DEPARTMENT OF COMPUTER SCIENCE

UNIVERSITY OF MANAGEMENT AND TECHNOLOGY

PROJECT REPORT

ANALYSIS OF ALGORITHMS
DYNAMIC PROGRAMMING REAL WORLD APPLICATION

Image Processing

Resource Person: Amjad Ali

GROUP MEMBERS:

Student Id	***Student Name***
F2021266461	*Muhammad Suban*
F2021266098	*Javeria Rizwan*
F2021266550	*Sibgha Zeeshan*
F2021266503	*Areeba Noor*

Table of Contents

ANALYSIS OF ALGORITHMS
DYNAMIC PROGRAMMING REAL WORLD APPLICATION
Image Processing

Abstract

Matrix Chain Multiplication (MCM) is a fundamental problem in computer science with applications in various fields such as optimization, computer graphics, and image processing. In this project, we explore the mapping of matrix chain multiplication to the context of image processing, specifically for RGB image reconstruction. The central objective of this research is to explore an application of Dynamic Programming in image processing. We integrate the concept of matrix multiplication, dynamic programming, and other advanced approaches within the context of image processing

Problem Statement:

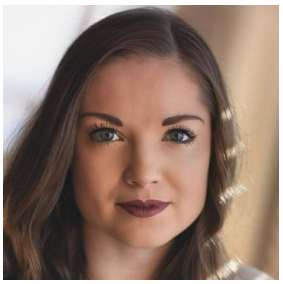
RGB image represented as three matrices (red, green, and blue channels), the goal is to find the optimal sequence of matrix chain multiplications to reconstruct the original image. Additionally, we aim to analyze the computational cost associated with this matrix chain multiplication.

1. Introduction

What is a Digital Image?

A digital image is a representation of a real image as a set of numbers that can be stored and handled by a digital computer. In order to translate the image into numbers, it is divided into small areas called pixels (picture elements). For each pixel, the imaging device records a number, or a small set of numbers, that describe some property of this pixel, such as its brightness (the intensity of the light) or its color. The numbers are arranged in an array of rows and columns that correspond to the vertical and horizontal positions of the pixels in the image.

A color image can have three colors, normally RGB (Red, Green, Blue)



Original Image

Pixels
(1023x1023)

RED: 158
GREEN: 123
BLUE: 93

Color Channels

What is Image Processing?

Image processing is a method to convert an image into digital form and perform some operations on it, in order to get an enhanced image or to extract some useful information from it. It is a type of signal dispensation in which input is image, like photograph and output may be image or characteristics associated with that image. Usually Image Processing system includes treating images as two dimensional signals while applying already set signal processing methods to them.

2. Tools and Techniques:

These are the following tools we have used in our project

Python:

Python is a popular programming language in the field of image processing and computer vision. Because of its readability, extensive libraries and open source nature we use python.

Libraries:

Some of the libraries that have been integrated in our project includes:

1. *Open CV:*

   OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. The library has more than 2500 optimized

algorithms, which includes a comprehensive set of both classic and state-of- the-art computer vision and machine learning algorithms. We use this library for:

- *Reading, Writing and Displaying Images*
- *Changing Color Spaces*

2. *Numpy:*

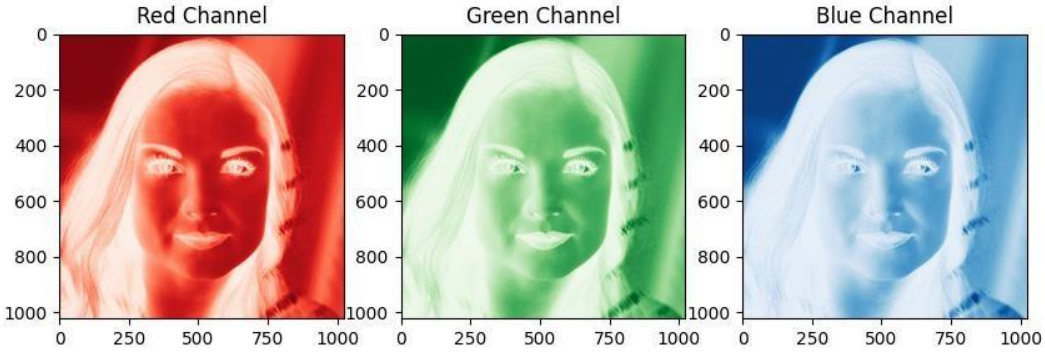
   NumPy is the fundamental package for scientific computing with Python. NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be denied. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

3. RGB Color Space:

Probably the most popular color space. It stands for Red, Green, and Blue. In this color space, each color is represented as a weighted combination of red, green, and blue. So every pixel value is represented as a tuple of three numbers corresponding to red, green, and blue. Each value ranges between 0 and 255.

Data Extraction

- *An RGB image loaded using OpenCV (cv2.imread)*
- *Visualizing and Displaying the channel images separately*



- *Matrices representing each channel*

RGB Matrices:

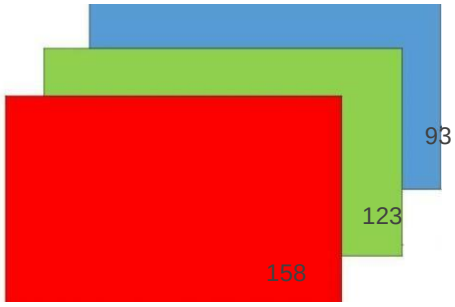
```
Red Channel Matrix: [[242 242 242 ... 196 195 195]
 [242 242 242 ... 196 195 195]
 [242 242 242 ... 196 195 195]
 ...
 [ 55  54  54 ... 158 158 158]
 [ 54  54  53 ... 158 158 158]
 [ 54  53  53 ... 158 158 158]]

Green Channel Matrix: [[242 242 242 ... 168 169 169]
 [242 242 242 ... 168 169 169]
 [242 242 242 ... 168 169 169]
 ...
 [ 47  46  46 ... 123 123 123]
 [ 46  46  45 ... 123 123 123]
 [ 46  45  45 ... 123 123 123]]

Blue Channel Matrix: [[242 242 242 ... 144 144 144]
 [242 242 242 ... 144 144 144]
 [242 242 242 ... 144 144 144]
 ...
 [ 44  43  43 ...  93  93  93]
 [ 43  43  42 ...  93  93  93]
 [ 43  42  42 ...  93  93  93]]
```

How these values obtained?

RGB Matrices values are obtained by:

1. ***Pixel Representation:***
   Each pixel in a digital image is represented by a combination of Red, Green, and Blue intensity values.
2. ***Color Channels:***
   The RGB color model uses three color channels: Red, Green, and Blue. Each channel is represented by a matrix with the same dimensions as the image
3. ***Matrix Elements:***
   The individual elements of the matrices represent the intensity values of the corresponding color channel for each pixel. These values typically range from 0 to 255, where 0 is the minimum intensity (no color contribution), and 255 is the maximum intensity (full color contribution)
4. ***Color Composition:***
   The color of each pixel is composed by combining the intensities from the three color channels. For example, if a pixel has (R, G, B) values of (255, 0, 0), it represents a fully saturated red color because the Red channel is at its maximum intensity.

Original Image (1024 x 1024)



RGB

Pixel at (1023 x 1023)

Pixels Values Generation:

Pixels are the smallest units that make up a digital image. These pixels collectively form the visual representation of an image on digital display. Each pixel is represented by a combination of numerical values. In RGB image each pixel is represented by three values corresponding to the intensity of Red, Green and Blue channels. The combination of these channels determines the overall color of the pixel.

As in an RGB image, each pixel has three color channels red, green and blue. Total numbers of pixels are obtained as

Total number of pixels = Width x Height x Number of Channels

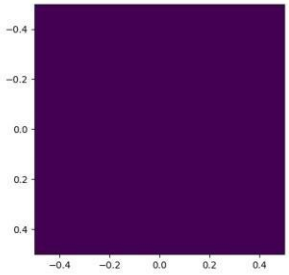
```
Total number of pixels in the image: 3145728
```

RGB PIXEL:

```
Pixel of original Image:
[[[158 110  88]]]
Pixel of Blue Image:
[[88]]
```

X = 0.327 Y=0.380
[158, 110, 88]

Blue Pixel:



X = 0.133, Y= -0.147

[88]

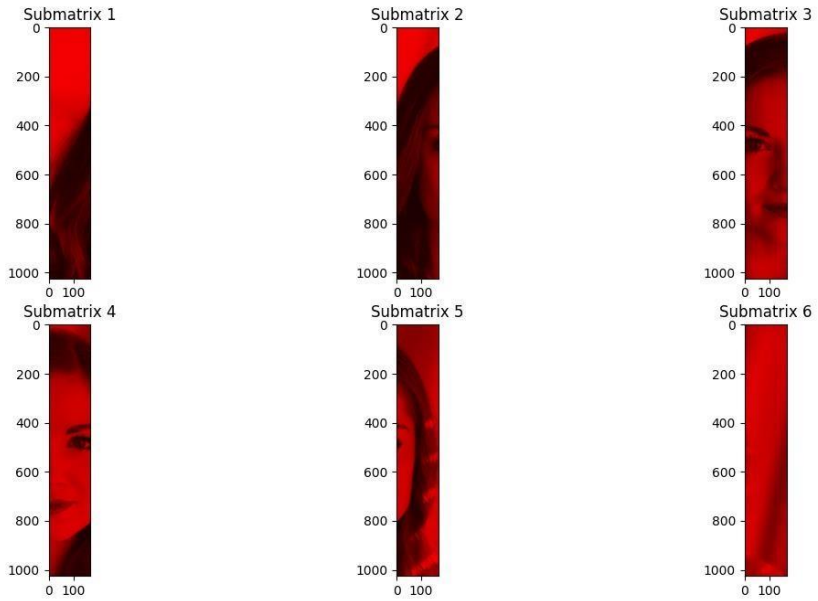
Matrix Division:

The division of original image into multiple submatrices is follows:

Red Channel:

```
Original Red Channel Matrix:
[[242 242 242 ... 196 195 195]
 [242 242 242 ... 196 195 195]
 [242 242 242 ... 196 195 195]
 ...
 [ 55  54  54 ... 158 158 158]
 [ 54  54  53 ... 158 158 158]
 [ 54  53  53 ... 158 158 158]]

Submatrix Data for Red Channel:

Submatrix 1 Dimensions: (1024, 170)
[[242 242 242 ... 242 242 242]
 [242 242 242 ... 242 242 242]
 [242 242 242 ... 242 242 242]
 ...
 [ 55  54  54 ...  41  39  38]
 [ 54  54  53 ...  41  39  38]
 [ 54  53  53 ...  41  39  38]]

Submatrix 1 Indices: Start Row: 0, End Row: 1024, Start Col: 0, End Col: 170

Submatrix 2 Dimensions: (1024, 171)
[[242 242 242 ... 211 213 215]
 [242 242 242 ... 210 212 214]
 [242 242 242 ... 210 212 213]
 ...
 [ 36  35  34 ... 145 144 141]
 [ 36  35  34 ... 145 144 143]
 [ 36  35  34 ... 144 144 144]]

Submatrix 2 Indices: Start Row: 0, End Row: 1024, Start Col: 170, End Col: 341

Submatrix 3 Dimensions: (1024, 171)
[[216 218 218 ... 216 216 216]
 [215 217 217 ... 215 215 215]
 [215 216 217 ... 214 214 213]
 ...
 [138 136 134 ... 154 153 153]
 [140 138 136 ... 153 152 152]
 [142 140 138 ... 153 152 151]]
```

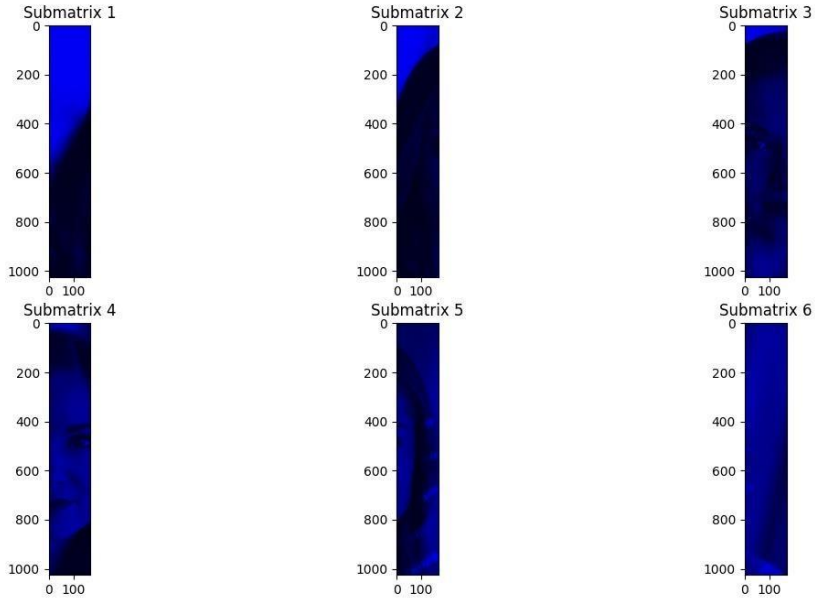
```
Submatrix 3 Indices: Start Row: 0, End Row: 1024, Start Col: 341, End Col: 512

Submatrix 4 Dimensions: (1024, 170)
[[216 216 216 ... 145 144 144]
 [215 214 214 ... 145 144 144]
 [212 212 212 ... 145 144 144]
 ...
 [152 150 148 ...  66  65  62]
 [150 149 147 ...  65  66  63]
 [149 148 147 ...  62  66  64]]

Submatrix 4 Indices: Start Row: 0, End Row: 1024, Start Col: 512, End Col: 682

Submatrix 5 Dimensions: (1024, 171)
[[144 144 144 ... 162 163 162]
 [144 144 144 ... 162 163 162]
 [144 144 144 ... 162 163 162]
 ...
 [ 60  59  60 ... 151 154 158]
 [ 61  60  61 ... 150 154 157]
 [ 61  60  61 ... 150 154 157]]

Submatrix 5 Indices: Start Row: 0, End Row: 1024, Start Col: 682, End Col: 853

Submatrix 6 Dimensions: (1024, 171)
[[163 164 165 ... 196 195 195]
 [163 164 165 ... 196 195 195]
 [163 164 165 ... 196 195 195]
 ...
 [161 164 165 ... 158 158 158]
 [160 162 163 ... 158 158 158]
 [159 161 162 ... 158 158 158]]

Submatrix 6 Indices: Start Row: 0, End Row: 1024, Start Col: 853, End Col: 1024
```



II *Blue Channel:*

```
Original Blue Channel Matrix:
[[242 242 242 ... 144 144 144]
 [242 242 242 ... 144 144 144]
 [242 242 242 ... 144 144 144]
 ...
 [ 44  43  43 ...  93  93  93]
 [ 43  43  42 ...  93  93  93]
 [ 43  42  42 ...  93  93  93]]

Submatrix Data for BLUE Channel:

Submatrix 1 Dimensions: (1024, 170)
[[242 242 242 ... 242 242 242]
 [242 242 242 ... 242 242 242]
 [242 242 242 ... 242 242 242]
 ...
 [ 44  43  43 ...  36  34  33]
 [ 43  43  42 ...  36  34  33]
 [ 43  42  42 ...  36  34  33]]

Submatrix 1 Indices: Start Row: 0, End Row: 1024, Start Col: 0, End Col: 170

Submatrix 2 Dimensions: (1024, 171)
[[242 242 242 ... 210 212 214]
 [242 242 242 ... 209 211 213]
 [242 242 242 ... 209 211 212]
 ...
 [ 31  30  29 ...  86  85  82]
 [ 31  30  29 ...  86  85  84]
 [ 31  30  29 ...  85  85  85]]

Submatrix 2 Indices: Start Row: 0, End Row: 1024, Start Col: 170, End Col: 341

Submatrix 3 Dimensions: (1024, 171)
[[215 217 217 ... 216 216 216]
 [214 216 216 ... 215 215 215]
 [214 215 216 ... 214 214 213]
 ...
 [ 79  77  75 ...  87  86  86]
 [ 81  79  77 ...  86  85  85]
 [ 83  81  79 ...  86  85  84]]
```

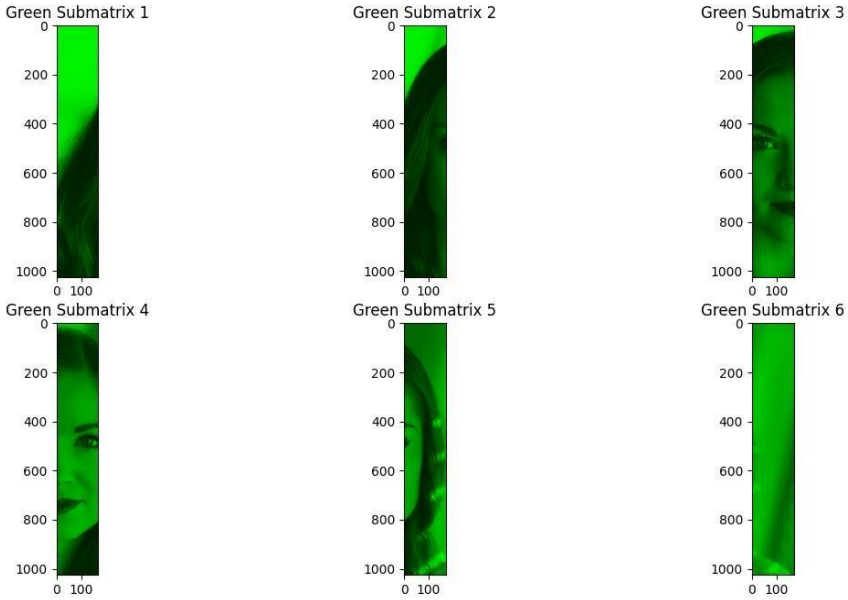
```
Submatrix 3 Indices: Start Row: 0, End Row: 1024, Start Col: 341, End Col: 512

Submatrix 4 Dimensions: (1024, 170)
[[216 216 216 ...  96  95  95]
 [215 214 214 ...  96  95  95]
 [212 212 212 ...  96  95  95]
 ...
 [ 83  81  79 ...  44  43  40]
 [ 81  80  78 ...  43  44  41]
 [ 80  79  78 ...  40  44  42]]

Submatrix 4 Indices: Start Row: 0, End Row: 1024, Start Col: 512, End Col: 682

Submatrix 5 Dimensions: (1024, 171)
[[ 95  95  95 ... 114 115 114]
 [ 95  95  95 ... 114 115 114]
 [ 95  95  95 ... 114 115 114]
 ...
 [ 38  37  38 ... 128 131 135]
 [ 39  38  39 ... 127 131 134]
 [ 39  38  39 ... 127 131 134]]

Submatrix 5 Indices: Start Row: 0, End Row: 1024, Start Col: 682, End Col: 853

Submatrix 6 Dimensions: (1024, 171)
[[115 116 117 ... 144 144 144]
 [115 116 117 ... 144 144 144]
 [115 116 117 ... 144 144 144]
 ...
 [138 141 142 ...  93  93  93]
 [137 139 140 ...  93  93  93]
 [136 138 139 ...  93  93  93]]

Submatrix 6 Indices: Start Row: 0, End Row: 1024, Start Col: 853, End Col: 1024
```



II *Green Channel:*

```
Original Green Channel Matrix:
[[242 242 242 ... 168 169 169]
 [242 242 242 ... 168 169 169]
 [242 242 242 ... 168 169 169]
 ...
 [ 47  46  46 ... 123 123 123]
 [ 46  46  45 ... 123 123 123]
 [ 46  45  45 ... 123 123 123]]

Submatrix Data for Green Channel:

Submatrix 1 Dimensions: (1024, 170)
[[242 242 242 ... 242 242 242]
 [242 242 242 ... 242 242 242]
 [242 242 242 ... 242 242 242]
 ...
 [ 47  46  46 ...  37  35  34]
 [ 46  46  45 ...  37  35  34]
 [ 46  45  45 ...  37  35  34]]

Submatrix 1 Indices: Start Row: 0, End Row: 1024, Start Col: 0, End Col: 170

Submatrix 2 Dimensions: (1024, 171)
[[242 242 242 ... 209 211 213]
 [242 242 242 ... 208 210 212]
 [242 242 242 ... 208 210 211]
 ...
 [ 32  31  30 ... 104 103 100]
 [ 32  31  30 ... 104 103 102]
 [ 32  31  30 ... 103 103 103]]

Submatrix 2 Indices: Start Row: 0, End Row: 1024, Start Col: 170, End Col: 341

Submatrix 3 Dimensions: (1024, 171)
[[214 216 216 ... 216 216 216]
 [213 215 215 ... 215 215 215]
 [213 214 215 ... 214 214 213]
 ...
 [ 97  95  93 ... 107 106 106]
 [ 99  97  95 ... 106 105 105]
 [101  99  97 ... 106 105 104]]
```

```
Submatrix 3 Indices: Start Row: 0, End Row: 1024, Start Col: 341, End Col: 512

Submatrix 4 Dimensions: (1024, 170)
[[216 216 216 ... 114 113 113]
 [215 214 214 ... 114 113 113]
 [212 212 212 ... 114 113 113]
 ...
 [106 104 102 ...  48  47  44]
 [104 103 101 ...  47  48  45]
 [103 102 101 ...  44  48  46]]

Submatrix 4 Indices: Start Row: 0, End Row: 1024, Start Col: 512, End Col: 682

Submatrix 5 Dimensions: (1024, 171)
[[113 113 113 ... 135 136 135]
 [113 113 113 ... 135 136 135]
 [113 113 113 ... 135 136 135]
 ...
 [ 42  41  42 ... 137 140 144]
 [ 43  42  43 ... 136 140 143]
 [ 43  42  43 ... 136 140 143]]

Submatrix 5 Indices: Start Row: 0, End Row: 1024, Start Col: 682, End Col: 853

Submatrix 6 Dimensions: (1024, 171)
[[136 137 138 ... 168 169 169]
 [136 137 138 ... 168 169 169]
 [136 137 138 ... 168 169 169]
 ...
 [147 150 151 ... 123 123 123]
 [146 148 149 ... 123 123 123]
 [145 147 148 ... 123 123 123]]

Submatrix 6 Indices: Start Row: 0, End Row: 1024, Start Col: 853, End Col: 1024
```



4. Matrix Chain Multiplication:

The MCM problem is solved using dynamic programming. The matrices are split into their respective RGB channels, and the dimensions of these matrices determine the chain of multiplications. The algorithm finds the optimal parenthesizing for matrix multiplication using a bottom-up approach, resulting in matrices M (cost table) and S (parenthesizing table)

Matrix M:

0	***1***	***2***	***3***	***4***	***5***	***6***
1	*0*	*1.07e+09*	*2.15e+09*	*3.22e+09*	*4.29e+09*	*5.37e+09*
2	*Inf*	*0*	*1.07e+09*	*2.15e+09*	*3.22e+09*	*4.29e+09*
3	*Inf*	*Inf*	*0*	*1.07e+09*	*2.15e+09*	*3.22e+09*
4	*Inf*	*Inf*	*Inf*	*0*	*1.07e+09*	*2.15e+09*
5	*Inf*	*Inf*	*Inf*	*Inf*	*0*	*1.07e+09*
6	*Inf*	*Inf*	*Inf*	*Inf*	*Inf*	*0*

Here e+09 represents 10^9 ***Matrix S:***

0	***1***	***2***	***3***	***4***	***5***	***6***
1		*1*	*1*	*1*	*1*	*1*
2			*2*	*2*	*2*	*2*
3				*3*	*3*	*3*
4					*4*	*4*
5						*5*
6						

```
Matrix M:
[[0.00e+00 1.07e+09 2.15e+09 3.22e+09 4.29e+09 5.37e+09]
 [     inf 0.00e+00 1.07e+09 2.15e+09 3.22e+09 4.29e+09]
 [     inf      inf 0.00e+00 1.07e+09 2.15e+09 3.22e+09]
 [     inf      inf      inf 0.00e+00 1.07e+09 2.15e+09]
 [     inf      inf      inf      inf 0.00e+00 1.07e+09]
 [     inf      inf      inf      inf      inf 0.00e+00]]

Matrix S:
[[0 1 1 1 1 1]
 [0 0 2 2 2 2]
 [0 0 0 3 3 3]
 [0 0 0 0 4 4]
 [0 0 0 0 0 5]
 [0 0 0 0 0 0]]
```

Total Matrices:

Channel	*Matrices*
Blue	***6***
Green	***6***
Red	***6***

Dimensions*: 1024 x 1024* ***Recursive***

Property:

$$A[i,j] = m[i, k] + m[k + 1, j] + p[i] * p[k] * p[j]$$

Optimal Solution:

RGB Channels:

```
Optimal Matrix Chain Multiplication Sequence with Dimensions:
Blue Channel:
(Matrix Blue A11 (1024x1024)(Matrix Blue A21 (1024x1024)(Matrix Blue A31 (1024x1024)(Matrix Blue A41 (1024x102
4)(Matrix Blue A51 (1024x1024)Matrix Blue A61 (1024x1024))))))
Green Channel:
(Matrix Green A12 (1024x1024)(Matrix Green A22 (1024x1024)(Matrix Green A32 (1024x1024)(Matrix Green A42 (1024
x1024)(Matrix Green A52 (1024x1024)Matrix Green A62 (1024x1024))))))
Red Channel:
(Matrix Red A13 (1024x1024)(Matrix Red A23 (1024x1024)(Matrix Red A33 (1024x1024)(Matrix Red A43 (1024x1024)(M
atrix Red A53 (1024x1024)Matrix Red A63 (1024x1024))))))Optimal Matrix Chain Multiplication Sequence:
(Matrix A10(Matrix A20(Matrix A30(Matrix A40(Matrix A50Matrix A60)))))
```

Cost Analysis:

The cost of multiplication is calculated based on the dimensions of matrices. The cost represents the total number of scalar multiplications required for the optimal sequence of matrix chain multiplication

RGB Channels:

```
Cost of Multiplication:
Blue Channel: 1073741824
Green Channel: 1073741824
Red Channel: 0
```

Optimal Sequence for image creation:

```
Optimal Matrix Chain Multiplication Sequence (Combined RGB - BGR) with Dimensions:
(Matrix Combined RGB A11 (1024x1024)(Matrix Combined RGB A21 (1024x1024)(Matrix Combined RGB A31 (1024x1024)(M
atrix Combined RGB A41 (1024x1024)(Matrix Combined RGB A51 (1024x1024)Matrix Combined RGB A61 (1024x1024))))))
```

Cost:

```
Total Cost of Image Recreation (Original Sequence): 2147483648
```

5. Smoothness:

There are following factors in smoothness of image:

Blurring:

Image blurring is achieved by convolving the image with a low-pass filter kernel. It is useful for removing noise. It actually removes high frequency content (e.g.: noise, edges) from the image resulting in edges being blurred when this is filter is applied

- *Averaging:*

  OpenCV provides a function, cv2.filter2D (), to convolve a kernel with an image. As an example, we will try an averaging filter on an image

Matrix Multiplication:

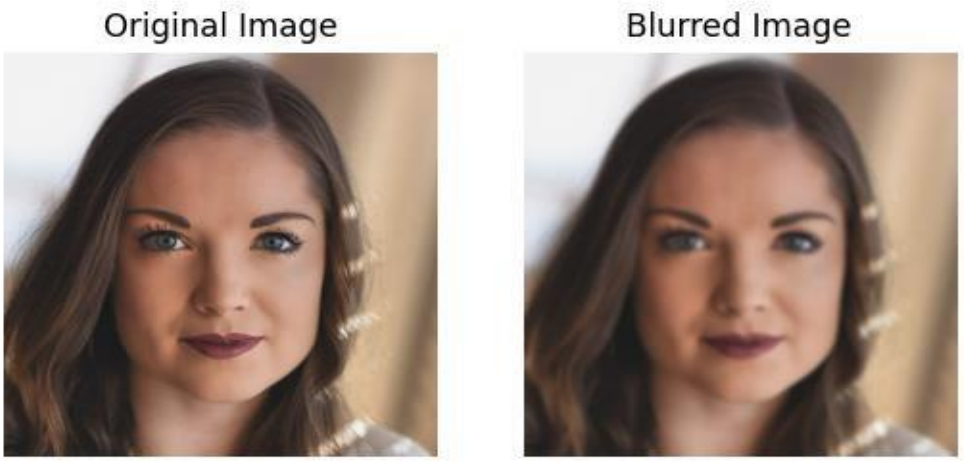
Here Matrix multiplication is involved in image blurring as:

1. ***Kernel Matrix:***

   The blurring kernel is a small matrix (usually square) that defines the weights for the convolution operation.

$$K = \frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

2. ***Pixel-wise Multiplication:***
   For each pixel in the image, the corresponding region of the image is multiplied element-wise with the kernel matrix.
3. ***Summation:***
   The results of the element-wise multiplications are then summed up to obtain the new pixel value.
4. ***Matrix Sliding:***
   This process is repeated for every pixel in the image by sliding the kernel matrix over the entire image.



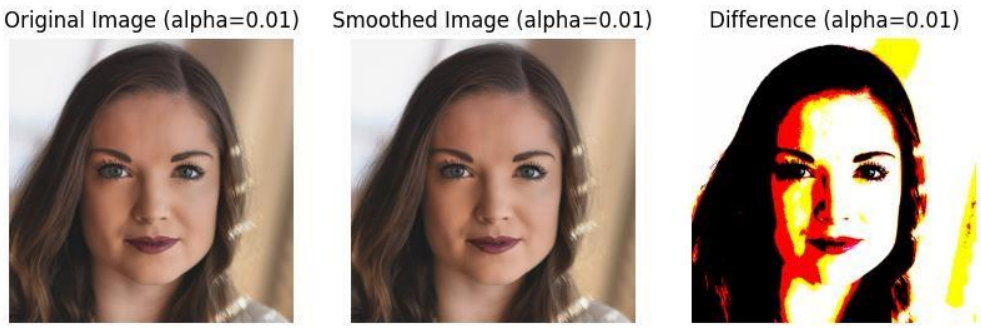
Noise Reduction:

Smoothing helps to reduce random variations or noise in an image, providing a clearer and more discernible structure.

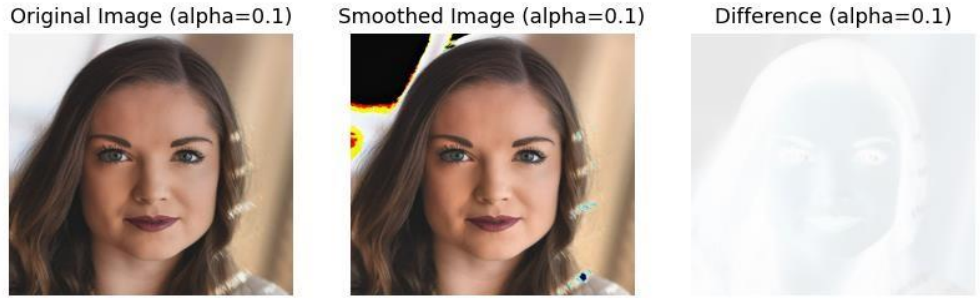
- ***Role of Dynamic Programming:***
  Dynamic programming is employed to efficiently find the optimal path for smoothing. In our case, the optimal path represents the sequence of pixels that minimizes the overall "cost" or difference between neighboring pixels. The goal is to find a smooth transition along the rows.

We use smoothing in image processing to reduce noise or variations in pixel intensities. We use a function that takes an image matrix and an alpha value as parameters. It iterates through the rows of the image, and for each pixel, it calculates a smoothed value based on its neighboring pixels and the specified alpha parameter.

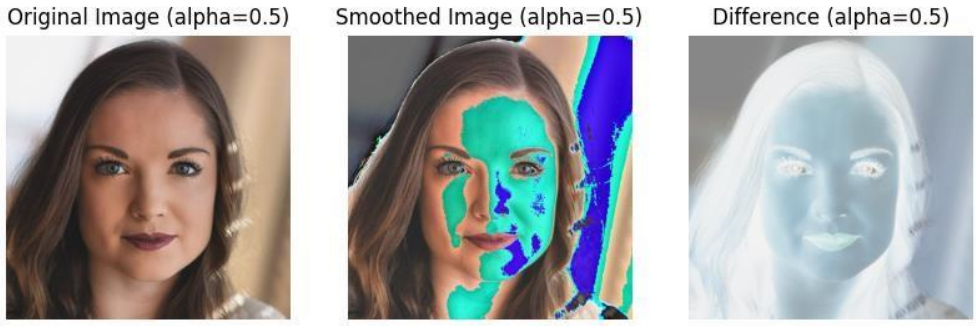
Alpha Parameter: 0.01



Alpha Parameter: 0.1



Alpha Parameter: 0.5



6. Conclusion:

We successfully explore the practical application of Matrix Chain Multiplication in image processing. We demonstrate the effectiveness of Dynamic Programming Algorithm in optimizing the reconstruction of RGB image The crux of this research contributes valuable insights into the computational efficiency of this approach.

References:

- **https://medium.com/advanced-deep-learning/decoding-image-representation-understanding-the-structure-of-rgb-images-6a211eb8800d**
- **https://asaibab.math.ncsu.edu/Module2ImageProcessing.pdf**
- **https://www.youtube.com/playlist?list=PL2zRqk16wsdoCCLpou-dGo7QQNks1Ppzo**
- **https://www.youtube.com/watch?v=Pcb7g3_WeXQ**
- **Google Scholar**