APPM 3310: Final Project

Claire Ely & Jasdeep Singh
110058417 — 109039912

Abstract

1 Attribution

Claire Ely worked on creating the program to compress the sample images using Fourier Transforms in MATLAB. Jasdeep Singh worked on creating the program to compress the sample images using SVD in MATLAB. Additionally Jasdeep and Claire analyzed the results from the two different image compressing approaches.

2 Introduction

Data storage, with a focus on image compression, is a relevant problem in day-to-day life. With more data being produced, the need for compression techniques to reduce storage demands is critical to maintaining how technology is currently used. Because of the prevalence of this topic, this report will explore two image compression techniques and analyze balancing data storage with compressed image quality. An important consideration in compression is lossy vs. lossless compression. Lossless image compression techniques produce a reconstructed image that is exactly the same as the original image. Lossy compression, on the other hand, leads to image quality loss but often requires less storage for the reconstructed image (Mathur and Mathur 2012). Both SVD and FFT are types of lossy compression. As such, they are a prime study ground for understanding image compression tradeoffs.

3 Mathematical Formulation

3.1 SVD

Given a $m \times n$ matrix A, the singular values of the matrix $\sigma_1, \ldots \sigma_r$ are the positive square roots of the eigenvalues $\sigma_i = \sqrt{\lambda_i}$ of the nonzero eigenvalues of the gram matrix $K = A^T A$. The eigenvectors of K are known as the singular vectors of A. (Olver and Peter J., Applied Linear Algebra)

Now given a real $n \times m$ matrix A that is nonzero, A can be factored into the form

$$A = P\Sigma Q^T \tag{1}$$

Where P is a $m \times r$ matrix ($r = rank(A)$) with orthonormal columns such that $P^T P = I$, Σ is a $r \times r$ matrix where $\Sigma = diag(\sigma_1, \ldots \sigma_r)$ has the singular values of A on its diagonal in decreasing order ($\sigma_1 > \sigma_2 \ldots \sigma_r$), and Q^T is a $r \times n$ matrix with orthonormal rows such that $Q^T Q = I$.

To solve for P, Σ, and Q^T the first step is to find the gram matrix $K = A^T A$ and solve for its eigenvalues and eigenvectors. Using the equation

$$det(K - \lambda I) = 0 \tag{2}$$

the next step is to solve for the eigenvalues of the matrix K and plug those eigenvalues back into this equation $K - \lambda I$ and find the kernel of this matrix or find

$$ker(K - \lambda I) \tag{3}$$

to get the eigenvectors of the matrix K. Now using the Gram-Schmidt process (See algorithm below in references). Now we can construct our Σ matrix by placing the square roots of the eigenvalues on the diagonal in decreasing order. Additionally we can construct our Q^T matrix by using the orthonormalized eigenvectors to form a matrix Q and taking the transpose of Q to get Q^T. Now to get the P matrix, one can solve the equation

$$Aq_i = \sigma_i p_i \tag{4}$$

for p_i so you get

$$\frac{Aq_i}{\sigma_i} = p_i \tag{5}$$

for i = 1 ... r

How can one relate this to image compression? The Σ matrix is very important in image compression. Suppose A is the greyscale pixel matrix of an image. By removing the lower valued σ_i's from the Σ matrix and replacing them with 0's we get a new matrix Σ'. If we multiply $P\Sigma' Q^T$ we get a new matrix A' that is a compressed image of the original image A. The more σ_i's there are in the matrix Σ the better the quality of the image. If very little σ_i's are in the Σ matrix then the image would be unrecognizable.

3.2 Fourier Transform

The Fast Fourier Transform (FFT), an algorithm to less expensively compute the Discrete Fourier Transform (DFT), is commonly used for image and audio compression. The DFT decomposes a function $f(x)$ into a sum of complex exponentials (or a linear combination of sines and cosines), essentially representing sampled values of a function or data set with Fourier series. This is used in place of the continuous Fourier transform for compression applications as most data sets or signals will be discrete. The DFT takes a signal from $Z \to C$

Given a discrete sequence of N complex numbers $X_k = X_0, X_1, ..., X_(N-1)$, the DFT is found by the following summation:

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{\frac{-i2pi}{N} kn}$$

$$= \sum_{n=0}^{N-1} x_n \cdot [cos(\frac{2pi}{N} kn) - i \cdot sin(\frac{2pi}{N} kn)]$$

The DFT is fairly computationally expensive, which goes against the intention of data compression. With N points, as defined above, the DFT requires O(N^2) multiplications and additions, while the FFT only needs O(NlogN) similar computations (Heckbert 98). The FFT is a more computationally efficient algorithm, which leverages the inherent symmetry of the DFT. Multiple FFT algorithms exist, but the Cooley-Tukey algorithm is most common (Mathur and Mathur 12). Without unnecessary discussion, a data set can be split into even and odd points, and each of the Fourier transforms of each set can be taken in isolation. The sts can then be merged, showing an overall reduction in the number of computations. FFT works recursively, by dividing the original data set in half and recombining the computed values.

Taking the FFT of an image produces Fourier coefficients, or values in the Fourier space. By the nature of the Fourier transform, the majority of the coefficients are quite small. As such, throwing out these negligible values has minimal effect on the norm of the

Fourier signal. Compression via removing the smallest Fourier coefficients allows less data storage and maintaining a similar image quality.

After removing the lowest values of the coefficients, the inverse Fourier transform can be taken, which maps back from Fourier space to pixel space and produces an image similar to the original. Ideally, this image will have minimal quality loss. The similarity between the original and compressed images can be evaluated with certain performance indicators, described later in this section.

Performance indicators that represent the success of compression with a given method were used to analyze SVD and the FFT. (Pandey, Singh, and Pandey 15). 1) Percent Error 2) Compression Ratio (CR) 3) Mean Square Error (MSE) 4) Peak Signal Noise Ratio (PSNR) These indicators describe how close the compressed image is to the original image, and the number of bits needed for the compressed image.

The mathematical formulation for these measures is detailed below.

3.2.1 Percent Error

The percent error measures the loss between the compressed and original image. It is calculated by:

$$\frac{1}{mn}\sum_{j=1}^{M}\sum_{i=1}^{N}(|X_{i,j}-Y_{i,j}|)$$

3.2.2 Compression Ratio

The compression ratio represents the number of bits in the original image relative to the number of bits in the compressed image. The smaller the CR value, the lower the quality of compression, or the more information lost.
CR = (Reconstructed image size)/(Original image size) ·100

3.2.3 Mean Square Error

Mean square error, while similar to the percent error, is an alternative measure of the differences between the original and compressed image. The lower the MSE, the less error in the output image.

$$MSE=\frac{1}{mn}\sum_{j=1}^{M}\sum_{i=1}^{N}(X_{i,j}-Y_{i,j})^2$$

3.2.4 Peak Signal to Noise Ratio

The PSNR can be expressed as a function of the MSE and similarly expresses the differences between original and compressed images. The higher the PSNR, the better compression.

$$PSNR=20\cdot log_{10}\frac{255^2}{MSE}$$

4 Examples and Numerical Results (if Appropriate)

4.1 SVD Example

Below is an example of calculating the Singular Value Decomposition of a matrix A.

$$A=\begin{pmatrix}1&-1&0\\-1&2&-1\\0&-1&1\end{pmatrix} \tag{6}$$

Lets solve for $K=A^TA$

$$K=A^TA=\begin{pmatrix}1&-1&0\\-1&2&-1\\0&-1&1\end{pmatrix}\begin{pmatrix}1&-1&0\\-1&2&-1\\0&-1&1\end{pmatrix} \tag{7}$$

$$=\begin{pmatrix}2&-3&1\\-3&6&-3\\1&-3&2\end{pmatrix} \tag{8}$$

The next step is to solve for the eigenvalues of K by finding $\det(K-\lambda I)$.

$$K-\lambda I=\begin{pmatrix}2-\lambda&-3&1\\-3&6-\lambda&-3\\1&-3&2-\lambda\end{pmatrix} \tag{9}$$

$$\begin{aligned}det(K-\lambda I)&=-\lambda^3+10\lambda^2-9\lambda\\&=-\lambda(\lambda-1)(\lambda-9)\end{aligned}$$

Now we must solve for the eigenvectors. $ker(K-\lambda I)$ where $\lambda=0,1,9$.

$\lambda_1=0$:

$$\left(\begin{array}{ccc|c}2&-3&1&0\\-3&6&-3&0\\1&-3&2&0\end{array}\right)\rightarrow\left(\begin{array}{ccc|c}1&0&-1&0\\0&1&-1&0\\0&0&0&0\end{array}\right) \tag{10}$$

$$\mathrm{v}_1=\begin{bmatrix}1\\1\\1\end{bmatrix}$$

$\lambda_2=1$:

$$\left(\begin{array}{ccc|c}1&-3&1&0\\-3&5&-3&0\\1&-3&1&0\end{array}\right)\rightarrow\left(\begin{array}{ccc|c}1&0&1&0\\0&1&0&0\\0&0&0&0\end{array}\right) \tag{11}$$

$$\mathrm{v}_2=\begin{bmatrix}-1\\0\\1\end{bmatrix}$$

$\lambda_3=9$:

$$\left(\begin{array}{ccc|c}7&-3&1&0\\-3&-3&-3&0\\1&-3&-7&0\end{array}\right)\rightarrow\left(\begin{array}{ccc|c}1&0&-1&0\\0&1&2&0\\0&0&0&0\end{array}\right) \tag{12}$$

$$\mathrm{v}_3=\begin{bmatrix}1\\-2\\1\end{bmatrix}$$

Now we must use the Gram-Schmidt process to get an orthonormal basis. But because all the eigenvectors are orthogonal with each other (their dot product is 0) so we dont have to do the Gram-Schmidt process. We do have to normalize them.

$$q_1=\frac{v_1}{||v_1||}=\begin{bmatrix}1/\sqrt{3}\\1/\sqrt{3}\\1/\sqrt{3}\end{bmatrix} \tag{13}$$

$$q_2 = \frac{v_2}{||v_2||} = \begin{bmatrix} -1/\sqrt{2} \\ 0 \\ 1/\sqrt{2} \end{bmatrix} \tag{14}$$

$$q_3 = \frac{v_3}{||v_3||} = \begin{bmatrix} 1/\sqrt{6} \\ -2/\sqrt{6} \\ 1/\sqrt{6} \end{bmatrix} \tag{15}$$

Now that we have our orthogonal basis, we can calculate our P matrix using the equation $\frac{1}{\sigma_i} A q_i = p_i$.

$$p_1 = \frac{1}{3} \begin{pmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{pmatrix} \begin{bmatrix} 1/\sqrt{6} \\ -2/\sqrt{6} \\ 1/\sqrt{6} \end{bmatrix} = \begin{bmatrix} 1/\sqrt{6} \\ -\sqrt{2/3} \\ 1/\sqrt{6} \end{bmatrix} \tag{16}$$

$$p_2 = \frac{1}{1} \begin{pmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{pmatrix} \begin{bmatrix} -1/\sqrt{2} \\ 0 \\ 1/\sqrt{2} \end{bmatrix} = \begin{bmatrix} -1/\sqrt{2} \\ 0 \\ 1/\sqrt{2} \end{bmatrix} \tag{17}$$

Now because we have a eigenvalue that is 0, we must find a vector that is orthogonal to the other found vectors. To do this we find the kernel of the matrix where the rows are the already founded p_i's. The vector we obtain is the $\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$ but we have to normalize it so we get

$$\begin{bmatrix} 1/\sqrt{3} \\ 1/\sqrt{3} \\ 1/\sqrt{3} \end{bmatrix}$$

Now that we have our p vectors, q vectors, and we know the values of sigma we can construct our matrix below:

$$A = P \Sigma Q^T \tag{18}$$

$$\begin{pmatrix} \frac{1}{\sqrt{6}} & \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{3}} \\ -\sqrt{\frac{2}{3}} & 0 & \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{6}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} \end{pmatrix} \begin{pmatrix} 3 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{6}} & \frac{-2}{\sqrt{6}} & \frac{1}{\sqrt{6}} \\ \frac{-1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \end{pmatrix} \tag{19}$$

Using this decomposition we will decompose a greyscale image matrix. We will remove many singular values from the Σ matrix and see the effects of the new image created by that Σ matrix.

In the references section is the Image Compression code for MATLAB using SVD.

4.2 FFT Example

Using the same example matrix as above for comparison:

$$A = \begin{pmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{pmatrix} \tag{20}$$

The 2D DFT of the matrix is taken. This is done by applying the summation formula from 3.2 to each individual value in the row, and the in each column. Because this is computationally time-consuming by hand (and to better display this project's methodology), the result of the MATLAB fft2() function is shown below.

$$fft2(A) = \begin{pmatrix} 0+0i & 0+0i & 0+0i \\ 0+0i & -1.5+2.5981i & 6+0i \\ 0+0i & 6+0i & -1.5+2.5981i \end{pmatrix} \tag{21}$$

Note the significant amount of zero (or truncated to zero through the MATLAB function) values. From here, a standard amount of values are chosen to be kept. If 33% of the values in the matrix are kept, then the largest (by absolute value) 66% of values are left. This is achieved in the code by transforming the matrix into a vector, ordering by size, and removing the smallest x%. The smallest values are set to 0. Here, five out of nine values are zero, so only one non-zero value is removed. This would be the -1.5 + 2.5981i value, resulting in the new matrix B.

$$B = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 6+0i \\ 0 & 6+0i & -1.5+2.5981i \end{pmatrix} \tag{22}$$

From here, B has to be shifted as the MATLAB function auto-shifts all low frequency values. The matrix is processed by the MATLAB ifft2() function, which is the inverse of fft2(), and the magnitude is found to return real values (and move out of the Fourier transform space).

$$ifft2(B) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 0 & 0 \end{pmatrix} \tag{23}$$

Notice the number of values sent to zero, especially considering the percentage of values removed. This matches the remaining three values expected with three values from the original matrix.

4.3 SVD Compressed Images

Below are the compressed images for sample images 1 and 2 with their singular values and % errors.

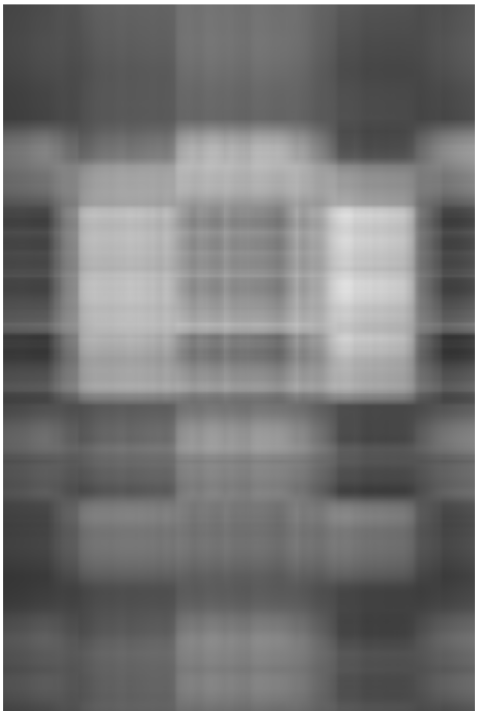

Figure 1: *3 singular values with 25% error using SVD*

Figure 3: *21 singular values with 5% error using SVD*

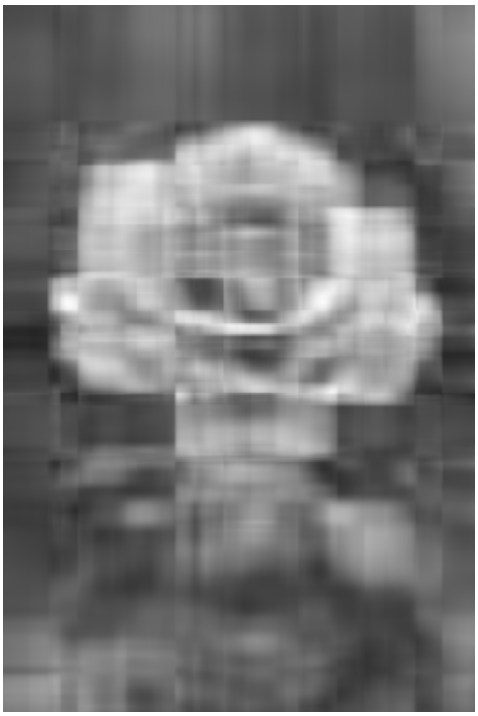

Figure 2: *9 singular values with 10% error using SVD*

Figure 4: *113 singular values with 1% error using SVD*

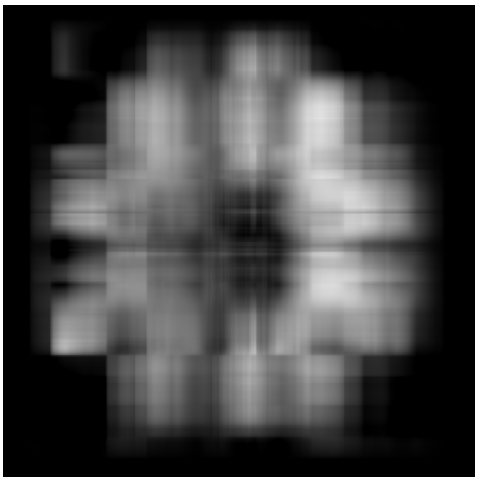

Figure 5: *4 singular values with 75% error using SVD*

Figure 8: *440 singular values with 1% error using SVD*

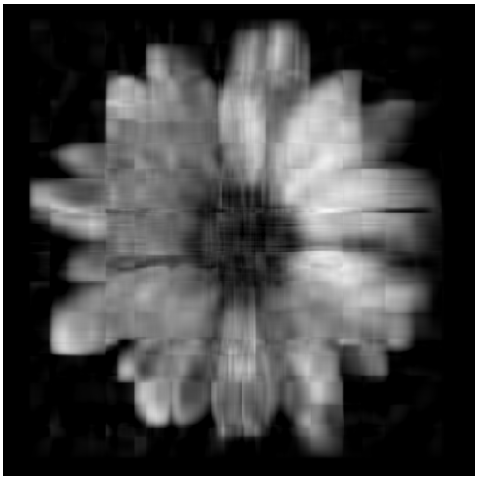

Figure 6: *12 singular values with 50% error using SVD*

4.4 Fourier Transform Compressed Images

Below are the compressed images for sample images 1 and 2 using Fourier Transform with their errors.

Figure 7: *36 singular values with 25% error using SVD*

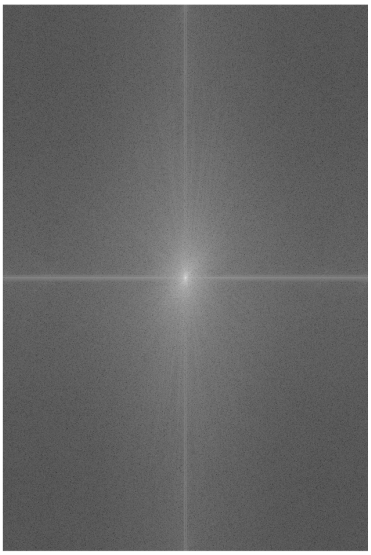

Figure 9: *Fourier Coefficients of Image 1, Visually*

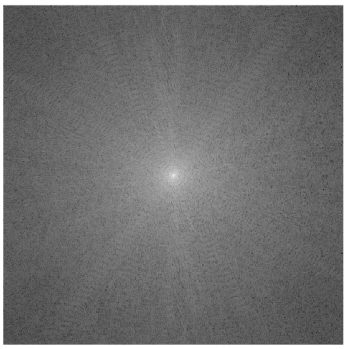

Figure 10: *Fourier Coefficients of Image 2, Visually*

Figure 12: *Fourier Compression with 50% error 75% of values removed*

Figure 11: *Fourier Compression with 75% error 99% of values removed*

Figure 13: *Fourier Compression with 25% error and 50% of values removed*

Figure 14: *Fourier Compression with 10% error 25% of values removed*

Figure 15: *Fourier Compression with 5% error 10% of values removed*

Figure 16: *Fourier Compression with 1% error 1% of values removed*

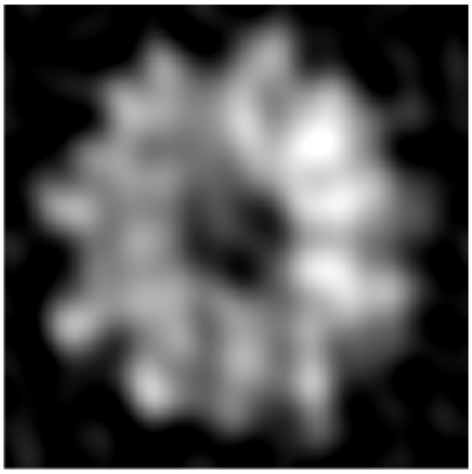

Figure 17: *Fourier Compression with 75% error 95% of values removed*

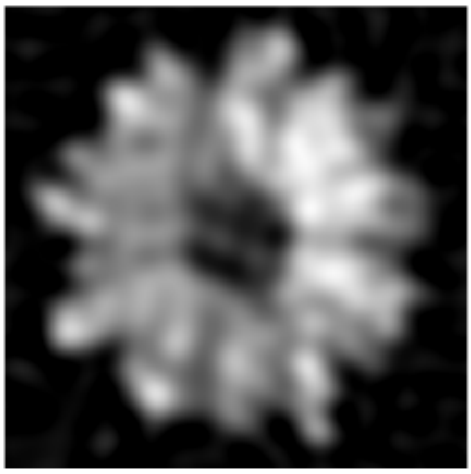

Figure 18: *Fourier Compression with 50% error 85% of values removed*

Figure 19: *Fourier Compression with 25% error 60% of values removed*

Figure 20: *Fourier Compression with 10% error 40% of values removed*

Figure 21: *Fourier Compression with 1% error 1% of values removed*

5 Discussion and Conclusions

5.1 FFT Conclusions

In order to see significant quality loss in the FFT - compressed images, a large number of coefficients had to be removed from the transformed matrix. This is representative of the insignificance of the majority of Fourier coefficients: many are so close to zero that removing them from the matrix entirely has little-to-no discernible image impact. This shows that data storage today often includes ”unnecessary” values in order to have higher quality, which is often unrecognizable by consumers.

Error in FFT-compression appears to proceed exponentially. The lower percent error values all seem to visually appear fairly close to the original image, but moving beyond 25% error (Figure 13 for Image 1 and 19 for Image 2) to the higher errors shows notably different images. The difference between 25% error and 50% error for both images is significant, with 50% error having an impressive quality declines, showing the loss-iness of the FFT method.

FFT Compression exemplified the differences in compression applications between different images. Comparing Image 1, the rose, and Image 2, the sunflower; different percentages of values needed to be removed to reach the same levels of error. For the sunflower, which had much darker regions and little variations in grey-color (as it was black and white to begin with), needed significantly higher percentages of Fourier coefficients removed to reach the same level of error. This logically follows from 1) the large amounts of low frequency regions in the image, and 2) minimal variation allowing for the display of the image to be similar regardless of the missing data values. This informs image compression techniques such that more complex images (contain more variation, are colorful, have important details) inherently require more storage space.

In terms of applying this knowledge to compression techniques, it is more effective for quality preservation to standardize the output error of the image rather than the percentage of values removed. On a larger scale, data collection methods must efficiently calculate the number of Fourier coefficients needed to reach a certain percent error. In this project, this was achieved through testing a variety of cases, however this opposes the purpose of image compression in the real world. As such, an alternative is needed.

5.2 SVD Conclusions

In SVD, the smaller singular values of a matrix are of almost no importance. Where as the larger singular values have a greater effect on the image. There is a certain point or threshold where including more singular values has no effect on the human eye. Meaning, after a certain point the human eye cannot tell the difference if you add a couple of smaller valued singular values. The larger singular values of a matrix have an interesting effect on an image as you can see in Figure 1 and 2. The image created by the 3 singular values in Figure 1 and the 9 singular values in Figure 2 create blocks. Almost as if the original image was split up into sections and the subsections pixel is equal to the average of all the pixels in the subsection. As more and more singular values are added (Figure 3 and 4), you can see the original matrix is split into even more sections and the pixel value of these sections were the average of all the pixels in that section.

Additionally in Figure 1, one can notice that even with 3 singular values there was a 25% error when compared with the original image matrix. Where as in Figure 4, with 4 singular values there was a 75% error when compared to the original image matrix. The 1st, 2nd, and 3rd singular values for image 1 and 2 are respectively (2.728e5, 4.341e4, 3.551e4) (7.611e4, 2.360e4, 1.297e4). You can notice that the singular values for image 1 are greater than the singular values for image 2. Hence the error for image 1 for a certain number of singular values may be less than the error for image 2 with the same number of singular values.

5.3 Comparison

The primary visual difference between SVD-compression and FFT-compression was the block-style compression used in SVD, versus even compression across the entire image for FFT. This ties to the difference in the numerical methods. SVD removes the singular values corresponding to an entire ”block” region while FFT removes insignificant Fourier coefficients at each pixel point based upon size. The removed points in FFT-compression are randomly spaced throughout the entire image, leading to a less drastic decrease in image quality between the numbers of values removed.

When comparing the average squared difference between the image matrices of Figures 4 and 16, the value (2.7021) is actually not that big. This shows that, the less compressed an image is the less the average squared difference is between SVD and FFT. But if you compares Figure 3 and 15 the average squared difference is 78.7738. This value is significantly bigger which would make sense because if you notice Figures 3 and 15. In Figure 3 the compressed image looks like blocks where as for Figure 15 the image looks uniformly compressed rather than blocks. Hence the average squared distance is significantly larger. When comparing the average squared distance of compressed images with larger errors of SVD and FFT, value grows exponentially as the error grows. Hence the two different compressions.

6 Code Appendix

6.1 SVDCompression.m

```
close all
clear all
clc

% We need to be able convert the image to matrix form
image = imread('image.jpeg');
image = rgb2gray(image);
imagematrix = double(image);

% Getting SVD decomposition of the imagematrix
[P, E, Q] = svd(imagematrix);

% Picking the number of singular values for the compressed image
singularvalues = 900;

% Keeping only # of singular values from above and setting the rest to 0
Ecompressed = E;
Ecompressed(singularvalues:end,:) = 0;
Ecompressed(:,singularvalues:end) = 0;

% Getting the compressed image matrix
imagecompressed = P * Ecompressed * Q';

% Displaying the compressed image
figure;
imshow(uint8(imagecompressed));
title("Image with " + singularvalues + " singular values");

% Initializing the variables used to display the error
errortable = [];
numberofsingularvalues = [];

% For loop that will calculate the difference-squared between the
% compressed image and the original image
for singularvalues = 2:2:800
    disp(singularvalues);
    Ecompressed = E;
    Ecompressed(singularvalues:end,:) = 0;
    Ecompressed(:,singularvalues:end) = 0;
    imagecompressed = P * Ecompressed * Q';
    errormatrix = abs(imagecompressed-imagematrix) ./ abs(imagematrix);
    errormatrix(isinf(errormatrix)|isnan(errormatrix)) = 0;
    error = mean(errormatrix, 'all');
    errortable = [errortable; error];
    numberofsingularvalues = [numberofsingularvalues; singularvalues];
end

% Displaying the error graph
figure;
title('Error in compression');
plot(numberofsingularvalues, errortable);
grid on
xlabel('Number of Singular Values used');
ylabel('Error between compress and original image');
```

6.2 FourierCompression.m

```
close all
clear
clc

%Read image from computer, alias as A, convert to grayscale for easier computation

A = imread('image.jpeg');
B = rgb2gray(A);

```

```
figure;
imshow(A);
title('Original Image');

%take the 2D fft of the b&w image
BFt = fft2(B);

%use log-scale for fourier coeffs (too small to plot), shift by one for
%plotting errors
BFtlog = log(abs(1+fftshift(BFt)));

figure;
imshow(BFtlog,[]);
title('Fourier Transform');

%Sort the Fourier coefficients by size
BFtsort = sort(abs(BFt(:))); %vectorize, sort largest to smallest
casenum = 1;
%initalize table for display
tblsz = [4 4];
varTypes = ["double","double","double","double"];
varNames = ["Percent Kept", "CR", "MSE", "PSNR"];
eval = table('Size',tblsz,'VariableTypes',varTypes,'VariableNames',varNames);
%initialize figure for compressed img disp
figure
sgtitle('Compressed Images')

%compress images for each percent of "kept" coefficients, calc performance
%indicators and add to table
for keep = [.99 .1 .01 .001]

        subplot(2,2,casenum)

        keepnum = BFtsort(floor((1-keep)*length(BFtsort)));
        indices = abs(BFt)>keepnum; %find values above keepnum threshold
        %multiplying indices matix w Fourier coeff matrix will zero out any
        %values below the keep number
        CFt = BFt.*indices;
        C = uint8(ifft2(CFt)); %inverse fourier 2D of C
        imshow(C) %plot
        title(['',num2str(keep*100),'%'])

        %Performance Indicators:
        %MSE with Matlab fxn
        MSE = immse(B,C);

        %CR - this must be calculated using the number of nonzero entries
        %this is producing a zero CR value for 0.1% compression, why??
        dimB = (size(B,1))*(size(B,2));
        CR = 100*(dimB-keepnum)/dimB;

        %find PSNR in dB
        PSNR = 20*log((255^2)/(MSE));

        %append table
        tablerow = {keep, CR, MSE, PSNR};
        eval(casenum,:) = tablerow;

        casenum = casenum + 1;
end

disp(eval)
imgsz = size(B,1)*size(B,2)
```

7 Additional Information

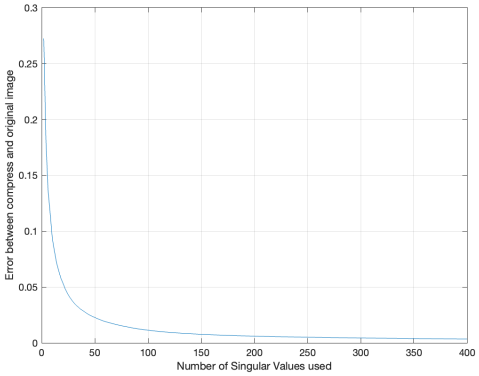

Figure 22: *Error vs. Singular Values for sample image 1*

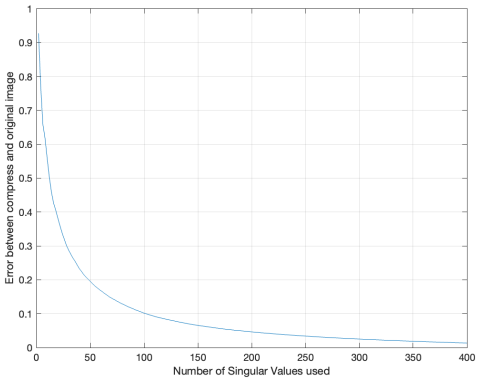

Figure 23: *Error vs. Singular Values for sample image 2*

7.1 FFT Performance Indicators

7.1.1 Image 1

Percent Removed	Percent Err	MSE	PSNR
99	75	0	Inf
75	50	0.00011677	402.76
50	25	0.015595	304.87
25	10	0.17311	256.73
10	5	0.57202	232.82
1	1	5.5208	187.48

7.1.2 Image 2

Percent Removed	Percent Err	MSE	PSNR
95	75	0	Inf
85	50	9.3712e-07	499.26
60	25	0.0031558	336.82
40	10	0.049726	281.67
1	1	5.5208	187.48

8 References

Heckbert, Paul. "Fourier Transforms and the Fast Fourier Transform (FFT) Algorithm." *Computer Graphics*, vol. 3, no. 2, pp. 15, Feb. 1995. https://www.cs.cmu.edu/afs/andrew/scs/cs/15-463/2001/pub/www/notes/fourier/fourier.pdf

Mathur, Mridul K. and Gunjan Mathur. "Image Compression Using DFT Through Fast Fourier Transform Technique." *International Journal of Emerging Trends & Technology in Computer Science*, vol. 1, no. 2, Summer 2012. ISSN 2278-6856.

Olver, Peter J., and Chehrzad Shakiban. Applied Linear Algebra. Prentice Hall, 2006.

Stefany Franco1, Dr. Tanvir Prince1, Ildefonso Salva2, and Charlie Windolf. "Mathematics Behind Image Compression"