Dhruv Parmar and Wyatt Bowen

Image Compression Using K-Means Clustering

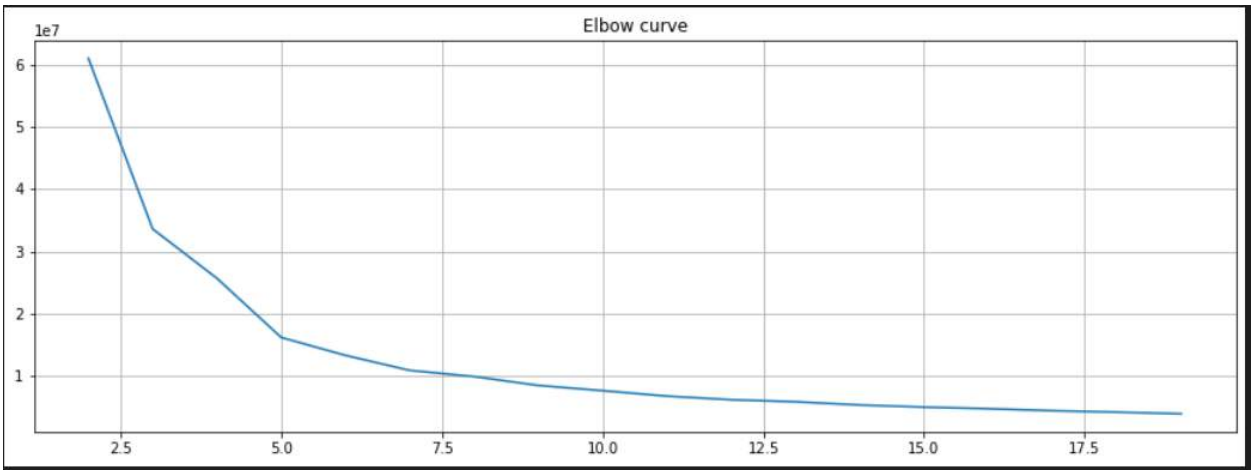
For the final project we used the K-means Clustering algorithm to compress many images from different types of datasets. Image compression is the reduction of the image's file size. This can be achieved through reducing the resolution of the image, or reducing the number of color channels the image uses (posterization). We compressed images by reducing the number of colors in the image and kept the resolution the same. The number of colors kept in the compressed image is the optimal K value which is determined by running the K-means Clustering algorithm with a range of K values and comparing the variance within the clusters for each iteration.

We have four directories (Landscapes, Animals, Gradients, and Simple) with multiple images in each directory that we run our algorithm on. For every image that we compress, we systematically random sampled 10% of the total number of pixels in the image. We decided to sample a fraction of the number of pixels instead of all the pixels because we found that this method is about 90% faster in runtime while losing only about 10% sharpness. The reason why we systematically sample is that we want to have a true representation of the complete image. We added some randomness to the systematic sampling because we want to avoid sampling pixels from the same rows or columns.

For each pixel selected we record its RGB value onto a 3D space of red, green, and blue. At this point we run the K-means Clustering algorithm with a range of K values. We find the variation within each of the groups that are produced. We then iterate k by one over and over until the variation within groups for (i-1)th and ith iteration reaches a threshold. The threshold we kept is

$$(variation\ in\ the\ i\ th\ iteration)\ /\ (variation\ in\ the\ (i\ -\ 1)th\ iteration) \geq 0.995$$

Originally, we wanted to use the elbow method to determine the best k values to compress our images. However, when we generated the graph of intra-cluster variation, the elbow point was at a k value that is too low for images.



As you can see from the graph above, if we were to select the elbow point of this graph as our k value, then the k values would be 5. That means the compressed image we produce would only have 5 colors. Instead of picking the elbow of this graph, we pick a different point of this graph to stop iterating k on. That point is when the value of variation of some k=n divided by the variation when k=n-1 is greater than 0.995. This means that the variation hardly changes as we increase k, meaning that the compressed image looks fairly similar to the original.We want to minimize the file size, yet make the image look good compared to the uncompressed version.

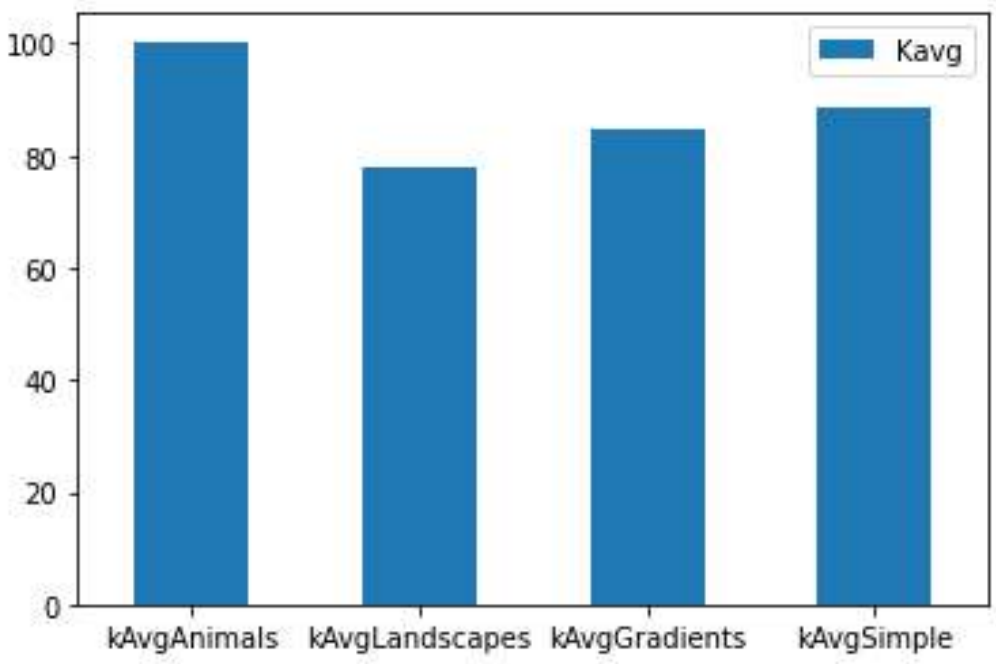
We tried different thresholds but we found that 0.995 was a sweet spot to stop the K means at. The sweet spot is when we are producing an image that is maximum compressed while still minimizing quality reduction. So we chose 0.995 as our threshold and nothing else that has a higher value than k value produced by the elbow point. Here are the results for the thresholds 0.98 and 0.995 respectively to show the difference in the quality:

when threshold is 0.98

when threshold is 0.995

We ran our image compression code on different types of images. High and low resolution images. Images with many colors, and images with not that many colors. We wanted our method to work on all of these types of images and not waste time having too many clusters for the image given. That's why we decided to start iterating k from a low value of 2.

We used four types of images: Animals, Landscapes, Gradients, and Simple. We wanted to ensure that our algorithm is sufficient enough to perform equally good for different types of images. We also wanted to compare the average optimal K value we get for each type. Here is a bar graph that showcases the differences:



Shown below are some examples of our imputed images followed by our compressed version of the image. Next to each image is its file size.

Input:

1.68 MB

Output:

957 KB

Input:

297 KB

Output:

134 KB

Input:

1.12 MB

Output:

507 KB

Input:

8.12 MB

Output:

6.66 MB

Input:

3.75 MB

Output:

1.47 MB

Input:

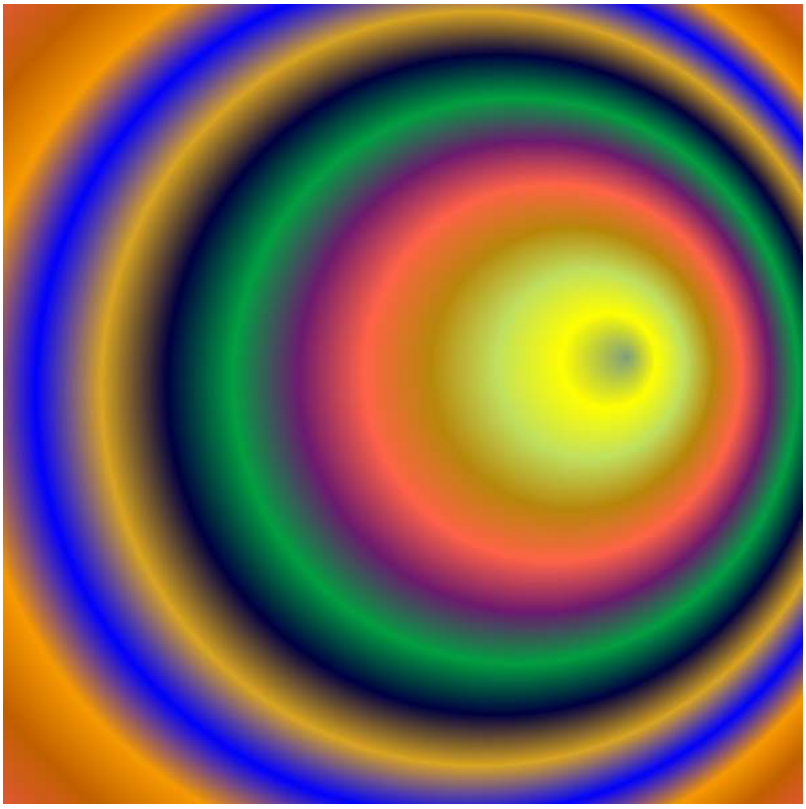
6.16MB

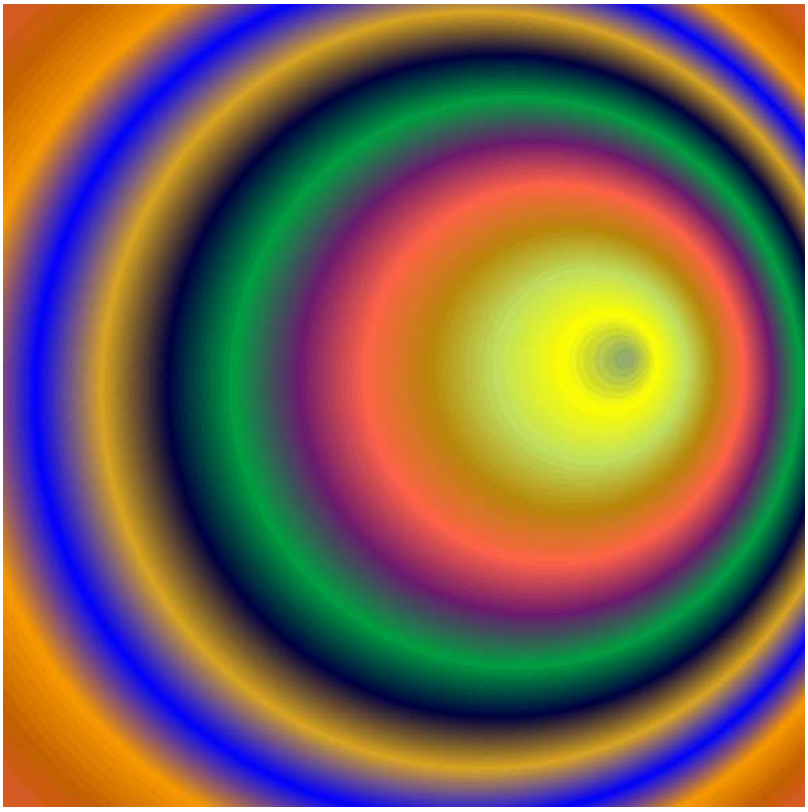
Output:

4.03 MB

Input:  1.24 MB

Output:  601 KB

Input:  227 KB

Output:  59.1 KB

Input:

71.8 KB

Output:

51.2 KB