ClusterCompress: K-Means Image Clustering

Amirthalingam Rajasundar

April 17, 2025

Abstract

ClusterCompress uses K-Means clustering to reduce the number of distinct colors in an image. This could significantly lower the file size requirements when an appropriate file encoding scheme is used. The K-Means algorithm is implemented from scratch using NumPy, and the project is dockerized for portability.

Features

- K-Means clustering implemented from scratch (no external ML libraries).
- Supports 512 x 512 x 3 images as input.
- Customizable number of clusters (K).
- Dockerized for simple build and execution.

How It Works

1. Load and normalize (range [0,1]) the input image.
2. Reshape image into a 2D array of RGB pixels.
3. Randomly select a pixel as the first cluster center.
4. For the remaining k-1 cluster centers, compute the distance of each point from the existing centers. Then, compute the square of the distances and normalize. Use the normalized value as the probability for selecting the next cluster center. This ensures that the next cluster center is chosen in a way such that it is far from the existing cluster centers. This is called as kmeans++ initialization.
5. For each point, compute the distance between the point and every other cluster centers. Assign the point to closest cluster center.
6. Update the cluster centers by taking the mean of all points assigned to the cluster.

7. Check if the cluster centers have changed and the max iteration is not reached. Go to step 5 if true. Else, stop.

8. Replace each pixel with the cluster center, reshape and save the image.

9. Compute and print the Mean Squared Error between the original image and the clustered image.

Project Structure

```
ClusterCompress/
          Dockerfile
          requirements.txt
          .dockerignore
          .gitignore
          main.py
          model.py
          utils.py
          image/
                input_image.jpg
          README.md
```

Installation

Run with Docker

Build the Docker image:

```
docker build -t clustercompress .
```

Run the container:

```
docker run -it --rm -v $(pwd):/app clustercompress /bin/bash
```

Run the script:

```
python main.py --image ./image/input_image.jpg --num_clusters 2
```

Notes:

- Input image must be saved as `image/input`$_{image.jpg.Output image will be saved as}$`image_clustered_<k>.jp`

Command-Line Arguments

- `--num_clusters (int)`: Number of clusters (e.g., 8, 32, 64).
- `--image (str)`: Path to input image (default: `image/input_image.jpg`).

Results

K	MSE
2	0.03206917602224996
5	0.009577312447572142
10	0.0048920057823762285
20	0.0026909937308160045
50	0.0012847510149189027

Table 1: Clustering results with different values of K

(a) Original Image

(b) K = 2

Figure 1: Original image and Clustered image K = 2

(a) K = 5

(b) K = 10

Figure 2: Clustered images with K = 5 and K = 10

(a) K = 20

(b) K = 50

Figure 3: Clustered images with K = 20 and K = 50

Error Metric

We use Mean Squared Error (MSE) to evaluate error:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(x_i - \hat{x}_i)^2$$

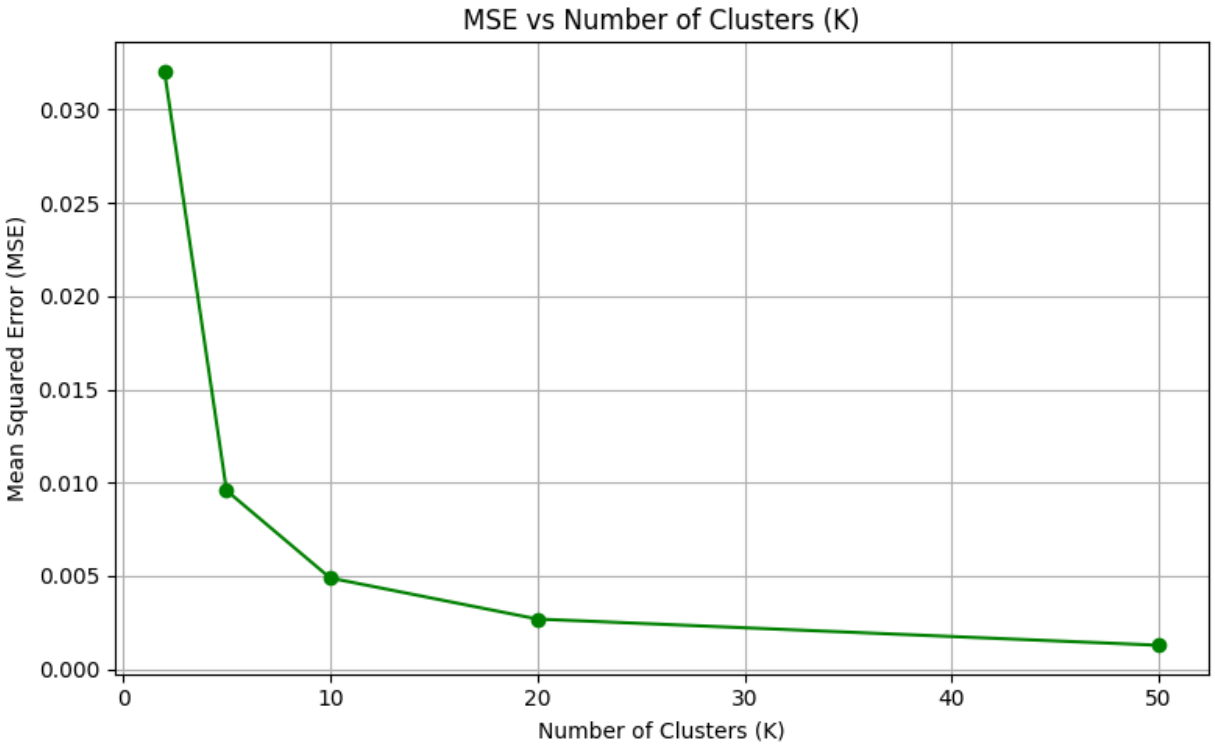

Figure 4: MSE vs k