INSTITUE OF SCIENCE AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

CS552 – DATA SCIENCE WITH PYTHON
Assignment 2

2020-2021 Fall

Assignment Report

Image Compression by K-Means Clustering

Güney Doruk Keser
S004575

1. Assignment Description

In this assignment, I implemented K-Means clustering by using python that solves the problem of image compression by reducing the color range of an image.

Image Compression

Image compression is a type of data compression that represents original image with fewer bits. The aim of image compression is reducing the redundancy on the image and store it more efficiently in terms of size of image in bytes.

There are two types of image compression which are lossless and lossy image compression. Lossless image compression is a method for reducing the size of an image while maintaining the same quality as before. Lossy image compression reduces size of image by discarding some parts/pixels of an image or reducing the variety of the pixels' features.

K-Means Clustering

K-means is one of the clustering techniques which belongs to unsupervised learning that finds and groups of similar samples and assign them into the specified number of clusters by looking at their Euclidian distances in plane.

K-means clustering performance is heavily depend on the correct selection of cluster count and first initialization of their centers. The algorithm's result/performance changes with the choice of these values.

The steps of K-means clustering:

1) Initialization of "k" value and cluster centroids
2) Assign corresponding data to each cluster center by minimizing Euclidian distance
3) After assignment of all data, determine new cluster centers
4) Repeat this process until new assigned centroids are not changing anymore

The quality of cluster assignments determined by computing sum of the squared error after the centroids converge.

2. Methodology and Objectives

In this assignment, I used python language to implement image compression by K-means clustering. Python enables us different kind of libraries to help us to achieve our goals. In my problem I used NumPy, Pandas, Scikit-learn libraries for scientific computations, Matplotlib, Seaborn for visualization, OpenCV and Pillow for Image Processing, webcolors for finding color names and io for calculating image file size. For my code organization, I used Jupyter Notebook for python to divide my code to blocks that enables me the analyze and explain my code better.

My objective in this assignment is implement K-means clustering in five different images. By implementing K-means, I reduced the number of colors on the image related the number of clusters' decision in K-means algorithm so that I compressed the images. For each image, I created an eight different compressed images which have 2^i colors for i in range(1,9). For each setup, I changed pixel's RGB values with it's centroid value so that I have compressed image with less representing colors(aim of the homework), then evaluate each compressed image by looking at their WCSS,BCSS, Explained variance and compressed image size. I used elbow method for choosing the best compressed image.

3. Implementation Details

I start my implementation of image compression by K-means clustering by importing necessary libraries.Then I need to read the images. In order to read the images, I used OpenCV library by using "imread" method. OpenCV reads the images in BGR form so I turn them to RGB by using "cvtColor" method. I started my implementation with

“Baboon.png” image and after reading image I showed baboon image by using “plt.imshow” method
Note: OpenCV imread method reads images in nparray format which is good, we do not need to convert them into nparray.

After showing the baboon image I looked at it’s shape which is (512x512x3, 3 means it has R,G and B values). For the speed of K-means algorithm(since we have 5 images and we do k-means 8 times for each image and also in kmeans initialization we define n_init and n_iter values for better results) I reduced image size to 256x256x3 by using “cv.resize” method.

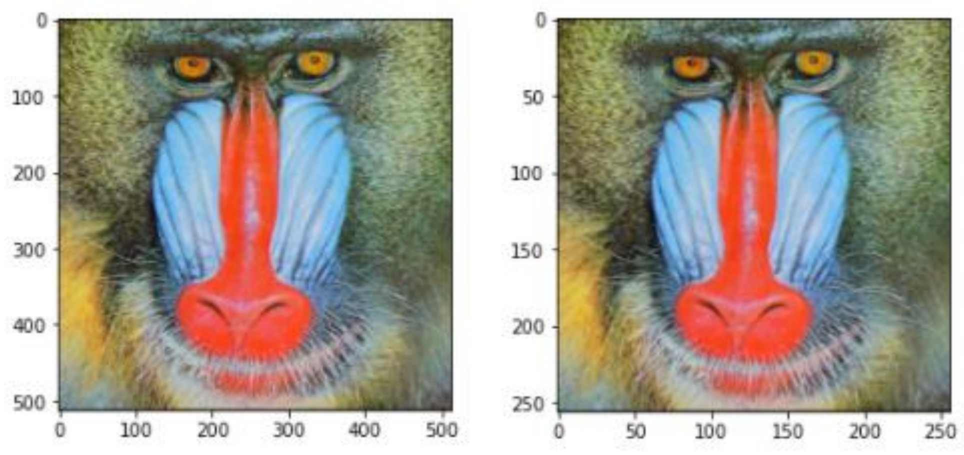


Figure 3.0: Original size and resized versions of baboon image

Then I find how many bytes(filesize) that resized original image consist of. I used BytesIO from io and pillow to save image. After that I need to fit the image in form of 2-d array(size: NX3) where N equals to “height x width” in order to find cluster centers of the pixel’s RGB values of the image. In order to reshape the image to “Nx3” I used “.reshape” method with the inputs of height x width and 3. Then, I need to find the number of unique colors of the original resized image to check the colors that representing my original image and evaluate with my k-means clusters count.

Note: These values can be seen in the results section together with it’s compressed versions.

In the homework description, we need to calculate the metrics (WCSS, BCSS, Explained Variance) and optimal elbow without using any library. Also, for some statements in the homework description, we need to write our own codes or functions to satisfy the statements. Therefore, I defined functions for calculating the required statements.

I started with defining a function that replaces pixels with their centroids. The aim of this function is that, when we labelled our pixels with their clusters, we need to convert their values to their cluster centroids so that we compress the image by reducing its color representations. K-means by itself only shows the which pixel belongs to which label without assignment of cluster pixels we cannot compress the image.

My function takes initialization of K-means as a parameter and loops the K-means labels, check which label belongs to which cluster center and assign label’s cluster center to list by method of “append”. Then convert list to numpy array with the shape of Nx3(parameter for other functions), later change its type to “uint8” and reshape back to height x width x 3. My function has two return value, one for parameter to other function and one for representing the compressed image.

My other function is for calculating the WCSS value. Function takes two parameters which are first return value of the above function(I will call this return value as compressed_image_2d for not making confusion in other function explanations) and original image in 2-d array form. Function have nested loop on original image’s dimensions and calculate WCSS by summing the Euclidian distances of compressed_image_2d pixel values and original image’s pixel values

My function for calculating BCSS value takes two parameters as input and its parameters are same with the WCSS function. This time, I calculate the mean of the pixels in axis zero in the original image and calculate BCSS by summing the Euclidian distances of compressed_image_2d pixel values and calculated mean.

I calculated explained variance by dividing BCSS to summation of BCSS and WCSS and in return I multiplied with one hundred to represent the value in percentage.

My function for optimal elbow calculation takes two parameters as input which are y and x values in the 2-d plane. I calculated optimal elbow by taking start and end points in the cartesian coordinate system. Then I find the equation of the line that passes through these points and I take the point that has most distance to the line as optimal elbow.

In the description, we need to find the name of the colors that represents compressed images. Webcolors library is a choice for displaying the name of the color with R, G, B values in color specs. However, some RGB combination colors do not have name representation on HTML 4, CSS 2, CSS 2.1 and CSS 3 color specs and when I try rgb_to_name(rgb) function, it gives value error. Therefore, I need a function to find the closest R G and B values of requested RGB values to represent its name in one of above specs. In function, I need to calculate Euclidian distance of requested RGB and RGB name representation on specs then assign name to requested RGB which has the minimum distance.

Sometimes maybe the RGB values are representing the name in CSS3(mychoice) color spec. In that case I do not need my own function to get name of the color, I simply use “rgb_to_name()” method in webcolors to get name of the color. Therefore, I describe a function that does exception handling that first tries “rgb_to_name()” method and if it gets a “ValueError” use my function to get the color name and return that name.

After definition of required functions in my homework, I can start initializing my K-means for baboon image. First of all, I created two list that stores K-means metric results and compressed images respectively, because for each image I need to do eight different K-means that have different number or clusters. After that, I created a loop in range 1 to 8 and initialize my K-means. For the initialization of the K-means, I assigned “n_clusters = 2^range value”, “random_state = 100” and default values for the other parameters. In default parameters “n_init = 10” which is the value of how many different center initialization occurs in K-means, “max_iter = 300” which is the value of how many times the iteration occurs to find convergence and “init = “kmeans++” which is the smart way of selecting initial cluster centers to speed up to convergence.

After initialization of the K-means, I did the following in an order:

1- I fit my 2-d form image to K-means and assign the pixel values to the closest cluster center values
2- I evaluated WCSS, BCSS, explained variance with my functions
3- I created a list that stores color names and assign the names to that list
4- I calculated compressed image sizes
5- I created a dictionary for storing each calculated value in above steps (Dataframe creation)
6- I added the dictionary to my list that stores metric results
7- I added the compressed images to my other list that stores compressed images

Note: Except creation of lists that for metric results and compressed images, all of other steps and K-means initialization done in the loop.

After K-means step, I created a table by using Pandas DatFrame that shows the metrics results.

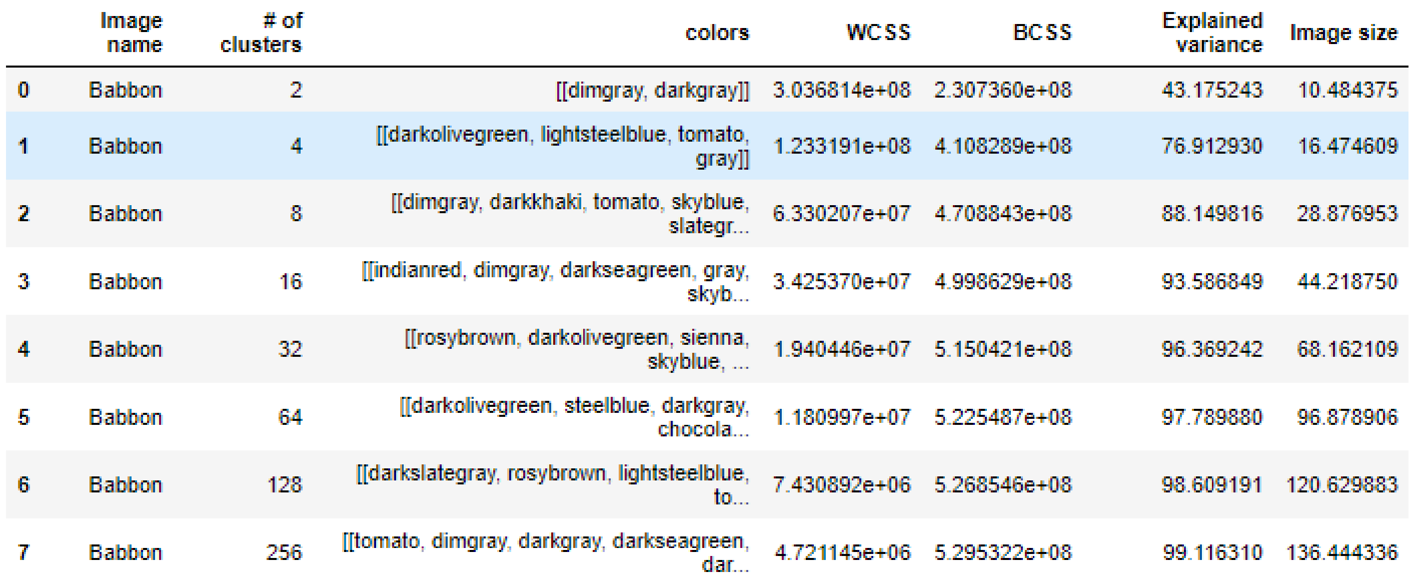


	Image name	# of clusters	colors	WCSS	BCSS	Explained variance	Image size
0	Babbon	2	[[dimgray, darkgray]]	3.036814e+08	2.307360e+08	43.175243	10.484375
1	Babbon	4	[[darkolivegreen, lightsteelblue, tomato, gray]]	1.233191e+08	4.108289e+08	76.912930	16.474609
2	Babbon	8	[[dimgray, darkkhaki, tomato, skyblue, slategr...	6.330207e+07	4.708843e+08	88.149816	28.876953
3	Babbon	16	[[indianred, dimgray, darkseagreen, gray, skyb...	3.425370e+07	4.998629e+08	93.586849	44.218750
4	Babbon	32	[[rosybrown, darkolivegreen, sienna, skyblue, ...	1.940446e+07	5.150421e+08	96.369242	68.162109
5	Babbon	64	[[darkolivegreen, steelblue, darkgray, chocola...	1.180997e+07	5.225487e+08	97.789880	96.878906
6	Babbon	128	[[darkslategray, rosybrown, lightsteelblue, to...	7.430892e+06	5.268546e+08	98.609191	120.629883
7	Babbon	256	[[tomato, dimgray, darkgray, darkseagreen, dar...	4.721145e+06	5.295322e+08	99.116310	136.444336

Table 3.0: Explanatory table of K-means results for Baboon image

After the creation of table as dataframe:

1- I plot resulted images as well as the original original image with their clusters/colors, image sizes, and explained variances
2- Calculate the optimal elbow with my function for WCSS, BCSS, explained variance vs number of clusters and image size vs explained variance
3- Plotting the curves of step 2
4- Comparison of best compressed images form elbow calculations and original image

For the rest of three "except landscape" images I did the same steps except redefinition of required functions.

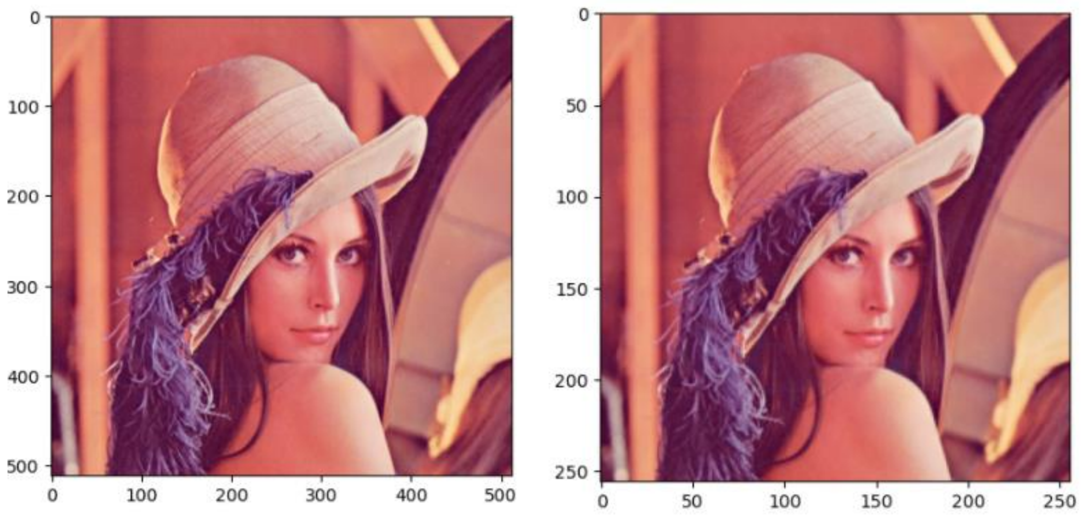


Figure 3.1: Original size and resized versions of Lenna image

	Image name	# of clusters	colors	WCSS	BCSS	Explained variance	Image size
0	Lenna	2	[[darksalmon, brown]]	1.469502e+08	2.600884e+08	63.897729	5.260742
1	Lenna	4	[[brown, lightcoral, burlywood, indianred]]	5.154029e+07	3.549426e+08	87.320428	11.097656
2	Lenna	8	[[indigo, lightcoral, sienna, wheat, brown, gr...	2.452664e+07	3.825101e+08	93.974344	19.202148
3	Lenna	16	[[sienna, darksalmon, indigo, wheat, brown, in...	1.230972e+07	3.944970e+08	96.974061	29.157227
4	Lenna	32	[[rosybrown, brown, indianred, indigo, navajow...	6.583708e+06	4.005240e+08	98.382809	44.350586
5	Lenna	64	[[indianred, brown, silver, rosybrown, sienna,...	3.839205e+06	4.032045e+08	99.056808	62.762695
6	Lenna	128	[[brown, palevioletred, navajowhite, sienna, l...	2.374632e+06	4.047321e+08	99.416705	82.034180
7	Lenna	256	[[brown, darksalmon, indigo, indianred, wheat,...	1.493630e+06	4.056023e+08	99.633101	96.882812

Table 3.1: Explanatory table of K-means results for Lenna image

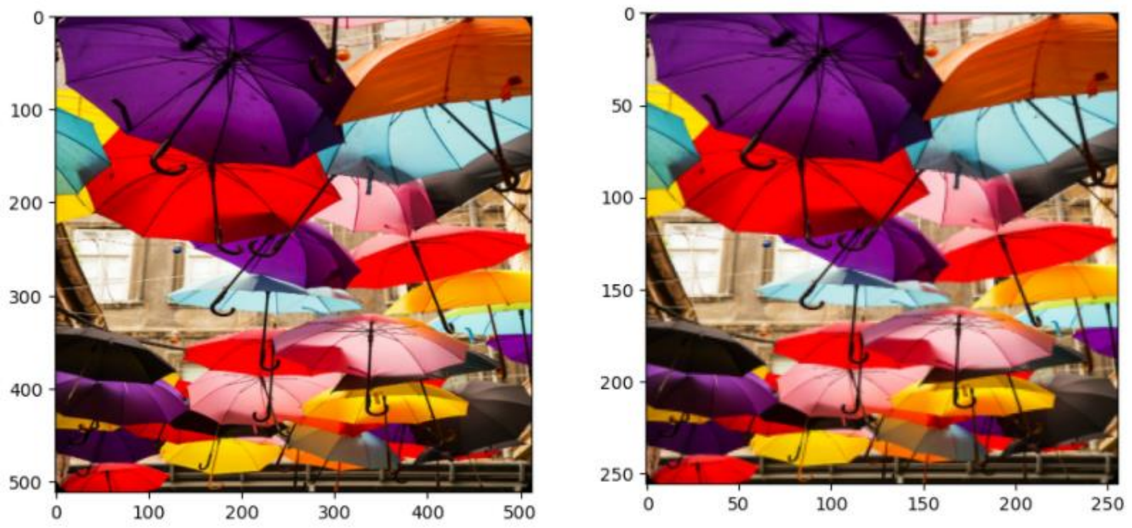


Figure 3.2: Original size and resized versions of Umbrella image

	Image name	# of clusters	colors	WCSS	BCSS	Explained variance	Image size
0	Umbrella	2	[[brown, tan]]	5.946038e+08	6.072839e+08	50.527507	5.000977
1	Umbrella	4	[[peru, silver, darkslategray, red]]	2.789784e+08	9.225666e+08	76.781692	9.450195
2	Umbrella	8	[[firebrick, tan, red, orange, dimgray, black,...	1.171431e+08	1.085065e+09	90.256008	15.324219
3	Umbrella	16	[[burlywood, darkred, darkorange, cadetblue, b...	5.278405e+07	1.146966e+09	95.600414	24.876953
4	Umbrella	32	[[purple, lightpink, firebrick, black, oranger...	2.670786e+07	1.174454e+09	97.776498	37.200195
5	Umbrella	64	[[indigo, indianred, orangered, black, khaki, ...	1.383623e+07	1.186812e+09	98.847603	50.394531
6	Umbrella	128	[[sienna, pink, black, purple, darkorange, red...	7.458535e+06	1.193457e+09	99.378929	63.761719
7	Umbrella	256	[[black, skyblue, firebrick, indigo, rosybrown...	4.146237e+06	1.196812e+09	99.654756	75.779297

Table 3.2: Explanatory table of K-means results for Umbrella image

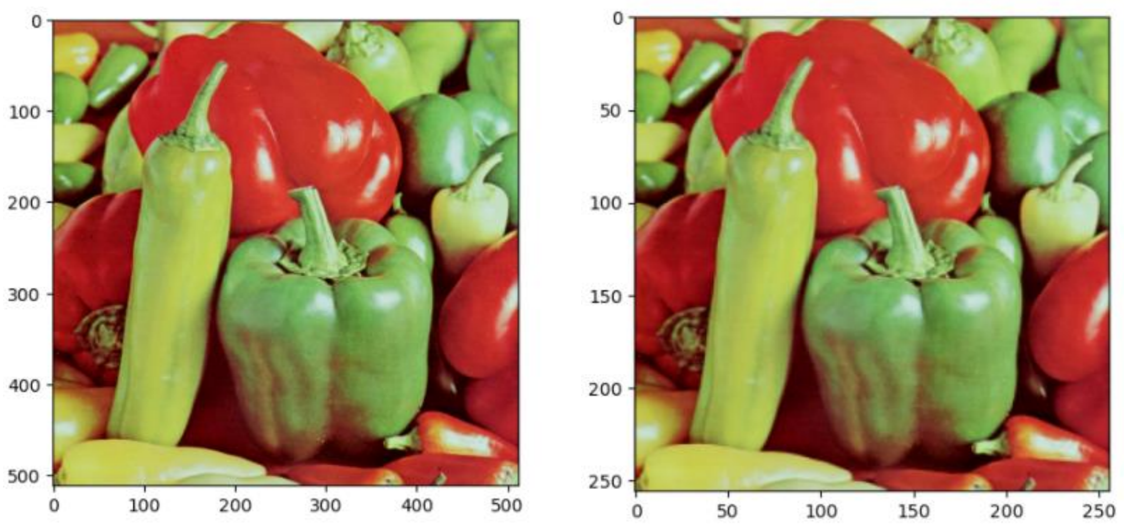


Figure 3.3: Original size and resized versions of Peppers image

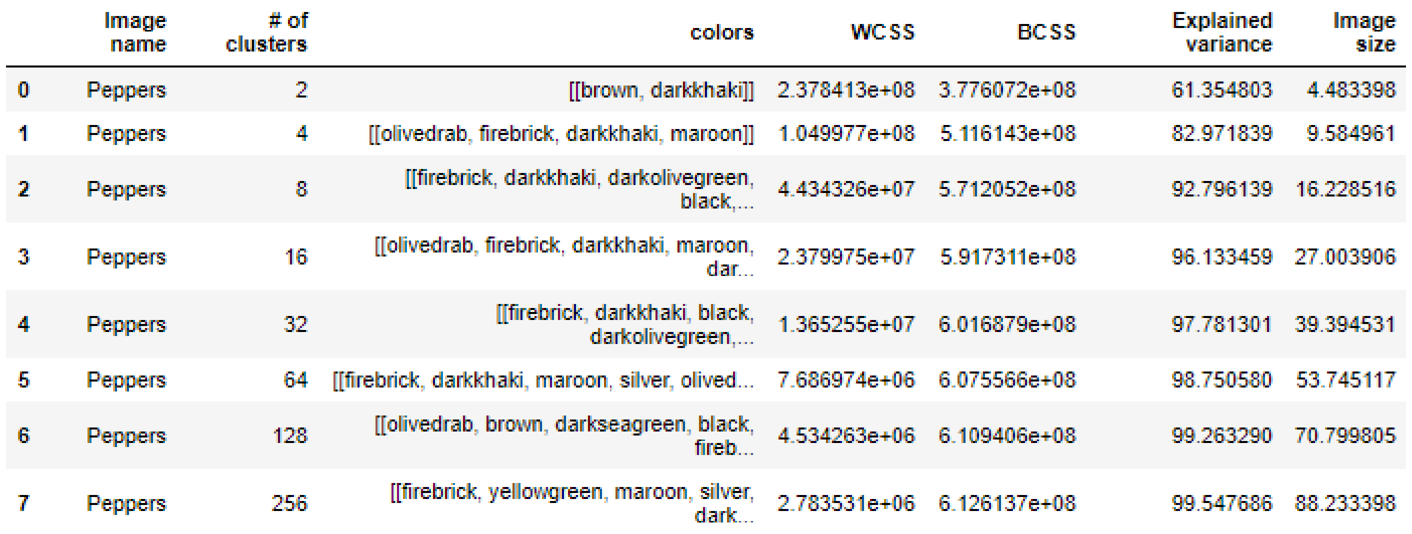


	Image name	# of clusters	colors	WCSS	BCSS	Explained variance	Image size
0	Peppers	2	[[brown, darkkhaki]]	2.378413e+08	3.776072e+08	61.354803	4.483398
1	Peppers	4	[[olivedrab, firebrick, darkkhaki, maroon]]	1.049977e+08	5.116143e+08	82.971839	9.584961
2	Peppers	8	[[firebrick, darkkhaki, darkolivegreen, black,...	4.434326e+07	5.712052e+08	92.796139	16.228516
3	Peppers	16	[[olivedrab, firebrick, darkkhaki, maroon, dar...	2.379975e+07	5.917311e+08	96.133459	27.003906
4	Peppers	32	[[firebrick, darkkhaki, black, darkolivegreen,...	1.365255e+07	6.016879e+08	97.781301	39.394531
5	Peppers	64	[[firebrick, darkkhaki, maroon, silver, olived...	7.686974e+06	6.075566e+08	98.750580	53.745117
6	Peppers	128	[[olivedrab, brown, darkseagreen, black, fireb...	4.534263e+06	6.109406e+08	99.263290	70.799805
7	Peppers	256	[[firebrick, yellowgreen, maroon, silver, dark...	2.783531e+06	6.126137e+08	99.547686	88.233398

Table 3.3: Explanatory table of K-means results for Peppers image

Landscape image is in ”jpg” format and images in jpg or jpeg format is already compressed(see jpeg) and when we use our K-means compression, some results size is bigger than original size of the image. Therefore, for landscape image size . After these steps, rest of the implementation of K-means image compression is same with other images.

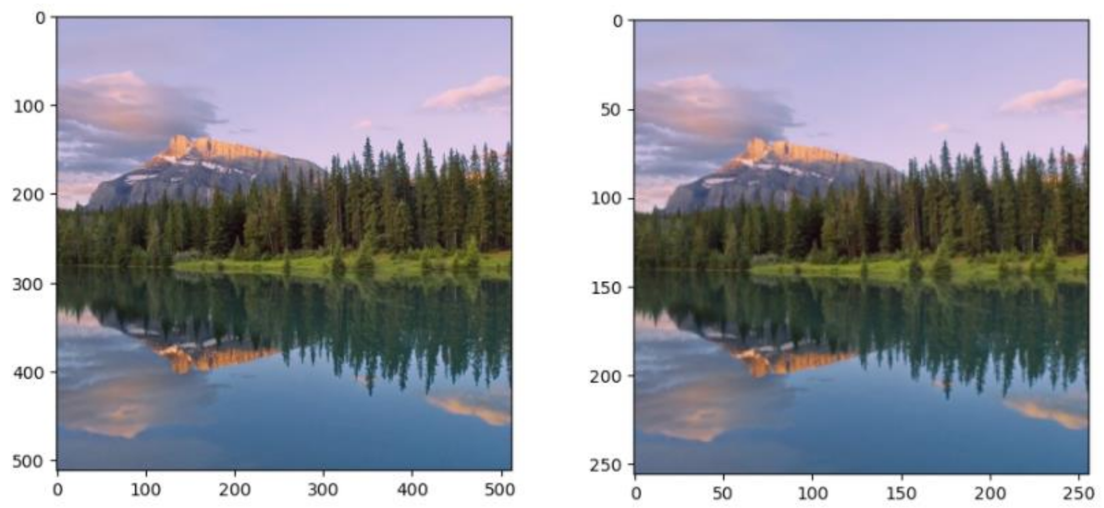


Figure 3.4: Original size and resized versions of Landscape image

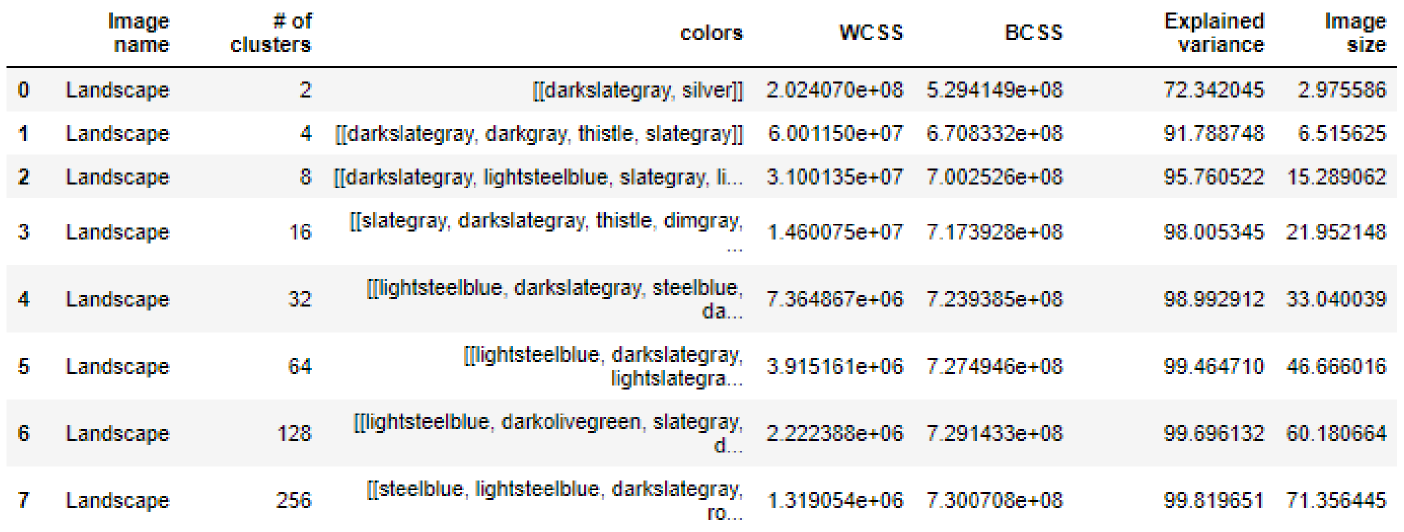


	Image name	# of clusters	colors	WCSS	BCSS	Explained variance	Image size
0	Landscape	2	[[darkslategray, silver]]	2.024070e+08	5.294149e+08	72.342045	2.975586
1	Landscape	4	[[darkslategray, darkgray, thistle, slategray]]	6.001150e+07	6.708332e+08	91.788748	6.515625
2	Landscape	8	[[darkslategray, lightsteelblue, slategray, li...	3.100135e+07	7.002526e+08	95.760522	15.289062
3	Landscape	16	[[slategray, darkslategray, thistle, dimgray, ...	1.460075e+07	7.173928e+08	98.005345	21.952148
4	Landscape	32	[[lightsteelblue, darkslategray, steelblue, da...	7.364867e+06	7.239385e+08	98.992912	33.040039
5	Landscape	64	[[lightsteelblue, darkslategray, lightslategra...	3.915161e+06	7.274946e+08	99.464710	46.666016
6	Landscape	128	[[lightsteelblue, darkolivegreen, slategray, d...	2.222388e+06	7.291433e+08	99.696132	60.180664
7	Landscape	256	[[steelblue, lightsteelblue, darkslategray, ro...	1.319054e+06	7.300708e+08	99.819651	71.356445

Table 3.3: Explanatory table of K-means results for Landscape image

Note: These tables are for general view of K-means results. Detailed results like No. of unique colors of images metric plots. Optimal elbow results and compressed image visualizations are in result section.

4. Results

I did image compression by using K-means for five different images with eight different number of clusters. In each K-means I found compressed image, how many colors that represents image(equals to number of clusters) and names, what is the size of the compressed images, WCSS, BCSS and explained variances and plot curves, calculate their optimal elbow, plot the curve for image size and explained variance and for each original image I choose the best results according to elbow calculation. We saw the results as table, now let’s see the visuals.

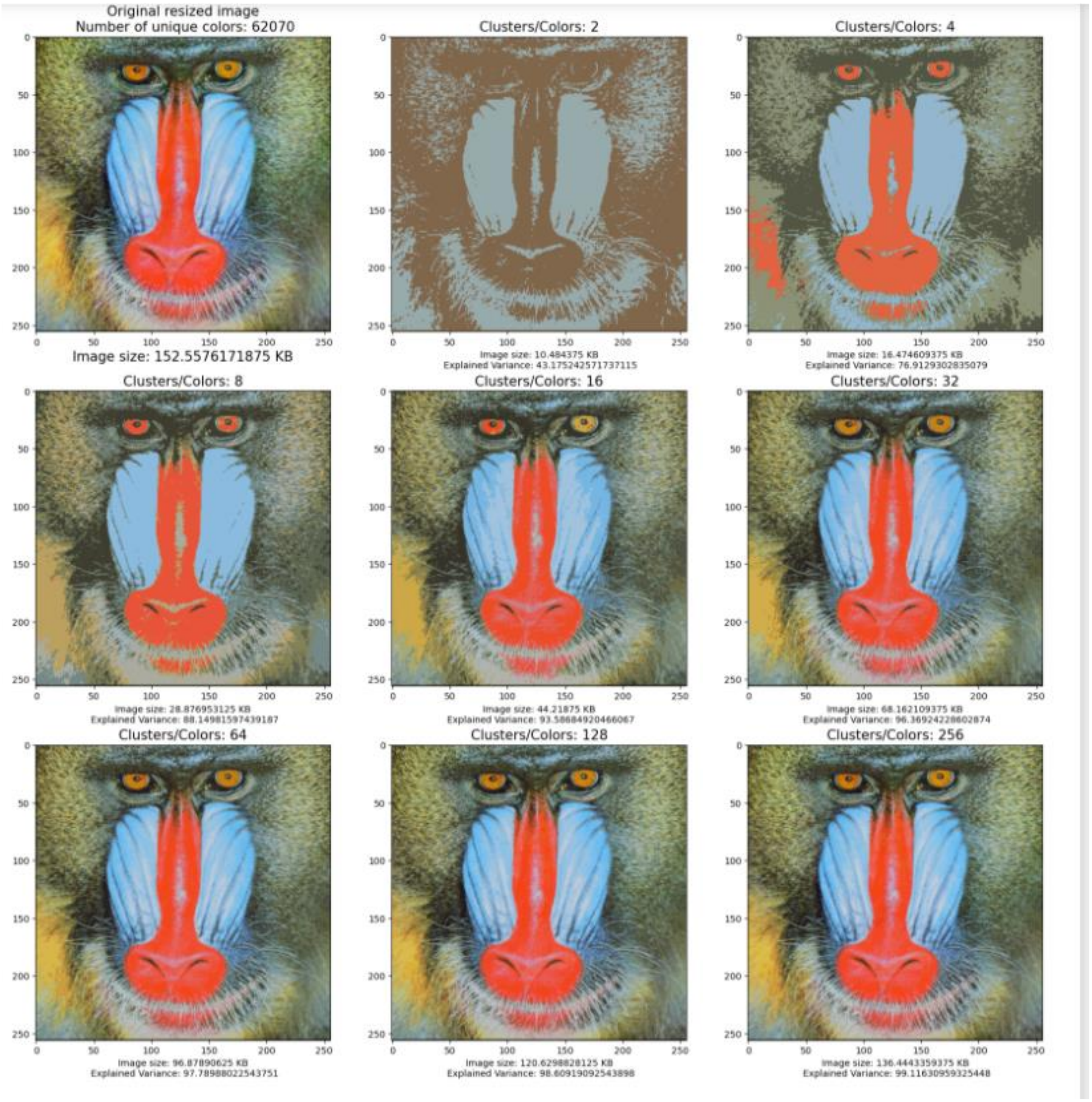

Original resized image
Number of unique colors: 62070
Image size: 152.5576171875 KB
Clusters/Colors: 2
Image size: 10.484375 KB
Explained Variance: 43.175242571737115
Clusters/Colors: 4
Image size: 16.474609375 KB
Explained Variance: 76.9129302835079
Clusters/Colors: 8
Image size: 28.876953125 KB
Explained Variance: 88.14981597439187
Clusters/Colors: 16
Image size: 44.21875 KB
Explained Variance: 93.58684920466067
Clusters/Colors: 32
Image size: 68.162109375 KB
Explained Variance: 96.36924228602874
Clusters/Colors: 64
Image size: 96.87890625 KB
Explained Variance: 97.78988022543751
Clusters/Colors: 128
Image size: 120.6298828125 KB
Explained Variance: 98.60919092543898
Clusters/Colors: 256
Image size: 136.4443359375 KB
Explained Variance: 99.11630959325448

Figure 4.0: K-means image compression on Baboon image

Figure 4.0 shows the eight different compressed image visuals, their size and explained variance(means how closer to original image), how many unique colors that they have . Original image has 62070 unique colors and it's size is 152KB.

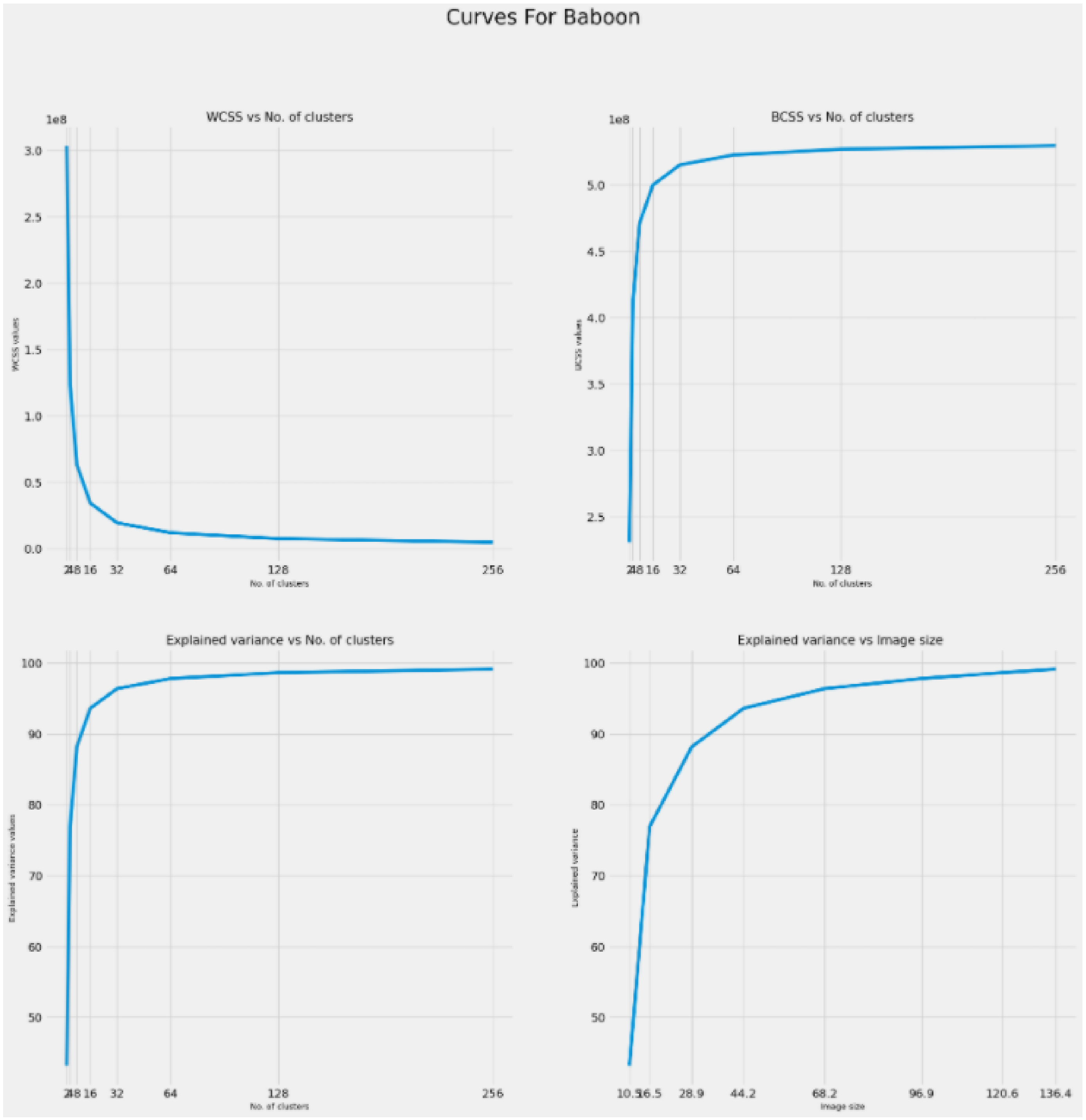
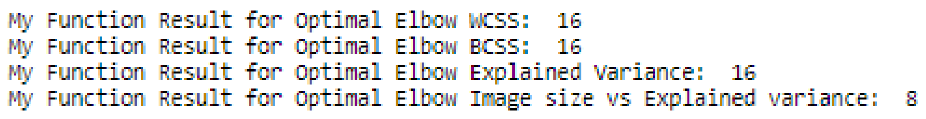


Figure 4.1: Metric curves for compressed Baboon images



```
My Function Result for Optimal Elbow WCSS:  16
My Function Result for Optimal Elbow BCSS:  16
My Function Result for Optimal Elbow Explained Variance:  16
My Function Result for Optimal Elbow Image size vs Explained variance:  8
```

Figure 4.2: My function results for optimal elbow for Baboon

When we look at Figure 4.1 we can clearly see that the optimal elbow calculation is correct in Figure 4.2. My function results give the No. of clusters which means the chosen image from elbow calculation has that much cluster. When we look at the curve of Explained variance and image size, the chosen compressed image for the best compression differs from others. The reason, why it differs from others is that, it has the best jump in terms of explained variance while keeping the increase in the image size low in comparison with the other compressed images.(see Figure 4.0)

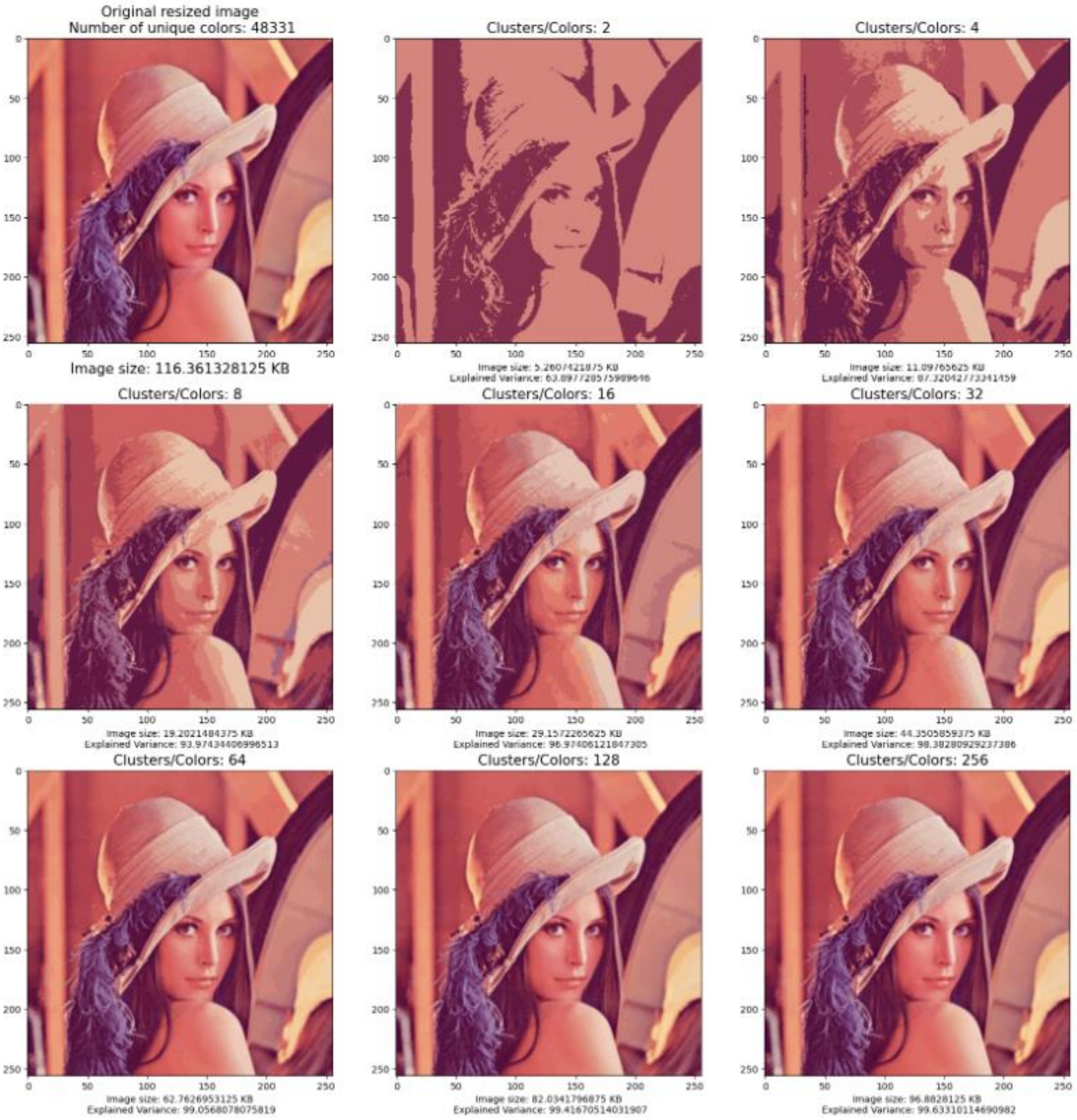

Original resized image
Number of unique colors: 48331
Image size: 116.361328125 KB
Clusters/Colors: 2
Image size: 5.2607421875 KB
Explained Variance: 63.0977285759896
Clusters/Colors: 4
Image size: 11.09765625 KB
Explained Variance: 87.32042773341459
Clusters/Colors: 8
Image size: 19.2021484375 KB
Explained Variance: 93.97434406996513
Clusters/Colors: 16
Image size: 29.1572265625 KB
Explained Variance: 96.97406121847305
Clusters/Colors: 32
Image size: 44.3505859375 KB
Explained Variance: 98.38280929237386
Clusters/Colors: 64
Image size: 62.7626953125 KB
Explained Variance: 99.0568078075819
Clusters/Colors: 128
Image size: 82.0341796875 KB
Explained Variance: 99.41670514031907
Clusters/Colors: 256
Image size: 96.8828125 KB
Explained Variance: 99.63310114690982

Figure 4.3: K-means image compression on Lenna image

Figure 4.3 shows the eight different compressed image visuals for Lenna, their size and explained variance(means how closer to original image), how many unique colors that they have. Original image has 48331 unique colors and it’s size is 116KB.

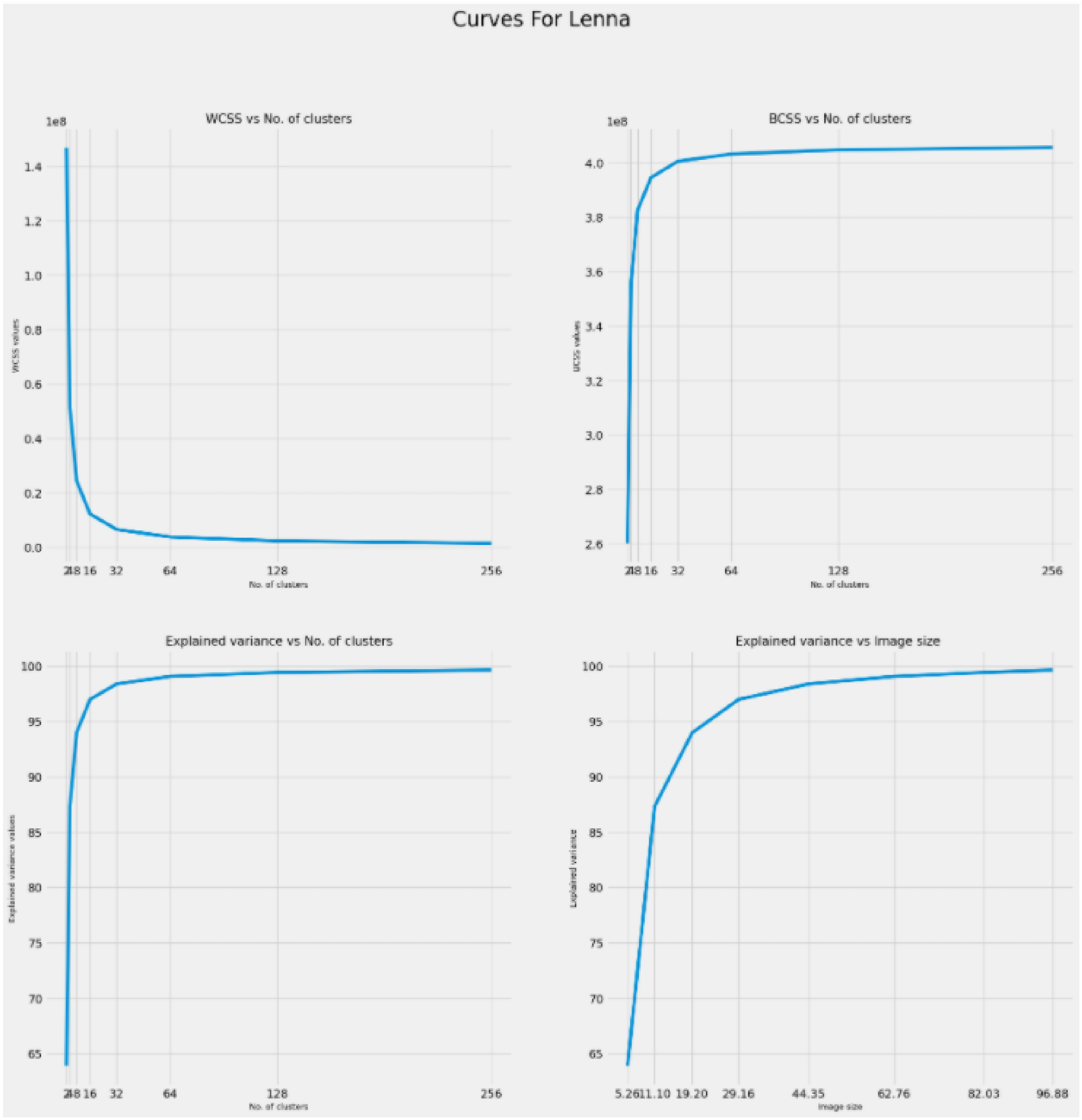


Figure 4.4: Metric curves for compressed Lenna images

```
My Function Result for Optimal Elbow WCSS:  16
My Function Result for Optimal Elbow BCSS:  16
My Function Result for Optimal Elbow Explained Variance:  16
My Function Result for Optimal Elbow Image size vs Explained variance:  8
```

Figure 4.5: My function results for optimal elbow for Lenna

When we look at Figure 4.4 we can clearly see that the optimal elbow calculation is correct in Figure 4.5. My function results give the No. of clusters which means the chosen image from elbow calculation has that much cluster. Function draws a line between start and end points and takes one that have most distance to the line. When we look at the curve of Explained variance and image size, the chosen compressed image for the best compression differs from others. The reason, why it differs from others is that, it has the best jump in terms of explained variance while keeping the increase in the image size low in comparison with the other compressed images.(see Figure 4.3)

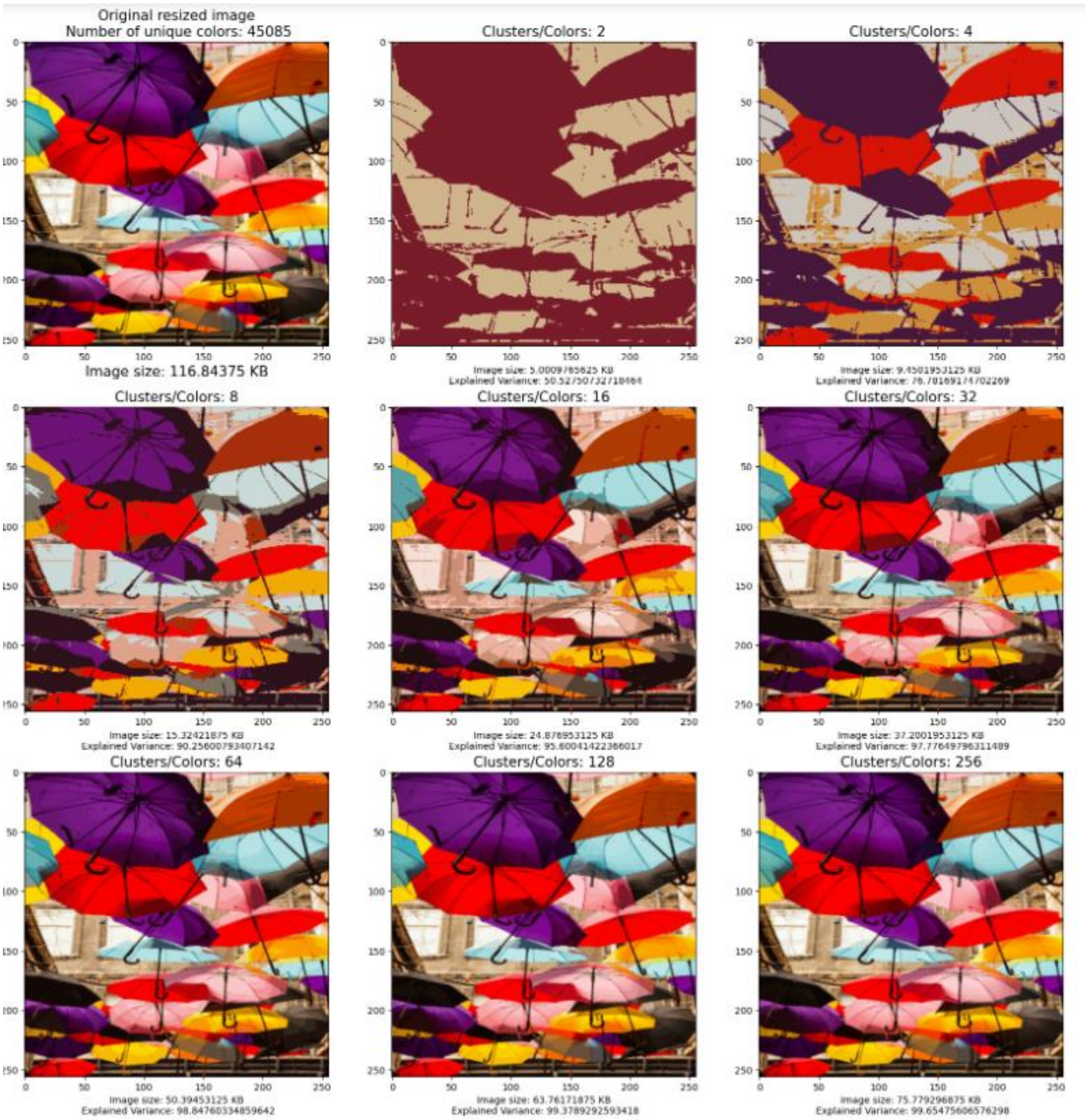

Original resized image
Number of unique colors: 45085
Image size: 116.84375 KB
Clusters/Colors: 2
Image size: 5.0009765625 KB
Explained Variance: 50.52750732710464
Clusters/Colors: 4
Image size: 9.4501953125 KB
Explained Variance: 76.70169174702269
Clusters/Colors: 8
Image size: 15.32421875 KB
Explained Variance: 90.25600793407142
Clusters/Colors: 16
Image size: 24.876953125 KB
Explained Variance: 95.60041422366017
Clusters/Colors: 32
Image size: 37.2001953125 KB
Explained Variance: 97.77649796311489
Clusters/Colors: 64
Image size: 50.39453125 KB
Explained Variance: 98.84760334859642
Clusters/Colors: 128
Image size: 63.76171875 KB
Explained Variance: 99.3789292593418
Clusters/Colors: 256
Image size: 75.779296875 KB
Explained Variance: 99.65475606576298

Figure 4.6: K-means image compression on Umbrella image

Figure 4.6 shows the eight different compressed image visuals for Umbrella, their size and explained variance(means how closer to original image), how many unique colors that they have. Original image has 45085 unique colors and it's size is 116KB

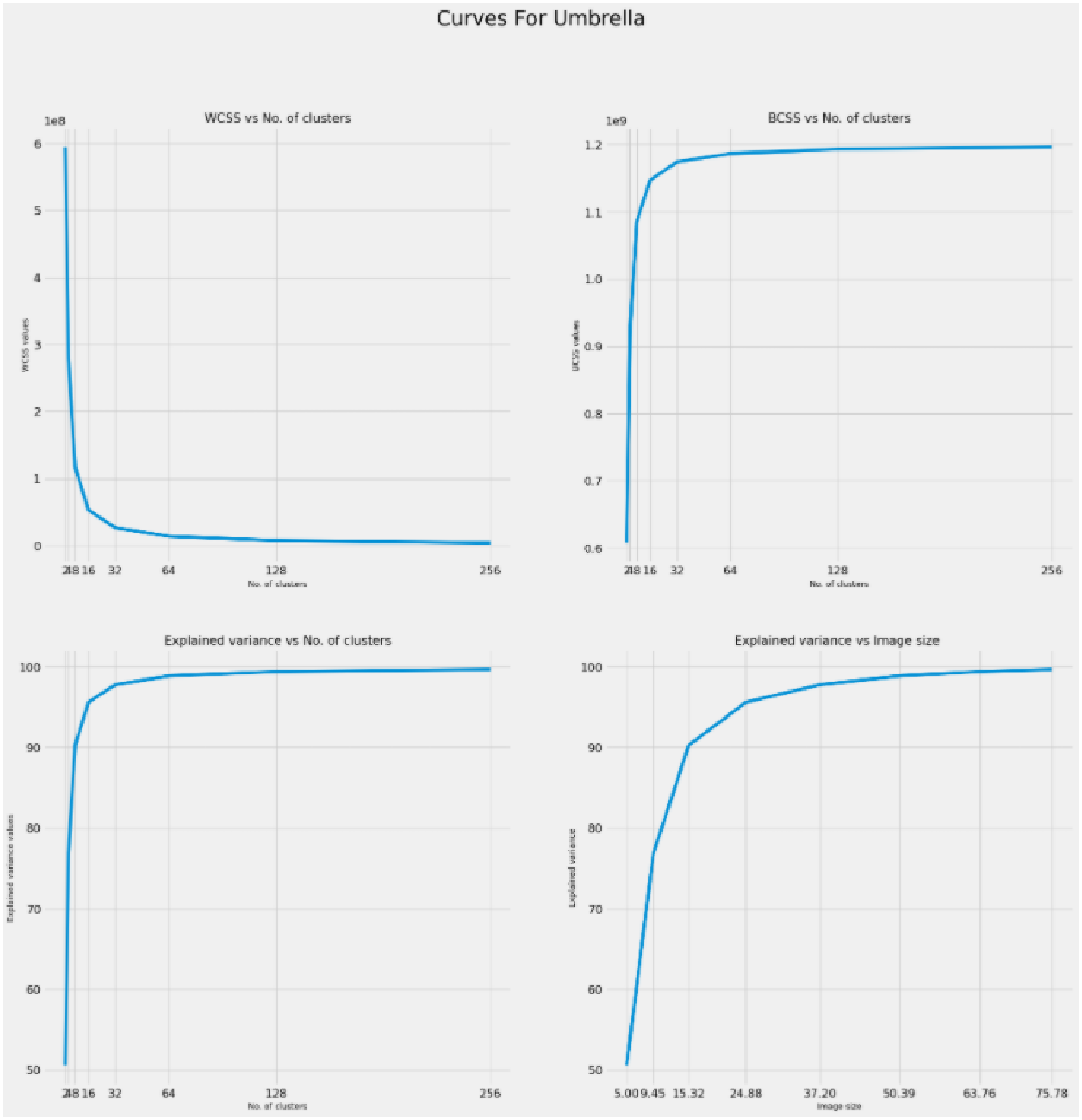
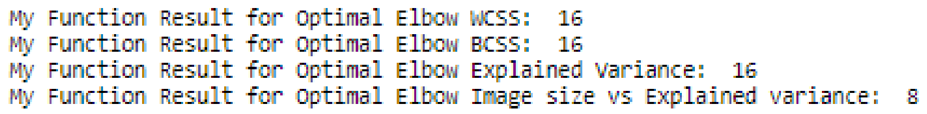


Figure 4.7: Metric curves for compressed Umbrella images



```
My Function Result for Optimal Elbow WCSS:  16
My Function Result for Optimal Elbow BCSS:  16
My Function Result for Optimal Elbow Explained Variance:  16
My Function Result for Optimal Elbow Image size vs Explained variance:  8
```

Figure 4.8: My function results for optimal elbow for Umbrella

When we look at Figure 4.7 we can clearly see that the optimal elbow calculation is correct in Figure 4.8. My function results give the No. of clusters which means the chosen image from elbow calculation has that much cluster. Function draws a line between start and end points and takes one that have most distance to the line. The difference in image size vs explained variance is same as the other two results(Maximizing the increase in the explained variance while keeping the increase in the image size as much as low).

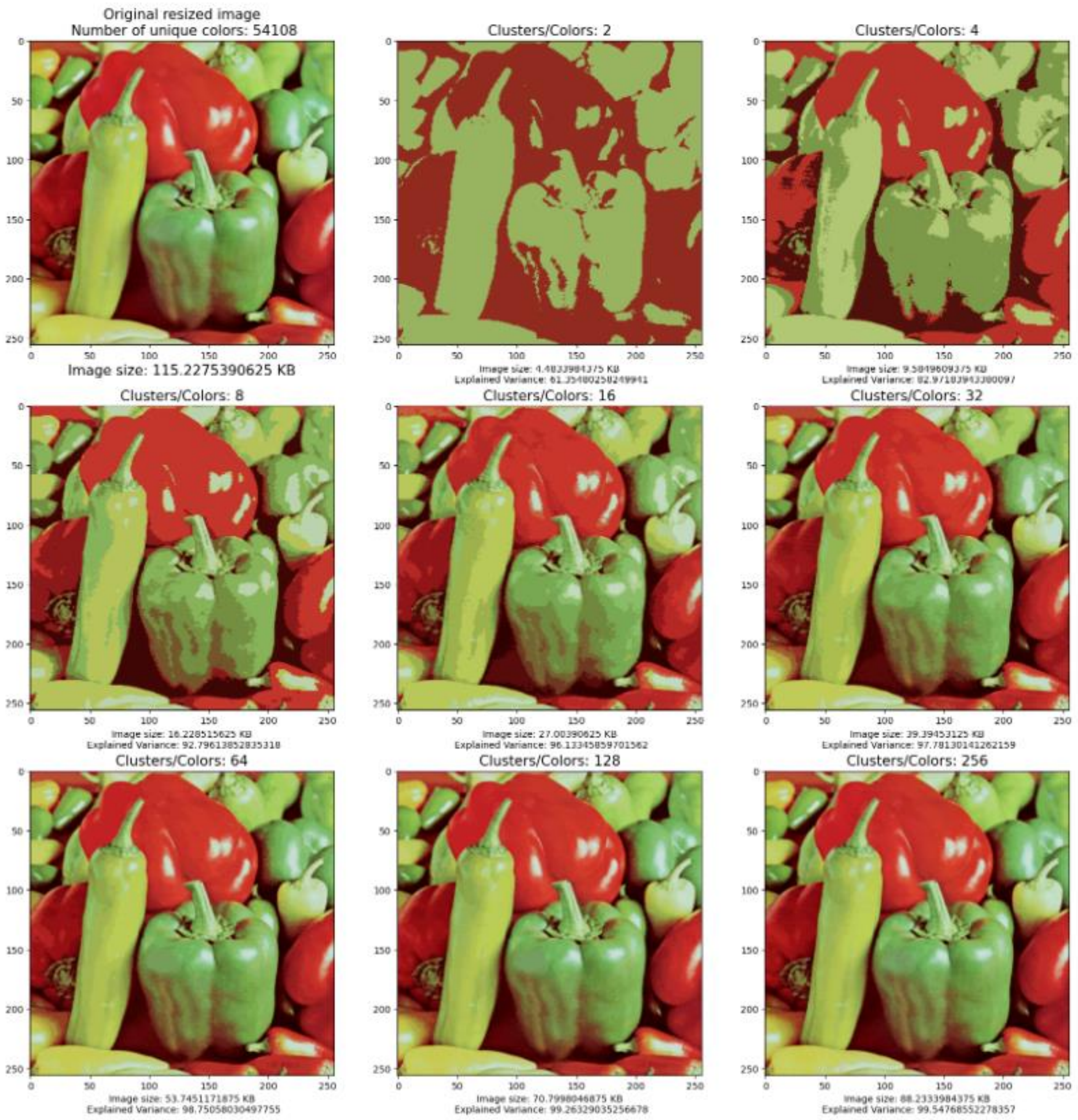

Original resized image
Number of unique colors: 54108
Image size: 115.2275390625 KB
Clusters/Colors: 2
Image size: 4.4833984375 KB
Explained Variance: 61.35480258249941
Clusters/Colors: 4
Image size: 9.584960937 5 KB
Explained Variance: 82.97183943380097
Clusters/Colors: 8
Image size: 16.228515625 KB
Explained Variance: 92.79613852835318
Clusters/Colors: 16
Image size: 27.00390625 KB
Explained Variance: 96.13345859701562
Clusters/Colors: 32
Image size: 39.39453125 KB
Explained Variance: 97.78130141262159
Clusters/Colors: 64
Image size: 53.7451171875 KB
Explained Variance: 98.75058030497755
Clusters/Colors: 128
Image size: 70.7998046875 KB
Explained Variance: 99.26329035256678
Clusters/Colors: 256
Image size: 88.2333984375 KB
Explained Variance: 99.54768552278357

Figure 4.9: K-means image compression on Peppers image

Figure 4.9 shows the eight different compressed image visuals for Peppers, their size and explained variance, how many unique colors that they have. Original image has 54108 unique colors and it’s size is 115KB

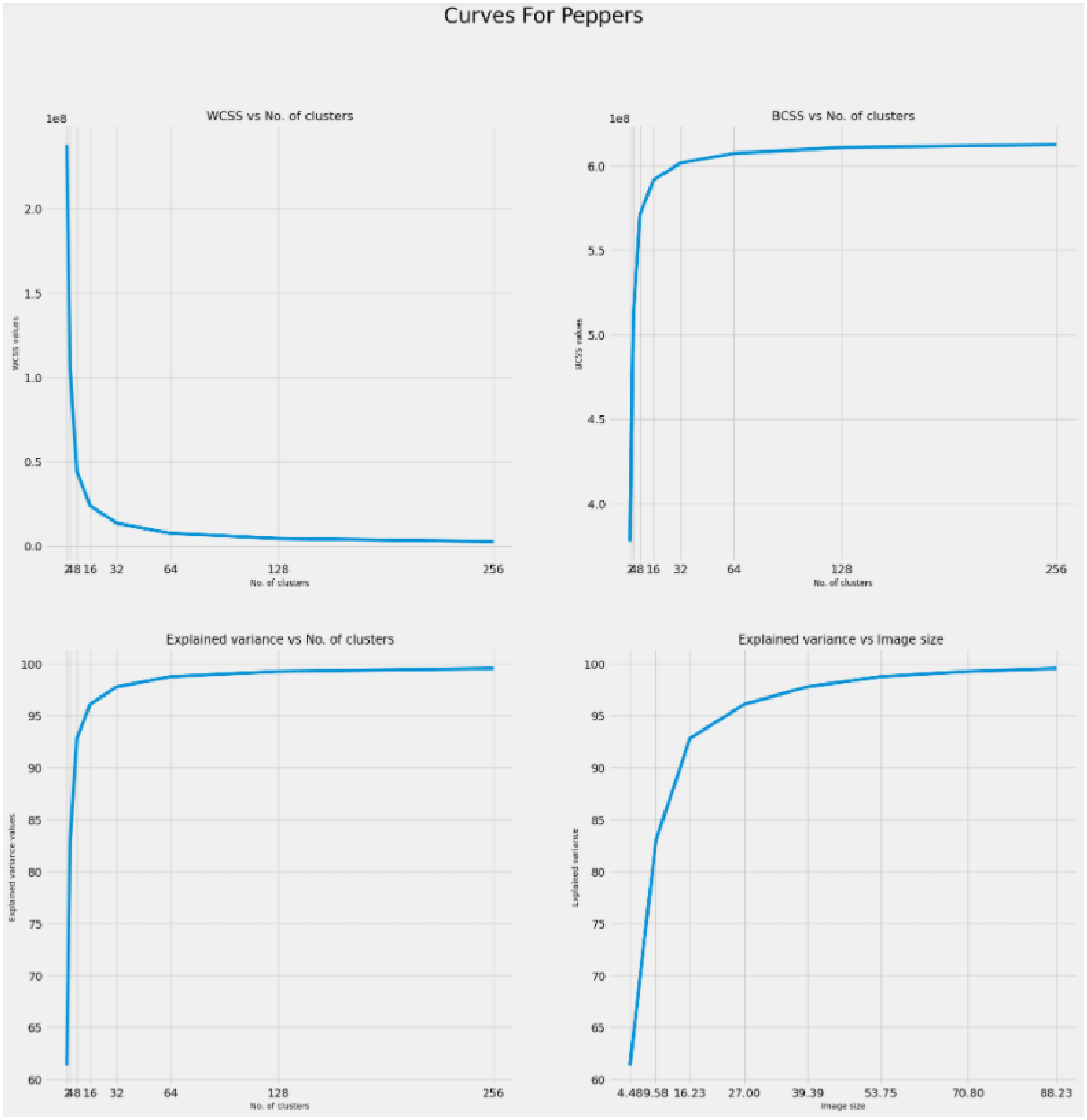


Figure 4.10: Metric curves for compressed Peppers images

```
My Function Result for Optimal Elbow WCSS:  16
My Function Result for Optimal Elbow BCSS:  16
My Function Result for Optimal Elbow Explained Variance:  16
My Function Result for Optimal Elbow Image size vs Explained variance:  8
```

Figure 4.11: My function results for optimal elbow for Peppers

When we look at Figure 4.10 we can clearly see that the optimal elbow calculation is correct in Figure 4.11. The difference comes from same reason that maximizing the increase in the explained variance while keeping the increase in the image size as much as low.

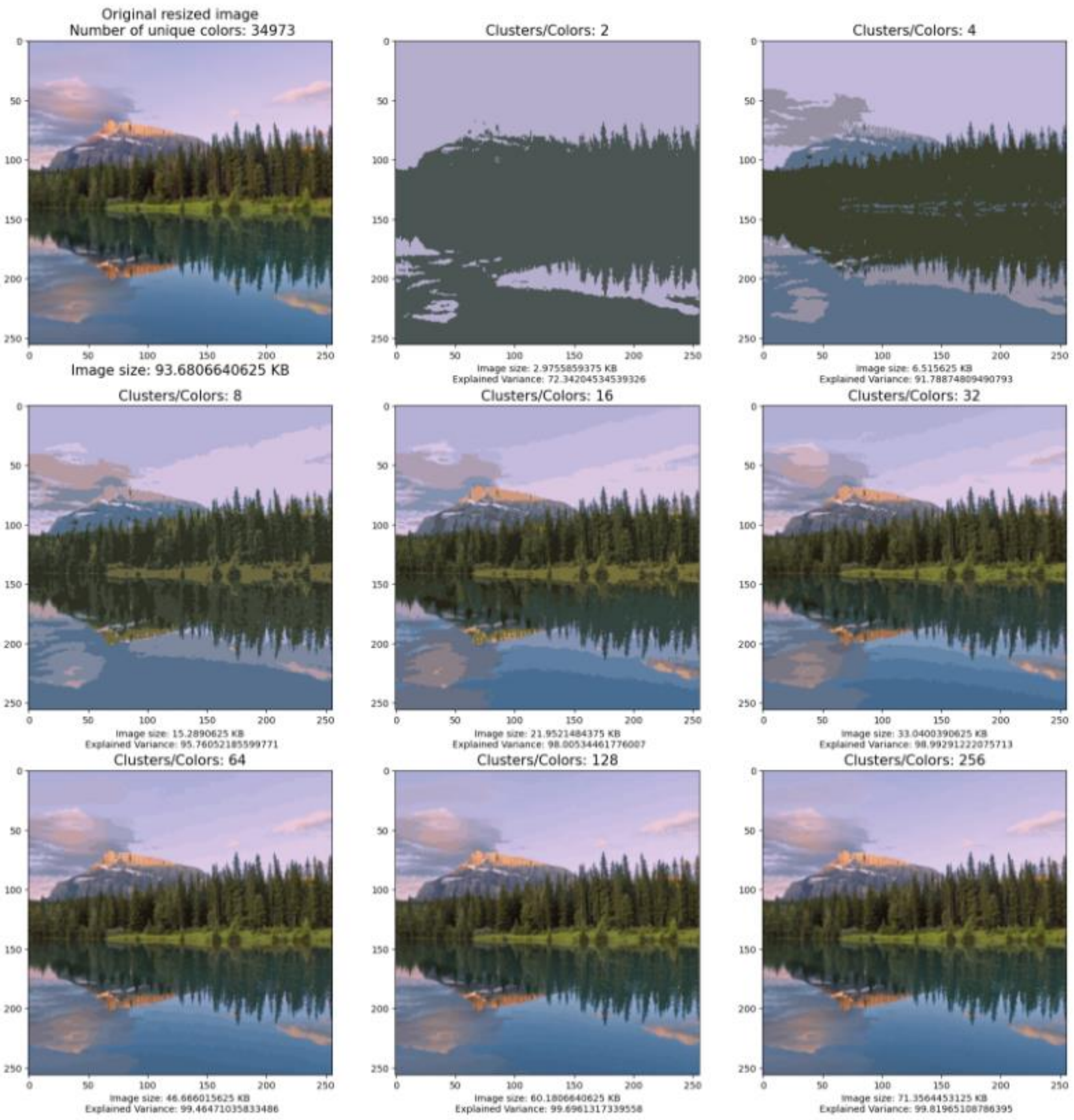

Original resized image
Number of unique colors: 34973
Image size: 93.6806640625 KB
Clusters/Colors: 2
Image size: 2.9755859375 KB
Explained Variance: 72.34204534539326
Clusters/Colors: 4
Image size: 6.515625 KB
Explained Variance: 91.78874809490793
Clusters/Colors: 8
Image size: 15.2890625 KB
Explained Variance: 95.76052185599771
Clusters/Colors: 16
Image size: 21.9521484375 KB
Explained Variance: 98.00534461776007
Clusters/Colors: 32
Image size: 33.0400390625 KB
Explained Variance: 98.99291222075713
Clusters/Colors: 64
Image size: 46.666015625 KB
Explained Variance: 99.46471035833486
Clusters/Colors: 128
Image size: 60.1806640625 KB
Explained Variance: 99.6961317339558
Clusters/Colors: 256
Image size: 71.3564453125 KB
Explained Variance: 99.81965108786395

Figure 4.12: K-means image compression on Landscape image

Figure 4.12 shows the eight different compressed image visuals for Landscape, their size and explained variance, how many unique colors that they have. Original image has 34973 unique colors and it’s size is 93KB in “png” format

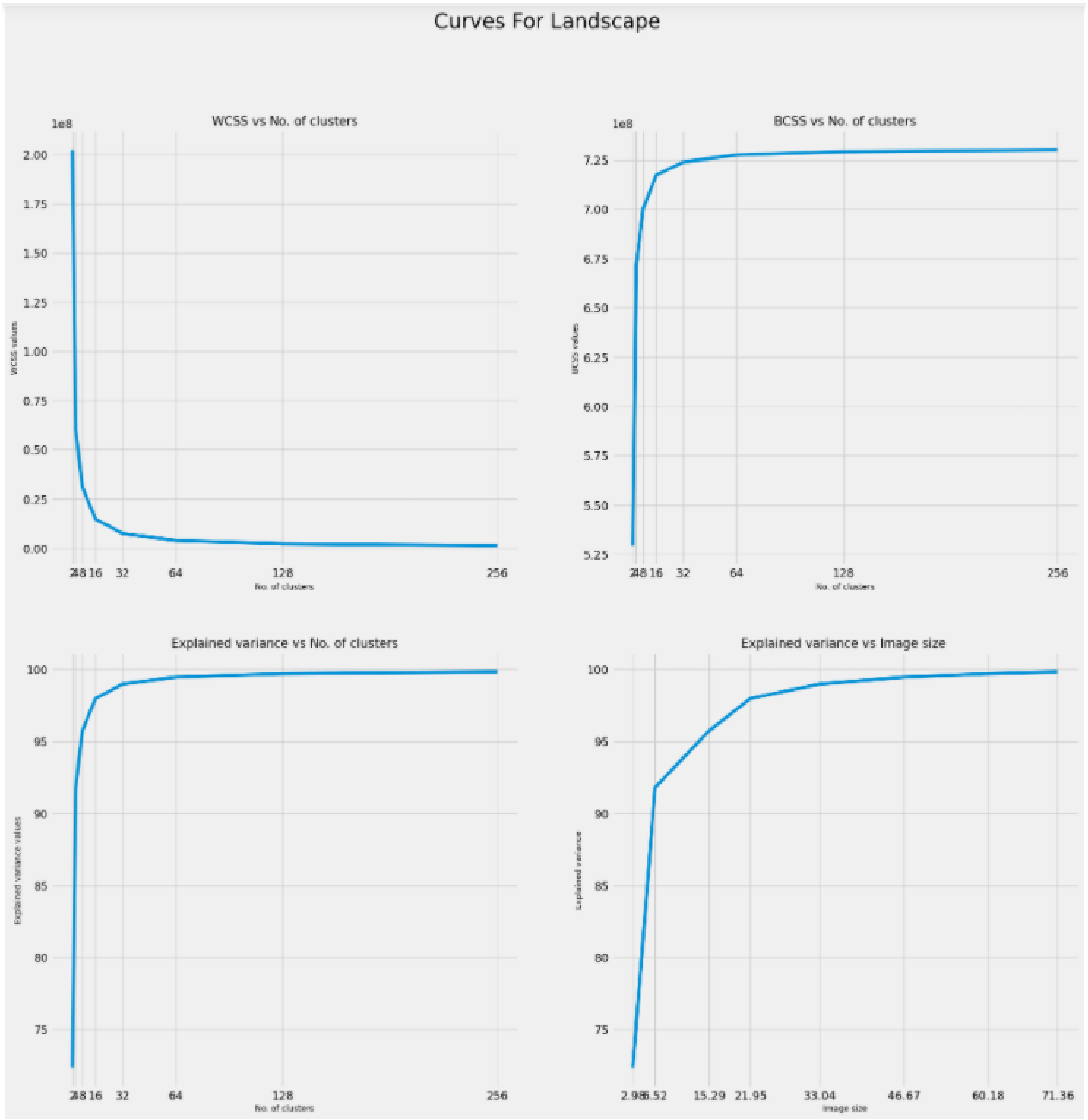


Figure 4.13: Metric curves for compressed Landscape images

```
My Function Result for Optimal Elbow WCSS:  16
My Function Result for Optimal Elbow BCSS:  16
My Function Result for Optimal Elbow Explained Variance:  16
My Function Result for Optimal Elbow Image size vs Explained variance:  8
```

Figure 4.14: My function results for optimal elbow for Landscape

When we look at Figure 4.13 we can clearly see that the optimal elbow calculation is correct in Figure 4.14. My function calculation based on distance to the line between start and end point and. For image size vs explained variance curve if you comment out the “print(distance)” you can see that third one has the most distance.

After showing the results and visuals, compare the original image and chosen compressed images from elbow calculations and my thougths.

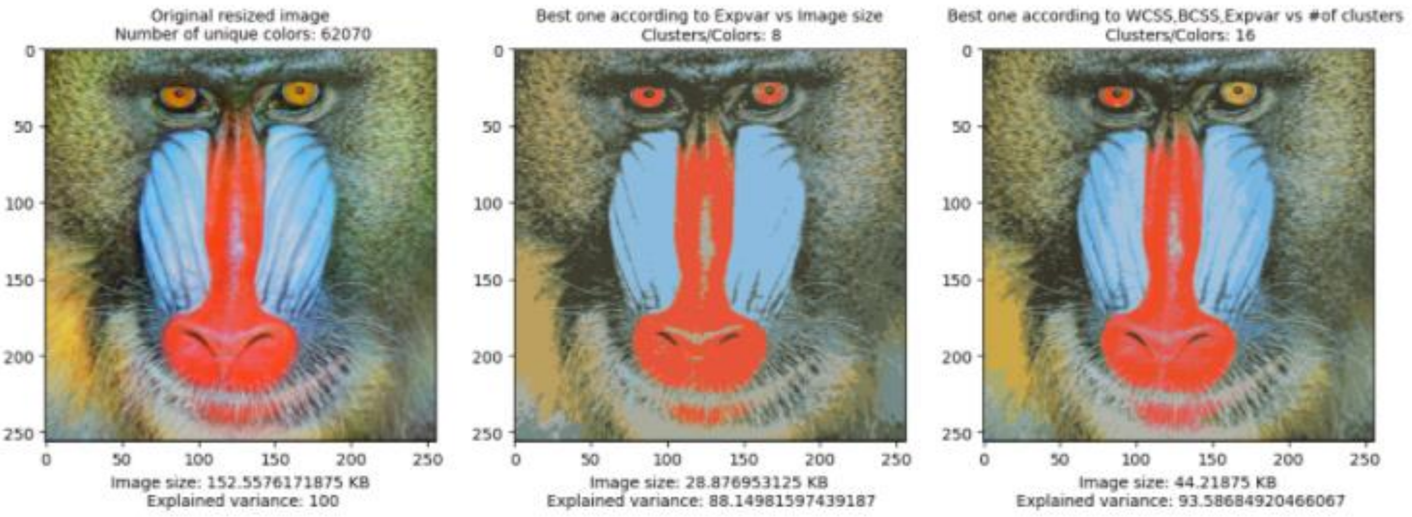

Original resized image
Number of unique colors: 62070
Image size: 152.5576171875 KB
Explained variance: 100
Best one according to Expvar vs Image size
Clusters/Colors: 8
Image size: 28.876953125 KB
Explained variance: 88.14981597439187
Best one according to WCSS,BCSS,Expvar vs #of clusters
Clusters/Colors: 16
Image size: 44.21875 KB
Explained variance: 93.58684920466067

Figure 4.15: Original image and chosen images from elbow calculation for Baboon

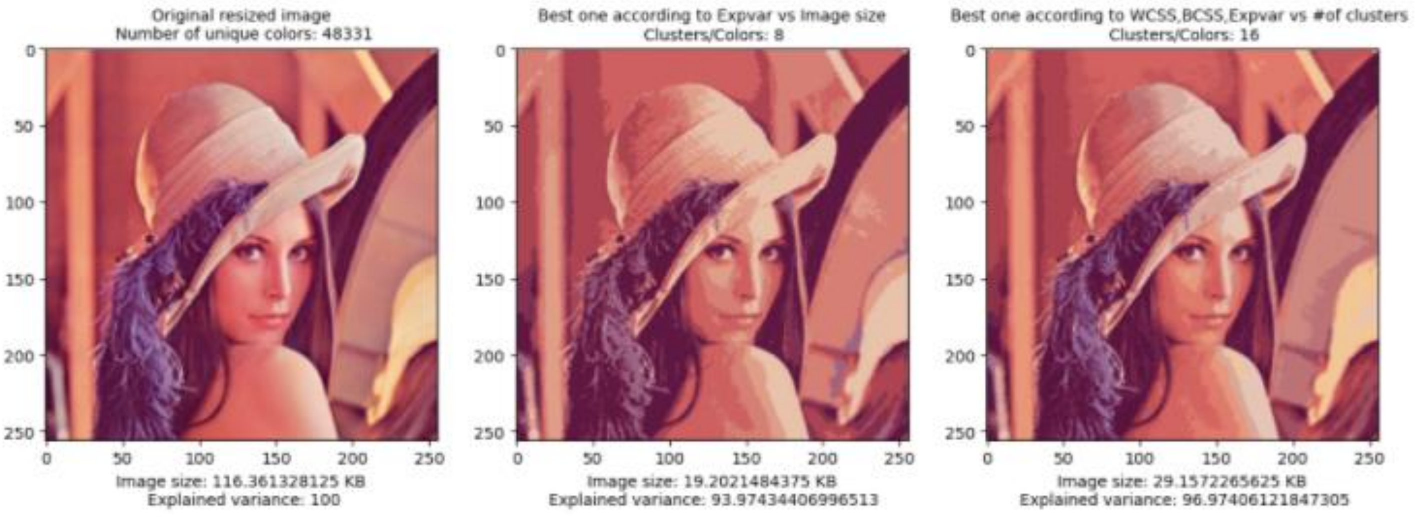

Original resized image
Number of unique colors: 48331
Image size: 116.361328125 KB
Explained variance: 100
Best one according to Expvar vs Image size
Clusters/Colors: 8
Image size: 19.2021484375 KB
Explained variance: 93.97434406996513
Best one according to WCSS,BCSS,Expvar vs #of clusters
Clusters/Colors: 16
Image size: 29.1572265625 KB
Explained variance: 96.97406121847305

Figure 4.16: Original image and chosen images from elbow calculation for Lenna

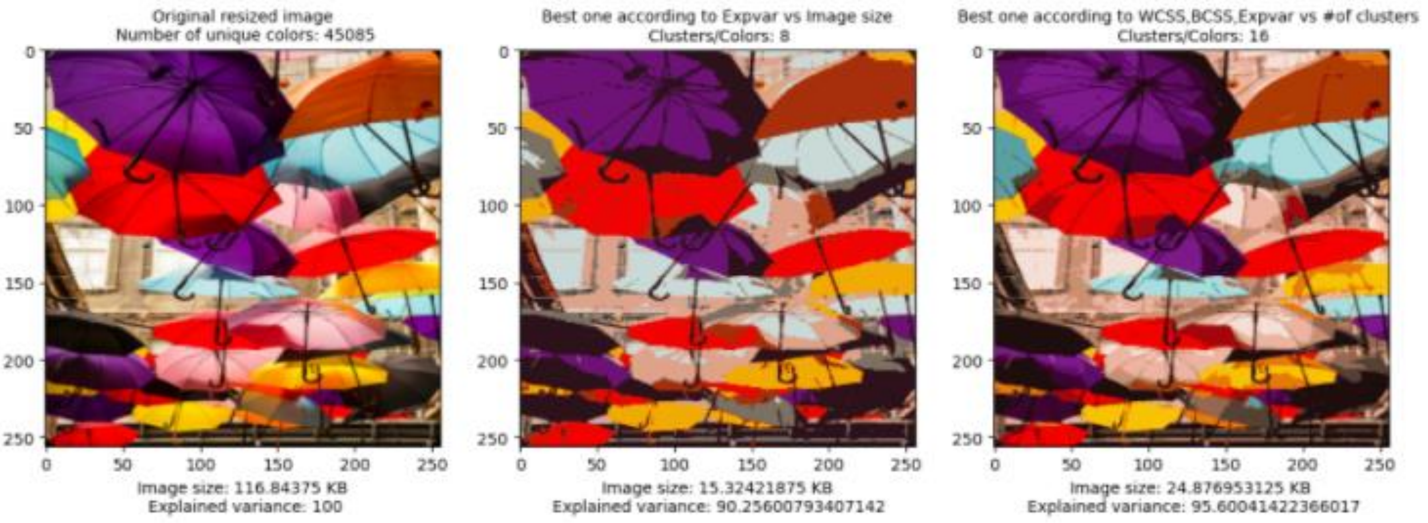

Original resized image
Number of unique colors: 45085
Image size: 116.84375 KB
Explained variance: 100
Best one according to Expvar vs Image size
Clusters/Colors: 8
Image size: 15.32421875 KB
Explained variance: 90.25600793407142
Best one according to WCSS,BCSS,Expvar vs #of clusters
Clusters/Colors: 16
Image size: 24.876953125 KB
Explained variance: 95.60041422366017

Figure 4.17: Original image and chosen images from elbow calculation for Umbrella

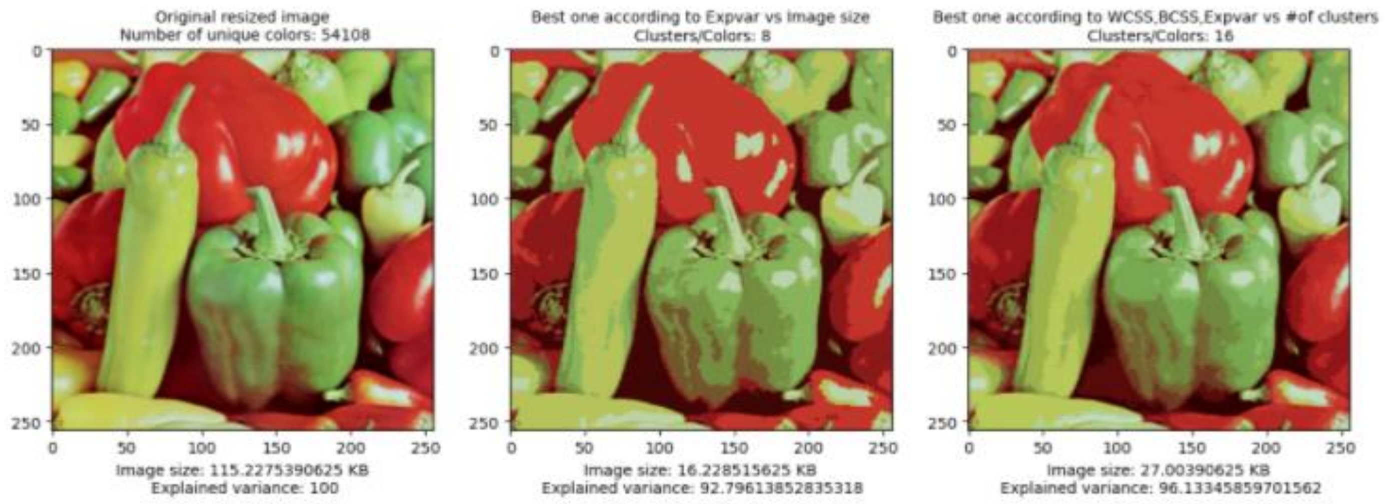

Original resized image
Number of unique colors: 54108
Image size: 115.2275390625 KB
Explained variance: 100
Best one according to Expvar vs image size
Clusters/Colors: 8
Image size: 16.228515625 KB
Explained variance: 92.79613852835318
Best one according to WCSS,BCSS,Expvar vs #of clusters
Clusters/Colors: 16
Image size: 27.00390625 KB
Explained variance: 96.13345859701562

Figure 4.18: Original image and chosen images from elbow calculation for Peppers

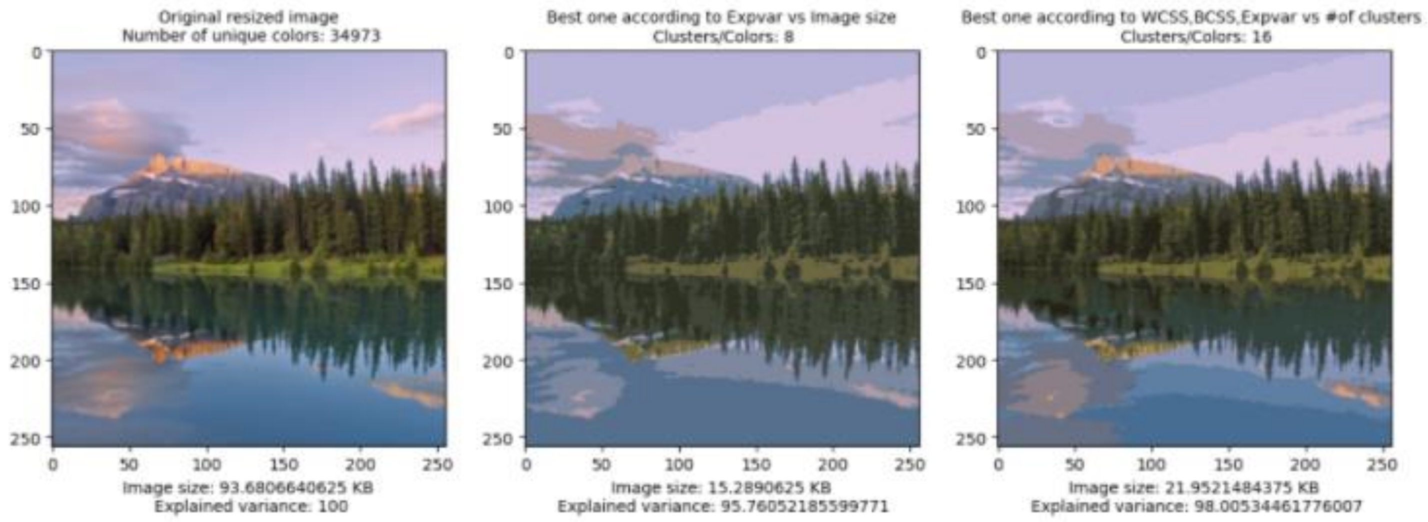

Original resized image
Number of unique colors: 34973
Image size: 93.6806640625 KB
Explained variance: 100
Best one according to Expvar vs image size
Clusters/Colors: 8
Image size: 15.2890625 KB
Explained variance: 95.76052185599771
Best one according to WCSS,BCSS,Expvar vs #of clusters
Clusters/Colors: 16
Image size: 21.9521484375 KB
Explained variance: 98.00534461776007

Figure 4.19: Original image and chosen images from elbow calculation for Landscape

Above five figures are representing best compared images from elbow calculation for five different images. In my opinion, compressed image that has 16 cluster is a better choice from elbow calculations, moreover compressed image that has 32 cluster is the best one according to me however, that is not the case. The point is the limitation of the choice for our storage capacity means that can we use that much size image, can we store it, can we use it in our code for processing instead of original image. For that approach size vs explained variance curve will help us more than other elbow calculation results because what is considering us in compression how much can I compress while keeping the similarity of the compressed one as much as high to original one.

5. Conclusion

In this assignment of “Data Science with Python” course, I implemented image compression by using K-means clustering. The aim of why I used K-means is that labeling the pixels RGB values and assign cluster center values to pixels that belongs to that cluster. By assigning center values to pixels that belongs to that cluster, we do compression by reducing the number of colors that represents the whole image. We apply this approach to five different images and for each image we reduce the number of colors that represents image by defining “K” cluster centers where K range in 2^1 to 2^8. Then, we calculate WCSS, BCSS, explained variance and image size for each compressed image. We calculate their elbow for choosing the best compressed image. The best approach for evaluating compression results is how much I compressed the image and how closer to the original one. For that approach, looking at image size and explained variance relation is the key. However, the aim of compression is reducing the size and choosing the best one according to evaluation metrics and according to your situation is important case. For example, evaluation result suggests me to choose third compressed image and my storage cannot handle the size of suggested image. In that case, I need to choose the more compressed one for that data/image or I can handle more size and choosing the less compressed one will give me more benefit then I will choose that one.