Image Compression Using Singular Value Decomposition

Kevin Johansmeyer

Fall 2021

1 Instructions

Written by Dr. Eric Forgoston for AMAT-532 Applied Linear Algebra:

1. Choose a black & white digital photo (either taken in black & white or converted to black white from a colour photo). The photo could be one that you have taken, or it could be a non-copyrighted image that you download from the web. Check the size of the original image (i.e. it is 10 MB). Import the image into Python or R and perform a singular value decomposition. Create a plot of the singular values and using that information make a hypothesis about how many modes you will need to reconstruct the image well. Using 5, 10, 20, and 50 modes reconstruct the image and discuss how good the approximation is with context to the singular value plot and your hypothesis. Lastly, reconstruct the image using as many modes as you need so that you cannot tell a difference between the original image and the reconstructed image. What is the size of the reconstructed image and what compression factor were you able to achieve.

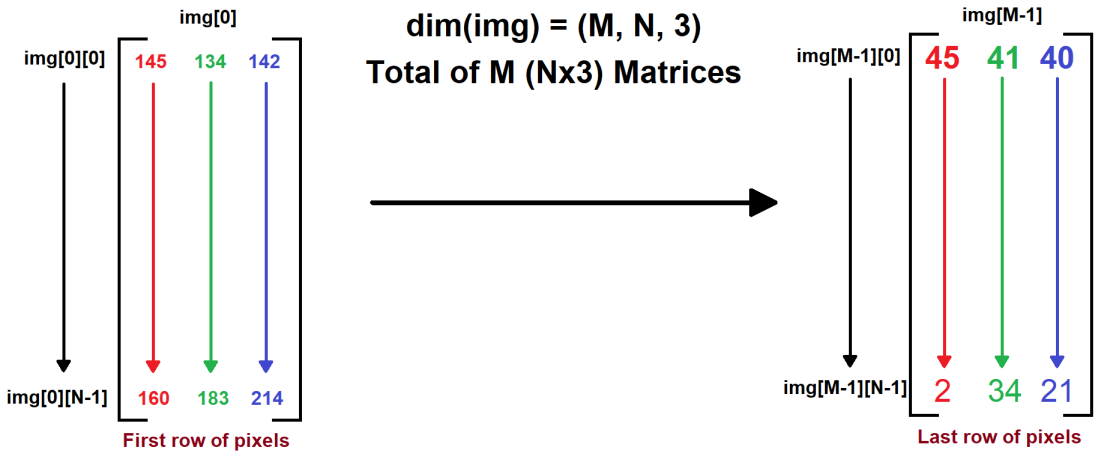
2. Do the above for a different photograph in colour.

Continue to the next page

2 Grayscale Image Compression

2.1 Theory

When importing an image into Python using Matplotlib, the dimensions are $(M, N, 3)$, where M is the height (in pixels), N is the width (in pixels), and 3 representing the three RGB values that correspond with that pixel.



Grayscale images also had RGB values. I was able to convert the three RGB values for each pixel into one value that represents it's grayscale value. These values can be 0, 255, and anywhere in between. The higher the number, the lighter (closer to white), the shade is.

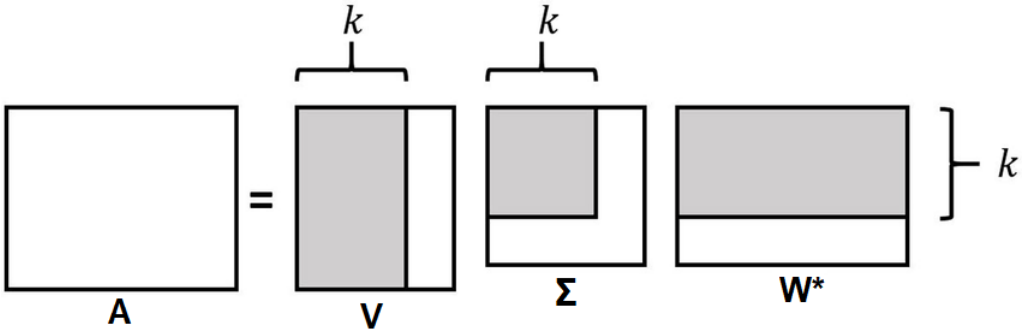
```
29  for m in range (0,M):
30      for n in range (0,N):
31          # RBG weights from NTSC: 0.2989 · Red + 0.5870 · Green + 0.1140 · Blue
32          imgBW[m][n] = (0.2989)*img[m][n][0] + 1/0.5870*img[m][n][1] + 1/0.1140*img[m][n][2]
```

This part of my code (entire code shown later), uses the weights provided by the National Television Standards Committee (NTSC) to convert RGB values into a single gray-scale value. This converts the $M \times N \times 3$ matrix into an $M \times N$.

The $M \times N$ matrix can be decomposed using Singular Value Decomposition (SVD). Singular Value Decomposition (SVD) transforms an $M \times N$ matrix into the product of three particular matrices. U is a complex, unitary matrix $(M \times M)$. Σ is a diagonal matrix with positive, singular values in decreasing order $(M \times N)$, and W^* is the adjoint of another complex, unitary matrix $(N \times N)$.

It is possible to truncate each matrix in the decomposition by a specified number of modes (k) to reduce the amount of information held in the matrix.

Image from Davide Chicco (click to view on ResearchGate):
(Modified to match notation in code)



The previous figure shows removing the white areas from V, Σ, and W^*. Notice that the new dimensions of each matrix are as follows:

$$V : M \times M \to M \times k$$

$$\Sigma : M \times N \to k \times k$$

$$W^* : N \times N \to k \times N$$

Multiplying the truncated matrices still results in an $M \times N$ matrix, so the original size of A is restored. The purpose of this process is to compress file sizes. An image is turned into matrix A, converted into V, Σ, and W^*, truncated to a specified number of modes (k), multiplied to find a new, reduced form of A, then converted into a new image. Throwing out singular values results in a smaller file size, but loss of image quality.

General procedure:
1. Convert image into a matrix.
2. Decompose the matrix into V, Σ, and W^* using singular value decomposition (SVD).
3. Truncate each matrix to the desired number modes/singular values.
4. Matrix multiply V, Σ, and W^* to obtain a compressed version of the original matrix.
5. Convert the compressed matrix into an image.

Question: Create a plot of the singular values and using that information make a hypothesis about how many modes you will need to reconstruct the image well.

Since singular values are in descending order, the beginning values hold the most information. Conversely, the ending values are smaller and hold less information. I hypothesize that the last 20% of singular values can safely be truncated without noticing a significant difference in image quality.

$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_q \text{ where } q = min\{m, n\}$$

The remainder of this paper will explore my code used to compress the images, and original images truncated to 5, 10, 20, and 50 modes.

2.2 Grayscale Images

Original Before Converted Into Matrix (JPG), file size = 784 KB

Image converted from JPG to PNG using the following code:

```
from matplotlib import pyplot as plt
# Converts JPG into PNG (this will increase the file size)
img = plt.imread('INSERT FILE NAME HERE.JPG')
plt.imsave('Uncompressed Image.png', img)
```

Original BW Image (PNG), file size = 2.7 MB
Dimensions: 1536 px × 2048 px

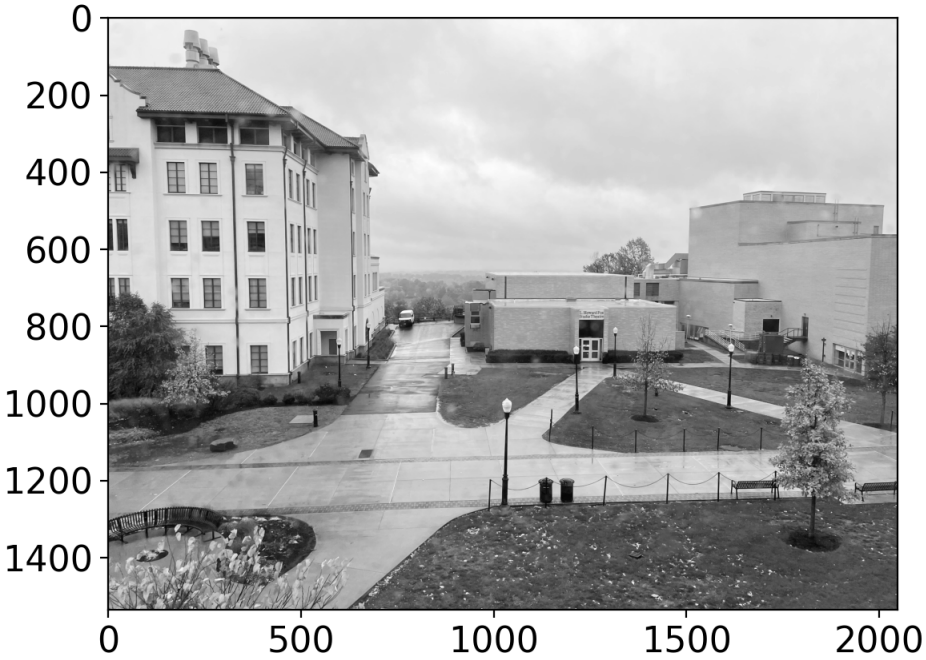


Image with 5 modes (PNG), file size = 1.2 MB
Dimensions: 1536 px × 2048 px

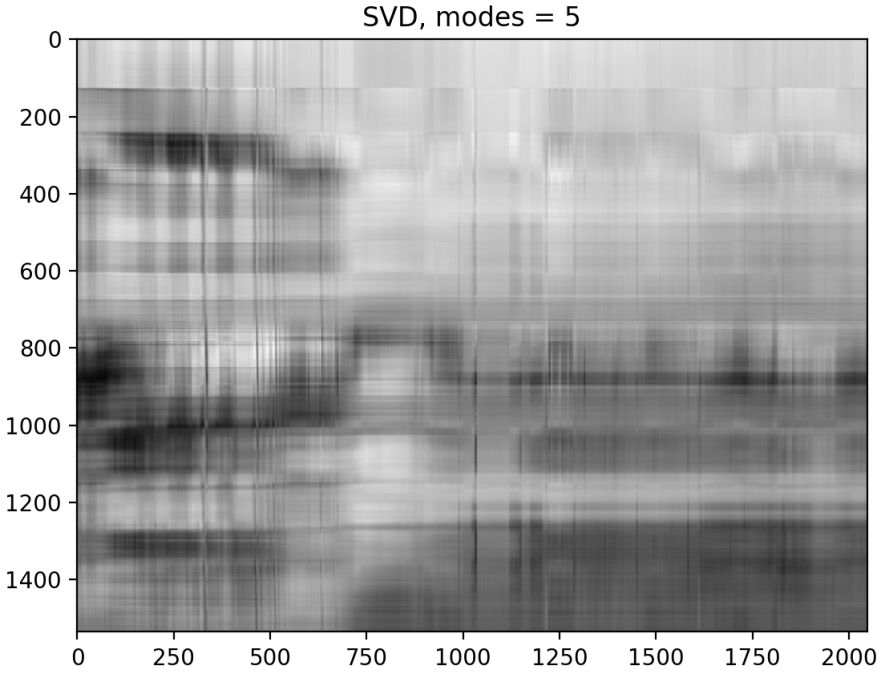


Image with 10 modes (PNG), file size = 1.4 MB
Dimensions: 1536 px × 2048 px

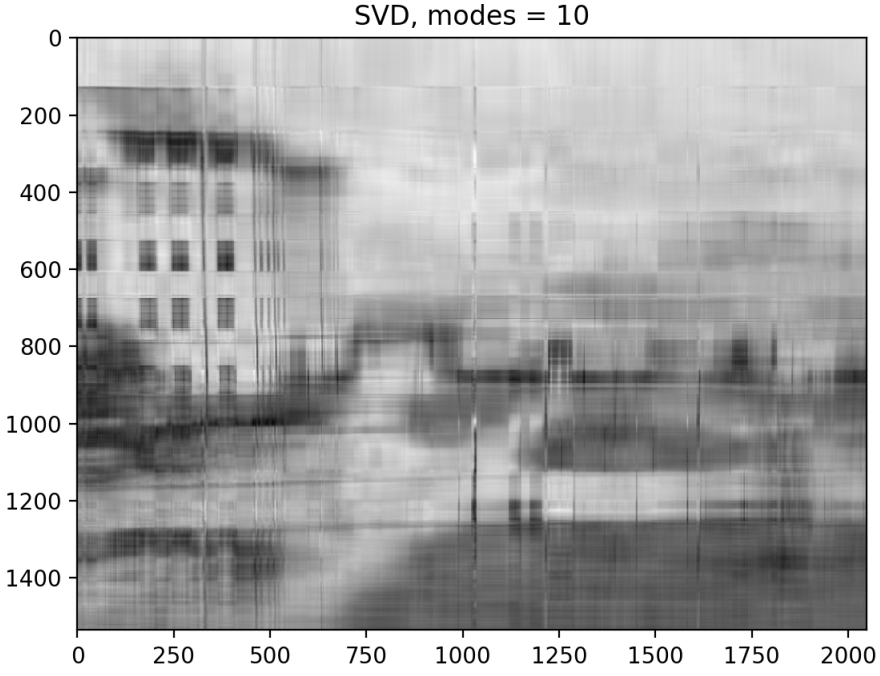


Image with 20 modes (PNG), file size = 1.6 MB
Dimensions: 1536 px × 2048 px

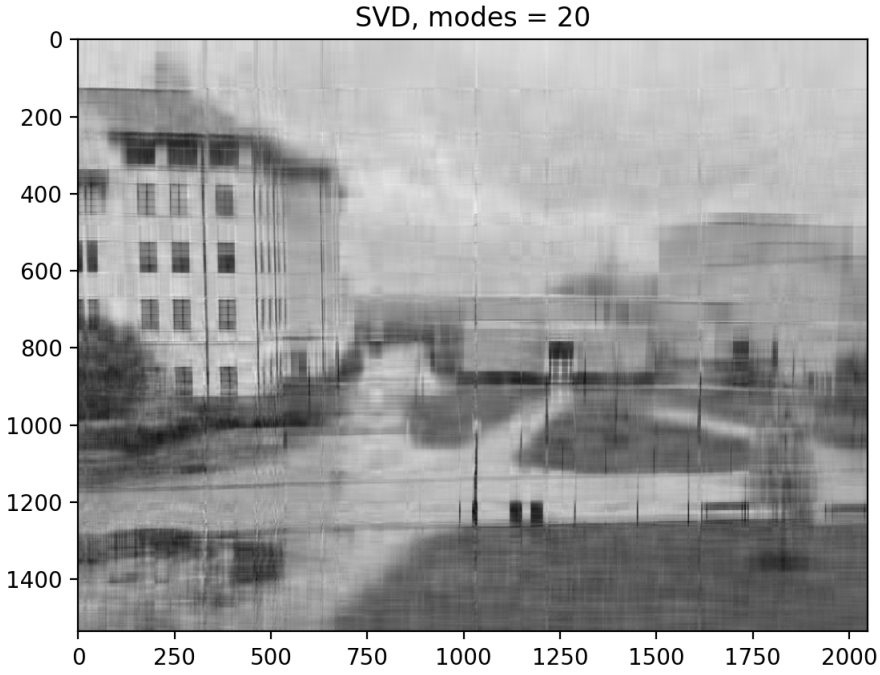
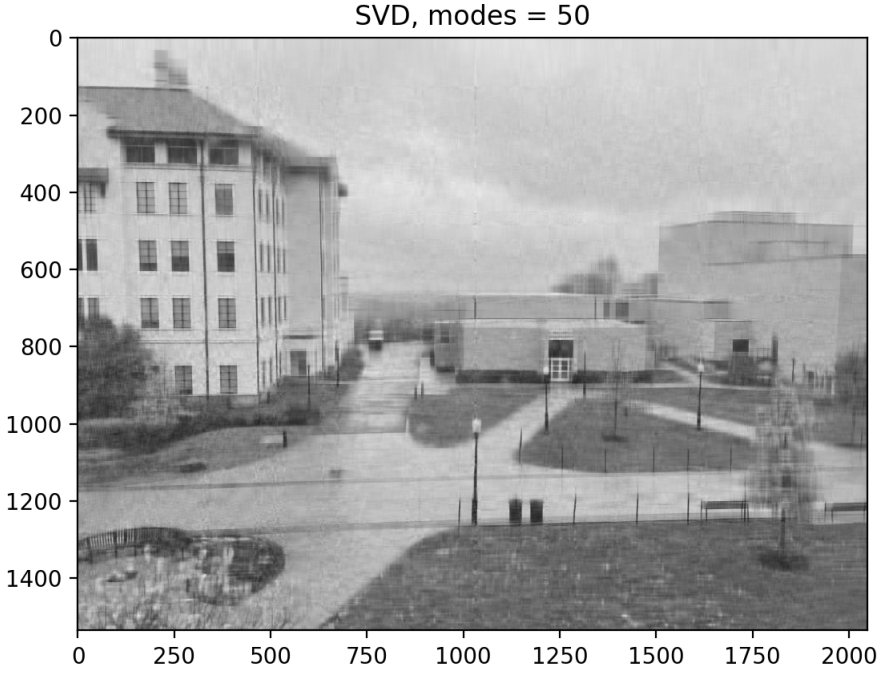


Image with 50 modes (PNG), file size = 1.9 MB
Dimensions: 1536 px × 2048 px



3 Color Image Compression

While the compression of color images is similar in theory to the previous procedure, there are some key differences. When importing an image into Python using Matplotlib, the dimensions remain $(M, N, 3)$. Each color channel (red, green, blue) had to be separated and place into its own $M \times N$ matrix. Singular value decomposition was computed for each channel individually, each matrix was truncated to the same number of modes, then the three channels were combined into a new $(M, N, 3)$ matrix. Below is code to show how the red channel [0] was extracted from the original matrix, decomposed using singular value decomposition, truncated, then recombined to produce a compressed red channel matrix.

```
23    # ============================== Red Channel ============================== #
24    imgRed = np.zeros((M,N))
25
26    for m in range (0,M):
27        for n in range (0,N):
28            imgRed[m][n] = img[m][n][0]
29
30    redV, redSigmaVals, redWAdjoint = np.linalg.svd(imgRed, full_matrices=True)
31
32    redTruncV = redV[:,:modes]
33    print(redTruncV.shape)
34    print(redTruncV.shape)
35    redTruncSigmaVals = redSigmaVals[:modes]
36    redTruncSigma = np.diag(redTruncSigmaVals)
37    redTruncWAdjoint = redWAdjoint[:modes]
38
39    redTruncImg = redTruncV @ redTruncSigma @ redTruncWAdjoint
40    print(redTruncImg.shape) # Recovers M x N dimensions
```

The code for the green [1] and blue [2] channels are nearly identical (see appendix for full code). The channels are then recombined to produce the compressed image.

General procedure:
1. Convert image into a matrix.
2. Separate the red, green, and blue channels into their own arrays.
3. Decompose the three matrices into V, Σ, and W^* for each color using singular value decomposition (SVD).
4. Truncate all nine matrices to the desired number modes/singular values.
5. Matrix multiply V, Σ, and W^* for each channel to obtain compressed versions of the original channels.
6. Combine the three channels into a new $(M, N, 3)$ matrix.
5. Convert the compressed matrix into an image.

3.1 Images

Original Before Converted Into Matrix (JPG), file size = 2.6 MB

Image converted from JPG to PNG using the following code:

```
from matplotlib import pyplot as plt
# Converts JPG into PNG (this will increase the file size)
img = plt.imread('INSERT FILE NAME HERE.JPG')
plt.imsave('Uncompressed Image.png', img)
```

Original Color Image (PNG), file size = 10.7 MB
Dimensions: 3024 px × 4032 px

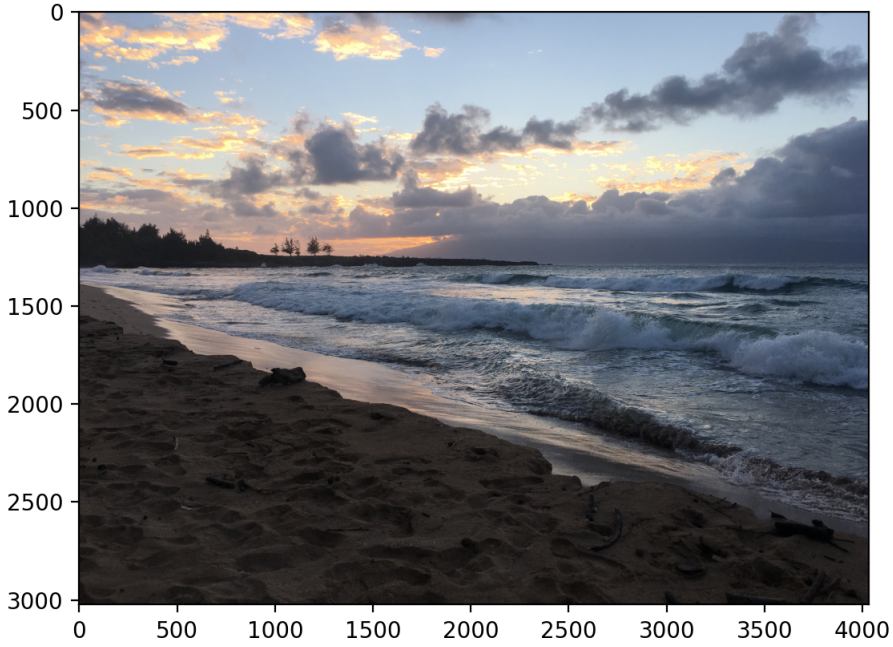


Image with 5 modes (PNG), file size = 5.9 MB
Dimensions: 3024 px × 4032 px

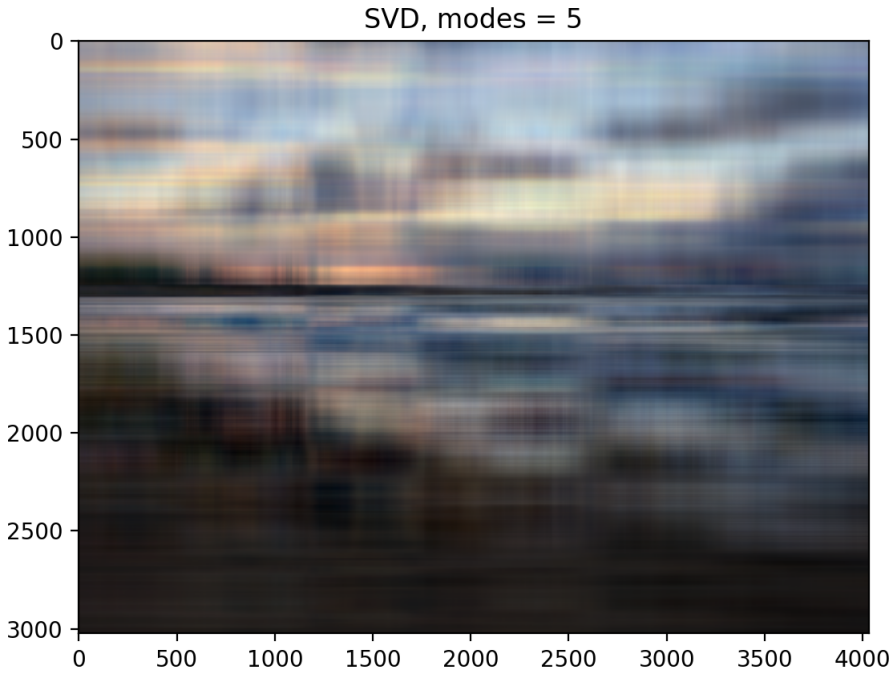


Image with 10 modes (PNG), file size = 7.1 MB
Dimensions: 3024 px × 4032 px

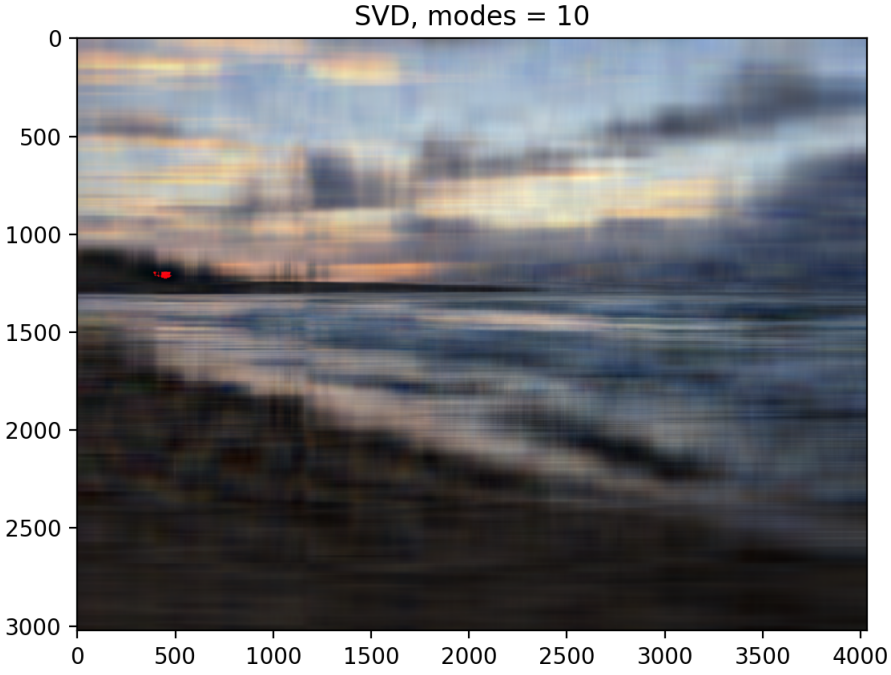


Image with 20 modes (PNG), file size = 8.7 MB
Dimensions: 3024 px × 4032 px

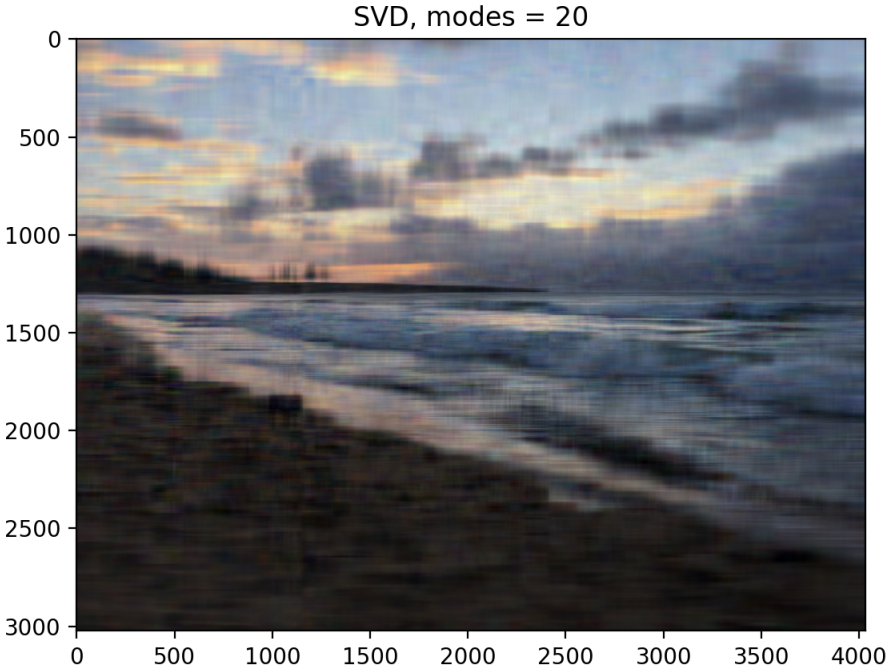
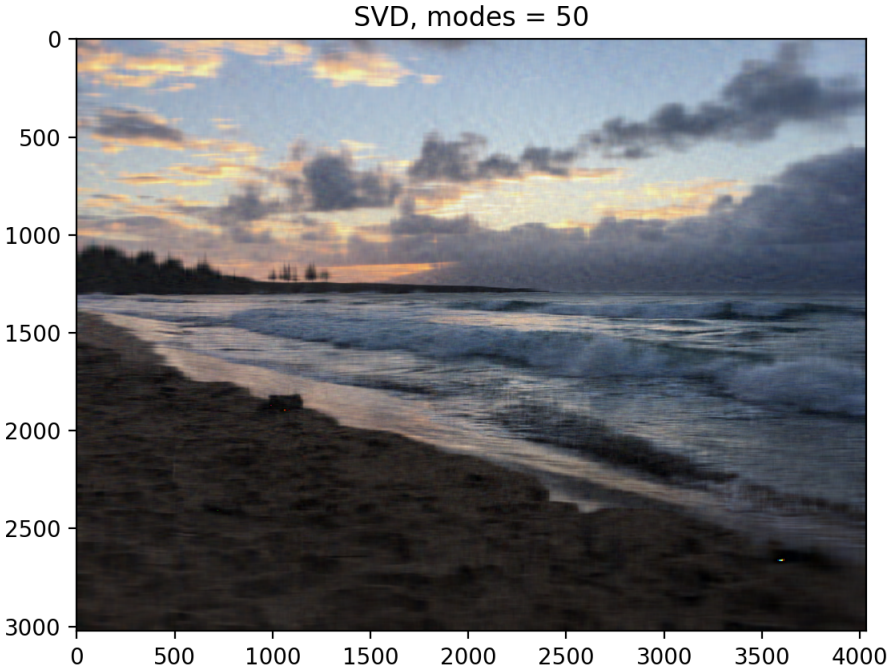


Image with 50 modes (PNG), file size = 10.7 MB
Dimensions: 3024 px × 4032 px



4 Comparison

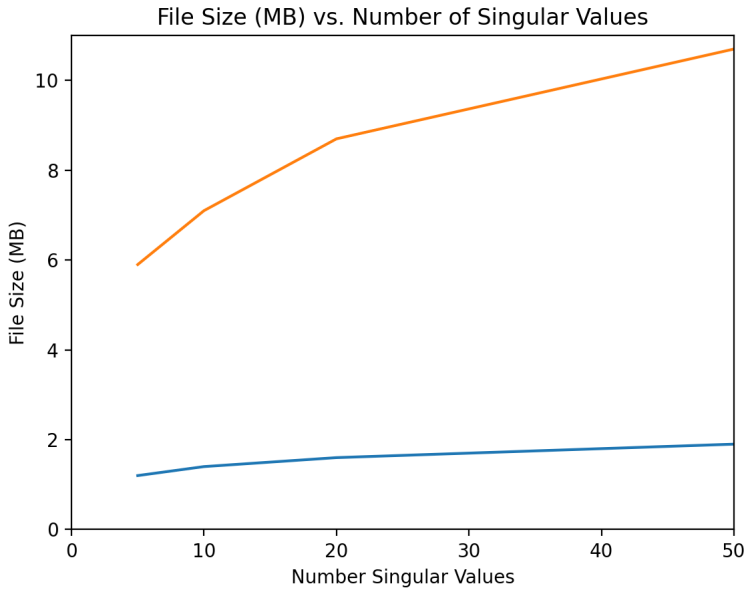
$$\text{Compression Factor} = \frac{\text{Original File Size}}{\text{Compressed File Size}}$$

Table 1: File Size and Compression Factor Comparison for Grayscale Image

Modes	**File Size (MB)**	**Compression Factor**
5	1.2	2.25
10	1.4	1.93
20	1.6	1.69
50	1.9	1.42
Full	2.7	1.000

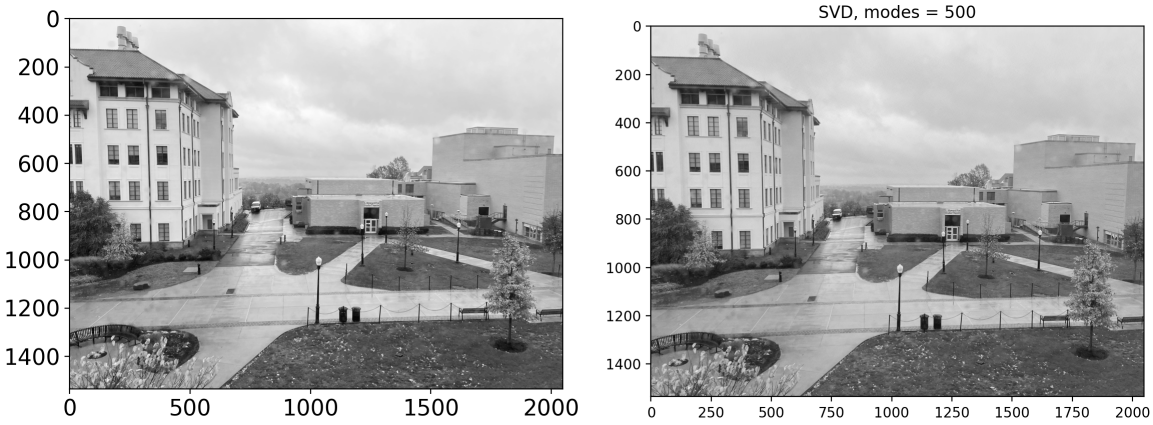
Table 2: File Size and Compression Factor Comparison for Color Image

Modes	**File Size (MB)**	**Compression Factor**
5	5.9	2.24
10	7.1	1.86
20	8.7	1.52
50	10.7	1.23
Full	13.2	1.000

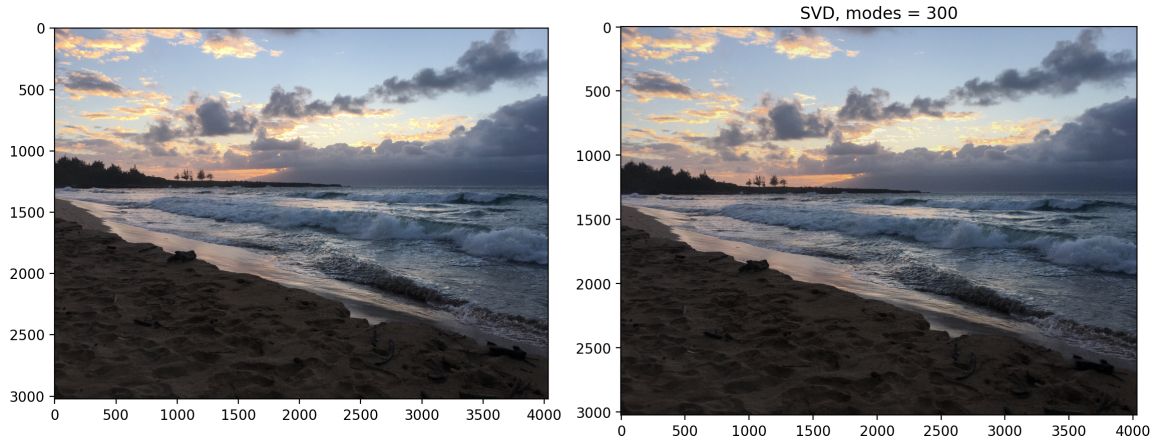


Question: Lastly, reconstruct the image using as many modes as you need so that you cannot tell a difference between the original image and the reconstructed image.

I manually adjusted the number of modes for each image, and compared it side-by-side with the original. With only 500 modes (out of 1536), I am unable to tell the difference between the photographs. The images were very close with 450 modes, however there were slight differences in the white of the building to the left and the water on the ground.



I used the same process for my color image. With 300 (out of 3024) singular values, I am unable to distinguish the pictures. Below 300 singular values, I am able to see pixelation in the clouds.



Reflection on Hypothesis:
Earlier in the paper I predicted the lowest 20% singular values could be safely truncated. For these two images, it appears you can remove more singular values than I anticipated without noticeable change. I constructed my guess conservatively, since I did not much information to base it on. I am surprised that the color image can maintain its quality with less modes, especially consider a larger percentage of them were removed when compared to the grayscale image.

5 Code Appendix

5.1 Grayscale Image Code

Click here to view code on GitHub (BWCompression.py)

```
# Written using Python 3.8
# Note: Running the program takes about 30 seconds (dependent on file size)

import numpy as np
from matplotlib import pyplot as plt

# ========================= Convert Image Into Matrix ========================= #
# Place image in same directory at this Python file:
img = plt.imread('Schmitt Hall 340.jpeg')
modes = 50 # Change number of modes (the more nodes, the higher image quality)

# ========================= Size and Shape of Matrix ========================= #
print('Original Matrix Shape:',img.shape)
print('Original Matrix Size:',img.size)
M = img.shape[0] # number of pixels (height)
N = img.shape[1] # number of pixels (width)

# ============================== Reshaping Data ============================== #
# Convert M x N x 3 matrix into M x N using RGB weights from NTSC:
imgBW = np.zeros((M,N))

for m in range (0,M):
    for n in range (0,N):
        # RBG weights from NTSC: 0.2989*Red + 0.5870*Green + 0.1140*Blue
        imgBW[m][n] = (0.2989)*img[m][n][0] + 1/0.5870*img[m][n][1]
                                            + 1/0.1140*img[m][n][2]

# ========== Uncomment Block to See Uncompressed, Grayscale Image ========== #
# Constructs new image
plt.imshow(imgBW, cmap='gray')
plt.show()
# ============================================================================ #

# ========================== Compress Using SVD ========================== #
# Documentation: https://numpy.org/doc/stable/reference/generated/numpy.linalg.svd.html
V, sigmaVals, wAdjoint = np.linalg.svd(imgBW, full_matrices=True)

print('V shape:', V.shape)
print('sigmaVals shape:', sigmaVals.shape)
print('w* shape:', wAdjoint.shape)
\newpage

# ================== Reconstruct imgBW ================== #
# simgaVals.shape = (1536,) [needs to be M x N with values on diagonal]


# ========= Uncomment Block to See Reconstruction of imgBW ========= #
# squareSigma = np.diag(sigmaVals)
# if M < N:
#     extraZeros = np.zeros((M,N-M))
#     sigma = np.hstack((squareSigma,extraZeros))
```

```
# elif M > N:
#     extraZeros = np.zeros((M-N,N))
#     sigma = np.vstack((squareSigma,extraZeros))

# print('sigma shape:', sigma.shape)

## Reconstruct by using: V @ sigma @ wAdjoint
# reconstructImg = V @ sigma @ wAdjoint
# print(reconstructImg.shape)
# ================================================================================ #

# ================== Truncate Matrices for Compression ================== #
truncV = V[:,:modes]
truncSigmaVals = sigmaVals[:modes]
truncSigma = np.diag(truncSigmaVals)
truncWAdjoint = wAdjoint[:modes]
print(truncV.shape)
print(truncSigmaVals.shape)
print(truncWAdjoint.shape)

# ======================== Construct Truncated imgBW ======================== #
# constructTruncImg = truncV @ truncSigma @ truncWAdjoint
# plt.imshow(constructTruncImg, cmap='gray')
# plt.title('SVD, nodes = {}'.format(modes))
# plt.show()
```

5.2 Color Image Code

Click here to view code on GitHub (ColorCompression.py)

```
# Written using Python 3.8
# Note: Running the program takes about 30 seconds (dependent on file size)

import numpy as np
from matplotlib import pyplot as plt

#========================= Convert Image Into Matrix ============================#
# Place image in same directory at this Python file:
img = plt.imread('INSERT FILE NAME HERE')
modes = 20 # Change number of modes (the more nodes, the higher image quality)

#========================= Size and Shape of Matrix ============================#
print('Original Matrix Shape:',img.shape)
print('Original Matrix Size:',img.size)
M = img.shape[0] # number of pixels (height)
N = img.shape[1] # number of pixels (width)

#================================ Red Channel ================================#
imgRed = np.zeros((M,N))

for m in range (0,M):
    for n in range (0,N):
        imgRed[m][n] = img[m][n][0]

redV, redSigmaVals, redWAdjoint = np.linalg.svd(imgRed, full_matrices=True)

redTruncV = redV[:,:modes]
print(redTruncV.shape)
print(redTruncV.shape)
redTruncSigmaVals = redSigmaVals[:modes]
redTruncSigma = np.diag(redTruncSigmaVals)
redTruncWAdjoint = redWAdjoint[:modes]

redTruncImg = redTruncV @ redTruncSigma @ redTruncWAdjoint
print(redTruncImg.shape) # Recovers M x N dimensions
#============================== Green Channel ================================#
imgGreen = np.zeros((M,N))

for m in range (0,M):
    for n in range (0,N):
        imgGreen[m][n] = img[m][n][1]

greenV, greenSigmaVals, greenWAdjoint = np.linalg.svd(imgGreen, full_matrices=True)

greenTruncV = greenV[:,:modes]
greenTruncSigmaVals = greenSigmaVals[:modes]
greenTruncSigma = np.diag(greenTruncSigmaVals)
greenTruncWAdjoint = greenWAdjoint[:modes]

greenTruncImg = greenTruncV @ greenTruncSigma @ greenTruncWAdjoint
print(greenTruncImg.shape) # Recovers M x N dimensions
#================================ Blue Channel ================================#
imgBlue = np.zeros((M,N))
```

```
for m in range (0,M):
    for n in range (0,N):
        imgBlue[m][n] = img[m][n][2]

blueV, blueSigmaVals, blueWAdjoint = np.linalg.svd(imgBlue, full_matrices=True)

blueTruncV = blueV[:,:modes]
blueTruncSigmaVals = blueSigmaVals[:modes]
blueTruncSigma = np.diag(blueTruncSigmaVals)
blueTruncWAdjoint = blueWAdjoint[:modes]

blueTruncImg = blueTruncV @ blueTruncSigma @ blueTruncWAdjoint
print(blueTruncImg.shape) # Recovers M x N dimensions
# ========== Construct Truncated Matrix ========== #
constructTruncImg = np.zeros((M,N,3))

for m in range (0,M):
    for n in range (0,N):
        constructTruncImg[m][n][0] = int(redTruncImg[m][n])
        constructTruncImg[m][n][1] = int(greenTruncImg[m][n])
        constructTruncImg[m][n][2] = int(blueTruncImg[m][n])

# Some integer values were not in range of [0,255] which resulted in clipping.
# Suggestion from Dr. Forgoston to scaled all the values using the code:
#scaledConstructImg = np.array(constructTruncImg/np.amax(constructTruncImg)*255, np.int32
scaledConstructImg = np.array(constructTruncImg/np.amax(constructTruncImg)*255, np.uint8

# print(constructTruncImg.shape)
# print(constructTruncImg)

plt.imshow(scaledConstructImg)
plt.title('SVD, modes = {}'.format(modes))
plt.show()
plt.imsave('ColorImageCompressed.png', scaledConstructImg)
```