```
from google.colab import drive

# Mount Google Drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

```
import torch
import torch.nn as nn
import torchvision.transforms as transforms
from torchvision import models
from torch.autograd import Variable
from sklearn.cluster import KMeans, SpectralClustering, AgglomerativeClustering, DBSCAN, Birch
from sklearn import metrics
import matplotlib.pyplot as plt
import os
from PIL import Image
import numpy as np

import warnings
warnings.filterwarnings("ignore")


# Set the path to your dataset
dataset_path = '/content/gdrive/MyDrive/DM1/Cropped'



# Function to extract features using ResNet18
def extract_resnet_features(img_path):
    resnet18 = models.resnet18(pretrained=True)
    resnet18 = torch.nn.Sequential(*(list(resnet18.children())[:-1]))
    resnet18.eval()

    transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
    ])

    img = Image.open(img_path).convert('RGB')
    img = transform(img)
    img = Variable(img.unsqueeze(0))
    # Register a forward hook to extract features from the last convolution layer
    features = []

    def hook(module, input, output):
        features.extend(output.flatten().cpu().detach().numpy())

    hook_handle = resnet18[-1].register_forward_hook(hook)

    with torch.no_grad():
        resnet18(img)

    hook_handle.remove()

    return features



# Extract features from all images in the dataset
features_list = []
labels = []

for breed_folder in os.listdir(dataset_path):
    breed_path = os.path.join(dataset_path, breed_folder)
    for img_file in os.listdir(breed_path):
        img_path = os.path.join(breed_path, img_file)
        features = extract_resnet_features(img_path)
        features_list.append(features)
        labels.append(breed_folder)
```

```
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to /root/.cache/torch/hub/checkpoints/resnet18-f37072fd.pth
100%|██████████| 44.7M/44.7M [00:00<00:00, 84.9MB/s]
```

```
# Convert lists to numpy arrays
features_array = np.array(features_list)
labels_array = np.array(labels)



# Perform dimensionality reduction
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
reduced_features = pca.fit_transform(features_array)



# Perform clustering using different algorithms
kmeans_random = KMeans(n_clusters=4, init='random').fit(reduced_features)
kmeans_kmeanspp = KMeans(n_clusters=4, init='k-means++').fit(reduced_features)
bisecting_kmeans = KMeans(n_clusters=2,  init='random', n_init=1).fit(reduced_features)
spectral_clustering = SpectralClustering(n_clusters=4).fit(reduced_features)
dbscan = DBSCAN(eps=0.5, min_samples=5).fit(reduced_features)
agglomerative_single = AgglomerativeClustering(n_clusters=4, linkage='single').fit(reduced_features)
agglomerative_complete = AgglomerativeClustering(n_clusters=4, linkage='complete').fit(reduced_features)
agglomerative_average = AgglomerativeClustering(n_clusters=4, linkage='average').fit(reduced_features)
agglomerative_ward = AgglomerativeClustering(n_clusters=4, linkage='ward').fit(reduced_features)



# Evaluate clustering performance
def evaluate_clustering_performance(true_labels, predicted_labels):
    fowlkes_mallows = metrics.fowlkes_mallows_score(true_labels, predicted_labels)
    silhouette_coefficient = metrics.silhouette_score(reduced_features, predicted_labels)

    return fowlkes_mallows, silhouette_coefficient


# Ground truth labels
true_labels = labels_array



# Predicted labels from different clustering algorithms
predicted_labels_kmeans_random = kmeans_random.labels_
predicted_labels_kmeans_kmeanspp = kmeans_kmeanspp.labels_
predicted_labels_bisecting_kmeans = bisecting_kmeans.labels_
predicted_labels_spectral_clustering = spectral_clustering.labels_
predicted_labels_dbscan = dbscan.labels_
predicted_labels_agglomerative_single = agglomerative_single.labels_
predicted_labels_agglomerative_complete = agglomerative_complete.labels_
predicted_labels_agglomerative_average = agglomerative_average.labels_
predicted_labels_agglomerative_ward = agglomerative_ward.labels_



 #Evaluate clustering performance
fowlkes_mallows_kmeans_random, silhouette_coefficient_kmeans_random = evaluate_clustering_performance(true_labels, predicted_labels_kmeans_r
fowlkes_mallows_kmeans_kmeanspp, silhouette_coefficient_kmeans_kmeanspp = evaluate_clustering_performance(true_labels, predicted_labels_kmea
fowlkes_mallows_bisecting_kmeans, silhouette_coefficient_bisecting_kmeans = evaluate_clustering_performance(true_labels, predicted_labels_bi
fowlkes_mallows_spectral_clustering, silhouette_coefficient_spectral_clustering = evaluate_clustering_performance(true_labels, predicted_lab
fowlkes_mallows_dbscan, silhouette_coefficient_dbscan = evaluate_clustering_performance(true_labels, predicted_labels_dbscan)
fowlkes_mallows_agglomerative_single, silhouette_coefficient_agglomerative_single = evaluate_clustering_performance(true_labels, predicted_l
fowlkes_mallows_agglomerative_complete, silhouette_coefficient_agglomerative_complete = evaluate_clustering_performance(true_labels, predict
fowlkes_mallows_agglomerative_average, silhouette_coefficient_agglomerative_average = evaluate_clustering_performance(true_labels, predicted
fowlkes_mallows_agglomerative_ward, silhouette_coefficient_agglomerative_ward = evaluate_clustering_performance(true_labels, predicted_label
```

```
# Print the results
print("Fowlkes-Mallows Index:")
print(f"KMeans (Random): {fowlkes_mallows_kmeans_random}")
print(f"KMeans (KMeans++): {fowlkes_mallows_kmeans_kmeanspp}")
print(f"Bisecting KMeans: {fowlkes_mallows_bisecting_kmeans}")
print(f"Spectral Clustering: {fowlkes_mallows_spectral_clustering}")
print(f"DBSCAN: {fowlkes_mallows_dbscan}")
print(f"Agglomerative (Single Link): {fowlkes_mallows_agglomerative_single}")
print(f"Agglomerative (Complete Link): {fowlkes_mallows_agglomerative_complete}")
print(f"Agglomerative (Group Average): {fowlkes_mallows_agglomerative_average}")
print(f"Agglomerative (Ward): {fowlkes_mallows_agglomerative_ward}")

print("\nSilhouette Coefficient:")
print(f"KMeans (Random): {silhouette_coefficient_kmeans_random}")
print(f"KMeans (KMeans++): {silhouette_coefficient_kmeans_kmeanspp}")
print(f"Bisecting KMeans: {silhouette_coefficient_bisecting_kmeans}")
print(f"Spectral Clustering: {silhouette_coefficient_spectral_clustering}")
print(f"DBSCAN: {silhouette_coefficient_dbscan}")
print(f"Agglomerative (Single Link): {silhouette_coefficient_agglomerative_single}")
print(f"Agglomerative (Complete Link): {silhouette_coefficient_agglomerative_complete}")
print(f"Agglomerative (Group Average): {silhouette_coefficient_agglomerative_average}")
print(f"Agglomerative (Ward): {silhouette_coefficient_agglomerative_ward}")
```

```
Fowlkes-Mallows Index:
KMeans (Random): 0.9724254389704909
KMeans (KMeans++): 0.9724254389704909
Bisecting KMeans: 0.6394037215005239
Spectral Clustering: 0.9762187170672412
DBSCAN: 0.5193180432481675
Agglomerative (Single Link): 0.5065662011209293
Agglomerative (Complete Link): 0.8728545961057501
Agglomerative (Group Average): 0.9595080441340221
Agglomerative (Ward): 0.972517881179286

Silhouette Coefficient:
KMeans (Random): 0.7113755345344543
KMeans (KMeans++): 0.7113755345344543
Bisecting KMeans: 0.4221753180027008
Spectral Clustering: 0.7105126976966858
DBSCAN: -0.12113349884748459
Agglomerative (Single Link): -0.3349498510360718
Agglomerative (Complete Link): 0.6016430258750916
Agglomerative (Group Average): 0.7076421976089478
Agglomerative (Ward): 0.7059998512268066
```

```
# Predicted labels
predicted_labels_bisecting_kmeans = bisecting_kmeans.labels_

# Evaluate clustering performance
fowlkes_mallows_bisecting_kmeans, silhouette_coefficient_bisecting_kmeans = evaluate_clustering_performance(true_labels, predicted_labels_bis

# Print the results
print("Fowlkes-Mallows Index:")
print(f"Bisecting KMeans: {fowlkes_mallows_bisecting_kmeans}")

print("\nSilhouette Coefficient:")
print(f"Bisecting KMeans: {silhouette_coefficient_bisecting_kmeans}")

# Perform clustering evaluation and ranking for all methods
methods = [
    ("KMeans (Random)", KMeans(n_clusters=4, init='random')),
    ("KMeans (KMeans++)", KMeans(n_clusters=4, init='k-means++')),
    ("Bisecting KMeans", KMeans(n_clusters=2, init='random', n_init=1)),
    ("Spectral Clustering", SpectralClustering(n_clusters=4)),
    ("DBSCAN", DBSCAN(eps=0.5, min_samples=5)),
    ("Agglomerative (Single Link)", AgglomerativeClustering(n_clusters=4, linkage='single')),
    ("Agglomerative (Complete Link)", AgglomerativeClustering(n_clusters=4, linkage='complete')),
    ("Agglomerative (Group Average)", AgglomerativeClustering(n_clusters=4, linkage='average')),
    ("Agglomerative (Ward)", AgglomerativeClustering(n_clusters=4, linkage='ward'))
]

fowlkes_mallows_scores = []
silhouette_coefficient_scores = []

for method_name, method in methods:
    predicted_labels = method.fit_predict(reduced_features)

    # Evaluate clustering performance
    fowlkes_mallows, silhouette_coefficient = evaluate_clustering_performance(true_labels, predicted_labels)
```

```
        fowlkes_mallows, silhouette_coefficient = evaluate_clustering_performance(true_labels, predicted_labels)
        fowlkes_mallows_scores.append(fowlkes_mallows)
        silhouette_coefficient_scores.append(silhouette_coefficient)

        # Print the results for each method
        print(f"\nResults for {method_name}:")
        print(f"Fowlkes-Mallows Index: {fowlkes_mallows}")
        print(f"Silhouette Coefficient: {silhouette_coefficient}")

    # Rank the methods based on Fowlkes-Mallows Index
    fowlkes_mallows_ranking = np.argsort(fowlkes_mallows_scores)[::-1]

    # Rank the methods based on Silhouette Coefficient
    silhouette_coefficient_ranking = np.argsort(silhouette_coefficient_scores)[::-1]

    print("\nRanking based on Fowlkes-Mallows Index:")
    for rank, idx in enumerate(fowlkes_mallows_ranking):
        print(f"{rank+1}. {methods[idx][0]}")

    print("\nRanking based on Silhouette Coefficient:")
    for rank, idx in enumerate(silhouette_coefficient_ranking):
        print(f"{rank+1}. {methods[idx][0]}")
```

```
Results for KMeans (Random):
Fowlkes-Mallows Index: 0.9724254389704909
Silhouette Coefficient: 0.7113755345344543

Results for KMeans (KMeans++):
Fowlkes-Mallows Index: 0.9724254389704909
Silhouette Coefficient: 0.7113755345344543

Results for Bisecting KMeans:
Fowlkes-Mallows Index: 0.6394037215005239
Silhouette Coefficient: 0.4221753180027008

Results for Spectral Clustering:
Fowlkes-Mallows Index: 0.9762187170672412
Silhouette Coefficient: 0.7105126976966858

Results for DBSCAN:
Fowlkes-Mallows Index: 0.5193180432481675
Silhouette Coefficient: -0.12113349884748459

Results for Agglomerative (Single Link):
Fowlkes-Mallows Index: 0.5065662011209293
Silhouette Coefficient: -0.3349498510360718

Results for Agglomerative (Complete Link):
Fowlkes-Mallows Index: 0.8728545961057501
Silhouette Coefficient: 0.6016430258750916

Results for Agglomerative (Group Average):
Fowlkes-Mallows Index: 0.9595080441340221
Silhouette Coefficient: 0.7076421976089478

Results for Agglomerative (Ward):
Fowlkes-Mallows Index: 0.972517881179286
Silhouette Coefficient: 0.7059998512268066

Ranking based on Fowlkes-Mallows Index:
1. Spectral Clustering
2. Agglomerative (Ward)
3. KMeans (KMeans++)
4. KMeans (Random)
5. Agglomerative (Group Average)
6. Agglomerative (Complete Link)
7. Bisecting KMeans
8. DBSCAN
9. Agglomerative (Single Link)

Ranking based on Silhouette Coefficient:
1. KMeans (KMeans++)
2. KMeans (Random)
3. Spectral Clustering
4. Agglomerative (Group Average)
5. Agglomerative (Ward)
6. Agglomerative (Complete Link)
7. Bisecting KMeans
8. DBSCAN
9. Agglomerative (Single Link)
```

References:

Dabbura, I. (2022, September 27). K-means Clustering: Algorithm, Applications, Evaluation Methods, and Drawbacks. Medium. https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a

Manna, S. (2021, January 6). Extracting Features from an Intermediate Layer of a Pretrained ResNet Model in PyTorch (Hard Way). Medium. https://medium.com/the-owl/extracting-features-from-an-intermediate-layer-of-a-pretrained-model-in-pytorch-c00589bda32b

Sharma, A. (2020, September 7). How to Master the Popular DBSCAN Clustering Algorithm for Machine Learning. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2020/09/how-dbscan-clustering-works/

sklearn.cluster.AgglomerativeClustering. (n.d.). Scikit-learn. https://scikit-learn/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html

Start coding or generate with AI.

+ Code + Text