1) Soil erosion is the process of loss and movement of soil due to the action of wind, water, ice, or other physical or chemical factors. It is a serious problem that affects human health, the environment, and the economy.

The main causes of soil erosion are:

* Careless handling of soil, such as excessive tillage, use of improper land use methods.
* Natural disasters such as floods, landslides, earthquakes, which lead to soil destruction and increased erosion rates.
* Climate change and the impact of global warming, which alter the water regime, as well as the distribution of precipitation, which can lead to increased erosion.
* Overgrazing, which damages vegetation and reduces soil quality.

Soil erosion can lead to the following consequences:

* Reduced soil fertility, resulting in decreased yields and increased fertilizer costs.
* Pollution of water resources and increased risk of floods.
* Decreased biodiversity and the spread of erosion processes to new areas.
* Intensification of global climate change.

Therefore, soil erosion is a serious problem that requires changes in land use practices, improvement of water regimes, and protection of vegetation.

One way to detect soil erosion is through aerial photography and satellite imaging: from a height, changes in the landscape can be observed, which may indicate soil erosion. Such methods can be useful for detecting erosion processes over large areas.

2) Definition of soil erosion detection algorithm:

We have been given a photo of size 10980 x 10980 pixels in .jp2 format, as well as a mask, which is provided as a GeoDataFrame object, where data on already identified erosion areas are stored. The main method of detecting erosion using aerial photography is image segmentation using a CNN. I will be using a pre-existing U-Net architecture written in the tensorflow.keras framework.

But first, we need to process the image. This task can be divided into subtasks:

• Create a binary mask for our image

• Split it into small photos with a size of 128 x 128 pixels

• Keep only the photos that contain the necessary information for us

1. The rasterio library function "rasterize" can be used to create a binary mask.

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. To split the image into smaller photos, I used the patchify function from the library of the same name.

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. To select only the necessary images, we simply discard the images that have a "blank" pixel fraction greater than a certain threshold..

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)

As a result, we will get a folder called "Data", which contains files with cropped images of land areas with soil erosion, as well as masks for these images. These data can already be fed to the neural network.

Defining the neural network:

As mentioned above, I used the U-Net network architecture.



#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)

Conclusion: Soil erosion is the process of detachment, transportation, and deposition of soil by water, wind, or ice. It is a serious ecological problem that affects agricultural productivity, water quality, and biodiversity. Many methods are used to detect it, including aerial photography and satellite imagery. Convolutional neural networks can be used for its implementation. In my work, the U-Net architecture with the IoU (Jaccard coefficient) metric was used. However, this method is not perfect and requires further improvement. Future research should include hyperspectral and SAR images with very high resolution for soil classification and better detection of soil erosion.