Color Quantization Using K-Means Clustering

Syed Muhammad Mehran 24L-8014

Abstract

This report presents an implementation of the K-Means clustering algorithm from scratch in Python to reduce color information in a colored image. The RGB image is compressed by grouping similar pixel colors into clusters. The resulting image replaces each pixel with the mean color of its cluster. The algorithm is applied for different values of clusters $K = 2$ to 10, 15, and 20, and results are analyzed and compared.

1 Introduction

Color quantization is a technique used to reduce the number of distinct colors in an image. This is especially useful in image compression and artistic stylization. K-Means clustering is an effective method for this task as it groups pixels with similar RGB values and replaces them with the centroid (mean) color of their group.

2 K-Means Algorithm

The K-Means algorithm used in this project includes the following steps:

1. Randomly initialize K cluster centroids by selecting K random pixels from the image.
2. Assign each pixel to the nearest centroid using Euclidean distance in RGB space.
3. Update each centroid by computing the mean RGB values of all pixels assigned to it.
4. Repeat steps 2 and 3 until the assignments stop changing (convergence) or a maximum number of iterations is reached.

The image is reconstructed by replacing each pixel with the RGB value of its assigned cluster centroid.

3 Results and Discussion

The algorithm was applied to a color image with different values of K. Below are visual results and observations for each case:

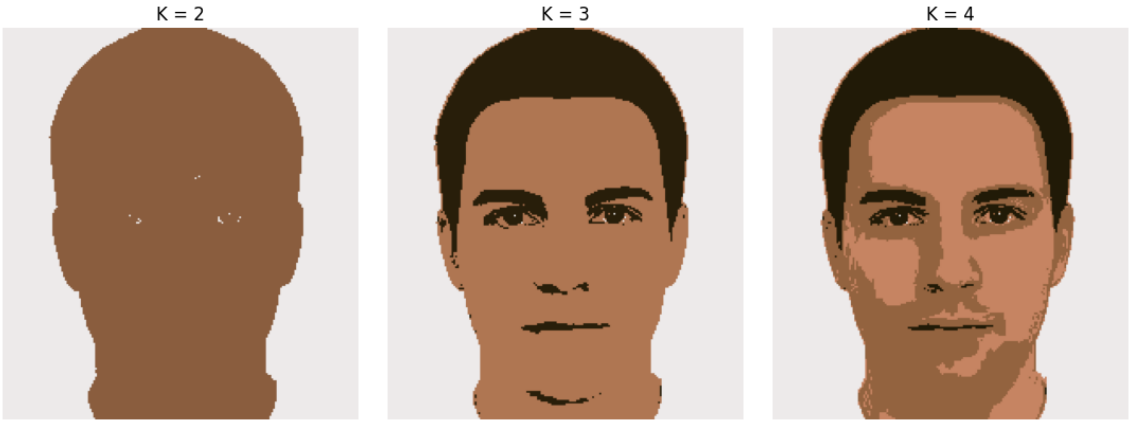

Figure 1: Quantized images for $K = 2, 3, 4$

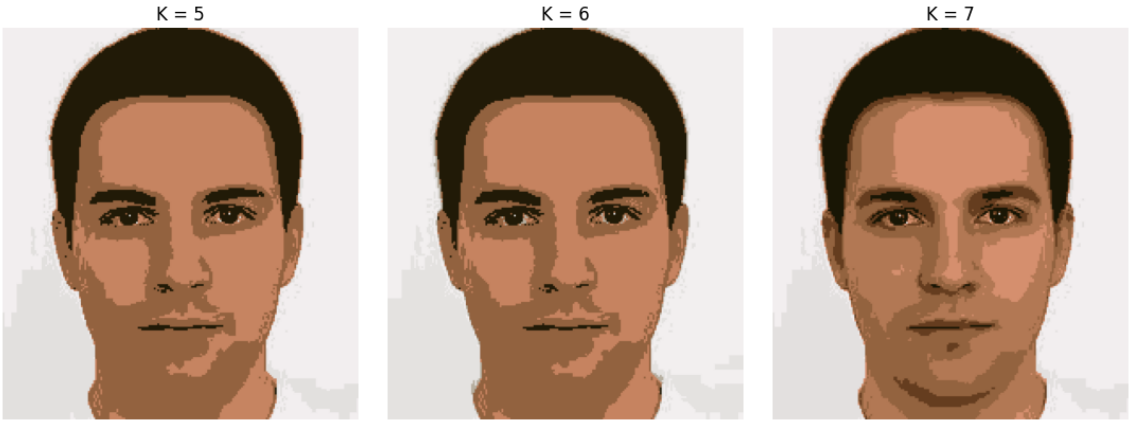

Figure 2: Quantized images for $K = 5, 6, 7$

Figure 3: Quantized images for $K = 8, 9, 10$

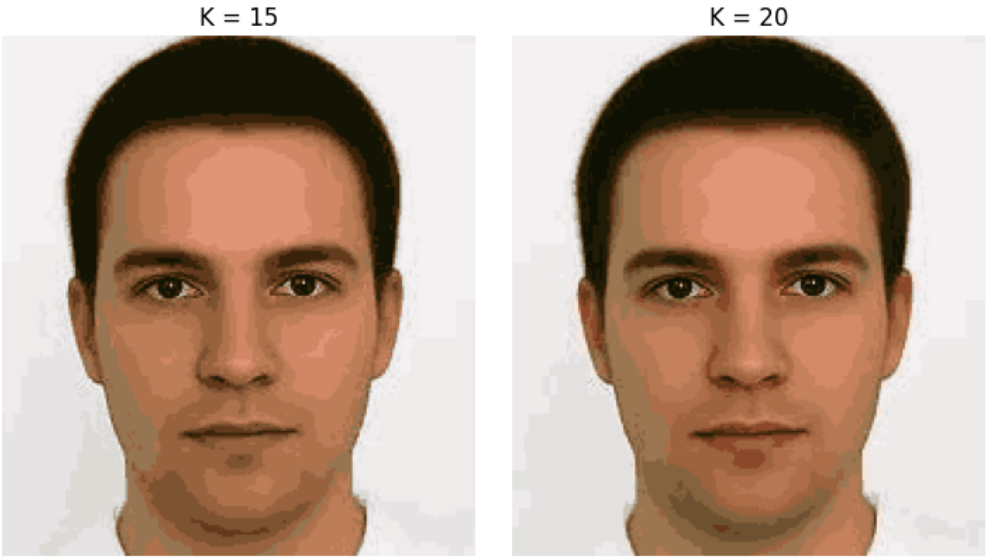

Figure 4: Quantized images for $K = 15, 20$

Observations

- **K = 2 to 4:** The image appears heavily stylized with major color blocks. Details are lost, but dominant color regions are preserved. Good for extreme compression.
- **K = 5 to 7:** More subtle color variations start to appear. The face and background start to regain some structure.
- **K = 8 to 10:** The image begins to resemble the original closely, while still reducing storage. This is a good balance between compression and quality.
- **K = 15 and 20:** The image quality is very close to the original. Further increases in K offer diminishing returns visually but increase computational cost.

4 Conclusion

The K-Means clustering algorithm effectively reduces the number of colors in an image while preserving important visual features. For most practical purposes, values of K between 8 and 10 provide a good trade-off between compression and quality. Higher values of K yield better visual results but increase computation time and memory usage.

Appendix: Python Code Summary

The following class was implemented for K-Means clustering:

```
class k_means:
    def __init__(self, no_of_clusters, data):
```

```
    ...
def euclidean_distance(self, arr1, arr2):
    ...
def assign_clusters(self):
    ...
def update_centroids(self):
    ...
def k_means(self, max_iter=100):
    ...
def generate_image(self):
    ...
```

The algorithm was applied using a loop for $K = 2$ to 10, 15, and 20. Visualizations were produced using matplotlib.