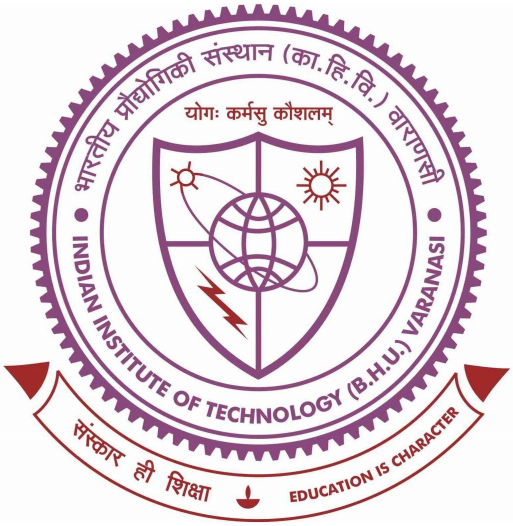
Hyperspectral Image Compression using Non - negative Tucker Decomposition Algorithm

Report submitted for fulfilling the requirements of
Algorithms Project
by

Dhriti Singh
18123006
Palakur Eshwitha Sai
18123014
Vishakha Tomar
18123020

Under the guidance of
Mr. Yaman Dua
Research Scholar



Department of Mathematical Sciences

INDIAN INSTITUTE OF TECHNOLOGY (BHU) VARANASI

Varanasi 221005, India

INDEX

TOOLS AND LANGUAGE USED

Language Used -To code our problem statement we prefered **Python** as our programming language. The diverse application of the Python language is a result of the combination of features which gives this language an edge over others.

Some of the benefits of programming in Python include:

- Presence of Third Party Modules
- Extensive Support Libraries
- Learning Ease and Support Available
- User-friendly Data Structures
- Productivity and Speed

Tools Used -

1. Decomposing 3D tensor -
    - numpy.
    - tensorly.
        - tensorly.decomposition - Contains the function used to decompose the 3D tensor - non_negative_tucker.
        - tensorly.fold , tensorly.unfold - For folding and unfolding the tensor contents respectively.
2. Image compression using NTD (png image) -
    - numpy.
    - matplotlib.
        - pyplot
        - pylab
    - tensorly.
        - tucker_to_tensor - To get back the original, full tensor from the tucker tensor.
        - tensorly.decomposition - Contains the function used to decompose the 3D tensor - non_negative_tucker.
    - time
    - PIL - Python Imaging Library
        - Image
    - Skimage.measure
        - compare_psnr
3. Image compression using NTD (Hyperspectral Image)
    - Scipy.io
        - Loadmat
    - matplotlib
        - matplotlib.pylab - plt - to plot the graph of required parameters.
        - pyplot
    - tensorly.
        - tucker_to_tensor - To get back the original, full tensor from the tucker tensor.
        - tensorly.decomposition - Contains the function used to decompose the 3D tensor - non_negative_tucker.
    - time
    - PIL - Python Imaging Library
        - Image
    - Skimage.measure
        - Compare_psnr

PROBLEM DEFINITION

Problem Statement -

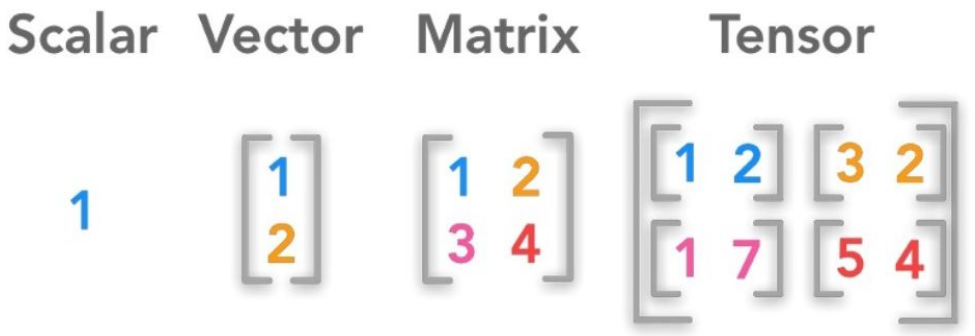
To compress a hyperspectral image using Non - negative Tucker Decomposition algorithm and plot the following graphs :

For different values of decomposed tensors -

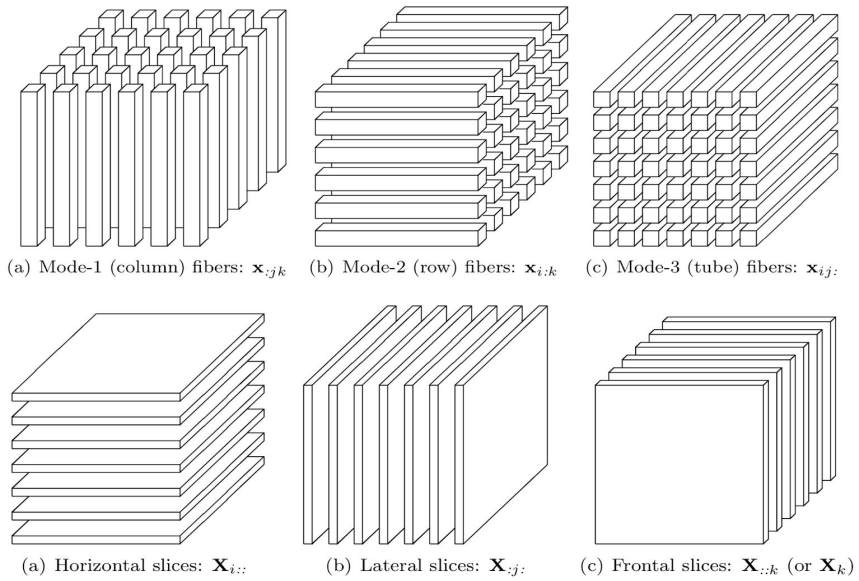
1. Compression Ratios vs Obtained Mean Square Error (MSE)
2. Compression Ratios vs Peak Signal-to-Noise Ratio (PSNR)

Concepts used -

1. Tensors - A Tensor is a container which can house data in N dimensions. Often and erroneously used interchangeably with the matrix (which is specifically a 2-dimensional tensor), tensors are generalizations of matrices to N-dimensional space.
   Tensors include descriptions of the valid linear transformations between tensors. examples of such transformations, or relations, include the cross product and the dot product.
   If we temporarily consider them simply to be *data structures*, below is an overview of where tensors fit in with scalars, vectors, and matrices, and some simple code demonstrating how Numpy can be used to create each of these data types.

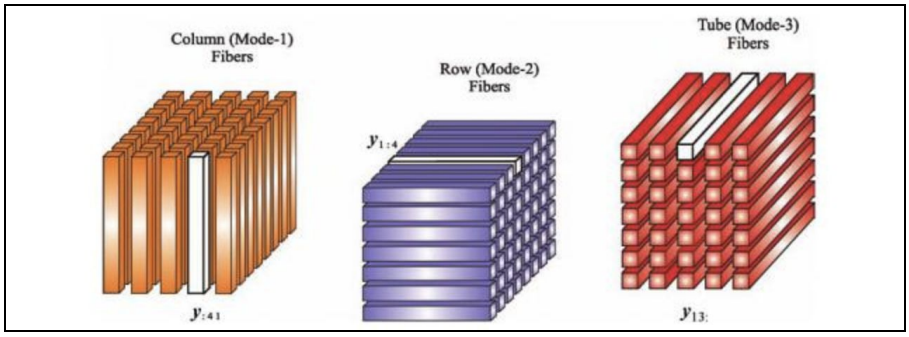


2. <u>Tensor Indexing</u>- We can create subarrays (or subfields) by fixing some of the given tensor's indices. Fibers are created when fixing all but one index, slices (or slabs) are created when fixing all but two indices. For a third order tensor the fibers are given as $x_{:jk} = x_{jk}$ (column), $x_{i:k}$ (row), and $x_{ij:}$ (tube); the slices are given as $X_{::k} = X_k$ (frontal), $X_{:j:}$ (lateral), $X_{i::}$ (horizontal).



Tensor Indexing

3. <u>Unfolding</u>- Also called matrization, unfolding a tensor is done by reading the element in a given way as to obtain a matrix instead of a tensor. It is done by stacking the fibers of the tensor into a matrix. For a tensor of size (I_1, I_2, …., I_n) , the k-mode unfolding of this tensor will be of size (I_k, I_1 x I_2 x ….. X I_{k-1} x I_{k+1} x ….. X I_n).



Example: For the previous tensor X, we have-

```
tl.unfold(X, mode=0)
```

```
array([[ 0,  1,  2,  3,  4,  5,  6,  7],
       [ 8,  9, 10, 11, 12, 13, 14, 15],
       [16, 17, 18, 19, 20, 21, 22, 23]])
```

```
tl.unfold(X, mode=1)
```

```
array([[ 0,  1,  8,  9, 16, 17],
       [ 2,  3, 10, 11, 18, 19],
       [ 4,  5, 12, 13, 20, 21],
       [ 6,  7, 14, 15, 22, 23]])
```

```
tl.unfold(X, mode=2)
```

```
array([[ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18, 20, 22],
       [ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19, 21, 23]])
```

4. Folding - Folding is the inverse operation: we can fold an unfolded tensor back from matrix to full tensor using the tensorly fold function.
   Example:

```
unfolding = tl.unfold(X, 1)
original_shape = X.shape
tl.fold(unfolding, mode=1, shape=original_shape)
```

```
array([[[ 0,  1],
        [ 2,  3],
        [ 4,  5],
        [ 6,  7]],

       [[ 8,  9],
        [10, 11],
        [12, 13],
        [14, 15]],

       [[16, 17],
        [18, 19],
        [20, 21],
        [22, 23]]])
```

5. N-mode product - It is also known as tensor contraction. This is a natural generalization of matrix-vector and matrix-matrix products. When multiplying a tensor by a matrix or a vector, we now have to specify the mode n along which to take the product.
   1. Tensor times matrix -
      We are doing an operation analogous to a matrix multiplication on the n-th mode. Given a tensor X of size (I_1, I_2, …., I_N) and a matrix M of size (D,I_n), the n-mode product of tensor X by matrix M is written X x_n M and is of size- (I_k, I_1 x I_2 x ….. X I_{n-1} x D x I_{n+1} x ….. X I_N).
   2. Tensor times vector-
      We are contracting over the nth mode by multiplying it with a vector. Given a tensor X of size (I_1, I_2, …., I_N) and a vector v of size (I_n), the n-mode product of X by v is written as X x_n v and is of size (I_k, I_1 x I_2 x ….. X I_{n-1} x D x I_{n+1} x ….. X I_N).

6. Matrix Decomposition - Matrix decompositions are important techniques used in mathematical settings, such as the implementation of numerical algorithms, the solution of linear equation systems, and the extraction of quintessential information from a matrix.the main focus will be on the rank decomposition of a matrix, an information extraction technique, which can be formally expressed

$$\boldsymbol{M} = \boldsymbol{A}\boldsymbol{B}^T \quad \text{with} \quad \boldsymbol{M} \in \mathbb{R}^{n\times m},\ \boldsymbol{A} \in \mathbb{R}^{n\times r},\ \boldsymbol{B}^T \in \mathbb{R}^{r\times m}$$

where r represents the rank of the decomposition. Intuitively, this decomposition aims at explaining the matrix M through r latent factors, which are encoded in the matrices A and B^T . The problem with lots of matrix factorization approaches is the fact that they are considered non-unique, meaning that a number of matrices A and B^T can give rise to a specific M .

In order to ensure uniqueness, additional constraints need to be imposed on a matrix, like positive-definiteness or orthogonality. In contrast, tensors do not require such strong constraints in order to oer a unique decomposition.

7. Spearman's Hypothesis - In 1904, Charles Spearman, a British psychologist, supposed that human intelligence can be broken down into two (hidden) factors: eductive (the ability to make sense out of complexity) and reproductive (the ability to store and reproduce information) intelligence .

Spearman therefore was one of the first scientists to carry out an unsupervised learning technique on a data set, which is nowadays referred to as factor analysis. To test his hypothesis, he invited s students to take t tests and noted down their scores in a matrix M $\in R^{s\times t}$. He was then wondering whether it would be possible to uniquely decompose the matrix M into two smaller matrices

$$\mathbf{A} \in R^{s\times h} \text{ and } B^T \in R^{h\times t}$$

through the h = 2 latent factors described above. According to his explanation, A would hold a list of students with their corresponding eductive and reproductive capabilities and B^T would hold a list of tests with their corresponding eductive and reproductive requirements.

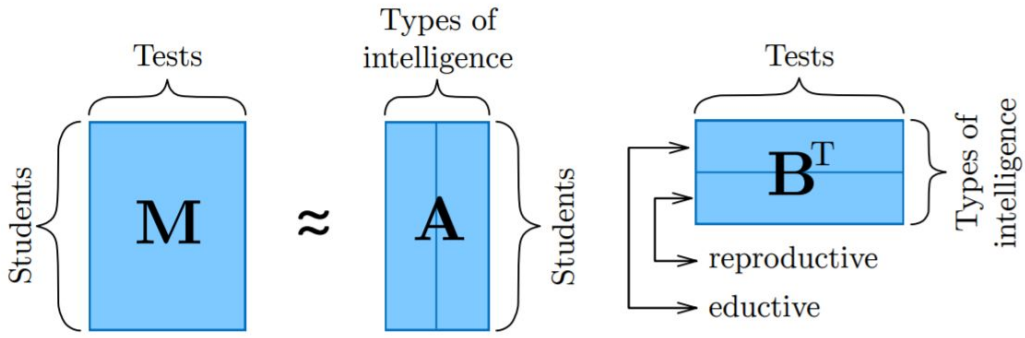


Figure 1: Spearman's hypothesis

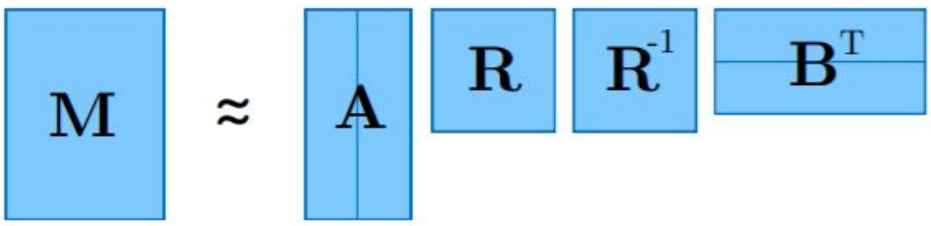
8. <u>Rotation Problem</u> - Given a matrix M, we would like to approximate it as well as possible with another matrix $\widehat{M}$ of a lower rank. Formally, the objective can be seen as minimizing the norm of the difference between the two matrices,

$$\hat{M} = AB^T = ARR^{-1}B^T = (AR)(R^{-1}B^T)$$
$$= (AR)(BR^{-T})^T = \tilde{A}\tilde{B}^T$$

However, this decomposition is not unique. By inserting an invertible rotation matrix R together with its inverse R^{-1} between A and B^T and absorbing R on the left with A and R^{-1} on the right with B^T we can again construct two matrices A˜ and $B^{\sim T}$.

$$\hat{M} = AB^T = ARR^{-1}B^T = (AR)(R^{-1}B^T)$$
$$= (AR)(BR^{-T})^T = \tilde{A}\tilde{B}^T$$

Representation of the rotation problem

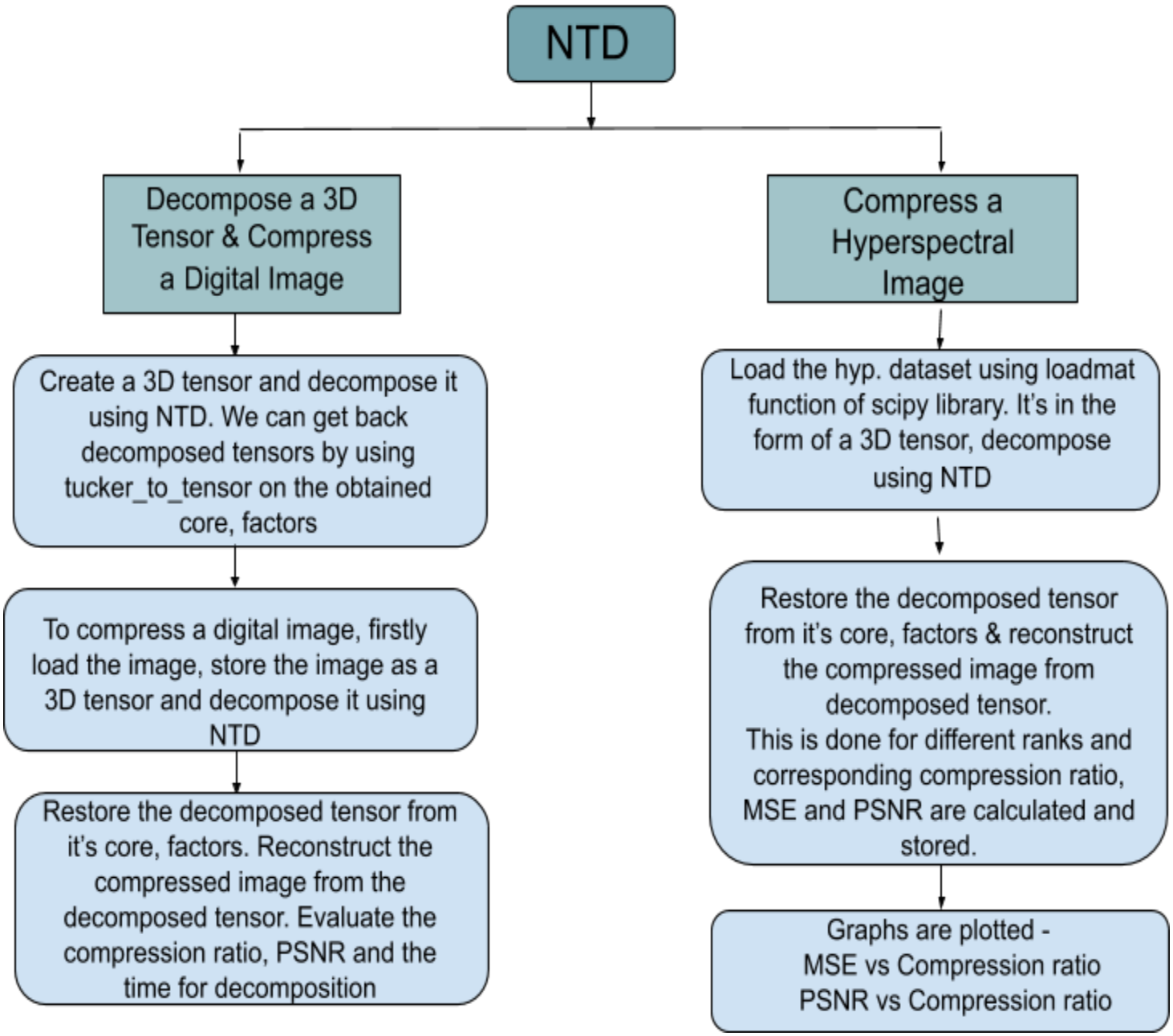


9. Non-Negative Tucker Decomposition - Nonnegative Tucker decomposition (NTD) is a powerful tool for the extraction of nonnegative parts-based and physically meaningful latent components from high-dimensional tensor data while preserving the natural multilinear structure of data.
   It decomposes a tensor into a so-called core tensor and multiple matrices which correspond to different core scalings along each mode. Therefore, the Tucker decomposition can be seen as a higher-order PCA(principal component analysis).

10. Image Compression
    Image compression is a type of data compression applied to digital images, to reduce their cost for storage or transmission. Algorithms may take advantage of visual perception and the statistical properties of image data to provide superior results compared with generic data compression methods which are used for other digital data.

    Lossy and Lossless Image Compression -
    The lossy compression technique does not restore the data in its original form, after decompression. Data's quality is compromised. It reduces the size of data and has more data-holding capacity.Lossy image compression (LIC), which aims to utilize inexact approximations to represent an image more compactly, is a classical problem in image processing.
    On the other hand lossless compression restores and rebuilds the data in its original form, after decompression. But Lossless Compression does not compromise the data's quality and does not reduce the size of data. Lossless Compression has less data-holding capacity than Lossy compression technique.
    Tensor Decomposition comes under lossy compression and hence can achieve a high level of data compression by removing the non-useful part of the data.

METHODOLOGY USED



ALGORITHM

Algorithm for Hyperspectral Image Compression using NTD -

1. All the libraries required for importing the hyperspectral dataset, compressing the tensor, and plotting required graphs, are imported.
2. Load the hyperspectral dataset using loadmat function of the class scipy.io
3. Read the required data (a 3D tensor) by accessing a specific key from the dataset which is in the form of a dictionary.
4. We obtained the hyperspectral image in the form of a 3D tensor in the previous step.
5. Decompose the 3D tensor using the non_negative_tucker function of the tensorly library, into core and factors.
6. The decomposed tensor is restored from its core, factors by using the tucker_to_tensor function of the tensorly library.
7. The image compression ratio, decomposition time, Mean Squared Error (MSE) and the Peak Signal-to-Noise Ratio (PSNR) are calculated.
8. The above decomposition algorithm is applied for different ranks, and each time, we calculate the compression ratio, PSNR, and MSE.
9. Graphs showing the variation of MSE with compression ratio and PSNR with compression ratio are plotted.

Function used - non_negative_tucker from tensorly.decomposition

Arguments of the function

1. rank - number of components
2. n_iter_max - maximum number of iterations
3. init - the initialization algorithms {‘svd’,’random’}
4. verbose - level of verbosity
5. ranks - size of core tensor
6. core- positive core of the tucker decomposition has shape ranks. (ndarray)
7. factors- ndarray list

Pseudocode for Hyperspectral Image Compression using NTD -

1. Import all the required libraries
    a. import spectral
    b. from spectral import *
    c. from scipy.io import loadmat
    d. import numpy as np
    e. import tensorly as tl
    f. import matplotlib.pylab as plt
    g. import time
    h. from tensorly.decomposition import non_negative_tucker
    i. from PIL import Image
    j. from skimage.measure import compare_psnr
    k. random_state = 1234
2. Set time0 = time.time()
3. Load the HSI using loadmat to X, where X is a 3D array
4. Create a 3D array, data = X
5. data_float = data.astype(np.float32)
6. compressing the data set using NTD
    a. tucker_ranks = [100, 50, 10] (variable)
    b. core, factors = non_negative_tucker(data_float, rank=tucker_ranks, n_iter_max=1000,init='random', tol=0.0001, random_state=random_state, verbose=True)
    c. tucker_tensor=(core, factors)
7. storing back the decomposed tensor
    a. data_reconstruction = tl.tucker_to_tensor(tucker_tensor, transpose_factors=False)
8. calculating the compression ratio
    a. size_decomposition = sum([factor.size for factor in factors]) + core.size
    b. compression_ratio = (data.size / size_decomposition)
9. calculating the psnr
    a. psnr1 = compare_psnr(data, im_reconstruction)
10. calculating the decomposition time
    a. 'Decomposition Time= str(time.time() - time0))
11. calculating the mean square error
    a. msqer = np.mean((im_reconstruction-data)**2)
12. Repeat from step 6 by varying the tucker_rank
13. Obtain different compression ratio, psnr value and mse value
14. Plot a graph showing variation of Mean Squared Error with the Compression Ratio
15. Plot a graph showing variation of PSNR with the Compression Ratio

DELIVERABLES

1. Source Code :
    - Compression of 3D tensor using NTD algorithm.
    - Compression of digital image (png) using NTD algorithm.
    - Compression of hyperspectral image using NTD algorithm, plotting the graphs of Compression Ratios vs Obtained Mean Square Error (MSE) and Compression Ratios vs Peak Signal-to-Noise Ratio (PSNR) for different values of decomposed tensors using the suitable libraries.

2. Power Point Presentation regarding the same.

LEARNINGS

- Overall this project was a very learning experience for us. It introduced us to a lot of new terminologies and their significance in the real world.

- Dealing with such common problems of handling large datasets.

- Having less experience in Python as a programming language, we came across a lot of libraries, like matplotlib and how to plot graphs for our results.

- Tensorly was one primary library that has a lot of inbuilt functions to make the work relatively easy, like tensorly.decomposition and tensorly.fold and tensorly.unfold. Then Scipy.io, through which we loaded a mat file. We used a few other libraries like NumPy, time, PIL(Python Image Library), and Skimage.measure.

- The main aim of this project was to decompose a 3D matrix or a tensor into components of a lesser dimension. Which would further be applied to an image to decompose it, but why is decomposition significant? Assuming that we have a vast dataset, where each column can be an observation and each row is a feature, it would be necessary to reduce such a large dataset to allow a more straightforward computation.

- Once we are done with the decomposition of the hyperspectral image, we restore the decomposed tensor from its core, factors, and this is done for different ranks of the core tensor. We store the different values of compression ratio, MSE, and PSNR values, respectively, for different ranks. Then studied how the MSE and PSNR values vary with the compression ratio of the image by plotting the respective graphs.

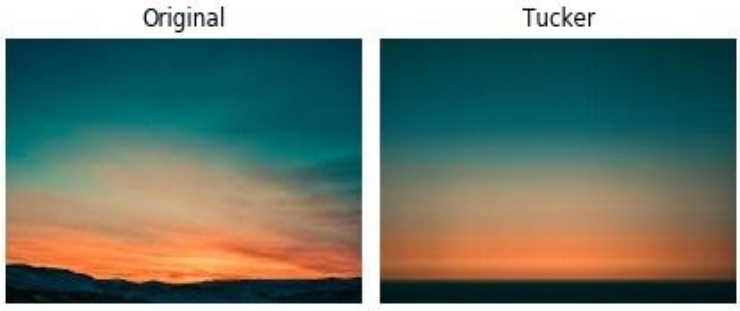
RESULTS

1. Decomposition of a 3D tensor using the NTD algorithm -

   Input- A 3D tensor of dimension (6 x 4 x 2)
   Output- Dimension of core tensor (3 x 2 x 1)

2. Image Compression using NTD - (png image)

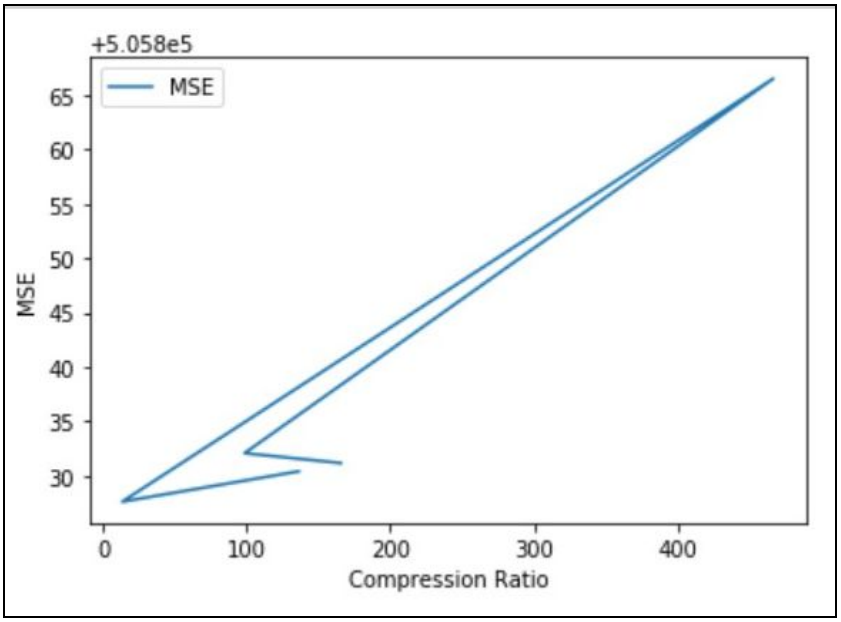
   Image Compression Ratio - 0.7989823394313772
   Image Compare PSNR - 21.102074700933812
   Decomposition time - 18.530953407287598



3. Image compression using NTD algorithm on Hyperspectral Data -

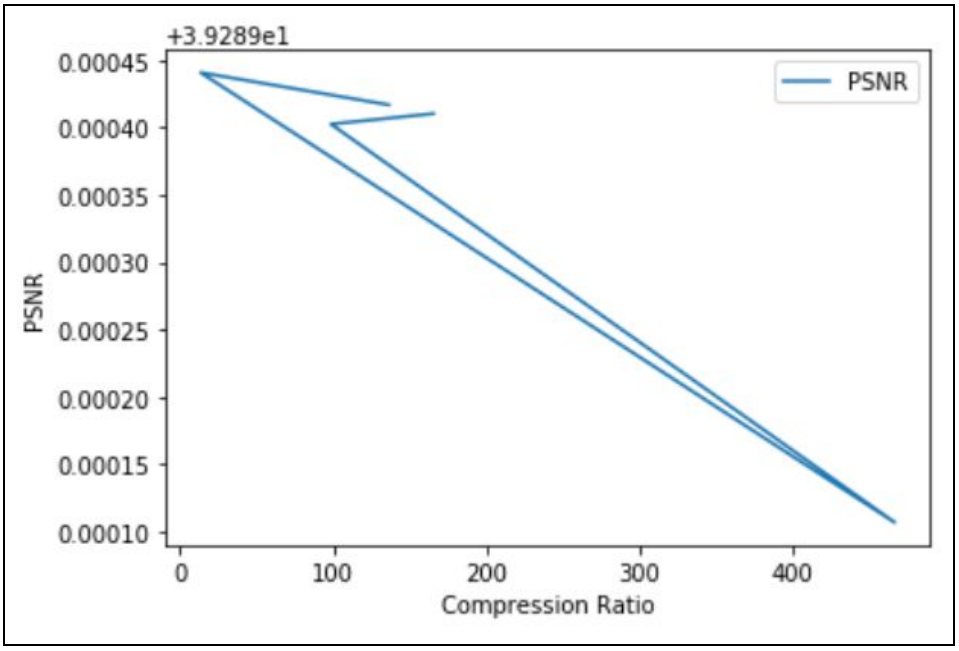
The following are the various values obtained for PSNR, Compression Ratio and Mean Square Error (MSE) - (for different values of ranks given)

Ranks	Compression Ratio	PSNR	Mean Square Error
[100, 50, 10]	165.5599472990777 2	39.28941015484052 5	505831.1751257361
[200, 40, 10]	98.61145732354706	39.28940237295465 5	505832.0814978326
[50, 25, 5]	466.7802906150988 6	39.28910692490837	505866.4941348737 3
[300, 150, 30]	13.45998021536270 7	39.28944062333619	505827.6264136184 5
[100, 40, 20]	136.3602706498148 8	39.28941667853918	505830.4152985179 7



Graph showing variation of Mean Squared Error with the Compression Ratio

(SCALED)



Graph showing variation of PSNR with the Compression Ratio (SCALED)

REFERENCES

1. Introduction to Tensor Decompositions and their Applications in Machine Learning by Stephan Rabanser,Oleksandr Shchur and Stephan Günnemann
2. TensorLy: Tensor Learning in Python by Jean Kossaifi , Yannis Panagakis and Maja Pantic
3. Nonnegative Tucker Decomposition by Yong-Deok Kim, Seungjin Choi
4. https://github.com/TensorDecompositions/NTFk.jl
5. http://tensorly.org/stable/auto_examples/decomposition/plot_image_compression.html
6. http://tensorly.org/stable/_modules/tensorly/decomposition/_tucker.html#non_negative_tucker
7. http://tensorly.org/stable/modules/generated/tensorly.decomposition.non_negative_tucker.html#tensorly.decomposition.non_negative_tucker
8. https://en.wikipedia.org/wiki/Tucker_decomposition
9. https://en.wikipedia.org/wiki/Tensor
10. https://en.wikipedia.org/wiki/Tucker_decomposition
11. Different repositories in GitHub