Ukrainian Catholic University

Faculty of Applied Sciences

Computer Science Bachelor Program

Singular Value Decomposition and Image Compression

Linear Algebra final project report

Authors:

Olena Matrunych

Danylo Nazaruk

(contributed equally, order chosen alphabetically)

May 20, 2020

Abstract

The Singular Value Decomposition (SVD) is a significant topic in Linear Algebra. In this report we introduce the topic of SVD of a matrix and image compression. Then we formulate the main problem and go over the general algorithm of image compression implementation as well as used linear algebra methods. Next, we provide some examples of our algorithm at work and compare different approaches. In the end we compare our project with similar work and make conclusions.
The code is available on our GitHub.

Introduction

Image Compression

Compression is a term used to quantify the reduction in data-representation size produced by a data compression algorithm.

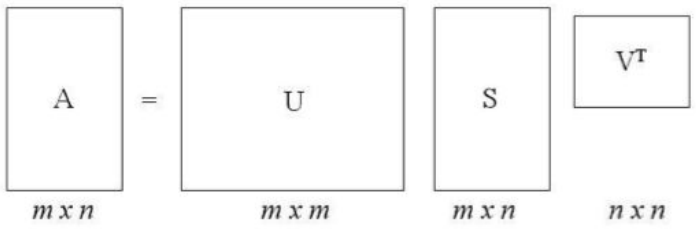
There are two forms of compression:

1. Those that are non-destructive and reversible. They do not change the content of files. Files can be compressed and decompressed and the data in the file remains the same, it has not changed. For example Flate/deflate compression, Huffman compression, RLE compression.
2. Those that are irreversible, get rid of some information in order to attain size reduction. For example Singular Value Decomposition.

Singular Value Decomposition

SVD has many practical and theoretical values; a special feature of SVD is that it can be performed on any real (m, n) matrix.

The **singular value decomposition** of a matrix A is the factorization of A into the product of three matrices $A = USV^T$ where the columns of U and V are orthonormal and the matrix S is a diagonal matrix with singular values (SV) on the diagonal.



The columns of U in the singular value decomposition, called the right singular vectors of A, always form an orthogonal set with no assumptions on A. The columns of V are called the left singular vectors and they also form an orthogonal set. A simple consequence of the orthogonality is that for a square and invertible matrix A, the inverse of *A* is $VD^{-1}U^T$.

Properties of the SVD

1. The singular values σ1, σ2, … ,σn are unique, however, the matrices U and V are not unique.
2. Since A^TA = $VS^T\ SV^T$, so V diagonalizes A^TA , it follows that the v_j are the eigenvectors of A^TA .
3. Since AA^T = USS^TU^T , so it follows that U diagonalizes AA^T and that the u_j 's are the eigenvectors of AA^T .
4. The rank of matrix A is equal to the number of its nonzero singular values.

Image compression deals with the problem of reducing the amount of data required to represent a digital image. In many applications, the singular values of a matrix decrease quickly with increasing rank. The 4th property allows us to compress the matrix data by eliminating the small singular values or the higher ranks.

When an image is SVD transformed, it is not compressed, but the data takes a form in which the first singular value has a great amount of the image information. With this, we can use only a few singular values to represent the image with little differences from the original.

Example

$A = USV^T$ = $\sigma_1\ u_1\ v_1^T$ + $\sigma_2\ u_2\ v_2^T$ + … + $\sigma_r\ u_r\ v_r^T$

While compressing the image, the sum is not performed to the very last SVs, the SVs with small enough values are dropped.

The closest matrix of rank k is obtained by truncating those sums after the first k terms:

$$A_k = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \ldots + \sigma_k u_k v_k^T$$

The total storage space for A_k will be k (m + n + 1).

Problem

The main problem we are solving is finding the best way of compressing colored images using singular value decomposition. We compare different approaches of image compression and determine which one is better by comparing the dependence of rank, compression ratio and mean square error for each approach.

Approaches to Solutions

In order to work with colored images we have to separate them into three matrices containing information about red, green and blue intensity.

Example of splitting an image into three new images, a red-scale, green-scale, and blue-scale image:

We discovered two possible solutions to solve given problem:

- Compress three different matrices separately
- Stack matrices horizontally and compress them together

Brief Explanation of the Algorithm

For coloured images we have a 3-dimensional array of size *n*×*m*×3, where *n* and *m* represent the number of pixels vertically and horizontally respectively, and for each pixel we store the intensity for colours red, green and blue.

To compress the image we will find a smaller rank k approximation using the SVD-decomposition.

- Firstly, we normalize the intensity values in each pixel, by dividing the values of the image matrix by 255.
- Then we perform an SVD-decomposition for a matrix A formed by stacking the matrices of red, green and blue intensity. (in second approach matrices are not stacked and decomposed separately)
- Take k columns from matrix U, k rows from matrix V^T, and k values from diagonal S.
- Multiply received matrices to reproduce k-rank approximation of given image.

Used Linear Algebra Methods

The main linear algebra method used in our algorithm is SVD decomposition. We used the following algorithm to decompose given matrix into USV^T:

- Find the AA^T matrix.
- Determine its eigenvalues and eigenvectors.
- Align eigenvectors as columns of matrix and sort eigenvalues and respective eigenvectors in descending order to get the U matrix.
- Find the square roots of eigenvalues to get the singular values which compose the diagonal of the S matrix.
- To find the ith column of V we multiply matrix transpose by ith column of U (meaning ith eigenvector) and divide received matrix by ith value of the diagonal of S (meaning ith singular value). The columns of V are the left eigenvectors of the matrix.
- Transpose received matrix to get V^T.

```
function svd(matrix):
  AAT = matrix * transpose(matrix)
  AAT_eigenvectors = sort_descending(eigenvectors(AAT))
  AAT_eigenvalues = sort_descending(eigenvalues(AAT))

  U = AAT_eigenvectors
  S = diagonal_matrix(sqrt(AAT_eigenvalues)

  for each iteration i out of length(U) do
    V[i] = transpose(matrix) * AAT_eigenvectors[i] / AAT_eigenvalues[i]
  end

  VT = transpose(V)

  return U, S, VT
```

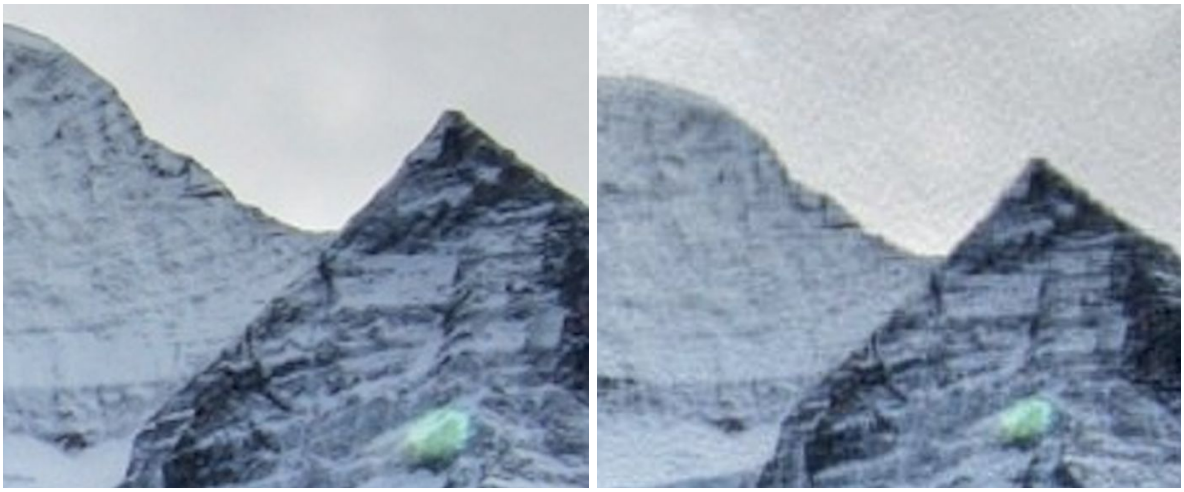
Example of Singular Value Decomposition Function:

```
Original:
[3, 2, 2, 4]
[2, 3, -2, 1]
[1, 2, 2, 1]
U:
 [[-0.81878969  0.3379932  -0.46405176]
 [-0.41638646 -0.9061123   0.07471827]
 [-0.39522874  0.25440342  0.8826512 ]]
S:
 [[6.80756553 0.         0.        ]
 [0.         3.48137979 0.        ]
 [0.         0.         1.59281082]]
VT:
 [[-0.54121708 -0.54016318 -0.23433692 -0.60032826]
 [-0.15621437 -0.44049881  0.86087069  0.20114552]
 [-0.22605795  0.66634008  0.43179161 -0.56430905]]
USVT:
 [[ 3.  2.  2.  4.]
 [ 2.  3. -2.  1.]
 [ 1.  2.  2.  1.]]
```

Comparing Original and Compressed Image

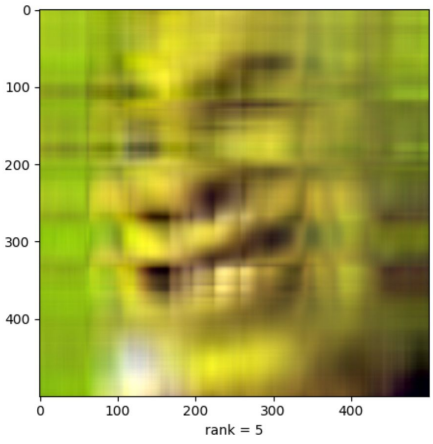
The left image is the original and the right one is compressed (using separated matrices method) with rank 300. When they are reduced they appear almost identical while the right one takes up almost 4 times less space. (77 785 920 and 21 607 200 bytes respectively)

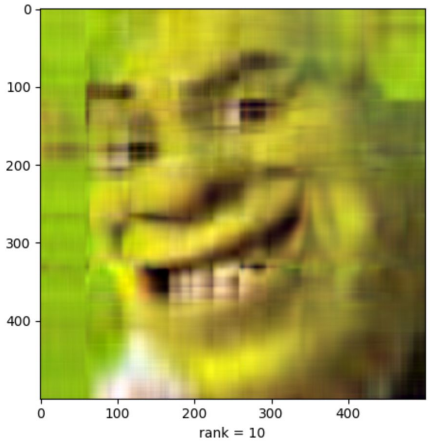
The distortion can only be spotted if we zoom in.

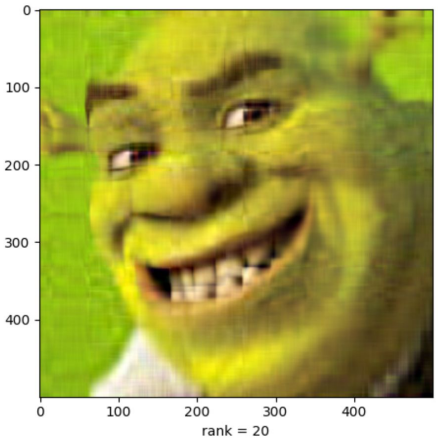


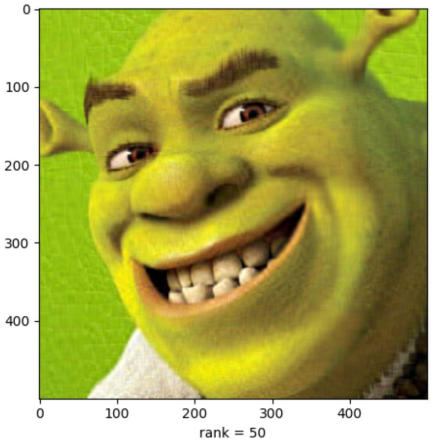
Example of Image Compression (matrix stacking method)

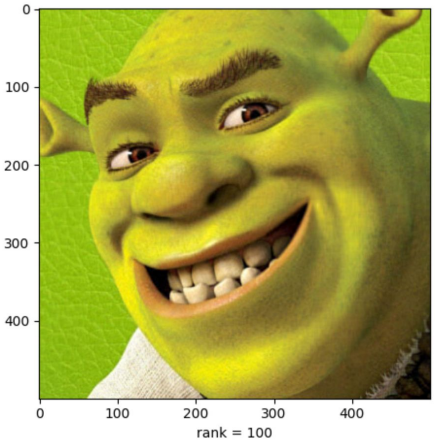
Image size: 500 x 500

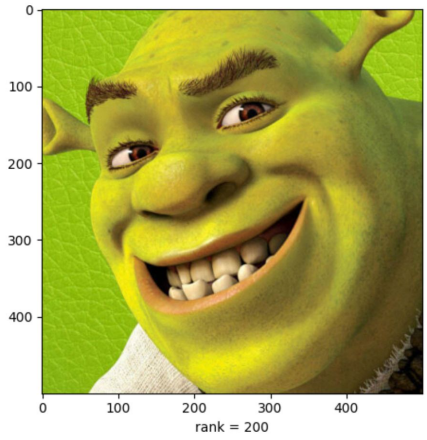






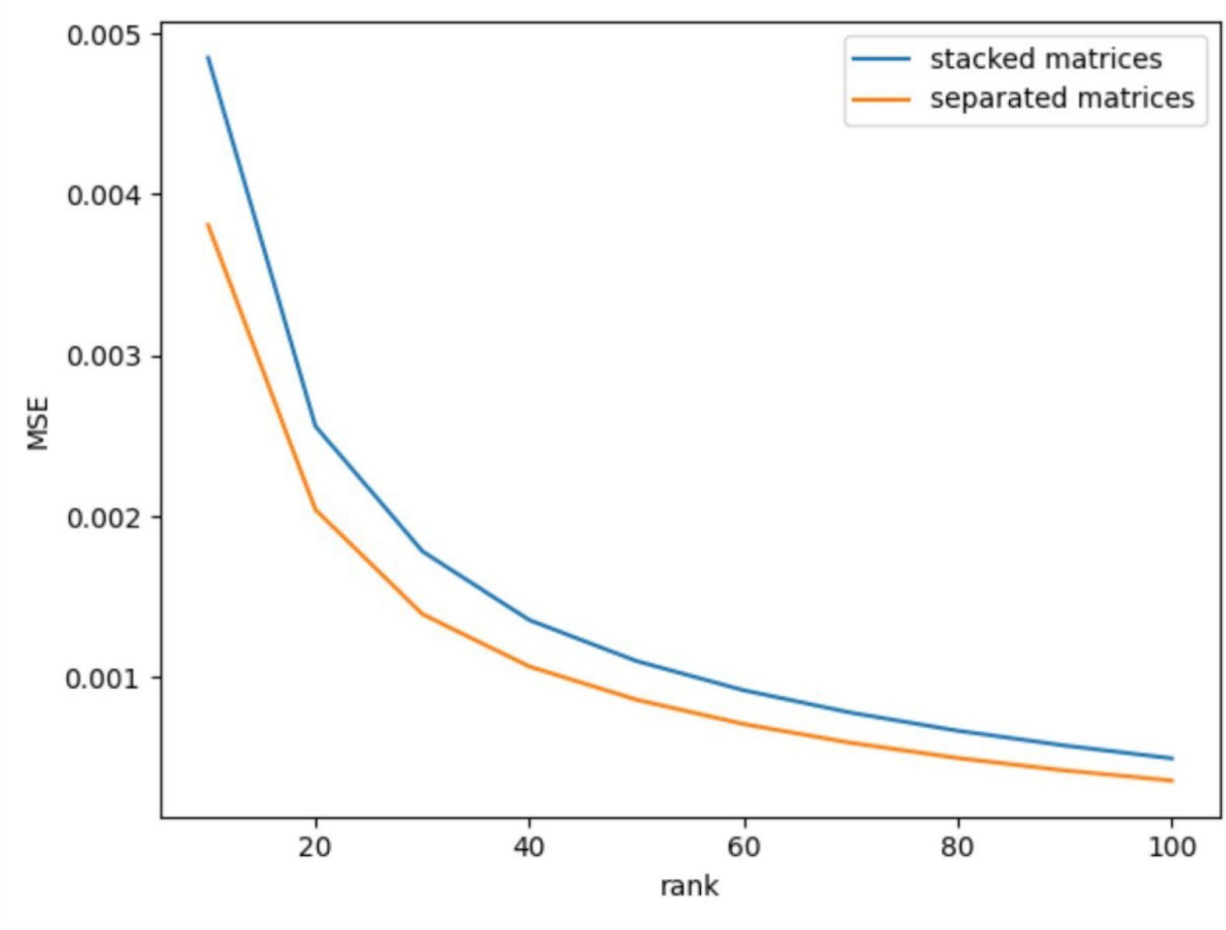




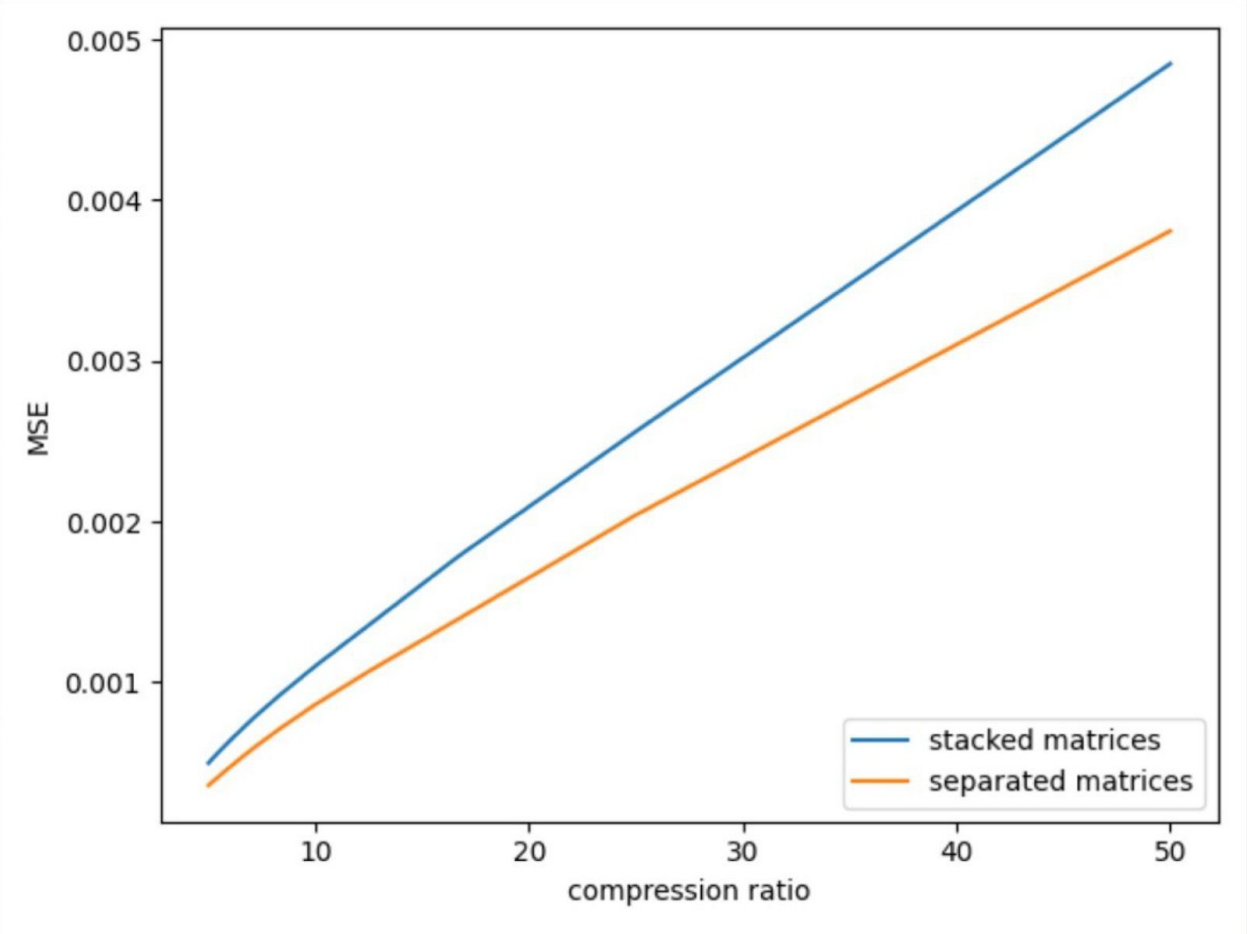


Analysis and Comparison

Our main problem is to compare two different approaches of image compression. Compression ratio for each approach turned out to be the same. Then we compared the dependence of rank and mean square error (between uncompressed and compressed images). The graphs for each approach are shown below. The MSE for stacked matrices approach is always higher.



Then we compared the dependence of compression ratio and MSE. Images compressed using separated matrices method have the same compression ratio while having lower MSE.



Despite appearing worse in the graphs, the stacked matrices method has its own advantages:

- decomposed image takes up less space (meaning that compressed matrix will take up less space as well)
- compressing using this method is much faster (only one matrix is decomposed instead of 3).

Separated matrices

Pros:

- Low MSE

Cons:

- Takes up more space
- Slow for big images

Stacked matrices

Pros:

- Takes up less space
- Faster

Cons:

- Higher MSE

Review of Related Work

We found a similar work "Image Compression using Singular Value Decomposition" by Brady Mathews from the University of Utah.

In this work it is explained in great detail what SVD is, how it works and how it is used for image compression. There are few examples of image compression provided as well as MatLab code for grayscale images compression.

The work focuses on the general topic of SVD compression and uses mostly grayscale images.

It includes one approach of compressing colored images by separating them into red, green and blue intensity matrices and compressing them separately, while we discovered a second approach of stacking matrices horizontally which has its own advantages.

They implement the algorithm using the SVD function provided by MatLab. In our project, we created our own SVD function.

Conclusion

Image is just numbers stored in matrices, and matrices can be easily manipulated. Using SVD we can reduce space taken up by matrix and store images more efficiently. There are different approaches to compress images using SVD. Each of them has its own advantages and disadvantages and each could appear better under specific circumstances.

Literature

1. https://www.cs.cmu.edu/~venkatg/teaching/CStheory-infoage/book-chapter-4.pdf
2. https://math.mit.edu/~gs/linearalgebra/linearalgebra5_7-1.pdf
3. https://www.math.cuhk.edu.hk/~lmlui/CaoSVDintro.pdf
4. http://db.cs.duke.edu/courses/cps111/spring07/notes/12.pdf