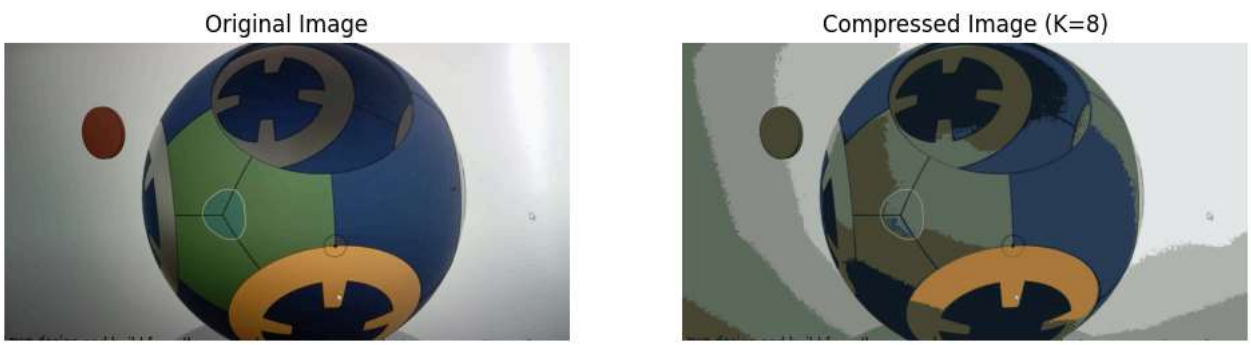
```
#BASIC V1

import numpy as np
import matplotlib.pyplot as plt
import cv2
from sklearn.cluster import KMeans

def compress_image(image_path, k=8):
    # Read the image
    img = cv2.imread(image_path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # Convert to RGB

    # Reshape image into a 2D array of pixels
    pixels = img.reshape((-1, 3))

    # Apply KMeans clustering
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(pixels)

    # Replace each pixel with its cluster center
    compressed_pixels = kmeans.cluster_centers_[kmeans.labels_]
    compressed_pixels = np.clip(compressed_pixels.astype('uint8'), 0, 255)
    compressed_img = compressed_pixels.reshape(img.shape)

    return img, compressed_img

def display_images(original, compressed, k):
    fig, ax = plt.subplots(1, 2, figsize=(12, 6))

    ax[0].imshow(original)
    ax[0].set_title("Original Image")
    ax[0].axis("off")

    ax[1].imshow(compressed)
    ax[1].set_title(f"Compressed Image (K={k})")
    ax[1].axis("off")

    plt.show()

# Example usage
image_path = "robot2.jpg"  # Replace with your image file
k = 8  # Number of clusters
original, compressed = compress_image(image_path, k)
display_images(original, compressed, k)
```



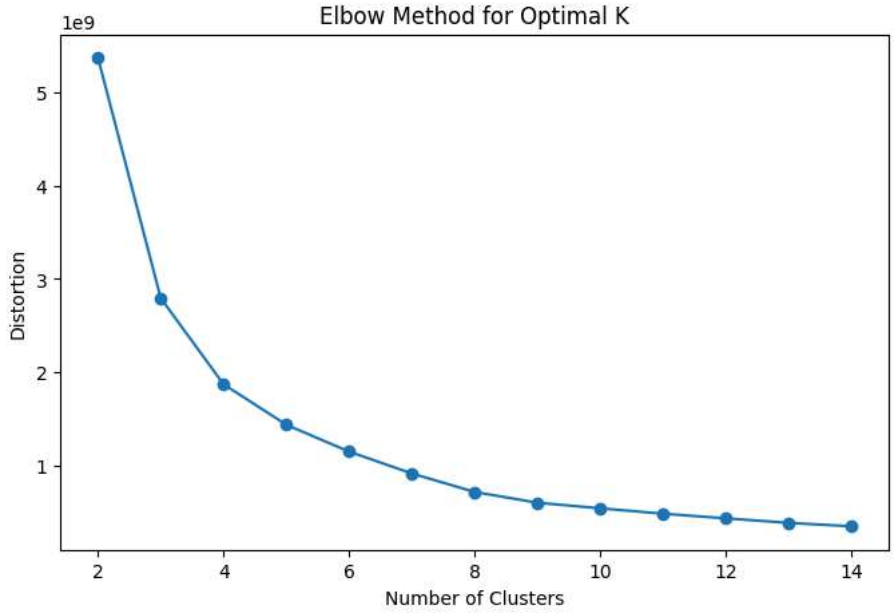

```
#V2 WITH GRAPH
import numpy as np
import matplotlib.pyplot as plt
import cv2
import os
from sklearn.cluster import KMeans
from google.colab import files

def upload_image():
    uploaded = files.upload()
    for filename in uploaded.keys():
        return filename

def compress_image(image_path, k=8):
```

```
        img = cv2.imread(image_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        pixels = img.reshape((-1, 3))

        kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
        kmeans.fit(pixels)

        compressed_pixels = kmeans.cluster_centers_[kmeans.labels_]
        compressed_pixels = np.clip(compressed_pixels.astype('uint8'), 0, 255)
        compressed_img = compressed_pixels.reshape(img.shape)

        return img, compressed_img

    def elbow_method(image_path, max_k=15):
        img = cv2.imread(image_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        pixels = img.reshape((-1, 3))
        distortions = []
        for k in range(2, max_k):
            kmeans = KMeans(n_clusters=k, random_state=42, n_init=10).fit(pixels)
            distortions.append(kmeans.inertia_)

        plt.figure(figsize=(8,5))
        plt.plot(range(2, max_k), distortions, marker='o', linestyle='-')
        plt.xlabel('Number of Clusters')
        plt.ylabel('Distortion')
        plt.title('Elbow Method for Optimal K')
        plt.show()

    def main():
        print("Please upload an image")
        image_path = upload_image()

        print("Running Elbow Method to Find Best K...")
        elbow_method(image_path)

        k = int(input("Enter the number of clusters (K): "))
        original, compressed = compress_image(image_path, k)

        # Display images
        fig, ax = plt.subplots(1, 2, figsize=(12, 6))
        ax[0].imshow(original)
        ax[0].set_title("Original Image")
        ax[0].axis("off")
        ax[1].imshow(compressed)
        ax[1].set_title(f"Compressed Image (K={k})")
        ax[1].axis("off")
        plt.show()

        # Save compressed image
        compressed_path = "compressed_image.jpg"
        cv2.imwrite(compressed_path, cv2.cvtColor(compressed, cv2.COLOR_RGB2BGR))
        print(f"Compressed image saved as {compressed_path}")

    if __name__ == "__main__":
        main()
```

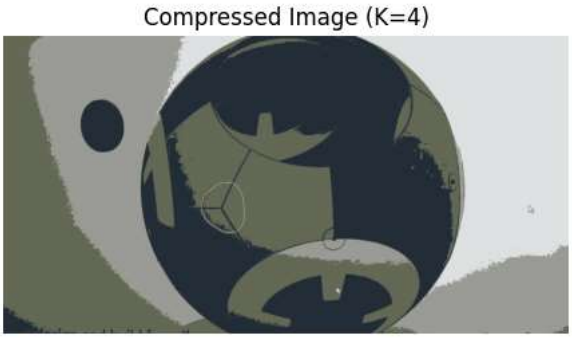
```
Please upload an image
```
Choose Files robot2.jpg

- **robot2.jpg**(image/jpeg) - 59385 bytes, last modified: 9/3/2024 - 100% done

```
Saving robot2.jpg to robot2 (1).jpg
Running Elbow Method to Find Best K...
```




```
Enter the number of clusters (K): 4
```



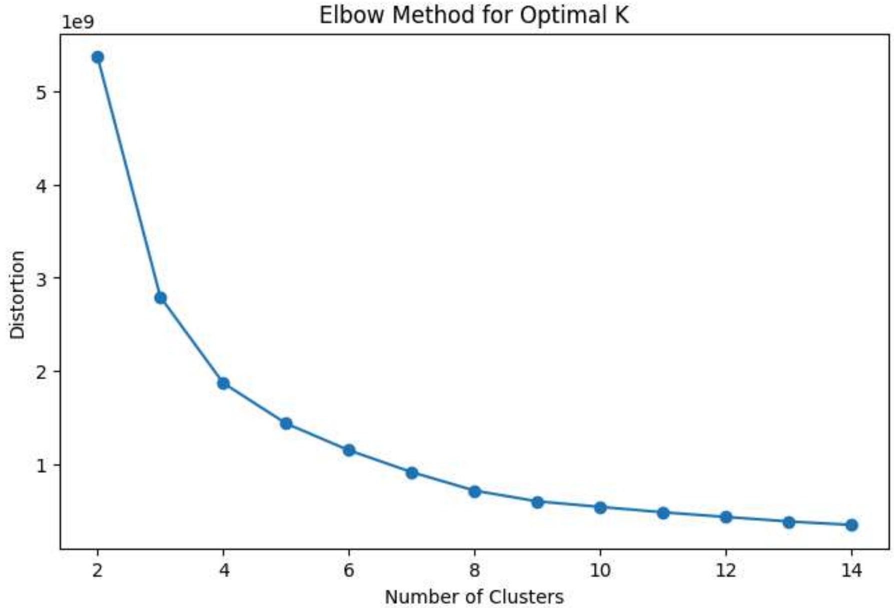

```
#V3 WITH ADDITIONAL FEATURES
import numpy as np
import matplotlib.pyplot as plt
import cv2
import os
import imageio
from sklearn.cluster import KMeans
from google.colab import files

def upload_image():
    uploaded = files.upload()
    for filename in uploaded.keys():
        return filename

def compress_image(image_path, k=8):
    img = cv2.imread(image_path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    pixels = img.reshape((-1, 3))

    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(pixels)

    compressed_pixels = kmeans.cluster_centers_[kmeans.labels_]
    compressed_pixels = np.clip(compressed_pixels.astype('uint8'), 0, 255)
    compressed_img = compressed_pixels.reshape(img.shape)

    return img, compressed_img

def elbow_method(image_path, max_k=15):
    img = cv2.imread(image_path)
```

```
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        pixels = img.reshape((-1, 3))
        distortions = []
        for k in range(2, max_k):
            kmeans = KMeans(n_clusters=k, random_state=42, n_init=10).fit(pixels)
            distortions.append(kmeans.inertia_)

        plt.figure(figsize=(8,5))
        plt.plot(range(2, max_k), distortions, marker='o', linestyle='-')
        plt.xlabel('Number of Clusters')
        plt.ylabel('Distortion')
        plt.title('Elbow Method for Optimal K')
        plt.show()

    def create_gif(image_path, k_values=[2, 4, 6, 8, 10]):
        images = []
        for k in k_values:
            _, compressed = compress_image(image_path, k)
            compressed_bgr = cv2.cvtColor(compressed, cv2.COLOR_RGB2BGR)
            compressed_path = f"compressed_k{k}.jpg"
            cv2.imwrite(compressed_path, compressed_bgr)
            images.append(imageio.imread(compressed_path))

        gif_path = "compression_steps.gif"
        imageio.mimsave(gif_path, images, duration=1)
        print(f"GIF saved as {gif_path}")

    def main():
        print("Please upload an image")
        image_path = upload_image()

        print("Running Elbow Method to Find Best K...")
        elbow_method(image_path)

        try:
            k = int(input("Enter the number of clusters (K): "))
        except ValueError:
            k = np.random.choice([4, 6, 8, 10])
            print(f"No input provided, using random K={k}")

        original, compressed = compress_image(image_path, k)

        # Display images
        fig, ax = plt.subplots(1, 2, figsize=(12, 6))
        ax[0].imshow(original)
        ax[0].set_title("Original Image")
        ax[0].axis("off")
        ax[1].imshow(compressed)
        ax[1].set_title(f"Compressed Image (K={k})")
        ax[1].axis("off")
        plt.show()

        # Save compressed image
        compressed_path = "compressed_image.jpg"
        cv2.imwrite(compressed_path, cv2.cvtColor(compressed, cv2.COLOR_RGB2BGR))
        print(f"Compressed image saved as {compressed_path}")

        # Generate GIF animation of compression steps
        create_gif(image_path)

    if __name__ == "__main__":
        main()
```

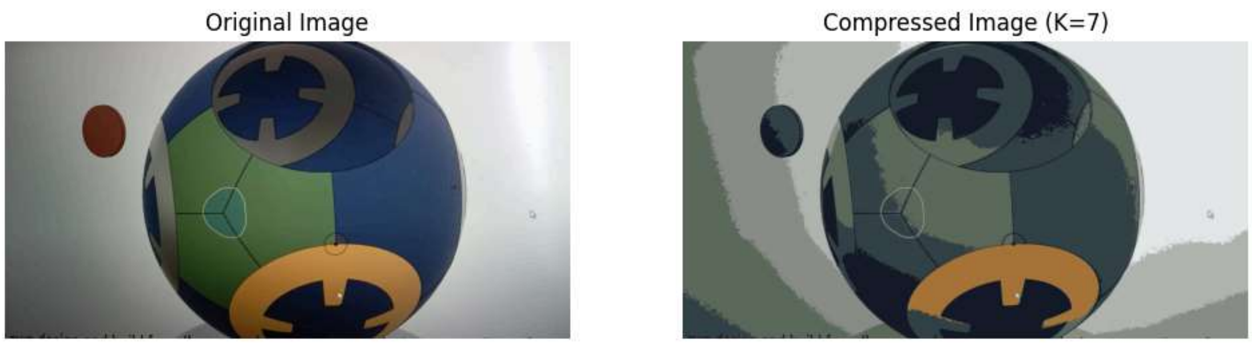
```
Please upload an image
```
Choose Files robot2.jpg

- **robot2.jpg**(image/jpeg) - 59385 bytes, last modified: 9/3/2024 - 100% done

```
Saving robot2.jpg to robot2 (2).jpg
Running Elbow Method to Find Best K...
```




```
Enter the number of clusters (K): 7
```




```
Compressed image saved as compressed_image.jpg
<ipython-input-6-8e09462fbbf1>:51: DeprecationWarning: Starting with ImageIO v3 the behavior of this function will switch to that of
  images.append(imageio.imread(compressed_path))
<ipython-input-6-8e09462fbbf1>:51: DeprecationWarning: Starting with ImageIO v3 the behavior of this function will switch to that of
  images.append(imageio.imread(compressed_path))
<ipython-input-6-8e09462fbbf1>:51: DeprecationWarning: Starting with ImageIO v3 the behavior of this function will switch to that of
  images.append(imageio.imread(compressed_path))
<ipython-input-6-8e09462fbbf1>:51: DeprecationWarning: Starting with ImageIO v3 the behavior of this function will switch to that of
  images.append(imageio.imread(compressed_path))
<ipython-input-6-8e09462fbbf1>:51: DeprecationWarning: Starting with ImageIO v3 the behavior of this function will switch to that of
  images.append(imageio.imread(compressed_path))
GIF saved as compression_steps.gif
```

```
#V4
import numpy as np
import matplotlib.pyplot as plt
import cv2
import os
import imageio
import time
from sklearn.cluster import KMeans
from google.colab import files

def upload_image():
    uploaded = files.upload()
    for filename in uploaded.keys():
        return filename

def compress_image(image_path, k=8):
    img = cv2.imread(image_path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    pixels = img.reshape((-1, 3))
```

```
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(pixels)

    compressed_pixels = kmeans.cluster_centers_[kmeans.labels_]
    compressed_pixels = np.clip(compressed_pixels.astype('uint8'), 0, 255)
    compressed_img = compressed_pixels.reshape(img.shape)

    return img, compressed_img

def elbow_method(image_path, max_k=15):
    img = cv2.imread(image_path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    pixels = img.reshape((-1, 3))
    distortions = []
    for k in range(2, max_k):
        kmeans = KMeans(n_clusters=k, random_state=42, n_init=10).fit(pixels)
        distortions.append(kmeans.inertia_)

    plt.figure(figsize=(8,5))
    plt.plot(range(2, max_k), distortions, marker='o', linestyle='-')
    plt.xlabel('Number of Clusters')
    plt.ylabel('Distortion')
    plt.title('Elbow Method for Optimal K')
    plt.show()

def create_gif(image_path, k_values=[2, 4, 6, 8, 10]):
    images = []
    for k in k_values:
        _, compressed = compress_image(image_path, k)
        compressed_bgr = cv2.cvtColor(compressed, cv2.COLOR_RGB2BGR)
        compressed_path = f"compressed_k{k}.jpg"
        cv2.imwrite(compressed_path, compressed_bgr)
        images.append(imageio.imread(compressed_path))

    gif_path = "compression_steps.gif"
    imageio.mimsave(gif_path, images, duration=1)
    print(f"GIF saved as {gif_path}")

def calculate_compression_ratio(original_path, compressed_path):
    original_size = os.path.getsize(original_path) / 1024  # Convert to KB
    compressed_size = os.path.getsize(compressed_path) / 1024
    compression_ratio = (original_size - compressed_size) / original_size * 100
    return original_size, compressed_size, compression_ratio

def main():
    print("Please upload an image")
    image_path = upload_image()

    print("Running Elbow Method to Find Best K...")
    elbow_method(image_path)

    try:
        k = int(input("Enter the number of clusters (K): "))
    except ValueError:
        k = np.random.choice([4, 6, 8, 10])
        print(f"No input provided, using random K={k}")

    original, compressed = compress_image(image_path, k)

    # Display images
    fig, ax = plt.subplots(1, 2, figsize=(12, 6))
    ax[0].imshow(original)
    ax[0].set_title("Original Image")
    ax[0].axis("off")
    ax[1].imshow(compressed)
    ax[1].set_title(f"Compressed Image (K={k})")
    ax[1].axis("off")
    plt.show()

    # Save compressed image
    compressed_path = "compressed_image.jpg"
    cv2.imwrite(compressed_path, cv2.cvtColor(compressed, cv2.COLOR_RGB2BGR))
    print(f"Compressed image saved as {compressed_path}")

    # Calculate compression ratio
    orig_size, comp_size, ratio = calculate_compression_ratio(image_path, compressed_path)
    print(f"Original Size: {orig_size:.2f} KB")
```

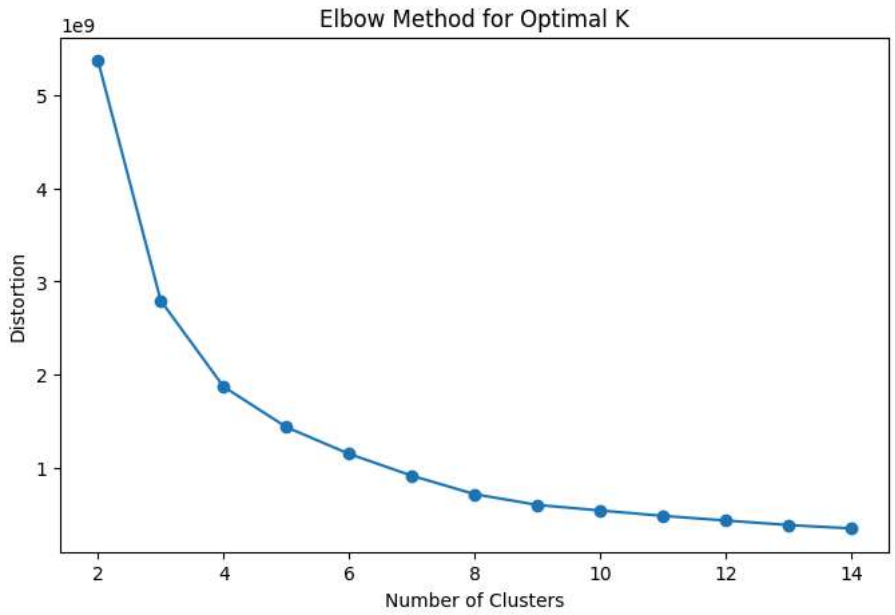
```
        print(f"Compressed Size: {comp_size:.2f} KB")
        print(f"Compression Ratio: {ratio:.2f}%")

        # Generate GIF animation of compression steps
        print("Generating GIF animation of compression process...")
        time.sleep(1)
        create_gif(image_path)
        print("GIF animation complete!")

if __name__ == "__main__":
    main()
```

```
Please upload an image
```

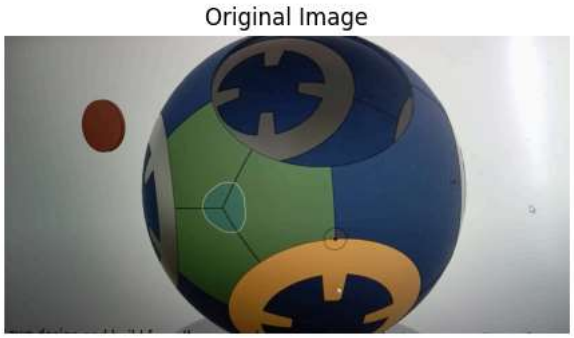
Choose Files robot2.jpg

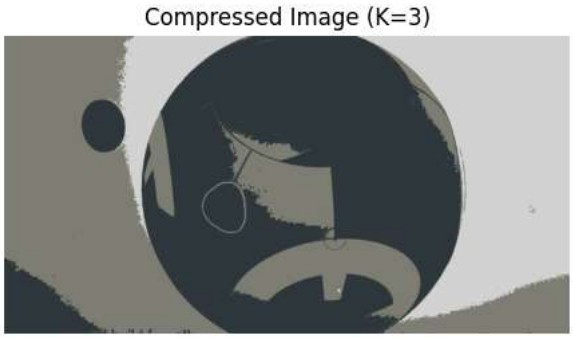
- **robot2.jpg**(image/jpeg) - 59385 bytes, last modified: 9/3/2024 - 100% done

```
Saving robot2.jpg to robot2 (2).jpg
Running Elbow Method to Find Best K...
```




```
Enter the number of clusters (K): 3
```






```
Compressed image saved as compressed_image.jpg
Original Size: 57.99 KB
Compressed Size: 86.04 KB
Compression Ratio: -48.36%
Generating GIF animation of compression process...
```