1 Image Compression using K-means

1.1 Data Visualisation

Figure 1: given RGB image

- Image size is (512 * 512) in terms of pixels
- Each pixel is a 3D features corresponding to R, G and B
- So converting this into a numpy array so easy operations we end up with (262144 * 3) as our input data

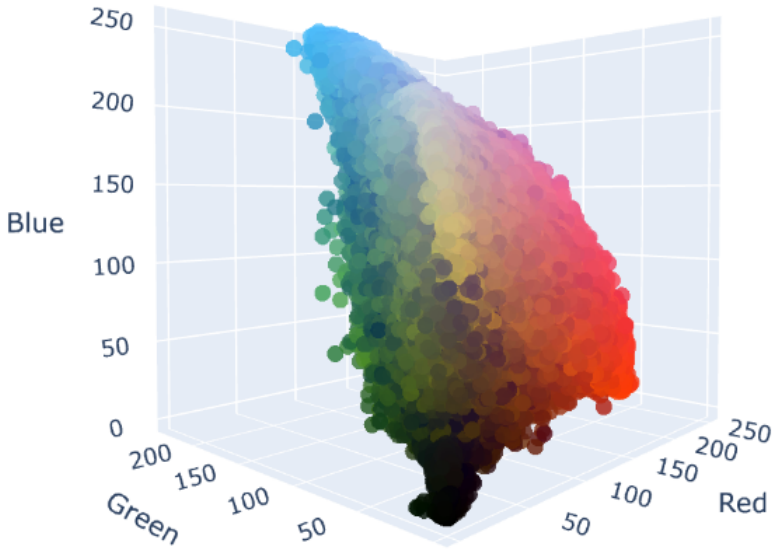


Figure 2: Pixel Values in RGB Color Space in 3D

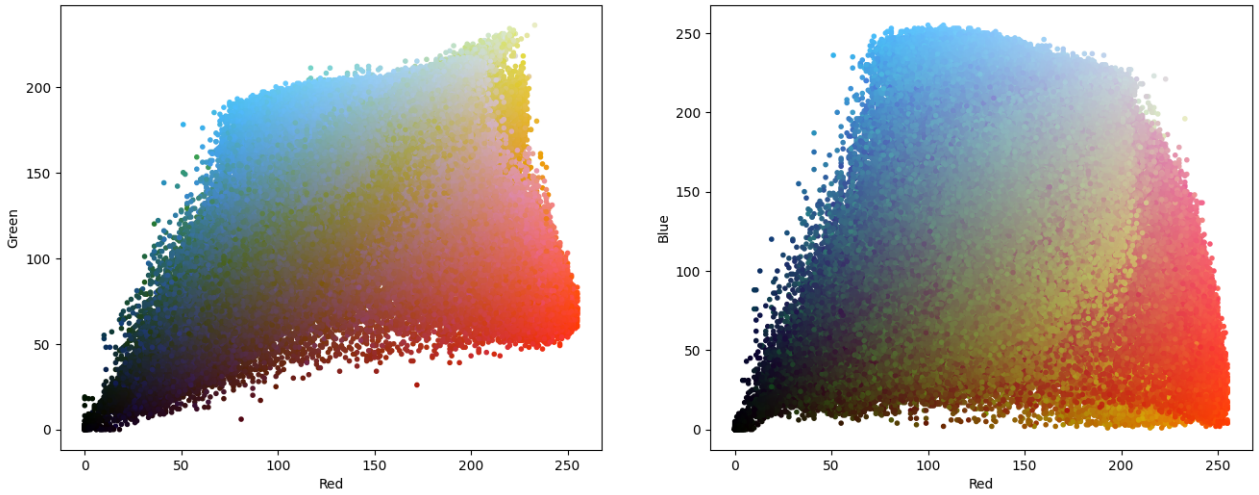


Figure 3: Input color space: 16 million possible colors in 2D

1.2 mykmeans from scratch

1. function named mykmeans takes two arguments: data_matrix (which represents the data points) and k (which represents the number of clusters)
2. initialize an empty list called means. This list will hold the centroids of the clusters.
3. select a random data point from imput data and add it to the list means. This initializes the first centroid randomly.
4. **K++ init method :** runs until the number of centroids in the means list reaches the desired number k
5. calculate the squared Euclidean distance between each data point in input data and the closest centroid in means
6. compute the probabilities of selecting each data point as the next centroid based on their distances.
7. generate a random number between 0 and 1 and based on that select the point from data input corresponding to that cumulative sum of probabilities ans add it to means
8. Now that random cluster means are generated its time for convergence
9. assign each point its cluster based on how closest it is
10. re-calculate the cluster means based on new classes
11. if the means change, repeat the process till they don't change and finally return them

1.3 Image Compression

Figure 4: Compressed Image k = 2

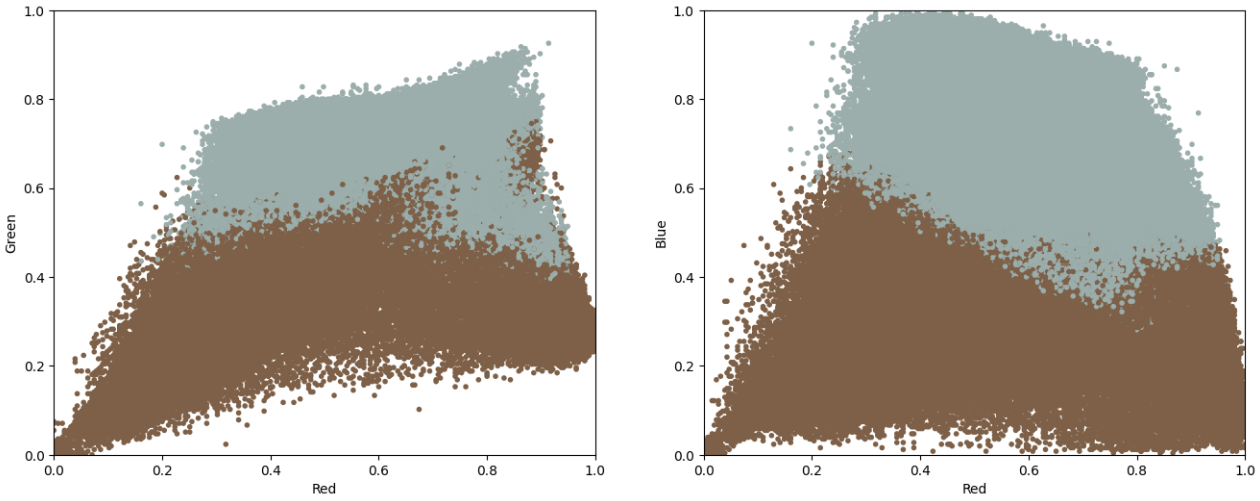


Figure 5: Reduced color space: 2 colors in 2D

Figure 6: Compressed Image k = 6

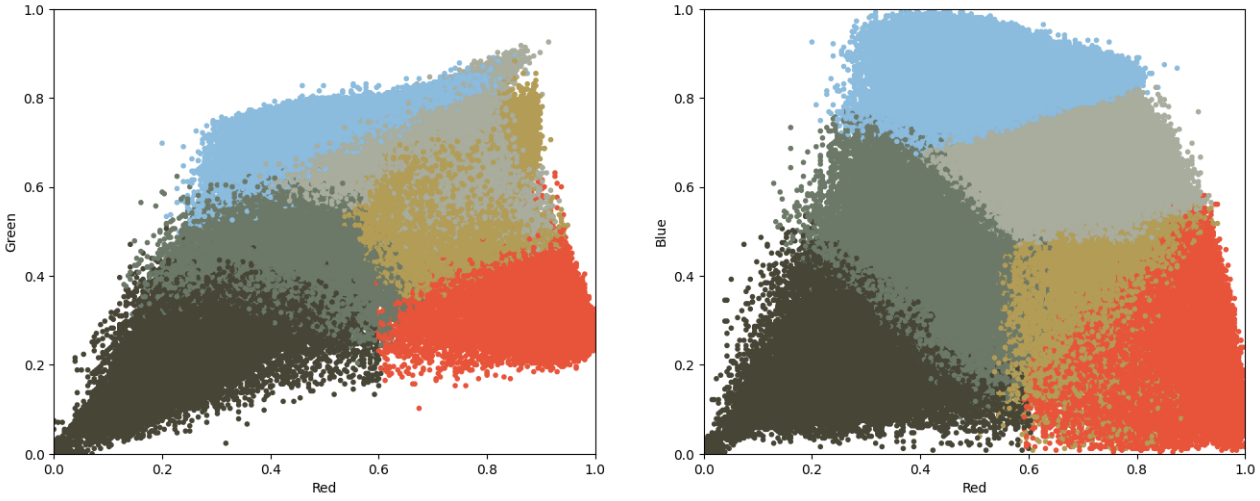


Figure 7: Reduced color space: 6 colors in 2D

Figure 8: Compressed Image k = 10

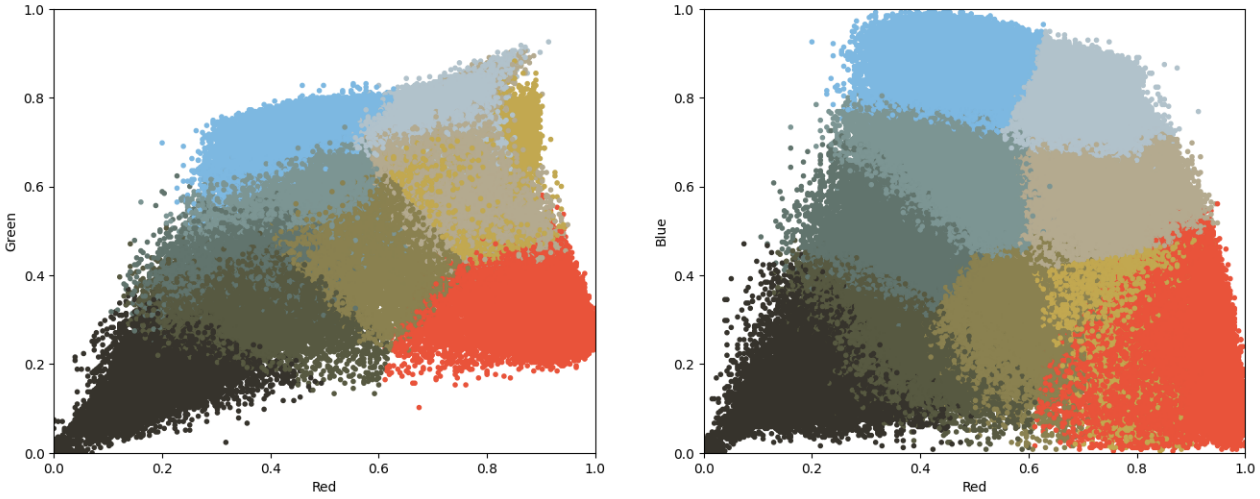


Figure 9: Reduced color space: 10 colors in 2D

1.4 Image Compression : sklearn

Figure 10: Compressed Image k = 2

Figure 11: Compressed Image k = 6

Figure 12: Compressed Image k = 10

1.5 Spatial coherence

- Pixels that are nearby in the original image are more likely to be assigned to the same cluster
- So, nearby here will be considered as squared peripheral boundary around one pixel

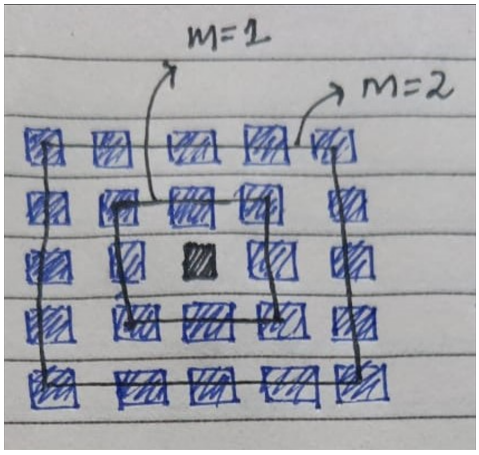


Figure 13: general case

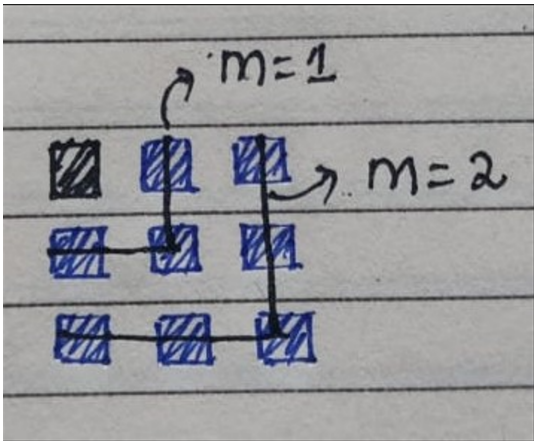


Figure 14: edge case

- Now including the pixel and surrounding neighbor pixels based on m will considered to determine the color of center pixel
- We first get note of the unique colors and their count. Based on that we assign the pixel color of one which has maximum count

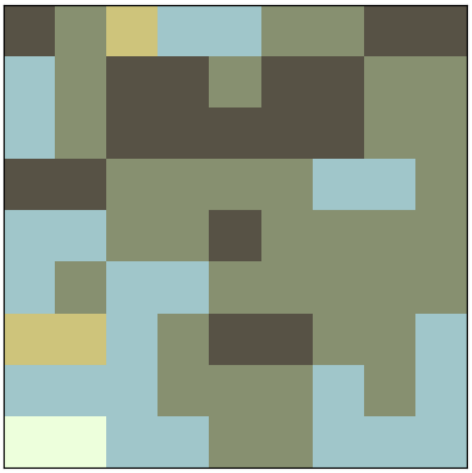


Figure 15: before Spatial coherence

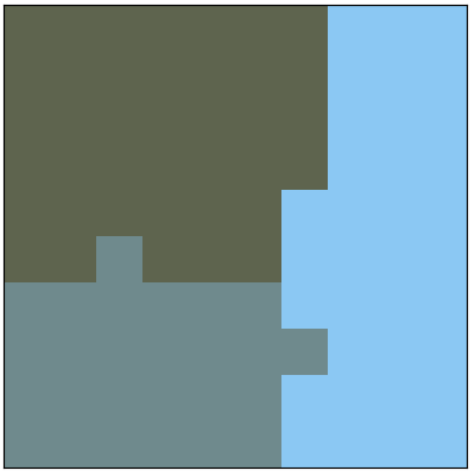
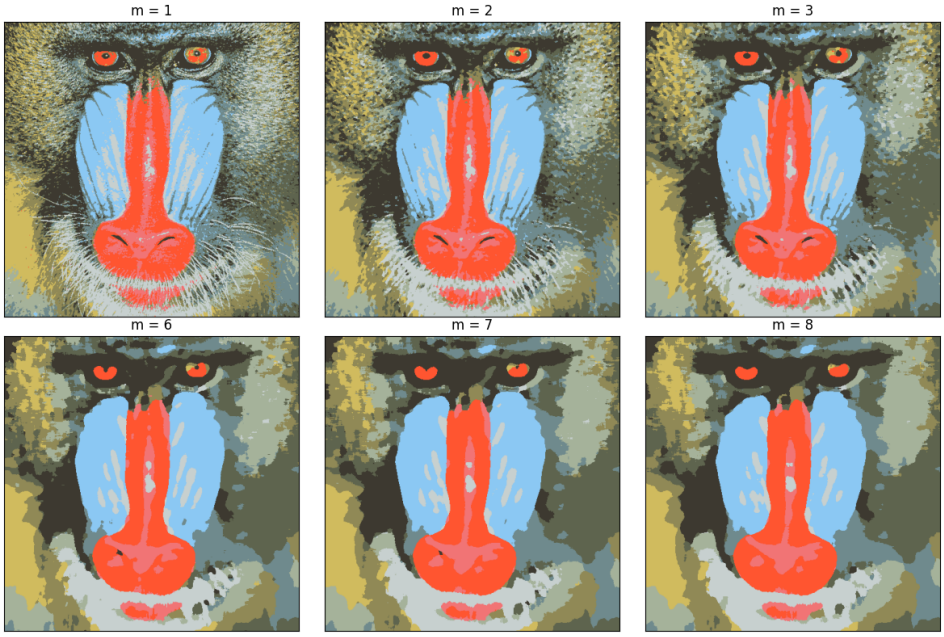


Figure 16: after Spatial coherence

Image after Spatial coherence for different m values



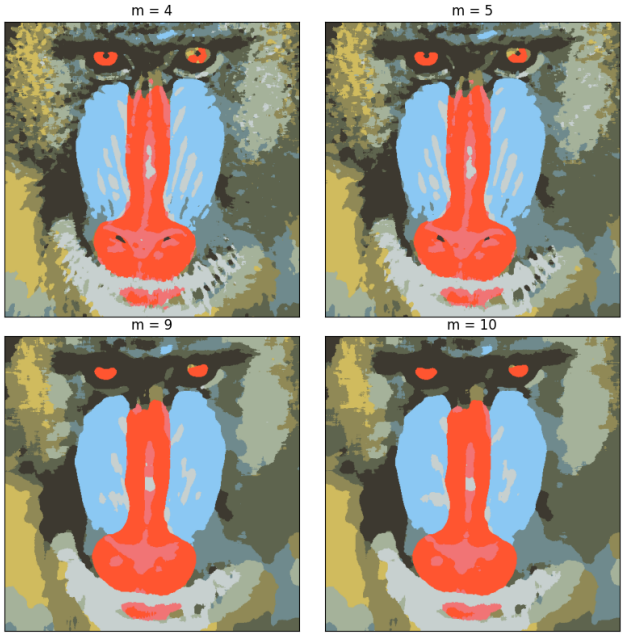

m = 4
m = 5
m = 9
m = 10

Figure 17: before Spatial coherence

Figure 18: after Spatial coherence m = 10