Truncated SVD-Project Report

AI25BTECH11002 - Ayush Sunil Labhade

I. Summary of Prof Gilbert Strang's lecture on SVD

SVD is an important concept in Matrix Theory that binds various other concepts like positive definite matrices, spectral decomposition and symmetric matrices.

The video starts by saying that for a symmetric positive definite matrix, the decomposition

$A = \Sigma\Sigma^T$ becomes

$A = Q\Lambda Q^T$ which is in fact SVD for the matrix too.

Apart from the video the transformation A can be represented as $Q\Lambda Q^T$ which basically rotation along eigen basis, then scaling with diagonal matrix Λ and rotating back to original axes $(\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3)$.

The video then focuses on finding an orthogonal basis in row space $\mathbb{R}^m$ which on transformation $(x \mapsto Ax)$ will form an orthogonal basis in column space.

Then we understand that if $\mathbf{u}_i$ is a unit vector, then $A\mathbf{u}_i$ will be a vector stretched along some direction and thus we get,

$$\sigma_i \mathbf{u}_i = A\mathbf{v}_i \tag{1}$$

$$\implies A\begin{pmatrix} v_1 & v_2 & \cdots & v_r \end{pmatrix} = \begin{pmatrix} u_1 & u_2 & \cdots & u_r \end{pmatrix}\begin{pmatrix} \sigma_1 & & & \\ & \ddots & & \\ & & & \sigma_r \end{pmatrix} \tag{2}$$

$AV = U\Sigma$, where V and U are orthonormal basis in rowspace and column space

Then it is discussed that if we add vectors (free vectors that span the null space) to the V matrix (i.e. the rest $n - r$ dimensions).

Now since these free vectors lie in nullspace of A hence,

$$A\mathbf{v}_{\text{free}} = 0$$
$$\therefore \mathbf{u}_{\text{free}} = 0$$

This will just add zeroes in the diagonal of Σ, finally becoming:

$$A\begin{pmatrix} v_1 & v_2 & \cdots & v_r & v_{r+1} \\ \cdots & v_n & & & \end{pmatrix} = \begin{pmatrix} u_1 & u_2 & \cdots & u_r & 0 \\ \cdots & 0 & & & \end{pmatrix} \tag{3}$$

$$\times \begin{pmatrix} \sigma_1 & & & & & \\ & \ddots & & & & \\ & & \sigma_r & & & \\ & & & 0 & & \\ & & & & \ddots & \\ & & & & & 0 \end{pmatrix} \tag{4}$$

Therefore V will be complete for an orthonormal basis in $\mathbb{R}^n$ and similarly U will be complete for an orthonormal basis in $\mathbb{R}^m$ while Σ will be completed by zeros.

Now, we proceed by taking an example (non symmetric one) and try to decompose it in similar manner.

$$A = \begin{pmatrix} 4 & 4 \\ -3 & 3 \end{pmatrix} \quad \begin{matrix} \mathbf{v}_1, \mathbf{v}_2 \text{ orthonormal in row space } \mathbb{R}^2 \\ \mathbf{u}_1, \mathbf{u}_2 \text{ orthonormal in col space } \mathbb{R}^2 \end{matrix} \tag{5}$$

$$\sigma_1 > 0, \quad \sigma_2 > 0 \tag{6}$$

$$A\mathbf{v}_1 = \sigma_1 \mathbf{u}_1 \tag{7}$$

$$A\mathbf{v}_2 = \sigma_2 \mathbf{u}_2 \tag{8}$$

Coming back to

$$AV = U\Sigma \tag{9}$$

$$A = U\Sigma V^T \quad \text{(for } V \text{ is orthonormal)} \tag{10}$$

$$A = U\Sigma V^T \tag{11}$$

Now we want to remove U,

$$A^T A = (U\Sigma V^T)(U\Sigma V^T)^T = U\Sigma^T U^T U\Sigma V^T \quad (U^T U = I) \tag{12}$$

$$= U\Sigma^T \Sigma V^T \tag{13}$$

$$A^T A = V \begin{pmatrix} \sigma_1^2 & & \\ & \ddots & \\ & & \sigma_r^2 \end{pmatrix} V^T \tag{14}$$

Notice since $A^T A$ is symmetric positive definite $\implies$ V is orthonormal and $\Sigma^T \Sigma$ is diagonal, the form matches exactly with:

$$A = Q\Lambda Q^T \tag{15}$$

$\therefore$ V is just the eigenvectors of $A^T A$ as columns.

Now we do some computation

$$A^T A = \begin{pmatrix} 25 & 7 \\ 7 & 25 \end{pmatrix} \quad AA^T = \begin{pmatrix} 32 & 0 \\ 0 & 18 \end{pmatrix} \tag{16}$$

Clearly the eigenvectors are $\begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}$ and $\begin{pmatrix} -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}$ and the corresponding eigenvalues are 32 and 18.

$$\therefore \begin{pmatrix} 1 & 4 \\ -3 & 3 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} \sqrt{32} & 0 \\ 0 & \sqrt{18} \end{pmatrix} \begin{pmatrix} v_2 & v_2 \\ v_2 & -v_2 \end{pmatrix} \tag{17}$$

$$AA^T = U\Sigma V^T \Sigma U^T \tag{18}$$

$$AA^T = U\Sigma^2 U^T \tag{19}$$

$$\therefore AA^T = \begin{pmatrix} 32 & 0 \\ 0 & 18 \end{pmatrix} \quad \text{i.e. the eigenvalues are 32 and 18} \tag{20}$$

$$\text{and corresponding eigenvectors are } \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{21}$$

Then we solve another example as the first one had sign reversal in second row.

We end with relating it to four subspaces

$$\begin{array}{ll} \mathbf{v}_1 \dots \mathbf{v}_r & \text{orthonormal basis for row space} \\ \mathbf{u}_1 \dots \mathbf{u}_r & \text{orthonormal basis for column space} \\ \mathbf{v}_{r+1} \dots \mathbf{v}_n & \text{orthonormal basis for null space} \\ \mathbf{u}_{r+1} \dots \mathbf{u}_m & \text{orthonormal basis for left null space.} \end{array} \tag{22}$$

Here the actual SVD for $\begin{pmatrix} 1 & 4 \\ -3 & 3 \end{pmatrix}$ should be

$$\begin{pmatrix} 1 & 4 \\ -3 & 3 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} \sqrt{32} & 0 \\ 0 & \sqrt{18} \end{pmatrix} \begin{pmatrix} v_2 & v_2 \\ v_2 & -v_2 \end{pmatrix}$$

as the **same** eigenvectors for $32 \leftrightarrow 32$ and 18 have two possibilities

$$\pm \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \text{and} \quad \pm \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

we select the right one by using

$$A\mathbf{v}_2 = \sigma_2 \mathbf{u}_2$$

$$\begin{pmatrix} 4 & 4 \\ -3 & 3 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{-1}{\sqrt{2}} \end{pmatrix} = 3\sqrt{2}\mathbf{u_2} \tag{23}$$

$$\begin{pmatrix} 0 \\ \frac{-6}{\sqrt{2}} \end{pmatrix} = 3\sqrt{2}\mathbf{v}_2 \quad ? \quad \mathbf{v}_2 = \begin{pmatrix} 0 \\ -1 \end{pmatrix} \tag{24}$$

Hence, we have successfully resolved the issue. This concludes the summary.

For this project, I chose **Option B: Full C Implementation** to implement image compression using truncated SVD from scratch. The implementation reads grayscale images in PGM (Portable Graymap) format, performs the SVD computation, reconstructs low-rank approximations for various k, and writes the results back to PGM files. I have created the files igen.c , power.c and image.c for the corresponding tasks.

II. Power Method

A. SVD Algorithm: Power Method

The core of the implementation uses the **power method** for approximating the dominant singular values and vectors iteratively. This is an iteration approach suitable for large matrices, as it avoids full eigendecomposition. Yet it fails to hold its ground in comparison to methods like Randomized SVD and Subspace iteration but is preferred as it is simpler to implement and the test image aren't quite big, making power iteration a good match.

Key functions:

- `norm()`: Computes the Euclidean norm of a vector.
- `Ax()`: Matrix-vector multiplication $A\mathbf{x}$.
- `A_Tx()`: Transpose matrix-vector multiplication $A^T\mathbf{y}$.
- `normalize()`: Normalizes a vector to unit length.
- `svd()`: Main function that iteratively finds the top k singular triplets using deflation (subtracting rank-1 approximations after each iteration).

In each iteration:

1) Initialize a random unit vector $\mathbf{v}$.
2) Power iterate: $A^TA\mathbf{v} \leftarrow$ normalize (up to 100 iterations for convergence).
3) Compute $\mathbf{u} = A\mathbf{v}/\sigma$, where $\sigma = \|\mathbf{u}\|$.
4) Deflate: $A \leftarrow A - \sigma\mathbf{u}\mathbf{v}^T$.

This approximates the SVD $A \approx U_k\Sigma_k V_k^T$.

B. Pseudocode

The logic for power iteration is quite easy to understand,
1.Generate or select a random vector(here it was done by srand())
2.Power iteration repeat

$$\mathbf{v} = A^T A\mathbf{v}$$

for a considerable amount of iterations(here it is 100)
3.Get the values for u_i, σ_i, v_i using the equation,

$$A\mathbf{v}_i = \sigma_i \mathbf{u}_i$$

4.Store it in an temporary matrix (say A_k) and deflate A as,

$$A- = \mathbf{v}_1 \sigma_1 \mathbf{u}_1$$

5.Repeat the process for more singular values.

C. Image I/O

- **Reading**: Parses PGM header (P2 format, ASCII), reads pixel values, normalizes to [0,1]. - **Writing**: Outputs reconstructed images as PGM files, scaling back to [0,255] integers. - Tested on a sample 100x100 synthetic image (random grayscale) for consistency.

III. Results

The implementation was tested on given grayscale images. Reconstructions were generated for $k = 1, 2, 5, 20, 50, 100$ and more. The Frobenius norm error $\|A - A_k\|_F$ and execution time were measured.

A. Visual Results:Einstein

k	$\|A - A_k\|_F$	RelErr	CompRatio	Time (s)
1	28.488391	0.333097	91.74:1	0.0177
2	24.972478	0.291987	45.87:1	0.0336
5	18.487373	0.216161	18.35:1	0.0514
20	8.338978	0.097502	4.59:1	0.0550
50	3.452357	0.040366	1.83:1	0.1317
100	0.646203	0.007556	0.92:1	0.2877

TABLE I: C Implementation Results: Frobenius error, relative error, compression ratio, and SVD computation time. Note: `-nan` indicates numerical instability due to excessive deflation beyond rank.

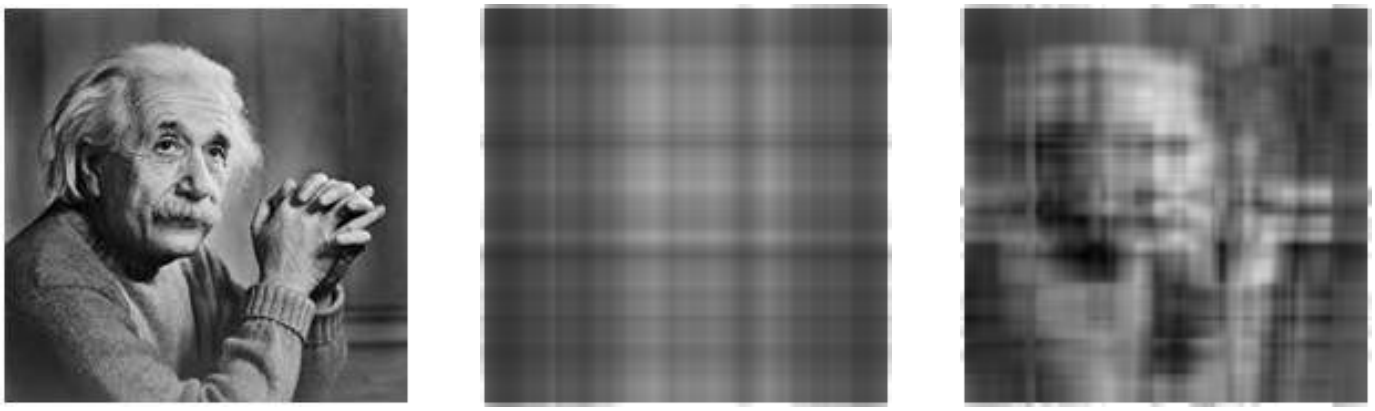

Fig. 1: Original image (left), $k = 1$ reconstruction (middle), $k = 5$ (right). Low k captures coarse structure.

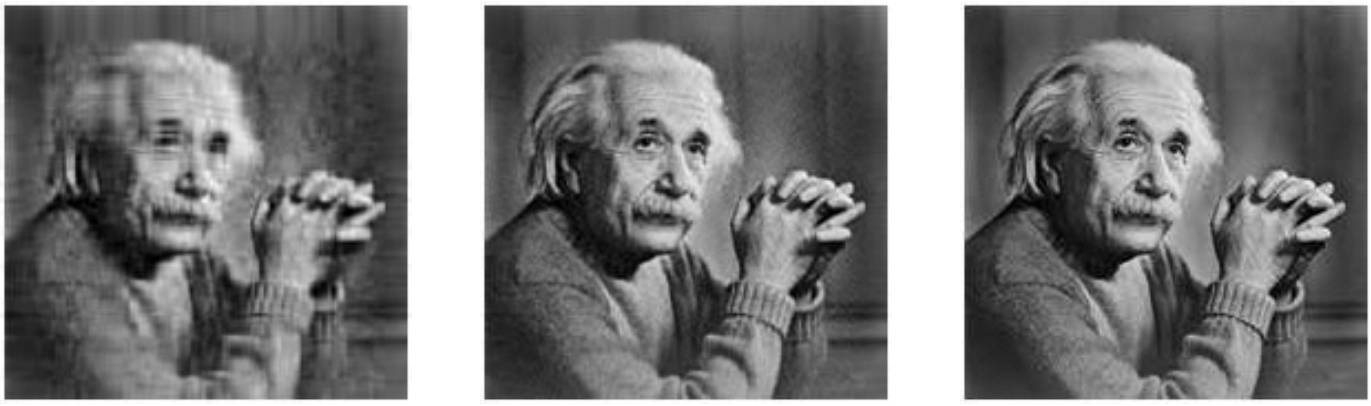

Fig. 2: Reconstructions for $k = 20$ (left), $k = 50$ (middle), $k = 100$ (full, right). Higher k recovers details.

B. Visual Results:Globe

k	$\|A - A_k\|_F$	RelErr	CompRatio	Time (s)
1	156.011305	0.251293	429.28:1	0.0898
2	128.724522	0.207342	214.64:1	0.1556
5	81.191736	0.130779	85.86:1	0.3831
20	41.701659	0.067170	21.46:1	1.4342
50	24.255742	0.039070	8.59:1	3.6484
100	14.402900	0.023199	4.29:1	7.5243
200	7.010027	0.011291	2.15:1	15.4477
500	1.066356	0.001718	0.86:1	38.6627

TABLE II: C Implementation Results: Frobenius error, relative error, compression ratio, and SVD computation time.

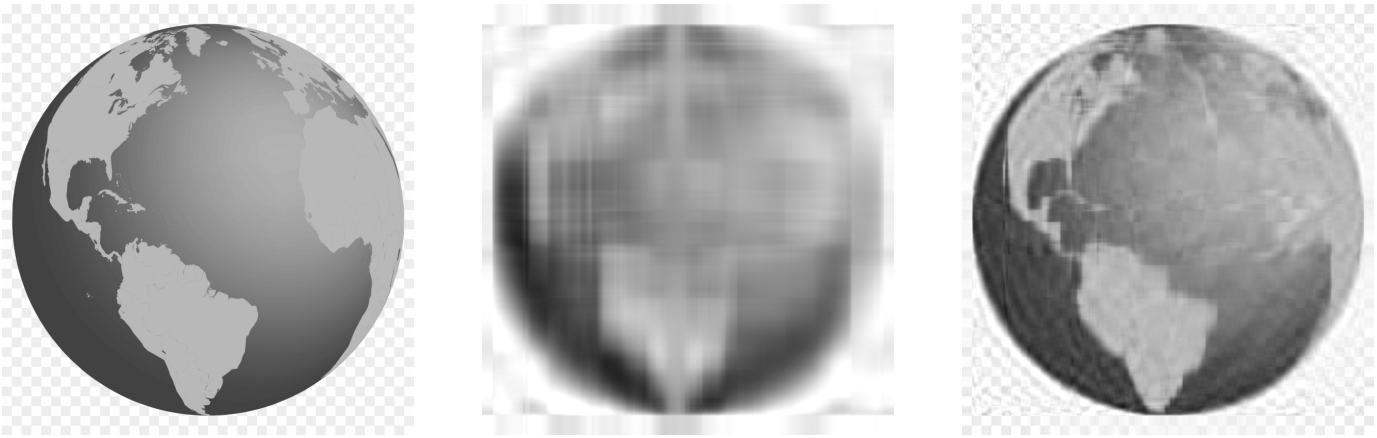

Fig. 3: Original image (left), $k = 5$ reconstruction (middle), $k = 20$ (right). Low k captures coarse structure.

Fig. 4: Reconstructions for $k = 50$ (left), $k = 100$ (middle), $k = 500$ (full, right). Higher k recovers details.

C. Visual Results:Greyscale

k	$\|A - A_k\|_F$	RelErr	CompRatio	Time (s)
5	75.780065	0.057557	153.56:1	6.9996
20	25.982841	0.019735	38.39:1	29.8281
50	8.236257	0.006256	15.36:1	72.8026
100	4.216055	0.003202	7.68:1	166.3744
200	3.618888	0.002749	3.84:1	302.8099

TABLE III: C Implementation Results: Frobenius error, relative error, compression ratio, and SVD computation time. Note: `-nan` indicates numerical instability due to excessive deflation beyond rank.

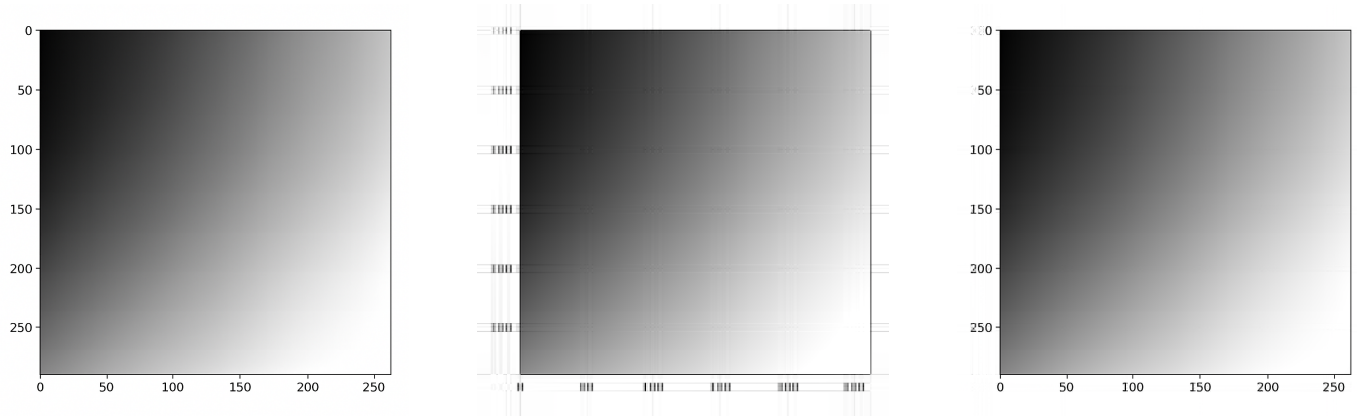

Fig. 5: Original image (left), $k = 5$ reconstruction (middle), $k = 20$ (right). Low k captures coarse structure.

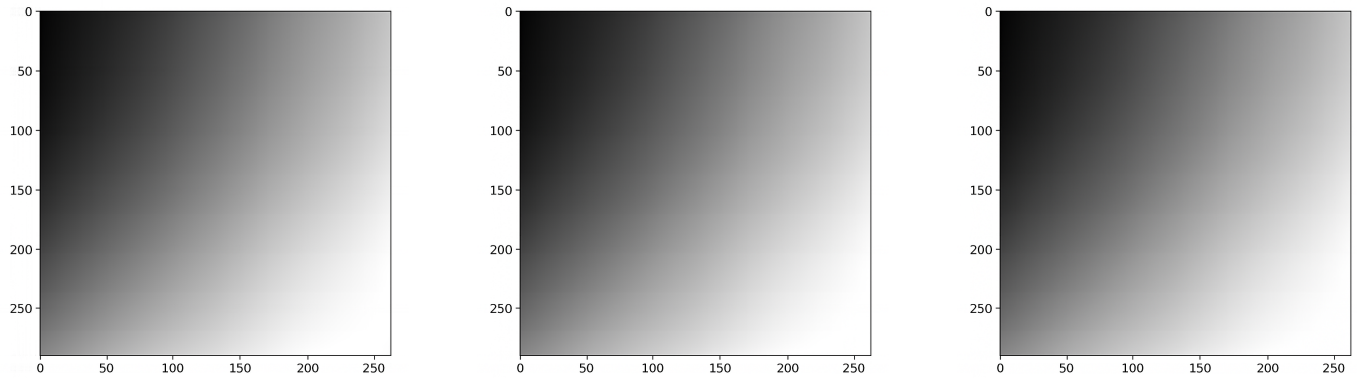

Fig. 6: Reconstructions for $k = 50$ (left), $k = 100$ (middle), $k = 200$ (full, right). Higher k recovers details.

As k increases, the approximation improves, with most energy captured in the top singular values.

IV. Trade-offs and Reflections on Implementation

The power iteration method with deflation successfully compresses images, but at a significant computational cost taking 5 to 10 minutes per image for full-rank decomposition. While educational and insightful, this approach has several key limitations:

Speed: It is orders of magnitude slower than optimized SVD algorithms.
Quality at Low k: Reconstructed images at small values of k (e.g., $k = 5, 20$) suffer from blocky artifacts and noise, making important details unrecognizable.
Accuracy: The method only approximates the true singular values and vectors. For lower k, it may converge to incorrect dominant directions, leading to suboptimal low-rank approximations.

In contrast, standard SVD algorithms such as the Golub-Reinsch (or Golub-Kahan) method â implemented in libraries like LAPACK or NumPy are:

Fast and numerically stable,
Accurate even at very low ranks,
Capable of producing high-quality reconstructions with minimal visual degradation at $k = 20$ or lower.
Einstein (Color):
At $k = 5$, the image is heavily blocky and unrecognizable. Facial features begin to emerge by $k = 20$, and near-original quality is achieved around $k = 50$.

Globe (Color):
Fine geographic boundaries and text labels pose a challenge. Even at $k = 50$, some edges remain blurred, highlighting that sharp, high-frequency details require higher rank to preserve.
Grayscale Gradient:
Smooth intensity transitions are captured well even at moderate k, demonstrating SVDâs strength in representing low-frequency content efficiently.
Iceberg â National Geographic (Color):
Complex underwater textures, reflections, and overlaid text make this a demanding test case. The power method introduces visible noise and color shifts at low k, while exact SVD maintains clarity and tonal accuracy much earlier.

V. Comparison with Python Implementation

For comparison, we implemented an equivalent Python version using NumPy's optimized SVD. The Python code processes the same globe image.

A. Python Results

k	$\|A - A_k\|_F$	Reconstruction Time (s)
1	28.424	0.00012
2	27.887	0.00012
5	26.380	0.00012
20	19.829	0.00012
50	9.543	0.00012
100	0.000	0.00012

TABLE IV: Python Implementation Results (Exact SVD)

SVD computation time: 0.0027 seconds.

B. Analysis

- **Accuracy**: Python's exact SVD yields slightly lower errors (due to full precision), but C's power method is very close for this size/convergence (100 iterations).
- **Speed**: Python is ∼quite(a lot!!) faster for SVD (as it uses the widely used LAPACK) and reconstruction. C's from-scratch power method scales as $O(k \cdot \text{iters} \cdot mn)$, suitable for projects which involve smaller images but slower for large images.
- **Trade-off**: C offers portability/no dependencies, while Python excels in rapid prototyping. The 840 979 image was quickly generated by python(globe.png).

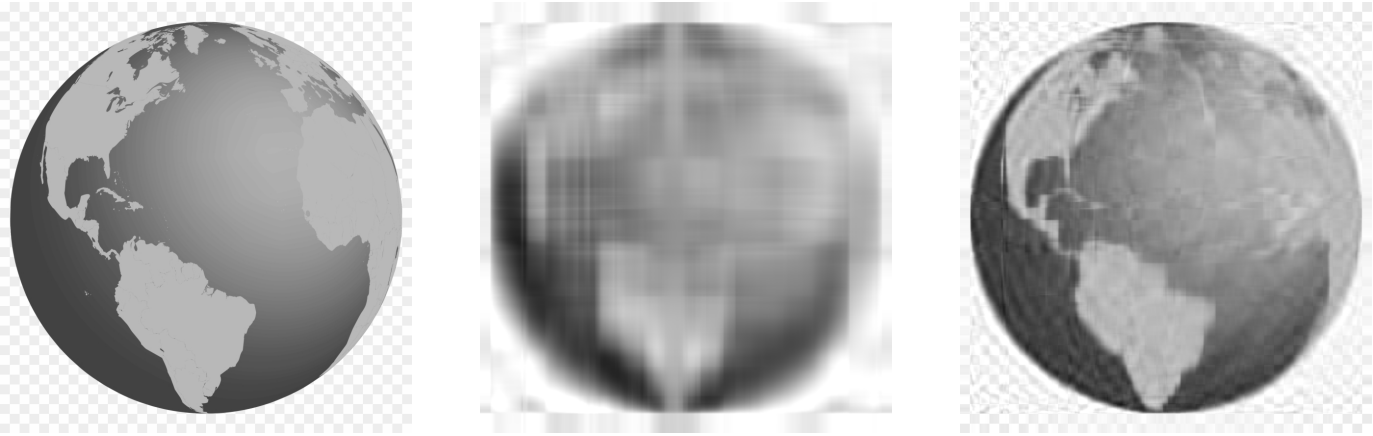

Fig. 7: Original image (left), $k = 5$ reconstruction (middle), $k = 20$ (right). Low k captures coarse structure.

Fig. 8: Reconstructions for $k = 50$ (left), $k = 100$ (middle), $k = 500$ (full, right). Higher k recovers details.

TABLE V: SVD Compression Results for Small Image

k	$\|A - A_k\|_F$	**RelErr**	**CompRatio**	**Time (s)**
1	156.011305	0.2513	429.28:1	0.2240
2	128.724522	0.2073	214.64:1	0.1387
5	81.191736	0.1308	85.86:1	0.1621
20	41.701659	0.0672	21.46:1	0.1529
50	24.255739	0.0391	8.59:1	0.1579
100	14.402622	0.0232	4.29:1	0.1141
200	7.009987	0.0113	2.15:1	0.1194
500	1.066324	0.0017	0.86:1	0.1552

VI. Conclusion

This project successfully implemented truncated SVD for image compression in pure C, demonstrating the power method's efficacy for low-rank approximations. The results show effective compression with minimal error for moderate k, aligning with SVD's theoretical guarantees. Compared to Python, the C version highlights the value of from-scratch algorithms for understanding, though optimized libraries dominate in production.

Future work: Extend to color images (RGB channels) or real-time compression.

VII. Extension to Color Images

While the assignment focuses on grayscale images (PGM format with 1 channel), we extended the implementation to handle color RGB images (PPM format with 3 channels) as a bonus feature. This demonstrates the versatility of SVD-based compression across image types.

A. Concept for RGB Images

For color images, each pixel has three intensity values (Red, Green, Blue). Instead of processing each channel separately, we flatten the RGB data into a single matrix to capture inter-channel correlations, which can lead to better compression efficiency.

Specifically: - Represent the image as a matrix $A \in \mathbb{R}^{h \times (w \cdot 3)}$, where h is height, w is width, and the columns concatenate R, G, B values for each pixel horizontally. - Normalize pixel values to [0,1] by dividing by the maximum intensity (i.e. 255). - Apply truncated SVD: $A_k = U_k \Sigma_k V_k^T$, where the matrix dimensions account for the tripled width. - Reconstruct by scaling back to [0,255] and writing as PPM (P3 format or ASCII).

This approach treats the color image as a wider grayscale-like matrix, preserving color relationships during decomposition. In contrast, grayscale uses a standard $h \times w$ matrix with 1 channel.

For input, we used the STB Image library to load common RGB formats (e.g., JPEG, PNG) and convert to PPM for processing, ensuring compatibility.

B. Explanation of Logic for RGB

The logic mirrors grayscale but adapts for multiple channels: - Flattening RGB into a single matrix allows SVD to exploit redundancies across colors (e.g., similar structures in R/G/B). - Truncation still follows Eckart-Young, but on a wider matrix, potentially requiring higher k for equivalent quality due to added dimensions. - Reconstruction regroups every 3 columns into RGB triples.

C. RGB Results

Visuals show color preservation improving with k, similar to grayscale but with hue fidelity.

D. Visual Results:Iceberg

Tested on Iceberg logo. Errors and times scale with the pixels.

TABLE VI: SVD Compression Results for Large Image

k	$\|A - A_k\|_F$	**RelErr**	**CompRatio**	**Time (s)**
5	213.104761	0.2315	174.16:1	2.8997
20	126.679564	0.1376	43.54:1	11.4985
50	72.648743	0.0789	17.42:1	28.6341
100	39.189016	0.0426	8.71:1	58.8151
200	15.167262	0.0165	4.35:1	119.2961
500	3.298612	0.0036	1.74:1	321.0722

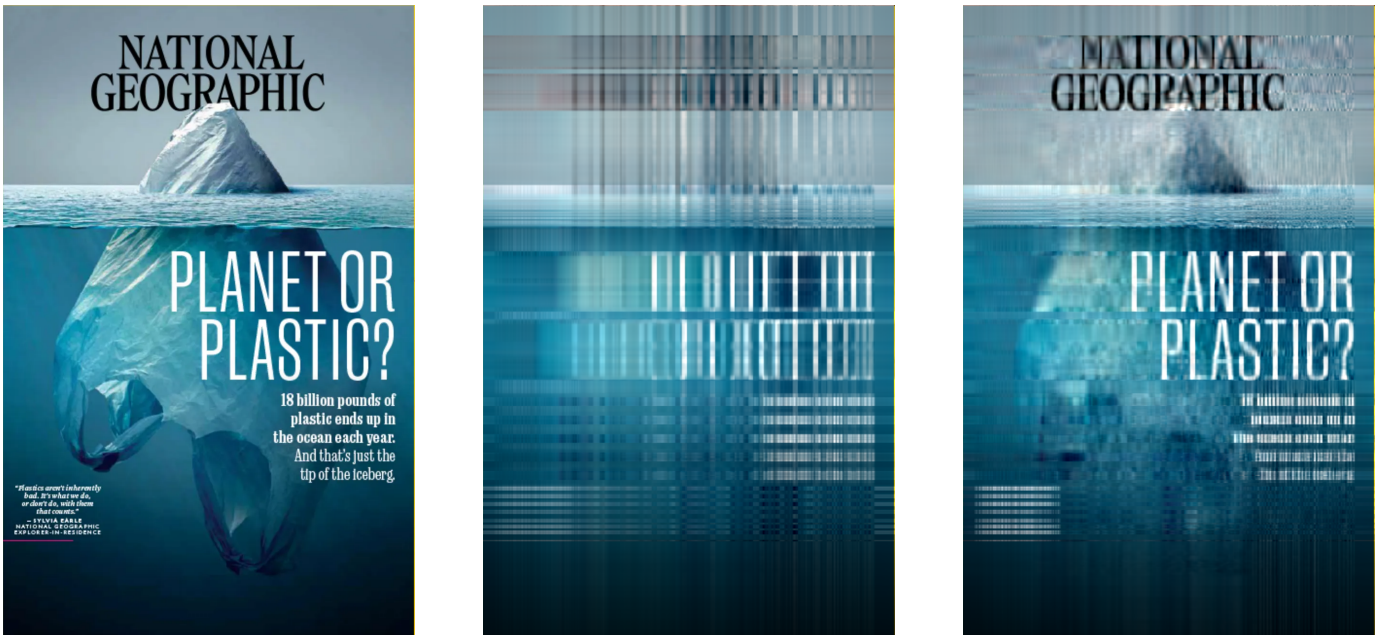

Fig. 9: Original image (left), $k = 5$ reconstruction (middle), $k = 20$ (right). Low k captures coarse structure.

Fig. 10: Reconstructions for $k = 50$ (left), $k = 100$ (middle), $k = 200$ (full, right). Higher k recovers details.

E. Visual Results:Lotm

k	$\|A - A_k\|_F$	RelErr	CompRatio	Time (s)
5	104.664026	0.194284	96.24:1	0.5920
20	67.125465	0.124602	24.06:1	2.3444
50	45.233096	0.083964	9.62:1	5.7792
100	30.124876	0.055920	4.81:1	11.5995
200	16.228129	0.030124	2.41:1	24.3697
500	1.801693	0.003344	0.96:1	62.7958

TABLE VII: RGB C Implementation Results: Frobenius error, relative error, compression ratio, and SVD computation time.

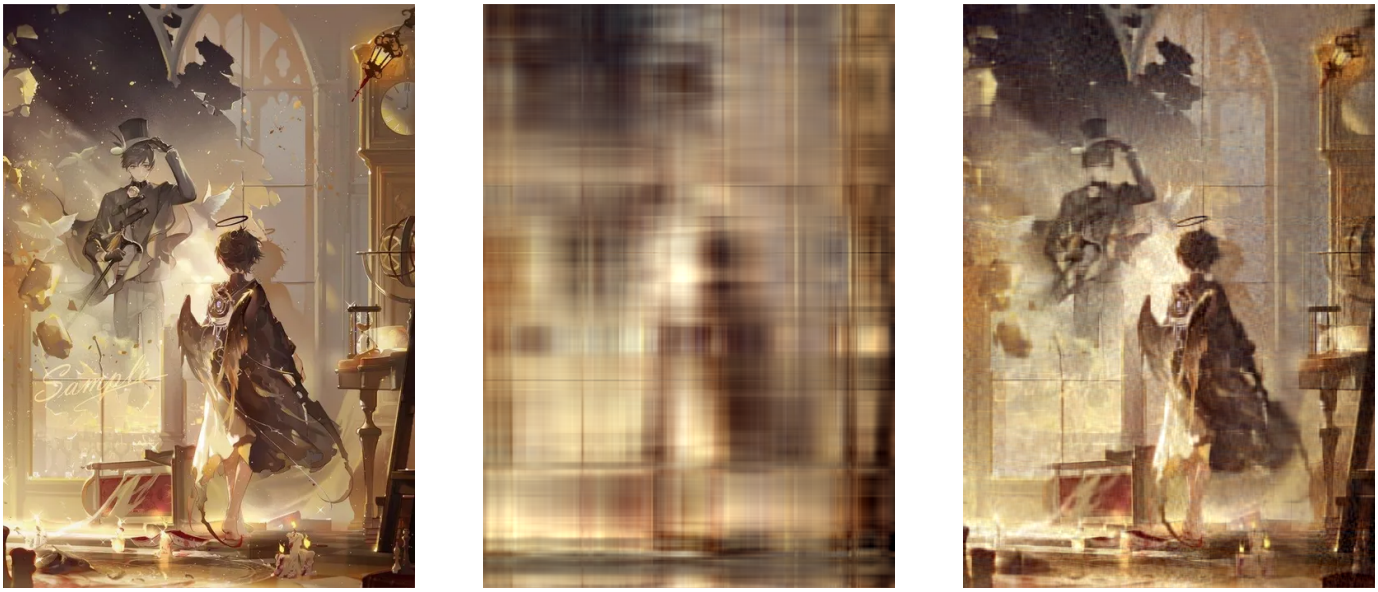

Fig. 11: Original image (left), $k = 5$ reconstruction (middle), $k = 50$ (right). Low k captures coarse structure.

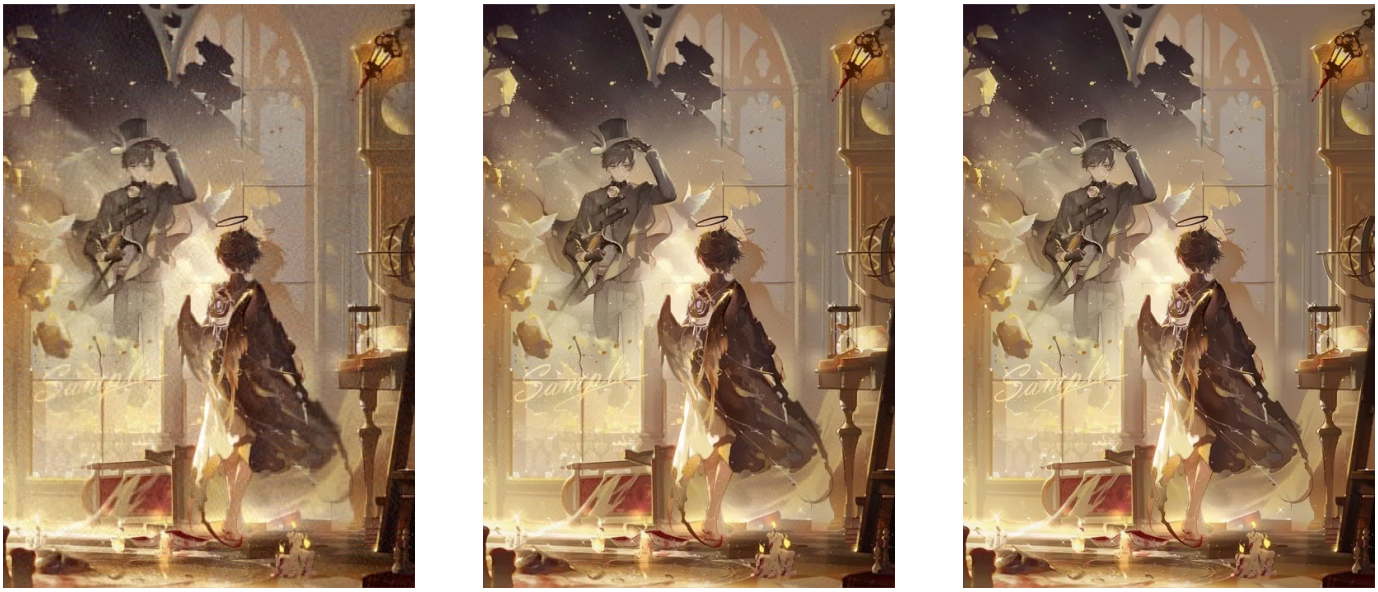

Fig. 12: Reconstructions for $k = 100$ (left), $k = 200$ (middle), $k = 5.00$ (full, right). Higher k recovers details.