SVD Based Approximation of Digital Images Report

Joseph Opitz & Angel Sanchez

M 491: Linear Algebra for Data Science and Engineering

Montana Technological University

11/24/24

Contents

1 Abstract

We will discuss how we can compress images using Singular Value Decomposition for grayscale and Red Green Blue (RGB) color scale. We are exploring the benefits of obtaining low-dimensional rank approximations for very high-dimensional data, in our case, by keeping key elements/features for large image matrices while reducing the size of the image.

2 Project Background

Images are stored in matrices with the corresponding pixel values. These matrices tend to be very large, which means that they require a lot of memory to store. Because of this, they can also be very computationally expensive to run algorithms on. However, many images contain overlapping information, also known as redundancies. Using these redundancies, you can compress images into smaller matrices better. Doing this allows you to run algorithms more efficiently on the images and save storage space. This project aims to obtain low-dimensional approximations to very high-dimensional data, in this case, picture elements with the image. An example is shown below with **Figure 1**.

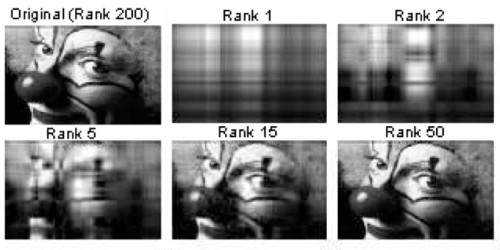

Figure 1: Example of Image Compression

In this figure, the original image has a rank of 200. By approximating the rank a couple of times the same exact original image can be produced with a much less reduced rank size. Displaying a good visual application of SVD in digital images.

3 Mathematical Background

Singular value decomposition (SVD) is a unique matrix that exists for any matrix X m-by-n that is real or complex where

$$X = U\Sigma V^T$$

All variables in the equation have different properties, where U is an m-by-m square orthogonal matrix, V^T is an n-by-n square orthogonal matrix and Σ is an m-by-n diagonal matrix with nonnegative diagonal entries. These entries of Σ are called the singular values and they order from the highest value to the lowest value diagonally, all other entries being 0.

To obtain the singular values in Σ, you must first find your eigenvalues (λ) using either of the following formulas. It does not matter which

$$\lambda = XX^T$$

$$\lambda = X^T X$$

You can use either one since both eigenvalues have the same rank and span the same nonzero eigenspace. We can then find the singular values by taking the square root of the found eigenvalues.

$$\sigma_i = \sqrt{\lambda_i}$$

These singular values will be arranged in descending order.

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\min\{m,n\}} \geq 0$$

Left singular vectors are obtained from finding eigenvectors of XX^T and right singular vectors are obtained from finding eigenvectors of $X^T X$. These vectors correspond to the columns of the matrices U and V in the singular decomposition of X. A depiction of the variables in matrix form can be seen in **Figure 2** below.

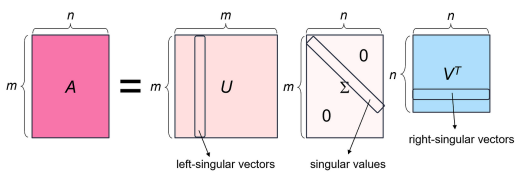

Figure 2: SVD equation with matrix representation of each variable.

Basically, left singular values explain how the columns of the matrix X are transformed by the singular values and the right singular vectors explain how the rows of matrix X are transferred by singular values. The rank of matrix X is equal to the number of its non-zero singular values. However, if the matrix X is not full to begin with, some singular values can still be zero. The magnitude of the singular values decreases so rapidly that only the first few singular values contain much of the information of X. Singular values are unique, but U and V are not unique.

4 Image to Matrix

Images are represented by a single matrix or an array, with each element of the matrix corresponding to one image pixel. These are square pixels arranged in columns and rows that represent picture details within the digital image. There are two different types of scales that the image follows. One being a grayscale and the other being a colorscale.

4.1 Grayscale

A grayscale image has an m-by-n matrix where m and n are vertical and horizontal pixel directions. Each pixel contains a grayscale value meaning it can range from 0 being black to 1 being white with every number in between those two being a shade of gray. An example of this can be seen in **Figure 3** below.

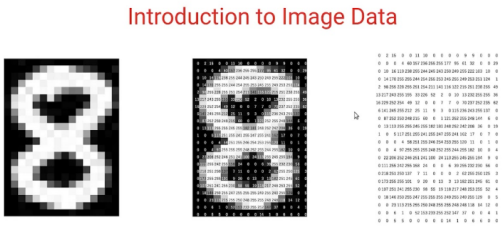

Figure 3: Diagram of the number 8 digital image being converted to matrix with a grayscale

Here the image is in grayscale however the parameters for each pixel can range from 0 (black) to 255 (white) and all other numbers in between being

a shade of gray. This is the same concept as 0 being black and 1 being white just with different number values. The code representation of this concept is shown below.

```
from skimage.io import imread
from skimage.color import rgb2gray

img = imread('flower.jpg')
print(f'A = \n{A}')
```

Which outputs:

```
    A =
[[0.22083843 0.21635137 0.20678902 ... 0.26594353 0.2816298  0.2816298 ]
 [0.22083843 0.21635137 0.20678902 ... 0.26594353 0.27378667 0.27378667]
 [0.21831569 0.21635137 0.20678902 ... 0.26677686 0.27069843 0.27069843]
 ...
 [0.23595922 0.2124298  0.20009961 ... 0.27411137 0.25506902 0.27859843]
 [0.27320275 0.23790863 0.21578588 ... 0.27467686 0.32341725 0.35086824]
 [0.30065373 0.25751647 0.22362902 ... 0.27467686 0.29988784 0.33518196]]
 [0.30065373 0.25751647 0.22362902 ... 0.27467686 0.29988784 0.33518196]]
```

4.2 Colorscale

Colorscale images are like grayscale images when it comes to being square pixels arranged in columns and rows that represent picture details within the digital image. A colorscale image has an m-by-n matrix where m and n are vertical and horizontal pixel directions as well. However, in this one each pixel contains a color value, and each pixel represents a vector of three numbers. The vectors represent a color from the red, green, and blue colorscale. The number inputs for each vector ranges from 0 to 255. For example (255, 0, 0) is the color red. A visual of the RBG colorscale can be seen in **Figure 4** below.

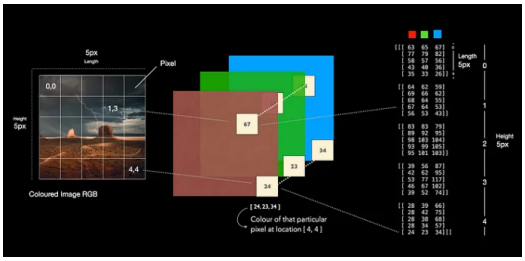

Figure 4: Diagram of a scenic digital image represented by the RGB colorscale

Each color channel has its own number inputted for the corresponding spot in the image to represent a color. Essentially the red channel has its own matrix, the green channel has its own matrix, and the blue channel has its own matrix. These are all then combined to produce a colorscale digital image.

A demonstration for how these channels are sliced in code is shown below in **Figure 5**:

```
A = imread('flower.jpg')

R = A[:, :, 0]
G = A[:, :, 1]
B = A[:, :, 2]
```

Which takes the original image and displays the following channels shown below in **Figure 5**:

Figure 5: Demonstration of RGB channels

5 Relative Error vs Rank Approximation

5.1 Rank Approximation

So far we have explained how to perform SVD, but not how to make approximations on the data in order to reduce the images, which is where rank-r approximation comes in. So, given matrix X, upon performing SVD, we are given the right (V^T) and left (U) Singular Vectors and singular values (Σ) that we mentioned previously. We are then able to calculate our energy matrix, which will help in finding our best r approximation.

$$E_r = \frac{\sum_{i=1}^{k} \sigma_i^2}{\sum_{i=1}^{n} \sigma_i^2}$$

Using this energy function, we will get a matrix of values that we can test against a certain threshold. The threshold for the equation below is 95%.

$$r = \min\{r : E_r \geq 0.95\}$$

This is what will give us the rank approximation (r) that will be used for compressing the image, as we will only keep the first "r" values in our singular vectors and singular values. In this case, we will retain 95% of the cumulative energy from the original matrix.

5.2 Relative Error

Once we recompress our image using our rank-r approximation, we have to have a way to effectively evaluate how well it performed. This is where the relative error function comes in, the equation for calculating this is shown below.

$$e(r) = \frac{\|X_r - X\|}{\|X\|}$$

Using this relative error function, we can effectively evaluate how well our image reconstruction was performed.

5.3 Comparison

The more that we increase the threshold, the better the rank approximation of the image will be. A graph demonstrating the comparison between relative

error and rank-r approximation is shown in **Figure 6**.

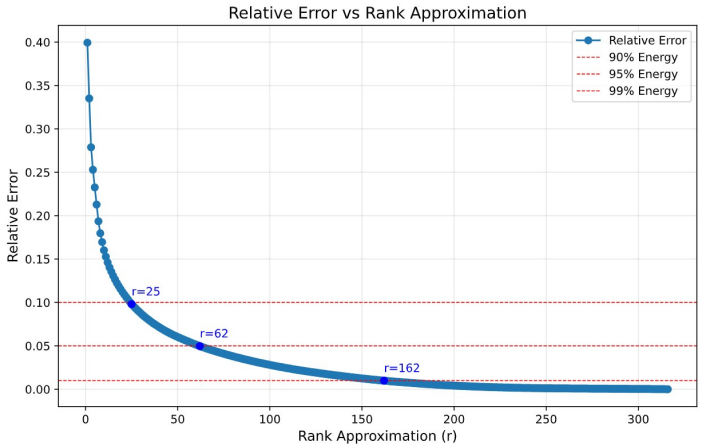

Figure 6: Graph demonstrating Rank Approximation vs Relative Error

However, we want to be careful to avoid becoming too obsessed with our relative error. The reason for this is that there is a point called the elbow point, demonstrated for this graph with the 95% threshold. The elbow point is basically the point on the graph where we retain enough of the information to still recognize it. But, not too much to where it no longer truly compresses the image. An image comparing all the different thresholds and r values are shown below in **Figure 7**.

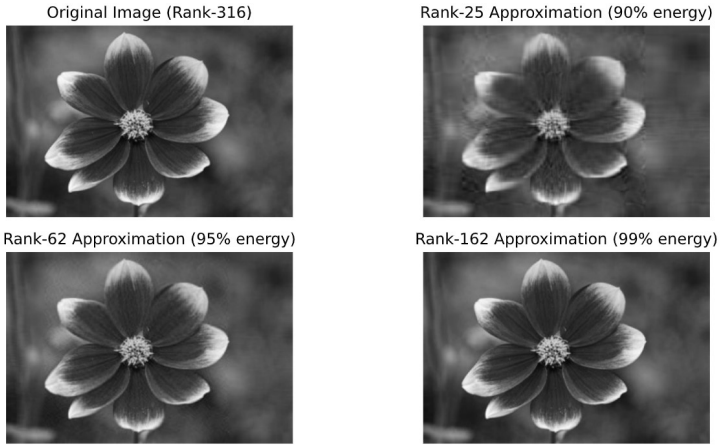

Figure 7: Images demonstrating Rank Approximation vs Relative Error

6 Using SVD to Approximate the Image Matrix to Reduce Size

6.1 Algorithms

A notable property of SVD, is it allows one to write the input matrix as a sum of rank-1 matrices. Specifically, the matrix X can be written as the following seen below.

$$X = \sum_{k=1}^{\min\{m,n\}} \sigma_k u_k v_k^T$$

Where σ_k is the kth diagonal entry of Σ, u_k and v_k are the kth columns of U and V. Assuming the singular values are distinct. Matrix X is expressed a sum of rank one matrix that are orthogonal with respect to the matric inner product. Each rank-1 matrix captures a distinct component of the matrix X. The Frobenius norm is used for the relative error which is the square root of the sum of the square differences between the original and the reduced images squared divided by the square root of the original matrix. The formula can be seen below.

$$\frac{|E_r|}{|A_r|} = \sqrt{\frac{\sum_{i=r+1}^{k} \sigma_i^2}{\sum_{i=1}^{k} \sigma_i^2}}$$

As "r" increases, the error decreases, but the matrix size increases. There is a point where increasing "r" provides diminishing returns in terms of reducing the error. Therefore, an optimal rank-r must be chosen to balance between error reduction and size efficiency.

6.1.1 Greedy Algorithm for Rank-r Approximation

The rank-one approximation means trying to represent the matrix as a sum of one matrix that has a rank of 1. Instead of just using a rank-one approximation, the approximation can be improved by including more matrices. The idea is to first choose the best possible rank-one matrix that approximates the original matrix. Then, take the leftover error and find the best rank-one approximation for it. This process is repeated each time choosing the best approximation for the remaining errors and adding them together.

This is referred to as a greedy algorithm because at each step, it tries to capture the largest possible chunk of the matrix's structure. After repeating this process for "r" steps, there is an approximation that has rank "r." If it keeps going, eventually, the error will be reduced to zero and goes back to the full SVD of the matrix. The SVD can be written as a sum of rank-one matrices that are multiplied together in a specific way. This allows for an approximation of the matrix using fewer components (lower rank) while still capturing the most important information. The algorithm used can be seen below.

6.2 Process

Those are the algorithms behind using SVD to approximate an image matrix to reduce size and here is the step-by-step process of what is actually happening. To reduce the size of the image while retaining its structure, approximate the matrix by truncating its Singular Value Decomposition to a lower rank. This process decomposes the image into its most important components, and then approximates the original image using fewer components, achieving compression while preserving key features. Each variable within the SVD equation represents something different within a digital image. Where the columns of U capture the directions in the image where the information is most concentrated, the columns of V capture the details or features of the image in those directions, and Σ measures the importance or strength of each feature. Large values represent important details, and small values are less significant. These singular values will order themselves from largest to greatest as SVD guarantees below. Recall that SVD arranges the sigular values in Σ in their decreasing order of importance:

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\min\{m,n\}} \geq 0$$

Relative error comes into play by minimizing when the largest singular values are retained, and small values are ignored. This error represents the "energy" threshold of the ignored singular values. In other words, it captures how much information was lost by truncating the SVD. For example, an energy threshold of 90% gives a relative error of 10% for singular values to abide by otherwise they are discarded.

7 Image Recompression

Upon performing SVD and reducing all of our singular vectors, along with our singular values. We must now recontruct our images that have been reduced using our rank-r approximation. We will be demonstrating how to do this with both grayscale and Red Green Blue (RGB) images.

7.1 Grayscale Images

For simplicity, we will start off with our grayscale images. Using our reduced singular vectors, U_r and V_r^T, as well as our reduced singular values, Σ_r. We now perform the dot product to get our reduced image matrix, X_r.

$$X_r = U_r \cdot \Sigma_r \cdot V_r^T$$

Using that formula, we now have our rank-r approximation compressed image X_r. The way that this looks in code is shown below.

```
    # Keep only the first r components
    Ur = U[:, :r]
    Sr = np.diag(S[:r])
    VTr = VT[:r, :]

    # Reconstruct the rank-r approximation
    compressed = Ur @ Sr @ VTr
```

Which gives you the following output, shown below in **Figure 8**:

Figure 8: Compressing a grayscale image

7.2 RGB Images

Image recompression gets more complicated when we are working with RGB images. As mentioned earlier, we had to split the image up into three seperate

channels-one red (R), one green (G), and one blue(B). So, we will have to recompress each reduced channel individually, which is demonstrated below.

$$R_r = U_{Rr} \cdot \Sigma_{Rr} \cdot V_{Rr}^T$$
$$G_r = U_{Gr} \cdot \Sigma_{Gr} \cdot V_{Gr}^T$$
$$B_r = U_{Br} \cdot \Sigma_{Br} \cdot V_{Br}^T$$

We will then want to combine these reduced channels back into their original channel, shown below.

$$\text{compressed\_image}(x, y, 0) = R_r(x, y)$$
$$\text{compressed\_image}(x, y, 1) = G_r(x, y)$$
$$\text{compressed\_image}(x, y, 2) = B_r(x, y)$$

This operation maps each color component R_r(x,y), G_r(x,y), and B_r(x,y) to the corresponding indices of the compressed image at pixel (x,y) for channels 0, 1, and 2, respectively. An example of how the image is split up into channels and compressed and then recombined back into the original image.

The code is demonstrated below:

```
    # Re-compress each channel using the maximum rank
    red_comp = compress_channels(red_image, r)
    green_comp = compress_channels(green_image, r)
    blue_comp = compress_channels(blue_image, r)

    # Create a temporary array to hold the new found
       channels
    comp_image = np.zeros_like(image, dtype=np.float32)
    comp_image[:, :, 0] = red_comp
    comp_image[:, :, 1] = green_comp
    comp_image[:, :, 2] = blue_comp
```

Which shows the image as follows, shown below in **Figure 9**:

Figure 9: Image showing three channels and compressed image

8 Principal Component Analysis vs SVD

Principal Component Analysis (PCA) is one of the main applications of SVD and follows many of the same steps. we still start with the X matrix that is still m-by-n. The principal components provide an orthogonal coordinate system regarding the mean data. These components can then be used to reveal the maximum variation within the data.

8.1 Calculations

To find the principal components, we must first find the mean for each row (feature). Which gives we a 1-by-n row vector $\bar{x}$ of mean values for each feature.

$$\bar{x}_j = \frac{1}{n}\sum_{i=1}^{m} x_{ij}$$

Once we find the means of all features within the matrix, we can then find the mean matrix - $\bar{X}$. This is found by multiplying $\bar{x}$ by a column vector of ones with size m-by-1, which will give us our m-by-n mean matrix.

$$\bar{X} = 1_{m\times 1}\bar{x}$$

We then subtract the mean matrix $barX$ from the data matrix X to get our mean-subtracted matrix.

$$B = \bar{X} - X$$

SVD is then applied to this mean-subtracted matrix.

$$B = U\Sigma V^T$$

This is another point where SVD and PCA differ, now we have to find our principal values using the following equation:

$$\frac{\sigma_k^2}{m-1}$$

The corresponding columns of V are called the principal components, which provide an orthogonal coordinate system for the data, otherwise known as the variances of the data.

9 Money and Time

Considering how powerful the SVD technique is, it can be used to save money and time for many different applications in various fields. First, by compressing images using SVD, it reduces the amount of data needed to Store them. Smaller files mean lower storage costs. This is specifically beneficial to the medical and engineering fields. Another way SVD saves money and time is that compressed images are sent quicker across networks in industries. Again, this is beneficial for areas such as telemedicine. Lastly, by using SVD for image compression, image processing becomes quicker with less data to compute which means less computational costs. This is important for surveillance footage of people, or license plates from afar.

10 Conclusion

SVD is a popular factorization technique used in linear algebra. Mainly it is used in the science and engineering fields. It is a powerful method for compressing image matrices by reducing their rank. By keeping the most significant singular values (and the corresponding vectors from U and V), we can create a low-rank approximation that retains the essential features of the image while dramatically reducing its size. The error in this approximation can be controlled by selecting an appropriate rank-r and balancing compression with quality.

Possible future work can include how your iPhone screen unlocks using important features from your face. As well as how self-driving cars efficiency and overall success could be improved through this technique. Self-driving cars usually analyze around 30 to 60 frames per second, so having these images be compressed will greatly increase their ability to make quick real time decisions.

11 References

1. Mitra A. and Majhi S. *Linear Algebra for Data Science and Engineering*. `https://linalg.mathematics.land`. Accessed: November 2024. Nov. 2024.

2. D. Kalman. "A Singularly Valuable Decomposition: The SVD of a Matrix". In: *The College Mathematics Journal* 27.1 (Feb. 2002), pp. 2–23. Accessed: November 29, 2024.

3. A. Ranade, S. Mahabalarao, and S. Kale. "A Variation on SVD Based Image Compression". In: *Image and Vision Computing* (June 2007). Accessed: December 1, 2024.

4. P. Bacher. *Practical Applications to SVD on RBG Images*. `https://www.kaggle.com`. Accessed: December 3, 2024. Oct. 2022.