# Install necessary libraries

!pip install pillow h5py opencv-python-headless scikit-image matplotlib

from PIL import Image

import numpy as np

import h5py

import cv2

from google.colab import files

from datetime import datetime

from skimage.measure import regionprops, label

import matplotlib.pyplot as plt

# Upload your files (.h5 or .png)

uploaded = files.upload()

# Check the uploaded file type and load accordingly

file\_path = list(uploaded.keys())[0]

file\_extension = file\_path.split('.')[-1]

if file\_extension == 'h5':

# Load the .h5 file

with h5py.File(file\_path, 'r') as h5\_file:

# Assuming the dataset is stored under the first key

data\_key = list(h5\_file.keys())[0]

dataset = h5\_file[data\_key][()]

# Convert the dataset to an image

# Normalize the dataset to the range [0, 255]

dataset\_normalized = (dataset - dataset.min()) / (dataset.max() - dataset.min()) \* 255

dataset\_normalized = dataset\_normalized.astype(np.uint8)

# Convert to image

dataset\_image = Image.fromarray(dataset\_normalized)

# Convert the image to a numpy array

converted\_image\_array = np.array(dataset\_image)

elif file\_extension == 'png':

# Load the .png file directly

converted\_image = Image.open(file\_path)

# Convert the image to a numpy array

converted\_image\_array = np.array(converted\_image)

# Identify red pixel areas

red\_pixels = converted\_image\_array[:, :, 0] > 127 # Assuming red channel values above 127 are considered "red"

classified\_image\_np = np.zeros\_like(converted\_image\_array)

classified\_image\_np[:, :, 0] = red\_pixels.astype(np.uint8) \* 255

broader\_aspect\_ratio\_threshold = 1.5

broader\_directionality\_range = (-90, 90)

broader\_directionality\_range\_rad = np.deg2rad(broader\_directionality\_range)

def calculate\_aspect\_ratio\_region(region):

minr, minc, maxr, maxc = region.bbox

height = maxr - minr

width = maxc - minc

return max(height / width, width / height)

def calculate\_directionality(region):

return region.orientation

def calculate\_circularity(region):

perimeter = region.perimeter

area = region.area

if perimeter == 0:

return 0

return (4 \* np.pi \* area) / (perimeter \*\* 2)

def calculate\_ovality(aspect\_ratio):

return abs(aspect\_ratio - 1)

def within\_broader\_orientation\_range(orientation):

return broader\_directionality\_range\_rad[0] <= orientation <= broader\_directionality\_range\_rad[1]

red\_mask = (classified\_image\_np == [255, 0, 0]).all(axis=2)

labeled\_red\_mask, num\_features = label(red\_mask, return\_num=True)

regions = regionprops(labeled\_red\_mask)

# List of ovality\_threshold values

ovality\_thresholds = [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]

for ovality\_threshold in ovality\_thresholds:

broader\_fracture\_filled\_labels = []

non\_circular\_labels = []

for region in regions:

aspect\_ratio = calculate\_aspect\_ratio\_region(region)

orientation = calculate\_directionality(region)

circularity = calculate\_circularity(region)

ovality = calculate\_ovality(aspect\_ratio)

if circularity < 0.5 and ovality < ovality\_threshold:

non\_circular\_labels.append(region.label)

if (aspect\_ratio >= broader\_aspect\_ratio\_threshold or within\_broader\_orientation\_range(orientation)) and circularity < 0.5 and ovality < ovality\_threshold:

broader\_fracture\_filled\_labels.append(region.label)

filtered\_broader\_fracture\_filled\_mask = np.isin(labeled\_red\_mask, broader\_fracture\_filled\_labels)

red\_pixels\_count = np.sum(red\_pixels)

yellow\_pixels\_count = np.sum(filtered\_broader\_fracture\_filled\_mask)

highlighted\_filtered\_broader\_fracture\_fills = classified\_image\_np.copy()

highlighted\_filtered\_broader\_fracture\_fills[filtered\_broader\_fracture\_filled\_mask] = [255, 255, 0]

plt.imshow(highlighted\_filtered\_broader\_fracture\_fills)

plt.title(f'Highlighted Filtered Broader Fracture Fills (ovality\_threshold = {ovality\_threshold})')

plt.show()

print(f"Processed with ovality\_threshold = {ovality\_threshold}")

print(f"Number of red pixels: {red\_pixels\_count}")

print(f"Number of yellow pixels (identified fracture-filled areas): {yellow\_pixels\_count}")