

```
from google.colab import drive
drive.mount('/content/drive')
```

```
from torchvision import transforms

def getTransform():
    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]
        )
    ])
    return transform
```

```
from torchvision.datasets import ImageFolder
from torch.utils.data import DataLoader

def loadImages(imageFolder, batchSize=32):

    transform = getTransform()

    dataset = ImageFolder(root=imageFolder, transform=transform)
    loader = DataLoader(dataset, batch_size=batchSize, shuffle=False)

    sampleImg, _ = dataset[0]
    print("Sample image shape after transform:", sampleImg.shape)

    return dataset, loader
```

```
from torchvision.models import resnet18
from torchvision.models.feature_extraction import create_feature_extractor
import torch

def getResnet18FeatureExtractor():
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    model = resnet18(pretrained=True).to(device)
    model.eval()

    # Extract last convolutional layer (last conv of layer4)
    returnNodes = {'layer4.1.conv2': 'features'}
    featureExtractor = create_feature_extractor(model, return_nodes=returnNodes)

    return featureExtractor, device
```

```
#Passes images in the loader through the feature extractor and
#returns flattened features and labels.

def extractFeatures(loader, featureExtractor, device):

    allFeatures = []
    allLabels = []

    with torch.no_grad():
        for images, labels in loader:
            images = images.to(device)
            features = featureExtractor(images)['features']
            features = features.view(features.size(0), -1) # flatten H x W x C
            allFeatures.append(features.cpu())
            allLabels.append(labels)
```

```
allFeatures = torch.cat(allFeatures)
allLabels = torch.cat(allLabels)

return allFeatures, allLabels
```

```
from sklearn.decomposition import PCA

def reduceTo2D(features):
    pca = PCA(n_components=2)
    reduced = pca.fit_transform(features)

    print("Original feature dim:", features.shape[1])
    print("Reduced feature dim:", reduced.shape[1])

    return reduced
```

```
from sklearn.cluster import KMeans, BisectingKMeans

def kmeansApproach(features2D, K=3):
    results = {}

    kmRandom = KMeans(n_clusters=K, init='random', n_init=10)
    results["KMeans_random"] = kmRandom.fit_predict(features2D)

    kmPlus = KMeans(n_clusters=K, init='k-means++', n_init=10)
    results["KMeans_kmeans++"] = kmPlus.fit_predict(features2D)

    kmBisect = BisectingKMeans(n_clusters=K, init='random')
    results["Bisecting_KMeans"] = kmBisect.fit_predict(features2D)

    return results
```

```
from sklearn.metrics import fowlkes_mallows_score, silhouette_score

def evaluateClustering(predLabels, trueLabels=None, features=None, method="fowlkes"):

    method = method.lower()

    if method == "fowlkes":
        return fowlkes_mallows_score(trueLabels, predLabels)

    elif method == "silhouette":
        return silhouette_score(features, predLabels)

    else:
        return None
```

```
from sklearn.cluster import SpectralClustering

def spectralClusteringApproach(X):
    spectral = SpectralClustering(n_clusters=3, assign_labels='kmeans')
    labels = spectral.fit_predict(X)
    return labels
```

```

from sklearn.metrics.pairwise import rbf_kernel
#First one was not producing desired clustering

def spectralClusteringApproach(features2D):
    # Use RBF kernel to compute similarity (default gamma=None uses 1/n_features)
    affinity_matrix = rbf_kernel(features2D)

    # Spectral Clustering with precomputed affinity to better separate clusters
    spectral = SpectralClustering(n_clusters=3, affinity='precomputed', assign_labels='kmeans', random_state=42)
    labels = spectral.fit_predict(affinity_matrix)
    return labels

```

```

from sklearn.cluster import AgglomerativeClustering

def agglomerativeClusteringAll(features2D, n_clusters=3):

    linkageMethods = ['single', 'complete', 'average', 'ward']
    labelsDict = {}

    for linkage in linkageMethods:
        agglo = AgglomerativeClustering(n_clusters=n_clusters, linkage=linkage)
        labels = agglo.fit_predict(features2D)
        labelsDict[linkage] = labels

    return labelsDict

```

```

def dbscanApproach(pca2Dfeatures, neighborhoodRadius, minPointsInCluster):
    from sklearn.cluster import DBSCAN

    dbscanModel = DBSCAN(eps=neighborhoodRadius, min_samples=minPointsInCluster)
    db = dbscanModel.fit(pca2Dfeatures)

    labels = db.labels_
    return labels

```

```

def testDbscanPair(features2D, eps, minSamples):
    labels = dbscanApproach(features2D, eps, minSamples)

    if labels is None or len(labels) == 0:
        return 0, []

    uniqueLabels = set(labels)
    if -1 in uniqueLabels:
        clusterCount = len(uniqueLabels) - 1
    else:
        clusterCount = len(uniqueLabels)

    return clusterCount, labels

```

```

def searchClusterSize(features2D, desired_clusters=3):
    #Search for DBSCAN parameters that produce the desired number of clusters.

    eps_values = [0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5]
    min_samples_values = [3, 4, 5, 6, 7, 8]

    for eps in eps_values:
        for minSamples in min_samples_values:
            clusterCount, labels = testDbscanPair(features2D, eps, minSamples)

            if clusterCount == desired_clusters:

```

```
    return eps, minSamples, labels

return None, None, None
```

```
def evaluateKMeansFamily():

    results = kmeansApproach(features2D, K=3)

    for name, labels in results.items():

        fmScore = evaluateClustering(predLabels=labels, trueLabels=allLabels, method="fowlkes")
        scScore = evaluateClustering(predLabels=labels, features=features2D, method="silhouette")

        print(name, "-> Fowlkes-Mallows:", round(fmScore, 3), "Silhouette:", round(scScore, 3))
```

```
def evaluateSpectralClustering(spectralLabels, allLabels, features2D):
    fmScore = evaluateClustering(predLabels=spectralLabels, trueLabels=allLabels, method="fowlkes")
    scScore = evaluateClustering(predLabels=spectralLabels, features=features2D, method="silhouette")

    print("SpectralClustering → Fowlkes-Mallows:", round(fmScore, 3), "Silhouette:", round(scScore, 3))
```

```
def evaluateAgglomerativeClustering():

    # Compute labels for all linkage methods
    aggloLabels = agglomerativeClusteringAll(features2D, n_clusters=3) #(Integers)

    # Evaluate each linkage method
    for linkage, labels in aggloLabels.items():
        fmScore = evaluateClustering(predLabels=labels, trueLabels=allLabels, method="fowlkes")
        scScore = evaluateClustering(predLabels=labels, features=features2D, method="silhouette")

        print("Agglomerative (", linkage, ") → Fowlkes-Mallows:", round(fmScore, 3), "Silhouette:", round(scScore, 3))
```

```
def evaluateDBSCAN(features2D, allLabels):

    eps, minSamples, labels = searchClusterSize(features2D, desired_clusters=3)

    if eps is None:
        print("DBSCAN could not find 3 clusters.")
        return None, None, None, None

    fmScore = evaluateClustering(predLabels=labels, trueLabels=allLabels, method="fowlkes")
    scScore = evaluateClustering(predLabels=labels, features=features2D, method="silhouette")

    return eps, minSamples, fmScore, scScore, labels
```

```
# 1 Load dataset
imageFolder = "/content/drive/MyDrive/DataSets/Covid/Covid19-dataset/allImages"
dataset, loader = loadImages(imageFolder)

# 2 Transforms
transform = getTransform()

# 3 Device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```
# 4 Feature extractor
featureExtractor, device = getResnet18FeatureExtractor()

# 5 Extract features
allFeatures, allLabels = extractFeatures(loader, featureExtractor, device)

# 6 PCA to 2D
features2D = reduceTo2D(allFeatures.numpy())

#7. Call Kmean Family
kmeanFamily=kmeansApproach(features2D)
print(kmeanFamily)

#9. Spectral Clustering
spectralLabels = spectralClusteringApproach(features2D)
print("Spectral clustering labels:", spectralLabels)

#10 Find parameter to get 3 clusters:
eps,minSamples,labels=searchClusterSize(features2D, 3)
print("Eps and min samples\t",eps, "\t",minSamples)
```

```
2 2 2 1 2 2 2 2 2 2 2 2 2 1 2 2 2 2  
Eps and min samples      0.2      4
```

```
evaluateKMeansFamily()  
evaluateAgglomerativeClustering()  
spectralLabels = spectralClusteringApproach(features2D)  
evaluateSpectralClustering(spectralLabels, allLabels, features2D)
```

```
KMeans_random → Fowlkes-Mallows: 0.679 Silhouette: 0.548  
KMeans_kmeans++ → Fowlkes-Mallows: 0.679 Silhouette: 0.548  
Bisection_KMeans → Fowlkes-Mallows: 0.763 Silhouette: 0.512  
Agglomerative ( single ) → Fowlkes-Mallows: 0.578 Silhouette: -0.273  
Agglomerative ( complete ) → Fowlkes-Mallows: 0.577 Silhouette: 0.492  
Agglomerative ( average ) → Fowlkes-Mallows: 0.634 Silhouette: 0.533  
Agglomerative ( ward ) → Fowlkes-Mallows: 0.621 Silhouette: 0.516  
SpectralClustering → Fowlkes-Mallows: 0.688 Silhouette: 0.544
```

```
eps, minSamples, fm, sc, db_labels = evaluateDBSCAN(features2D, allLabels)
```

```
if eps is not None:  
    print("== DBSCAN Evaluation ==")  
    print("eps:", eps)  
    print("min_samples:", minSamples)  
    print("Fowlkes-Mallows:", round(fm, 3))  
    print("Silhouette Score:", round(sc, 3))  
else:  
    print("DBSCAN could not produce 3 clusters with provided grid.")
```

```
== DBSCAN Evaluation ==  
eps: 0.2  
min_samples: 4  
Fowlkes-Mallows: 0.554  
Silhouette Score: -0.086
```

```
#Not necessary, but illustrates the search  
eps_values = [0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.6, 0.7, 0.8, 0.9]  
min_samples_values = [2,3,4,5,6]
```

```
for eps in eps_values:  
    for minSamples in min_samples_values:  
        clusterCount, labels = testDbscanPair(features2D, eps, minSamples)  
        print(f"eps={eps}, minSamples={minSamples} → clusters={clusterCount}")
```

```
eps=0.1, minSamples=2 → clusters=19  
eps=0.1, minSamples=3 → clusters=0  
eps=0.1, minSamples=4 → clusters=0  
eps=0.1, minSamples=5 → clusters=0  
eps=0.1, minSamples=6 → clusters=0  
eps=0.15, minSamples=2 → clusters=27  
eps=0.15, minSamples=3 → clusters=1  
eps=0.15, minSamples=4 → clusters=1  
eps=0.15, minSamples=5 → clusters=1  
eps=0.15, minSamples=6 → clusters=0  
eps=0.2, minSamples=2 → clusters=44  
eps=0.2, minSamples=3 → clusters=7  
eps=0.2, minSamples=4 → clusters=3  
eps=0.2, minSamples=5 → clusters=1  
eps=0.2, minSamples=6 → clusters=0  
eps=0.25, minSamples=2 → clusters=50  
eps=0.25, minSamples=3 → clusters=10  
eps=0.25, minSamples=4 → clusters=4  
eps=0.25, minSamples=5 → clusters=1  
eps=0.25, minSamples=6 → clusters=0  
eps=0.3, minSamples=2 → clusters=57  
eps=0.3, minSamples=3 → clusters=13  
eps=0.3, minSamples=4 → clusters=7  
eps=0.3, minSamples=5 → clusters=2  
eps=0.3, minSamples=6 → clusters=0  
eps=0.35, minSamples=2 → clusters=58  
eps=0.35, minSamples=3 → clusters=24  
eps=0.35, minSamples=4 → clusters=7  
eps=0.35, minSamples=5 → clusters=4
```

```
eps=0.35, minSamples=6 → clusters=1
eps=0.4, minSamples=2 → clusters=60
eps=0.4, minSamples=3 → clusters=32
eps=0.4, minSamples=4 → clusters=12
eps=0.4, minSamples=5 → clusters=4
eps=0.4, minSamples=6 → clusters=3
eps=0.45, minSamples=2 → clusters=53
eps=0.45, minSamples=3 → clusters=26
eps=0.45, minSamples=4 → clusters=19
eps=0.45, minSamples=5 → clusters=9
eps=0.45, minSamples=6 → clusters=3
eps=0.5, minSamples=2 → clusters=50
eps=0.5, minSamples=3 → clusters=24
eps=0.5, minSamples=4 → clusters=19
eps=0.5, minSamples=5 → clusters=10
eps=0.5, minSamples=6 → clusters=8
eps=0.6, minSamples=2 → clusters=40
eps=0.6, minSamples=3 → clusters=23
eps=0.6, minSamples=4 → clusters=17
eps=0.6, minSamples=5 → clusters=15
eps=0.6, minSamples=6 → clusters=10
eps=0.7, minSamples=2 → clusters=28
eps=0.7, minSamples=3 → clusters=20
eps=0.7, minSamples=4 → clusters=11
eps=0.7, minSamples=5 → clusters=10
eps=0.7, minSamples=6 → clusters=13
eps=0.8, minSamples=2 → clusters=19
eps=0.8, minSamples=3 → clusters=16
```