MAT 167 Final Project

Summer Session I 2025

Instructor: Kelli Gutierrez

August 1, 2025

Lauren Le | Jasmine Saldano | Jood Alhenaki

Table of Contents

1. Introduction

In an increasingly data driven world, efficient data storage and transmission is crucial. Images can consume significant amounts of storage and in order to address that, compression techniques are implemented to reduce the file size while preserving as much of the necessary content as possible.

The goals of this project are:

- To understand and apply SVD for image approximation.
- To analyze the relationship between the number of singular vectors used and the resulting image quality.
- To define and assess compressibility across various images.

By the end of this project, we aim to gain a deeper understanding of matrix decomposition, image data structures, and the practical trade-offs in compression techniques.

2. Implementation Details

This project will involve developing three primary MATLAB functions to perform image compression. Each function will focus on a different aspect of the overall compression pipeline, from displaying the image, calculating compression metrics, to reconstructing compressed versions.

2.1. DisplayPicture()

This function takes an image matrix of size $m \times n \times 3$ with double entries and produces a figure displaying that picture. The first if statement checks the size of the image to ensure that the third dimension is 3. If the inputted matrix does not pass this condition, a message will be displayed warning that the dimensions of the image are incorrect. The nested if statement then inspects the numeric type of the entries. If the entries are not doubles, a warning message will be displayed. If the inputted matrix passes both conditions, a figure will then display the image. The mat2gray() function was used to normalize the values to be within 0 and l, and to scale the pixels, the values were multiplied by 255.

2.2. NumSingVecs()

This function takes an image matrix of size $m \times n \times 3$ and a percentage value X, which represents how much of the original image we would like to preserve after compression, denoted as the proportion of the image's total energy (information). To determine the number of singular values (k) needed to reconstruct X% of the image, the function first separates the image into its

RGB (Red, Green, Blue) color channels. Then, it performs singular value decomposition (SVD) on each channel individually, squaring the sum of the singular values.

$$A = U \Sigma V\text{^}T$$

This sum is also equal to the Frobenius norm squared, which represents the total energy of the matrix.

$$\text{Total Energy: } \sum_{i=1}^{r} \sigma_i^2$$

X can be found by dividing the cumulative sum of a certain amount of singular values over the total energy of the matrix.

$$\text{Find smallest k: } \frac{\sum_{i=1}^{k} \sigma_i^2}{\sum_{i=1}^{r} \sigma_i^2} \geq \frac{X}{100}$$

We used the find() function to find the smallest k's for each color that are needed to preserve at least X% of the image. Finally, we plotted progress curves to visualize how many singular values each color channel used.

2.3. CompressPicture()

This function takes an image matrix of size $m \times n \times 3$ and a percent X which represents how much of the original image we would like to retain after compression. For each RGB color channel, we apply SVD to get the singular values and vectors from the matrix. We then square the singular values and find their sum. We used the find() function to find the smallest k's needed to preserve at least X% of the image. Then we reconstructed each color channel using the top k singular vectors.

$$\text{Reconstruction: } A_k = U_k \Sigma_k V_k^T$$

2.4. ProportionNumVecs()

This function calculates the proportion of singular values used to achieve a specified percentage of preservation, and takes the image and amount of preservation that we want to achieve. The function first calls NumSingVecs() to get the total number of singular values per channel. After summing the number of singular values per channel, the function then calls NumSingVecs() again to retrieve the number of singular values needed to preserve the specified amount of the

image. Finally, the function divides the number of singular values needed to preserve the specified amount of the image by the total number of singular values and displays the calculated proportion.

2.5. RunAll()

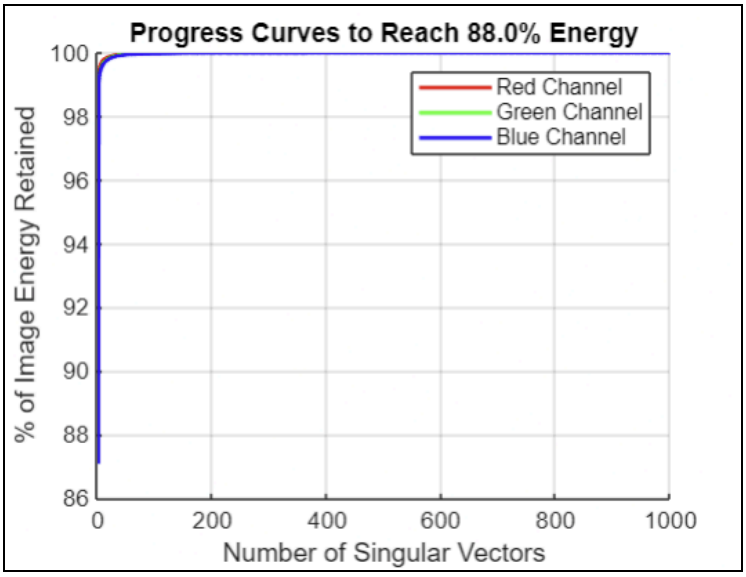
This function takes an image and executes all previously mentioned functions on said image. It first calls NumSingVecs() to determine how many singular values are needed for each color channel to preserve a specified amount of the image. It then displays those values in a progression plot and also displays the explicit amount for each RGB channel. The image is then compressed with CompressPicture(). ProportionNumVecs() is then called to determine what proportion of singular values is needed to preserve a specified amount of the original image.

3. Project Questions

3.1. Ranking Images by Compressibility

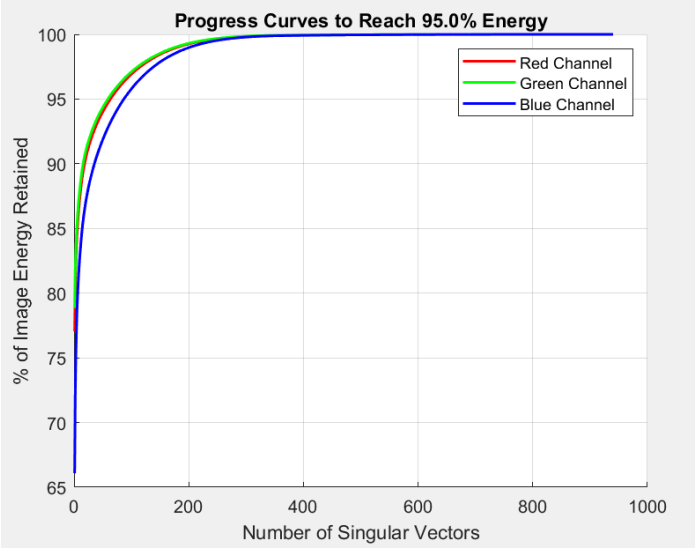
Of the six images, the most compressible was the chessboard. This image required the least amount of singular values to retain 88% of the original image compared to the other images. To preserve 88% of the image, it required only 0.13% of the singular values, which is a very minimal amount.

To retain 88% energy:	
Colors	Vectors needed
Red	1
Green	1
Blue	2



Of the six images, the least compressible was the picture of TV Snow. Any level of preservation less than 99% did not bear any resemblance to the original image. Additionally, the TV Snow needed 7.23% of the singular values to be visually recognizable. This is a much larger proportion of the singular values compared to the Chess board.

To retain 95% energy:	
Colors	Vectors needed
Red	64
Green	59
Blue	87



Compressibility Rank (MOST to LEAST compressible)
Chessboard
Black Hole
Cow
Lake Tahoe
UC Davis
TV Snow

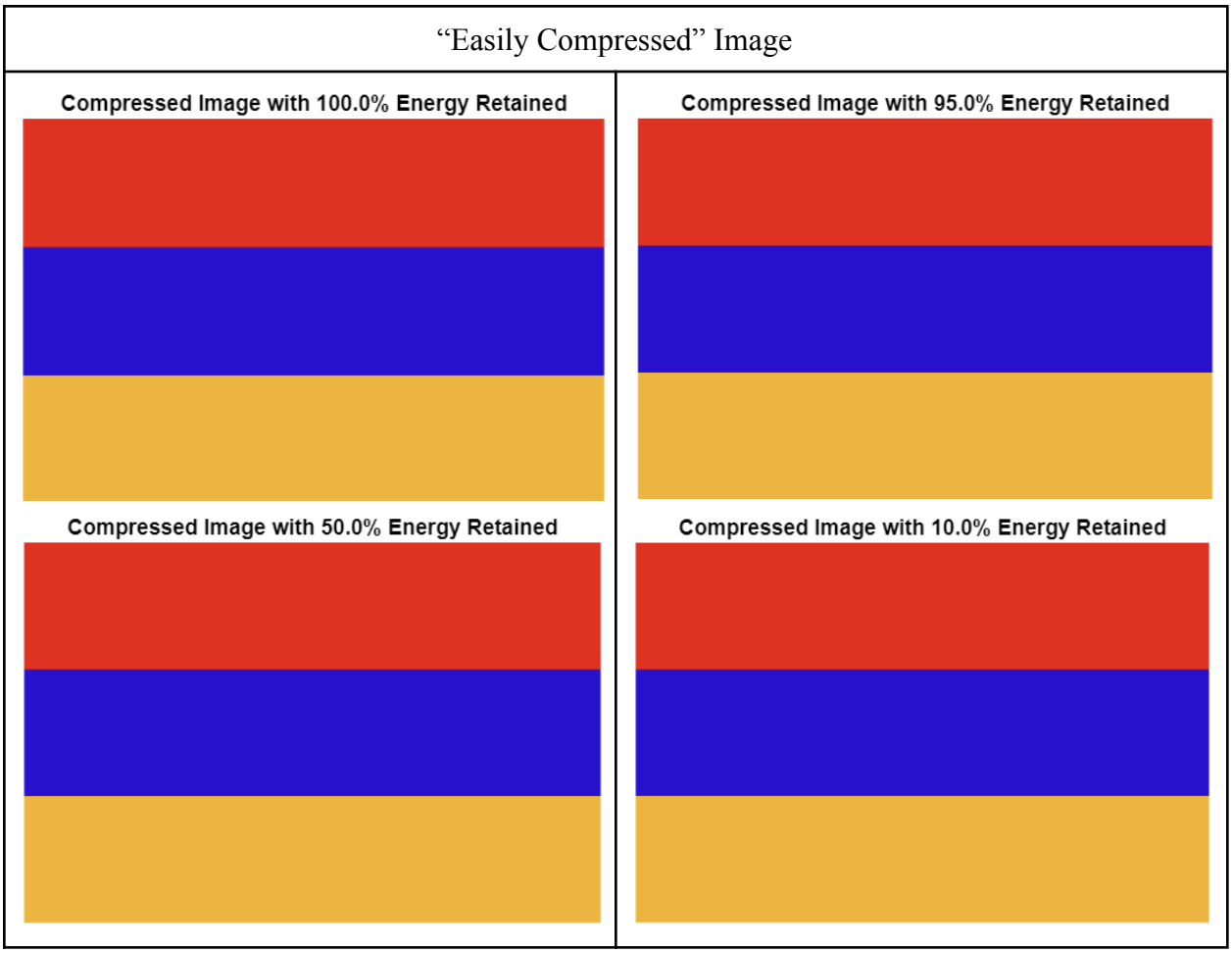
3.2. Criteria for Easy vs. Difficult to Compress Images

An easily compressible image requires a smaller amount of singular vectors to retain a larger proportion of the original image. To resemble the original image, an easily compressible image can preserve less of the image and still be recognizable. In comparison, a more difficult to compress image requires a larger amount of singular vectors to retain a larger proportion of the image. Additionally, an image is more difficult to compress if preserving a large amount of the image yields a result that bears minimal resemblance to the original.

To demonstrate this, we compressed an image of the Armenian flag. Visually, the flag has a uniform pattern and a limited amount of colors. The colors are also fully solid, with little variation in shade. The colors featured are also high in contrast with one another, making them easily distinguishable when the image is compressed to a smaller proportion of the original

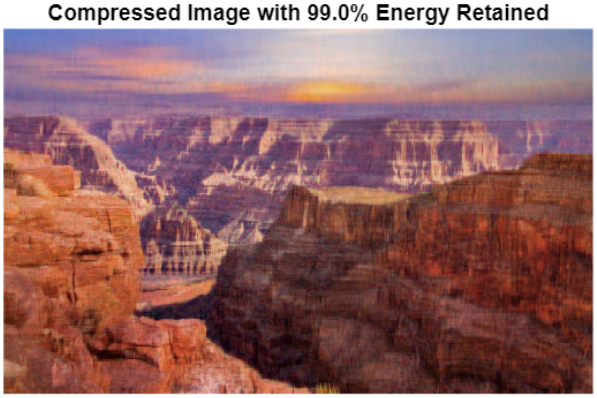
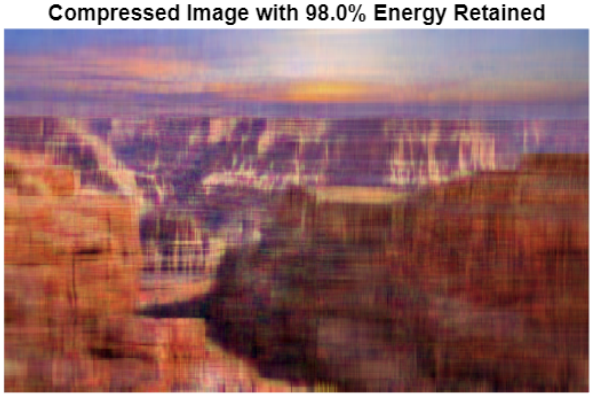
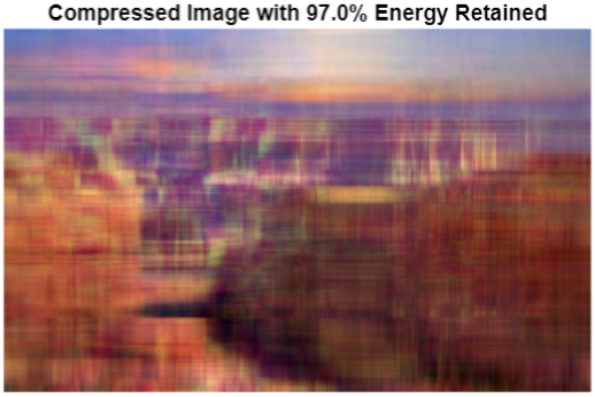
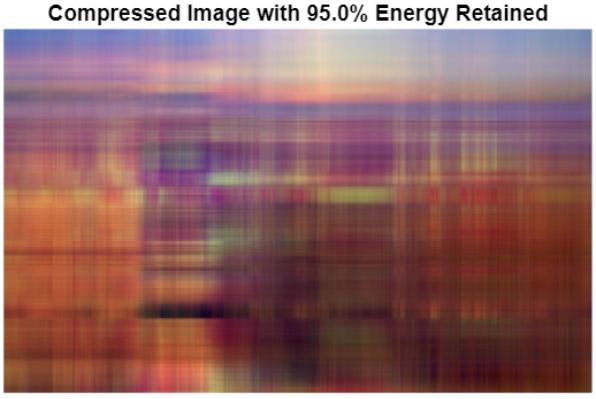
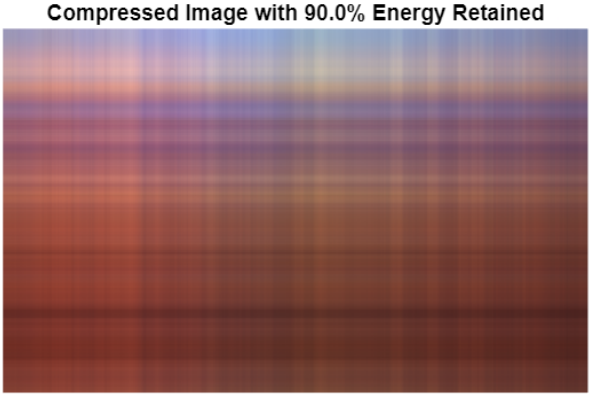
image. These visual components are similar to that of the image of the chess board, which was ranked the most compressible image of the six.

Compressing the flag to retain a 95% of the original image required only 1 vector for each rgb channel. Further compression also only required 1 vector for each rgb channel, as the compressed image continued to very closely resemble the original flag. At a proportion of 10%, the compressed image looks almost identical to the original flag.



Alternatively, to preserve a larger proportion of the original image, images that are more difficult to compress use a larger amount of singular values. Compared to the flag, an image of the Grand Canyon is more difficult to compress due to its low contrast visual components. The rock formations featured in the image are a variation of a single color and contain multiple shades, meaning it is more difficult to distinguish between various elements of the landscape the more we compress the image. These visual components differ from the Armenian flag's high contrast and solid colors.

As a result, to retain 95% of the original matrix, the image of the Grand Canyon requires 3 vectors in the red channel, 5 in the green channel, and 2 in the blue channel. Additionally, when preserving 95% of the flag, the compressed image still very closely resembles the original flag, whereas the Grand Canyon hardly resembles the original at all. The rock formations displayed in the original image are unrecognizable in images with a retention of 97% or less.

<table>
<tr><th colspan="2">“Compression Resistant” Image</th></tr>
<tr><td>Compressed Image with 100.0% Energy Retained
  </td><td>Compressed Image with 99.0% Energy Retained
  </td></tr>
<tr><td>Compressed Image with 98.0% Energy Retained
  </td><td>Compressed Image with 97.0% Energy Retained
  </td></tr>
<tr><td>Compressed Image with 95.0% Energy Retained
  </td><td>Compressed Image with 90.0% Energy Retained
  </td></tr>
</table>

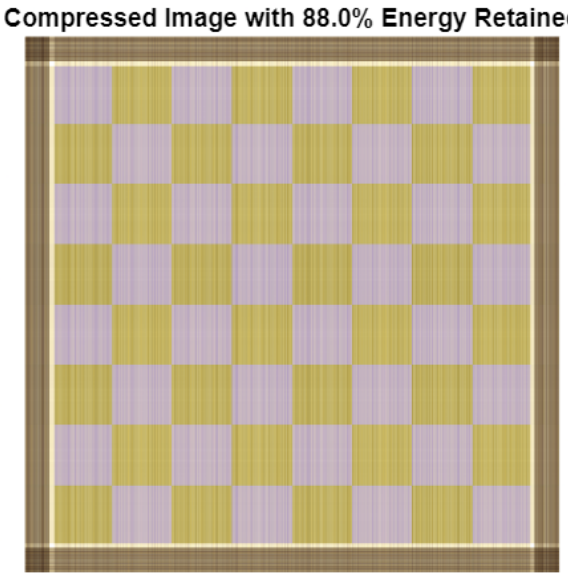
3.3. Average Proportion of Singular Values Needed to Resemble the Original Image

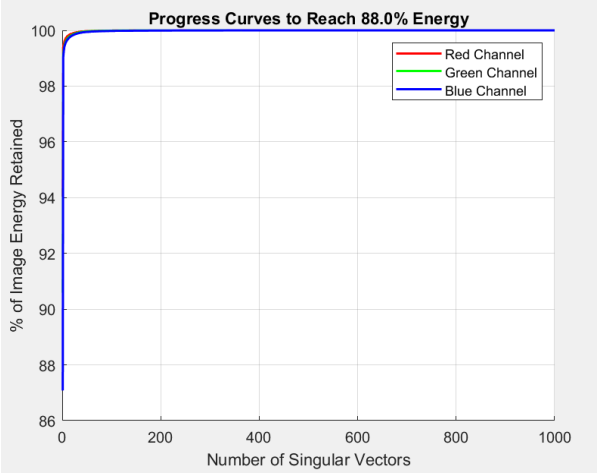
After comparing multiple degrees of compression, the chess board needed a minimum proportion of 88% which used 0.13% of the singular values. The black hole needed a proportion of 95% to be recognizable which utilized about 0.06% of the singular values, and the cow needed a proportion of 95% which required 0.31% of the singular values. Lake Tahoe is recognizable at a minimum proportion of 94% and needed 1.91%, the TV snow was recognizable at a proportion of 99% and needed 7.23% of its singular values, and lastly UC Davis needed 2.52% of its singular values to be recognizable at a minimum proportion of 90%. On average, the images required at least 2.0267% of their singular values to resemble its original. The overall average proportion needed to resemble the original image was approximately 93.5% which is a very large portion of the overall amount of singular values.

Image	Proportion that Resembles Original	% of Singular Values Needed
Chess	88%	0.13%
Black Hole	95%	0.06%
Cow	95%	0.31%
Lake Tahoe	94%	1.91%
UC Davis	90%	2.52%
TV Snow	99%	7.23%

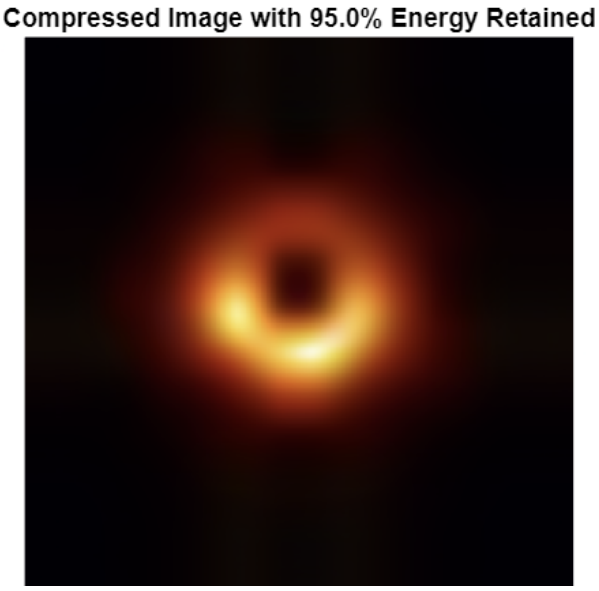
This demonstrates that the more difficult an image is to compress, the larger proportion of singular values is required for the compressed image to resemble the original. Lake Tahoe, TV Snow, and Davis Campus images were more difficult to compress and required the largest proportion of singular values compared to chess board, black hole, and cow which were more easy to compress and were ranked as so. The chess board, the easiest image to compress, only required 0.13% to resemble its original, and the black hole required the least proportion of singular values, 0.09261%.

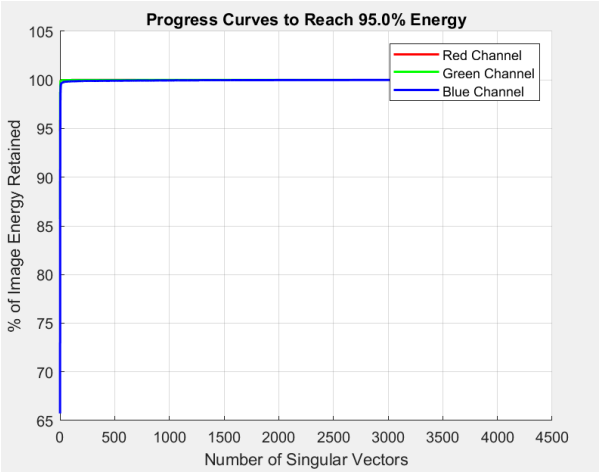
Minimum Proportion of Singular Values Needed to Resemble Original Image



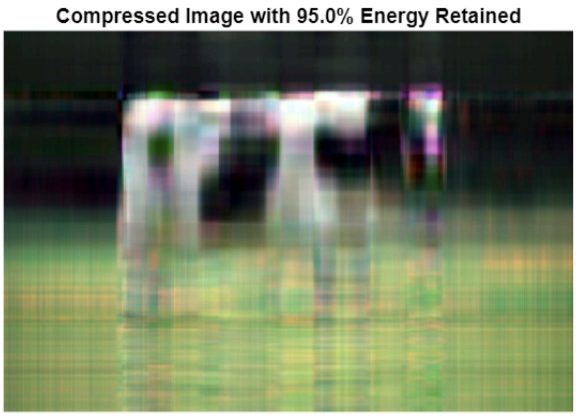


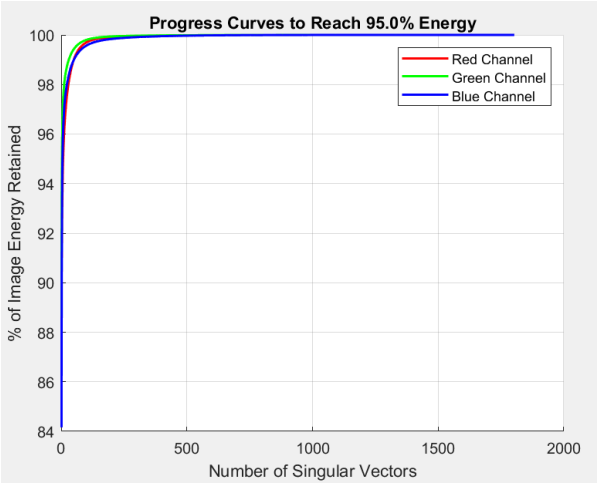
To retain 88% energy:
- Red: 1 vectors
- Green: 1 vectors
- Blue: 2 vectors



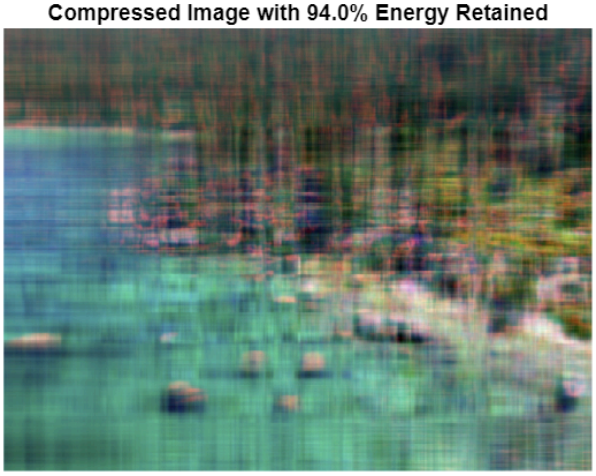


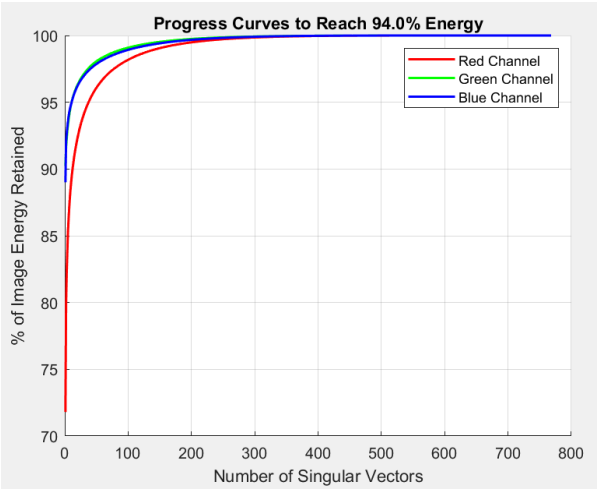
To retain 95% energy:
- Red: 2 vectors
- Green: 2 vectors
- Blue: 4 vectors



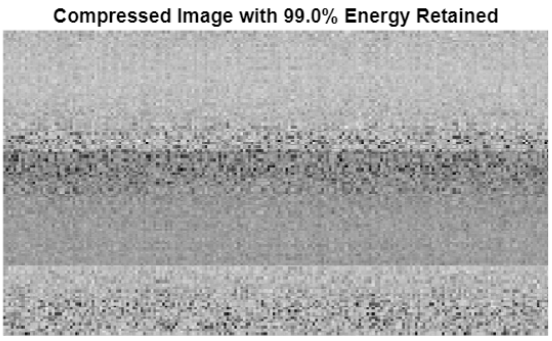


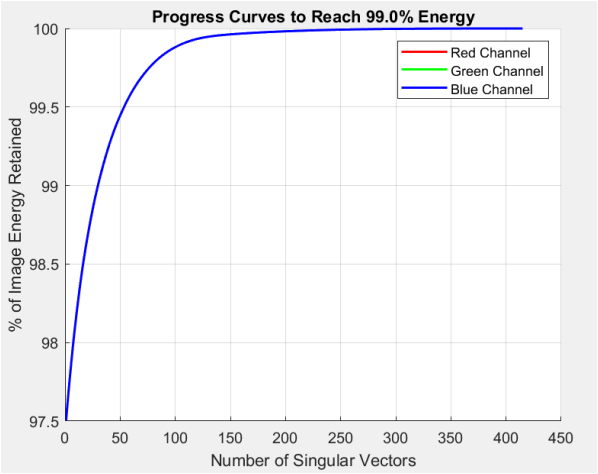
To retain 95% energy:
- Red: 8 vectors
- Green: 4 vectors
- Blue: 5 vectors



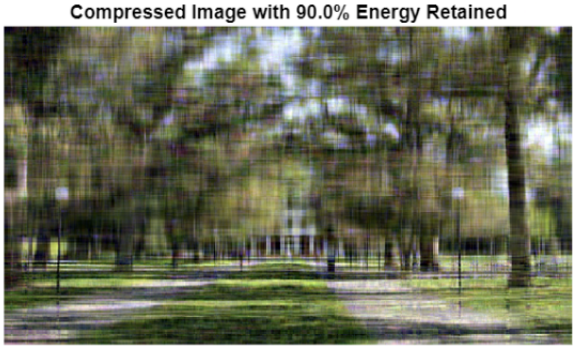


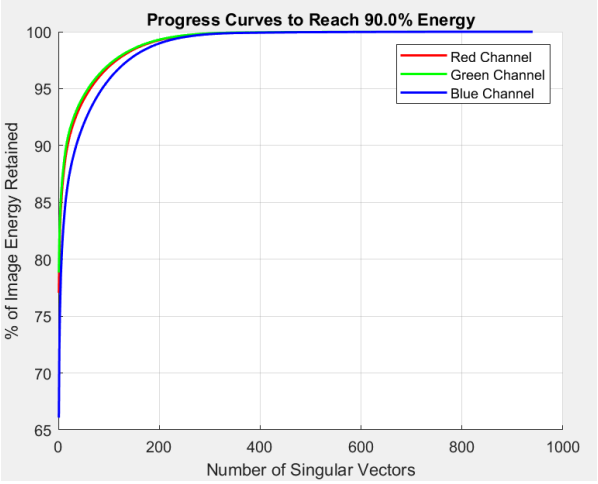
To retain 94% energy:
- Red: 30 vectors
- Green: 7 vectors
- Blue: 7 vectors





To retain 99% energy:
- Red: 30 vectors
- Green: 30 vectors
- Blue: 30 vectors





To retain 90% energy:
- Red: 19 vectors
- Green: 16 vectors
- Blue: 36 vectors

3.4. Strengths and Weaknesses of the Algorithm

Visually, the algorithm was able to capture patterns, like the checkered print that appeared in the image of the chess board and the stripes on the Armenian Flag. Uniform colors were also captured well, which were visual components that both the chess board and flag also possessed.

Noisier images that had multiple shades of a limited selection of colors were more difficult to compress, and these visual features were often lost. For example, the TV snow was in greyscale

making the dots more difficult to distinguish the more the image was compressed. The dots were also very fine, which is another feature that's more difficult for the algorithm to pick up. Similarly, the image of the Grand Canyon had lots of noise, colors that were not flat, and multiple shades of the same color, causing the image to bear little resemblance to the original even after 95% of the image was preserved.

To improve upon the algorithm and ensure that fine details and color variation is preserved, rather than applying SVD to each RGB channel, we can apply it to even smaller components. In our original algorithm, fine details contained in the smaller singular values were lost after compression. Preserving less and less of the original image meant that those singular values were not used to reconstruct the image, leading to loss of those details and shade variation. To minimize this we can further separate the image into uniform blocks and apply SVD to each block. Within each block, the fine details become more notable due to the smaller area that they are contained in. This method will help to better preserve textured features or features with non-solid colors that were previously overlooked due to their containment in smaller singular values in the original algorithm.

4. Conclusion

Our results demonstrate that the effectiveness of SVD-based image compression depends heavily on the image's characteristics. Those with simpler patterns, solid colors, and higher contrast variation are easier to compress, requiring fewer singular values to retain 95% of the original image in proportion to 100% preservation. The best examples of this can be seen with the chessboard, which has uniform patterns, and the Armenian flag, which only contains 3 high contrast colors to distinguish between. In contrast, images with complex textures, lower contrast gradients, or higher frequencies of noise (ie. TV snow) demand more singular values for similar preservation, making them harder to compress efficiently. The image of the Grand Canyon was selected as our "compression resistant" example due to its complex textures and low contrast landscape.

Some difficulties the algorithm encountered included unexpected color variations when we attempted to "compress" the image such that it preserves 100% of the original information. For example, when we executed the CompressPicture() function to retain 100% of the original energy on the Black Hole image, it reproduced a green black hole (Appendix B). This was resolved by adding another k value condition, such that if the X% value given was 100, then it returns the original number of Σ. After catching this issue with 100%, we added a third k condition for when X=0.

On average, the images required at least 2.03% of their singular values to resemble its original. The overall average amount of preservation needed to resemble the original image was approximately 93.5% which is a very large portion of the original image to be retained. In order

to calculate the minimum averages between each channel, we simply divided the amount of vectors needed for each color channel at the minimum X% to resemble the original image over the total number of vectors for X=100.

Altogether, the algorithm efficiently maintains the global structure of images between 0-100% of information, but it struggles to preserve finer details as seen with the TV snow image.

5. References

MathWorks. (n.d) *MATLAB documentation.* https://www.mathworks.com/help/matlab/

MathWorks. (n.d) *imshow.* https://www.mathworks.com/help/images/ref/imshow.html

MathWorks. (n.d) *cumsum.* https://www.mathworks.com/help/matlab/ref/double.cumsum.html

MathWorks. (n.d) *find.* https://www.mathworks.com/help/matlab/ref/find.html

MathWorks. (n.d) *double.* https://www.mathworks.com/help/matlab/ref/double.html

MathWorks. (n.d) *zeros.* https://www.mathworks.com/help/matlab/ref/zeros.html

Kelli Gutierrez. (2025). [*Lecture notes weeks 4–5]*. MAT 167 Applied Linear Algebra, UC Davis. July 26 2025

Kelli Gutierrez. (2025). [*SVD In-Class Demo] svd_approx_demo.m*. MAT 167 Applied Linear Algebra, UC Davis. July 25, 2025.

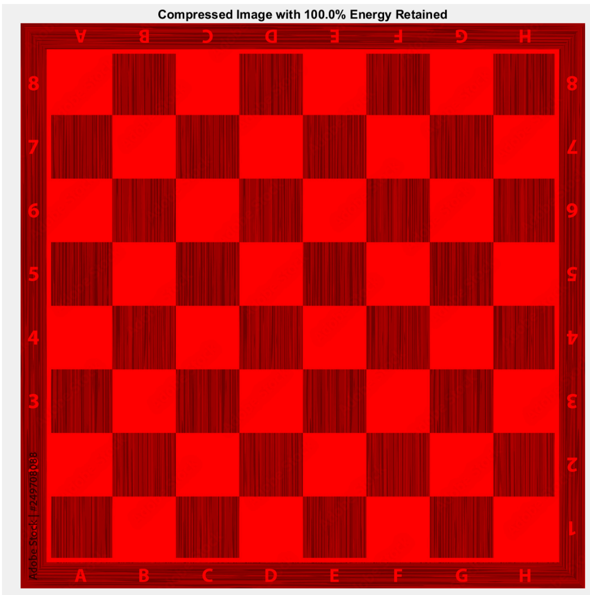
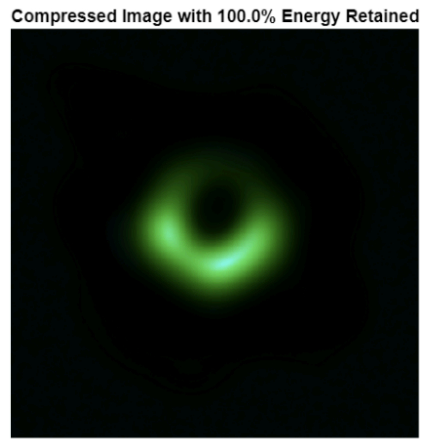
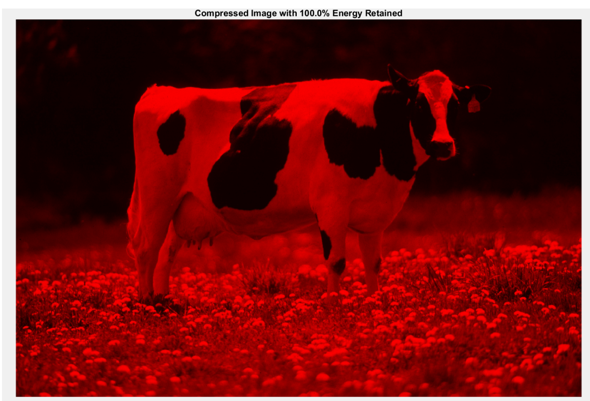
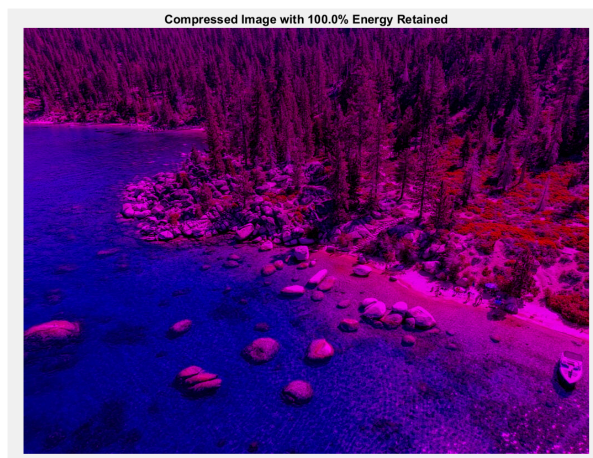
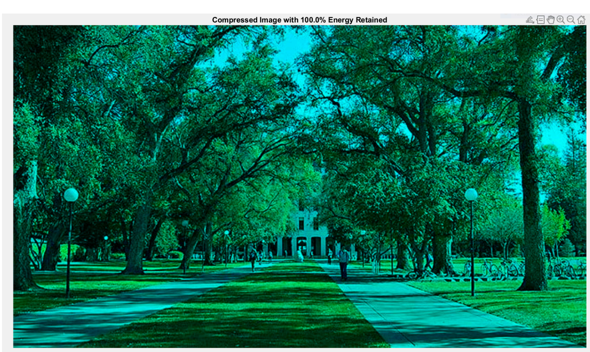
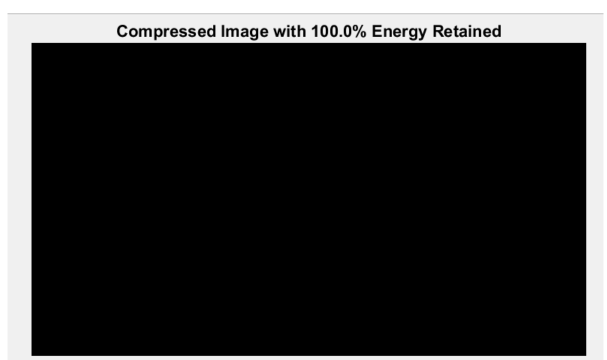
6. Appendices

Appendix A: Commands used

```
>> DisplayPicture(my_image)          % Displays image


>> img = imread('lake_tahoe.jpg');    % example: lake_tahoe.jpg | can replace with any image

>> [kRed, kGreen, kBlue] = NumSingVecs(img, 65, true);   % example: 65 | can use any number 0-100

>> fprintf('To retain 65%% energy:\n');         % fprintf commands for readability | match number above

>> fprintf('  Red:   %d vectors\n', kRed);

>> fprintf('  Green: %d vectors\n', kGreen);

>> fprintf('  Blue:  %d vectors\n', kBlue);


>> img = imread('cow.jpg');   % example: cow.jpg | can replace with any image

>> img_double = double(img);

>> compressed = CompressPicture(img_double, 90);          % example: 90 | can use any number 0-100
```

Alternatively, all functions were compiled into one file, **Final_Code.mlx**, to execute all functions in one place.

Appendix B: Challenges with X=100% for CompressPicture()

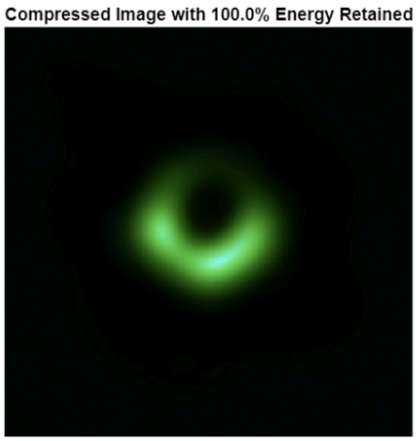
Solution:

```
if X == 0
   k = 0;  % Minimal info retained
elseif X == 100
   k = size(S, 1);  % Return original S
else
   k = find(cumulative_energy >= (X/100)*total_energy, 1);
end
```

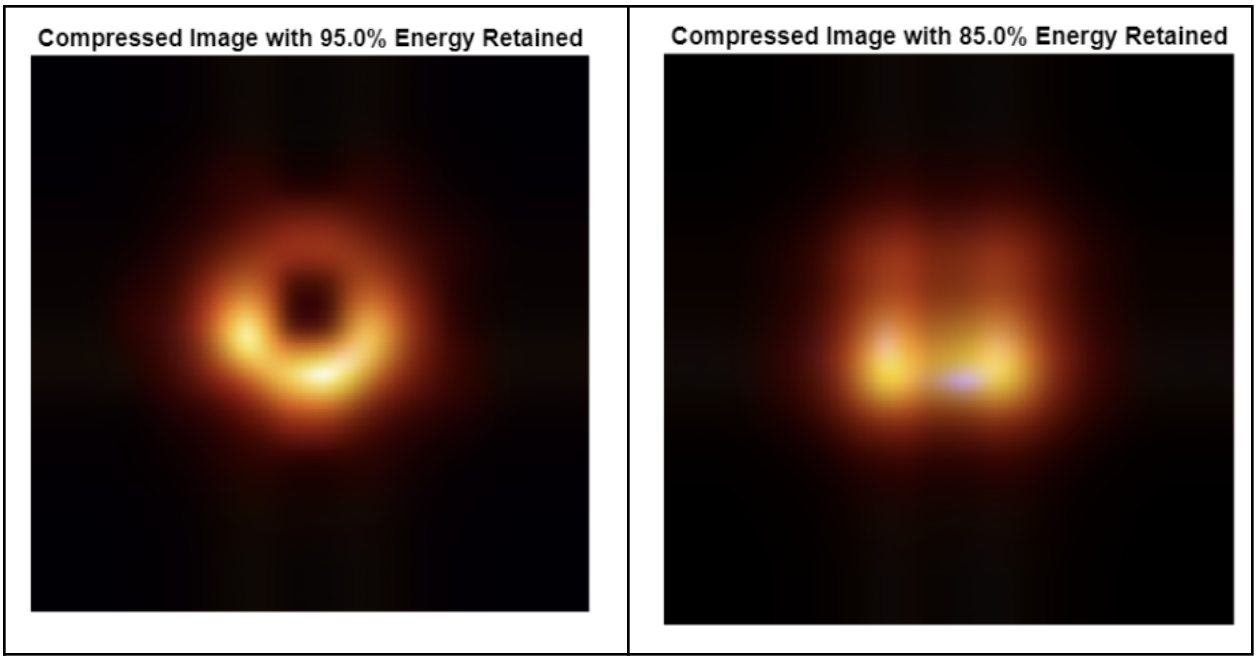
Some Figures

Black Hole Figures:

Minimum needed: 95



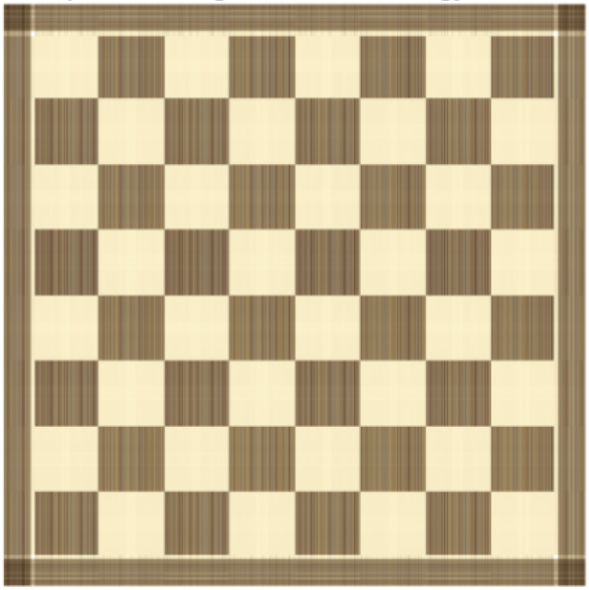
Compression before correcting algorithm vs after correction



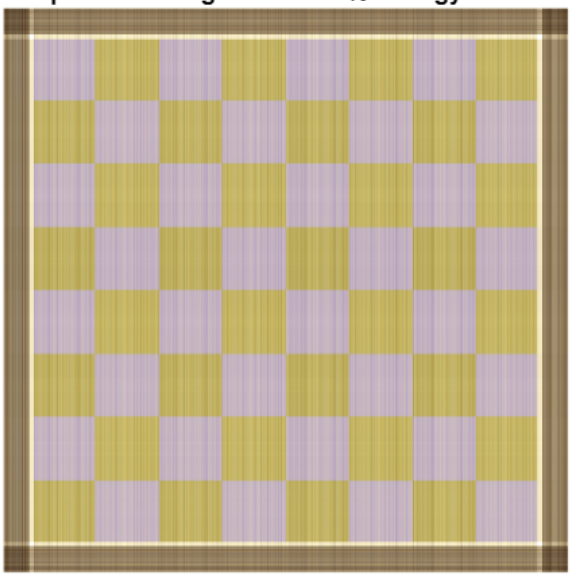
Chess:

Minimum needed: 90%

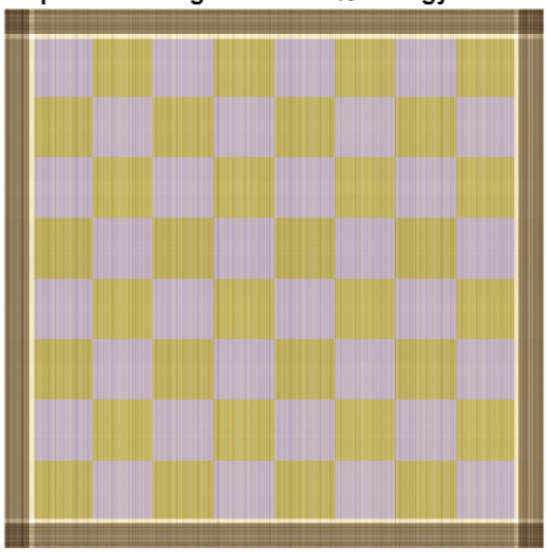
Compressed Image with 95.0% Energy Retained



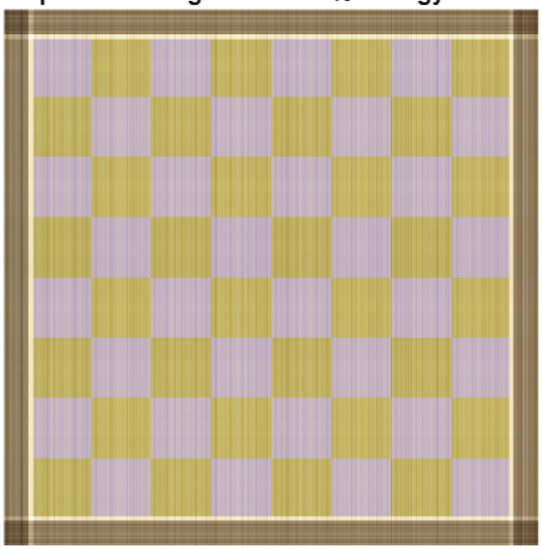
Compressed Image with 90.0% Energy Retained



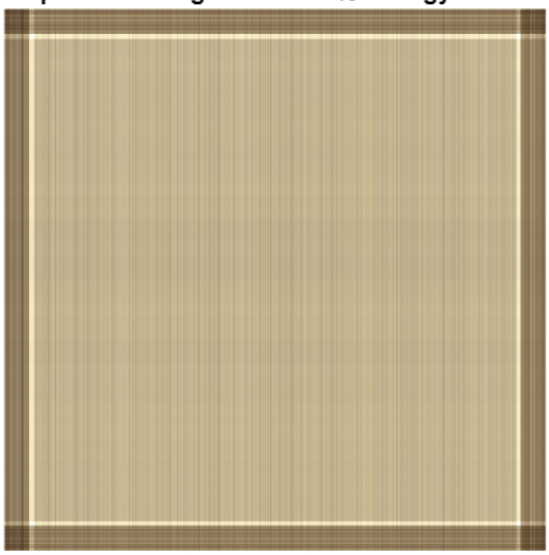
Compressed Image with 89.0% Energy Retained



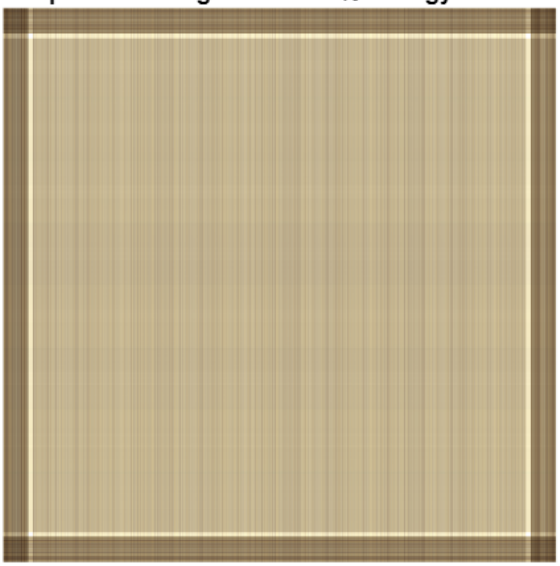
Compressed Image with 88.0% Energy Retained



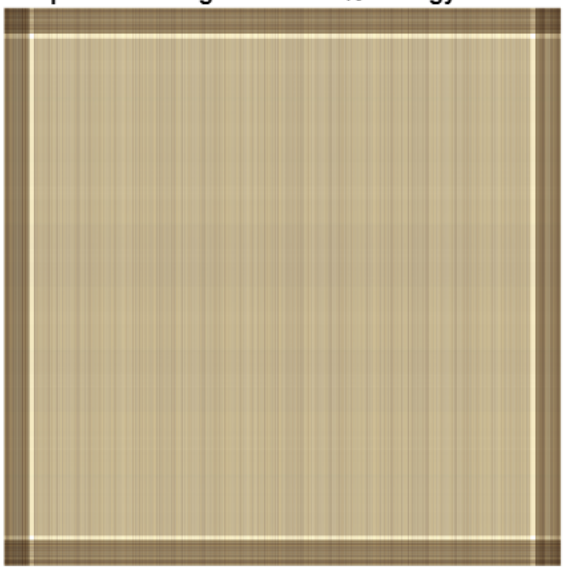
Compressed Image with 87.0% Energy Retained



Compressed Image with 85.0% Energy Retained



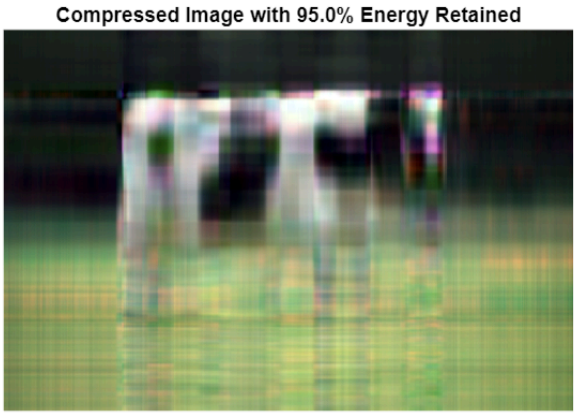
Compressed Image with 80.0% Energy Retained

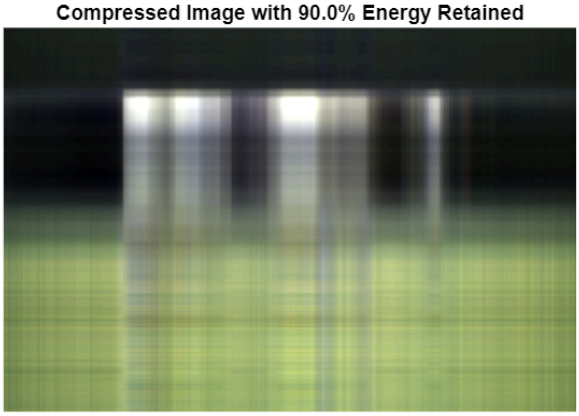


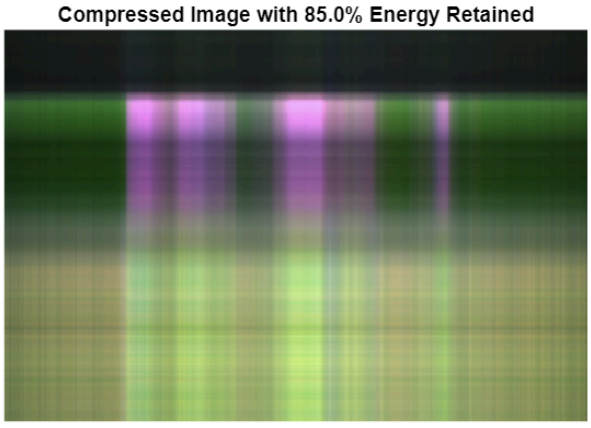
Cow:

Minimum: 90%-95%

```
To retain 95% energy:
  Red:   8 vectors
  Green: 4 vectors
  Blue:  5 vectors
```



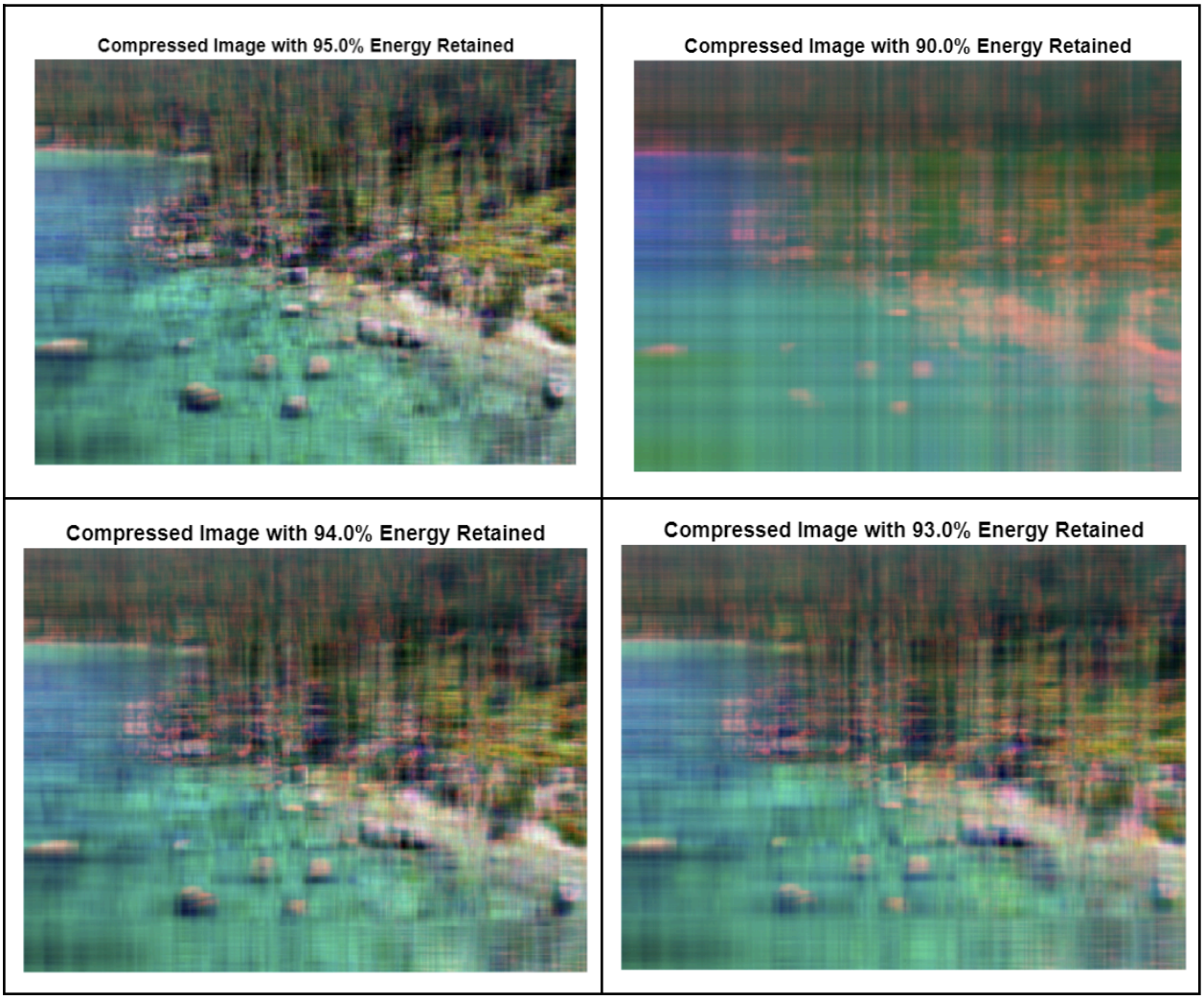




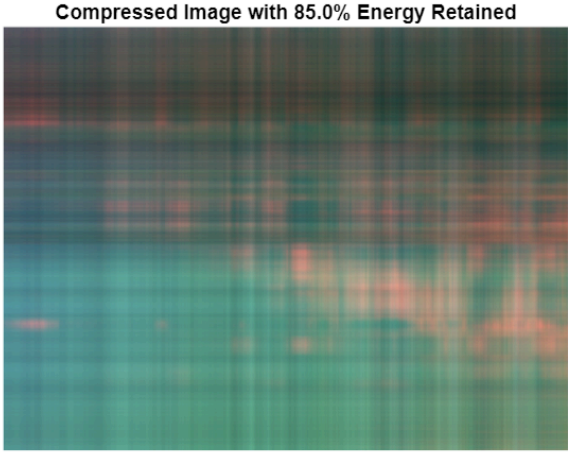
Lake Tahoe

Minimum: 95%

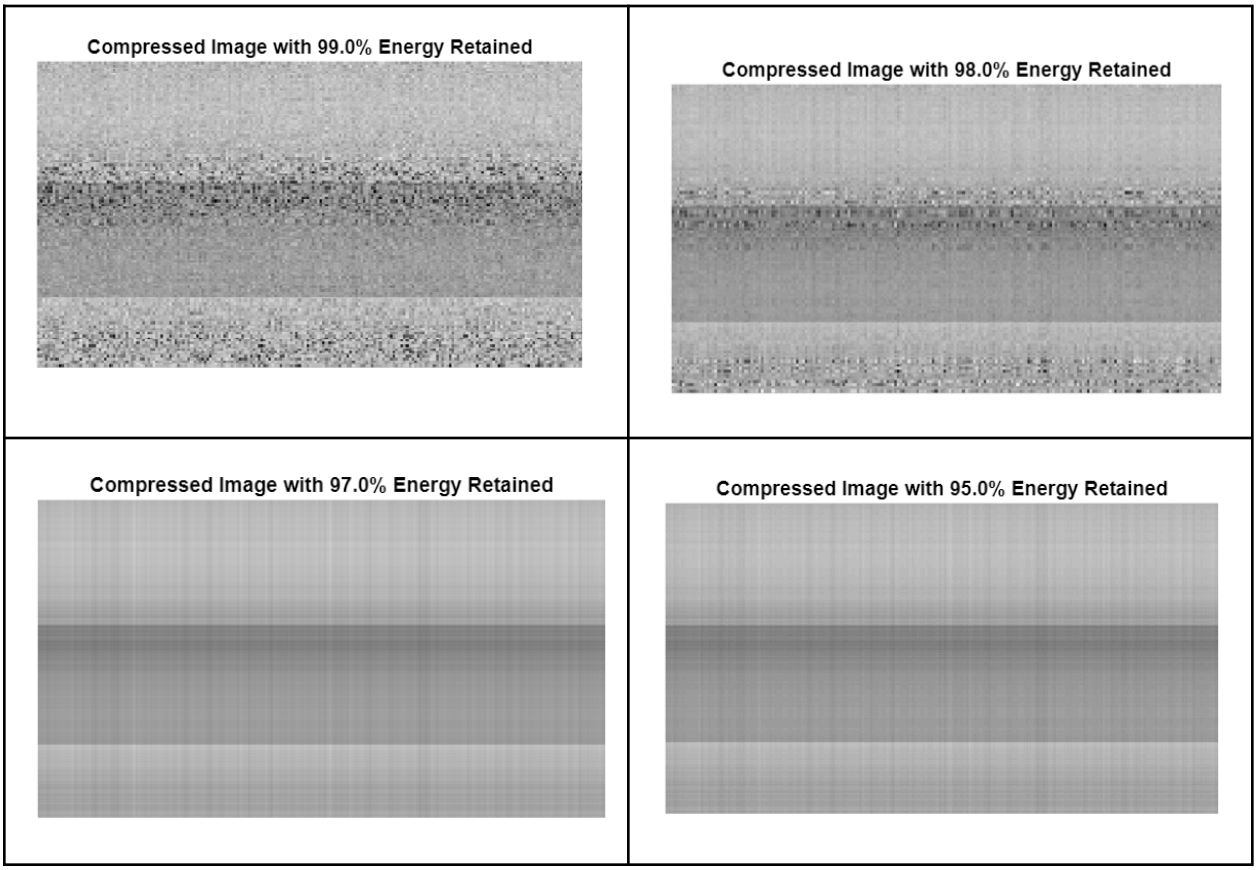
```
To retain 95% energy:
  Red:   38 vectors
  Green: 11 vectors
  Blue:  11 vectors
```



Compressed Image with 85.0% Energy Retained



TV Snow

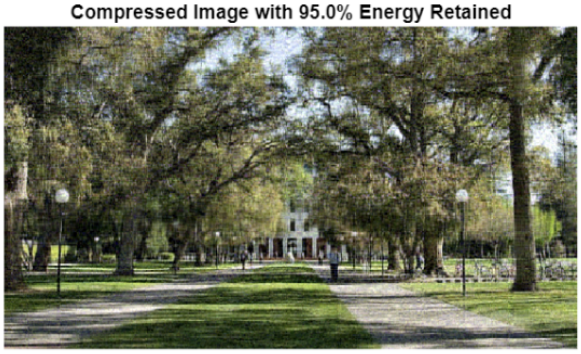

Compressed Image with 99.0% Energy Retained
Compressed Image with 98.0% Energy Retained
Compressed Image with 97.0% Energy Retained
Compressed Image with 95.0% Energy Retained

UC Davis

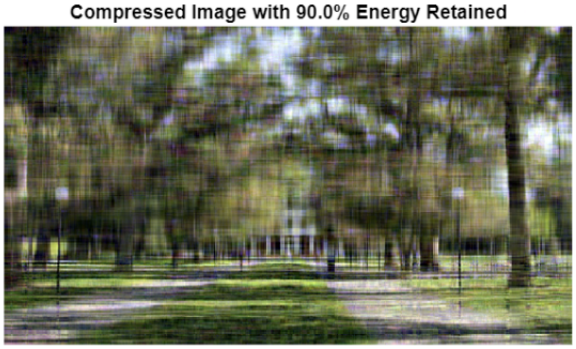
Minimum: 90%

```
To retain 95% energy:
  Red:   64 vectors
  Green: 59 vectors
  Blue:  87 vectors
```

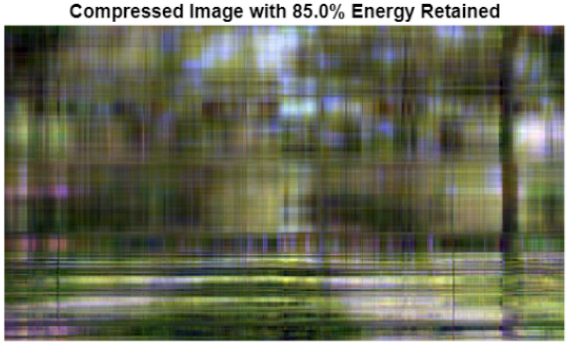
Compressed Image with 95.0% Energy Retained



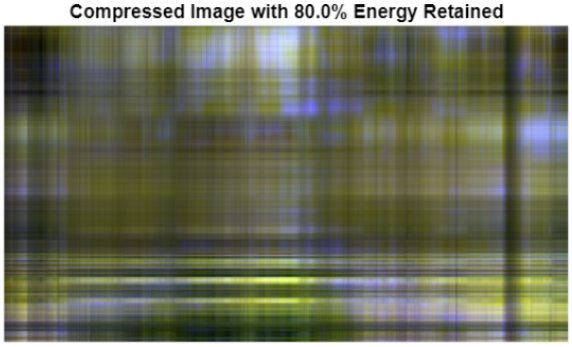
Compressed Image with 90.0% Energy Retained



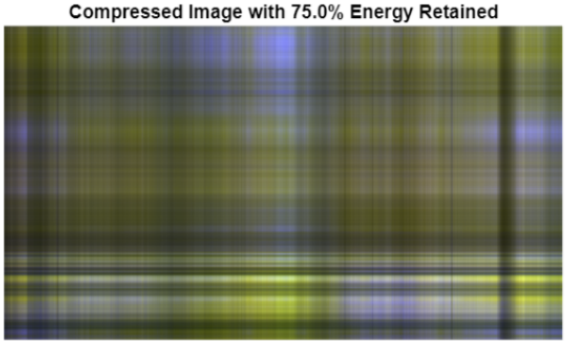
Compressed Image with 85.0% Energy Retained



Compressed Image with 80.0% Energy Retained

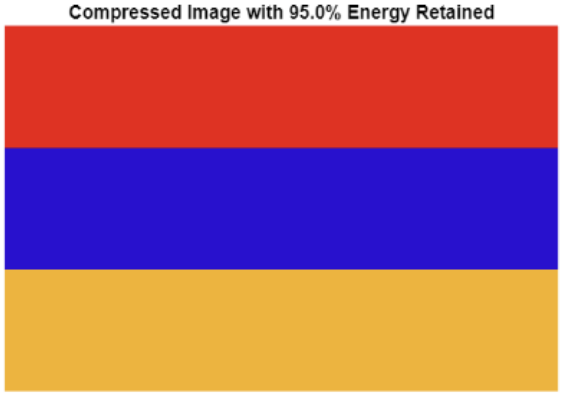


Compressed Image with 75.0% Energy Retained



Armenian Flag

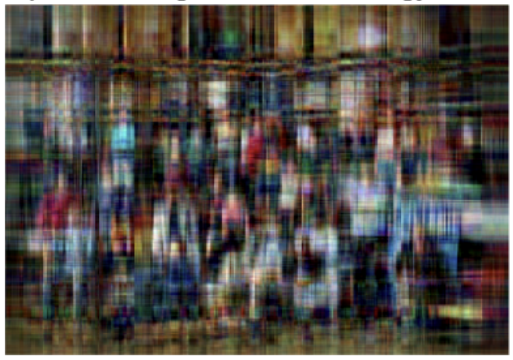
Minimum:

Compressed Image with 95.0% Energy Retained 	Compressed Image with 10.0% Energy Retained 

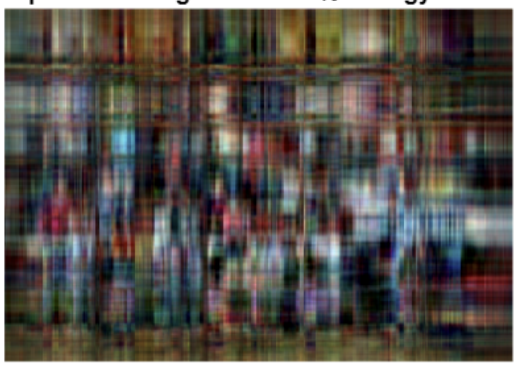
Group/Tree

Compressed Image with 95.0% Energy Retained 	Compressed Image with 90.0% Energy Retained 

Compressed Image with 80.0% Energy Retained

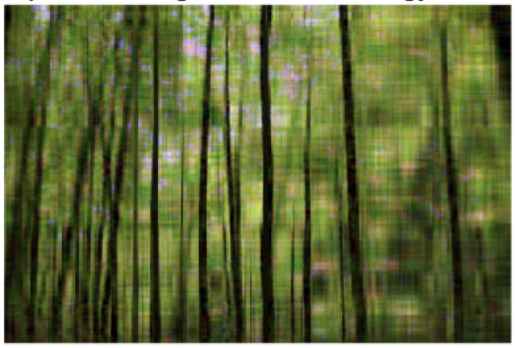


Compressed Image with 75.0% Energy Retained

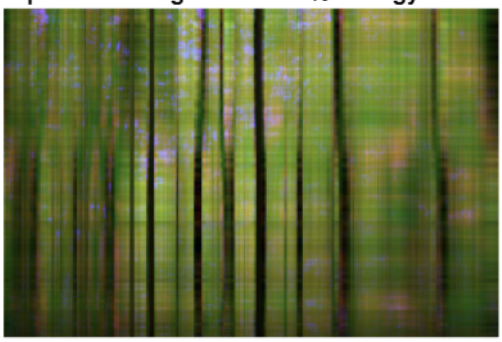


Compressed Image with 95.0% Energy Retained

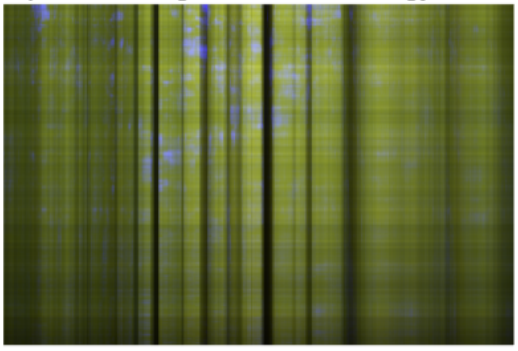
Compressed Image with 90.0% Energy Retained



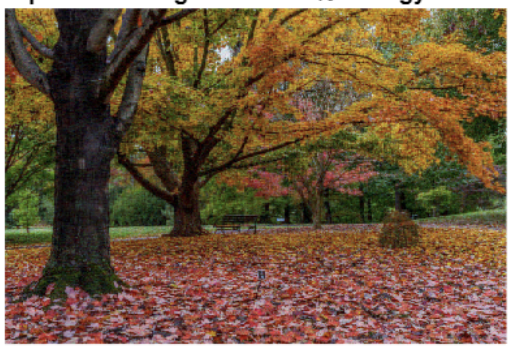
Compressed Image with 85.0% Energy Retained



Compressed Image with 75.0% Energy Retained

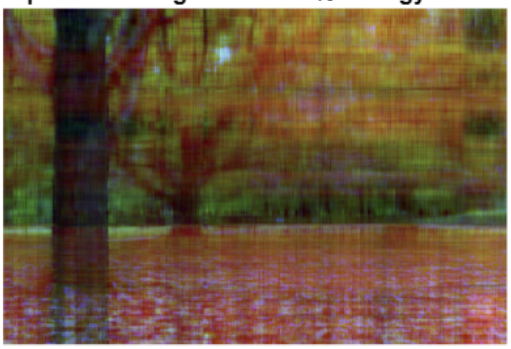


Compressed Image with 95.0% Energy Retained



Compressed Image with 90.0% Energy Retained

Compressed Image with 85.0% Energy Retained



Compressed Image with 80.0% Energy Retained