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")

2ND PHAGE

import cv2

import numpy as np

import matplotlib.pyplot as plt

# Load the image in grayscale

image = cv2.imread("/content/drive/MyDrive/TieScan4/TileScan\_001--Stage263.tif", cv2.IMREAD\_GRAYSCALE)

# Apply global histogram equalization

hist\_eq = cv2.equalizeHist(image)

# Display Original vs Histogram Equalized

plt.figure(figsize=(10,5))

plt.subplot(1,2,1)

plt.imshow(image, cmap="gray")

plt.title("Original Image")

plt.axis("off")

plt.subplot(1,2,2)

plt.imshow(hist\_eq, cmap="gray")

plt.title("Histogram Equalized")

plt.axis("off")

plt.show()

CANNY

# Apply Canny Edge Detection

edges = cv2.Canny(gray, threshold1=50, threshold2=150)

# Display Results

plt.figure(figsize=(10,5))

plt.subplot(1,2,1)

plt.imshow(gray, cmap="gray")

plt.title("/content/drive/MyDrive/TieScan4/TileScan\_001--Stage042.tif")

plt.axis("off")

plt.subplot(1,2,2)

plt.imshow(edges, cmap="gray")

plt.title("Canny Edge Detection")

plt.axis("off")

plt.show()

HOG

import cv2

import numpy as np

import matplotlib.pyplot as plt

from skimage.feature import hog

from skimage import color

# Load an image (coal image, for example)

image = cv2.imread("/content/drive/MyDrive/TieScan4/TileScan\_001--Stage263.tif")  # Replace with your coal image path

# Convert the image to grayscale (required for HOG)

gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

# Compute HOG features and visualize the HOG image

features, hog\_image = hog(gray\_image, pixels\_per\_cell=(8, 8), cells\_per\_block=(2, 2),

                          orientations=9, visualize=True, block\_norm="L2-Hys")

# Display the original grayscale image and the HOG features

plt.figure(figsize=(10,5))

plt.subplot(1, 2, 1)

plt.imshow(gray\_image, cmap='gray')

plt.title("Original Grayscale Image")

plt.axis('off')

plt.subplot(1, 2, 2)

plt.imshow(hog\_image, cmap='gray')

plt.title("HOG Features")

plt.axis('off')

plt.show()

# Print the HOG feature vector size (number of features extracted)

print(f"Size of HOG feature vector: {features.shape}")

HOG-UPM

import cv2

import numpy as np

import matplotlib.pyplot as plt

from skimage.feature import hog

from sklearn.decomposition import PCA

import umap

# Load the coal image

image\_path = '/content/drive/MyDrive/TieScan4/TileScan\_001--Stage235.tif'  # Replace with the path to your coal image

image = cv2.imread(image\_path)

# Convert the image to grayscale (HOG requires grayscale images)

gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

# Extract HOG features from the image

features, hog\_image = hog(gray\_image, pixels\_per\_cell=(8, 8), cells\_per\_block=(2, 2),

                          orientations=9, visualize=True, block\_norm="L2-Hys")

# Now, let's apply UMAP for dimensionality reduction on HOG features

# Reshape HOG features into a 2D array (each feature vector as a row)

hog\_features\_reshaped = features.reshape(-1, features.shape[-1])  # Flatten feature map

# Apply UMAP for dimensionality reduction to 2D

umap\_model = umap.UMAP(n\_components=2)

umap\_result = umap\_model.fit\_transform(hog\_features\_reshaped)

# Visualize the original HOG image and the UMAP result

plt.figure(figsize=(12, 6))

# Display the original HOG image

plt.subplot(1, 2, 1)

plt.imshow(hog\_image, cmap='gray')

plt.title("HOG Features Visualization")

plt.axis('off')

# Display the UMAP 2D projection

plt.subplot(1, 2, 2)

plt.scatter(umap\_result[:, 0], umap\_result[:, 1], c='blue', marker='.', alpha=0.7)

plt.title("UMAP Projection of HOG Features")

plt.xlabel("UMAP 1")

plt.ylabel("UMAP 2")

plt.grid(True)

plt.show()

ORGINAL HOG AND UPMA

import cv2

import numpy as np

import matplotlib.pyplot as plt

from skimage.feature import hog

import umap

# Load the coal image (replace with the path to your image)

#image\_path = 'coal\_sample.jpg'  # Replace with your coal image path

image\_path = '/content/drive/MyDrive/TieScan4/TileScan\_001--Stage235.tif' # Corrected path to the image file

image = cv2.imread(image\_path)

# Convert the image to grayscale (HOG requires grayscale images)

gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

# Extract HOG features from the grayscale image

features, hog\_image = hog(gray\_image, pixels\_per\_cell=(8, 8), cells\_per\_block=(2, 2),

                          orientations=9, visualize=True, block\_norm="L2-Hys")

# Now, let's apply UMAP for dimensionality reduction on HOG features

# Reshape HOG features into a 2D array (each feature vector as a row)

hog\_features\_reshaped = features.reshape(-1, features.shape[-1])  # Flatten feature map

# Apply UMAP for dimensionality reduction to 2D

umap\_model = umap.UMAP(n\_components=2)

umap\_result = umap\_model.fit\_transform(hog\_features\_reshaped)

# Visualize the original image, HOG features, and UMAP projection

plt.figure(figsize=(15, 10))

# Display the original image

plt.subplot(1, 3, 1)

plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))  # Convert from BGR to RGB for correct display

plt.title("/content/drive/MyDrive/TieScan4/TileScan\_001--Stage042.tif")

plt.axis('off')

# Display the HOG features image

plt.subplot(1, 3, 2)

plt.imshow(hog\_image, cmap='gray')

plt.title("HOG Features Visualization")

plt.axis('off')

# Display the UMAP 2D projection

plt.subplot(1, 3, 3)

plt.scatter(umap\_result[:, 0], umap\_result[:, 1], c='blue', marker='.', alpha=0.7)

plt.title("UMAP Projection of HOG Features")

plt.xlabel("UMAP 1")

plt.ylabel("UMAP 2")

plt.grid(True)

plt.show()

HOG TO DBSCAN

import cv2

import numpy as np

from sklearn.cluster import DBSCAN

from skimage.feature import hog

import matplotlib.pyplot as plt

# Load the coal image (replace with your coal image path)

image\_path = '/content/drive/MyDrive/TieScan4/TileScan\_001--Stage235.tif'  # Replace with the path to your coal image

image = cv2.imread(image\_path)

# Convert to grayscale

gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

# Extract HOG features

features, hog\_image = hog(gray\_image, pixels\_per\_cell=(8, 8), cells\_per\_block=(2, 2),

                          orientations=9, visualize=True, block\_norm="L2-Hys")

# Flatten the features for clustering

features\_reshaped = features.reshape(-1, features.shape[-1])  # Flatten feature map

# Apply DBSCAN

dbscan = DBSCAN(eps=0.3, min\_samples=10)

labels = dbscan.fit\_predict(features\_reshaped)

# Visualize the results

plt.figure(figsize=(12, 6))

# Display the HOG feature visualization

plt.subplot(1, 2, 1)

plt.imshow(hog\_image, cmap='gray')

plt.title("HOG Features Visualization")

plt.axis('off')

# Display the DBSCAN clustering result (scatter plot of labels)

plt.subplot(1, 2, 2)

plt.scatter(features\_reshaped[:, 0], features\_reshaped[:, 1], c=labels, cmap='viridis', alpha=0.7)

plt.title("DBSCAN Clustering of HOG Features")

plt.xlabel("Feature Dimension 1")

plt.ylabel("Feature Dimension 2")

plt.grid(True)

plt.show()