Small Images with a Big Impact:

Photo Compression Using Singular Value Decomposition

Jasmine Yabut

The Comp Comp (Compression Company)

jyabut3042@sdsu.edu

858-386-2115

github.com/jyabut

December 7, 2024

Abstract

By using Singular Value Decomposition (SVD) in R, photos can be compressed to save storage space without compromising on the details in the image itself. The hopes were to create an app with a dropdown menu that provides percentages of how much of the photo the user would like to keep. This would allow users to quickly compress the photo. Through SVD, a photo is broken down into matrices containing metadata of how to reconstruct the photo. It was discovered that a small portion of the metadata can recreate a photo. From there, different photos were evaluated, and varying amounts of metadata were used to recreate them. This gave our team ideas for the compression levels, but there were conflicting decisions. Due to varying amounts of details, the photos could not be compressed at the same levels, and it occasionally came down to personal preference. To have a fully functioning app, a maximum amount of compression will be provided to the user based on the photo's dimensions.

Introduction

"A picture is worth a thousand words." This idiom has been around for years and resonates with people around the world. Most, if not all, people take photos daily. Approximately 660 billion photos were taken in 2013, and 1.81 trillion photos were taken in 2023 (Broz, 2024). By tripling the number of photos taken in 2013, the number of photos taken has grown at an incredible rate. As of September 2024, the average American takes twenty photos a day, and with 94% of photos are taken on smartphones. As the number of photos keeps growing, the amount of storage needed increases. Storage can be a problem for the average person and social media companies. With users uploading photos every day, those companies' servers constantly need more storage and will increase costs. If there was a way to compress the photo without losing too many details, this could decrease storage restraints and will allow people to keep more photos and memories. So, how can we decrease the quantity (size) without compromising the quality? How can we store more photos while also paying less?

Our app's goal is to combat this issue. After uploading to the app, the photos will be compressed with the help of Singular Value Decomposition (SVD) in R programming. SVD is a process that decomposes a matrix into three matrices that represent the relationships between the rows, the relationships between the columns, and the strength of the relations (Neokai, 2023). If the three matrices with the photo's metadata were multiplied together, they will recreate the original photo. By only using some of the rows and columns, it is possible to create a variation of the photo with the same detail but less data, leading to less required storage. The app will also allow users to choose a percentage of how much of the photo they would like to keep. The rest of the paper will go into more detail about this method and our findings throughout the development process.

Data and Methods

Throughout this section and Results, the code used to gather the tables and figures will be below the correlating table/figure. The code can also be found at the end of the paper or the GitHub link in the References. The images used for our dataset will also be provided in the repository.

With matrix **A** of dimensions *m* rows and *n* columns $(m \leq n)$¸ SVD will decompose $\mathbf{A}_{n \times m}$ into

$$\mathbf{A}_{n\times m} = \mathbf{U}_{n \times m}\, \mathbf{D}_{m \times m}\, (\mathbf{V}^{\mathbf{t}})_{m \times m} \tag{1}$$

If $m \geq n$, the SVD will then decompose $\mathbf{A}_{n \times m}$ into

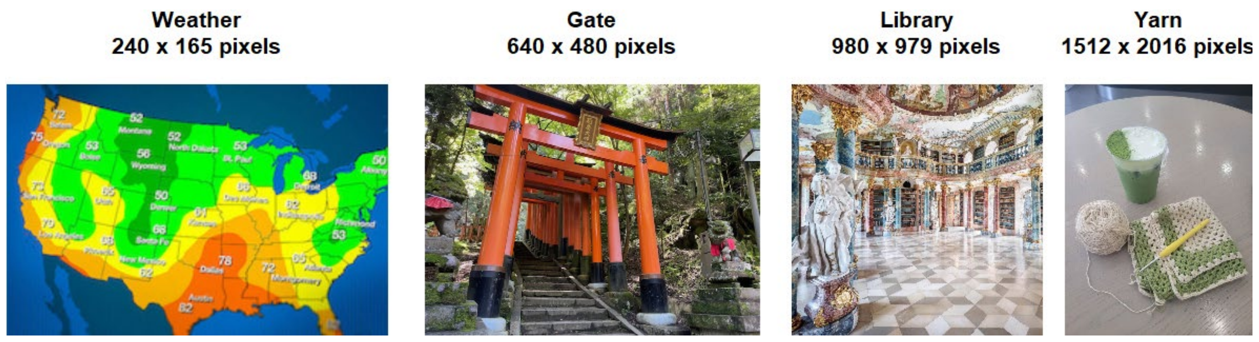
$$\mathbf{A}_{n\times m} = \mathbf{U}_{n \times n}\, \mathbf{D}_{n \times n}\, (\mathbf{V}^{\mathbf{t}})_{n \times m} \tag{2}$$

U has *m* orthonormal column vectors that are unit eigenvectors of $\mathbf{AA^t}$, **V** has *m* orthonormal column vectors that are unit eigenvectors of $\mathbf{A^tA}$, and **D** is a diagonal matrix that holds the square root of $\mathbf{AA^t}$'s eigenvalues (Shen, 2024). In R Studio, the photo was represented as matrix **A,** with matrices **U** and **V** providing information about the photo, and **D** telling us how the information affects the photo. The values in **D** are also called eigenvalues or modes. By multiplying these matrices, **A** is reconstructed. With that knowledge, multiplying parts of the matrices will create a partial version of **A**. Our team learned about this process and applied it to photos.

The goal was to find the levels of compression to put in a dropdown menu: original, medium, and small. The team also wanted to check for the limit. To decide the suggested percentages, we each chose a photo to evaluate the SVD compression. We made sure they varied in size and detail, and they are all shown in Figure 1. The sizes ranged from 240 x 165 pixels to 1512 x 2016 pixels. In the photos, bigger dimensions are the length (240, 640, 980, 2016), and the smaller dimensions are the width (165, 480, 979, 1512). Through SVD, **D**'s dimensions depend on the smaller value of **A**'s dimensions, so **D**'s size would be width x width.

Figure 1

Photos Used in Testing SVD Compression Levels

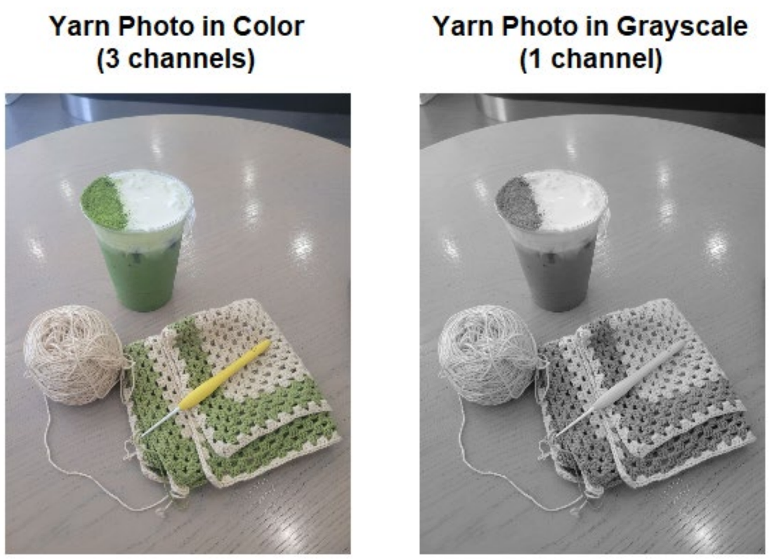


```r
##### importing data (photos) #####
lbry <- load.image('library.jpg')
gate <- load.image('gate.jpg')
wthr <- load.image('weather.jpg')
yarn <- load.image('yarn.jpg')

##### Figure 1: creating a chart of all the photos to put in paper #####
layout(matrix(c(1,2,3,4), nrow = 1),widths = c(33,29,22,16))
par(mar=c(1,0.5,8,0.5))

plot(wthr, axes=FALSE)
title(main="Weather\n240 x 165 pixels",cex.main=2)

plot(gate, axes=FALSE)
title(main="Gate\n640 x 480 pixels",cex.main=2)

plot(lbry, axes=FALSE)
title(main="Library\n980 x 979 pixels",cex.main=2)

plot(yarn, axes=FALSE)
title(main="Yarn\n1512 x 2016 pixels",cex.main=2)
```

From here, most of this section will focus on the last photo containing a matcha drink and yarn. To simplify the research, we worked with the photos in grayscale (1 color channel). A colored photo can consist of three (RGB) or four (CMYK) channels, and this would have required a SVD analysis for each channel. Figure 2 compares how the photo looks in color and grayscale.

Figure 2

Comparison of the Yarn photo in Color and Grayscale



```r
##### get dimensions to calculate modes for percentages #####
# the matrices' sizes depend on the smaller dimension
dims <- dim(yarn)
row = dims[1]
col = dims[2]

if (row <= col) {
  modes = row
}else {
  modes = col
}

##### Make the photo data black-and-white #####
graydat = grayscale(yarn)

##### Figure 2: Showing Colored vs Grayscale #####

par(mfrow=c(1,2), mar=c(1,1,4,1))

plot(yarn, xlim = c(0, row), ylim = c(col,0), axes = FALSE,
     main = 'Yarn Photo in Color\n(3 channels)')

plot(graydat, xlim = c(0, row), ylim = c(col,0), axes = FALSE,
     main = 'Yarn Photo in Grayscale\n(1 channel)')
```

After converting to grayscale, the photo was put through SVD and turned into matrices **U, D,** and **V**. The singular values in **D** were the focus of understanding what contributed the most to the photo. Typically, the values with the most impact are in the beginning of the matrices. Since the data was decomposed, the matrices had the data decreasing. Percentages were calculated from **D** and represented on a scree plot. This helped visualize the eigenvalues' contributions to the cumulative photo. The first 20 values were used as a benchmark to see how far the decomposition can go.

Based on the plot, we were encouraged to find the minimum amount of *k* modes to reconstruct the photo. To reconstruct the yarn photo ($m \geq n,$ where $m = 2016$ and $n = 1512$) with only *k* modes, Formula 2 would be changed to

$$\mathbf{A}_{n \times m} = \mathbf{U}_{n \times n}\, \mathbf{D}_{n \times n}\, (\mathbf{V}^{\mathbf{t}})_{k \times m} \tag{3}$$

After, our team tested different values of *k* in R to check their photo's state. This helped decide what percentage worked for the medium sized, small sized, and limit of compression levels on their photo. Once these levels were found, we reconvened and discussed their findings. These values helped decide what common values were found and would be used in the app.

Results

After running SVD on the yarn photo and retrieving **U**, **D**, and **V**, the percentages of variance/strength were calculated from the first 20 values in **D** with the code below. Table 1 shows the resulting percentages and revealed that the first six eigenvalues contributed at least 0.1% variation.

Table 1

Percentage of Variance for Photo's SVD Analysis

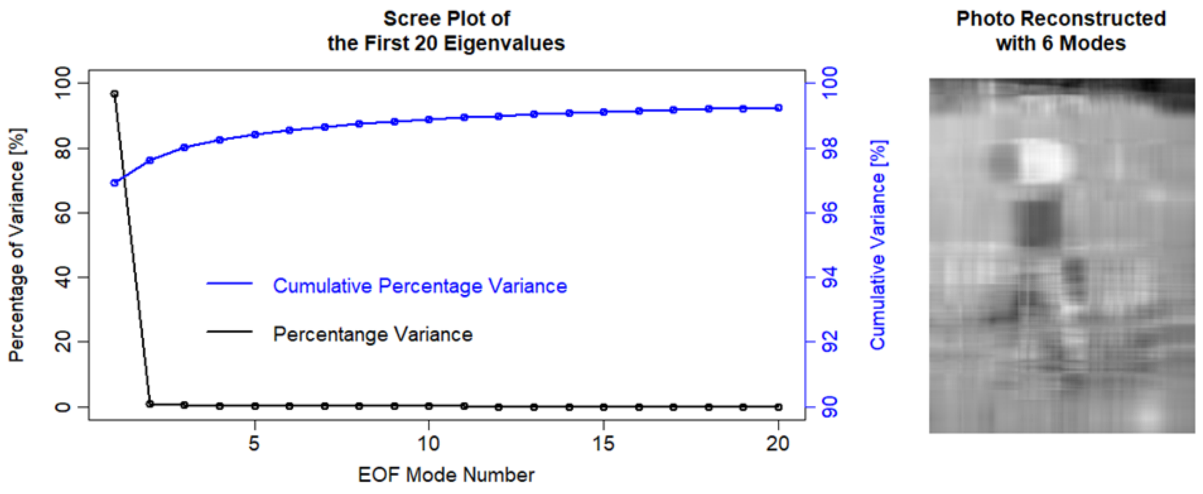
Mode	Variance %	Mode	Variance %	Mode	Variance %	Mode	Variance %
1	96.91	6	0.15	11	0.06	16	0.03
2	0.72	7	0.10	12	0.05	17	0.03
3	0.40	8	0.09	13	0.05	18	0.03
4	0.23	9	0.07	14	0.04	19	0.03
5	0.15	10	0.06	15	0.03	20	0.02

```r
##### SVD analysis of the grayscale data #####
svdDat = svd(graydat)
SVDd = svdDat$d

# calculating the variance of the data
# for the first 20 modes (benchmark)
# which modes contribute the most to the photo?
K = 20
lam = (svdDat$d)^2
lamK=lam[1:K]
lamK

variancePercent = 100*lamK/sum(lam)
round(variancePercent,2)
```

On the scree plot (Figure 3), the line flatlined and showed that the remaining modes did not contribute as much compared to the first six. With the first six eigenvalues contributing the most, the photo was reconstructed using Formula 3 where $k = 6$, $m = 2016$, and $n = 1512$.

Figure 3

Scree Plot of Grayscale Photo's Eigenvalues and Reconstruction

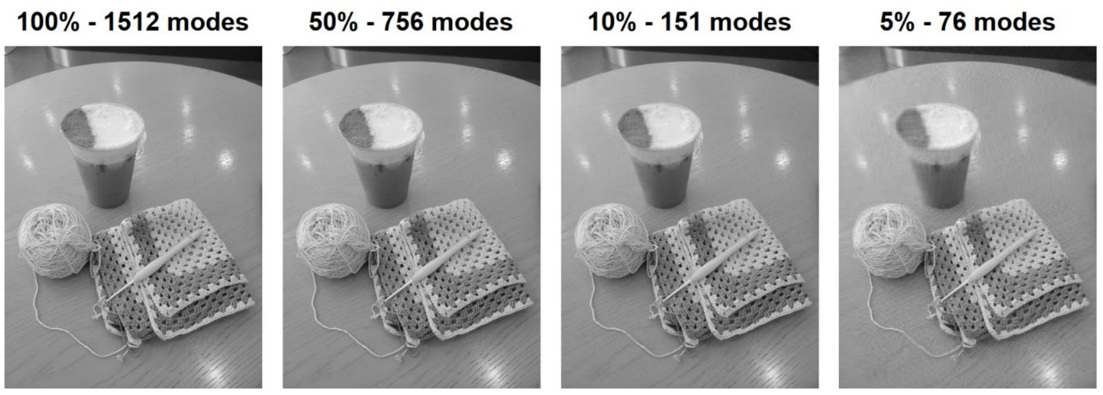


```
##### Figure 3: plotting scree plot with variance and cumulative variance #####
# allows to put the scree plot and photo next to each other
layout(matrix(c(1,2), nrow = 1),widths = c(3,1))

par(mar=c(4,5,4,5), mgp=c(2.2,0.7,0))

# creating the scree plot
plot(1:K, 100*lamK/sum(lam), ylim=c(0,100), type="o",
     ylab="", xlab="EOF Mode Number",
     cex.lab=1.2, cex.axis = 1.2, lwd=2,
     main="Scree Plot of\nthe First 20 Eigenvalues")

# adding axis label and legend for % of variance
mtext("Percentage of Variance [%]",col="black",
      cex=1.1,side=2,line=3)
legend(3,30, col=c("black"),lty=1, lwd=2.0,
       legend=c("Percentange Variance"),bty="n",
       text.font=1,cex=1.2, text.col="black")

# adding cumulative % variance to the plot
par(new=TRUE)
plot(1:K,cumsum(100*lamK/sum(lam)),
     ylim = c(90,100), type="o",
     col="blue",lwd=2, axes=FALSE,
     xlab="",ylab="")

# add axis, lable, and legend
axis(4, col="blue", col.axis="blue", mgp=c(3,0.7,0), cex.axis=1.2)
mtext("Cumulative Variance [%]",col="blue",
      cex=1.1,side=4,line=3)
legend(3,94.5, col=c("blue"),lty=1,lwd=2.0,
       legend=c("Cumulative Percentage Variance"),bty="n",
       text.font=1,cex=1.2, text.col="blue")

# plotting the photo reconstructed with some of data
# from scree plot calculations
par(mar=c(3,1,4,1))

# using only first 6 modes
k6 = 6
R6 = as.cimg(U[,1:k6]%*%D[1:k6, 1:k6]%*%t(V[,1:k6]))
plot(R6, xlim = c(0, row), ylim = c(col,0), axes=FALSE,
     main = "Photo Reconstructed\nwith 6 Modes")
```

Seeing how blurry the reconstruction came out; it was decided that more modes were needed to create an acceptable compressed photo. For the grayscale yarn photo from Figure 2, the compression benchmarks decided on were 100% (original), 50% (medium), 10% (small), and 5% (limit). Figure 4 shows the reconstructed photos with these percentages and number of nodes.

Figure 4

Effects of Different Compression Levels on Yarn Photo



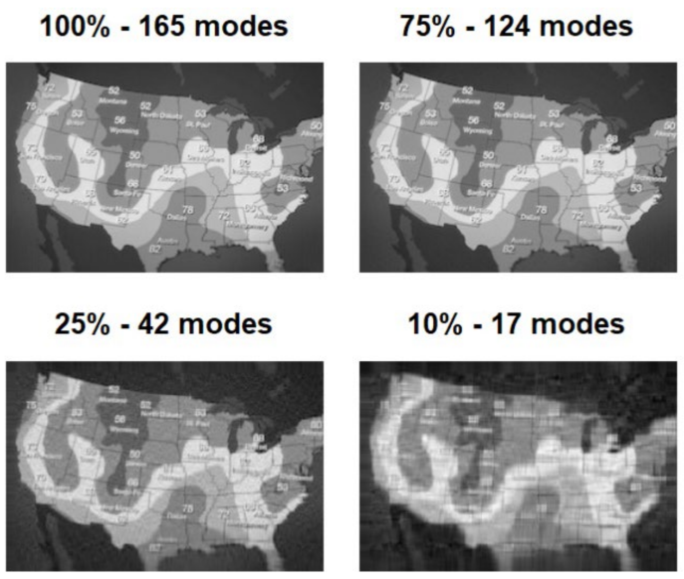
```
##### Figure 4: plotting all 4 reconstructions #####
par(mfrow = c(1,4), mar=c(0.5,1,4,1))

### PROCESS OF CODE BELOW ###
# grabbing the modes
# reconstruct image from the modes
# plot reconstructed image
# add title to the plot
```

```
k100 = modes # 100% recon
R100 = as.cimg(U[,1:k100]%*%D[1:k100, 1:k100]%*%t(V[, 1:k100]))
plot(R100, xlim = c(0, row), ylim = c(col,0), axes=FALSE)
title(main = "100% - 1512 modes",cex.main = 3)

k50 = round(modes/2) # grabbing 50% of modes
R50 = as.cimg(U[,1:k50]%*%D[1:k50, 1:k50]%*%t(V[, 1:k50]))
plot(R50, xlim = c(0, row), ylim = c(col,0), axes=FALSE)
title(main = "50% - 756 modes", cex.main = 3)

k10 = round(modes/10) # grabbing 10% of modes
R10 = as.cimg(U[,1:k10]%*%D[1:k10, 1:k10]%*%t(V[, 1:k10]))
plot(R10, xlim = c(0, row), ylim = c(col,0), axes=FALSE)
title(main = "10% - 151 modes", cex.main = 3)

k5 = round(modes/20) # grabbing 5% of modes
R5 = as.cimg(U[,1:k5]%*%D[1:k5, 1:k5]%*%t(V[, 1:k5]))
plot(R5, xlim = c(0, row), ylim = c(col,0), axes=FALSE)
title(main = "5% - 76 modes", cex.main = 3)
```

Each team member used similar code. The only difference is how many modes/percentages. The formula with the correct numbers worked for all our photos. After our team found their benchmarks, the percentages were compared to one another. My team members' results are below in Figures 5 through 7.

Figure 5

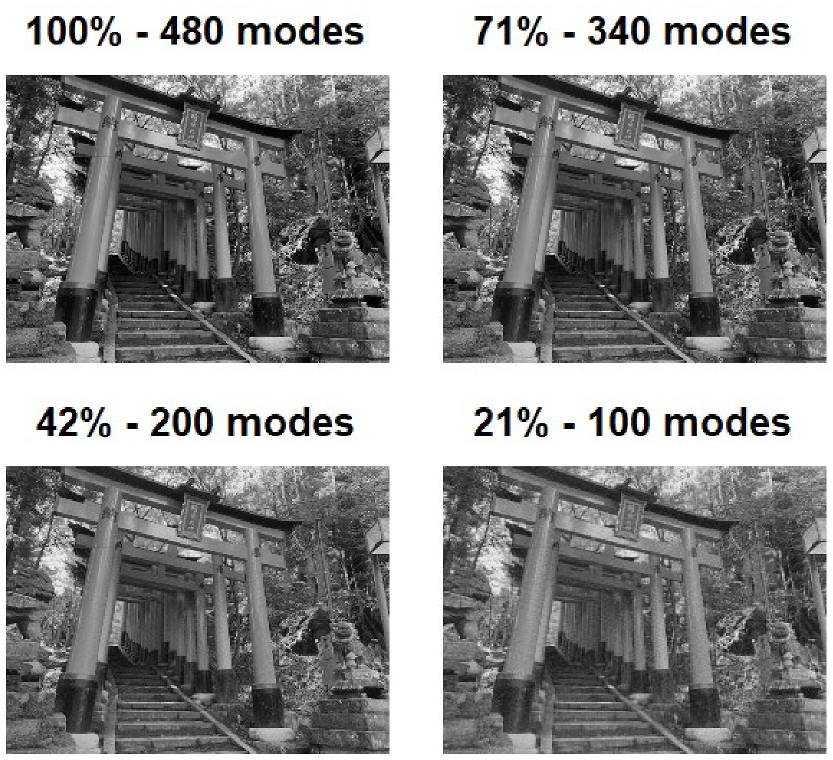
Effects of Different Compression Levels on Weather Photo



```
##### Figure 5: SVD for WEATHER photo and plot reconstructions #####

grayWthr = grayscale(wthr)
svdWthr = svd(grayWthr)

wU = svdWthr$u
wD = diag(svdWthr$d)
wV = svdWthr$v

row = dim(wthr)[1]
col = dim(wthr)[2]

par(mfrow = c(2,2), mar=c(0.5,1,4,1))

w1 = 165
wthr1 = as.cimg(wU[,1:w1]%*%wD[1:w1, 1:w1]%*%t(wV[, 1:w1]))
plot(wthr1, xlim = c(0, row), ylim = c(col,0), axes=FALSE)
title(main = "100% - 165 modes",cex.main = 2)

w2 = 124
wthr2 = as.cimg(wU[,1:w2]%*%wD[1:w2, 1:w2]%*%t(wV[, 1:w2]))
plot(wthr2, xlim = c(0, row), ylim = c(col,0), axes=FALSE)
title(main = "75% - 124 modes",cex.main = 2)

w3 = 42
wthr3 = as.cimg(wU[,1:w3]%*%wD[1:w3, 1:w3]%*%t(wV[, 1:w3]))
plot(wthr3, xlim = c(0, row), ylim = c(col,0), axes=FALSE)
title(main = "25% - 42 modes",cex.main = 2)

w4 = 17
wthr4 = as.cimg(wU[,1:w4]%*%wD[1:w4, 1:w4]%*%t(wV[, 1:w4]))
plot(wthr4, xlim = c(0, row), ylim = c(col,0), axes=FALSE)
title(main = "10% - 17 modes",cex.main = 2)
```

Figure 6

Effects of Different Compression Levels on Gate Photo

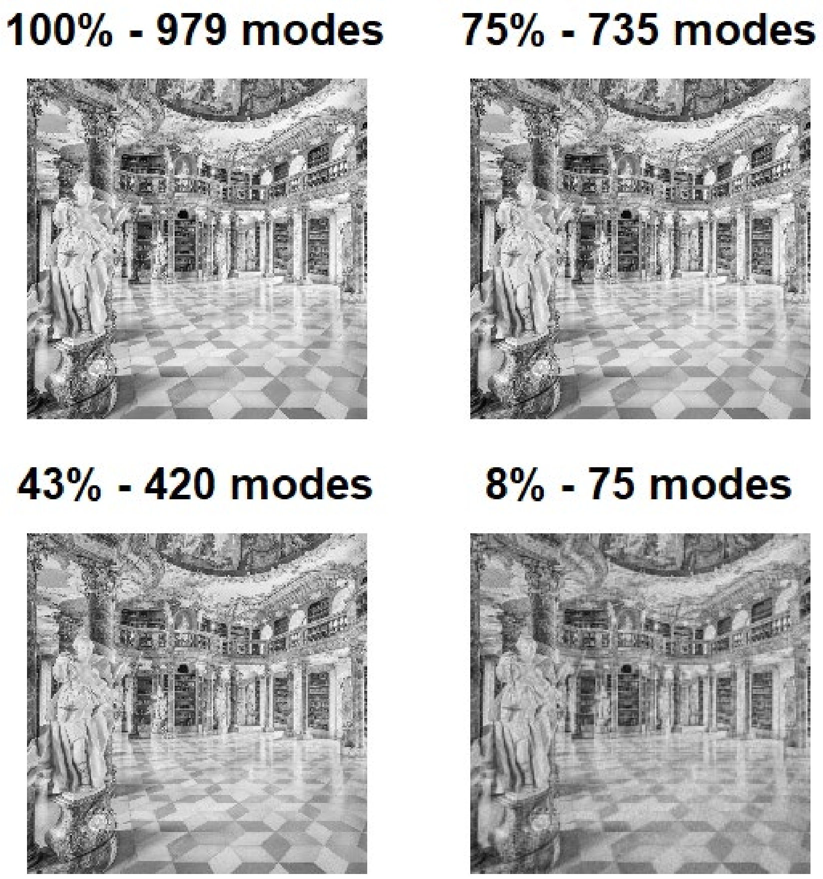


```
##### Figure 6: SVD for GATE photo and plot reconstructions #####

grayGate = grayscale(gate)
svdGate = svd(grayGate)

gU = svdGate$u
gD = diag(svdGate$d)
gV = svdGate$v

row = dim(gate)[1]
col = dim(gate)[2]

par(mfrow = c(2,2), mar=c(0.5,1,4,1))

g1 = 480
gate1 = as.cimg(gU[,1:g1]%*%gD[1:g1, 1:g1]%*%t(gV[, 1:g1]))
plot(gate1, xlim = c(0, row), ylim = c(col,0), axes=FALSE)
title(main = "100% - 480 modes",cex.main = 2)
```

```
g2 = 340
gate2 = as.cimg(gU[,1:g2]%*%gD[1:g2, 1:g2]%*%t(gV[, 1:g2]))
plot(gate2, xlim = c(0, row), ylim = c(col,0), axes=FALSE)
title(main = "71% - 340 modes",cex.main = 2)

g3 = 200
gate3 = as.cimg(gU[,1:g3]%*%gD[1:g3, 1:g3]%*%t(gV[, 1:g3]))
plot(gate3, xlim = c(0, row), ylim = c(col,0), axes=FALSE)
title(main = "42% - 200 modes",cex.main = 2)

g4 = 100
gate4 = as.cimg(gU[,1:g4]%*%gD[1:g4, 1:g4]%*%t(gV[, 1:g4]))
plot(gate4, xlim = c(0, row), ylim = c(col,0), axes=FALSE)
title(main = "21% - 100 modes",cex.main = 2)
```

Figure 7

Effects of Different Compression Levels on Library Photo



```
##### Figure 7: SVD for LIBRARY photo and plot reconstructions #####

grayLbry = grayscale(lbry)
svdLbry = svd(grayLbry)

lU = svdLbry$u
lD = diag(svdLbry$d)
lV = svdLbry$v

row = dim(lbry)[1]
col = dim(lbry)[2]

par(mfrow = c(2,2), mar=c(0.5,1,4,1))

l1 = 979
lbry1 = as.cimg(lU[,1:l1]%*%lD[1:l1, 1:l1]%*%t(lV[, 1:l1]))
plot(lbry1, xlim = c(0, row), ylim = c(col,0), axes=FALSE)
title(main = "100% - 979 modes",cex.main = 2)

l2 = 735
lbry2 = as.cimg(lU[,1:l2]%*%lD[1:l2, 1:l2]%*%t(lV[, 1:l2]))
plot(lbry2, xlim = c(0, row), ylim = c(col,0), axes=FALSE)
title(main = "75% - 735 modes",cex.main = 2)

l3 = 420
lbry3 = as.cimg(lU[,1:l3]%*%lD[1:l3, 1:l3]%*%t(lV[, 1:l3]))
plot(lbry3, xlim = c(0, row), ylim = c(col,0), axes=FALSE)
title(main = "43% - 420 modes",cex.main = 2)

l4 = 75
lbry4 = as.cimg(lU[,1:l4]%*%lD[1:l4, 1:l4]%*%t(lV[, 1:l4]))
plot(lbry4, xlim = c(0, row), ylim = c(col,0), axes=FALSE)
title(main = "8% - 75 modes",cex.main = 2)
```

In Table 2 below, the percentages for the medium, small, and limit benchmarks varied:

- Medium: 50 to 75% (25% difference)
- Small: 10 to 43% (33% difference)
- Limit: 5 to 21% (16% difference)

Table 2

Compression Percentages of Photos from Dataset

Photo	Original Size Modes (%)	Medium Modes (%)	Small Modes (%)	Limit Modes (%)
Weather	165 (100%)	124 (75%)	42 (25%)	17 (10%)
Gate	480 (100%)	340 (≈ 71%)	200 (≈ 42%)	100 (≈ 21%)
Library	979 (100%)	735 (≈ 75%)	420 (≈ 43%)	75 (≈ 8%)
Yarn	1512 (100%)	756 (50%)	151 (10%)	75 (5%)

Note: Photo column's order corresponds with the photos and names in Figure 1.

Based on these results, there are no clear percentages to use as benchmarks in the app's dropdown menu. The dimensions and amount of detail were major factors in what our team members chose for their photo. It helps a lot to understand that not all photos work the same way, and we saw that our team found percentages that worked well for our individual photos. This also told us that specific percentages may not be needed for the app. Instead of having a dropdown menu, a scroll bar can be used to show all the percentages.

Conclusion

As more photos are taken every day, the need for storage grows as well. By developing this app, our team hoped to find ways to compress photos without taking away from the details in them and require less storage. Through the use of Singular Value Decomposition (SVD) in R, we were able to compress sample photos. However, the quality of the results depended on the original size and details, leading to varied results. The only downside is that we need more data and opinions on the compression benchmarks for varying sized photos. With some more research, an alternative could be to create limits depending on the size. Our team can create a table with length range, width range, and maximum compression columns. If research time is an issue, our team can create a survey and ask for volunteers to provide feedback. We can quickly interpret the data with some machine learning. We would then implement code that checks the photos dimensions, finds the ranges the dimensions fit in, and retrieves the maximum compression. This will then output to the user and only allow them to compress up to the limit. This can quickly be added and provide better functionality. If the app needs to be updated, this table can prevent our team from dissecting the entire code. Ultimately, this will save labor costs and quickly bring more value to the app. Lastly, the upside was that the compression process is quick. Users can use it for hundreds or thousands of photos without extensively waiting.

References

Broz, M. (2024, September 12). *How many photos are taken every day?* Photutorial.com. https://photutorial.com/photos-statistics/

Maclennan, A., Fleck, C., Khoshaba, M., & Yabut, J. (2024, December 7). Image Compression Through SVD Version (1.0). Retrieved from https://github.com/jyabut/MATH524Project.

Neokai. (2023, March 30). *How to use Singular Value Decomposition for Image Compression*. Medium. https://medium.com/@sudhanshumukherjeexx/how-to-use-singular-value-decomposition-for-image-compression-7d45882f9f23

Shen, S. (2024). *Interactive Linear Algebra with R and Python*.

Project Code

```r
library(imager)

setwd('C:/Users/hello/LinAlg')
getwd()

## [1] "C:/Users/hello/LinAlg"

##### importing data (photos) #####
lbry <- load.image('library.jpg')
gate <- load.image('gate.jpg')
wthr <- load.image('weather.jpg')
yarn <- load.image('yarn.jpg')

##### Figure 1: creating a chart of all the photos to put in paper #####
layout(matrix(c(1,2,3,4), nrow = 1),widths = c(33,29,22,16))
par(mar=c(1,0.5,8,0.5))

plot(wthr, axes=FALSE)
title(main="Weather\n240 x 165 pixels",cex.main=2)

plot(gate, axes=FALSE)
title(main="Gate\n640 x 480 pixels",cex.main=2)

plot(lbry, axes=FALSE)
title(main="Library\n980 x 979 pixels",cex.main=2)

plot(yarn, axes=FALSE)
title(main="Yarn\n1512 x 2016 pixels",cex.main=2)

dev.off()

##### get dimensions to calculate modes for percentages #####
# the matrices' sizes depend on the smaller dimension
dims <- dim(yarn)
row = dims[1]
col = dims[2]

if (row <= col) {
  modes = row
}else {
  modes = col
}

##### Make the photo data black-and-white #####
graydat = grayscale(yarn)

##### Figure 2: Showing Colored vs Grayscale #####

par(mfrow=c(1,2), mar=c(1,1,4,1))

plot(yarn, xlim = c(0, row), ylim = c(col,0), axes = FALSE,
     main = 'Yarn Photo in Color\n(3 channels)')
```

```
plot(graydat, xlim = c(0, row), ylim = c(col,0), axes = FALSE,
     main = 'Yarn Photo in Grayscale\n(1 channel)')

dev.off()

##### SVD analysis of the grayscale data #####
svdDat = svd(graydat)
SVDd = svdDat$d

# calculating the variance of the data
# for the first 20 modes (benchmark)
# which modes contribute the most to the photo?
K = 20
lam = (svdDat$d)^2
lamK=lam[1:K]
lamK

##  [1] 1076574.8851    8041.9880    4416.1034    2585.8575    1721.5428
##  [6]    1622.0220    1069.7657    1043.2345     774.2973     688.4329
## [11]     642.7652     544.9433     512.6458     429.7089     363.5073
## [16]     350.7314     326.3880     301.9426     289.2680     273.5275

variancePercent = 100*lamK/sum(lam)
round(variancePercent,2)

##  [1] 96.91  0.72  0.40  0.23  0.15  0.15  0.10  0.09  0.07  0.06  0.06  0.05
## [13]  0.05  0.04  0.03  0.03  0.03  0.03  0.03  0.02

# Accessing SVD matrices #
U = svdDat$u
D = diag(svdDat$d)
V = svdDat$v

dim(U)

## [1] 1512 1512

dim(D)

## [1] 1512 1512

dim(V)

## [1] 2016 1512

##### Figure 3: plotting scree plot with variance and cumulative variance #####
# allows to put the scree plot and photo next to each other
layout(matrix(c(1,2), nrow = 1),widths = c(3,1))

par(mar=c(4,5,4,5), mgp=c(2.2,0.7,0))

# creating the scree plot
plot(1:K, 100*lamK/sum(lam), ylim=c(0,100), type="o",
     ylab="", xlab="EOF Mode Number",
     cex.lab=1.2, cex.axis = 1.2, lwd=2,
     main="Scree Plot of\nthe First 20 Eigenvalues")
```

```
# adding axis label and legend for % of variance
mtext("Percentage of Variance [%]",col="black",
      cex=1.1,side=2,line=3)
legend(3,30, col=c("black"),lty=1, lwd=2.0,
       legend=c("Percentange Variance"),bty="n",
       text.font=1,cex=1.2, text.col="black")

# adding cumulative % variance to the plot
par(new=TRUE)
plot(1:K,cumsum(100*lamK/sum(lam)),
     ylim = c(90,100), type="o",
     col="blue",lwd=2, axes=FALSE,
     xlab="",ylab="")

# add axis, lable, and legend
axis(4, col="blue", col.axis="blue", mgp=c(3,0.7,0), cex.axis=1.2)
mtext("Cumulative Variance [%]",col="blue",
      cex=1.1,side=4,line=3)
legend(3,94.5, col=c("blue"),lty=1,lwd=2.0,
       legend=c("Cumulative Percentage Variance"),bty="n",
       text.font=1,cex=1.2, text.col="blue")

# plotting the photo reconstructed with some of data
# from scree plot calculations
par(mar=c(3,1,4,1))

# using only first 6 modes
k6 = 6
R6 = as.cimg(U[,1:k6]%*%D[1:k6, 1:k6]%*%t(V[,1:k6]))
plot(R6, xlim = c(0, row), ylim = c(col,0), axes=FALSE,
     main = "Photo Reconstructed\nwith 6 Modes")

dev.off()

##### Figure 4: plotting all 4 reconstructions #####
par(mfrow = c(1,4), mar=c(0.5,1,4,1))

### PROCESS OF CODE BELOW ###
# grabbing the modes
# reconstruct image from the modes
# plot reconstructed image
# add title to the plot

k100 = modes # 100% recon
R100 = as.cimg(U[,1:k100]%*%D[1:k100, 1:k100]%*%t(V[, 1:k100]))
plot(R100, xlim = c(0, row), ylim = c(col,0), axes=FALSE)
title(main = "100% - 1512 modes",cex.main = 3)

k50 = round(modes/2) # grabbing 50% of modes
R50 = as.cimg(U[,1:k50]%*%D[1:k50, 1:k50]%*%t(V[, 1:k50]))
plot(R50, xlim = c(0, row), ylim = c(col,0), axes=FALSE)
title(main = "50% - 756 modes", cex.main = 3)

k10 = round(modes/10) # grabbing 10% of modes
```

```
R10 = as.cimg(U[,1:k10]%*%D[1:k10, 1:k10]%*%t(V[, 1:k10]))
plot(R10, xlim = c(0, row), ylim = c(col,0), axes=FALSE)
title(main = "10% - 151 modes", cex.main = 3)

k5 = round(modes/20) # grabbing 5% of modes
R5 = as.cimg(U[,1:k5]%*%D[1:k5, 1:k5]%*%t(V[, 1:k5]))
plot(R5, xlim = c(0, row), ylim = c(col,0), axes=FALSE)
title(main = "5% - 76 modes", cex.main = 3)

dev.off()

##### Figure 5: SVD for WEATHER photo and plot reconstructions #####

grayWthr = grayscale(wthr)
svdWthr = svd(grayWthr)

wU = svdWthr$u
wD = diag(svdWthr$d)
wV = svdWthr$v

row = dim(wthr)[1]
col = dim(wthr)[2]

par(mfrow = c(2,2), mar=c(0.5,1,4,1))

w1 = 165
wthr1 = as.cimg(wU[,1:w1]%*%wD[1:w1, 1:w1]%*%t(wV[, 1:w1]))
plot(wthr1, xlim = c(0, row), ylim = c(col,0), axes=FALSE)
title(main = "100% - 165 modes",cex.main = 2)

w2 = 124
wthr2 = as.cimg(wU[,1:w2]%*%wD[1:w2, 1:w2]%*%t(wV[, 1:w2]))
plot(wthr2, xlim = c(0, row), ylim = c(col,0), axes=FALSE)
title(main = "75% - 124 modes",cex.main = 2)

w3 = 42
wthr3 = as.cimg(wU[,1:w3]%*%wD[1:w3, 1:w3]%*%t(wV[, 1:w3]))
plot(wthr3, xlim = c(0, row), ylim = c(col,0), axes=FALSE)
title(main = "25% - 42 modes",cex.main = 2)

w4 = 17
wthr4 = as.cimg(wU[,1:w4]%*%wD[1:w4, 1:w4]%*%t(wV[, 1:w4]))
plot(wthr4, xlim = c(0, row), ylim = c(col,0), axes=FALSE)
title(main = "10% - 17 modes",cex.main = 2)

dev.off()

##### Figure 6: SVD for GATE photo and plot reconstructions #####

grayGate = grayscale(gate)
svdGate = svd(grayGate)
```

```
gU = svdGate$u
gD = diag(svdGate$d)
gV = svdGate$v

row = dim(gate)[1]
col = dim(gate)[2]

par(mfrow = c(2,2), mar=c(0.5,1,4,1))

g1 = 480
gate1 = as.cimg(gU[,1:g1]%*%gD[1:g1, 1:g1]%*%t(gV[, 1:g1]))
plot(gate1, xlim = c(0, row), ylim = c(col,0), axes=FALSE)
title(main = "100% - 480 modes",cex.main = 2)

g2 = 340
gate2 = as.cimg(gU[,1:g2]%*%gD[1:g2, 1:g2]%*%t(gV[, 1:g2]))
plot(gate2, xlim = c(0, row), ylim = c(col,0), axes=FALSE)
title(main = "71% - 340 modes",cex.main = 2)

g3 = 200
gate3 = as.cimg(gU[,1:g3]%*%gD[1:g3, 1:g3]%*%t(gV[, 1:g3]))
plot(gate3, xlim = c(0, row), ylim = c(col,0), axes=FALSE)
title(main = "42% - 200 modes",cex.main = 2)

g4 = 100
gate4 = as.cimg(gU[,1:g4]%*%gD[1:g4, 1:g4]%*%t(gV[, 1:g4]))
plot(gate4, xlim = c(0, row), ylim = c(col,0), axes=FALSE)
title(main = "21% - 100 modes",cex.main = 2)

dev.off()

##### Figure 7: SVD for LIBRARY photo and plot reconstructions #####

grayLbry = grayscale(lbry)
svdLbry = svd(grayLbry)

lU = svdLbry$u
lD = diag(svdLbry$d)
lV = svdLbry$v

row = dim(lbry)[1]
col = dim(lbry)[2]

par(mfrow = c(2,2), mar=c(0.5,1,4,1))

l1 = 979
lbry1 = as.cimg(lU[,1:l1]%*%lD[1:l1, 1:l1]%*%t(lV[, 1:l1]))
plot(lbry1, xlim = c(0, row), ylim = c(col,0), axes=FALSE)
title(main = "100% - 979 modes",cex.main = 2)
```

```r
l2 = 735
lbry2 = as.cimg(lU[,1:l2]%*%lD[1:l2, 1:l2]%*%t(lV[, 1:l2]))
plot(lbry2, xlim = c(0, row), ylim = c(col,0), axes=FALSE)
title(main = "75% - 735 modes",cex.main = 2)

l3 = 420
lbry3 = as.cimg(lU[,1:l3]%*%lD[1:l3, 1:l3]%*%t(lV[, 1:l3]))
plot(lbry3, xlim = c(0, row), ylim = c(col,0), axes=FALSE)
title(main = "43% - 420 modes",cex.main = 2)

l4 = 75
lbry4 = as.cimg(lU[,1:l4]%*%lD[1:l4, 1:l4]%*%t(lV[, 1:l4]))
plot(lbry4, xlim = c(0, row), ylim = c(col,0), axes=FALSE)
title(main = "8% - 75 modes",cex.main = 2)

dev.off()
```

Cole Fleck: Peer Review for Jasmine Yabut

General Notes: I'd recommend indenting the start of each paragraph, otherwise the layout is all good! Also this is personal preference but when you talk about what we did don't be afraid to say stuff like "our team" and "we" as it helps get across that this is all of our work! It's personal preference though and saying "the team" works perfectly too!

Title Page: Everything looks great and everything seems to be included! Great header and title too!

Abstract: Great summarization of the most important stuff, very to the point! I feel that the sentence "Due to the varying sizes…" could be made a bit more clear as it was a bit difficult for me to understand. Maybe something like "Due to varying amounts of detail, not all photos can be reliably compressed to the same level, occasionally coming down to personal preference."

Introduction: Great hook and great statistics! Really helps emphasize how important photos are in our lives! Could change "This is almost…" to something like "The number of photos taken each year has been growing at an incredible rate, just about tripling since 2013". Change "and 94%" to "with 94%" possibly? Really like how you brought up storage and introduced the pitch for the app! The "If only some…" is a bit awkward. My suggestion would be to merge it with the next sentence like "Moreover, by only using some of our rows and columns it's possible to create a variation of our matrix that has around the same amount of detail, but with less data, leading to less required storage" but that's a bit of a run-on so I don't know! Great stuff though! Loved your explanation of SVD so far!

Data and Methods: Great use of formulas, I like how you establish everything early on and with great detail! "Through R programming…" feels a bit awkward, maybe "In R studio, our image would be represented by the matrix A, with matrices U and V providing information about the photo, and matrix D telling us how that information affects our photo.” Typo with ‘effect’, should be ‘effects’ or ‘affects’, I don’t know the difference lol! Typo with “medium small”, instead should be “medium and small” I think! For the “All the photos…” sentence I feel like this would be a good chance to talk about the photo’s dimensions and what they represent! Could change “different amounts of k modes” to “different values of k” as you did a great job establishing what k represents! Otherwise everything in this last section looks good! Especially with the discussion about what we did!

Results: Overall everything is just really really good, great figures and tables and such! Also though this is completely just personal preference but for the last sentence I feel that all the photos having differing percentage sizes shows that compression differs from photo to photo, and that there may not be an exact percentage that needs to be found ^_^ I feel like that may be a nice way to end the results section on a high note as I feel our team did a great job choosing percentages!

Conclusion: I’d suggest merging the second and third sentences maybe! For better flow change “By using” to “Through the use of” too. I like your perspective on the downsides! I feel that the “An alternative…” sentence could be moved somewhere else. Maybe combine the two sentences surrounding that one to talk about how with more data our team could’ve gained more insight or something? Great conclusion though! I like how much stuff you went over!!!!