CS 452
Data Science with Python
Assignment 3 Report

Image Compression by K-Means Clustering

Author	*Uygar KAYA*
Instructor	*Assistant Prof. Reyhan Aydoğan*
Submission Date	*07.01.2022*

1 Introduction

In this assignment, we implement a K-means Clustering model that solves the problem of Image Compression.

In this assignment, the K-Means Cluster model, which finds the C number of cluster centers by pre-processing the 5 different images provided to us and assigns the pixel value of the closest cluster center to each pixel, was applied. After the model was applied, the performance values were measured and recorded in this report.

Image compression is a type of data compression that is done to digital images without compromising the image's quality. There are two kinds of image compression methods.

1. *Lossless*
2. *Lossy*

1.1 Lossless

This is a technique for shrinking a file's size while keeping the same quality as before it was compressed. The data quality is not damaged by lossless compression, and the file may be restored to its original format. This form of compression has no effect on the file's size.

Algorithms used for lossless compression are;

1. *Run Length encoding*
2. *Huffman Coding*
3. *Arithmetic Coding*

1.2 Lossy

Lossy compression is a compression approach that removes data that isn't perceptible. Lossy compression reduces the size of a photograph by discarding less relevant sections of the image.

Algorithms used for lossy compression are;

1. *Discrete Cosine Transform*
2. *Fractal compression*
3. *Transform Coding*

2 Methodology

In this assignment, we apply a K-Means Clustering model methodology using Python Programming Language.

During this assignment, using only 7 different Python programming language libraries these are;

1. *Pandas*
2. *NumPy*
3. *OpenCV*
4. *Scikit-Learn*
5. *Matplotlib*
6. *Pillow*

7. *Webcolors were used.*

First of all, 5 different images provided to us were read with the help of the OpenCV library and taken as raw data, then the images that came as 'BGR' was converted to 'RGB'. K-Means Clustering model was applied to these converted images after preprocessing processes such as reshaping and resizing. Converted to cluster sizes of 2^k where k = {2, 4, 8, 16, 32, 64, 128, 256} when applying the K-Means model. After performing these steps, the Within Cluster Sum of Squares (WCSS) and Between Cluster Sum of Squares (BCSS) metrics were coded and used to measure the variance of the model. Also, this report is illustrated by visualizing all the relevant information we collect and putting the relevant information into a pandas dataframe.

2.1 K-Means Clustering

The K-Means algorithm is a clustering method based on centroid value systems. This technique divides the data into k distinct clusters. The centroid point represents each cluster in the k-means clustering technique.

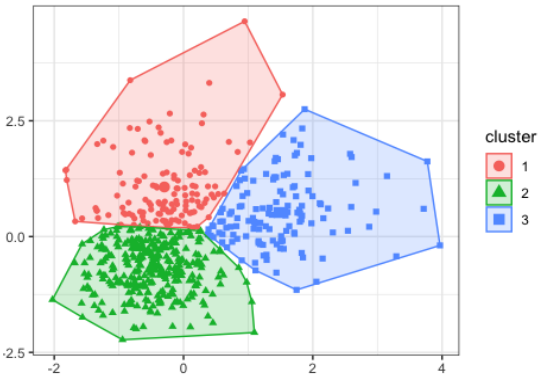


Figure 1 - Example of K-Means Clustering

1. Initialize **cluster centroids** $\mu_1, \mu_2, \ldots, \mu_k \in \mathbb{R}^n$ randomly.
2. Repeat until convergence: {

For every i, set

$$c^{(i)} := \arg\min_j \|x^{(i)} - \mu_j\|^2.$$

For each j, set

$$\mu_j := \frac{\sum_{i=1}^{m} 1\{c^{(i)} = j\} x^{(i)}}{\sum_{i=1}^{m} 1\{c^{(i)} = j\}}.$$

}

Figure 2 – Algorithm of the K-Means Clustering

2.2 K-Means Evaluation Metrics

***2.2.1 Within Cluster Sum of Squares (WCSS)** is a statistic that evaluates the variability of data within a cluster. A cluster with a small sum of squares is often more compact than one with a big number of squares. Higher-valued clusters have more variability among the observations inside the cluster.*

$$\text{WCSS} = \sum_{C_k}^{C_n} \Big(\sum_{d_i in\, C_i}^{d_m} distance(d_i, C_k)^2 \Big)$$

Where,
C is the cluster centroids and d is the data point in each Cluster.

Figure 3 – Mathematical Formula of the WCSS

***2.2.2 Between Cluster Sum of Squares (BCSS)** is a formula that calculates the average squared distance between all centroids. The Euclidean distance between a particular cluster centroid and all other cluster centroids is used to compute BCSS. Then you repeat the process for all the*

clusters, adding up all the values. This is the BCSS value

Between Cluster Sums of Squares: $$\mathrm{BSS} = \sum_{i=1}^{N_C} |C_i| \cdot d(\bar{\mathbf{x}}_{\mathbf{C_i}}, \bar{\mathbf{x}})^2$$

Figure 4 – Mathematical Formula of the BCSS

Where C_i = Cluster, N_c = Number of Clusters, X_{Ci} = Cluster Centroid and X = Sample Mean.

***2.2.3 Explained Variance (Silhouette Coefficients)** is calculated using the mean intra-cluster distance (a) and the mean nearest-cluster distance (b) for each sample. The Silhouette Coefficient for a sample is **(b - a) / max(a, b)**.*

$$s = \frac{b - a}{max(a, b)}$$

Figure 5 – Mathematical Formula of the Explained Variance

3 Implementation Details

3.1 Pre-Processing Part

In this assignment, firstly, we read the five different images provided to us with the help of the OpenCV library.

```
# using the plot_images() to plot the each images
plot_images(imageList)
```

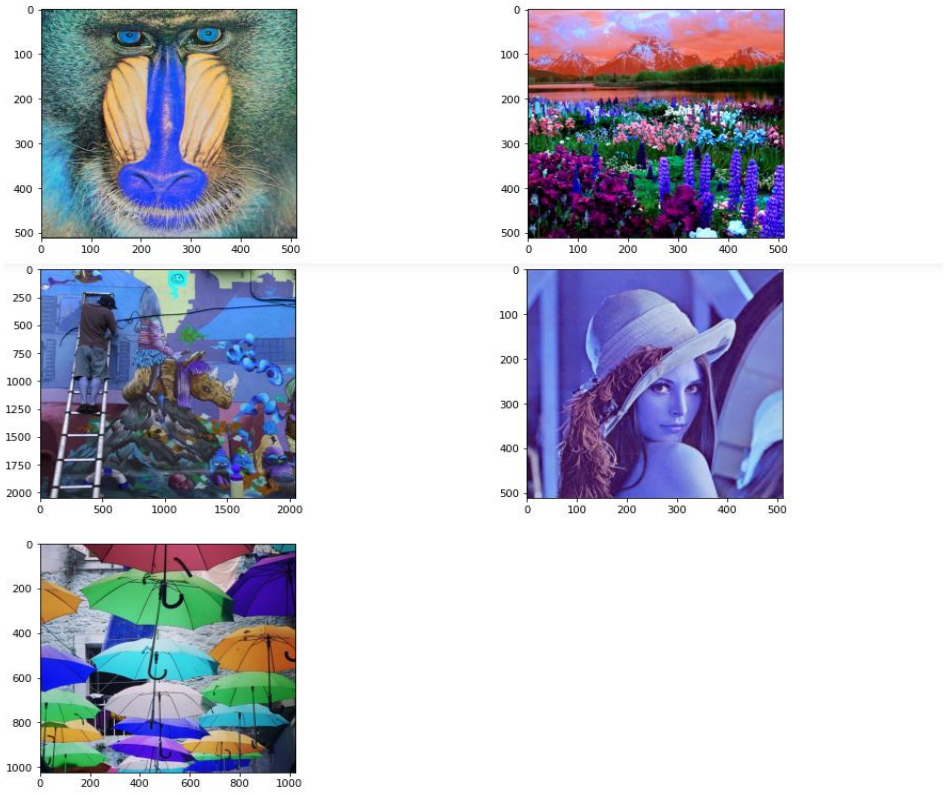


Figure 6 – BGR visualization of five different images

As you can see in Figure 6 since we drew the images with the help of the Matplotlib library, they came as BGR, so we convert the BGR incoming images to RGB.

```
# since we visualize the images with the help of the Matplotlib library, they are visualized with 'BGR'
# converting the 'BGR' to 'RGB'

imageListRGB = []
for image in imageList:
    image = cv.cvtColor(image, cv.COLOR_BGR2RGB)
    imageListRGB.append(image)
```

```
# using the plot_images() to visualize the each 'RGB' images
plot_images(imageListRGB)
```

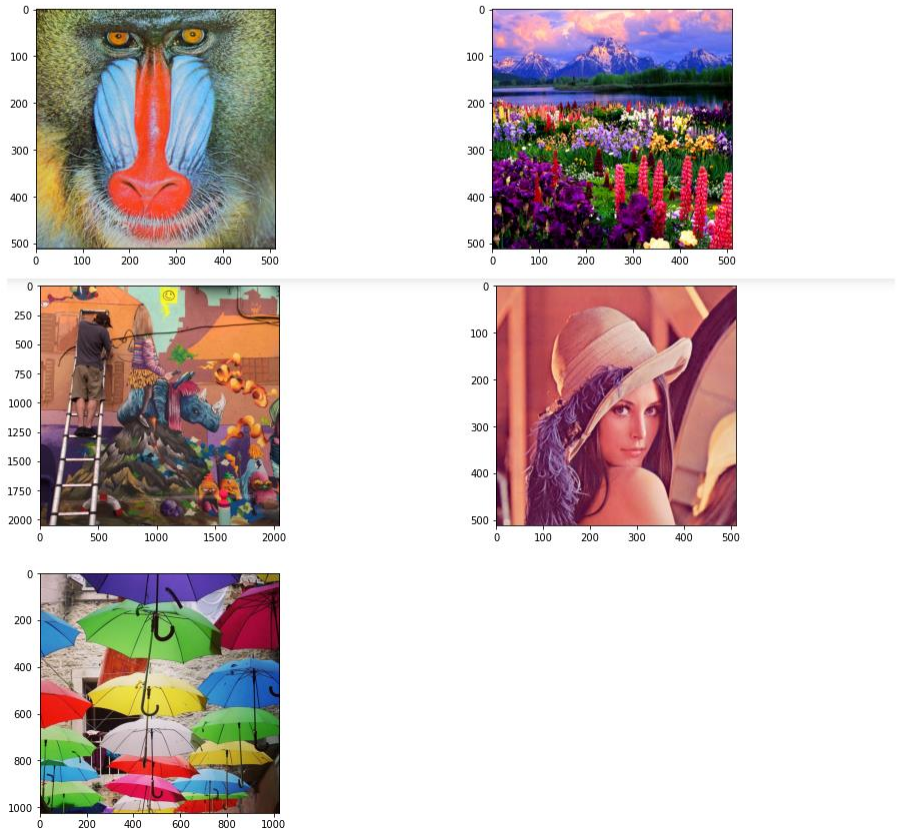


Figure 7 – RGB visualization of five different images

When we print the original shape and size of each of the RGB images in bytes, we get the following result. Also, we reshape each image to 256x256x3.

```
(512, 512, 3)
(512, 512, 3)
(2048, 2048, 3)
(512, 512, 3)
(1024, 1024, 3)
```

```
File Size In Bytes: 648010
File Size In Bytes: 614992
File Size In Bytes: 1074472
File Size In Bytes: 473777
File Size In Bytes: 278455
```

```
(256, 256, 3)
(256, 256, 3)
(256, 256, 3)
(256, 256, 3)
(256, 256, 3)
```

Figure 8 – Original image shape *Figure 9 – Original image size* *Figure 10 – Reshaped image*

*After this process, flattening input images to 2d * D (RGB) arrays from (256, 256, 3); therefore, we obtain the (65536, 3) result.* ***Note that*** *normally, we would use the Standard Scaler at this stage for various scaled features, but since every pixel value is between (0, 255), we don't need to scale our data. After these parts, we also find the unique colors in a single image.*

```
unique colors of baboon: 62070
unique colors of flowers: 57848
unique colors of graffiti: 47091
unique colors of lena: 48331
unique colors of umbrella: 49461
```

Figure 11 – Unique colors of the single images

3.2 Applying K-Means Clustering Model

First, we initialize the K-Means Clustering model algorithm. While applying this algorithm, we need to enter 3 different parameters, which are n_clusters, init, and random_state parameters, respectively. n_clusters is the number of clusters and in this parameter, clusters are specified for the values {2, 4, 8, 16, 32, 64, 128, 256}, init is which k-means initialization method should be used. I'm using k-means++, which is the default value for the init parameter. Random_state is the seed used for code repeatability and here I set it to 0. After doing these steps, we fit each image by opening an empty list for each image, and then we categorically append it to the empty lists that are opened to use these values.

```
# initializing K-Means Clustering models & fitting the each image
baboonKMeans, flowersKMeans, graffitiKMeans, lenaKMeans, umbrellaKMeans = ([] for i in range(5))

for index in range(len(2**np.arange(1,9))):
    KMeansBaboon, KMeansFlowers, KMeansGraffiti, KMeansLena, KMeansUmbrella = (KMeans(n_clusters=2**np.arange(1,9)[index], random

    KMeansBaboon.fit(baboon_numpyArray)
    KMeansFlowers.fit(flowers_numpyArray)
    KMeansGraffiti.fit(graffiti_numpyArray)
    KMeansLena.fit(lena_numpyArray)
    KMeansUmbrella.fit(umbrella_numpyArray)

    baboonKMeans.append(KMeansBaboon)
    flowersKMeans.append(KMeansFlowers)
    graffitiKMeans.append(KMeansGraffiti)
    lenaKMeans.append(KMeansLena)
    umbrellaKMeans.append(KMeansUmbrella)

imagesKMeansList = [baboonKMeans, flowersKMeans, graffitiKMeans, lenaKMeans, umbrellaKMeans]
```

Figure 12 – Initializing & fitting the k-means clustering algorithm

After these operations are done, for each setup with a different number of clusters, each pixel value is re-assigned as the cluster center closest to it. ***Note that compressed images will be displayed in the result section.***

```
# re-assigning each pixel value as the closest cluster center to it & clip in order to fit between (0, 255)
baboonCenteredImages, flowersCenteredImages, graffitiCenteredImages, lenaCenteredImages, umbrellaCenteredImages = ([] for i in ra

for index in range(len(2**np.arange(1,9))):
    baboonCenteredImages.append(np.clip(np.reshape(np.array([list(baboonKMeans[index].cluster_centers_[label]) for label in baboo
    flowersCenteredImages.append(np.clip(np.reshape(np.array([list(flowersKMeans[index].cluster_centers_[label]) for label in flo
    graffitiCenteredImages.append(np.clip(np.reshape(np.array([list(graffitiKMeans[index].cluster_centers_[label]) for label in g
    lenaCenteredImages.append(np.clip(np.reshape(np.array([list(lenaKMeans[index].cluster_centers_[label]) for label in lenaKMean
    umbrellaCenteredImages.append(np.clip(np.reshape(np.array([list(umbrellaKMeans[index].cluster_centers_[label]) for label in u
```

Figure 13 – Re-assigning each pixel value

To get each cluster size of the compressed images, the compressed images were saved in the local folder and each image was taken one by one and the compressed sizes were suppressed. ***Note that these results will be explained in detail in the result section.***

3.3 Metrics Calculations

In this assignment, we calculate 3 different metrics, these are Within Cluster Sum of Squares (WCSS), Between Cluster Sum of Squares (BCSS) and Explained Variance (Silhouette Coefficients).

```
# calculating the Within Cluster Sum of Squares (WCSS)
def WCSS(image, imageKMeans):
    wcss = 0
    for i in range(len(image)):
        distances = np.zeros(len(imageKMeans.cluster_centers_))
        for j in range(len(imageKMeans.cluster_centers_)):
            distances[j] = np.linalg.norm(imageKMeans.cluster_centers_[j] - image[i])
        wcss = wcss + distances[np.argmin(distances)]**2
    return wcss
```

```
# calculating the Between Cluster Sum of Squares (BCSS)
def BCSS(image, wcss):

    bcss = 0
    imgLength, imgDimension = image.shape

    for dimension in range(imgDimension):
        np.zeros(imgDimension)[dimension] = np.mean(image[:][dimension])
    for lenght in range(imgLength):
        bcss = bcss + np.linalg.norm(np.zeros(imgDimension)-image[lenght])**2

    return bcss-wcss
```

```
# calculating the explained variance(silhoette coeffients)
def explained_variance(image, bcss):

    exp_var = 0
    imgLength, imgDimension = image.shape

    for dimension in range(imgDimension):
        np.zeros(imgDimension)[dimension] = np.mean(image[:][dimension])
    for lenght in range(imgLength):
        exp_var = exp_var + np.linalg.norm(np.zeros(imgDimension)-image[lenght])**2

    return bcss/exp_var
```

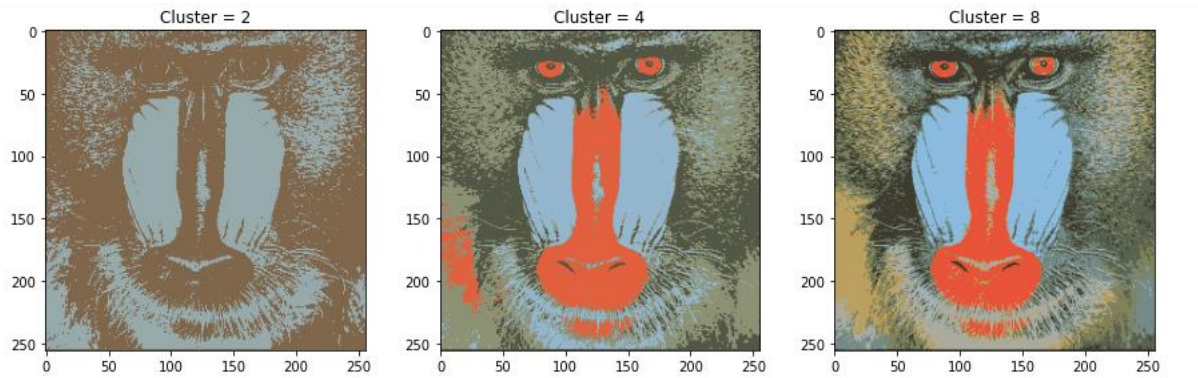
Figure 14 – Calculation the WCSS, BCSS & Explained Variance

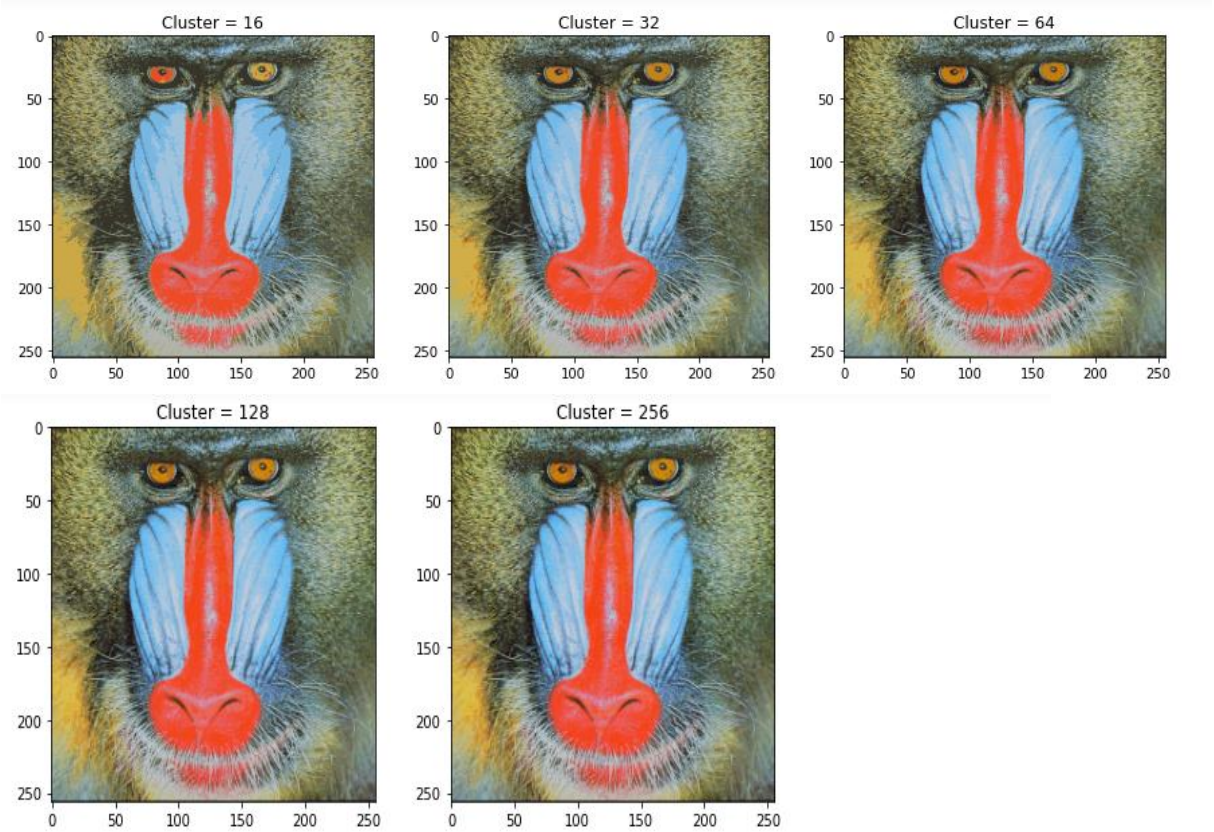
After this necessary calculation method, we apply these methods for all each image and the result will discuss about the next part which is result section.

4 Results

In this section, we look at metric values such as WCSS and BCSS to measure the accuracy of the K-Means Clustering model, which will be seen at the end of the Implementation Details section. Furthermore, we show the compressed images and the size of each compressed image in bytes.

4.1 Compress Images of Baboon



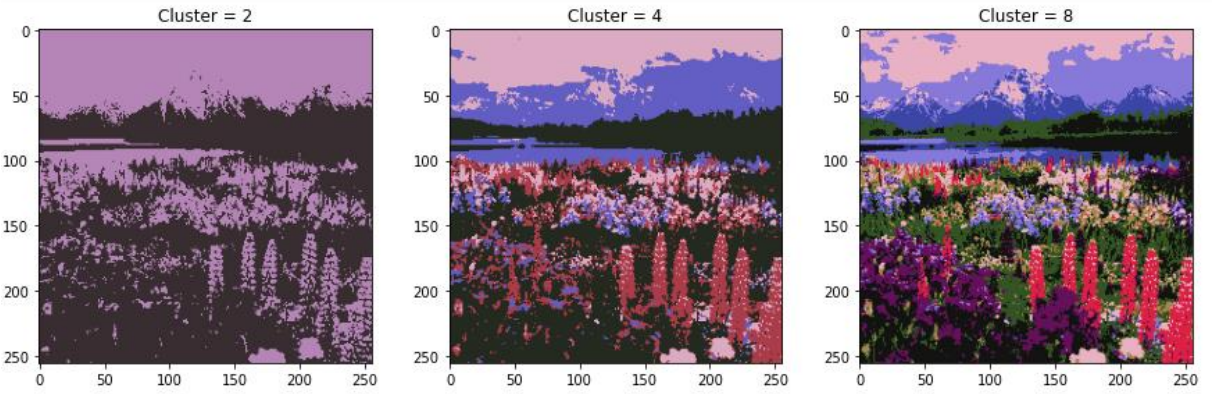


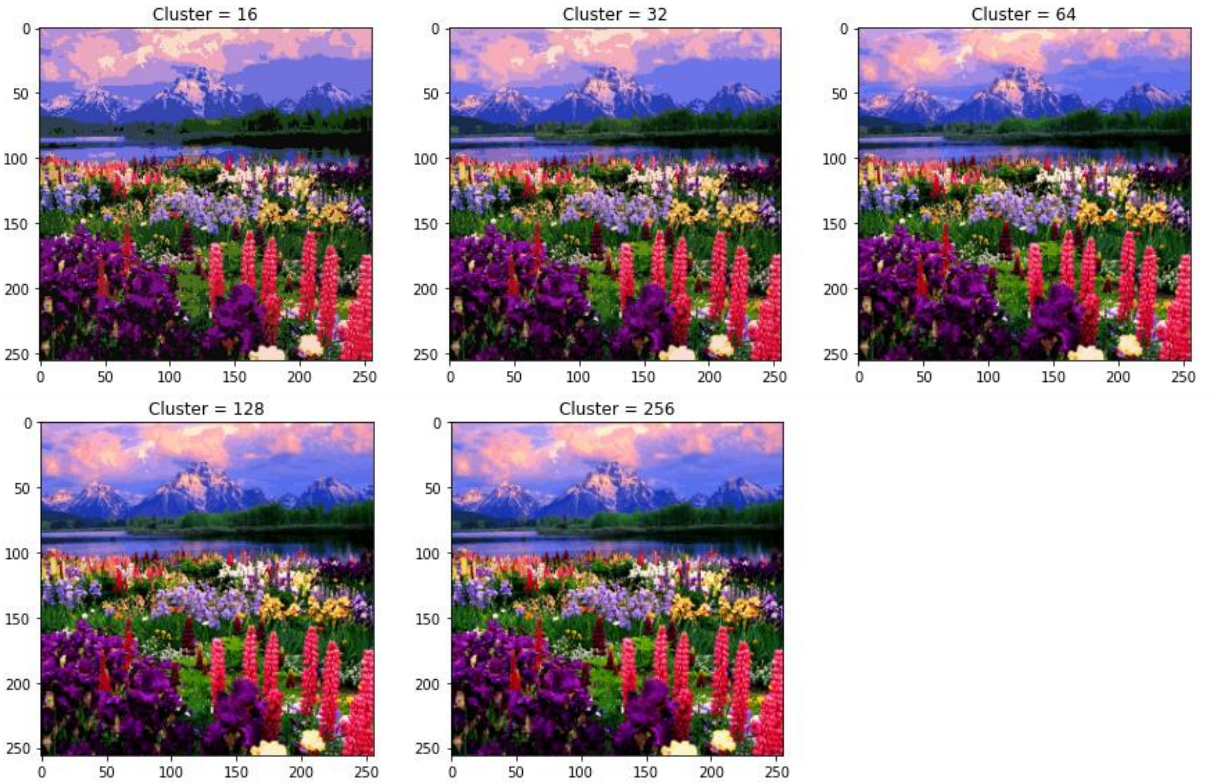

```
Compressed File Size In baboon With Clusters 2: 21876 bytes
Compressed File Size In baboon With Clusters 4: 36238 bytes
Compressed File Size In baboon With Clusters 8: 69590 bytes
Compressed File Size In baboon With Clusters 16: 94811 bytes
Compressed File Size In baboon With Clusters 32: 118244 bytes
Compressed File Size In baboon With Clusters 64: 135288 bytes
Compressed File Size In baboon With Clusters 128: 146075 bytes
Compressed File Size In baboon With Clusters 256: 152326 bytes
```

Figure 15 – Compress size of the baboon with clusters

As you can see in the implementation detail's part, Baboon's original size was 648010 bytes, but when we compress it, it went down to 152326 bytes with clusters 256. Also, there was no loss of image quality with cluster 256 after compressing.

4.2 Compress Images of Flowers



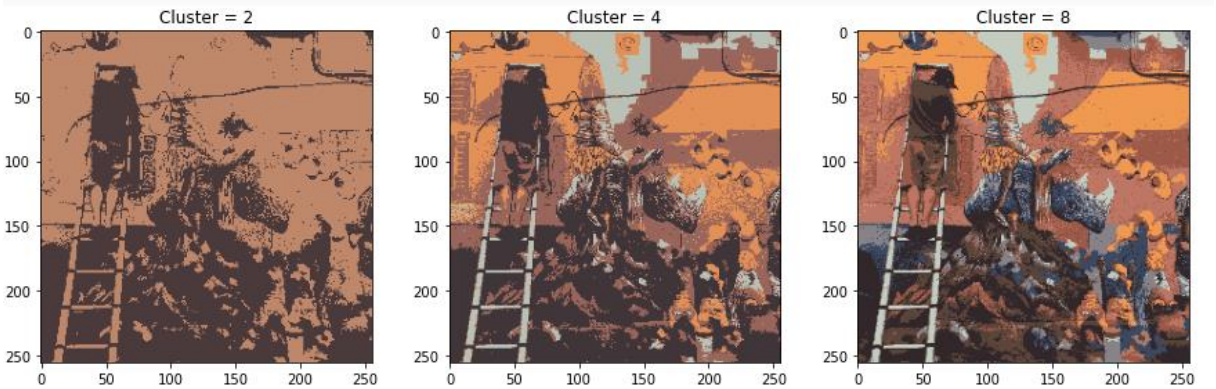


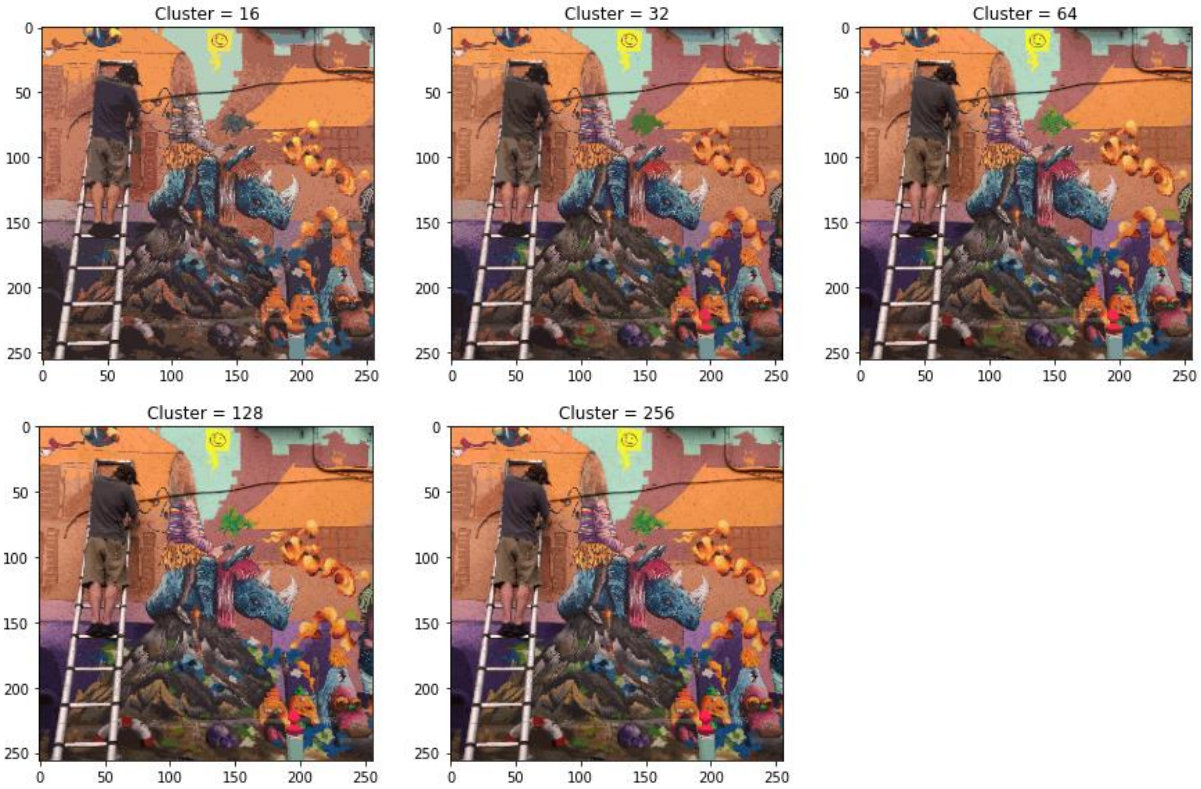

```
Compressed File Size In flowers With Clusters 2: 16905 bytes
Compressed File Size In flowers With Clusters 4: 31789 bytes
Compressed File Size In flowers With Clusters 8: 56345 bytes
Compressed File Size In flowers With Clusters 16: 77389 bytes
Compressed File Size In flowers With Clusters 32: 100253 bytes
Compressed File Size In flowers With Clusters 64: 115299 bytes
Compressed File Size In flowers With Clusters 128: 127715 bytes
Compressed File Size In flowers With Clusters 256: 136400 bytes
```

Figure 16 – Compress size of the flowers with clusters

As you can see in the implementation detail's part, Flowers's original size was 614992 bytes, but when we compress it, it went down to 136400 bytes with clusters 256. Also, there was no loss of image quality with cluster 256 after compressing.

4.3 Compress Images of Graffiti



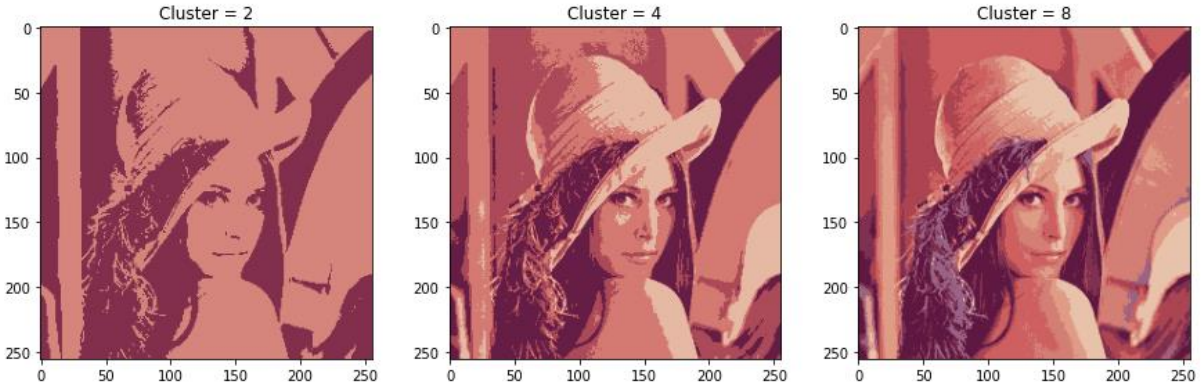


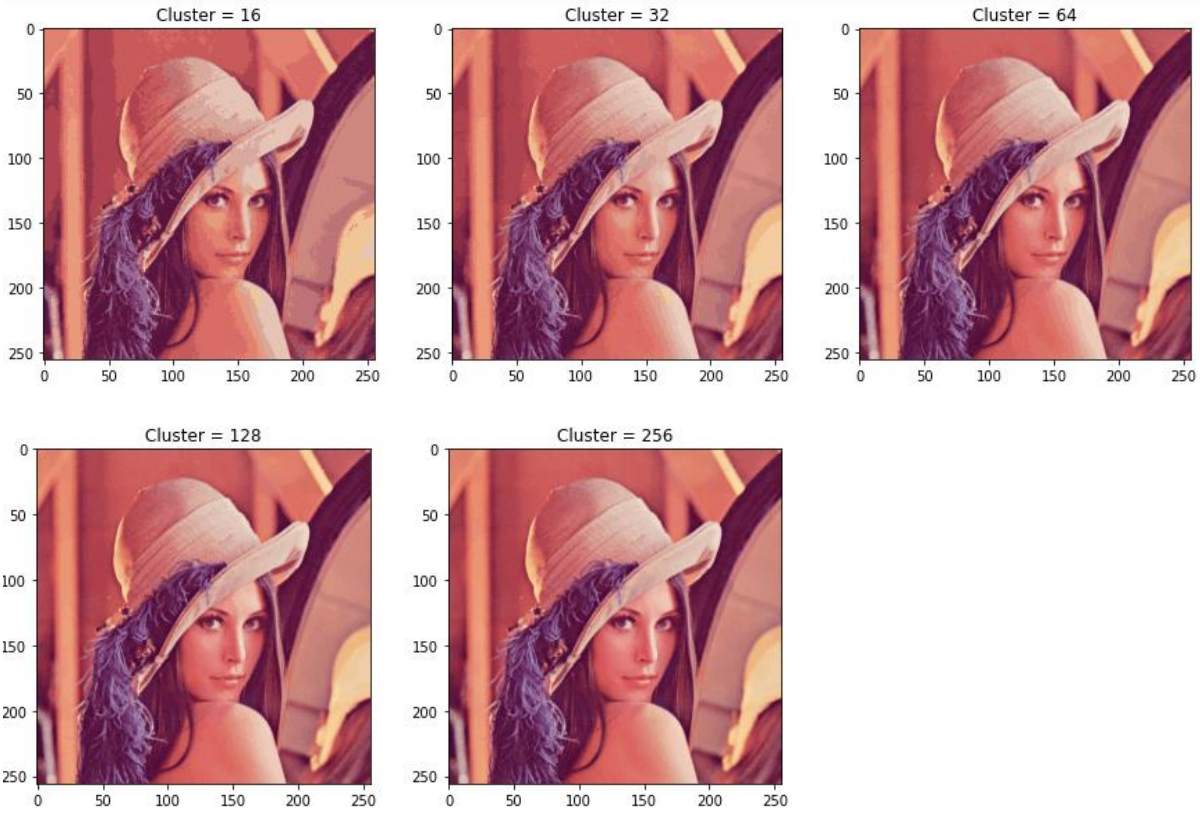

```
Compressed File Size In graffiti With Clusters 2: 17092 bytes
Compressed File Size In graffiti With Clusters 4: 32963 bytes
Compressed File Size In graffiti With Clusters 8: 61016 bytes
Compressed File Size In graffiti With Clusters 16: 78350 bytes
Compressed File Size In graffiti With Clusters 32: 96435 bytes
Compressed File Size In graffiti With Clusters 64: 111949 bytes
Compressed File Size In graffiti With Clusters 128: 125609 bytes
Compressed File Size In graffiti With Clusters 256: 137210 bytes
```

Figure 17 – Compress size of the graffiti with clusters

As you can see in the implementation detail's part, Graffiti's original size was 1074472 bytes, but when we compress it, it went down to 137210 bytes with clusters 256. Also, there was no loss of image quality with cluster 256 after compressing.

4.4 Compress Images of Lena



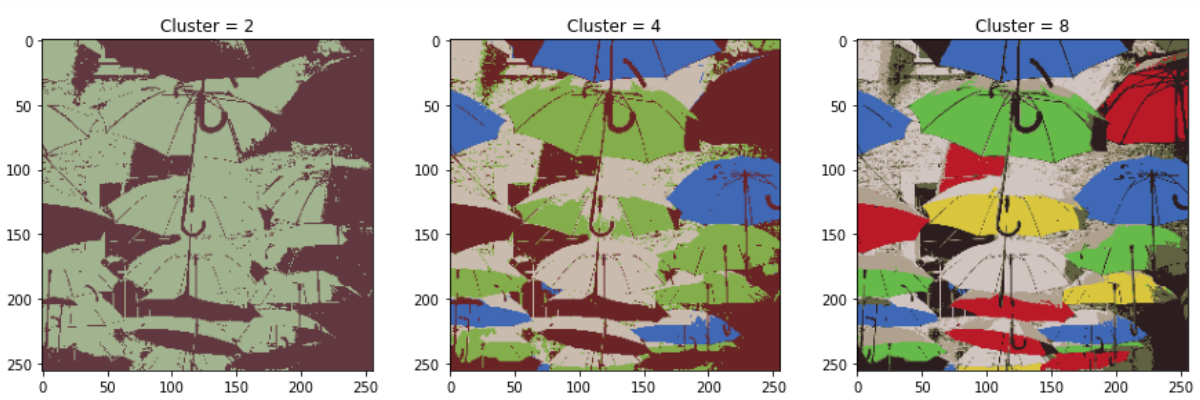


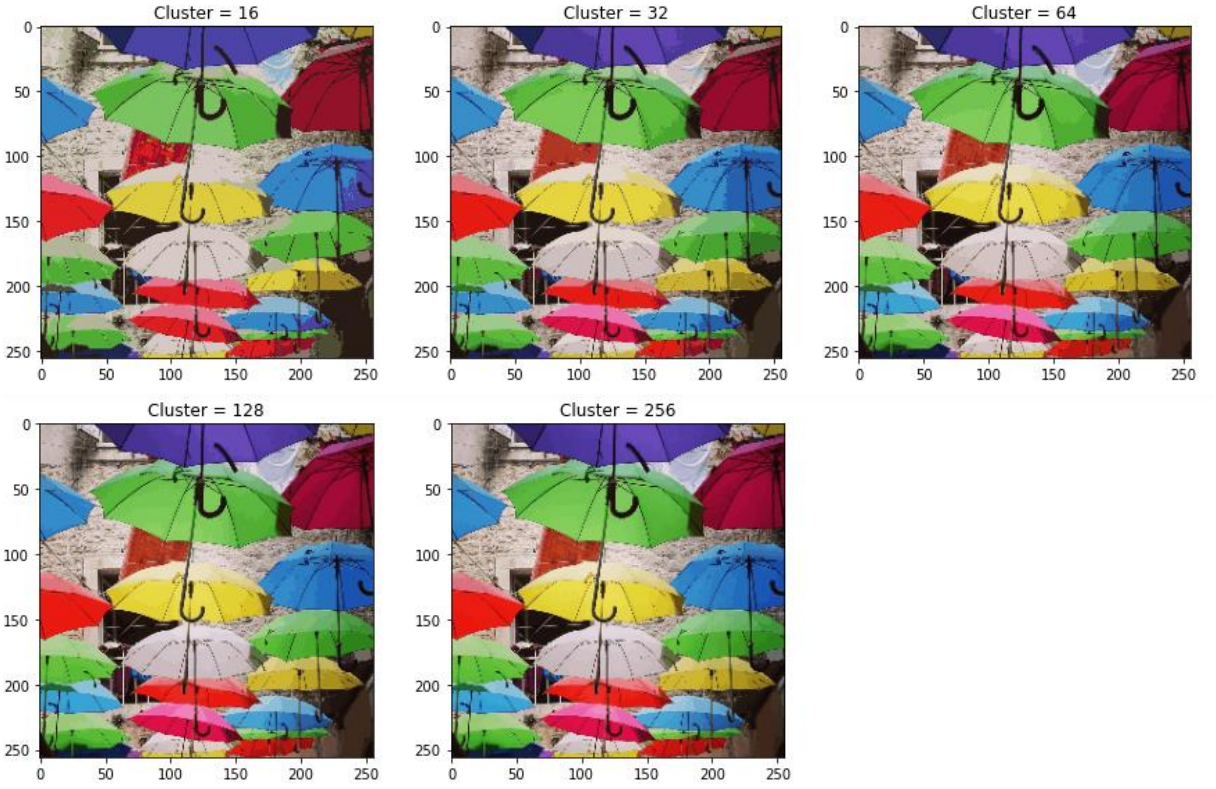

```
Compressed File Size In lena With Clusters 2: 11721 bytes
Compressed File Size In lena With Clusters 4: 27837 bytes
Compressed File Size In lena With Clusters 8: 49917 bytes
Compressed File Size In lena With Clusters 16: 66245 bytes
Compressed File Size In lena With Clusters 32: 83831 bytes
Compressed File Size In lena With Clusters 64: 100301 bytes
Compressed File Size In lena With Clusters 128: 112703 bytes
Compressed File Size In lena With Clusters 256: 121559 bytes
```

Figure 18 – Compress size of the lena with clusters

As you can see in the implementation detail's part, Lena's original size was 473777 bytes, but when we compress it, it went down to 121559 bytes with clusters 256. Also, there was no loss of image quality with cluster 256 after compressing.

4.5 Compress Images of Umbrella






```
Compressed File Size In umbrella With Clusters 2: 13353 bytes
Compressed File Size In umbrella With Clusters 4: 23133 bytes
Compressed File Size In umbrella With Clusters 8: 35954 bytes
Compressed File Size In umbrella With Clusters 16: 48067 bytes
Compressed File Size In umbrella With Clusters 32: 63023 bytes
Compressed File Size In umbrella With Clusters 64: 75514 bytes
Compressed File Size In umbrella With Clusters 128: 88476 bytes
Compressed File Size In umbrella With Clusters 256: 99545 bytes
```

Figure 19 – Compress size of the umbrella with clusters

As you can see in the implementation detail's part, Umbrella's original size was 278455 bytes, but when we compress it, it went down to 99545 bytes with clusters 256. Also, there was no loss of image quality with cluster 256 after compressing.

4.6 Metric Results

After the necessary metric calculations are made, I print the values we obtained for each image.

```
# printing the all Within Cluster Sum of Squares (WCSS) of each image
print('Baboon WCSS:', baboonWCSS)
print('Flowers WCSS:', flowersWCSS)
print('Graffiti WCSS:', graffitiWCSS)
print('Lena WCSS:', lenaWCSS)
print('Umbrella WCSS:', umbrellaWCSS)
```

```
Baboon WCSS: 4724506.3589000655
Flowers WCSS: 9332285.69065606
Graffiti WCSS: 3962399.3079191856
Lena WCSS: 1496036.2525108755
Umbrella WCSS: 4521749.490752896
```

Figure 20 – WCSS result of each image

```
# printing the all Between Cluster Sum of Squares (BCSS) of each image
print('Baboon BCSS:', baboonBCSS)
print('Flowers BCSS:', flowersBCSS)
print('Graffiti BCSS:', graffitiBCSS)
print('Lena BCSS:', lenaBCSS)
print('Umbrella BCSS:', umbrellaBCSS)
```

```
Baboon BCSS: 3699579831.6411
Flowers BCSS: 3082632911.309344
Graffiti BCSS: 2985794781.692081
Lena BCSS: 3911777006.747489
Umbrella BCSS: 3895276050.5092473
```

Figure 21 – BCSS result of each image

```
# printing the all explained variance(silhoette coeffients) of each image
print('Baboon Explained Variance:', baboonExpVar)
print('Flowers Explained Variance:', flowersExpVar)
print('Graffiti Explained Variance:', graffitiExpVar)
print('Lena Explained Variance:', lenaExpVar)
print('Umbrella Explained Variance:', umbrellaExpVar)
```

```
Baboon Explained Variance: 0.9987245901179246
Flowers Explained Variance: 0.9969817623756856
Graffiti Explained Variance: 0.998674675210047
Lena Explained Variance: 0.9996177020524578
Umbrella Explained Variance: 0.998840516938916
```

Figure 22 – Explained Variance result of each image

4.7 Pandas DataFrame

```
dataFrame
```

	Image	Number of Clusters	Fie Size in Byte	WCSS	BCSS	Explained Variance
0	new_baboon_2.png	2	File Size In Bytes: 21876	2.403698e+07	1.757506e+10	4.992839
1	new_baboon_4.png	4	File Size In Bytes: 36238	2.403698e+07	1.757506e+10	4.992839
2	new_baboon_8.png	8	File Size In Bytes: 69590	2.403698e+07	1.757506e+10	4.992839
3	new_baboon_16.png	16	File Size In Bytes: 94811	2.403698e+07	1.757506e+10	4.992839
4	new_baboon_32.png	32	File Size In Bytes: 118244	2.403698e+07	1.757506e+10	4.992839
5	new_baboon_64.png	64	File Size In Bytes: 135288	2.403698e+07	1.757506e+10	4.992839
6	new_baboon_128.png	128	File Size In Bytes: 146075	2.403698e+07	1.757506e+10	4.992839
7	new_baboon_256.png	256	File Size In Bytes: 152326	2.403698e+07	1.757506e+10	4.992839
8	new_flowers_2.png	2	File Size In Bytes: 16905	2.403698e+07	1.757506e+10	4.992839
9	new_flowers_4.png	4	File Size In Bytes: 31789	2.403698e+07	1.757506e+10	4.992839
10	new_flowers_8.png	8	File Size In Bytes: 56345	2.403698e+07	1.757506e+10	4.992839
11	new_flowers_16.png	16	File Size In Bytes: 77389	2.403698e+07	1.757506e+10	4.992839
12	new_flowers_32.png	32	File Size In Bytes: 100253	2.403698e+07	1.757506e+10	4.992839
13	new_flowers_64.png	64	File Size In Bytes: 115299	2.403698e+07	1.757506e+10	4.992839
14	new_flowers_128.png	128	File Size In Bytes: 127715	2.403698e+07	1.757506e+10	4.992839
15	new_flowers_256.png	256	File Size In Bytes: 136400	2.403698e+07	1.757506e+10	4.992839
16	new_graffiti_2.png	2	File Size In Bytes: 17092	2.403698e+07	1.757506e+10	4.992839
17	new_graffiti_4.png	4	File Size In Bytes: 32963	2.403698e+07	1.757506e+10	4.992839
18	new_graffiti_8.png	8	File Size In Bytes: 61016	2.403698e+07	1.757506e+10	4.992839
19	new_graffiti_16.png	16	File Size In Bytes: 78350	2.403698e+07	1.757506e+10	4.992839
20	new_graffiti_32.png	32	File Size In Bytes: 96435	2.403698e+07	1.757506e+10	4.992839
21	new_graffiti_64.png	64	File Size In Bytes: 111949	2.403698e+07	1.757506e+10	4.992839
22	new_graffiti_128.png	128	File Size In Bytes: 125609	2.403698e+07	1.757506e+10	4.992839
23	new_graffiti_256.png	256	File Size In Bytes: 137210	2.403698e+07	1.757506e+10	4.992839
24	new_lena_2.png	2	File Size In Bytes: 11721	2.403698e+07	1.757506e+10	4.992839
25	new_lena_4.png	4	File Size In Bytes: 27837	2.403698e+07	1.757506e+10	4.992839
26	new_lena_8.png	8	File Size In Bytes: 49917	2.403698e+07	1.757506e+10	4.992839
27	new_lena_16.png	16	File Size In Bytes: 66245	2.403698e+07	1.757506e+10	4.992839
28	new_lena_32.png	32	File Size In Bytes: 83831	2.403698e+07	1.757506e+10	4.992839
29	new_lena_64.png	64	File Size In Bytes: 100301	2.403698e+07	1.757506e+10	4.992839
30	new_lena_128.png	128	File Size In Bytes: 112703	2.403698e+07	1.757506e+10	4.992839

31	new_lena_256.png	256	File Size In Bytes: 121559	2.403698e+07	1.757506e+10	4.992839
32	new_umbrella_2.png	2	File Size In Bytes: 13353	2.403698e+07	1.757506e+10	4.992839
33	new_umbrella_4.png	4	File Size In Bytes: 23133	2.403698e+07	1.757506e+10	4.992839
34	new_umbrella_8.png	8	File Size In Bytes: 35954	2.403698e+07	1.757506e+10	4.992839
35	new_umbrella_16.png	16	File Size In Bytes: 48067	2.403698e+07	1.757506e+10	4.992839
36	new_umbrella_32.png	32	File Size In Bytes: 63023	2.403698e+07	1.757506e+10	4.992839
37	new_umbrella_64.png	64	File Size In Bytes: 75514	2.403698e+07	1.757506e+10	4.992839
38	new_umbrella_128.png	128	File Size In Bytes: 88476	2.403698e+07	1.757506e+10	4.992839
39	new_umbrella_256.png	256	File Size In Bytes: 99545	2.403698e+07	1.757506e+10	4.992839

5 Conclusion

Five different images which are baboon, flowers, graffiti, lena, umbrella provided to us in this assignment were compressed by pre-processing and then using the K-Means Clustering model. In the basic working logic of the algorithm, it performs by quantizing images by minimizing the WCSS score, so that it can represent all image colors with 8 different photos using only 2, 4, 8, 16, 32, 64, 128 256 unique colors. After compressing the images, we encode the mathematical formula of the metrics and measure each image in these metrics and note the results in this report.