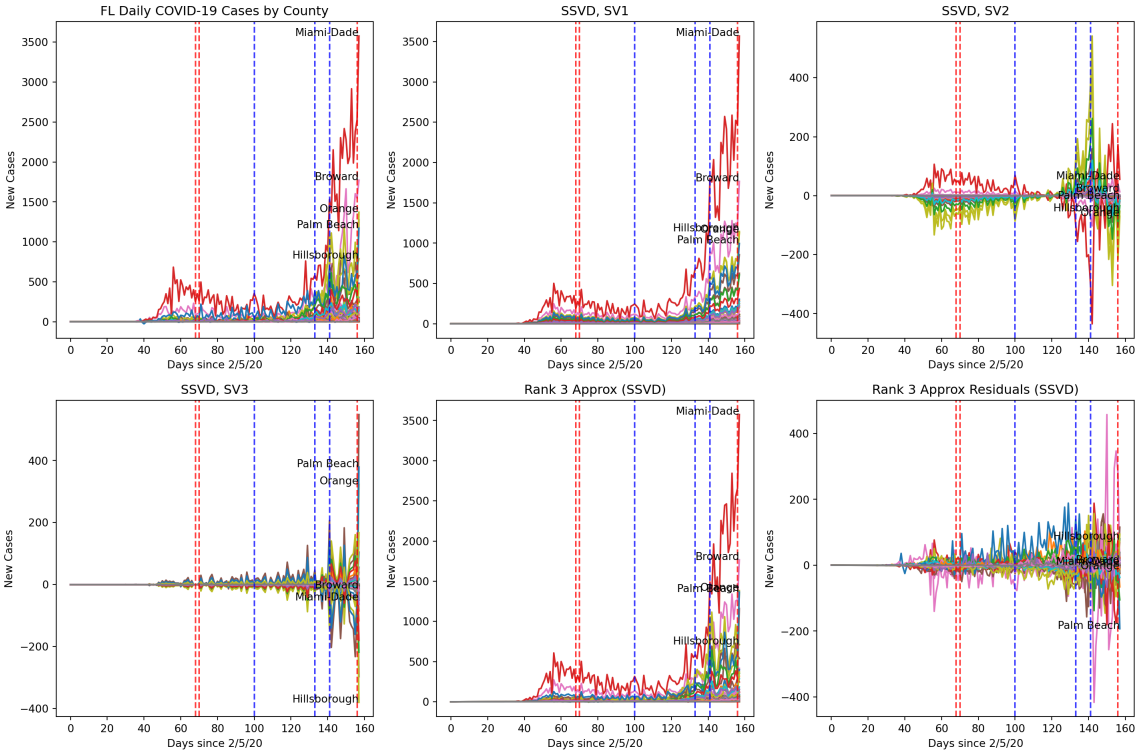


Figure 27: SVD Plots for Daily New COVID-19 Cases in Florida, with points shifted to the left by fourteen days, and with red and blue dotted lines to represent closing and reopening of facilities.

In Figure 27, we can see that the final two blue vertical lines, which represent the reopening of various facilities at half capacity, are quickly followed by a significant spike in cases, especially in Miami-Dade county. The rank 1 plots, however, are quite noisy and difficult to interpret.

The plots in Figure 26 are much less noisy. Similar to the COVID-19 death plots, we see a prominant "bump" in the SV2 and SV3 plots. Interestingly, the decision to allow education systems to reopen aligns is exactly fourteen days before SV2 plot's intersection with the horizontal axis.

To generate Figure 28, we apply the function $f(n) = \log(1+n)$ to each of the entries in the data matrix and then perform the same analysis as above to the modified matrix.

An interesting feature of this plot is that in the SV3 plot, the zero-points all come within two weeks of an opening or closing policy decision made by

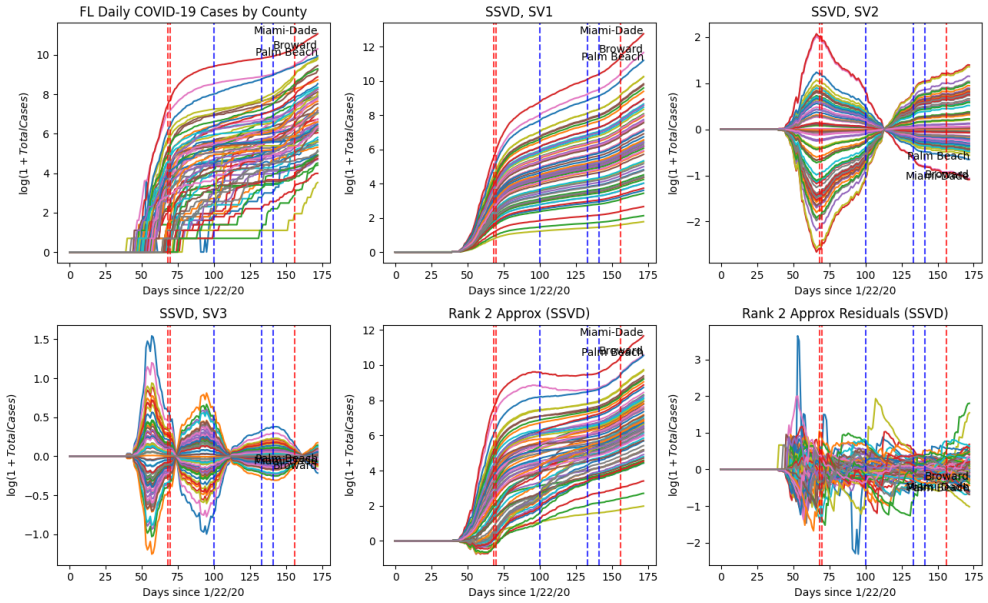


Figure 28: SVD Log Plots for Daily Cumulative COVID-19 Cases in Florida

the Florida government. It is feasible to imagine that the SV3 plot, to some extent, captures the effects of these decisions on the number of cumulative cases in each county.

Additionally, notice that in the SV2 plot, Miami-Dade and Broward counties peak at very similar values, while Palm Beach peaks at a much lower value. This is interesting because Miami-Dade, Palm Beach, and Broward counties are all beach counties, geographically adjacent to one another. One would expect that they behave very similarly in terms of case numbers, but palm beach deviates from Miami-Dade and Broward Counties. Additionally, this peak coincides with the statewide stay-at-home order issued on April 1st, 2020.

Finally, in the Rank 2 approximation plot in Figure 28 Miami-Dade and Broward counties show a "dip" after the stay-at-home order issued around April 1st, while Palm beach does not. This is consistent with the Miami-Dade and Broward curve "flattening" in the original data, while the cases in Palm Beach continued to grow exponentially. These observations, along with the Palm Beach's peak in the SV2 plot being much lower than the Miami-Dade and Broward counties, indicate that the SV2 plot captures the "flattening" of the case curve, relative to the general trend in the state, after the issuance

of the statewide stay-at-home order.

4.4 Watermarking and Steganography

4.4.1 Liu & Tan Watermarking Scheme

The Liu & Tan watermarking scheme changes the singular value spectrum of the watermarked image while leaving the principal components unaffected. Because of this, embedding a watermark in an image only enhances some of the already-present features of the image, making the watermark remarkably imperceptible for reasonably high values of α. Using the image and watermark shown in Figure 29, this is demonstrated in Figure 30.

(a) Original image [1] (b) Watermark [2]

Figure 29: Raccoon and fox images used in experiments

(a) $\alpha = 1$ (b) $\alpha = 0.5$ (c) $\alpha = 0.25$ (d) $\alpha = 0.1$

Figure 30: Images watermarked using Liu & Tan scheme

[1] "Young Raccoon in Crab Apple Tree" by Bill Buchanan, licensed under Public Domain Mark 1.0

[2] "Red Fox" by Jean Beaufort, licensed under CC0 1.0

One advantage to the Liu & Tan watermarking scheme is that it is extremely robust to distortions, including geometric distortions such as rotations. For this experiment, we embed the fox watermark into the raccoon image with $\alpha = 0.1$, rotate the watermarked image at various angles, and extract the watermark from the distorted images. The results are shown in Figure 31.

(a) 1° (b) 25° (c) 45° (d) 90°

Figure 31: Top row: Rotated watermarked images; Bottom row: Extracted watermarks

The main disadvantage of the Liu & Tan watermarking scheme, as noted in [15], is that it fails the basic security test outlined in 3.4.4. Figure 32 shows the results of the basic security test for the Liu & Tan watermarking scheme.

(a) Original image

(b) Watermark

(c) Phony watermark [3]

(d) Watermarked image

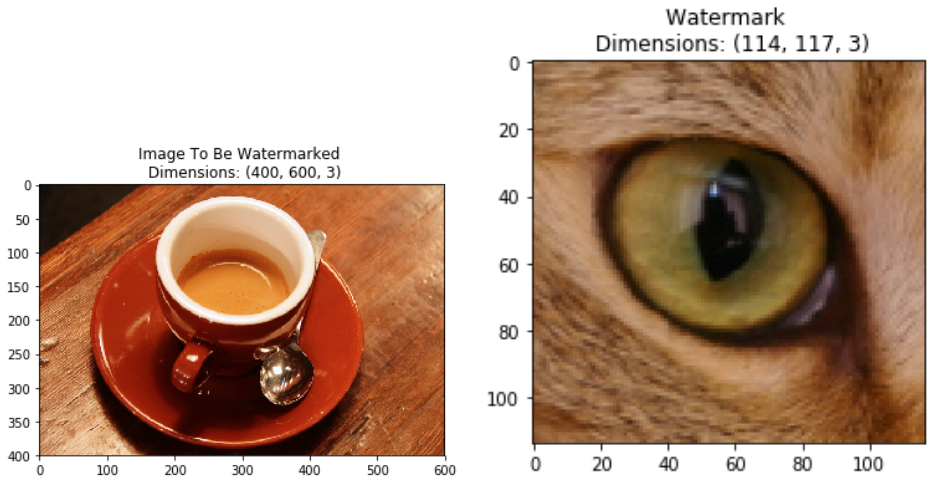
(e) Phony watermark extracted

Figure 32: Basic security test for Liu & Tan watermarking scheme

The optimal scaling factor may be different depending on our desired robustness against specific attacks, the nature of the images being used, and the extent to which we would like to conceal the existence of a watermark. Using these parameters, we explore various avenues that can help us determine this optimal scaling factor.

For the following plots, we used images from the Imageio image library. The cat's eye photo was cropped from a larger photo of the cat's face found in [1] in order to ensure that it was small enough to be embedded in the coffee cup image [14].

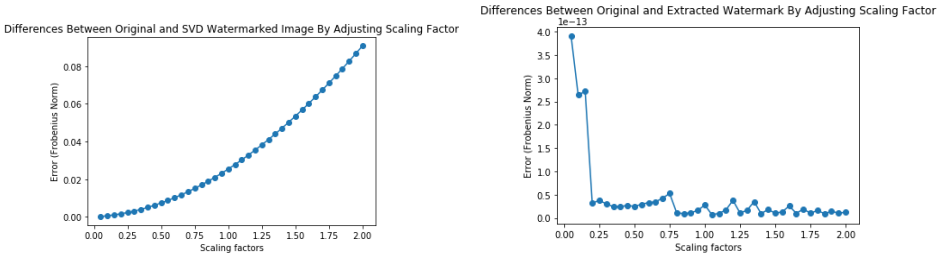
[3] "Siberian Husky Dog Pet" by Andrea Stöckel, licensed under CC0 1.0



(a) Coffee Cup (b) Cat's Eye

Figure 33: Imageio Sample Images

The following plots demonstrate how increasing the scaling factor results in a larger effect on the image, providing us with a measure of perceptibility. However, an increased scaling factor also allows us to extract our watermark with greater accuracy (although this improved accuracy may be too small to be significant).



(a) Perceptibility (b) Extraction Error

Figure 34: Perceptibility and Extraction Error for Image Watermark, Liu & Tan Scheme

To visualize this, we provide some examples using various scaling factors below. Note that a is equivalent to α in the labeling of these examples.

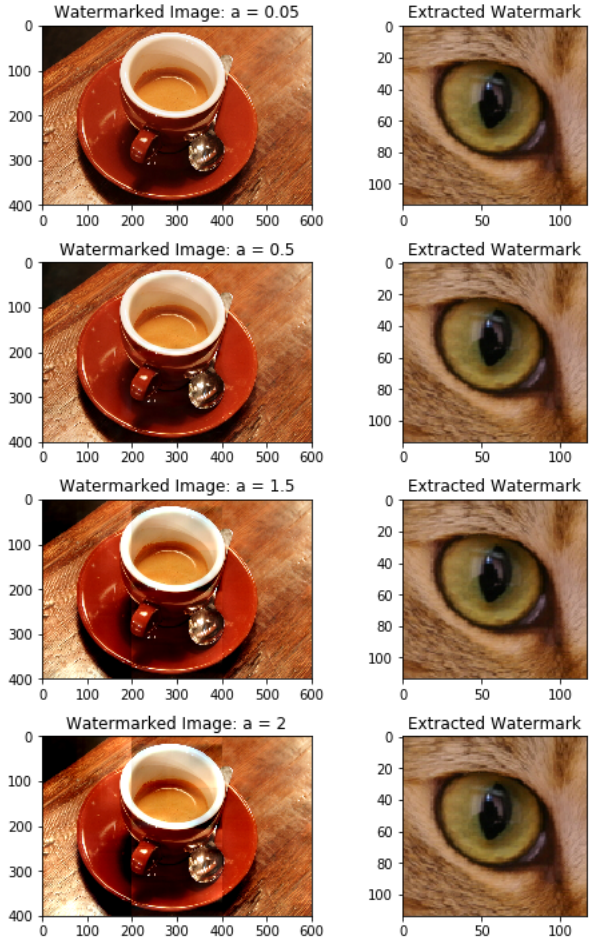


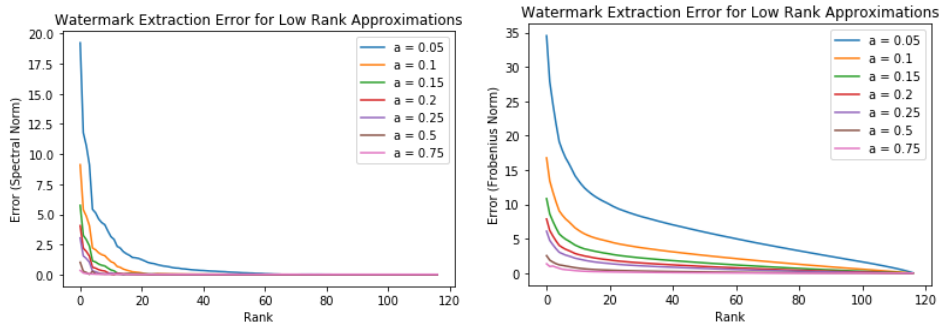
Figure 35: Examples for Image Watermark, Liu & Tan Scheme

These plots and examples illustrate the tradeoff between the scaling factor a, the accuracy of the extracted watermark. This issue is explained in further detail in [13].

As images and other forms of digital media are sent and shared through the internet, they often undergo data compression. Thus, we are curious as to whether a watermark can still be extracted from such a compressed image using the Liu & Tan scheme.

To construct the following plot, we compute low-rank deterministic SVD approximations of the watermarked image using various scaling factors and attempt to extract our watermark from the image. This allows us to study

how robust various scaling factors are against a simple distortion such as a reduced rank approximation via the SVD. The relative error is taken by calculating the absolute difference between the norms of the extracted watermark and original watermark and then dividing this difference by the norm of the original watermark.



(a) Spectral Norm Relative Error (b) Frobenius Norm Relative Error

Figure 36: Low Rank Extraction Plots, Liu & Tan Scheme

Regardless of the norm, we unsurprisingly see that a lower scaling factor is the least robust against low rank approximations, while a higher scaling factor is more robust. However, we notice that this difference is less extreme for scaling values greater than $a = 0.15$.

To better understand this, the following side-by-side examples have been generated to visualize the Liu-Tan algorithm's robustness against low-rank approximations. Note that in all example images using low rank approximations, including following sections, we use $a = 0.05$ as our embedding scaling value.

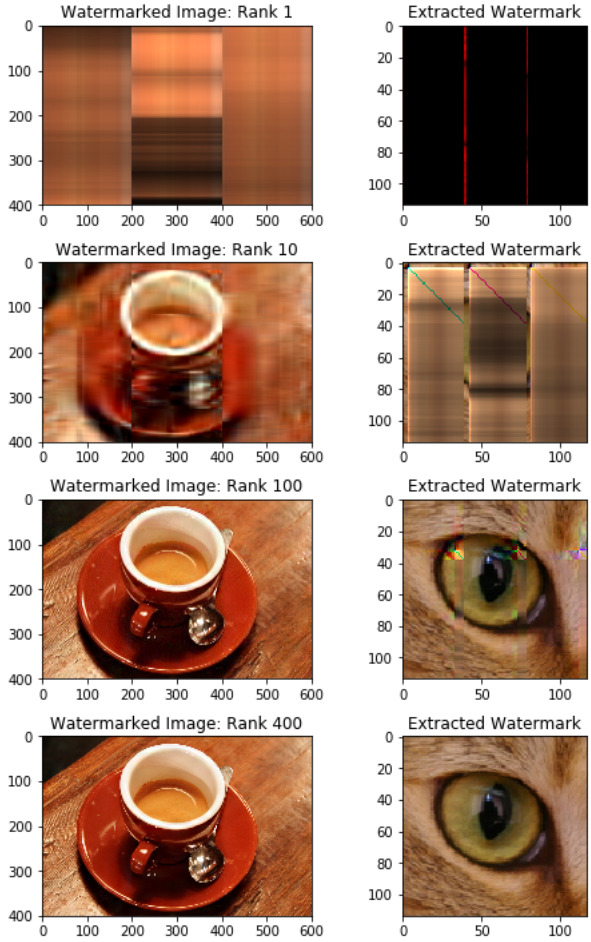


Figure 37: Low Rank Examples for Image Watermark, Liu & Tan Scheme

Cropping is one very straightforward image processing operation that can easily be utilized as an attack. Robustness against such simple attacks is a particularly relevant concern for watermarking and steganography.

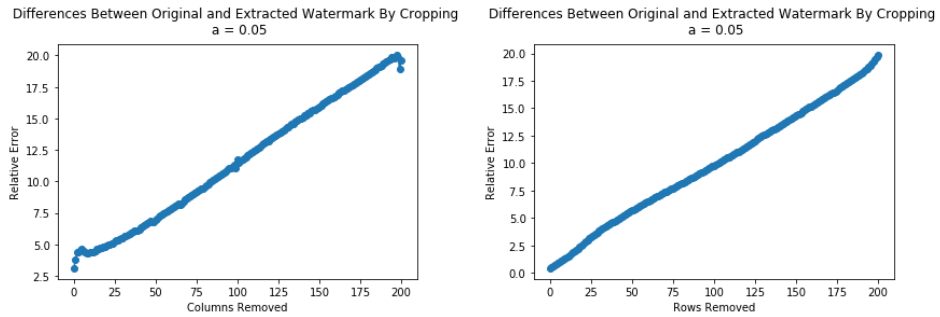
To assess the Liu-Tan algorithm's robustness against cropping, we can calculate the relative error between the extracted watermark from the uncropped image and the extracted watermark from the cropped image.

Note that this process requires knowledge of the original dimensions of our watermarked image, since we pad our cropped image before computing the SVD so that the singular value and vector matrix will match dimensions. Thus, if we were given a cropped image without any knowledge of its original

size, we would be unable to recover our watermark.

We pad our matrices with zeros several times throughout this process. We first compute our usual watermark and watermark extraction scheme, but leave our extracted watermark unstacked in terms of color channels at the end. Then, we pad our stacked, cropped, watermarked image with zeros to match the shape of the stacked watermarked image. We then attempt to extract the watermark from the padded cropped image using U_W, S, $V_W^\top$, and a (all of which need to be known from the original image and watermarked image). We then remove the padded zeros from this extracted watermark so that it matches the shape of our original watermark. We now have two stacked watermarks, one extracted without cropping and one extracted with cropping, which we can use to compute error.

The below plots are calculated by plotting the relative error described earlier. This allows us to incrementally simulate the process of cropping.



(a) Cropping Columns (b) Cropping Rows

Figure 38: Cropping Plots for Image Watermark, Liu & Tan Scheme

To visualize this, some examples of watermarks extracted from cropped images are provided in Figure 39.

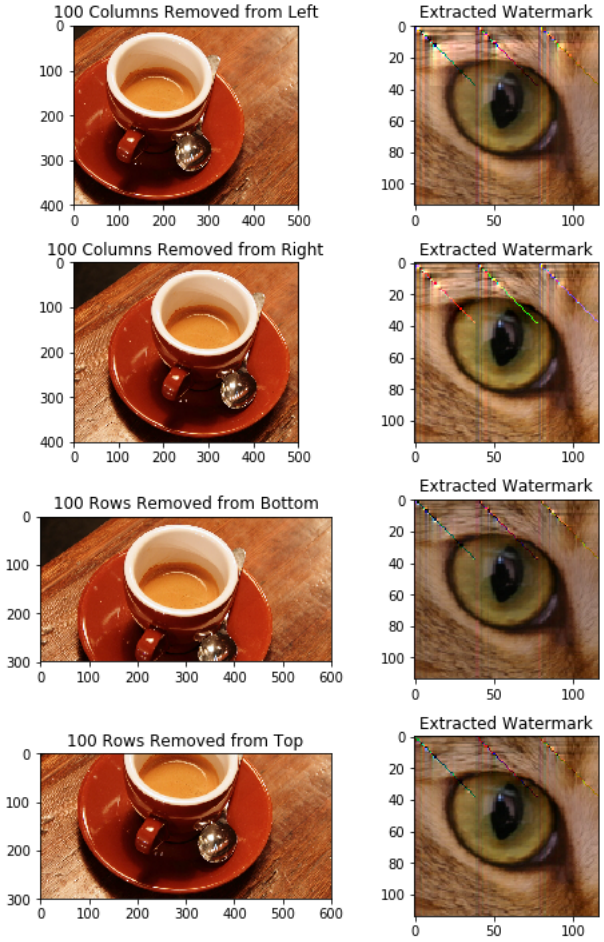
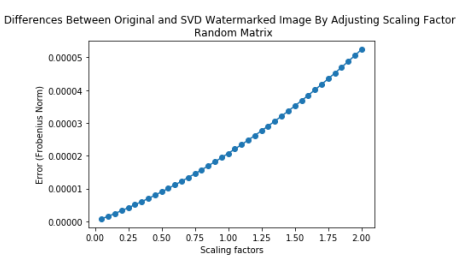
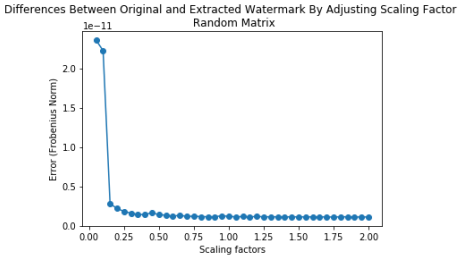


Figure 39: Examples of Cropping and Watermark Extraction, Liu & Tan Scheme

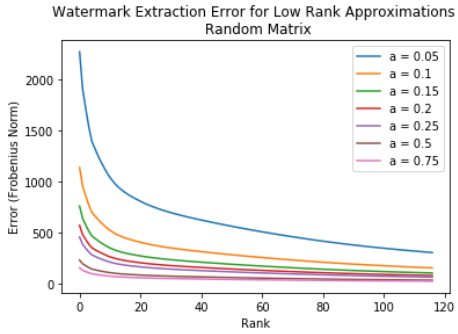
It is also suggested in [13] that a randomly generated matrix would be a stronger watermark than a meaningful one. To test this, we compute the above plots for a randomly generated watermark matrix with dimensions matching the shape of the singular value matrix of the image we are watermarking. These new plots are displayed in Figure 40. In the case of the coffee image, the singular value matrix size is 600×600.



(a) Perceptibility



(b) Error



(c) Low Rank Extraction Error

Figure 40: Random Matrix Watermark Analysis, Liu & Tan Scheme

For the following examples, we use a random 600×600 matrix of random values from a uniform distribution over the interval [0,1). This matrix is displayed as a grayscale image in Figure 41.

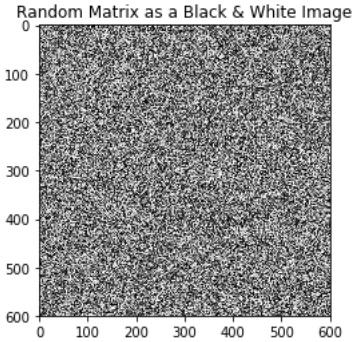
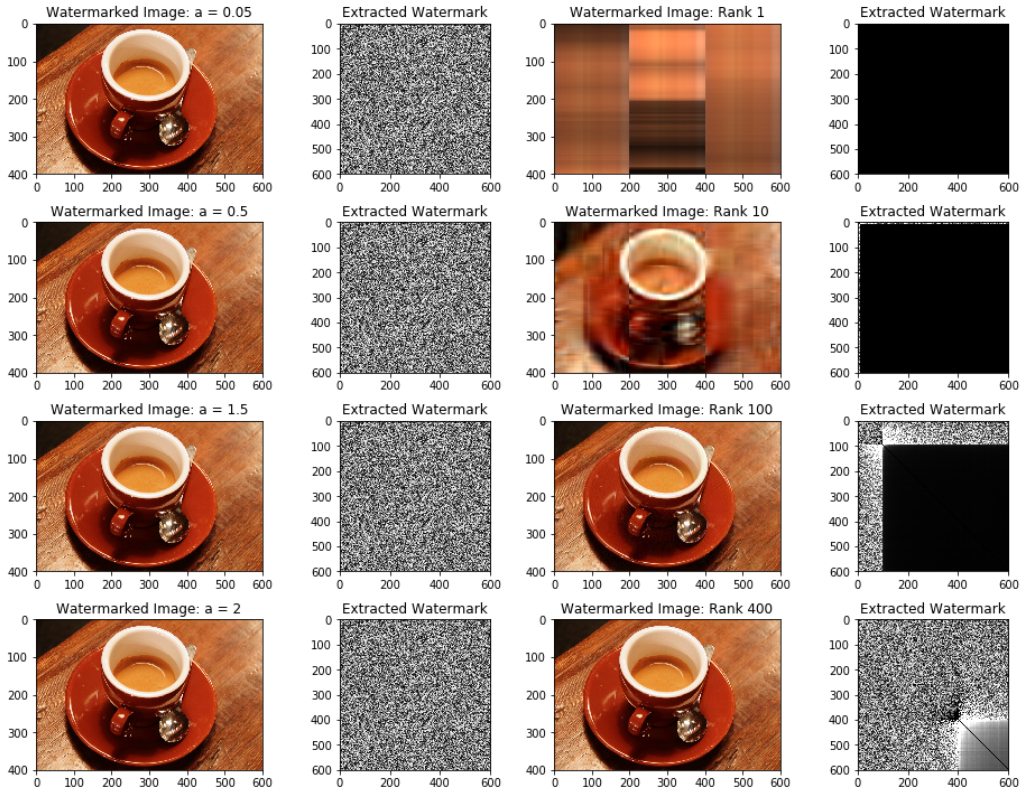


Figure 41: Random Matrix Watermark

Figure 42 shows some visual examples of how the scaling factor affects

the image using the above random watermark. Likewise, we also illustrate the robustness of a random watermark against a simple distortion such as a low rank approximation.



(a) Adjusted Scaling Factor (b) Low Rank Approximation

Figure 42: Example Images for Random Matrix Watermark, Liu & Tan Scheme

From these plots and examples, we can make some interesting conclusions about the Liu-Tan algorithm. First, we see that even with relatively large scaling factors, a randomly generated matrix that is even larger than an image matrix in terms of dimensions (the cat's eye watermark is 342×177 when stacked) has much lower of an effect on the image when inserted as a watermark. However, a random watermark is much more vulnerable to simple distortions like a low-rank approximation than a "meaningful" watermark like an image.

4.4.2 Jain et al. Watermarking Scheme

The watermarking scheme proposed by Jain et al. in [10] improves on the security of the Liu & Tan watermarking scheme while trading off robustness to distortions—especially geometric distortions such as rotation—and imperceptibility. Figure 43 shows the images resulting from embedding the fox image into the raccoon image (see Figure 29) using the Jain et al. watermarking scheme.

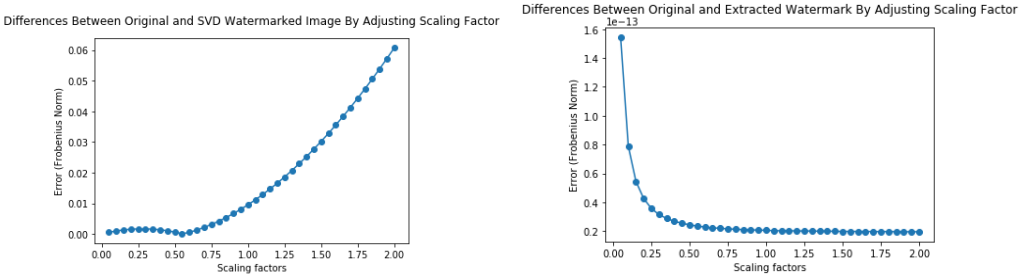
(a) $\alpha = 1$ (b) $\alpha = 0.5$ (c) $\alpha = 0.25$ (d) $\alpha = 0.1$

(e) $\alpha = 0.01$

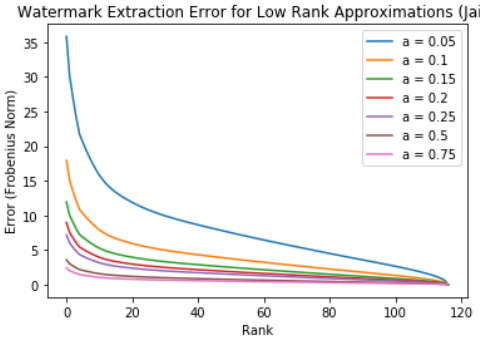
Figure 43: Images watermarked using Jain et al. scheme

As opposed to the Liu & Tan watermarking scheme, the differences between the watermarked image and the original image are quite apparent, even for $\alpha = 0.1$. Thus, for practical usage, we opt to use very low values such as $\alpha = 0.01$. The Jain watermarking scheme is somewhat robust to geometry-preserving distortions, but this robustness decreases with the value of α, so this proves to be a major disadvantage.

The above phenomenon is described in the plots displayed in Figure 44, computed using the same Imageio cat and coffee pictures as earlier. We also include a plot to test the robustness of the Jain et al. watermarking scheme against low-rank approximations as a distortion.



(a) Perceptibility (b) Extraction Error



(c) Low Rank Extraction Error

Figure 44: Image Watermark Analysis, Jain et al. Scheme

Some visual examples for the above plots are provided in Figure 45.

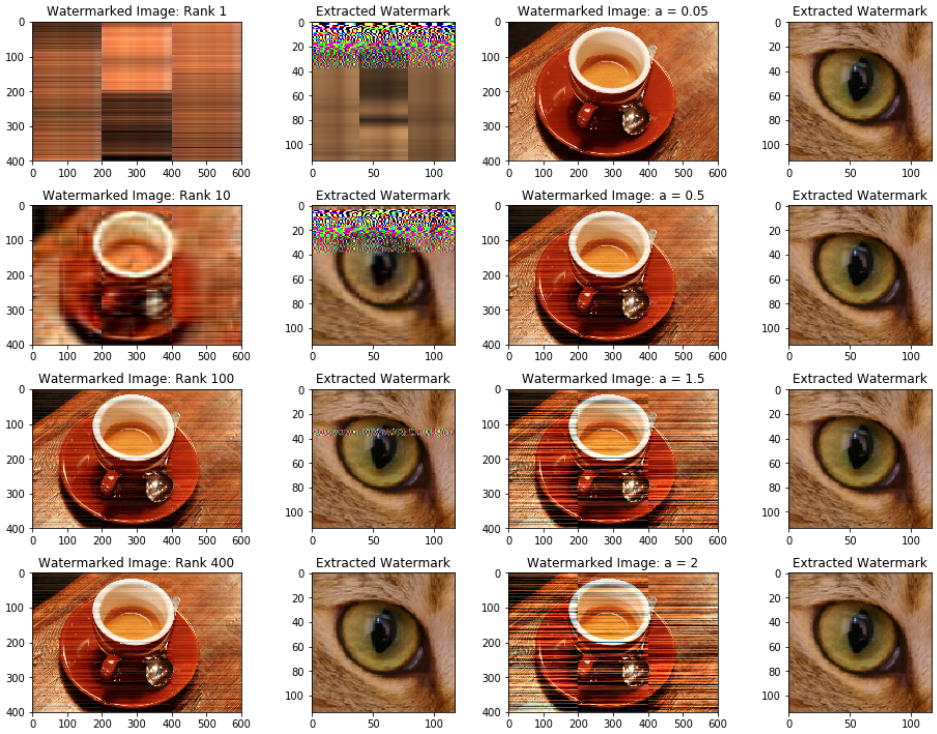
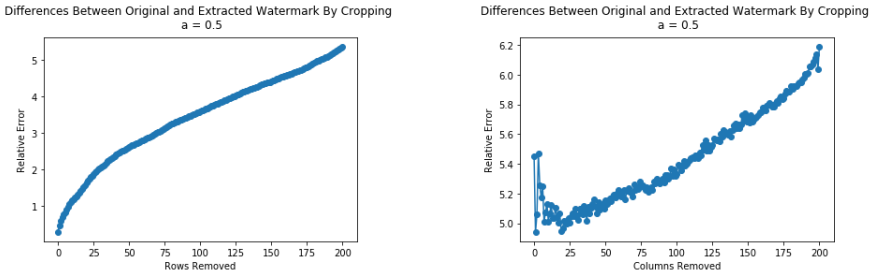


Figure 45: Example Images for Image Watermark, Jain et al. Scheme

We also test the robustness of the Jain et al. watermarking scheme against incrementally cropping by computing the same plots displayed in 4.4.1. These are shown in Figure 46



(a) Cropping Rows (b) Cropping Columns

Figure 46: Cropping Plots for Image Watermark, Jain et al. Scheme

These results are visualized through example images in Figure 47.

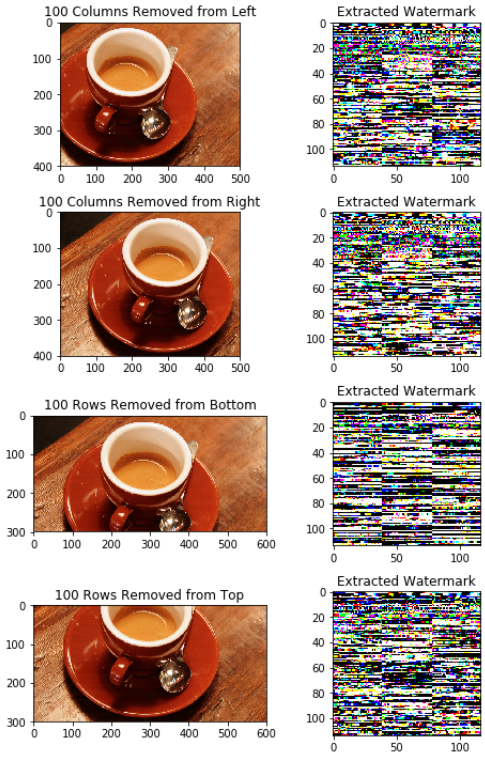
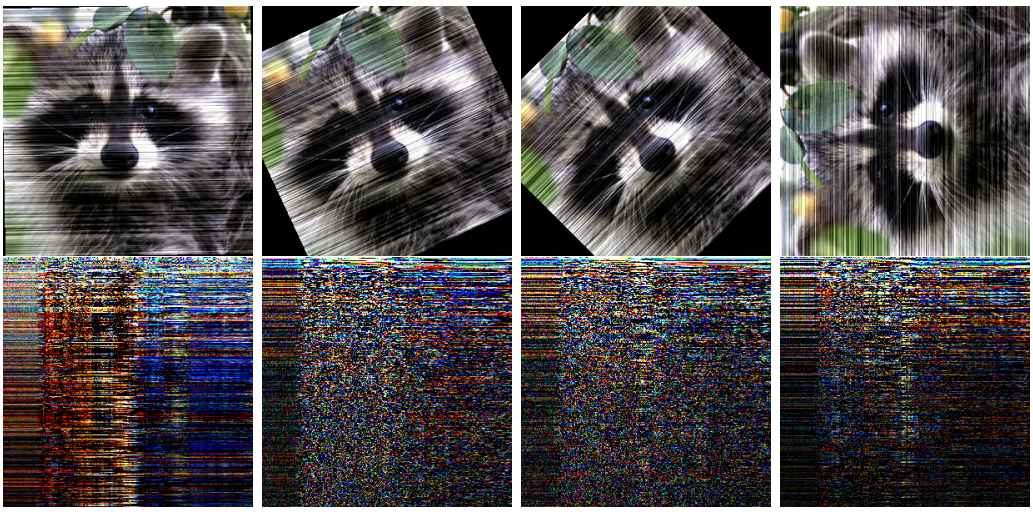


Figure 47: Examples of Cropping and Watermark Extraction, Jain et al. Scheme

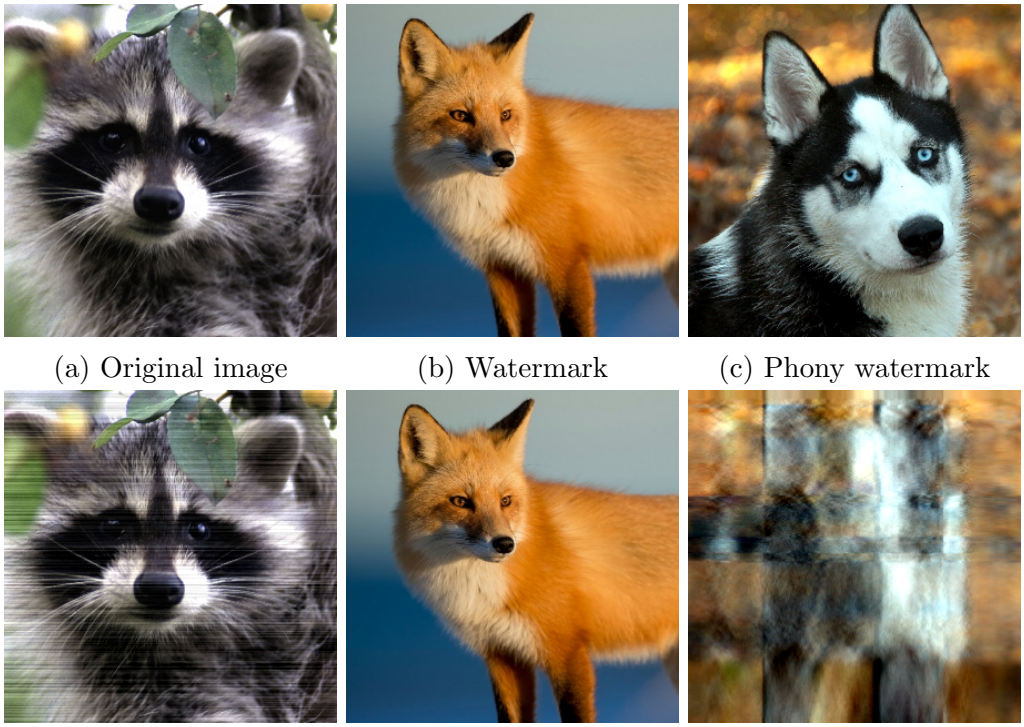
The Jain et al. watermarking scheme does not respond well to geometric distortions at all. Running a similar experiment to that shown in Figure 31, we see that the extracted watermark from the rotated watermarked image is not recognizable, even with a rotation of just one degree. Figure 48 shows us the results of the experiment.



(a) 1° (b) 25° (c) 45° (d) 90°

Figure 48: Rotated watermarked images and their extracted watermarks for $\alpha = 0.25$

Although the Jain et al. watermarking scheme is less robust to distortions, its main advantage over the Liu & Tan watermarking scheme is its improved security. Here, as in Figure 32, we perform the basic security test for the Jain et al. watermarking scheme. As we can see in Figure 49, extracting the incorrect watermark from the image produces an image that barely resembles the watermark.

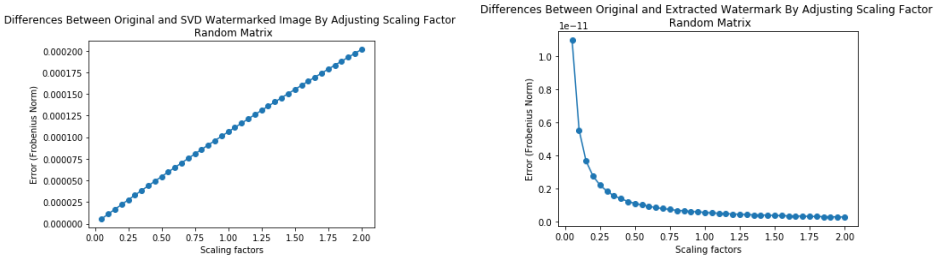


(a) Original image (b) Watermark (c) Phony watermark

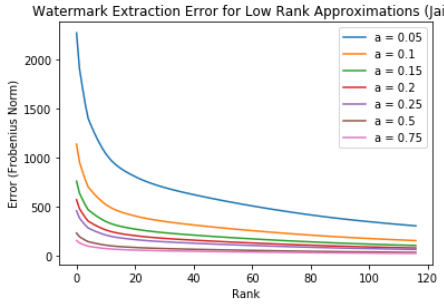
(d) Watermarked image (e) Watermark Extracted (f) Phony watermark extracted

Figure 49: Basic security test for Jain et al. watermarking scheme

Knowing the above information about the Jain et al. scheme, we are interested in the effectiveness of a random matrix watermark. We compute the same plots as in 4.4.1 to test the robustness of a random matrix watermark using this scheme against the Liu & Tan watermarking scheme.



(a) Perceptibility (b) Extraction Error

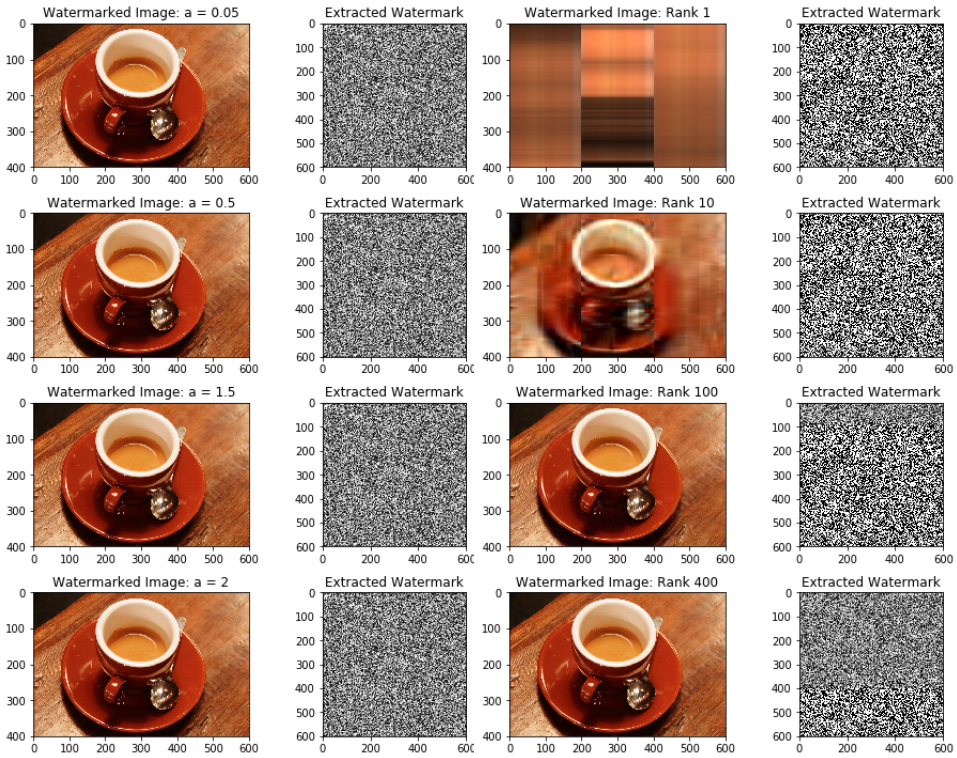


(c) Low Rank Extraction Error

Figure 50: Random Watermark Analysis, Jain et al. Scheme

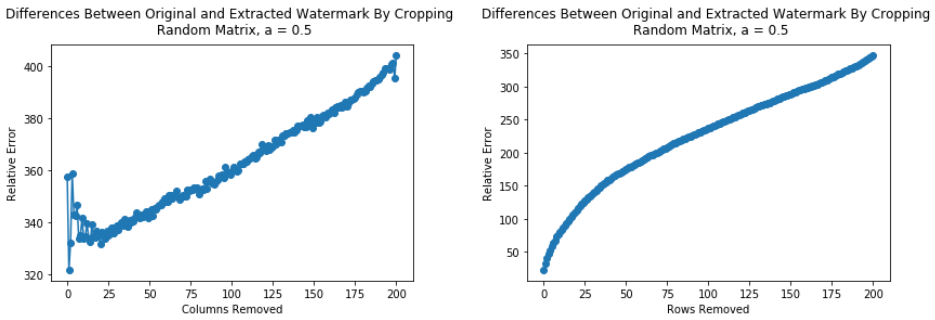
We use the same random matrix watermark as in 4.4.1 for the following images.

In addition, we see that a random matrix watermark is very much not secure against a distortion by cropping using this watermarking scheme in the following plots.



(a) Adjusted Scaling Factor (b) Low Rank Approximation

Figure 51: Example Images for Random Matrix Watermark, Jain et al. Scheme



(a) Cropping Columns (b) Cropping Rows

Figure 52: Cropping Plots for Random Matrix Watermark, Jain et al. Scheme

4.4.3 Modified Jain et al. Watermarking Scheme

In this section, we examine the various properties of the modification to the Jain et al. watermarking scheme proposed in Section 3.4.3. As in the previous sections. Figure 53 shows the results of embedding the fox watermark in the raccoon image (Figure 29) using various scaling factors α.

(a) $\alpha = 1$ (b) $\alpha = 0.5$ (c) $\alpha = 0.25$ (d) $\alpha = 0.1$

Figure 53: Images watermarked using the modified Jain et al. watermarking scheme

We visualize these terms with an example using images in Figure (54)

We examine the behavior of the Modified Jain et al. watermarking scheme as the scaling factor is modified in the following plots. In addition, as in previous schemes, we can test the robustness of the Modified Jain et al. watermarking scheme against various distortions such as being subjected to a low-rank SVD approximation.

(a) Original Image

(b) Watermarked Image (Jain)

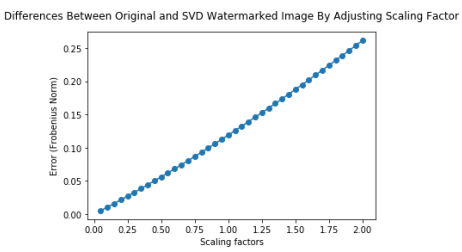
(c) Watermarked Image (Jain Modified)

(d) Watermark

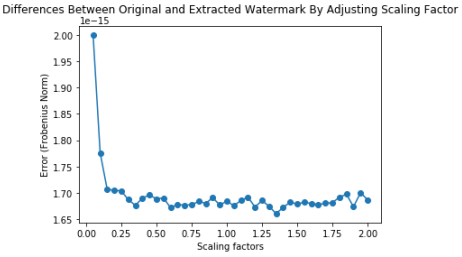
(e) Additive Term (Jain)

(f) Additive Term (Jain Modified)

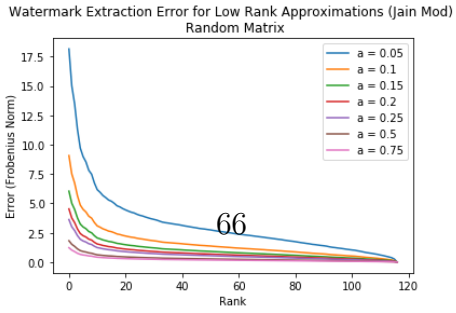
Figure 54: Examples for modified and unmodified Jain et al. watermarking schemes



(a) Perceptibility



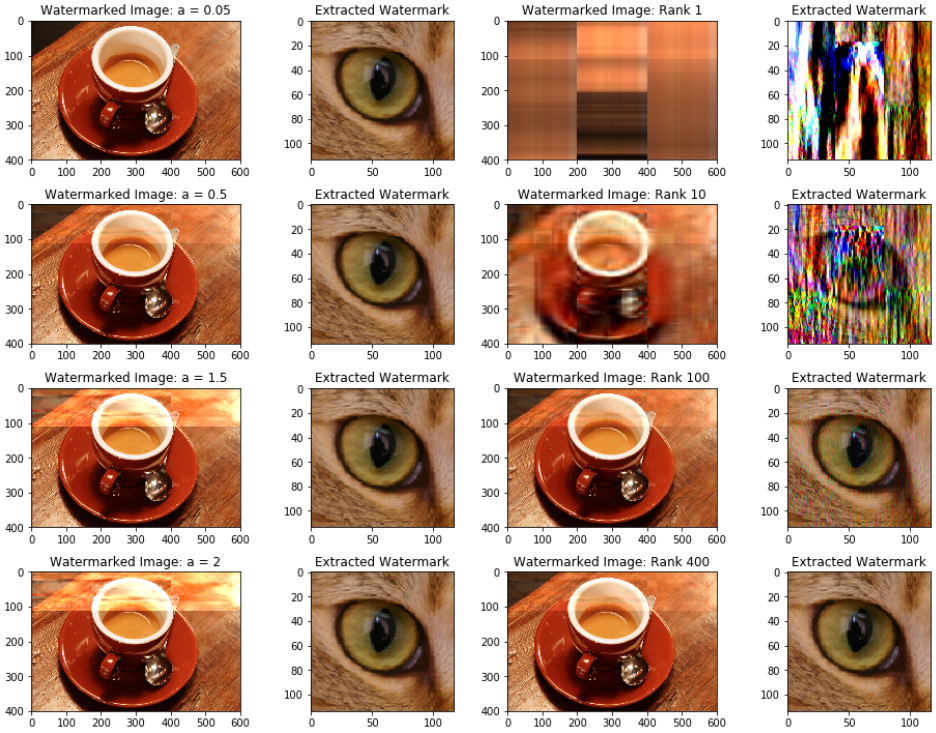
(b) Extraction Error



(c) Low Rank Extraction Error

Figure 55: Image Watermark Analysis, Modified Jain et al. Scheme

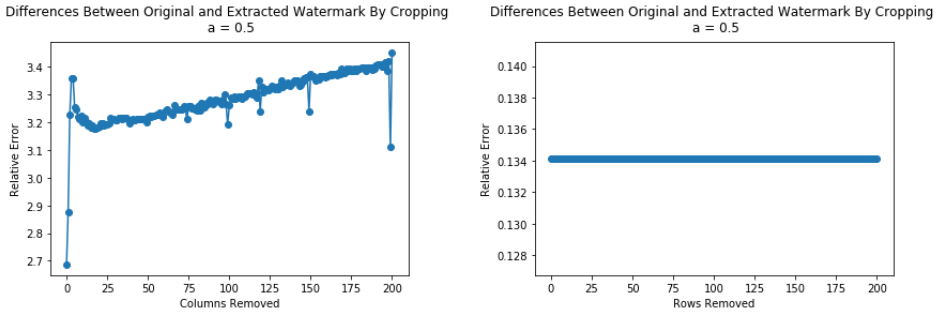
Examples of the previous results are displayed in Figure 56.



(a) Adjusted Scaling Factor (b) Low Rank Approximation

Figure 56: Example Images for Image Watermark, Modified Jain et al. Scheme

Likewise, we also analyze the modified scheme's robustness against incrementally cropping the watermarked image in Figure 57.



(a) Cropping Columns (b) Cropping Rows

Figure 57: Cropping Plots for Image Watermark, Modified Jain et al. Scheme

These results are visualized in Figure 58.

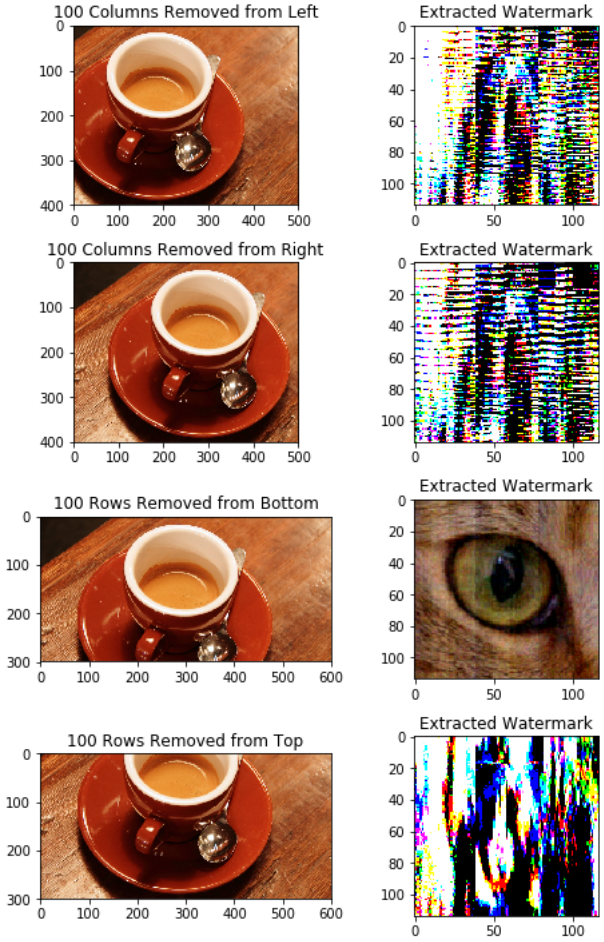


Figure 58: Examples of Cropping and Watermark Extraction, Modified Jain et al. Scheme

The behavior in Figure 57(b) is particularly interesting to us. We have not yet determined why our error seems to be constant, even when removing. When we adjust our code so that we are incrementally cropping more rows, we get the results displayed in Figure 59.

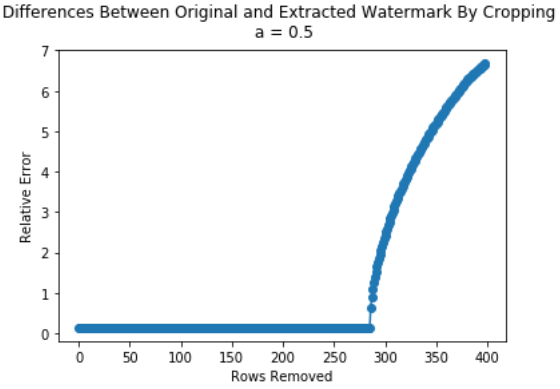
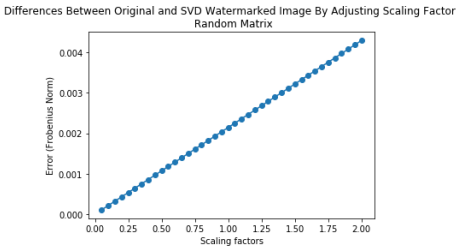


Figure 59: Further Analysis of Cropping Rows for Image Watermark, Modified Jain et al. Scheme

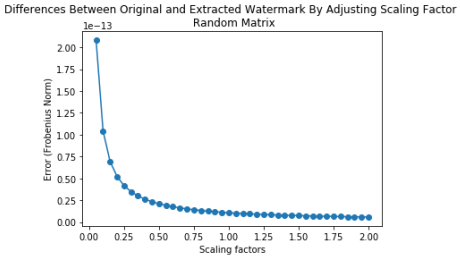
We see that the error remains constant until briefly before 300 rows are removed, at which point the error begins increasing with an interesting pattern.

In the future, we plan to further investigate our code to ensure that there are no errors in order to determine whether these plots truly do indicate an unusual robustness against cropping for the Modified Jain et al. cheme. We also remember that our watermark is much smaller than our embedding image, requiring us to pad our watermark with zeros. Perhaps experimenting with different sizes of watermarks and increasing the size will results that are more representative of the Modified Jain et al. scheme's true robustness against cropping.

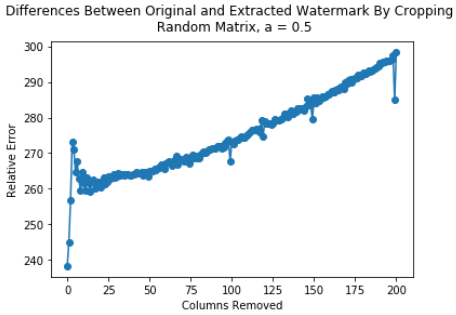
We also test the robustness of the Modified Jain et al. watermarking scheme if a random matrix watermark is employed in Figure 60.



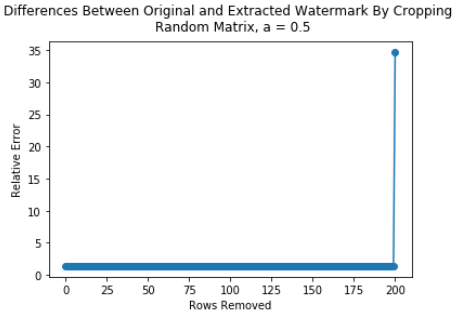
(a) Perceptibility



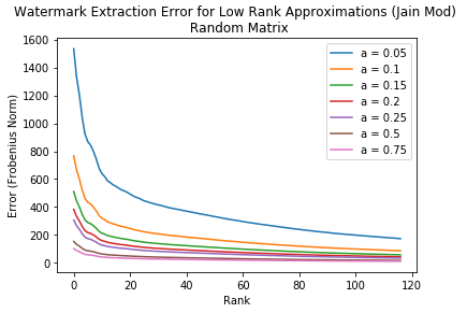
(b) Extraction Error



(c) Cropping Columns



(d) Cropping Rows



(e) Low Rank Extraction Error

Figure 60: Random Watermark Analysis, Modified Jain et al. Scheme

Again, we notice some strange behavior in Figure 60. In addition, the behavior in the column-cropping plots for the Jain et al. and Modified Jain et al. schemes appears to be much more erratic when compared to the same plots for the Liu & Tan watermarking scheme. We will continue to investigate both of these issues in the future.

Note that for both the image and random watermark, cropping rows and attempting to extract a watermark according to the method described in Section 4.4.1 yields strange results. While we have not yet determined

whether these are erroneous or reflective of an unusual robustness against cropping for the modified Jain et al. scheme, we plan to further explore alternate means of watermark extraction from a watermarked image distorted by cropping in order to remedy these potential problems in our error plots for row cropping. Some of our ideas include padding each color channel of the cropped image before stacking it and removing rows/columns in the SVD outer product expansion accordingly if the number of rows/columns removed during cropping is known.

We also display some example images for the earlier plots in Figure 61. Again, the random matrix watermark being used is the same 600×600 matrix used in Sections 4.4.2 and 4.4.1.

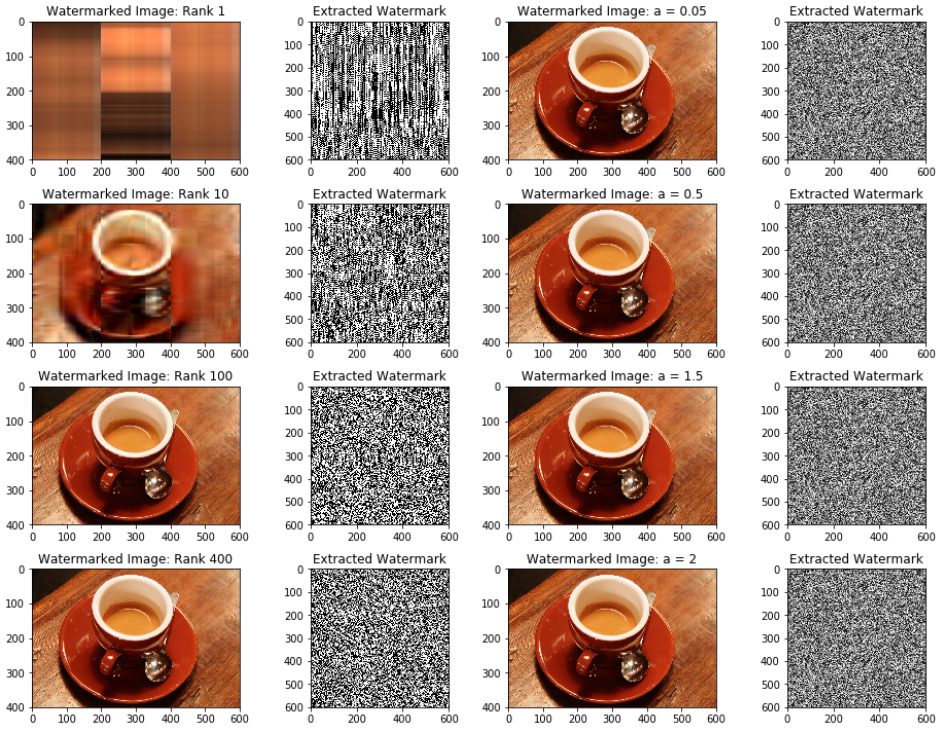
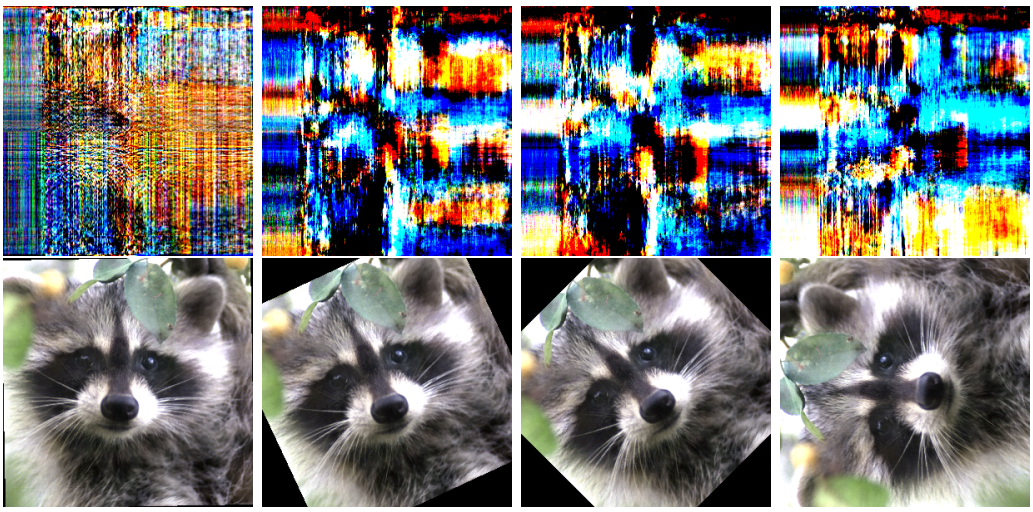


Figure 61: Example Images for Random Watermark, Modified Jain et al. Scheme

We see that, as in previous schemes, a random matrix watermark allows for good imperceptibility and accurate extraction under ideal circumstances, but is not robust against simple distortions and attacks.

Finally, as in section 4.4.2, we test how the modified Jain watermarking scheme responds to rotations. Figure 62 shows the results of this experiment.

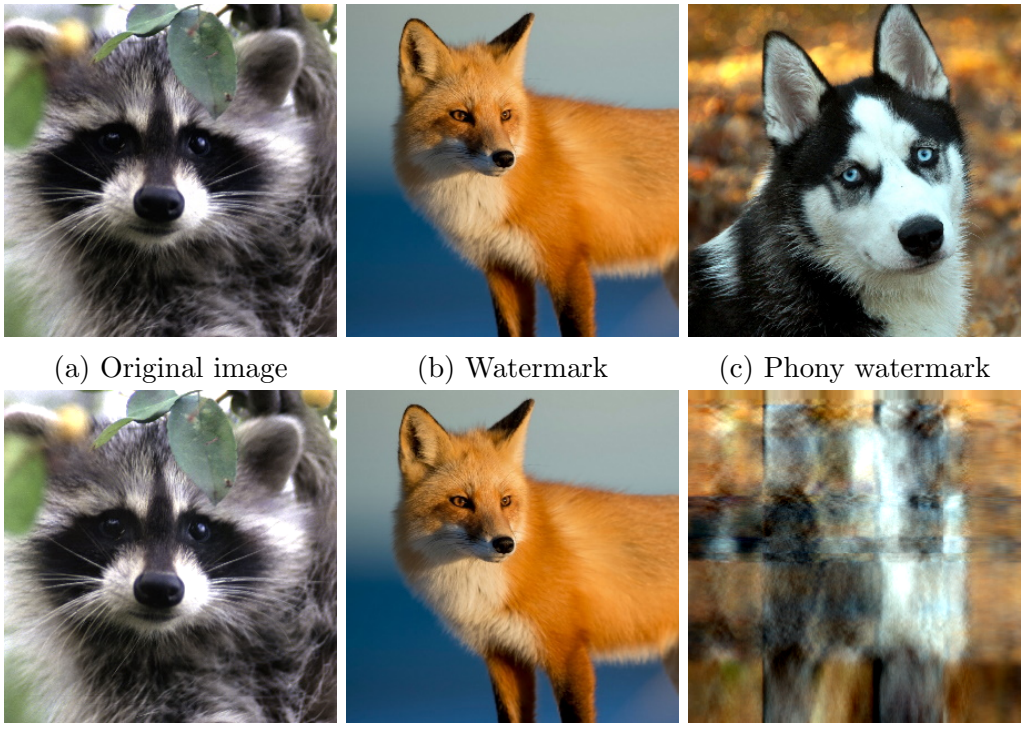


(a) 1° (b) 25° (c) 45° (d) 90°

Figure 62: Rotated watermarked images and their extracted watermarks for $\alpha = 0.25$

The extracted fox watermark is somewhat recognizable after a rotation of the watermarked image by one degree, but any larger rotation, similarly to the Jain et al. scheme, results in significant distortions to the extracted watermarked image. The modified Jain et al. watermarking scheme is not robust to distortions.

Finally, we perform the basic security test on the modified Jain et al. watermarking scheme, as in 4.4.2. The results are shown in Figure 63



(a) Original image (b) Watermark (c) Phony watermark

(d) Watermarked image (e) Watermark (f) Phony watermark extracted

Figure 63: Basic security test for modified Jain et al. watermarking scheme

As we can see, the extracted watermark bears only slight resemblance to the phony watermark. The results here are very similar to that of Jain et al., showing that the modified Jain et al. watermarking scheme has similar properties with respect to the protection of rightful ownership.

4.4.4 Comparing Jain et al. and Modified Jain et al. Watermarking Schemes

The above experimental results seem to indicate that the modified Jain et al. watermarking scheme preserves the desirable robustness properties of the Jain et al. watermarking scheme while improving on the imperceptibility of the embedded watermark. In this section, we support this inclination with some theoretical results and some additional experimental results.

Let $A = USV^\top$ be a matrix and $W = U_W S_W V_W^\top$ a watermark matrix.

Recall that the Jain et al. watermarking scheme has an embedding function

$$E_J : (A, W, \alpha) \;\mapsto\; \begin{array}{c} A_J = A + \alpha U U_W S_W V^\top \\ K = (A, V_W, \alpha), \end{array}$$

and extraction function

$$E_J^{-1} : (A_J^*, K) \;\mapsto\; W_J^* = \alpha^{-1} U^\top (A_J^* - A) V V_W^\top.$$

Also recall that the Jain et al. watermarking scheme has an embedding function

$$E_M : (A, W, \alpha) \;\mapsto\; \begin{array}{c} A_M = A + \alpha U_W S_W V^\top \\ K = (A, V_W, \alpha), \end{array}$$

and extraction function

$$E_M^{-1} : (A_M^*, K) \;\mapsto\; W_M^* = \alpha^{-1} (A_M^* - A) V V_W^\top.$$

Using the above, we now show that the Frobenius norm error incurred in the extracted watermark in response to an additive perturbation P on the watermarked matrix is the same for the modified and unmodified Jain et al. scheme, and this error can be expressed in terms of $\|P\|_F$ and α.

Proposition 1. *Let A, W, and P be matrices, and let $\alpha \in \mathbb{R}$. With notation as above, Let*

$$\begin{aligned} (A_J, K_J) &= E_J(A, W, \alpha), \\ (A_M, K_M) &= E_M(A, W, \alpha), \\ W_J^* &= E^{-1}(A_J + P, K_J), \text{ and} \\ W_M^* &= E_M^{-1}(A_M + P, K_M). \end{aligned}$$

Then we have

$$\|W - W_J^*\|_F = \|W - W_M^*\|_F = \frac{\|P\|_F}{|\alpha|}.$$

Proof. Apply the SVD to obtain $A = USV^\top$ and $W = U_W S_W V_W^\top$. Under the Jain et al. watermarking scheme, if we attempt to extract the distorted

watermark W_J^* from the perturbed matrix $A_J + P$, we find that

$$\begin{aligned} W_J^* &= \alpha^{-1}(U^\top(A_J^* - A)VV_W^\top \\ &= \alpha^{-1}U^\top(A_J + P - A)VV_W^\top \\ &= \alpha^{-1}U^\top(A + \alpha UU_W S_W V^\top + P - A)VV_W^\top \\ &= \alpha^{-1}U^\top(\alpha UU_W S_W V^\top + P)VV_W^\top \\ &= U_W S_W V_W^\top + \alpha^{-1}U^\top PVV_W^\top \\ &= W + \alpha^{-1}U^\top PVV_W^\top \end{aligned}$$

Computing the error with respect to the Frobenius norm, we find that

$$\begin{aligned} \|W_J^* - W\|_F &= \left\|\alpha^{-1}U^\top PVV_W^\top\right\|_F \\ &= \frac{\|P\|_F}{|\alpha|}, \end{aligned}$$

since the matrices U, V, and V_W are unitary.

Similarly, for the modified Jain et al. watermarking scheme, we find that

$$\begin{aligned} W_J^* &= \alpha^{-1}(A_M^* - A)VV_W^\top \\ &= \alpha^{-1}(A_M + P - A)VV_W^\top \\ &= \alpha^{-1}(A + \alpha U_W S_W V^\top + P - A)VV_W^\top \\ &= \alpha^{-1}(\alpha U_W S_W V^\top + P)VV_W^\top \\ &= U_W S_W V_W^\top + \alpha^{-1}PVV_W^\top \\ &= W + \alpha^{-1}PVV_W^\top, \end{aligned}$$

and

$$\begin{aligned} \|W_M^* - W\|_F &= \left\|\alpha^{-1}PVV_W^\top\right\|_F \\ &= \frac{\|P\|_F}{|\alpha|}, \end{aligned}$$

since V, V_W are unitary matrices. □

Note that an additive perturbation to the watermarked matrix results in an additive perturbation to the extracted watermark. The proof of Proposition 1 also shows that the error in the extracted watermark increases with $\|P\|_F$ and decreases as α increases, as expected.

Next, informed by the experimental examples in Section 4.4.3, we examine how the perceptibly of the watermark differs for the modified and unmodified Jain et al. watermarking schemes.

Examining errors in terms of the Frobenius norm, we find that

$$\begin{aligned}\|A_J - A\|_F &= \left\|\alpha U U_W S_W V^\top\right\|_F \\ &= \alpha \|W\|_F ,\end{aligned}$$

and

$$\begin{aligned}\|A_M - A\|_F &= \left\|\alpha U_W S_W V^\top\right\|_F \\ &= \alpha \|W\|_F ,\end{aligned}$$

since the matrices U, U_W, and V are unitary. Thus, the error between the watermarked and original matrix for the modified and unmodified Jain et al. watermarking schemes are the same. However, the removal of the multiplier U in the additive term $\alpha U U_W S_W V^\top$ seems to have desirable effects on the imperceptibility of the watermark in the image. Thus, we use other methods to examine the relationships between A_J and A, and A_M and A.

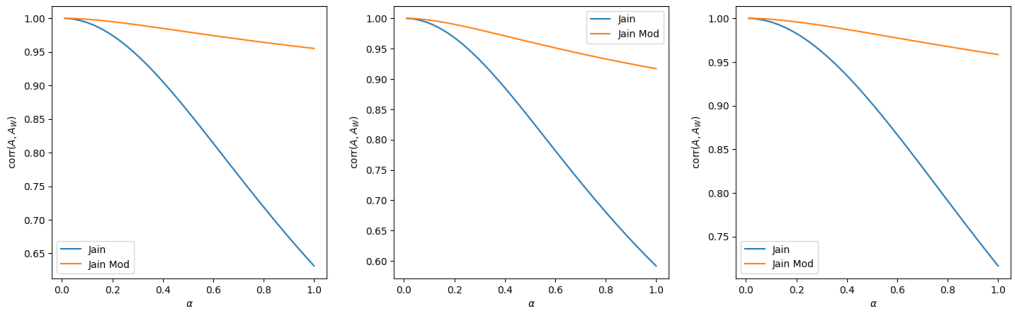
We define the correlation between two matrices X and Y to be

$$\operatorname{corr}(X, Y) = \frac{\langle X, Y\rangle_F}{\|X\|_F \|Y\|_F},$$

where $\langle X, Y\rangle_F$ is the Frobenius inner product $\langle X, Y\rangle_F = \operatorname{Tr}(X^\top Y)$. Note that $-1 \leq \langle X, Y\rangle_F \leq 1$, and if X is a scalar multiple of Y, then $\operatorname{corr}(X, Y) = 1$. This is the cosine of the angle between X and Y, as described in [7]. In Figure 64, we display the results of some experiments in which we embedded watermarks into images with various scaling factors α and examined the correlation between the resulting watermarked image and the original image.

Figure 64: Correlations between image (A) and watermarked image (A_W) after watermark embedding with various scaling factors α

(a) Raccoon watermarked by fox (b) Raccoon watermarked by noise (c) Fox watermarked by husky



The correlation between the watermarked image and the original image remains above 0.95 for the modified Jain et al. scheme, while it drops quickly for the unmodified version. This means that with the modified Jain et al. watermarking scheme, the watermarked image is approximately a scalar multiple of the original image. This, combined with the results on the Frobenius norm error between the watermarked image and the original image, indicate that the modified Jain et al. watermarking scheme has better imperceptibility than the unmodified for any given scaling factor α, which is consistent with observation.

Although the error in the extracted watermark after additive perturbation is not different between the modified and unmodified Jain et al. schemes, the property of the modified Jain et al. scheme described in the previous paragraph allows us to use larger scaling factors α in practice without a negative impact on perceptibility, resulting in less error in the extracted watermark.

4.4.5 Audio

The watermarking technique can be applied onto audio files similarly as we perform Short Time Fourier Transformation on audio data. In this section we embed and extract audio watermark from and into audio data and perform experiments with different distortions.

We perform audio watermarking on a 17-second recording clip of Bach Cello Suite No.1[6], and the embedded watermark is a 7-second clip of news

introduction music[2]. The news clip is embedded with a scaling factor α.

The watermarking scheme we are using here is the Liu & Tan scheme, which embeds the watermark along the diagonal entries of Σ, the singular values. As stated in Section 4.2.2, singular value modification for audio leads to changes in amplitude and noise levels. This is demonstrated through Figure 65. As α increases, the increase in noise level is depicted in the spectrogram. The optimal value α for image is around 0.5, but for audio we choose 0.4 as our optimal scalar, for noise are specifically perceptible in the beginning of the audio when the original clip is silent.

(a) Original (b) $\alpha = 0.1$ (c) $\alpha = 0.4$ (d) $\alpha = 1.6$

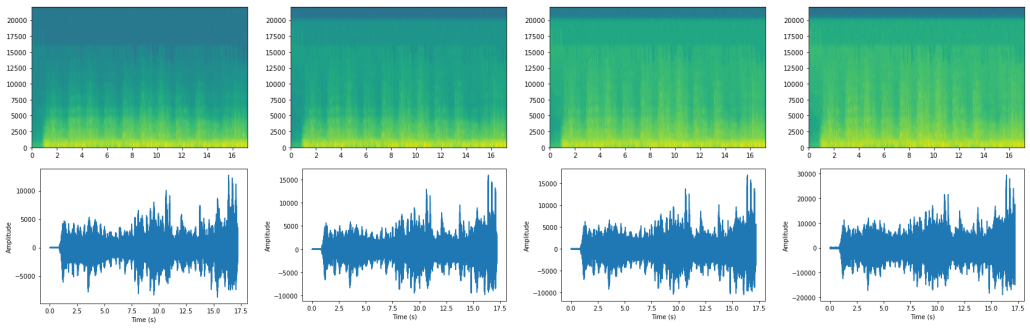


Figure 65: Audio watermarked using the Liu & Tan Watermarking Scheme

To check the robustness of the Liu & Tan Algorithm on audio watermarking, we perform four separate distortions on the watermarked audio with $\alpha = 0.4$ using external audio software. The four modifications are to add reverb, to lower all pitches, to limit high frequencies, and to remove the noise. The result shows that the most dominant feature, melody, is still preserved in the extracted watermark, confirming the robustness of the algorithm.

(a) Lower Pitch (b) Frequency (c) Remove Noise (d) Add Reverb

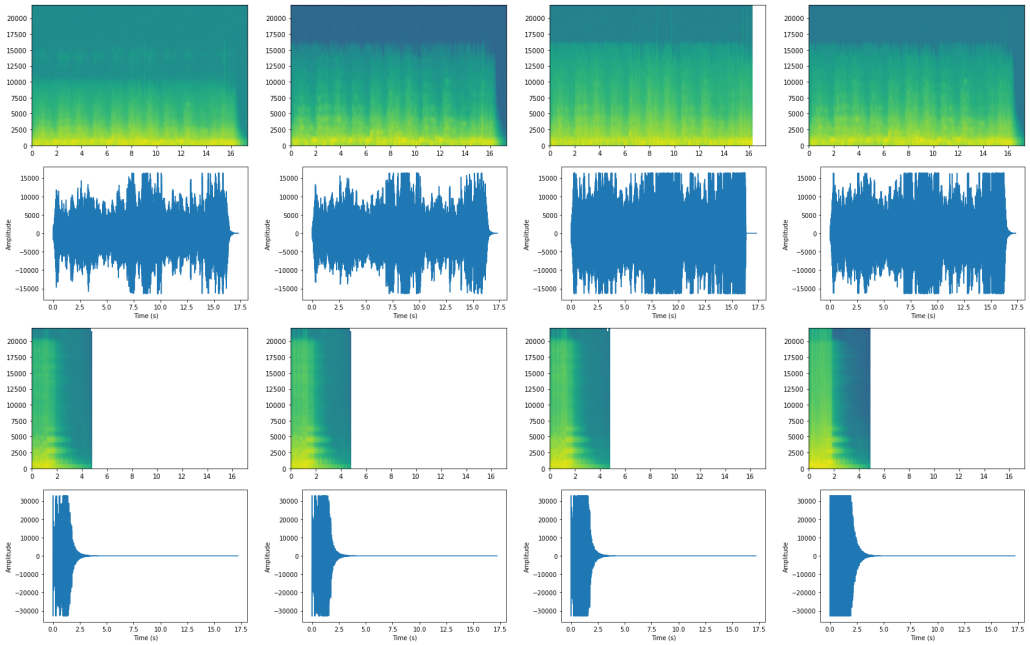


Figure 66: Distortions on Watermarked Audio using the Liu & Tan Watermarking Scheme and the Extracted Watermarks

5 Concluding Remarks

In this report, we demonstrated three applications of the SVD in media compression, data analysis, and media watermarking.

The first application is in media compression for image, video, and audio. Low-rank approximation using SVD effectively preserves the dominant features of the media while reducing the degrees of freedom. As the most dominant features in videos are the surveillance backgrounds, the application can be potentially used in video background removal in further research. Modification of singular values also result in changes in various characteristics of the media.

We performed three different algorithms for the computation of SVD. The deterministic SVD is the most accurate, and the randomized SVD and the compressed SVD are more efficient in different applications.

The second application provides a thorough understanding of the given

data sets. By projecting the data matrix onto different singular vectors and producing approximations, we can effectively visualize the different characteristics and correlations among the data set.

The third application protects the ownership of medias, specifically images and audios. We investigate in the two algorithms adopted from Liu & Tan [13] and Jain [10] in embedding and extracting a watermark matrix into the media to be protected. The watermark matrix can be in the format of an image, a piece of audio, or a random matrix. With the Liu & Tan scheme is insecure and the Jain et al. scheme produces highly perceptible embedding, we produced a modified algorithm that increases imperceptibility of the embedded watermark and remains secure.

We intend to test the different watermarking schemes on audio watermarking in further research.

References

[1] *chelsea.png*. Imageio, 2014-2020.

[2] *news opening 5*. David Fesliyan, Aug 2016.

[3] Coronavirus locations: Covid-19 map by county and state, Jul 2020.

[4] *Coronavirus Resource Center*. Johns Hopkins University of Medicine, 2020.

[5] Most aggressive states against the coronavirus, Apr 2020.

[6] Johann Sebastian Bach. *BACH, J.S.: Cello Suite No. 1, BWV 1007*. Naxos Digital Services US Inc.

[7] Michael W. Berry, Zlatko Drmac, and Elizabeth R. Jessup. Matrices, vector spaces, and information retrieval. *SIAM Review*, 41:335–362, 1999.

[8] Steven Brunton and J. Kutz. *Singular Value Decomposition (SVD)*, pages 3–46. 02 2019.

[9] N. Erichson, S. L. Brunton, and J. Kutz. Compressed singular value decomposition for image and video processing. In *2017 IEEE International Conference on Computer Vision Workshop (ICCVW)*, pages 1880–1888, Los Alamitos, CA, USA, oct 2017. IEEE Computer Society.

[10] Chirag Jain, Siddharth Arora, and Prasanta K. Panigrahi. A reliable SVD based watermarking schem. *CoRR*, abs/0808.0309, 2008.

[11] Dan Kalman. A singularly valuable decomposition: The svd of a matrix. *The College Mathematics Journal*, 27(1):2–23, 1996.

[12] Nasser Kehtarnavaz. Chapter 7 - frequency domain processing. In Nasser Kehtarnavaz, editor, *Digital Signal Processing System Design (Second Edition)*, pages 175 – 196. Academic Press, Burlington, second edition edition, 2008.

[13] Ruizhen Liu and Tieniu Tan. An svd-based watermarking scheme for protecting rightful ownership. *IEEE Transactions on Multimedia*, 4:121–128, 08 2002.

[14] Rachel Michetti. *coffee.png*. Imageio, 2014-2020.

[15] L. Zhang and A. Li. Robust watermarking scheme based on singular value of decomposition in dwt domain. In *2009 Asia-Pacific Conference on Information Processing*, volume 2, pages 19–22, 2009.

[16] Lingsong Zhang, J. S. Marron, Haipeng Shen, and Zhengyuan Zhu. Singular value decomposition and its visualization. *Journal of Computational and Graphical Statistics*, 16:1–22, 2007.