```python
# -*- coding: utf-8 -*-
"""KNN CLASSIFICATION 20 FEBRUARY 2021 QB

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1XGv4aPO3yuvz1i026ZgL02E8D24SGHaQ
"""

pip install rasterio

import rasterio

# Define the new path
new_path = "/content/drive/MyDrive/RS RESULTS/CNN DRAFT/GROUNDWATER FROM GGIS/20 FEBRUARY 2021/LC08_L1TP_170074_20210220_20210303_02_T1_qb.tif"

# Open the image file using rasterio
with rasterio.open(new_path) as image:
    print(image.crs)

# Import necessary libraries
import pandas as pd
import numpy as np
import rasterio
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier  # Import KNeighborsClassifier
from sklearn.metrics import accuracy_score  # Import accuracy_score for evaluation
from sklearn.impute import SimpleImputer

# Load the data
data = pd.read_csv("/content/drive/MyDrive/RS RESULTS/CNN DRAFT/GROUNDWATER FROM GGIS/CSV DATA/ESTIMATED DEPTHS TO GROUNDWATER.csv")

# Function to convert range values to their average
def convert_range_to_average(val):
    if '-' in val:
        val_range = val.split('-')
        average = np.mean([float(i) for i in val_range])
        return average
    elif '<' in val:
        return float(val[1:]) / 2  # If value is less than a number, take half of that number
    else:
        return float(val)

# Convert range values to their average
for col in ["Estimated Depth To Groundwater/mbgl", "Groundwater Storage (Depth in m)", "Aquifer Productivity (l/s)"]:
    data[col] = data[col].apply(convert_range_to_average)

# Load the GeoTIFF file
with rasterio.open('/content/drive/MyDrive/RS RESULTS/CNN DRAFT/GROUNDWATER FROM GGIS/20 FEBRUARY 2021/LC08_L1TP_170074_20210220_20210303_02_T1_qb.tif') as src
    # Transform lat/lon to the coordinate system of the GeoTIFF
    lon = data['longitude'].values
    lat = data['latitude'].values
    row, col = src.index(lon, lat)

    # Initialize an array for the GeoTIFF values
    geotiff_values = np.empty_like(lon)

    # Extract the values at these points
    for i in range(len(lon)):
        try:
            geotiff_values[i] = src.read(1)[row[i], col[i]]
        except IndexError:
            geotiff_values[i] = np.nan  # Assign a default value

    data['geotiff_value'] = geotiff_values

# Define bins and labels
bins = [0, 10, 20, 30, 40, 50, np.inf]  # Change these values based on your specific use case
labels = ['0-10', '10-20', '20-30', '30-40', '40-50', '50+']

# Use pd.cut function to convert continuous values into categorical
data['Average Depth Category'] = pd.cut(data['Average Depth'], bins=bins, labels=labels)

# Define features and target
features = data[["longitude", "latitude", "Groundwater Storage (Depth in m)", "Aquifer Productivity (l/s)", "geotiff_value"]]
target = data['Average Depth Category']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)

# Create an imputer
imputer = SimpleImputer(strategy='mean')

# Fit on the training data and transform both training and test data
X_train_imputed = imputer.fit_transform(X_train)
X_test_imputed = imputer.transform(X_test)

# Train the KNN classifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train_imputed, y_train)

# Predict on the test set
y_pred = knn.predict(X_test_imputed)

# Evaluate the model using accuracy score
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

from sklearn.metrics import precision_score  # Import precision_score for evaluation

# Predict on the test set
y_pred = knn.predict(X_test_imputed)

# Evaluate the model using precision score
precision = precision_score(y_test, y_pred, average='micro')  # Set the `average` parameter to your preferred method
print(f"Precision: {precision:.2f}")
```

```python
from sklearn.metrics import precision_score  # Import precision_score for evaluation

# Predict on the test set
y_pred = knn.predict(X_test_imputed)

# Evaluate the model using precision score with 'macro' average
precision_macro = precision_score(y_test, y_pred, average='macro')
print(f"Macro Precision: {precision_macro:.2f}")

# Evaluate the model using precision score with 'weighted' average
precision_weighted = precision_score(y_test, y_pred, average='weighted')
print(f"Weighted Precision: {precision_weighted:.2f}")

# Import necessary libraries
from sklearn.metrics import recall_score  # Import recall_score

# ... (rest of your code)

# Predict on the test set
y_pred = knn.predict(X_test_imputed)

# Compute recall
recall_micro = recall_score(y_test, y_pred, average='micro')
recall_macro = recall_score(y_test, y_pred, average='macro')
recall_weighted = recall_score(y_test, y_pred, average='weighted')

print(f"Micro Recall: {recall_micro:.2f}")
print(f"Macro Recall: {recall_macro:.2f}")
print(f"Weighted Recall: {recall_weighted:.2f}")

# Compute recall
recall_micro = recall_score(y_test, y_pred, average='micro', zero_division=0)
recall_macro = recall_score(y_test, y_pred, average='macro', zero_division=0)
recall_weighted = recall_score(y_test, y_pred, average='weighted', zero_division=0)

# Import necessary libraries
from sklearn.metrics import f1_score  # Import f1_score

# ... (rest of your code)

# Predict on the test set
y_pred = knn.predict(X_test_imputed)

# Compute F1 score
f1_micro = f1_score(y_test, y_pred, average='micro')
f1_macro = f1_score(y_test, y_pred, average='macro')
f1_weighted = f1_score(y_test, y_pred, average='weighted')

print(f"Micro F1 Score: {f1_micro:.2f}")
print(f"Macro F1 Score: {f1_macro:.2f}")
print(f"Weighted F1 Score: {f1_weighted:.2f}")

# Import necessary libraries
import matplotlib.pyplot as plt

# Assume you have computed the following metrics
accuracy = 0.82
precision_micro = 0.82
precision_macro = 0.5
precision_weighted = 1
recall_micro = 0.82
recall_macro = 0.41
recall_weighted = 0.82
f1_micro = 0.82
f1_macro = 0.45
f1_weighted = 0.90

# Create a dictionary of metrics
metrics = {
    'Accuracy': accuracy,
    'Micro Precision': precision_micro,
    'Macro Precision': precision_macro,
    'Weighted Precision': precision_weighted,
    'Micro Recall': recall_micro,
    'Macro Recall': recall_macro,
    'Weighted Recall': recall_weighted,
    'Micro F1 Score': f1_micro,
    'Macro F1 Score': f1_macro,
    'Weighted F1 Score': f1_weighted
}

# Create a bar plot
plt.figure(figsize=(10, 6))
plt.barh(list(metrics.keys()), list(metrics.values()), color='skyblue')
plt.xlabel('Score')
plt.title('Classification Metrics')
plt.xlim(0, 1)
plt.show()

# Import necessary libraries
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Get unique classes
unique_classes = np.unique(np.concatenate((np.unique(y_test), np.unique(y_pred))))

# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred, labels=unique_classes)

# Convert confusion matrix to DataFrame for easier plotting
cm_df = pd.DataFrame(cm, index=unique_classes, columns=unique_classes)

# Create a heatmap
plt.figure(figsize=(10, 8))
```

```python
sns.heatmap(cm_df, annot=True, cmap='Blues', fmt='g')
plt.title('Confusion Matrix')
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()

# Import libraries
import rasterio
from rasterio.plot import show
import numpy as np
from skimage.transform import resize

# Define the paths to the files
old_predictions_path = '/content/drive/MyDrive/RS RESULTS/CNN DRAFT/GROUNDWATER FROM GGIS/20 FEBRUARY 2021/LC08_L1TP_170074_20210220_20210303_02_T1_qb.tif'
new_predictions_path = '/content/drive/MyDrive/RS RESULTS/CNN DRAFT/GROUNDWATER FROM GGIS/10 FEBRUARY 2023/LC08_L1TP_170074_20230210_20230217_02_T1_qb.tif'

# Open the two tif files
with rasterio.open(old_predictions_path) as src1, rasterio.open(new_predictions_path) as src2:
    # Read the data and metadata
    old_data, old_meta = src1.read(), src1.meta
    new_data, new_meta = src2.read(), src2.meta

    # Resample the new data to match the old data
    new_data_resampled = resize(new_data, old_data.shape)

    # Calculate the difference between the old and new predictions
    diff_data = new_data_resampled - old_data

    # Stack the old data, resampled new data, and difference data along a new third dimension
    merged_data = np.dstack([old_data, new_data_resampled, diff_data])

    # Visualize the results
    show(merged_data, cmap='viridis')

# Import libraries
import rasterio
import numpy as np

# Define the path to the file
new_predictions_path = '/content/drive/MyDrive/RS RESULTS/CNN DRAFT/GROUNDWATER FROM GGIS/10 FEBRUARY 2023/LC08_L1TP_170074_20230210_20230217_02_T1_qb.tif'

# Open the tif file
with rasterio.open(new_predictions_path) as src:
    # Read the data
    new_data = src.read()

# Print out some of the predicted values
print("The shape of the new data is:", new_data.shape)
print("Some predicted values from the 2023 image are:", new_data[0, :10, :10])
```

```python
# -*- coding: utf-8 -*-
"""CNN MODEL

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1jLKiHT9Km0mm8qG4KtjOPceXFBHogIBA
"""

pip install rasterio

import numpy as np
from tensorflow.keras.utils import to_categorical
import rasterio
from skimage.transform import resize

# Define the paths to your GeoTIFF files
file_paths = [
    '/content/drive/MyDrive/RS RESULTS/GEOSPATIAL TECHNIQUES/MIMOSA MINING COMPANY/GROUNDWATER EXCERCISE FROM GGIS/MIMOSA FEBRUARY 2021/LC08_L1TP_170074_202102
    '/content/drive/MyDrive/RS RESULTS/GEOSPATIAL TECHNIQUES/MIMOSA MINING COMPANY/GROUNDWATER EXCERCISE FROM GGIS/MIMOSA FEBRUARY 2024/LC09_L1TP_170073_202402
    '/content/drive/MyDrive/RS RESULTS/GEOSPATIAL TECHNIQUES/UNKI MINE/SATELLITE IMAGERY/UNKI FEBRUARY 2021/LC08_L1TP_170073_20210204_20210303_02_T1_qb.tif',
    '/content/drive/MyDrive/RS RESULTS/GEOSPATIAL TECHNIQUES/UNKI MINE/SATELLITE IMAGERY/UNKI FEBRUARY 2024/LC09_L1TP_170073_20240205_20240205_02_T1_refl.tif',
    '/content/drive/MyDrive/RS RESULTS/GEOSPATIAL TECHNIQUES/ZIMPLATS/ZIMPLATS GROUNDWATER EXCERCISE FROM GGIS/SATELLITE IMAGERY/FEBRUARY 2021/LC08_L1TP_170073_
    '/content/drive/MyDrive/RS RESULTS/GEOSPATIAL TECHNIQUES/ZIMPLATS/ZIMPLATS GROUNDWATER EXCERCISE FROM GGIS/SATELLITE IMAGERY/FEBRUARY 2024/LC09_L1TP_170073_
]

# Define the desired dimensions (change these to your desired dimensions)
desired_height = 7681
desired_width = 7681

# Initialize an empty list to hold your training data
X_train = []

# Load the GeoTIFF files and extract the feature bands
for file_path in file_paths:
    dataset = rasterio.open(file_path)
    bands = []
    for b in dataset.indexes:
        band = dataset.read(b)

        # Resize the band to the desired dimensions
        band = resize(band, (desired_height, desired_width))

        bands.append(band)

    # Stack the bands to create a 3D array
    bands = np.dstack(bands)

    # Add the 3D array to your training data
    X_train.append(bands)

# Define the classes
class Stability:
    def __init__(self, band1, band2, band3):
        self.band1 = band1
        self.band2 = band2
        self.band3 = band3

    def classify(self):
        stability = np.empty_like(self.band1, dtype=object)
        stability[(0 <= self.band1) & (self.band1 <= 32) & (148 <= self.band2) & (self.band2 <= 216) & (94 <= self.band3) & (self.band3 <= 144)] = 'Very good s
        stability[(32 <= self.band1) & (self.band1 <= 64) & (108 <= self.band2) & (self.band2 <= 148) & (57 <= self.band3) & (self.band3 <= 94)] = 'Good stabil
        stability[(64 <= self.band1) & (self.band1 <= 96) & (68 <= self.band2) & (self.band2 <= 108) & (20 <= self.band3) & (self.band3 <= 57)] = 'Poor stabili
        stability[(96 <= self.band1) & (self.band1 <= 128) & (0 <= self.band2) & (self.band2 <= 68) & (0 <= self.band3) & (self.band3 <= 20)] = 'Very poor stab
        stability[stability == None] = 'Invalid parameters'
        return stability

class WaterPresence:
    def __init__(self, band1):
        self.band1 = band1

    def classify(self):
        water_presence = np.empty_like(self.band1, dtype=object)
        water_presence[self.band1 < 25] = 'Very good'
        water_presence[(25 <= self.band1) & (self.band1 < 50)] = 'Good'
        water_presence[(50 <= self.band1) & (self.band1 < 75)] = 'Poor'
        water_presence[self.band1 >= 75] = 'Very poor'
        water_presence[water_presence == None] = 'Invalid parameters'
        return water_presence

class VegetationDensity:
    def __init__(self, band3):
        self.band3 = band3

    def classify(self):
        vegetation_density = np.empty_like(self.band3, dtype=object)
        vegetation_density[self.band3 > 0.7] = 'Very good (Dense and healthy vegetation)'
        vegetation_density[(0.4 <= self.band3) & (self.band3 <= 0.7)] = 'Good (Moderate and diverse vegetation)'
        vegetation_density[(0.1 <= self.band3) & (self.band3 < 0.4)] = 'Poor (Sparse and unhealthy vegetation)'
        vegetation_density[self.band3 < 0.1] = 'Very poor (Absent or dead vegetation)'
        vegetation_density[vegetation_density == None] = 'Invalid parameters'
        return vegetation_density

class LandUse:
    def __init__(self, band2):
        self.band2 = band2

    def classify(self):
        land_use = np.empty_like(self.band2, dtype=object)
        land_use[self.band2 == 100] = 'Very good (Forest)'
        land_use[self.band2 == 75] = 'Good (Grassland)'
        land_use[self.band2 == 25] = 'Poor (Cropland)'
        land_use[self.band2 == 0] = 'Very poor (Urban)'
        land_use[land_use == None] = 'Invalid parameters'
        return land_use
```

```python
# Define the label mappings
label_mapping = {
    'Very good stability': 0,
    'Good stability': 1,
    'Poor stability': 2,
    'Very poor stability': 3,
    'Invalid parameters': 4,
    'Very good': 5,
    'Good': 6,
    'Poor': 7,
    'Very poor': 8,
    'Very good (Dense and healthy vegetation)': 9,
    'Good (Moderate and diverse vegetation)': 10,
    'Poor (Sparse and unhealthy vegetation)': 11,
    'Very poor (Absent or dead vegetation)': 12,
    'Very good (Forest)': 13,
    'Good (Grassland)': 14,
    'Poor (Cropland)': 15,
    'Very poor (Urban)': 16
}

# Use the classes to classify the bands
for i in range(len(X_train)):
    stability = Stability(X_train[i][0], X_train[i][1], X_train[i][2]).classify()
    water_presence = WaterPresence(X_train[i][0]).classify()
    vegetation_density = VegetationDensity(X_train[i][2]).classify()
    land_use = LandUse(X_train[i][1]).classify()

    # Create your labels
    y_train = np.concatenate([stability, water_presence, vegetation_density, land_use])

    # Replace the category strings with their corresponding integers
    y_train = np.vectorize(label_mapping.get)(y_train)

    # One-hot encode the labels
    y_train = to_categorical(y_train)

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

```python
# -*- coding: utf-8 -*-
"""GAN MODEL

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1uL333uHd6QDbOKhQ3mW72uwEil0wgF-8
"""

pip install rasterio

import numpy as np
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.utils import to_categorical
import rasterio
from skimage.transform import resize

# Define the paths to your GeoTIFF files
file_paths = [
    '/content/drive/MyDrive/RS RESULTS/GEOSPATIAL TECHNIQUES/MIMOSA MINING COMPANY/GROUNDWATER EXCERCISE FROM GGIS/MIMOSA FEBRUARY 2021/LC08_L1TP_170074_202102
    '/content/drive/MyDrive/RS RESULTS/GEOSPATIAL TECHNIQUES/MIMOSA MINING COMPANY/GROUNDWATER EXCERCISE FROM GGIS/MIMOSA FEBRUARY 2024/LC09_L1TP_170073_202402
    '/content/drive/MyDrive/RS RESULTS/GEOSPATIAL TECHNIQUES/UNKI MINE/SATELLITE IMAGERY/UNKI FEBRUARY 2021/LC08_L1TP_170073_20210204_20210303_02_T1_qb.tif',
    '/content/drive/MyDrive/RS RESULTS/GEOSPATIAL TECHNIQUES/UNKI MINE/SATELLITE IMAGERY/UNKI FEBRUARY 2024/LC09_L1TP_170073_20240205_20240205_02_T1_refl.tif',
    '/content/drive/MyDrive/RS RESULTS/GEOSPATIAL TECHNIQUES/ZIMPLATS/ZIMPLATS GROUNDWATER EXCERCISE FROM GGIS/SATELLITE IMAGERY/FEBRUARY 2021/LC08_L1TP_170073_
    '/content/drive/MyDrive/RS RESULTS/GEOSPATIAL TECHNIQUES/ZIMPLATS/ZIMPLATS GROUNDWATER EXCERCISE FROM GGIS/SATELLITE IMAGERY/FEBRUARY 2024/LC09_L1TP_170073_
]

# Define the desired dimensions (change these to your desired dimensions)
desired_height = 7681
desired_width = 7681

# Initialize an empty list to hold your training data
X_train = []

# Load the GeoTIFF files and extract the feature bands
for file_path in file_paths:
    dataset = rasterio.open(file_path)
    bands = []
    for b in dataset.indexes:
        band = dataset.read(b)

        # Resize the band to the desired dimensions
        band = resize(band, (desired_height, desired_width))

        bands.append(band)

    # Stack the bands to create a 3D array
    bands = np.dstack(bands)

    # Add the 3D array to your training data
    X_train.append(bands)

# Define the generator model
def make_generator_model():
    model = tf.keras.Sequential()
    model.add(layers.Dense(desired_height*desired_width*3, use_bias=False, input_shape=(100,)))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Reshape((desired_height, desired_width, 3)))
    model.add(layers.Conv2DTranspose(128, (5, 5), strides=(1, 1), padding='same', use_bias=False))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same', use_bias=False))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(3, (5, 5), strides=(2, 2), padding='same', use_bias=False, activation='tanh'))

    return model

# Define the discriminator model
def make_discriminator_model():
    model = tf.keras.Sequential()
    model.add(layers.Conv2D(64, (5, 5), strides=(2, 2), padding='same', input_shape=[desired_height, desired_width, 3]))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

    model.add(layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same'))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

    model.add(layers.Flatten())
    model.add(layers.Dense(1))

    return model

# Instantiate the generator and discriminator
generator = make_generator_model()
discriminator = make_discriminator_model()

# Define the loss function and optimizers
cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)
generator_optimizer = tf.keras.optimizers.Adam(1e-4)
discriminator_optimizer = tf.keras.optimizers.Adam(1e-4)

@tf.function
def train_step(images):
    noise = tf.random.normal([BATCH_SIZE, 100])

    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
        generated_images = generator(noise, training=True)

        real_output = discriminator(images, training=True)
```

```python
        fake_output = discriminator(generated_images, training=True)

        gen_loss = cross_entropy(tf.ones_like(fake_output), fake_output)
        disc_loss = cross_entropy(tf.ones_like(real_output), real_output) + cross_entropy(tf.zeros_like(fake_output), fake_output)

    gradients_of_generator = gen_tape.gradient(gen_loss, generator.trainable_variables)
    gradients_of_discriminator = disc_tape.gradient(disc_loss, discriminator.trainable_variables)

    generator_optimizer.apply_gradients(zip(gradients_of_generator, generator.trainable_variables))
    discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator, discriminator.trainable_variables))

# Train the GAN for a certain number of epochs
def train(dataset, epochs):
    for epoch in range(epochs):
        for image_batch in dataset:
            train_step(image_batch)

# Convert your list of images to a numpy array
X_train = np.array(X_train)

# Normalize your images to the range [-1, 1]
X_train = (X_train - 127.5) / 127.5

# Define your batch size and number of epochs
BATCH_SIZE = 32
EPOCHS = 50

# Create TensorFlow Dataset object for shuffling and batch processing
train_dataset = tf.data.Dataset.from_tensor_slices(X_train).shuffle(X_train.shape[0]).batch(BATCH_SIZE)

# Train the GAN
train(train_dataset, EPOCHS)
```

```python
# -*- coding: utf-8 -*-
"""SSD

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1cNbEhQzsCVM9uYoiUgdFSlLnsfer40mg
"""

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

import pandas as pd

# Specify the path of your CSV file
file_path = '/content/drive/MyDrive/RS RESULTS/CLASSIFICATION/SSD /GGIS GROUNDWATER RESOURCES IN AFRICA FEBRUARY 2021.xlsx - SELECT (1).csv'

# Read the CSV file
df = pd.read_csv(file_path)

# Select the required columns
df = df[['longitude', 'latitude', 'Classification']]

# Specify the path of your new CSV file
new_file_path = '/content/drive/MyDrive/RS RESULTS/CLASSIFICATION/SSD /SSD CLASSES.csv'

# Write the data to a new CSV file
df.to_csv(new_file_path, index=False)

pip install rasterio

pip install torch

# Import necessary libraries
import pandas as pd
from PIL import Image
import torch
from torchvision.models.detection import fasterrcnn_resnet50_fpn
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
from torchvision.transforms import functional as F

# Load the CSV file containing data
df = pd.read_csv('/content/drive/MyDrive/RS RESULTS/CLASSIFICATION/SSD /SSD CLASSES.csv')

# Define the classes
classes = df['Classification'].unique().tolist()
num_classes = len(classes)

# Load a pre-trained model for classification and return only the features
model = fasterrcnn_resnet50_fpn(pretrained=True)
in_features = model.roi_heads.box_predictor.cls_score.in_features

# Replace the pre-trained head with a new one
model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)

# Assuming that you are on a CUDA machine, this should print a CUDA device:
device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')

# Move model to the right device
model.to(device)

# Your code for training the model goes here

# For inference
model.eval()
with torch.no_grad():
    for i, row in df.iterrows():
        # Open the image file
        img_path = f"/content/drive/MyDrive/RS RESULTS/CLASSIFICATION/SSD /satellite images/{row['Classification']}.tif"
        img = Image.open(img_path).convert("RGB")
        img_tensor = F.to_tensor(img).unsqueeze_(0).to(device)

        # Make a prediction
        prediction = model(img_tensor)

        # Print the prediction
        print(prediction)



# For inference
model.eval()
tsf_count = 0
with torch.no_grad():
    for i, row in df.iterrows():
        # Open the image file
        img_path = f"/content/drive/MyDrive/RS RESULTS/CLASSIFICATION/SSD /satellite images/{row['Classification']}.tif"
        img = Image.open(img_path).convert("RGB")
        img_tensor = F.to_tensor(img).unsqueeze_(0).to(device)

        # Make a prediction
        prediction = model(img_tensor)

        # Check if the predicted class is 'dam', 'waste water', or 'tailings beach'
        tsf_classes = ['dam', 'waste water', 'tailings beach']
        predicted_class = classes[prediction[0]['labels'][0].item()]
        if predicted_class in tsf_classes:
            print(f"The location in image {img_path} is classified as a TSF.")
            tsf_count += 1
        else:
```

```python
            print(f"The location in image {img_path} is classified as {predicted_class}.")

print(f"\nA total of {tsf_count} TSFs were detected.")

# For inference
model.eval()
tsf_count = 0
with torch.no_grad():
    for i, row in df.iterrows():
        # Open the image file
        img_path = f"/content/drive/MyDrive/RS RESULTS/CLASSIFICATION/SSD /satellite images/{row['Classification']}.tif"
        img = Image.open(img_path).convert("RGB")
        img_tensor = F.to_tensor(img).unsqueeze_(0).to(device)

        # Make a prediction
        prediction = model(img_tensor)

        # Check if the predicted class is 'dam', 'waste water', or 'tailings beach'
        tsf_classes = ['dam', 'waste water', 'tailings beach']
        predicted_class = classes[prediction[0]['labels'][0].item()]
        if predicted_class in tsf_classes:
            tsf_count += 1

print(f"A total of {tsf_count} TSFs were detected.")

# For inference
model.eval()
tsf_count = 0
with torch.no_grad():
    for i, row in df.iterrows():
        # Open the image file
        img_path = f"/content/drive/MyDrive/RS RESULTS/CLASSIFICATION/SSD /satellite images/{row['Classification']}.tif"
        img = Image.open(img_path).convert("RGB")
        img_tensor = F.to_tensor(img).unsqueeze_(0).to(device)

        # Make a prediction
        prediction = model(img_tensor)

        # Check if the predicted class is 'dam' or 'tailings beach'
        tsf_classes = ['dam', 'tailings beach']
        predicted_class = classes[prediction[0]['labels'][0].item()]
        if predicted_class in tsf_classes:
            tsf_count += 1

print(f"A total of {tsf_count} TSFs were detected.")

pip install rasterio

import rasterio
import pandas as pd
import numpy as np

# Load the CSV file containing data
df = pd.read_csv('/content/drive/MyDrive/RS RESULTS/CLASSIFICATION/SSD /SSD CLASSES.csv')

# Open the TIFF file
with rasterio.open('/content/drive/MyDrive/RS RESULTS/CLASSIFICATION/SSD /satellite images/LC08_L1TP_170075_20210220_20210303_02_T1_refl.tif') as src:
    dam_band = src.read(1)  # assuming 'dam' is the first band
    tailings_beach_band = src.read(2)  # assuming 'tailings beach' is the second band

# Define your criteria for each characteristic
dam_criteria = ...
tailings_beach_criteria = ...

# Apply the criteria to each band
dam_mask = dam_band == dam_criteria
tailings_beach_mask = tailings_beach_band == tailings_beach_criteria

# A location is a TSF if it meets all three criteria
tsf_mask = dam_mask & tailings_beach_mask

# Count the number of TSFs
tsf_count = np.count_nonzero(tsf_mask)

print(f"A total of {tsf_count} TSFs were detected.")
```

```python
# -*- coding: utf-8 -*-
"""RNN MODEL

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1tMqdyqi5s1TYDDz6aAGa5HY7XKsAbjhd
"""

from google.colab import auth
auth.authenticate_user()

import os
PROJECT_ID = "TSF MONITOR"
os.environ["GOOGLE_CLOUD_PROJECT"] = PROJECT_ID

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

pip install rasterio

import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import GRU, Dense, Reshape
from tensorflow.keras.utils import to_categorical
import rasterio
from skimage.transform import resize

# Define the paths to your GeoTIFF files
file_paths = [
    '/content/drive/MyDrive/RS RESULTS/GEOSPATIAL TECHNIQUES/MIMOSA MINING COMPANY/GROUNDWATER EXCERCISE FROM GGIS/MIMOSA FEBRUARY 2021/LC08_L1TP_170074_202102
    '/content/drive/MyDrive/RS RESULTS/GEOSPATIAL TECHNIQUES/MIMOSA MINING COMPANY/GROUNDWATER EXCERCISE FROM GGIS/MIMOSA FEBRUARY 2024/LC09_L1TP_170073_202402
    '/content/drive/MyDrive/RS RESULTS/GEOSPATIAL TECHNIQUES/UNKI MINE/SATELLITE IMAGERY/UNKI FEBRUARY 2021/LC08_L1TP_170073_20210204_20210303_02_T1_qb.tif',
    '/content/drive/MyDrive/RS RESULTS/GEOSPATIAL TECHNIQUES/UNKI MINE/SATELLITE IMAGERY/UNKI FEBRUARY 2024/LC09_L1TP_170073_20240205_20240205_02_T1_refl.tif',
    '/content/drive/MyDrive/RS RESULTS/GEOSPATIAL TECHNIQUES/ZIMPLATS/ZIMPLATS GROUNDWATER EXCERCISE FROM GGIS/SATELLITE IMAGERY/FEBRUARY 2021/LC08_L1TP_170073_
    '/content/drive/MyDrive/RS RESULTS/GEOSPATIAL TECHNIQUES/ZIMPLATS/ZIMPLATS GROUNDWATER EXCERCISE FROM GGIS/SATELLITE IMAGERY/FEBRUARY 2024/LC09_L1TP_170073_
]
# Define the desired dimensions
desired_height = 7681
desired_width = 7681

# Define a generator function to load and preprocess the data in chunks
def data_generator(file_paths, chunk_size):
    for file_path in file_paths:
        dataset = rasterio.open(file_path)
        bands = []
        for b in dataset.indexes:
            band = dataset.read(b)
            for i in range(0, band.shape[0], chunk_size):
                for j in range(0, band.shape[1], chunk_size):
                    chunk = band[i:i+chunk_size, j:j+chunk_size]
                    chunk = resize(chunk, (desired_height, desired_width))
                    bands.append(chunk)
        bands = np.dstack(bands)
        yield bands

# Define your RNN model
model = Sequential()
model.add(Reshape((-1, desired_height * desired_width), input_shape=(None, desired_height, desired_width)))
model.add(GRU(30, activation='relu'))
model.add(Dense(4, activation='softmax'))  # num_classes is the number of classes in your classification task

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Use the generator to train the model
chunk_size = 32
steps_per_epoch = sum(band.shape[0] * band.shape[1] for band in bands) // (chunk_size * chunk_size)

# Train the model on a subset of the data
subset_size = 100  # Adjust this value based on the size of your data and the capacity of your environment
for i in range(0, len(file_paths), subset_size):
    subset_paths = file_paths[i:i+subset_size]
    model.fit(data_generator(subset_paths, chunk_size), epochs=10, steps_per_epoch=steps_per_epoch)

import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import GRU, Dense, Reshape
from tensorflow.keras.utils import to_categorical
import rasterio
from skimage.transform import resize

# Define the paths to your GeoTIFF files
file_paths = [
    '/content/drive/MyDrive/RS RESULTS/GEOSPATIAL TECHNIQUES/MIMOSA MINING COMPANY/GROUNDWATER EXCERCISE FROM GGIS/MIMOSA FEBRUARY 2021/LC08_L1TP_170074_202102
    '/content/drive/MyDrive/RS RESULTS/GEOSPATIAL TECHNIQUES/MIMOSA MINING COMPANY/GROUNDWATER EXCERCISE FROM GGIS/MIMOSA FEBRUARY 2024/LC09_L1TP_170073_202402
    '/content/drive/MyDrive/RS RESULTS/GEOSPATIAL TECHNIQUES/UNKI MINE/SATELLITE IMAGERY/UNKI FEBRUARY 2021/LC08_L1TP_170073_20210204_20210303_02_T1_qb.tif',
    '/content/drive/MyDrive/RS RESULTS/GEOSPATIAL TECHNIQUES/UNKI MINE/SATELLITE IMAGERY/UNKI FEBRUARY 2024/LC09_L1TP_170073_20240205_20240205_02_T1_refl.tif',
    '/content/drive/MyDrive/RS RESULTS/GEOSPATIAL TECHNIQUES/ZIMPLATS/ZIMPLATS GROUNDWATER EXCERCISE FROM GGIS/SATELLITE IMAGERY/FEBRUARY 2021/LC08_L1TP_170073_
    '/content/drive/MyDrive/RS RESULTS/GEOSPATIAL TECHNIQUES/ZIMPLATS/ZIMPLATS GROUNDWATER EXCERCISE FROM GGIS/SATELLITE IMAGERY/FEBRUARY 2024/LC09_L1TP_170073_
]
# Define the desired dimensions
desired_height = 7681
desired_width = 7681

# Define a generator function to load and preprocess the data in chunks
def data_generator(file_paths, chunk_size):
    for file_path in file_paths:
        dataset = rasterio.open(file_path)
        bands = []
        for b in dataset.indexes:
            band = dataset.read(b)
            for i in range(0, band.shape[0], chunk_size):
                for j in range(0, band.shape[1], chunk_size):
                    chunk = band[i:i+chunk_size, j:j+chunk_size]
                    chunk = resize(chunk, (desired_height, desired_width))
```

```
                    bands.append(chunk)
        bands = np.dstack(bands)
        yield bands

# Define your RNN model
model = Sequential()
model.add(Reshape((-1, desired_height * desired_width), input_shape=(None, desired_height, desired_width)))
model.add(GRU(30, activation='relu'))
model.add(Dense(4, activation='softmax'))  # num_classes is the number of classes in your classification task

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model on the entire dataset
chunk_size = 32
steps_per_epoch = sum(band.shape[0] * band.shape[1] for band in bands) // (chunk_size * chunk_size)
model.fit(data_generator(file_paths, chunk_size), epochs=10, steps_per_epoch=steps_per_epoch)
```

```python
# -*- coding: utf-8 -*-
"""SOM CLASSIFICATION

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1cabEkC1TOvllZPszC9VfWuvcMfYVoMUw
"""

pip install rasterio

pip install minisom

# Import necessary libraries
import rasterio
import pandas as pd
import numpy as np
from shapely.geometry import Point
import geopandas as gpd
from minisom import MiniSom

# Open the GeoTIFF file
with rasterio.open('/content/drive/MyDrive/RS RESULTS/CLASSIFICATION/GROUNDWATER EXCERCISE FROM GGIS/USGS IMAGERY FOR 4 FEBRUARY 2021/LC08_L1TP_170074_20210204
    # Read the raster data
    data = src.read(1)
    # Get the metadata
    meta = src.meta

# Read the CSV file
df = pd.read_csv('/content/drive/MyDrive/RS RESULTS/CLASSIFICATION/GROUNDWATER EXCERCISE FROM GGIS/CSV DATA GENERATED FROM GGIS/ESTIMATED DEPTHS TO GROUNDWATER

# Create a GeoDataFrame from the CSV data
geometry = [Point(xy) for xy in zip(df['longitude'], df['latitude'])]
gdf = gpd.GeoDataFrame(df, geometry=geometry)

# Extract the raster values at the CSV points
gdf['raster_value'] = [next(src.sample([(pt.x, pt.y)]))[0] for pt in gdf.geometry]

# Prepare data for SOM
data = gdf[['raster_value', 'Average Depth']].values

# Initialize and train the SOM
som = MiniSom(1, 3, 2, sigma=0.5, learning_rate=0.5)  # 1x3 SOM
som.train_random(data, 100)  # trains the SOM with 100 iterations

# Classify each input vector
gdf['cluster'] = [som.winner(x) for x in data]

# Print the average depth of groundwater for each cluster
for i in range(3):
    cluster_gdf = gdf[gdf['cluster'] == (0, i)]
    average_depth = cluster_gdf['Average Depth'].mean()
    print(f"Cluster {i+1}: Average depth = {average_depth}")
```

```python
# -*- coding: utf-8 -*-
"""AUTOENCODER

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1vp5G0nwr79j1ztLe2y3SkKBdu7SJ9SUT
"""

pip install rasterio

import numpy as np
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, Flatten, Reshape
from tensorflow.keras.optimizers import Adam
import rasterio
from skimage.transform import resize

# Define the paths to your GeoTIFF files
file_paths = [
    '/content/drive/MyDrive/RS RESULTS/GEOSPATIAL TECHNIQUES/MIMOSA MINING COMPANY/GROUNDWATER EXCERCISE FROM GGIS/MIMOSA FEBRUARY 2021/LC08_L1TP_170074_202102
    '/content/drive/MyDrive/RS RESULTS/GEOSPATIAL TECHNIQUES/MIMOSA MINING COMPANY/GROUNDWATER EXCERCISE FROM GGIS/MIMOSA FEBRUARY 2024/LC09_L1TP_170073_202402
    '/content/drive/MyDrive/RS RESULTS/GEOSPATIAL TECHNIQUES/UNKI MINE/SATELLITE IMAGERY/UNKI FEBRUARY 2021/LC08_L1TP_170073_20210204_20210303_02_T1_qb.tif',
    '/content/drive/MyDrive/RS RESULTS/GEOSPATIAL TECHNIQUES/UNKI MINE/SATELLITE IMAGERY/UNKI FEBRUARY 2024/LC09_L1TP_170073_20240205_20240205_02_T1_refl.tif',
    '/content/drive/MyDrive/RS RESULTS/GEOSPATIAL TECHNIQUES/ZIMPLATS/ZIMPLATS GROUNDWATER EXCERCISE FROM GGIS/SATELLITE IMAGERY/FEBRUARY 2021/LC08_L1TP_170073_
    '/content/drive/MyDrive/RS RESULTS/GEOSPATIAL TECHNIQUES/ZIMPLATS/ZIMPLATS GROUNDWATER EXCERCISE FROM GGIS/SATELLITE IMAGERY/FEBRUARY 2024/LC09_L1TP_170073_
]

# Define the desired dimensions
desired_height = 7681
desired_width = 7681

# Define the size of the encoded representations
encoding_dim = 32

# Define the input shape
input_img = Input(shape=(desired_height, desired_width, len(file_paths)))

# Flatten the input
flattened_img = Flatten()(input_img)

def data_generator(file_paths, batch_size):
    while True:
        for i in range(0, len(file_paths), batch_size):
            batch_paths = file_paths[i:i+batch_size]
            X_batch = []

            for file_path in batch_paths:
                dataset = rasterio.open(file_path)
                bands = []
                for b in dataset.indexes:
                    band = dataset.read(b)
                    band = resize(band, (desired_height, desired_width))
                    bands.append(band)
                bands = np.dstack(bands)
                X_batch.append(bands)

            yield np.array(X_batch, dtype=object)

# Define the encoder layers
encoded = Dense(128, activation='relu')(flattened_img)
encoded = Dense(encoding_dim, activation='relu')(encoded)

# Define the decoder layers
decoded = Dense(128, activation='relu')(encoded)
decoded = Dense(desired_height*desired_width*len(file_paths), activation='sigmoid')(decoded)

# Reshape the decoded output to the original image shape
decoded = Reshape((desired_height, desired_width, len(file_paths)))(decoded)

# Define the autoencoder model
autoencoder = Model(input_img, decoded)

# Compile the autoencoder model
autoencoder.compile(optimizer=Adam(), loss='binary_crossentropy')

# Use the generator to train the model
batch_size = 32
steps_per_epoch = len(file_paths) // batch_size
model.fit(data_generator(file_paths, batch_size), epochs=10, steps_per_epoch=steps_per_epoch)

# Initialize an empty list to hold your training data
X_train = []

# Load the GeoTIFF files and extract the feature bands
for file_path in file_paths:
    dataset = rasterio.open(file_path)
    bands = []
    for b in dataset.indexes:
        band = dataset.read(b)

        # Resize the band to the desired dimensions
        band = resize(band, (desired_height, desired_width))

        bands.append(band)

    # Stack the bands to create a 3D array
    bands = np.dstack(bands)

    # Add the 3D array to your training data
    X_train.append(bands)

# Convert your list to a numpy array
X_train = np.array(X_train)

# Train the autoencoder
```

```
autoencoder.fit(X_train, X_train, epochs=50, batch_size=32, shuffle=True)
```