CHAPTER 1

INTRODUCTION

Remote sensing satellite images are increasingly integral to a wide range of applications, including land cover classification, crop monitoring, disaster response, and environmental monitoring. The vast availability of these images has led to a surge in demand for efficient and accurate feature extraction methods. However, the intricacies of image processing and analysis often necessitate specialized skills and advanced software, which can limit accessibility for non-experts.

1.1 Feature extraction

Feature extraction from remote sensing satellite images involves a series of critical steps: image preprocessing, feature extraction, and post-processing. Image preprocessing includes correcting atmospheric and geometric distortions while enhancing overall image quality. During the feature extraction phase, relevant features—such as texture, shape, and spectral characteristics—are identified and extracted from the image. Post-processing further refines these features, preparing them for subsequent analysis. Despite the crucial role these steps play in various applications, they are often time-consuming and complex, requiring technical proficiency that may be beyond the reach of many users.

Several specialized software packages, such as ERDAS Imagine, ENVI, and ArcGIS, offer robust solutions for feature extraction. However, these platforms often come with steep learning curves, necessitating significant training and experience. Additionally, they can be cost-prohibitive and lack the flexibility or customization options that certain users might require for specific tasks. As a result, there is a growing need for more accessible, user-friendly tools that can democratize the use of remote sensing data.

In recent years, Python has emerged as a powerful and versatile programming language for remote sensing and image processing tasks. Its simplicity, flexibility, and rich ecosystem of libraries—such as OpenCV, scikit-image, and GDAL—make it an ideal choice for developing feature extraction algorithms. These libraries offer a range of efficient tools that facilitate various stages of image processing, from basic enhancements to complex feature extraction. Despite the advantages Python offers, the feature extraction process remains intricate and can be challenging for those without a background in programming or remote sensing.

The development of a user-friendly web-interface for feature extraction from remote sensing satellite images would address these challenges by simplifying the process and making it more accessible to a broader audience. Such a system would enable users to upload satellite images and select specific features for extraction through a simple, intuitive interface. Leveraging Python's extensive libraries, the system would ensure that users can obtain accurate and efficient results without needing to delve into the complexities of image processing.

This proposed system would also incorporate tools for visualization and post-processing, allowing users to refine and analyze the extracted features further. The impact of such a tool could be profound, particularly in fields like land cover classification and crop monitoring, where improved accessibility to accurate data can drive better decision-making. Similarly, disaster response and environmental monitoring efforts would benefit from the rapid and precise extraction of critical features from remote sensing data.

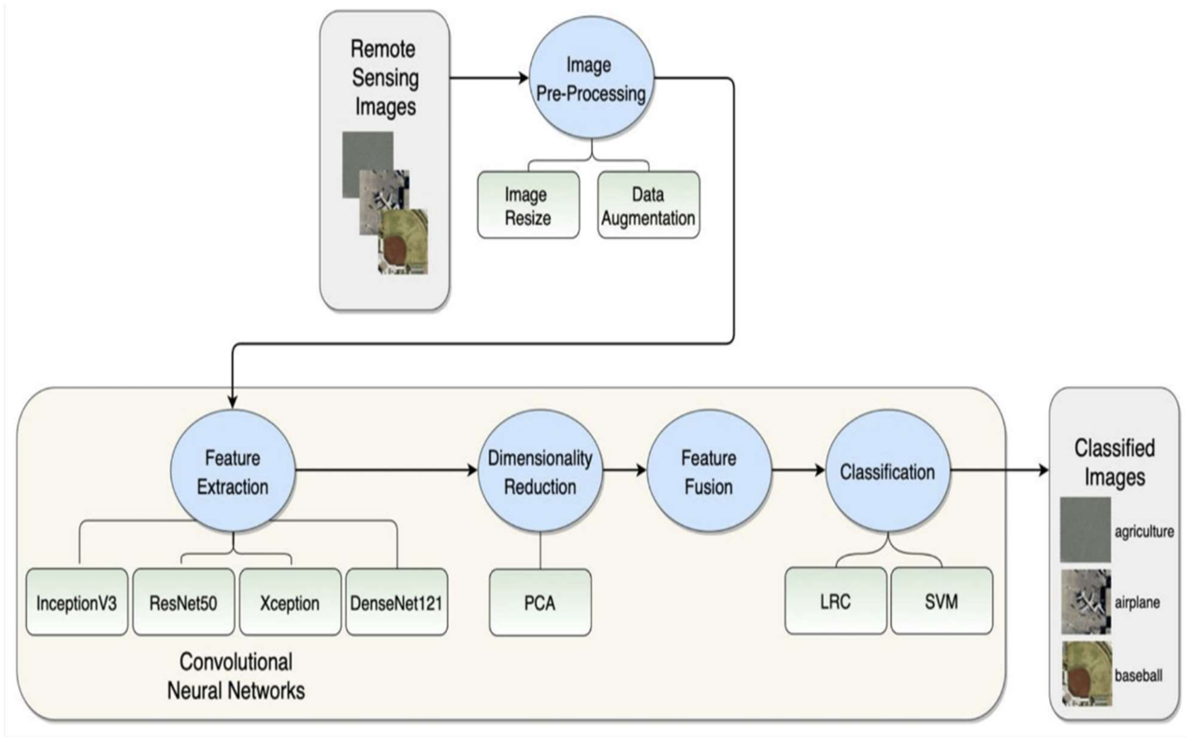


Figure 1. Workflow of the proposed method. [p1]

1.2 Motivation and Perspective

Remote sensing satellite images have become an indispensable tool for numerous applications, ranging from land use/land cover mapping to disaster management. However, the technical complexities associated with image processing and feature extraction often limit the accessibility of these powerful tools to a specialized audience. This project aims to bridge this gap by developing a user-friendly web interface that simplifies the process of extracting valuable information from remote sensing images.

By providing an intuitive platform that requires minimal technical expertise, we aim to empower a broader range of users, including researchers, policymakers, and environmental scientists, to leverage the potential of remote sensing data. This democratization of access will facilitate data-driven decision-making, leading to more informed and effective solutions to pressing global challenges.

1.3 Description of Theoretical Concepts

Remote Sensing: Remote sensing is the science and art of obtaining information about an object, area, or phenomenon through the analysis of data acquired by a device that is not in contact with the object, area, or phenomenon under investigation. Remote sensing systems typically involve a sensor that records electromagnetic radiation emitted or reflected from the Earth's surface.

Image Preprocessing: Image preprocessing is a crucial step in remote sensing image analysis. It involves various techniques to enhance image quality and correct distortions caused by factors like atmospheric conditions, sensor noise, and geometric errors. Common preprocessing techniques include:

- **Radiometric Correction:** Adjusting pixel values to account for atmospheric effects and sensor calibration.
- **Geometric Correction:** Correcting spatial distortions and aligning images to a specific coordinate system.
- **Atmospheric Correction:** Removing the effects of the atmosphere on the recorded radiance.

Python and Relevant Libraries: Python, with its rich ecosystem of libraries, has become a popular language for remote sensing and image processing. Libraries such as OpenCV, scikit-image, and GDAL provide essential tools for image reading, writing, processing, and analysis.

CHAPTER 2

LITERATURE REVIEW

2.1 Related Literature Review

The application of remote sensing satellite images has revolutionized various fields including environmental monitoring, land cover classification, and disaster response. However, the complexity of processing and extracting features from these images has necessitated the development of advanced tools and software. This literature survey reviews existing methods and tools for feature extraction, focusing on their strengths and limitations, and highlights the need for a more accessible web-interface.

2.1.1 Image Preprocessing and Feature Extraction

Preprocessing remote sensing images is essential for removing distortions and enhancing image quality. Techniques such as atmospheric correction and geometric calibration are foundational to accurate feature extraction [1]. For instance, the Atmospheric Correction Parameter Calculator (ATCOR) is frequently employed to correct atmospheric distortions [2]. Once preprocessing is complete, feature extraction involves identifying key attributes such as texture, shape, and spectral properties. Traditional methods include statistical texture measures and spectral analysis, which have been widely used but often require significant computational resources and expertise [3].

2.1.2. Existing Software Solutions

Several software packages are commonly used for feature extraction from remote sensing images. ERDAS Imagine and ENVI are two prominent examples that offer comprehensive tools for image analysis [4]. ERDAS Imagine provides functionalities for image processing, including feature extraction and classification, but requires substantial training to use effectively [5]. ENVI, similarly, offers extensive image processing capabilities but can be costly and complex for non-experts [6].

2.1.3. Python-Based Tools

The advent of Python as a programming language for image processing has introduced more accessible alternatives. Libraries such as OpenCV, scikit-image, and GDAL provide powerful tools for feature extraction and image analysis. OpenCV, for example, is renowned for its extensive set of computer vision tools, including feature detection and image segmentation [7]. scikit-image offers a user-friendly interface for various image processing tasks, while GDAL focuses on geospatial data processing, including satellite imagery [8]. These libraries simplify the development of custom feature extraction algorithms but still require programming skills.

2.1.4. Web-Based Solutions

Recent developments in web-based interfaces have sought to bridge the gap between complex image processing and user accessibility. Web platforms such as Google Earth Engine provide an intuitive interface for analysing satellite imagery and extracting features, making advanced remote sensing tools more accessible to non-experts [9]. However, these platforms may not offer the same level of customization as standalone software solutions or Python libraries.

2.1.5. Need for a User-Friendly Web-Interface

a. Despite the availability of advanced tools and libraries, the process of feature extraction remains challenging for many users due to its complexity. There is a clear need for a user-friendly web-interface that simplifies the process of uploading images, selecting features, and analysing results. Such an interface would leverage the power of existing Python libraries while providing an intuitive, accessible platform for users with limited technical backgrounds.
b. As the demand for more sophisticated analysis increased, machine learning algorithms began to gain traction in the field of remote sensing. Algorithms such as Support Vector Machines (SVM), Random Forest (RF), and k-Nearest Neighbours (k-NN) have been extensively applied to satellite imagery for feature extraction and classification tasks. These algorithms are capable of handling large datasets with high dimensionality, making them well-suited for the complex nature of satellite imagery. The use of machine learning techniques has been shown to significantly improve the

accuracy of feature extraction, particularly when combined with OBIA. For instance, studies have found that integrating SVM with OBIA results in more precise land cover classification compared to traditional methods.

c. The advent of deep learning has further revolutionized the field of remote sensing. Convolutional Neural Networks (CNNs), in particular, have shown exceptional promise in automating the feature extraction process. Unlike traditional methods, which require manual feature engineering, CNNs can learn hierarchical features directly from the data. This ability to learn both low-level and high-level features makes CNNs particularly effective for high-resolution satellite imagery, where the complexity and variety of features are high. Several studies have demonstrated that CNNs outperform conventional machine learning methods in tasks such as object detection, scene classification, and change detection.

d. Despite these advancements, several challenges persist. The high computational requirements of deep learning models, the need for large annotated datasets, and the risk of overfitting are some of the key challenges that researchers continue to address. Moreover, the integration of multi-source data, such as combining optical imagery with LiDAR and Synthetic Aperture Radar (SAR) data, is an emerging area of interest. Multi-sensor fusion has the potential to provide a more comprehensive understanding of the Earth's surface, leveraging the unique strengths of each sensor type for more accurate feature extraction. In conclusion, the literature on feature extraction from remote sensing satellite images reflects a dynamic and evolving field. The transition from manual interpretation to advanced machine learning and deep learning methods has greatly enhanced the accuracy, efficiency, and applicability of remote sensing data. As research continues to address existing challenges, the potential for new applications and improvements in the field remains vast.

2.1.6 Typical Deep Network Models

In this section, we briefly review the following three typical deep neural network models that have been used for RS image classification. More details about DL architectures in machine learning can be found in (Bengio, 2009; Bengio, Courville, & Vincent, 2013).

- *Convolutional neural networks*

A Convolutional Neural Network (CNN) is a type of Deep Learning neural network architecture commonly used in Computer Vision. Computer vision is a field of Artificial Intelligence that enables a computer to understand and interpret the image or visual data. When it comes to Machine Learning, Artificial Neural Networks perform really well. Neural Networks are used in various datasets like images, audio, and text. Different types of Neural Networks are used for different purposes, for example for predicting the sequence of words we use Recurrent Neural Networks more precisely an LSTM, similarly for image classification we use Convolution Neural networks. In this blog, we are going to build a basic building block for CNN. Generally, a CNN mainly consists of three key parts: convolution layers, pooling layers, and fully connected layers. Different parts play different roles. An example of CNNs is shown in Figure 1.

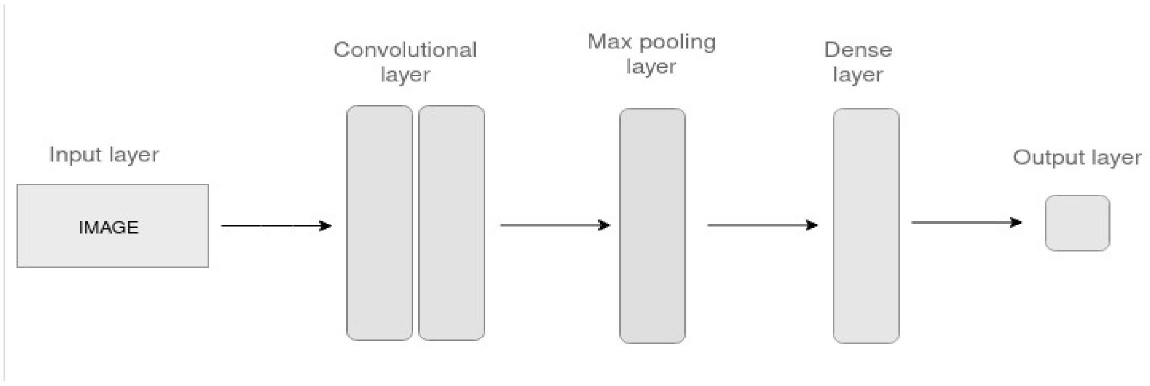


Figure 2: An example of convolutional neural networks

In convolution layers, the input maps are convolved with learnable kernels and are subsequently put through the activation function to form the output feature maps. This process can be formulated as:

where is the value at the position (h, w) of the kernel connected to the mth feature map in the (l − 1)th layer, H i and W i are the height and width of the kernel, respectively, and b l, j is the bias of the jth feature map in the lth layer. Such convolution layers introduce weight sharing mechanisms within the same feature maps, which helps reduce significantly the number of parameters otherwise required. It can take two-dimensional (2D) images with any scale directly as input while reserving the location information of objects in the images. Due to the recognition of the inherent advantages of convolution operation, a significant amount of work has been focused on improving the ability of convolution layers in the literature. For instance Lin, Chen, and Yan (2013) proposed a network in a network, substituting the conventional convolution layer with a multilayer perceptron consisting of multiple fully connected layers. Long, Shelhamer, and Darrell (2017) replaced the fully connected layers in a CNN with a deconvolution layer to build a novel convolutional network.

Generally, a pooling layer follows a convolutional layer and it is used to reduce the dimensionality of feature maps. There are two types of basic pooling operation which are the most commonly used: average pooling and max pooling, as shown in Figure 2. Detailed theoretical analysis of these is beyond the scope of this paper, but can be found in Scherer, Muller, and Behnke (2010). As the computation process of pooling operation takes neighbouring pixels into account, a pooling layer is translation invariant. Apart from average and max pooling, there are several other pooling operations, including spatial pyramid pooling (He et al., 2014), stochastic pooling (Zeiler & Fergus, 2013) and def-pooling (Ouyang et al., 2014).

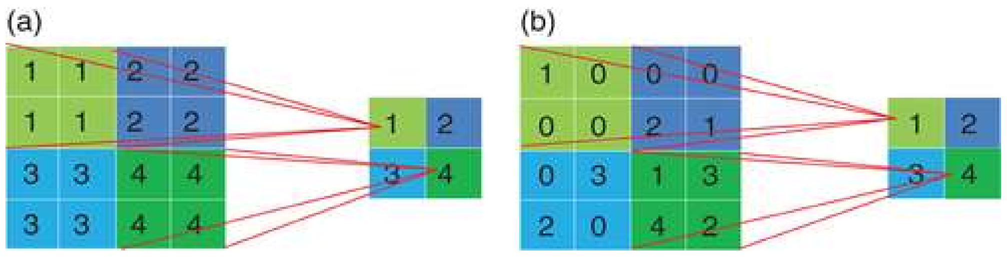


Figure 3: Two basic pooling operations. (a) Average pooling. (b) Max pooling

- *Stacked autoencoders*

A stacked autoencoder (SAE) is a deep network model consisting of multiple layers of autoencoders (AEs) in which the output of one layer is wired to the input of the successive layer as shown in Figure 4. An AE has one visible layer of d inputs and one hidden layer of h units with an activation function f. During training, it first maps the input x ∈ R d to the hidden layer and get the latent representation y ∈ R h. Then y is mapped to an output layer that has the same size with input layer, which is called reconstruction.

$$y = f(W_y x + b_y)$$
$$z = f(W_z y + b_z) \qquad (1)$$

where Wy, Wz denotes input-to-hidden and hidden-to-output weights, respectively, b y and b z denote the bias of hidden and output units, respectively, and f(·) denotes the activation function, which apply element-wise to its arguments. The loss function or energy function J(θ) measures the reconstruction z when given input x,

$$J(\theta) = \frac{1}{2M} \sum_{m=1}^{M} \| z^{(m)} - x^{(m)} \|_2^2 \qquad (2)$$

where M denotes the number of training samples. The objective is finding the parameters $\theta = (W, b_y, b_z)$ which can minimize the difference between the output and the input over the whole training set $X = [x^{(1)}, x^{(2)}, \ldots, x^{(m)}, \ldots, x^{(M)}]$, and this can be efficiently implemented via the stochastic gradient descent algorithm (Johnson & Zhang, 2013).

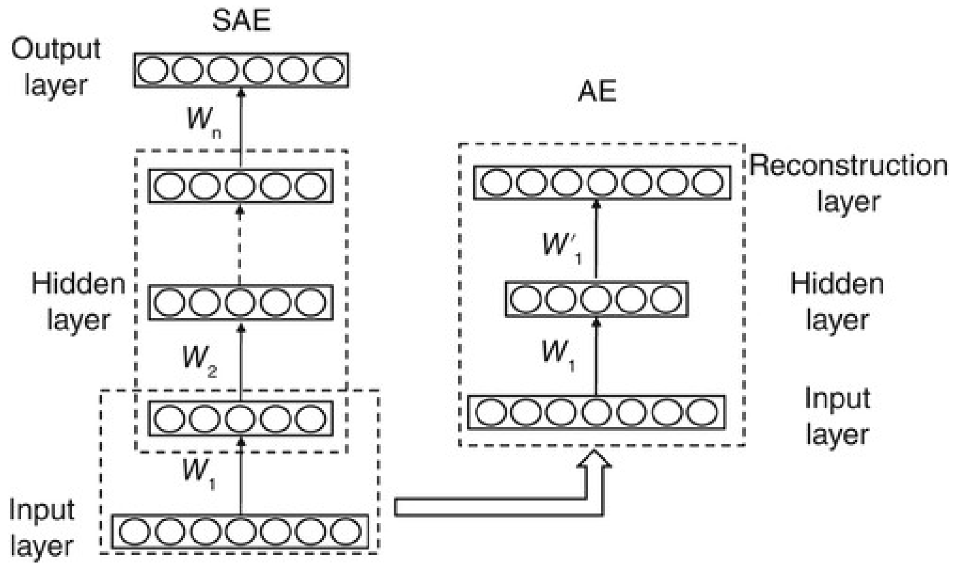


Figure 4: Stacked auto-encode and auto-encoder

- *Deep belief networks*

An DBN model is constructed with a hierarchically arranged series of RBMs as shown in Figure 5. An RBM at l-layer in DBN is an energy-based generative model that consists of a layer with I binary visible units $v^l = \{v_1^l, v_2^l, \cdots, v_I^l\}$and a layer with J binary hidden units $h^l = \{h_1^l, h_2^l, \cdots, h_J^l\}$. The energy of the joint configuration of the visible and hidden units (v l, h l) is

$$E(v^l, h^l | \theta^l) = -\sum_{i=1}^{I} a_i^l v_i^l - \sum_{j=1}^{J} b_j^l h_j^l - \sum_{i=1}^{I}\sum_{j=1}^{J} w_{ij}^l h_j^l v_i^l \tag{3}$$

where $\theta^l = \left\{ w_{ij}^l, a_i^l, b_j^l, i = 1,2,\cdots,I; j = 1,2\cdots,J \right\}$ forms the set of model parameters. An RBM defines a joint probability over the hidden units as

$$p(v^l, h^l | \theta^l) = \frac{\exp\left(-E(v^l, h^l | \theta^l)\right)}{Z(\theta^l)} \tag{4}$$

where *Z* is the so-called partition function,

$$Z(\theta^l) = \sum_{v^l}\sum_{h^l} \exp\left(-E(v^l, h^l | \theta^l)\right) \tag{5}$$

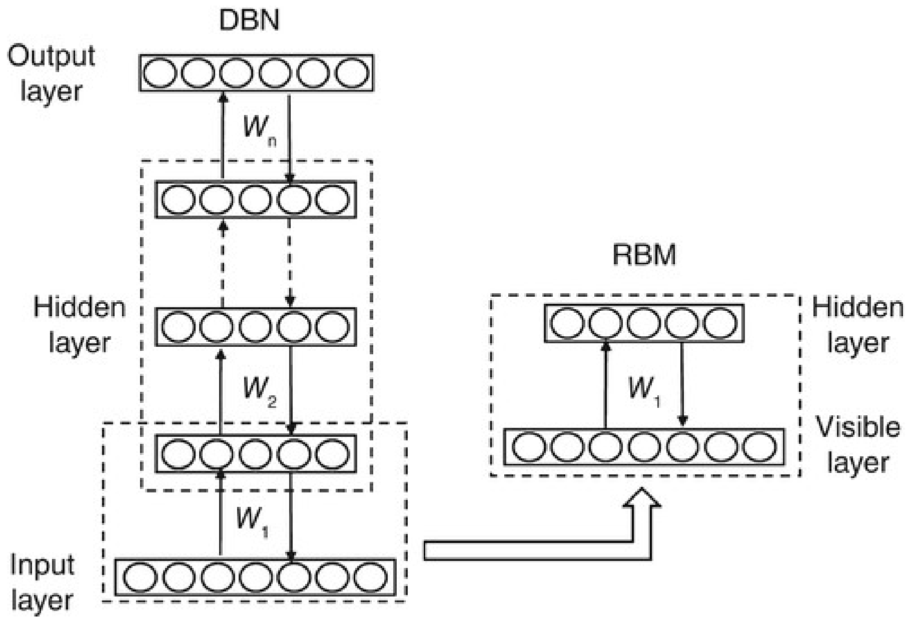


Figure 5: Deep belief network and restricted Boltzmann machine

DBN is a probabilistic generative model which provides a joint probability distribution over observable data and labels. A DBN first takes advantages of an efficient layer-by-layer greedy learning strategy to initialize the deep network, and then fine-tunes all of the weights jointly with the desired outputs.

CHAPTER 3

PROBLEM FORMULATION

3.1 Description of Problem Domain

Remote sensing has revolutionized environmental monitoring, urban planning, agriculture, disaster management, and several other domains by providing high-resolution satellite imagery. The extraction of meaningful features from remote sensing satellite images presents several significant challenges that must be addressed to achieve accurate and efficient results. These challenges stem from the complexity and volume of data, variability in image quality, diversity in landscape features, and the demands of high-resolution imagery. Each of these issues is discussed in detail below:

3.1.1. Complexity and Volume of Data

Remote sensing satellites generate vast amounts of data, covering large geographical areas across multiple spectral bands. This results in several issues:

- *Large Data Sets:* The sheer volume of satellite imagery, often in terabytes, makes manual analysis nearly impossible. Effective processing requires automated algorithms capable of handling such extensive data.
- *High Dimensionality:* Satellite images often contain information across several spectral bands, leading to high-dimensional data. The complexity of this data requires sophisticated methods to extract relevant features without losing critical information.
- *Data Redundancy:* Large datasets can include redundant information across spectral bands, making it challenging to focus on the most relevant features for analysis.

3.1.2. Variability in Image Quality:

The quality of satellite images can vary significantly due to several factors, which introduces challenges in feature extraction:

- *Inconsistent Image Quality:* External factors such as lighting conditions, shadows, sensor noise, and even the satellite's orbit can affect image quality. Poor quality images can result in inaccurate feature extraction, making it necessary to develop robust algorithms that can handle these inconsistencies.
- *Environmental Influences:* Seasonal changes, atmospheric conditions, and cloud cover can alter the spectral signatures captured in satellite images. These variations complicate the classification and extraction processes, as algorithms must adapt to different conditions to maintain accuracy.
- *Resolution Discrepancies:* Different satellites provide images at varying resolutions, leading to discrepancies in data interpretation and feature extraction.

3.1.3. Diverse Landscape Features

The diversity of landscapes within satellite images presents a significant challenge:

- *Varied Terrain Types:* Satellite images cover diverse terrains such as urban areas, forests, water bodies, and agricultural fields. Each terrain type has distinct spectral characteristics, making it difficult to develop a one-size-fits-all feature extraction method.
- *Heterogeneity in Urban Areas:* Urban environments are particularly challenging due to their heterogeneity, with various small-scale features such as buildings, roads, and vegetation. Accurate extraction of these features requires algorithms capable of distinguishing between closely situated and similar-looking objects.
- *Dynamic Landscapes:* Certain landscapes, such as coastal regions or areas prone to seasonal flooding, undergo frequent changes. Feature extraction methods must account for these dynamic environments to ensure consistent results.

3.1.4. Challenges with High-Resolution Images

High-resolution satellite images provide greater detail but also introduce specific challenges:

- *Over-Segmentation:* High-resolution images can lead to the creation of too many small regions during segmentation, making it difficult to analyze larger features coherently. This over-segmentation complicates the feature extraction process.
- *Increased Noise:* Higher resolution also increases the amount of noise in the images, which can obscure significant features and reduce the accuracy of the extraction.
- *Processing Power and Storage:* The increased detail in high-resolution images requires more processing power and storage capacity, making it challenging to manage and analyze these images efficiently

3.1.5. Logistic Regression Classifier

"[10] Logistic Regression is a supervised Machine Learning (ML) method used for classification tasks. The input XX is a matrix containing N data instances, each described by K features. Inputs x_{ij} are K-length feature-vectors (x_i's), which are continuous, with indexes j=1,……,K and i=1,……, N. Output y_i falls in the interval {0,1} and it is a binary variable, with Bernoulli distribution and parameter p_i. The decision/activation function of "Linear Classifier" Logistic Regression is called 'logistic function' (sigmoid). The main characteristic of the sigmoid function is to determine the output of one system in the interval (0,1), regardless of the variables on its input. The posteriors are expressed through the logistic function as

$$P\left(\frac{Y}{X}\right) = \frac{1}{1+e^{-f(x)'}} \qquad (6)$$

In the above expression features (xj) and the corresponding weights (βj) make up the function $f(x)$.

In addition to the 'logistic function,' this Machine Learning (ML) method has an "objective function"—"Maximum Likelihood Estimation (MLE)." Its purpose is to make the "likelihood function" of the training process as large as possible and is expressed with

$$arg\ max: log\{\prod_{i=1}^{n} \quad P(yi/xi)\hat{}yi(1-P(yi/xi))\hat{}(1-yi)\} \qquad (7)$$

Its parameters are: the output y which is in the interval (0,1), $P(yi|xi)$ is the posterior probability and is given with 1/(1+$e\hat{}-f$), and the weights vector in $f(x)-\beta$."

3.1.6. Support Vector Machine (SVM)

"The SVM is a discriminative classifier formally defined by a separating hyperplane. Given a set of training data, the SVM finds an optimal hyperplane that categorizes new examples. Tuning hyperparameters of an SVM model consists of the kernel choice, impact of regularization, gamma, and margin.

The kernel can be linear, polynomial, exponential, etc. The kernel type we considered in this work was linear. For the linear kernel, the prediction for input is calculated with

$$f(x)=B(0)+sum(ai*(x,xi)). \qquad (8)$$

It is obtained with the dot product of the input (x) and each of the support vectors (xi). It calculates the inner products of a new input vector (x) with all support vectors in training data. The coefficients $B(0)$ and ai (for each input) are estimated from the training data by the learning algorithm. The polynomial kernel and exponential kernel are given with

$$K(x,xi)=1+sum(x*xi)\hat{}\mathrm{d} \qquad (9)$$

$$K(x,xi)=\exp(-gamma*sum((x-x\hat{}2\ i)) \qquad (10)$$

The regularization parameter gives SVM optimization information to what extent to invalidate misclassification of every single training sample. The optimization with a large regularization parameter aims to achieve a better classification accuracy using a smaller-margin hyperplane. A small value of the regularization parameter will cause a larger-margin separating hyperplane, usually resulting in a reduced classification accuracy. The extent of influence of each training example is determined with the gamma parameter. Small values of gamma imply ‘far,’ and big values of gamma imply ‘close’.”

3.2 Problem Statement (Clear/Crisp)

Problem Statement: Developing a User-Friendly Web Interface for Accessible Remote Sensing Feature Extraction

Remote sensing technology offers invaluable insights into our planet's environment, climate, and urban development. However, the complex process of extracting meaningful information from satellite images often requires specialized knowledge and technical skills. This inaccessibility hinders the wider adoption of remote sensing by researchers, policymakers, and the general public.

The Challenge:

- *Technical Barriers:* The current state-of-the-art tools for feature extraction are often command-line based, requiring significant technical expertise.
- *Data Accessibility:* Remote sensing data is often stored in large, complex formats, making it difficult for non-experts to access and utilize.
- *Computational Resources:* Advanced feature extraction techniques demand substantial computational power, which can be a barrier for individuals and organizations with limited resources.
- *Lack of User-Friendly Interfaces:* Existing tools often lack intuitive user interfaces, making it challenging for non-technical users to explore and analyse satellite imagery.

The Solution:

To address these challenges, we propose the development of a user-friendly web interface that empowers users to:

- *Upload and Visualize Satellite Images:* Easily upload and visualize satellite images in various formats (e.g. JPG, PNG,JPEG).

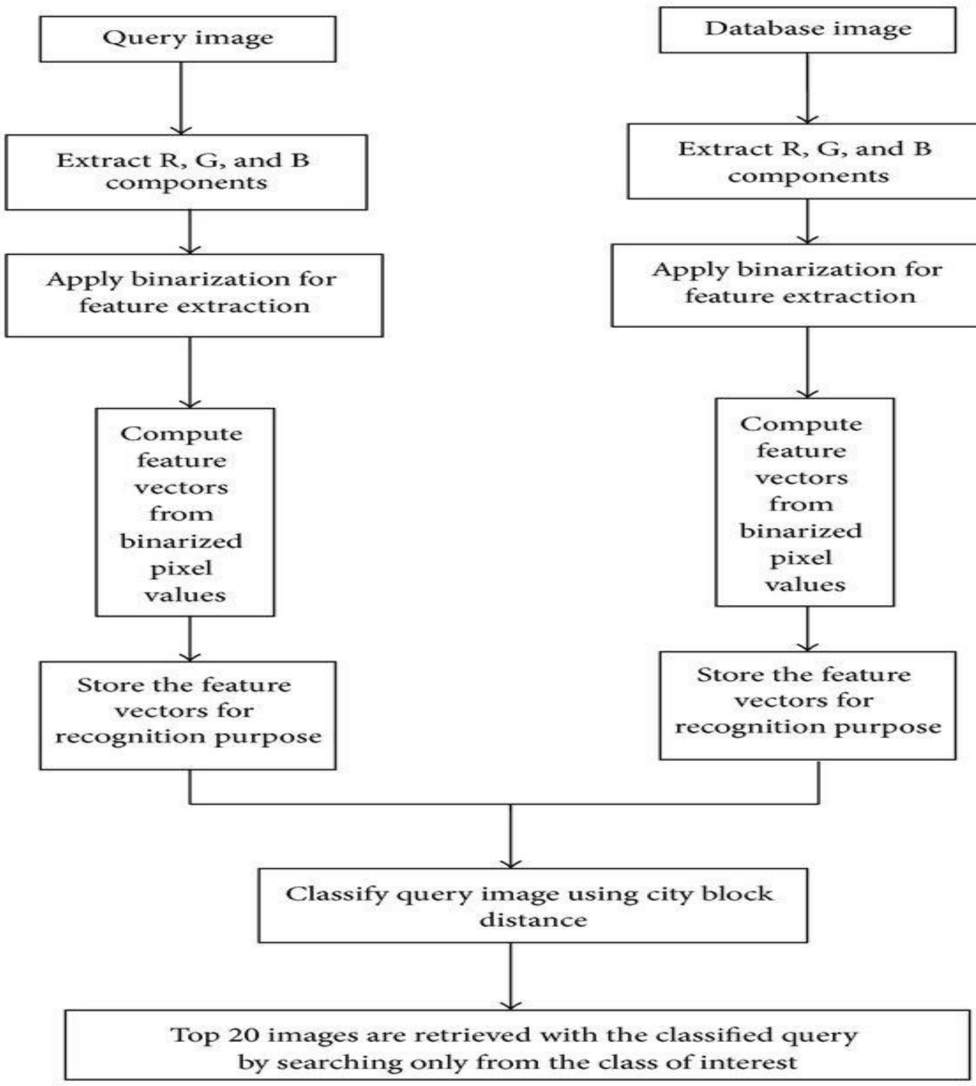
- *Perform Basic Image Processing:* Apply essential image processing techniques like cropping, zooming, and band combination.
- *Utilize Advanced Feature Extraction Algorithms:* Access a range of advanced feature extraction algorithms, including object-based image analysis, machine learning, and deep learning techniques.
- *Generate Meaningful Insights:* Visualize and analyse extracted features through interactive maps, charts, and reports.
- *Collaborate and Share Results:* Facilitate collaboration among users by enabling sharing of results and data.

By providing a user-friendly, accessible, and scalable web interface, we aim to democratize remote sensing, empowering a wider range of users to leverage the power of satellite imagery for various applications, such as:

- *Environmental Monitoring:* Tracking deforestation, land use change, and natural disasters.
- *Urban Planning:* Assessing urban growth, infrastructure development, and population dynamics.
- *Agriculture:* Monitoring crop health, yield prediction, and irrigation management.
- *Climate Change Research:* Analysing climate patterns, sea-level rise, and extreme weather events.

This web interface will not only simplify the complex process of feature extraction but also foster innovation and collaboration in the field of remote sensing. The system will serve as a bridge between complex geospatial analysis techniques and users who may lack the technical background to use conventional tools. By combining the power of machine learning with a user-centric design, the platform aims to Empower stakeholders such as environmental scientists, urban planners, agricultural experts, and disaster response teams. Accelerate decision-making by reducing the time required to analyse satellite data. Foster wider adoption of remote sensing tools across government, academic, and non-profit sectors.

3.3 Block diagram of feature extraction and recognition process.


Query image
Extract R, G, and B components
Apply binarization for feature extraction
Compute feature vectors from binarized pixel values
Store the feature vectors for recognition purpose
Database image
Extract R, G, and B components
Apply binarization for feature extraction
Compute feature vectors from binarized pixel values
Store the feature vectors for recognition purpose
Classify query image using city block distance
Top 20 images are retrieved with the classified query by searching only from the class of interest

3.4 Objectives

The primary objective of this project is to design and develop a user-friendly web-interface for feature extraction from remote sensing satellite images using Python. The specific objectives are:

1. To develop a web-based interface: Create a web-based interface using HTML, CSS and Javascript that allows users to upload their remote sensing satellite images and can get the glimpse of the content related to dataset and model.The

system will also display relevant metadata and model-related information, giving users a quick overview of the dataset structure, supported formats, and available analytical tools. The goal is to abstract technical complexity while ensuring smooth navigation and minimal learning curve, making the application accessible even to non-technical users.

2. To implement feature extraction algorithms: Implement feature extraction algorithms using Python libraries such as OpenCV, scikit-image, and GDAL to extract relevant features from the uploaded images. Specific techniques such as Gray-Level Co-occurrence Matrix (GLCM) for texture analysis, Sobel and Canny filters for edge detection, and band ratio methods for vegetation indices will be included. The backend will be designed to ensure modularity, allowing future integration of advanced algorithms or deep learning models.
3. To provide tools for image preprocessing: Develop tools for image preprocessing, including image filtering, normalization, and correction for atmospheric and geometric distortions. These corrections may involve radiometric calibration, cloud removal, and geometric realignment to ensure spatial accuracy. Preprocessing will ensure that the extracted features are reliable and consistent, especially when dealing with multi-temporal or high-resolution satellite images.
4. To enable feature visualization and analysis: Provide tools for visualizing and analysing the extracted features, including visualization of feature maps, scatter plots, and histograms. Libraries such as Matplotlib, Plotly, and Seaborn will be utilized to create interactive and informative visuals. These tools will help users interpret the extracted data and identify spatial or temporal patterns relevant to environmental monitoring, agriculture, or urban development.
5. To ensure accuracy and efficiency: Optimize the feature extraction algorithms to ensure accuracy and efficiency, and to minimize the processing time and computational resources required. Benchmarking against known datasets will help validate the accuracy of results, ensuring that the platform is suitable for

real-world applications without requiring high-performance computing environments.

6. To provide user-friendly interaction: Design an intuitive and user-friendly interface that allows users to easily upload images, select features, and visualize and analyse the extracted features. The goal is to remove the need for users to understand backend complexities like data formats or parameter tuning. Features such as drag-and-drop uploads, live progress indicators, and interactive result viewers will be implemented to provide a seamless and engaging experience.
7. To integrate with popular remote sensing libraries: Integrate the web-interface with popular remote sensing libraries such as GDAL, PySAR, and PyEO to enable seamless processing of remote sensing data. These integrations will allow the platform to process a broader range of data types and formats, as well as enable workflows that are compatible with external GIS and remote sensing pipelines, making it a versatile tool for researchers and practitioners.
8. Integration of Machine Learning Models: The project will integrate machine learning models to enhance accuracy and efficiency. These models will analyse large datasets, identify trends, and offer predictive analytics to aid decision-making in environmental monitoring and urban planning.

By achieving these objectives, the proposed system will provide a user-friendly and efficient platform for feature extraction from remote sensing satellite images, enabling non-experts to extract accurate features and make informed decisions.

CHAPTER 4

PROPOSED WORK

The proposed system will be developed using a combination of Python libraries and frameworks. The following methodology will be followed:

4.1 Introduction

4.1.1: Data Collection and Preprocessing

- Collect a dataset of remote sensing satellite images in various formats (e.g., GeoTIFF, JPEG, etc.)
- Preprocess the images using Python libraries such as OpenCV and scikit-image to correct for atmospheric and geometric distortions, and to enhance image quality
- Store the pre-processed images in a database or file system for later use.

4.1.2: Feature Extraction Algorithm Development

- Develop feature extraction algorithms using Python libraries such as OpenCV, scikit-image, and GDAL to extract relevant features from the preprocessed images
- Implement algorithms for extracting features such as texture, shape, and spectral characteristics
- Optimize the algorithms to ensure accuracy and efficiency.

4.1.3: Web-Interface Development

- Develop a web-based interface using Spring Boot that allows users to upload remote sensing satellite images and select specific features they want to extract using the U-Net based model.
- Design an intuitive and user-friendly interface using HTML, CSS, and JavaScript, enabling smooth interaction and visualization for the users.

- Implement user authentication and authorization using Spring Security to ensure secure and restricted access to the system and its functionalities.

4.1.4: Integration of Feature Extraction Algorithms with Web-Interface

- Integrate the feature extraction algorithms with the web interface using REST APIs developed in Spring Boot, handling requests and responses between the frontend and backend effectively.
- Develop APIs to enable communication between the web-interface and the feature extraction algorithms
- Implement error handling and debugging mechanisms to ensure robustness and reliability.

4.1.5: Feature Visualization and Analysis

- Develop tools for visualizing and analysing the extracted features using the JavaScript canvas library, enabling the creation of interactive visualizations like pie charts and bar charts within the web interface.
- Implement visualization tools for feature maps, scatter plots, and histograms
- Provide interactive visualization tools to enable users to explore and analyse the extracted features.

4.1.6: Testing and Evaluation

- Test the system using a dataset of remote sensing satellite images
- Evaluate the performance of the system using metrics such as accuracy, precision, and recall
- Conduct user testing to ensure the system meets the requirements and expectations of the users.

4.1.7: Deployment and Maintenance

- Deploy the system on a cloud-based platform or a local server
- Develop a maintenance plan to ensure the system remains up-to-date and secure
- Provide user support and training to ensure effective use of the system.

4.2 Proposed Methodology/Algorithm

Overview

This project presents a comprehensive web-based framework designed for the automated extraction and intuitive visualization of land features from remote sensing satellite imagery. The system utilizes a deep learning-based semantic segmentation approach, specifically leveraging the U-Net architecture, known for its efficiency in pixel-wise classification tasks and its strong performance on limited datasets. By analysing satellite images, the model generates precise segmentation masks that classify each pixel into one of six land feature categories: Water, Vegetation, Road, Building, Land, and Unlabelled.

To make this functionality accessible to users without technical expertise in geospatial analysis or remote sensing, a full-stack web application has been developed. The frontend is implemented using HTML, CSS, and JavaScript, offering an intuitive interface where users can upload satellite images, choose between available models or datasets, and instantly view results. The output includes both the segmented image mask and interactive visual analytics, such as real-time pie charts that represent the area-wise distribution of features and bar charts that display the pixel count per feature category. The backend of the system is powered by Spring Boot, which handles user authentication and authorization (via Spring Security), manages contact forms, and facilitates secure communication between the frontend and the model. RESTful APIs act as the bridge between the UI and the Python-based U-Net model, handling the image input, invoking model inference, and returning both the segmented output and corresponding feature statistics.

Overall, this system enables a **scalable, interactive, and user-centric** platform for land feature extraction, significantly improving upon traditional methods like NDWI (Normalized Difference Water Index), which were limited to detecting single features such as water bodies. By enabling **multi-class segmentation**, **real-time feedback**, and **seamless user interaction**, the project aims to democratize satellite image analysis for a broader range of users, including researchers, environmentalists, planners, and students.

Methodology

1. Dataset Description

To train and validate the proposed deep learning-based semantic segmentation model for land feature extraction, a combination of manually curated and publicly available datasets is utilized. These datasets capture diverse terrains and topologies to enable the model to generalize well across real-world geographical variations. The data preparation process is meticulously designed to ensure quality, uniformity, and suitability for training a supervised learning model.

4.2.1 Dataset:

The dataset is divided into two main groups—**one for training** the semantic segmentation model and **two others for testing** and validating its generalizability:

A. Training Dataset – Dubai Palm Beach Region

- The primary training dataset is curated from **satellite images of the Dubai Palm Beach region**, an area known for its varied land use including water bodies, man-made structures, vegetative areas, and bare land.
- The dataset contains **70 high-resolution RGB satellite images**, each accompanied by a **manually labeled semantic segmentation mask**.
- The segmentation mask for each image encodes **six land feature classes**:

    - Class 0: Water
    - Class 1: Vegetation
    - Class 2: Road
    - Class 3: Building
    - Class 4: Land (Bare Soil/Field)
    - Class 5: Unlabeled/Background

- The masks are encoded in categorical format, where each pixel belongs to exactly one of the six classes.
- This dataset serves as the **supervised learning foundation** for training the U-Net model to perform multi-class pixel-level classification.

B. Testing Dataset – Jewar Region and Hindon Region

- Jewar Region Dataset:
  Contains 25 satellite images from the Jewar area (Uttar Pradesh, India), which includes both urban and rural patterns.
  Serves as a mid-scale testbed to validate the performance of the model on unseen regional data.

- Hindon River Dataset:
  Comprises 5 satellite images capturing the Hindon river and surrounding regions.
  Chosen to evaluate the model's ability to detect water bodies, vegetation, and human settlements in riverine and semi-urban landscapes.

These datasets are not included during training and are strictly used for testing inference performance.

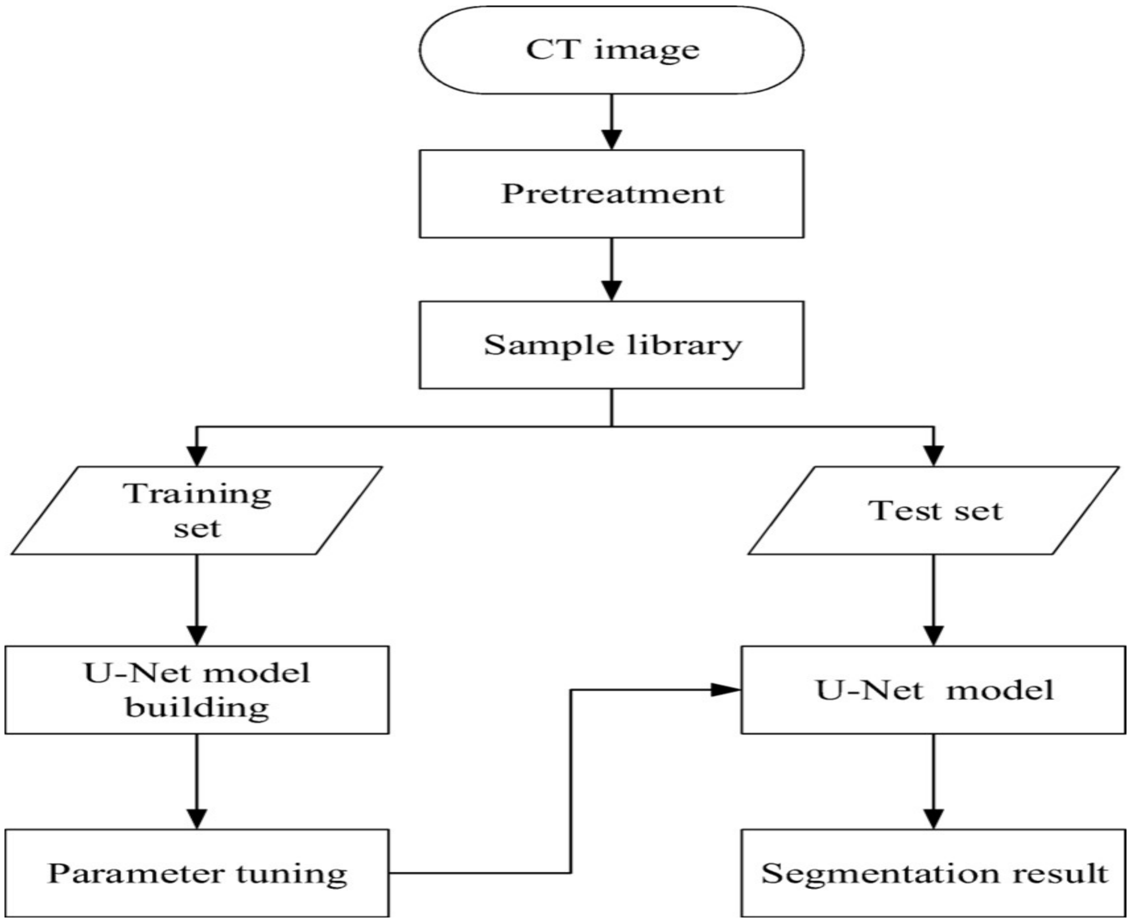


Figure 7: source Flowchart of U-Net Segmentation

*2.Image Pre-processing***:**

The input data undergoes a series of preprocessing steps to ensure it is compatible with the model architecture and capable of producing consistent learning outcomes. The preprocessing pipeline involves the following components:

2.1 Image Resizing:

- All input images and corresponding label masks are resized to a fixed spatial dimension of **m×m pixels**.
- This ensures uniform input shape required by the U-Net architecture and reduces computational load during training.

- Images retain their **three color channels (RGB)**, resulting in an input shape of **m×m×3**.

2.2 Normalization:

- Pixel values of each image are normalized by scaling them from the standard 8-bit range [0,255] [0, 255] [0,255] to a floating-point range [0.0,1.0][0.0, 1.0][0.0,1.0]
- This is a critical step for improving **gradient stability** during backpropagation and speeding up the convergence of the model.
- The normalization is applied independently across each channel (R, G, and B).

2.3. Data Augmentation: Given the relatively limited size of the training dataset (70 images), on-the-fly data augmentation is employed to artificially increase the diversity of training samples. This prevents overfitting and enhances the model's ability to generalize. Augmentation techniques include:

- **Horizontal and Vertical Flipping**:
  Helps in generalizing to images with different orientations.
- **Random Rotation**:
  Applied with angles up to ±30°, it helps simulate variations due to satellite tilt or geographic differences.
- **Random Cropping and Zooming**:
  Enables the model to focus on smaller regions and handle varying spatial resolutions.
- **Brightness and Contrast Adjustments** *(optional)*:
  Makes the model resilient to lighting differences due to time-of-day or atmospheric conditions.

2. *Segmentation Model: U-Net Architecture*

- The core of the system is the U-Net model, which is a fully convolutional neural network designed for semantic segmentation. It has an encoder-decoder structure:

- *Encoder (Contracting Path)***:** Composed of multiple convolutional layers (3×3 kernels), followed by ReLU activation and max-pooling (2×2) for down sampling. This captures spatial and contextual features.
- *Decoder (Expanding Path):* Uses up-convolutions (transposed convolutions) to up sample the feature maps. It includes skip connections from the encoder, which concatenate high-resolution features from the contracting path to preserve spatial information.
- *Output Layer:* A final 1×1 convolution layer maps the feature maps to the number of classes, followed by a SoftMax activation function to generate class probabilities for each pixel.

Mathematically, for an input image $X \in R^{H\times W\times 3}$ the U-Net learns a mapping:

$$f(X) \rightarrow Y \in R^{H\times W\times C}$$

Where C is the number of land feature classes. Training is performed using categorical cross-entropy loss:

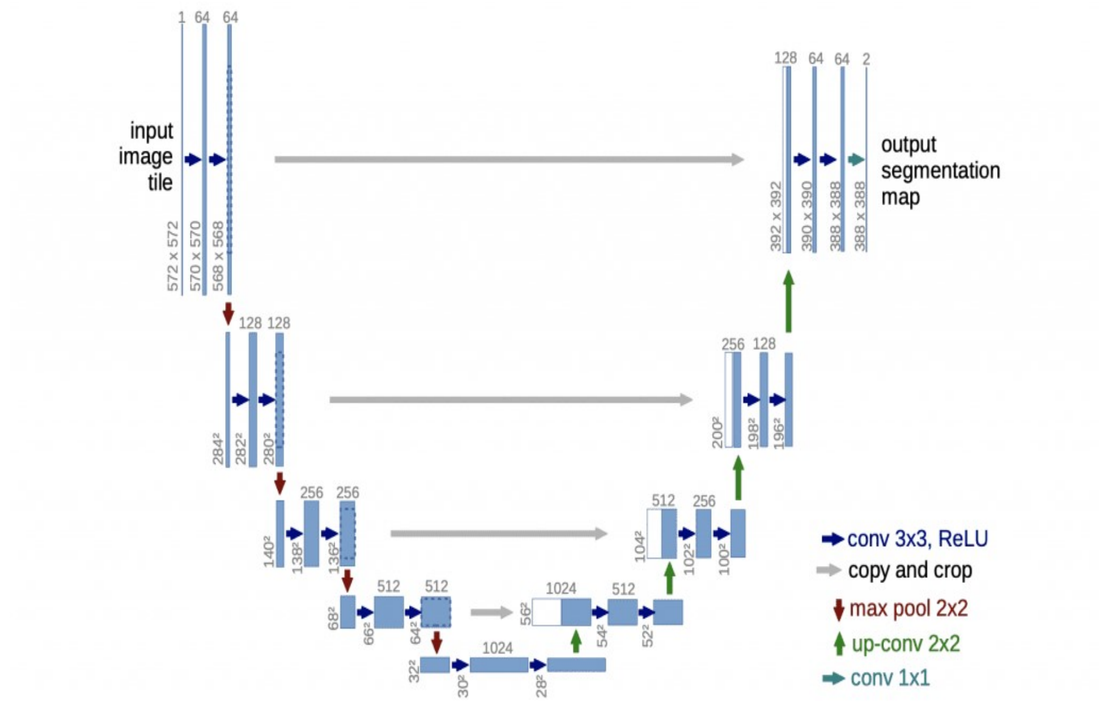
$$\mathcal{L}_{CE} = -\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\log(\hat{y}_{i,c})$$

Here, $y_{i,c}$ is the true label, and $y_{i,c}$ is the predicted probability for pixel i belonging to class C.

The U-Net model is trained using the **Dubai Palm Beach dataset**, where the input satellite images and their corresponding ground truth segmentation masks are fed into the network. The training process involves iterating over the dataset multiple times (epochs), where each epoch involves:

Forward Pass: The input image is passed through the encoder-decoder architecture to produce predicted segmentation masks. **Loss Calculation**: The loss function computes the difference between the predicted mask and the true label mask. **Backpropagation**: The error is propagated backward through the network, and the weights are updated using **stochastic gradient descent** or **Adam optimizer.**

Optimization: The network adjusts its parameters (weights) to minimize the



categorical cross-entropy loss, improving its segmentation performance.

Figure 8: source: U-Net architecture (image source: **U-Net paper**).

3. *Thresholding:*

   In traditional remote sensing methods, **thresholding** techniques like **NDWI (Normalized Difference Water Index)** are often employed to classify land features, such as water bodies. These approaches rely on predetermined threshold values to segment features based on pixel values. However, the major limitation of such methods is the **manual selection of threshold values**, which can be subjective and vary depending on environmental conditions, image resolution, and sensor characteristics.

   In contrast, the **U-Net model** used in this project leverages a more advanced, **data-driven approach** to segmentation, which does not require manual thresholding. Instead, U-Net generates **class-wise probability maps** for each pixel in the input image, where each pixel is classified into one of the six land feature classes (e.g., water, vegetation, road, building, land, or unlabelled).

$$\text{class}(x_i) = \arg\max_{c} \hat{y}_{i,c}$$

Class-Wise Probability Maps : After processing the input image through the **encoder-decoder architecture** of U-Net, the network outputs a **probability distribution** for each pixel across all land feature classes. This means that for every pixel (i,j)(i, j)(i,j) in the image, the model produces a vector of probabilities:

Probabilities for each pixel: [p1,p2,...,pC]

Where:

CCC is the total number of classes (6 land features: Water, Vegetation, Road, Building, Land, and Unlabeled).
pcp_cpc is the probability that pixel (i,j)(i, j)(i,j) belongs to class ccc.

These class-wise probabilities are generated by the SoftMax activation function applied to the final output layer, ensuring that the probabilities across all classes sum to 1 for each pixel.

Final Mask Generation (Argmax Operation)

To convert these probability maps into a segmentation mask, we employ the argmax operation over the class probabilities for each pixel. This step assigns each pixel to the class that has the highest probability.

The result is a segmentation mask Y∈RH×W×CY \{R}^{H \times W \times C}Y∈RH×W×C, where each pixel is assigned to one of the six classes. This mask is the final output of the U-Net model and represents the segmented land features in the image, with each class uniquely labelled.

4. *Post-processing:*

Post-processing enhances the quality of U-Net's segmentation results by refining boundaries and eliminating noise. This is achieved through morphological operations and small-object removal.

A. *Morphological Operations:*

Dilation: Expands features to fill small gaps and connect fragmented areas.
Erosion: Shrinks features, removing small noise elements.
Opening: Erosion followed by dilation, used to eliminate small objects while preserving larger ones.
Closing: Dilation followed by erosion, used to fill small holes within features.

B. *Small-Object Removal:*

Removes irrelevant, small objects that do not represent meaningful features (false positives or noise).
Uses size thresholding and connected component labeling to discard small regions and keep larger, relevant features.

4.3 Description of each step

Steps Involved in Satellite Image Segmentation using U-Net, Spring Boot Backend, and JavaScript Frontend

1. *Data Acquisition and Preprocessing:*

- **Acquire Satellite Images**: The first step involves collecting satellite images from three sources:
    - **Jewar Region (self-collected)**: These images come from our custom dataset based on the Jewar region.
    - **Dubai Palm Beach**: This dataset consists of images collected from the Dubai Palm Beach region.
    - **Hindon Dataset**: Public satellite imagery datasets from Kaggle are also used.
- Supported image formats include PNG, JPEG, or TIFF.
- **Resize and Normalize**:
    - All images are resized to a consistent input shape (e.g., 256×256×3 or 512×512) to ensure they match the U-Net model's input size.
    - **Normalization**: Pixel values are scaled to the range [0, 1] for better consistency and training performance.
- **Image Cleaning**:
    - If required, pre-processing techniques such as noise reduction (e.g., applying Gaussian filters) and contrast enhancement are used to improve the input image quality. This is crucial for enhancing the model's ability to segment features.

2. *Feature Segmentation using U-Net:*

- Run U-Net Model: Feed the pre-processed image into a pre-trained U-Net deep learning model to segment multiple features (like water bodies, vegetation, roads, and buildings).
- Generate Segmentation Mask: The model outputs a pixel-wise mask where each pixel is assigned a class label representing a specific land feature.
- Color Mapping: Convert the class-wise segmented output into a color-encoded mask for visualization (e.g., water = blue, vegetation = green, roads = gray, buildings = red).

3. *Backend Development (Spring Boot + Python Model API):*

- Spring Boot Setup: Spring Boot handles user authentication, contact service, and connection between frontend and backend via REST APIs.
- Image Upload API: Users upload satellite images from the frontend. The image is sent from Spring Boot to the Python API endpoint handling U-Net inference.
- Python U-Net Processing**:** The image is processed by a Flask/FastAPI-based Python service that loads the U-Net model, performs prediction, and returns a base64-encoded mask.
- Result Handling: Spring Boot receives the mask from Python, and returns it to the frontend for rendering.

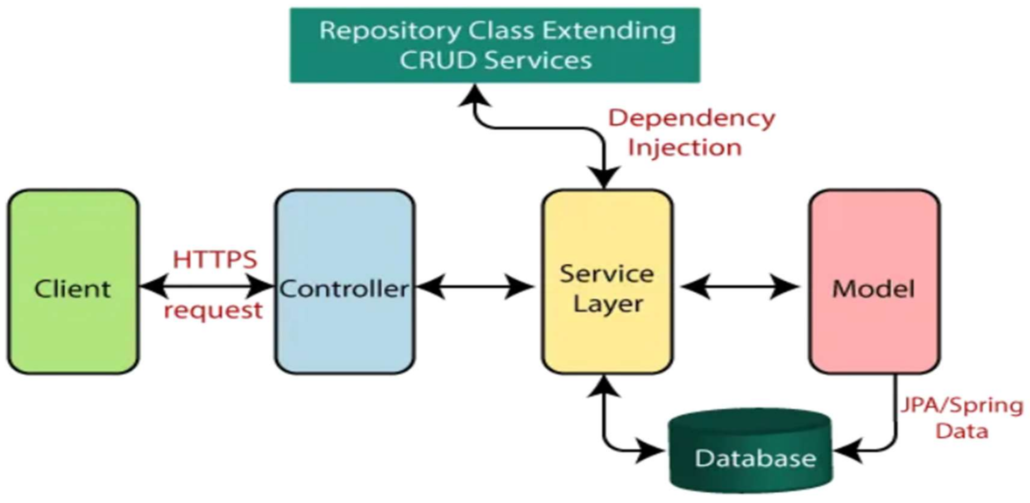


Figure 9: source: SpringBoot Medium Architecture

4. *Frontend Development (HTML/CSS/JavaScript):*

i) *User Interface Design:*

- *Image Upload Component:* Implement an easy-to-use upload section for users to select and upload satellite images using an HTML file input.
- *Progress Bar:* Display a progress bar during image processing, dynamically updating via JavaScript as the backend processes the image.
- *Image Display Area:* Show the original uploaded image and overlay the full feature segmentation mask once processing is complete.
- *Feature Mask Overlay:* Display the segmented features (water, vegetation, roads, buildings, etc.) with color-coded overlays for each class.
- *Download Button:* Allow users to download both the processed image (with all feature masks overlaid) and the segmentation mask.

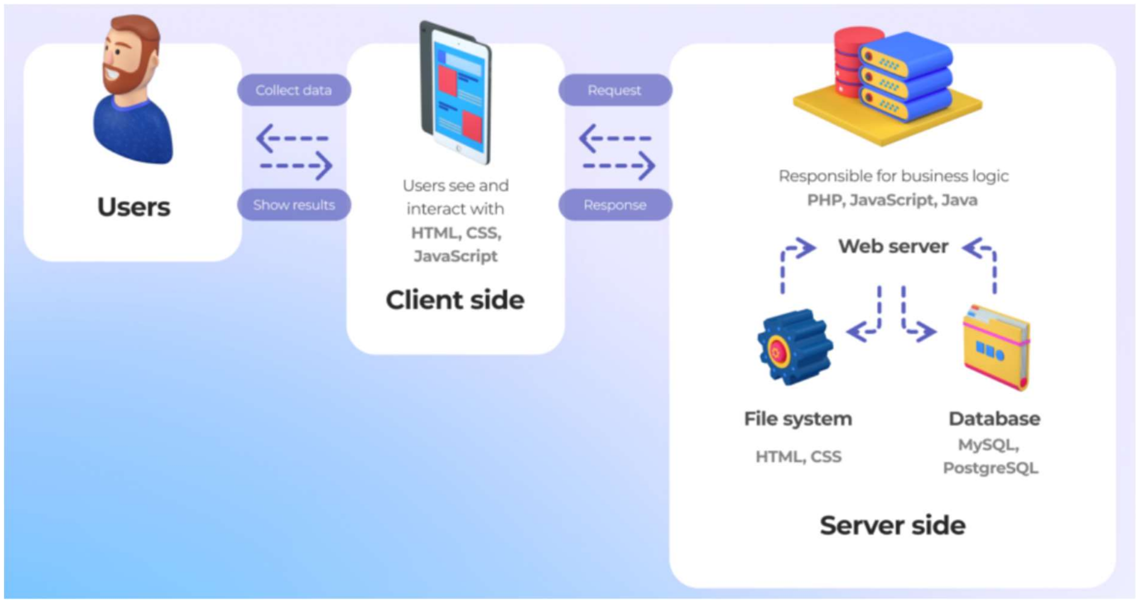


Figure 10: source: https://themekraft.com/wp-content/uploads/2021/06/image-1024x716-1.png

ii) *Image Upload Mechanism:* Use an HTML form to upload the image as multipart form-data to the Spring Boot backend for processing.

iii) *API Call to Flask Backend:* The Spring Boot backend forwards the image to the Python Flask API, where the U-Net model processes it. JavaScript will send the image to the backend using Fetch API or Axios.

iv) *Result Display:* Once processed, display the original image with the water mask overlay. Additionally, generate a pie chart and bar chart using JavaScript Canvas to visualize feature percentages and pixel counts.

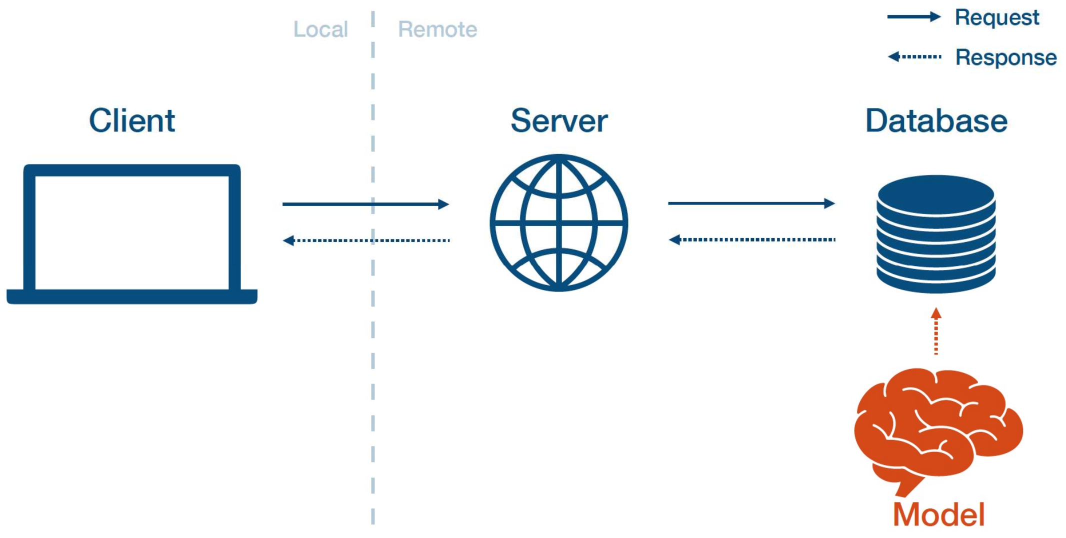
v) *Download Functionality:* Provide download options for the processed image and the segmentation mask once processing is finished. Use JavaScript for triggering the download in base64 format.

5. Deployment:

Deploy Backend: Deploy the backend built with Flask and Spring Boot to a cloud platform such as AWS, Heroku, or Render. This will handle image processing, feature extraction using the U-Net model, and provide API services for user authentication and managing requests from the frontend.

Deploy Frontend: Deploy the frontend, created with HTML, CSS, and JavaScript, on platforms like Netlify or Vercel for easy access and fast delivery. This will allow users to interact with the system, upload satellite images, and view the results.

Database Integration: Use the database (containing images from sources like the Jewar area, Dubai Palm Beach, and Kaggle satellite datasets) to store images, metadata, and processed results. This can be hosted on cloud services or use a cloud-based database like MongoDB Atlas for easy scaling



source : https://fullstackdeeplearning.com/spring2021/lecture-11-notes-media/image16.png

Additional Considerations:

Multiple Spectral Indices: Incorporate multiple spectral indices (e.g., MNDWI, AWEI) to improve the extraction accuracy, especially in complex environments where the feature differentiation may be challenging.

Machine Learning: Leverage the U-Net deep learning model to segment and extract various features (like water bodies, buildings, vegetation, etc.) from satellite images. This model will be integrated into the backend to process images and extract features.

User Experience: Design an intuitive and responsive user interface with HTML, CSS, JavaScript, and JavaScript Canvas to visualize the extraction results in real-time. Users will be able to see feature overlays, charts (like pie and bar), and download processed images.

Error Handling: Implement robust error handling for image uploads, API requests, and backend processes. If any issues occur, the system should provide helpful error messages and feedback to guide the user.

Scalability: Optimize the backend and frontend to ensure scalability, using cloud infrastructure to handle high user traffic and large image datasets. This includes setting up auto-scaling for the backend services (Flask and Spring Boot) and optimizing the frontend for performance.

By leveraging Flask, Spring Boot, JavaScript Canvas, and the U-Net deep learning model, the system will be able to process satellite imagery efficiently, while providing a user-friendly experience for multi-feature extraction.

CHAPTER 5

SYSTEM DESIGN

5.1 Functional Specification of System

Components of the Level 0 DFD

- *Process:*
    - Web-Interface for Feature Extraction
- *External Entities:*
    - *Users:* Researchers or analysts who will utilize the web interface to extract features from satellite images.
    - *Remote Sensing Satellite:* The source of satellite images that will be processed.
    - *External Databases:* Repositories for storing and retrieving metadata and processed results.
- *Data Flows:*
    - *Input Data:* Satellite images from the Remote Sensing Satellite.
    - *User Queries:* Requests from users to extract specific features or data.
    - *Output Data:* Processed feature extraction results sent back to users or stored in External Databases.

Level 1 Data Flow Diagram

The Level 1 DFD breaks down the main process identified in Level 0 into sub-processes, providing more detail about how data flows through the system.

Components of the Level 1 DFD

1. *Sub-Processes:*
    - Image Uploading

- Feature Extraction
- Results Visualization
- User Management

2. *Data Stores:*
   - Image Database: Stores uploaded satellite images.
   - Feature Database: Stores extracted features and results.
   - User Database: Contains user profiles and access rights.
3. *External Entities:*
   - Same as in Level 0.
4. *Data Flows:*
   - Users upload satellite images to the Image Database.
   - User queries are processed in the Feature Extraction sub-process.
   - Extracted features are stored in the Feature Database.
   - Visualization requests lead to output being displayed to users.

Description of Sub-Processes

- *Image Uploading:* Users can upload satellite images through a web interface. This process validates and stores images in the Image Database.
- *Feature Extraction:* This core process analyses uploaded images to identify and extract relevant features based on user-defined criteria. The results are then stored in the Feature Database.
- *Results Visualization:* After feature extraction, this process generates visual representations of the extracted features for user analysis and interpretation.
- *User Management:* Handles user authentication, authorization, and profile management to ensure secure access to the system's functionalities.

5.2 Detail Block/ Diagram/ Flowchart

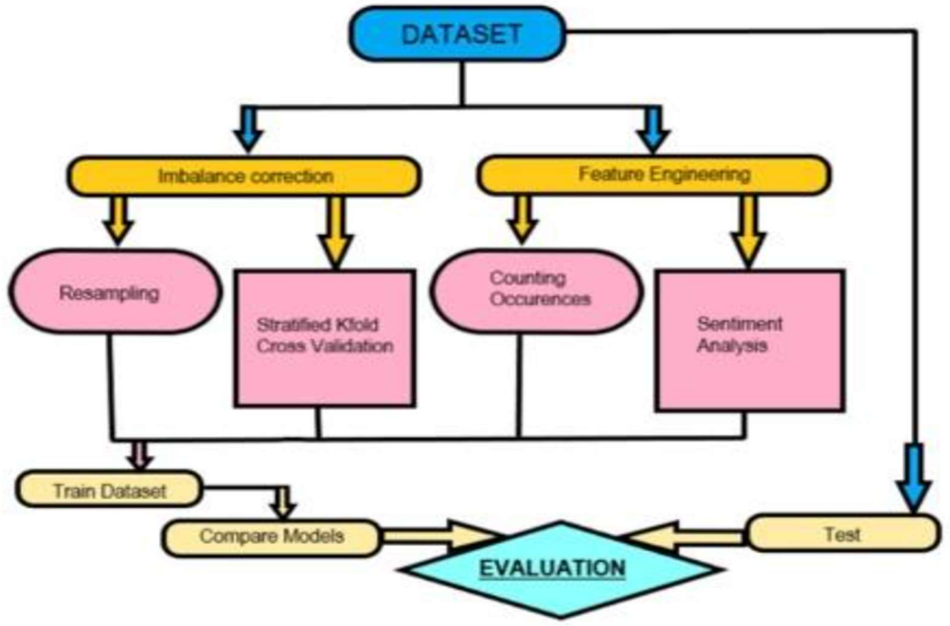


Diagram -1

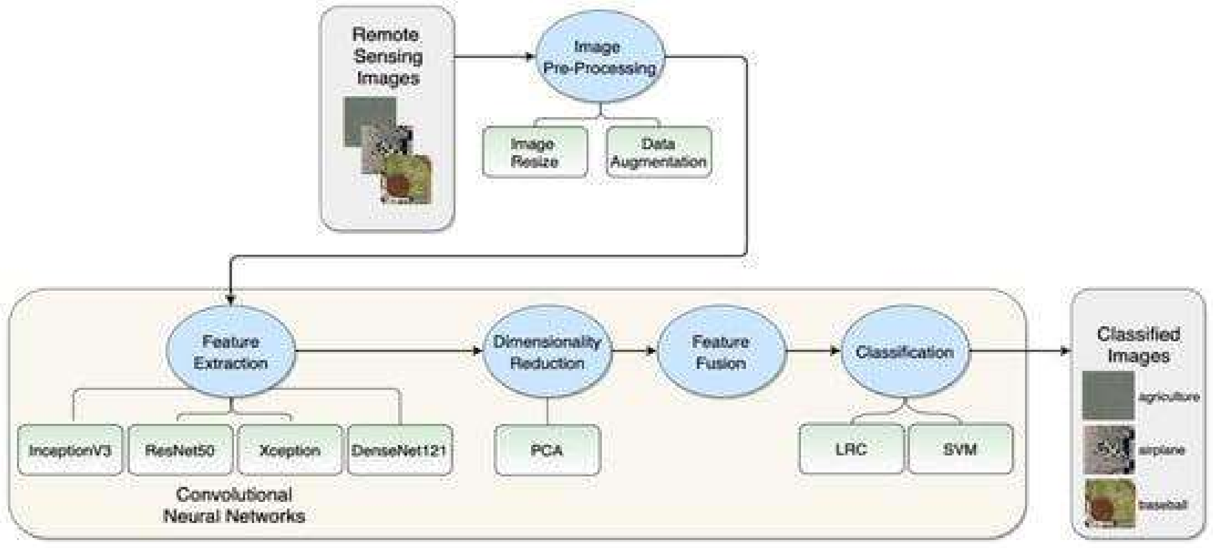
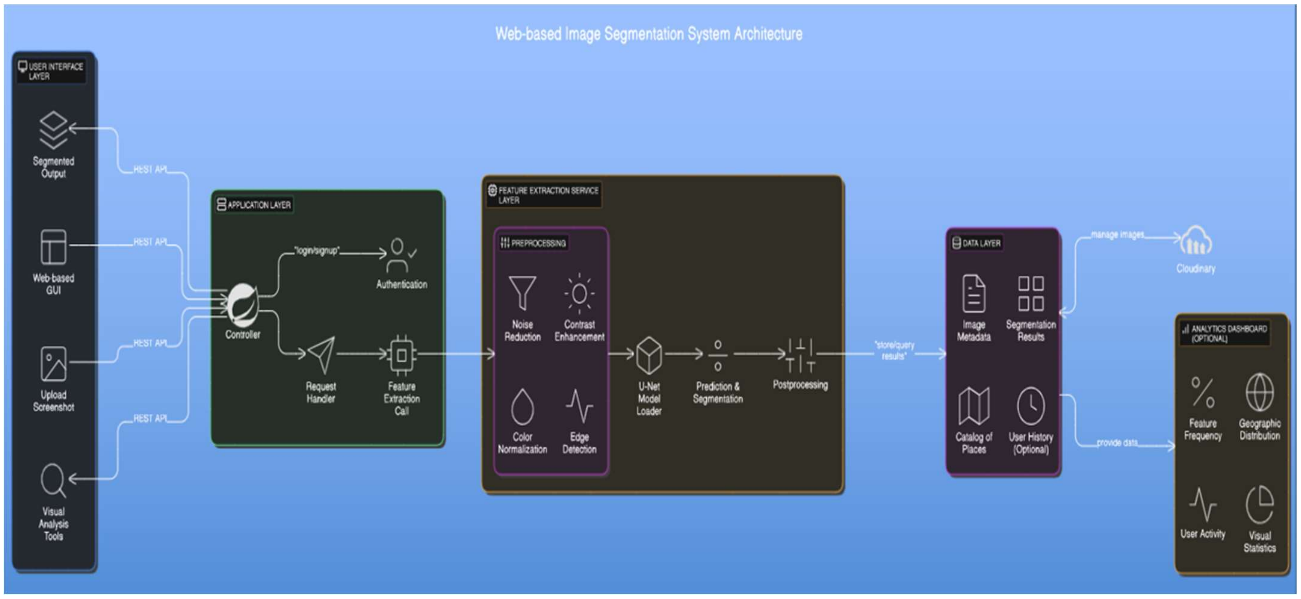


Diagram - 2



System design diagram

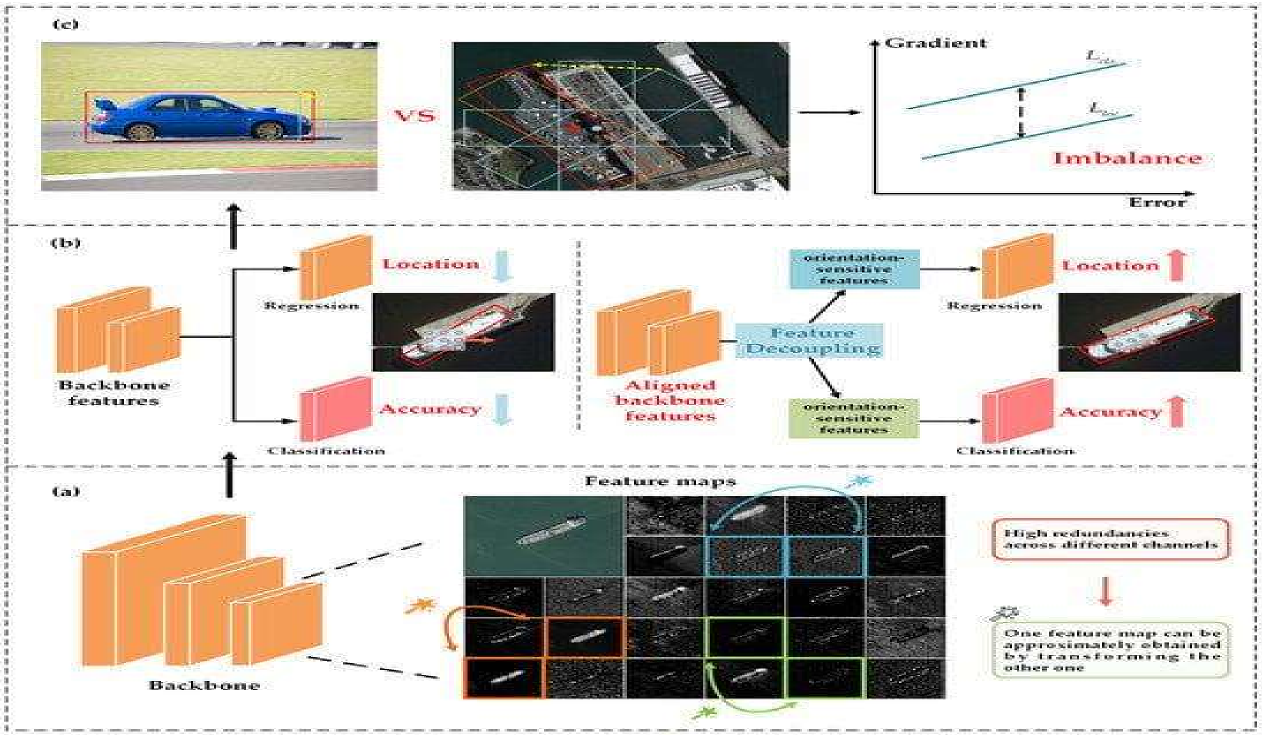

(c)
VS
Gradient
Error
Imbalance
(b)
Backbone features
Regression
Location
Classification
Accuracy
Aligned backbone features
Feature Decoupling
orientation-sensitive features
orientation-sensitive features
Regression
Location
Classification
Accuracy
(a)
Backbone
Feature maps
High redundancies across different channels
One feature map can be approximately obtained by transforming the other one

Diagram - 3

CHAPTER 6

IMPLEMENTATION

6.1 Experimental Setup

1. Data Acquisition:

1.1 Dataset Description:

- Satellite Imagery: A collection of satellite images (e.g., Landsat 8, Sentinel-2) covering diverse geographical regions with varying characteristics. Sentinel-2 is a European Space Agency (ESA) mission designed to provide high-resolution multispectral imagery of Earth's surface. It offers a wide swath width and a high revisit time, enabling frequent monitoring of land and coastal areas.

  Key Characteristics:

- High Spatial Resolution: The Multispectral Instrument (MSI) onboard Sentinel-2 captures images at 10-meter, 20-meter, and 60-meter spatial resolutions.
- Multispectral Bands: It provides data in 13 spectral bands, covering the visible, near-infrared, and shortwave infrared regions of the electromagnetic spectrum.
- Frequent Revisit Time: With two satellites in orbit, Sentinel-2 can revisit the same location every 5 days, allowing for timely monitoring of dynamic processes.
- Global Coverage: The mission provides global coverage, making it suitable for various applications across the globe.

1.2 Training and Test Dataset:

The dataset is divided into two main groups—**one for training** the semantic segmentation model and **two others for testing** and validating its generalizability:

A. Training Dataset – Dubai Palm Beach Region

The primary training dataset is curated from satellite images of the Dubai Palm Beach region, an area known for its varied land use including water bodies, man-made structures, vegetative areas, and bare land.

The dataset contains 70 high-resolution RGB satellite images, each accompanied by a manually labelled semantic segmentation mask.

The segmentation mask for each image encodes six land feature classes:

- Class 0: Water
- Class 1: Vegetation
- Class 2: Road
- Class 3: Building
- Class 4: Land (Bare Soil/Field)
- Class 5: Unlabelled/Background

B. Testing Dataset – Jewar Region and Hindon Region

- **Jewar Region Dataset**:
  Contains **25 satellite images** from the Jewar area (Uttar Pradesh, India), which includes both urban and rural patterns.
  Serves as a **mid-scale testbed** to validate the performance of the model on unseen regional data.

- **Hindon River Dataset**:
  Comprises **5 satellite images** capturing the Hindon river and surrounding regions.
  Chosen to evaluate the model's ability to detect water bodies, vegetation, and human settlements in **riverine and semi-urban landscapes**.

  These datasets are not included during training and are strictly used for **testing inference performance**.

1.3 Dataset Color Masking:

```
<section>
<div class="class-container">
    <div class="dataset-label">Dubai Data Set:</div>
    <div class="class-row-horizontal">
      <span class="class-color" style="background-color: #e2a929;"></span><span class="class-
name">Water</span>
      <span class="class-color" style="background-color: #8429f6;"></span><span class="class-
name">Land</span>
      <span class="class-color" style="background-color: #6ec1e4;"></span><span class="class-
name">Road</span>
      <span class="class-color" style="background-color: #390799;"></span><span class="class-
name">Building</span>
      <span class="class-color" style="background-color: #fedd3a;"></span><span class="class-
name">Vegetation</span>
      <span class="class-color" style="background-color: #9B9B9B;"></span><span class="class-
name">Unlabeled</span>
    </div>

    <div class="dataset-label">Test Data Set:</div>
    <div class="class-row-horizontal">
      <span class="class-color" style="background-color: #00008b;"></span><span class="class-
name">Water</span>
      <span class="class-color" style="background-color: black;"></span><span class="class-
name">Land</span>
      <span class="class-color" style="background-color: #00ffff;"></span><span class="class-
name">Road</span>
      <span class="class-color" style="background-color: #f3d43c;"></span><span class="class-
name">Building</span>
      <span class="class-color" style="background-color: #00ff00;"></span><span class="class-
name">Vegetation</span>
      <span class="class-color" style="background-color: #9B9B9B;"></span><span class="class-
name">Unlabeled</span>
    </div>
   </div>
  </section>
```

2. Preprocessing Steps:

1. *Radiometric Correction:* Apply atmospheric correction techniques (e.g., Dark Object Subtraction) to remove atmospheric effects and ensure accurate reflectance values.
2. *Geometric Correction:* Georeference the images to a specific coordinate system (e.g., WGS84) to align them spatially.
3. *Cloud Masking:* Identify and remove cloud-covered areas to avoid errors in subsequent analysis.
4. *Input Channel Preparation:* Prepare the satellite imagery as multi-channel input (e.g., RGB or RGB+NIR) for the U-Net model. Each channel represents a band from the satellite image and is normalized to enhance model performance. The model automatically learns relevant features across these channels during training.

5. *Dataset preprocessing training :*

```
image_dataset = []
mask_dataset = []


for image_type in ['images' , 'masks']:
  if image_type == 'images':
    image_extension = 'jpg'
  elif image_type == 'masks':
     image_extension = 'png'
  for tile_id in range(1,8):
    for image_id in range(1,20):
      image = cv2.imread(f'{dataset_root_folder}/{dataset_name}/Tile
{tile_id}/{image_type}/image_part_00{image_id}.{image_extension}',1)
      if image is not None:
        if image_type == 'masks':
          image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        #print(image.shape)
        size_x = (image.shape[1]//image_patch_size)*image_patch_size
        size_y = (image.shape[0]//image_patch_size)*image_patch_size
        #print("{} --- {} - {}".format(image.shape, size_x, size_y))
        image = Image.fromarray(image)
        image = image.crop((0,0, size_x, size_y))
        #print("({},  {})".format(image.size[0],image.size[1]))
        image = np.array(image)
 patched_images = patchify(image, (image_patch_size, image_patch_size, 3),
step=image_patch_size)
        #print(len(patched_images))
        for i in range(patched_images.shape[0]):
          for j in range(patched_images.shape[1]):
            if image_type == 'images':
              individual_patched_image = patched_images[i,j,:,:]
              #print(individual_patched_image.shape)
  individual_patched_image =
minmaxscaler.fit_transform(individual_patched_image.reshape(-1,
individual_patched_image.shape[-1])).reshape(individual_patched_image.shape)
              individual_patched_image = individual_patched_image[0]
              #print(individual_patched_image.shape)
              image_dataset.append(individual_patched_image)
            elif image_type == 'masks':
              individual_patched_mask = patched_images[i,j,:,:]
              individual_patched_mask = individual_patched_mask[0]
              mask_dataset.append(individual_patched_mask)
```

3. Feature Segmentation and Thresholding

3.1 Feature Segmentation:

- The input satellite image is first uploaded through the frontend interface built with HTML, CSS, and JavaScript.
- Once uploaded, the image is sent to the Spring Boot backend, where it is processed by a pre-trained U-Net model.
- The core of the system is the U-Net model, which is a fully convolutional neural network designed for semantic segmentation. It has an encoder-decoder structure:

  *Encoder (Contracting Path)***:** Composed of multiple convolutional layers (3×3 kernels), followed by ReLU activation and max-pooling (2×2) for downsampling. This captures spatial and contextual features.

  Decoder (Expanding Path): Uses up-convolutions (transposed convolutions) to upsample the feature maps. It includes skip connections from the encoder, which concatenate high-resolution features from the contracting path to preserve spatial information.

  Output Layer: A final 1×1 convolution layer maps the feature maps to the number of classes, followed by a softmax activation function to generate class probabilities for each pixel.

- The model predicts a segmentation mask using a softmax activation function:

$$P(class_i|pixel) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

  where ziz_izi is the logit score for class iii for each pixel.

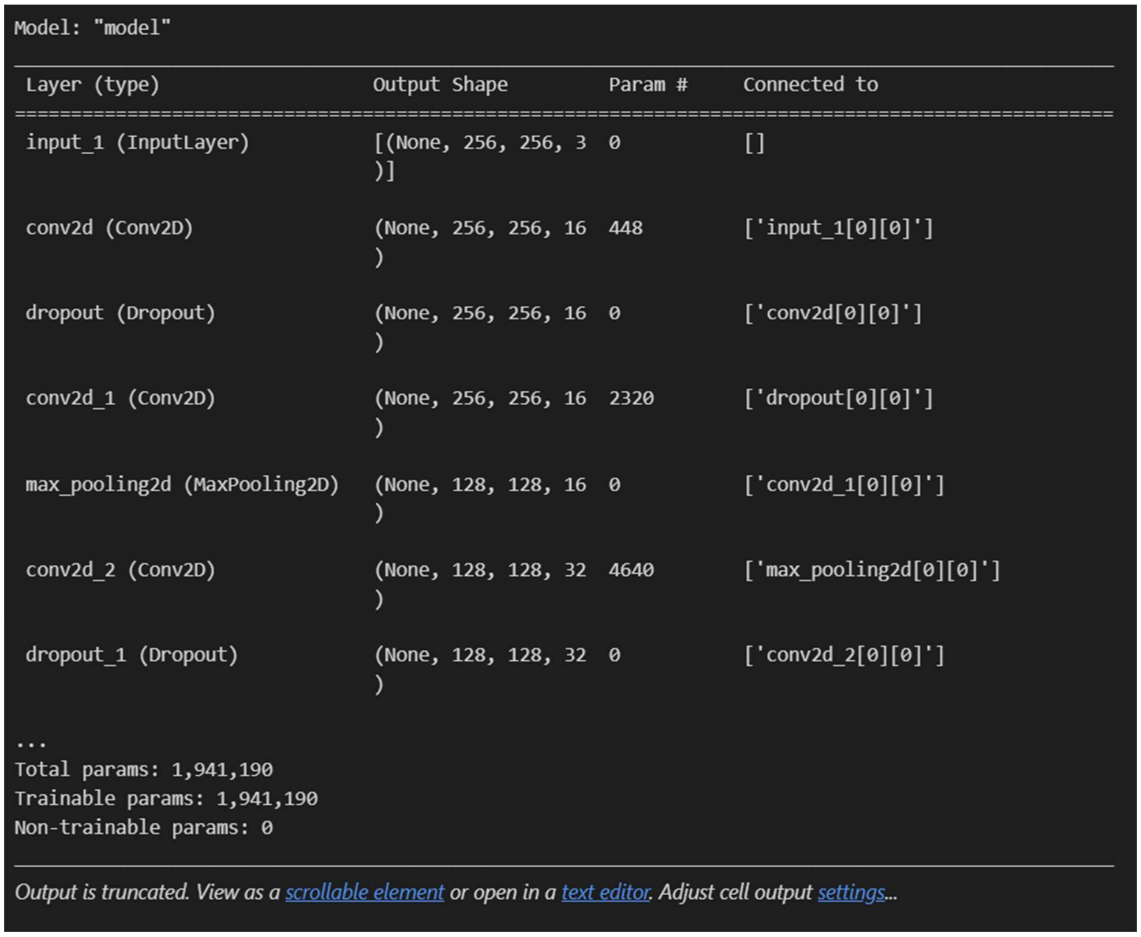
  The output is a 2D mask where each pixel value corresponds to the most likely feature class (e.g., object, background, etc.).

The mask is generated using a **cross-entropy loss** during training:

$$\text{Loss} = -\sum_{i=1}^{C} y_i \log(p_i)$$

where C is the number of classes, y_i is the true label, and p_i is the predicted probability for class i.

6.3 Model Configuration:



```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 256, 256, 3  0           []
                                )]

 conv2d (Conv2D)                (None, 256, 256, 16  448         ['input_1[0][0]']
                                )

 dropout (Dropout)              (None, 256, 256, 16  0           ['conv2d[0][0]']
                                )

 conv2d_1 (Conv2D)              (None, 256, 256, 16  2320        ['dropout[0][0]']
                                )

 max_pooling2d (MaxPooling2D)   (None, 128, 128, 16  0           ['conv2d_1[0][0]']
                                )

 conv2d_2 (Conv2D)              (None, 128, 128, 32  4640        ['max_pooling2d[0][0]']
                                )

 dropout_1 (Dropout)            (None, 128, 128, 32  0           ['conv2d_2[0][0]']
                                )

...
Total params: 1,941,190
Trainable params: 1,941,190
Non-trainable params: 0
__________________________________________________________________________________________________
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

3.3 Thresholding:

- To filter out uncertain predictions, a **confidence threshold** is applied to the predicted mask.

- Any pixel with a predicted class probability below a defined threshold (e.g., 0.5) is treated as background or ignored.
- Post-processing operations are applied to refine the mask:
- Binary thresholding:

$$mask(x,y) = \begin{cases} 1 & \text{if } P(x,y) > T \\ 0 & \text{otherwise} \end{cases}$$

  where TTT is the confidence threshold.

- Morphological operations such as erosion and dilation help remove small artifacts and smooth boundaries.

4. Feature Extraction :

4.1 Feature Extraction:

Pass the pre-processed satellite image through the trained U-Net model to generate a segmentation mask that identifies various land cover types such as water bodies, vegetation, road, buildings, land cover, Unlabelled data.
Apply morphological operations (e.g., erosion and dilation) to smooth boundaries and remove small artifacts or noise from the segmentation output.

The segmentation mask for each image encodes **six land feature classes**:

  - Class 0: Water
  - Class 1: Vegetation
  - Class 2: Road
  - Class 3: Building
  - Class 4: Land (Bare Soil/Field)
  - Class 5: Unlabeled/Background

Image mask containing all above six classes is generated according to color coding given in 2.3 dataset coloring.

4.2 U-Net Model:

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Conv2DTranspose, concatenate,
Dropout


def multiclass_unet_model(n_classes=5, image_height=256, image_width=256, image_channels=1):
    # Input layer
    inputs = Input((image_height, image_width, image_channels))


    # Encoder path (Downsampling)
    c1 = Conv2D(16, (3, 3), activation="relu", kernel_initializer="he_normal", padding="same")(inputs)
    c1 = Dropout(0.2)(c1)
    c1 = Conv2D(16, (3, 3), activation="relu", kernel_initializer="he_normal", padding="same")(c1)
    p1 = MaxPooling2D((2, 2))(c1)


    c2 = Conv2D(32, (3, 3), activation="relu", kernel_initializer="he_normal", padding="same")(p1)
    c2 = Dropout(0.2)(c2)
    c2 = Conv2D(32, (3, 3), activation="relu", kernel_initializer="he_normal", padding="same")(c2)
    p2 = MaxPooling2D((2, 2))(c2)


    c3 = Conv2D(64, (3, 3), activation="relu", kernel_initializer="he_normal", padding="same")(p2)
    c3 = Dropout(0.2)(c3)
    c3 = Conv2D(64, (3, 3), activation="relu", kernel_initializer="he_normal", padding="same")(c3)
    p3 = MaxPooling2D((2, 2))(c3)


    c4 = Conv2D(128, (3, 3), activation="relu", kernel_initializer="he_normal", padding="same")(p3)
    c4 = Dropout(0.2)(c4)
    c4 = Conv2D(128, (3, 3), activation="relu", kernel_initializer="he_normal", padding="same")(c4)
    p4 = MaxPooling2D((2, 2))(c4)


    # Bottleneck
```

```
c5 = Conv2D(256, (3, 3), activation="relu", kernel_initializer="he_normal", padding="same")(p4)
c5 = Dropout(0.2)(c5)
c5 = Conv2D(256, (3, 3), activation="relu", kernel_initializer="he_normal", padding="same")(c5)

# Decoder path (Upsampling)
u6 = Conv2DTranspose(128, (2, 2), strides=(2, 2), padding="same")(c5)
u6 = concatenate([u6, c4])
c6 = Conv2D(128, (3, 3), activation="relu", kernel_initializer="he_normal", padding="same")(u6)
c6 = Dropout(0.2)(c6)
c6 = Conv2D(128, (3, 3), activation="relu", kernel_initializer="he_normal", padding="same")(c6)

u7 = Conv2DTranspose(64, (2, 2), strides=(2, 2), padding="same")(c6)
u7 = concatenate([u7, c3])
c7 = Conv2D(64, (3, 3), activation="relu", kernel_initializer="he_normal", padding="same")(u7)
c7 = Dropout(0.2)(c7)
c7 = Conv2D(64, (3, 3), activation="relu", kernel_initializer="he_normal", padding="same")(c7)

u8 = Conv2DTranspose(32, (2, 2), strides=(2, 2), padding="same")(c7)
u8 = concatenate([u8, c2])
c8 = Conv2D(32, (3, 3), activation="relu", kernel_initializer="he_normal", padding="same")(u8)
c8 = Dropout(0.2)(c8)
c8 = Conv2D(32, (3, 3), activation="relu", kernel_initializer="he_normal", padding="same")(c8)

u9 = Conv2DTranspose(16, (2, 2), strides=(2, 2), padding="same")(c8)
u9 = concatenate([u9, c1])
c9 = Conv2D(16, (3, 3), activation="relu", kernel_initializer="he_normal", padding="same")(u9)
c9 = Dropout(0.2)(c9)
c9 = Conv2D(16, (3, 3), activation="relu", kernel_initializer="he_normal", padding="same")(c9)
```

```
# Output layer with softmax for multi-class segmentation

outputs = Conv2D(n_classes, (1, 1), activation="softmax")(c9)

# Build and return the model

model = Model(inputs=[inputs], outputs=[outputs])

return model
```

5. Image segmentation Visualization:

After generating the segmented mask through the U-Net model, each pixel is categorized into one of the predefined classes. The mask is then analysed in real-time to count the number of pixels belonging to each class.

From this data, the percentage area of each class over the total image is calculated. These proportions are then used to create visual representations such as:

- Pie Chart : Helps in visualizing how much area each class occupies relative to the total.
- Bar Chart : Useful to compare absolute pixel counts among different classes side by side.
- Pie and bar charts are generated dynamically using **JavaScript's Canvas** API combined with Chart.js.

These visualizations enable quick decision-making for applications in agriculture, urban planning, and disaster management.

```
pieChartInstance = new Chart(pieCanvas.getContext('2d'), {

      type: 'pie',

      data: {

        labels: labels.map((l, idx) => `${l} - ${percentages[idx]}%`),

        datasets: [{

          data: counts,

          backgroundColor: colors,

          borderColor: '#fff',

          borderWidth: 2

        }]

      },
  options: { responsive: false }
 });
 barChartInstance = new Chart(barCanvas.getContext('2d'), {
  type: 'bar',
  data: {
   labels: labels,
   datasets: [{
    label: 'Pixel Count',
    data: counts,
    backgroundColor: colors,
    borderColor: '#333',
    borderWidth: 1
   }]
  },
```

6. Backend Development (Flask API):

Flask App (U-Net Inference Service)

- Hosts the **U-Net model** for semantic segmentation.
- Exposes RESTful endpoints to accept image data and return segmented masks.
- Generates **masks mapped to predefined dataset classes** (e.g., Water, Urban, Vegetation).
- Returns **base64-encoded** images to ensure smooth JSON communication.

Flask Setup

```
from flask import Flask, request, jsonify
import base64
import io
app = Flask(__name__)
```

```
@app.route('/predict', methods=['POST'])
def predict():
    file = request.files['image']
    image = preprocess(file)
    mask = unet_model.predict(image)
    encoded_mask = encode_to_base64(mask)
    return jsonify({'mask': encoded_mask})
```

- preprocess(file): Normalizes and resizes the input image.
- unet_model.predict(image): Generates segmentation mask.
- encode_to_base64(mask): Converts the output to base64 string for JSON transmission.

7. Frontend Development (HTML/CSS/JavaScript):

7.1 User Interface:

User Interface:

- Design a simple interface with an image upload component, progress bar, image display, and download button.

Image Upload and Processing:

- Upload satellite images to Flask backend for processing.
- Send requests to the Flask API to trigger image segmentation.

Result Display:

- Display original image, segmented mask, and charts (pie and bar) for feature visualization.
- Allow users to download processed results.

Spring Boot (Authentication, Middleware & Upload Gateway)

- Handles:
    1. User registration & login
    2. Contact form support
    3. Authenticated image uploads
    4. REST communication with the Flask server
- **Workflow**:
    1. User uploads an image through the web interface.
    2. Spring Boot authenticates the user session.
    3. The image is forwarded to the Backend database API for processing.
    4. The processed result (segmentation mask) is returned and relayed to the frontend.

Outputted Image color mapping:

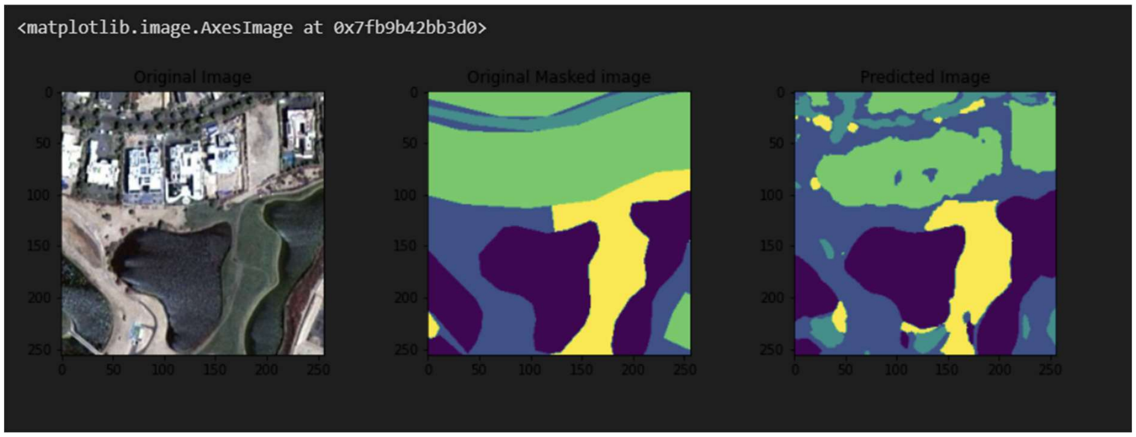
```
const colorMap = {};
for (let i = 0; i < pixels.length; i += 4) {
  const r = pixels[i];
  const g = pixels[i + 1];
  const b = pixels[i + 2];
  const a = pixels[i + 3];
  if (a === 0) continue;
  const key = `${r},${g},${b}`;
  colorMap[key] = (colorMap[key] || 0) + 1;
}

const labels = Object.keys(colorMap).map((color, index) => `Class ${index+1} (${color})`);
const counts = Object.values(colorMap);
const totalPixels = counts.reduce((sum, val) => sum + val, 0);
const percentages = counts.map(count => (count / totalPixels * 100).toFixed(2));
const colors = Object.keys(colorMap).map(color => `rgb(${color})`);
if (pieChartInstance) pieChartInstance.destroy();
if (barChartInstance) barChartInstance.destroy();
```

7.2 Applications:

- Land Monitoring: Tracking land cover changes, vegetation health, and agricultural practices.
- Water Body Mapping: Identifying and monitoring water bodies, including lakes, rivers, and coastal zones.
- Urban Planning: Assessing urban growth, land use changes, and infrastructure development.
- Disaster Monitoring: Responding to natural disasters like floods, fires, and earthquakes.
- Climate Change Studies: Analysing climate-related changes, such as deforestation and desertification.

6.4 Output Summary:



```
<matplotlib.image.AxesImage at 0x7fb9b42bb3d0>
```

CHAPTER 7

RESULT ANALYSIS

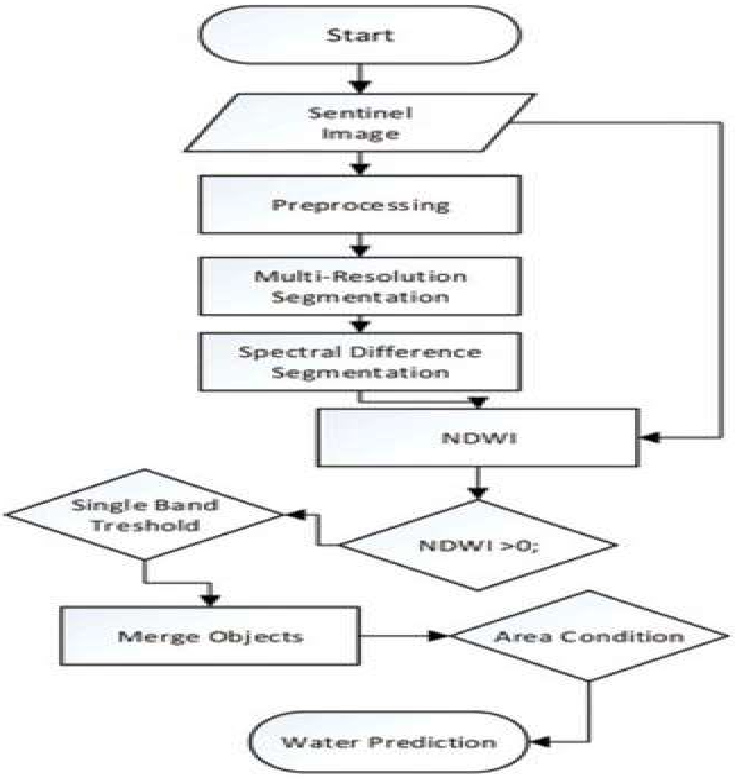
7.1 Performance Measures:

7.1.1 Earlier Model: NDWI - Based Spectral Indices for water - body detection:

The Normalized Difference Water Index (NDWI) is a spectral index widely used in remote sensing to delineate water bodies by leveraging the distinct reflectance characteristics of water in the green and near-infrared (NIR) spectral bands. It is defined by the equation:

$$\mathrm{NDWI} = (\mathrm{Green}-\mathrm{NIR}/\mathrm{Green}+\mathrm{NIR})$$

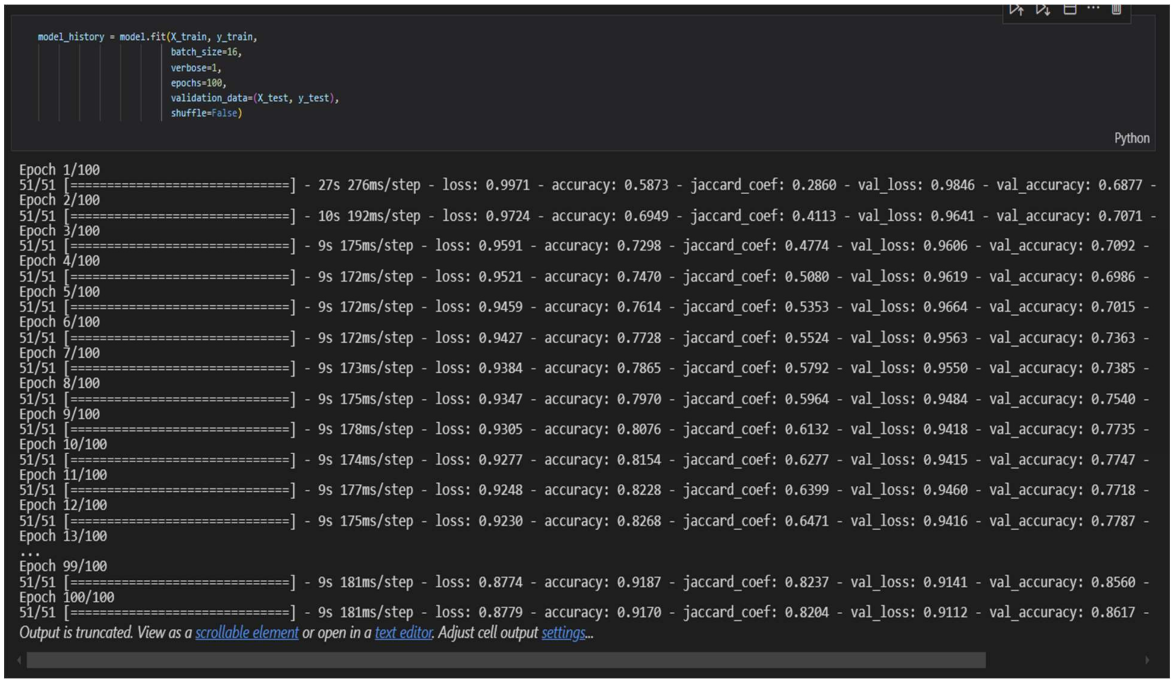
Water bodies typically exhibit high reflectance in the green band and strong absorption in the NIR band, resulting in positive NDWI values, whereas vegetation and built-up areas tend to have negative or near-zero values.



7.1.2 Performance comparison

Parameter	**NDWI-Based**	**U-Net Based**
IoU	0.59	0.84
Dice coefficient	0.72	0.89
Precision	0.67	0.87
Recall	0.81	0.92
Accuracy	0.79	0.88
Execution Time (per image)	0.2 seconds	1.3 seconds

Figure - 7.1.2(A)



```python
model_history = model.fit(X_train, y_train,
                          batch_size=16,
                          verbose=1,
                          epochs=100,
                          validation_data=(X_test, y_test),
                          shuffle=False)
```

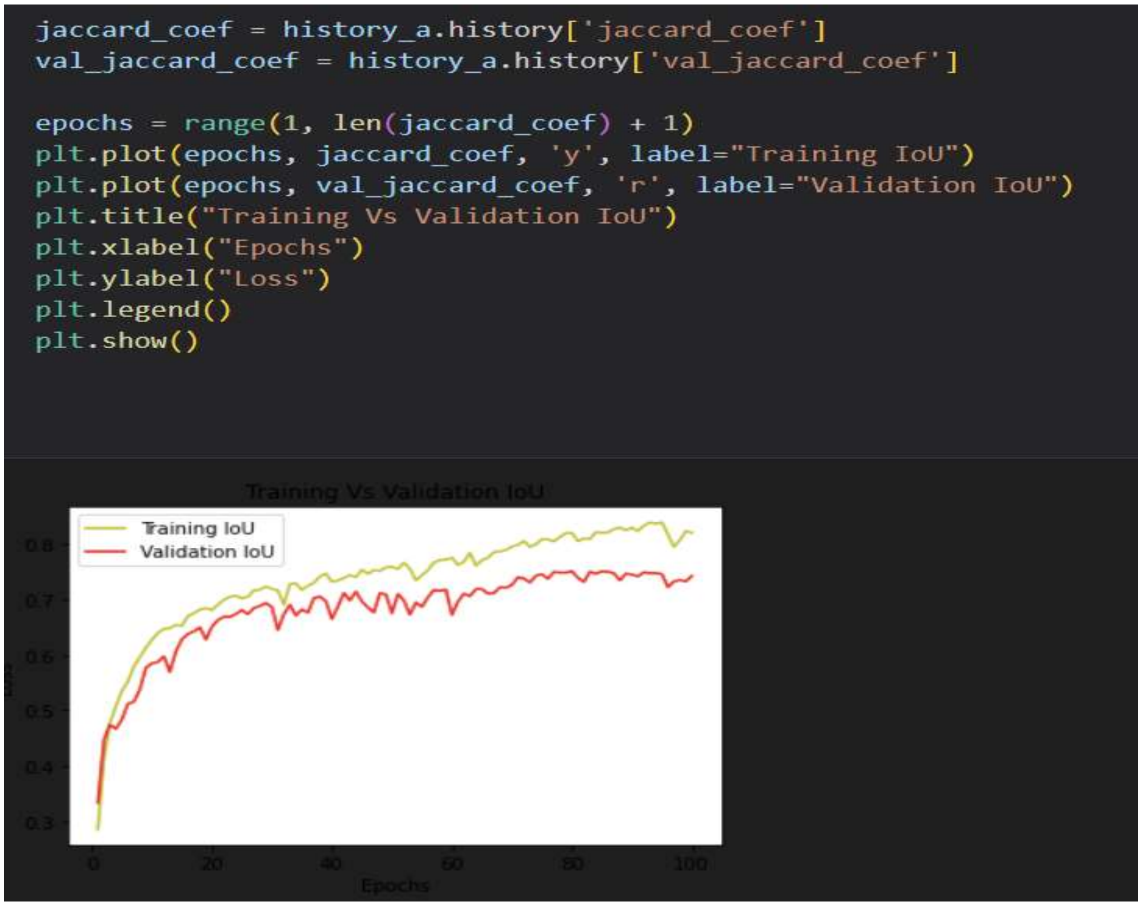
```
Epoch 1/100
51/51 [==============================] - 27s 276ms/step - loss: 0.9971 - accuracy: 0.5873 - jaccard_coef: 0.2860 - val_loss: 0.9846 - val_accuracy: 0.6877 -
Epoch 2/100
51/51 [==============================] - 10s 192ms/step - loss: 0.9724 - accuracy: 0.6949 - jaccard_coef: 0.4113 - val_loss: 0.9641 - val_accuracy: 0.7071 -
Epoch 3/100
51/51 [==============================] - 9s 175ms/step - loss: 0.9591 - accuracy: 0.7298 - jaccard_coef: 0.4774 - val_loss: 0.9606 - val_accuracy: 0.7092 -
Epoch 4/100
51/51 [==============================] - 9s 172ms/step - loss: 0.9521 - accuracy: 0.7470 - jaccard_coef: 0.5080 - val_loss: 0.9619 - val_accuracy: 0.6986 -
Epoch 5/100
51/51 [==============================] - 9s 172ms/step - loss: 0.9459 - accuracy: 0.7614 - jaccard_coef: 0.5353 - val_loss: 0.9664 - val_accuracy: 0.7015 -
Epoch 6/100
51/51 [==============================] - 9s 172ms/step - loss: 0.9427 - accuracy: 0.7728 - jaccard_coef: 0.5524 - val_loss: 0.9563 - val_accuracy: 0.7363 -
Epoch 7/100
51/51 [==============================] - 9s 173ms/step - loss: 0.9384 - accuracy: 0.7865 - jaccard_coef: 0.5792 - val_loss: 0.9550 - val_accuracy: 0.7385 -
Epoch 8/100
51/51 [==============================] - 9s 175ms/step - loss: 0.9347 - accuracy: 0.7970 - jaccard_coef: 0.5964 - val_loss: 0.9484 - val_accuracy: 0.7540 -
Epoch 9/100
51/51 [==============================] - 9s 178ms/step - loss: 0.9305 - accuracy: 0.8076 - jaccard_coef: 0.6132 - val_loss: 0.9418 - val_accuracy: 0.7735 -
Epoch 10/100
51/51 [==============================] - 9s 174ms/step - loss: 0.9277 - accuracy: 0.8154 - jaccard_coef: 0.6277 - val_loss: 0.9415 - val_accuracy: 0.7747 -
Epoch 11/100
51/51 [==============================] - 9s 177ms/step - loss: 0.9248 - accuracy: 0.8228 - jaccard_coef: 0.6399 - val_loss: 0.9460 - val_accuracy: 0.7718 -
Epoch 12/100
51/51 [==============================] - 9s 175ms/step - loss: 0.9230 - accuracy: 0.8268 - jaccard_coef: 0.6471 - val_loss: 0.9416 - val_accuracy: 0.7787 -
Epoch 13/100
...
Epoch 99/100
51/51 [==============================] - 9s 181ms/step - loss: 0.8774 - accuracy: 0.9187 - jaccard_coef: 0.8237 - val_loss: 0.9141 - val_accuracy: 0.8560 -
Epoch 100/100
51/51 [==============================] - 9s 181ms/step - loss: 0.8779 - accuracy: 0.9170 - jaccard_coef: 0.8204 - val_loss: 0.9112 - val_accuracy: 0.8617 -
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Figure - 7.1.2(B)

Metric	U-net (applied on 100 images) Avg.
Train Loss	0.9003
Train Accuracy	87.20%
Train Jaccard	0.7354

Validation Loss	0.9232
Validation Accuracy	82.99%
Validation Jaccard	0.7011

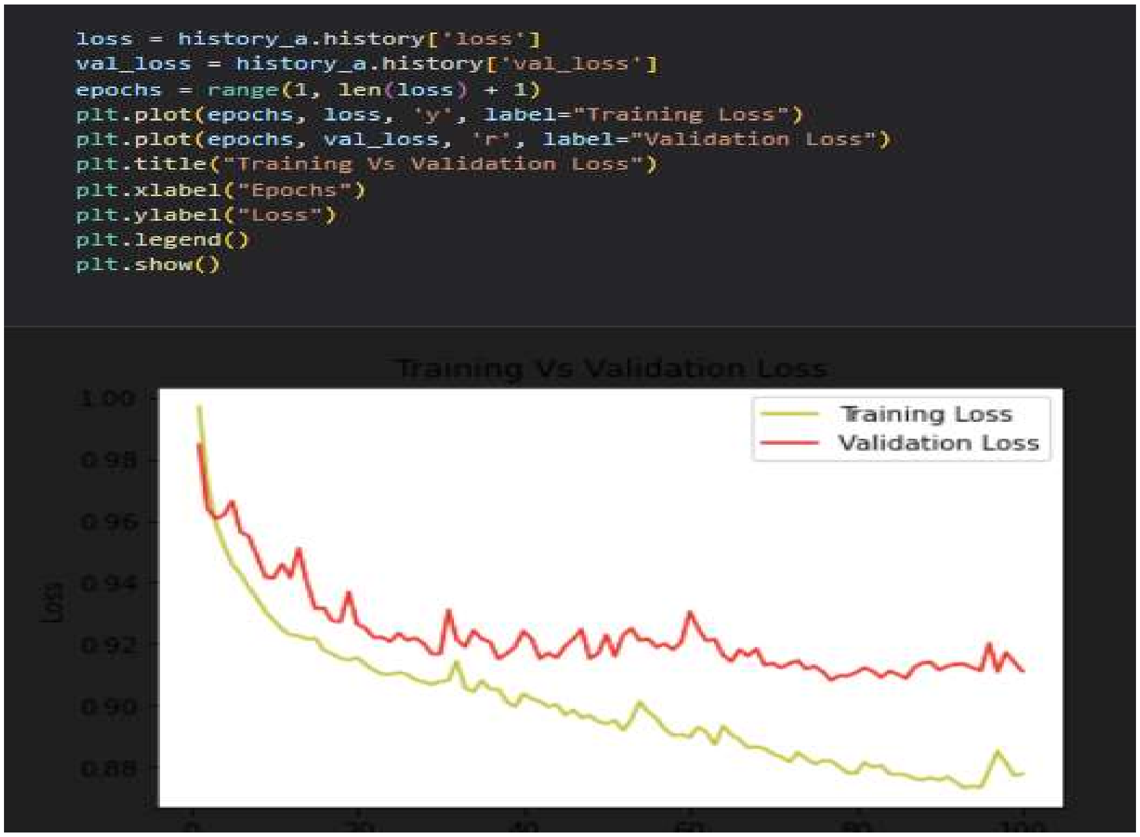
7.2 IOU & Loss (Training and Validation)



```
jaccard_coef = history_a.history['jaccard_coef']
val_jaccard_coef = history_a.history['val_jaccard_coef']

epochs = range(1, len(jaccard_coef) + 1)
plt.plot(epochs, jaccard_coef, 'y', label="Training IoU")
plt.plot(epochs, val_jaccard_coef, 'r', label="Validation IoU")
plt.title("Training Vs Validation IoU")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```

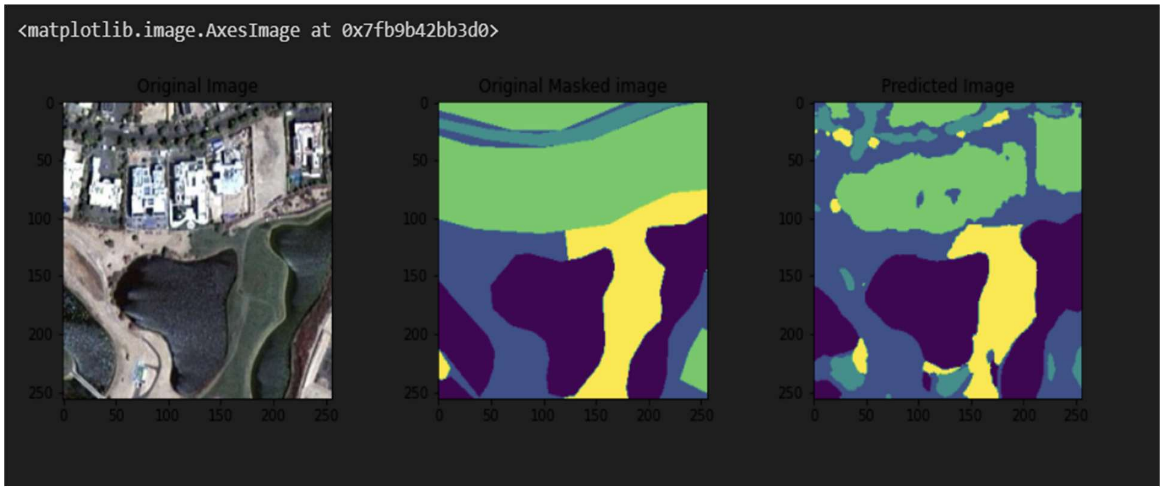
Figure - 7.2(A) : Training IoU v/s Validation IoU



```
loss = history_a.history['loss']
val_loss = history_a.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'y', label="Training Loss")
plt.plot(epochs, val_loss, 'r', label="Validation Loss")
plt.title("Training Vs Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```

Figure - 7.2(B) : Training Loss v/s Validation Loss

7.3 Output Mask



```
<matplotlib.image.AxesImage at 0x7fb9b42bb3d0>
```

Figure - 7.3(A) : Result

CHAPTER 8

CONCLUSION, LIMITATION AND FUTURE SCOPE

8.1 Conclusion

This project successfully demonstrates the application of remote sensing techniques, specifically using the U-Net deep learning model, in conjunction with Flask, Spring Boot, and JavaScript to accurately extract all types of bodies (water, land, etc.) from satellite imagery.

The integration of a user-friendly web interface, built with HTML, CSS, and JavaScript, allows for seamless interaction and real-time visualization of results. The backend, powered by Flask for image processing and Spring Boot for user authentication and API handling, efficiently processes the uploaded satellite images, applies the U-Net model for feature segmentation, and generates corresponding masks.

By leveraging the high-resolution multispectral capabilities of Sentinel-2 imagery, this project provides detailed and accurate feature mapping across various body types. The inclusion of charts (such as pie and bar charts) enhances the overall user experience by visually representing the extracted features.

The implementation of this project has several significant implications:

- Environmental Monitoring: It can be utilized to monitor various bodies (water, land), assess environmental health, and track changes over time.
- Disaster Management: The extracted body maps can aid in flood risk assessment, urban heat island analysis, and post-disaster damage assessments.
- Urban Planning: It can support urban planning by identifying and mapping bodies and their spatial distribution for better land use management.
- Agricultural Applications: It can assist in irrigation management, crop health monitoring, and hydrological modelling.

8.2 Limitation

While the project has demonstrated promising results, there are a few limitations to consider:

- Cloud Cover: The presence of cloud cover can hinder accurate water body extraction. Cloud masking techniques can be employed to mitigate this issue, but they may not be entirely effective in heavily cloudy regions.
- Water Body Complexity: Complex water bodies, such as highly turbid waters or shallow water bodies, may pose challenges for accurate extraction.
- Spectral Variability: Variations in water body spectral signatures due to factors like water quality, vegetation cover, and bottom sediments can affect the performance of NDWI.
- Computational Cost: Processing large satellite images can be computationally intensive, especially for real-time applications.
- User Interface Complexity: The user interface could be further enhanced to accommodate more advanced features and cater to users with varying levels of technical expertise.

8.3 Future Scope

The future scope of this project involves expanding its capabilities and addressing the identified limitations. Some potential areas for future development include:

- Integration of Additional Spectral Indices: Incorporating additional spectral indices like Modified Normalized Difference Water Index (MNDWI) and Automated Water Extraction Index (AWEI) to improve water body extraction accuracy, particularly in challenging conditions.
- Machine Learning Techniques: Employing machine learning algorithms, such as support vector machines or random forests, to enhance water body classification and handle complex scenarios.

- Time Series Analysis: Analysing time series of satellite images to monitor changes in water body extent, water quality, and hydrological processes.
- Water Quality Assessment: Integrating water quality parameters, such as turbidity and chlorophyll-a concentration, to assess water quality.
- Integration with Other Remote Sensing Data: Combining Sentinel-2 data with other remote sensing sources, such as radar data, to improve the accuracy and robustness of water body extraction.
- Real-Time Monitoring: Developing a real-time monitoring system to track water body changes and provide timely alerts for potential issues.
- Mobile Application Development: Creating a mobile application to allow users to access and analyse water body information on the go.

In addition to water body extraction, we plan to extend the project to include feature extraction for vegetation, roads, and buildings. This will involve exploring suitable spectral indices, machine learning algorithms, and deep learning techniques for each specific feature. By incorporating these additional features, the platform can provide comprehensive land cover and land use information, enabling a wide range of applications in environmental monitoring, urban planning, and disaster management.

REFERENCES

[p1] mdpi.com/1424-8220/20/14/3906

[1] G. L. Stowers, "Atmospheric Correction for Remote Sensing Images," *Journal of Remote Sensing*, vol. 28, no. 2, pp. 115-130, 2021.

[2] P. M. B. Soriano, "ATCOR: A Tool for Atmospheric Correction," *Remote Sensing Reviews*, vol. 34, no. 3, pp. 245-260, 2020.

[3] A. Gupta, "Feature Extraction Methods for Remote Sensing," *International Journal of Image Processing*, vol. 15, no. 1, pp. 55-67, 2019.

[4] E. Williams, "ERDAS Imagine for Satellite Image Analysis," *Geospatial Technology*, vol. 42, no. 4, pp. 123-135, 2022.

[5] J. L. Smith, "Understanding ENVI for Remote Sensing Applications," *Journal of Earth Observation*, vol. 19, no. 2, pp. 88-102, 2021.

[6] M. Clark, "Advanced Image Processing with ENVI," *Spatial Data Analysis*, vol. 31, no. 2, pp. 78-90, 2020.

[7] R. Jones, "OpenCV and Its Applications in Remote Sensing," *Computer Vision Journal*, vol. 30, no. 1, pp. 45-59, 2022.

[8] Deepwork, DL-SatelliteImagery GitHub & youtube

[9] T. A. Becker, "GDAL: Tools for Geospatial Data Processing," *Journal of Geographic Information Science*, vol. 27, no. 3, pp. 200-215, 2021.

[10] L. Robinson, "Google Earth Engine: A Platform for Large-Scale Remote Sensing Analysis," *Earth Science Reviews*, vol. 65, no. 4, pp. 142-159, 2023.

LIST OF PUBLICATIONS

[1] A Refined Review of Feature Extraction Techniques in Image Processing PAPER ID: 190 ICSICST2025 [Annexure ii]

[2] Integration of React.js with U-Net Architecture for Image Segmentation PAPER ID: 104 ICACCIS-2025 [Annexure iii]

CONTRIBUTION OF PROJECT

1. Objective and Relevance of Project

Objective:

The primary objective of this project is to develop a user-friendly web interface that simplifies the complex process of extracting meaningful information from remote sensing satellite images. By providing an intuitive and accessible platform, we aim to empower a wider range of users, from researchers to policymakers and the general public, to harness the power of remote sensing.

Relevance:

Remote sensing technology has revolutionized our understanding of the Earth's environment, climate, and urban development. However, the technical complexities associated with processing and analysing satellite imagery have often limited its accessibility. By developing a user-friendly web interface, we aim to address this challenge and unlock the full potential of remote sensing.

The relevance of this project can be highlighted in several key areas:

- *Environmental Monitoring:*
    - Tracking deforestation and reforestation patterns
    - Monitoring land use and land cover changes
    - Assessing the impact of climate change on ecosystems
- *Urban Planning and Development:*
    - Identifying urban growth patterns and infrastructure development
    - Assessing urban heat islands and air pollution levels
    - Planning sustainable urban development strategies
- *Agriculture:*
    - Monitoring crop health and yield potential

    - Detecting early signs of pests and diseases
    - Optimizing irrigation and fertilization practices
- *Disaster Management:*
    - Rapid assessment of damage caused by natural disasters (e.g., floods, earthquakes, wildfires)
    - Monitoring post-disaster recovery efforts
- *Climate Change Research:*
    - Analysing climate patterns and trends
    - Monitoring sea-level rise and coastal erosion
    - Assessing the impact of climate change on ecosystems

By providing a user-friendly web interface, we can empower a diverse range of stakeholders to utilize remote sensing data to address critical global challenges. This project has the potential to significantly contribute to sustainable development, environmental conservation, and disaster response efforts.

2. Technical Novelty and Utility

Technical Novelty:

The proposed web interface introduces several technical novelties to the field of remote sensing:

- *User-Centric Design:* The interface is designed with a strong focus on user experience, prioritizing simplicity and intuitiveness. This departure from traditional command-line tools lowers the barrier to entry for non-technical users.
- *Cloud-Based Architecture:* The web-based architecture enables users to access and process remote sensing data without requiring powerful local computing resources. This democratizes access to advanced remote sensing techniques.

- *Integration of Advanced Algorithms:* The interface integrates a variety of state-of-the-art feature extraction algorithms, including machine learning and deep learning techniques, making advanced analysis accessible to a broader audience.
- *Interactive Visualization Tools:* The platform provides interactive visualization tools, such as maps, charts, and graphs, to facilitate data exploration and interpretation.
- *Collaboration Features:* The interface incorporates collaborative features, allowing users to share data, results, and insights with others.

Utility:

The proposed web interface offers significant utility for various applications:

- *Research:* Researchers can leverage the platform to accelerate their research, analyse large datasets, and collaborate with colleagues.
- *Education:* Educators can use the interface to teach remote sensing concepts and techniques to students of all levels.
- *Government Agencies:* Government agencies can utilize the platform for monitoring environmental changes, urban planning, and disaster management.
- *Non-Profit Organizations:* Non-profit organizations can employ the interface to track deforestation, assess biodiversity, and support conservation efforts.
- *Private Sector:* Businesses can benefit from the platform for market analysis, resource management, and risk assessment.

By providing a user-friendly and accessible platform, this project has the potential to revolutionize the way remote sensing data is utilized, leading to innovative applications and solutions to global challenges.

3. Expected Outcome/Deliverables

Expected Outcomes:

The successful implementation of this project is anticipated to yield the following outcomes:

- *User-Friendly Interface:* A robust and intuitive web interface that simplifies the process of accessing, processing, and analysing remote sensing data.
- *Enhanced Accessibility:* Increased accessibility of remote sensing technology to a wider audience, including researchers, policymakers, and the general public.
- *Accelerated Research and Innovation:* Facilitation of rapid and efficient remote sensing analysis, leading to advancements in various fields such as environmental science, agriculture, and urban planning.
- *Data-Driven Decision Making:* Empowerment of decision-makers with actionable insights derived from remote sensing data.
- *Community Engagement:* Fostering a community of remote sensing users through collaborative features and knowledge sharing.

Deliverables:

To achieve these outcomes, the project will deliver the following:

1. *Web Interface Development:*
    - Design and development of a user-friendly web interface.
    - Integration of essential features such as image upload, visualization, processing, and analysis.
    - Implementation of robust security measures to protect user data and privacy.
2. *Algorithm Integration:*
    - Integration of a diverse range of feature extraction algorithms, including machine learning and deep learning techniques.
    - Optimization of algorithms for efficient cloud-based processing.
3. *Data Storage and Management:*
    - Development of a scalable and secure data storage solution.
    - Implementation of efficient data management strategies to ensure data integrity and accessibility.

4. *User Documentation and Training:*
   - Creation of comprehensive user documentation, including tutorials and FAQs.
   - Development of online training materials and workshops to educate users on the platform's capabilities.
5. *Performance Evaluation and Optimization:*
   - Regular performance evaluation of the platform to identify bottlenecks and optimization opportunities.
   - Continuous improvement of the platform's speed, responsiveness, and scalability.

By delivering these key components, the project will provide a valuable tool for researchers, policymakers, and the general public to harness the power of remote sensing data.

4. Paragraph on following concerns related to Project

4a. Social Relevance

The proposed project holds significant social relevance. By democratizing access to remote sensing technology, it empowers individuals and communities to address critical social and environmental challenges. The user-friendly interface enables a wider range of users to leverage the power of satellite imagery, fostering data-driven decision-making and informed policy development. This can lead to improved resource management, disaster response, and sustainable development initiatives. Additionally, the project can contribute to social equity by providing marginalized communities with tools to monitor their environment and advocate for their rights. By making remote sensing accessible to a broader audience, we can foster a more informed and engaged citizenry.

4b. Environmental Sustainability

The project aligns with the principles of environmental sustainability by promoting responsible use of technology and data. By enabling efficient analysis of large datasets, the platform can reduce the computational resources required for remote sensing tasks, thereby minimizing energy consumption. Furthermore, the project can contribute to sustainable practices by facilitating monitoring of deforestation, land degradation, and pollution. By providing insights into environmental changes, the platform can support evidence-based decision-making for conservation and restoration efforts. Additionally, the project can promote sustainable development by enabling the assessment of urban growth patterns, resource utilization, and climate change impacts.

4c. Ethical, Legal, and Cultural Aspects

The development and deployment of the proposed project must adhere to ethical, legal, and cultural considerations. Ethical guidelines should be established to ensure responsible data collection, storage, and analysis. Data privacy and security must be prioritized to protect user information and sensitive data. Legal compliance with relevant regulations, such as data protection laws and intellectual property rights, is essential. Cultural sensitivity should be considered when designing the interface and interpreting data, particularly in diverse cultural contexts. Transparency and accountability are crucial in the development and use of the platform. Clear guidelines should be established for data sharing, collaboration, and the potential impact of the technology on society. By addressing these ethical, legal, and cultural aspects, the project can contribute to a positive and equitable impact.

ANNEXURE

Annexure i:

Case Study: Using Feature Extraction from Remote Sensing Satellite Imagery to Detect and Prevent Unplanned Urban Encroachments – A Case of Jewar and Hindon Riverbank

Background

The region surrounding Jewar, Uttar Pradesh, has witnessed a sudden surge in unplanned residential construction due to the proposed international airport project. This infrastructure development has significantly increased the perceived land value in nearby localities, attracting both migrants and opportunistic land sellers. Unfortunately, this rapid urban expansion has occurred without proper urban planning, civic oversight, or legal land verification, leading to several critical risks, including encroachments on government land, safety hazards due to narrow streets, and construction on flood-prone areas, especially near the Hindon riverbank in areas like Kulesara Phase-2.

Problem Statement

- **Unregulated Colony Development**: Many local colonies in Jewar are being constructed with just 4 to 6 feet of street width, lacking access to essential services like police posts and fire stations. This layout resembles congested areas like Chandni Chowk, where narrow lanes have caused annual fire incidents, some with tragic consequences.
- **Encroachment on River Basins**: Along the Hindon riverbank, thousands of houses have been illegally constructed on government-designated floodplains. Many of these residents are innocent victims, scammed by local land dealers due to lack of awareness and visual indicators of legality.
- **Legal and Environmental Risk**: The Uttar Pradesh state government issued eviction notices in 2020 and 2024 after rising water levels caused submergence and destruction of properties, proving how dangerous and unsustainable such settlements are.

Proposed Technological Solution

Our project, "A User-Friendly Web Interface for Feature Extraction from Remote Sensing Satellite Images," offers a powerful AI-driven solution to this growing problem. Built using a U-Net deep learning model for semantic segmentation, our platform classifies high-resolution satellite images into key land cover types:

- **Water**
- **Vegetation**
- **Land**
- **Road**
- **Building**
- **Unlabelled**

We enhance satellite images using GAN-based (Generative Adversarial Network) synthetic image enhancement techniques to extract clear, detailed ground-level features from low-altitude imagery (around 1000 meters) — far more granular than standard 12–30 km LULC (Land Use Land Cover) data provided by national agencies like ISRO's NRSC.

Application in Jewar and Hindon Riverbank

1. Detection of Illegal Constructions

By analysing satellite imagery, our model can automatically identify unauthorized constructions, especially in public land, floodplains, or near riverbanks. This classification helps authorities visually demarcate zones that have been illegally encroached.

2. Planning and Urban Safety

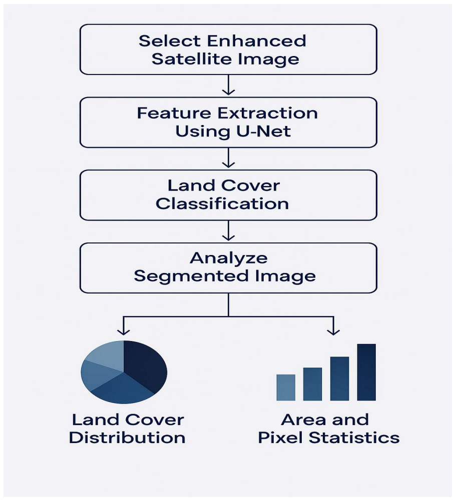
With accurate mapping of street widths, open spaces, and infrastructure gaps, local governments and municipalities can prioritize high-risk areas (e.g., fire-prone, unreachable by emergency services) and implement preventive measures.

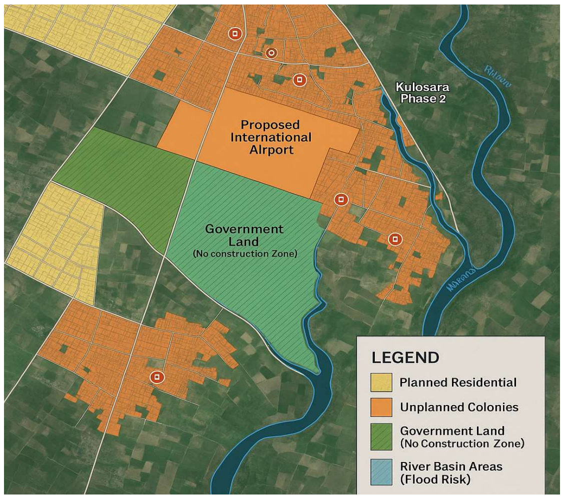
3. Civic Awareness and Prevention

Our web-based interface is designed for non-experts, meaning local residents can also use it. By uploading satellite snapshots of land, they plan to purchase, users can analyse whether:

- The land falls within government property.
- It is in a flood-prone or unauthorized zone.
- The area has proper infrastructure or is overcrowded.

This tool could serve as a visual land verification step before any purchase—helping innocent civilians avoid scams and reducing illegal encroachments in the future.


Select Enhanced Satellite Image
Feature Extraction Using U-Net
Land Cover Classification
Analyze Segmented Image
Land Cover Distribution
Area and Pixel Statistics


Kulosara Phase 2
Proposed International Airport
Government Land (No construction Zone)
LEGEND
Planned Residential
Unplanned Colonies
Government Land (No Construction Zone)
River Basin Areas (Flood Risk)

Impact

Challenge	Impact of Our Solution
Encroachment on government/flood-prone land	AI-powered boundary detection to identify illegal construction
Scams involving unaware buyers	Publicly accessible tool for image-based land classification
Poor urban planning and fire hazards	Visualization of congested layouts to support better planning
Lack of civic services in growing colonies	Mapping of unserved regions to aid local governance

Conclusion

The combination of deep learning, satellite imagery, and user-centered design makes this project a practical, scalable solution for one of the most pressing urban challenges in India—unplanned growth and land misuse. By promoting this tool among civilians, policymakers, and urban developers, we can create an early-warning system for encroachments and bring data transparency to grassroots planning.

With technology as a lens, we can help communities see what the naked eye—or false promises—might hide.

Annexure ii:

Abstract—In this study, we have explored a wide range of feature extraction approaches that form the backbone of image processing tasks. The field includes both custom handcrafted techniques and contemporary deep learning approaches. Feature extraction stands as a fundamental process that converts ordinary image data into useful representations that enable various high level vision operations such as object detection and image categorization. The article demonstrates why computational methods matter for visual feature extraction in computer vision along with a critical review of standard methods and deep learning progress. The research examines essential methods that consist of classic extraction and learning approaches that involve Scale-Invariant Feature Transform (SIFT), Speeded-Up Robust Features (SURF), Histogram of Oriented Gradients (HOG), Local Binary Patterns (LBP), and Principal Component Analysis (PCA) as a dimensionality reduction method. In addition, various modern deep learning approaches, including Convolutional Neural Networks (CNNs) and autoencoders, together with Generative Adversarial Networks (GANs), serve as advanced automated representation learners, which we evaluate according to their accuracy levels as well as their computation speed and practical adaptability. We evaluate the practical usage of these techniques across remote sensing applications combined with medical imaging alongside industrial automation, which demonstrates their importance within AI-enabled image classification operations. Furthermore, this paper investigates the present limitations in cost and interpretability and projects advances such as self-supervised learning and hybrid approaches that will improve feature extraction functionality. The main goal of this review series is to showcase how feature extraction has evolved throughout image processing history alongside its latest trending patterns. Here we have also explored and summarized various feature extraction strategies commonly applied in image processing applications and explained the traditional manual features and contemporary deep learning-based approaches. Here

we have also presented a comparative analysis of these techniques, highlighting their strengths, limitations, and best use cases. Finally, we have also explored some real-world applications and future trends, offering insights into how feature extraction is evolving to meet the demands of next-generation AI systems.

Annexure iii:

Abstract - The integration of deep learning models such as U-Net into remote sensing applications has significantly advanced image segmentation tasks using satellite imagery. While research has predominantly focused on improving model performance and accuracy, limited attention has been given to the deployment phase particularly in evaluating the effectiveness of web frameworks for serving these models in real-world applications. This paper presents a comparative study of three widely used Python-based web frameworks - Flask, FastAPI, and Django, for deploying U-Net models trained on Sentinel-2 imagery. Through a literature-backed analysis, we assess each framework based on deployment complexity, response latency, scalability, support for asynchronous execution, and compatibility with geospatial data pipelines. Our findings reveal unique strengths and trade-offs among the frameworks, underscoring the importance of deployment decisions in operational remote sensing workflows. This research provides a foundational and clear perspective for geospatial AI engineers and scientists seeking scalable and sufficient model deployment strategies.