In [1]:
```
pip install torchvision
```

```
Requirement already satisfied: torchvision in c:\users\admin\anaconda3\lib\site-packages (0.20.1)
Requirement already satisfied: numpy in c:\users\admin\anaconda3\lib\site-packages (from torchvision) (1.26.4)
Requirement already satisfied: torch==2.5.1 in c:\users\admin\anaconda3\lib\site-packages (from torchvision) (2.5.1)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in c:\users\admin\anaconda3\lib\site-packages (from torchvision) (10.3.0)
Requirement already satisfied: filelock in c:\users\admin\anaconda3\lib\site-packages (from torch==2.5.1->torchvision) (3.13.1)
Requirement already satisfied: typing-extensions>=4.8.0 in c:\users\admin\anaconda3\lib\site-packages (from torch==2.5.1->torchvision) (4.11.0)
Requirement already satisfied: networkx in c:\users\admin\anaconda3\lib\site-packages (from torch==2.5.1->torchvision) (3.2.1)
Requirement already satisfied: jinja2 in c:\users\admin\anaconda3\lib\site-packages (from torch==2.5.1->torchvision) (3.1.4)
Requirement already satisfied: fsspec in c:\users\admin\anaconda3\lib\site-packages (from torch==2.5.1->torchvision) (2024.3.1)
Requirement already satisfied: setuptools in c:\users\admin\anaconda3\lib\site-packages (from torch==2.5.1->torchvision) (69.5.1)
Requirement already satisfied: sympy==1.13.1 in c:\users\admin\anaconda3\lib\site-packages (from torch==2.5.1->torchvision) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in c:\users\admin\anaconda3\lib\site-packages (from sympy==1.13.1->torch==2.5.1->torchvision) (1.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\admin\anaconda3\lib\site-packages (from jinja2->torch==2.5.1->torchvision) (2.1.3)
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```
import os, glob
import cv2
import xml.etree.ElementTree as ET
from PIL import Image
from pathlib import Path
from matplotlib import pyplot as plt
import numpy as np
import pandas as pd
import torch
from torchvision import transforms
import torchvision.models as models
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans, SpectralClustering, AgglomerativeClustering, DBSCAN, BisectingKMeans
from sklearn.metrics import fowlkes_mallows_score, silhouette_score
from sklearn.preprocessing import LabelEncoder
from torchvision.models import resnet18, ResNet18_Weights
```

In [3]:
```
dog_images = glob.glob(r"C:\Users\ADMIN\Desktop\nidhi python\images\\*\\*")
breeds = glob.glob(r"C:\Users\ADMIN\Desktop\nidhi python\annotation\\*\\*")
annotations = glob.glob(r"C:\Users\ADMIN\Desktop\nidhi python\annotation\\*\\*")
```

In [4]:
```
def get_bounding_boxes(annot):
    xml = annot
    tree = ET.parse(xml)
    root = tree.getroot()
    objects = root.findall('object')
    bbox = []
    for o in objects:
        bndbox = o.find('bndbox')
        xmin = int(bndbox.find('xmin').text)
        ymin = int(bndbox.find('ymin').text)
        xmax = int(bndbox.find('xmax').text)
        ymax = int(bndbox.find('ymax').text)
        bbox.append((xmin,ymin,xmax,ymax))
    return bbox
```

In [5]:
```
def get_image(annot):
    img_path = r"C:\Users\ADMIN\Desktop\nidhi python\images"
    file = annot.split('\\')
    img_filename = img_path + "\\" + file[-2]+"\\"+file[-1]+'.jpg'
    return img_filename
```

In [6]:
```
for i in range(len(dog_images)):
    bbox = get_bounding_boxes(annotations[i])
    dog = get_image(annotations[i])
    im = Image.open(dog)
    for j in range(len(bbox)):
        im2 = im.crop(bbox[j])
        im2 = im2.resize((224,224), Image.Resampling.LANCZOS)
        new_path = dog.replace(r"C:\Users\ADMIN\Desktop\nidhi python\images",
                               r"C:\Users\ADMIN\Desktop\nidhi python\Cropped_224_v1")
        new_path = new_path.replace('.jpg',"-" + str(j) + '.jpg')
        im2=im2.convert('RGB')
        head, tail = os.path.split(new_path)
        Path(head).mkdir(parents=True, exist_ok=True)
        im2.save(new_path)
```

In [7]:
```
cropped_images_path = r"C:\Users\ADMIN\Desktop\nidhi python\Cropped_224_v1"
dog_classes = ['n02092002-Scottish_deerhound','n02093991-Irish_terrier','n02097474-Tibetan_terrier','n02106166-Border_collie']
```

In [8]:
```
dog_images = []
labels = []
for breed in dog_classes:
    images = glob.glob(cropped_images_path + "\\" + breed + "\\*")
    for image in images:
        image_array = cv2.imread(image, cv2.IMREAD_COLOR)
        normalized_image = ((image_array - image_array.mean()) / image_array.std())
        dog_images.append(normalized_image)
        labels.append(breed)
label_encoder = LabelEncoder()
labels = label_encoder.fit_transform(labels)
```

In [9]:
```
X_tensor = [transforms.functional.to_tensor(img) for img in dog_images]
model = resnet18(weights=ResNet18_Weights.IMAGENET1K_V1)
model.eval()
```

Out[9]:
```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=1000, bias=True)
)
```

In [10]:
```
features = {}
def get_features(name):
    def hook(model, input, output):
        features[name] = output.detach()
    return hook
model.layer4[1].conv2.register_forward_hook(get_features('final_conv_outputs'))
```

Out[10]:
```
<torch.utils.hooks.RemovableHandle at 0x1841b7b7e30>
```

In [11]:
```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model.to(device)
final_conv_features = []
with torch.no_grad():
    for img in X_tensor:
        img = img.unsqueeze(0).to(device).float()
        model(img)
        final_conv_features.append(features['final_conv_outputs'].cpu())
```

In [12]:
```
# Flatten the features for PCA
flattened_features = [f.view(f.size(0), -1).numpy() for f in final_conv_features]
flattened_features = np.concatenate(flattened_features, axis=0)
# PCA to reduce dimensions to 2
pca = PCA(n_components=2)
X_pca = pca.fit_transform(flattened_features)
```

In [13]:
```
# K-means with random initialization
kmeans_random = KMeans(n_clusters=4, init='random').fit(X_pca)
 # K-means with k-means++ initialization
kmeans_plus = KMeans(n_clusters=4, init='k-means++').fit(X_pca)
 # Bisecting K-means
bisect_kmeans = BisectingKMeans(n_clusters=4, init='random').fit(X_pca)
 # Spectral Clustering
spectral = SpectralClustering(n_clusters=4).fit(X_pca)
```

In [14]:
```
eps = 0.18
min_samples = 5
dbscan = DBSCAN(eps=eps, min_samples=min_samples)
dbscan_labels = dbscan.fit_predict(X_pca)
print("Min sample: " + str(min_samples))
print('eps: ' + str(eps))
print("Number of clusters: " + str(len(np.unique(dbscan_labels))))
```

```
Min sample: 5
eps: 0.18
Number of clusters: 6
```

In [15]:
```
# Different linkage criteria
single_link = AgglomerativeClustering(n_clusters=4, linkage='single').fit(X_pca)
complete_link = AgglomerativeClustering(n_clusters=4, linkage='complete').fit(X_pca)
average_link = AgglomerativeClustering(n_clusters=4, linkage='average').fit(X_pca)
ward_link = AgglomerativeClustering(n_clusters=4, linkage='ward').fit(X_pca)
```

In [16]:
```
# Calculate Fowlkes-Mallows index and Silhouette Coefficient for each clustering method
 # K-means with random initialization
fm_kmeans_random = fowlkes_mallows_score(labels, kmeans_random.labels_)
silhouette_kmeans_random = silhouette_score(X_pca, kmeans_random.labels_)
 # K-means with k-means++ initialization
fm_kmeans_plus = fowlkes_mallows_score(labels, kmeans_plus.labels_)
silhouette_kmeans_plus = silhouette_score(X_pca, kmeans_plus.labels_)
 # Bisecting K-means
fm_bisect_kmeans = fowlkes_mallows_score(labels, bisect_kmeans.labels_)
silhouette_bisect_kmeans = silhouette_score(X_pca, bisect_kmeans.labels_)
 # Spectral Clustering
fm_spectral = fowlkes_mallows_score(labels, spectral.labels_)
silhouette_spectral = silhouette_score(X_pca, spectral.labels_)
 # DBSCAN
fm_dbscan = fowlkes_mallows_score(labels, dbscan.labels_)
silhouette_dbscan = silhouette_score(X_pca, dbscan.labels_)
 # Hierarchical Clustering - Single Link
fm_single_link = fowlkes_mallows_score(labels, single_link.labels_)
silhouette_single_link = silhouette_score(X_pca, single_link.labels_)
 # Hierarchical Clustering - Complete Link
fm_complete_link = fowlkes_mallows_score(labels, complete_link.labels_)
silhouette_complete_link = silhouette_score(X_pca, complete_link.labels_)
 # Hierarchical Clustering - Average Link
fm_average_link = fowlkes_mallows_score(labels, average_link.labels_)
silhouette_average_link = silhouette_score(X_pca, average_link.labels_)
 # Hierarchical Clustering - Ward's method
fm_ward_link = fowlkes_mallows_score(labels, ward_link.labels_)
silhouette_ward_link = silhouette_score(X_pca, ward_link.labels_)
```

In [17]:
```
# Storing the method names and scores in tuples
clustering_methods = [
    ("K-means (Random Init)", fm_kmeans_random, silhouette_kmeans_random),
    ("K-means (k-means++)", fm_kmeans_plus, silhouette_kmeans_plus),
    ("Bisecting K-means", fm_bisect_kmeans, silhouette_bisect_kmeans),
    ("Spectral Clustering", fm_spectral, silhouette_spectral),
    ("DBSCAN", fm_dbscan, silhouette_dbscan),
    ("Hierarchical Clustering (Single Link)", fm_single_link, silhouette_single_link),
    ("Hierarchical Clustering (Complete Link)", fm_complete_link, silhouette_complete_link),
    ("Hierarchical Clustering (Average Link)", fm_average_link, silhouette_average_link),
    ("Hierarchical Clustering (Ward's method)", fm_ward_link, silhouette_ward_link)
]
 # Sorting based on Fowlkes-Mallows index
fm_ranking = sorted(clustering_methods, key=lambda x: x[1], reverse=True)
 # Sorting based on Silhouette Coefficient
silhouette_ranking = sorted(clustering_methods, key=lambda x: x[2], reverse=True)
 # Printing the rankings
print("Ranking based on Fowlkes-Mallows Index:")
for rank, method in enumerate(fm_ranking, 1):
    print(f"{rank}. {method[0]} - Score: {method[1]}")
print("\nRanking based on Silhouette Coefficient:")
for rank, method in enumerate(silhouette_ranking, 1):
    print(f"{rank}. {method[0]} - Score: {method[2]}")
```

```
Ranking based on Fowlkes-Mallows Index:
1. K-means (k-means++) - Score: 0.7241245977782262
2. K-means (Random Init) - Score: 0.7235991742582496
3. Spectral Clustering - Score: 0.7103454240905379
4. Hierarchical Clustering (Average Link) - Score: 0.7095826556838983
5. Hierarchical Clustering (Ward's method) - Score: 0.7094914835067307
6. Hierarchical Clustering (Complete Link) - Score: 0.6524803115547986
7. Bisecting K-means - Score: 0.6475193717058999
8. Hierarchical Clustering (Single Link) - Score: 0.5037185534021091
9. DBSCAN - Score: 0.4831147714818596

Ranking based on Silhouette Coefficient:
1. K-means (Random Init) - Score: 0.5400379896916394
2. K-means (k-means++) - Score: 0.5396698117256165
3. Spectral Clustering - Score: 0.5383628606796265
4. Hierarchical Clustering (Ward's method) - Score: 0.5330082178115845
5. Hierarchical Clustering (Average Link) - Score: 0.5289923348698425
6. Hierarchical Clustering (Complete Link) - Score: 0.4729314148426056
7. Bisecting K-means - Score: 0.4598406235245209
8. Hierarchical Clustering (Single Link) - Score: -0.2424754649400711
9. DBSCAN - Score: -0.3579606435295105
```

In [ ]:

In [ ]: