Graduate School of Engineering and Science

CS 552 – Data Science with Python

2020 FALL

Instructor : Dr. Reyhan AYDOĞAN

Assignment 2 : Image Compression by K-Means Clustering

Salih Emre KAYNAR – S011496

Introduction

This assignment demonstrates an image compression method by K-Means Clustering in Python programming language. K-Means Clustering method is one of the well-known clustering techniques and in this assignment the technique is used for image compression.

K-Means clustering method is an unsupervised learning technique which finds groups inside a dataset which has not been labeled.

The main goal of clustering is finding optimal centroid of clusters. All the data inside a cluster should be similar to the centroid and different clusters should differ between themselves.

As the compression technique, each pixel of an image is replaced with the centroid of clusters. So, the number of centroids will represent the colors and number of centroids represent the number of colors.

In this assignment different numbers of clusters will be implemented. The numbers decided are 2 to the powers of 1 to 8. (2^1 -- 2^8)

The aim of this assignment is finding the optimal number of clusters regarding to different metrics which is calculated after K-Means clustering of 5 images.

Methodology

Python 3.8.6 64-bit version is used on Jupyter IPython notebook in this assignment. The purpose of using IPython notebook is, we can benefit an interactive environment for coding.

For K-Means Clustering KMeans from Scikit-Learn library is used. Math, Numpy, BytesIO and webcolors libraries are used for calculating different metrics for compressed images. OpenCV and Pillow libraries are used for importing and saving images. Pandas library is used for constructing data frames. For the last, matplotlib library is used to show images inside IPython Notebook.

A total of 5 images that used in this assignment are given by our instructor.

Implementation Details

1. Data Analysis

First, images given by our instructor must be imported to the Jupyter Notebook with ‘imread’ function from OpenCV library.

‘imread’ function reads the image as a pixel array in Blue Green Red format. For showing the images matplotlib library is used and it plots the image in Red Green Blue format. In order to overcome this issue, `cvtColor(image, cv2.COLOR_BGR2RGB)` function from OpenCV library is used. This function converts BGR format to RGB format.

Before starting k-means clustering and compression, it’s good to check some information of original image such as size in KB, frame size, number of unique colors and how the image is imported.

All of the images are imported as numpy.ndarray and stored as in Figure[2]

```
In [4]: #Finding the type how images are stored
        a = 0
        for image in images:
            print(type(images[a]))
            a = a + 1

<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
```

Figure[1] – Storing type of Images

```
In [5]: #Having a sight how data is stored in this
        images[0][:3,:3]

Out[5]: array([[[164, 150,  71],
                [ 63,  57,  31],
                [ 75,  43,  10]],

               [[120, 125,  62],
                [135,  97,  33],
                [ 55,  35,  23]],

               [[ 99,  74,  31],
                [132, 118,  46],
                [ 60,  41,  36]]], dtype=uint8)
```

Figure[2] – A brief look atImage Array

Finding the image size is done by using BytesIO library. The image as an array is converted to image and saved as a png file. Then tell() function from BytesIO returns the file size in Bytes.

Our images have a frame size of 512x512 pixels. This number of pixels will take too much time in k-means clustering algorithm. While we have 5 images if we reduce our size to 256x256 pixels we will significantly decrease our computation speed of k-means clusterings.

To calculate unique colors, each image is reshaped and turned from a 3D array into a matrix. This is done by reshape() function from numpy. So, number of unique colors are calculated by counting unique RGB pixel lists inside reshaped image matrix.

2. k-means Clustering

Now the image arrays are ready for clustering. The image arrays will go through a loop which performs k-means clustering with cluster numbers ranging [2^1 2^2 2^3 2^4 2^5 2^6 2^7 2^8]. Each clustering is performed with k-means++ initialization method, 10 runs will be performed and best will be selected as output and number of iterations are 300 for a single run. All performed clustering is appended inside a list called kmeans_arr, for not calling k-means again and again. K-Means clustering has a long computation time. This loop is performed for all 5 images

After constructing a list of k-means, each setup with different number of clusters, each pixel value is replaced with the closest cluster center to it. In order to perform this operation pixel_as_centroids() function is defined. This function takes kmeans, output type and reshape option as input arguments. If the user wants an output for showing the image, this function removes the decimals from each centroid (color) rgbs, reassigns each pixel value as the closest center to it and reshapes the image in back to 3D array. If the user wants an output for calculating metrics, this function only replaces pixels with its closest centroids. Each k-means with different clusters are appended to 2 lists for these two output types after pixel_as_centroids() function.

Image Compression is kind of finished for now.

3. Metrics Calculation

In order to compare the images with different number of clusters and evaluate our design we have different techniques for that. In this assignment we are using 3 types of metrics. These metrics are : WCSS: Within Cluster Sum of Squares, BCSS: Between Cluster Sum of Squares and Explained Variance (Silhouette Coefficients).

WCSS is the sum of squared difference between the points and the corresponding cluster centroids.

BCSS is the sum of squared difference between the centroids and the total sample mean multiplied by the number of points within each cluster.

Explained Variance is how compressed image can explain the variance of the original image. It is calculated by BCSS divided by sum of WCSS and BCSS.

4. Choosing Optimal Number of Colors

Hence the metrics calculations are done for each number of cluster (color), we can use these metrics for choosing an optimal number for colors by comparing these metrics' elbow points.

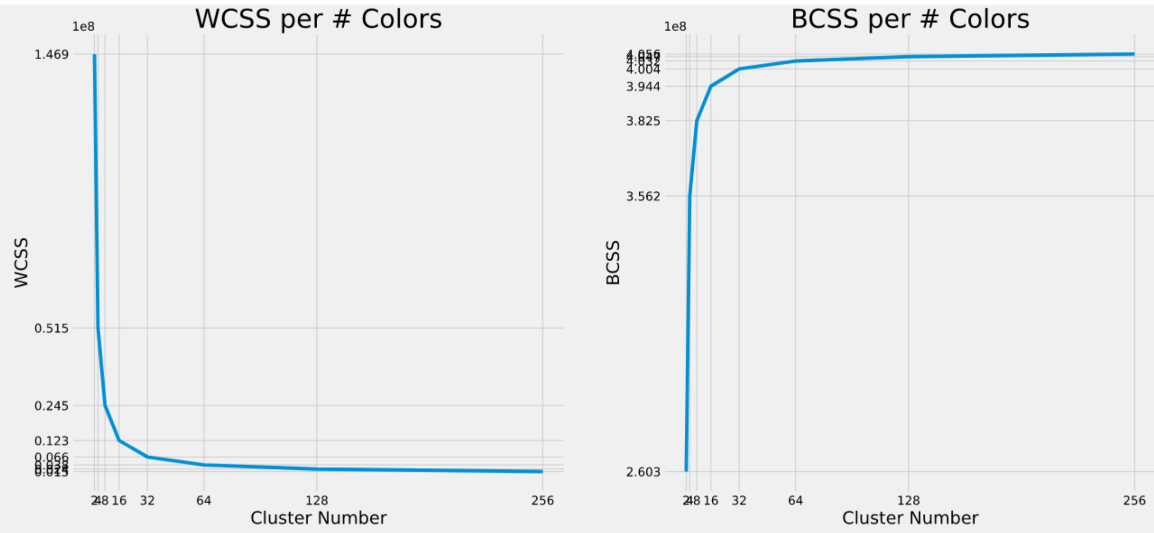


Figure [3] – WCSS and BCSS values for each number of colors of baboon image

Elbow method is used for determining for an optimal number of clusters by comparing the metrics explained in the part above and the number of clusters. Elbow point gives us the optimal number of clusters. Elbow point is calculated by drawing a straight line between the first and last element, calculating the perpendicular distance between each point and the line. The furthermost distance is the elbow point. In Figure [3] above elbow point can be determined by assuming the graph is an arm and the elbow of the arm is our elbow point.

Finally, we compare the variance and image size in order to choose the optimal image with the best possible quality(variance), but least image size. The optimal image is chosen by calculating the elbow point of variance vs. image size plotted.

4. Results

The whole algorithm described above is for compressing the images and finding the optimal number of clusters according different metrics. The Result of each image compression is shown below.

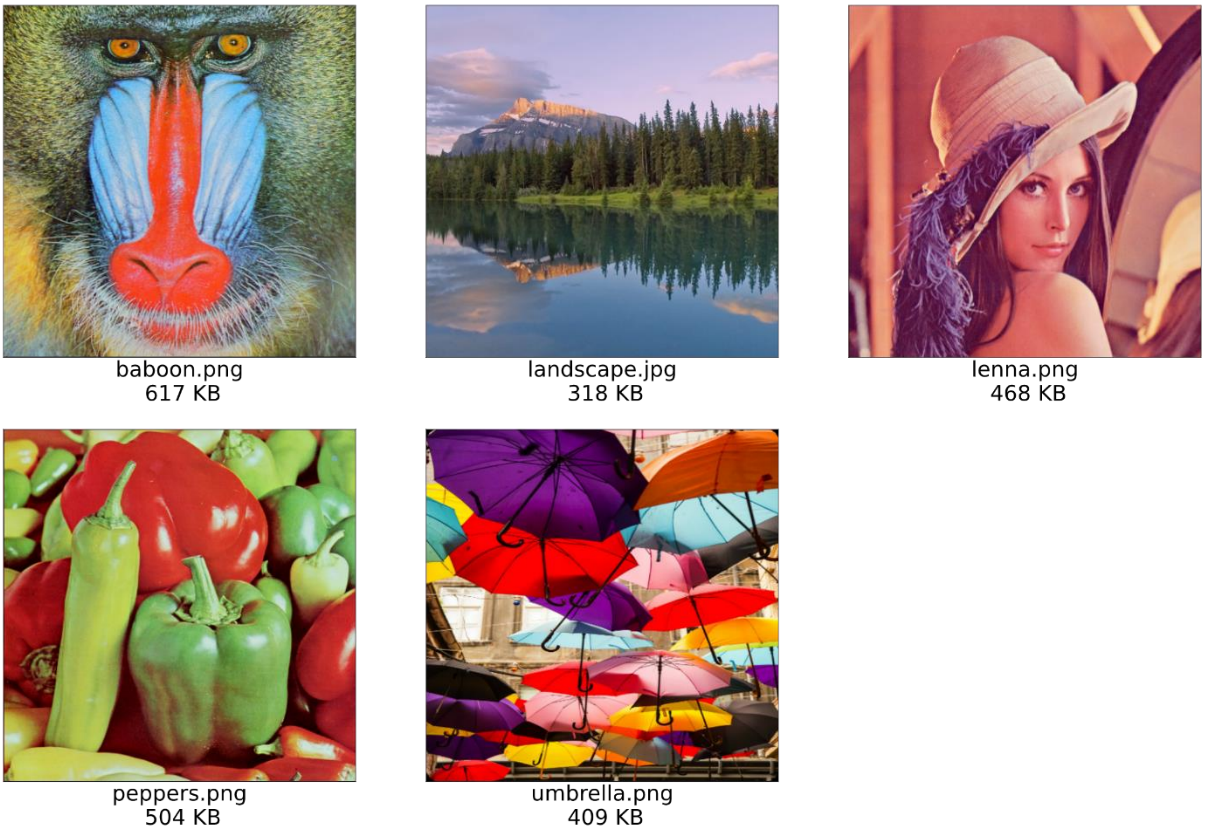


Figure [4] - Original images and their file sizes

In Figure[4] above all images are shown in their original sizes 512x512 pixels

```
Number of unique colors in image 1 is 230427
Number of unique colors in image 2 is 57600
Number of unique colors in image 3 is 148279
Number of unique colors in image 4 is 183525
Number of unique colors in image 5 is 135560
```

```
Number of unique colors in image baboon is 62070
Number of unique colors in image landscape is 33194
Number of unique colors in image lenna is 48331
Number of unique colors in image peppers is 54108
Number of unique colors in image umbrella is 45085
```

Table [1] – Number of Unique Colors 512x512 – 256x256

```
Size of image 1 is : 152.5576171875 KB
Size of image 2 is : 93.4609375 KB
Size of image 3 is : 116.361328125 KB
Size of image 4 is : 115.2275390625 KB
Size of image 5 is : 116.84375 KB
```

Table [2] – Size of each image after reducing size

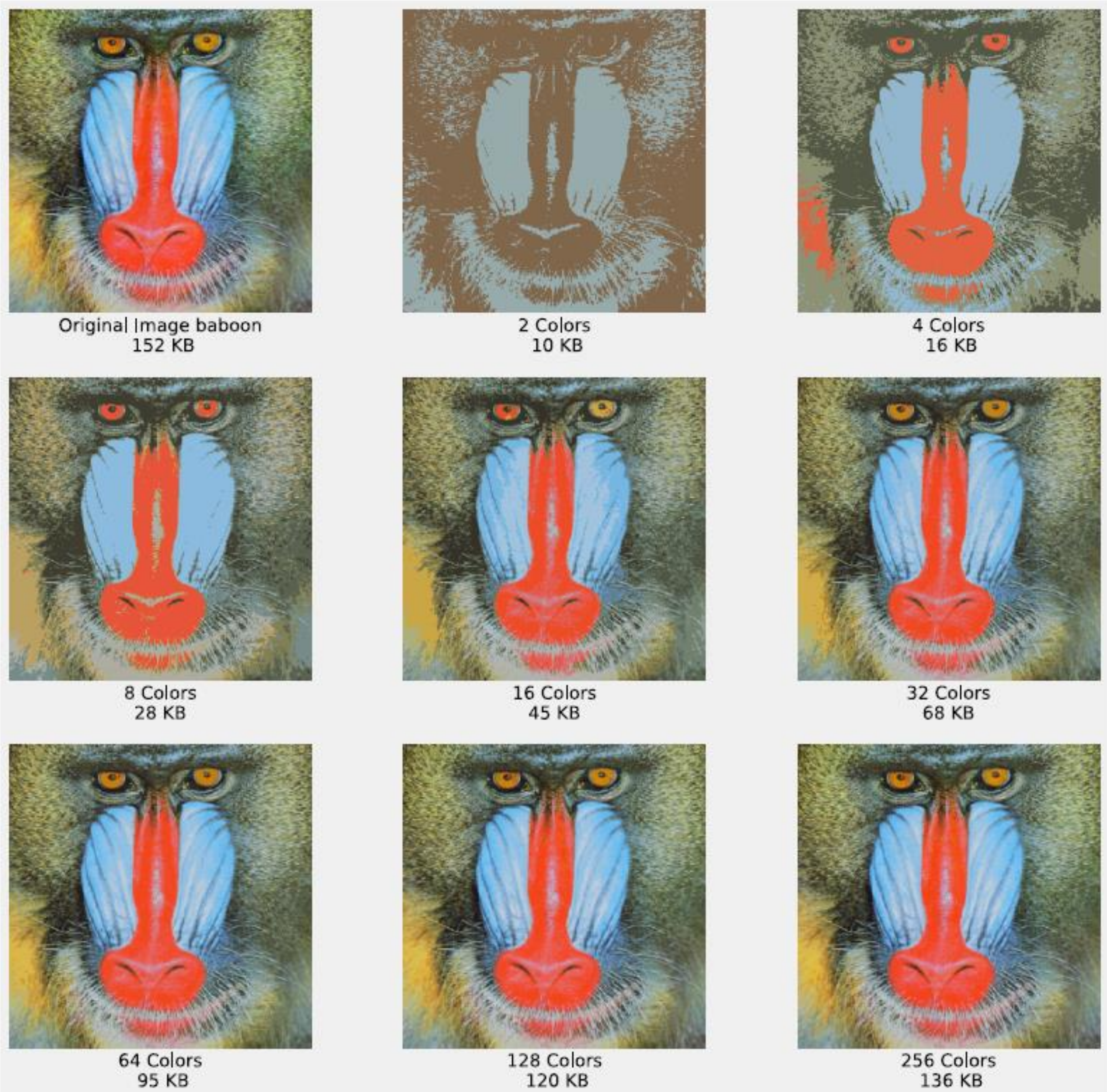
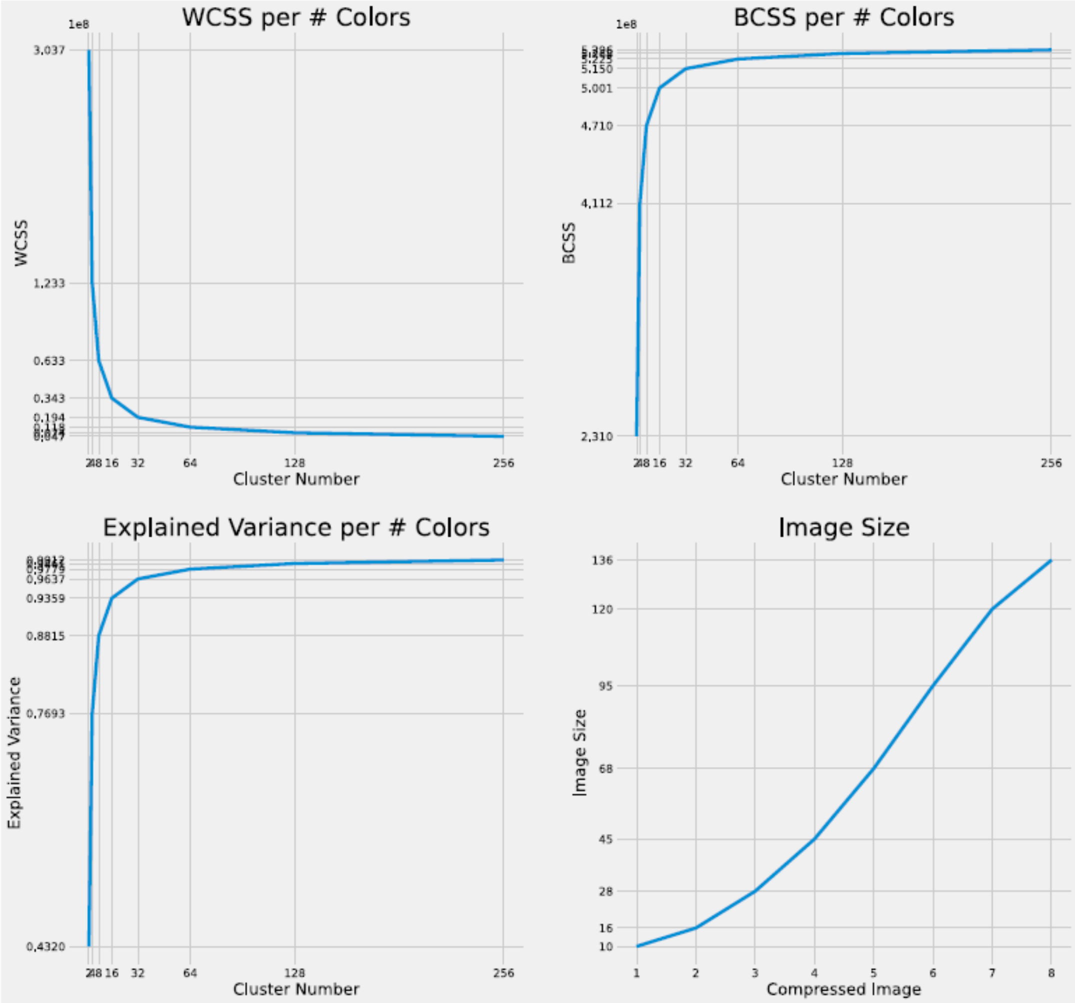


Figure [5] – Baboon image compression results

When we look at Figure[5] we can see that after 32 colored image there is not much of visible improvement in the quality of the image.

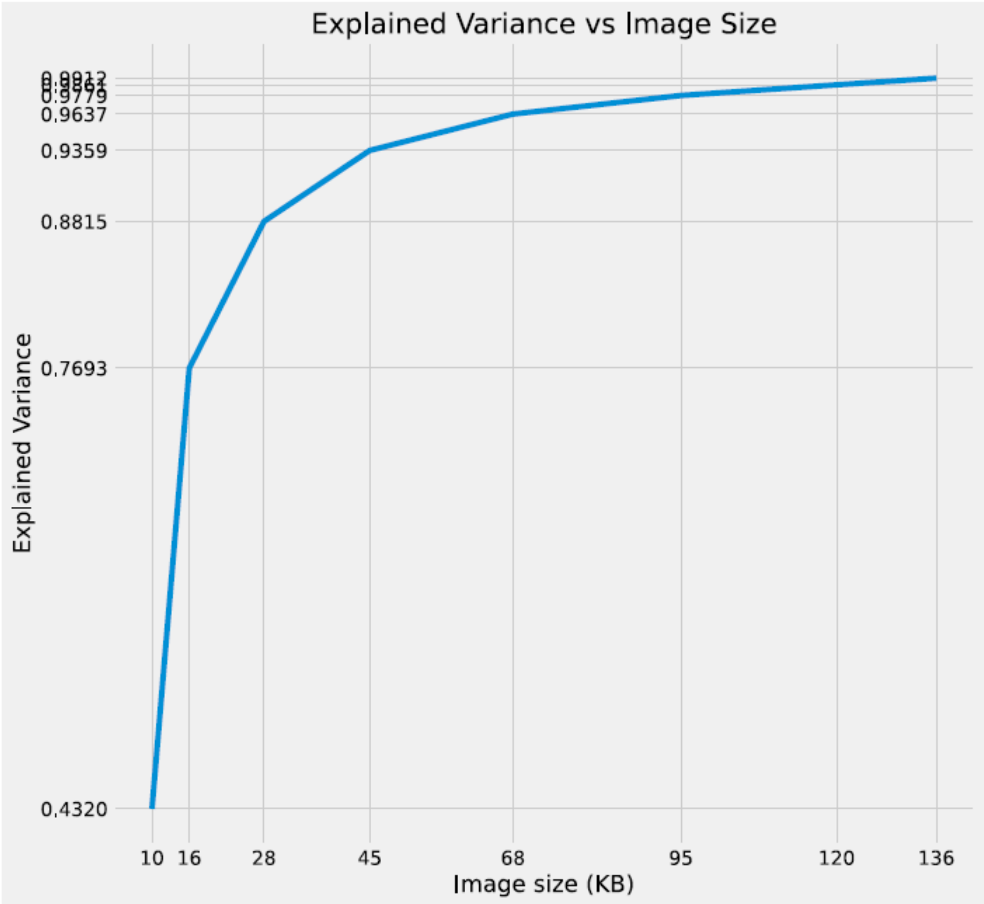


Graph[1] – Metrics Results for baboon image

```
Optimal elbow for WCCS :  16 clusters
Optimal elbow for BCSS :  16 clusters
Optimal elbow for Explained Variance :  16 clusters
Optimal elbow for Explained Variance and Image Size :  8 clusters
```

Table[3] – Optimal elbows for each metric of baboon

16 Color Image has chosen for the optimal compression for WCSS, BCSS and Explained Variance. But when we look at Figure[5] above, we can see some red spots on the right eye of baboon, which is the only obvious difference between 16 and 32 Color images. The reason that 16 Color image is chosen as elbow points, the error on right eye is not a big error in case of mathematics. But it is obvious that if you want good quality compression you should make at least 32 clusters in this image.



Graph[2] – Explained Variance vs Image Size for baboon image

	# Clusters	Used Color Names	WCSS	BCSS	Explained Variance	Image Size (KB)
0	2	[dimgray, darkgray]	3.036874e+08	2.309540e+08	0.431979	10
1	4	[lightsteelblue, darkolivegreen, gray, tomato]	1.233246e+08	4.111799e+08	0.769273	16
2	8	[dimgray, darkgray, dimgray, tomato, darkkhaki...	6.329599e+07	4.710469e+08	0.881544	28
3	16	[darkolivegreen, darkkhaki, gray, lightsteelbl...	3.426908e+07	5.000869e+08	0.935868	45
4	32	[darkgray, darkolivegreen, sienna, dimgray, sk...	1.939763e+07	5.150076e+08	0.963702	68
5	64	[dimgray, darkseagreen, tomato, darkgray, dimg...	1.179094e+07	5.225102e+08	0.977932	95
6	128	[dimgray, skyblue, darkolivegreen, tomato, dar...	7.432199e+06	5.267909e+08	0.986088	120
7	256	[chocolate, slategray, skyblue, dimgray, darkk...	4.710510e+06	5.295741e+08	0.991184	136

Table[4] – Baboon Dataframe

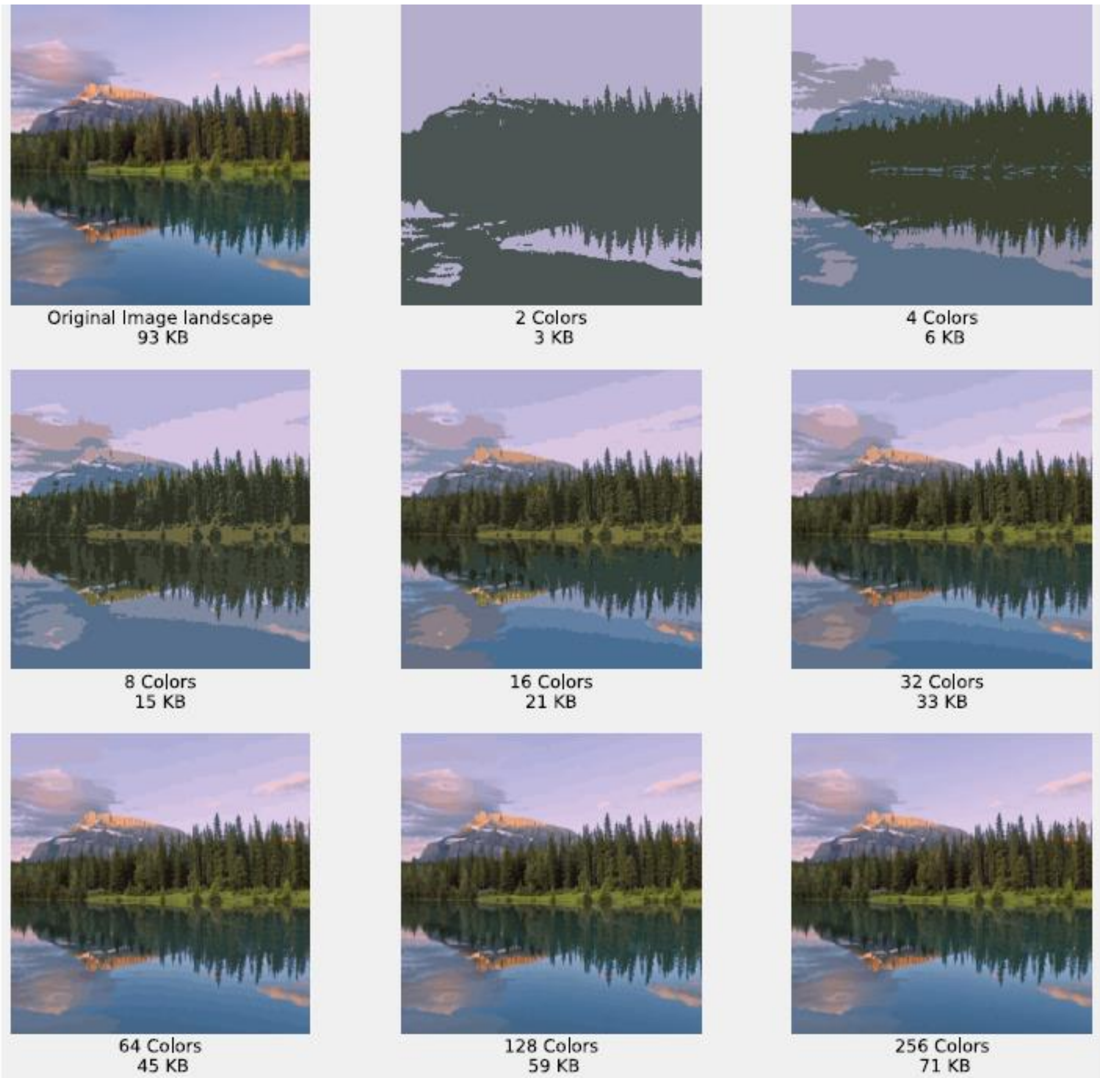
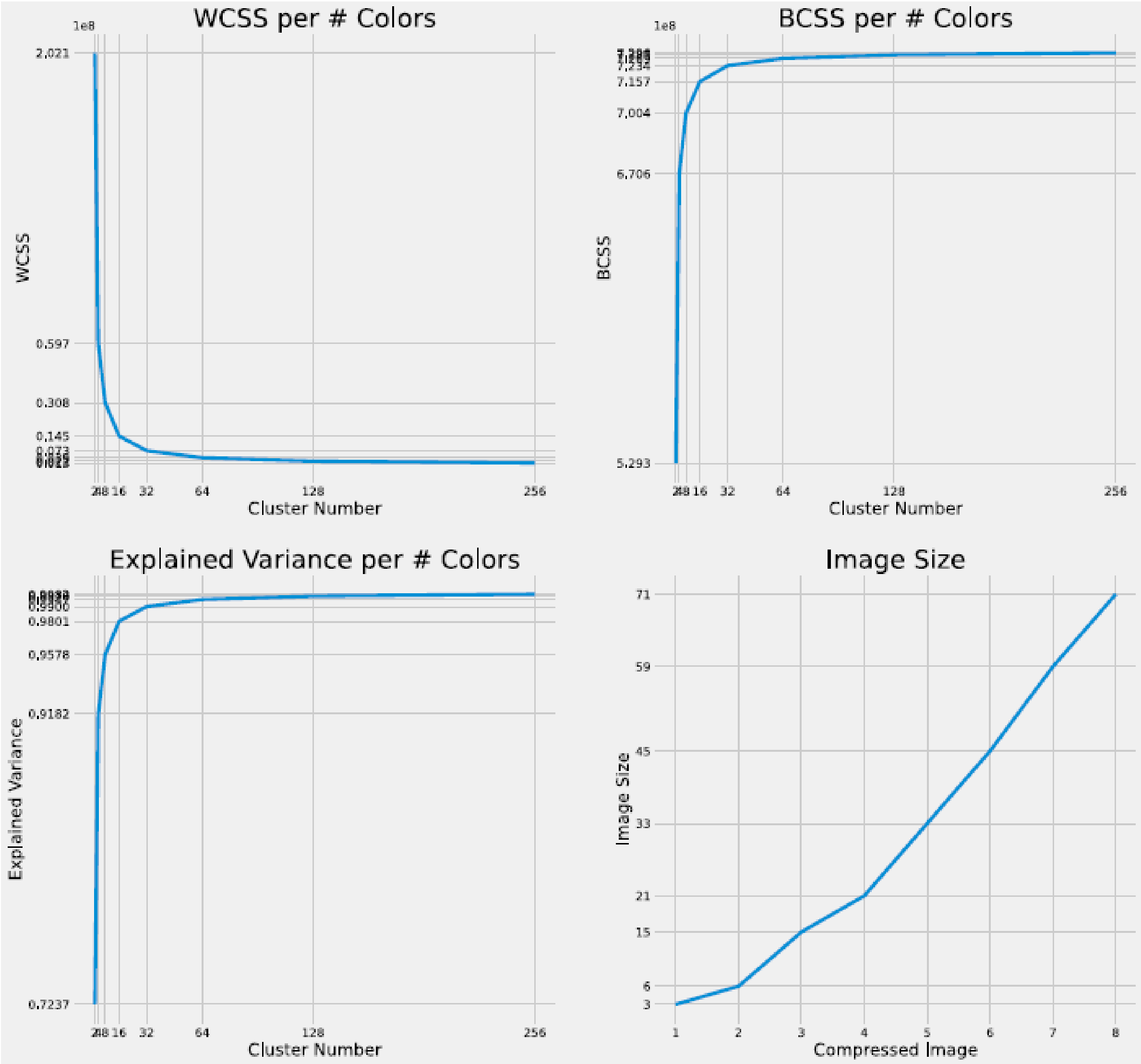


Figure [6] – Landscape image compression results



Graph[3] – Metrics Results for landscape image

```
Optimal elbow for WCCS :  16 clusters
Optimal elbow for BCSS :  16 clusters
Optimal elbow for Explained Variance :  16 clusters
Optimal elbow for Explained Variance and Image Size :  8 clusters
```

Table[6] – Optimal elbows for each metric of landscape

Graph[4] – Explained Variance vs Image Size for landscape image

	# Clusters	Used Color Names	WCSS	BCSS	Explained Variance	Image Size (KB)
0	2	[darksalmon, brown]	1.469500e+08	2.603398e+08	0.639200	5
1	4	[indianred, lightcoral, brown, burlywood]	5.153359e+07	3.562150e+08	0.873614	11
2	8	[wheat, sienna, gray, darksalmon, indigo, ligh...	2.452743e+07	3.824505e+08	0.939733	19
3	16	[indigo, indianred, burlywood, brown, rosybrow...	1.229589e+07	3.944226e+08	0.969768	28
4	32	[indigo, indianred, burlywood, darksalmon, sie...	6.604994e+06	4.004146e+08	0.983772	44
5	64	[tan, sienna, darksalmon, indianred, brown, in...	3.844411e+06	4.031926e+08	0.990555	63
6	128	[silver, sienna, darksalmon, indianred, brown,...	2.375125e+06	4.046978e+08	0.994165	81
7	256	[indianred, brown, burlywood, brown, rosybrown...	1.497101e+06	4.055992e+08	0.996322	96

Table[7] – Landscape Dataframe

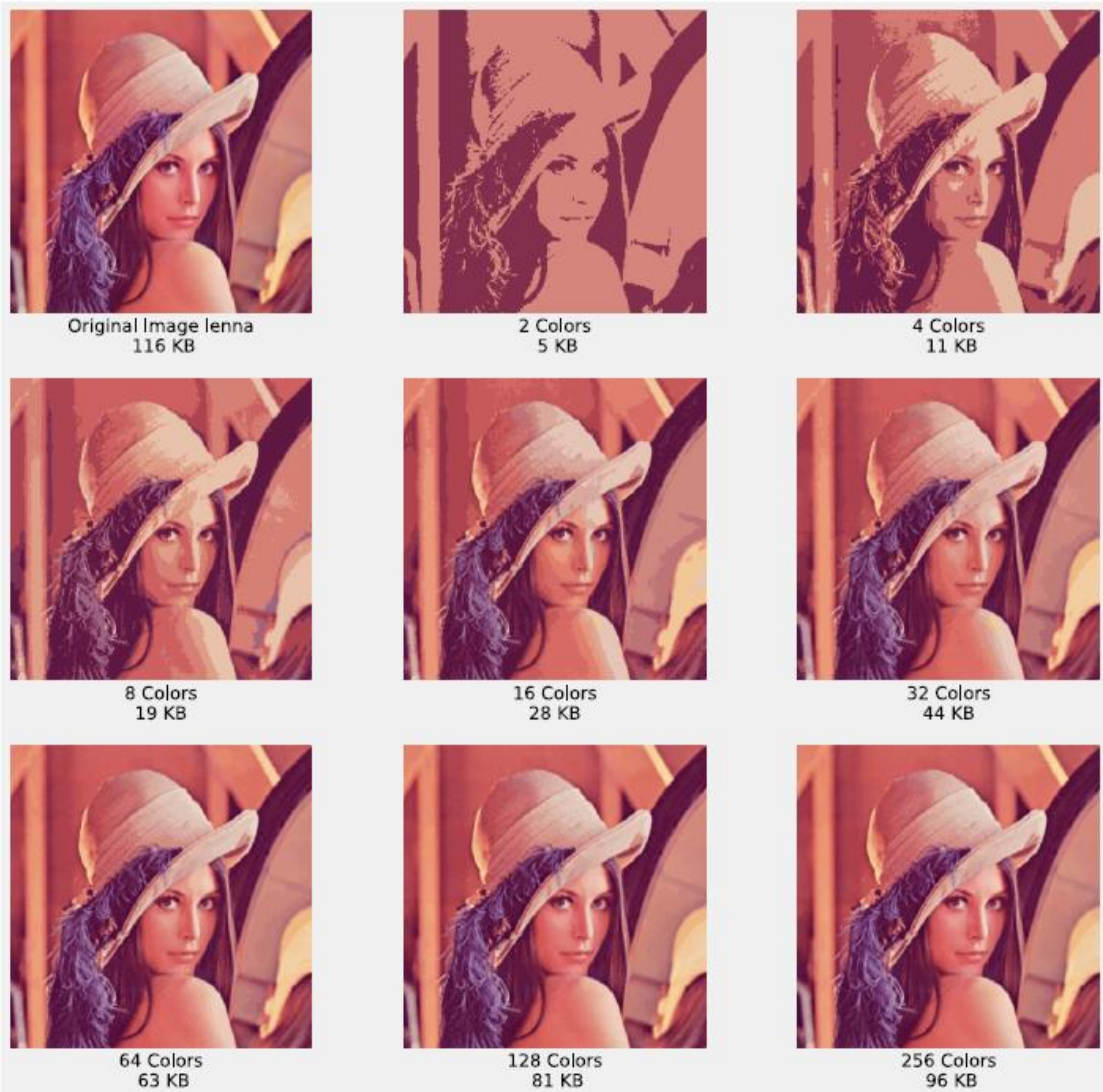
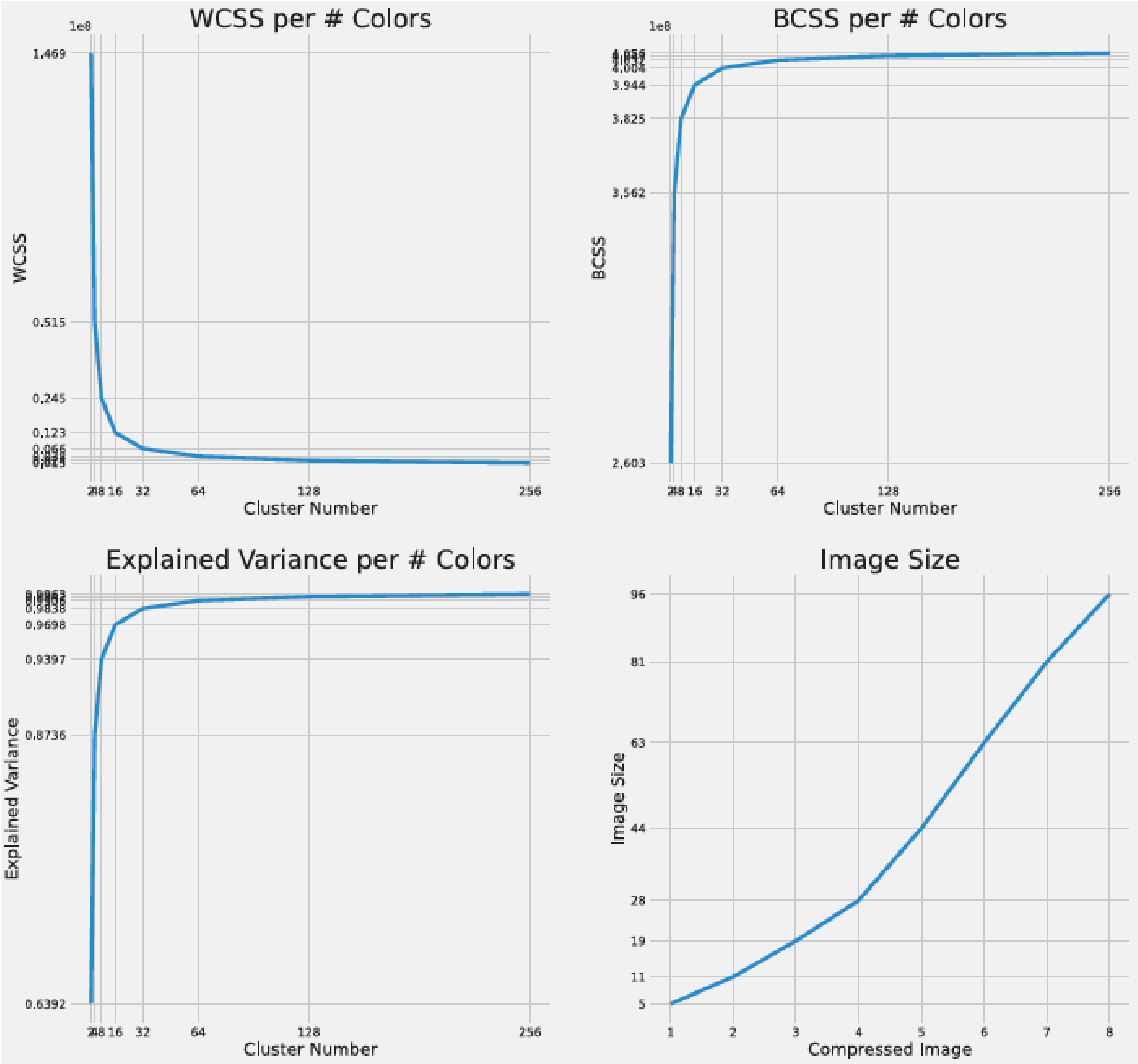


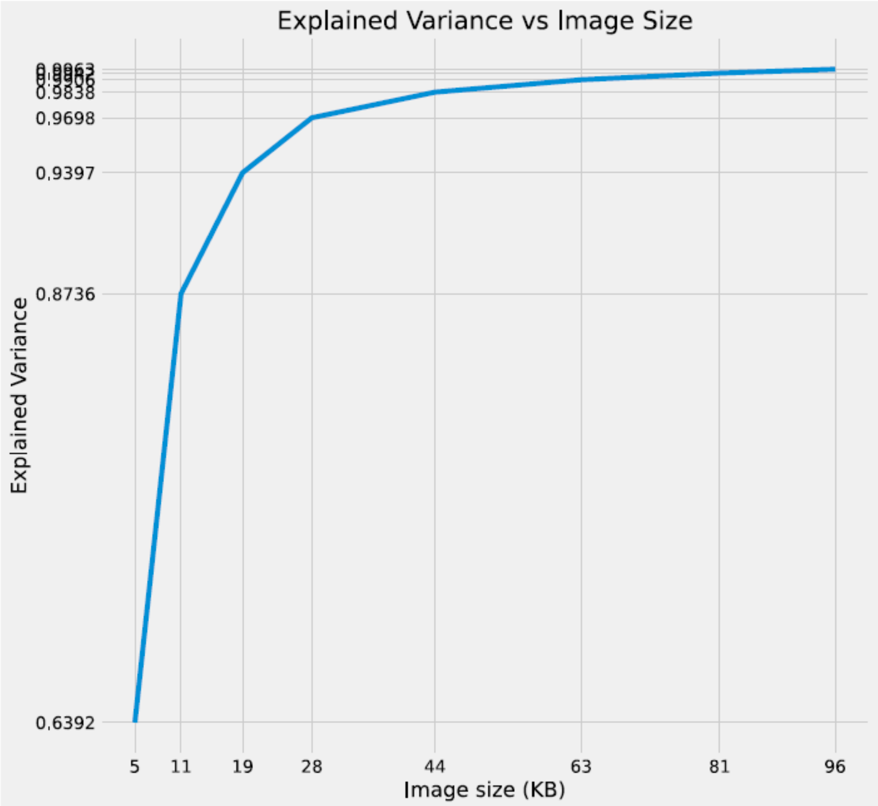
Figure [7] – Lenna image compression results



Graph[6] – Metrics Results for lenna image

```
Optimal elbow for WCCS :  16 clusters
Optimal elbow for BCSS :  16 clusters
Optimal elbow for Explained Variance :  16 clusters
Optimal elbow for Explained Variance and Image Size :  8 clusters
```

Table[8] – Optimal elbows for each metric of lenna



Graph[7] – Explained Variance vs Image Size for lenna image

	# Clusters	Used Color Names	WCSS	BCSS	Explained Variance	Image Size (KB)
0	2	[darksalmon, brown]	1.469500e+08	2.603398e+08	0.639200	5
1	4	[indianred, lightcoral, brown, burlywood]	5.153359e+07	3.562150e+08	0.873614	11
2	8	[wheat, sienna, gray, darksalmon, indigo, ligh...	2.452743e+07	3.824505e+08	0.939733	19
3	16	[indigo, indianred, burlywood, brown, rosybrow...	1.229589e+07	3.944226e+08	0.969768	28
4	32	[indigo, indianred, burlywood, darksalmon, sie...	6.604994e+06	4.004146e+08	0.983772	44
5	64	[tan, sienna, darksalmon, indianred, brown, in...	3.844411e+06	4.031926e+08	0.990555	63
6	128	[silver, sienna, darksalmon, indianred, brown,...	2.375125e+06	4.046978e+08	0.994165	81
7	256	[indianred, brown, burlywood, brown, rosybrown...	1.497101e+06	4.055992e+08	0.996322	96

Table[9] – Lenna Dataframe

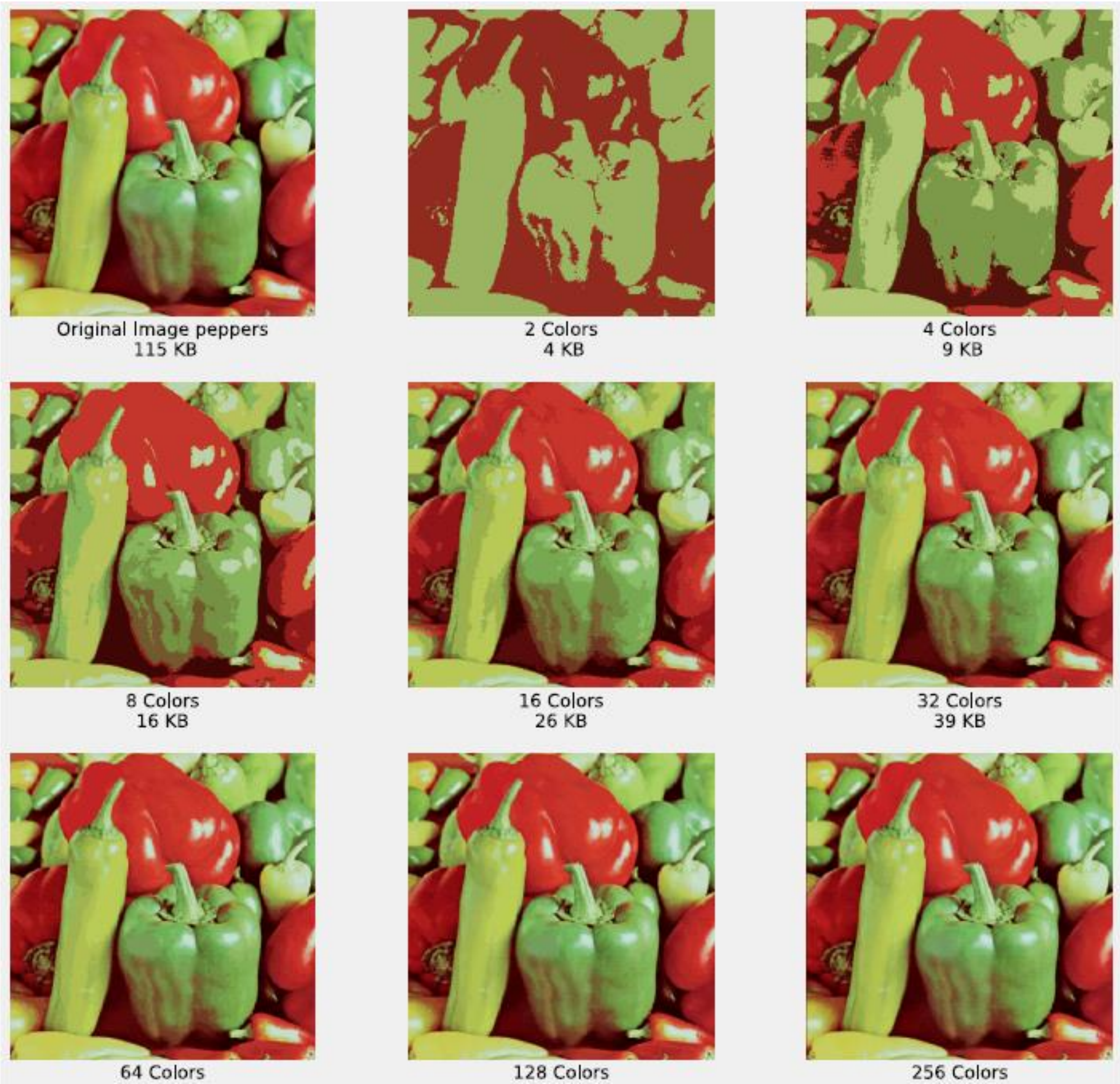
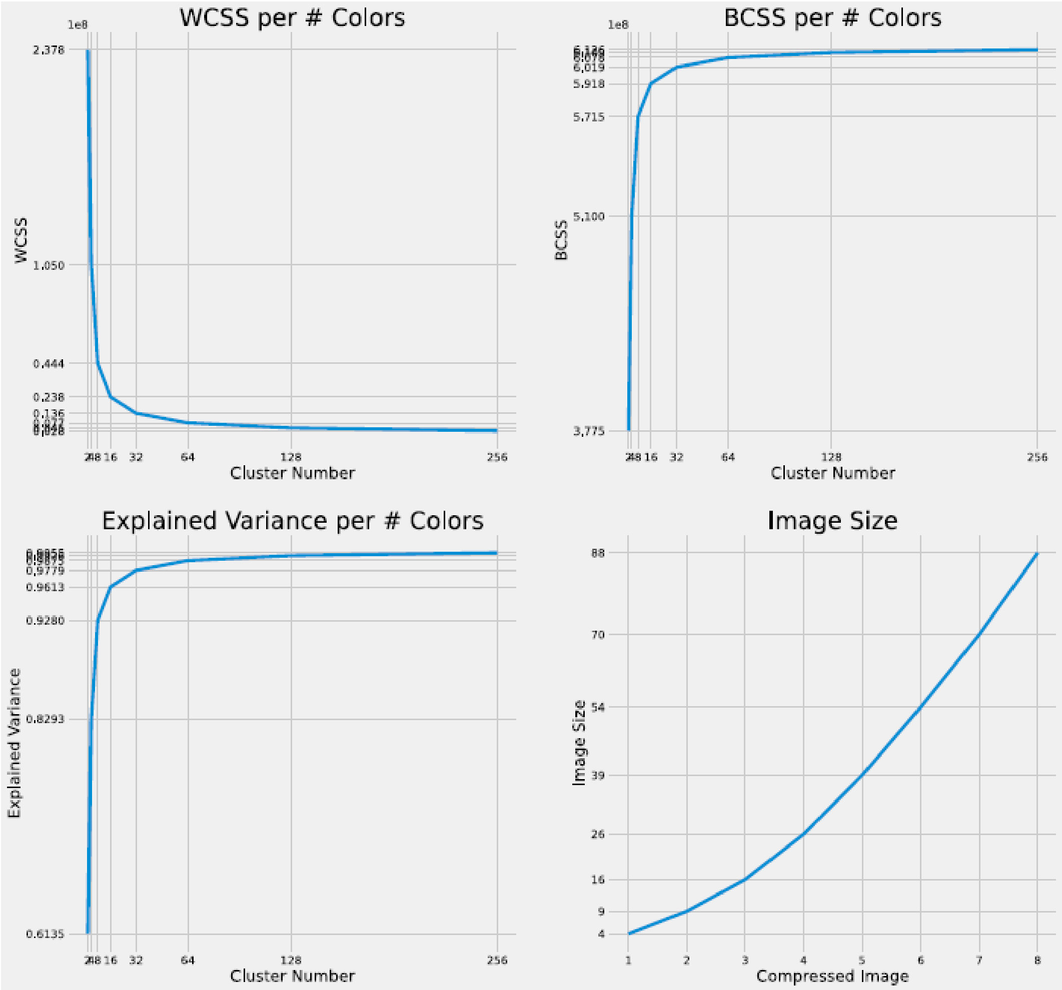


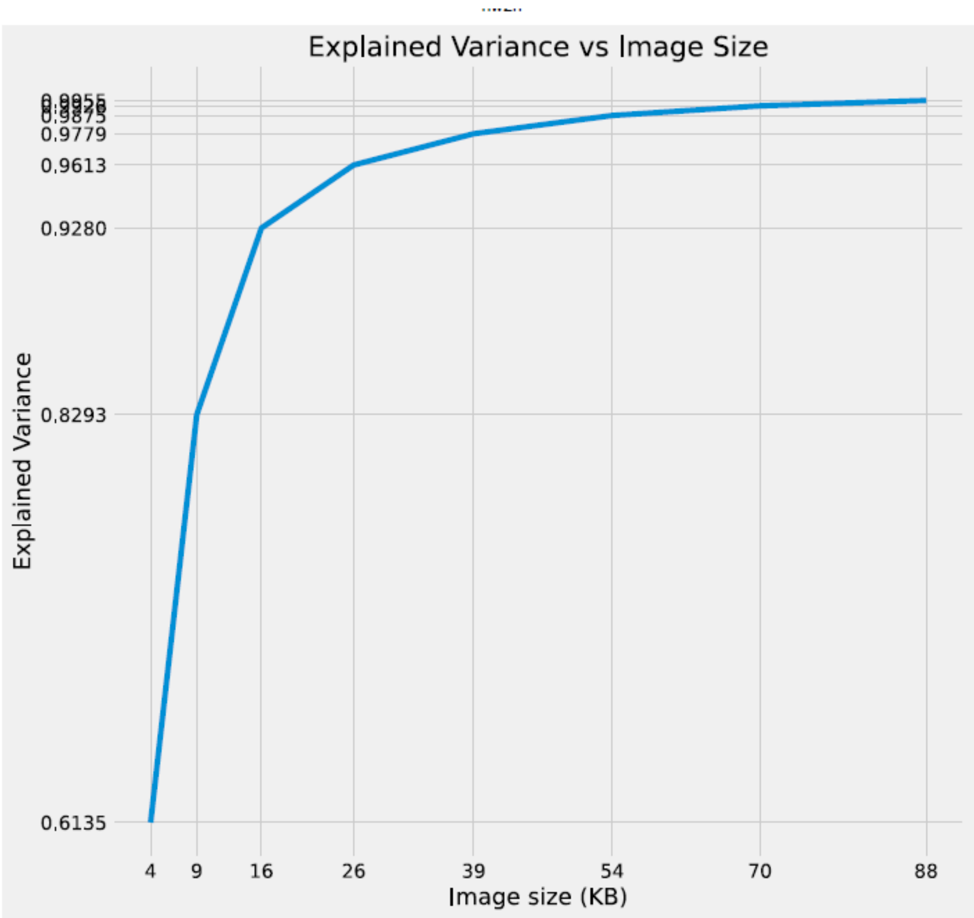
Figure [8] – Peppers image compression results



Graph[8] – Metrics Results for peppers image

```
Optimal elbow for WCCS :  16 clusters
Optimal elbow for BCSS :  16 clusters
Optimal elbow for Explained Variance :  16 clusters
Optimal elbow for Explained Variance and Image Size :  8 clusters
```

Table[8] – Optimal elbows for each metric of peppers



Graph[9] – Explained Variance vs Image Size for peppers image

91]:

	# Clusters	Used Color Names	WCSS	BCSS	Explained Variance	Image Size (KB)
0	2	[darkkhaki, brown]	2.378420e+08	3.774879e+08	0.613472	4
1	4	[firebrick, darkkhaki, olivedrab, maroon]	1.049961e+08	5.099873e+08	0.829270	9
2	8	[yellowgreen, darkolivegreen, black, darkkhaki...	4.435448e+07	5.714637e+08	0.927975	16
3	16	[olivedrab, firebrick, darkseagreen, maroon, s...	2.380090e+07	5.917762e+08	0.961336	26
4	32	[firebrick, olivedrab, saddlebrown, darkkhaki,...	1.361877e+07	6.018952e+08	0.977874	39
5	64	[firebrick, mediumseagreen, maroon, darkkhaki,...	7.674435e+06	6.077740e+08	0.987530	54
6	128	[darkseagreen, brown, maroon, tan, olivedrab, ...	4.534550e+06	6.108662e+08	0.992632	70
7	256	[firebrick, mediumseagreen, maroon, darkkhaki,...	2.785762e+06	6.126303e+08	0.995473	88

Table[9] – Peppers Dataframe

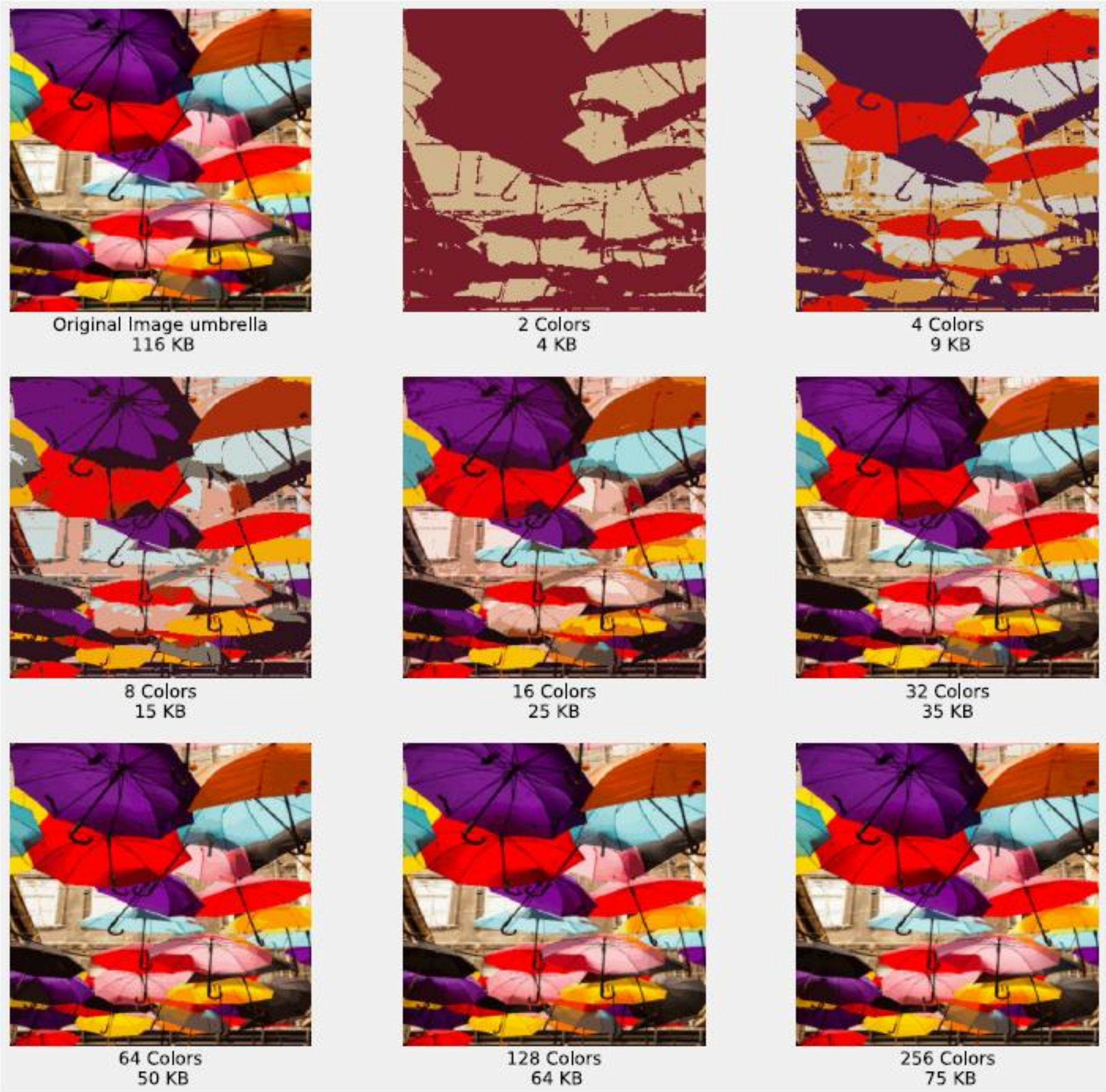
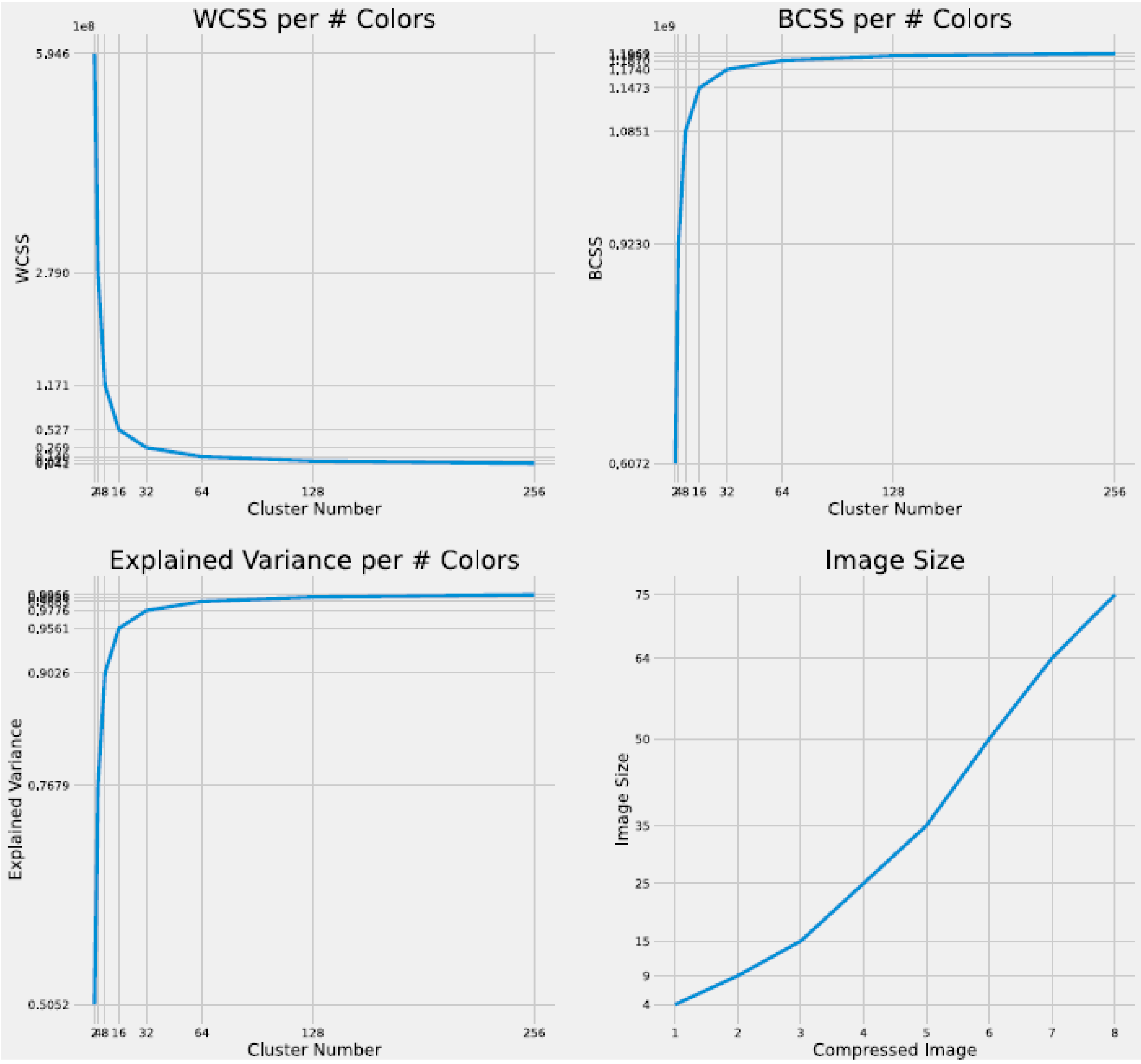


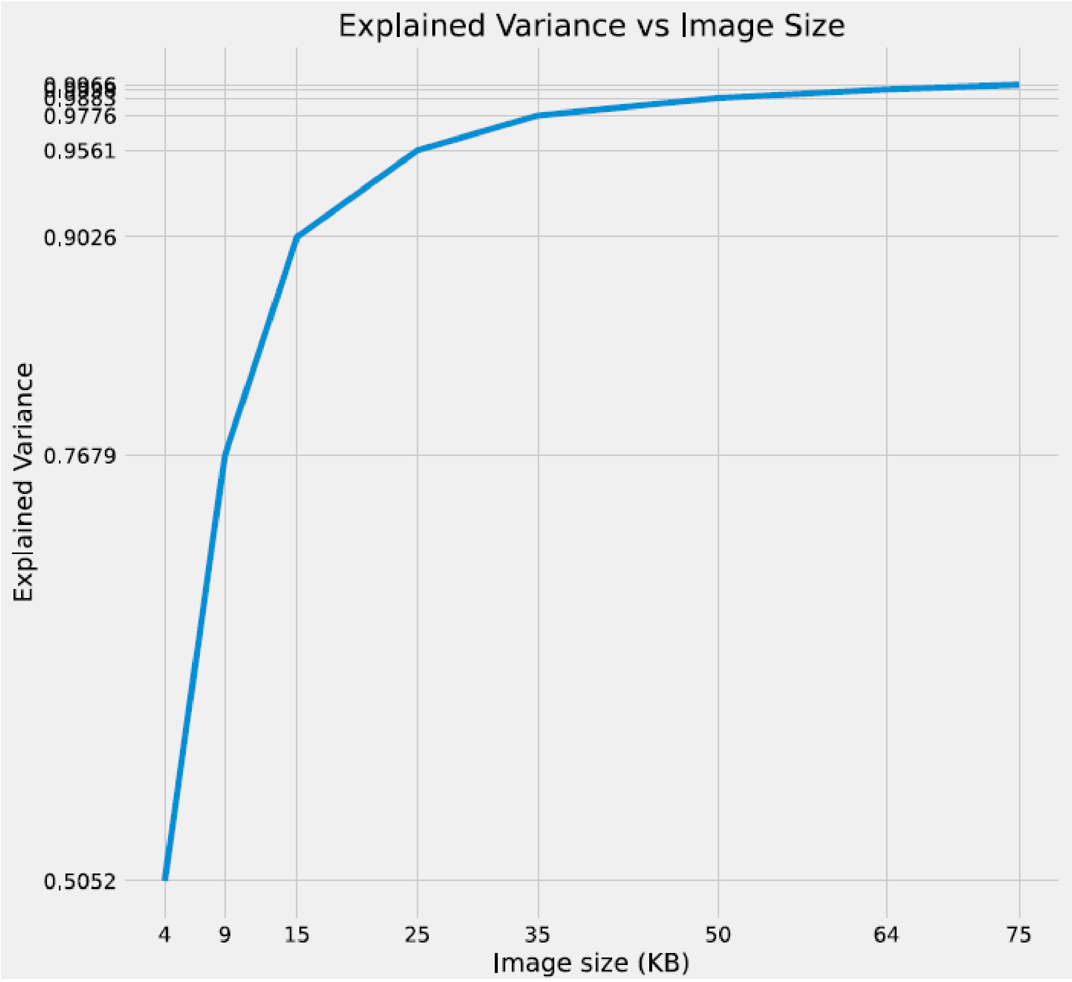
Figure [9] – Umbrella image compression results



Graph[10] – Metrics Results for umbrella image

```
Optimal elbow for WCCS :  16 clusters
Optimal elbow for BCSS :  16 clusters
Optimal elbow for Explained Variance :  16 clusters
Optimal elbow for Explained Variance and Image Size :  8 clusters
```

Table[10] – Optimal elbows for each metric of umbrella



Graph[11] – Explained Variance vs Image Size for peppers image

93]:

	# Clusters	Used Color Names	WCSS	BCSS	Explained Variance	Image Size (KB)
0	2	[tan, brown]	5.946029e+08	6.072108e+08	0.505245	4
1	4	[darkslategray, peru, red, silver]	2.789862e+08	9.230125e+08	0.767898	9
2	8	[purple, orange, red, lightgray, tan, black, f...	1.171431e+08	1.085065e+09	0.902560	15
3	16	[black, burlywood, darkred, indigo, lightblue,...	5.273193e+07	1.147297e+09	0.956058	25
4	32	[linen, saddlebrown, palevioletred, indigo, bl...	2.692456e+07	1.174032e+09	0.977581	35
5	64	[lightblue, brown, black, tan, red, orange, di...	1.400055e+07	1.186963e+09	0.988342	50
6	128	[black, lightsalmon, red, purple, lightblue, g...	7.498503e+06	1.193659e+09	0.993757	64
7	256	[black, lightsalmon, red, indianred, powderblu...	4.136542e+06	1.196906e+09	0.996556	75

Table[11] – Umbrella Dataframe

Conclusion

In this assignment, 5 different images are compressed in 8 different number of clusters ranging 2 to the powers of 1 to 8. When we look at the results, the optimal number of clusters for all 3 metrics are 16 while the optimal number for Explained Variance vs Image Size is 8 for every image. Finding elbow points is very closely related min and max number of clusters. For example, if we have increased the number of clusters to 2^9 and 2^{10} the number in the middle increases which gives us a result. We also should not expect 2^2, 2^7 or 2^6 number of clusters to become elbow points in our case because they are very close to the start and ending points. In conclusion for elbow points, they are giving an optimal point depending on min max number of clusters. It's not an optimal number for a compression of an image because as we can see it's not depending on image.

Before concluding this report, I would like to compare optimal results respect to my eye and the calculations.

Original Image baboon
152 KB

16 Colors
45 KB

8 Colors
28 KB

32 Colors
68 KB

Metrics Optimal – Variance vs Size Optimal

My optimal is 32 Colors because in 16 Colors the eye colors have some obvious errors while in 32 Colors it looks very similar to original image and the size is halved

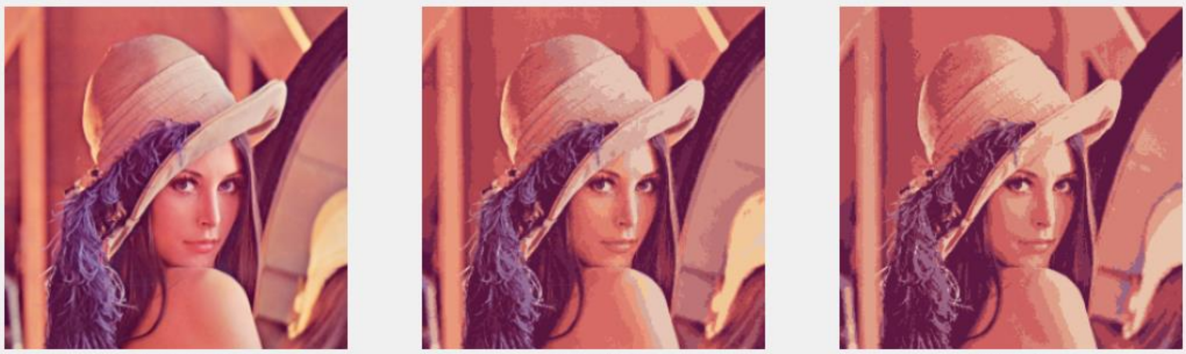
Metrics Optimal – Variance vs Size Optimal

Original Image landscape 93 KB | 16 Colors 21 KB | 8 Colors 15 KB

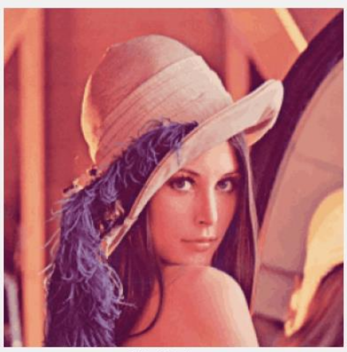
256 Colors 71 KB

My optimal is 256 Colors beacuse there are no shadows and the image size is less than 3 times size of 16 Colors while having 16 times more colors.

Metrics Optimal – Variance vs Size Optimal



Original Image lenna 116 KB | 16 Colors 28 KB | 8 Colors 19 KB



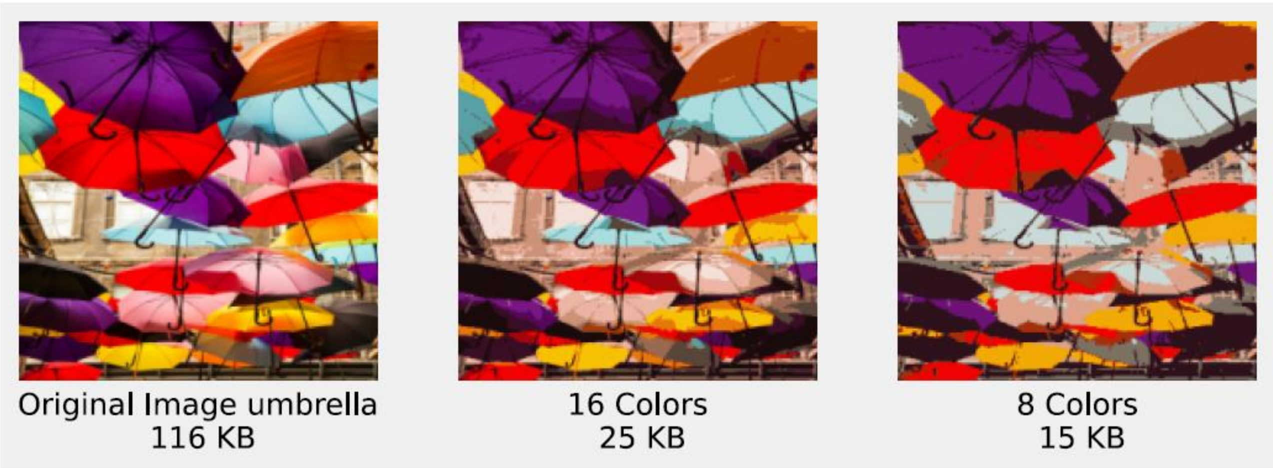
64 Colors 63 KB

My optimal is 64 colors because there are no obvious shadows

Metrics Optimal – Variance vs Size Optimal

My optimal is 16 Colors because shadows are not so obvious in this size

Metrics Optimal – Variance vs Size Optimal



My optimal is 16 Colors because shadows are not so obvious in this size and the size is reduced more than 5 times.

My optimal are depends on my eye and the frame sizes of each image. If you want more quality and in bigger size, the number of colors should go higher, or you'll have obvious shadows.