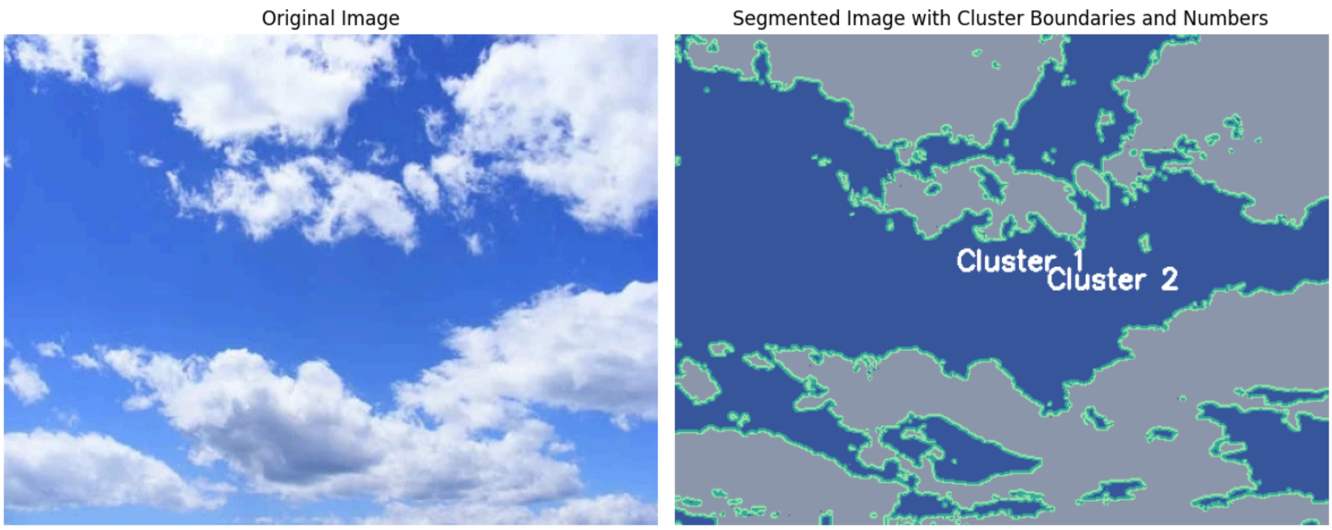
```
import numpy as np
import cv2
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Step 1: Load the image
image = cv2.imread('sky.jpeg')  # Replace with your image path
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  # Convert from BGR to RGB

# Step 2: Reshape the image to a 2D array of pixels
pixels = image_rgb.reshape(-1, 3)

# Step 3: Apply K-Means Clustering to the pixel data
k = 2  # Number of clusters (you can adjust this)
kmeans = KMeans(n_clusters=k, random_state=42)
kmeans.fit(pixels)

# Step 4: Get the clustered labels and centroids
labels = kmeans.labels_  # Cluster labels for each pixel
centroids = kmeans.cluster_centers_.astype(np.uint8)  # RGB values of the centroids of clusters

# Step 5: Recreate the segmented image
segmented_image = centroids[labels].reshape(image_rgb.shape).astype(np.uint8)

# Step 6: Find the boundaries of the clusters using contours
# Convert segmented image to grayscale
gray_segmented = cv2.cvtColor(segmented_image, cv2.COLOR_RGB2GRAY)

# Apply Canny edge detection to find edges
edges = cv2.Canny(gray_segmented, 100, 200)

# Step 7: Find contours in the edge-detected image
contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# Draw the contours on a black background
boundary_image = np.zeros_like(image_rgb, dtype=np.uint8)

# Draw each contour on the boundary image in green
for contour in contours:
    cv2.drawContours(boundary_image, [contour], -1, (0, 255, 0), 2)  # Green color

# Step 8: Overlay the boundaries on the segmented image
boundary_overlay = cv2.addWeighted(segmented_image, 0.7, boundary_image, 0.3, 0)

# Step 9: Annotate the number of clusters on the image
font = cv2.FONT_HERSHEY_SIMPLEX

for i in range(k):
    # Find pixel positions belonging to this cluster
    cluster_pixels = np.column_stack(np.where(labels.reshape(image_rgb.shape[:2]) == i))
    if cluster_pixels.size > 0:
        # Compute mean position (row, column) -> convert to (x, y)
        mean_position = cluster_pixels.mean(axis=0).astype(int)
        text_position = (int(mean_position[1]), int(mean_position[0]))  # (x, y)

        # Put text at computed position
        cv2.putText(boundary_overlay, f"Cluster {i + 1}", text_position, font, 0.7, (255, 255, 255), 2)

# Step 10: Plot the original and the boundary-overlayed segmented images
fig, axes = plt.subplots(1, 2, figsize=(12, 6))

# Original image
axes[0].imshow(image_rgb)
axes[0].set_title("Original Image")
axes[0].axis('off')

# Segmented image with boundaries
axes[1].imshow(boundary_overlay)
axes[1].set_title("Segmented Image with Cluster Boundaries and Numbers")
axes[1].axis('off')

plt.tight_layout()
plt.show()
```




```
import numpy as np
import cv2
import os
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from tensorflow.keras.datasets import cifar10
# Load CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
# Step 1: Function to extract color histogram features from an image
def extract_color_histogram(image, bins=32):
 # Convert image to HSV color space
 hsv_image = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)

 # Compute the histogram for each channel (Hue, Saturation, Value)
 hist_hue = cv2.calcHist([hsv_image], [0], None, [bins], [0, 256]) # Hue
 hist_saturation = cv2.calcHist([hsv_image], [1], None, [bins], [0, 256]) # Saturation
 hist_value = cv2.calcHist([hsv_image], [2], None, [bins], [0, 256]) # Value

 # Normalize histograms and flatten them into a single feature vector
 hist_hue = hist_hue / hist_hue.sum()
 hist_saturation = hist_saturation / hist_saturation.sum()
 hist_value = hist_value / hist_value.sum()

 return np.concatenate([hist_hue.flatten(), hist_saturation.flatten(), hist_value.flatten()])
# Step 2: Extract features from CIFAR-10 images
features = []
for image in x_train[:100]: # Take a subset of images (100 samples for faster processing)
 feature_vector = extract_color_histogram(image)
 features.append(feature_vector)
features = np.array(features)
# Step 3: Apply K-Means clustering on the extracted features
k = 10 # Number of clusters (same as the number of classes in CIFAR-10)
kmeans = KMeans(n_clusters=k, random_state=42)
kmeans.fit(features)
# Step 4: Assign each image to its corresponding cluster
image_labels = kmeans.labels_
# Step 5: Visualize the images grouped by clusters
fig, axes = plt.subplots(k, 1, figsize=(10, 10))
axes = axes.flatten()
for i in range(k):
 # Find images belonging to the current cluster
 cluster_images = [x_train[j] for j in range(len(image_labels)) if image_labels[j] == i]

 # Display the first image in each cluster
 axes[i].imshow(cluster_images[0])
 axes[i].set_title(f"Cluster {i + 1} - Example Image")
 axes[i].axis('off')
plt.tight_layout()
plt.show()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
```

Cluster 1 - Example Image

Cluster 2 - Example Image

Cluster 3 - Example Image

Cluster 4 - Example Image

Cluster 5 - Example Image

Cluster 6 - Example Image

Cluster 7 - Example Image

Cluster 8 - Example Image

Cluster 9 - Example Image

Cluster 10 - Example Image

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

# Load the Oxford 102 Flowers dataset (only the training split)
dataset, info = tfds.load('oxford_flowers102', split='train', with_info=True, as_supervised=True)

# Resize and extract a subset (for faster processing)
def preprocess_image(image, label, target_size=(64, 64)):
    image = tfds.as_numpy(image)
    image = cv2.resize(image, target_size)
    return image

images = []
for i, (img, label) in enumerate(dataset.take(100)):  # Take only 100 images
    img = preprocess_image(img, label)
    images.append(img)
images = np.array(images)

# Function to extract color histogram features
def extract_color_histogram(image, bins=32):
    hsv_image = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
```

```
    hist_hue = cv2.calcHist([hsv_image], [0], None, [bins], [0, 256])
    hist_saturation = cv2.calcHist([hsv_image], [1], None, [bins], [0, 256])
    hist_value = cv2.calcHist([hsv_image], [2], None, [bins], [0, 256])
    hist_hue = hist_hue / hist_hue.sum()
    hist_saturation = hist_saturation / hist_saturation.sum()
    hist_value = hist_value / hist_value.sum()
    return np.concatenate([hist_hue.flatten(), hist_saturation.flatten(), hist_value.flatten()])

# Extract features
features = [extract_color_histogram(img) for img in images]
features = np.array(features)

# Apply K-Means clustering
k = 10
kmeans = KMeans(n_clusters=k, random_state=42)
kmeans.fit(features)
image_labels = kmeans.labels_

# Visualize clustered images
fig, axes = plt.subplots(k, 1, figsize=(10, 15))
axes = axes.flatten()
for i in range(k):
    cluster_images = [images[j] for j in range(len(image_labels)) if image_labels[j] == i]
    axes[i].imshow(cluster_images[0])
    axes[i].set_title(f"Cluster {i + 1} - Example Image")
    axes[i].axis('off')
plt.tight_layout()
plt.show()
```

```
WARNING:absl:Variant folder /root/tensorflow_datasets/oxford_flowers102/2.1.1 has no dataset_info.json
Downloading and preparing dataset Unknown size (download: Unknown size, generated: Unknown size, total: Unknown size
```

Dl Completed...: 100% 3/3 [00:36<00:00, 5.08s/ url]

Dl Size...: 100% 328/328 [00:36<00:00, 27.12 MiB/s]

Extraction completed...: 100% 8189/8189 [00:36<00:00, 488.63 file/s]

```
Dataset oxford_flowers102 downloaded and prepared to /root/tensorflow_datasets/oxford_flowers102/2.1.1. Subsequent ca
```

Cluster 1 - Example Image

Cluster 2 - Example Image

Cluster 3 - Example Image

Cluster 4 - Example Image

Cluster 5 - Example Image

Cluster 6 - Example Image

Cluster 7 - Example Image

Cluster 8 - Example Image

Cluster 9 - Example Image

Cluster 10 - Example Image

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from tensorflow.keras.datasets import mnist

# Load MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Step 1: Function to extract grayscale histogram features from an image
def extract_gray_histogram(image, bins=32):
    # Compute grayscale histogram
    hist = cv2.calcHist([image], [0], None, [bins], [0, 256])
    hist = hist / hist.sum()  # Normalize
    return hist.flatten()

# Step 2: Extract features from MNIST images
features = []
for image in x_train[:100]:  # Use only 100 samples for faster processing
    feature_vector = extract_gray_histogram(image)
    features.append(feature_vector)

features = np.array(features)

# Step 3: Apply K-Means clustering
k = 20
# Number of clusters (0-9 digits)
kmeans = KMeans(n_clusters=k, random_state=42)
kmeans.fit(features)

# Step 4: Assign each image to its corresponding cluster
image_labels = kmeans.labels_

# Step 5: Visualize the images grouped by clusters
fig, axes = plt.subplots(k, 1, figsize=(8, 15))
axes = axes.flatten()

for i in range(k):
    # Find images belonging to the current cluster
    cluster_images = [x_train[j] for j in range(len(image_labels)) if image_labels[j] == i]

    # Display the first image in each cluster
    axes[i].imshow(cluster_images[0], cmap='gray')
    axes[i].set_title(f"Cluster {i + 1} - Example Image")
    axes[i].axis('off')

plt.tight_layout()
plt.show()
```