Analysis and Implementation of SVD

Author: Theodore Li

November 19, 2024

Reproducibility

This work was done on a MSI Summit laptop using 13th Gen Intel Core i7 with 32 GB main memory. This computer is running on Windows 11 and was executing on VSCODE version 1.95.2
The coding segments were executing on Python 3.9.19. Here are the following libraries and the versions used in python:
pillow: 10.4.0
numpy: 2.1.3
matplotlib: 3.9.0
pandas: 2.2.0
scipy: 1.11.4

Introduction

This report is meant to analyze the use of singular value decomposition and its mathematical properties. We will begin by first analyzing the structure of SVD for a matrix and how they can be deconstructed. Next we will move on to more practical applications of SVD specifically for image processing, through the use of Python.

In brief, SVD is to decronstruct a matrix into three smaller matrices consisting of U in $\mathbb{R}^{n\times n}$, S in $\mathbb{R}^{n\times m}$, and V in $\mathbb{R}^{m\times m}$ such that we can obtain the matrix A as the matrix product of USV^T.

A specific property to keep in mind is that S takes on different values depending on the values of n and m. S has the form,

$$\begin{pmatrix}\Sigma\\0\end{pmatrix} \text{ if n} \geq \text{m},$$

$$[\Sigma \;\; 0] \text{ if n} < \text{m}.$$

Example 1

Let matrix A be in $\mathbb{R}^{n\times m}$ and assume $m \geq n$. By SVD we know we can rewrite A as USV^T.
Where U is in $\mathbb{R}^{n\times n}$, S is in $\mathbb{R}^{n\times m}$, and V is in $\mathbb{R}^{m\times m}$.
And S is the diagonal matrix of singular values $\sigma_1, \ldots, \sigma_{\min(n,m)}$.

Since we are assuming $m \geq n$, let's break it down into separate cases to explore all possibilities. Case 1: m = n:

By definition on the construction of S in SVD, we know that:

$$S = \begin{bmatrix} \Sigma \\ 0 \end{bmatrix},$$

where by applying the definition of Σ stated above we know $\Sigma = diag(\sigma_1, \ldots, \sigma_n)$. Thus $\Sigma \in \mathbb{R}^{nxn}$.

However, since $m = n$ we know S being in $\mathbb{R}^{n\times m}$ is equivalent to S being in $\mathbb{R}^{n\times n}$, therefore $S = \Sigma$ as S contains Σ and we know both have the same dimensions therefore the two matrices must be equivalent.

Thus we can now rewrite A as $U\Sigma V^T$.
Computing these matrices we get,

$U\Sigma = (u_1\sigma_1, \ldots, u_n\sigma_n)$ where u_i is the ith element from the ith column of the matrix U.

Now we can finally compute USV^T.
Let $v_1^T, \ldots, v_n^T$ be column vectors of V^T. Thus,

We know that, $USV^T = (u_1\sigma_1, \ldots, u_n\sigma_n) \begin{pmatrix} v_1^T \\ \vdots \\ v_n^T \end{pmatrix}$ which can be expanded into $\sum_{i=1}^{n} \sigma_i u_i v_i^T$.

And since $u_i v_i^T$ results in a vector let us pick a arbitrary $\sigma_i u_i v_i^T$ from i = $1, \ldots, n$,

$$\text{Expanding } \sigma_i u_i v_i^T \text{ we get } \sigma_i \begin{pmatrix} u_{1i} \\ \vdots \\ u_{ni} \end{pmatrix} \begin{pmatrix} v_{1i}^T, \ldots, v_{ni}^T \end{pmatrix}.$$

$$\text{Which can be further expanded into } \sigma_i \begin{pmatrix} u_{1i}v_1^T, \ldots, u_{1i}v_n^T \\ u_{2i}v_1^T, \ldots, u_{2i}v_n^T \\ \vdots \\ u_{ni}v_1^T, \ldots, u_{ni}v_n^T \end{pmatrix}.$$

Therefore we can see that since each row is a multiple of the vector $(v_1^T, \ldots, v_n^T)$ they are all linearly dependent. Thus there is only 1 linearly independent row indicating that A can

be written as a sum of rank-1 matrices. Therefore we have proven the case of the row and column dimensions being the same.

Next let us explore the case of m > n. By definition, S, becomes $[\Sigma, 0]$, where $\Sigma = diag(\sigma_1, \ldots, \sigma_n)$ as $n < m$.

Now let us try and compute USV^T.

First, let's compute US.
Since only the first n columns from S are nonzero where the remaining $n+1, \ldots, m$ columns are zero and taking any product of a zero vector results in zero. Thus the resulting matrix product US also only has n many nonzero columns with the remaining columns also being zero. This results in

$$(u_1\sigma_1, \ldots, u_n\sigma_n, 0, \ldots, 0)$$

where this resulting matrix product is in $\mathbb{R}^{n \times m}$ and u_i denotes the ith column of the matrix U.

Now taking on USV^T we get

$$(u_1\sigma_1, \ldots, u_n\sigma_n, 0, \ldots, 0) \begin{bmatrix} v_1^T \\ \vdots \\ v_n^T \\ \\ v_{n+1}^T \\ \vdots \\ v_m^T \end{bmatrix}$$

which can be expanded into,

$$\sum_{i=1}^{n} \sigma_i u_i v_i^T + \sum_{i=n+1}^{m} 0 v_i^T$$

which simplifies to,

$$\sum_{i=1}^{n} \sigma_i u_i v_i^T$$

which gets us the same result as from Case 1 of when row and column dimensions were the same. Thus in both cases we can express A as a sum of rank 1 matrices by using SVD.

So now that we know that A can be constructed as a sum of rank 1 matrices, we can use this summation form of A to explore other properties. Additionally, we know that the 2-norm of a matrix A is its largest singular value which will be at the smallest ith value corresponding to σ_i.

Example 2

Let $A \in \mathbb{R}^{n\times m}$ have the singular decomposition as

$$A = USV^T = \sum_{i=1}^{\min(n,m)} \sigma_i u_i v_i^T$$

and let $A_k \in \mathbb{R}^{n\times m}$ be the best rank k approximation to A. Thus A_k can be written as $\sum_{i=1}^{k} \sigma_i u_i v_i^T$.
Without loss of generality, let us assume that $min(n, m) = n$ thus we can now rewrite A as $\sum_{i=1}^{n} \sigma_i u_i v_i^T$.
Thus we can now express $A - A_k$ as

$$\sum_{i=1}^{n} \sigma_i u_i v_i^T - \sum_{i=1}^{k} \sigma_i u_i v_i^T$$

which is simplified into

$$\sum_{i=k+1}^{n} \sigma_i u_i v_i^T.$$

To find $||A-A_k||_2$ would be the max eigenvalue among $A-A_k$ and since $n > n-1 > \cdots > k+1$ by the order of singular values σ_{k+1} is the max singular value. Thus, $||A - A_k||_2 = \sigma_{k+1}$.

So now that we have analyzed the properties of SVD, we also need to know how to deconstruct a matrix A into its components. Specifically we need to know how to compute the left singular and right singular vectors along with the diagonal matrix. One way we can compute the left singular vectors in U is through $Av_i = \sigma_i u_i$. Let's look into this further.

Problem 3

Given A in $\mathbb{R}^{n\times n}$, let $A = USV^T$ to be the singular value decomposition (SVD) of A where

$$S = \text{diag}(\sigma_1, \ldots, \sigma_n), \quad U = \begin{bmatrix} u_1 & \ldots & u_n \end{bmatrix}, \quad V = \begin{bmatrix} v_1 & \ldots & v_n \end{bmatrix}, \quad \text{and} \quad U^TU = V^TV = I_n.$$

Let us expand AA^T as

$$(USV^T)(USV^T)^T \text{ which is equivalent to } (USV^T)(VS^TU^T).$$

And since $V^TV = I_n$ we can further simply the expression into

$$USI_nS^TU^T \text{ which by the identity matrix is equivalent to } USS^TU^T.$$

Now let us choose a arbitrary ith column from U, let's call u_i.

$$\text{Now let's compute } AA^Tu_i, \text{ which is expressed as } USS^TU^Tu_i$$

Since $U^TU = I_n$, $U^Tu_i = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} = e_i$ which is in $\mathbb{R}^{n\times 1}$ which marks the ith row as 1 and everywhere else as

Additionally, since S is a diagonal matrix, $SS^T = diag(\sigma_1^2, \ldots, \sigma_n^2)$.
Thus, let's continue finding $USS^TU^Tu_i$.

Let's expand the value of SS^T in $USS^TU^Tu_i$ to get $Udiag(\sigma_1^2, \ldots, \sigma_n^2)e_i$

which is equivalent to

$$Udiag(\sigma_1^2, \ldots, \sigma_n^2) \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} = U\sigma_i^2 e_i$$

$$= \sigma_i^2 U e_i = \sigma_i^2 u_i.$$

Thus, $AA^Tu_i = \sigma_i^2 u_i$ therefore σ_i^2 is the corresponding eigenvalue for u_i
Next let us take a arbitrary v_i from V.
Using this vector, let us compute A^TAv_i,
First, let's expand the value of A^TA into,

$$(USV^T)^T(USV^T) = VS^TU^TUSV^T = VS^TSV^T = Vdiag(\sigma_1^2, \ldots, \sigma_n^2)V^T.$$

Now that we have a new expression for A^TA, we can see A^TAv_i to be

$$Vdiag(\sigma_1^2, \ldots, \sigma_n^2)V^Tv_i.$$

And since we know

$V^Tv_i = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} = e_i \in \mathbb{R}^{n\times 1}$ where the ith row is 1 and everywhere else is 0, we can use this to show

$$Vdiag(\sigma_1^2, \ldots, \sigma_n^2)V^Tv_i = Vdiag(\sigma_1^2, \ldots, \sigma_n^2)e_i = V\sigma_i^2 e_i = \sigma_i^2 V e_i = \sigma_i^2 v_i$$

Thus, $A^TA_i^v = \sigma_i^2 v_i$ therefore σ_i^2 is the corresponding eigenvalue for v_i

Finally let v_i be a arbitrary ith vector in V, let's multiply Av_i and we can get

$$USV^Tv_i = USe_i = U\sigma_i e_i = \sigma_i U e_i = \sigma_i u_i.$$

Thus, $Av_i = \sigma_i u_i$.

Now that we have a full understanding of SVD, it's about time we solve a matrix step by step to get its SVD. We'll need to first compute the eigenvalues and eigenvectors of A^TA. Then normalize the eigenvectors. Then we would be able to find Σ and V^T. The last step would be to find U which would be done as shown on the proof above to get each left singular value.

Example 4

Compute the SVD and optimal rank-1 approximation of the matrix

$$A = \begin{pmatrix} 2 & 3 \\ 0 & 2 \end{pmatrix}$$

by hand.

Answer.)

To compute the SVD we need to first begin with finding A^TA and its corresponding eigenvalues and eigenvectors.

Performing matrix multiplication of A^TA we get

$$\begin{pmatrix} 4 & 6 \\ 6 & 13 \end{pmatrix}$$

which we will use to find the eigenvalues and eigenvectors of A^TA.

$$\text{Next we will find } det \begin{pmatrix} 4-\lambda & 6 \\ 6 & 13-\lambda \end{pmatrix} = 0.$$

By calculating the determinant we get

$$(4-\lambda)(13-\lambda) - 36 = 0,$$

$$\lambda^2 - 17\lambda + 16 = 0,$$

$$(\lambda - 16)(\lambda - 1) = 0,$$

$$\text{Which gives us the eigenvalues of } \lambda = 16, 1.$$

Now let's find the corresponding eigenvector the $\lambda = 1$ by solving for v in $(A^TA - 1\mathbb{I})v = 0$. Thus we get,

$$\begin{pmatrix} 3 & 6 \\ 6 & 12 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = 0,$$

which by using row reduction can be simplified into

$$\begin{pmatrix} 1 & 2 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = 0.$$

We can see that v_2 is a free variable and $v_1 = -2v_2$. Therefore we can choose $v_2 = 1$ which makes $v_1 = -2$ expressed as,

$$\begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} -2 \\ 1 \end{pmatrix}.$$

Next to find the corresponding eigenvectors for $\lambda = 16$ we will need to repeat the process for $(A^T A - 16\mathbb{I})v = 0$.

Thus we get,

$$\begin{pmatrix} -12 & 6 \\ 6 & -3 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = 0,$$

which can be row reduced into the form

$$\begin{pmatrix} 6 & -3 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = 0$$

which gives us v_2 as a free variable and $6v_1 = 3v_2$. Thus, we can choose $v_2 = 1$ and so $v_1 = \frac{1}{2}$ which can be written into

$$\begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} \frac{1}{2} \\ 1 \end{pmatrix}.$$

Now we need to construct $S = diag(\sigma_1, \sigma_2)$. To find this we will need to take the square root of the eigenvalues and have them as the diagonal elements in descending order, meaning the largest eigenvalues on the top left and smallest on the bottom right. Therefore we get the matrix

$$S = \begin{pmatrix} \sqrt{16} & 0 \\ 0 & \sqrt{1} \end{pmatrix} = \begin{pmatrix} 4 & 0 \\ 0 & 1 \end{pmatrix}.$$

Now let us normalize the eigenvectors by taking

$$||v_1|| = \sqrt{\frac{1}{4} + 1} = \sqrt{\frac{5}{4}} = \frac{\sqrt{5}}{2}.$$

Now we need to divide each element in v_1 by its magnitude to get,

$$v_1 = \frac{2}{\sqrt{5}} \begin{pmatrix} \frac{1}{2} \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} \end{pmatrix}.$$

Now to find the magnitude of v_2,

$$||v_2|| = \sqrt{5}.$$

Now we'll take that value and divide it accross the elements in v_2 such as,

$$v_2 = \frac{1}{\sqrt{5}} \begin{pmatrix} -2 \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{-2}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} \end{pmatrix}.$$

Thus we can now construct V with the normalized values of v_1 and v_2 which gives us

$$V = \begin{pmatrix} \frac{1}{\sqrt{5}} & \frac{-2}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} \end{pmatrix}.$$

Now that we have the matrix V we can simply flip its columns into rows to get,

$$V^T = \begin{pmatrix} \frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} \\ \frac{-2}{\sqrt{5}} & \frac{1}{\sqrt{5}} \end{pmatrix}.$$

Finally we need to construct the matrix U. Let us calculate the first element through

$$u_1 = \frac{1}{4}\begin{pmatrix} 2 & 3 \\ 0 & 2 \end{pmatrix}\begin{pmatrix} \frac{1}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} \end{pmatrix} = \frac{1}{4}\begin{pmatrix} \frac{2}{\sqrt{5}} + \frac{6}{\sqrt{5}} \\ \frac{4}{\sqrt{5}} \end{pmatrix} = \begin{pmatrix} \frac{2}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} \end{pmatrix}.$$

Now to calculate the second element with its corresponding normalized eigenvector and sigma by

$$u_2 = \begin{pmatrix} 2 & 3 \\ 0 & 2 \end{pmatrix}\begin{pmatrix} \frac{-2}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} \end{pmatrix} = \begin{pmatrix} \frac{-1}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} \end{pmatrix}.$$

This putting together u_1 and u_2 we get the outcome

$$U = \begin{pmatrix} \frac{2}{\sqrt{5}} & \frac{-1}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} \end{pmatrix}.$$

Thus the SVD of matrix A is

$$A = USV^T = \begin{pmatrix} \frac{2}{\sqrt{5}} & \frac{-1}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} \end{pmatrix}\begin{pmatrix} 4 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} \frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} \\ \frac{-2}{\sqrt{5}} & \frac{1}{\sqrt{5}} \end{pmatrix}.$$

To validate our answer, let's compute USV^T.

First to compute US we get

$$US = \begin{pmatrix} \frac{2}{\sqrt{5}} & \frac{-1}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} \end{pmatrix}\begin{pmatrix} 4 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} \frac{8}{\sqrt{5}} & \frac{-1}{\sqrt{5}} \\ \frac{4}{\sqrt{5}} & \frac{2}{\sqrt{5}} \end{pmatrix}.$$

Next we will multiply with V^T to get

$$(US)V^T = \begin{pmatrix} \frac{8}{\sqrt{5}} & \frac{-1}{\sqrt{5}} \\ \frac{4}{\sqrt{5}} & \frac{2}{\sqrt{5}} \end{pmatrix}\begin{pmatrix} \frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} \\ \frac{-2}{\sqrt{5}} & \frac{1}{\sqrt{5}} \end{pmatrix},$$

$$= \begin{pmatrix} \frac{8}{5} + \frac{2}{5} & \frac{16}{5} - \frac{1}{5} \\ \frac{4}{5} - \frac{4}{5} & \frac{8}{5} + \frac{2}{5} \end{pmatrix} = \begin{pmatrix} 2 & 3 \\ 0 & 2 \end{pmatrix} = A.$$

We can see that our SVD is equivalent to A, therefore we are confident in our answer.

Next let us find the optimal rank 1 approximation of A.
Since this is rank 1 approximation we only need to take $\sigma_1 u_1 v_1^T$ which can be extracted as

$$4\begin{pmatrix}\frac{2}{\sqrt{5}}\\ \frac{1}{\sqrt{5}}\end{pmatrix}\begin{pmatrix}\frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}}\end{pmatrix} = 4\begin{pmatrix}\frac{2}{5} & \frac{4}{5}\\ \frac{1}{5} & \frac{2}{5}\end{pmatrix} = \begin{pmatrix}\frac{8}{5} & \frac{16}{5}\\ \frac{4}{5} & \frac{8}{5}\end{pmatrix}.$$

Therefore the optimal rank 1 approximation of A is

$$\begin{pmatrix}\frac{8}{5} & \frac{16}{5}\\ \frac{4}{5} & \frac{8}{5}\end{pmatrix}.$$

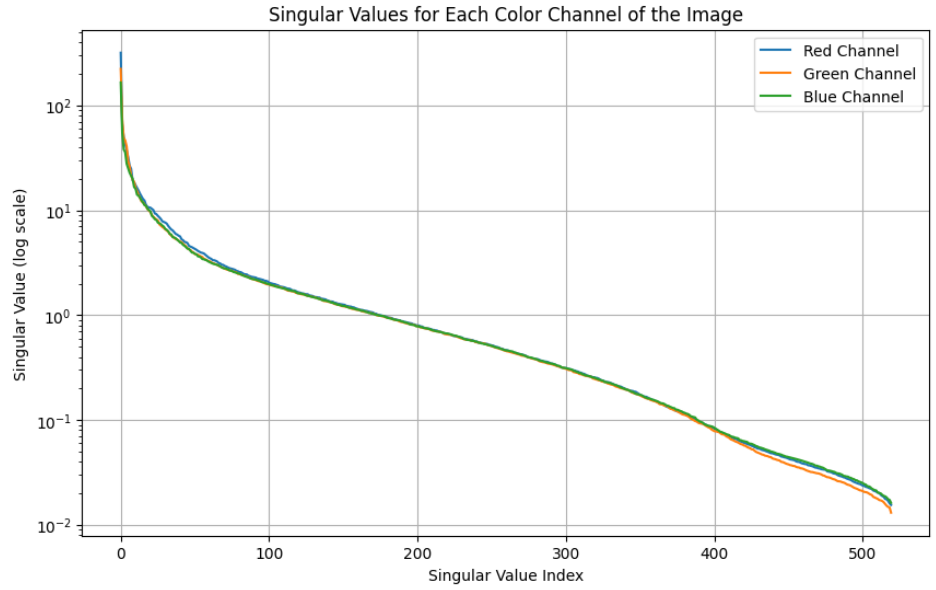
Now that we are confident in our understanding of SVD we can apply what we learned in real-world problems. Let's see how SVD is used in rank-k approximations on images and get visualizations on the effect. We would also need to analyze the error and how much the approximations are diverging from the original image to get a statistic on the effect of SVD on image compression.

Example 5

Here we would do a example of manipulating a JPG image in python by separating our image into red, green, blue channels and then analyzing their best-k approximation given various increasing values of k.

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt

# Load the image
hokie_cat = Image.open("hokie_cat.jpg")

hokie_cat_array = np.array(hokie_cat, dtype = np.double)

# Normalize pixel values to range [0, 1]
normalized_hokie_cat = hokie_cat_array / 255.0

#Split into color channels
red_channel = normalized_hokie_cat[:, :, 0]
green_channel = normalized_hokie_cat[:, :, 1]
blue_channel = normalized_hokie_cat[:, :, 2]
```

```
# Compute the SVD for each channel
U_r, S_r, Vt_r = np.linalg.svd(red_channel, full_matrices=False)
U_g, S_g, Vt_g = np.linalg.svd(green_channel, full_matrices=False)
U_b, S_b, Vt_b = np.linalg.svd(blue_channel, full_matrices=False)


# Plot singular values on a logarithmic scale using semilogy
plt.figure(figsize=(10, 6))
plt.semilogy(S_r, label="Red Channel")
plt.semilogy(S_g, label="Green Channel")
plt.semilogy(S_b, label="Blue Channel")

plt.xlabel("Singular Value Index")
plt.ylabel("Singular Value (log scale)")
plt.title("Singular Values for Each Color Channel of the Image")
plt.legend()
plt.grid(True)
plt.show()
```



These curves all follow the theory behind the relationship behind singular values and their indices. The largest singular value for any matrix will always be at index 0 with their following values being less than or equal to the one before. Thus we can see a descent throughout the graph on all three channels.

```
    import pandas as pd
# Part B
```

```
# List of k values to consider for rank-k approximation
k_values = [1, 5, 10, 25, 50, 75, 100, 150]

# Dictionary to store errors for each color channel
errors = {
    'Red': [],
    'Green': [],
    'Blue': []
}

for k in k_values:
    # Red channel
    #We take top k columns from U and k singular values and k rows from Vt to make the s
    A_r_k = U_r[:, :k] @ np.diag(S_r[:k]) @ Vt_r[:k, :]


    #Get the 2 norm of the k approximation and the 2 norm of the original
    A_r_k_norm2 = np.linalg.norm(A_r_k - red_channel, ord=2)

    #Get the absolute errors:

    errors['Red'].append(A_r_k_norm2) #Add the error to the corresponding key in diction


    # Green channel
    A_g_k = U_g[:, :k] @ np.diag(S_g[:k]) @ Vt_g[:k, :]

    A_g_k_norm2 = np.linalg.norm(A_g_k - green_channel, ord=2)

    errors['Green'].append(A_g_k_norm2)

    # Blue channel
    A_b_k = U_g[:, :k] @ np.diag(S_b[:k]) @ Vt_b[:k, :]

    A_b_k_norm2 = np.linalg.norm(A_b_k - blue_channel, ord=2)
    #Get the absolute errors:
    errors['Blue'].append(A_b_k_norm2)

# Convert the dictionary to a DataFrame to be plotted
error_df = pd.DataFrame(errors, index=k_values)

error_df.index.name = 'k'

col_labels = ["k", "Red", "Green", "Blue"]
```

```
table_data = error_df.reset_index().values  # Include the "k" values in the table data

fig, ax = plt.subplots(figsize=(8, 4))
ax.axis('tight')
ax.axis('off')

fig.suptitle("Table of Errors Among Channels", fontsize=14, fontweight='bold')

table = ax.table(cellText=table_data, colLabels=col_labels, cellLoc='center', loc='cente
table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1.2, 1.2)

plt.show()
```

Here is a table of the corresponding error values among red, green, and blue channels in the 2 norm for every value of k.

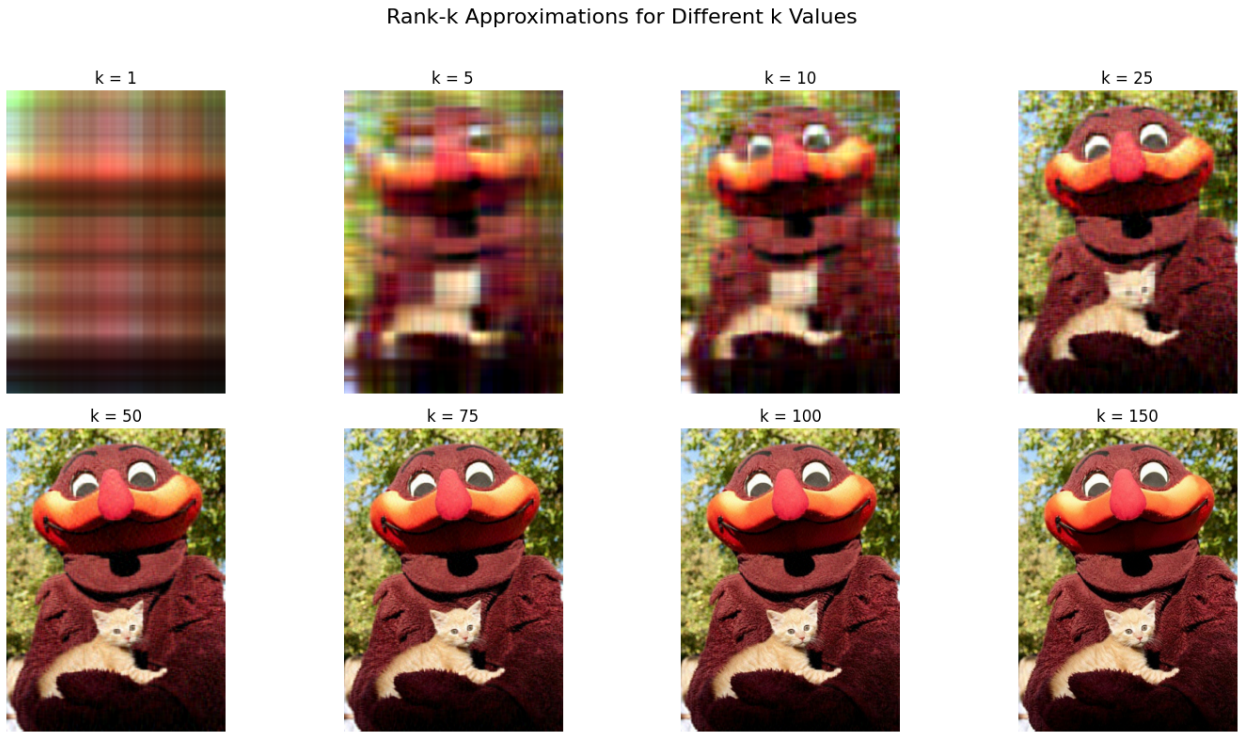
Table of Errors Among Channels

k	Red	Green	Blue
1.0	66.92332226393316	74.00929345287703	55.28856894204164
5.0	32.10665610999809	29.184313211521843	50.69661136958626
10.0	17.33357148507629	16.33411317316412	58.04303943118257
25.0	8.851613277624915	8.04086175528002	58.07674927603051
50.0	4.282291523402	3.815325469153946	58.079555038528554
75.0	2.7649035390982366	2.650150377643869	58.07999961775558
100.0	2.0391468661721404	1.959520802701354	58.0799890700428
150.0	1.2609896317435394	1.2148432409182652	58.08001514455656

Now let us visualize these images among different values of k.

```
k_values = [1, 5, 10, 25, 50, 75, 100, 150]


fig, axes = plt.subplots(2, 4, figsize=(15, 8))
fig.suptitle("Rank-k Approximations for Different k Values", fontsize=16)

for i, k in enumerate(k_values):
```

```
    # Compute the best rank-k approximation for each color channel based on the k value
    best_Rk = U_r[:, :k] @ np.diag(S_r[:k]) @ Vt_r[:k, :]
    best_Gk = U_g[:, :k] @ np.diag(S_g[:k]) @ Vt_g[:k, :]
    best_Bk = U_b[:, :k] @ np.diag(S_b[:k]) @ Vt_b[:k, :]

    # Combine the channels into a single image
    approximation = np.zeros_like(hokie_cat_array)
    approximation[:, :, 0] = best_Rk  # Red channel
    approximation[:, :, 1] = best_Gk  # Green channel
    approximation[:, :, 2] = best_Bk  # Blue channel

    # Select the appropriate subplot
    ax = axes[i // 4, i % 4]

    # Display the approximation in the subplot
    ax.imshow(np.clip(approximation, 0, 1))
    ax.axis('off')
    ax.set_title(f"k = {k}")

plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```



Here is the original image for reference

As anticipated, a small k value drastically decompresses the image but gives a much lower quality image visualization. However, as the values of k increase, there is a point of diminishing return as shown by both the plot and visualizing these images. We can see that eventually there is minimal differences that can be inspected starting from k = 75 until k = 150.

Example 6.)

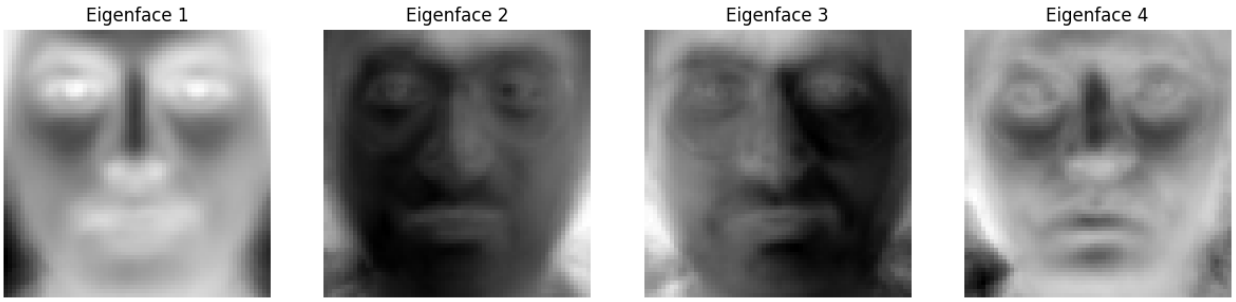
Instead of just using SVD to compress images, we can also use this to reconstruct a image. This is very useful if we have limited access to data due to unforseen events. Thus we will need to reconstruct a new matrix (in our case a image) to get as close as possible to what is actual.
We will attempt to reconstruct a image based on a dataset of faces in finding the average face among the dataset. Then we will use SVD to reconstruct 3 images not found in the dataset and visualize how close we can get to the original.

```
import scipy.io

#load the mat file
mat = scipy.io.loadmat('yalefaces.mat')

Yale = mat['Y']

sum = np.zeros(Yale[0][0].shape, dtype = 'float64')
```

```
for i in range(Yale.shape[1]):
    sum += Yale[0][i]

avg = sum /Yale.shape[1]

#Plotting the average face
plt.imshow(avg, cmap='gray')
plt.axis('off')
```

```
#Vectorizing the average face
vectorized_average = avg.ravel().T
vectorized_average = vectorized_average[0]

vectorized_images = []

for i in range(Yale.shape[1]):  # 132 images
    image = Yale[0][i]  # Extract each individual image
    vectorized_image = image.ravel()  # Flatten the image into a 1D vector
```

```
    vectorized_images.append(vectorized_image)

vectorized_images = np.array(vectorized_images).T

N = vectorized_images.shape[0]

# Center each image by subtracting the vectorized average
centered_images = vectorized_images - vectorized_average

# Normalize by dividing by sqrt(N)
D = centered_images / np.sqrt(N)

#Compute the SVD:

U, S, V_t = np.linalg.svd(D, full_matrices=False)

eigenfaces = U[:, :4]  # The first 4 columns of U are the leading eigenfaces

# Plot the leading four eigenfaces
fig, axes = plt.subplots(1, 4, figsize=(15, 5))


for i in range(4):
    eigenface = eigenfaces[:, i].reshape(64, 64)

    # Plot the eigenface
    axes[i].imshow(eigenface, cmap='gray')
    axes[i].axis('off')
    axes[i].set_title(f"Eigenface {i+1}")

plt.show()
```

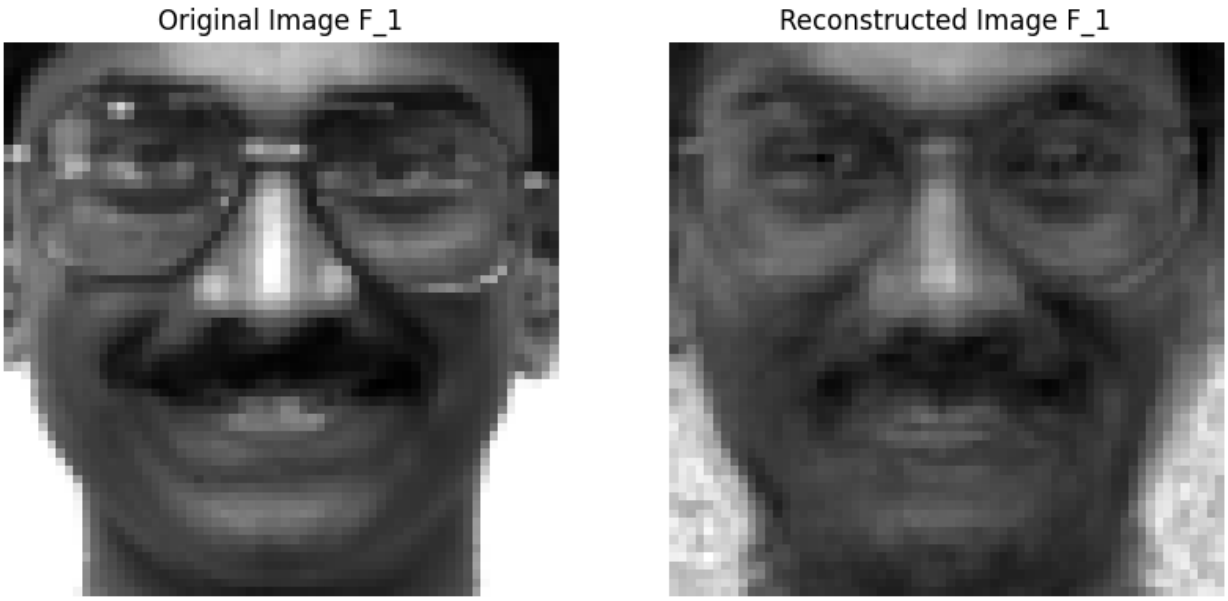


The first eigenface is exactly the same as the average face computed from Part A. However,

all following eigenfaces look significantly more different.

```
#load the mat file
mat = scipy.io.loadmat('images_to_reconstruct.mat')

print(mat.keys())

F_1 = mat['F1']
F_2 = mat['F2']
F_3 = mat['F3']

F_1_vector = F_1.ravel()
F_2_vector = F_2.ravel()
F_3_vector = F_3.ravel()

rank = np.linalg.matrix_rank(D)

U_r = U[:, :rank]
U_r_t = U_r.T


f_1 = F_1_vector - vectorized_average
f_1_approx = U_r @ U_r_t @ f_1 + vectorized_average

f_2 = F_2_vector - vectorized_average
f_2_approx = U_r @ U_r_t @ f_2 + vectorized_average

f_3 = F_3_vector - vectorized_average
f_3_approx = U_r @ U_r_t @ f_3 + vectorized_average


height, width = F_1.shape

# Reshape the approximation back to the original image shape
f_1_approx_image = f_1_approx.reshape(height, width)
f_2_approx_image = f_2_approx.reshape(height, width)
f_3_approx_image = f_3_approx.reshape(height, width)

for i in range(3):
    original_image = eval(f"F_{i+1}_image")
    approx_image = eval(f"f_{i+1}_approx_image")

    # Create a side-by-side subplot
```

```
plt.figure(figsize=(10, 5))

# Display the original image on the left
plt.subplot(1, 2, 1)
plt.imshow(original_image, cmap='gray')
plt.axis('off')
plt.title(f"Original Image F_{i+1}")

# Display the reconstructed image on the right
plt.subplot(1, 2, 2)
plt.imshow(approx_image, cmap='gray')
plt.axis('off')
plt.title(f"Reconstructed Image F_{i+1}")

# Show the side-by-side plot
plt.show()
```



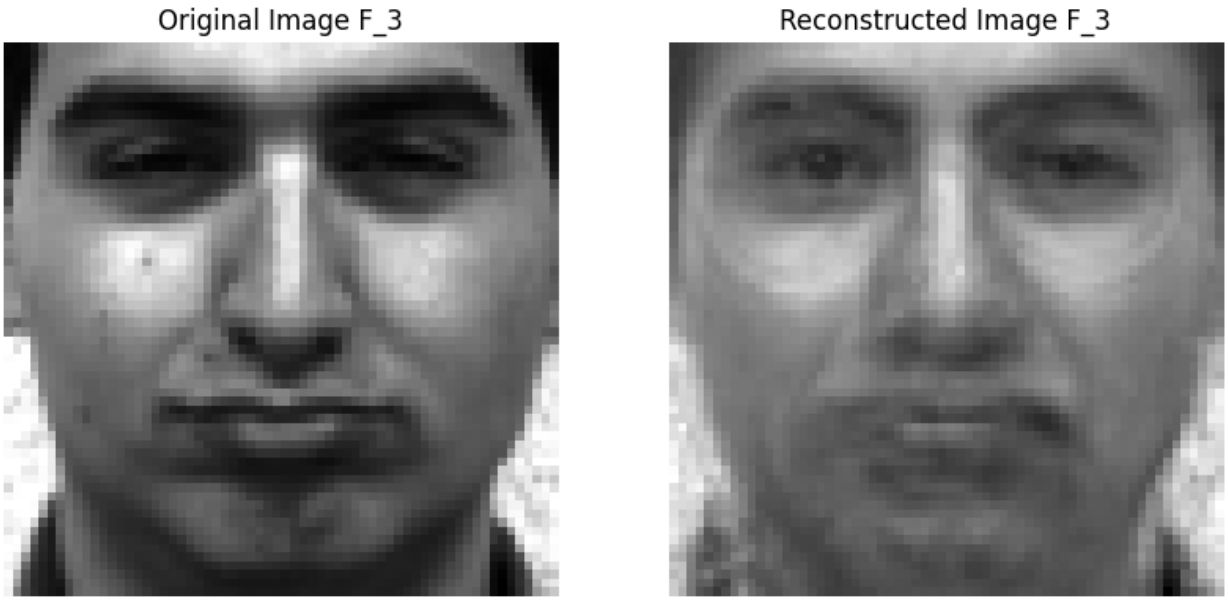
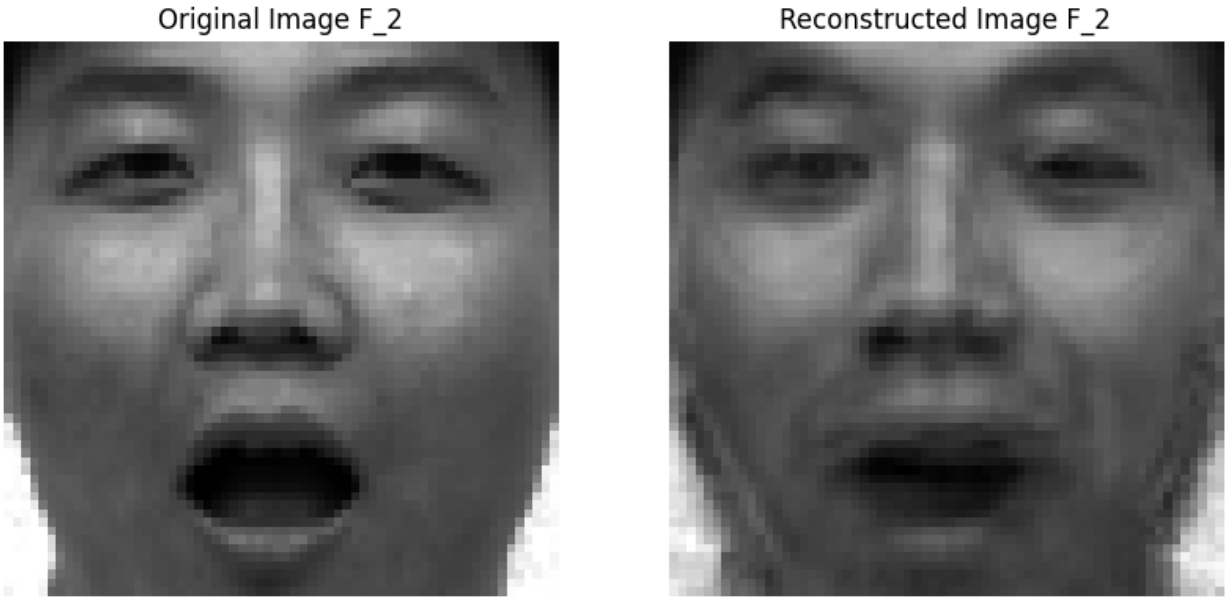


Figure 1: Enter Caption



All 3 reconstructed images good a good approximation to the original image especially considering that these images aren't from the database. However, this is still improvements to be made specifically with getting a better picture quality and a more accurate representation. An example would be how in image F_1 the man is wearing glasses and all following reconstructed images has a brief outline of glasses in the picture even though the original photo didn't have glasses. Therefore images used could cloud the reconstructed image's output.

Conclusion

We have analyzed the basic structure of singular value decomposition by breaking it apart and analyzing its different components. We then performed a singular value decomposition on a matrix from scratch step by step. We can see the use case of SVD by compressing information into a smaller matrix to get a approximate of a much larger matrix. We utilized this idea through compressing a image of a hokie bird where we noticed the stark drop in image quality if we make too much of a approximation but we are able to find a medium where we can have a more compressed image to take less space but still be pleased by the image quality.
Finally, we used SVD to perform a reconstruction of images of faces based on a SVD constructed from a dataset of 132 faces. We are able to get a reasonable quality ouut of the 3 unseen images which shows the power in utilizing this tool for data approximations in the real world.