Implementation 3: Image Compression

In this section we want to compress an image with an algorithm named “Quantizing color information” using LUT which is abbreviation of Look Up Table. The approach is to cluster all the present colors in the image into 16-256 batches and pick the centroid of each cluster as that cluster representative and put the RGB value of this centroid in LUT. After that replace all the pixel’s RGB value with the centroid of it’s cluster RGB in the picture.

For Implementation of this method we use the k-means method that we have implemented in the first section as clustering method. Also we know that the image is a 3D array where the first two dimensions are height and width of the image and the third dimension is the RGB value of each pixel of image. So first we reshape this 3D array into a 2D array using numpy.reshape function.

We define a method named **image_compressor** which will get the image and convert it to 2D numpy array and pass it through k-means method and get the labels and centorids. After that it will replace the RGB values and render the image by imshow() method.

We have done some changes in k-means method:

1- Set a limit for k-means convergence and if it hasn’t been converged in that limit number of iterations the algorithm will stop and return the current centroids.

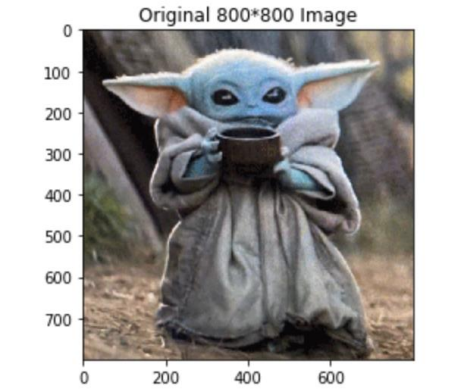
2- We have add an input argument named “init_method” which indicates that if we want to use k-means++ algorithm for centroids initiation or not. Because it will take a lot of time if we want to initiate the centroids with this algorithm when we are going to have 256 centroids, hence we choose initiator centroids through random choose among the data points. For init_method=”k” the initiation method will be k-means++ and for init_method=”r” the initiation is random choose.

3- We have removed the plot and other details like clustering error measuring which were not important for this image compression application.

In the following we will test the functionality of this method for small and original input images and will show the results of compression through k=16 and k=256 for each image.

Before clustering we check how many colors are there in the pictures by using numpy.unique method on images arrays.

```
img = image.imread(file_name1)
imgSmall = image.imread(file_name2)
plt.title("Original 800*800 Image")
plt.imshow(img)
plt.show()
plt.title("Original 200*200 Image")
plt.imshow(imgSmall)
plt.show()
```



These are loaded images into the program by imread method. Now we're going to check how many colors are in these two images.

```
img_reshaped = img.reshape((img.shape[0]*img.shape[1], img.shape[2]))
colors_in_img = np.unique(img_reshaped, return_counts=False)
print("There are ",len(colors_in_img)," different colors in img")

imgSmall_reshaped = imgSmall.reshape((imgSmall.shape[0]*imgSmall.shape[1], imgSmall.shape[2]))
colors_in_imgSmall = np.unique(imgSmall_reshaped, return_counts=False)
print("There are ",len(colors_in_imgSmall)," different colors in imgSmall")
```

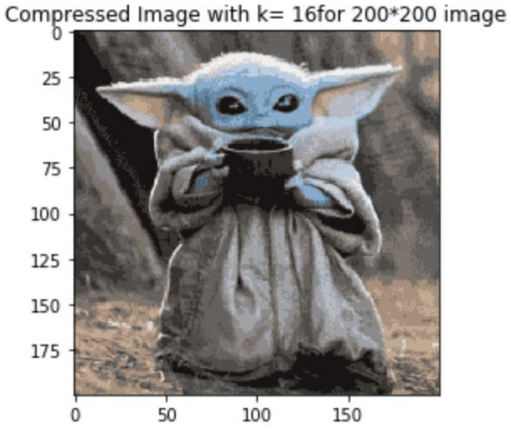
```
There are  198  different colors in img
There are  195  different colors in imgSmall
```

As you can see there are 198 different colors in big image and 195 different colors in small image. hence, reducing colors into 16 by clustering them in 16 batches will reduce the size of image but clustering colors in 256 clusters will not decrease the image size. We will see this after saving images and compare the compressed ones to the original ones.

In the following figures we will see the compression method have been run on images with k=16 and k= 256.

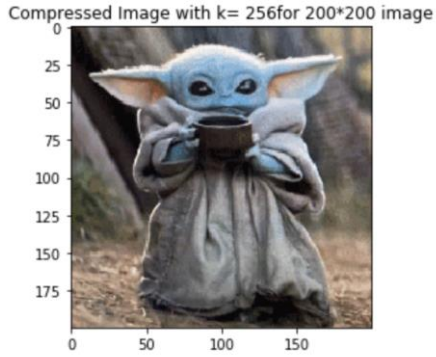
-new_image = image_compressor(imgSmall, 16, kmeans_init_method="k")

```
Kmeans starts
kmeans will stop when convergence occure
Iteration number:  0
Iteration number:  1
Iteration number:  2
Iteration number:  3
Iteration number:  4
Iteration number:  5
Iteration number:  6
(40000, 3)
(200, 200, 3)
```



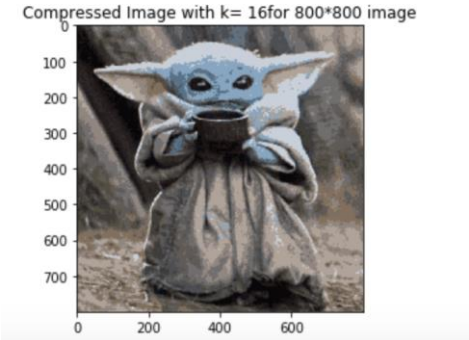
-new_image = image_compressor(imgSmall, 256, kmeans_init_method="r")

```
Kmeans starts
kmeans will stop when convergence occure
Iteration number:  0
Iteration number:  1
Iteration number:  2
Iteration number:  3
Iteration number:  4
(40000, 3)
(200, 200, 3)
```



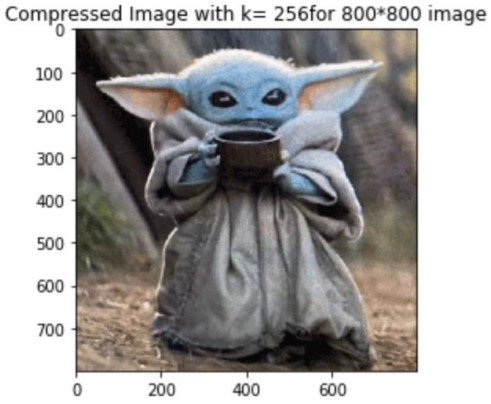
-new_image = image_compressor(img, 16, kmeans_init_method="r")

```
kmeans will stop when convergence occure
Iteration number:  0
Iteration number:  1
Iteration number:  2
Iteration number:  3
Iteration number:  4
Iteration number:  5
Iteration number:  6
Iteration number:  7
Iteration number:  8
Iteration number:  9
Iteration number:  10
Iteration number:  11
Iteration number:  12
Iteration number:  13
Iteration number:  14
Iteration number:  15
Iteration number:  16
Iteration number:  17
Iteration number:  18
(640000, 3)
(800, 800, 3)
```



-new_image = image_compressor(img, 256, kmeans_init_method="r")

```
Kmeans starts
kmeans will stop when convergence occure
Iteration number:  0
Iteration number:  1
Iteration number:  2
Iteration number:  3
Iteration number:  4
(640000, 3)
(800, 800, 3)
```



As you can see the number of iterations for clustering all the data point's decreases when we are using 256 centroids to cluster the colors because there are a lot of overlapping centroids on the same colors (Cause there are just 195 colors in the picture).

Result images are saved and **stored in folder named "compression_results"**. In the image below you can see the information of saved images in comparison with the original image. the left one is small image and right one is the big image and the bottom left result is k=16 compression and bottom right result is k=256 compression.