```python
import torch
from torchvision import models, transforms
from PIL import Image
import os
import numpy as np

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

# Define feature extraction function
def extract_features(image_path):
    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]),
    ])
    image = Image.open(image_path).convert('RGB')
    image = transform(image).unsqueeze(0)
    with torch.no_grad():
        features = model(image)
    return features

# Extract features for all images in a folder
image_folder = "/content/drive/MyDrive/cropped"
features = []
for img in os.listdir(image_folder):
    # Check if the item is a file before processing
    image_path = os.path.join(image_folder, img)
    if os.path.isfile(image_path):  # Only process files, skip
directories
        features.append(extract_features(image_path))

print("Feature extraction completed.") # Optional: Print a message to
indicate completion
```

```
/usr/local/lib/python3.10/dist-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(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:2
23: 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)
```

```
Feature extraction completed.
```

```
import os
import torch
from torchvision import models, transforms
from PIL import Image
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans, SpectralClustering,
AgglomerativeClustering, DBSCAN
from sklearn.cluster import BisectingKMeans
from sklearn.metrics import fowlkes_mallows_score, silhouette_score

# Set the root directory for the dataset
dataset_dir = "/content/drive/MyDrive/cropped"  # Update this to your
directory path

# Step 1: Feature Extraction
# Load ResNet18 pre-trained model
model = models.resnet18(pretrained=True)
model.eval()

# Define feature extraction function
def extract_features(image_path):
    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]),
    ])
    image = Image.open(image_path).convert('RGB')
    image = transform(image).unsqueeze(0)
    with torch.no_grad():
        features = model(image)
    return features.flatten().numpy()

# Extract features and collect ground truth labels
features = []
labels = []
class_names = os.listdir(dataset_dir)  # Get subdirectory names
(classes)
for label, class_name in enumerate(class_names):
    class_dir = os.path.join(dataset_dir, class_name)
    for image_name in os.listdir(class_dir):
        image_path = os.path.join(class_dir, image_name)
        features.append(extract_features(image_path))
        labels.append(label)  # Assign numerical labels for each class

# Step 2: Dimensionality Reduction
# Perform PCA to reduce dimensionality to 2
pca = PCA(n_components=2)
```

```
reduced_features = pca.fit_transform(features)

# Step 3: Clustering Methods
# K-means clustering
kmeans_random = KMeans(n_clusters=4, init='random',
random_state=42).fit(reduced_features)
kmeans_plus = KMeans(n_clusters=4, init='k-means++',
random_state=42).fit(reduced_features)

# Bisecting K-means
bisect_kmeans = BisectingKMeans(n_clusters=4,
random_state=42).fit(reduced_features)

# Spectral clustering
spectral = SpectralClustering(n_clusters=4, random_state=42,
affinity='nearest_neighbors').fit(reduced_features)

# DBSCAN
dbscan = DBSCAN(eps=0.5, min_samples=5).fit(reduced_features)

# Agglomerative clustering
agg_single = AgglomerativeClustering(n_clusters=4,
linkage='single').fit(reduced_features)
agg_complete = AgglomerativeClustering(n_clusters=4,
linkage='complete').fit(reduced_features)
agg_average = AgglomerativeClustering(n_clusters=4,
linkage='average').fit(reduced_features)
agg_ward = AgglomerativeClustering(n_clusters=4,
linkage='ward').fit(reduced_features)

# Step 4: Clustering Evaluation
# Evaluate each clustering method
methods = {
    "KMeans Random": kmeans_random.labels_,
    "KMeans Plus": kmeans_plus.labels_,
    "Bisecting KMeans": bisect_kmeans.labels_,
    "Spectral": spectral.labels_,
    "DBSCAN": dbscan.labels_,
    "Agglomerative Single": agg_single.labels_,
    "Agglomerative Complete": agg_complete.labels_,
    "Agglomerative Average": agg_average.labels_,
    "Agglomerative Ward": agg_ward.labels_,
}


evaluation_results = {}
for method, predicted_labels in methods.items():
    fmi = fowlkes_mallows_score(labels, predicted_labels)
    # Check if predicted labels have more than one unique label
    if len(np.unique(predicted_labels)) > 1:
```

```
        silhouette = silhouette_score(reduced_features,
predicted_labels)
    else:
        silhouette = -1  # Assign a default value for single-cluster
cases
    evaluation_results[method] = {"FMI": fmi, "Silhouette":
silhouette}


# Rank methods by FMI and Silhouette
ranked_by_fmi = sorted(evaluation_results.items(), key=lambda x: x[1]
["FMI"], reverse=True)
ranked_by_silhouette = sorted(evaluation_results.items(), key=lambda
x: x[1]["Silhouette"], reverse=True)

# Display rankings
print("Ranking by Fowlkes-Mallows Index:")
for rank, (method, scores) in enumerate(ranked_by_fmi, start=1):
    print(f"{rank}. {method}: FMI={scores['FMI']}")

print("\nRanking by Silhouette Coefficient:")
for rank, (method, scores) in enumerate(ranked_by_silhouette,
start=1):
    print(f"{rank}. {method}: Silhouette={scores['Silhouette']}")
```

```
Ranking by Fowlkes-Mallows Index:
1. Spectral: FMI=0.9399393480851093
2. Bisecting KMeans: FMI=0.9399341044331888
3. Agglomerative Average: FMI=0.9366335483807186
4. KMeans Random: FMI=0.9353106596092164
5. KMeans Plus: FMI=0.9353106596092164
6. Agglomerative Complete: FMI=0.9324979746800472
7. Agglomerative Ward: FMI=0.9280427868785972
8. DBSCAN: FMI=0.5003361862284268
9. Agglomerative Single: FMI=0.49833288338614196

Ranking by Silhouette Coefficient:
1. KMeans Random: Silhouette=0.6111951248315856
2. KMeans Plus: Silhouette=0.6111951248315856
3. Spectral: Silhouette=0.6110473945624377
4. Bisecting KMeans: Silhouette=0.610433589299064
5. Agglomerative Ward: Silhouette=0.6100224128753775
6. Agglomerative Complete: Silhouette=0.6059990465423257
7. Agglomerative Average: Silhouette=0.6059027589412035
8. Agglomerative Single: Silhouette=-0.1994733995415742
9. DBSCAN: Silhouette=-1
```