प्रगत संगणन विकास केंद्र, नोएडा
Centre for Development of Advanced Computing, Noida

A Project on

Feature Extraction & Mapping of Land use Feature from Aerial Imagery

Submitted In Partial Fulfilment of the Requirement for the Award of

Post Graduate Diploma in Artificial Intelligence (PG-DAI)

Under the Guidance of

Ms Saruti Gupta and Mr. Shivam Pandey

(Project Guide)

CDAC, B-30, Institutional Area, Sector-62
Noida (Uttar Pradesh)-201307
CDAC, NOIDA

Submitted By:

Aditya Kuche	Sanket Gosavi	Vasu Upadhyay	Abhishek Thakur
PRN: 230320528003	**PRN: 230320528005**	**PRN: 230320528024**	**PRN: 230320528002**

CONTENTS

CERTIFICATE

This is to certify that Report entitled **"Feature Extraction & Mapping of Buildings, Woodlands, Water and Roads from Aerial Imagery"** which is submitted Aditya Kuche, Sanket Gosavi, Vasu Upadhyay and Abhishek Thakur in partial fulfilment of the requirement for the award of **Post Graduate Diploma in Artificial Intelligence** (PG-DAI) to **CDAC, Noida** is a record of the candidates own work carried out by them under my supervision.

The documentation embodies results of original work, and studies are carried out by the student themselves and the contents of the report do not from the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

Mrs Saruti Gupta and Mr. Shivam Pandey

(Project Guide)

ACKNOWLEDGEMENT

We would like to express our best sense of gratitude & endeavour with respect to **Mrs Saruti Gupta and Mr. Shivam Pandey (Project Guide)** CDAC, Noida for suggesting the problems scholarly guidance and expert supervision during this project. Special thanks to **Mr. Ravi Payal (Program Coordinator).**

We are very thankful to our project guides for constant simulated discussion, encouraging new ideas about this project.

Aditya Kuche
Sanket Gosavi
Vasu Upadhyay
Abhishek Thakur

ABSTRACT

Effective monitoring of land cover and land use is vital for sustainable natural resources management. Our project explores the transformative potential of automatic visual mapping through the aerial imagery with advanced techniques like Convolutional Neural Networks (CNNs) such as U-Net and ResNet. This integration enables precise and efficient land cover assessment and rapid change detection. Aerial imagery offers the advantage of significantly higher pixel resolution compared to satellite data, facilitating detailed mapping. However, a critical challenge lies in the scarcity of specialized aerial datasets tailored for semantic segmentation, especially in rural areas with sub-decimetre resolution. These datasets demand manual fine-labelling and encompass critical environmental features such as buildings, woodlands, water bodies, and roads. This highlights the importance of addressing this gap and emphasizes the role of deep learning, with pre-trained weights from ImageNet, in advancing semantic segmentation for accurate land use prediction, paving the way for informed decision-making in forestry and public administration. The Landcover.ai dataset is a valuable resource in this context, facilitating research and development in land cover classification.

INTRODUCTION

Introduction to the Problem Statement:

To analyze the present situation of the landscape for the Local planning and development department and help to reduce the cost and time required for the survey. As to produce the different feature layer required many manual processes, ultimately increase in cost and time for analyzing the urban pattern changes.

The Possible Solution:

The integration of high-resolution aerial imagery and advanced machine learning techniques, particularly deep learning, offers a promising solution to the challenges associated with precise land cover and land use monitoring. By harnessing the power of neural networks, it becomes feasible to automate the segmentation and classification of various land features. This approach not only enhances accuracy but also significantly accelerates the process of change detection. The higher pixel resolution of aerial imagery, when compared to satellite data, enables the identification of finer details, such as individual buildings, small water bodies, and intricate road networks.

To address the scarcity of suitable datasets, initiatives can be undertaken to create comprehensive aerial datasets that cover rural areas with the necessary granularity, encompassing features like buildings, forests, water bodies, and roads. These datasets should also include finely labeled instances for training deep learning models effectively. Collaborations between remote sensing experts and machine learning practitioners are crucial in curating these datasets, as they require domain knowledge in both fields.

The proposed solution's efficacy lies in its potential to provide near, aiding policymakers, urban planners, and environmentalists in making informed decisions. The automation of land cover classification and change detection contributes to efficient resource allocation, early detection of environmental changes, and the formulation of sustainable land management strategies.

In conclusion, leveraging high-resolution aerial imagery coupled with deep learning techniques presents a robust solution to the challenges faced in accurate land cover and land use monitoring. This interdisciplinary approach holds the key to unlocking actionable insights for better land management, fostering environmental preservation, and facilitating sustainable development

Data Pre-processing

The process begins with an exploration of the dataset's characteristics to gain insights into the challenges of accurate land cover and land use monitoring. Satellite and aerial imagery have transformed this field, but limitations persist in rural areas due to data resolution. To overcome these challenges, a comprehensive solution involving high-resolution aerial imagery and advanced machine learning techniques is proposed. Images are cropped into patches that align with the desired patch size, and similarly, corresponding mask patches are created. Subsequently, a selection process identifies patches containing meaningful information, ensuring they encompass non-zero labels of at least 5% in the mask. These useful image and mask pairs are then saved for further processing. The data is further organized by splitting it into training and testing subsets. This streamlined process, combining image pacification, mask processing, data curation, and splitting, forms the foundation for effective land cover and land use monitoring using semantic segmentation with deep learning algorithms.

Need of dividing the large image into small patches

Dividing a large image and its respective mask into smaller patches is a common technique in deep learning for several reasons:

Dividing large images into patches is a practical strategy in deep learning to manage memory, improve training efficiency, and enhance the model's ability to generalize and handle large-scale images. The choice of patch size may depend on the specific task and dataset characteristics.

1. Memory Efficiency: Large images can require a significant amount of memory to process, especially in deep neural networks with many layers and parameters. By breaking them down into smaller patches, you can fit them into the memory of a GPU or other hardware accelerator, making it possible to train and process them efficiently.

2. Parallelism: GPUs are designed to process data in parallel, and by splitting the image into patches, you can take advantage of this parallelism. Each patch can be processed independently, speeding up training and inference times.

3. Translation Invariance: In many computer vision tasks, it's important to capture translation-invariant features. By using patches, you ensure that the network can learn features at different positions in the image, which helps it generalize better to unseen data. Without patches, the network might focus too much on specific spatial locations and struggle with translation-invariant tasks.

4. Reducing Overfitting: Deep learning models can have a large number of parameters, and dividing the image into patches can act as a form of regularization. It forces the model to learn features that are relevant across different parts of the image, reducing the risk of overfitting to specific spatial patterns.

5. Data Augmentation: Patches can also be used as a form of data augmentation. By randomly selecting patches during training, you can introduce variability and increase the effective size of your training dataset. This helps the model become more robust to variations in object position, scale, and orientation.

6. Scalability: Dividing images into patches makes it easier to scale your deep learning system to handle larger images. Instead of completely redesigning your model and hardware infrastructure, you can simply adjust the patch size to accommodate larger inputs.

7. Semantic Segmentation: In tasks like semantic segmentation, where you need to classify each pixel in an image, dividing it into patches can simplify the problem. Each patch can be treated as a separate input, and the model can predict the class of each pixel within the patch. Then, the results from different patches can be combined to produce the final segmentation map.

Sorting the useful dataset from the whole dataset using the threshold of 5%. This approach can save computational resources and improve model performance.

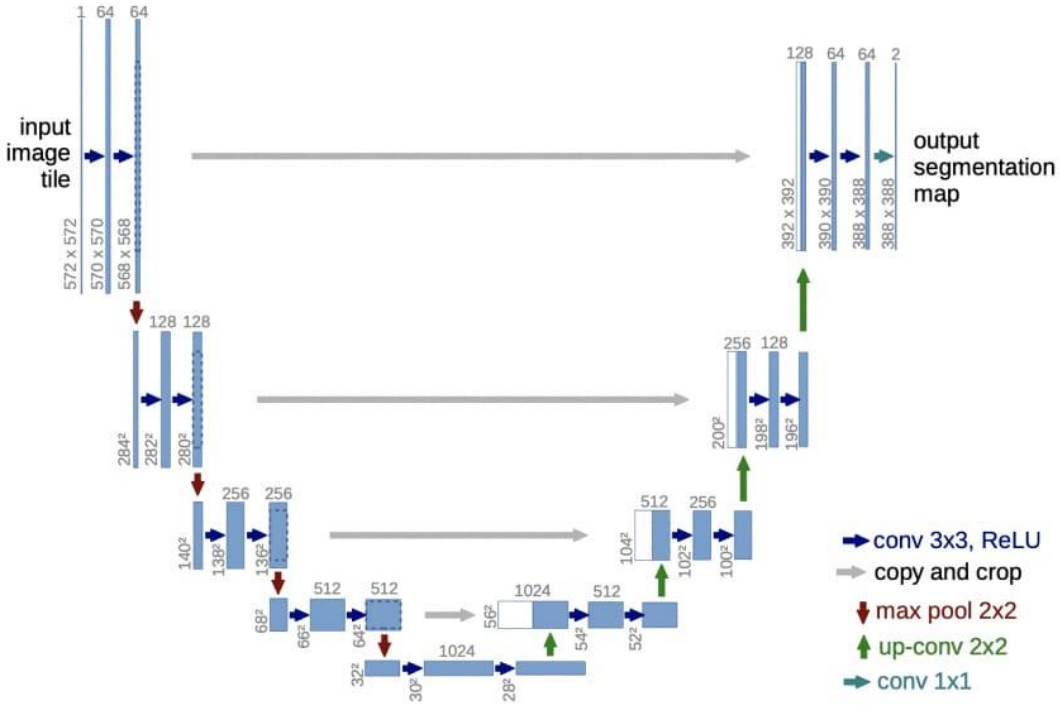
1. Label Thresholding: Determine a threshold for what constitutes a "useful" label. In your case, you mentioned labels that are not equal to 0, so your threshold is 0.Keep the images and labels that meet the threshold criteria for further processing. These are the images with enough useful information.

2. Filtering: Set a threshold for the minimum percentage of useful area that an image should have to be considered for further processing. You mentioned at least 5% useful area, so any image with less than 5% useful area can be discarded. You can adjust this threshold based on your specific requirements and dataset characteristics. For example, in semantic segmentation tasks, you might have pixel-wise labels where 0 represents the background, and other values represent different object classes. You can create a binary mask or multiple class where pixels with non-zero values are set to 1, and all others are set to 0.

3. Calculate Useful Area: Compute the percentage of useful area in each image based on the binary mask you generated. You can do this by counting the number of non-zero pixels and dividing it by the total number of pixels in the mask.

Model Training

As U-net is the best model for the segmentation of the images (non-structured dataset) and Res-Net as a backbone structure. Increasing the effects of object feature detection weights of ImageNet is also use while compiling it.



Post-processing

Georeferenced is required to the segmented output image as it does not contain any spatial reference system and projection. Implementing the image-to-image georeferencing to the output image. As then feature extraction and segmentation from the image is never the end result but it is a intermediatory step to obtain result. For the analysis of data and the feature converting the raster image into the polygon using Gdal library and Geopandas is used to handle the shapefile format files and to address the attributes about the shape and geometry of the various features.

CODE IN PYTHON

Data Preprocessing

```
# importing the necessary library
import os
import cv2
import numpy as np

from matplotlib import pyplot as plt
from patchify import patchify
from PIL import Image
import random

import tensorflow as tf
from tensorflow import keras
import segmentation_models as sm
from tensorflow.keras.metrics import MeanIoU
```

```
root_directory = 'landcover.ai.v1/'
patch_size = 256
```

```
#divide all images into patches of 256x256x3.
img_dir=root_directory+"images/"
for path, subdirs, files in os.walk(img_dir):
    #print(path)
    dirname = path.split(os.path.sep)[-1]
    #print(dirname)
    images = os.listdir(path)  #List of all image names in this subdirectory
    #print(images)
    for i, image_name in enumerate(images):
        if image_name.endswith(".tif"):
            #print(image_name)
            image = cv2.imread(path+"/"+image_name, 1)  #Read each image as BGR
            SIZE_X = (image.shape[1]//patch_size)*patch_size #Nearest size divisible by our patch size
            SIZE_Y = (image.shape[0]//patch_size)*patch_size #Nearest size divisible by our patch size
            image = Image.fromarray(image)
            image = image.crop((0 ,0, SIZE_X, SIZE_Y))  #Crop from top left corner
            #image = image.resize((SIZE_X, SIZE_Y))  #Try not to resize for semantic segmentation
            image = np.array(image)

            #Extract patches from each image
            print("Now patchifying image:", path+"/"+image_name)
            patches_img = patchify(image, (256, 256, 3), step=256)  #Step=256 for 256 patches means no overlap

            for i in range(patches_img.shape[0]):
                for j in range(patches_img.shape[1]):

                    single_patch_img = patches_img[i,j,:,:]
                    #single_patch_img = (single_patch_img.astype('float32')) / 255. #We will preprocess using one of the backbone
                    single_patch_img = single_patch_img[0] #Drop the extra unecessary dimension that patchify adds.

                    cv2.imwrite(root_directory+"256_patches/images/"+
                                image_name+"patch_"+str(i)+str(j)+".tif", single_patch_img)
                    #image_dataset.append(single_patch_img)
```

```
mask_dir=root_directory+"masks/"
for path, subdirs, files in os.walk(mask_dir):
    #print(path)
    dirname = path.split(os.path.sep)[-1]

    masks = os.listdir(path)  #List of all image names in this subdirectory
    for i, mask_name in enumerate(masks):
        if mask_name.endswith(".tif"):
            mask = cv2.imread(path+"/"+mask_name, 0)  #Read each image as Grey (or color but remember to map each color to an int
            SIZE_X = (mask.shape[1]//patch_size)*patch_size #Nearest size divisible by our patch size
            SIZE_Y = (mask.shape[0]//patch_size)*patch_size #Nearest size divisible by our patch size
            mask = Image.fromarray(mask)
            mask = mask.crop((0 ,0, SIZE_X, SIZE_Y))  #Crop from top left corner
            #mask = mask.resize((SIZE_X, SIZE_Y))  #Try not to resize for semantic segmentation
            mask = np.array(mask)

            #Extract patches from each image
            print("Now patchifying mask:", path+"/"+mask_name)
            patches_mask = patchify(mask, (256, 256), step=256)  #Step=256 for 256 patches means no overlap

            for i in range(patches_mask.shape[0]):
                for j in range(patches_mask.shape[1]):

                    single_patch_mask = patches_mask[i,j,:,:]
                    #single_patch_img = (single_patch_img.astype('float32')) / 255. #No need to scale masks, but you can do it if
                    #single_patch_mask = single_patch_mask[0] #Drop the extra unecessary dimension that patchify adds.
                    cv2.imwrite(root_directory+"256_patches/masks/"+
                                mask_name+"patch_"+str(i)+str(j)+".tif", single_patch_mask)
```

```
train_img_dir = "landcover.ai.v1/256_patches/images/"
train_mask_dir = "landcover.ai.v1/256_patches/masks/"
```

```
img_list = os.listdir(train_img_dir)
msk_list = os.listdir(train_mask_dir)
#Now, let us copy images and masks with real information to a new folder.
# real information = if mask has decent amount of labels other than 0.

useless=0  #Useless image counter
for img in range(len(img_list)):   #Using t1_list as all lists are of same size
    img_name=img_list[img]
    mask_name = msk_list[img]
    print("Now preparing image and masks number: ", img)

    temp_image=cv2.imread(train_img_dir+img_list[img], 1)

    temp_mask=cv2.imread(train_mask_dir+msk_list[img], 0)
    #temp_mask=temp_mask.astype(np.uint8)

    val, counts = np.unique(temp_mask, return_counts=True)

    if (1 - (counts[0]/counts.sum())) > 0.05:  #At least 5% useful area with labels that are not 0
        print("Save Me")
        cv2.imwrite('landcover.ai.v1/256_patches/images_with_useful_info/images/'+img_name, temp_image)
        cv2.imwrite('landcover.ai.v1/256_patches/images_with_useful_info/masks/'+mask_name, temp_mask)

    else:
        print("I am useless")
        useless +=1

print("Total useful images are: ", len(img_list)-useless)  #21924
print("Total useless images are: ", useless) #19722
```

```
import splitfolders  # or import split_folders

input_folder = 'landcover.ai.v1/256_patches/images_with_useful_info/'
output_folder = 'landcover.ai.v1/data_for_training_and_testing/'
# Split with a ratio.
# To only split into training and validation set, set a tuple to `ratio`, i.e, `(.8, .2)`.
splitfolders.ratio(input_folder, output=output_folder, seed=42, ratio=(.75, .25), group_prefix=None)
```

```
"""
Your current directory structure:
Data/
    train/
        images/
            img1, img2, ...
        masks/
            msk1, msk2, ....
    val/
        images/
            img1, img2, ...
        masks/
            msk1, msk2, ....

Copy the folders around to the following structure...


Data/
    train_images/
                train/
                    img1, img2, img3, ......

    train_masks/
                train/
                    msk1, msk, msk3, ......

    val_images/
                val/
                    img1, img2, img3, ......

    val_masks/
                val/
                    msk1, msk, msk3, ......


"""
```

Model Training

```
import os
import cv2
import numpy as np
from matplotlib import pyplot as plt
import segmentation_models as sm
from tensorflow.keras.metrics import MeanIoU
import random
```

```
#Getting an dunderstanding few random images
train_img_dir = "landcover.ai.v1/data_for_keras_aug/train_images/train/" #images
train_mask_dir = "landcover.ai.v1/data_for_keras_aug/train_masks/train/" #mask respective

img_list = os.listdir(train_img_dir)
msk_list = os.listdir(train_mask_dir)

num_images = len(os.listdir(train_img_dir))


img_num = random.randint(0, num_images-1) #taking the random images to show the respective masks

img_for_plot = cv2.imread(train_img_dir+img_list[img_num], 1)
img_for_plot = cv2.cvtColor(img_for_plot, cv2.COLOR_BGR2RGB)

mask_for_plot =cv2.imread(train_mask_dir+msk_list[img_num], 0)
#repesenting the images and respective mask side by side
plt.figure(figsize=(12, 8))
plt.subplot(121)
plt.imshow(img_for_plot)
plt.title('Image')
plt.subplot(122)
plt.imshow(mask_for_plot, cmap='gray')
plt.title('Mask')
plt.show()
```

```
seed=24
batch_size= 16
n_classes=4

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
from keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder #check the labelEncoder

#Use this to preprocess input for transfer learning
BACKBONE = 'resnet34'
preprocess_input = sm.get_preprocessing(BACKBONE)

#Define a function to perform additional preprocessing after datagen.
#For example, scale images, convert masks to categorical, etc.
def preprocess_data(img, mask, num_class):
    #Scale images
    img = scaler.fit_transform(img.reshape(-1, img.shape[-1])).reshape(img.shape)
    img = preprocess_input(img)  #Preprocess based on the pretrained backbone...
    mask = to_categorical(mask, num_classes = num_class)
    return (img,mask)
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
def trainGenerator(train_img_path, train_mask_path, num_class):

    img_data_gen_args = dict(horizontal_flip=True,
                      vertical_flip=True,
                      fill_mode='reflect')

    image_datagen = ImageDataGenerator(**img_data_gen_args)
    mask_datagen = ImageDataGenerator(**img_data_gen_args)

    image_generator = image_datagen.flow_from_directory(
        train_img_path,
        class_mode = None, #no classification type code
        batch_size = batch_size,
        seed = seed)

    mask_generator = mask_datagen.flow_from_directory(
        train_mask_path,
        class_mode = None,
        color_mode = 'grayscale',
        batch_size = batch_size,
        seed = seed)

    train_generator = zip(image_generator, mask_generator)
    print(train_generator)
    for (img, mask) in train_generator:
        img, mask = preprocess_data(img, mask, num_class) #############
        yield (img, mask)


train_img_path = "landcover.ai.v1/data_for_keras_aug/train_images/"
train_mask_path = "landcover.ai.v1/data_for_keras_aug/train_masks/"
train_img_gen = trainGenerator(train_img_path, train_mask_path, num_class=5) #to remove the index4 out of bound use 5 instead of

val_img_path = "landcover.ai.v1/data_for_keras_aug/val_images/"
val_mask_path = "landcover.ai.v1/data_for_keras_aug/val_masks/"
val_img_gen = trainGenerator(val_img_path, val_mask_path, num_class=5) #to remove the index4 out of bound use 5 instead of 4
```

```
x, y = train_img_gen.__next__()

for i in range(0,3):
    image = x[i]
    mask = np.argmax(y[i], axis=2)
    plt.subplot(1,2,1)
    plt.imshow(image)
    plt.subplot(1,2,2)
    plt.imshow(mask, cmap='gray')
    plt.show()
```

```
x_val, y_val = val_img_gen.__next__()

for i in range(0,3):
    image = x_val[i]
    mask = np.argmax(y_val[i], axis=2)
    plt.subplot(1,2,1)
    plt.imshow(image)
    plt.subplot(1,2,2)
    plt.imshow(mask, cmap='gray')
    plt.show()
```

```
#Define the model metrcis and load model.

num_train_imgs = len(os.listdir('landcover.ai.v1/data_for_keras_aug/train_images/train/'))
num_val_images = len(os.listdir('landcover.ai.v1/data_for_keras_aug/val_images/val/'))
steps_per_epoch = num_train_imgs//batch_size
val_steps_per_epoch = num_val_images//batch_size


IMG_HEIGHT = x.shape[1]
IMG_WIDTH  = x.shape[2]
IMG_CHANNELS = x.shape[3]

n_classes=5 #as we had given the size of the
```

```
model = sm.Unet(BACKBONE, encoder_weights='imagenet',
                input_shape=(IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS),
                classes=n_classes, activation='softmax')
model.compile('Adam', loss=sm.losses.categorical_focal_jaccard_loss, metrics=[sm.metrics.iou_score])
#iou - intersection over union score, you can also use the
#Other losses to try: categorical_focal_dice_loss, cce_jaccard_loss, cce_dice_loss, categorical_focal_loss

#model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['metrics']) #["accuracy"]
print(model.summary())
print(model.input_shape) #(256,256,3) as per the expectation
```

```
history=model.fit(train_img_gen,
                  steps_per_epoch=steps_per_epoch,
                  epochs=2,
                  verbose=1,
                  validation_data=val_img_gen,
                  validation_steps=val_steps_per_epoch)
```

```
model.save('landcover_2_epochs_RESNET_backbone_batch16.hdf5')
```

```
#plot the training and validation IoU and loss at each epoch
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'y', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

acc = history.history['iou_score']
val_acc = history.history['val_iou_score']

plt.plot(epochs, acc, 'y', label='Training IoU')
plt.plot(epochs, val_acc, 'r', label='Validation IoU')
plt.title('Training and validation IoU')
plt.xlabel('Epochs')
plt.ylabel('IoU')
plt.legend()
plt.show()
```

```
from keras.models import load_model
model = load_model("landcover_2_epochs_RESNET_backbone_batch16.hdf5", compile=False)

#batch_size=32 #Check IoU for a batch of images

#Test generator using validation data.

test_image_batch, test_mask_batch = val_img_gen.__next__()

#Convert categorical to integer for visualization and IoU calculation
test_mask_batch_argmax = np.argmax(test_mask_batch, axis=3)
test_pred_batch = model.predict(test_image_batch)
test_pred_batch_argmax = np.argmax(test_pred_batch, axis=3)

n_classes = 5
IOU_keras = MeanIoU(num_classes=n_classes)
IOU_keras.update_state(test_pred_batch_argmax, test_mask_batch_argmax)
print("Mean IoU =", IOU_keras.result().numpy())
```

```
#View a few images, masks and corresponding predictions.
img_num = random.randint(0, test_image_batch.shape[0]-1)

plt.figure(figsize=(12, 8))
plt.subplot(231)
plt.title('Testing Image')
plt.imshow(test_image_batch[img_num])
plt.subplot(232)
plt.title('Testing Label')
plt.imshow(test_mask_batch_argmax[img_num])
plt.subplot(233)
plt.title('Prediction on test image')
plt.imshow(test_pred_batch_argmax[img_num])
plt.show()
```

Model Prediction

```
import cv2
import numpy as np

from matplotlib import pyplot as plt
import segmentation_models as sm

from keras.models import load_model #loading the model
from smooth_tiled_predictions import predict_img_with_smooth_windowing #this is use to smothing and blending of segmented images
from sklearn.preprocessing import MinMaxScaler #scaliing and preprocessing of input image
```

```
scaler = MinMaxScaler() #declaring the object for scaling purpose

BACKBONE = 'resnet34' #declaration of base network of resnet34
preprocess_input = sm.get_preprocessing(BACKBONE)
```

```
#Reading the image
img = cv2.imread(r"C:\Python310\tf\Scripts\Project_main\landcover.ai.v1\Result\image_s1.tif")
#scaling the input image and reshapeing it(resizing it)
input_img = scaler.fit_transform(img.reshape(-1, img.shape[-1])).reshape(img.shape)
input_img = preprocess_input(input_img)
```

```
model = load_model("landcover_2_epochs_RESNET_backbone_batch16.hdf5", compile=False) #loading the model

patch_size =256 # size of patches
n_classes = 5 # Number of classes
```

```
predictions_smooth = predict_img_with_smooth_windowing(
    input_img,
    window_size=patch_size,
    subdivisions=2,  # Minimal amount of overlap for windowing. Must be an even number.
    nb_classes=n_classes,
    pred_func=(lambda img_batch_subdiv: model.predict((img_batch_subdiv))
    )
)

final_prediction = np.argmax(predictions_smooth, axis=2)
```

```
plt.imsave(r"C:\Python310\tf\Scripts\Project_main\landcover.ai.v1\Result\R2.tiff",final_prediction)
```

```
#This block of code is to represent the originalimages, original mask,and predicted segmented images side by side to compare it
plt.figure(figsize=(12, 12))
plt.subplot(221)
plt.title('Testing Image')
plt.imshow(img)
plt.subplot(222)
plt.title('Testing Label')
plt.imshow(original_mask)
plt.subplot(223)
plt.title('Prediction with smooth blending')
plt.imshow(final_prediction)
plt.show()
```

Post-Processing of the predicted dataset

```
import rasterio
from rasterio.warp import calculate_default_transform, reproject, Resampling
from osgeo import gdal,ogr,osr
import os
import geopandas as gpd

import rasterio
from rasterio.features import shapes
from shapely.geometry import shape
import fiona
import plotly.express as px
import pyproj
```

```
#open source raster #original image input ot the model
inras_path=r"C:\Python310\tf\Scripts\Project_main\landcover.ai.v1\Result\Try_1\image_s1.tif"
inras = gdal.Open(inras_path)

#raster input which is getting form segmentation modeling( result out of model)
outras_path=r"C:\Python310\tf\Scripts\Project_main\landcover.ai.v1\Result\Try_1\R2.tiff"
outras=gdal.Open(outras_path, gdal.GA_Update)

#This is the path to the out shapefile/polygon files
output_path = r"C:\Python310\tf\Scripts\Project_main\landcover.ai.v1\Result\Try_1\op.shp"
```

```
def Boundbox(dataset):
    #print('x')
    # Get the image's geotransform
    geotransform = dataset.GetGeoTransform()

    # Get the dimensions of the image
    width = dataset.RasterXSize
    height = dataset.RasterYSize

    # Calculate the coordinates of the corners
    top_left_x = geotransform[0]
    top_left_y = geotransform[3]
    top_right_x = geotransform[0] + width * geotransform[1]
    top_right_y = geotransform[3]
    bottom_left_x = geotransform[0]
    bottom_left_y = geotransform[3] + height * geotransform[5]
    bottom_right_x = geotransform[0] + width * geotransform[1]
    bottom_right_y = geotransform[3] + height * geotransform[5]

    return(top_left_x, top_left_y,top_right_x, top_right_y, bottom_left_x, bottom_left_y, bottom_right_x, bottom_right_y)

in_bb= Boundbox(inras) #calling the function
out_bb= Boundbox(outras) #getting the corrdenated value

#separating the value
a,b,c,d,e,f,g,h=in_bb
i,j,k,l,m,n,o,p=out_bb

#Getting the value to the GCP's (Ground Control Points) one image respect to another images
gcps = [gdal.GCP(a,b,0,i,j),
        gdal.GCP(c,d,0,k,l),
        gdal.GCP(e,f,0,m,n),
        gdal.GCP(g,h,0,o,p)]
```

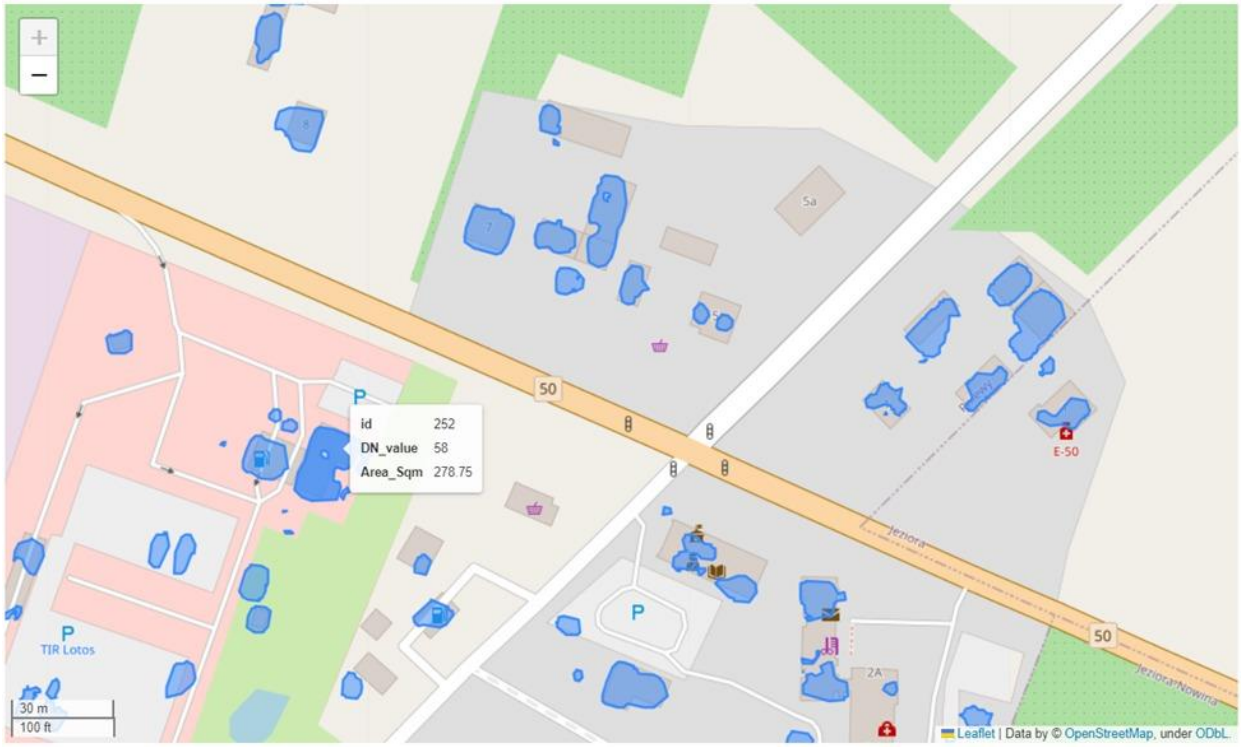
```
# Set spatial reference: Georeferencing
sr = osr.SpatialReference()
#sr.ImportFromEPSG(32634) #My projection system
sr.ImportFromWkt(inras.GetProjection())

outras.SetGCPs(gcps, sr.ExportToWkt()) #setting the projection and GCP's(image to image reference)
outras= None #closing the file
```

```
# Open the raster dataset
#raster_path = r"C:\Python310\tf\Scripts\Project_main\landcover.ai.v1\Result\R2.tiff"
raster_ds = rasterio.open(outras_path)

#Getting the projection out of raster
raster_is= rasterio.open(inras_path)
transform = raster_is.transform
crs = raster_is.crs
#print(crs)

# Read the raster data
raster_data = raster_ds.read(1) #reading the first band from the 3 band images

# Convert raster to polygons
polygons = list(shapes(raster_data,transform=transform))# mask=None))

# Create a new Shapefile to save the polygons
#output_path = r"C:\Python310\tf\Scripts\Project_main\landcover.ai.v1\Result\Try_1\op.shp"
schema = {
    'geometry': 'Polygon',
    'properties': {'id': 'int','DN_value':'int'},
}
with fiona.open(output_path, 'w', 'ESRI Shapefile', schema, crs=crs) as output_shp:
    for idx, (geom, val) in enumerate(polygons):
        output_shp.write({
            'geometry':shape(geom).__geo_interface__, #geometry.mapping(shape(geom)),
            'properties': {'id': idx + 1,'DN_value':val},
        })

# Clean up
raster_ds.close() #closing raster relesing memory from the computer
```

```
layer = gpd.read_file(output_path) #reading the .shapefile
layer['Area_Sqm']=layer['geometry'].area #Calculating the area of the each polygon
```

```
print('Total_Feature_count--',layer.count()[1])
print('Number_of_Buildings--',layer[layer['DN_value']==58].count()[1])
print('Number_of_water_Body--',layer[layer['DN_value']==94].count()[1])
print('Number_of_Woods_patches--',layer[layer['DN_value']==32].count()[1])
```

```
#This expresion is use for the Building layer only if required we can have it for another feature
layer[layer['DN_value']==58].explore()
```

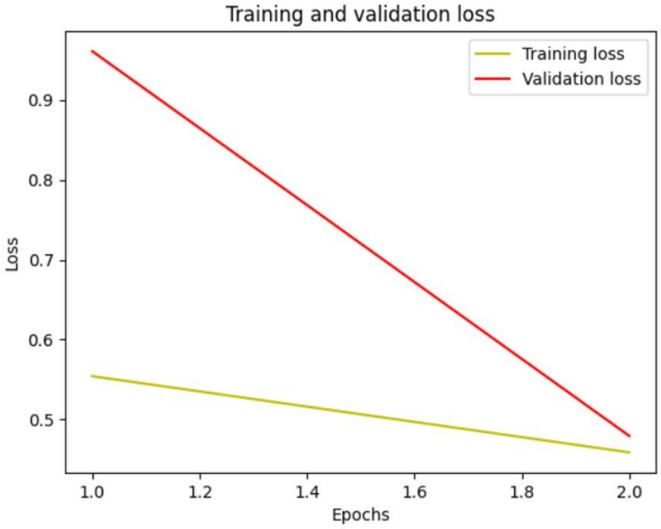


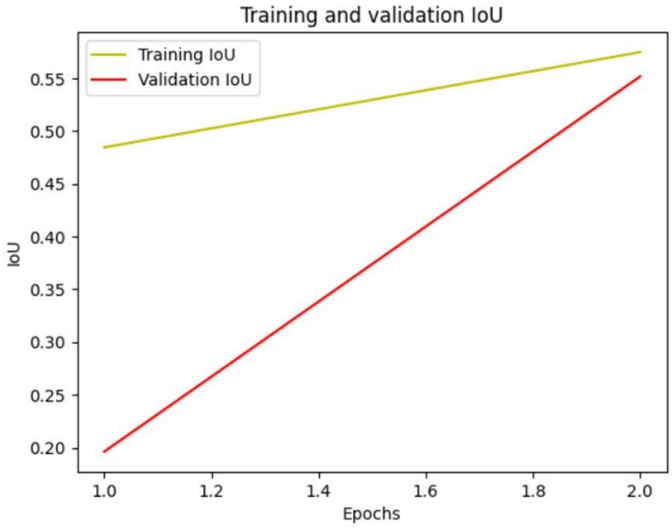

```
#Saving layer's attribute table to csv for the furture use as per the requirement
layer.to_csv(r"C:\Python310\tf\Scripts\Project_main\landcover.ai.v1\Result\Try_1\op.csv")
```

```
a=layer[layer['DN_value']==58].explore()
a.save('op.html') #saving the interactive map into htm file for portability
```

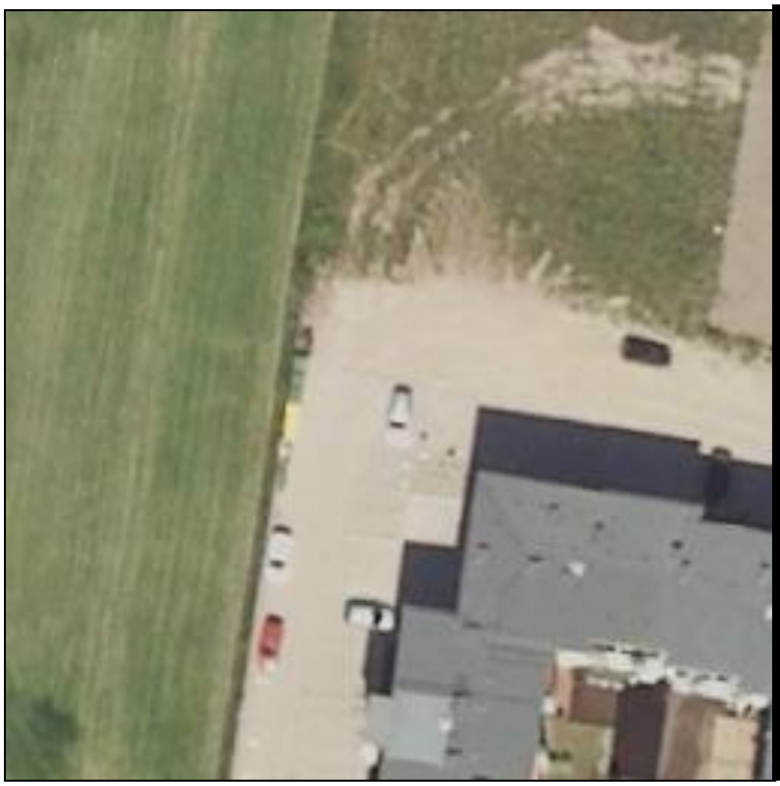
RESULTS

```
Epoch 10/15
513/513 [==============================] - 9780s 19s/step - loss: 0.3004 - iou_score: 0.7224 - val_loss: 0.3559 - val_iou_scor
e: 0.6740
Epoch 11/15
513/513 [==============================] - 9927s 19s/step - loss: 0.2973 - iou_score: 0.7250 - val_loss: 0.4074 - val_iou_scor
e: 0.6319
Epoch 12/15
513/513 [==============================] - 9883s 19s/step - loss: 0.2910 - iou_score: 0.7307 - val_loss: 0.3405 - val_iou_scor
e: 0.6888
Epoch 13/15
513/513 [==============================] - 9836s 19s/step - loss: 0.2881 - iou_score: 0.7336 - val_loss: 0.2998 - val_iou_scor
e: 0.7212
Epoch 14/15
513/513 [==============================] - 9662s 19s/step - loss: 0.2866 - iou_score: 0.7348 - val_loss: 0.3123 - val_iou_scor
e: 0.7127
Epoch 15/15
513/513 [==============================] - 8355s 16s/step - loss: 0.2841 - iou_score: 0.7372 - val_loss: 0.3394 - val_iou_scor
e: 0.6871
```





Result samples



CONCLUSION

In conclusion, this project underscores the critical importance of advancing land cover and land use monitoring through cutting-edge technology and interdisciplinary collaboration. By addressing the scarcity of high-resolution aerial datasets with fine labelling for rural regions, we pave the way for more accurate and detailed semantic segmentation of land features. Leveraging deep learning models like U-Net and ResNet, with pre-trained weights from ImageNet, enhances the efficiency of land classification. The future scope, including geographic expansion and interdisciplinary engagement, promises to deliver a comprehensive solution that empowers sustainable natural resource management and informed decision-making. Through this project, we aim to contribute significantly to the responsible stewardship of our environment and resources.

FUTURE SCOPE

1. **Expansion of Geographic Coverage:** While the Landcover.ai dataset provides a valuable foundation; future endeavour's should focus on expanding the geographic coverage to encompass diverse regions and landscapes. This expansion would enhance the model's adaptability to different environmental conditions and improve its utility on a global scale.
2. **Fine-Grained Land Classification:** The project can evolve by delving into fine-grained land classification. This entails identifying specific land cover types with greater precision, including distinguishing between various vegetation types, urban infrastructure components, and water bodies.
3. **Multi-Temporal Analysis:** Incorporating multi-temporal analysis can capture land use changes over time. Leveraging historical satellite or aerial imagery, the system can monitor how land cover evolves seasonally and yearly, providing valuable insights for land management and climate change studies.
4. **Machine Learning Advancements:** As machine learning techniques continue to evolve, incorporating the latest advancements, architectures, and algorithms can further improve the accuracy and efficiency of land cover classification and change detection.
5. **Cross-Disciplinary Collaborations:** Collaborations with experts in ecology, urban planning, disaster management, and agriculture can help tailor the system to address specific domain needs and challenges.
6. **Policy and Decision Support:** Integrating the system into policy and decision-making processes can have a substantial societal impact. Governments and organizations can use the insights for informed land management and environmental policies.

REFERENCES & BIBLIOGRAPHY

[1] LandCover.ai: Dataset for Automatic Mapping of Buildings, Woodlands, Water and Road from Aerial Imagery

Adrian Boguszewski, Dominik Batorski, Natalia Ziemba-Jankowska, Tomasz Dziedzic, Anna Zambrzycka

[2] 10m Annual Land Use Land Cover (9-class)

https://registry.opendata.aws/io-lulc/

[3] Global land use / land cover with Sentinel 2 and deep learning

Krishna Karra; Caitlin Kontgis; Zoe Statman-Weil; Joseph C. Mazzariello; Mark Mathis; Steven P. Brumby