import os

import numpy as np

from PIL import Image

from sklearn.cluster import MiniBatchKMeans

from sklearn.metrics import silhouette\_score, calinski\_harabasz\_score, davies\_bouldin\_score

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

from skimage import util

# Define paths

data\_folder = "/content/drive/MyDrive/TileScan\_003"  # Path to folder with coal images (input)

n\_clusters = 4  # Number of clusters

wsize = 3     # Tile size (window size)

stride = 3      # Stride

seed = 197208   # Random seed for reproducibility

def pad(img, wsize, stride=1):

    """Add symmetric padding to ensure tiles divide the image evenly."""

    height, width = img.shape

    adj = 1 if stride != 'block' else 0

    px = wsize - height % stride - adj

    py = wsize - width % stride - adj

    px = (px // 2, px // 2 + px % 2)

    py = (py // 2, py // 2 + py % 2)

    return np.pad(img, pad\_width=(px, py), mode='symmetric')

def evaluate\_clustering\_without\_ground\_truth(data\_folder, n\_clusters, wsize, stride, seed):

    file\_names = [f for f in os.listdir(data\_folder) if f.endswith('.tif')]

    silhouette\_scores = []

    calinski\_scores = []

    davies\_scores = []

    for file\_name in file\_names:

        # Load the input image

        img\_path = os.path.join(data\_folder, file\_name)

        img = np.array(Image.open(img\_path).convert('L'))  # Convert to grayscale

        # Tile the image

        X = util.view\_as\_windows(

            pad(img, wsize=wsize, stride=stride),

            window\_shape=(wsize, wsize),

            step=stride

        ).reshape(-1, wsize\*\*2)

        # Standardize the data

        scaler = StandardScaler()

        X\_scaled = scaler.fit\_transform(X)

        # Apply PCA for dimensionality reduction

        max\_components = min(X\_scaled.shape[0], X\_scaled.shape[1])  # Minimum of samples and features

        n\_components = min(10, max\_components)  # Retain up to 10 components or fewer if limited

        pca = PCA(n\_components=n\_components)

        X\_pca = pca.fit\_transform(X\_scaled)

        # Perform MiniBatchKMeans clustering

        minibatch\_kmeans = MiniBatchKMeans(n\_clusters=n\_clusters, random\_state=seed, batch\_size=100)

        labels = minibatch\_kmeans.fit\_predict(X\_pca)

        # Compute clustering evaluation metrics

        if len(np.unique(labels)) > 1:  # Ensure there are at least two clusters for evaluation

            silhouette\_scores.append(silhouette\_score(X\_pca, labels))

            calinski\_scores.append(calinski\_harabasz\_score(X\_pca, labels))

            davies\_scores.append(davies\_bouldin\_score(X\_pca, labels))

    # Calculate mean metrics

    mean\_silhouette = np.mean(silhouette\_scores) if silhouette\_scores else None

    mean\_calinski = np.mean(calinski\_scores) if calinski\_scores else None

    mean\_davies = np.mean(davies\_scores) if davies\_scores else None

    return mean\_silhouette, mean\_calinski, mean\_davies

# Call the function and print results

mean\_silhouette, mean\_calinski, mean\_davies = evaluate\_clustering\_without\_ground\_truth(

    data\_folder, n\_clusters, wsize, stride, seed

)

print(f"Mean Silhouette Score: {mean\_silhouette:.4f}" if mean\_silhouette else "Silhouette Score: Not Computable")

print(f"Mean Calinski-Harabasz Index: {mean\_calinski:.4f}" if mean\_calinski else "Calinski-Harabasz Index: Not Computable")

print(f"Mean Davies-Bouldin Index: {mean\_davies:.4f}" if mean\_davies else "Davies-Bouldin Index: Not Computable")