COMPARISON OF SVD AND FFT IN IMAGE COMPRESSION

[1]Vinita Cheepurupalli, [2]Sierra Tubbs, [3]Khadijah Boykin, [4]Dr. Naima Naheed

[1] Spring Valley High School, 120 Sparkleberry Ln, Columbia, SC 29229

[2]Biology, Chemistry, and Environmental Health Science Department

[3]Physics and Engineering Department

[4]Math and Computer Science Department

Benedict College, 1600 Harden Street, Columbia SC 29204

Email addresses for 4 persons mentioned above:

cheepurupalli@yahoo.com

sierratubbs424@gmail.com

kboykin13@sljhs.org

naheedn@benedict.edu

Abstract: Two image compression methods are compared: Singular Value Decomposition (SVD) and Fast Fourier Transform (FFT). SVD is the factorization of a real or complex matrix, while FFT is an algorithm which allows low pass and high pass filtering with a great degree of accuracy. FFT is also a process that vastly reduces the time needed to compute large matrices. Distortion and compression ratios for each method were calculated at different parameters. Images were compressed without sacrificing significant image quality. Comparing the compression ratio, distortion, and visual quality of the images, FFT was determined to be the better of the two compression methods.

Contact Person: Dr. Naima Naheed

Email: naheedn@benedict.edu

keywords: Image, SVD, FFT, Compression Ratio, Distortion

978-1-4673-9795-7/15 $31.00 © 2015 IEEE
DOI 10.1109/CSCI.2015.56

Section I: Introduction

Image compression is in high demand because it reduces the computational time and consequently the cost in image storage and transmission. The basis for image compression is to remove unimportant and redundant data while maintaining the compressed image quality in an acceptable manner.

In this paper, two different still image compression methods were compared: Singular Value Decomposition (SVD) and fast Fourier transform (FFT). These two lossy compression techniques were applied to a single image, and the methods' performances were compared using measures of compression ratio, distortion, and visual quality.

Previous studies have compared image compression methods, but none have compared these two. Also, an image processing toolbox was utilized, and all MATLAB codes related to this work are listed in the paper.

Graphical analysis and visual quality of the images led to the conclusion that FFT is more effective in image compression than SVD. Section II describes the basic idea about SVD and FFT. Section III contains the experimental data and Section IV contains the conclusion.

Section II: Formulation of the Problem

SVD: If A is an m x n matrix, the singular values of A are the square roots of the eigenvalues of A^TA and are denoted by $\sigma_1, \dots \sigma_n$. It is conventional to arrange the singular values such that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$. There exist orthogonal matrices U and V and a diagonal matrix $\sum$, such that $A = U\sum V^T$. Here, U is an m x m matrix and V is an n x n matrix, so that $\sum$ is rectangular with the same dimension as A.

The SVD of an m x n matrix A will be observed [1-4]. Here, the transformation is a mapping from the domain $\mathbb{R}^n$ to the range $\mathbb{R}^m$, so it is rational to inquire for a natural basis for each of the domain and the range. The columns of matrices V and U are used to represent vectors in the domain and the range of transformation. According to the magnitude of the singular values, the transformation simply dilates, contracts, or possibly discards some components. From this perspective, the SVD describes how to choose an orthonormal basis so that the transformation is represented by a diagonal matrix with its simplest possible form.

To construct an orthogonal matrix V, an orthonormal basis $\{v_1, \dots, v_n\}$ for $\mathbb{R}^n$ must be found consisting of eigenvectors of the n x n symmetric matrix A^TA. This orthonormal basis $V = [v_1 \ \dots \ v_n]$ is an orthogonal n x n matrix.

For an orthogonal matrix U, $[Av_1, \dots, Av_n]$ is an orthogonal set of vectors in $\mathbb{R}^m$. If it is given that v_i is an eigenvector of A^TA corresponding to an eigenvalue λ_i, then, for $i \neq j$, we have

$$(Av_i).(Av_j) = (Av_i)^T(Av_j)$$

$$= v_i^T A^T A v_j$$

$$= v_i^T \lambda_j v_j$$

$$= \lambda_i(v_i.v_j) = 0$$

since eigenvectors v_i are orthogonal. Furthermore, the lengths of the vectors $Av_1, \dots, Av_n$ are the singular values of A, and there are r nonzero singular values $Av_i \neq 0$ if and only if $1 \leq i \leq r$ [1-4]. Therefore, $Av_1, \dots, Av_r$ are linearly independent vectors found in *Col A*. Each Av_i is normalized to obtain an orthonormal basis $\{u_1 \dots u_r\}$, where $u_i = \frac{1}{\|Av_i\|}Av_i = \frac{1}{\sigma_i}Av_i$ for $i = 1, \dots, r$.

This normalization guarantees that $\{u_1 \dots u_r\}$, is an orthonormal set in $\mathbb{R}^m$, but if r < m, it will not be a basis for $\mathbb{R}^m$. The set $\{u_1 \dots u_r\}$ is prolonged to an orthonormal basis $\{u_1 \dots u_m\}$ for $\mathbb{R}^m$. Then, U becomes equal to $[u_1 \dots u_m]$.

Verification is necessary that with U, V, and $\sum$ as defined, we have $A = U\sum V^T$. Since, $V^T = V^{-1}$, this is equivalent to showing that $AV = U\sum$

It is known that $Av_i = \sigma_i u_i$ for $i = 1, \dots, r$ and $\|Av_i\| = \sigma_i = 0$ for $r + 1, \dots, n$. Hence, $Av_i = 0$ for $i = r + 1, \dots, n$. Therefore,

$AV = [Av_1 \dots Av_r \ 0 \dots 0] = [\sigma_1 u_1 \dots \sigma_r u_r \ 0 \quad 0]$

$$= [u_1 \dots \dots u_m] \begin{bmatrix} \sigma_1 & \dots & \dots & \dots & 0 \\ \dots & \dots & \dots & \dots & 0 \\ \dots & \dots & \dots & \dots & 0 \\ \dots & \dots & \dots & \sigma_r & 0 \\ 0 & 0 & \dots & 0 & 0 \end{bmatrix} = U\sum$$

Properties:

If we let A represent an m x n matrix with a singular value decomposition of $U\sum V^T$, then

1. The singular values $\sigma_1, \ldots \sigma_n$ of A are unique; however, the matrices U and V are not unique.
2. Since $A^TA = V\sum^T\sum V^T$, it follows that V diagonalizes A^TA and that the $v_j{}'$s are eigenvectors of A^TA
3. As $AA^T = U\sum\sum^T U^T$, it is clear that U diagonalizes AA^T and that the $u_j{}'$s are eigenvectors of AA^T.
4. The m x n matrix A can be written as the sum of rank-one matrices
$$A = \sum_{i=1}^{r} \sigma_i u_i v_i^T, \qquad (1)$$
where r is the rank of A, and u_i and v_i are the i^{th} columns of U and V, respectively.

Proof: 4. Given that A= $U\sum V^T$:

$$= U\begin{bmatrix}\sigma_1 & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & \sigma_r\end{bmatrix}V^T = U\left(\begin{bmatrix}\sigma_1 & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & 0\end{bmatrix}+\right.$$

$$\left.\begin{bmatrix}0 & \cdots & 0\\ \vdots & \sigma_2 & \vdots\\ 0 & \cdots & 0\end{bmatrix}+\ldots+\begin{bmatrix}0 & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & \sigma_r\end{bmatrix}\right)V^T$$

$=\sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \ldots + \sigma_r u_r v_r^T$

Property 4 is the low rank approximation property of SVD. The best least square approximation of A whose rank $p \le r$ is provided by retaining the first p terms of equation (1).

FFT: For simplicity, one dimensional signal is first addressed, followed by two dimensional images. A continuous signal $x(t)$ can be defined on a time interval [0, T] and therefore can be sampled at N times where $t = nT/N$ for $n = 0, 1, \ldots, N-1$. We get a discretized signal $x = (x_0, x_1, \ldots, x_{N-1})$, a vector in $\mathbb{R}^N$. The $x(t)$ can be written as an infinite sum of basic waveforms $e^{2\pi ikt/T}$ for $k \in Z$ and sampled at $t = nT/N$. The sequence of basic waveform which we denote by $E_{N,k}$, indexed by *k,* is of the form:

$$E_{N,k} = \begin{bmatrix} e^{2\pi ik0/N} \\ \vdots \\ \vdots \\ \vdots \\ e^{2\pi ik(N-1)/N} \end{bmatrix}.$$

$E_{N,k}$ forms an orthogonal basis for C^N. Any $x \in C^N$ can be written as $x = \sum_{k=0}^{N-1} \frac{(x,E_{N,k})}{(E_{N,k},E_{N,k})} E_{N,k} = \frac{1}{N}\sum_{k=0}^{N-1}(x, E_{N,k})E_{N,k}$ using the fact that $(E_{N,k}, E_{N,k}) = N$ for each k.

The discrete Fourier transform (DFT) of x is the vector $\boldsymbol{X} \in C^N$ with components

$$X_k = (x, E_{N,k}) = \sum_{m=0}^{N-1} x_m\, e^{-2\pi ikm/N} \qquad (2)$$

for $0 \le k \le N-1$. Then we can let $\boldsymbol{X} \in C^N$ be a vector $(X_0, X_1, \ldots, X_{N-1})$.

The inverse discrete Fourier transform (IDFT) of X is the vector $x = \frac{1}{N}\sum_{m=0}^{N-1} X_m\, e^{2\pi ikm/N}$.

The IDFT shows how to synthesize x from $E_{N,k}$, which are the basic waveforms. $\boldsymbol{X}$ =DFT($\boldsymbol{x}$) and $\boldsymbol{x}$ =IDFT(X) are used to indicate the DFT or IDFT of a given vector. The DFT and IDFT were used for analyzing sampled signals and images. Every object that exists in the time domain has a representation in the frequency domain. The DFT and IDFT allow us to easily move back and forth between the discrete time and the frequency domains.

The computational cost for evaluating the sums of equation (2) was examined. The computation for any given X_k requires N complex multiplication followed by N-1 additions for a total of 2N-1 operations. Since there are X_k for $0 \le k \le N-1$, we had to perform $N(2N-1) = 2N^2 - N$ operations.

The quantity $\|\boldsymbol{x}\|^2$ is interpreted as the energy of a sampled signal *x.* Since signals often contain unnecessary components, a graphical analysis of the signal's spectrum was used that revealed the fact that high frequency components contribute very little energy. To compress a signal or an image, higher frequencies needed to be removed.

In 1965, Cooley and Tuckey published an algorithm called FFT, similar to DFT. This algorithm shortens the work for computing an N sampled DFT from $2N^2 - N$ floating point operations (flop) down to $CN\log(N)$ operations for some constants C. This reduction is a significant saving of computation when N is large [5].

Section III: Method

Any *m x n* image is *m x n* matrix, where the entry *(i, j)* is interpreted as the brightness of pixel *(i, j)*. Here, 2 methods, SVD and FFT, were used to compress an image. The image "peppers.png," which is available in MATLAB, was used as a visualization demonstration file. The image was first converted from PNG to JPEG format since the latter is used for lossy compression.

The following command was used to view the image.

A=imread (‘peppers.tif’); imwrite (A, ‘peppers.jpg’); imshow (A)

The RGB or true color image can be seen such that the total number of bits required for each pixel is 24 (8 for red, 8 for green, and 8 for blue). To reduce the storage space, the image can be converted to grayscale, in which only 8 bits are required per pixel.

SVD: The image was compressed using SVD, and the compression ratio was calculated. The MATLAB [5-6] code is given below.

```
function [Ak,CR1] = svdPartSum(A, k )
B = double(rgb2gray(A));
[m, n] = size(B);
% m= number of row, n= number of col.
[u, s, v] = svd (B);
Ak=u(:,1:k)*s(1:k,1:k)*v(:,1:k)';
% k is the index of the singular value.
CR1= k*(m+n)/m/n; % Compression ratio
imshow (Ak, []);
end
```

FFT: After converting the RGB image to grayscale, the image was converted to the frequency domain using the command “fft2.” Seeking for the highest frequency “M”, different thresh parameters were chosen [5]. All frequencies in the image which were below a thresh parameter times M were zeroed out, resulting in less entries than the original image. Using “ifft2” command, the compressed image was converted to the time domain using the MATLAB code given below:

```
function [FO,CR2] = FFTthresh( A, t )
B=double(rgb2gray(A));
[m,n] = size(B);
A1 = fft2(B); % DFT of the image
M=max(max(abs(A1)));
Ath = (abs(A1)>t*M).*A1;
% zeroing out all the frequencies below t*M
CR2 = sum(sum(abs(Ath)>0))/m/n;
%Compression ratio
Ath = real(ifft2(Ath)); % IDFT of Ath
FO = ath; imshow(FO, [])
end
```

When the image A is approximated as O (using singular values k or thresh parameter t), the measure of the distortion is approximated as $D = \frac{\|A-O\|^2}{\|A\|^2}$ [5]. It is also called relative error in any physical application. When multiplied by 100, relative error becomes percent distortion. The MATLAB code to calculate the distortion is given below:

```
function  D  = Distortion( A, O )
% This equation calculates the distortion between
% the original image and the compressed image
B = double(rgb2gray(A));
D = 100*norm(B-O,'fro').^2/norm(B,'fro').^2;
end
```

The “fro” argument indicates that the Frobenius norm should be used for the matrices, which can be obtained by taking the square root of the sum of the squares of the matrix entries.

Section IV: Result and Conclusion

The images were created using 150, 100, 50 and 43 singular values. At a singular value of less than 43, the visual quality is inadequate. The compression ratio is given by $CR1 = \frac{k(m+n)}{m.n}$, where k indicates the index of singular values [7]. It is clear that the relation between singular values and compression ratio is linear since m and n are constants.

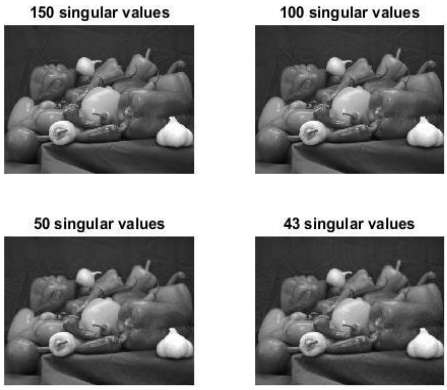

Figure 1: Compressed images using singular values

We chose different thresh parameters in the FFT method to compress the image.

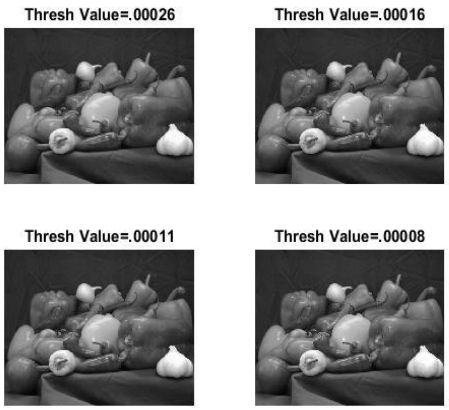

Figure 2: Compressed images using thresh values

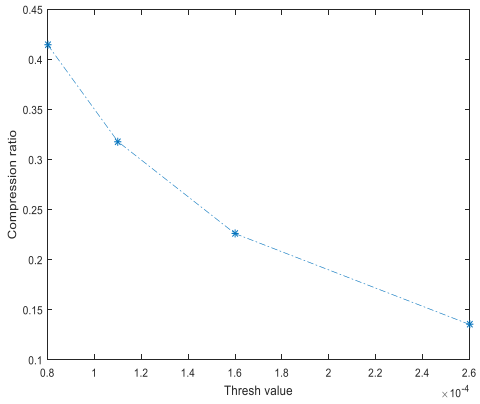

Figure 3: Relation between Thresh value and compression ratio

In Fig. 3, we see that the relation between thresh value and compression ratio is not linear.

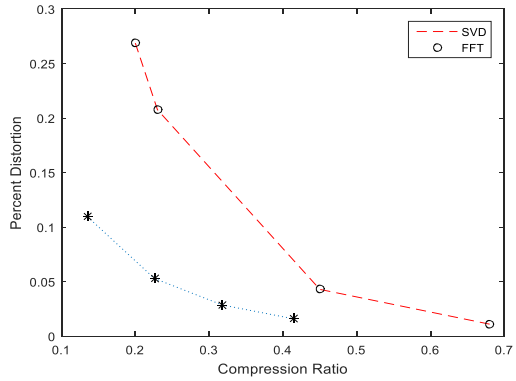

Figure 4: Comparison of SVD and FFT methods

The visual quality of the images becomes unacceptable when the number of singular values is decreased. From Figure 4, we see that at the same compression ratio, distortion is 0.27 for the image that was compressed using SVD, while the percent distortion is only 0.06 for the image when using FFT. Therefore, compressing an image using SVD is nearly five times more deteriorated than FFT. Therefore, it was concluded that FFT is a more effective approach for image compression than SVD. Further research can be conducted by comparing FFT with a wavelet method.

REFERENCES:

1. Strang, G. (2006). *Linear algebra and its applications* (4th ed.). Boston, MA: Brooks/Cole Cengage Learning.
2. Poole, D. (2006). *Linear algebra* (4th ed.). Boston, MA: Cengage Learning.
3. Lay, D. C. (2012). *Linear algebra and its applications* (4th ed.). Boston, MA: Addison-Wesley.
4. Leon, S. J. (2006). *Linear algebra with applications* (9th ed.). New York, NY: Pearson.
5. Broughton, S. A. & Bryan, K. (2009). *Discrete Fourier analysis and wavelets*. Hoboken: NJ. John Wiley & Sons, Inc.
6. Chapman, S. J. (2008). *MATLAB programming for engineers*. Natick, MA: Cengage Learning.
7. Demmel, J. W. (1997). *Applied numerical linear algebra*. Philadelphia, PA: Society for Industrial and Applied Mathematics.

ACKNOWLEDGEMENTS:

This research was supported by NSF Grant #1436222.