**ADVANCED ALGORITHMS**

**Project #2**

**Image Compression Using SVD and Dimensionality Reduction Using PCA**

**By**

**Sukesh Reddy (sg121)**

**Shashank Reddy (sa123)**

**Vandana Rao(vk34)**

**Rishi Reddy (rc81)**

**Professor**

**Dr. Zhong-Hui Duan**

**Academic Integrity Pledge**

Program: - Image compression using SVD and Dimensionality Reduction Using PCA

Programmer: - Sukesh Reddy, Shashank Reddy, Vandana Rao and Rishi Reddy

Academic Integrity Pledge

* All source code and documentation used in my program is either my original work, or was derived, by me, from the source code published in the text book for this course or presented in class. Any source code in this project that is derived from code published elsewhere is documented as such so the original author receives due credit.
* I have not used source code obtained from another student, or any other unauthorized source, either modified or unmodified. I have not helped another student write their program by providing a printed or electronic copy of my solution.
* I have not discussed coding details about this project with anyone other than my instructor. I understand that I may discuss the concepts of this program with other students, and that another student may help me debug my program. However, the responsibility to write each program belongs solely to the program's author.
* I have violated neither the spirit nor letter of these restrictions.

Sukesh Reddy

Shashank Reddy

Vandana Rao

Rishi Reddy 3/29/2014

Signature of Programmer

**Introduction:**

SINGULAR VALUE DECOMPOSITION (SVD) is an effective tool for minimizing data storage and data transfer. The singular value decomposition of an m×n matrix M is a factorization of the form M=UΣV\* where U is a m×m unitary matrix, Σ is an m×n rectangular diagonal matrix with non-negative numbers on the diagonal, and V\* is the transpose of a unitary matrix V. The objective of this part of the project is to understand how SVD works and the application of SVD to image compression.

Principal component analysis (PCA) is a very popular technique for dimensionality reduction. Given a set of data on “n” dimensions, PCA aims to find out a linear subspace of dimension “d” lower than “n” such that data points lie mainly on this linear subspace. Such a reduced subspace attempts to main most of the variability of the data. The linear subspace can be specified by “d” orthogonal vectors that form a new coordinate system called the principal components. These principal components are orthogonal. The objective of this part is to understand how PCA works, the connection between SVD and PCA, and the application of PCA to dimensionality reduction.

**Implementation:**

The Program has been implemented basically in 4 steps as follows:

**Step1:** Converting decimal pgm values to ascii values:

1. Initially the pgm file contains decimal values. The first part of the project is where the decimal values are further converted to Binary values and then to ascii values to reduce the data storage. We finally store the ascii values in a pgm file.

Storing the height, width, greyscale and remaining values in bytes:

1. The width is saved in 2 bytes and height are saved in 2 bytes and the greyscale value is saved using 1 byte and the total number of pixels are saved in rows (m) \* columns (n) bytes.

**TestCase1**: mona\_lisa.pgm

To store the entire matrix

250 360

255

……………

We store them in 250 in 2 bytes and 360 in 2 bytes and 255 in 1 byte and the total rows and columns in 250\*360=90000 bytes so therefore the total 90000+2+2+1=90005 bytes.

**Output:**

|  |  |
| --- | --- |
|  |  |

Figure 1: The image size comparison between original image and ascii image.

**Time** **Complexity:** The Time Complexity for this O(n²) since we traverse through each row and column to store in a matrix and then we have O(n²) to convert each row and column to ascii values storing each row and column and greyscale values it takes O(1) time therefore the time complexity for this matrix would be O(n²).

|  |  |  |
| --- | --- | --- |
| **File Size** | **Need to compressed** | **Compressed File size** |
| 360761 bytes | 90005 bytes | 90005 bytes |
| 519 bytes | 173 bytes | 173 bytes |
| 1191978 bytes | 262149 bytes | 262149 bytes |
| 299466 bytes | 65541 bytes | 65541 bytes |

**Table 1:** Illustrates the file compression for various files.

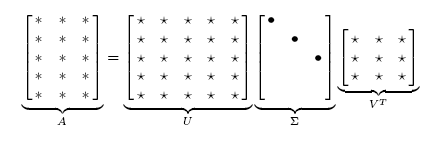
**Step2:** Convert this binary file to ascii file:

In Step2 we would convert the ASCII image file back to the original decimal image file.

**Time Complexity**: The Time Complexity for this O(n²) since we traverse through each row and column to convert to each ASCII pixel back to binary and then we have O(n²) to convert each row and column to decimal values storing each row and column and greyscale values it takes O(1) time therefore the Time complexity for this matrix would be O(n²).

**Step 3:** Storing the necessary information for the given SVD and header file based on rank.

1. Given the SVD and Header file we store the values in three separate matrices.
2. We then Transpose V to get V^T.
3. We store the row, column and rank from the header file into a image\_b.pgm.SVD file.
4. We convert the U, S and V^T matrices to ascii and store them in the image\_b.pgm.SVD file.
5. Here we will use 2 bytes for row , column and rank each, 3 bytes for S values and 2 bytes for U and V^T matrices.



**Figure 2**: The USV^T matrices to store the matrices in ASCII format along with their respective rows and columns.

JAMA is a basic linear algebra package for Java. It provides user-level classes for constructing

and manipulating real dense matrices. It is meant to provide sufficient functionality for routine

Problems, packaged in a way that is natural and understandable to non-experts. It is mainly comprises of 6 Java classes Matrix, CholeskyDecomposition, LUDecomposition, QRDecomposition, SingularValueDecomposition and EigenvalueDecomposition.

**Step 4:** Converting the image back to the original image.

1. We read the row, column and rank from the ascii file.
2. Then read the U, S and V^T to the matrices.
3. We are using Jama jar file to multiply the matrices.

**TESTCASES:**

**Test Case 1:**

|  |  |
| --- | --- |
| C:\Users\Shashank\Desktop\pic outputs\moriginal.PNG  Original Image | C:\Users\Shashank\Desktop\pic outputs\moriginal_dup.PNG  Rank: 256 |
| **Rank : 150** | **Rank: 20** |

**Test Case 2:**

|  |  |
| --- | --- |
| C:\Users\Shashank\Desktop\pic outputs\cuporiginal.PNG | C:\Users\Shashank\Desktop\pic outputs\kduplicate.PNG |

**Time complexity:**

We need to multiply the Matrices U, S and V^T so multiplying these matrices will take O(n³) complexity.

**Individual Contribution:**

* Program 1 : Shashank Reddy and Vandana
* Program 2: Shashank Reddy and Vandana
* Program 3: Sukesh and Rishi
* Program 4: Sukesh and Rishi
* PCA : Sukesh , Vandana , Shashank and Rishi
* Documentation and presentation: Sukesh , Vandana , Shashank and Rishi

**Problems faced:**

While implementing we need to save the USV values in a file after converting to ascii values but we need to precise the value of U, S and V^T so we have rounded up up to 2 decimal value after the dot. Values which have exponent when read as integer the value is not getting converted into its original form so we need to read them as double and then convert them back to string.

**Conclusion:**

We have implemented the two parts of the project successfully and have compressed the image and have obtained the original image back. We have checked for the compressed image size with the original image.

**References:**

1. <http://en.wikipedia.org/wiki/Singular_value_decomposition>
2. <http://www.ling.ohio-state.edu/~kbaker/pubs/Singular_Value_Decomposition_Tutorial.pdf>