Report for Imihigo June 2025

Task: Train existing ML models for fields delineation and land cover classification to improve and achieve acceptable accuracy

These tasks concerns to land cover and land use classification which is crucial and important for national statistical offices (NSOs) to produce an official statistic especially in seasonal agricultural surveys.

During this project, I have used earth observation data such as imagery for field delineation and other measurements taken by satellite sensors for land cover classification. I explored how such data can be used to improve the way agricultural statistics are produced.

In this report I will explain in details what I have done.

Task 1: Field delineation

Field delineation is the process of identifying and outlining individual agricultural fields or farm plots from satellite or aerial imagery using image processing or machine learning techniques. It is very important for crop mapping, area estimation, and official statistics production.

In this way, I have done the following steps to make it happen: data collection from sentinel-2, data annotation, data preprocessing, splitting data, model selection, model validation data testing.

1. Data collection

Sentinel-2 satellite has been used as source of data; whereby, I subscribed to google earth engine in order to access its API that allowed me to connect to image cloud storage.

```
Import ee
ee.Authenticate()
ee.Initialize(project="ee-hatangimanabenon0")
```

The image collection used in this study was **"COPERNICUS/S2_SR_HARMONIZED"**, filtered to the date range March 1, 2023, to June 30, 2023. To ensure better visual quality and minimize cloud interference, only images with less than 10% cloud cover were retained. Using specific georeferenced points obtained from survey data, Sentinel-2 imagery was extracted around those coordinates in Rwanda, with each image covering approximately a 1.11 km × 1.11 km area.

The resulting imagery was then used for annotation to generate segmentation masks, which served as training data for a U-Net deep learning model aimed at performing land cover segmentation.

```
def get_rectangle_from_point(lon, lat, delta=0.01):
    return ee.Geometry.Rectangle([lon - delta, lat - delta, lon + delta, lat + delta])
```

```python
def maskClouds(image):
    qa = image.select('QA60')
    cloudBitMask = 1 << 10
    cirrusBitMask = 1 << 11
    mask = qa.bitwiseAnd(cloudBitMask).eq(0).And(qa.bitwiseAnd(cirrusBitMask).eq(0))
    return image.updateMask(mask).copyProperties(image, ['system:time_start'])

# Store export tasks
task_list = []

for idx, row in sample_df.iterrows():
    lon, lat = row['x_coord'], row['y_coord']
    crop_type = row['crop_name_eng']

    rect = get_rectangle_from_point(lon, lat)

    # Sentinel-2 collection and cloud masking
    collection = ee.ImageCollection('COPERNICUS/S2_SR_HARMONIZED') \
        .filterBounds(rect) \
        .filterDate('2023-03-01', '2023-06-30') \
        .filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 10)) \
        .map(maskClouds)

    # Composite and bands
    composite = collection.median()
    rgb = composite.select(['B4', 'B3', 'B2']).toFloat()
    ndvi = composite.normalizedDifference(['B8', 'B4']).rename('NDVI').toFloat()

    stacked = rgb.addBands(ndvi).clip(rect)

    task = ee.batch.Export.image.toDrive(
        image=stacked,
        description=f'{crop_type}_{idx}_export',
        folder='Sentinel_CropType_Images',
        fileNamePrefix=f'{crop_type}_{idx}',
        region=rect,
        scale=10,
        maxPixels=1e13
    )

    task.start()
    task_list.append(task)

print(f"Started {len(task_list)} export tasks to Google Drive.")
```

2. Annotation

After acquiring Sentinel-2 satellite imagery containing the RGB bands, I used these images to create masks that served as the ground truth labels. This labeling process enables a deep learning model to learn and recognize specific land cover features from the input imagery.

To accomplish this, I used the LabelMe annotation tool, which allowed me to manually draw boundaries around features of interest and assign class labels. The annotations were saved as JSON files containing the polygon coordinates and their corresponding class information. These JSON files were then processed using a Python script to convert them into mask images, which were used as input for training a U-Net segmentation model. **I labeled a total of 79 images, focusing on two classes: crop land and non-crop land. This binary classification enabled me to generate binary masks that served as ground truth labels for the corresponding images.**

3. Data pre processing

Preprocessing of data is really crucial for better performance of a model. That's why I preprocessed an image as well as corresponding labels by resizing them at 128x128 spatial dimension, augmenting them and loading them into segmentation model in appropriate order in order to ensure good performance. I achieved this stage by creating a pipeline that run a script for this action.

```
# --- Augmentation pipeline ---
augmentation = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.VerticalFlip(p=0.5),
    A.Rotate(limit=30, p=0.5),
    A.RandomBrightnessContrast(p=0.3),
    A.ShiftScaleRotate(shift_limit=0.05, scale_limit=0.1, rotate_limit=15, p=0.5),
], additional_targets={'mask': 'mask'})

# --- Define load + augment function ---
def load_and_augment(img_path, mask_path, augment_n=5):
    img = load_img(img_path, target_size=(IMG_SIZE, IMG_SIZE))
    img = img_to_array(img).astype(np.uint8)

    mask = load_img(mask_path, target_size=(IMG_SIZE, IMG_SIZE), color_mode='grayscale')
    mask = img_to_array(mask).astype(np.uint8)[..., 0]

    augmented_images, augmented_masks = [], []

    # Include original image
    img_norm = img.astype(np.float32) / 255.0
    mask_onehot = tf.one_hot(mask, NUM_CLASSES)
    augmented_images.append(img_norm)
    augmented_masks.append(mask_onehot)

    # Generate augmentations
    for _ in range(augment_n):
        augmented = augmentation(image=img, mask=mask)
        aug_img = augmented['image'].astype(np.float32) / 255.0
        aug_mask = tf.one_hot(augmented['mask'], NUM_CLASSES)

        augmented_images.append(aug_img)
        augmented_masks.append(aug_mask)

    return augmented_images, augmented_masks

# --- Load and augment dataset ---
image_paths = sorted(glob(os.path.join(image_dir, "*.jpg")))
mask_paths = sorted(glob(os.path.join(mask_dir, "*.png")))

assert len(image_paths) == len(mask_paths), "Mismatch image/mask counts"

all_images, all_masks = [], []

for i_path, m_path in zip(image_paths, mask_paths):
    imgs, msks = load_and_augment(i_path, m_path, augment_n=AUG_PER_IMAGE)
    all_images.extend(imgs)
    all_masks.extend(msks)

images = np.array(all_images)
masks = np.array(all_masks)

print("Augmented dataset shape:")
print("Images:", images.shape)
print("Masks:", masks.shape)
```

4. Data splitting

To ensure that model will not overfit during the model training as well as bias when you want to test or validate a model; it is good practice to split your dataset into train set and test set. I have done this by splitting dataset into two parts; one for training and other for testing model.

```
from sklearn.model_selection import train_test_split

# Adjust ratio and random state as needed
X_train, X_test, y_train, y_test = train_test_split(
    images, masks, test_size=0.2, random_state=42
)
```

5. Model Selection and training

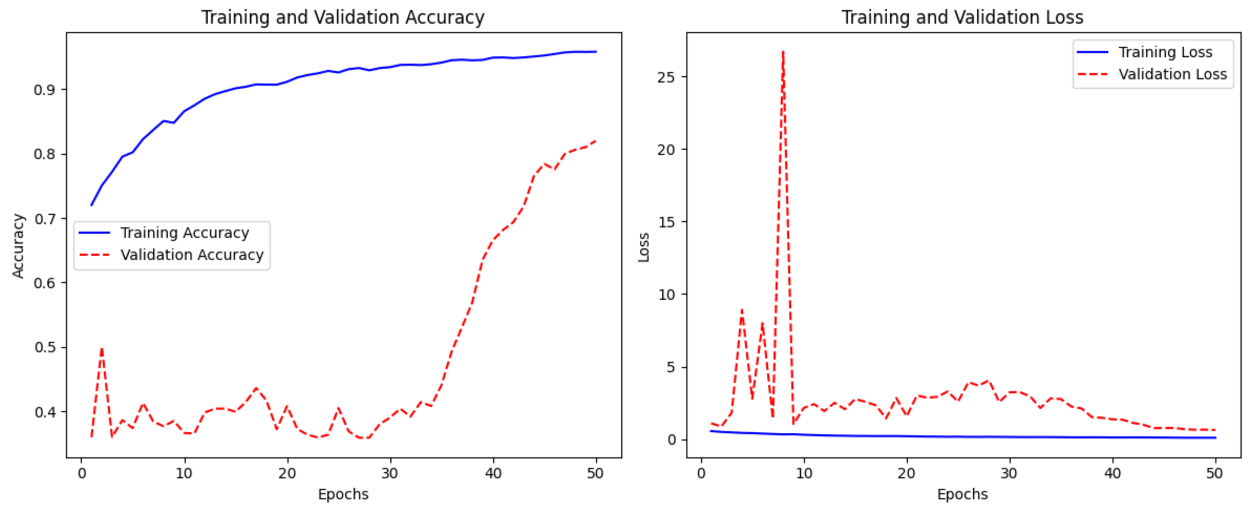
Since the task involves semantic segmentation, I selected a model specifically designed to label each pixel in an image enabling the identification of features based on predefined class labels. For this purpose, I chose the U-Net architecture, which is well-suited for segmentation tasks.

To enhance its performance, I used a pretrained ResNet50 encoder, trained on the large-scale ImageNet dataset. The use of pretrained weights is particularly beneficial when working with a limited dataset, as the model has already learned to extract a wide range of useful visual features. This helps improve both convergence speed and model accuracy during training.

```
def build_unet_resnet50(input_shape=(128, 128, 3), num_classes=2):
    base_model = ResNet50(include_top=False, weights='imagenet', input_shape=input_shape)
    skip1 = base_model.get_layer("conv1_relu").output       # 64x64
    skip2 = base_model.get_layer("conv2_block3_out").output  # 32x32
    skip3 = base_model.get_layer("conv3_block4_out").output  # 16x16
    skip4 = base_model.get_layer("conv4_block6_out").output  # 8x8

    x = base_model.output  # 4x4

    def upsample_concat(x, skip, filters):
        x = layers.Conv2DTranspose(filters, 2, strides=2, padding='same')(x)
        x = layers.Concatenate()([x, skip])
        x = layers.Conv2D(filters, 3, padding='same', activation='relu')(x)
        x = layers.Conv2D(filters, 3, padding='same', activation='relu')(x)
        return x

    x = upsample_concat(x, skip4, 512)  # 4->8
    x = upsample_concat(x, skip3, 256)  # 8->16
    x = upsample_concat(x, skip2, 128)  # 16->32
    x = upsample_concat(x, skip1, 64)   # 32->64

    # Add one more upsampling to go 64 -> 128 (final output size)
    x = layers.Conv2DTranspose(32, 2, strides=2, padding='same')(x)  # 64->128
    x = layers.Conv2D(32, 3, padding='same', activation='relu')(x)
    x = layers.Conv2D(32, 3, padding='same', activation='relu')(x)

    x = layers.Conv2D(num_classes, 1, padding='same')(x)
    outputs = layers.Activation('softmax')(x)

    model = Model(base_model.input, outputs)
    return model

model=build_unet_resnet50(input_shape=(128, 128, 3), num_classes=2)
model.summary()

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

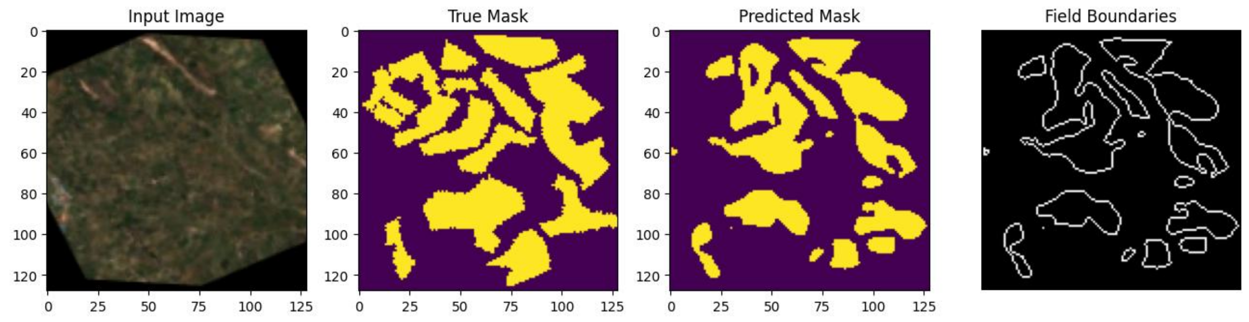
```
# --- Train ---
history=model.fit(train_dataset, validation_data=test_dataset, epochs=50)
```

Model was trained for 50 number of epochs. And final attained training accuracy is 95.9 % and validation accuracy is 81.98%. the bellow graph is showing accuracies and losses obtained on each epoch for tracking the learning process.



6. Model testing and Results

I have tested a trained model using sample from test set; therefore, the following result has been attained.



As clearly illustrated in the figure, the first image (from the left) represents the original input image, while the second image is the corresponding ground truth mask (true label). When this original image is passed through the trained model, it produces the third image, which is the predicted mask. This predicted mask is then used to extract field boundaries, enabling the accomplishment of the field delineation task.

In both the true and predicted masks:

- The yellow regions represent crop land fields
- The purple regions indicate non-crop areas

In the final field boundary image, the outlined regions correspond to crop land fields derived from the prediction.

From these results, we can observe that the model performs quite well, as the predicted mask closely matches the ground truth. However, while the current performance is promising, further improvements are still necessary to achieve higher accuracy and robustness especially for deployment in real-world applications.

By using the detected field boundaries, the following insights are possible: crop mapping, area estimation, and official statistics which are crucial to national statistical offices (NSOs) and decision makers.

Challenges

- Computation resources
- Low quality of an images (low resolution)
- Bad annotation

Future work

I plan to continue exploring ways to make this project feasible and scalable, with the goal of enabling National Statistical Offices (NSOs) to generate valuable insights using Earth Observation (EO) data.

Additionally, I aim to improve the annotation process to enhance the efficiency and quality of model training, which is critical for achieving reliable and accurate results in practical applications.

Task 2: Land cover classification

Land cover classification is a second task that I worked on. It is a process of identifying and outlining which feature occupied a certain region a particular period of time. It is very significant to NSOs, researchers, and government agencies for better decision making.

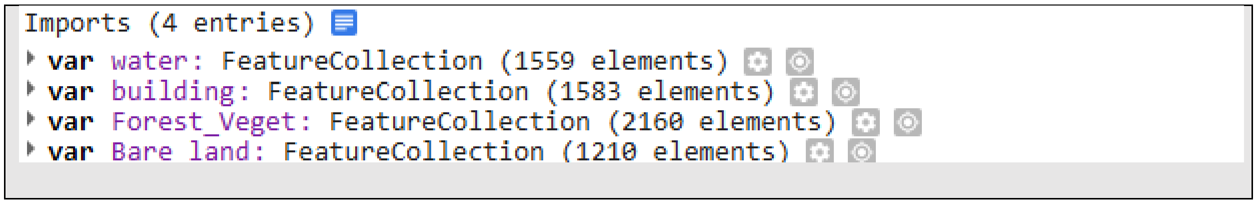
For this task, I have involved in data collection (collecting training samples from sentinel-2 satellite with help of google earth engine capabilities). In this section, I'm going to explain in details how data for training model obtained.

After login to google earth engine, I have been able to access code editor where I wrote the following code to filter region of Rwanda using global administrative boundary available to google earth engine as a region of interest.

```
// 1. Load Rwanda boundary
var rwanda = ee.FeatureCollection("FAO/GAUL/2015/level0")
  .filter(ee.Filter.eq('ADM0_NAME', 'Rwanda'));
Map.centerObject(rwanda, 7);
Map.addLayer(rwanda, {color: 'black'}, 'Rwanda');
```

I extracted Sentinel-2 imagery for the defined boundary, using the following bands and indices: B2, B3, B4, B5, B6, B7, B8, B8A, B11, B12, NDVI, NDWI, and NDMI. Only images with cloud cover less than 10% were considered, and the date range was set from January 1, 2023 to December 31, 2023.

From the extracted imagery, I generated training samples by assigning labels to specific pixels corresponding to the target land cover features. The considered classes were: water, building, forest vegetation, and bare land. These training samples were created manually, ensuring accurate representation of each class. The resulting training dataset consists of 14 features (columns) and a total of 6,510 observations (rows).

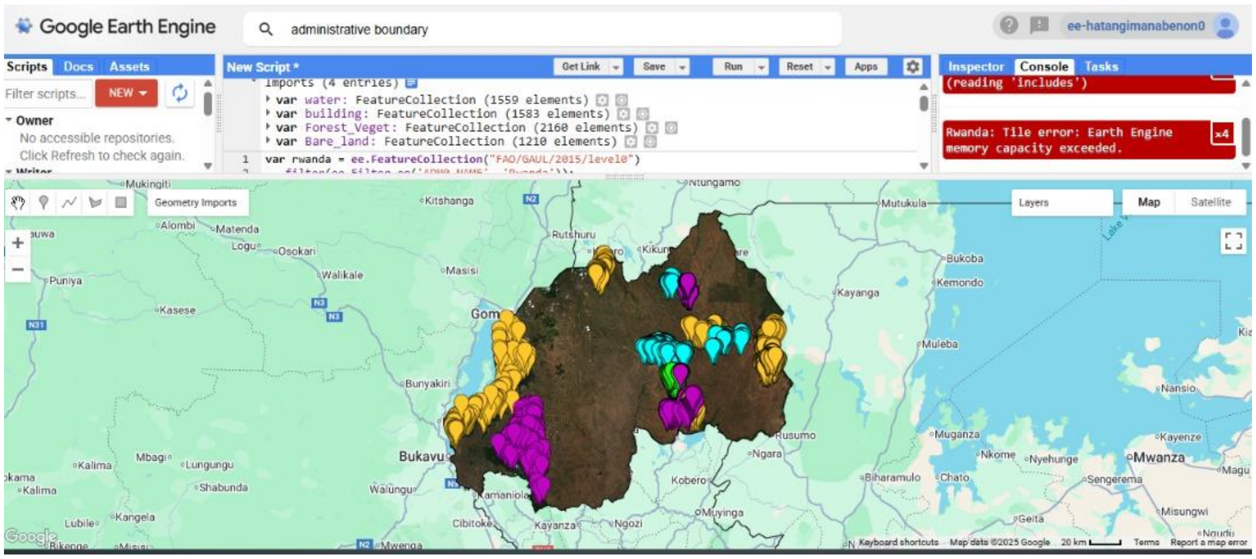


```
Imports (4 entries)
▸ var water: FeatureCollection (1559 elements)
▸ var building: FeatureCollection (1583 elements)
▸ var Forest_Veget: FeatureCollection (2160 elements)
▸ var Bare land: FeatureCollection (1210 elements)
```

```
// 1. Load Rwanda boundary
var rwanda = ee.FeatureCollection("FAO/GAUL/2015/level0")
  .filter(ee.Filter.eq('ADM0_NAME', 'Rwanda'));
Map.centerObject(rwanda, 7);
Map.addLayer(rwanda, {color: 'black'}, 'Rwanda');

// 2. Cloud masking and vegetation indices
function maskClouds(image) {
  var qa = image.select('QA60');
  var cloudBitMask = 1 << 10;
  var cirrusBitMask = 1 << 11;
  var mask = qa.bitwiseAnd(cloudBitMask).eq(0)
             .and(qa.bitwiseAnd(cirrusBitMask).eq(0));
  return image.updateMask(mask).copyProperties(image, ['system:time_start']);
}

function computeIndices(image) {
  var ndvi = image.normalizedDifference(['B8', 'B4']).rename('NDVI');
  var ndwi = image.normalizedDifference(['B3', 'B8']).rename('NDWI');
  var ndmi = image.normalizedDifference(['B8', 'B11']).rename('NDMI');
  return image.addBands([ndvi, ndwi, ndmi]);
}

// 3. Prepare Sentinel-2 composite
var s2 = ee.ImageCollection("COPERNICUS/S2_SR_HARMONIZED")
  .filterBounds(rwanda)
  .filterDate('2023-01-01', '2023-12-31')
  .filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 10))
  .map(maskClouds)
  .map(computeIndices)
  .median()
  .clipToCollection(rwanda);

var bands = ['B2','B3','B4','B5','B6','B7','B8','B8A','B11','B12','NDVI','NDWI','NDMI'];
var composite = s2.select(bands);
```

```
Map.addLayer(composite, {bands: ['B4', 'B3', 'B2'], min: 0, max: 3000}, 'S2 RGB');

// 4. Assign label_class to each imported polygon collection
var water_labeled = water.map(function(f) {
  return f.set('label_class', 0);
});
var building_labeled = building.map(function(f) {
  return f.set('label_class', 1);
});
var Forest_Veget_labeled = Forest_Veget.map(function(f) {
  return f.set('label_class', 2);
});
var Bare_land_labeled = Bare_land.map(function(f) {
  return f.set('label_class', 3);
});

// 5. Merge labeled polygons into one FeatureCollection
var trainingPolygons = water_labeled
  .merge(building_labeled)
  .merge(Forest_Veget_labeled)
  .merge(Bare_land_labeled);

Map.addLayer(trainingPolygons, {}, 'Training Polygons');

// 6. Sample pixels from the composite using labeled polygons
var trainingSamples = composite.sampleRegions({
  collection: trainingPolygons,
  properties: ['label_class'],
  scale: 10,
  geometries: true
})

Map.addLayer(trainingSamples, {}, 'Training Samples');
print('Number of training samples:', trainingSamples.size());

// 7. Export the samples table to Google Drive
Export.table.toDrive({
  collection: trainingSamples,
  description: 'Rwanda_LandCover_TrainingSampless_1',
  fileFormat: 'CSV'
});
```

The image below shows the Google Earth Engine Code Editor, highlighting the regions where training samples were collected for the four land cover classes: water, building, forest & vegetation, and bare land. All samples were manually selected within the boundaries of Rwanda. This approach was chosen due to the absence of survey data, which would otherwise provide verified ground truth labels for the features observed in the Sentinel-2 imagery.



```
Imports (4 entries)
 var water: FeatureCollection (1559 elements)
 var building: FeatureCollection (1583 elements)
 var Forest_Veget: FeatureCollection (2160 elements)
 var Bare_land: FeatureCollection (1210 elements)
var rwanda = ee.FeatureCollection("FAO/GAUL/2015/level0")
```

```
(reading 'includes')

Rwanda: Tile error: Earth Engine memory capacity exceeded.   x4
```