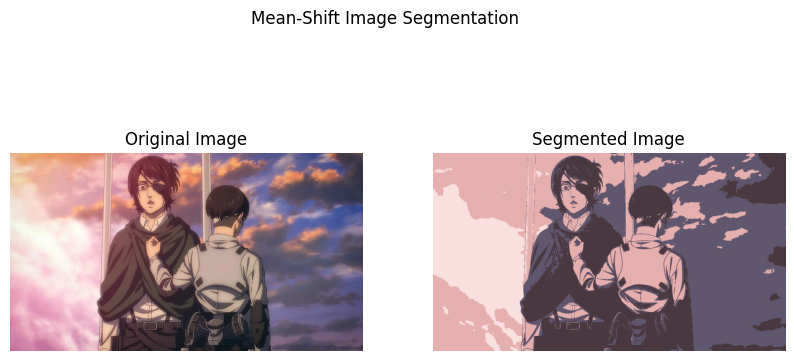
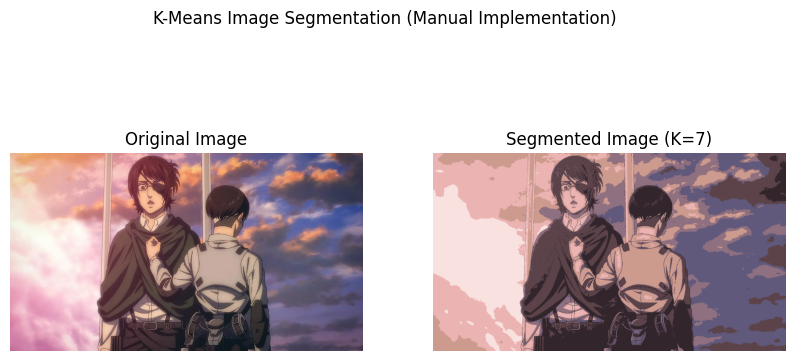
**1. Mean Shift Algorithm**

# Importing the necessary libraries  
from sklearn.cluster import MeanShift, estimate\_bandwidth  
import numpy as np  
import cv2  
import matplotlib.pyplot as plt  
  
# Reading the input image  
image\_path = "aot.jpg" # Replace with your image path  
image = cv2.imread(image\_path)  
image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB) # Convert to RGB  
  
# Reshaping the image to a 2D array of pixels  
flat\_image = image.reshape((-1, 3))  
  
# Estimating bandwidth for MeanShift  
bandwidth = estimate\_bandwidth(flat\_image, quantile=0.1, n\_samples=500)  
  
# Applying Mean-Shift clustering  
ms = MeanShift(bandwidth=bandwidth, bin\_seeding=True)  
ms.fit(flat\_image)  
  
# Extracting labels and cluster centers  
labels = ms.labels\_  
cluster\_centers = ms.cluster\_centers\_  
  
# Reconstructing the segmented image  
segmented\_image = cluster\_centers[labels].reshape(image.shape).astype(np.uint8)  
  
# Plotting the original and segmented images  
plt.figure(figsize=(10, 5))  
  
plt.subplot(1, 2, 1)  
plt.imshow(image)  
plt.title("Original Image")  
plt.axis("off")  
  
plt.subplot(1, 2, 2)  
plt.imshow(segmented\_image)  
plt.title("Segmented Image")  
plt.axis("off")  
  
plt.suptitle("Mean-Shift Image Segmentation")  
plt.show()



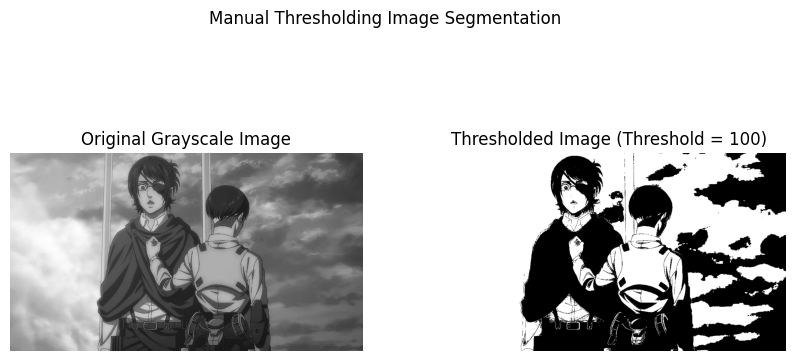
**2. K-Means Clustering**

# Reshaping the image to a 2D array of pixels  
pixels = image.reshape((-1, 3))  
pixels = np.float32(pixels)  
  
# Parameters  
K = 7 # Number of clusters (adjust as needed)  
max\_iterations = 100  
tolerance = 1e-4  
  
# Step 1: Initialize cluster centers randomly  
centers = pixels[np.random.choice(pixels.shape[0], K, replace=False)]  
  
# Function to compute the distance between points and centers  
def compute\_distance(a, b):  
 return np.sqrt(np.sum((a - b) \*\* 2, axis=1))  
  
# Step 2: K-Means clustering manually  
for iteration in range(max\_iterations):  
 # Step 3: Assign each point to the nearest cluster center  
 distances = np.array([compute\_distance(pixels, center) for center in centers])  
 labels = np.argmin(distances, axis=0)  
  
 # Step 4: Update cluster centers as the mean of assigned points  
 new\_centers = np.array([pixels[labels == k].mean(axis=0) if np.any(labels == k) else centers[k] for k in range(K)])  
  
 # Check for convergence (if centers do not change significantly)  
 if np.all(np.linalg.norm(new\_centers - centers, axis=1) < tolerance):  
 break  
 centers = new\_centers  
  
# Step 5: Reconstruct the segmented image  
segmented\_pixels = centers[labels].astype(np.uint8)  
segmented\_image = segmented\_pixels.reshape(image.shape)  
  
# Plotting the original and segmented images  
plt.figure(figsize=(10, 5))  
  
plt.subplot(1, 2, 1)  
plt.imshow(image)  
plt.title("Original Image")  
plt.axis("off")  
  
plt.subplot(1, 2, 2)  
plt.imshow(segmented\_image)  
plt.title(f"Segmented Image (K={K})")  
plt.axis("off")  
  
plt.suptitle("K-Means Image Segmentation (Manual Implementation)")  
plt.show()



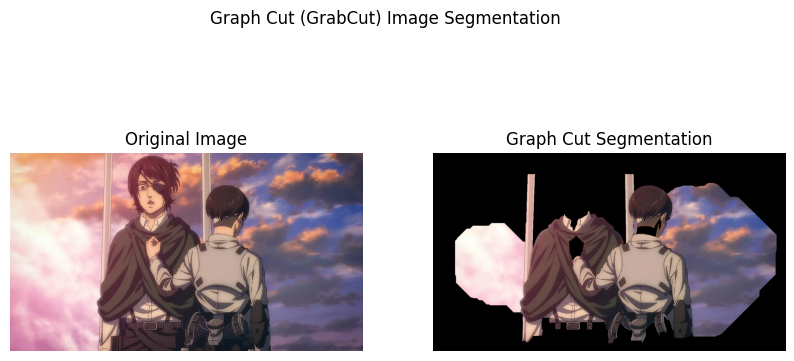
**3. Thresholding**

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY) # Convert to grayscale  
  
# Manually setting a threshold value  
threshold\_value = 100 # Adjust as needed (0-255)  
  
# Manual thresholding: Binary segmentation  
binary\_image = np.where(gray > threshold\_value, 255, 0).astype(np.uint8)  
  
# Plotting the original and thresholded images  
plt.figure(figsize=(10, 5))  
  
plt.subplot(1, 2, 1)  
plt.imshow(gray, cmap='gray')  
plt.title("Original Grayscale Image")  
plt.axis("off")  
  
plt.subplot(1, 2, 2)  
plt.imshow(binary\_image, cmap='gray')  
plt.title(f"Thresholded Image (Threshold = {threshold\_value})")  
plt.axis("off")  
  
plt.suptitle("Manual Thresholding Image Segmentation")  
plt.show()



**4. Graph Cut Segmentation**

mask = np.zeros(image.shape[:2], np.uint8)  
  
# Creating background and foreground models (required by GrabCut)  
bgd\_model = np.zeros((1, 65), np.float64)  
fgd\_model = np.zeros((1, 65), np.float64)  
  
# Defining a rectangle for the initial foreground region (manually)  
rect = (75, 75, image.shape[1] - 100, image.shape[0] - 100) # Adjust as needed  
  
# Applying GrabCut algorithm (Graph Cut)  
cv2.grabCut(image, mask, rect, bgd\_model, fgd\_model, 5, cv2.GC\_INIT\_WITH\_RECT)  
  
# Modifying the mask: setting background and probable background to 0, others to 1  
mask\_2 = np.where((mask == 2) | (mask == 0), 0, 1).astype('uint8')  
  
# Applying the mask to the original image  
segmented\_image = image \* mask\_2[:, :, np.newaxis]  
  
# Plotting the original and segmented images  
plt.figure(figsize=(10, 5))  
  
plt.subplot(1, 2, 1)  
plt.imshow(image)  
plt.title("Original Image")  
plt.axis("off")  
  
plt.subplot(1, 2, 2)  
plt.imshow(segmented\_image)  
plt.title("Graph Cut Segmentation")  
plt.axis("off")  
  
plt.suptitle("Graph Cut (GrabCut) Image Segmentation")  
plt.show()



**5. Region Growing**

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY) # Convert to grayscale  
  
# Seed point (manually chosen or based on some criteria)  
seed\_point = (100, 100) # Adjust as needed  
threshold = 10 # Threshold for similarity (intensity difference)  
  
# Creating an empty mask for the segmented region  
segmented = np.zeros\_like(gray, dtype=np.uint8)  
  
# Creating a list of points to be processed  
to\_process = [seed\_point]  
  
# Region Growing Algorithm  
while to\_process:  
 x, y = to\_process.pop(0) # Get the current point  
 # Check if the point is within image boundaries and not processed yet  
 if segmented[y, x] == 0:  
 # Mark the point as processed  
 segmented[y, x] = 255  
  
 # Check the 8-connected neighborhood  
 for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1), (-1, -1), (1, 1), (-1, 1), (1, -1)]:  
 nx, ny = x + dx, y + dy  
 # Ensure the neighbor is within bounds  
 if 0 <= nx < gray.shape[1] and 0 <= ny < gray.shape[0]:  
 # Check similarity condition  
 if segmented[ny, nx] == 0 and abs(int(gray[ny, nx]) - int(gray[y, x])) < threshold:  
 to\_process.append((nx, ny))  
  
# Plotting the original and segmented images  
plt.figure(figsize=(10, 5))  
  
plt.subplot(1, 2, 1)  
plt.imshow(gray, cmap='gray')  
plt.title("Original Grayscale Image")  
plt.axis("off")  
  
plt.subplot(1, 2, 2)  
plt.imshow(segmented, cmap='gray')  
plt.title("Region Growing Segmentation")  
plt.axis("off")  
  
plt.suptitle("Manual Region Growing Image Segmentation")  
plt.show()

