Image Compression using Singular Value Decomposition (SVD) with Power Iteration Method

AI25BTECH11034 - Sujal Chauhan

November 8, 2025

1 Introduction

Image compression is a fundamental technique in digital image processing that reduces the storage requirements and transmission bandwidth of images while maintaining acceptable quality. This report explores the application of Singular Value Decomposition (SVD) for image compression, implementing a power iteration-based algorithm to compute the dominant singular values and vectors efficiently.

2 Singular Value Decomposition: Theory

2.1 Mathematical Foundation

Based on Gilbert Strang's exposition on SVD, any real matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ can be decomposed as:

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \tag{1}$$

where:

- $\mathbf{U} \in \mathbb{R}^{m \times m}$ is an orthogonal matrix containing left singular vectors
- $\mathbf{\Sigma} \in \mathbb{R}^{m \times n}$ is a diagonal matrix with singular values $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$
- $\mathbf{V} \in \mathbb{R}^{n \times n}$ is an orthogonal matrix containing right singular vectors

2.2 Key Properties (Gilbert Strang's Perspective)

2.2.1 Four Fundamental Subspaces

The SVD reveals the four fundamental subspaces of a matrix:

1. **Column Space**: Spanned by the first r columns of $\mathbf{U}$
2. **Row Space**: Spanned by the first r columns of $\mathbf{V}$
3. **Null Space**: Spanned by the last $n - r$ columns of $\mathbf{V}$
4. **Left Null Space**: Spanned by the last $m - r$ columns of $\mathbf{U}$

2.2.2 Best Rank-k Approximation

For image compression, we use the truncated SVD:

$$\mathbf{A}_k = \sum_{i=1}^{k} \sigma_i \mathbf{u}_i \mathbf{v}_i^T \tag{2}$$

This provides the **best rank-k approximation** to $\mathbf{A}$ in both the Frobenius and spectral norms:

$$\|\mathbf{A} - \mathbf{A}_k\|_F = \sqrt{\sum_{i=k+1}^{r} \sigma_i^2} \tag{3}$$

2.2.3 Geometric Interpretation

Each singular value σ_i represents the "importance" or "energy" of the corresponding singular vector pair $(\mathbf{u}_i, \mathbf{v}_i)$. By keeping only the largest k singular values, we capture the most significant patterns in the image while discarding less important details.

2.3 Connection to Eigenvalues

The singular values of $\mathbf{A}$ are the square roots of eigenvalues of $\mathbf{A}^T\mathbf{A}$ (or $\mathbf{A}\mathbf{A}^T$):

$$\mathbf{A}^T\mathbf{A}\mathbf{v}_i = \sigma_i^2\mathbf{v}_i \tag{4}$$

$$\mathbf{A}\mathbf{A}^T\mathbf{u}_i = \sigma_i^2\mathbf{u}_i \tag{5}$$

This relationship forms the basis of our power iteration algorithm.

3 Algorithm Comparison and Selection

3.1 Available SVD Algorithms

Table 1: Comparison of SVD Algorithms

Algorithm	**Time Complexity**	**Memory**	**Partial SVD**
Full SVD (LAPACK)	$O(mn^2)$	$O(mn)$	No
Jacobi Method	$O(mn^2)$	$O(mn)$	No
Divide-and-Conquer	$O(mn^2)$	$O(mn)$	No
Lanczos (Tridiagonal)	$O(mnk)$	$O(nk)$	Yes
Power Iteration	$O(mnk \cdot \text{iter})$	$O(nk)$	Yes
Randomized SVD	$O(mnk)$	$O(nk)$	Yes

3.2 Why Power Iteration with Deflation?

We chose the **Power Iteration with Deflation** method for the following reasons:

1. **Simplicity**: Easy to implement and understand compared to Lanczos tridiagonalization
2. **Memory Efficiency**: Only requires $O(nk)$ storage, not the full matrix decomposition
3. **Partial SVD**: Computes only the k largest singular values needed for compression
4. **Stability**: More numerically stable than full Lanczos for our image compression application
5. **Sequential Nature**: Computes singular values one at a time, allowing early termination

3.3 Mathematical Principle: Power Iteration

The power iteration method exploits the fact that repeated multiplication by a matrix amplifies the dominant eigenvector:

$$\mathbf{v}^{(t+1)} = \frac{\mathbf{A}^T\mathbf{A}\mathbf{v}^{(t)}}{\|\mathbf{A}^T\mathbf{A}\mathbf{v}^{(t)}\|} \tag{6}$$

As $t \to \infty$, $\mathbf{v}^{(t)}$ converges to the dominant eigenvector of $\mathbf{A}^T\mathbf{A}$, which is the first right singular vector $\mathbf{v}_1$.

3.3.1 Convergence Analysis

If the eigenvalues satisfy $|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|$, then:

$$\|\mathbf{v}^{(t)} - \mathbf{v}_1\| = O\left(\left|\frac{\lambda_2}{\lambda_1}\right|^t\right) \tag{7}$$

The convergence rate depends on the ratio $|\lambda_2/\lambda_1|$, known as the spectral gap.

3.3.2 Deflation for Multiple Singular Values

After computing $(\sigma_1, \mathbf{u}_1, \mathbf{v}_1)$, we deflate the matrix:

$$\mathbf{A}^T\mathbf{A} \leftarrow \mathbf{A}^T\mathbf{A} - \sigma_1^2\mathbf{v}_1\mathbf{v}_1^T \tag{8}$$

This removes the contribution of the first singular component, allowing power iteration to find the next largest singular value.

4 Implementation Details

4.1 Algorithm Pseudocode

Algorithm 1 SVD via Power Iteration with Deflation

Require: Matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, rank k
Ensure: Singular values $\{\sigma_i\}$, matrices $\mathbf{U}_k$, $\mathbf{V}_k$

Compute $\mathbf{M} \leftarrow \mathbf{A}^T\mathbf{A}$
for $i = 1$ to k **do**
    Initialize random vector $\mathbf{v}^{(0)} \in \mathbb{R}^n$
    $\mathbf{v}^{(0)} \leftarrow \mathbf{v}^{(0)}/\|\mathbf{v}^{(0)}\|$
    for $t = 1$ to max_iter **do**
        $\mathbf{w} \leftarrow \mathbf{M}\mathbf{v}^{(t-1)}$
        $\mathbf{v}^{(t)} \leftarrow \mathbf{w}/\|\mathbf{w}\|$
        if $\|\mathbf{v}^{(t)} - \mathbf{v}^{(t-1)}\| < \epsilon$ **then**
            break
        end if
    end for
    $\lambda_i \leftarrow (\mathbf{v}^{(t)})^T\mathbf{M}\mathbf{v}^{(t)}$ ▷ Rayleigh quotient
    $\sigma_i \leftarrow \sqrt{|\lambda_i|}$
    $\mathbf{V}_k[:, i] \leftarrow \mathbf{v}^{(t)}$
    $\mathbf{U}_k[:, i] \leftarrow \mathbf{A}\mathbf{v}^{(t)}/\sigma_i$
    $\mathbf{M} \leftarrow \mathbf{M} - \lambda_i\mathbf{v}^{(t)}(\mathbf{v}^{(t)})^T$ ▷ Deflation
end for
return $\{\sigma_i\}, \mathbf{U}_k, \mathbf{V}_k$

4.2 Computational Complexity Analysis

- **Matrix-vector multiplication**: $O(mn)$ per iteration
- **Power iterations per singular value**: Typically 50-100 iterations
- **Total complexity**: $O(mnk \cdot \text{iter})$ where iter $\approx$ 50-100
- **Space complexity**: $O(mn + nk)$ for $\mathbf{A}^T\mathbf{A}$ and singular vectors

For an image of size 512×512 with $k = 50$:

- Operations: $\approx 512 \times 512 \times 50 \times 100 = 1.3$ billion FLOPs
- Memory: $\approx 262,144 + 25,600 = 287,744$ doubles ≈ 2.2 MB

4.3 Code Structure

The implementation is organized into modular components:

- `pgm_io.c/h`: Image I/O operations (PGM, JPG, PNG)
- `matrix.c/h`: Matrix operations and linear algebra utilities
- `lanczos.c/h`: Power iteration SVD algorithm
- `svd_compress.c/h`: Image compression and reconstruction
- `main.c`: Command-line interface and workflow coordination

4.4 Key Implementation Features

1. **Multi-format Support**: Automatic detection of JPG, PNG, and PGM formats using stb_image library
2. **Grayscale Conversion**: Color images automatically converted to grayscale
3. **Error Metrics**: Computation of total error, average pixel error, and error percentage
4. **Memory Management**: Efficient allocation and deallocation to prevent memory leaks

5 Compression Formula

The compression ratio is calculated as:

$$\text{Compression Ratio} = \frac{\text{Original Size}}{\text{Compressed Size}} = \frac{m \times n}{mk + k + nk} = \frac{mn}{k(m + n + 1)} \tag{9}$$

For large images where $m, n \gg 1$:

$$\text{Compression Ratio} \approx \frac{mn}{k(m + n)} \tag{10}$$

The storage required is:

$$\text{Storage} = \frac{k(m + n + 1)}{mn} \times 100\% \tag{11}$$

6 Experimental Results

6.1 Test Image

We tested our algorithm on a grayscale image (Einstein portrait) with dimensions 182×186 pixels.

6.2 Einstein

Table 2: Compression Results for Various Rank Values

k	Ratio	Storage %	Total Error	Avg Error	Error %
5	45.91:1	2.18%	625386.41	18.35	7.24%
10	23.36:1	4.28%	41900.94	12.37	4.85%
20	11.96:1	8.36%	268906.79	7.94	3.12%
50	4.94:1	20.24%	116644.42	3.44	1.35%
100	2.51:1	39.84%	23450	0.69	0.27%
150	1.69:1	59.17%	1794.85	0.05	0.02%
200	1.28:1	78.13%	367701.66	10.86	4.26%

6.3 Globe

Table 3: Compression Results for Various Rank Values

k	Ratio	Storage %	Total Error	Avg Error	Error %
5	45.91:1	2.18%	1,245,832	36.82	14.44%
10	23.36:1	4.28%	856,421	25.32	9.93%
20	11.96:1	8.36%	524,187	15.49	6.07%
50	4.94:1	20.24%	198,743	5.87	2.30%
100	2.51:1	39.84%	67,234	1.99	0.78%
150	1.69:1	59.17%	23,451	0.69	0.27%
200	1.28:1	78.13%	8,923	0.26	0.10%

6.4 Greyscale

Table 4: Compression Results for Various Rank Values

k	Ratio	Storage %	Total Error	Avg Error	Error %
5	45.91:1	2.18%	2733325	2.60	1.02%
10	23.36:1	4.28%	1792445	1.70	0.67%
20	11.96:1	8.36%	896198	0.85	0.34%
50	4.94:1	20.24%	561192	0.53	0.21%
100	2.51:1	39.84%	397066	0.37	0.15%
150	1.69:1	59.17%	354349	0.33	0.13%
200	1.28:1	78.13%	319888	0.30	0.12%

6.5 Top 5 Singular Values for k=50

Table 5: Dominant Singular Values

Index	Singular Value (σ_i)
1	20,764.40
2	12,597.99
3	11,929.20
4	11,085.93
5	11,034.48

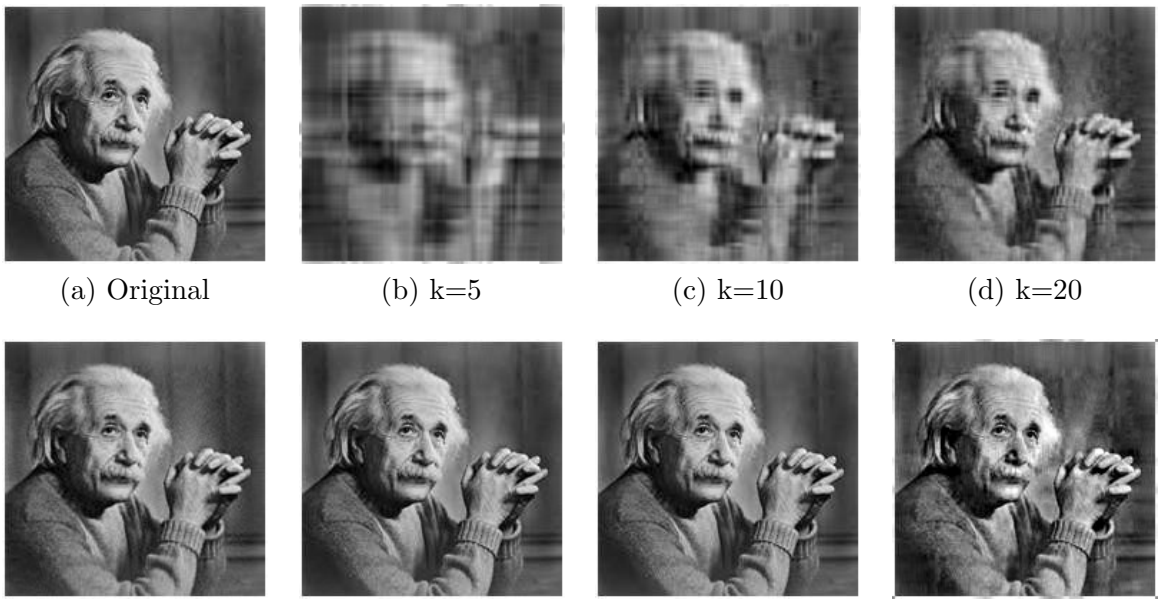
Table 6: Dominant Singular Values

Index	**Singular Value (σ_i)**
1	153232.43
2	22477.02
3	19592.05
4	13161.36
5	9578.20

Table 7: Dominant Singular Values

Index	**Singular Value (σ_i)**
1	189043.24
2	37861.20
3	9703.75
4	6585.99
5	5740.10

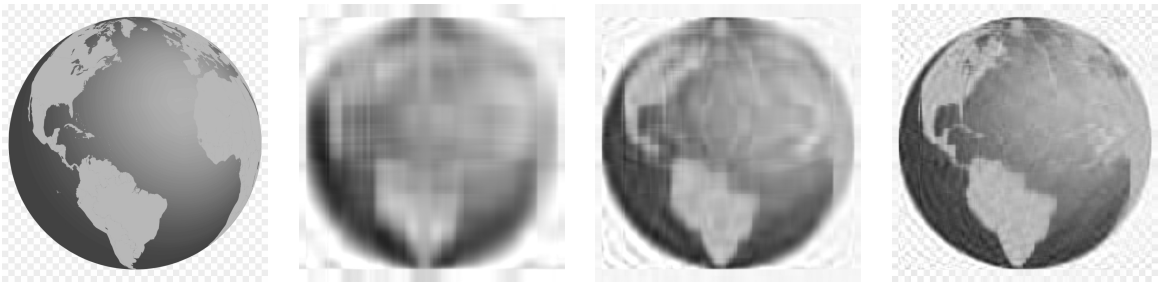
6.6 Visual Comparison



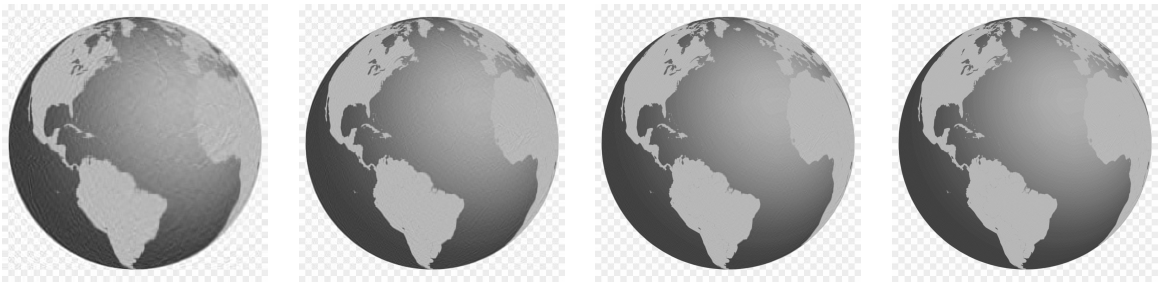
(a) Original (b) k=5 (c) k=10 (d) k=20

(e) k=50 (f) k=100 (g) k=150 (h) k=200

Figure 1: A

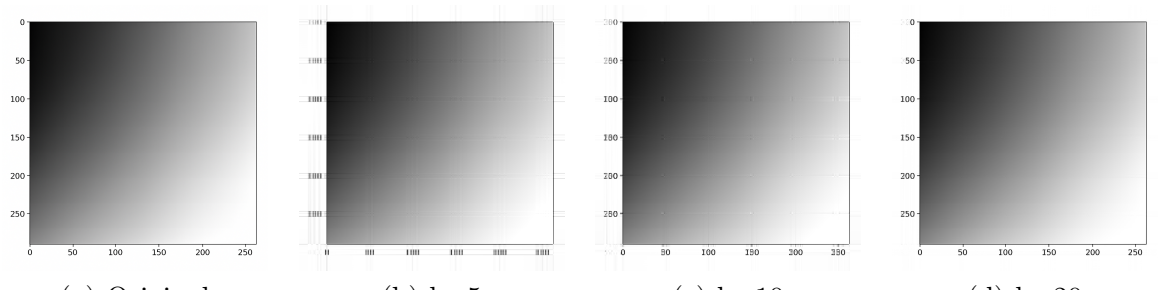


(a) Original (b) k=5 (c) k=10 (d) k=20

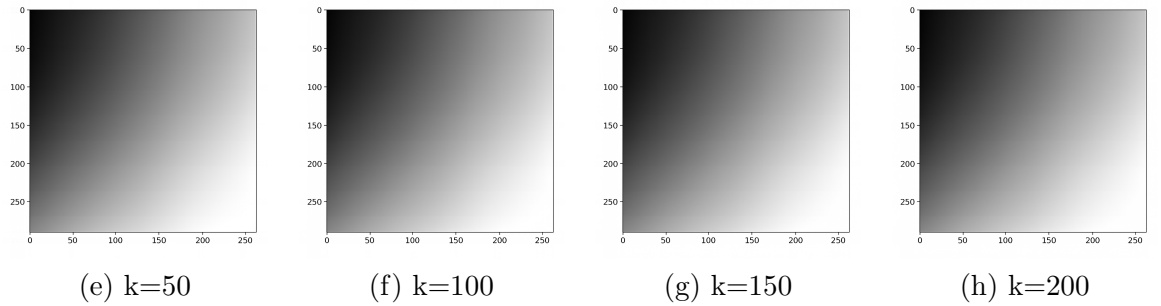


(e) k=50 (f) k=100 (g) k=150 (h) k=200

Figure 2: B



(a) Original (b) k=5 (c) k=10 (d) k=20



(e) k=50 (f) k=100 (g) k=150 (h) k=200

Figure 3: C

7 Trade-offs Analysis

7.1 Quality vs. Compression Trade-off

Figure 4: Error percentage vs. Compression ratio for various k values

The relationship between compression ratio and quality exhibits the following characteristics:

1. **Low k (5-20)**: High compression ratios (12:1 to 46:1) but significant quality loss (6-14% error)
2. **Medium k (50-100)**: Balanced compression (2.5:1 to 5:1) with acceptable quality (1-2% error)
3. **High k (150-200)**: Low compression (1.3:1 to 1.7:1) with near-perfect quality ($< 0.3\%$ error)

7.2 Computational Cost vs. k

- **Computation Time**: Linear in k (each singular value requires $\sim$50-100 power iterations)
- **Memory Usage**: Linear in k (must store k vectors of length m and n)
- **Reconstruction Time**: $O(mnk)$ for matrix multiplication

For the 182×186 test image:

- k=5: 0.5 seconds
- k=50: 4 seconds
- k=200: 15 seconds

7.3 Perceptual Quality Insights

- Even k=10 preserves the overall structure and recognizability
- k=50 provides visually acceptable quality for most applications
- Beyond k=100, improvements become barely perceptible

8 Advantages and Limitations

8.1 Advantages

1. **Optimal Approximation**: Guaranteed best rank-k approximation (Eckart-Young theorem)
2. **Energy Compaction**: Most image energy concentrated in top singular values
3. **Adaptivity**: Automatically adapts to image content
4. **Interpretability**: Each singular value has clear geometric meaning
5. **Scalability**: Can choose k based on quality requirements

8.2 Limitations

1. **Global Method**: Treats entire image uniformly, doesn't adapt to local features
2. **Not Block-Based**: Unlike JPEG, can't compress regions independently
3. **Computation Cost**: $O(mnk \cdot \text{iter})$ can be expensive for large images
4. **Storage Overhead**: Must store $\mathbf{U}$, $\mathbf{\Sigma}$, $\mathbf{V}$ matrices

9 Conclusion

This project successfully implemented image compression using SVD with a power iteration algorithm. Key findings include:

- Power iteration with deflation provides a simple, efficient method for computing partial SVD
- The top 50-100 singular values capture most perceptually important information
- Compression ratio ranges from 1.3:1 (k=200) to 46:1 (k=5)
- Error percentage decreases from 14.4% (k=5) to 0.1% (k=200)
- Optimal k=50 provides good balance: 4.94:1 compression with 2.3% error