1) Ерозія ґрунту - це процес втрати та переміщення ґрунту внаслідок дії вітру, води, льоду або інших фізичних або хімічних факторів. Це серйозна проблема, яка впливає на здоров'я людей, середовище та економіку.

Основні причини ерозії ґрунту:

1. Недбале поводження з ґрунтом, наприклад, надмірне обробітку ґрунту, використання неправильних методів землекористування.
2. Природні катастрофи, такі як повені, зсуви, землетруси, які призводять до зруйнування ґрунту та збільшення швидкості ерозії.
3. Зміна клімату та вплив глобального потепління, які змінюють водний режим, а також розподіл опадів, що може призвести до зростання ерозії.
4. Надмірне випасання тварин, що призводить до пошкодження рослинності та зниження якості ґрунту.

Ерозія грунту може призвести до наступних наслідків:

1. Зниження родючості ґрунту, що призводить до зменшення врожаю та зростання витрат на добрива.
2. Забруднення водних ресурсів та збільшення ризику повені.
3. Зниження біорізноманіття та поширення ерозійних процесів на нові території.
4. Посилення глобальних змін клімату.

Отже, ерозія ґрунту є серйозною проблемою, яка потребує внесення змін у спосіб землекористування, поліпшення водного режиму та захисту рослинності.

Одним зі способів виявлення ерозії ґрунту є аерофотозйомка та космічне зондування: з висоти можна спостерігати зміни в ландшафті, що може свідчити про ерозію ґрунту. Такі методи можуть бути корисними для виявлення ерозійних процесів на великих територіях.

2) Визначення алгоритму виявлення ерозії грунту:

Нам дано фото розміром 10980 x 10980 пікселів формату .jp2, а також маску, яка подана в вигляді об’єкту GeoDataFrame, де зберігаються дані про вже визначенні області ерозії.

Основним методом виявлення ерозії за допомогою аерофотозйомки є

сегментація зображення за допомогою CNN. Я буду використовувати уже готову архітектуру U-Net написану на фреймворкі tensorflow.keras.

Але спочатку нам потрібно буде обробити зображення. Цю задачу можна розбити на підзадачі:

* Створити бінарну маску для нашого зображення
* Розбити її на малі фото розміром 128 x 128 пікселів
* Залишити лише фото, що містять потрібну нам інформацію

1. Для створення бінарної маски можна використати функцію бібліотеки rasterio: rasterize.

def poly\_from\_utm(polygon, transform):

poly\_pts = []

# make a polygon from multipolygon

poly = unary\_union(polygon)

for i in np.array(poly.exterior.coords):

# transfrom polygon to image crs, using raster meta

poly\_pts.append(~transform \* tuple(i))

# make a shapely Polygon object

new\_poly = Polygon(poly\_pts)

return new\_poly

# creating binary mask for field/not\_filed segmentation.

poly\_shp = []

im\_size = (src.meta['height'], src.meta['width'])

for num, row in train\_df.iterrows():

if row['geometry'].geom\_type == 'Polygon':

poly = poly\_from\_utm(row['geometry'], src.meta['transform'])

poly\_shp.append(poly)

else:

for p in row['geometry']:

poly = poly\_from\_utm(p, src.meta['transform'])

poly\_shp.append(poly)

mask = rasterize(shapes=poly\_shp,

out\_shape=im\_size)

bin\_mask\_meta = src.meta.copy()

bin\_mask\_meta.update({'count': 1})

with rasterio.open("mask.jp2", 'w', \*\*bin\_mask\_meta) as dst:

dst.write(mask \* 255, 1)

2. Для розбиття на менші фото я використав функцію patchify одноіменної бібліотеки.

def patch\_img(img, root\_dir, shape):

size\_x = (img.shape[1]//128)\*128

size\_y = (img.shape[1]//128)\*128

img = img[0:size\_x, 0:size\_y]

print(img.shape)

patches\_img = patchify(img, shape, step=128)

patches\_img = np.squeeze(patches\_img)

print(patches\_img.shape)

for i in range(patches\_img.shape[0]):

print(i)

for j in range(patches\_img.shape[1]):

#print(j)

single\_patch = patches\_img[i,j,:,:]

try:

img\_image = cv2.cvtColor(single\_patch, cv2.COLOR\_RGB2BGR)

except:

img\_image = single\_patch

cv2.imwrite(root\_dir+str(i)+str(j)+".tif", img\_image)

img\_dir = r"\train\_img\_1\train\_img"

mask\_dir = r"\train\_mask\_1\train\_mask"

patch\_img(satellite\_img, img\_dir, (128,128,3))

patch\_img(mask\_img, mask\_dir, (128,128))

3. Для того, щоб обрати лише потрібні нам фото, ми просто викинемо ті зображення, які мають частку «пустих» пікселів більше чим певний трешхолд.

img\_dir = r"train\_img\_1"

mask\_dir = r"train\_mask\_1"

for img in range(len(img\_list)):

img\_name=img\_list[img]

mask\_name=mask\_list[img]

temp\_img = cv2.imread(img\_dir+f"\{img\_name}")

temp\_mask = cv2.imread(mask\_dir+f"\{mask\_name}", 0)

val, counts = np.unique(temp\_mask, return\_counts=True)

if (1 - (counts[0]/counts.sum())) > 0.008:

cv2.imwrite(r"Data\train\_images\train"+f"\{img\_name}", temp\_img)

cv2.imwrite(r"Data\train\_mask\train"+f"\{img\_name}", temp\_mask)

В результаті ми отримаємо папку Data, в якій містяться файли з обрізаними зображеннями ділянок земель з ерозією грунту, а також маски для цих зображень. Ці дані вже можна подавати для нейромережі.

Визначення нейромережі:

Як було сказано вище, я використовував архітектуру мережі U-Net.



#ЇЇ реалізація в бібліотеці tensorflow:

inputs = layers.Input((128,128,3))

s = layers.Lambda(lambda x: x / 255)(inputs)

c1 = layers.Conv2D(16, (3,3), activation="relu", kernel\_initializer="he\_normal", padding="same")(s)

c1 = layers.Dropout(0.1)(c1)

p1 = layers.MaxPooling2D((2,2))(c1)

c2 = layers.Conv2D(32, (3,3), activation="relu", kernel\_initializer="he\_normal", padding="same")(p1)

c2= layers.Dropout(0.1)(c2)

c2 = layers.Conv2D(32, (3,3), activation="relu", kernel\_initializer="he\_normal", padding="same")(c2)

p2 = layers.MaxPooling2D((2,2))(c2)

c3 = layers.Conv2D(64, (3,3), activation="relu", kernel\_initializer="he\_normal", padding="same")(p2)

c3 = layers.Dropout(0.1)(c3)

c3 = layers.Conv2D(64, (3,3), activation="relu", kernel\_initializer="he\_normal", padding="same")(c3)

p3 = layers.MaxPooling2D((2,2))(c3)

c4 = layers.Conv2D(128, (3,3), activation="relu", kernel\_initializer="he\_normal", padding="same")(p3)

c4 = layers.Dropout(0.2)(c4)

c4 = layers.Conv2D(128, (3,3), activation="relu", kernel\_initializer="he\_normal", padding="same")(c4)

p4 = layers.MaxPooling2D((2,2))(c4)

c5 = layers.Conv2D(256, (3,3), activation="relu", kernel\_initializer="he\_normal", padding="same")(p4)

c5 = layers.Dropout(0.3)(c5)

c5 = layers.Conv2D(256, (3,3), activation="relu", kernel\_initializer="he\_normal", padding="same")(c5)

u6 = layers.Conv2DTranspose(128, (2, 2), strides=(2,2), padding="same")(c5)

u6 = layers.concatenate([u6, c4])

c6 = layers.Conv2D(128, (3,3), activation="relu", kernel\_initializer="he\_normal", padding="same")(u6)

c6 = layers.Dropout(0.2)(c6)

c6 = layers.Conv2D(128, (3,3), activation="relu", kernel\_initializer="he\_normal", padding="same")(c6)

u7 = layers.Conv2DTranspose(128, (2, 2), strides=(2,2), padding="same")(c6)

u7 = layers.concatenate([u7, c3])

c7 = layers.Conv2D(128, (3,3), activation="relu", kernel\_initializer="he\_normal", padding="same")(u7)

c7 = layers.Dropout(0.2)(c7)

c7 = layers.Conv2D(128, (3,3), activation="relu", kernel\_initializer="he\_normal", padding="same")(c7)

u8 = layers.Conv2DTranspose(128, (2, 2), strides=(2,2), padding="same")(c7)

u8 = layers.concatenate([u8, c2])

c8 = layers.Conv2D(128, (3,3), activation="relu", kernel\_initializer="he\_normal", padding="same")(u8)

c8 = layers.Dropout(0.2)(c8)

c8 = layers.Conv2D(128, (3,3), activation="relu", kernel\_initializer="he\_normal", padding="same")(c8)

u9 = layers.Conv2DTranspose(128, (2, 2), strides=(2,2), padding="same")(c8)

u9 = layers.concatenate([u9, c1])

c9 = layers.Conv2D(128, (3,3), activation="relu", kernel\_initializer="he\_normal", padding="same")(u9)

c9 = layers.Dropout(0.1)(c9)

c9 = layers.Conv2D(128, (3,3), activation="relu", kernel\_initializer="he\_normal", padding="same")(c9)

outputs = layers.Conv2D(1, (1, 1), activation="sigmoid")(c9)

model = Model(inputs=[inputs], outputs=(outputs))

model.compile(optimizer="adam", loss="binary\_crossentropy", metrics=[IoU(num\_classes=2, target\_class\_ids=[1])])

model.summary()

callbacks = [

keras.callbacks.ModelCheckpoint(".h5", save\_best\_only=True),

keras.callbacks.EarlyStopping(patience = 2, monitor="val\_io\_u")

]

epochs = 300

model.fit(train\_x, train\_y, epochs=epochs, validation\_split=0.1, callbacks=callbacks)

Висновок: Ерозія ґрунту — це процес відшарування, перенесення та осадження ґрунту водою, вітром або льодом. Це серйозна екологічна проблема, яка впливає на продуктивність сільського господарства, якість води та біорізноманіття. Для її виявлення використовують багато методів, одним з яких є аерофотозйомка та космічне зондування. Для його реалізаці можна використовувати згорткові нейроні мережі. В моїй роботі було використано архітектуру U-Net, з метрикою IoU(Коефіцієнт Жаккарта). Однак цей метод не ідеальний та потребує доопрацювання. Майбутні дослідження повинні включати гіперспектральні та SAR-зображення з дуже високою роздільною здатністю для класифікації ґрунтів та кращого виявлення ерозії ґрунту.