Software Assignment - Report

AI25BTECH11021 - Abhiram Reddy N

1 Aim

The aim of this Software Assignment is to write code using algorithms of SVD to compress the images by using truncated SVD by computing the highest eigen values containing the most properrties of the image and neglecting the lower eigrn values as they mostly contains noise.

2 Outcomes

We get to know the differences of the algorithms and also usage of SVD and image compression.

I used three Coloured Images taken from Internet and compressed them with 5 different K values and calculated errors also and plotted them and also for the 3 B/W images given are compressed with different K values.

3 Extending to RGB colours

I also extended the SVD to RGB colors by applying SVD to three channels instead of 1 and also by taking three dimensional array handle color components separately for good compression accuracy.

The c code for coloured images is colour.c it works for both colour and B/W and I gave the c code for only grey as grey.c

4 Summary of Strang's Video

In this lecture, Strang tells about the singular value decomposition (SVD) as the most useful and general factorization for any matrix :

$$A = U\,\Sigma\,V^T$$

where U and V are orthogonal matrices and Σ is a rectangular diagonal matrix of *non-negative singular values*. He tells that eigen-decomposition works nicely for symmetric, square, positive-definite matrices and the SVD works for arbitrary matrices.

He then explains the meaning of each matrix: the right singular vectors (columns of V) come from diagonalizing A^TA, and the left singular vectors (columns of U) come from diagonalizing AA^T. The singular values are the square roots of the eigenvalues of A^TA.

He also interpret the EigenValues and the relation between the A and A_k with Genes.

Geometrically, he describes the SVD as: apply an orthogonal transformation via V^T, then stretch by Σ, then rotate by U. This decomposition gives a good view into the row space and column space of a matrix and sets up applications like low-rank approximations.

$$A_k \approx U_k\,\Sigma_k\,V_k^T$$

5 Implemented Algorithm

I used the **Randomized Power Iteration Algorithm**, which is a variant of *Randomized SVD*, was proposed by Halko, Martinsson, and Tropp (2011).

Purpose

To compute a low-rank approximation

$$A_k \approx U_k \, \Sigma_k \, V_k^T$$

without performing a full singular value decomposition (SVD) — ideal for image compression in a fast way.

Explanation

Let $A \in \mathbb{R}^{m \times n}$ be a given matrix (for example, an image matrix). We want to get a low-rank approximation A_k of rank k, such that $A_k \approx A$, without performing a full singular value decomposition, so that it takes less time.

The **Randomized Power Iteration Algorithm** proceeds as follows:

1. **Generate a random test matrix.** Construct a random matrix
$$\Omega \in \mathbb{R}^{n \times k},$$
whose entries are drawn from a uniform or Gaussian distribution. This matrix serves to randomly sample the column space of A.

2. **Project A to a lower-dimensional subspace.** Compute
$$Y = A\Omega,$$
which gives a matrix $Y \in \mathbb{R}^{m \times k}$ that captures most of the range of A. Each column of Y is a random linear combination of the columns of A.

3. **Apply power iterations to get more accuracy.** To better approximate the dominant singular subspace, repeat the following steps for p iterations:
$$Y = A(A^T Y).$$
Each power iteration amplifies the contribution of the largest singular values of A, improving the quality of the approximation when singular values decay slowly.

4. **Compute an orthonormal basis.** Perform QR decomposition (or Gram–Schmidt orthogonalization) on Y to obtain
$$Q = \text{orth}(Y),$$
where $Q \in \mathbb{R}^{m \times k}$ has orthonormal columns. The columns of Q form an approximate basis for the column space of A.

5. **Project A onto the subspace spanned by Q.** Compute
$$B = Q^T A,$$
resulting in a smaller matrix $B \in \mathbb{R}^{k \times n}$. This matrix B is the representation of A in the reduced subspace.

6. **Form the rank-k approximation.** The low-rank approximation of A is given by
$$A_k = QB.$$
Thus, A_k is an $m \times n$ matrix of rank at most k that approximates A.

7. **Compute the approximation error.** The quality of the approximation can be measured using the Frobenius norm:
$$\|A - A_k\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} (A_{ij} - (A_k)_{ij})^2}.$$
The relative Frobenius error ratio is then given by
$$\frac{\|A - A_k\|_F}{\|A\|_F}.$$

8. **Interpretation.** The matrix A_k retains the dominant features of A while discarding the less significant ones (noise). In the context of image compression, this means A_k reconstructs an image visually similar to the original but using only the top k components, leading to a significant reduction in storage.

Pseudo Code

Algorithm 1 Randomized Power Iteration for Image Compression (SVD-based)

Input: Image file (JPG or PNG), target ranks $K_1, K_2, \ldots, K_n$
Output: Compressed images and error file (Frobenius norms)
Read the image from `figs/subfolder/image.(jpg/png)` and convert to grayscale
Convert image pixels to a double matrix A of size $m \times n$
Compute Frobenius norm of A for error normalization
for each K value **do**
    Generate random matrix Ω of size $n \times K$
    Compute $Y = A\Omega$
    for $p = 1$ to P **do**
        $Y = A(A^T Y)$ ▷ Power iteration for stability
    end for
    Orthonormalize Y to form matrix Q
    Compute $B = Q^T A$
    Compute low-rank approximation $A_K = QB$
    Compute Frobenius error: $\|A - A_K\|_F$ and ratio $\frac{\|A - A_K\|_F}{\|A\|_F}$
    Save compressed image as `subfolder_K.(jpg/png)`
    Write error and ratio to `errors.txt`
end for
Free all allocated matrices and memory
End

6 Comparing Different Algorithms

Why did I use randomised SVD power iteration

I'm using the Randomized SVD algorithm as :

The randomized power iteration algorithm is faster and more memory-efficient than traditional SVD methods such as Golub–Reinsch or Lanczos when only a few dominant singular values are required. It avoids computing the full decomposition by working with a randomly projected subspace of the original matrix, reducing computational complexity and allowing efficient processing of large dense matrices such as images. This algorithm is well-suited for large-scale applications and parallel computation.

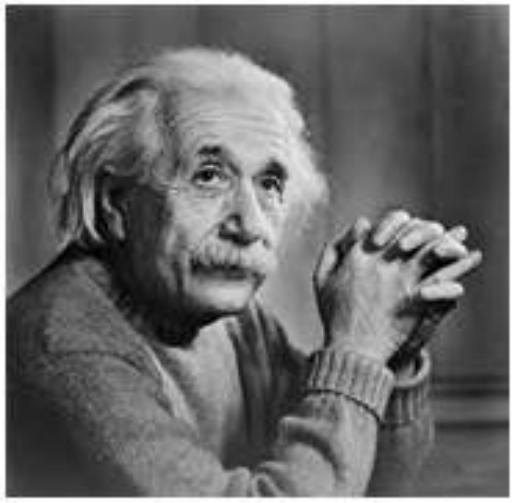
Another major advantage is its number of power iterations can be tuned to balance accuracy and speed, and even with a small number of iterations, it achieves results close to the exact truncated SVD. Randomization also helps reduce numerical errors and improves robustness for ill-conditioned data. Overall, the algorithm provides a simple, scalable, and accurate approach to image compression and low-rank approximation without the high cost of classical SVD methods.

Comparision

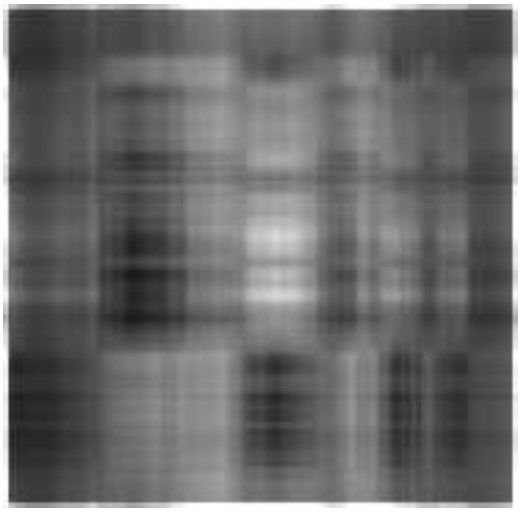
This image provides a brief overview of the comparison.

#	Method	Idea	Complexity	Accuracy	Notes / When used
	Lanczos Bidiagonall-zation (ARPACK)	Iterative Krylov subspace method building $B_k == A$	$O(kmn)$	High	Most classical top-k method.
2	Block Lanczos / Thick-Restart Lanczos	Block variant for faster convergence / fower restarts.	$O(kmn)$	High	Fastest for large dense data.
3	Randomized SVD (Halko-Martinsoon- Tropp)	Project A to randno'low-dim subspace, then SVD that.	$O(mn \log k + k^2(m+n)$	Near-optimal	Conceptual or teaching: slower
4	Power iteration (Subspace Iteration)	Repeat $A^{\text{“}} A$ multtplications to Gulate dominant sing vects.	$O(kmn \cdot iteru)$	Medium	Conceptual or teaching: slower
5	Incremental / Online SVD	Update top-k as new data streame in trank-1 updates	$O(kmn \cdot iters)$	Medium	Conceptual or same as Lanczos
6.	Orthogonal iteration / Simultaneous Iteration	Variant of power iteration for subspace of dimen: 8. $\tilde{\kappa}$	$O(kmn)$	High	Sometimes used in PCA code.
7	Rayleigh–Ritz projection (Krylov variant)	Approximate eigenpairs of $A^{\text{“}} A$ in Krylov subspace	$O(kmn)$	High	Sometimes used in PCA code

7 Reconstructed Images

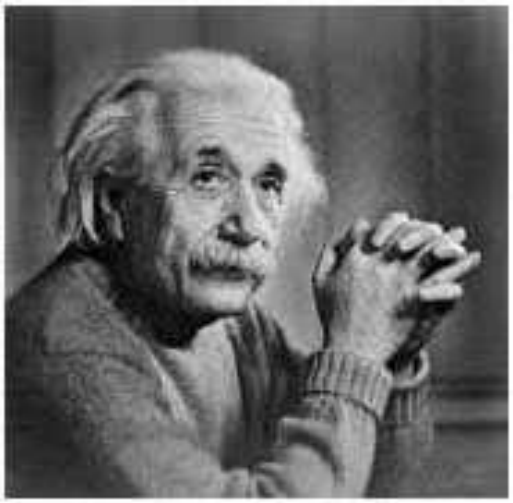


(a) Original

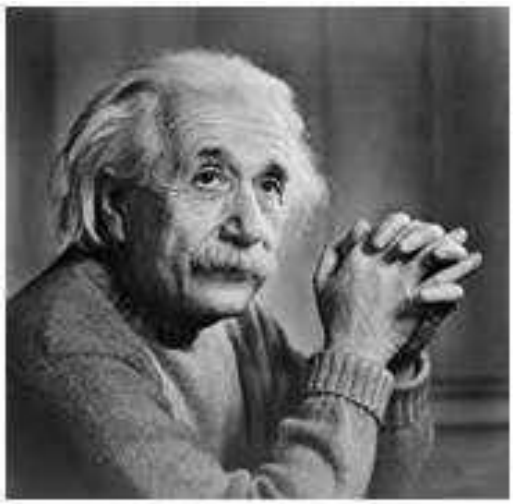


(b) K = 2

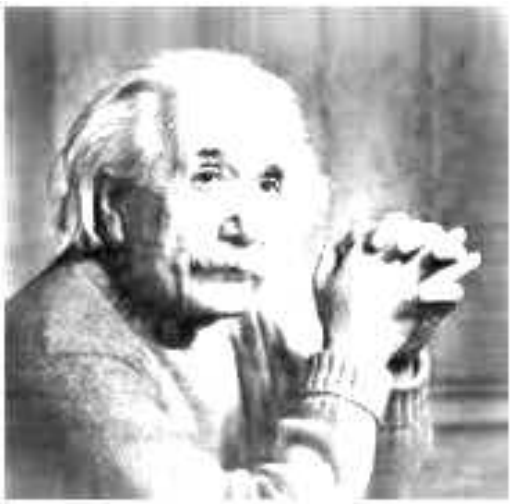
(c) K = 20



(d) K = 50

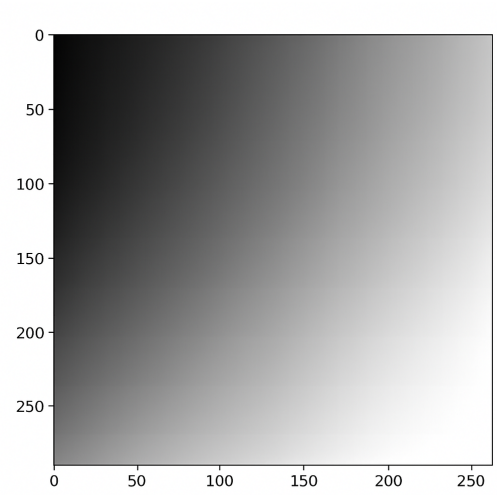


(e) K = 100

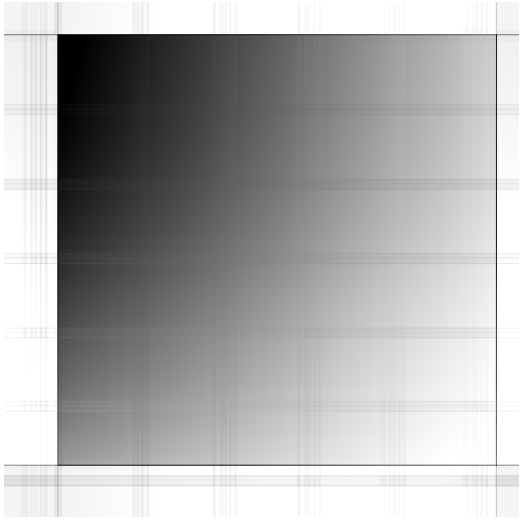


(f) K = 180

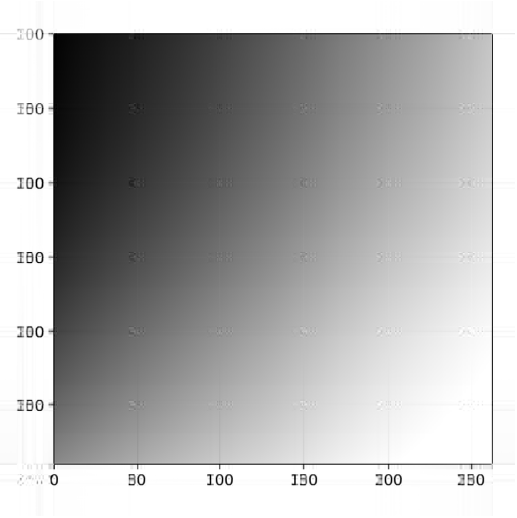
Figure 1: Grey set 1



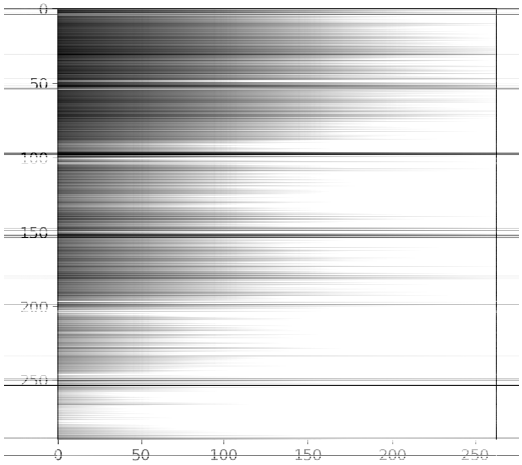
(a) Original



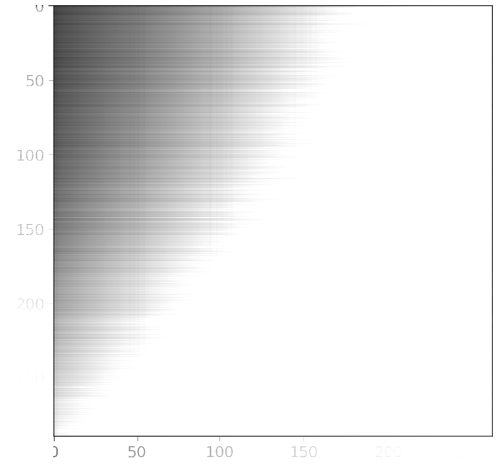
(b) K = 2



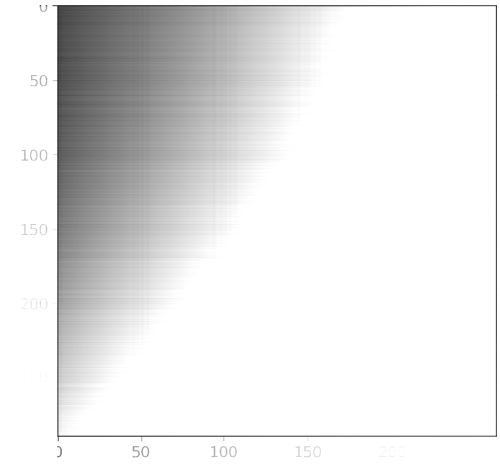
(c) K = 10



(d) K = 70



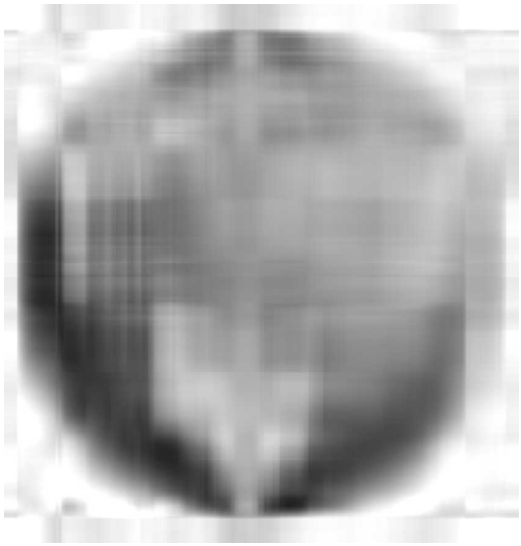
(e) K = 200



(f) K = 500

Figure 2: Grey set 2

(a) Original



(b) K = 5

(c) K = 50

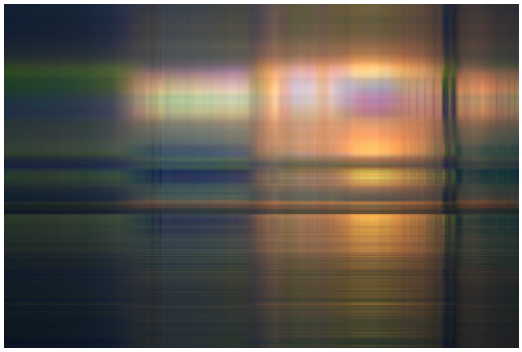
(d) K = 100

(e) K = 200

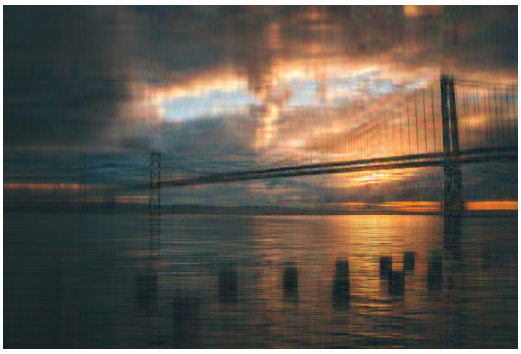
(f) K = 400

Figure 3: Grey set 3

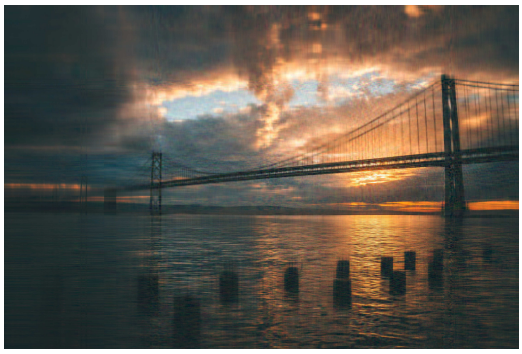
(a) Original



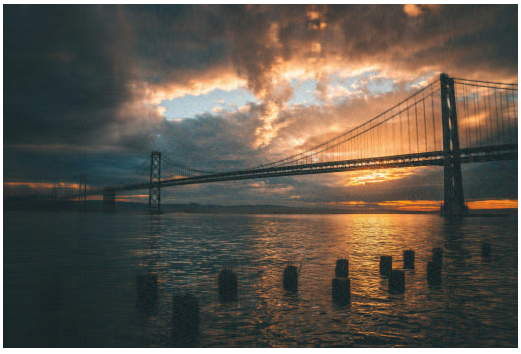
(b) K = 2



(c) K = 20



(d) K = 50

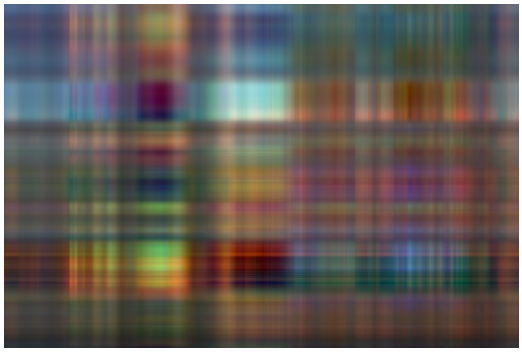


(e) K = 100

(f) K = 200

Figure 4: Colour set 1

(a) Original



(b) K = 2

(c) K = 20

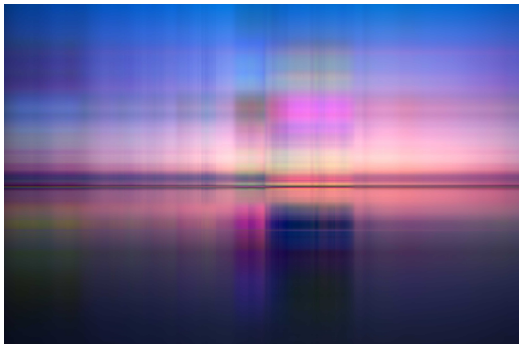
(d) K = 50

(e) K = 100

(f) K = 200

Figure 5: Colour set 2

(a) Original



(b) K = 2

(c) K = 20

(d) K = 50

(e) K = 100

(f) K = 200

Figure 6: Colour set 3

8 Error Analysis

Image Name	Metrics		
	K	**Frobenius Error**	**Frobenius Ratio**
Einstein	2	6472.393091	0.296775
	20	2195.303065	0.100660
	50	912.638918	0.041847
	100	193.096088	0.008854
	200	21680.125406	0.994086
Greyscale	2	17171.704800	0.088599
	10	7303.555135	0.037683
	70	157194.745688	0.811060
	200	188987.222872	0.975096
	500	189289.303833	0.976655
Globe	2	32845.296916	0.207471
	20	10805.790233	0.068256
	50	6295.838063	0.039768
	100	3797.060301	0.023985
	200	42965.980630	0.271400
1colour	2	213410.651161	0.260709
	20	139105.191656	0.169935
	50	115800.525691	0.141466
	100	95203.374085	0.116304
	200	71603.495348	0.087473
2colour	2	396299.080451	0.453137
	20	170131.567972	0.194532
	50	113686.270837	0.129991
	100	82304.470620	0.094109
	200	52691.535595	0.060249
3colour	2	184081.067433	0.165437
	20	68535.871369	0.061594
	50	42394.308302	0.038100
	100	36010.441302	0.032363
	200	403713.555469	0.362824

Table 1: SVD compression results for six images showing K, Frobenius / Relative Error, and Frobenius Ratio.

When we make the number of iteration more, the error start going more less because the matrix get more refine each time. Every extra iteration make the SVD approximation better and capture more detail of image, so Frobenius error and ratio come down. If we do only few iteration then it not converge properly and give big error, but if we do many time then result become more clear and accurate.

Plots

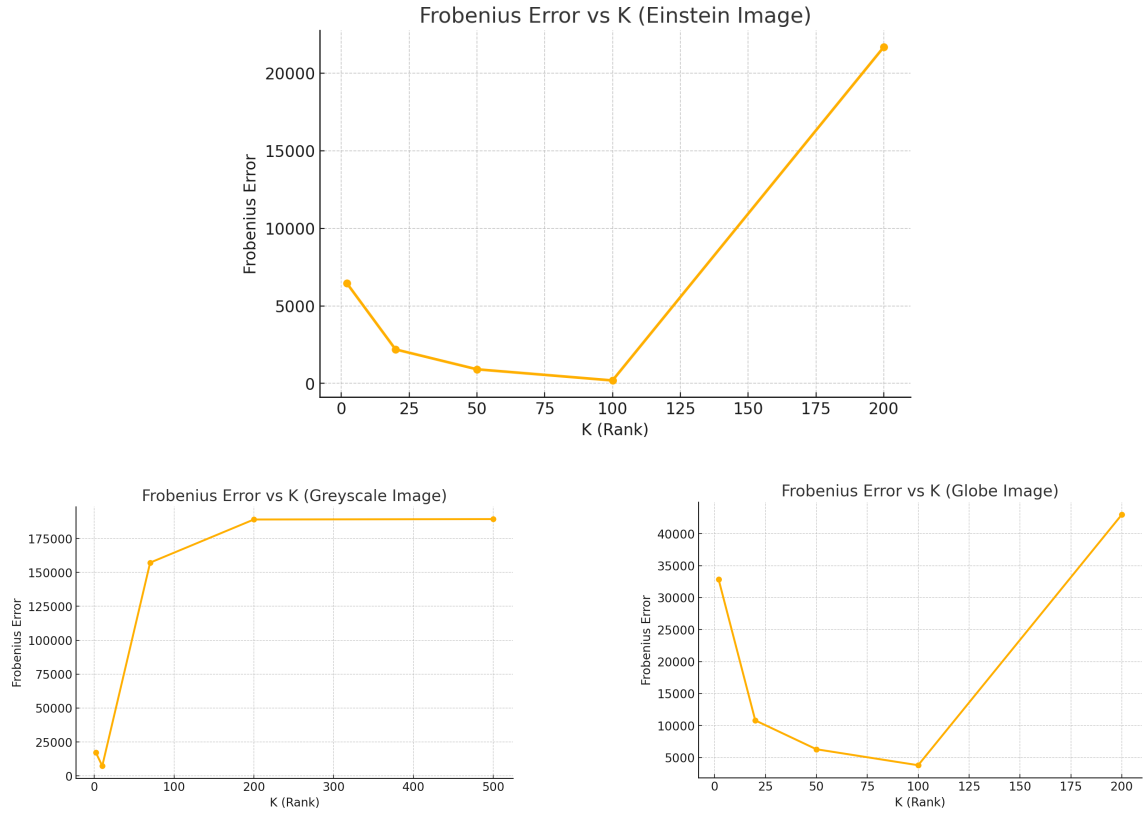


Figure 7: Comparison of Frobenius metrics across images: (top) Einstein, (bottom left) Greyscale, (bottom right) Globe.

9 Discussion between k, image quality, and compression

In SVD image compression, the value of k decides how many singular values we keep. If k is small, the image becomes light in size but also blurry or low in quality. When k increases, the picture looks more clear and close to the original, but the file size also grows bigger, reducing compression. So, we need to pick a k that gives good balance between quality and storage.

And also the quality of the output image matters a lot for the file size of the output so I set it to be 40 default.

So, we need to pick a k that gives a good balance between quality and storage.

10 Discussion of trade-offs and reflections on implementation choice

When I do SVD image compression, I see that choosing k is not easy. If k is small, the image look kind of blur but the size get small. If k is big, the image look more clear but take more space, so not good for compression.

From this, I tried many k values and see which one give better output. I think best way is to take k in middle, so image not too blur and also not too big in size. It depend on what we want, good quality or small size.

11 Visualization of eigen values

Visualizing singular values helps you see how image information is spread across different ranks, and choose a suitable k for compression.

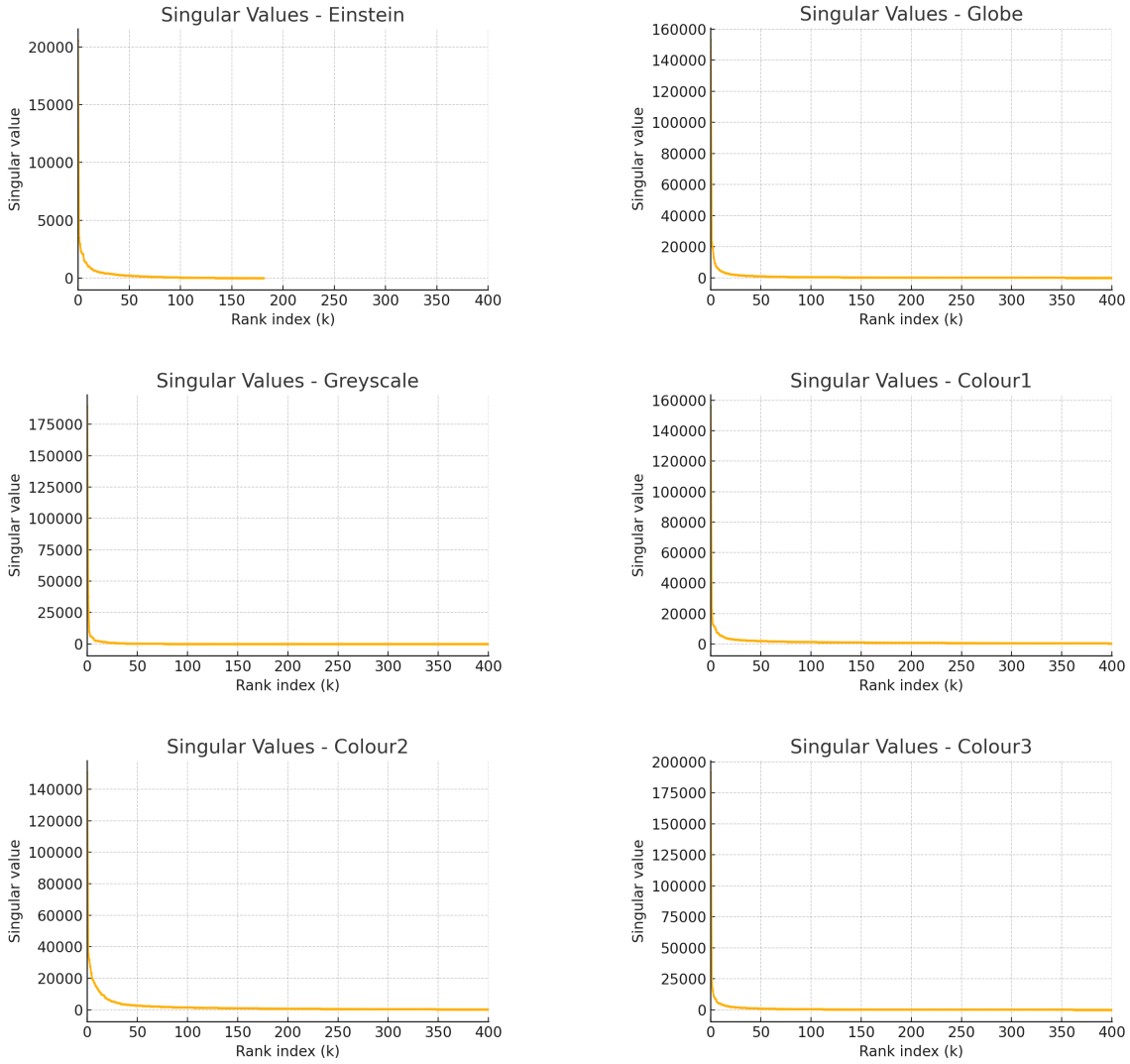


Figure 8: Visualization