Assignment4Final

December 14, 2023

[ ]:
```
# Resizing and cropping each image to a 224 × 224 pixel image.

import os
import xml.etree.ElementTree as ET
from PIL import Image
import cv2
import random
import matplotlib.pyplot as plt
from skimage import exposure
import numpy as np
from scipy.spatial import distance
from scipy.stats import wasserstein_distance
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from skimage import io, color
```

[ ]:
```
# Cropping and resizing the images for Beagle Data set (Data set 1)

annotations_dir = 'D:\Dataset\Beagle annonations\\n02088364-beagle'
annotations_files = os.listdir(annotations_dir)

# Step 2: Load the Images Dataset
images_dir = 'D:\Dataset\Beaglepng'
output_dir = 'D:\Dataset\Beagle 224'

if not os.path.exists(output_dir):
    os.makedirs(output_dir)

# Step 3: Parse XML Data and Crop & Resize Images
for xml_file in annotations_files:
    xml_path = os.path.join(annotations_dir, xml_file)
    image_filename = os.path.splitext(xml_file)[0] + '.png'
    image_path = os.path.join(images_dir, image_filename)

    if os.path.exists(image_path):
        tree = ET.parse(xml_path)
        root = tree.getroot()
        # Extract bounding box coordinates
        xmin = int(root.find(".//xmin").text)
        ymin = int(root.find(".//ymin").text)
        xmax = int(root.find(".//xmax").text)
        ymax = int(root.find(".//ymax").text)

        # Load and crop the image
        image = Image.open(image_path)
        cropped_image = image.crop((xmin, ymin, xmax, ymax))

        # Resize the cropped image to 224x224 pixels
        resized_image = cropped_image.resize((224,224), Image.ANTIALIAS)

        # Save the resized image as PNG
        output_path = os.path.join(output_dir, image_filename)
        resized_image.save(output_path, "PNG")

        print(f"Processed: {image_filename}")

print("All images processed and saved.")


# Cropping and resizing the images for Dhole Data set (Data set 2)

annotations_dir = 'D:\Dataset\Dhole annonations\\n02115913-dhole'
annotations_files = os.listdir(annotations_dir)

# Step 2: Load the Images Dataset
images_dir = 'D:\Dataset\Dholepng'
output_dir = 'D:\Dataset\Dhole 224'

if not os.path.exists(output_dir):
    os.makedirs(output_dir)

# Step 3: Parse XML Data and Crop & Resize Images
for xml_file in annotations_files:
    xml_path = os.path.join(annotations_dir, xml_file)
    image_filename = os.path.splitext(xml_file)[0] + '.png'
    image_path = os.path.join(images_dir, image_filename)

    if os.path.exists(image_path):
        tree = ET.parse(xml_path)
        root = tree.getroot()

        # Extract bounding box coordinates
        xmin = int(root.find(".//xmin").text)
        ymin = int(root.find(".//ymin").text)
        xmax = int(root.find(".//xmax").text)
        ymax = int(root.find(".//ymax").text)

        # Load and crop the image
        image = Image.open(image_path)
        cropped_image = image.crop((xmin, ymin, xmax, ymax))

        # Resize the cropped image to 224x224 pixels
        resized_image = cropped_image.resize((224,224), Image.ANTIALIAS)

        # Save the resized image as PNG
        output_path = os.path.join(output_dir, image_filename)
        resized_image.save(output_path, "PNG")

        print(f"Processed: {image_filename}")

print("All images processed and saved.")



# Cropping and resizing the images for Golden retriever Data set (Data set 3)

annotations_dir = 'D:\Dataset\Golden Retriever␣
 ↪annonations\\n02099601-golden_retriever'
annotations_files = os.listdir(annotations_dir)

# Step 2: Load the Images Dataset
images_dir = 'D:\Dataset\Golden Retrieverpng'
output_dir = 'D:\Dataset\Golden retriever 224'

if not os.path.exists(output_dir):
    os.makedirs(output_dir)

# Step 3: Parse XML Data and Crop & Resize Images
for xml_file in annotations_files:
    xml_path = os.path.join(annotations_dir, xml_file)
    image_filename = os.path.splitext(xml_file)[0] + '.png'
    image_path = os.path.join(images_dir, image_filename)

    if os.path.exists(image_path):
        tree = ET.parse(xml_path)
        root = tree.getroot()

        # Extract bounding box coordinates
        xmin = int(root.find(".//xmin").text)
        ymin = int(root.find(".//ymin").text)
        xmax = int(root.find(".//xmax").text)
        ymax = int(root.find(".//ymax").text)

        # Load and crop the image
        image = Image.open(image_path)
        cropped_image = image.crop((xmin, ymin, xmax, ymax))

        # Resize the cropped image to 224x224 pixels
        resized_image = cropped_image.resize((224,224), Image.ANTIALIAS)

        # Save the resized image as PNG
        output_path = os.path.join(output_dir, image_filename)
        resized_image.save(output_path, "PNG")

        print(f"Processed: {image_filename}")

print("All images processed and saved.")







# Cropping and resizing the images for Great Pyreness Data set (Data set 4)

annotations_dir = 'D:\Dataset\Great Pyreness␣
  ↪annonations\\n02111500-Great_Pyrenees'
annotations_files = os.listdir(annotations_dir)

# Step 2: Load the Images Dataset
images_dir = 'D:\Dataset\Great pyrenesspng'
output_dir = 'D:\Dataset\Great Pyreness 224'

if not os.path.exists(output_dir):
    os.makedirs(output_dir)

# Step 3: Parse XML Data and Crop & Resize Images
for xml_file in annotations_files:
    xml_path = os.path.join(annotations_dir, xml_file)
    image_filename = os.path.splitext(xml_file)[0] + '.png'
    image_path = os.path.join(images_dir, image_filename)

    if os.path.exists(image_path):
        tree = ET.parse(xml_path)
        root = tree.getroot()

        # Extract bounding box coordinates
        xmin = int(root.find(".//xmin").text)
        ymin = int(root.find(".//ymin").text)
        xmax = int(root.find(".//xmax").text)
        ymax = int(root.find(".//ymax").text)

        # Load and crop the image
        image = Image.open(image_path)
        cropped_image = image.crop((xmin, ymin, xmax, ymax))

        # Resize the cropped image to 224x224 pixels
        resized_image = cropped_image.resize((224,224), Image.ANTIALIAS)

        # Save the resized image as PNG
        output_path = os.path.join(output_dir, image_filename)
        resized_image.save(output_path, "PNG")

        print(f"Processed: {image_filename}")

print("All images processed and saved.")
```

[ ]:
```
# Normalize the resized image dataset.
from sklearn.preprocessing import StandardScaler

# Function to normalize a given dataset
def normalize_images(dataset_path, output_path):
    image_files = os.listdir(dataset_path)

    if not os.path.exists(output_path):
        os.makedirs(output_path)

    # Load, normalize, and save each image
    for image_file in image_files:
        image_path = os.path.join(dataset_path, image_file)

        if os.path.exists(image_path):
            # Load the image
            image = Image.open(image_path)

            # Convert the image to a NumPy array
            image_array = np.array(image)

            # Flatten the array to a 1D vector
            flattened_array = image_array.flatten().reshape(1, -1)

            # Use StandardScaler to normalize the pixel values
            scaler = StandardScaler()
            normalized_array = scaler.fit_transform(flattened_array)
            # Reshape the normalized array back to the original shape
            normalized_image_array = normalized_array.reshape(image_array.shape)

            # Convert the array back to an image
            normalized_image = Image.fromarray(normalized_image_array.
 ↪astype('uint8'))

            # Save the normalized image
            output_image_path = os.path.join(output_path, image_file)
            normalized_image.save(output_image_path, "PNG")



# Normalize the Beagle dataset
beagle_resized_path = 'D:\Dataset\Beagle 224'
beagle_normalized_path = 'D:\Dataset\Beagle 224 Normalized'
normalize_images(beagle_resized_path, beagle_normalized_path)

# Normalize the Dhole dataset
dhole_resized_path = 'D:\Dataset\Dhole 224'
dhole_normalized_path = 'D:\Dataset\Dhole 224 Normalized'
normalize_images(dhole_resized_path, dhole_normalized_path)

# Normalize the Golden Retriever dataset
golden_retriever_resized_path = 'D:\Dataset\Golden retriever 224'
golden_retriever_normalized_path = 'D:\Dataset\Golden retriever 224 Normalized'
normalize_images(golden_retriever_resized_path,␣
 ↪golden_retriever_normalized_path)

# Normalize the Great Pyrenees dataset
great_pyrenees_resized_path = 'D:\Dataset\Great Pyreness 224'
great_pyrenees_normalized_path = 'D:\Dataset\Great Pyreness 224 Normalized'
normalize_images(great_pyrenees_resized_path, great_pyrenees_normalized_path)
```

[ ]:
```
import os
import numpy as np
from PIL import Image
import torch
import torch.nn as nn
import torchvision.transforms as transforms
from torchvision import models
from sklearn.decomposition import PCA
from sklearn.cluster import AgglomerativeClustering
import matplotlib.pyplot as plt
```

```
[ ]: # Reference: https://kozodoi.me/blog/20210527/extracting-features
     # Function to extract features from the last convolutional layer of ResNet18
     def extract_resnet18_features(dataset_path):
         image_files = os.listdir(dataset_path)
         PREDS = []  # To store Labels
         FEATS = []  # To store Features

         # Load the pre-trained ResNet18 model
         model = models.resnet18(pretrained=True)

         # Remove the fully connected layers (classification head)
         model = nn.Sequential(*list(model.children())[:-2])
         model.eval()  # Set the model to evaluation mode

         # Define the transformation to be applied to each image
         transform = transforms.Compose([
             transforms.ToTensor(),
             transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.
      ↪225]),
         ])

         # Extract features for each image
         for image_file in image_files:
             image_path = os.path.join(dataset_path, image_file)

             if os.path.exists(image_path):
                 # Load and preprocess the image
                 image = Image.open(image_path).convert('RGB')
                 image = transform(image)
                 image = image.unsqueeze(0)

                 # Forward pass through the model
                 with torch.no_grad():
                     features = model(image)

                 # Flatten the feature tensor
                 feature_vector = features.mean([2, 3]).squeeze().numpy()

                 # Store the feature vector and label
                 FEATS.append(feature_vector)
                 PREDS.append(os.path.basename(dataset_path))


         return np.array(FEATS), np.array(PREDS)

     # Define the paths for all 4 datasets
     beagle_resized_path = 'D:\Dataset\Beagle 224 Normalized'
```

```
dhole_resized_path = 'D:\Dataset\Dhole 224 Normalized'
golden_retriever_resized_path = 'D:\Dataset\Golden retriever 224 Normalized'
great_pyrenees_resized_path = 'D:\Dataset\Great Pyreness 224 Normalized'

# Extract features for each dataset
beagle_feats, beagle_preds = extract_resnet18_features(beagle_resized_path)
dhole_feats, dhole_preds = extract_resnet18_features(dhole_resized_path)
golden_retriever_feats, golden_retriever_preds =␣
  ↪extract_resnet18_features(golden_retriever_resized_path)
great_pyrenees_feats, great_pyrenees_preds =␣
  ↪extract_resnet18_features(great_pyrenees_resized_path)

# Combine features and labels for all datasets
all_feats = np.concatenate((beagle_feats, dhole_feats, golden_retriever_feats,␣
  ↪great_pyrenees_feats), axis=0)
all_preds = np.concatenate((beagle_preds, dhole_preds, golden_retriever_preds,␣
  ↪great_pyrenees_preds), axis=0)

print('Labels shape:', all_preds.shape)
print('Features shape:', all_feats.shape)

# Perform PCA to reduce dimensionality to 2D
pca = PCA(n_components=2)
reduced_feats = pca.fit_transform(all_feats)

print("Reduced features shape:",reduced_feats.shape)
```

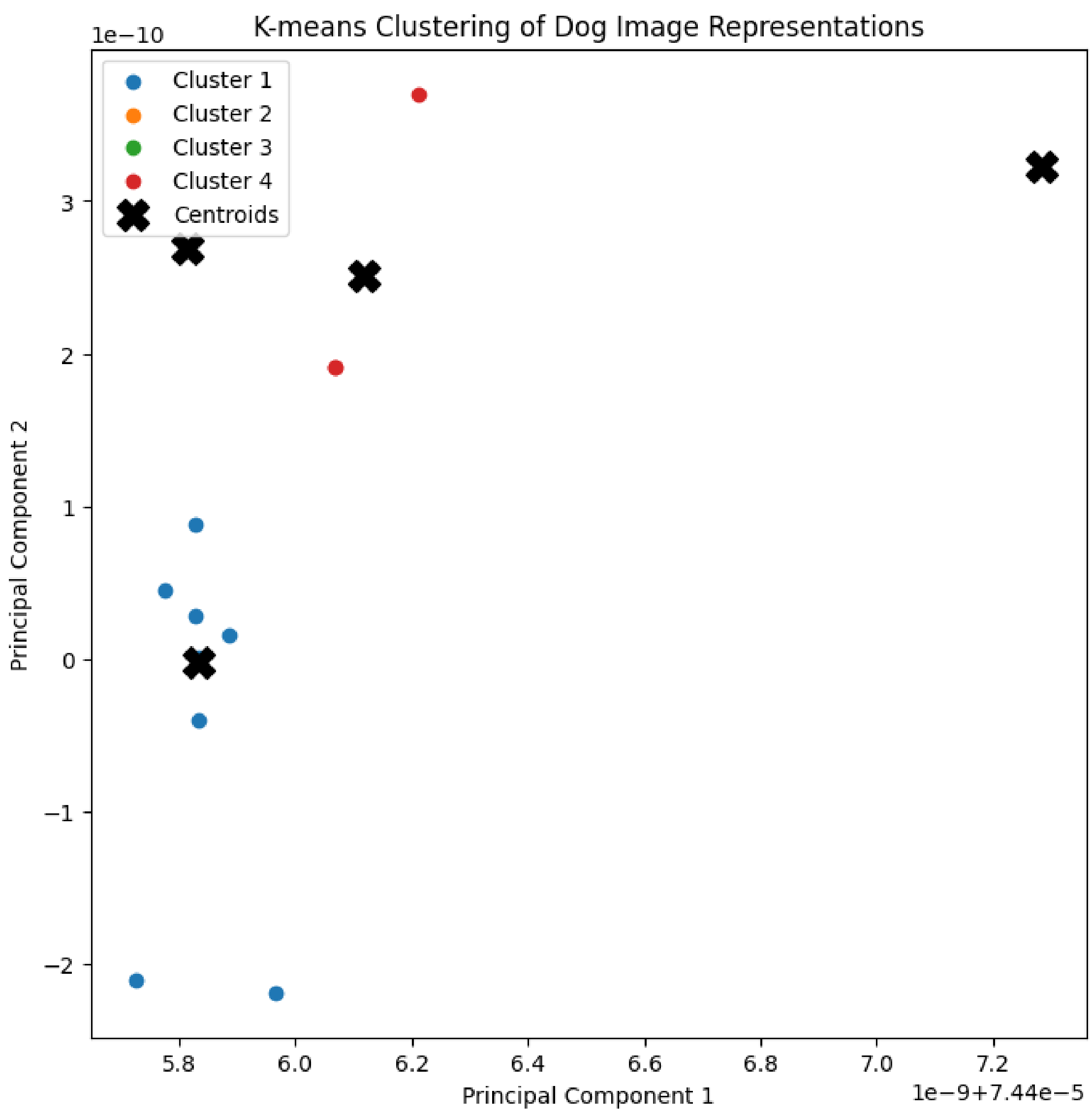
```
c:\Users\User\AppData\Local\Programs\Python\Python39\lib\site-
packages\torchvision\models\_utils.py:208: UserWarning: The parameter
'pretrained' is deprecated since 0.13 and may be removed in the future, please
use 'weights' instead.
  warnings.warn(
c:\Users\User\AppData\Local\Programs\Python\Python39\lib\site-
packages\torchvision\models\_utils.py:223: UserWarning: Arguments other than a
weight enum or `None` for 'weights' are deprecated since 0.13 and may be removed
in the future. The current behavior is equivalent to passing
`weights=ResNet18_Weights.IMAGENET1K_V1`. You can also use
`weights=ResNet18_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
```

```
Labels shape: (708,)
Features shape: (708, 512)
Reduced features shape: (708, 2)
```

[ ]:
```
# CLustering
from sklearn.cluster import KMeans
# K-means clustering with init='random'
kmeans = KMeans(n_clusters=4, init='random', random_state=42)
```

```python
cluster_labels = kmeans.fit_predict(reduced_feats)

# Visualizing the K-means clustering results
plt.figure(figsize=(8, 8))
for cluster_label in np.unique(cluster_labels):
    indices = np.where(cluster_labels == cluster_label)
    plt.scatter(reduced_feats[indices, 0], reduced_feats[indices, 1],␣
  ↪label=f'Cluster {cluster_label + 1}')

plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],␣
  ↪marker='X', s=200, c='black', label='Centroids')
plt.title('K-means Clustering of Dog Image Representations')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.show()
```

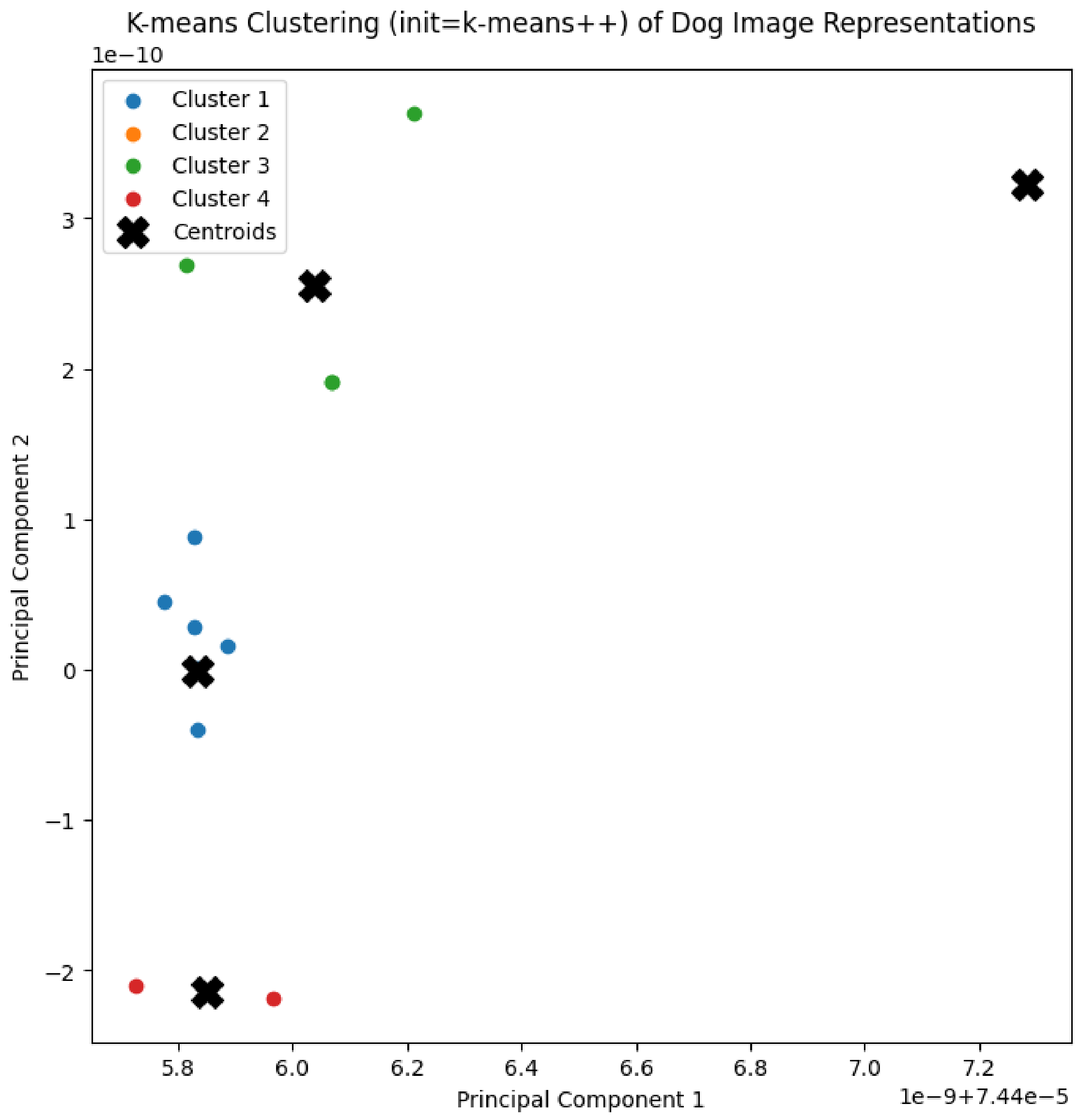
```
c:\Users\User\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init`
explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```



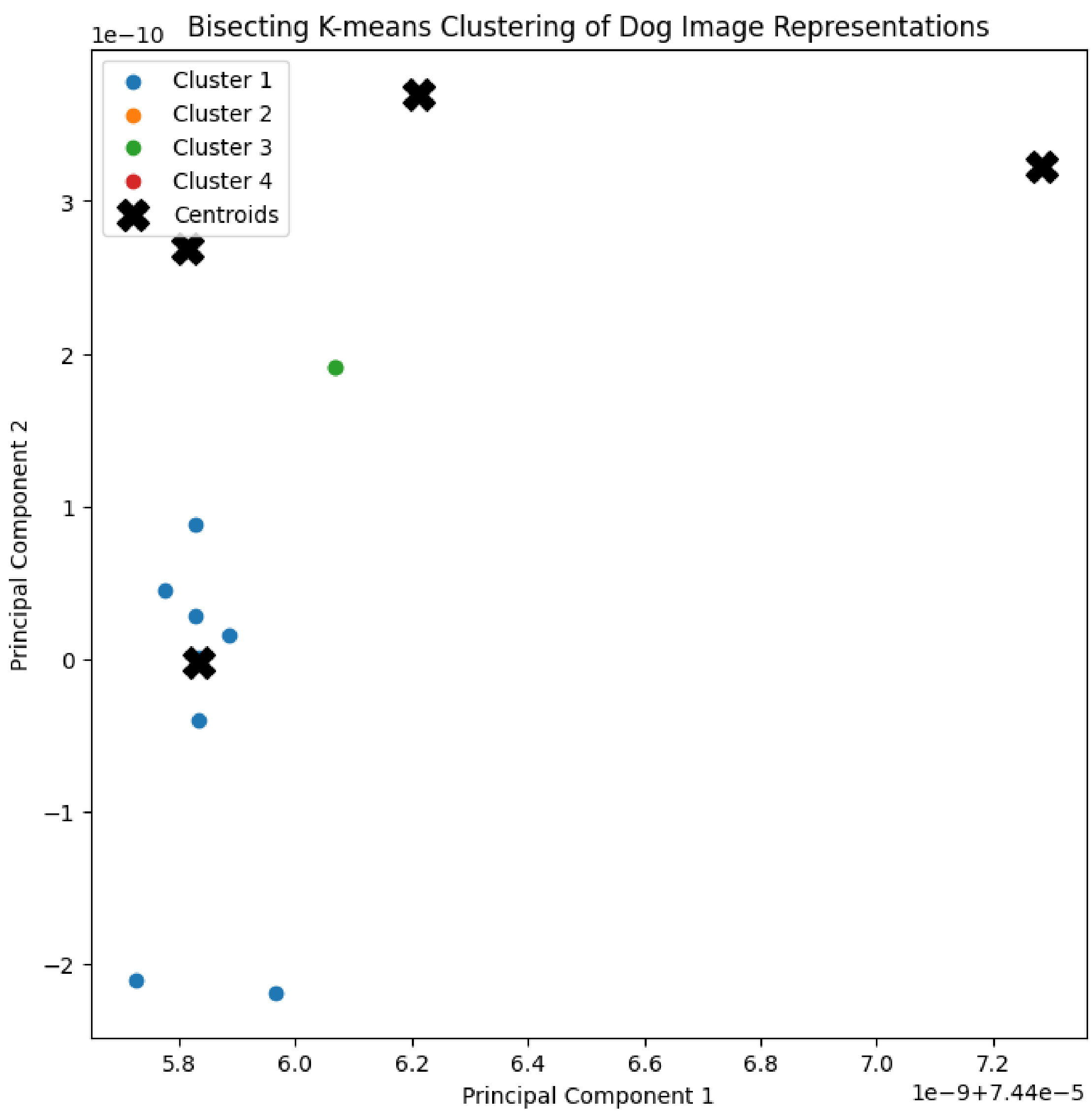
```
[ ]: # K-means clustering with init='k-means++'
     kmeans_pp = KMeans(n_clusters=4, init='k-means++', random_state=42)
     cluster_labels_pp = kmeans_pp.fit_predict(reduced_feats)

     # Visualizing the K-means clustering results with k-means++ initialization
     plt.figure(figsize=(8, 8))
     for cluster_label in np.unique(cluster_labels_pp):
         indices = np.where(cluster_labels_pp == cluster_label)
         plt.scatter(reduced_feats[indices, 0], reduced_feats[indices, 1],␣
       ↪label=f'Cluster {cluster_label + 1}')
```

```
plt.scatter(kmeans_pp.cluster_centers_[:, 0], kmeans_pp.cluster_centers_[:, 1],␣
  ↪marker='X', s=200, c='black', label='Centroids')
plt.title('K-means Clustering (init=k-means++) of Dog Image Representations')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.show()
```

```
c:\Users\User\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init`
explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

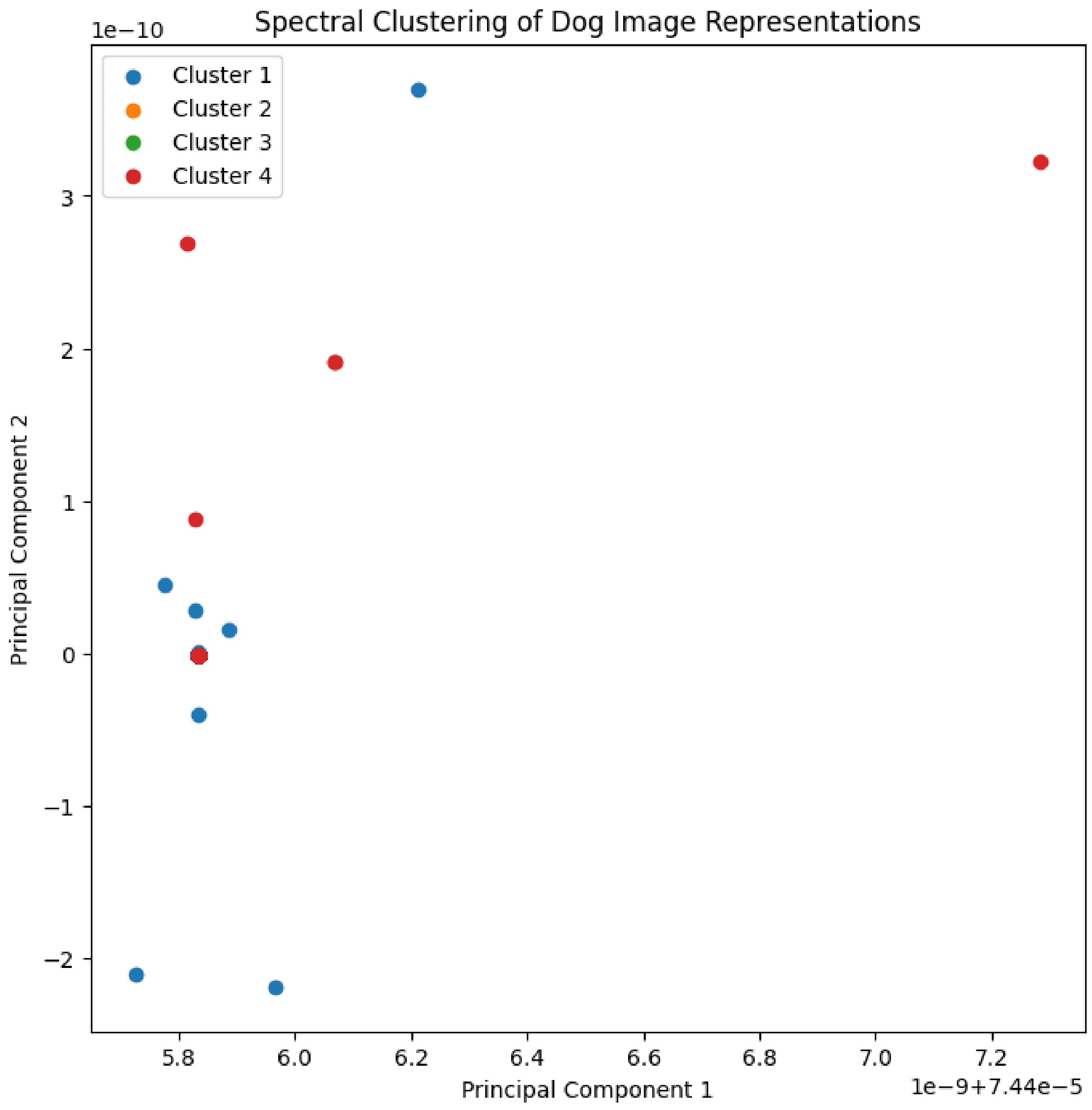


```
[ ]: from sklearn.cluster import BisectingKMeans
     # Bisecting K-means clustering
     bisecting_kmeans = BisectingKMeans(n_clusters=4, init='random', random_state=42)
     cluster_labels_bisecting = bisecting_kmeans.fit_predict(reduced_feats)

     # Visualizing the Bisecting K-means clustering results
     plt.figure(figsize=(8, 8))
     for cluster_label in np.unique(cluster_labels_bisecting):
         indices = np.where(cluster_labels_bisecting == cluster_label)
         plt.scatter(reduced_feats[indices, 0], reduced_feats[indices, 1],␣
      ↪label=f'Cluster {cluster_label + 1}')

     plt.scatter(bisecting_kmeans.cluster_centers_[:, 0], bisecting_kmeans.
      ↪cluster_centers_[:, 1], marker='X', s=200, c='black', label='Centroids')
     plt.title('Bisecting K-means Clustering of Dog Image Representations')
     plt.xlabel('Principal Component 1')
     plt.ylabel('Principal Component 2')
     plt.legend()
     plt.show()
```



```
[ ]: from sklearn.cluster import SpectralClustering
     # Spectral Clustering with default parameters
     spectral_clustering = SpectralClustering(n_clusters=4, random_state=42)
     cluster_labels_spectral = spectral_clustering.fit_predict(reduced_feats)

     # Visualizing the Spectral Clustering results
     plt.figure(figsize=(8, 8))
     for cluster_label in np.unique(cluster_labels_spectral):
         indices = np.where(cluster_labels_spectral == cluster_label)
         plt.scatter(reduced_feats[indices, 0], reduced_feats[indices, 1],␣
       ↪label=f'Cluster {cluster_label + 1}')
```

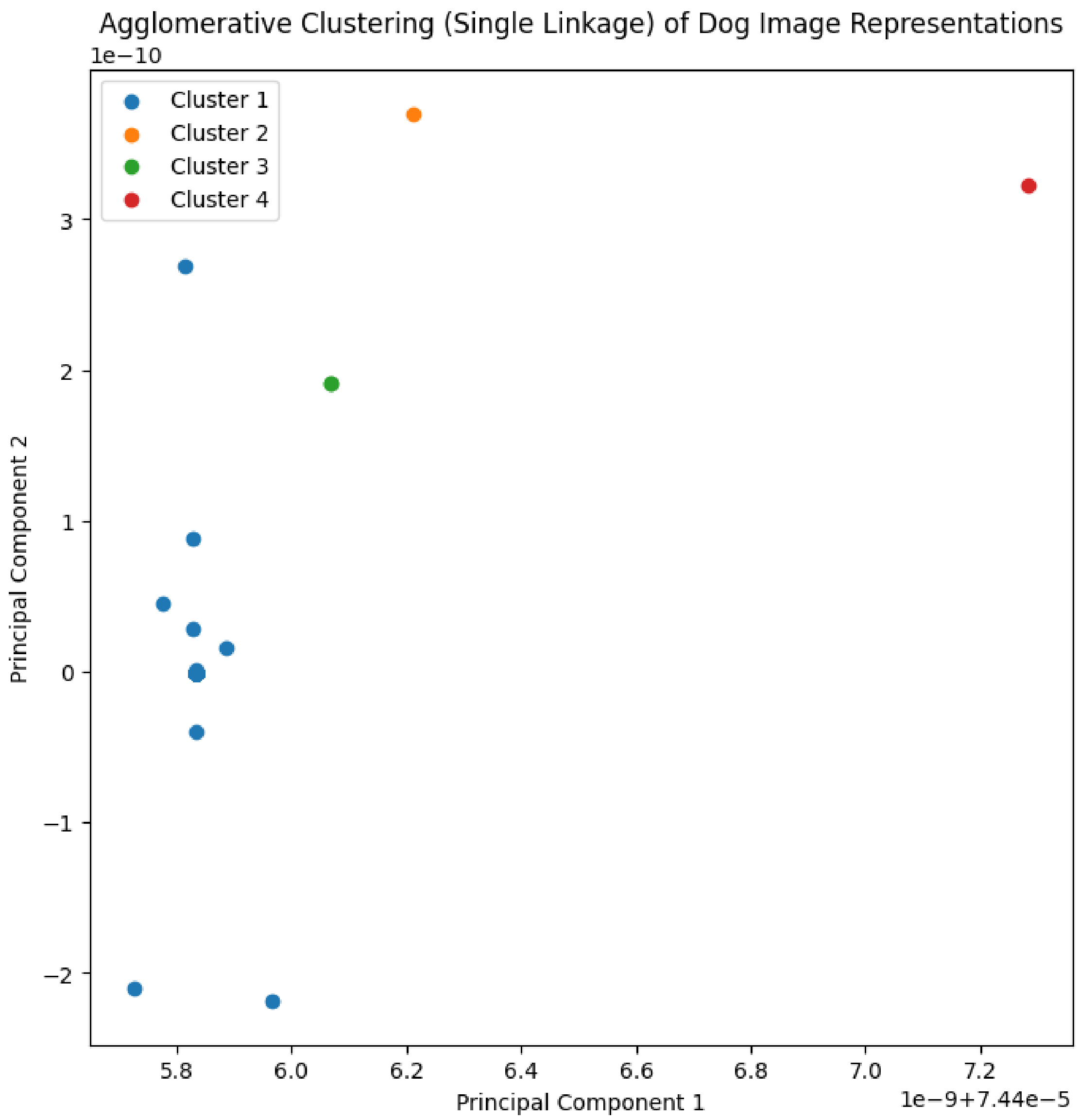
```
plt.title('Spectral Clustering of Dog Image Representations')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.show()
```



[ ]:
```
from sklearn.cluster import DBSCAN


# DBSCAN
for eps in [0.5]:
    for min_samples in [3]:
        dbscan = DBSCAN(eps=eps, min_samples=min_samples)
        labels = dbscan.fit_predict(reduced_feats)

        # Check the number of unique labels
        unique_labels = np.unique(all_preds)
        print(f"For eps={eps}, min_samples={min_samples}, Number of clusters:␣
  ↪{len(unique_labels)}")
```

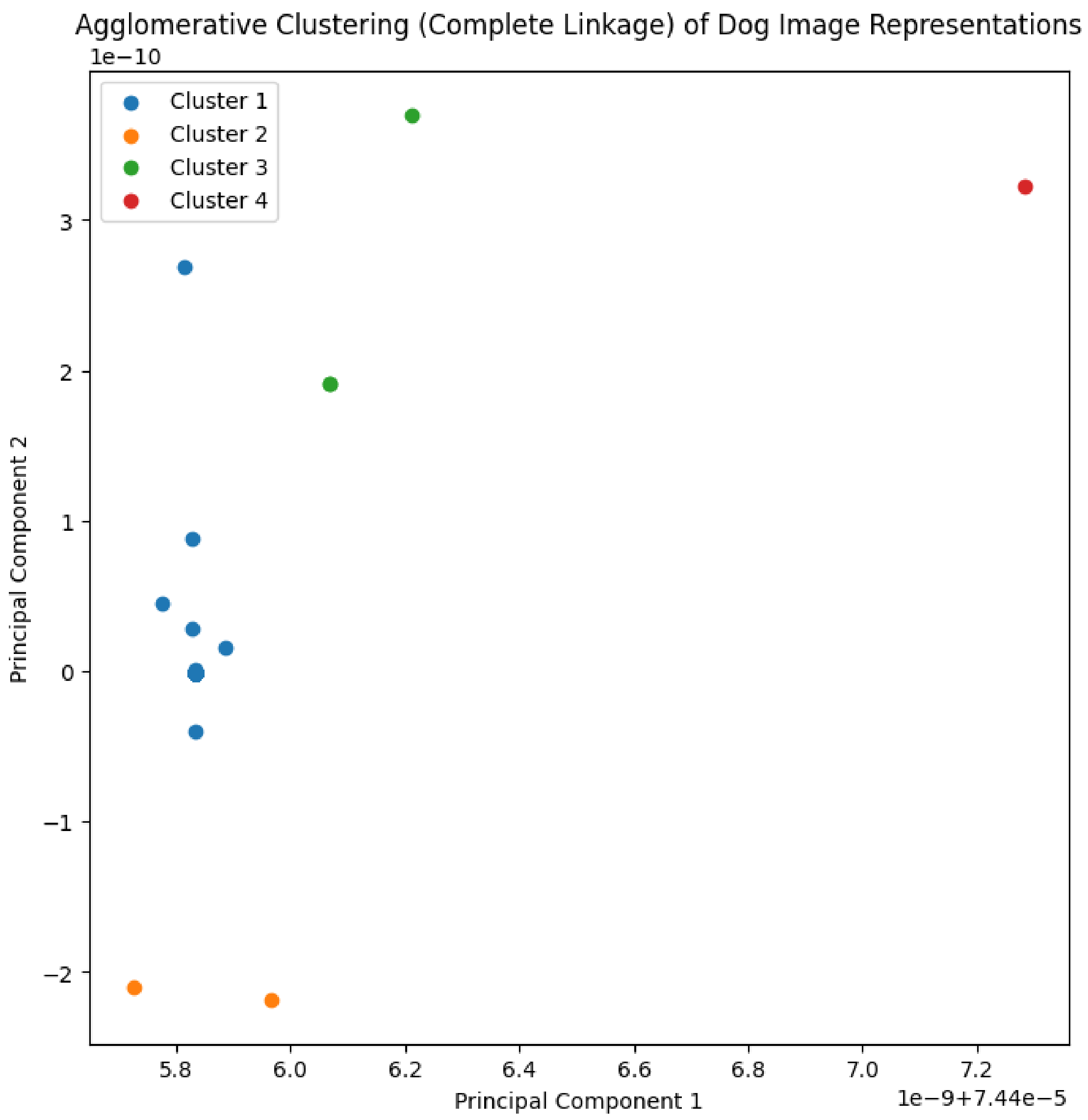
```
For eps=0.5, min_samples=3, Number of clusters: 4
```

[ ]:
```
# Agglomerative Clustering with 'single' linkage
agglomerative_single = AgglomerativeClustering(n_clusters=4, linkage='single')
cluster_labels_agglomerative_single = agglomerative_single.
  ↪fit_predict(reduced_feats)

# Visualizing the Agglomerative Clustering (Single Linkage) results
plt.figure(figsize=(8, 8))
for cluster_label in np.unique(cluster_labels_agglomerative_single):
    indices = np.where(cluster_labels_agglomerative_single == cluster_label)
    plt.scatter(reduced_feats[indices, 0], reduced_feats[indices, 1],␣
  ↪label=f'Cluster {cluster_label + 1}')

plt.title('Agglomerative Clustering (Single Linkage) of Dog Image␣
  ↪Representations')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.show()
```

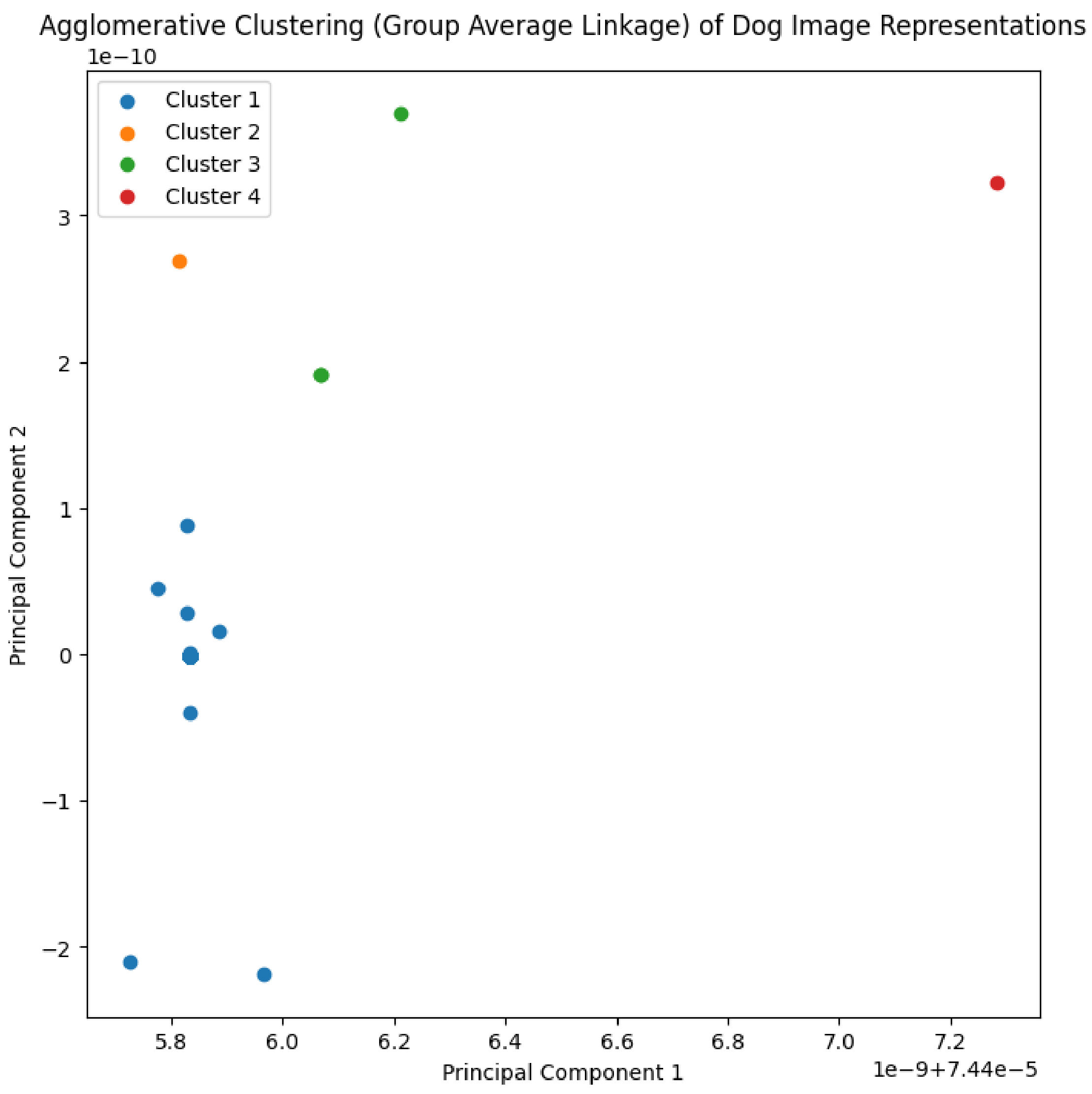


```
[ ]: # Agglomerative Clustering with 'complete' (MAX) linkage
     agglomerative_complete = AgglomerativeClustering(n_clusters=4,␣
      ↪linkage='complete')
     cluster_labels_agglomerative_complete = agglomerative_complete.
      ↪fit_predict(reduced_feats)

     # Visualizing the Agglomerative Clustering (Complete Linkage) results
     plt.figure(figsize=(8, 8))
     for cluster_label in np.unique(cluster_labels_agglomerative_complete):
         indices = np.where(cluster_labels_agglomerative_complete == cluster_label)
```

```
    plt.scatter(reduced_feats[indices, 0], reduced_feats[indices, 1],␣
  ↪label=f'Cluster {cluster_label + 1}')

plt.title('Agglomerative Clustering (Complete Linkage) of Dog Image␣
  ↪Representations')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.show()
```

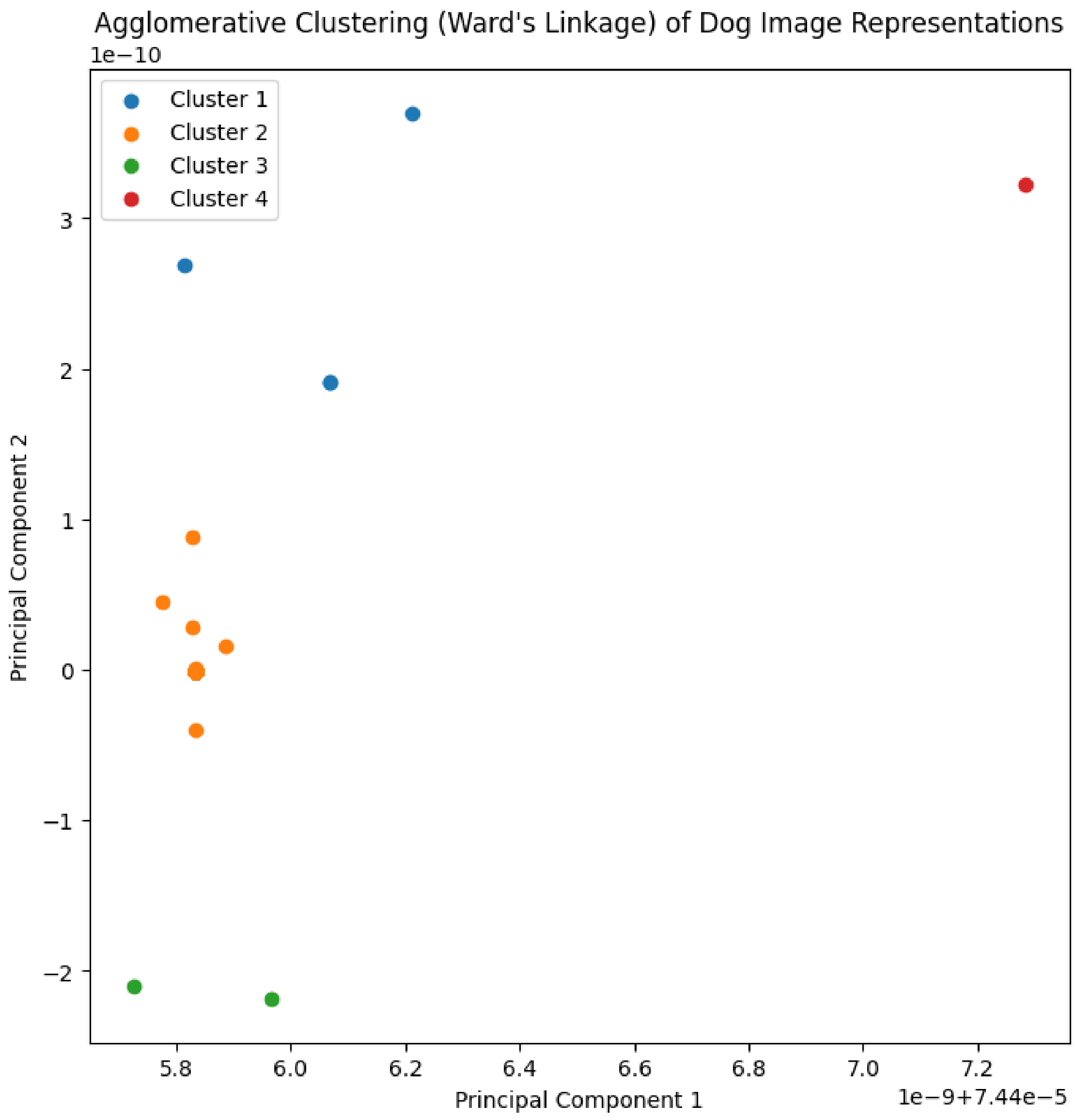


```
[ ]: # Agglomerative Clustering with 'average' linkage
agglomerative_average = AgglomerativeClustering(n_clusters=4, linkage='average')
cluster_labels_agglomerative_average = agglomerative_average.
 ↪fit_predict(reduced_feats)

# Visualizing the Agglomerative Clustering (Group Average Linkage) results
plt.figure(figsize=(8, 8))
for cluster_label in np.unique(cluster_labels_agglomerative_average):
    indices = np.where(cluster_labels_agglomerative_average == cluster_label)
    plt.scatter(reduced_feats[indices, 0], reduced_feats[indices, 1],␣
 ↪label=f'Cluster {cluster_label + 1}')

plt.title('Agglomerative Clustering (Group Average Linkage) of Dog Image␣
 ↪Representations')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.show()
```



[ ]:
```python
# Agglomerative Clustering with 'ward' linkage
agglomerative_ward = AgglomerativeClustering(n_clusters=4, linkage='ward')
cluster_labels_agglomerative_ward = agglomerative_ward.
 ↪fit_predict(reduced_feats)

# Visualizing the Agglomerative Clustering (Ward's Linkage) results
plt.figure(figsize=(8, 8))
for cluster_label in np.unique(cluster_labels_agglomerative_ward):
    indices = np.where(cluster_labels_agglomerative_ward == cluster_label)
    plt.scatter(reduced_feats[indices, 0], reduced_feats[indices, 1],␣
 ↪label=f'Cluster {cluster_label + 1}')
```

```
plt.title("Agglomerative Clustering (Ward's Linkage) of Dog Image␣
  ↪Representations")
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.show()
```



[ ]:
```
import numpy as np
from sklearn.metrics import fowlkes_mallows_score

true_labels = np.concatenate((np.zeros(beagle_feats.shape[0]),
                              np.ones(dhole_feats.shape[0]),
                              2 * np.ones(golden_retriever_feats.shape[0]),
                              3 * np.ones(great_pyrenees_feats.shape[0])))

# function to evaluate and print Fowlkes-Mallows index
def evaluate_fowlkes_mallows(method_name, predicted_labels):
    score = fowlkes_mallows_score(true_labels, predicted_labels)
    print(f"Fowlkes-Mallows index for {method_name}: {score}")

# Evaluate for K-means (init='random')
evaluate_fowlkes_mallows("K-means (init='random')", cluster_labels)

# Evaluate for K-means (init='k-means++')
evaluate_fowlkes_mallows("K-means (init='k-means++')", cluster_labels_pp)

# Evaluate for Bisecting K-means
evaluate_fowlkes_mallows("Bisecting K-means", cluster_labels_bisecting)

# Evaluate for Spectral Clustering
evaluate_fowlkes_mallows("Spectral Clustering", cluster_labels_spectral)

# Evaluate for Agglomerative Clustering (Single Linkage)
evaluate_fowlkes_mallows("Agglomerative Clustering (Single Linkage)",␣
 ↪cluster_labels_agglomerative_single)

# Evaluate for Agglomerative Clustering (Complete Linkage)
evaluate_fowlkes_mallows("Agglomerative Clustering (Complete Linkage)",␣
 ↪cluster_labels_agglomerative_complete)

# Evaluate for Agglomerative Clustering (Group Average Linkage)
evaluate_fowlkes_mallows("Agglomerative Clustering (Group Average Linkage)",␣
 ↪cluster_labels_agglomerative_average)

# Evaluate for Agglomerative Clustering (Ward's Linkage)
evaluate_fowlkes_mallows("Agglomerative Clustering (Ward's Linkage)",␣
 ↪cluster_labels_agglomerative_ward)
```

```
Fowlkes-Mallows index for K-means (init='random'): 0.5010296961472338
Fowlkes-Mallows index for K-means (init='k-means++'): 0.4994758234603681
Fowlkes-Mallows index for Bisecting K-means: 0.5010296961472338
Fowlkes-Mallows index for Spectral Clustering: 0.35255621518337577
Fowlkes-Mallows index for Agglomerative Clustering (Single Linkage):
0.5018152540885414
Fowlkes-Mallows index for Agglomerative Clustering (Complete Linkage):
0.50024291569637
Fowlkes-Mallows index for Agglomerative Clustering (Group Average Linkage):
0.5010296961472338
```

```
Fowlkes-Mallows index for Agglomerative Clustering (Ward's Linkage):
0.4994758234603681
```

[ ]:
```
from sklearn.metrics import silhouette_score

# function to evaluate and print Silhouette Coefficient
def evaluate_silhouette_coefficient(method_name, predicted_labels,␣
 ↪feature_matrix):
    # Silhouette Coefficient requires more than one cluster
    if len(set(predicted_labels)) > 1:
        score = silhouette_score(feature_matrix, predicted_labels)
        print(f"Silhouette Coefficient for {method_name}: {score}")
    else:
        print(f"Silhouette Coefficient for {method_name}: Not applicable␣
 ↪(requires more than one cluster)")

# Evaluate for K-means (init='random')
evaluate_silhouette_coefficient("K-means (init='random')", cluster_labels,␣
 ↪reduced_feats)

# Evaluate for K-means (init='k-means++')
evaluate_silhouette_coefficient("K-means (init='k-means++')",␣
 ↪cluster_labels_pp, reduced_feats)

# Evaluate for Bisecting K-means
evaluate_silhouette_coefficient("Bisecting K-means", cluster_labels_bisecting,␣
 ↪reduced_feats)

# Evaluate for Spectral Clustering
evaluate_silhouette_coefficient("Spectral Clustering", cluster_labels_spectral,␣
 ↪reduced_feats)

# Evaluate for Agglomerative Clustering (Single Linkage)
evaluate_silhouette_coefficient("Agglomerative Clustering (Single Linkage)",␣
 ↪cluster_labels_agglomerative_single, reduced_feats)

# Evaluate for Agglomerative Clustering (Complete Linkage)
evaluate_silhouette_coefficient("Agglomerative Clustering (Complete Linkage)",␣
 ↪cluster_labels_agglomerative_complete, reduced_feats)

# Evaluate for Agglomerative Clustering (Group Average Linkage)
evaluate_silhouette_coefficient("Agglomerative Clustering (Group Average␣
 ↪Linkage)", cluster_labels_agglomerative_average, reduced_feats)

# Evaluate for Agglomerative Clustering (Ward's Linkage)
evaluate_silhouette_coefficient("Agglomerative Clustering (Ward's Linkage)",␣
 ↪cluster_labels_agglomerative_ward, reduced_feats)
```

```
Silhouette Coefficient for K-means (init='random'): 0.9876362681388855
Silhouette Coefficient for K-means (init='k-means++'): 0.9884199500083923
Silhouette Coefficient for Bisecting K-means: 0.9876362681388855
Silhouette Coefficient for Spectral Clustering: -0.9139062762260437
Silhouette Coefficient for Agglomerative Clustering (Single Linkage):
0.9877052903175354
Silhouette Coefficient for Agglomerative Clustering (Complete Linkage):
0.9879322052001953
Silhouette Coefficient for Agglomerative Clustering (Group Average Linkage):
0.9876362681388855
Silhouette Coefficient for Agglomerative Clustering (Ward's Linkage):
0.9884199500083923
```

[ ]:
```
# Collecting Fowlkes-Mallows scores for each method
method_scores = {
    "K-means (init='random')": fowlkes_mallows_score(true_labels,␣
 ↪cluster_labels),
    "K-means (init='k-means++')": fowlkes_mallows_score(true_labels,␣
 ↪cluster_labels_pp),
    "Bisecting K-means": fowlkes_mallows_score(true_labels,␣
 ↪cluster_labels_bisecting),
    "Spectral Clustering": fowlkes_mallows_score(true_labels,␣
 ↪cluster_labels_spectral),
    "Agglomerative (Single Linkage)": fowlkes_mallows_score(true_labels,␣
 ↪cluster_labels_agglomerative_single),
    "Agglomerative (Complete Linkage)": fowlkes_mallows_score(true_labels,␣
 ↪cluster_labels_agglomerative_complete),
    "Agglomerative (Group Average Linkage)": fowlkes_mallows_score(true_labels,␣
 ↪cluster_labels_agglomerative_average),
    "Agglomerative (Ward's Linkage)": fowlkes_mallows_score(true_labels,␣
 ↪cluster_labels_agglomerative_ward),
}

# Sorting the methods based on their Fowlkes-Mallows scores
sorted_methods = sorted(method_scores.items(), key=lambda x: x[1], reverse=True)

# Print the ranked methods
print("Ranking based on Fowlkes-Mallows index:")
for rank, (method, score) in enumerate(sorted_methods, start=1):
    print(f"{rank}. {method}: {score}")
```

```
Ranking based on Fowlkes-Mallows index:
1. Agglomerative (Single Linkage): 0.5018152540885414
2. K-means (init='random'): 0.5010296961472338
3. Bisecting K-means: 0.5010296961472338
4. Agglomerative (Group Average Linkage): 0.5010296961472338
5. Agglomerative (Complete Linkage): 0.50024291569637
```

```
6. K-means (init='k-means++'): 0.4994758234603681
7. Agglomerative (Ward's Linkage): 0.4994758234603681
8. Spectral Clustering: 0.35255621518337577
```

[ ]:
```
# Collecting Silhouette Coefficient scores for each method
method_silhouette_scores = {
    "K-means (init='random')": silhouette_score(reduced_feats, cluster_labels),
    "K-means (init='k-means++')": silhouette_score(reduced_feats,␣
  ↪cluster_labels_pp),
    "Bisecting K-means": silhouette_score(reduced_feats,␣
  ↪cluster_labels_bisecting),
    "Spectral Clustering": silhouette_score(reduced_feats,␣
  ↪cluster_labels_spectral),
    "Agglomerative (Single Linkage)": silhouette_score(reduced_feats,␣
  ↪cluster_labels_agglomerative_single),
    "Agglomerative (Complete Linkage)": silhouette_score(reduced_feats,␣
  ↪cluster_labels_agglomerative_complete),
    "Agglomerative (Group Average Linkage)": silhouette_score(reduced_feats,␣
  ↪cluster_labels_agglomerative_average),
    "Agglomerative (Ward's Linkage)": silhouette_score(reduced_feats,␣
  ↪cluster_labels_agglomerative_ward),
}

# Sorting the methods based on their Silhouette Coefficient scores
sorted_methods_silhouette = sorted(method_silhouette_scores.items(), key=lambda␣
  ↪x: x[1], reverse=True)

# Print the ranked methods
print("Ranking based on Silhouette Coefficient:")
for rank, (method, score) in enumerate(sorted_methods_silhouette, start=1):
    print(f"{rank}. {method}: {score}")
```

```
Ranking based on Silhouette Coefficient:
1. K-means (init='k-means++'): 0.9884199500083923
2. Agglomerative (Ward's Linkage): 0.9884199500083923
3. Agglomerative (Complete Linkage): 0.9879322052001953
4. Agglomerative (Single Linkage): 0.9877052903175354
5. K-means (init='random'): 0.9876362681388855
6. Bisecting K-means: 0.9876362681388855
7. Agglomerative (Group Average Linkage): 0.9876362681388855
8. Spectral Clustering: -0.9139062762260437
```