1. Environment Setup

This section outlines the tools and dependencies needed and provides step-by-step instructions for installation. (If needed refer to the readme.txt file for detailed instructions).

1.1 Install OpenJDK

OpenJDK is required to ensure compatibility with ESA SNAP software. Versions 8 or 11 are recommended. Use the provided file for installation:

- Locate the executable files in the installers folder or download from the official website.
- Run the installer and follow on-screen instructions to complete installation.
- Add environment variables.
- Verify installation by running the command:
  java -version

1.2 Install ESA SNAP Software

ESA SNAP is a vital tool for working with satellite imagery, and its *esa-snappy* plugin enables Python integration:

- Locate the executable files in the installers folder or download the latest ESA SNAP installer from the official ESA SNAP website.
- Run the installer and follow prompts to complete installation.
- Enable the esa-snappy python installation when prompted.

1.3 Install Anaconda and Create a Python Environment

Anaconda simplifies package management and creates isolated environments for your project:

- Download and install Anaconda from https://www.anaconda.com/
- Use the provided *environment.yml* file to create a compatible Python environment. Recommended Python versions for ESA SNAP are 2.7 and 3.3 to 3.10. For newer installations, Python 3.8 is preferred.
- Run the following command to create the environment:
  conda env create -f snappy.yml

2. Datasets Required

This section outlines the datasets needed, their sources, and how they integrate into the data preparation workflow.

2.1 Sentinel-1 Satellite Data

Sentinel-1 provides radar imagery, enabling analysis of surface structures and textures.

- Source: Sentinel-1 imagery can be manually downloaded from the Copernicus Open Access Hub or automatically retrieved using the *get_sentinel.py* script during preprocessing step.

2.2 Sentinel-2 Satellite Data

Sentinel-2 offers high-resolution optical imagery.

- Source: Download manually from the Copernicus Open Access Hub or automate the process using *get_sentinel.py* script during preprocessing step.

2.3 Global Human Settlement Layer (GHSL) Built-Up Fraction

GHSL data provides built-up and non-built-up land cover classifications.

- Source: Download from Copernicus Human Settlement (https://human-settlement.emergency.copernicus.eu/).
- Purpose: Creating binary land cover raster masks.

2.4 Global Human Settlement Layer (GHSL) Population Grid

Population data facilitates demographic analysis of urban areas.

- Source: Download from Copernicus Human Settlement (https://human-settlement.emergency.copernicus.eu/).
- Purpose: Computing zonal statistics, such as population counts within DUAs.

2.5 Deprived Urban Areas (DUA) Polygons

DUA polygons outline specific urban zones of interest.

- Source: Obtain predefined location polygons from urban research datasets or local administrative sources.
- Purpose: Generating a third-class raster for land cover classification.

2.6 City Administrative Boundaries

City boundary data defines the Area of Interest (AOI) for the mapping process.

2.7 Building Footprint Dataset

Building footprint datasets provide detailed information on built-up structures.

- Source: Choose between Google Open Buildings and Microsoft Building Footprints. Data can be manually downloaded or automated with the provided *create_density.py* script.

- Purpose: Computing morphometric metrics such as built-up density, orientation, and size.

3. Data Preprocessing

The following subsections provide step-by-step instructions for downloading, aligning, sampling and structuring the data.

3.1 Download and Preprocess Sentinel Data

The script *get_sentinel.py* automates the download and preprocessing of Sentinel-1 and Sentinel-2 data:

- Required inputs:
  - City AOI vector file to define the bounding box.
- Process:
  - Run the script to fetch Sentinel imagery within the AOI bounds and specified period.
  - Apply preprocessing chains to prepare the images for downstream task.

3.2 Create Reference Raster Mask

The script *create_ref.py* generates a reference raster mask with three land cover classes:

- Required inputs:
  - GHSL Built-Up Fraction Layer.
  - Deprived Urban Areas (DUA) polygons.
  - City AOI polygon.
- Process:
  - Use the script to combine input layers into a raster mask.
  - The final output is a raster with three land cover classes.

3.3 Generate Built-Up Density Raster

The script *create_density.py* computes built-up density using the building footprint dataset:

- Required inputs:
  - Building footprint dataset (manually provided or automatically fetched).
- Process:
  - The script calculates density metrics and outputs a single band raster.

3.4 Align Raster Pixels

The script *align_rasters.py* ensures that raster dimensions and pixel alignments match:

- Required inputs:
  - Reference raster (master).
  - Sentinel-1, Sentinel-2, and built-up density raster (to be aligned).
- Process:
  - Align each raster (one at a time) with the reference raster.

3.5 Generate Stratified Sampling Grid

The script *grid_sampling.py* creates an equal-sized grid over the study area with stratified random sampling:

- Required inputs:
  - Reference raster.
  - Grid size in pixels (e.g., 128x128 or 256x256).
- Process:
  - Assign 70% for training, 15% for validation, and 15% for testing.
  - Output: GeoJSON vector file.

3.6 Extract Train, Validation, and Test Patches

The script *extract_patch.py* extracts data patches for training, validation, and testing:

- Required inputs:
  - Grid polygons (from the previous step).
  - Raster data (Sentinel-1, Sentinel-2, density raster, reference mask).
- Process:
  - Apply the grid file to each raster pair (e.g., Sentinel-2 raster with the grid).
  - Extract patches and organize them by dataset type (train, valid, test).

Output of Preprocessing

At the end of preprocessing, you will have aligned training, validation, and testing patches for:

- Sentinel-1
- Sentinel-2
- Built-up density
- Reference masks

These outputs are ready for deep learning model training.