In [ ]:
```
import torch
import torch.nn as nn
from torchvision.models import resnet18
import os
import xml.etree.ElementTree as ET
import cv2
from PIL import Image
from sklearn.decomposition import PCA
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from torch.utils.data import Dataset,DataLoader
from torchvision import transforms
from torchsummary import  summary
import timm
```

```
WARNING:tensorflow:From c:\Users\kaasa\AppData\Local\Programs\Python\Python310\lib\site-packages\keras\src\losses.py:
2976: The name tf.losses.sparse_softmax_cross_entropy is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cr
oss_entropy instead.
```

In [ ]:
```
# referred from https://pytorch.org/docs/stable/data.html#torch.utils.data.Dataset

labmap = {0: "n02089078-black-and-tan_coonhound",
          1: "n02091831-Saluki",
          2: "n02092002-Scottish_deerhound",
          3: "n02095314-wire-haired_fox_terrier"}

class CustomDataset(Dataset):
    def __init__(self, root_dir, transform=None):
        self.root_dir = root_dir
        self.transform = transform
        self.classes = sorted(os.listdir(root_dir))
        self.class_to_idx = {cls: idx for idx, cls in enumerate(self.classes)}
        self.images = self.load_images()

    def load_images(self):
        images = []
        for class_name in self.classes:
            class_path = os.path.join(self.root_dir, class_name)
            for filename in os.listdir(class_path):
```

```
                image_path = os.path.join(class_path, filename)
                images.append((image_path, self.class_to_idx[class_name]))
        return images

    def __len__(self):
        return len(self.images)

    def __getitem__(self, idx):
        img_path, label = self.images[idx]
        image = Image.open(img_path).convert('RGB')

        if self.transform:
            image = self.transform(image)

        return image, label


root_folder = '../DataSet/ProcessedDatasets/'
transform = transforms.Compose([transforms.Resize((224, 224)),
                                transforms.ToTensor(),
                                transforms.Normalize((0.5),(0.5))
                                ])

dog_dataset = CustomDataset(root_folder, transform=transform)

batch_size = 16
data_loader = DataLoader(dog_dataset, batch_size=batch_size, shuffle=True)

# Load the pre-trained ResNet-18 model
resnet_model = resnet18(pretrained=True)
# Remove the final fully connected layer
resnet_model = torch.nn.Sequential(*(list(resnet_model.children())[:-2]))
resnet_model = resnet_model.to('cuda')
# Set the model to evaluation mode
resnet_model.eval()
```

```
c:\Users\kaasa\AppData\Local\Programs\Python\Python310\lib\site-packages\torchvision\models\_utils.py:208: UserWarnin
g: The parameter 'pretrained' is deprecated since 0.13 and may be removed in the future, please use 'weights' instea
d.
  warnings.warn(
c:\Users\kaasa\AppData\Local\Programs\Python\Python310\lib\site-packages\torchvision\models\_utils.py:223: UserWarnin
g: Arguments other than a weight enum or `None` for 'weights' are deprecated since 0.13 and may be removed in the fut
ure. The current behavior is equivalent to passing `weights=ResNet18_Weights.IMAGENET1K_V1`. You can also use `weight
s=ResNet18_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
```

Out[ ]:
```
Sequential(
  (0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): ReLU(inplace=True)
  (3): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (4): 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)
    )
  )
  (5): 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)
    )
  )
  (6): 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)
      )
    )
    (7): 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)
      )
    )
  )
```

In [ ]:
```
#referred from https://pytorch.org/vision/stable/feature_extraction.html
#referred from https://kozodoi.me/blog/20210527/extracting-features

batch_size = 32
data_loader = DataLoader(dog_dataset, batch_size=batch_size, shuffle=False)

# Extract features using the pre-trained ResNet-18 model
all_features = []
all_labels = []


with torch.no_grad():
    for images, labels in data_loader:
        images, labels = images.to('cuda'), labels.to('cuda')
        features = resnet_model(images)
        all_features.append(features.cpu().squeeze().numpy())
        all_labels.append(labels.cpu().numpy())

# Concatenate features and labels
features_reshaped = np.concatenate(all_features, axis=0).reshape(-1, 512 * 7 * 7)
all_labels = np.concatenate(all_labels, axis=0)
```

In [ ]:
```
#referred from https://scikit-learn.org/0.15/modules/generated/sklearn.decomposition.PCA.html
pca = PCA(n_components=2)
principalComponents_dog = pca.fit_transform(features_reshaped)
```

In [ ]:
```
#referred from https://scikit-learn.org/stable/modules/clustering.html#overview-of-clustering-methods


from sklearn.cluster import KMeans, SpectralClustering, BisectingKMeans


# (a) K-means clustering
kmeans = KMeans(n_clusters=4, init='random')
kmeans_labels = kmeans.fit_predict(principalComponents_dog)

# (b) KMeans with init='k-means++'
kmeans_pp = KMeans(n_clusters=4, init='k-means++')
kmeans_pp_labels = kmeans_pp.fit_predict(principalComponents_dog)

# (c) Bisecting K-means
```

```
bisecting_kmeans = BisectingKMeans(n_clusters=4, init='random')
bisecting_kmeans_labels = bisecting_kmeans.fit_predict(principalComponents_dog)

# (d) Spectral clustering
spectral_clustering = SpectralClustering(n_clusters=4)
spectral_labels = spectral_clustering.fit_predict(principalComponents_dog)
```

```
c:\Users\kaasa\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarni
ng: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning
  super()._check_params_vs_input(X, default_n_init=10)
c:\Users\kaasa\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarni
ng: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning
  super()._check_params_vs_input(X, default_n_init=10)
c:\Users\kaasa\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\manifold\_spectral_embedding.py:392:
UserWarning: Exited at iteration 217 with accuracies
[3.27773152e-13 3.00833888e-05 2.12821777e-05 1.23635816e-05
 2.02417273e-05]
not reaching the requested tolerance 2.3653836898059478e-05.
  _, diffusion_map = lobpcg(
c:\Users\kaasa\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\base.py:1152: ConvergenceWarning: Nu
mber of distinct clusters (2) found smaller than n_clusters (4). Possibly due to duplicate points in X.
  return fit_method(estimator, *args, **kwargs)
```

In [ ]:

```
#refered from https://scikit-learn.org/stable/modules/clustering.html#dbscan
#referred from https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html
#referred from https://scikit-learn.org/stable/modules/generated/sklearn.metrics.fowlkes_mallows_score.html
#referred from https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html

from sklearn.cluster import DBSCAN, AgglomerativeClustering
from sklearn.metrics import fowlkes_mallows_score, silhouette_score

# (e) DBSCAN
dbscan = DBSCAN(eps=0.99, min_samples=3)
dbscan_labels = dbscan.fit_predict(principalComponents_dog)
print(f"The no of clusters for eps 0.99 and min_samples 3 is {len(set(dbscan_labels))}")
#  Agglomerative clustering
# (f) Single link (MIN)
single_link = AgglomerativeClustering(n_clusters=4, linkage='single')
single_link_labels = single_link.fit_predict(principalComponents_dog)
```

```
# (g) Complete link (MAX)
complete_link = AgglomerativeClustering(n_clusters=4, linkage='complete')
complete_link_labels = complete_link.fit_predict(principalComponents_dog)

# (h) Group Average
group_average = AgglomerativeClustering(n_clusters=4, linkage='average')
group_average_labels = group_average.fit_predict(principalComponents_dog)

# (i) Ward's method
ward = AgglomerativeClustering(n_clusters=4, linkage='ward')
ward_labels = ward.fit_predict(principalComponents_dog)

# Clustering evaluation metrics
def evaluate_clustering(labels, true_labels):
    fowlkes_mallows = fowlkes_mallows_score(true_labels, labels)
    silhouette = silhouette_score(principalComponents_dog, labels)
    return fowlkes_mallows, silhouette

# Ground truth labels (assuming you have them)
true_labels = all_labels
```

```
The no of clusters for eps 0.99 and min_samples 3 is 4
```

To get 4 clusters for dbscan I used eps = 0.99 and min_samples = 3

In [ ]:
```
dbscan_scores = evaluate_clustering(dbscan_labels, true_labels)
# Evaluate Agglomerative clustering methods
single_link_scores = evaluate_clustering(single_link_labels, true_labels)
complete_link_scores = evaluate_clustering(complete_link_labels, true_labels)
group_average_scores = evaluate_clustering(group_average_labels, true_labels)
ward_scores = evaluate_clustering(ward_labels, true_labels)

# Print the evaluation scores
print("DBSCAN Scores (Fowlkes-Mallows, Silhouette):", dbscan_scores)
print("Single Link Scores (Fowlkes-Mallows, Silhouette):", single_link_scores)
print("Complete Link Scores (Fowlkes-Mallows, Silhouette):", complete_link_scores)
print("Group Average Scores (Fowlkes-Mallows, Silhouette):", group_average_scores)
print("Ward Scores (Fowlkes-Mallows, Silhouette):", ward_scores)

# Rank methods based on Fowlkes-Mallows index
methods_fm_rank = sorted([(dbscan_scores[0], 'DBSCAN'),
```

```
                           (single_link_scores[0], 'Single Link'),
                           (complete_link_scores[0], 'Complete Link'),
                           (group_average_scores[0], 'Group Average'),
                           (ward_scores[0], "Ward's Method")], reverse=True)

# Rank methods based on Silhouette Coefficient
methods_silhouette_rank = sorted([(dbscan_scores[1], 'DBSCAN'),
                                  (single_link_scores[1], 'Single Link'),
                                  (complete_link_scores[1], 'Complete Link'),
                                  (group_average_scores[1], 'Group Average'),
                                  (ward_scores[1], "Ward's Method")], reverse=True)

# Print the rankings
print("\nRankings based on Fowlkes-Mallows Index:")
for rank, method in enumerate(methods_fm_rank, 1):
    print(f"{rank}. {method[1]}")

print("\nRankings based on Silhouette Coefficient:")
for rank, method in enumerate(methods_silhouette_rank, 1):
    print(f"{rank}. {method[1]}")
```

```
DBSCAN Scores (Fowlkes-Mallows, Silhouette): (0.5005859868587448, -0.2579051)
Single Link Scores (Fowlkes-Mallows, Silhouette): (0.5037631189299173, 0.034081228)
Complete Link Scores (Fowlkes-Mallows, Silhouette): (0.5657348052505528, 0.42517817)
Group Average Scores (Fowlkes-Mallows, Silhouette): (0.5948711505892431, 0.45314533)
Ward Scores (Fowlkes-Mallows, Silhouette): (0.5952182342715089, 0.44958904)

Rankings based on Fowlkes-Mallows Index:
1. Ward's Method
2. Group Average
3. Complete Link
4. Single Link
5. DBSCAN

Rankings based on Silhouette Coefficient:
1. Group Average
2. Ward's Method
3. Complete Link
4. Single Link
5. DBSCAN
```

References

1. https://pytorch.org/docs/stable/data.html#torch.utils.data.Dataset
2. https://pytorch.org/vision/stable/feature_extraction.html
3. https://kozodoi.me/blog/20210527/extracting-features
4. https://scikit-learn.org/0.15/modules/generated/sklearn.decomposition.PCA.html
5. https://scikit-learn.org/stable/modules/clustering.html#overview-of-clustering-methods
6. https://scikit-learn.org/stable/modules/clustering.html#dbscan
7. https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html
8. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.fowlkes_mallows_score.html
9. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html