Image Analysis Using Singular Value Decomposition

Vince Verdugo

San Diego State University

vverdugo8822@sdsu.edu

(571) 314 - 6927

April 26, 2024

Abstract

The purpose of this report is to evaluate the results of performing singular value decomposition (SVD) for image compression using Python. SVD essentially decomposes one matrix into three separate matrices, from which we can perform operations to approximate our original matrix. As Elizabeth A. Compton and Stacey L. Ernstberger, PhD write in *Singular Value Decomposition: Applications to Image Processing*, SVD is helpful in analyzing images because it allows us to "identify the components of the image which contribute the least to overall image quality". This helps us reduce the amount of data an image contains, further improving the amount of memory required.

Introduction

When dealing with large portions of data, a common solution may be to analyze it using SVD, this is especially helpful when analyzing images. Black and white digital pictures are composed of elements on a saturation scale of 0 to 255 per pixel (Compton & Ernstberger, PhD, 2020). Considering that photos taken on a smartphone are most commonly 640 x 320 pixels, these images easily contain over 200,000 values. In an effort to minimize the amount of memory storage needed for these pictures, we can perform SVD on the derived matrices to determine which values can be removed without sacrificing the quality. We begin this process by formulating our equation,

$$A_{n\times m} = U_{n\times s} D_{s\times s} V^T_{s\times m}$$

in which A, U, D, and V^T are matrices, V^T being the transpose of matrix V. The values n and m are the dimensions of our image in pixels, and s is the minimum of n and m. With our data now consisting of three matrices, we can examine each to determine where the most important data is located. The matrix U contains the import information from the rows of matrix A. Similarly,

matrix V^T contains the important information from the columns of matrix A. Matrix D is significant in that it is a diagonal matrix that only contains the singular values (eigenvalues) of matrix A (Mathews, 2014). These singular values are important because it allows us to determine the point in which the additional data becomes insignificant to our reconstruction. The method we will use to determine this will be by calculating the variance of the singular values. We will define "modes" as the rows and columns of our matrices U, D, and V^T. The first mode will be defined as row 1 of U, the first singular value along the diagonal of D, and the first column of V^T; mode 2 will be rows 1 and 2 of U, the first two singular values of D, and columns 1 and 2 of V^T; and so forth. As we can see, it is not an easy task to determine the values of U, D, and V^T by hand. So instead, we will use Python to calculate the SVD matrices.

Data and Method

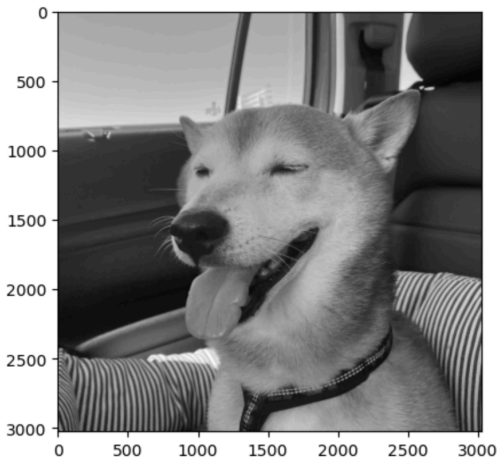
In this example, we will be using a picture of my dog Heath. Using Python, we will import the photo, convert its information into a matrix, perform SVD on the matrix, and analyze how much data we should retain to make a high-quality reconstruction. Below is the 3204 x 3204 pixel photo.

To begin we will import the numpy, matplotlib.pyplot, and io libraries. We can then use io to read the image into a grayscale matrix representation titled 'heath'. The 'heath' matrix is a

two-dimensional array with the same dimensions as our original picture, 3204 x 3024. Although grayscale images are composed of elements on a scale of 0 to 255 (Compton & Ernstberger, PhD, 2020), when read into Python, our values are set to a scale of 0 to 1. Some data from the matrix can be seen below.

	0	1	2	3	4	5	6	...	...	...	...	...
0	0.668239	0.668239	0.668239	0.668239	0.668239	0.668239	0.668239					
1	0.668239	0.668239	0.668239	0.668239	0.668239	0.668239	0.668239					
2	0.672161	0.672161	0.672161	0.668239	0.668239	0.668239	0.668239					
3	0.672161	0.672161	0.672161	0.672161	0.668239	0.668239	0.668239					
4	0.672161	0.672161	0.672161	0.672161	0.668239	0.668239	0.668239					
...	...	...	...	...	...	...	...	...				
3019	0.179864	0.132805	0.148491	0.105354	0.140082	0.181247	0.102816					
3020	0.128883	0.172020	0.175942	0.095538	0.102816	0.165561	0.122424					
3021	0.142597	0.170048	0.158284	0.099460	0.087130	0.122424	0.122424					
3022	0.130833	0.119068	0.166127	0.091617	0.094973	0.153796	0.098895					
3023	0.154362	0.134754	0.138676	0.119068	0.110659	0.087130	0.059679					

Using the matplotlib.pyplot library, we will call the command imshow to plot our image in grayscale. This will plot each value from the matrix ‘heath’ on a 3024 x 3024 graph, which can be seen here.



Now that we have our full matrix representation of our image, we can use the linear algebra SVD function from the numpy library to create the U, D, and V^T matrices. To do this we will declare objects ‘u’, ‘d’, and ‘vt’ and assign them with the multidimensional arrays that correspond to the SVD solutions. As the lingalg function will do the heavy work for us, when we call this function it is important that we pass “full_matrices=False” as the second parameter. This will ensure that our matrices ‘d’ and ‘vt’ will end up with dimension s x s and s x m. In our case, since the image is a square with dimensions 3024 x 3024, s is equal to n and m. However, if our image was not square, failing to include our second parameter would disrupt our results.

Results

Now that we have performed our SVD, we can begin to analyze our results. By looking at the first 5 rows and columns of ‘u’, ‘d’, and ‘vt’ we can see these new values follow our intuition.

u = [[-0.02134584, -0.01819523, 0.00406495, -0.00188764, -0.00465745]

[-0.02135427, -0.018183, 0.0040972, -0.00189568, -0.00472049]

[-0.02136184, -0.01817625, 0.0040585, -0.00188026, -0.00473762]

[-0.02135726, -0.01821018 , 0.0040353, -0.0020057, -0.00483122]

[-0.02135998, -0.01822915, 0.00402907, -0.00195587, -0.00476086]]

d = [1335.82524278, 372.40314358, 213.30203433, 161.59289574, 123.33736901]

vt = [[-0.01785366, -0.01787269, -0.01785669, -0.01785636, -0.01784794]

[-0.02567887, -0.0256322, -0.02560005, -0.02555965, -0.02551142]

[0.00818932, 0.00815293, 0.00800387, 0.00799232, 0.0079415]

[0.01520698, 0.0151179, 0.01493119, 0.01492067, 0.01470859]

[0.00430646, 0.0042027, 0.00369228, 0.00307828, 0.00260619]]

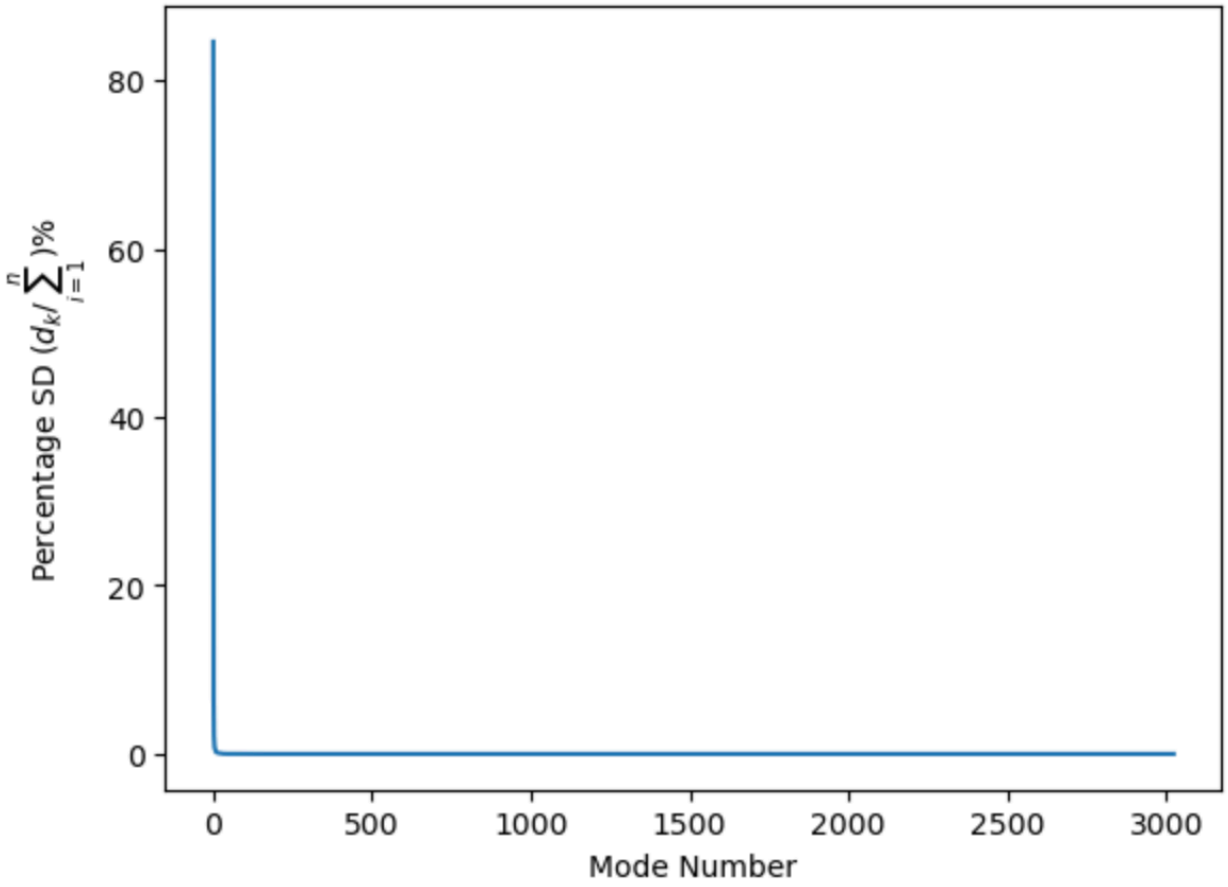
To verify that the matrix product of 'u', 'd', and 'vt' is our original 'heath' matrix, we will use the '@' operator to mimic matrix multiplication between 'u', 'd', 'vt'. We will then calculate A = u@np.diag(d)@vt, where np.diag converts our one dimensional array into a two dimensional diagonal array. This gives us:

	0	1	2	3	4	5	6	...	...	...	...	...
0	0.668239	0.668239	0.668239	0.668239	0.668239	0.668239	0.668239					
1	0.668239	0.668239	0.668239	0.668239	0.668239	0.668239	0.668239					
2	0.672161	0.672161	0.672161	0.668239	0.668239	0.668239	0.668239					
3	0.672161	0.672161	0.672161	0.672161	0.668239	0.668239	0.668239					
4	0.672161	0.672161	0.672161	0.672161	0.668239	0.668239	0.668239					
...	...	...	...	...	...	...	...	...				
3019	0.179864	0.132805	0.148491	0.105354	0.140082	0.181247	0.102816					
3020	0.128883	0.172020	0.175942	0.095538	0.102816	0.165561	0.122424					
3021	0.142597	0.170048	0.158284	0.099460	0.087130	0.122424	0.122424					
3022	0.130833	0.119068	0.166127	0.091617	0.094973	0.153796	0.098895					
3023	0.154362	0.134754	0.138676	0.119068	0.110659	0.087130	0.059679					

.

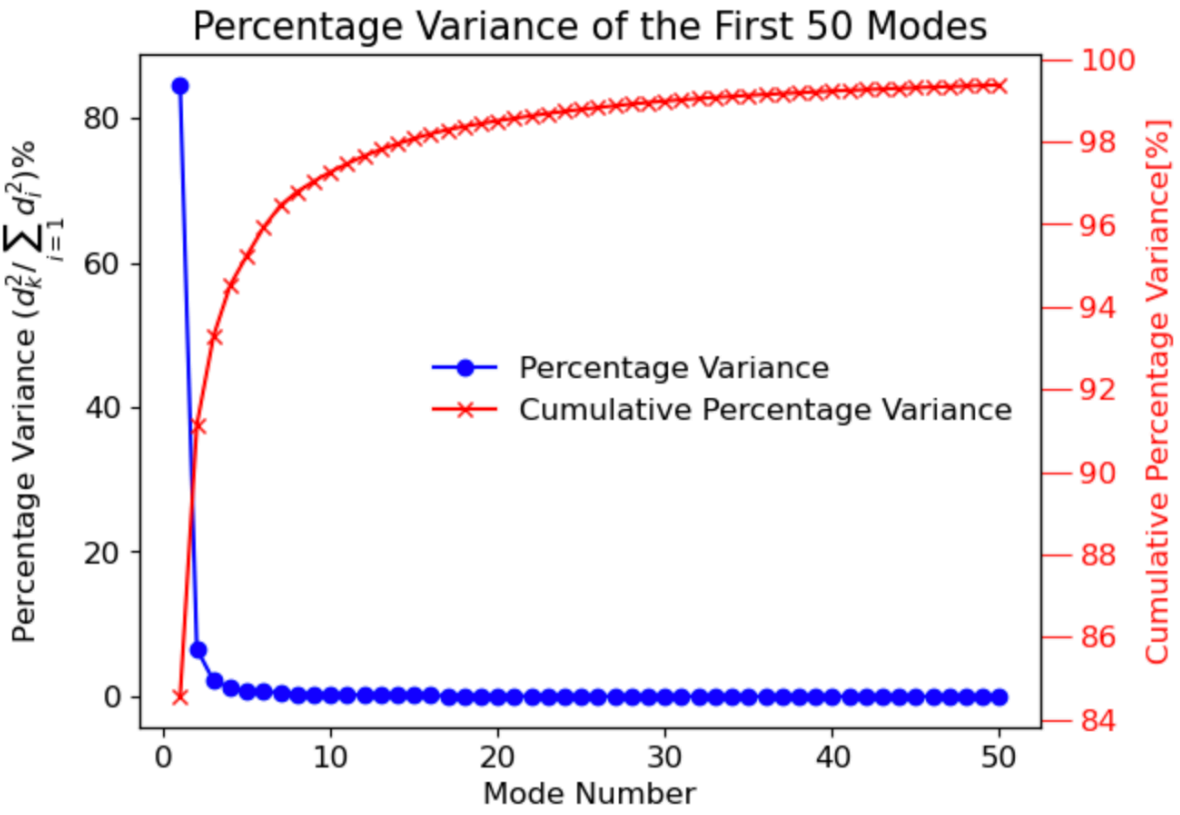
Looking back to our results from earlier, we can see that 'A' and 'heath' contain the same values, and therefore are equal. After we have confirmed our SVD has been done correctly, we will start by creating a graph that visualizes the variance in our singular values.

In determining the variance of ‘d’, we will better determine the composition of our data as the number of modes increases. For this we will first calculate the variance of ‘d’, which is simply ‘varD = d^2’. We can then plot ‘varD’ over the sum of ‘varD’ (varD/np.sum(varD)). This gives us the following graph.



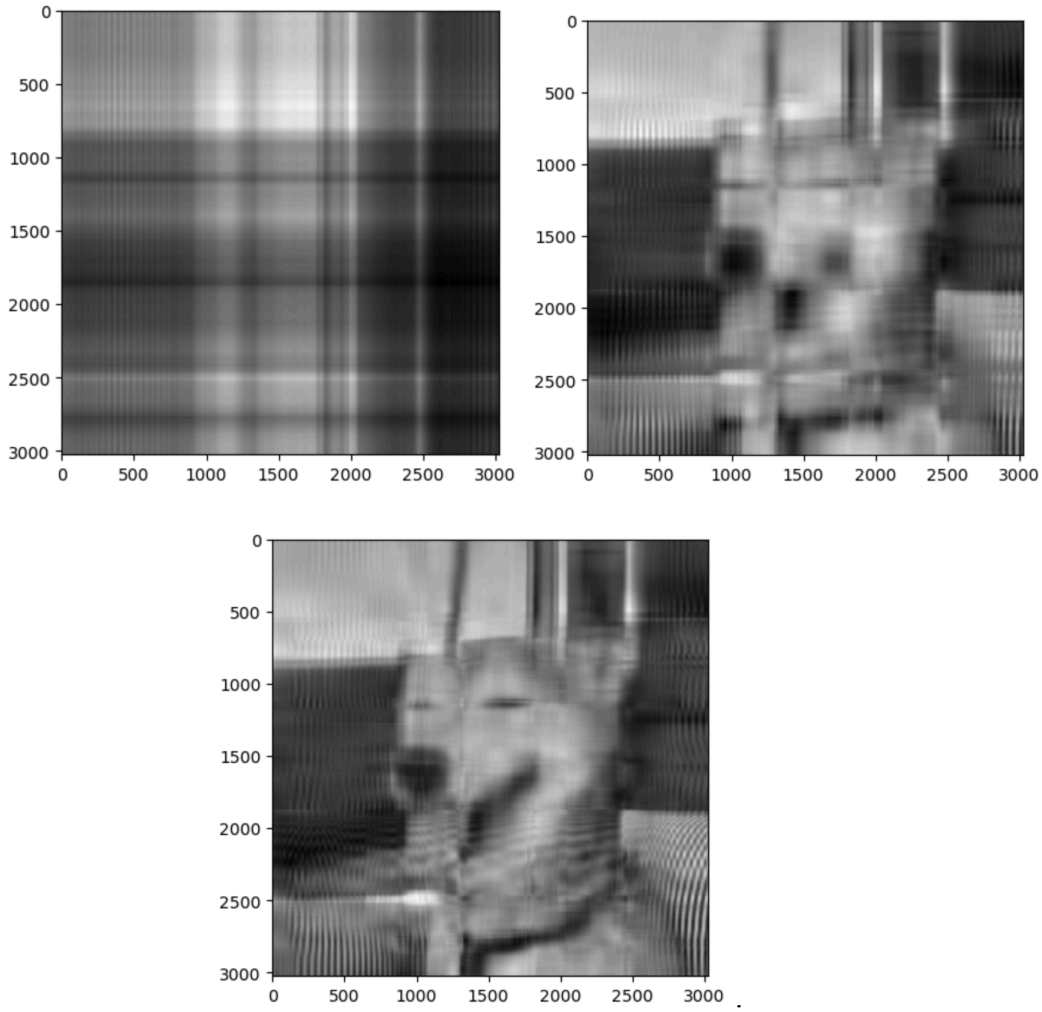
This graph shows us that as the number of modes increases, the information added to the image related to that mode rapidly decreases and becomes almost zero. As we can see, the percentage of variance begins very high at the first mode, at over 80%. This is due to the first singular value of ‘d’ being the largest. However, it is unclear at what point our function starts to converge towards zero. To better understand what happens in the first few modes, we will inspect a smaller portion of modes. More than this, we will also plot the cumulative variance percentage.

For our purposes, we will think of this cumulative variance percentage as the total amount of data from the image that has been reconstructed. For reference, we will consider 100% cumulative variance as 100% image reconstruction. Creating the object ‘cvarD’ which is simply the sum of ‘varD’, will show us what percent of the overall image composition is derived from “k” number of modes. The following is the plot:



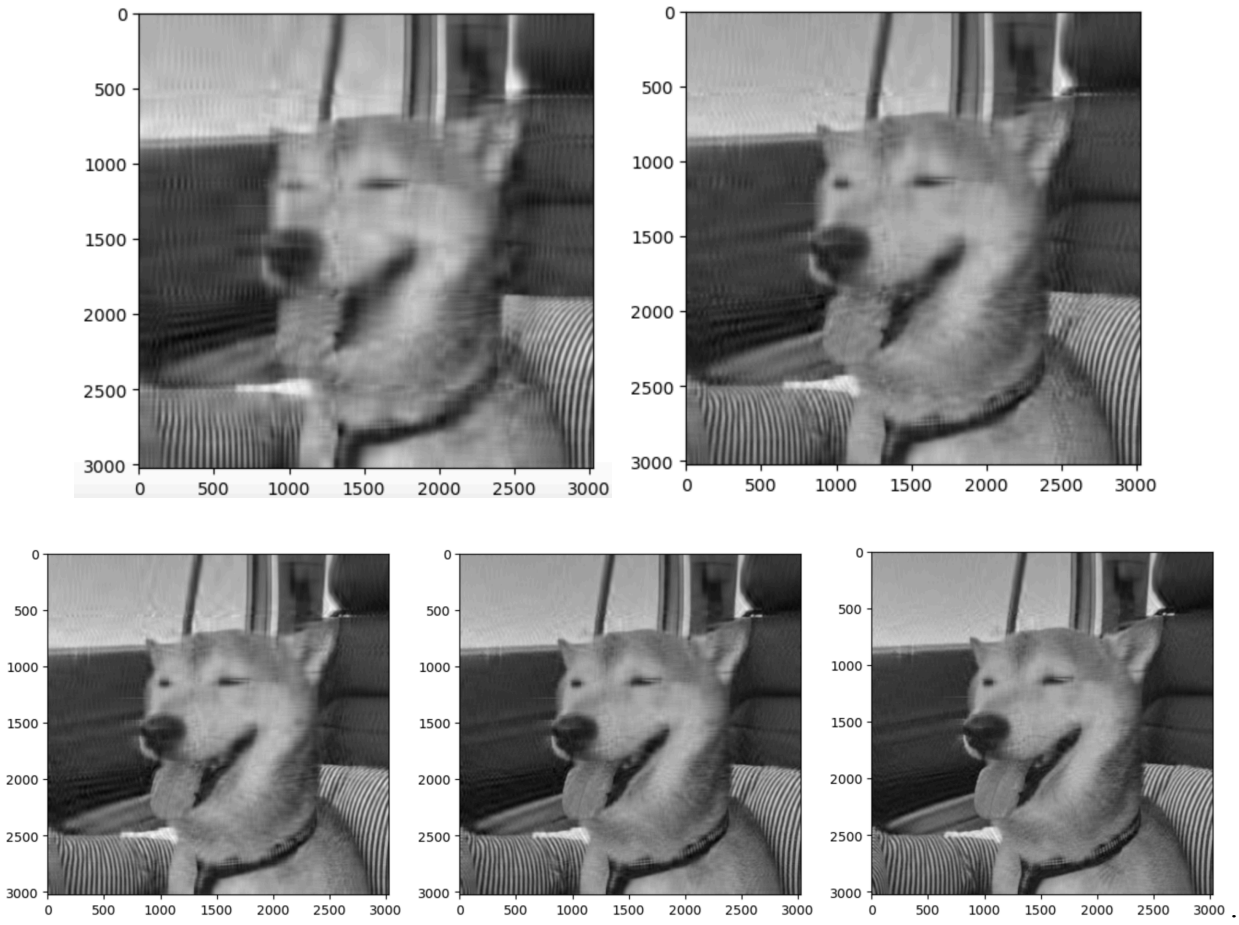
For example, as shown in the graph above, we can see that while the percentage variance of the first 10 modes has reached almost zero, it is at 10 modes where we see just under 98% cumulative percentage variance. While this information is extremely helpful to see the variance in our singular value matrix, we must still test reconstructing at different numbers of modes to determine where we will get our best quality. To do this we will create an object named ‘heathRe’ that equals ‘u[:,0:k+1]@np.diag(d[0:k+1])@vt[0:k+1,:]’. We will first compare the

graphs we obtain for k = 1, 5, and 10. When evaluated at these values of k we obtain the respective graphs:

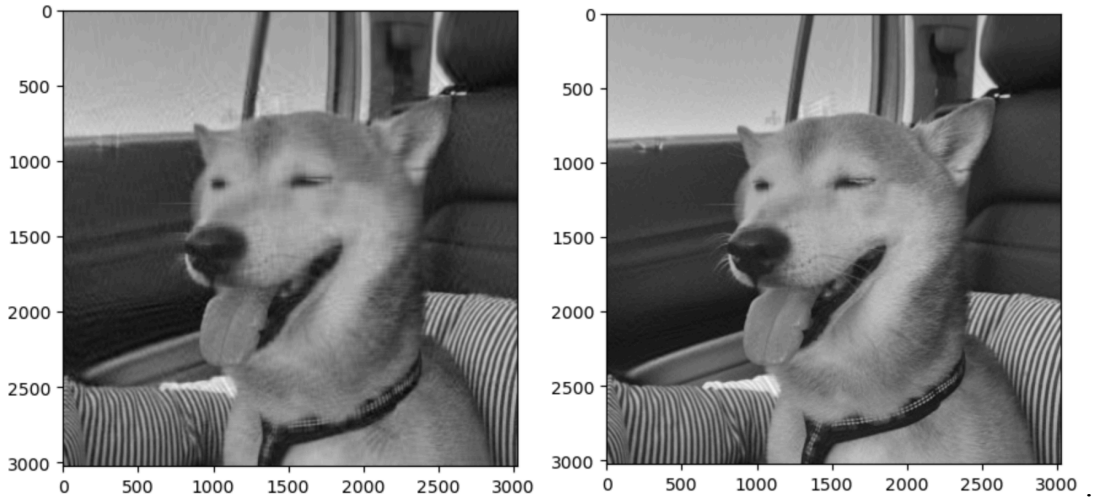


It is obvious by looking at the first graph, that although the first mode obtains over 80% cumulative variance, it is not enough to give us a good reconstruction of the image. As we increase to 5 modes and then 10, we start to see more detail and our reconstruction begins to take its original form. Referencing our plot “Percentage Variance of the First 50 Modes”, we can see that the cumulative percentage variance continues to increase exponentially from mode 10 to approximately mode 25. After that, we see that as the number of modes increases, its growth is almost linear and its slope close to zero. That being said, let us look at how the cumulative

variance at these modes visually looks in reconstruction. These are the respective ‘heathRe’ plots for k = 15, 25, 30, 40, and 50:



In the first row we can clearly see the jump in composition from 15 to 25 modes, being able to better identify areas around the eyes and mouth. In row two, we see the gradual increase in quality from modes 30 to 40 to 50. The photos begin to become more refined and start becoming less grainy. Evaluating ‘cvarD’ at [49] we get the value 99.407, which tells us that with only 50 modes, we already have over 99% cumulative variance. Let us take this one step further, and compare the plots of mode 50 and mode 100. Let us also compare the respective values of ‘cvarD’ at each mode. Plotting and comparing gives us:



We see by doubling our number of modes we receive an image that looks almost identical to our original composition. Deeper analysis shows that the cumulative variance of 100 modes equals 99.68%. Considering the difference between the cumulative variance percentage for 50 modes and 100 modes is approximately equal to 0.27%, we could argue that the increase in cumulative percentage is not significant enough. Doubling the number of modes and therefore doubling the required memory might not be justified. However, might we consider that by using 100 modes in our reconstruction, we are using only 3% of our total number of modes. So by increasing the number of modes to 100, we have still dramatically decreased the amount of overall data. In doing so we have also arrived at a reconstruction that thus far is closest to 100% cumulative variance and the highest quality.

In collecting all this data, we might ask at what point do we no longer need more modes. For this, we can construct a sensitivity analysis that helps us determine at what rate the accuracy of reconstruction changes as the number of modes increases. Understanding where the rate of change slows can help us determine the number of modes that are necessary. For this example, having an increase < 0.2% seemed an appropriate mode to stop at. We can see these values in the table below.

Number of modes (k)	Change in number of modes (Δk)	Cumulative Variance % (cvarD[k])	Change in Cumulative Variance % (ΔcvarD[k])	S[%] (ΔcvarD[k])(Δk)
1	0	84.5590457608596	0	0
5	+4	95.24514801177585	+10.68610225091625%	+2.672
10	+9	97.26620953324148	+12.707163772381875%	+1.412
15	+14	98.07248174536353	+13.513435984503928%	+0.965
25	+24	98.78133499962871	+13.513435984503928%	+0.593
30	+29	98.9821277032727	+14.423081942413091%	+0.497
40	+39	99.2252643389066	+14.666218578046994%	+0.376
50	+49	99.37305345286009	+14.814007692000487%	+0.302
100	+99	99.67262249663924	+15.113576735779631%	+0.152

Conclusion

When dealing with large matrices, SVD can be a very helpful method in identifying key elements. This report has shown that this method of SVD can be applied to compression of an image. As images begin to grow in quality, they require increased dimensions, data, and memory space. By converting the data of the image into a matrix, we can perform SVD and obtain three different matrices whose product is equal to the original matrix. By manipulating the singular values of our matrix, we are able to reconstruct the image into one that is of similar quality yet a fraction of the size. In our case, we derived a reconstruction of our image of Heath that used 3% of the data yet was 99.67% accurate to the original photo. While this process is computationally extensive, programming languages such as Python have built in methods that aid in computation. Another benefit of using SVD is its other applications such as compression of a video using each

frame (Compton & Ernstberger, PhD, 2020). However SVD is considered a lossy compression technique which means that the data is irreversible (Swathi et al, 2017). That being said, JPEG, which is the most commonly used compression technique, is also considered lossy. Since JPEG is a built-in feature to most digital cameras and digital services, it is an extremely common method of image compression. Still, SVD should still be considered a useful technique when performing image compression in Python.

References

Compton, E. A., & Ernstberger, PhD, S. L. (2020). *Singular Value Decomposition: Applications to Image Processing*. 18-Citations 2020.Compton.pdf. Retrieved April 17, 2023, from https://www.lagrange.edu/academics/undergraduate/undergraduate-research/citations/18-Citations2020.Compton.pdf

Jackson, C. (n.d.). *How to Size Photos for Cell Phones*. Small Business - Chron.com. Retrieved April 17, 2024, from https://smallbusiness.chron.com/size-photos-cell-phones-44095.html

Mathews, B. (2014, December 12). *Image Compression using Singular Value Decomposition (SVD) - NTNU*. Personal webpages at NTNU. Retrieved April 17, 2024, from https://folk.ntnu.no/cloverm/115BWinter2020/MathewsSVDImages.pdf

Swathi, H. R., & et al. (2017). *Image compression using singular value decomposition*. Open Access proceedings Journal of Physics: Conference series. Retrieved Apr 20, 2024, from https://iopscience.iop.org/article/10.1088/1757-899X/263/4/042082/pdf#:~:text=Another%20advantage%20of%20using%20SVD,R%2C%20G%20and%20B%20matrices.

Python Code

```
#import libraries we will need
import numpy as np
import matplotlib.pyplot as plt
from skimage import io
import pandas as pd
#import photo of heath and read in data
heath = io.imread('/Users/vinceverdugo/Downloads/mathmodeling/heathySVD.jpg',
                          as_gray=True)
#create data frame to better visualize data
heath_df = pd.DataFrame(data = heath)
print(heath_df)
#examine first 5x5 of heath
heath[0:5, 0:5]
#plot heath
plt.imshow(heath, cmap=plt.cm.gray)
#SVD calculation
u, d, vt = np.linalg.svd(heath, full_matrices=False)
#look at first five of u, d, and vt
print(f'u = \n{u[0:5, 0:5]}')
print(f'd = \n{d[0:5]}')
print(f'vt = \n{vt[0:5, 0:5]}')
#matrix multiplication to verify we have performed SVD correctly
A = u@np.diag(d)@vt
#convert to data frame to visualize data in comparison to heath
A_df = pd.DataFrame(data = A)
print(A_df)
#calculate variance
```

```
varD = d**2
#variance plot
#we will plot variance as a function of varD over the sum of varD
#pvarD will be represented as a percentage, hence *100
pvarD = (varD/np.sum(varD))*100
plt.plot(pvarD)
plt.ylabel(r"Percentage SD $(d_k/\sum_{i=1}^n)\%$")
plt.xlabel(r"Mode Number")
plt.show()
#cumulative variance percentage calculation and comparison plot
cvarD = np.cumsum(pvarD)
plt.figure(figsize = (6, 4))
fig, ax = plt.subplots()
modeNo = np.arange(1, 51)
p1, = plt.plot(modeNo, pvarD[0:50],
        'b',marker = 'o',
         label = 'Variance Percentage')
plt.ylabel(r"Percentage Variance $(d_k^2/\sum_{i=1}^nd_i^2)\%$",
          fontsize = 12)
plt.xlabel(r"Mode Number", fontsize = 12)
plt.title(r"Percentage Variance of the First 50 Modes",
         fontsize = 15)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
ax2 = ax.twinx()
p2, = plt.plot(modeNo, cvarD[0:50],
               'r', marker = 'x',
               label = 'Cumulative Variance Percentage')
ax2.tick_params('y', colors='r', size = 12, labelsize = 12)
```

```
ax2.set_ylabel("Cumulative Percentage Variance[%]", size = 12)
ax2.yaxis.label.set_color('red')
plt.legend(handles=[p1, p2],loc='center right',
           fontsize = 12, frameon = False)
plt.show()
#heath reconstruction from the first mode
k= 1
heathRe = u[:,0:k] @ np.diag( d[0:k] ) @ vt[0:k, :]
plt.imshow(heathRe, cmap=plt.cm.gray)
plt.show()
#heath reconstruction from the first 5 modes
k= 5+1
heathRe = u[:,0:k] @ np.diag( d[0:k] ) @ vt[0:k, :]
plt.imshow(heathRe, cmap=plt.cm.gray)
plt.show()
#heath reconstruction from the first 10 modes
k = 10+1
heathRe = u[:,0:k] @ np.diag( d[0:k] ) @ vt[0:k, :]
plt.imshow(heathRe, cmap=plt.cm.gray)
plt.show()
#heath reconstruction from the first 15 modes
k = 15+1
heathRe = u[:,0:k] @ np.diag( d[0:k] ) @ vt[0:k, :]
plt.imshow(heathRe, cmap=plt.cm.gray)
plt.show()
#heath reconstruction from the first 25 modes
k = 25+1
heathRe = u[:,0:k] @ np.diag( d[0:k] ) @ vt[0:k, :]
plt.imshow(heathRe, cmap=plt.cm.gray)
```

```
plt.show()
#heath reconstruction from the first 30 modes
k = 30+1
heathRe = u[:,0:k] @ np.diag( d[0:k] ) @ vt[0:k, :]
plt.imshow(heathRe, cmap=plt.cm.gray)
plt.show()
#heath reconstruction from the first 50 modes
k = 40+1
heathRe = u[:,0:k] @ np.diag( d[0:k] ) @ vt[0:k, :]
plt.imshow(heathRe, cmap=plt.cm.gray)
plt.show()
#heath reconstruction from the first 100 modes
k = 50+1
heathRe = u[:,0:k] @ np.diag( d[0:k] ) @ vt[0:k, :]
plt.imshow(heathRe, cmap=plt.cm.gray)
plt.show()
#heath reconstruction from the first 100 modes
k = 100+1
heathRe = u[:,0:k] @ np.diag( d[0:k] ) @ vt[0:k, :]
plt.imshow(heathRe-, cmap=plt.cm.gray)
plt.show()
print(cvarD[9]-cvarD[0])
print(cvarD[14]-cvarD[0])
print(cvarD[24]-cvarD[0])
print(cvarD[29]-cvarD[0])
print(cvarD[39]-cvarD[0])
print(cvarD[49]-cvarD[0])
print(cvarD[99]-cvarD[0])
```