Software Assignment: Image Compression Using Truncated SVD

AI25BTECH11023 - Pratik R

1 Summary

The singular value decomposition of a matrix A is the factorization of A into the product of three matrices $A = U\Sigma V^\top$ where the columns of U and V are orthonormal and the matrix Σ is diagonal with positive real entries. Let A be some $m \times n$ matrix. We are interested in finding the basis vectors $\mathbf{v_i}$ of the row space of A in R^m such that after transformation gives scaled basis vectors $\mathbf{u_i}$ of the column space of A in R^n, that is

$$A\mathbf{v_i} = \sigma_i \mathbf{u_i} \tag{1}$$

$\mathbf{v_i}$ are the columns of V and $\mathbf{u_i}$ are the columns of U. σ_i is the factor by which the transformed vector is scaled, σ_i being the diagonals of matrix Σ. Adding Eq 1 for all i's upto r, we get $AV = U\Sigma$.

multipying by $V^\top$ on right

$$AVV^\top = U\Sigma V^\top \tag{2}$$

Since V is orthonormal matrix

$$A = U\Sigma V^\top \tag{3}$$

In this video it is shown that by computing $A^\top A$ we can compute these singular values σ_i and $\mathbf{v_i}$ in the following manner.

$$A = U\Sigma V^\top \tag{4}$$

$$A^\top A = V\Sigma U^\top U\Sigma V^\top \tag{5}$$

$$A^\top A = V\Sigma\Sigma V^\top = V\Sigma^2 V^\top \tag{6}$$

This is a case of spectral decomposition in which Σ^2 contains eigenvalues of $A^\top A$ as diagonals. Matrix V can be found by computing eigenvectors of $A^\top A$.For U eliminate $V^\top V$ by taking $AA^\top$.

2 Abstract

The SVD is useful in many tasks. The data matrix A is close to a matrix of low rank(k) and it is useful to find a low rank matrix which is a good approximation to the data matrix .From the singular value decomposition of A, we can get the matrix A_k of rank k which best approximates A. This is what we will be using as the basis for image compression.

3 Algorithm Selection

There are several algorithms that can be used to calculate the eigenvalues of a matrix. Some of these algorithms are listed below.

Algorithm	**Description**
Power Iteration	Iteratively finds the largest eigenvalue by repeatedly multiplying the matrix with a vector.
Inverse Iteration	Finds the smallest eigenvalue by inverting the matrix and applying power iteration.
QR Algorithm	Uses QR decomposition iteratively to compute all eigenvalues of a matrix.
Lanczos Algorithm	Reduces a large sparse symmetric matrix to tridiagonal form for eigenvalue computation.
Arnoldi Iteration	Generalizes the Lanczos method for non-symmetric matrices to compute a few eigenvalues.
Jacobi Method	Rotates pairs of rows and columns to diagonalize the matrix iteratively.
Davidson Algorithm	Specialized for large sparse matrices, builds a subspace to approximate eigenvalues.

Table 1: Eigenvalue Calculation Algorithms and Their Descriptions

Nevertheless, for this project I have use Power Iteration method due to relatively code for algorithm can be found in *codes/c_main/main.c*

4 Algorithm

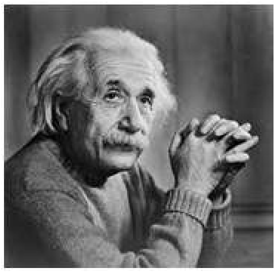
1. Read image using *stbi_load* and storing it in 1D Array($m \times n$)
2. multiply to get $A^\top A$

3. Use Power iteration method to get most dominant eigenvector of $A^\top A$. Make sure to normalise it during each iteration.
4. Corresponding eigenvalue(λ) can be found by norm of $A^\top A\mathbf{v}$ where $\mathbf{v}$ is the eigenvector
5. To find the next eigenvector we remove the contribution of $\mathbf{v}$ from $A^\top A$, that is
6. $A^\top A \leftarrow A^\top A - \lambda v v^\top$ and repeat from step 3 to find the next dominant eigen vector
7. These eigenvectors will form basis of row space for the matrix A and eigenvalues of $A^\top A$ will serve as squares of singular values. Basis of column space can be computed by $u = \frac{Av}{||Av||} = \frac{Av}{\sigma}$
8. Take only best k eigenvalues to reconstruct the matrix A_k via Truncated SVD.
9. Reconstruct the matrix A_k by the given equation $A_k = U_k \Sigma_k V_k^\top$
10. In the code, A_k is reconstructed by adding $\sigma_i \mathbf{u_i} \mathbf{v_i}^\top$ over i = 1 to k, that is
11. $A = \sum_{i=1}^{k} \sigma_i \mathbf{u_i} \mathbf{v_i}^\top$
12. use *stbi_write_jpg* to write 1D Array back to jpg.

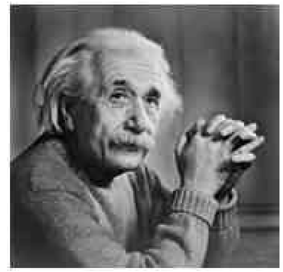
Note: iteration is set to over 200 and the reconstructed image for different values of k is shown in the next section.

5 Image output

For Einstein.jpg For globe.jpg

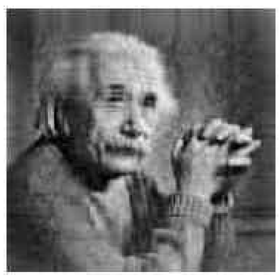


(a) Original

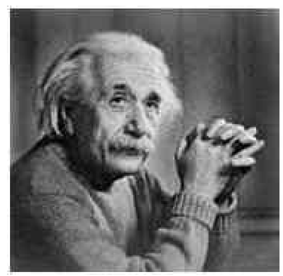


(b) k = 100

(a) k = 10



(b) k = 20

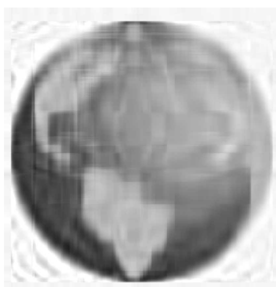


(c) k = 50

Figure 2: Effect of rank k on compression quality.

(a) Original

(b) k = 100



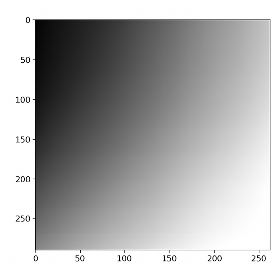
(a) k = 10

(b) k = 20

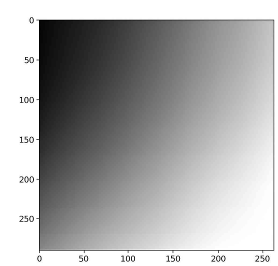
(c) k = 50

Figure 4: Effect of rank k on compression quality.

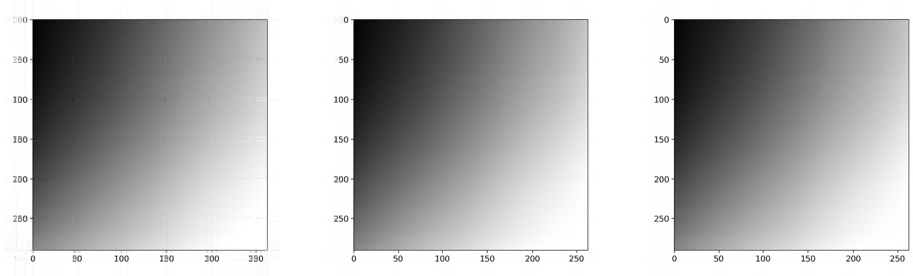
For greyscale.png



(a) Original



(b) k = 100



(a) k = 10 (b) k = 20 (c) k = 50

Figure 6: Effect of rank k on compression quality.

6 Error Analysis

K	Average Forbeanius Error
10	3249.14
20	1568.472
50	1048.647
100	893.768
200	567.423

Table 2: Error

7 Conclusion

Image compression through power iteration is efficient and easy to implement and hence a good choice for image compression.