Image Compression Using Singular Value Decomposition

Ian Cooper and Craig Lorenc

December 15, 2006

Abstract

Singular value decomposition (SVD) is an effective tool for minimizing data storage and data transfer in the digital community. This paper explores image compression through the use of SVD on image matrices.

Introduction

Data compression is an important application of linear algebra. The need to minimize the amount of digital information stored and transmitted is an ever growing concern in the modern world. Singular Value Decomposition is an effective tool for minimizing data storage and data transfer.

Introduction
SVD Overview
SVD Example
Images and SVD . . .
SVD vs. Memory
MATLAB

Home Page
Title Page
Go Back
Full Screen
Close
Quit

SVD Overview

Let A be any $m \times n$ matrix. SVD is the factorization of A into $U\Sigma V^T$, where U and V are orthonormal matrices. Σ is a diagonal matrix comprised of the singular values of A. The singular values $\sigma_1 \geqslant \cdots \geqslant \sigma_n \geqslant 0$ appear in descending order along the main diagonal of Σ. The numbers $\sigma_1^2 \geq \cdots \geqslant \sigma_n^2$ are the eigenvalues of AA^T and A^TA.

$$A = U\Sigma V^T$$

$$U = \begin{bmatrix} \mathbf{u_1} & \cdots & \mathbf{u_n} \end{bmatrix}, \quad \Sigma = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \sigma_n \end{bmatrix}, \quad V^T = \begin{bmatrix} \mathbf{v_1}^T \\ \vdots \\ \mathbf{v_n}^T \end{bmatrix}$$

SVD Example

A 2×2 matrix of rank 2 is a natural choice for an example of the SVD approach as most images will undoubtedly have full rank. It also provides the proper stage for an efficient illustration of the process at hand. Consider matrix A,

$$A = \begin{bmatrix} 2 & 2 \\ -1 & 1 \end{bmatrix}$$

it follows that,

$$A^T = \begin{bmatrix} 2 & -1 \\ 2 & 1 \end{bmatrix}$$

Both of these matrices are needed in order to perform singular value decomposition. We need to mulitply them together in order to produce the matrices necessary for SVD.

Step 1: Calculate AA^T and A^TA

The first step in the SVD algorythm is calculating AA^T and A^TA

$$AA^T = \begin{bmatrix} 2 & 2 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 2 & -1 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 8 & 0 \\ 2 & 0 \end{bmatrix}$$

$$A^TA = \begin{bmatrix} 2 & -1 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 2 & 2 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} 5 & 3 \\ 3 & 5 \end{bmatrix}$$

Step 2: Eigenvalues and Σ

The second step of the process is to find eigenvalues for AA^T and A^TA. Once we have those we can easily compute the singular values by taking the square roots of the eigenvalues, λ_1 and λ_2. With the acquisition of σ_1 and σ_2 we can form Σ,

$$|AA^T - \lambda I| = 0 \qquad (1)$$

$$|A^TA - \lambda I| = 0 \qquad (2)$$

Introduction
SVD Overview
SVD Example
Images and SVD...
SVD vs. Memory
MATLAB

Home Page
Title Page
◀◀ ▶▶
◀ ▶
Go Back
Full Screen
Close
Quit

Solving equation (1) yields λ_1 and λ_2:

$$\begin{aligned}
|AA^T - \lambda I| &= 0 \\
\left|\begin{bmatrix} 8 & 0 \\ 0 & 2 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right| &= 0 \\
\begin{vmatrix} 8-\lambda & 0 \\ 0 & 2-\lambda \end{vmatrix} &= 0 \\
(8-\lambda)(2-\lambda) &= 0 \\
\lambda_1 &= 8 \\
\lambda_2 &= 2
\end{aligned}$$

Now that we have λ_1 and λ_2 we can compute the singular values of AA^T. With σ_1 and σ_2 we can form Σ.

$$\begin{aligned}
\sigma_1 &= \sqrt{\lambda_1} = \sqrt{8} \\
\sigma_2 &= \sqrt{\lambda_2} = \sqrt{2}
\end{aligned}$$

$$\Sigma = \begin{bmatrix} \sqrt{8} & 0 \\ 0 & \sqrt{2} \end{bmatrix}$$

Step 3: Finding U

The columns of U are the unit eigenvectors of AA^T. We will need to solve equation (3) using both eigenvalues to find two eigenvectors that we can use for the columns of U.

$$(AA^T - \lambda I)\mathbf{x} = 0 \tag{3}$$

$$(A^T A - \lambda I)\mathbf{x} = 0. \tag{4}$$

Solve for the first eigenvector of AA^T. This will be used to generate the first column of U.

$$\begin{aligned}
(AA^T - \lambda I)\mathbf{x} &= 0 \\
\left(\begin{bmatrix} 8 & 0 \\ 0 & 2 \end{bmatrix} - \lambda_1 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right)\mathbf{x}_1 &= 0 \\
\begin{bmatrix} 8-\lambda_1 & 0 \\ 0 & 2-\lambda_1 \end{bmatrix}\mathbf{x}_1 &= 0 \\
\begin{bmatrix} 8-8 & 0 \\ 0 & 2-8 \end{bmatrix}\mathbf{x}_1 &= 0 \\
\begin{bmatrix} 0 & 0 \\ 0 & -6 \end{bmatrix}\mathbf{x}_1 &= 0 \\
\mathbf{x}_1 &= \begin{bmatrix} -1 \\ 0 \end{bmatrix}
\end{aligned}$$

Introduction
SVD Overview
SVD Example
Images and SVD...
SVD vs. Memory
MATLAB

Solve for the second eigenvector of AA^T. This will be used to generate the second column of U.

$$(AA^T - \lambda I)\mathbf{x} = 0$$

$$\left(\begin{bmatrix} 8 & 0 \\ 0 & 2 \end{bmatrix} - \lambda_2 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right)\mathbf{x}_2 = 0$$

$$\begin{bmatrix} 8-\lambda_2 & 0 \\ 0 & 2-\lambda_2 \end{bmatrix}\mathbf{x}_2 = 0$$

$$\begin{bmatrix} 8-8 & 0 \\ 0 & 2-8 \end{bmatrix}\mathbf{x}_2 = 0$$

$$\begin{bmatrix} 0 & 0 \\ 0 & -6 \end{bmatrix}\mathbf{x}_2 = 0$$

$$\mathbf{x}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

With the acquisition of $\mathbf{x}_1$ and $\mathbf{x}_2$ we can form both columns of the matrix U by normalizing these vectors.

$$\mathbf{x}_1 = \begin{bmatrix} -1 \\ 0 \end{bmatrix} \qquad \mathbf{x}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Generate the columns of U by turning $\mathbf{x}_1$ and $\mathbf{x}_2$ into unit vectors (U and V must be orthonormal).

$$\mathbf{u}_1 = \frac{\mathbf{x}_1}{\|\mathbf{x_1}\|} \qquad \mathbf{u}_2 = \frac{\mathbf{x}_2}{\|\mathbf{x_2}\|}$$

$$\mathbf{u}_1 = \begin{bmatrix} -1 \\ 0 \end{bmatrix} \qquad \mathbf{u}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Now that we have unit eigenvectors $\mathbf{u}_1$ and $\mathbf{u}_2$ we can form U.

$$U = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

Step 4: Finding V

We use a similar process using equation (4) to find unit eigenvectors of V...

$$\mathbf{v}_1 = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix} \qquad \mathbf{v}_2 = \begin{bmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$$

Now that we have unit eigenvectors $\mathbf{v}_1$ and $\mathbf{v}_2$ we can form V.

Introduction
SVD Overview
SVD Example
Images and SVD...
SVD vs. Memory
MATLAB

$$V = \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} \\ 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix}$$

Step 5: The complete SVD

By securing U, Σ, and V the factorization of A is realized and the SVD is complete.

$$A = U\Sigma V^T$$

$$\begin{bmatrix} 2 & 2 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \sqrt{8} & 0 \\ 0 & \sqrt{2} \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ -1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix}$$

Images and SVD Compression

Image Matrices

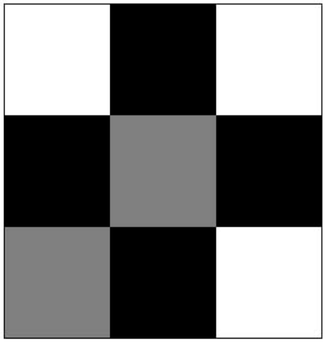
So what in the world does SVD have to do with image compression? Well before diving into this question, it's important to understand that an image can be represented as an m by n matrix, where m, the number of rows, is the pixel height of the image, and n, the number of columns, is the pixel width of the image. Every subsequent value inside the matrix tells the computer how bright to display the corresponding pixel. This is most easily understood in grayscale images, where every value within the matrix runs from 0 (black) to 1 (white). For example, suppose you have a 3 pixel by 3 pixel image as shown in Figure 1.

As predicted, these matrices can become exceeding large and taxing on storage space as their dimensions increase. For example, a typical desktop wallpaper

Introduction
SVD Overview
SVD Example
Images and SVD...
SVD vs. Memory
MATLAB

Home Page
Title Page
◀◀ ▶▶
◀ ▶

Go Back
Full Screen
Close
Quit



$$= \begin{bmatrix} 1 & 0 & 1 \\ 0 & .5 & 0 \\ .5 & 0 & 1 \end{bmatrix}$$

Figure 1: Example 3×3 grayscale image: This is an enlarged version of course, since a 3 pixel by 3 pixel image would be almost impossible to discern.

is 1280 by 1024 pixels, which equates to 1.31×10^6 pixel values stored within the image's matrix. And this is only a grayscale image. A color image's matrix, in fact, contains three times as many values for the same size picture.

To display any given color value, a computer typically breaks the color into its three primary components: red, green, and blue, commonly referred to as RGB. Thus each pixel has three values associated with it, one value for red, one for green, and one for blue, each ranging from 0 (color is absent) to 1 (completely saturated). It stores each of these colors in a different layer (three total) where each individual layer can be treated as a grayscale image. See Figure 2.

Once it's understood how an image is represented as a matrix, it can be understood how useful SVD is.

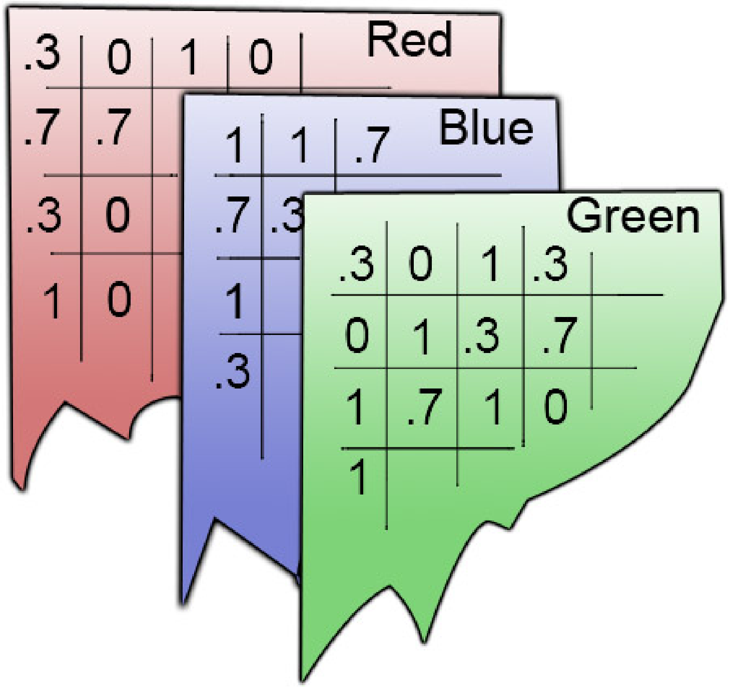


Figure 2: Each layer is a grayscale image matrix, but when overlayed using color, produce the full RGB spectrum

Introduction
SVD Overview
SVD Example
Images and SVD . . .
SVD vs. Memory
MATLAB

SVD Compression

Before discussing SVD's relationship with images, we will explore how SVD can compress any form of data.

SVD takes a matrix, square or non-square, and divides it into two orthogonal matrices and a diagonal matrix (See previous Section). This allows us to rewrite our original matrix as a sum of much simpler rank one matrices.

$$A = U\Sigma V^T$$

$$A = \begin{bmatrix} \mathbf{u_1} & \mathbf{u_2} & \cdots & \mathbf{u_{n-1}} & \mathbf{u_n} \end{bmatrix} \begin{bmatrix} \sigma_1 & & & & \\ & \sigma_2 & & & \\ & & \ddots & & \\ & & & \sigma_{n-1} & \\ & & & & \sigma_n \end{bmatrix} \begin{bmatrix} \mathbf{v_1}^T \\ \mathbf{v_2}^T \\ \vdots \\ \mathbf{v_{n-1}}^T \\ \mathbf{v_n}^T \end{bmatrix}$$

$$A = \mathbf{u_1}\sigma_1\mathbf{v_1}^T + \mathbf{u_2}\sigma_2\mathbf{v_2}^T + \cdots + \mathbf{u_{n-1}}\sigma_{n-1}\mathbf{v_{n-1}}^T + \mathbf{u_n}\sigma_n\mathbf{v_n}^T$$

$$A = \sum_{i=1}^{n} \sigma_i \mathbf{u_i}\mathbf{v_i}^T$$

Although this may not seem impressive at first glance, it holds the key to compression. Since $\sigma_1 \geq \cdots \geq \sigma_n$, the first term of this series will have the largest impact on the total sum, followed by the the second term, then the third term, etc. **This means we can approximate the matrix A by adding only the first few terms of the series!** This addition of rank one matrices results in a rank k matrix, where k is the total number of rank one matrices added together.

$$A_{approx.} = \sum_{i=1}^{k} \sigma_i \mathbf{u_i} \mathbf{v_i}^T$$

As k increases, the image quality increases, but so too does the amount of memory needed to store the image. This means smaller ranked SVD approximations (smaller value for k) are preferable. Under rare conditions, usually with geometric shapes, an image can be almost perfectly replicated with a rank one or rank two SVD. Figure 3 exemplifies this rare case,

By storing only the first two columns of U and V and their respective singular values, the square image can be replicated *exactly* while taking up only .02% of the original storage space. (We'll discuss how to calculate the amount of memory saved via SVD compression in the next section). The circle, however, can't be replicated exactly without using a full ranked SVD. This is due to the "rectangular"nature of SVD compression. You can actually *see* how the compression breaks down the matrix for a rank one approximation. Notice that every row of pixels is the same row, just multiplied by a different constant, which changes the overall intensity of the row. The same goes for columns; every column of pixels is actually the same column multiplied by a different constant. See Figure 4

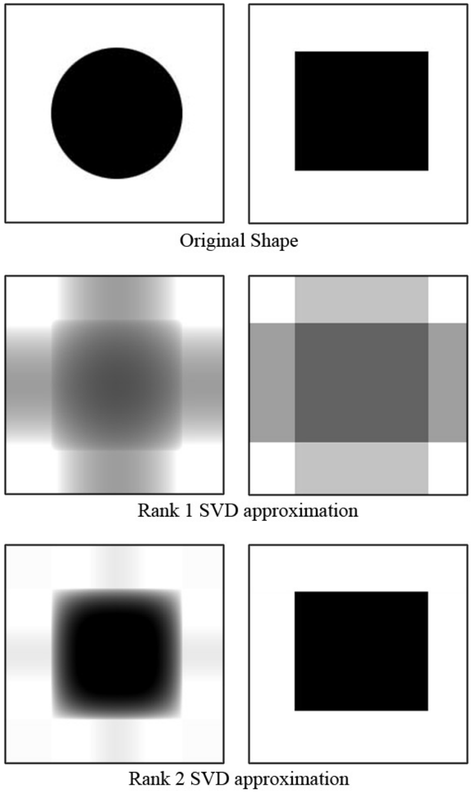


Figure 3: Rank 1 and rank 2 SVD approximations of a circle and a square.

Introduction
SVD Overview
SVD Example
Images and SVD...
SVD vs. Memory
MATLAB

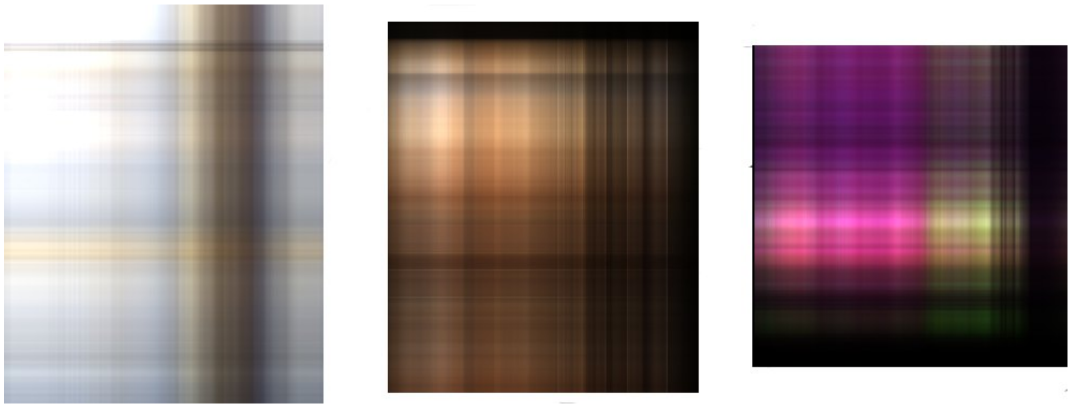


Figure 4: Each picture is simply one row and one column multiplied together.

SVD vs. Memory

The next question becomes, “How much memory does this save?” Short answer: a lot! But it’s important to understand exactly how much “a lot” is. (Note: from here on out, when referring to the number of values (memory) required to store a given image C, I will use the notation C_M)

Calculating Memory

First, we need to see how much memory a non-compressed image, I, requires. Recall that for an $m \times n$ pixel grayscale image, the computer must store mn values, one value for each pixel.

$$I_M = mn$$

While a rank k SVD approximation of I (I’ll refer to this approximation as A), on the other hand, consists of three matrices, U, Σ, and V, so the number of values the computer must store is a little trickier to compute. Let’s start with U. Originally U is an $m \times m$ matrix, but we only want the first k columns. This means U can be stored as an $m \times k$ matrix with mk values.

$$U_M = mk$$

Similarly, we only want the first k columns of V (which become the rows of V^T), so we only need to store V as an $n \times k$ matrix with nk values.

$$V_M = nk$$

Lastly, because we're stripping off only the first k columns of U and V, we only need the first k singular values.

$$\Sigma_M = k$$

Now it's possible to compute the total number of values required to store this rank k approximation,

$$\begin{aligned} A_M &= U_M + V_M + \Sigma_M \\ A_M &= mk + nk + k \\ A_M &= k(m + n + 1) \end{aligned}$$

Maximum rank k

Here's where a problem arises. Most images aren't nice geometric shapes, as in Figure 3. In fact, almost all image matrices have full rank, meaning that to replicate an image exactly using SVD, k would have to equal n (where n is the smaller dimension of the original matrix I). Solving for A_M in this case,

$$\begin{aligned} A_M &= n(m + n + 1) \\ A_M &= mn + n^2 + n \end{aligned}$$

But this causes A_M, our "compressed" image, to take up far more space than the original image ($I_M = mn$), utterly defeating the purpose of SVD compression! This implies there are important limits on k for which SVD actually *saves* memory. We want $A_M \leq I_M$.

$$\begin{aligned} A_M &< I_M \\ k(m+n+1) &< mn \\ k &< \frac{mn}{m+n+1} \end{aligned}$$

Although it has not been mentioned thus far, the same rule for k applies to color images. In the case of color $I_M = 3mn$, while

$$\begin{aligned} A_M &= 3(U_M + V_M + \Sigma_M) \\ &= 3k(m+n+1). \end{aligned}$$

Thus,

$$\begin{aligned} k &< \frac{3mn}{3(m+n+1)} \\ k &< \frac{mn}{m+n+1}. \end{aligned}$$

Introduction
SVD Overview
SVD Example
Images and SVD...
SVD vs. Memory
MATLAB

MATLAB

Here's where MATLAB joins the game. Throughout this paper, all of the compressed images have been generated via code similar to the following,

```
I = imread('pig.jpg');
I = im2double(I);
k=5;
for
i = 1:3
    [U,S,V] = svd(I(:,:,i));
    A(:,:,i) = U(:,1:k)*S(1:k,1:k)*V(:,1:k)';
end
A(A>1)=1; A(A<0)=0;
imshow(A) axis off
imwrite(A,'pig_compressed.jpg')
```

Step 1: Obtain an Image Matrix.

The first line of code uses the function `imread` to generate an image matrix based on the file `pig.jpg`. By default, all images loaded into matrices are formatted as 8 bit unsigned integers in order to save memory. Unfortunately the MATLAB function `svd` will only accept single-layer matrices of double precision format. The second line of code converts I to the double precision format (the single-layer requirement gets taken care of later).

Introduction
SVD Overview
SVD Example
Images and SVD...
SVD vs. Memory
MATLAB

Introduction
SVD Overview
SVD Example
Images and SVD...
SVD vs. Memory
MATLAB

Step 2: Decompose the Matrix

This step is relatively easy due to MATLAB's built in `svd` function. We send `svd` our original matrix one layer at a time, which means we have to compute the SVD of each layer of I independently by running the `svd` command three times; hence the `for` loop, where `i` goes from 1 to 3. We save U, Σ, and V to the variables `U`, `S`, and `V` respectively.

Step 3: Reconstruct a Compressed Matrix

Each time a layer's SVD is computed, it multiplies the first k columns of U, the first k values in Σ, and the first k columns of V transposed, and stores the result in the corresponding layer of A. Just as in the previous step, we have to reconstruct the matrix one layer at a time. Thus, it, too, is inside the `for` loop.

As a final step in reconstruction (the first line after the `for` loop), the program finds all values in A that are out of bounds of our color spectrum and corrects them. Specifically it finds values less than zero and greater than 1 and sets them equal to 0 and 1 respectively.

Step 4: Write the Image

This step is fairly self explanatory. The program first displays the image with the `imshow` command, then writes the compressed image to a file named `pig_compressed.jpg` with the `imwrite`. MATLAB automatically chooses the format of this file based on the extension you include in the name of the file.

The Result!

Figure 5 shows some example compressed images using different values for k based off of a 383×290 pixel color image.

The final k was chosen so that $A_M = I_M$, or in other words, the compressed image takes up the same amount of memory as the uncompressed image. Can you tell the difference? Normally a subjective question has no place in math, but compressing an “image” is entirely subjective. A better question might be, what’s the smallest acceptable value k for which the image is “good enough”? And the answer is completely up to you.

The link below leads to a GUI program created in MATLAB that allows the user to compress any given image with different values of k.

http://online.redwoods.edu/darnold/linalg/fall2006/SVD/SVDgui.m

Introduction
SVD Overview
SVD Example
Images and SVD...
SVD vs. Memory
MATLAB

Home Page
Title Page
◀◀ ▶▶
◀ ▶
Page 20 of 22
Go Back
Full Screen
Close
Quit

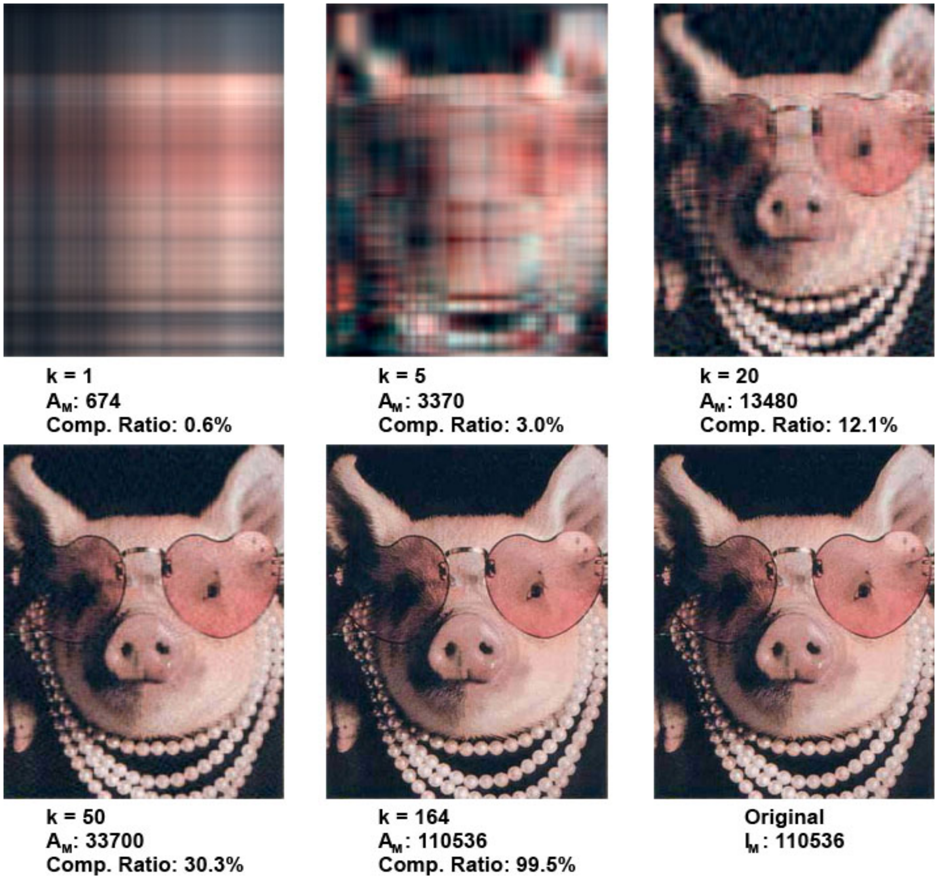


Figure 5: Example compressions created via the MATLAB code included in this document

Introduction
SVD Overview
SVD Example
Images and SVD...
SVD vs. Memory
MATLAB
Home Page
Title Page
◀◀ ▶▶
◀ ▶
Page 21 of 22
Go Back
Full Screen
Close
Quit

References

[1] Strang, Gilbert.*Introduction to Linear Algebra.* revised 2005, Wellesley-Cambridge Press.

[2] Hourigan, Jody S., and Lynn V. McIndoo. "The Singular Value Decomposition." Dec. 1998. College of the Redwoods.

[3] Manuel, Nina, and Adams, Bethany. "The SVD and Image Compression." Dec. 2005. College of the Redwoods.

[4] Arnold, Ben "An Investigation into using Singular Value Decomposition as a method of Image Compression" Dec. 2005. College of the Redwoods. September 2000. University of Canterbury

[5] Leon, Steven, and Eugene, Herman, and Faulkenberry, Richard. *ATLAST: Computer Excercises for Linear Algebra.* 1996. Prentice Hall.

[6] Dave Arnold. Thanks to our beloved instructor for all of his time and effort, without whom we would have failed.

Introduction
SVD Overview
SVD Example
Images and SVD...
SVD vs. Memory
MATLAB

Home Page
Title Page
Page 22 of 22
Go Back
Full Screen
Close
Quit