MTH-573 Numerical Linear Algebra Project Report

"Matrix Factorization for Image Optimization"

Authors

Nikhil Premachandra rao (02105149)

Matrix Factorization for Image Optimization

OBJECTIVE:

The objective of this project is to explore and implement advanced image compression techniques using Singular Value Decomposition (SVD) and QR Factorization. The objective is to obtain high reconstruction accuracy while compressing efficiently. The project's goal is to give insights into the trade-offs and benefits of alternative compression methodologies by assessing their performance, therefore adding to the knowledge of image processing and optimization.

MOTIVATION:

Image compression is important in a variety of applications, from digital storage and transmission to computer vision and machine learning. The necessity for effective compression methods that strike a compromise between lowering data size and keeping important information for correct reconstruction drives this study. We want to unveil the theoretical underpinnings and practical ramifications of Singular Value Decomposition (SVD) and QR Factorization in the context of picture optimization by diving into them. The findings of the study have the potential to inform advances in image processing techniques, therefore contributing to the larger disciplines of data compression and computer vision.

BACKGROUND:

Picture compression is a critical field of digital signal processing that aims to minimize picture storage and transmission needs while maintaining visual quality. Singular Value Decomposition (SVD) and QR Factorization, for example, provide a new perspective by employing linear algebra techniques to accomplish more subtle compression. These approaches lay the groundwork for understanding the intrinsic structure of pictures and enable efficient representation and restoration.

Linear algebra concepts are used in this project's image compression approach Singular Value Decomposition (SVD) is divided into three basic pieces: U, S, and Vt. These are the image's building blocks: the left singular vectors, singular values, and right singular vectors. The brilliance of SVD comes in its capacity to either maintain or reject these parts, allowing us to compress the image without losing crucial elements. In contrast, QR Factorization builds an orthogonal component (Q) and an upper triangular (R) by dissecting the picture differently. Linear algebra is a crucial tool for improving picture data as it helps to understand the arithmetic behind image compression.

Python is used as the primary platform for implementing the SVD and QR compression techniques. NumPy which is a powerful numerical computing library for matrix operations and factorization. Matplotlib is used for visualizations, interpretation of graphs, singular value plots, and reconstructed images. The accuracy of the compression and decompression processes is calculated using the Mean Squared Error (MSE) metric and accuracy percentage.

MY WORK

For picture compression, two numerical linear algebra methods are used in this project: Singular Value Decomposition (SVD) and QR Factorization. The original picture matrix is decomposed into three matrices (U, S, and Vt) via SVD, collecting the key information necessary for compression. QR Factorization, on the other hand, entails dividing the matrix into two orthogonal matrices (Q) and an upper triangular matrix (R).

The project entails utilizing the NumPy module to build both SVD and QR compression algorithms in Python. Visualizations such as cumulative energy graphs and single-value plots are developed to better comprehend the compression process. To quantify the accuracy of compression and decompression, the Mean Squared Error (MSE) and accuracy % metrics are used.

The results show how well SVD and QR Factorization compress pictures. Singular value plots and cumulative energy graphs are two examples of visualizations that show how these techniques maintain important data. MSE and accuracy percentage (%) metrics provide a quantitative assessment of the compressed pictures to the originals. This emphasizes how important linear algebra is to the creation of compression techniques that maximize file size without compromising reconstruction accuracy.

APPENDIX A: Code Implementation

The full code is available on GitHub: https://github.com/nikhil9066/Image_Compression

1. SVD Compression Code:

```
def svd_compress(image, k):
    U, S, Vt = np.linalg.svd(image, full_matrices=False)
    singular_values_squared = S**2
    cumulative_energy = np.cumsum(singular_values_squared) /
np.sum(singular_values_squared)

    compressed_U = U[:, :k]
    compressed_S = np.diag(S[:k])
    compressed_Vt = Vt[:k, :]
    compressed_image = np.dot(compressed_U, np.dot(compressed_S, compressed_V

    # Plotting the singular values for compression
    plt.plot(S, marker='o')
    plt.title('Singular Values (Compression)')
    plt.xlabel('Index')
    plt.ylabel('Singular Value')
    plt.show()

    accuracy_before = calculate_mse(image, compressed_image)
    max_possible_mse = np.max(image)**2  # Maximum possible MSE when all pixe
values are different
    accuracy_percentage_before = calculate_accuracy_percentage(accuracy_befor
max_possible_mse)
    print(f'MSE Before Compression (SVD): {accuracy_before}')
    print(f'Accuracy Before Compression (SVD): {accuracy_percentage_before}%\

    print('U matrix (Left Singular Vectors):')
    print(compressed_U)
    print('\nS matrix (Diagonal Singular Values):')
    print(compressed_S)
    print('\nVt matrix (Right Singular Vectors):')
    print(compressed_Vt)
    return compressed_image, compressed_U, compressed_S, compressed_Vt
```

2. SVD Decompression Code:

```
# SVD Decompression function
def svd_decompress(compressed_U, compressed_S, compressed_Vt):
    reconstructed_image = np.dot(compressed_U, np.dot(compressed_S,
compressed_Vt))

    # Plotting the singular values for decompression
    singular_values = np.diag(compressed_S)
    plt.plot(singular_values, marker='o')
    plt.title('Singular Values (Decompression)')
    plt.xlabel('Index')
    plt.ylabel('Singular Value')
    plt.show()

    accuracy_after = calculate_mse(original_image, reconstructed_image)
    max_possible_mse = np.max(original_image)**2  # Maximum possible MSE for
original image
    accuracy_percentage_after = calculate_accuracy_percentage(accuracy_after,
max_possible_mse)

    print(f'MSE After Decompression (SVD): {accuracy_after}')
    print(f'Accuracy After Decompression (SVD): {accuracy_percentage_after}%\

    print('U matrix (Left Singular Vectors):')
    print(compressed_U)
    print('\nS matrix (Diagonal Singular Values):')
    print(compressed_S)
    print('\nVt matrix (Right Singular Vectors):')
    print(compressed_Vt)

    return reconstructed_image, accuracy_after, accuracy_percentage_after
```

3. QR Compression Code:

```
# QR Compression function
def qr_compress(image, k):
    Q, R = np.linalg.qr(image)
    compressed_Q = Q[:, :k]
    compressed_R = R[:k, :]
    compressed_image = np.dot(compressed_Q, compressed_R)

    accuracy_before = calculate_mse(normalize_image(image),
normalize_image(compressed_image))
    max_possible_mse = 1.0
    accuracy_percentage_before = calculate_accuracy_percentage(accuracy_befor
max_possible_mse)

    print(f'MSE Before Compression (QR): {accuracy_before}')
    print(f'Accuracy Before Compression (QR): {accuracy_percentage_before}%\r

    print('Q matrix (Orthogonal Matrix):')
    print(compressed_Q)
    print('\nR matrix (Upper Triangular Matrix):')
    print(compressed_R)
```

4. QR Decompression Code:

```
# QR Decompression function
def qr_decompress(compressed_Q, compressed_R):
    reconstructed_image = np.dot(compressed_Q, compressed_R)

    accuracy_after = calculate_mse(normalize_image(original_image),
normalize_image(reconstructed_image))
    max_possible_mse = 1.0  # Maximum possible MSE for the original image
(normalized)
    accuracy_percentage_after = calculate_accuracy_percentage(accuracy_after,
max_possible_mse)

    print(f'MSE After Decompression (QR): {accuracy_after}')
    print(f'Accuracy After Decompression (QR): {accuracy_percentage_after}%\r

    print('Q matrix (Orthogonal Matrix):',compressed_Q)
    print('\nR matrix (Upper Triangular Matrix):',compressed_R)
Return reconstructed_image,accuracy_after,accuracy_percentage_after
```

APPENDIX B: Output:

1. Compression and Decompression using QR Factorization

MSE Before Compression (QR): 0.044908335181034625

```
Q matrix (Orthogonal Matrix):
[[-0.03688021 -0.00134302  0.00230474 ...  0.04201829 -0.00318034
  -0.05311094]
 [-0.03556306 -0.00221373  0.00045215 ...  0.04292656  0.00653032
  -0.04443292]
 [-0.03278241 -0.01253092 -0.00133766 ...  0.02742652  0.01814457
  -0.00373247]
 ...
 [-0.02868461  0.01658037  0.0090478  ... -0.02768661  0.0206912
  -0.01716509]
 [-0.02956271  0.01573179  0.01060004 ... -0.03683951  0.02079254
   0.00843442]
 [-0.02927001  0.01315655  0.01263133 ... -0.04642338  0.01442737
   0.02337287]]

R matrix (Upper Triangular Matrix):
[[-6832.93304519 -6844.69183156 -6730.68061634 ... -4936.72274804
  -4940.71517703 -4945.49935972]
 [    0.         -1166.27643848 -1475.44896536 ...  -436.50297221
   -437.76001027  -437.70662998]
 [    0.             0.         -1392.98413179 ...  -692.64538682
   -694.38448778  -697.0233938 ]
 ...
 [    0.             0.             0.         ...   -84.14244747
    -82.53377225   -82.48329629]
 [    0.             0.             0.         ...    28.06297075
     29.83425696    31.57047538]
 [    0.             0.             0.         ...   -31.19896199
    -29.49307476   -29.00807958]]
```

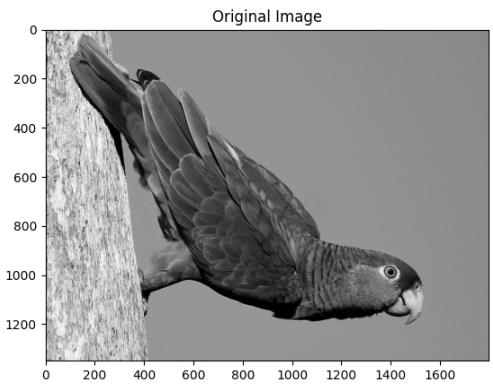
2. Compression and Decompression using QR Factorization

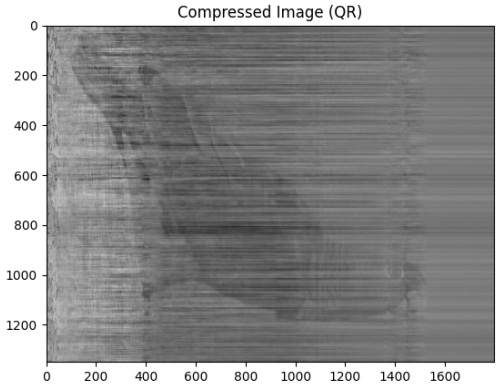
MSE After Decompression (QR): 0.044908335181034625

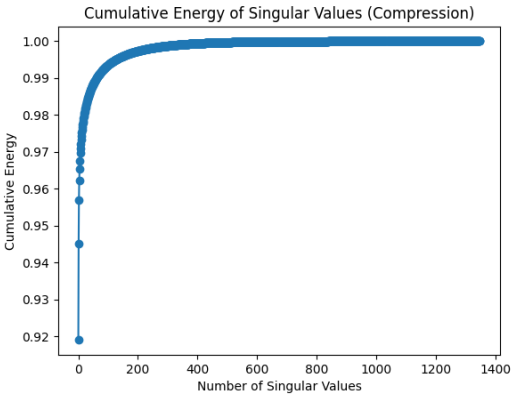
```
Q matrix (Orthogonal Matrix):
[[-0.03688021 -0.00134302  0.00230474 ...  0.04201829 -0.00318034
  -0.05311094]
 [-0.03556306 -0.00221373  0.00045215 ...  0.04292656  0.00653032
  -0.04443292]
 [-0.03278241 -0.01253092 -0.00133766 ...  0.02742652  0.01814457
  -0.00373247]
 ...
 [-0.02868461  0.01658037  0.0090478  ... -0.02768661  0.0206912
  -0.01716509]
 [-0.02956271  0.01573179  0.01060004 ... -0.03683951  0.02079254
   0.00843442]
 [-0.02927001  0.01315655  0.01263133 ... -0.04642338  0.01442737
   0.02337287]]

R matrix (Upper Triangular Matrix):
[[-6832.93304519 -6844.69183156 -6730.68061634 ... -4936.72274804
  -4940.71517703 -4945.49935972]
 [    0.         -1166.27643848 -1475.44896536 ...  -436.50297221
   -437.76001027  -437.70662998]
 [    0.             0.         -1392.98413179 ...  -692.64538682
   -694.38448778  -697.0233938 ]
 ...
 [    0.             0.             0.         ...   -84.14244747
    -82.53377225   -82.48329629]
 [    0.             0.             0.         ...    28.06297075
     29.83425696    31.57047538]
 [    0.             0.             0.         ...   -31.19896199
    -29.49307476   -29.00807958]]

[Finished in 4.4s]
```

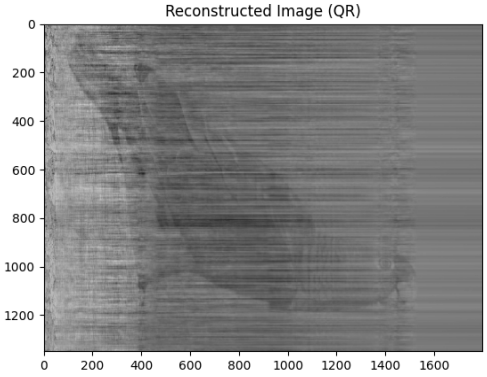
3. **Compression and Decompression using SVD**

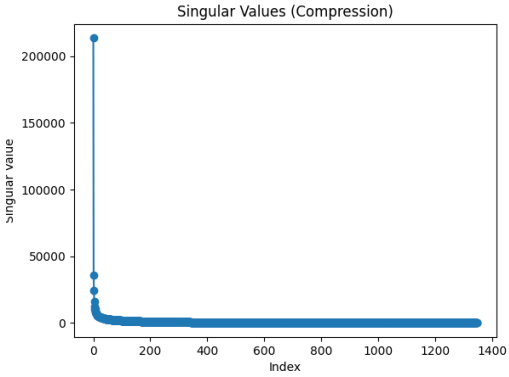
```
U matrix (Left Singular Vectors):
[[ 0.03021285  0.01151863  0.01357403 ...  0.00554388  0.00632614
  -0.03687928]
 [ 0.03017679  0.01153934  0.01372085 ...  0.00721355  0.00704979
  -0.03673613]
 [ 0.03018419  0.0115484   0.01375704 ...  0.01510859  0.00170288
  -0.03340773]
 ...
 [ 0.0314831   0.00536634  0.01823658 ... -0.00299569 -0.00829893
  -0.00620265]
 [ 0.03145539  0.00617913  0.01814806 ... -0.0044569  -0.01169668
   0.00577176]
 [ 0.03141777  0.00667531  0.01796835 ... -0.00039595 -0.00584335
   0.01550378]]
S matrix (Diagonal Singular Values):
[[213635.78028477      0.              0.          ...      0.
       0.              0.        ]
 [     0.          36043.28228688      0.          ...      0.
       0.              0.        ]
 [     0.              0.          24235.90836385 ...      0.
       0.              0.        ]
 ...
 [     0.              0.              0.          ...   2932.71004974
       0.              0.        ]
 [     0.              0.              0.          ...      0.
    2892.07528732      0.        ]
 [     0.              0.              0.          ...      0.
       0.           2839.93370698]]
Vt matrix (Right Singular Vectors):
[[ 0.03066354  0.03116244  0.0316553  ...  0.02392021  0.02393803
   0.0239593 ]
 [-0.01002519 -0.01225055 -0.01533887 ... -0.00653109 -0.00656321
  -0.00657109]
 [ 0.00996018  0.00727247  0.00886023 ... -0.00108825 -0.00103624
  -0.00096991]
 ...
 [-0.02963934 -0.02447281  0.0439194  ... -0.00371744 -0.00352632
  -0.00379538]
 [ 0.04517417  0.04190096  0.02953706 ...  0.00192218  0.00227302
   0.00163   ]
 [-0.01131636 -0.03303761 -0.1059563  ... -0.00057798 -0.00083132
  -0.00046749]]
```

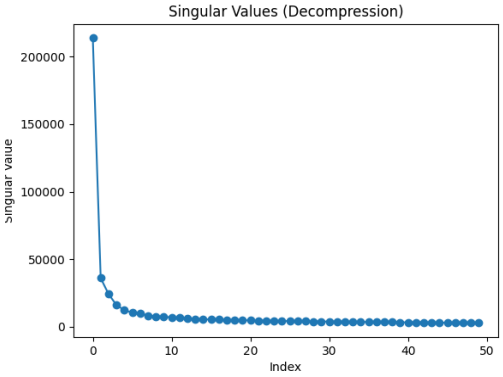

Original Image

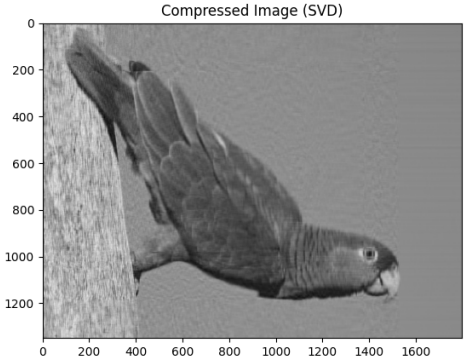

Compressed Image (QR)

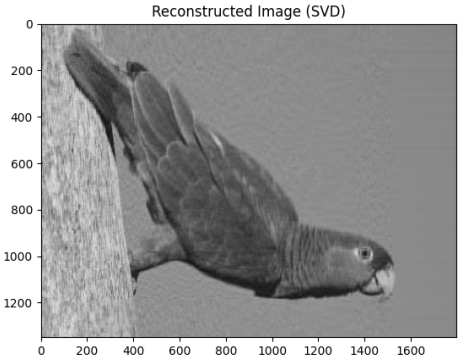

Cumulative Energy of Singular Values (Compression)
Cumulative Energy
Number of Singular Values


Reconstructed Image (QR)


Singular Values (Compression)
Singular value
Index


Singular Values (Decompression)
Singular value
Index


Compressed Image (SVD)


Reconstructed Image (SVD)

In conclusion, the project successfully demonstrates the application of numerical linear algebra methods, specifically SVD and QR Factorization, for image compression. The findings emphasize the significance of these techniques in achieving a balance between compression efficiency and reconstruction accuracy. The implementation and results contribute to a deeper understanding of the mathematical principles behind image compression.

REFERENCE

Class Notes:
- Dong, B., Ph.D. (2023, December 5). QR Algorithm - Section 4.4 Class Notes.
- Dong, B., Ph.D. (2023, December 7). Section 4.5 Class Notes.

GitHub Repository:
zaibatsu. (2021, March 22). Compressor. GitHub.

YouTube Channel:
Eigensteve. (2020, February 2). Title of the Video. [Video]. YouTube.