# Project 01: Color Compression

**Name: Châu Tấn Kiệt**

**ID: 21127329**

**Class: 21CLC04**

## 1. Problem analysis and programming idea:

- Step 1: Read the image from the user input and determine its width, height, and color of each pixel (an array of 3 elements - R,G,B) - or a 3D array (width, height & RGB)
- Step 2: Randomize k centroids with values ranging from 0 to 255.
- Step 3: Create an array to store the cluster assigned for each pixel
- Step 4: Loop through the number of iterations over each pixel in the image
- Step 5: Use Euclidean distance to check the distance between the pixel and the centroid
- Step 6: Update the centroids
- Step 7: Rejoining the pixels to create an image
- Step 8: Export to file

## 2. Function explanations:

- display_image: Use matplotlib.pyplot to show the image.

```
image = Image.open("test2.jpg")

def display_image(image):
    plt.imshow(image)
    plt.show()

display_image(image)
```

- calc_distance: Calculate Euclidean distance.

```
distance = np.square(x1 - x2) + np.square(y1 - y2)
    distance = np.sqrt(distance)
    return distance
```

- k_means: Implements the K-means clustering algorithm.

```
def k_means(pixels, means, clusters):
    iter = 10
    m, n = pixels.shape

    # Index correspond to the cluster where each pixel belongs to.
    index = np.zeros(m)

    # K-means
    while iter > 0:
        for j in range(m):
            min_dist = float('inf')

            for k in range(clusters):
                x1, y1 = pixels[j, 0], pixels[j, 1]
                x2, y2 = means[k, 0], means[k, 1]

                if calc_distance(x1, y1, x2, y2) <= min_dist:
                    min_dist = calc_distance(x1, y1, x2, y2)
                    index[j] = k

        for k in range(0,clusters):
            cluster_points = pixels[index == k]
            if len(cluster_points) > 0:
                means[k] = np.mean(cluster_points, axis=0)
        iter -= 1
    return means, index
```

- initialize_means: Chooses the initial centroids randomly from among the colors in 3D matrix.

```
def initialize_means(img_np, clusters):
    # Reshaping into a 2d matrix
    pixels = img_np.reshape((-1,3))
    pixels = np.float32(pixels)
    m, n =  pixels.shape

    means = np.zeros((clusters, n))

    # Randomized initialization of means.
    for i in range(0,clusters):
```

```
        rand_indices = np.random.choice(m, size=10, replace=False)
        means[i] = np.mean(pixels[rand_indices], axis=0)
    return pixels, means
```

- save_img & compress_img: Saves the processed image as .jpg file.

```
def save_img(img_np, clusters):
    filename = f"result{clusters}.jpg"
    mimg.imsave(filename, img_np)

def compress_img(means, index, img_np, clusters):
    centroid = np.array(means)
    recovered = centroid[index.astype(np.int32), :]
    # Getting back the 3d matrix (row, col, rgb(3))

    recovered = (recovered / 255).reshape(img_np.shape)
    # Plotting the compressed image.

    display_image(recovered)
    save_img(recovered, clusters)
```

- save_pdf & compress_pdf: Saves the processed image as .pdf file.

```
def save_pdf(img_np, clusters):
    filename = f"result{clusters}.pdf"
    mimg.imsave(filename, img_np)

def compress_pdf(means, index, img_np, clusters):
    centroid = np.array(means)
    recovered = centroid[index.astype(np.int32), :]
    # Getting back the 3d matrix (row, col, rgb(3))

    recovered = (recovered / 255).reshape(img_np.shape)
    # Plotting the compressed image.

    display_image(recovered)
    save_pdf(recovered, clusters)
```

- main function: Allows user to enter image filename, number of clusters and file saving format.

```
if __name__ == '__main__':
    filename = input("Enter filename: ")
    image = Image.open(filename)
    img_np = np.array(image)
```

```
    img_np = np.float32(img_np)

    clusters = 10
    clusters = int(input('Enter the number of clusters (default = 10): '))

    pixels, means = initialize_means(img_np,clusters)
    means, index = k_means(pixels,means,clusters)

    fmt = ''
    while (fmt != "jpg" or fmt != "pdf"):
        fmt = input("Enter file saving format ( jpg / pdf ) :" )
        if fmt == "jpg":
            compress_img(means, index, img_np, clusters)
            break
        elif fmt == "pdf":
            compress_pdf(means, index, img_np, clusters)
            break
        else:
            print("Invalid file format, please re-enter:" )
```

## 3. Testcase with Image 1 (695 x 666 pixels):

Original Image:

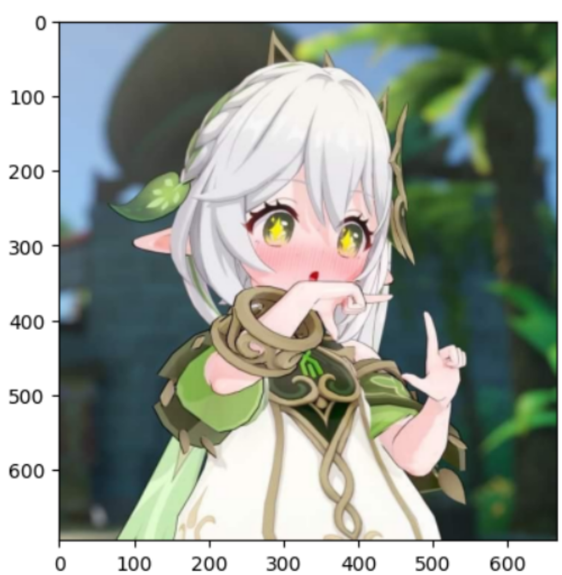



Image after processing with k = 3:

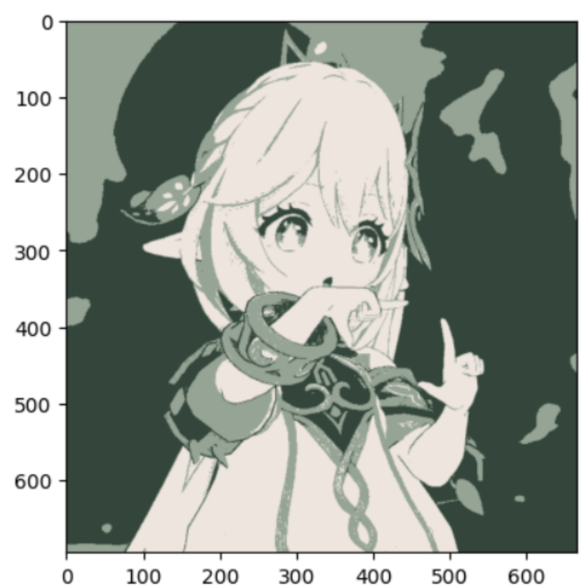
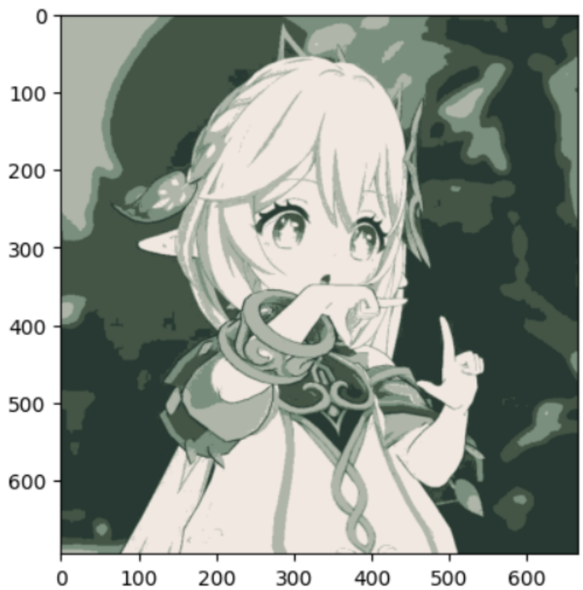
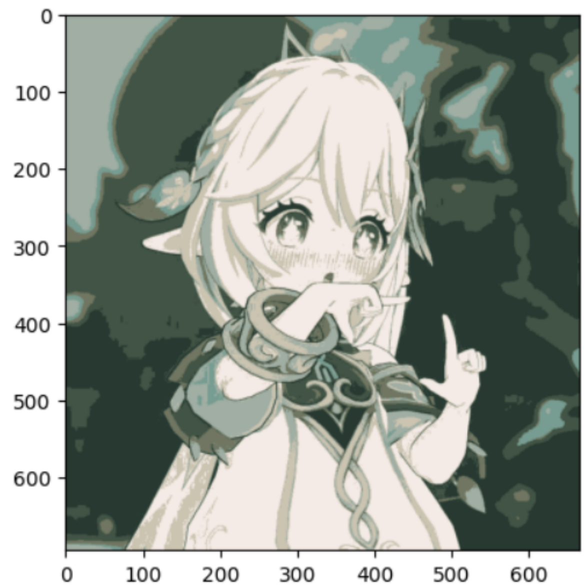



Image after processing with k = 5:

Image after processing with k = 7:




Runtime for each process

| | Total time (10 iterations) | Average time with 1 iteration |
|---|---|---|
| k = 3 | 2 minutes 14.2 seconds | 13.4 seconds |
| k = 5 | 4 minutes 9.8 seconds | 25 seconds |
| k = 7 | 5 minutes 15.6 seconds | 31.5 seconds |

## 4. Testcase with Image 2 (298 x 224 pixels):

Original Image:

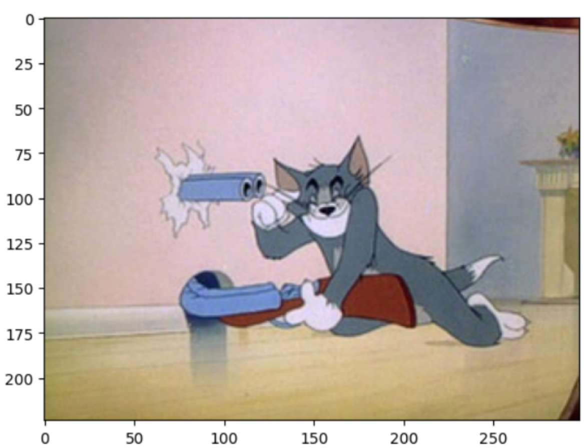



Image after processing with k = 3:

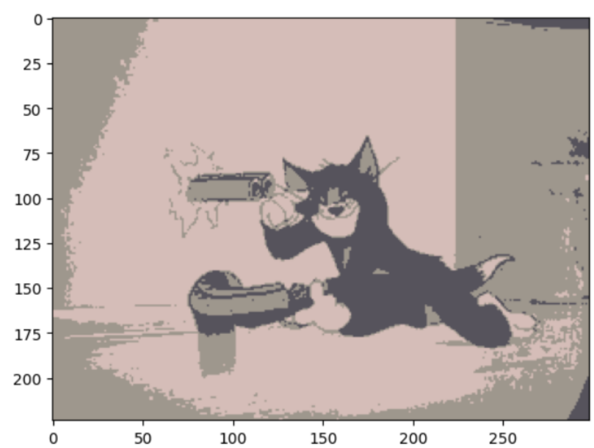


Image after processing with k = 5:

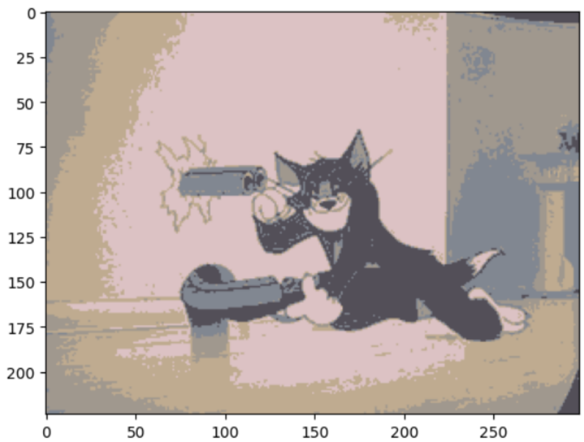
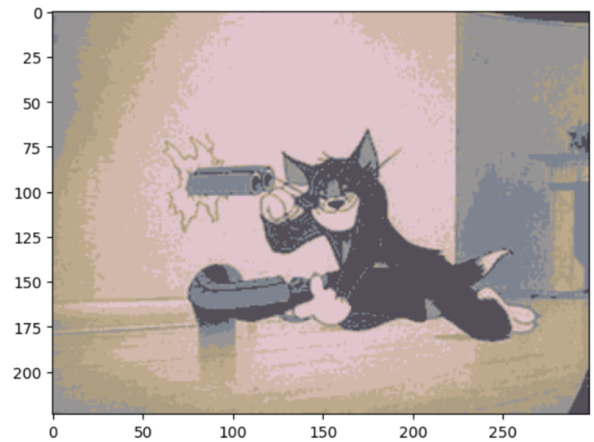


Image after processing with k = 7:



| | Total time (10 iterations) | Average time with 1 iteration |
|---|---|---|
| k = 3 | 16.7 seconds | 1.7 seconds |
| k = 5 | 26.2 seconds | 2.6 seconds |
| k = 7 | 32.2 seconds | 3.2 seconds |

## 5. Comments:

- Runtime: Average runtime and total runtime for smaller k is significantly faster than bigger k
- File size: Directly affect runtime, due to the number of data points to compare (as in 695 x 666 pixels in image 1 and 298 x 224 pixels in image 2 leads to drastically large gap between the time processing these images)
- Image quality: Larger k will retain more details for the images than smaller k, but smaller k will still have some defining contents of the images.
- File size after processing: Smaller k means the number of colors is grouped into less groups, thus making the file size smaller, which is suitable for data compression.

| | | | |
|---|---|---|---|
| result3.jpg | 7/16/2023 4:38 PM | JPG File | 50 KB |
| result5.jpg | 7/16/2023 4:42 PM | JPG File | 52 KB |
| result7.jpg | 7/16/2023 4:47 PM | JPG File | 53 KB |

## 6. References:


k-means clustering - Wikipedia

(192) K-Means Clustering Algorithm with Python Tutorial - YouTube

(192) Machine Learning Tutorial Python - 13: K Means Clustering Algorithm - YouTube

Machine-Learning-without-Libraries/K-Means-Clustering/K-Means-Clustering-without-ML-libraries.ipynb at master · CihanBosnali/Machine-Learning-without-Libraries · GitHub

Image Segmentation using K Means Clustering - GeeksforGeeks

Machine Learning cơ bản (machinelearningcoban.com)

Python Machine Learning - K-means (w3schools.com)

Image Clustering Using k-Means. Using transfer learning model for… | by Shubham Gupta | Towards Data Science

How to Use K-Means Clustering for Image Segmentation using OpenCV in Python - Python Code (thepythoncode.com)