VNUHCM – UNIVERSITY OF SCIENCE

FACULTY OF INFORMATION TECHNOLOGY

....

Applied Mathematics and Statistics for Information Technology

PROJECT 01: COLOR COMPRESSION

Prepared by:

Dinh Xuan Khuong - 23127398

Instructor:

Nguyen Ngoc Toan

Tran Ha Son

June 2025

Contents

I. Introduction

1. About the project

Nowadays, with the rapid development of technology and digital content, images have become more detailed and complex. As a result, storage and bandwidth requirements are becoming increasingly demanding. One of the key techniques to address this challenge is **image compression**, which reduces the amount of data required to represent an image without significantly compromising its visual quality.

In this project, I explore an effective and widely used clustering algorithm in machine learning, the **K-means algorithm**, to perform image compression by reducing the number of unique colors in an image. The project applies K-means clustering with different values of K (specifically K = 3, 5, 7, and more) to compress the image and analyze the trade-off between compression ratio and visual fidelity.

2. Input and output

A. Input

An RGB image (in common formats such as PNG or JPEG). Each pixel in the image is represented by a 3-dimensional vector corresponding to the Red, Green, and Blue color intensities.

B. Output:

A compressed version of the original image where the number of unique colors is reduced to exactly K. The compressed image maintains the general appearance of the original image but with fewer colors, which results in lower storage requirements.

C. Example Outputs:

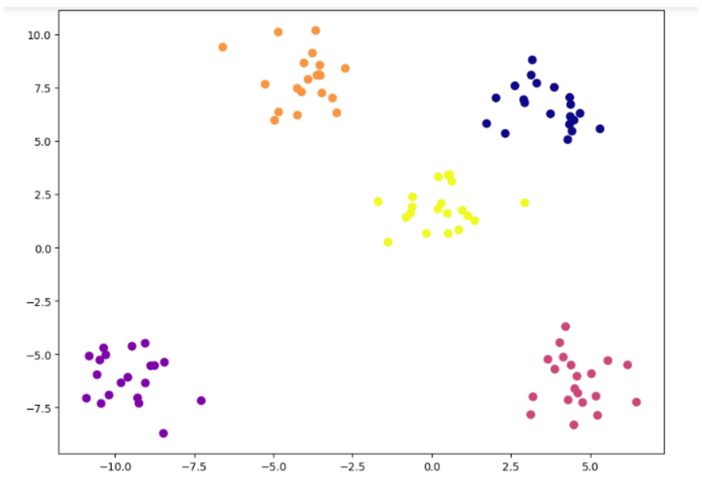
- Image compressed using K = 3 colors
- Image compressed using K = 5 colors
- Image compressed using K = 7 colors
- And more.

3. Primary goals

- To apply the K-means clustering algorithm to compress images.
- To observe the impact of different values of K on the visual quality and compression ratio of the image.
- To understand the trade-off between image quality and compression efficiency.
- To visualize and compare the results of different K values.
- To gain practical experience with machine learning algorithm – k-means.

4. Idea

After reading some documents about K-means and watching a few tutorial videos on YouTube, I came up with the idea of applying this clustering algorithm to an RGB image. The documents I read mostly visualized K-means on 2-dimensional data points like this:



I wondered how I could apply this algorithm to 3-dimensional data points. Then, I realized that I would simply be working with one more parameter (the 3D space Oxyz) instead of just two (Oxy).

By treating all the pixels of an image as a collection of such points, I can use K-means clustering to group similar colors together. The algorithm will find K cluster centers in this 3D space, and each pixel will be assigned to the closest cluster center. Then, I can replace the original pixel color with the color of its cluster center. As a result, the image will have only K unique colors.

II. Methods

1. Program Structure

The program is organized into modular functions, each with a specific role:

- **read_img(img_path)**: Reads an image from the given file path and converts it into a NumPy array.
- **show_img(img_2d)**: Displays the image using matplotlib.
- **save_img(img_2d, img_path)**: Saves the image in PNG, PDF, or both formats based on user input.
- **convert_img_to_1d(img_2d)**: Reshapes a 2D image into a 1D array suitable for clustering.
- **generate_2d_img(img_2d_shape, centroids, labels)**: Reconstructs the compressed 2D image from the cluster centroids and pixel labels.
- **choose_centroids(init_centroids, img_1d, k_clusters)**: Initializes centroids either randomly or by selecting random pixels from the image.
- **kmeans(img_1d, k_clusters, max_iter, init_centroids='random')**: The core function that implements the K-Means clustering algorithm.

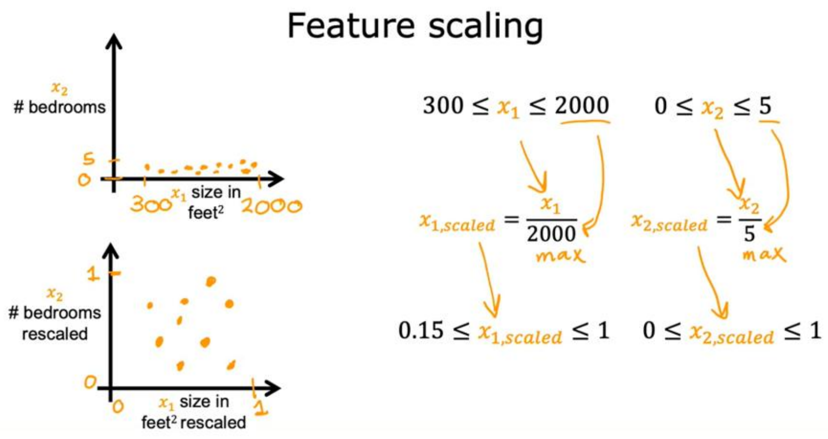
To compress an RGB image into a palette of k colors using k-means clustering, a series of steps were implemented in Python using the Python Imaging Library (PIL), NumPy, and Matplotlib. The process involves reading the image, converting its pixel data into a 2-dimensional array of pixels, applying k-means clustering, and reconstructing the compressed image.

First, the image is loaded from a specified file path using the read_img function, which utilizes PIL to open the image and convert it to RGB format. The image is then transformed into a 2D NumPy array of shape (height of the image, width of the image, 3), where each pixel is represented by its red, green, and blue intensity values. A common question is why not directly create a 1D array if the goal is to process pixel data points. A 2D array is used initially because it preserves the spatial structure of the image, enabling straightforward visualization and saving with Matplotlib and PIL.

Second, after having a 2D array of pixels, the image data is reshaped into a 1D list of RGB pixels which is suitable for clustering. This transformation is performed using a helper function `convert_img_to_1d`, which reshapes the image from `(height, width, 3)` to `(height × width, 3)`. Each row in the resulting array represents a pixel's RGB values and serves as an individual data point in 3-dimensional space.

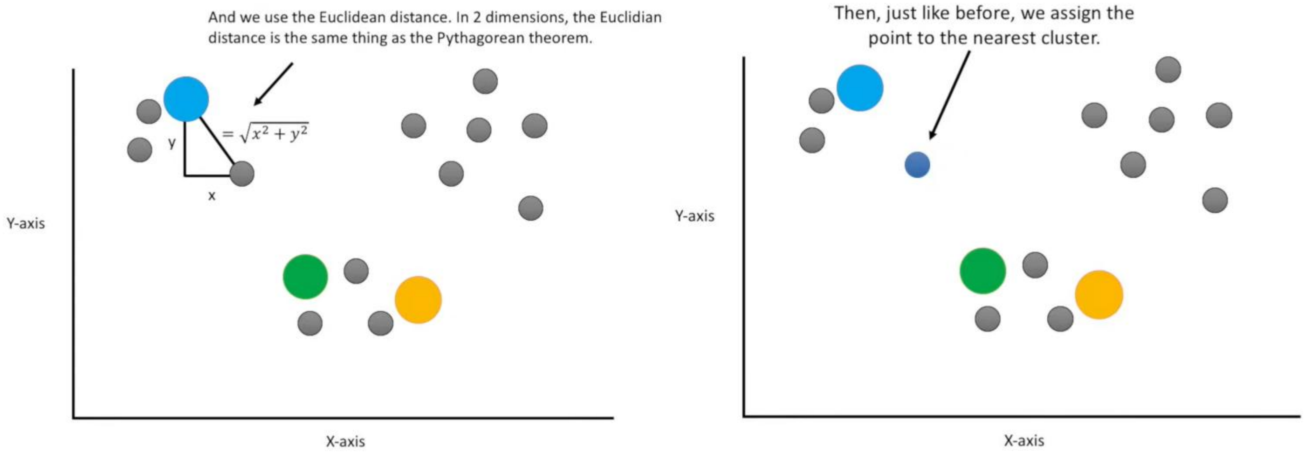
Next, the k-means clustering algorithm is applied using a custom implementation defined in the `kmeans` function. The algorithm starts by choosing `k` initial centroids either by selecting random RGB values (“"random”) or random pixels from the image (“in_pixels”). The selection is done through the `choose_centroids` function, which ensures all centroids are unique by using set data structure in Python.

Once initialized, the algorithm normalizes the pixel data and centroids to the range [0, 1] for numerical stability by dividing by the maximum of the range.

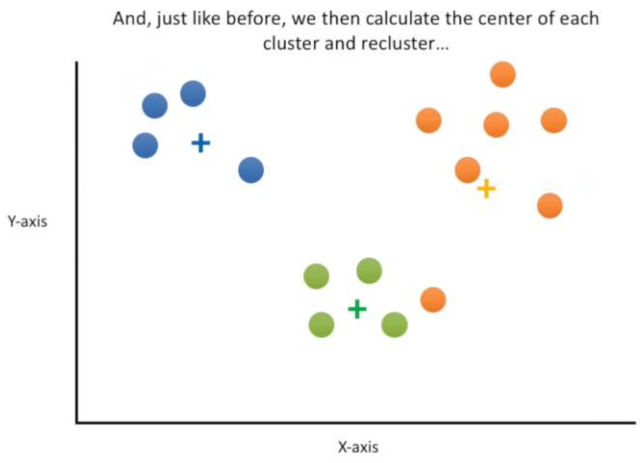

Feature scaling
x_2 # bedrooms
5
0
300
x_1 size in feet²
2000
$300 \le x_1 \le 2000$
$0 \le x_2 \le 5$
$x_{1,scaled} = \frac{x_1}{2000}$
max
$x_{2,scaled} = \frac{x_2}{5}$
max
$0.15 \le x_{1,scaled} \le 1$
$0 \le x_{2,scaled} \le 1$
x_2 # bedrooms rescaled
1
0
0
x_1 size in feet² rescaled
1

Then, it proceeds iteratively:

1. **Assignment Step:** Each pixel is assigned to the nearest centroid based on Euclidean distance (with improvement) in RGB space.


And we use the Euclidean distance. In 2 dimensions, the Euclidian distance is the same thing as the Pythagorean theorem.
$= \sqrt{x^2 + y^2}$
y
x
Y-axis
X-axis
Then, just like before, we assign the point to the nearest cluster.
Y-axis
X-axis

2. **Update Step:** After assigning pixels to clusters, each centroid is updated by computing the average (mean) color of all the pixels assigned to that cluster. This is done by summing the RGB values of the assigned pixels and dividing by the number of pixels in the cluster, effectively repositioning the centroid to the center of its assigned data points.


And, just like before, we then calculate the center of each cluster and recluster...
Y-axis
X-axis

3. **Convergence Check:** If the maximum change in any centroid's position is less than a small threshold (e.g., 0.001), the algorithm halts, this is when the loss function converges.

The iteration continues until convergence or the maximum number of iterations ('max_iter') is reached. Once finished, the centroids are rescaled to [0, 255] and returned along with a label for each pixel indicating which cluster it belongs to.

After obtaining the cluster labels, the compressed image is reconstructed using the `generate_2d_img` function. This function replaces each original pixel with its corresponding centroid color. The labels array is reshaped to match the original image's height and width, and each pixel in the new image is assigned the RGB values of its cluster centroid.

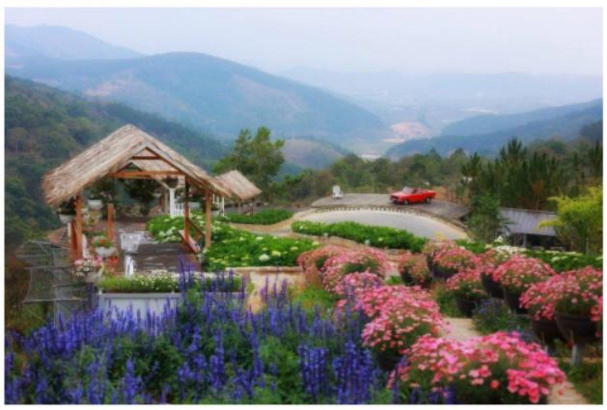


Figure 1: Dalat before compression

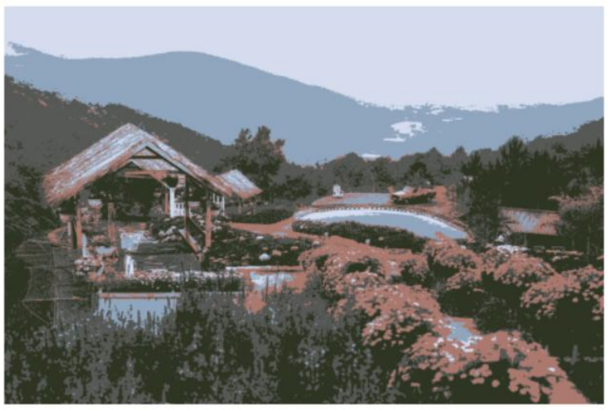


Figure 2: Dalat after compression with k = 5

Finally, the image can be displayed using the 'show_img' function or saved in PDF and/or PNG format using the 'save_img' function. The output image maintains the original spatial resolution but is limited to exactly 'k' distinct colors, effectively achieving lossy compression. The result retains the essential visual features while reducing the color space and storage requirements.

2. Core Functions and Improvements

- The **kmeans function** has been optimized using a mathematical formulation for computing Euclidean distances:

$$\| x - c \|^2 = \| x \|^2 - 2 \cdot x^T \cdot c + \| c \|^2$$

This reformulation avoids directly computing the squared differences and instead uses dot products and precomputed norms, improving performance and scalability.

- The **choose_centroids function** ensures that randomly initialized centroids are unique, reducing the chance of poor clustering results due to duplicate centroids.

III. Results and conclusion

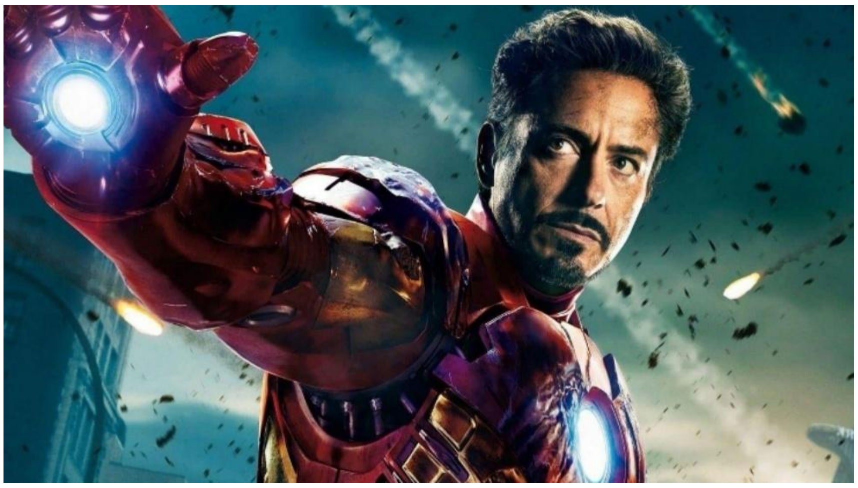
1. Results

These experiments were conducted on the following setup:

- Device: Laptop HP 15s fq2045TU
- CPU: Intel core i7 1165G7
- Ram: Gskill Ripjaws 16GB DDR4 3200MHz
- OS: Window 11
- Environment: Python 3.13.0, Numpy, Matplotlib, PIL

Let's begin with my favorite character from Avengers, Iron Man.

Original image: Iron Man (Dimension: 1920 x 1080 with Size: 1.51 MB)



“Random” method with max iterations = 10000:

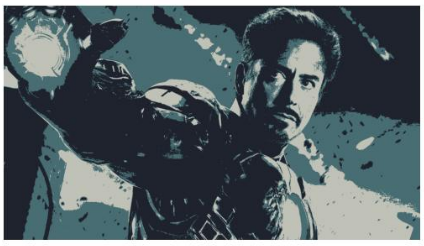


Figure 1: k = 3, runtime = 1.232475s

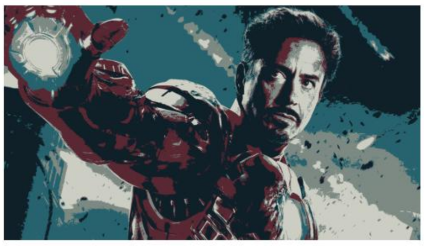


Figure 2: k = 5, runtime = 3.762998s

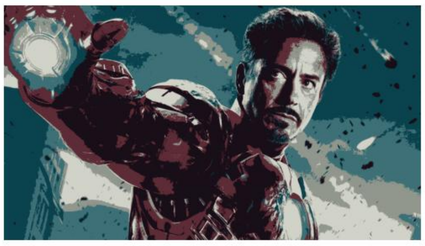


Figure 3: k = 7, runtime = 3.900795s

“In pixels” method with max iterations = 10000:

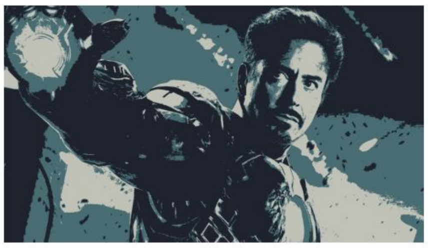


Figure 4: k = 3, runtime = 1.330149s

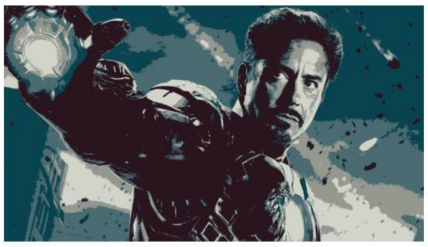


Figure 5: k = 5, runtime = 5.734336s

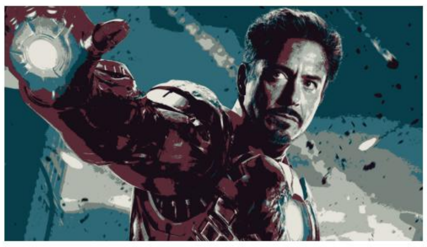


Figure 6: k = 7, runtime = 19.569252s

Runtime Comparison

- At **k = 3**, both methods show similar runtimes:
  - Random: 1.2325s
  - In pixels: 1.3301s
- As **k increases**, runtime increases significantly for both methods, but especially for the “In pixels” method:
  - At **k = 5**:
    - Random: 3.7629s
    - In pixels: 5.7343s
  - At **k = 7**:
    - Random: 3.9008s
    - In pixels: 19.5693s (more than 5 times slower)

Quality and Visual Results

- As **k increases**, the image becomes more detailed and less posterized, which is expected due to more color centroids capturing more nuance.
- Both methods visually improve the image as k increases, but the differences between methods are subtle to moderate. The "In pixels" results may slightly preserve more accurate color tones, especially at higher k values, possibly due to better initial centroid selection.

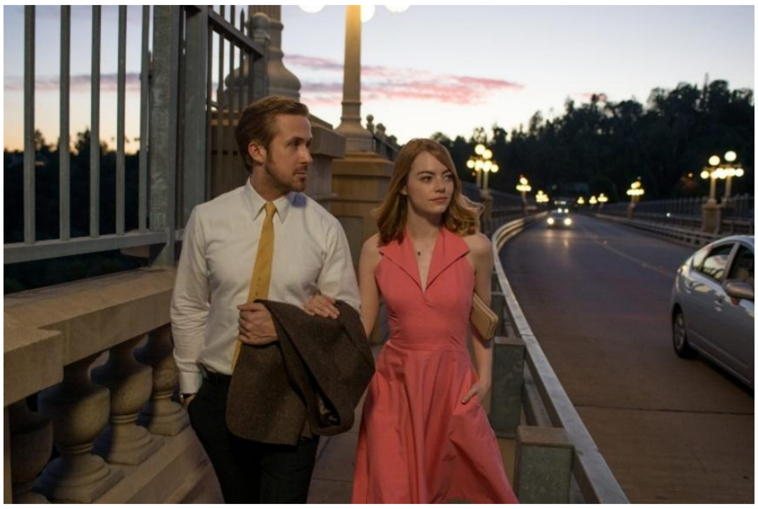
Method Evaluation

- **Random Initialization**:
  - Faster on average.
  - More efficient for small to medium k values.
  - May suffer from local minima or inconsistent results depending on the randomness.

- **In Pixels Initialization**:
    - Slower, especially at higher k values.
    - Potentially better initial centroids as they are based on actual pixel values.
    - Likely to produce more stable or slightly better visual quality, especially for complex images or larger k.

Now, let's test on the second image from La La Land.

Original image: La La Land (Dimension: 2560 x 1700 with Size: 397 KB)



"Random" method with max iterations = 10000:

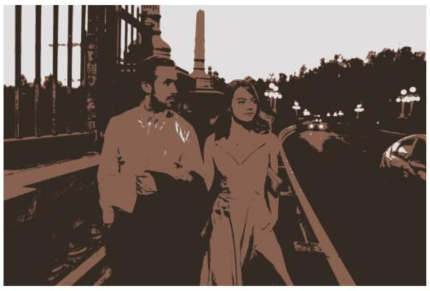


Figure 8: k = 3, runtime = 2.987760s

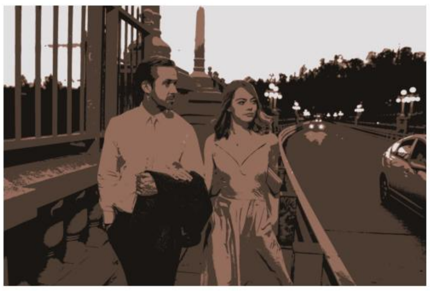


Figure 9: k = 5, runtime = 5.293447s

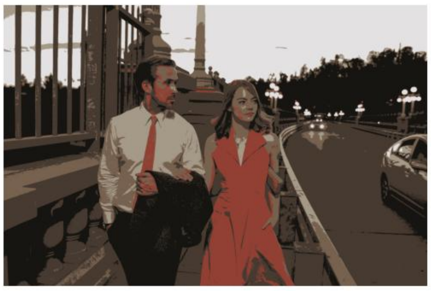


Figure 10: k = 7, runtime = 6.042157s

"In pixels" method with max iterations = 10000:

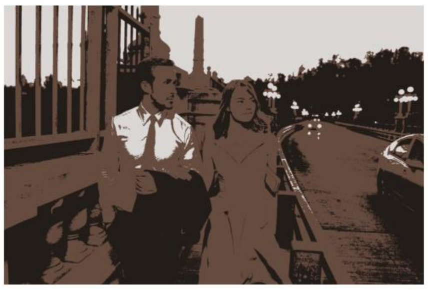


Figure 11: k = 3, runtime = 3.627527s

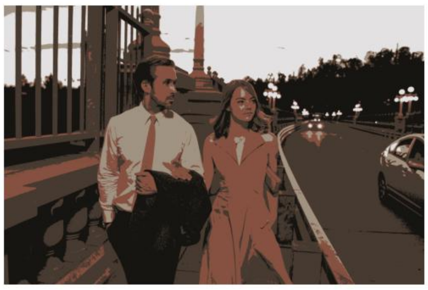


Figure 12: k = 5, runtime = 7.832237s

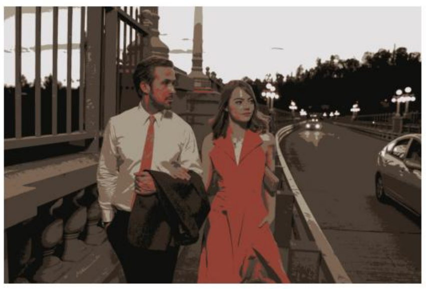


Figure 13: k = 7, runtime = 9.433557s

How about k = 15 and k = 30 with “in_pixels” method?

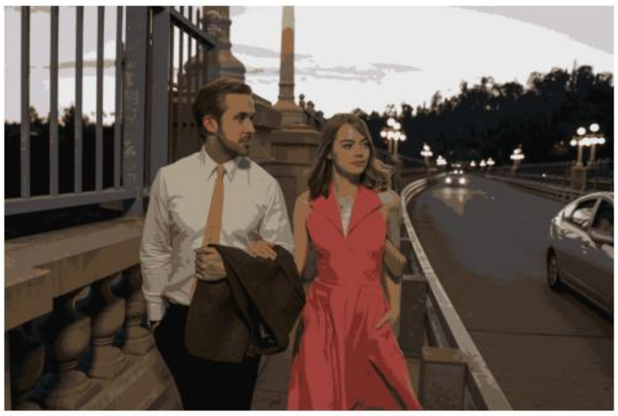


Figure 14: k = 15, runtime = 49.185343s

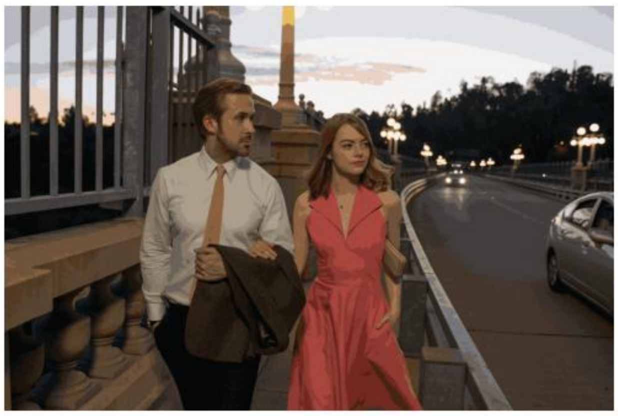


Figure 15: k = 30, runtime = 56.759814s

Visual Quality

- At **k = 3**, both methods produce highly **posterized** images with significant color loss. “In pixels” appears slightly more natural in the facial and background tones.
- As **k increases to 5 and 7**, images become **smoother and more detailed**, especially in the face, hair, clothes, and background (street lights, shadows).
- At **k = 15 and k = 30**, especially in “In pixels”:
  - The images become **visually close to the original**, retaining fine details like fabric texture and shading.
  - There's a **diminishing return** in quality improvement compared to the jump in runtime.

Method Comparison

- **Random Initialization**
  - **Pros**:
    - Faster at lower k
    - Simple and efficient
  - **Cons**:
    - May lead to suboptimal clustering
    - Visual quality at higher kkk not as consistent or smooth
- **In Pixels Initialization**
  - **Pros**:
    - Better **initial centroids** → more stable convergence
    - **Improved visual quality** at higher kkk
  - **Cons**:
    - **Significantly slower**, especially at k≥15k \geq 15k≥15
    - Computational cost becomes high and may not be worth it if performance is a concern

2. Conclusion

In this study, I applied the K-means clustering algorithm for image compression using two centroid initialization strategies: **“Random”** and **“In pixels”.** My experiments demonstrated that the number of clusters k and the initialization method both significantly impact the **runtime performance** and **visual quality** of the compressed images.

- The **“Random” method** achieved faster runtimes across all k values but occasionally produced less stable or suboptimal visual results.
- The **“In pixels” method**, while more computationally expensive especially at higher k yielded smoother, more accurate images, thanks to better centroid initialization.
- The higher value of k, the higher qualitity of the compressed image.

Overall, the **choice of k** should be a balance between **compression quality** and **computational efficiency**. These findings suggest that while K-means is a viable method for image compression, its effectiveness can be significantly influenced by both parameter selection and initialization strategy.

IV. References

- StatQuest with Josh Starmer, StatQuest: K-means clustering, YouTube link, 8:31am - 12/6/2025.
- Real Python, Start Using the Pillow Library to Process Images in Python, YouTube link, 9:17am - 12/6/2025.
- Pillow (PIL Fork) 11.2.1 documentation, url, 9:54am - 12/6/2025.
- Machinelearningcoban, K-means Clustering, url, 10:43am - 12/6/2025.
- NeuralNine, K-Means Clustering From Scratch in Python (Mathematical), YouTube link, 13:45am - 12/6/2025.
- Geeksforgeeks, K-means++ Algorithm – ML, url, 14:42am - 12/6/2025.

V. Acknowledgement

This report was prepared with the assistance of ChatGPT, Grok in translating, grammar fixing, writing Sections II, III and improving k-means function.