

# Configuration Manual

MSc Research Project  
Data Analytics

Jawed Siddique  
Student ID: x22232915

School of Computing  
National College of Ireland

Supervisor: Paul Stynes, Musfira Jilani & Mark Cudden

**National College of Ireland**  
**MSc Project Submission Sheet**  
**School of Computing**



**Student Name:** Jawed Siddique  
**Student ID:** x22232915  
**Programme:** MSc. Data Analytics **Year:** 2023 - 2024  
**Module:** MSc. Research Project  
**Lecturer:** Paul Stynes, Musfira Jilani & Mark Cudden  
**Submission Due Date:** 12/08/2024  
**Project Title:** Deep Learning Framework for Land Use and Land Cover Classification and Change Detection  
**Word Count:** N/A **Page Count:** 14

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

**Signature:** Jawed Siddique

**Date:** 11/08/2024

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST**

Attach a completed copy of this sheet to each project (including multiple copies)	<input type="checkbox"/>
<b>Attach a Moodle submission receipt of the online project submission,</b> to each project (including multiple copies).	<input type="checkbox"/>
<b>You must ensure that you retain a HARD COPY of the project,</b> both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	<input type="checkbox"/>

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

<b>Office Use Only</b>	
Signature:	
Date:	
Penalty Applied (if applicable):	

# Configuration Manual

Jawed Siddique  
Student ID: x22232915

## 1. Introduction

This document outlines the software tools and configuration used for the successful completion of this research titled '*Deep Learning Framework for Land Use and Land Cover Classification and Change Detection*'. It provides a step-by-step workflow that can be followed to replicate the study.

## 2. Computing Requirements

This section describes the hardware and software specifications and settings required for the completion of this research.

### 2.1. Hardware Requirements

- Apple MacbookPro
- Processor: M1 chip, 8-core CPU
- Memory: 8 GB RAM, 512 GB HDD
- Operating System: macOS Sonoma version 14.5

This research was completed using a macOS, however it can very well be completed on a Windows or Linux OS.

### 2.2. Software Requirements

- QGIS (version 3.28) and ArcGIS Pro (version 2.9): These are geospatial software used for data pre-processing as detailed in section 3.2.
- Jupyter Notebook: The complete Python code for this research including data preprocessing, data transformation, modelling and evaluation was developed and executed within the Jupyter Notebook environment, version 6.4.3.
- UTM: Currently, the ArcGIS software is not available for a machine with macOS. Thus, UTM was required to run a Windows virtual machine on a Mac to work with the ArcGIS software.

### 2.3. Additional Requirements

- Python Libraries: Several Python libraries including Pandas, Numpy, GeoPandas (gpd), rasterio, sci-kit learn ('sklearn'), 'matplotlib', and 'seaborn' were used throughout the research.
- PyTorch: The modelling using the ResNET50 model was implemented using PyTorch version 2.3.1

# 3. Data Collection, Data Pre-Processing, Data Transformation and Modelling Workflow

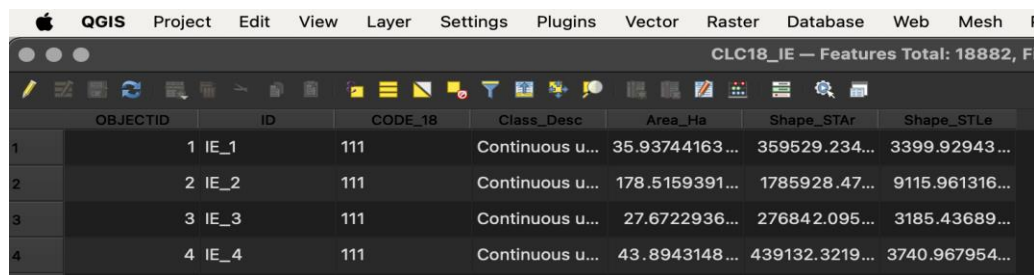
## 3.1 Data Collection

- Download the ground truth vector data and an outline map of the Greater Dublin Area from the Environmental Protection Agency<sup>[1]</sup>, Ireland and Copernicus Land Monitoring Service Urban Atlas<sup>[2]</sup> respectively.
- Download the MultiSpectral Instrument, Level-1C Sentinel 2 imagery having 5% cloud cover from the Copernicus online portal<sup>[3]</sup> for the area of interest for the years 2018 and 2024.

## 3.2. Data PreProcessing

### 3.2.1. Ground Truth Vector Data

- The downloaded ground truth vector data will have a low-level classification as shown in Figure 1. Reclassify this using the reclassification code file '*ROI GT ShapeFile - Reclassification & Color Adjustment - 2018.ipynb*' and save the new reclassified vector data. Figure 2 shows the code snippet for reclassification.



	OBJECTID	ID	CODE_18	Class_Desc	Area_Ha	Shape_STAR	Shape_STLe
1	1	IE_1	111	Continuous u...	35.93744163...	359529.234...	3399.92943...
2	2	IE_2	111	Continuous u...	178.5159391...	1785928.47...	9115.961316...
3	3	IE_3	111	Continuous u...	27.6722936...	276842.095...	3185.43689...
4	4	IE_4	111	Continuous u...	43.8943148...	439132.3219...	3740.967954...

Figure 1: Attributes Table for Ground Truth vector data with low-level classification

```
In [2]: shapefile_path_2018 = '../Data/2018/Ground Truth - 2018/Original_ROI_Shapefile - 2018/CLC18_IE.shp'

In [3]: gdf_2018 = gpd.read_file(shapefile_path_2018)

In [5]: high_level_mapping = {
        range(100, 199): 101,
        range(200, 299): 201,
        range(300, 399): 301,
        range(400, 499): 401,
        range(500, 599): 501,
    }

    def reclassify_code(code):
        for key in high_level_mapping:
            if code in key:
                return high_level_mapping[key]
        return None # If no matching high-level class is found

    gdf_2018['High_Level_Class'] = gdf_2018['CODE_18'].apply(lambda x: reclassify_code(int(x)))

In [7]: class_descriptions = {
        101: 'Artificial Surfaces',
        201: 'Agricultural Areas',
        301: 'Forest and Seminatural Areas',
        401: 'Wetlands',
        501: 'Water Bodies'
    }

    def get_class_description(high_level_class):
        return class_descriptions.get(high_level_class, 'Unknown') # Default to 'Unknown' if not found

    gdf_2018['Class_Description'] = gdf_2018['High_Level_Class'].apply(get_class_description)
```

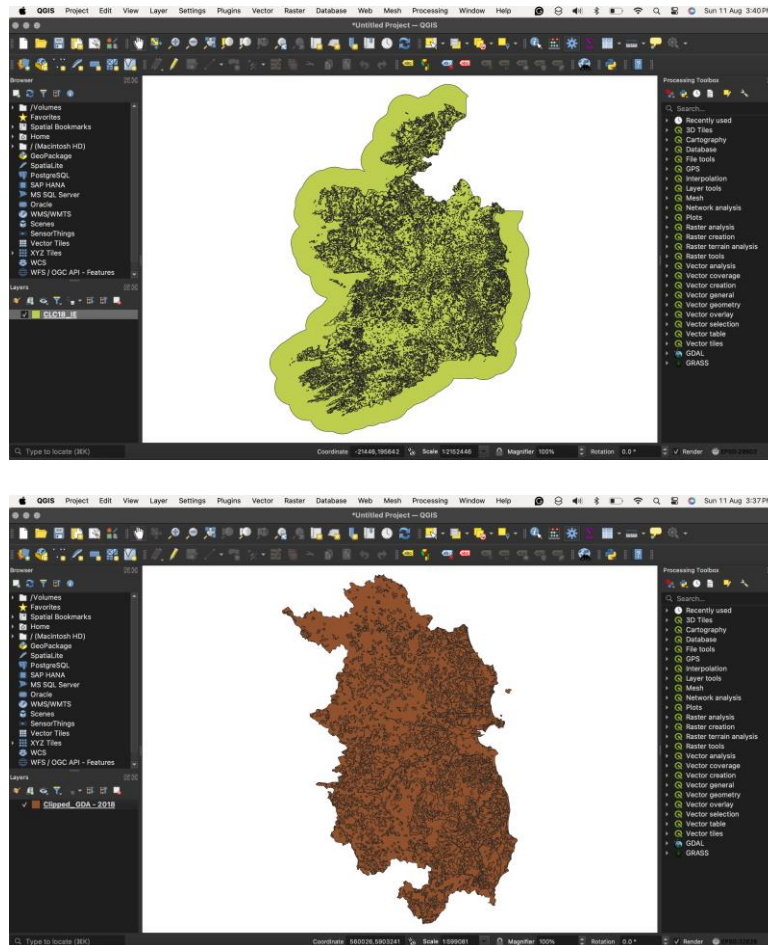
Figure 2: Script for class labels reclassification

<sup>1</sup><https://gis.epa.ie/GetData/Download>

<sup>2</sup><https://land.copernicus.eu/en/products/urban-atlas/urban-atlas-2018#download>

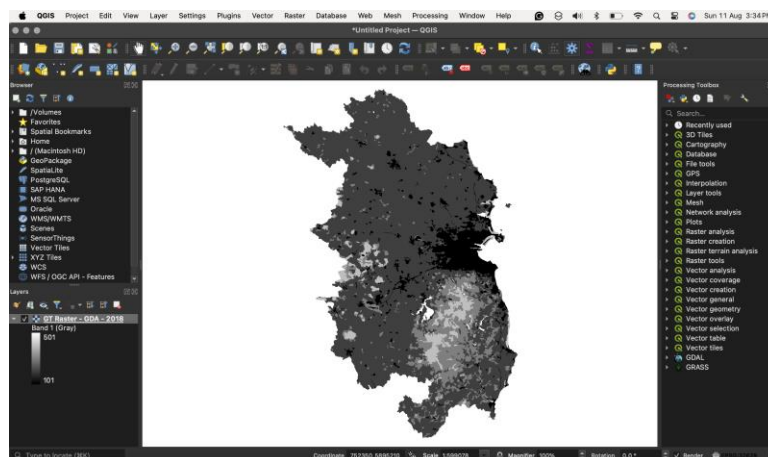
<sup>3</sup><https://dataspace.copernicus.eu/explore-data/data-collections/sentinel-data/sentinel-2>

- After reclassification, using the ArcGIS and the downloaded outline map for the Greater Dublin Area clip the ground truth vector data to the area of interest. Figures 3(a) and 3(b) show the before and after clipping shape of the vector data.



**Figures 3(a) and 3(b):** Before and after clipping shape of the vector data

- For this research, the vector data cannot be used directly. Thus rasterize the vector data by class labels and a resolution of 10m. This can be achieved using ArcGIS. Figure 4 shows the raster image generated.

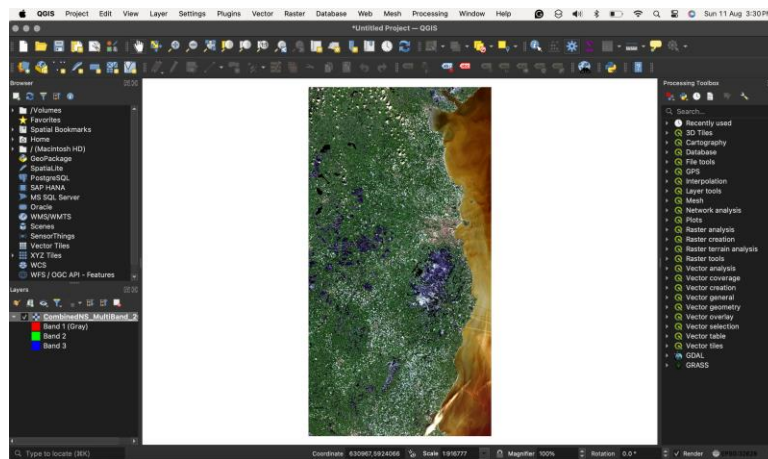


**Figure 4:** Raster image generated from vector data

- Reproject the raster file to EPSG: 32629 WGS 84 / UTM zone 29N Coordinate Reference System and save it for future use.

### 3.2.2. Sentinel-2 Data

- Use QGIS to merge the multiple spectral bands (B2, B3, B4 and B8), all at 10m resolution and then combine images representing the north and south regions of the area of interest for both 2018 and 2024. This would generate a single image representing the comprehensive multi-temporal and multi-spectral dataset. Figure 5 shows the merged multi-spectral image generated at 10m resolution for 2018. A similar image is generated for 2024.



**Figure 5:** Merged multi-spectral image generated at 10m resolution for the year 2018

- Using the outline atlas for the area of interest, clip the mosaiced images to the area of interest and reproject the images to EPSG: 32629 WGS 84 / UTM zone 29N Coordinate Reference System.

The generated ground truth from section 3.2.1 has the shape (13707, 9253) while the Sentinel Data has a shape (13708, 9265). Since the shapes of both files do not match it would not be possible to layer the files on top of each other and perform a pixel-by-pixel comparison to generate labelled satellite images. Thus, using the raster image preprocessing code '*Raster + Sentinel Image Processing.ipynb*' trim the shape of the clipped sentinel image to remove the extra row/column. Figure 6 shows the code snippet for trimming the sentinel data.

```
In [33]: with rasterio.open(sentinel2024_raster_path) as sentinel, rasterio.open(gt_raster_path) as gt:
    sentinel_data = sentinel.read()
    gt_height, gt_width = gt.shape

    sentinel_trimmed = sentinel_data[:, :gt_height, :gt_width]

    with rasterio.open(
        '../Data/2024/TrimmedNS_Sentinel2024.tif',
        'w',
        driver='GTiff',
        height=gt_height,
        width=gt_width,
        count=sentinel.count,
        dtype=sentinel_trimmed.dtype,
        crs=sentinel.crs,
        transform=sentinel.transform,
    ) as dst:
        dst.write(sentinel_trimmed)
```

```
In [34]: trimmed_sentinel_path_2018 = "../Data/2018/Sentinel 2 - 2018/TrimmedNS_Sentinel2018.tif"

with rasterio.open(trimmed_sentinel_path_2018) as dataset:
    metadata = dataset.meta
    spatial_resolution = dataset.res

    print(f'Spatial Resolution: {spatial_resolution}')
    print(f'Metadata: {metadata}')

Spatial Resolution: (10.0, 10.0)
Metadata: {'driver': 'GTiff', 'dtype': 'uint8', 'nodata': None, 'width': 9253, 'height': 13707, 'count': 4, 'crs':
CRS.from_wkt('PROJCS["WGS 84 / UTM zone 29N",GEOGCS["WGS 84",DATUM["World Geodetic System 1984",SPHEROID["WGS 84",6
378137,298.257223563]],PRIMEM["Greenwich",0],UNIT["degree",0.0174532925199433,AUTHORITY["EPSG","9122"]]],PROJECTION
["Transverse_Mercator"],PARAMETER["latitude_of_origin",0],PARAMETER["central_meridian",-9],PARAMETER["scale_facto
r",0.9996],PARAMETER["false_easting",500000],PARAMETER["false_northing",0],UNIT["metre",1,AUTHORITY["EPSG","900
1"]],AXIS["Easting",EAST],AXIS["Northing",NORTH]]'), 'transform': Affine(10.0, 0.0, 609070.0,
0.0, -10.0, 5976750.0)}

In [35]: trimmed_sentinel_path_2024 = "../Data/2024/TrimmedNS_Sentinel2024.tif"

with rasterio.open(trimmed_sentinel_path_2024) as dataset:
    metadata = dataset.meta
    spatial_resolution = dataset.res

    print(f'Spatial Resolution: {spatial_resolution}')
    print(f'Metadata: {metadata}')

Spatial Resolution: (10.0, 10.0)
Metadata: {'driver': 'GTiff', 'dtype': 'uint8', 'nodata': None, 'width': 9253, 'height': 13707, 'count': 4, 'crs':
CRS.from_wkt('PROJCS["WGS 84 / UTM zone 29N",GEOGCS["WGS 84",DATUM["World Geodetic System 1984",SPHEROID["WGS 84",6
378137,298.257223563]],PRIMEM["Greenwich",0],UNIT["degree",0.0174532925199433,AUTHORITY["EPSG","9122"]]],PROJECTION
["Transverse_Mercator"],PARAMETER["latitude_of_origin",0],PARAMETER["central_meridian",-9],PARAMETER["scale_facto
r",0.9996],PARAMETER["false_easting",500000],PARAMETER["false_northing",0],UNIT["metre",1,AUTHORITY["EPSG","900
1"]],AXIS["Easting",EAST],AXIS["Northing",NORTH]]'), 'transform': Affine(10.0, 0.0, 609070.0,
0.0, -10.0, 5976750.0)}
```

Figure 6: Code snippet for trimming the Sentinel data.

Generate tiled images of 64\*64 pixels from the ground truth raster data and trimmed Sentinel image using the code 'Convert Image to Tiles of 64 X 64.ipynb'. Figures 7(a), 7(b), and 7(c) show the code snippet for generating tiled images.

## 2. Generate tiles from GT

```
In [4]: gt_raster_path = "../Data/2018/Ground Truth - 2018/GT Raster - GDA - 2018.tif"
output_dir = "../Data/2018/Ground Truth - 2018/Tiled + Classified GT - 2018"
os.makedirs(output_dir, exist_ok=True)
```

```
In [5]: with rasterio.open(gt_raster_path) as dataset:
    metadata = dataset.meta
    spatial_resolution = dataset.res

    print(f'Spatial Resolution: {spatial_resolution}')
    print(f'Metadata: {metadata}')
```

```
In [6]: with rasterio.open(gt_raster_path) as src:
    # Get the dimensions of the image
    rows, cols = src.shape

    tile_id = 0
    for i in range(0, rows, tile_size):
        for j in range(0, cols, tile_size):
            # Define the window for the current tile
            window = Window(j, i, tile_size, tile_size)

            # Read the data from the defined window
```

## 3. Generate tiled image from Sentinel-2 2018

```
In [8]: trimmed_sentinel_path = "../Data/2018/Sentinel 2 - 2018/TrimmedNS_Sentinel2018.tif"
output_dir = "../Data/2018/Sentinel 2 - 2018/Tiled + Classified Sentinel - 2018"
os.makedirs(output_dir, exist_ok=True)
```

```
In [9]: with rasterio.open(trimmed_sentinel_path) as src:
    # Get the dimensions of the image
    rows, cols = src.shape

    tile_id = 0
    for i in range(0, rows, tile_size):
        for j in range(0, cols, tile_size):
            # Define the window for the current tile
            window = Window(j, i, tile_size, tile_size)

            # Read the data from the defined window
            transform = src.window_transform(window)
            tile_data = src.read(window=window)

            # Ensure the tile is the correct size (in case of edge tiles)
            if tile_data.shape[1] == tile_size and tile_data.shape[2] == tile_size:
                # Define the profile for the tile
                profile = src.profile
                profile.update({
                    'height': tile_size,
                    'width': tile_size,
                    'transform': transform
                })

            # Save the tile
            tile_filename = os.path.join(output_dir, f'tile_{tile_id}.tif')
            with rasterio.open(tile_filename, 'w', **profile) as dst:
                dst.write(tile_data)

            tile_id += 1
```



#### 4. Generate tiled image from Sentinel-2 2024

```
In [13]: trimmed_sentinel_path = "../Data/2024/TrimmedNS_Sentinel2024.tif"
output_dir = "../Data/2024/Tiled Sentinel - 2024"
os.makedirs(output_dir, exist_ok=True)

In [14]: with rasterio.open(trimmed_sentinel_path) as src:
# Get the dimensions of the image
rows, cols = src.shape

tile_id = 0
for i in range(0, rows, tile_size):
    for j in range(0, cols, tile_size):
        # Define the window for the current tile
        window = Window(j, i, tile_size, tile_size)

        # Read the data from the defined window
        transform = src.window_transform(window)
        tile_data = src.read(window=window)

        # Ensure the tile is the correct size (in case of edge tiles)
        if tile_data.shape[1] == tile_size and tile_data.shape[2] == tile_size:
            # Define the profile for the tile
            profile = src.profile
            profile.update({
                'height': tile_size,
                'width': tile_size,
                'transform': transform
            })

            # Save the tile
            tile_filename = os.path.join(output_dir, f'tile_{tile_id}.tif')
            with rasterio.open(tile_filename, 'w', **profile) as dst:
                dst.write(tile_data)

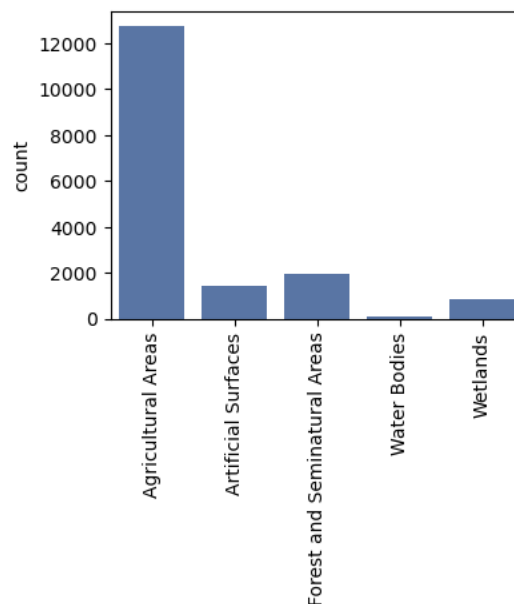
            tile_id += 1
```

Figures 8(a), 8(b), 8(c): Code snippet for generating tiled images for ground truth, 2018 and 2024 data

Lastly, generate 5 folders representing each class - Agricultural Areas, Artificial Surfaces, Forest and Seminatural Areas, Water Bodies, and Wetlands and depending on the class labels of each tile image move it to the respective class folder. Additionally, map the sentinel 2018 tiled images to the labelled raster image and generate a labelled Sentinel-2 image dataset. This step can be achieved using the code '*Processing of Tiled Images.ipynb*'. Figures 8(a), 8(b), and 8(c) show the code snippet for generating tiled images.

### 3.3. Data Transformation

The generated image set was imbalanced as there is no equal representation of each class. Figure 9 shows a histogram of the representation of each class in the labelled 2018 image dataset before data augmentation.





**Figure 9:** Histogram representing each class in the labelled 2018 image dataset before data augmentation

Thus, before modelling it is required to perform data augmentation and balance the dataset. Code '*Data Augmentation.ipynb*' is used to perform data augmentation. Figure 10 shows the code snippet to perform data augmentation and Figure 11 shows the representation of each class after data augmentation.

```
In [2]: main_folder = '../Data/2018/Sentinel 2 - 2018/Tiled + Classified + Augmented Sentinel - 2018'
        target_size = 12000

In [3]: # Define the transformations for augmentation
        transform = transforms.Compose([
            transforms.RandomHorizontalFlip(),
            transforms.RandomVerticalFlip(),
            transforms.RandomRotation(degrees=20),
            transforms.RandomResizedCrop(size=(224, 224), scale=(0.8, 1.0)),
            transforms.ColorJitter(brightness=0.1, contrast=0.1, saturation=0.1, hue=0.1),
            transforms.ToTensor()
        ])

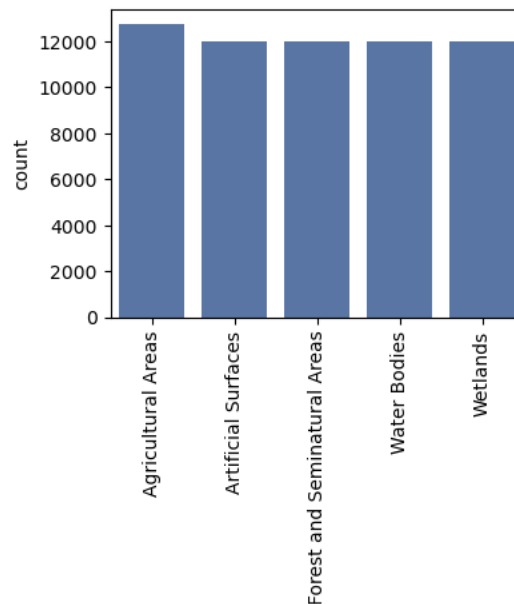
In [4]: def augment_images(folder_path, num_images_needed):
        image_files = [f for f in os.listdir(folder_path) if f.endswith('.tif')]
        num_existing_images = len(image_files)

        if num_existing_images >= num_images_needed:
            print(f"No augmentation needed for {folder_path}.")
            return

        print(f"Augmenting images in {folder_path} to reach {target_size} images.")

        while num_existing_images < num_images_needed:
            for img_file in random.sample(image_files, len(image_files)):
                img_path = os.path.join(folder_path, img_file)
                with Image.open(img_path) as img:
                    augmented_img = transform(img)
                    new_img_name = f"aug_{num_existing_images}.tif"
                    save_image(augmented_img, os.path.join(folder_path, new_img_name))
                    num_existing_images += 1
                if num_existing_images >= num_images_needed:
                    break
```

**Figure 10:** Code snippet to perform data augmentation



**Figure 11:** Histogram representing each class in the labelled 2018 image dataset after data augmentation

### 3.4. Modelling using 2018 Sentinel-2 labelled data

The modelling process is achieved using the script '*Model - Training & Validation.ipynb*'. Below is a step-by-step explanation of the code and a corresponding snippet highlighting its implementation in Python using Jupyter Notebook.

3.4.1. Import all the required libraries and set the 'SEED' value to 20 for reproducibility.

```
In [2]: import os
import random
from tqdm.notebook import tqdm

# Data manipulation and visualization
import matplotlib.pyplot as plt
from PIL import Image
import seaborn as sns
import pandas as pd
import numpy as np

# Deep Learning libraries
import torch
import torchvision
import torchsummary
from torch.utils.data import DataLoader, random_split
from torchvision import datasets, models, transforms

from sklearn.metrics import confusion_matrix, accuracy_score, cohen_kappa_score, precision_score, recall_score, f1_s

In [3]: print(torch.__version__)

2.3.1

In [4]: # Set seed for reproducibility
SEED = 20
np.random.seed(SEED)
torch.manual_seed(SEED)
torch.cuda.manual_seed(SEED)

torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False

# Check is GPU is enabled
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print("Device: {}".format(device))

# Get specific GPU model
if str(device) == "cuda:0":
    print("GPU: {}".format(torch.cuda.get_device_name(0)))

Device: cpu
```

3.4.2. This research applies transfer learning using the ResNet50 model. The ResNet50 model is pre-trained on the ImageNet dataset, thus it is crucial to standardize the input dataset to match the data distribution of the pre-trained model. Define the image normalization parameters.

```
In [5]: input_size = 224
imagenet_mean, imagenet_std = [0.485, 0.456, 0.406], [0.229, 0.224, 0.225]

train_transform = transforms.Compose([
    transforms.Resize(input_size),
    transforms.ToTensor(),
    transforms.Normalize(imagenet_mean, imagenet_std)
])

val_test_transform = transforms.Compose([
    transforms.Resize(input_size),
    transforms.ToTensor(),
    transforms.Normalize(imagenet_mean, imagenet_std)
])
```

3.4.3. Load the labelled dataset and randomly split the data into 60:20:20 train, validation and test ratios respectively.

```
In [7]: data_dir_no_aug = './Data/2018/Sentinel 2 - 2018/Tiled + Classified Sentinel - 2018'
full_dataset_no_aug = datasets.ImageFolder(data_dir_no_aug)

In [6]: data_dir = './Data/2018/Sentinel 2 - 2018/Tiled + Classified + Augmented Sentinel - 2018'
full_dataset = datasets.ImageFolder(data_dir)

In [8]: class_names = full_dataset.classes
print("Class names: {}".format(class_names))
print("Total number of classes: {}".format(len(class_names)))

Class names: ['Agricultural Areas', 'Artificial Surfaces', 'Forest and Seminatural Areas', 'Water Bodies', 'Wetlands']
Total number of classes: 5
```

```
In [9]: # Apply different transformations to the training and test sets
train_size = int(0.60 * len(full_dataset))
val_size = int(0.20 * len(full_dataset))
test_size = len(full_dataset) - train_size - val_size

train_data, val_data, test_data = random_split(full_dataset, [train_size, val_size, test_size])

print("Train/val/test sizes: {}/{}{}".format(len(train_data), len(val_data), len(test_data)))

Train/val/test sizes: 36466/12155/12157
```

3.4.4. Set the data transformation i.e. normalization for the training, validation, and test datasets. Then creates data loaders for each dataset, specifying the batch size, number of worker threads for data loading, and whether the data should be shuffled. These data loaders are used to efficiently feed the data into a model during training and evaluation.

```
In [10]: train_data.dataset.transform = train_transform
val_data.dataset.transform = val_test_transform
test_data.dataset.transform = val_test_transform

In [11]: num_workers = 2
batch_size = 16

train_loader = DataLoader(train_data, batch_size=batch_size, num_workers=num_workers, shuffle=True)
val_loader = DataLoader(val_data, batch_size=batch_size, num_workers=num_workers, shuffle=False)
test_loader = DataLoader(test_data, batch_size=batch_size, num_workers=num_workers, shuffle=False)
```

3.4.5. Load a pre-trained ResNet50 model that is optimized on the ImageNet dataset with default weights. Adapt the model for this research by replacing the fully connected layer with a new linear layer that matches the number of classes in the dataset. The model was then transferred to the appropriate device (CPU or GPU) to leverage efficient computation. Finally, a summary of the model architecture is generated, detailing the structure and parameters for an input size of 224x224 pixels.

```
In [14]: model = models.resnet50(weights=models.ResNet50_Weights.DEFAULT)
model.fc = torch.nn.Linear(model.fc.in_features, len(class_names))
model = model.to(device)
torchsummary.summary(model, (3, 224, 224))
```

Conv2d-166	[-1, 512, 7, 7]	2,359,296
BatchNorm2d-167	[-1, 512, 7, 7]	1,024
ReLU-168	[-1, 512, 7, 7]	0
Conv2d-169	[-1, 2048, 7, 7]	1,048,576
BatchNorm2d-170	[-1, 2048, 7, 7]	4,096
ReLU-171	[-1, 2048, 7, 7]	0
Bottleneck-172	[-1, 2048, 7, 7]	0
AdaptiveAvgPool2d-173	[-1, 2048, 1, 1]	0
Linear-174	[-1, 5]	10,245

```

=====
Total params: 23,518,277
Trainable params: 23,518,277
Non-trainable params: 0
-----
Input size (MB): 0.57
Forward/backward pass size (MB): 286.55
Params size (MB): 89.72
Estimated Total Size (MB): 376.84
-----
```

3.4.6. Define the key parameters of the experimental setup, including the activation function, optimizer, learning rate, loss function, and number of epochs.

```
In [15]: n_epochs = 10
lr = 1e-3

criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=lr)
```

3.4.7. Next, define three key functions to train, evaluate, and fit a deep learning model.

The 'train' function handles the training loop, where it calculates the loss, updates the model's weights through backpropagation, and tracks the training accuracy across the dataset.

```
In [16]: def train(model, dataloader, criterion, optimizer):
    model.train()
    running_loss = 0.0
    running_total_correct = 0.0

    for inputs, labels in tqdm(dataloader):
        inputs = inputs.to(device)
        labels = labels.to(device)

        optimizer.zero_grad()

        outputs = model(inputs)
        loss = criterion(outputs, labels)

        loss.backward()
        optimizer.step()

        _, preds = torch.max(outputs, 1)

        running_loss += loss.item() * inputs.size(0)
        running_total_correct += torch.sum(preds == labels)

    epoch_loss = running_loss / len(dataloader.dataset)
    epoch_accuracy = (running_total_correct / len(dataloader.dataset)) * 100
    print(f"Train Loss: {epoch_loss:.2f}; Accuracy: {epoch_accuracy:.2f}")

    return epoch_loss, epoch_accuracy
```

The 'evaluate' function assesses the model's performance on the validation set, computing loss and accuracy without altering the model's weights, and gathers predictions for further evaluation.

```
In [17]: def evaluate(model, dataloader, criterion, phase="val"):
    model.eval()
    running_loss = 0.0
    running_total_correct = 0.0

    all_labels = []
    all_preds = []

    for inputs, labels in tqdm(dataloader):
        inputs = inputs.to(device)
        labels = labels.to(device)

        with torch.no_grad():
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            _, preds = torch.max(outputs, 1)

        running_loss += loss.item() * inputs.size(0)
        running_total_correct += torch.sum(preds == labels)

        all_labels.extend(labels.cpu().numpy())
        all_preds.extend(preds.cpu().numpy())

    epoch_loss = running_loss / len(dataloader.dataset)
    epoch_accuracy = (running_total_correct / len(dataloader.dataset)) * 100
    print(f"{phase.title()} Loss: {epoch_loss:.2f}; Accuracy: {epoch_accuracy:.2f}")

    return epoch_loss, epoch_accuracy, all_labels, all_preds
```

The fit function manages the training process across multiple epochs and saves the model's best-performing weights and metrics for future use.

```
In [18]: def fit(model, train_loader, val_loader, n_epochs, lr, criterion, optimizer):
    best_loss = np.inf
    best_model_wts = None

    train_losses, val_losses = [], []
    train_accuracies, val_accuracies = [], []

    for epoch in range(n_epochs):
        print(f"Epoch {epoch+1}/{n_epochs}")

        train_loss, train_accuracy = train(model, train_loader, criterion, optimizer)
        val_loss, val_accuracy, _, _ = evaluate(model, val_loader, criterion)

        train_losses.append(train_loss)
        val_losses.append(val_loss)
        train_accuracies.append(train_accuracy)
        val_accuracies.append(val_accuracy)

        if val_loss < best_loss:
            best_loss = val_loss
            best_model_wts = model.state_dict()






    # Save best model weights
    if best_model_wts is not None:
        torch.save(best_model_wts, './Model + Metrics - pickled/best_fit_model111111.pth')

    # Save metrics
    np.savez('./Model + Metrics - pickled/best_fit_model_metrics111111.npz', train_losses=train_losses, val_losses=val_losses,
            train_accuracies=train_accuracies, val_accuracies=val_accuracies)

    return best_model_wts, train_losses, val_losses, train_accuracies, val_accuracies
```

3.3.8. Lastly, run the fit function, which trains the model over multiple epochs using the provided training and validation data loaders. It returns the best model weights (best\_model\_wts) based on validation loss, along with the lists of training and validation losses (train\_losses, val\_losses), and accuracies (train\_accuracies, val\_accuracies).

```
In [17]: best_model_wts, train_losses, val_losses, train_accuracies, val_accuracies = fit(model, train_loader, val_loader, n_

Epoch 1/10
100%  1056/1056 [1:17:16<00:00, 4.42s/it]
Train Loss: 0.63; Accuracy: 78.12
100%  59/59 [03:36<00:00, 2.54s/it]
Val Loss: 0.46; Accuracy: 85.61
Epoch 2/10
100%  1056/1056 [1:20:06<00:00, 3.09s/it]
Train Loss: 0.39; Accuracy: 86.15
100%  59/59 [02:24<00:00, 1.68s/it]
Val Loss: 0.31; Accuracy: 87.21
Epoch 3/10
100%  1056/1056 [1:12:44<00:00, 3.03s/it]
```

## 4. Model Testing and Evaluation Workflow

### 4.1. Testing and Evaluation of 2018 Data

Test the model on the 2018 test dataset and check the model performance using the script '*Model Testing - 2018.ipynb*'.

```
In [16]: model.load_state_dict(torch.load('./Model + Metrics - pickled/best_fit_model.pth'))
model.eval()

(conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
(bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
(bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(relu): ReLU(inplace=True)
)
(2): Bottleneck(
  (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
)
)
(avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
(fc): Linear(in_features=2048, out_features=5, bias=True)
)
```

```
In [19]: test_model(model, test_loader, criterion, class_names)
```

```
0%|          | 0/760 [00:00<?, ?it/s]

Test Loss: 0.19; Accuracy: 92.48
Overall Accuracy: 92.48%
Precision: 92.41%
Recall: 92.51%
F-Score: 92.45%
Cohen Kappa Coefficient: 0.91
```

## 4.2. Testing and Evaluation of 2024 Data

4.2.1. Using script '*Model Predictions – 2024*' and the saved model '*best\_fit\_model.pth*' make class predictions on the unlabelled 2024 dataset.

4.2.2. Using code '*Manual Labeling + Performance evaluation of 10% Predicted Data.ipynb*' generate a 10% sample from the predicted data and manually label them. Save the labelling information in Excel with true labels and predicted label information. Using the same script, upload the Excel to a pandas data frame and evaluate the model's performance in predicting labels.

```
In [22]: file_path = './Data/2024/Manual Labeled True Class vs Predicted Class - Before Pseudo Testing.xlsx'
df = pd.read_excel(file_path)

df['True Class Numeric'] = df['True Class']
df['Predicted Class Numeric'] = df['Predicted Class']

class_mapping = {
    'Artificial': 'Artificial Surfaces',
    'Agriculture': 'Agricultural Areas',
    'Forest': 'Forest and Seminatural Areas',
    'Water': 'Waterbodies',
    'Wetlands': 'Wetlands'
}

class_mapping_Numeric = {
    'Artificial': 0,
    'Agriculture': 1,
    'Forest': 2,
    'Water': 3,
    'Wetlands': 4
}

### Compute confusion matrix

df['True Class'] = df['True Class'].map(class_mapping)
df['Predicted Class'] = df['Predicted Class'].map(class_mapping)

true_classes = df['True Class']
predicted_classes = df['Predicted Class']

class_labels = sorted(set(true_classes) | set(predicted_classes))
cm = confusion_matrix(true_classes, predicted_classes, labels=class_labels)

# Plot the confusion matrix with updated class labels
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_labels, yticklabels=class_labels)
plt.xlabel('Predicted Class')
plt.ylabel('True Class')
plt.title('Confusion Matrix')
plt.show()
```

4.2.3. Add the 10% sampled manually labelled data to the existing 2018 labelled data. Re-train the model using this pseudo-labelled + original labelled dataset. Repeat steps 4.2.1. and 4.2.2. to make predictions using the new model *'best\_fit\_model\_pseudotraining.pth'* and re-evaluate the prediction performance.

```
In [21]: file_path = './Data/2024/Manual Labeled True Class vs Predicted Class - After Pseudo Testing.xlsx'
df = pd.read_excel(file_path)

df['True Class Numeric'] = df['True Class']
df['Predicted Class Numeric'] = df['Predicted Class']

class_mapping = {
    'Artificial': 'Artificial Surfaces',
    'Agriculture': 'Agricultural Areas',
    'Forest': 'Forest and Seminatural Areas',
    'Water': 'Waterbodies',
    'Wetlands': 'Wetlands'
}

class_mapping_Numeric = {
    'Artificial': 0,
    'Agriculture': 1,
    'Forest': 2,
    'Water': 3,
    'Wetlands': 4
}

df['True Class'] = df['True Class'].map(class_mapping)
df['Predicted Class'] = df['Predicted Class'].map(class_mapping)

true_classes = df['True Class']
predicted_classes = df['Predicted Class']

class_labels = sorted(set(true_classes) | set(predicted_classes))
cm = confusion_matrix(true_classes, predicted_classes, labels=class_labels)

# Plot the confusion matrix with updated class labels
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_labels, yticklabels=class_labels)
plt.xlabel('Predicted Class')
plt.ylabel('True Class')
plt.title('Confusion Matrix')
plt.show()
```

### 4.3. Change Detection

Finally, calculate the total area covered by each class for 2018 and 2024. Difference the 2018 values from the 2024 values to calculate the change in area.

```
In [2]: def calculate_area_per_class(base_directory, resolution):
    """
    Calculate the total area covered by each class based on tiled images.

    Args:
        base_directory (str): The path to the base directory containing class folders.
        resolution (float): The spatial resolution of the images in meters per pixel.

    Returns:
        dict: A dictionary where keys are class names and values are areas in km².
    """
    area_per_pixel = (resolution ** 2) / 1e6 # Convert area to km² (1e6 m² in 1 km²)
    class_areas = {}

    # Iterate through each class folder
    for class_folder in os.listdir(base_directory):
        class_path = os.path.join(base_directory, class_folder)

        # Only proceed if it's a directory
        if os.path.isdir(class_path):
            # Count the number of tiles (images) in the folder
            num_tiles = len([f for f in os.listdir(class_path) if f.endswith('.tif')])

            # Calculate total pixels
            total_pixels = num_tiles * 64 * 64

            # Calculate total area for the class
            total_area_km2 = total_pixels * area_per_pixel

            # Store the result
            class_areas[class_folder] = total_area_km2

    return class_areas
```



## Calculate Area -2018

```
In [3]: base_directory = "./Data/2018/Sentinel 2 - 2018/Tiled + Classified Sentinel - 2018"
resolution = 10
```

```
class_areas = calculate_area_per_class(base_directory, resolution)
```

```
for class_name, area_km2 in class_areas.items():
    print(f"Class '{class_name}' covers an area of {area_km2:.2f} km².")
```

Class 'Wetlands' covers an area of 351.85 km².  
Class 'Forest and Seminatural Areas' covers an area of 796.26 km².  
Class 'Artificial Surfaces' covers an area of 594.33 km².  
Class 'Agricultural Areas' covers an area of 5233.87 km².  
Class 'Water Bodies' covers an area of 32.77 km².

## Calculate Area -2024

```
In [4]: base_directory = "./Data/2024/Predicted Class After Pseudo Training - 2024"
resolution = 10
```

```
class_areas = calculate_area_per_class(base_directory, resolution)
```

```
for class_name, area_km2 in class_areas.items():
    print(f"Class '{class_name}' covers an area of {area_km2:.2f} km².")
```

Class 'Wetlands' covers an area of 453.84 km².  
Class 'Forest and Seminatural Areas' covers an area of 571.80 km².  
Class 'Artificial Surfaces' covers an area of 681.16 km².  
Class 'Agricultural Areas' covers an area of 5276.06 km².  
Class 'Water Bodies' covers an area of 26.21 km².