Capstone Project Phase B

Urban Microclimate Analysis through Machine Learning: A SegFormer-Based Study

24-2-R-19

Students:

Erik Pinhasov – erik.pinhasov@e.braude.ac.il

Nave Cohen – nave.cohen@e.braude.ac.il

Lecturer:

Zakharia Frenkel

Project GitHub Repository

Table of contents

Abstract. Urban microclimates, influenced by the built environment, are becoming more pronounced with the rapid expansion of cities, intensifying the challenges of an increasingly extreme climate. Traditional climate models generally lack the resolution to capture the complexities of urban areas, such as the Urban Heat Island effect. This study utilizes high-resolution aerial and satellite imagery to fine-tune the SegFormer-B4 model, a transformer-based segmentation framework, for analyzing urban microclimate conditions. Fine-tuning the model allowed us to accurately classify key urban features like buildings, roads, vegetation, and water bodies. Hyperparameter tuning was performed to refine learning rates, batch sizes, and loss functions, focusing on maximizing model accuracy. Additionally, data augmentation techniques were applied to enhance the model's generalization across urban landscapes. The fine-tuned model has been integrated into a user-friendly graphical interface that generates segmentation maps and performs analyses by comparing multi-year imagery with historical climate data. This functionality enables users to assess changes in urban features and climate features over time, providing insights into the relationship between urban growth and climate trends, such as rising temperatures and reduced green space. These tools support stakeholders and urban planners make informed decisions for sustainable urban development.

Keywords: Urban Microclimate ・ Aerial Image Segmentation ・ SegFormer ・ Urban Heat Island (UHI) ・ Remote Sensing Analysis

1 General Description

- Project Goal:

  The project intends to study the relationship between microclimate change and urban development through high-resolution segmentation maps, which identify urban objects with historical temperature records. The focal issue is the identification of patterns, especially the Urban Heat Island (UHI) phenomenon, and correlations to patterns of urbanization, including changes in buildings, roads, parks, and bodies of water.
  This goal is achieved by developing a system that combines optimized SegFormer-B4 models with an easily understandable interface, thus providing tangible insights for stakeholders such as urban planners and environmental scientists.

- Project Implementation:

  Key steps of implementation included:

  1. Data Collection and Preparation: Gathered aerial and satellite imagery data along with historical climate data. Preprocessed the data using several techniques for consistency.
  2. Model Training: SegFormer-B4 is trained with high-resolution datasets, which helps increase the accuracy of segmenting both urban and rural features.

3. System Development: Designed GUI for data visualization and analysis, integrating model outputs with climate data to show trends.
4. Validation and Testing: Carried out thorough testing to validate model performance and the accuracy of climate correlation analyses.

Our project consists of those three main products:

1. Fine-Tuned Segmentation Model: Utilized the SegFormer-B4 segmentation model, fine-tuned with high-resolution aerial and satellite imagery datasets, to classify urban and rural features, such as buildings, roads, vegetation, and water bodies.
2. GUI Application with Analysis Tools: Created a user-friendly interface for analyzing remote-sensing images. The tool allows uploading aerial images and generating segmentation maps for them. The analysis compares multi-year segmentation results with historical climate data, highlighting correlations through visualizations.
3. Analysis Results Sample Dataset: Created a focused dataset with segmentation maps and historical climate data for some selected areas with UHI potential. Established a relationship between increased urban features and deteriorated climate conditions. This dataset illustrates the motive of the study and further shows how it can be scaled up in the future.

- Stakeholders:
  1. Urban Planners and Environmental Researchers: Use the system for strategy development to mitigate UHI and increase climate resilience.
  2. Policy Makers and Municipal Authorities: Utilize the analysis for informed decision-making in urban development and environmental policies.
  3. Academic Researchers and Students: Use the project for educational purposes and further research in the field of urban climate studies.
  4. Data Scientists: Apply the methodologies to related applications in environmental data analysis and urban planning. In addition, scaled up the fine-tuned SegFormer model for urban/rural segmentation use cases.

2 Solution Description

2.1 Algorithm Flow

- **Load Configuration**

Load the configuration file or use default to set training parameters.

- **Dataset Parameters:** Dataset source, batch size, number of workers for parallel data loading, and preprocessing options.
- **Model & Result Directories:** Create directories for models, logs, checkpoints, and results.

- **Training Hyperparameters:** Define values for learning rate, weight decay, max epochs, class weights strategy, and loss functions to optimize model performance.
- **Callback Parameters:** Define metrics to monitor, callback mode, and patience parameters.

- **Data Preparation:**

    - **Load Dataset:** Load dataset from path or Hugging Face repository.
    - **Create PyTorch Dataset**: Utilize the SegFormer image processor with a given augmentation pipeline.
    - **Preprocessing Steps**: Apply normalization, augmentation, and resizing to prepare images for model training.

- **Training Process**

    - **Model Initialization**: Load the SegFormer model with the specified backbone (e.g., B0, B4)..
    - **Optimizer and Scheduler Setup**: Use AdamW optimizer with weight decay and a Cosine Annealing Learning Rate scheduler.
    - **Class weights:** Calculate class weights based on class frequency, inverse the weights and normalize to sum 1 for handling class imbalance.
    - **Loss Function**: Compute segmentation loss using a combination of Cross Entropy and Dice Loss, with tunable α and β weights.
    - **Data Loading**: Use PyTorch DataLoader to feed pipelined augmented batches into the model.
    - **Training Loop**:
        - Forward pass through the SegFormer model.
        - Compute loss between predictions and ground truth masks.
        - Backpropagate gradients and update model parameters.
        - Track training and validation metrics, including IoU and Pixel Accuracy.
    - **Validation**: Periodically evaluate the model on the validation dataset, monitoring metrics like Validation Loss, Mean IoU and Dice Score to detect overfitting.
    - **Checkpointing and Early Stopping**: Save the best-performing model based on given metrics to monitor (e.g., Mean IoU), and stop the training if no improvement is observed for a set number of epochs.

- **Evaluate the Model**

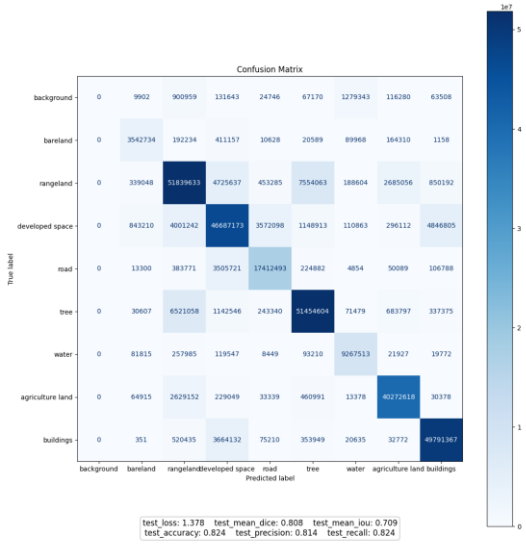
    - Evaluate the final trained model on a separate test dataset.
    - Measure performance using IoU, Dice Score, Accuracy, Precision, and Recall to assess segmentation quality.
    - Generate a confusion matrix comparing ground truth labels to model predictions.

The confusion matrix provides information about class-based performance.

- **Logs and Results**

  The model logs **training and validation loss** along with key metrics using **TensorBoard**, allowing real-time monitoring during training and post-evaluation.
  After testing, the model generates a **confusion matrix plot**, similar to the one shown below, visualizing the model's classification performance across different classes.

Confusion matrix output example:



2.2 Software Architecture

- **Main Entry Point (ucs/main.py)**

  The main.py script serves as the main entry point to code, accept path to config.yaml configuration file, and command-line arguments that override specific configuration settings.
  If no configuration file or arguments are provided, the script falls back to default settings defined within the code.
  - **config:** A path to a config.yaml file to load configurations from.
  - **evaluate:**
    - **arguments**
      - **version:** The experiment version number to evaluate.
      - **checkpoints_dir:** Directory to load the model checkpoint from.
      - **results_dir:** Directory to save test results.
  - **train:**
    - **arguments**
      - **dataset_path:** local path or Hugging Face path to load the dataset.
      - **batch_size:** Number of samples to process in each batch during training.
      - **num_workers:** Number of cpu workers for parallel data loading.
      - **do_reduce_label:** Used to reduce data set labels by one.

- **pin_memory:** Enable pinned memory for faster GPU transfer.
- **checkpoints_dir:** Directory to save model checkpoint during training.
- **logs_dir:** Directory to save tensorboard logs during training.
- **pretrained_dir:** Directory to save model as pretrained weights model.
- **model_name:** SegFormer backbone to use (e.g., b0)
- **max_epochs:** The number of epochs to train the model.
- **learning_rate:** Learning rate for AdamW optimizer.
- **weight_decay:** L2 regularization value for AdamW optimizer.
- **ignore_index:** Index to ignore during loss and metrics calculations.
- **weighting_strategy:** A strategy to normalize class weights (e.g., raw, sum, none)
- **gamma:** Gamma parameter for Focal Loss.
- **id2label:** Dictionary mapping class index to semantic names.
- **early_stop_paitance:** Define how many epochs without improvement before stop training.
- **early_stop_monitor:** The metric to monitor for early stop.
- **early_stop_mode:** Define what to consider as improvement in monitor metric (e.g., max, min).
- **save_model_monitor:** The metric to monitor for saveing best checkpoint and pretrained model.
- **save_model_mode:** Define what to consider as improvement in monitor metric (e.g., max, min).

- **Train Entry Point (ucs/model/train.py)**

  The train.py file takes care of the whole training pipeline for the SegFormer model. It manages dataset loading, model, trainer and callbacks initialization, also handles logging, and training execution with the integration of PyTorch Lightning to make it easier.

  Prepares Paths for Logging and Checkpoints

  - The script creates directories for saving logs, model checkpoints, and pre-trained weights.
  - If resuming from a previous run, it ensures the correct versioning of logs and checkpoints.

  Initializes TensorBoard Logging

  - Training metrics such as loss, accuracy, and IoU are logged to TensorBoard.
  - If training is being resumed, it continues logging from the previous session.

  Loads the Dataset and Augmentations pipeline

  - Load raw dataset from the given configuration path.
  - The raw dataset is loaded using our SegmentationDataModule, handling train, validation, and test splits.
  - Create a train/validation/test dataloaders.
  - Apply given Augmentations pipeline to train dataloader

Computes Class Weights for Loss Balancing.

- Compute class weights if weighting strategy is given for class balancing during loss calculation.
- Load precomputed class weights if available, otherwise they are calculated and saved.

Initializes the SegFormer Model

- Load Initialize our SegFormerFineTuner pytorch_lightning model, from pretrained SegFormer model using the given configuration (.e.g, b0, b1)
    - Initialize the Focal Loss function with ignore index, class_weights and gamma.
    - Initialize the score metrics, with provided ignore index.

Initializes Callbacks

- Initialize early stop callback using given configurations.
- Initialize save model callback using given configurations.

Configures the PyTorch Lightning Trainer

- The script automatically detects GPU availability and adjusts acceleration accordingly.
- Integrates the logger and callbacks

Handles Resume from Checkpoint

- Check if a resume version is given from the command-line.
- Load checkpoint with hyperparameters and model weights.

Starts the Training Process

- The training loop runs for the specified number of epochs, optimizing the model while monitoring validation performance.
- Intermediate results and best-performing models are saved for later evaluation.

- **Train Entry Point (ucs/model/evaluate.py)**

The evaluate.py script is responsible for loading a trained model from checkpoint, using a test dataset that the model never saw before, compute evaluation metrics and loss according to predictions and ground truth, retrieve all predictions and ground truth and generate confusion matrix plot with metrics score.

Load the model for the test

- If a version is provided, load our SegformerFinetuner from the saved version checkpoint.
- If no version is provided, load the last trained model.

Load test dataset

- Load the test dataset using our SegmentationDataModule, prepare the test dataloader and the image processor.

Run Model Evaluation

- Initialize pytorch lightning Trainer to start the evaluation loop.
- The model processes the test dataset, making predictions for each sample.

- Evaluation metrics such as IoU, Dice Score, Precision, and Recall are computed from predictions against ground truth.

Save Results

- Retrieve all predictions and ground truth and generate a confusion matrix.
- Save confusion matrix as plot with the evaluation matrix score.

- **Utils module (ucs/utils)**

  The utils module provides utility functions and classes that support different aspects of the project, for model training, evaluation, etc. These utilities include configuration handling, metric computation, and callbacks.

- **Data module (ucs/data)**

  The data module manages all dataset-related functionalities, such as creation of train-validation-test dataloaders, data augmentation pipeline, and creation pytorch dataset from raw data.This module allows efficient loading, transformation, and batching of data while supporting both local and remote dataset sources.

- **Core module (ucs/core)**

  The core module provides core components that define error handling and argument parsing behavior. It includes custom error definitions to improve debugging and structured argument parsing for command-line execution.

- **Model module (ucs/model)**

  The model module is responsible for defining the model architecture, training procedures, and evaluation pipeline. It provides the core functionality needed to train, fine-tune, and test the segmentation model using PyTorch Lightning.

2.3 Analysis Tool - GUI Application Structure

The Microclimate Analysis Tool GUI is designed using PyQt5 with modular controllers to manage data creation, segmentation, and climate data analysis. The architecture follows a maintainable and scalable MVC-like structure.

- **Main Application Entry Point (main.py)**

The main.py file serves as the entry point for the application. It initializes the PyQt application, loads the UI from the .ui file, and integrates the sidebar along with individual page controllers.

Main Components:

- MainApp Class: Inherits from QMainWindow, initializes the UI, loads styles from styles.json, and integrates all page controllers.

- Controllers Integration: SidebarController, CreateDataController, SegmentDataController, and AnalysisPageController are instantiated here.
- Dynamic Styles: The load_styles() function loads UI styles from a JSON file for consistent theming.

- **Sidebar Controller (sidebar_controller.py)**

Controls page-to-page transition in the application.

Main Features:

- Sidebar Toggle: Controls the expand/collapse behavior of the sidebar.
- Navigation: Maps buttons to pages, ensuring the active button is highlighted with dynamic styles.
- Page Switching: Controls page switching between "Create Data", "Segment Data", and "View Analysis" seamlessly.

- **Base Page Controller (page_controller.py)**

Abstract base class, providing shared functionalities across all pages.

Main Features:

- File Handling: Integrates FileHandler for file operations (browsing, validation).
- Image Display: Uses ImageDisplayHandler to manage how images are displayed in scroll areas.
- Input Toggling: Enabling/disabling UI components based on user actions.

- **Create Data Controller (create_data_controller.py)**

Manages the "Create Data" page where users upload images, specify coordinates, and save session data.

Main Features:

- File Upload & Metadata: Handles image uploads with year input validation, and stores metadata including geolocation and timestamps.
- Coordinate Handling: Supports both automatic locations fetching via country/city selection and manual latitude/longitude entry.
- Session Saving: Metadata and images are saved into structured folders, aiding further segmentation and analysis.

- **Segment Data Controller (segment_data_controller.py)**

Manages the "Segment Data" page, enabling users to process images through the segmentation model.

Main Features:

- Dual Input Mode: Supports both new image uploads and existing datasets for segmentation.
- Threaded Segmentation: Implements SegmentationThread to run segmentation asynchronously, ensuring UI responsiveness.
- Visualization: Displays original images side-by-side with their corresponding segmentation maps post-processing.

- **Analysis Page Controller (analysis_controller.py)**

Manages the "View Analysis" page where users can analyze climate data in correlation with segmentation outputs.

Main Features:

- Dataset Management: Populates available datasets for selection.
- Climate Data Integration: Fetches historical climate data using coordinates and image timestamps, integrating it into the analysis workflow.
- Analysis Visualization: Generates analysis based on segmentation classes percentage change through the years vs climate data. Provide results graph and correlation heatmap.

- **Segmentation Model Integration (model.py)**

The model.py file integrates the SegFormer model for semantic segmentation.

Main Features:

- Model Loading: Pretrained SegFormer model loaded with specific land cover classes.
- Segmentation Pipeline: Processes images, applies Conditional Random Fields (CRF) for post-processing, and generates segmentation maps.
- Metadata Updates: Calculates class-wise pixel frequencies and updates the corresponding metadata.json for each dataset.

Utility Files Explanation

- **file_handler.py**

This utility handles file browsing and validation. It provides two key functions:

- browse_files(): Opens a file dialog to select image files, filtering by .png, .jpg, .jpeg, and .tif formats.
- validate_file_paths(): Cleans and validates file paths entered manually, ensuring they exist before proceeding.

- **image_display.py**

This utility is responsible for dynamically managing image displays within the GUI. Key features include:

- create_image_widget(): Generates an image display widget with an optional year input field and a delete button.
- create_delete_button(): Creates a delete button that removes an image from the list.
- clear_layout(): Clears all widgets from a given layout.
- populate_scroll_area(): Populates a scroll area with image widgets, supporting a grid layout (configurable for 1 or 2 images per row).
- get_images_with_years(): Extracts filenames and associated years from the images.

- **location_handler.py**

This module manages country, city, and coordinate handling using external APIs:

- get_countries(): Fetches a list of all countries using the pycountry library.

- get_cities_by_country(): Retrieves a list of cities for a selected country from the Countries Now API.
- get_coordinates(): Uses OpenStreetMap's Nominatim API to fetch latitude and longitude based on a city-country combination.
- setup_country_combobox() / setup_city_combobox(): Populate the respective dropdowns dynamically.

- **save_handler.py**

This file manages data-saving operations, including:

- ensure_base_directory_exists(): Ensures the central directory (Microclimate Analysis Data) exists.
- create_session_directory(): Creates a structured session folder with subdirectories (images/).
- save_images(): Copies uploaded images into the session directory.
- save_metadata(): Saves metadata (metadata.json) related to the session.

- **alert_handler.py**

This utility provides alert dialogs and a loading indicator for user feedback:

- show_error(): Displays a critical error message.
- show_info(): Displays an informational message.
- LoadingDialog: A progress dialog with a spinning bar, used during long operations (e.g., segmentation and climate data fetching).

- **climate_data_handler.py**

This module fetches and updates historical climate data from the Open-Meteo API:

- fetch_climate_data(): Fetches both daily and hourly climate data for each selected year at a given latitude and longitude.
- update_metadata(): Updates metadata.json with climate data, ensuring numerical values are rounded to four decimal places.

- **dataset_handler.py**

This utility manages dataset folders inside Microclimate Analysis Data:

- get_dataset_path(): Retrieves the full path of a dataset.
- get_dataset_folders(): Lists all datasets.
- populate_dataset_combo(): Populates a dropdown with dataset names.
- get_images_from_dataset(): Fetches image file paths from a dataset's images/ folder.

2.4 Research and Development Process

- Problem Identification: We aimed to understand how urban growth affects local climates, especially the Urban Heat Island (UHI) effect. Our goal was to create a system that tracks changes in urban features over time and connects them with climate data, focusing on temperature trends.
- Literature Review: We studied urban microclimates, remote sensing, machine learning models, and techniques involving image segmentation research. This led us to the choice

of the SegFormer model due to its excellent performance on high-resolution imagery. We also studied ways to link climate data with urban features. Our research focuses on the process of urbanization in contributing to climate change, particularly UHI, by understanding those works that explain how building density, green spaces, and surface materials affect the local temperature. Additionally, we investigated various climate data analysis techniques, including temperature trend modeling and spatial-temporal data correlation. We also learned coding practices regarding model training, data preprocessing, and GUI development. The technical research helped us streamline the process of training and further improve the efficiency of our model.

- Model Design: The model we propose is inspired by SegFormer-B4, a transformer-based semantic segmentation model that is known to classify urban features such as buildings, roads, vegetation, and water bodies with very high accuracy. We further fine-tuned this model using high-resolution imagery to improve the segmentation result. We utilized the model along with a GUI application that a user can upload images into, generate segmentation maps, and analyze changes over multiple years besides historical climate data. The app facilitates visualization and statistical analysis, making the results accessible to a broad range of users.
- Data Collection: We used the Global Land Cover Mapping dataset from OpenEarthMap, available on Kaggle, which consists of several datasets: xBD, Inria, Open Cities AI, SpaceNet, Landcover.ai, AIRS, GeoNRW, and HTCD. This dataset provides diverse high-resolution imagery and was essential for training and validating our model, ensuring it performs well across different environments.
- Hyperparameter Selection: We tested different settings such as learning rate, batch size, dropout rate, and specific SegFormer parameters like alpha and gamma.
- Model Training: The SegFormer-B4 model was trained using the collected data. We applied data augmentation to improve model robustness and used early stopping and learning rate adjustments to prevent overfitting.
- Evaluation: We evaluated the model's performance using metrics like Intersection over Union (IoU), accuracy, and mean Dice coefficient. These metrics provided a comprehensive assessment of segmentation quality. Tests on different urban areas ensured the model worked well beyond the training data.
- Unit Testing: Unit tests were used throughout development to ensure both the model and GUI worked correctly. This helped catch and fix issues early.
- Iteration: Based on evaluation results, we refined the model and the GUI. Continuous feedback and testing helped improve accuracy and usability.

Climate Data Parameters for Analysis:

The following table describes the climatic data parameters utilized in analysis, including their respective real-life labels, relevant Open-Meteo API parameter IDs, brief definitions, and their importance in relation to climate impacts.

Parameter	**API attribute**	**Explanation**	**Climate Effect**

Maximum Temperature	temperature_2m_max	Highest daily air temperature at 2 meters above ground.	Indicates heatwaves, urban heat island (UHI) effect.
Minimum Temperature	temperature_2m_min	Lowest daily air temperature at 2 meters above ground.	Reflects nighttime cooling, UHI intensity.
Mean Temperature	temperature_2m_mean	Average daily air temperature at 2 meters above ground.	Measures overall temperature trends.
Maximum Wind Speed	wind_speed_10m_max	Highest daily wind speed at 10 meters above ground.	Affects heat dispersion and cooling in urban areas.
Maximum Wind Gusts	wind_gusts_10m_max	Strongest wind gusts recorded at 10 meters above ground.	Influences extreme weather events, ventilation.
Integrated Water Vapour	total_integrated_water_vapour	Total atmospheric water vapor in a vertical column.	Influences precipitation potential and humidity.
Relative Humidity	relative_humidity_2m	Ratio of current air moisture to maximum possible moisture.	Influences heat perception, evaporation rates.
Dew Point Temperature	dew_point_2m	Temperature at which air becomes saturated with moisture.	Indicates atmospheric moisture content.
Surface Pressure	surface_pressure	Atmospheric pressure at the Earth's surface.	Affects weather systems, air mass movement.
Vapor Pressure Deficit	vapour_pressure_deficit	Difference between saturation and actual vapor pressure.	Indicates atmospheric dryness, plant stress.
Deep Soil Temperature	soil_temperature_100_to_255cm	Temperature of deep soil layers.	Reflects long-term ground heat storage.
Deep Soil Moisture	soil_moisture_100_to_255cm	Moisture content in deep soil layers.	Indicates groundwater recharge, drought resilience.
Direct Radiation	direct_radiation	Solar radiation received directly from the sun.	Affects heating of surfaces without scattering.

2.5 Tools Used

- Deep Learning Frameworks: PyTorch and PyTorch Lightning for developing and training the SegFormer-B4 model, providing flexibility and efficiency in handling large-scale datasets and coding simplicity.
- Segmentation Model: The SegFormer-B4 model was sourced from Hugging Face Transformers, leveraging its pre-trained weights for improved segmentation performance on urban/rural features.
- Dataset Management: Kaggle for accessing the Global Land Cover Mapping dataset from OpenEarthMap. Additionally, we created a custom dataset repository on Hugging Face, storing data in Parquet format for optimized performance.
- Data Processing and Analysis: Python libraries such as Pandas, NumPy, and Scikit-learn were used for data manipulation, preprocessing, and statistical analysis.
- Data Augmentation: Albumentations library was employed to enhance model generalization through advanced image augmentation techniques.

- Visualization: Matplotlib was used for data visualization and result presentation, enabling clear graphical representations of segmentation outcomes and climate correlations.
- User Interface Development: The GUI was developed using PyQt5, providing an intuitive interface for image upload, segmentation visualization, and climate data analysis.
- APIs: We integrated APIs such as OpenStreetMap for geospatial data, CountriesNow for country and city information, and Open-Meteo for accessing historical climate data.
- Hyperparameter Optimization: Optuna was used for hyperparameter tuning, optimizing key parameters, and finding the best hyperparameter combination.
- Model Evaluation: Torchmetrics implementations used evaluation metrics, including Intersection over Union (IoU) and mean Dice coefficient, while TQDM provided progress tracking during model training.
- Testing Framework: Pytest and Pytest-Qt were utilized for unit and GUI testing to ensure the reliability and stability of both the machine learning model and the application.
- Version Control: Git and GitHub were used for version control and collaboration, maintaining code integrity and supporting efficient teamwork throughout the project.
- Cloud Computing: We utilized Google Cloud Virtual Machines equipped with Nvidia A100 GPUs to train the SegFormer-B4 model. The high computational demands of the model, combined with large datasets, exceeded the capabilities of our laptops.
- Remote Sensing Imagery: For creating the project sample dataset, we used Google Earth Pro to find aerial images from different years.

2.6 Supervisor Interface

Throughout the development process, we maintained regular communication with our supervising lecturer through meetings and progress updates. The lecturer provided valuable guidance, especially regarding the climate data aspect of the project. His insights helped us understand which climate features to use, how to interpret the data effectively, and how to integrate it seamlessly with our urban segmentation analysis. This support was crucial in ensuring that the climate-related components of our project were both accurate and relevant to our objectives.

2.7 Challenges and Solutions

- Remote Sensing Dataset Selection
  One of the most significant challenges we faced was selecting a reliable dataset for training the segmentation model. First, we chose the LoveDA dataset, but after training and checking, we found that the segmentation masks were inaccurate, resulting in bad performance of the model. So, we decided to change to GeoNRW dataset, which seemed to be more professional and promising. However, mask quality was also inaccurate, and they didn't have the precision necessary for correct segmentation.

- ✓ Solution: Realizing that mask accuracy was critical for our project, we shifted our focus to finding a dataset where the masks were created with high attention to detail. After extensive searching, we found the Global Land Cover Mapping dataset from OpenEarthMap. This dataset explicitly stated that the masks were created manually by human annotators, and upon verification, we confirmed the masks were of high quality. This significantly improved our model's segmentation performance and the overall reliability of our analysis.

- Inconsistent Model Predictions across Environments
  We encountered significant variation in the performance of the model when applied to different geographical areas indicating poor generalization.
  - ✓ Solution: We diversified the training data by including images captured in different environments and slightly modified the model architecture to focus on multi-scale feature learning. We also fine-tuned the model with smaller datasets taken from low-performing regions.

- Hyperparameter Tuning
  Identifying the optimal hyperparameter for SegFormer-B4 such as learning rate, batch size, alpha, or gamma. Every small change in these parameters impacted performance.
  - ✓ Solution: We used Optuna framework to optimize hyperparameters so that a quick, automatic search for optimal parameters can help in enhancing our model's accuracy.

- Historical Climate Data Selection
  Finding accurate historical climate data was a challenge. Initially, we found NCEI (National Centers for Environmental Information) API that provided precise climate data, but it was limited to areas near weather stations, leaving large gaps in regions without nearby stations. This limitation affected our ability to perform consistent climate analysis across different geographical locations.
  - ✓ Solution: We searched for alternative climate data sources that offered comprehensive coverage. We found Open-Meteo API that provides reliable historical climate data globally, regardless of proximity to weather stations.

- Model Overfitting
  The model overfitted during training because it performed quite well on the training data, but poorly when it came to validation data. This was probably due to a limited variability of some datasets, or the model specialized in specific patterns.
  - ✓ Solution: We applied data augmentation using Albumentations framework for introducing variability. In addition, we used different regularization methods such as dropout layers and early stopping mechanisms to avoid overfitting.

2.8 Results and Conclusions

Model results and conclusions

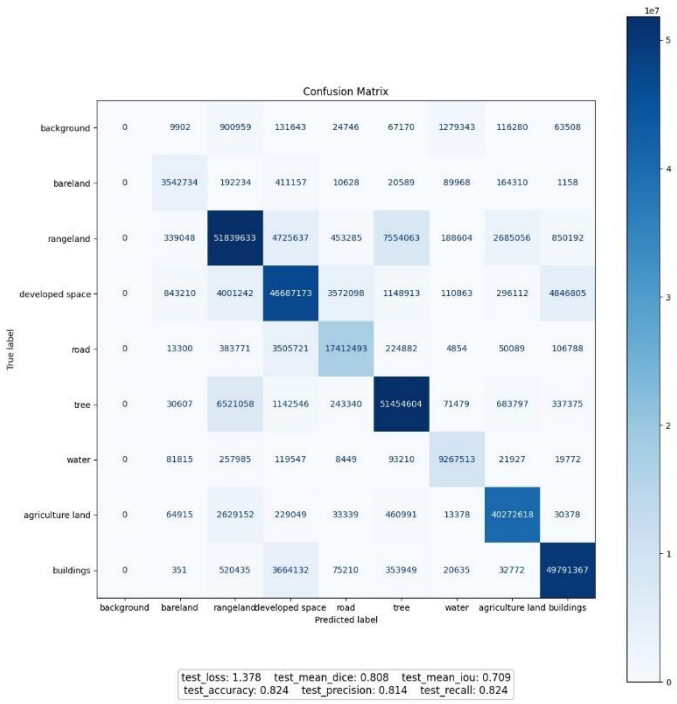
We evaluated different variant of SegFormer backbones, with different hyperparameters, and loss functions to optimize segmentation. We used search grid to find the best learning rate, batch size, and weight decay (L1 Regularization) for better results and faster convergence. Among the loss functions we tried we find that Weighted Cross Entropy + Dice Loss outperformed Focal Loss and pure Weighted CrossEntropy. SegFormer B4 provided the best metrics results, leading to its selection as the final model.

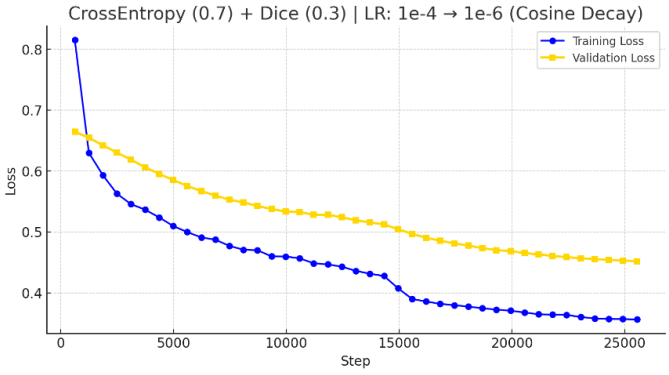
Chosen hyperparameters:

- Learning-Rate: 1e-4 with CosineAnnealing linear reduce over epochs to 1e-6.
- Batch size: 16
- Loss Function: 0.7 * Cross-Entropy + 0.3 * Dice Loss, pixel wise from CE as priority and better ground truth overlap from Dice.
- Weight Decay (L1): 1e-3, allow the model to adjust the weights for the new task without risk of overfitting.

Class Weights: We find that land cover datasets suffer from severe class imbalance, where some classes have significantly fewer pixels than others. We address this issue by calculate the inverse frequency of each class and use these weights by multiplying calculated loss with class weights, ensuring that underrepresented classes contribute more to the overall optimization.

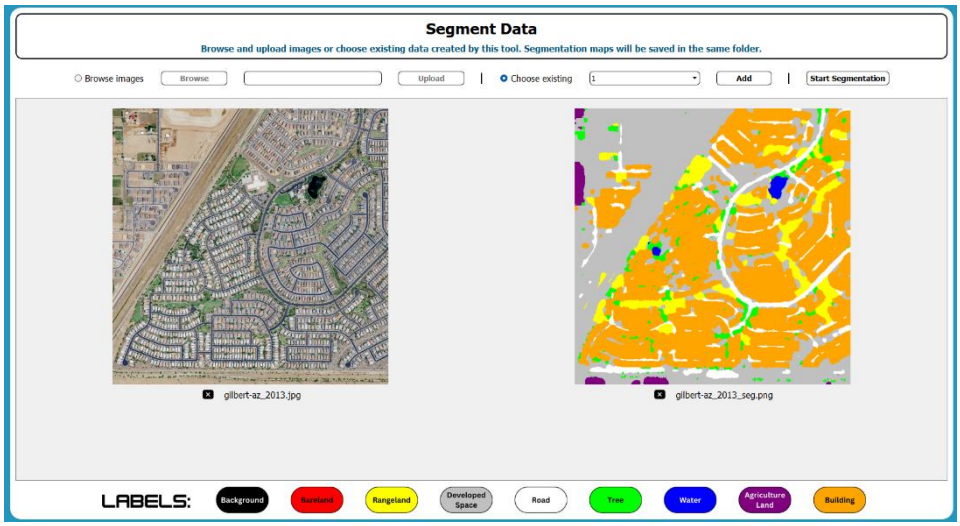
Best train metrics results:





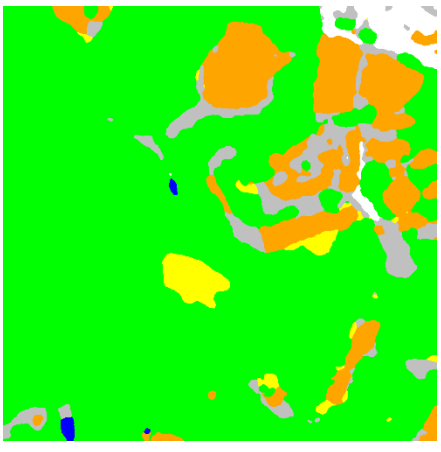
Segmentation example result (from validation):

Segmentation with GUI Application:

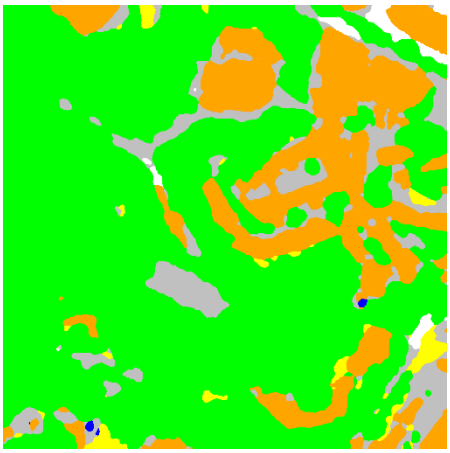


Segmentation and Analysis:

Guangzhou, China: 2007



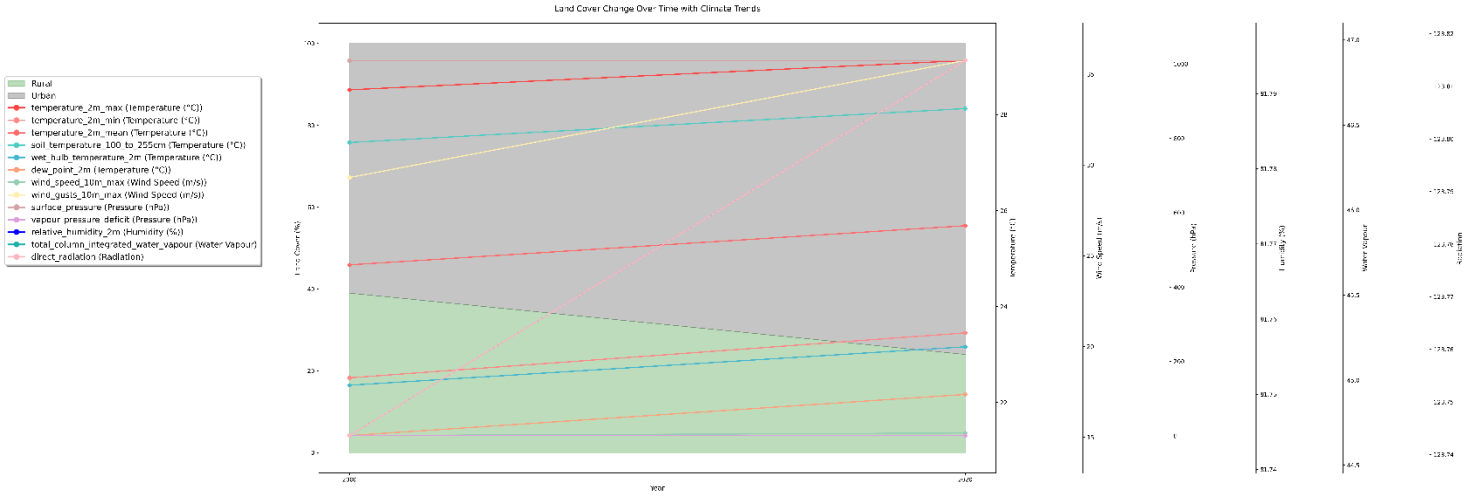
Guangzhou, China: 2022

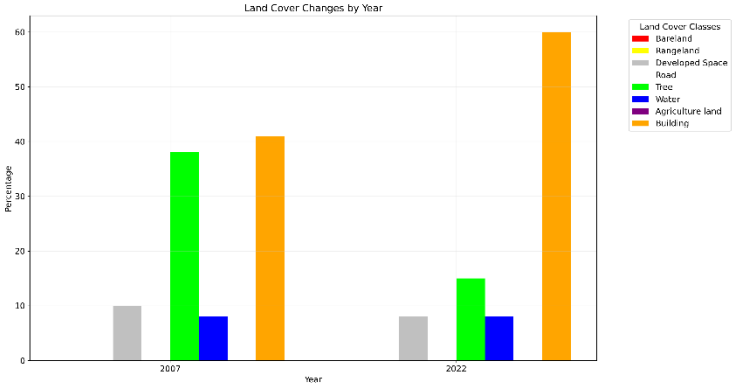


Analysis Results:

year	Bareland (%)	Rangeland (%)	Developed Space (%)	Road (%)	Tree (%)	Water (%)	Agriculture land (%)	Building (%)
2007.0	0.0	0.0	10.0	8.0	38.0	8.0	0.0	41.0
2022.0	0.0	0.0	8.0	8.0	15.0	8.0	0.0	60.0

year	Temp Max 2m (°C)	Temp Min 2m (°C)	Temp Mean 2m (°C)	Wind Speed 10m (m/s)	Wind Gusts 10m (m/s)	Relative Humidity (%)	Dew Point 2m (°C)	Surface Pressure (hPa)	Vapour Pressure	Soil Temp 100-255cm (°C)	Soil Moisture 100-255cm	Wet Bulb Temp (°C)	Integrated Water Vapour	Direct Radiation	urban	rural
2007.0	26.9	19.34	22.72	13.61	30.93	74.47	17.47	1009.5	0.74	23.82	0.38	19.39	39.55	108.39	59.0	41.0
2022.0	27.44	20.15	23.41	13.18	29.83	75.65	17.48	1010.01	0.72	24.02	0.34	20.2	43.12	112.56	76.0	24.0





Conclusion: We can see clear positive correlation between rise of urban features and rise of climate parameters

2.9 Lessons Learned

- The Importance of Dataset Quality
  One of the most significant lessons was the importance of evaluating datasets before committing to model training. Initially, we spent time working with the datasets, only to discover that the segmentation masks were inaccurate and inconsistent. This led to wasted time and resources retraining the model. Early dataset validation would have saved us considerable effort.
- Investing Time in Learning Machine Learning Techniques
  One of the key lessons we learned was the importance of dedicating more time to understanding the machine learning techniques and methodologies before diving into extensive model training. We spent a significant amount of time running multiple training sessions, only to receive suboptimal results due to gaps in our knowledge about model optimization, data preprocessing, and evaluation strategies. This not only delayed our progress but also led to unnecessary resource consumption. Investing more time in the beginning to improve our knowledge of relevant machine learning concepts would have helped us achieve better results and efficient use of our time and computational resources.
- Early Testing and Iteration
  Another lesson learned was to include testing and iteration during development. Early unit tests of both the model and the GUI might have caught integration issues earlier and reduced last-minute adjustments that were needed. Continuous evaluation helped us to improve both the technical and analytical components of the project.
- Finding the Right Computational Platform
  Identifying the right platform to train the SegFormer-B4 model also took us time. Initially, we tried several platforms for model training, but we faced limitations in terms of performance, resource availability, cost efficiency, and friendly UI. We found that Google Cloud offered the best platform, yet we had to spend time learning how to use the platform the best way. Even after selecting Google Cloud, we encountered delays because accessing NVIDIA A100 GPUs required a Google quota request, which took additional time to be reviewed and approved. This experience taught us the importance of early planning in the project, including understanding platform requirements and potential administrative delays.

2.10 Project Benchmarks

- **Development of a Segmentation Model**
  We successfully developed and fine-tuned a SegFormer-B4 model for semantic segmentation of urban and rural features such as buildings, roads, vegetation, and water bodies. In preparing the model, high-resolution imagery was used from the OpenEarthMap dataset to create segmentation masks. The achieved model results in strong performance in classifying different land cover types across diverse environments.
- **Model Evaluation Metrics**
  The model's performance was evaluated using standard metrics, including Intersection over Union (IoU), Cross-Entropy (CE), and the Dice coefficient. These metrics provided

an assessment of the model's segmentation accuracy and consistency which showed that the model performed well.

- **Correlation Analysis Between Urban Features and Climate Data**
  We conducted a correlation analysis to explore the relationship between urban feature changes and climate data, focusing on temperature variations over time. The analysis determined clear trends, such as the increase in urban features like buildings and roads correlating with rising temperatures, supporting the Urban Heat Island (UHI) effect hypothesis. This correlation provided valuable insights into how urbanization impacts local climate conditions.
- **Sample Dataset Result for Researchers**
  A sample dataset was created, combining segmented urban feature maps with historical climate data from various locations. This dataset includes multi-year segmentation results and corresponding climate records of areas we found had a significant rise in building in the last decades. The analysis results show a clear relationship between the rise of urban features and deteriorating climate conditions which proves our research claim.
- **Development of an Interactive GUI for Analysis**
  We developed a user-friendly GUI that allows users to use our trained model and perform analysis with our research approach. The application integrates climate data, enabling users to visualize the correlation of urban growth with climate variations through graphs and statistical outputs. The intuitive design and functionality of the GUI make it accessible to researchers, urban planners, and other stakeholders.
- **Integration of Climate Data with Urban Analysis**
  The integration of historical climate data with the result of urban segmentation was one of the project's objectives. We linked multi-year climate data with segmented urban features to enable comprehensive analyses of how trends in urbanization affect local climates. This helped identify meaningful insight into the dynamics of urban microclimates.

3 User Operating Guide

3.1 Model Guide

3.1.1 System Requirements

To fine-tune the SegFormer model, ensure your system meets the following requirements:

- Operating System: Cross-platform (Linux, Windows, macOS)
- Python Version: 3.9
- GPU:
  - For SegFormer B0-B1: Minimum 8 GB
  - For SegFormer B2-B4: Recommended 16 GB+
- RAM:

    - For SegFormer B0-B1: Minimum 16GB
    - For SegFormer B2-B4: Recommended 32GB
- CUDA Version: 11.3+ (for NVIDIA GPUs)
- Disk Space: At least 15 GB of free space

3.1.2 Local Fine-tuning

To install and set up the project, follow these steps:

```
pip install git+https://github.com/MMA-org/ML-Microclimate-Analysis.git  # Install the package
ucs -h # To verify installation, view all available commands/options.
ucs train # Default training configurations.
ucs train --batch_size 8 # change configurations from cli.
ucs -c path/to/config.yaml train # With custom configurations.
ucs evaluate -v {version number} # To evaluate the model after. training
```

3.1.3 Configuration file

For custom fine-tuning, edit conf.yaml, or create a new yaml file from the template, every configuration can be overridden using the command line.

```yaml
# DATA
dataset:
  dataset_path: "erikpinhasov/landcover_dataset"
  batch_size: 16
  num_workers: 8
  do_reduce_labels: False
  pin_memory: True

# PROJECT DIRECTORIES
directories:
  models: models
  pretrained: models/pretrained_models
  logs: models/logs
  checkpoints: models/logs/checkpoints
  results: results

# TRAINING SETTINGS
training:
  model_name: b0
  max_epochs: 50
  learning_rate: 2e-5
  weight_decay: 1e-3
  ignore_index: 0 # int | None
  weighting_strategy: "raw" # Options: "none", "balanced", "max", "sum", or "raw"
  gamma: 2.0
  id2label:
    0: "background"
```

```
  1: "bareland"
  2: "rangeland"
  3: "developed space"
  4: "road"
  5: "tree"
  6: "water"
  7: "agriculture land"
  8: "buildings"

# CALLBACKS SETTINGS
callbacks:
  early_stop_patience: 5
  early_stop_monitor: "val_loss"
  early_stop_mode: "min"
  save_model_monitor: "val_mean_iou"
  save_model_mode: "max"
```

For further detailed instructions on installation, configuration, and usage, please refer to the official documentation available at docs

3.2 GUI Application Tool Guide

- **System Requirements**
  - Operating System: Windows 10/11, Linux, or macOS
  - Python Version: Python 3.10 or above
  - Required Libraries: PyQt5, matplotlib, pandas, numpy, requests, pycountry, pytest, pytest-qt, torch, transformers, pydensecrf, pyinstaller, wheel, setuptools, seaborn
  - Hardware Recommendations:
    - RAM: Minimum 4 GB (8-16 GB recommended)
    - Disk Space: 200MB free storage (Standalone executable file)
    - GPU: Optional but recommended for faster segmentation processing (NVIDIA CUDA-compatible GPU)

- **Installation Instructions**
  - Clone the Repository:
    git clone https://github.com/MMA-org/Microclimate-Analysis-Tool.git
    cd Microclimate-Analysis-Tool
  - Install Dependencies:
    pip install -r requirements.txt
  - Microsoft Visual C++ Build Tools: Required to compile pydensecrf
    1. Download from Microsoft Build Tools.

2. During installation, select "Desktop development with C++".
3. Ensure that the C++ build tools are checked.

- Run the Application:
  python main.py
- Creating Executable standalone file with PyInstaller command:
  pyinstaller --onefile --windowed --icon=assets/icons/main.ico --name "Microclimate Analysis" --add-data "app/ui_main.ui;app" --add-data "app/utils/styles.json;app/utils" --add-data "assets/icons;assets/icons" main.py

- **Testing**

  The Tool includes unit tests to ensure the reliability and correctness of its core functionalities. These tests are implemented using pytest and pytest-qt for GUI testing.

  Run all tests with: pytest tests/

  - Test Files Overview:
    - conftest.py: Sets up the testing environment by initializing a shared QApplication instance required for PyQt5 GUI tests.
    - test_create_data_controller.py: Tests the Create Data page, focusing on UI interactions, input validation (e.g., location selection, image uploads), file handling, and metadata management.
    - test_segment_data_controller.py: Tests the Segment Data page, covering UI controls, image upload/removal functionality, and segmentation process.

4 Maintenance Guide

4.1 Model Maintenance Guide

The project's software structure is detailed in the documentation pages. Running instructions and setup guidelines can be found in the official documentation. Our goal is to maintain flexibility, allowing seamless adjustments between SegFormer B0-B4 models, datasets, and training configurations. By following this maintenance guide alongside the documentation, you can ensure that the project remains efficient, scalable, and easy to maintain.

4.1.1 Key Components

- **Data Module:** Manages dataset loading and preprocessing.
- **Model Module:** Contains the machine learning models and training routines.
- **Utils Module:** Provides utility functions for configuration, metrics, and callbacks.
- **Core Module:** Core functionality and model pipelines.

more details in documentation API Reference.

4.1.2 Install The Package Development Purposes

```
git clone https://github.com/MMA-org/ML-Microclimate-Analysis.git # Clone the repo
git install -e .[dev]  # Install package in edit mode, and necessary dev dependencies.
```

4.1.3 Running the Project

- Configure and launch experiments through structured configuration files.
- Use logging and monitoring tools to track progress and debug potential issues.

4.1.4 Maintenance Best Practices

- **Dependency Management:** Regularly update software dependencies and verify compatibility, make sure to update pyproject.toml dependencies for the package.
- **Modularity:** Group functions and classes logically within their respective modules, for e.g., data, model, util. Ensure new features are added to the appropriate module rather than mixing concerns.
- **Code Documentation:** Maintain structured code comments and update project documentation when changes occur.
- **Version Control:** Keep track of modifications using Git and maintain a structured branching strategy.
- **Logging & Debugging**: Ensure all critical functions log errors and warnings to help with quick debugging.

4.2 GUI Application Tool Maintenance Guide

Provided as standalone executable, in GitHub repository: ‘Microclimate Analysis.exe’.

Key operations that users can perform in the analysis tool, following a step-by-step workflow:

- Uploading Images (Create Data Page):
  - Click Browse to select remote sensing images from your device.
  - Click Upload to add the selected images to the session.
- Choosing Location or Adding Coordinates:
  - Option 1: Select a country and city from the dropdown menus to automatically fetch coordinates.
  - Option 2: Manually enter latitude and longitude.
- Adding Image Metadata:
  - Enter the corresponding year for each uploaded image in the provided input fields.
- Saving the Dataset:
  - Enter a session name to identify your dataset.
  - Click Save to store the images, metadata, and location information.
- Segmenting Images (Segment Data Page):

- Upload Images for segmentation or select an Existing Dataset from the dropdown.
- Click Start Segmentation to generate segmentation maps, which will be displayed side-by-side with the original images.

- Analyzing Data (Analysis Page):
  - Select a saved dataset from the dropdown menu.
  - Click View Analysis to fetch historical climate data based on the dataset's coordinates, graph with climate parameters for each years, along with the calculated percentage of urban/rural objects will be viewed in graph trough the selected years, in order to check correlations with urbanization trends.