Satellite Image Classification and Enhancement

Project Report submitted in partial fulfillment of

The Requirements for the degree of

Bachelor of Technology

in

INFORMATION TECHNOLOGY

of

MAULANA ABUL KALAM AZAD UNIVERSITY OF TECHNOLOGY, WEST BENGAL

By

Sneha Pradhan - (10900221029)

Baishnavi Srivastava - (10900221052)

Simran Bhadani - (10900221056)

Sneha Rani - (10900221066)

Soumi Ghosh - (10900221079)

Under the guidance of

Prof. Purbasha Das

DEPARTMENT OF INFORMATION TECHNOLOGY

NETAJI SUBHASH ENGINEERING COLLEGE,
Techno City, Kolkata-700152

Affiliated to Maulana Abul Kalam Azad University of Technology, W.B

NETAJI SUBHASH ENGINEERING COLLEGE,

Techno City, Kolkata-700152

Certificate

This is to certify that this project report titled "**SATELLITE IMAGE CLASSIFICATION & ENHANCEMENT"** submitted in partial fulfillment of requirements for award of the degree Bachelor of Technology (B. Tech) in INFORMATION TECHNOLOGY of MAULANA ABUL KALAM AZAD UNIVERSITY OF TECHNOLOGY is a faithful record of the original work carried out by,

Sneha Pradhan,	**Roll no**. 10900221029,	**Regd. No.** 211090100210095 (2021-22)
Baishnavi Srivastava,	**Roll no**. 10900221052,	**Regd. No.** 211090100210115 (2021-22)
Simran Bhadani,	**Roll no**. 10900221056,	**Regd. No.** 211090100210118 (2021-22)
Sneha Rani,	**Roll no**. 10900221066,	**Regd. No.** 211090100210159 (2021-22)
Soumi Ghosh,	**Roll no**. 10900221079,	**Regd. No.** 211090100210080 (2021-22)

under my guidance and supervision.

It is further certified that it contains no material, which to a substantial extent has been submitted for the award of any degree/diploma in any institute or has been published in any form, except the assistance drawn from other sources, for which due acknowledgement has been made.

____

Date………..

Guide's signature

Purbasha Das

*Sd/*__________________

Head of the Department

Information Technology

NETAJI SUBHASH ENGINEERING COLLEGE

TECHNO CITY, GARIA, KOLKATA – 700 152

NETAJI SUBHASH ENGINEERING COLLEGE,

Techno City, Kolkata-700152

DECLARATION

We hereby declare that this project report entitled **"Satellite Image Classification and Enhancement"**, is our own original work carried out as an undergraduate student in NETAJI SUBHASH ENGINEERING COLLEGE except to the extent that assistance from other sources is duly acknowledged.

All sources used for this project have been fully and properly cited. It contains no material which to a substantial extent has been submitted for the award of any degree/diploma in any institute or has been published in any form, except where due acknowledgement is made.

Student's Name	University Roll	Signature	Date
Sneha Pradhan	10900221029	…………………....	
Baishnavi Srivastava	10900221052	……………………	
Simran Bhadani	10900221056	……………………	
Sneha Rani	10900221066	……………………	
Soumi Ghosh	10900221079	……………………	

CERTIFICATE OF APPROVAL

We hereby approve this dissertation titled

"Satellite Image Classification & Enhancement"

carried out by

Sneha Pradhan,	**Roll no**. 10900221029,	**Regd. No.** 211090100210095 (2021-22)
Baishnavi Srivastava,	**Roll no**. 10900221052,	**Regd. No.** 211090100210115 (2021-22)
Simran Bhadani,	**Roll no**. 10900221056,	**Regd. No.** 211090100210118 (2021-22)
Sneha Rani,	**Roll no**. 10900221066,	**Regd. No.** 211090100210159 (2021-22)
Soumi Ghosh,	**Roll no**. 10900221079,	**Regd. No.** 211090100210080 (2021-22)

under the guidance of

Prof. Purbhasa Das

of Netaji Subhash Engineering College, Kolkata in partial fulfillment of requirements for award of the degree Bachelor of Technology (B. Tech) in INFORMATION TECHNOLOGY of MAULANA ABUL KALAM AZAD UNIVERSITY OF TECHNOLOGY

Date………...

Examiners' signatures:

1. ………………………………………….
2. ………………………………………….

Acknowledgement

Before getting into the thickest of things, we would like to present our gratitude towards the personalities under whose supervision this project work was carried out. Giving outstanding guidance be it directly or indirectly since the birth of the project. I solely recognize our indebtedness for guidance and assistance of the project adviser and other members of the faculty.

The project opportunity we had with our mentor was a great chance for learning and professional development. Therefore, I consider myself as a very lucky individual to be a part of it.

We are highly indebted to Prof. Purbasha Das, Assistant Professor, Department of Information Technology, for her support during the tenure of the project.

We wish to extend my sincere gratitude to faculties of Department of Information Technology, for their guidance, encouragement and valuable suggestion which proved extremely useful and helpful in completion of this Project.

We wish to express my profound gratitude for all advice and guidance while We were a trainee. We very much appreciate their entire kindness for helping and teaching me. We find no words to acknowledge the sacrifice, help and inspiration rendered by my parents to take up this study and also to my friends and teammates who helped a lot.

With thanks to all,

SNEHA RANI

SOUMI GHOSH

SNEHA PRADHAN

SIMRAN BHADANI

BAISHNAVI SRIVASTAVA

B. Tech (IT)

Dated:…………………

Abstract

This project addresses the challenge of accurately classifying satellite images, which often suffer from varying quality due to environmental and imaging conditions. To improve classification performance, the project integrates image enhancement techniques with a deep learning-based classification model.OpenCV is used to enhance image quality through methods such as Contrast Limited Adaptive Histogram Equalization (CLAHE), noise reduction filters, and sharpening. These preprocessing steps help reveal important features in satellite images, making them more suitable for analysis.For classification, an EfficientNet-B0 Convolutional Neural Network is employed to categorize images from the EuroSAT dataset into ten land cover classes. The model is trained with data augmentation to improve generalization and robustness.The combined approach demonstrates that preprocessing satellite imagery leads to more accurate and reliable classification, making it valuable for applications in environmental monitoring, agriculture, and urban planning.

Keywords: Image Classification, Image Enhancement, CNN, OpenCV, Satellite Imagery, EfficientNet-B0.

Contents

List of Figure

Chapter 01

Problem Definition and Proposed solution

1.1 Problem Statement

Satellite images play a critical role in diverse applications such as environmental monitoring, urban planning, disaster management, agriculture, and defense. However, satellite images are often degraded due to factors like atmospheric conditions, sensor limitations, and environmental interference. Common issues include low contrast, noise, haze, and blurriness, which can obscure essential features, making it challenging for machine learning models to classify these images accurately.

Accurate classification of satellite images is essential for extracting valuable insights, such as identifying land cover types, monitoring deforestation, or detecting urban expansion. The degradation of image quality often results in reduced feature extraction capability, which negatively impacts the classification accuracy of models, particularly in real-world scenarios where timely and reliable decisions are critical.

Challenges

1. **Degraded Image Quality**:
   Satellite images frequently suffer from noise, low contrast, and distortions caused by environmental and atmospheric conditions, leading to a loss of important details.

2. **Complexity of Satellite Imagery**:
   Satellite images exhibit diverse features such as different landforms, vegetation types, or urban structures, requiring robust feature extraction for accurate classification.

3. **Data Dependency**:
   Classification models like CNNs require high-quality labeled datasets for training, which can be difficult to obtain, especially for rare or complex satellite features.

4. **Generalization Issues**:
   Classification models trained on one set of satellite images may fail to generalize well to images from different sensors, resolutions, or environmental conditions

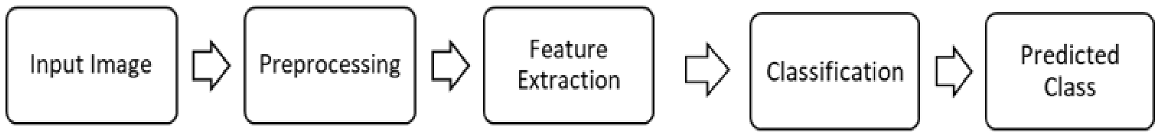
1.2. Proposed Solution

To effectively address the challenges associated with poor-quality satellite imagery and achieve accurate land cover classification, this project presents an integrated system combining advanced image enhancement techniques with a deep learning-based classification model. The pipeline begins with an image preprocessing phase where raw satellite images are enhanced using a combination of computer vision methods implemented via OpenCV. Specifically, **Contrast Limited Adaptive Histogram Equalization (CLAHE)** is employed to improve contrast and highlight local features. This enhancement process ensures that important visual cues in the satellite data—such as boundaries of vegetation, water bodies, and built-up regions—are preserved and made more distinguishable.

Once enhanced, the images are fed into a Convolutional Neural Network (CNN)-based classification model. For this project, the **EfficientNetB0** architecture is utilized due to its optimized balance between accuracy and computational efficiency. The model is trained on a curated and preprocessed dataset representing various land cover types, including vegetation, urban infrastructure, and water bodies. **Data augmentation** techniques such as rotation, scaling, and horizontal flipping are employed during training to increase dataset variability and improve model generalization.

The classification performance is evaluated using standard metrics such as **accuracy, precision, recall**, and **F1-score**, demonstrating marked improvements over baseline models that do not incorporate image enhancement. The results underscore the effectiveness of this two-stage approach, where enhancement significantly contributes to better feature extraction and, consequently, higher classification accuracy.

This solution holds strong potential for practical applications in remote sensing, including **disaster management**, **urban growth monitoring**, **deforestation detection**, and **environmental surveillance**, where the clarity and accurate interpretation of satellite imagery are crucial for timely and informed decision-making.



Chapter 02

Introduction

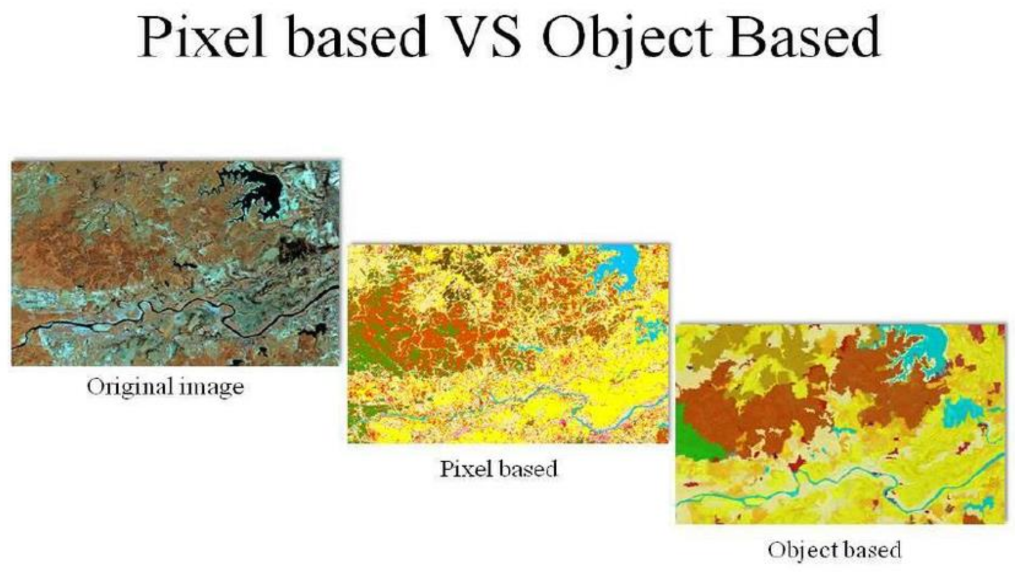
Satellite imagery plays a crucial role in various fields such as environmental monitoring, urban planning, agriculture, and disaster management. The ability to analyze and interpret satellite images has significantly evolved with advancements in technology, particularly through the integration of machine learning and image processing techniques. This report delves into the methodologies and applications of satellite image classification and enhancement, emphasizing their importance in remote sensing.

2.1 Overview of Satellite Image Classification

Satellite image classification involves the process of assigning labels to pixels or groups of pixels in an image based on their characteristics. This process is essential for understanding land cover types, monitoring changes in the environment, and managing natural resources. Classification can be broadly categorized into:

- **Object-Level Classification:** Assigns a single label to an entire image, useful for broader categorizations such as urban, forest, or agricultural land.
- **Pixel-Level Classification (Semantic Segmentation)**: Assigns a label to each pixel, allowing for detailed analysis of complex scenes where multiple land cover types coexist.

The classification process typically utilizes various algorithms ranging from traditional methods like decision trees and support vector machines to advanced techniques involving deep learning, such as Convolutional Neural Networks (CNNs).



2.2 Importance of Image Enhancement

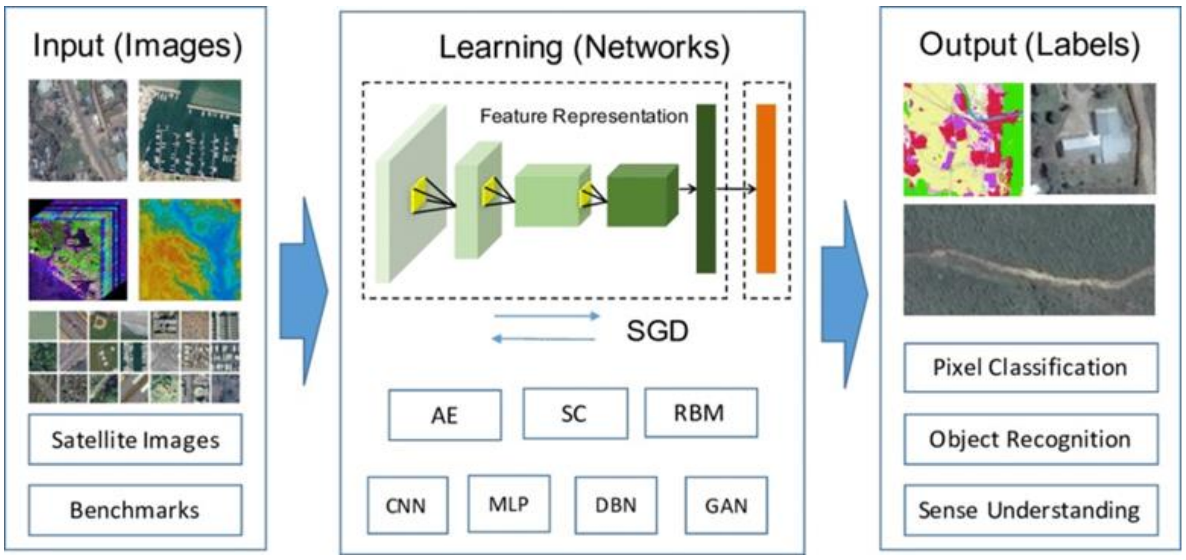
Enhancing satellite images is crucial for improving visual quality and interpretability. Raw satellite data often suffers from issues such as atmospheric interference, sensor noise, and varying illumination conditions. Image enhancement techniques aim to improve the clarity and detail of images through methods including:

- **Contrast Adjustment:** Enhances the visibility of features by stretching the range of pixel values.
- **Noise Reduction:** Removes unwanted variations in pixel values that can obscure important details.
- **Geometric Corrections:** Aligns images to a specific coordinate system to ensure accurate spatial representation.

2.3. Methodologies in Satellite Image Processing

The methodologies employed in satellite image classification and enhancement can be summarized as follows:

- **Data Acquisition**: Involves capturing satellite images using various sensors. The choice of sensor impacts the resolution and quality of the images obtained.
- **Preprocessing:** Includes geometric correction, radiometric correction, and noise removal to prepare raw data for analysis.
- **Feature Extraction:** Identifies relevant characteristics from images that are crucial for classification tasks.
- **Classification Algorithms:** Utilizes machine learning techniques to classify the extracted features into predefined categories.
- **Post-Processing:** Involves refining the classification results through techniques such as majority filtering or morphological operations to enhance accuracy.



2.4. Applications

The integration of satellite image enhancement and classification has significant practical relevance across a wide range of domains. By improving image clarity and classification accuracy, this system supports critical decision-making processes in the following areas:

Urban Planning and Infrastructure Development: Enhanced satellite imagery enables accurate analysis of land use and land cover changes, supporting planners in identifying expansion patterns, optimizing land allocation, and managing resources effectively. This helps in designing sustainable urban layouts, monitoring unauthorized constructions, and forecasting future urban growth.

Environmental Monitoring and Conservation: Accurate classification of land cover types is essential for tracking environmental changes such as deforestation, wetland degradation, and glacial retreat. Enhanced imagery allows for precise detection of subtle changes in vegetation and landforms, aiding environmental agencies and researchers in implementing conservation strategies and assessing ecological health.

Agricultural Monitoring and Crop Management: By classifying vegetation types and analysing plant health using vegetation indices (e.g., NDVI), the system assists in monitoring crop conditions, detecting stress or disease, and estimating yields. This supports precision agriculture practices, optimizing fertilizer and water usage, and improving food security outcomes.

Disaster Management and Response: In events like floods, wildfires, and landslides, the system can rapidly classify and map affected areas. Enhanced imagery ensures that critical features are not obscured by noise or poor contrast, facilitating accurate damage assessment and efficient deployment of relief resources.

Water Resource Management: Classification of water bodies from satellite images supports the monitoring of rivers, lakes, and reservoirs. This is crucial for managing water supply, detecting drought conditions, and ensuring sustainable usage of water resources.
Climate Change Studies: Long-term monitoring of land cover transitions, such as desertification or melting permafrost, contributes to understanding the regional and global effects of climate change. Enhanced and accurately classified satellite images provide valuable data for climate models and impact studies.

This system, therefore, plays a critical role in domains where remote sensing data is a primary input, offering enhanced reliability and accuracy for large-scale, real-time geospatial analysis.

Chapter 03

Background and Related Studies

The field of image classification and enhancement has seen significant advancements over the years, with numerous researchers proposing solutions to address the challenges of degraded image quality and the need for accurate classification. Below, we present a review of the related work in this area, categorized into two key domains: image classification using Convolutional Neural Networks (CNNs) and image enhancement techniques.

3.1. Image Classification Using Convolutional Neural Networks (CNNs)

CNNs have become the cornerstone of modern image classification tasks due to their powerful ability to learn spatial hierarchies of features from raw pixel data. Unlike traditional methods that rely on handcrafted features, CNNs learn feature representations directly through backpropagation, leading to superior performance on complex visual tasks.

- **AlexNet (2012):** Introduced by Krizhevsky et al., AlexNet was a groundbreaking CNN architecture that won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) by a significant margin. The model demonstrated the feasibility of deep learning for image classification and popularized the use of GPUs for large-scale training.

- **VGGNet(2014) and ResNet (2015):** VGGNet, developed by Simonyan and Zisserman, introduced a simple yet effective architecture using very small (3×3) convolutional filters, stacked deeply to improve classification performance. ResNet, proposed by He et al., introduced **residual connections** or "skip connections" to allow training of extremely deep networks by mitigating the vanishing gradient problem. ResNet set new benchmarks in accuracy and training efficiency.

- **EfficientNet(2019):** Tan and Le proposed EfficientNet, a family of models that use a **compound scaling method** to balance depth, width, and resolution, resulting in state-of-the-art performance with fewer parameters and reduced computational cost. In this project, **EfficientNetB0** is adopted for classification due to its efficiency and effectiveness on limited-resource environments, making it suitable for real-time or large-scale satellite image analysis.

- **Satellite Image Classification Applications:** CNNs have been increasingly adapted for remote sensing. For example, **Zhang et al. (2016)** designed a deep CNN for land cover classification from high-resolution satellite imagery, achieving significant gains over traditional classifiers like SVM and Random Forest. **Ayyub et al. (2020)** enhanced this line of work by leveraging **transfer learning** on pre-trained CNN models to classify multi-spectral images, improving model generalization and reducing training time, especially when labeled datasets are limited.

3.2 Image Enhancement Techniques

Image enhancement techniques aim to improve the visual quality of images, enabling better performance for downstream tasks like classification, detection, and recognition.

- **Traditional Techniques**: Early methods for image enhancement, such as histogram equalization and adaptive histogram equalization (AHE), focused on improving contrast and brightness in images. Pizer et al. (1987) introduced Contrast Limited Adaptive Histogram Equalization (CLAHE), a popular method to avoid over-enhancement and noise amplification.

- **Noise Reduction Filters**: Techniques such as Gaussian blur and median filtering have been widely used to remove noise from images. However, these methods often result in a loss of fine details.

- **Deep Learning for Image Enhancement**: Recently, deep learning methods have been employed for more sophisticated image enhancement. For example, Zhang et al. (2017) introduced a CNN-based denoising model (DnCNN), which effectively removes noise while preserving fine details. Similarly, Chen et al. (2018) proposed a deep learning-based low-light image enhancement technique using a Retinex-inspired network.

- **Generative Adversarial Networks (GANs)**: GANs have been employed for advanced image enhancement tasks such as super-resolution and deblurring. Ledig et al. (2017) introduced the Super-Resolution GAN (SRGAN) for generating high-resolution images, while Kupyn et al. (2018) proposed the DeblurGAN for motion blur removal.

3.3 Integration of Image Enhancement and Classification

Several researchers have explored the synergy between image enhancement and classification to address challenges in real-world scenarios.

- **Medical Imaging**: Researchers have applied image enhancement techniques to improve the accuracy of medical image classification. For instance, Singh et al. (2020) combined CLAHE with CNNs to enhance the detection of lung abnormalities in chest X-rays, achieving higher diagnostic accuracy.

- **Underwater Image Analysis**: Anwar and Li (2020) proposed the Underwater Image Enhancement Network (UIE-Net) to address underwater distortions. When coupled with CNN-based classification models, their approach demonstrated superior performance in aquatic object detection.

- **Satellite Imagery**: Kim et al. (2021) combined noise reduction and contrast enhancement techniques with CNNs to classify land-use patterns in satellite images, showing significant improvement in classification accuracy under challenging conditions.

3.4 Challenges and Gaps in Related Work

While significant progress has been made in both satellite image enhancement and classification using deep learning, several challenges and research gaps remain unaddressed. These limitations hinder the development of robust, scalable, and real-time systems for practical geospatial applications.

- **Real-World Generalization and Dataset Limitations**: One of the primary limitations in current research is the lack of diverse, high-quality training datasets that adequately represent real-world variability. Many models are trained on curated datasets with consistent conditions—such as fixed spatial resolution, sensor types, or geographical regions—which limits their generalization across different terrains, seasons, and atmospheric conditions. This results in performance drops when models are applied to unseen data, especially in complex or underrepresented environments.

- **Trade-off Between Enhancement and Feature Preservation:** Image enhancement techniques are intended to improve the visibility of important features. However, aggressive or poorly tuned enhancement algorithms—such as over-sharpening or excessive contrast adjustment—can introduce artifacts, distort natural textures, or alter the statistical properties of the image. These changes may confuse classification models, leading to reduced accuracy. Striking the right balance between enhancement and preservation of critical spatial-spectral information is still a key research challenge.

- **Computational Complexity and Real-Time Constraints:** Many existing solutions rely on sequential processing pipelines that separately perform enhancement and classification. While this modularity offers flexibility, it often results in increased computational cost, latency, and memory usage—especially for high-resolution satellite imagery. This limits the feasibility of deploying such models in real-time or edge computing environments, where resources are constrained. Efficient integration of enhancement and classification in a unified architecture is an area with substantial room for innovation.

- **Limited Adaptation to Multi-Spectral and Hyper-Spectral Data:** Although CNNs have shown success on RGB satellite images, many applications require the use of multi-spectral or hyper-spectral data. Existing models often fail to fully exploit the spectral richness of such data due to architectural constraints or a lack of pre-trained weights in these domains. This results in suboptimal performance in tasks like crop type discrimination or mineral mapping.

Chapter 04

Tools and Technologies Used

4.1. Application Workflow

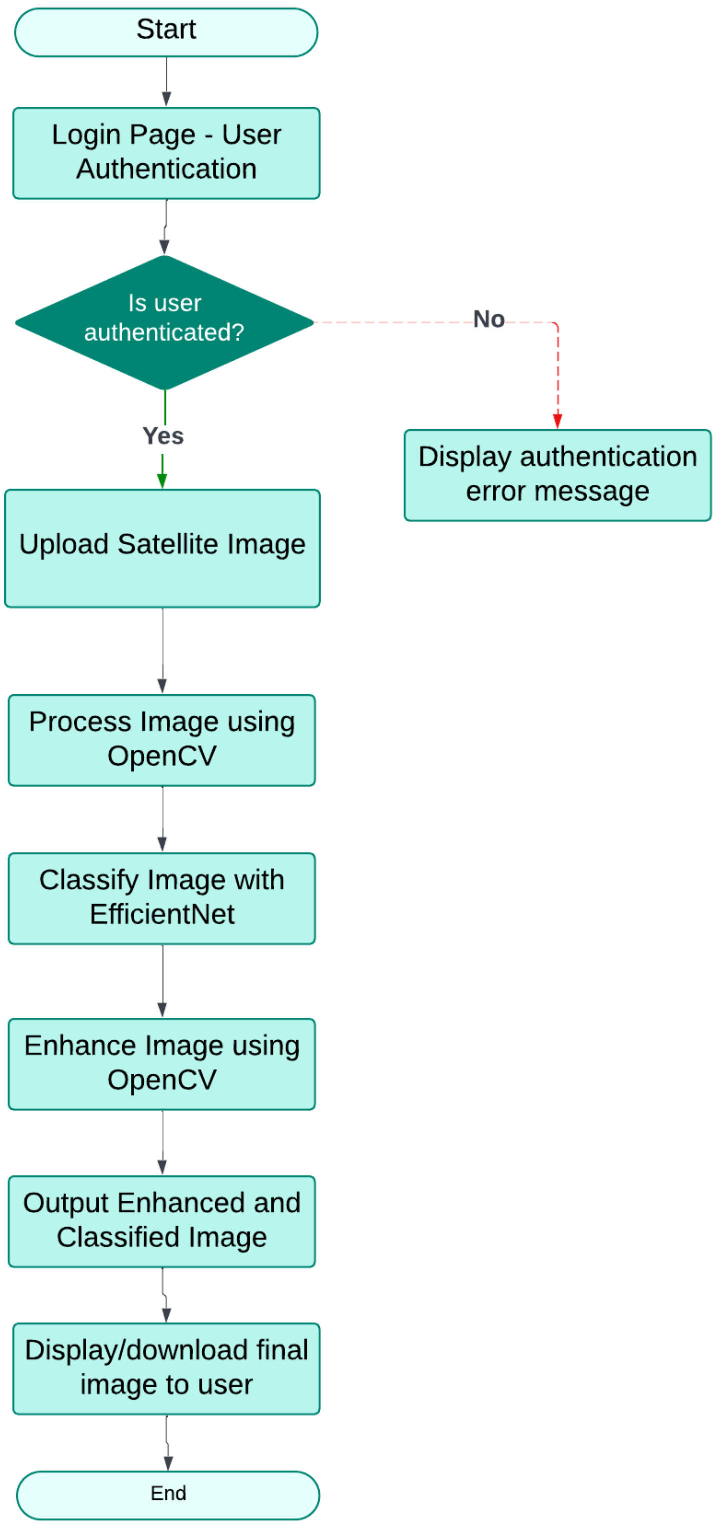

Start
Login Page - User Authentication
Is user authenticated?
No
Display authentication error message
Yes
Upload Satellite Image
Process Image using OpenCV
Classify Image with EfficientNet
Enhance Image using OpenCV
Output Enhanced and Classified Image
Display/download final image to user
End

Figure 1. User flow in the web application

4.2. Hardware & Software Used

Hardware Requirements

- High-Performance Machine: Processor
- RAM: At least 16 GB (32 GB recommended for larger datasets).
- GPU: to support machine learning and deep learning tasks, especially for training GANs and Efficient Net.
- Storage: SSD with at least 500 GB (for datasets, models, and project files)..

Software Requirements

- Operating System: Windows 10/11, macOS, or Linux (e.g., Ubuntu 20.04) depending on your preference.
- Jupyter notebook: A web application that allows you to run live code, embed visualization and explanatory text all in one place.

Programming Languages:

- Python: Primary language for ML/DL models and image processing tasks.
- JavaScript: For web application development (frontend/backend).

Development Frameworks and Libraries:

Machine Learning & Deep Learning:

- TensorFlow/Keras: For implementing EfficientNet.
- PyTorch: Alternative framework for model building and training.
- EuroSAT: A dataset specifically for land use classification using satellite images.

Image Processing:

- OpenCV: For pre-processing, enhancement, and feature extraction.

4.3 Frontend Development

Frontend:

HTML, CSS, JS: To build a responsive and dynamic user interface.

4.4 Backend Development

Backend:

- Node.js: To create APIs for model integration.
- Express.js: For server-side routing and API handling.

Database: MongoDB for storing image data and metadata.

APIs and Frameworks for Deployment:

- FastAPI (Python): For hosting the ML/DL models and exposing APIs.
- TensorFlow.js: To run machine learning models directly in the browser.

Integrated Development Environment (IDE): VS Code, Google Colab for model development and debugging.

Version Control and Collaboration Tools:

GitHub: For version control and team collaboration.

Visualization Tools:

- Matplotlib/Seaborn: For visualizing model performance and image enhancements.
- TensorBoard: For tracking training metrics and results.

Deployment Platforms:

- Heroku: Quick deployment for small-scale applications.
- Docker: To containerize the application for easy deployment across platforms.
- AWS/GCP/Azure: For large-scale production deployment.

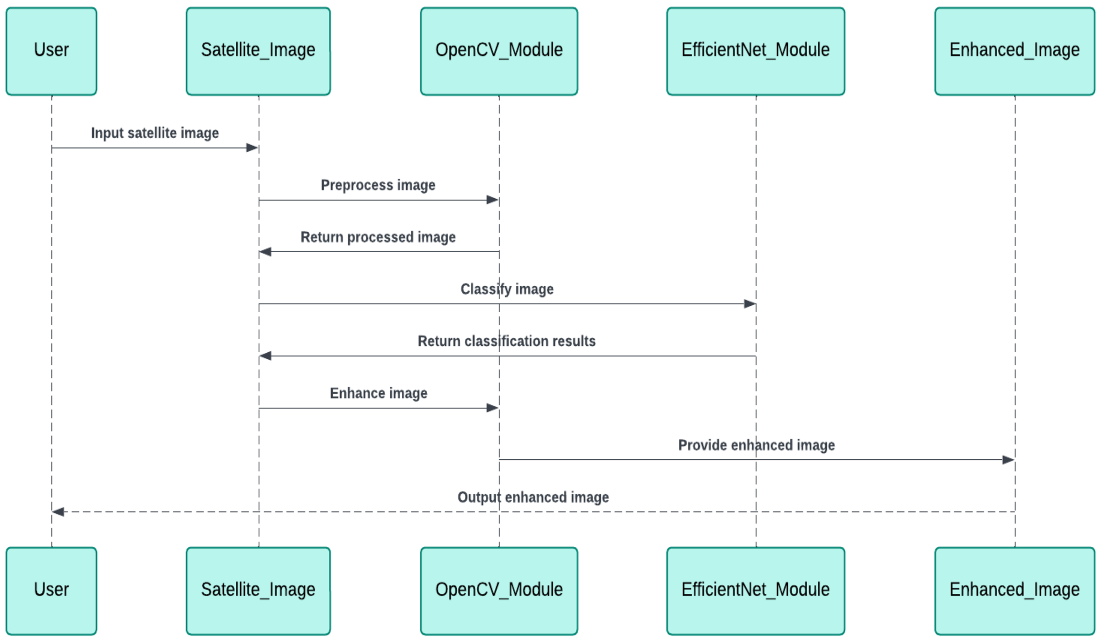


Figure 2. Sequence Diagram of the architecture

Chapter 05

Flowchart and Information Gathering

5.1. Designing of the Flowchart

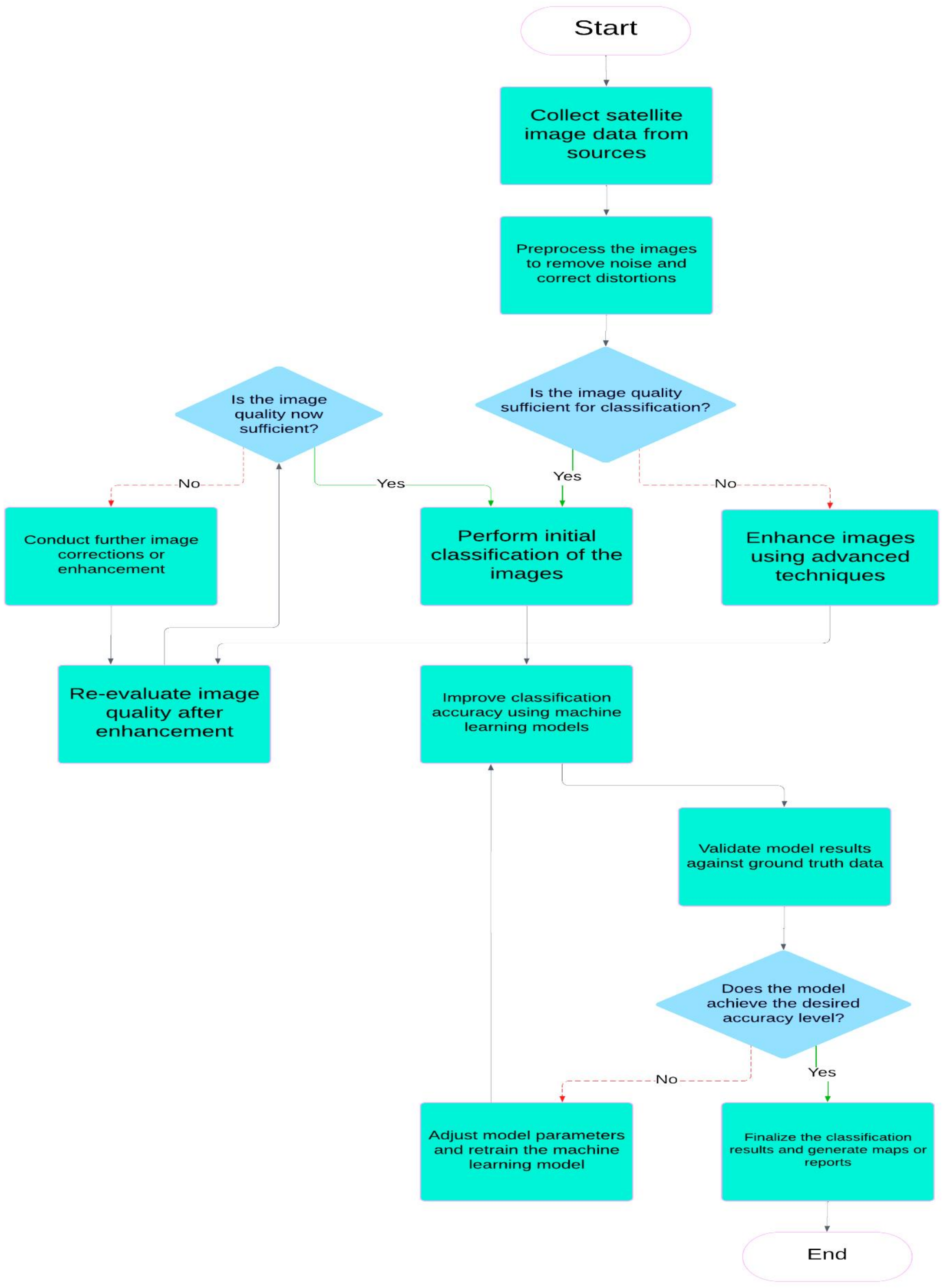

Start
Collect satellite image data from sources
Preprocess the images to remove noise and correct distortions
Is the image quality sufficient for classification?
Yes
No
Is the image quality now sufficient?
No
Yes
Conduct further image corrections or enhancement
Perform initial classification of the images
Enhance images using advanced techniques
Re-evaluate image quality after enhancement
Improve classification accuracy using machine learning models
Validate model results against ground truth data
Does the model achieve the desired accuracy level?
No
Yes
Adjust model parameters and retrain the machine learning model
Finalize the classification results and generate maps or reports
End

Figure 3. Flowchart of the system

5.2. Flowchart Explanation

The flowchart for the Satellite Image Classification and Enhancement project outlines the key phases involved in processing satellite images to achieve accurate classification and resolution enhancement. Each phase is designed to ensure an efficient and systematic workflow. Below is a detailed breakdown of the project:

Step 1: Input Data Acquisition

The pipeline starts with satellite image collection from platforms such as Sentinel, EuroSAT, or Google Earth Engine. These images may include RGB or grayscale bands and are converted into .jpg or .png formats to ensure compatibility with image processing libraries. Due to the variability in resolution, noise, and contrast, standardization is essential before further processing.

Step 2: Image Enhancement (OpenCV)

To improve the quality of raw satellite images, OpenCV is used for enhancement:

- **Resizing:** All images are resized to a standard dimension (e.g., 256x256) to ensure consistency.
- **Noise Reduction:** Filters such as Gaussian Blur, Median Blur, or Bilateral Filtering are applied to reduce noise while preserving structural integrity.
- **Contrast Enhancement**: Techniques like CLAHE are used to improve visibility in low-contrast areas.
- **Edge Enhancement:** Methods like Sobel and Canny help highlight structural elements like roads or rivers.
- **Segmentation:** Optional techniques like k-means clustering may be used to separate distinct regions for better clarity before classification.

These enhancement steps improve image sharpness, brightness, and detail visibility, making the data more suitable for accurate classification.

Step 3: Image Classification (EfficientNet)

After enhancement, the processed images are fed into the EfficientNet model for classification:

- **Model Input:** Enhanced images are formatted and normalized as required by the EfficientNetB0 architecture.

- **Feature Extraction & Classification:** EfficientNet extracts deep spatial and contextual features and classifies each image into categories such as Forest, Water Body, Urban Area, or Agricultural Land.

- **Training & Validation:** The model is trained using labeled satellite image datasets (e.g., EuroSAT) and validated using metrics like accuracy, precision, recall, and F1-score to ensure high reliability.

EfficientNet's compound scaling capability ensures both accuracy and efficiency, making it well-suited for handling high-resolution satellite imagery.

Step 4: Post-Processing & Output Generation

Once classification is complete, the outputs are refined for final use:

- **Label Mapping**: Classified results are mapped back onto the enhanced images to create labelled visual outputs.
- **Artifact Correction:** Any misclassifications or model uncertainties are addressed, if applicable.
- **Output Conversion:** Final classified images are exported in user-friendly formats such as PNG or GeoTIFF for use in GIS tools like QGIS or for web deployment.

Step 5: Evaluation & Performance Metrics

The complete system is evaluated using:

- **Classification Metrics:** Accuracy, Precision, Recall, F1-score
- **Image Quality Metrics:** PSNR (Peak Signal-to-Noise Ratio), SSIM (Structural Similarity Index)
- **Computation Time:** Total runtime and processing efficiency

5.3 Understanding the Purpose of Components

Each component of the **Satellite Image Classification and Enhancement** system plays a crucial role in producing high-quality, classified satellite images. The modular design ensures that individual components can be optimized or extended independently, while the system as a whole works cohesively to deliver accurate and interpretable results. Below is a detailed explanation of the role and significance of each module:

Satellite Image Input

The input module initiates the workflow by acquiring satellite images, which are the foundation for the entire project. These images can be sourced from publicly available datasets such as **Sentinel-2**, **Landsat**, or **EuroSAT**, or directly from custom satellite feeds. The images vary widely in resolution, spectral bands (e.g., RGB, near-infrared), and quality. To standardize these for processing, they are converted into compatible formats like .jpg or .png. The choice and quality of input data significantly affect downstream performance, influencing the results of both enhancement and classification stages. Proper selection and preprocessing of input data ensure better generalizability and robustness.

Preprocessing Module (OpenCV)

The **OpenCV-based preprocessing module** prepares raw images for enhancement and classification by performing essential image enhancement tasks. It improves visual quality and reduces inconsistencies that might hinder model performance:

- **Resizing** images to standard dimensions (e.g., 256x256 pixels) ensures consistent input across all stages.
- **Noise reduction** is achieved using techniques like **Gaussian Blur**, **Median Blur**, or **Bilateral Filtering**, which smoothen images without losing critical details.
- **Contrast enhancement**, particularly with **CLAHE (Contrast Limited Adaptive Histogram Equalization)**, brings out details in poorly illuminated or low-contrast regions.
- **Edge detection** using **Canny** or **Sobel filters** helps highlight structural boundaries such as roads, rivers, or urban outlines.
- **Segmentation** methods like **k-means clustering** or thresholding isolate important regions (e.g., vegetation vs. built-up areas) to improve the clarity of features for classification.

This module ensures that the input data is visually and structurally optimized before classification, laying the groundwork for accurate model predictions.

Feature Extraction

After preprocessing, key image characteristics are extracted to form numerical representations used for classification. Although **EfficientNetB0** inherently performs deep feature extraction as part of its neural architecture, traditional feature descriptors can support exploratory analysis or comparison:

- **Texture features** can be computed using **HOG (Histogram of Oriented Gradients)**.
- **Color distribution** is captured using **color histograms** from OpenCV's cv2.calcHist() function.
- **Shape and edge-based features** assist in identifying terrain types like coastlines or forest boundaries.

While these handcrafted features are optional in the current deep learning-based pipeline, they can be useful for hybrid models or for enhancing understanding of dataset characteristics.

Classification Model

The **EfficientNetB0 model** serves as the primary classification engine. It is a lightweight yet powerful convolutional neural network architecture that balances depth, width, and resolution through compound scaling, making it suitable for resource-efficient deployment on edge or mobile devices.

- **Input**: Receives enhanced satellite images from the OpenCV pipeline.
- **Architecture**: Utilizes mobile inverted bottleneck convolution (MBConv) layers to extract multiscale spatial features efficiently.
- **Classification**: Trained to recognize and categorize land cover types such as **Forest**, **Water Body**, **Urban Area**, and **Agricultural Land**.
- **Training & Evaluation**: Fine-tuned using labeled datasets like **EuroSAT**, with performance validated using metrics such as **accuracy**, **precision**, **recall**, and **F1-score**.

EfficientNetB0 combines high classification accuracy with computational efficiency, making it ideal for scalable real-world deployment in satellite image interpretation tasks.

Enhancement Module

The enhancement module in this project utilizes EfficientNet, a state-of-the-art deep learning architecture, to refine the processing and classification of satellite images. While preprocessing techniques such as CLAHE and noise reduction improve the raw image quality, EfficientNet is employed to extract detailed and meaningful features from these enhanced images. Its compound scaling strategy balances depth, width, and resolution, ensuring efficient handling of high-resolution images. This enables the system to achieve high classification accuracy and robust feature representation, making EfficientNet integral to the enhancement module and its real-world applications in urban planning, environmental monitoring, and disaster management.

Post-Processing

Post-processing refines the outputs of the classification and enhancement models to ensure usability and accuracy. Enhanced images are annotated with labels indicating land-use categories or other classification results, making them easier to interpret. Additionally, any artifacts introduced during enhancement or misclassifications in the prediction stage are corrected during this phase. This step ensures that the final output meets the required quality standards and is suitable for further analysis or presentation. Post-processing also includes saving the outputs in formats like PNG or GeoTIFF for seamless integration with GIS systems.

Output Visualization

The final output is a high-resolution, classified satellite image that visually represents land-use types or other terrain features. This component provides a user-friendly way to analyze and interpret the results, enabling stakeholders to make informed decisions. For example, planners can use the outputs for urban development, while environmental scientists can monitor deforestation or water resources. Visualization tools ensure that the results are accessible and actionable for a wideaudience.

i) Code snippet :

```
category_list = os.listdir(source)
    for category in category_list:
        category_path = os.path.join(source, category)
        category_img_list = os.listdir(category_path)
        random_selected = random.choice(category_img_list)

         img = enhance_image(img)
        result = model.predict(img)
        max_proba_result = max(result[0], key=result[0].get)

        print("--"*20)
        print(f"Ground truth: {category}")
        print(f"Predicted: {max_proba_result}")
        print(
            f"Result: {'Correct' if category == max_proba_result else 'Incorrect'}")

        if is_display:
      display(img, category, max_proba_result, is_colab, save_path)
```

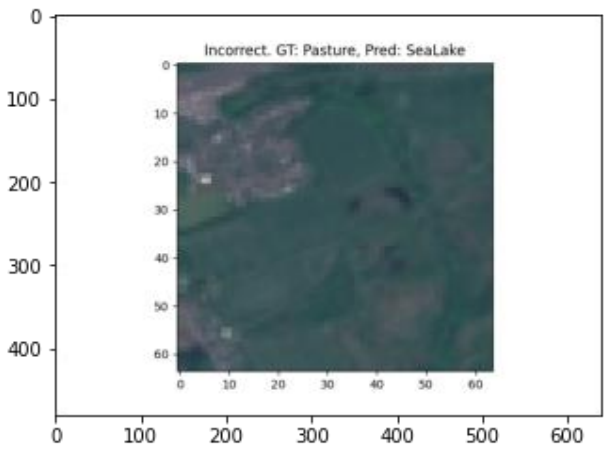
Output:

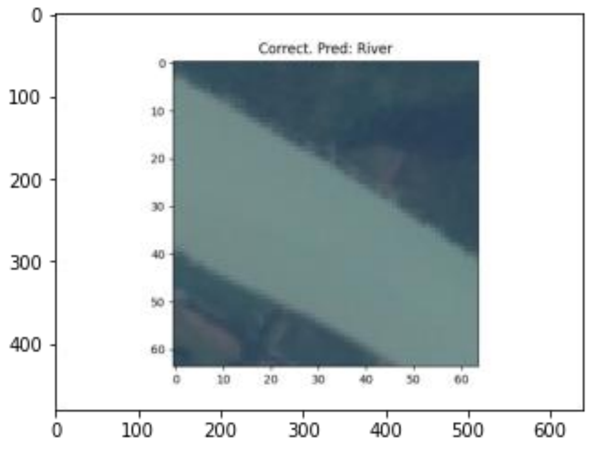
```
Ground truth: SeaLake
Predicted: SeaLake
Result: Correct
----------------------------------------
Ground truth: Highway
Predicted: SeaLake
Result: Incorrect
----------------------------------------
Ground truth: PermanentCrop
Predicted: SeaLake
Result: Incorrect
----------------------------------------
Ground truth: River
Predicted: River
Result: Correct
----------------------------------------
Ground truth: Residential
Predicted: SeaLake
Result: Incorrect
----------------------------------------
Ground truth: Industrial
Predicted:Industrial
Result: Correct
```

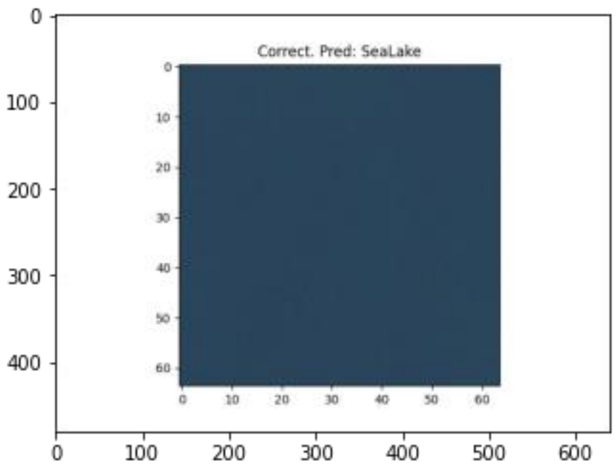
ii) Code snippet:

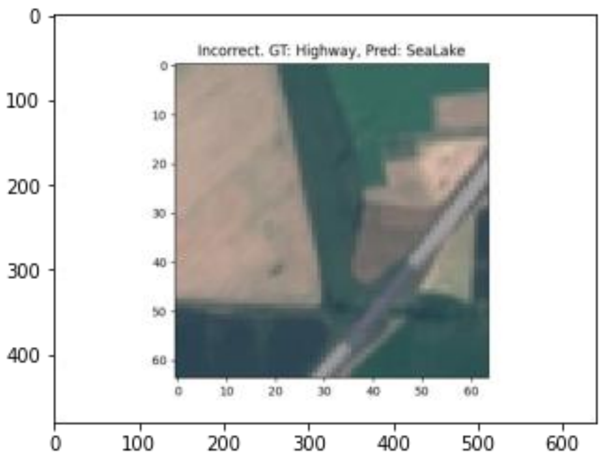
```
from IPython.display import clear_output
import matplotlib.pyplot as plt
from numpy.random import randn
from time import sleep
from cv2 import cv2
import os
%matplotlib inline

target_path = "predict_results"
arr = os.listdir(target_path)
for x in arr:
  img = cv2.imread(os.path.join(target_path, x))
  img_cvt=cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
  plt.imshow(img)
  plt.show()
  sleep(1)
```

Output:



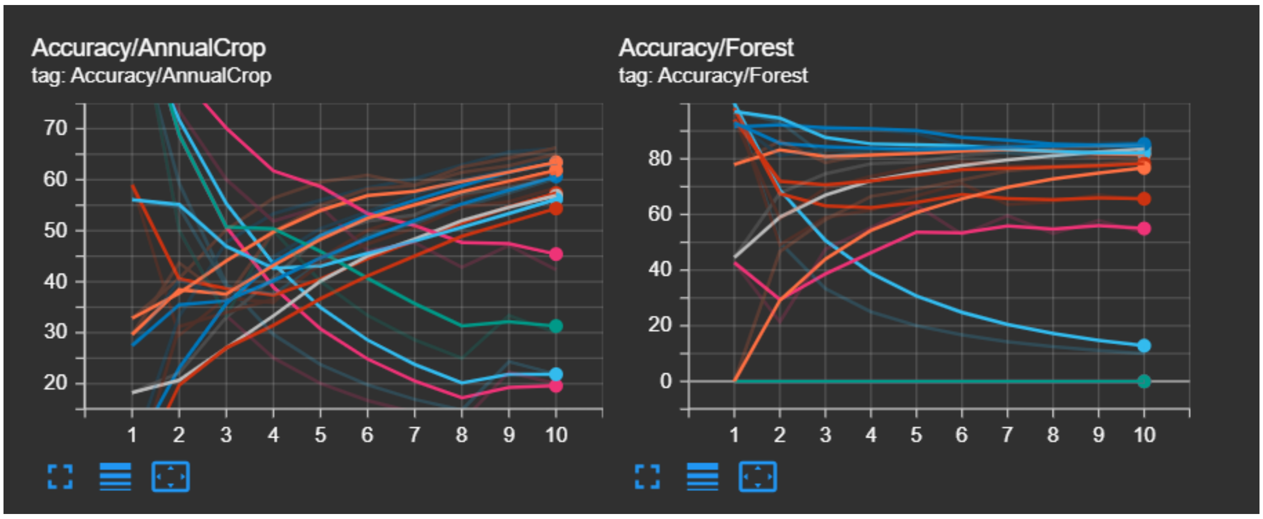


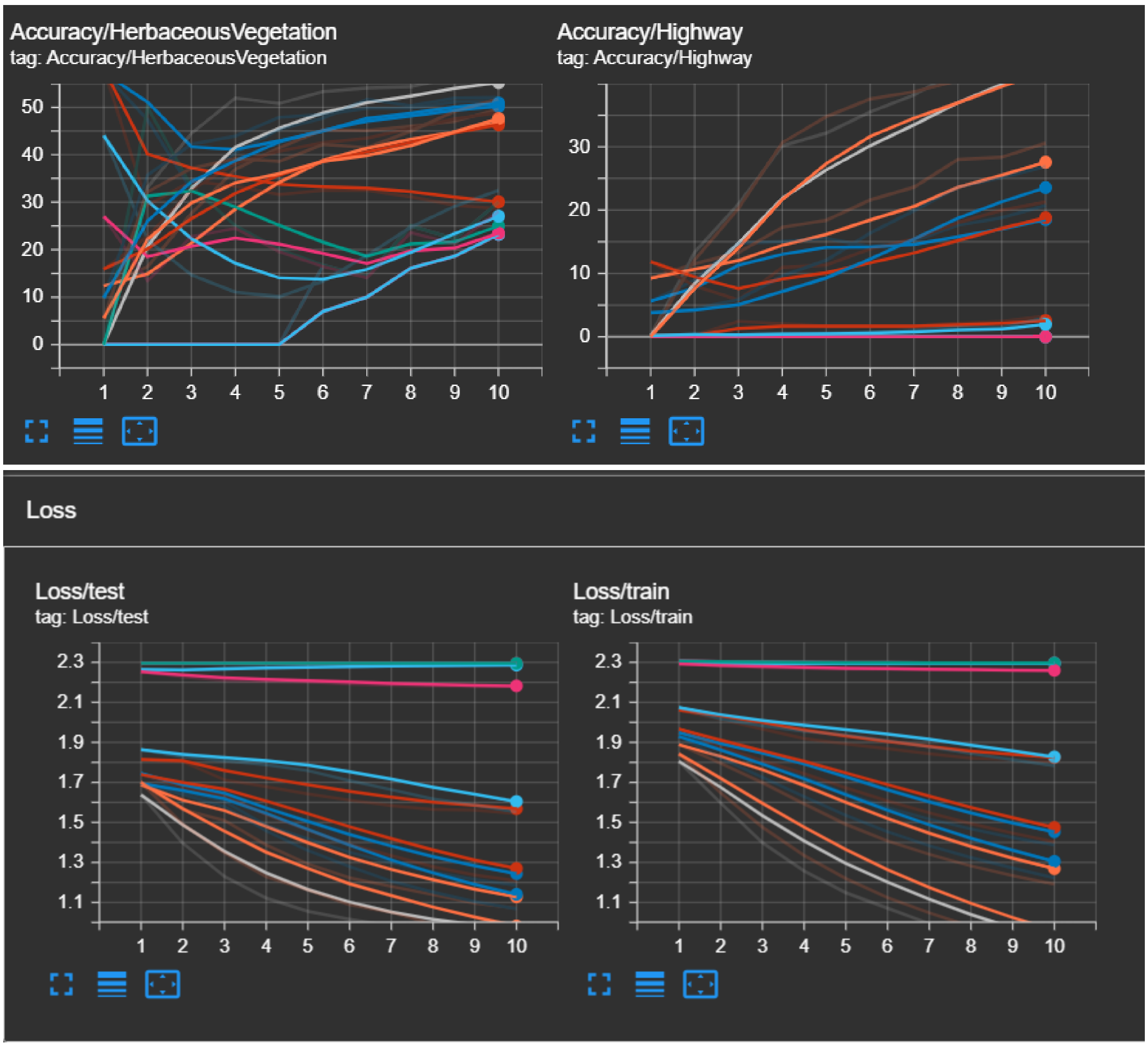




Let's examine the results:

```
%load_ext tensorboard
%tensorboard --logdir runs
```





Modular Integration

The modular design of the system ensures that each component operates independently while contributing to the overall workflow. This approach allows for easy updates or replacements of individual modules, such as upgrading the classification algorithm. This flexibility makes the system scalable and adaptable to different datasets or evolving requirements. Modular integration not only simplifies the development process but also ensures that future improvements can be incorporated without overhauling the entire pipeline.

By understanding the purpose of these components in detail, the interconnectivity and functionality of the system become apparent. Each component contributes uniquely to the project's goals, creating a robust and effective solution for satellite image classification and enhancement. This detailed breakdown underscores the importance of every stage in achieving accurate and high-quality results.

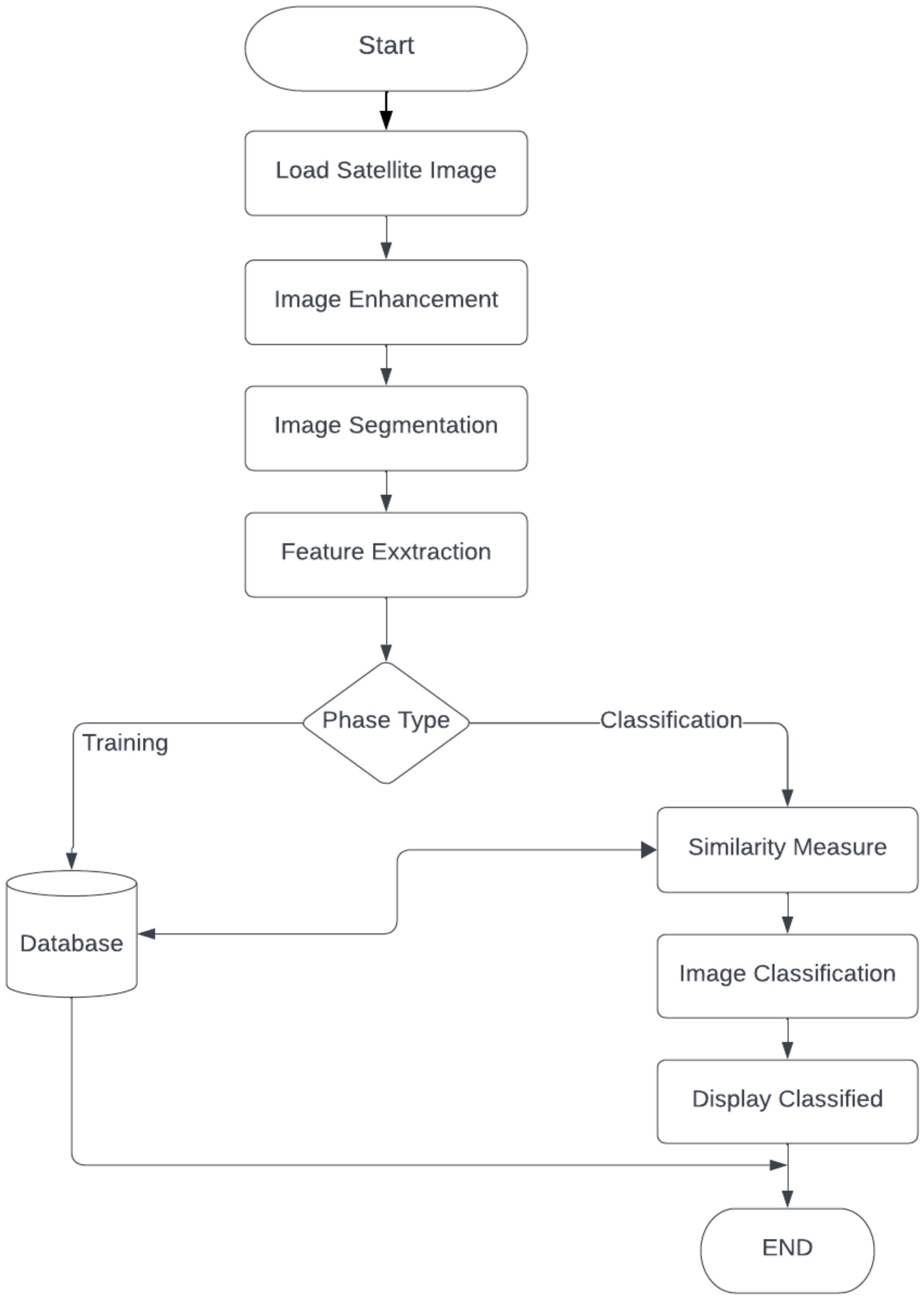


Figure 4. Block diagram of components

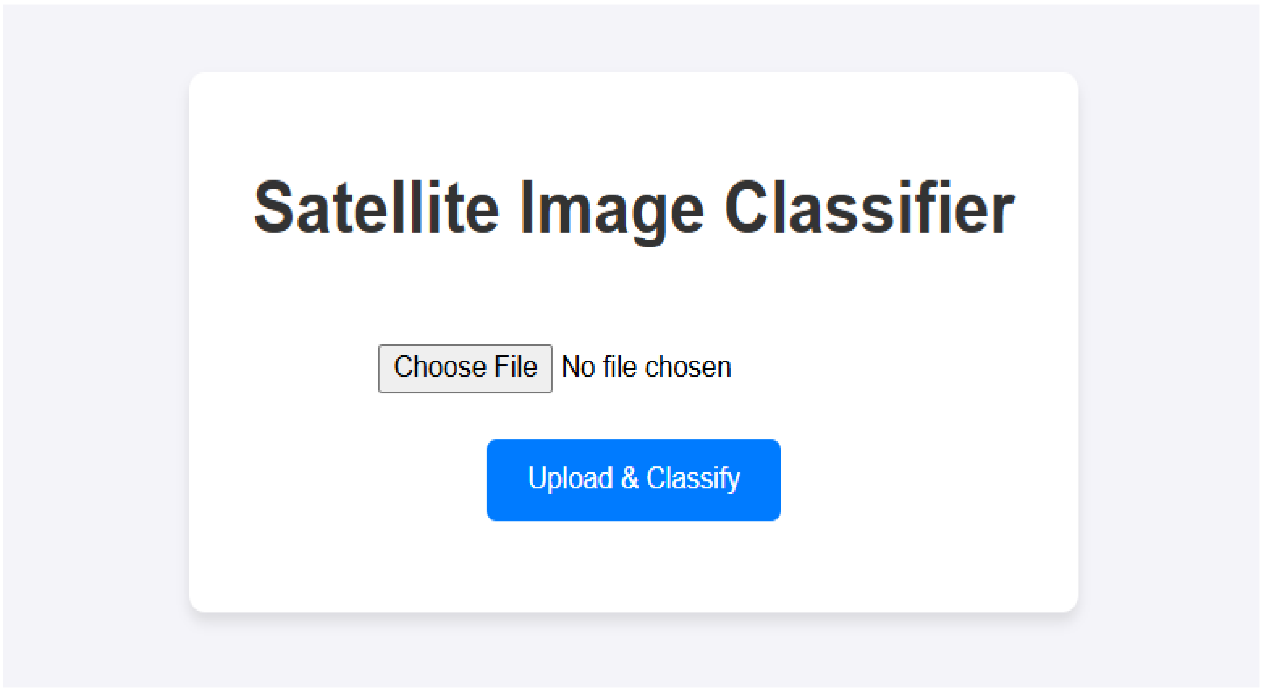
5.4. UI Implementation

The User Interface (UI) for the system is developed using HTML, CSS, and JavaScript, providing a simple and interactive web-based platform. Users can upload satellite images through the UI, which are then sent to a backend API for classification.

The backend is built in Python using frameworks like Flask or FastAPI, and it integrates the trained image classification model. Upon receiving the image from the frontend, the API performs image enhancement (using OpenCV techniques such as CLAHE, sharpening, and denoising), followed by classification using the EfficientNet-B0 model.

The predicted class label is then returned to the frontend and displayed to the user. This setup ensures a smooth and responsive user experience and allows real-time interaction with the classification model through a modern web interface.

This modular structure also makes the system scalable and easily deployable as a web service.



Chapter 06

6.1. Conclusion

The Satellite Image Classification and Enhancement project successfully demonstrates the application of modern computer vision techniques, combining OpenCV for preprocessing and classification with EfficientNet for enhancement. By leveraging OpenCV, we implemented efficient image processing pipelines for tasks such as feature extraction, segmentation, and classification, enabling accurate identification of different land covers and terrain types. The integration of EfficientNet allowed us to enhance the quality of low-resolution satellite images, improving clarity and detail, which is critical for tasks like environmental monitoring, urban planning, and disaster management.

This project showcases the potential of combining traditional image processing methods with deep learning frameworks to address complex challenges in satellite imagery. The results indicate significant improvements in both classification accuracy and image resolution, validating the effectiveness of our approach. Furthermore, the modular design ensures scalability and adaptability, making the system applicable to a wide range of datasets and real-world scenarios.

Future work could focus on refining the efficientNet model for even higher resolution outputs, integrating real-time data streams for dynamic classification, and extending the application to multi-spectral and hyper-spectral data. This project highlights the transformative role of AI in remote sensing, paving the way for more efficient and insightful analysis of satellite imagery.

6.2. Challenges

While developing this project, several challenges were encountered:

- **Data Quality and Availability:** Access to high-resolution, labeled satellite datasets was limited. Many publicly available datasets lacked diversity, affecting the model's ability to generalize well to unseen geographical regions.
- **Computational Resources**: Training deep learning models like EfficientNet requires significant GPU resources. Without access to high-end GPUs or cloud infrastructure, training time increased and model experimentation was restricted.
- **Balancing Enhancement and Accuracy:** Image enhancement methods improved feature visibility but sometimes introduced artifacts that negatively impacted classification accuracy. Fine-tuning enhancement parameters was necessary to avoid over-processing.
- **Integration Complexity:** Combining enhancement (OpenCV) and classification (EfficientNet) in a seamless pipeline required careful design of preprocessing steps, input formatting, and consistent scaling of images.
- **Real-time Deployment:** Integrating heavy models into a real-time web application posed performance challenges, particularly on resource-constrained environments or browsers.

6.3 Future Scope of the Project

The successful implementation of this **satellite image enhancement and classification system** opens multiple promising avenues for future expansion, both in terms of research innovation and practical deployment:

1. **Real-Time Image Processing and Classification:** The system can be further optimized for real-time deployment by applying model compression techniques such as pruning, quantization, or knowledge distillation to the EfficientNetB0 architecture, making it lightweight and efficient enough to run on edge devices like drones, mobile units.

2. **Stream-based Classification:** The classification pipeline can be extended to support continuous processing of live satellite image streams from sources like CubeSats or UAVs, enabling immediate and automated analysis for time-sensitive use cases such as disaster response, real-time traffic monitoring, and environmental event detection.

3. **Multispectral and Hyperspectral Image Support:** The model can be enhanced to incorporate multispectral and hyperspectral imagery by integrating additional bands such as near-infrared or thermal channels.

4. **Cross-Region and Domain Adaptation**: To improve generalization across diverse geographic regions, domain adaptation methods can be implemented to make the model robust against variations in vegetation, climate, and terrain.

5. **Self-Supervised and Semi-Supervised Learning**: The reliance on large manually labelled datasets can be reduced by incorporating self-supervised learning strategies that allow the model to learn from unlabelled satellite imagery, and by leveraging semi-supervised approaches to train the model effectively with a combination of labelled and unlabelled data.

6. **Advanced Enhancement Techniques:** The image enhancement pipeline can be improved by integrating deep learning models such as SRGAN or Real-ESRGAN for super-resolution and denoising autoencoders for noise reduction.

7. **Geospatial Integration:** The classified output can be integrated with geospatial mapping tools such as QGIS or ArcGIS to visualize results on real-world maps, and extended to support spatiotemporal analysis for monitoring changes over time, including urban growth, deforestation, and seasonal variations in land use.

8. **Interactive Web and Mobile Interface**: An intuitive and responsive user interface can be developed using modern frontend technologies like React, Plotly, or D3.js for interactive map visualization, and complemented by a mobile application that allows field researchers to upload images and receive real-time classification feedback in remote areas.

References

1. A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” *Advances in Neural Information Processing Systems*, 2012.

2. K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” arXiv:1409.1556, 2014.

3. K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” *Proceedings of the IEEE CVPR*, 2016.

4. B. Zhang, X. Du, and L. Zhang, “Deep learning-based classification of high-resolution satellite images,” *IEEE JSTARS*, vol. 9, no. 5, pp. 1868–1878, 2016.

5. P. Ayyub, R. Thomas, and S. Alam, “Satellite image classification using transfer learning,” *Remote Sensing Applications: Society and Environment*, vol. 17, 2020.

6. S. Pizer et al., “Adaptive histogram equalization and its variations,” *Computer Vision, Graphics, and Image Processing*, vol. 39, no. 3, 1987.

7. K. Zhang et al., “Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising,” *IEEE Transactions on Image Processing*, 2017.

8. C. Chen et al., “Learning to See in the Dark,” *Proceedings of the IEEE CVPR*, 2018

.

9. C. Ledig et al., “Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network,” *IEEE CVPR*, 2017.

10. O. Kupyn et al., “DeblurGAN: Blind Motion Deblurring Using Conditional Adversarial Networks,” *IEEE CVPR*, 2018

11. M. Tan and Q. Le, “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks,” *ICML*, 2019.

Appendix (Program Code)

1. dataset.py file

```
import os
import imageio
import numpy as np
import torch
from skimage import img_as_float32, img_as_ubyte
from skimage.transform import resize
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from tqdm import tqdm
from torchvision.datasets.utils import download_and_extract_archive
from colorama import Fore


class EurosatDataset(torch.utils.data.Dataset):
    """
    EuroSAT: Land Use and Land Cover Classification with Sentinel-2
    Eurosat is a dataset and deep learning benchmark for land use and land cover classification.
The dataset is based on Sentinel-2 satellite images covering 13 spectral bands and consisting out
of 10 classes with in total 27,000 labeled and geo-referenced images.
    """

    def __init__(self, is_train, root_dir="data/EuroSAT/", transform=None, seed=42,
download=False):
        """
        EurosatDataset

        Args:
            is_train (bool): If true returns training set, else test set.
            root_dir (str, optional): Root directory of dataset. Defaults to "data/EuroSAT/2750/".
            transform ([type], optional): Optional transform to be applied on a sample. Defaults to
None.
            seed (int, optional): Seed used for train/test split. Defaults to 42.
            download (bool, optional): If True, downloads the dataset from the internet and puts it in
root directory. If dataset is already downloaded it is not downloaded again. Defaults to False.
        """

        self.seed = seed
        self.root_dir = root_dir
        self.transform = transform
        self.is_train = is_train
        self.download = download
        self.size = [64, 64]
        self.num_channels = 3
        self.num_classes = 10
        self.test_ratio = 0.2
        self.N = 27000
        self.extaraced = '2750'
        self._load_data()

    def _load_data(self):
        images = np.zeros(
            [self.N, self.size[0], self.size[1], 3], dtype="uint8")
        labels = []
        filenames = []

        if self.download:
            self.download_dataset()

        if not self._check_exists():
            raise RuntimeError(
                "Dataset not found. You can use download=True to download it"
            )

        i = 0
        data_dir = os.path.join(self.root_dir, self.extaraced)

        with tqdm(os.listdir(data_dir), bar_format="{l_bar}%s{bar}%s{r_bar}" % (Fore.GREEN,
Fore.RESET)) as dir_bar:
            for item in dir_bar:
                f = os.path.join(data_dir, item)
                if os.path.isfile(f):
                    continue
                for subitem in os.listdir(f):
                    sub_f = os.path.join(f, subitem)
                    filenames.append(sub_f)

                    # a few images are a few pixels off, we will resize them
                    image = imageio.imread(sub_f)
                    if image.shape[0] != self.size[0] or image.shape[1] != self.size[1]:
                        # print("Resizing image...")
                        image = img_as_ubyte(
                            resize(
                                image, (self.size[0], self.size[1]), anti_aliasing=True)
                        )
                    images[i] = img_as_ubyte(image)
                    i += 1
                    labels.append(item)

                dir_bar.set_description(
                    f"{'Train' if self.is_train else 'Test'} images are reading..")
                dir_bar.set_postfix(category=item)

        labels = np.asarray(labels)
        filenames = np.asarray(filenames)

        # sort by filenames
        images = images[filenames.argsort()]
        labels = labels[filenames.argsort()]

        # convert to integer labels
        label_encoder = preprocessing.LabelEncoder()
        label_encoder.fit(np.sort(np.unique(labels)))
        labels = label_encoder.transform(labels)
        labels = np.asarray(labels)
        # remember label encoding
        self.label_encoding = list(label_encoder.classes_)

        # split into a is_train and test set as provided data is not presplit
        x_train, x_test, y_train, y_test = train_test_split(
            images,
            labels,
            test_size=self.test_ratio,
            random_state=self.seed,
            stratify=labels,
        )

        if self.is_train:
            self.data = x_train
            self.targets = y_train
        else:
            self.data = x_test
            self.targets = y_test

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        if torch.is_tensor(idx):
            idx = idx.tolist()

        img = self.data[idx]
        # doing this so that it is consistent with all other datasets
        # to return a PIL Image
        # img = Image.fromarray(img)

        if self.transform:
            img = self.transform(img)

        image = np.asarray(img / 255, dtype="float32")

        return image.transpose(2, 0, 1), self.targets[idx]

    def _check_exists(self) -> bool:
        return os.path.exists(self.root_dir)

    def download_dataset(self) -> None:
        if self._check_exists():
            return

        os.makedirs(self.root_dir, exist_ok=True)
        download_and_extract_archive(
            "https://madm.dfki.de/files/sentinel/EuroSAT.zip",
            download_root=self.root_dir,
            md5="c8fa014336c82ac7804f0398fcb19387",
        )

if __name__ == '__main__':
    dset = EurosatDataset(is_train=False, seed=42, download=True)
    print(len(dset))
    print(dset.label_encoding)
```

2. model.py

```
from typing import Dict, List
import numpy as np
import torch
from torch import nn
import torchsummary
from torchvision import models

class Classifier(nn.Module):
    """
    Image Classifier using EfficientNet-B0
    """
    # pylint: disable=no-member
    # pylint: disable=not-callable
    __default_config__ = {
        "input_size": 224,
        "labels": ['AnnualCrop', 'Forest', 'HerbaceousVegetation', 'Highway', 'Industrial', 'Pasture',
'PermanentCrop', 'Residential', 'River', 'SeaLake'],
        "loss_function": "crossentropy",
        'dropout': 0.2,
    }

    def __init__(self, config: Dict = None):
        super(Classifier, self).__init__()

        self.config = self.__default_config__ if config is None else config

        self.labels = list(self.config["labels"])
        self.num_classes = len(self.labels)

        if self.config['loss_function'] == 'crossentropy':
            self.loss_fcn = nn.CrossEntropyLoss()
        else:
            raise ValueError('Unknown criterion')

        # Load EfficientNet-B0 backbone
        self.backbone =
models.efficientnet_b0(weights=models.EfficientNet_B0_Weights.DEFAULT)
        in_features = self.backbone.classifier[1].in_features
        self.backbone.classifier = nn.Sequential(
            nn.Dropout(p=self.config['dropout']),
            nn.Linear(in_features, self.num_classes)
        )

    def forward(self, inputs: torch.Tensor) -> torch.Tensor:
        return self.backbone(inputs)

    def predict(self, inputs: np.ndarray) -> List[str]:
        data = self.preprocess_input(inputs)
        preds = torch.softmax(self.backbone(data), axis=-1)
        outputs = []
        for scores in preds.detach().cpu().numpy().tolist():
            output = {label: round(score, 3) for score, label in zip(scores, self.labels)}
            outputs.append(output)
        return outputs

    @classmethod
    def from_pretrained(cls, model_path: str, *args, **kwargs) -> nn.Module:
        state_dict = torch.load(model_path)
        pretrained_model = cls(config=state_dict['config'], *args, **kwargs)
        pretrained_model.load_state_dict(state_dict['state_dict'])
        return pretrained_model

    @classmethod
    def summarize(cls, input_size=(3, 224, 224)):
        return torchsummary.summary(cls().to(device), input_size=input_size)

    def to(self, device: str, *args, **kwargs):
        self.device = device
        return super().to(device, *args, **kwargs)

    def preprocess_input(self, input_array: np.ndarray) -> torch.Tensor:
        data = input_array.copy()
        if len(data.shape) == 3:
            new_inputs = torch.from_numpy(np.expand_dims(data, axis=0)).float().permute(0, 3, 1,
2).contiguous()
        elif len(data.shape) == 4:
            new_inputs = torch.from_numpy(data).float().permute(0, 3, 1, 2).contiguous()
        else:
            raise AssertionError(f"input shape not supported: {data.shape}")
        return new_inputs.to(self.device)

    def loss(self, y_pred, y_true):
        return self.loss_fcn(y_pred, y_true)


if __name__ == "__main__":
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model = Classifier().to(device)

    input_array = np.random.rand(224, 224, 3)
    result = model.predict(input_array)
    print(result)
```

3. train.py

```
# import torch.nn.init
from colorama import Fore
from dataset import EurosatDataset
from model import Classifier
from sklearn.metrics import accuracy_score
from torch.autograd import Variable
from torch.utils.tensorboard import SummaryWriter
from tqdm import tqdm
import argparse
import numpy as np
import random
import torch
import torch.optim as optim
import warnings
import os
# suppress warnings
warnings.filterwarnings("ignore")


def parse_arguments():
    arg = argparse.ArgumentParser()

    arg.add_argument("--epoch", type=int, default=10)
    arg.add_argument(
        "--device", type=int, default=torch.device("cuda" if torch.cuda.is_available() else "cpu"),
choices=['cuda', 'cpu'])
    arg.add_argument("--save_dir", type=str, default="./saved_models")
    arg.add_argument("--data_dir", type=str, default="data/EuroSAT/")

    arg.add_argument("--batch_size", type=int, default=128,
                     help="total number of batch size of labeled data")

    arg.add_argument("--eval_batch_size", type=int, default=256,
                     help="batch size of evaluation data loader")

    arg.add_argument("--criterion", type=str,
                     default="crossentropy", choices=['crossentropy'])
    arg.add_argument("--optimizer", type=str,
                     default="sgd", choices=['sgd', 'adam'])

    arg.add_argument("--learning_rate", type=float, default=0.01)
    arg.add_argument("--momentum", type=float, default=0.9)
    arg.add_argument("--weight_decay", type=float, default=5e-4)
    arg.add_argument("--dropout", type=float, default=0.0,
                     choices=[0.0, 0.5, 0.7, 0.9, 0.99, 0.999])
    arg.add_argument("--num_workers", type=int, default=3)
    arg.add_argument("--seed", type=int, default=42)

    return arg.parse_args()


def softmax(x):
    return np.exp(x)/sum(np.exp(x))
```

```
def accuracy(gt_S, pred_S):
    _, alp = torch.max(torch.from_numpy(pred_S), 1)
    return accuracy_score(gt_S, np.asarray(alp))



def main(args):

    # pylint: disable=not-callable

    # Seed for reproducibility

    torch.manual_seed(args.seed)
    np.random.seed(args.seed)
    torch.cuda.manual_seed(args.seed)
    random.seed(args.seed)

    writer = SummaryWriter(
        comment=f"Learn_{args.learning_rate}_Drop{args.dropout}")

    # Construct Dataset

    train_dataset = EurosatDataset(
        is_train=True, seed=args.seed, root_dir=args.data_dir)

    print(
        f'Number of data on Train Dataset is {len(train_dataset)}')

    test_dataset = EurosatDataset(
        is_train=False, seed=args.seed, root_dir=args.data_dir)

    print(
        f'Number of data on Test Dataset is {len(test_dataset)}')

    train_loader = torch.utils.data.DataLoader(
        train_dataset,
        batch_size=args.batch_size,
        shuffle=True,
        num_workers=args.num_workers,
        drop_last=True,
    )

    test_loader = torch.utils.data.DataLoader(
        test_dataset,
        batch_size=args.eval_batch_size,
        shuffle=True,
        num_workers=args.num_workers,
        drop_last=True,
    )

    print(
        f'The images split in train and test is based on the seed {args.seed}')

    config = {
        "input_size": train_dataset.size[0],
        "labels": train_dataset.label_encoding,
        "loss_function": args.criterion,
        'dropout': args.dropout,
    }
    model = Classifier(config=config)
    model = model.to(args.device)

    if args.optimizer == 'adam':
        optimizer = optim.Adam(model.parameters(), lr=args.learning_rate,
                               weight_decay=args.weight_decay)
    elif args.optimizer == 'sgd':
        optimizer = optim.SGD(model.parameters(), lr=args.learning_rate,
                              momentum=args.momentum, weight_decay=args.weight_decay)
    else:
        raise ValueError('Unknown optimizer')

    test_losses = []
    train_losses = []
    test_acc = []
    train_acc = []

    best_acc = 0
    correct_pred = {classname: 0 for classname in train_dataset.label_encoding}
    total_pred = {classname: 0 for classname in train_dataset.label_encoding}

    if not os.path.exists(args.save_dir):
        os.makedirs(args.save_dir, exist_ok=True)

    print(f'Start training with {args.epoch} epochs')

    for e in range(1, args.epoch + 1):
        with tqdm(train_loader, unit="batch", bar_format="{l_bar}%s{bar}%s{r_bar}" %
(Fore.BLUE, Fore.RESET)) as train_epoch:
            model.train()
            for data, target in train_epoch:
                # get the inputs; train_epoch is a list of [data, target]
                data, target = (
                    Variable(data.to(args.device)),
                    Variable(target.to(args.device)),
                )
                # zero the parameter gradients
                optimizer.zero_grad()
                # forward + backward + optimize
                output = model(data)
                loss = model.loss(output, target)
                loss.backward()
                optimizer.step()

                train_losses = np.append(train_losses, loss.item())
                pred = output.data.cpu().numpy()  # [0]
                pred = softmax(pred)
                gt = target.data.cpu().numpy()  # [0]
                train_acc = np.append(train_acc, accuracy(gt, pred))

                train_epoch.set_description(f"Epoch {e}")
                train_epoch.set_postfix(
                    loss=loss.item(), acc=train_acc[-1], mean_loss=np.mean(train_losses),
mean_acc=np.mean(train_acc))

            writer.add_scalar('Loss/train', np.mean(train_losses), e)
            writer.add_scalar('Accuracy/train', np.mean(train_acc), e)

        with tqdm(test_loader, unit="batch", bar_format="{l_bar}%s{bar}%s{r_bar}" %
(Fore.RED, Fore.RESET)) as test_epoch:
            model.eval()
            # since we're not training, we don't need to calculate the gradients for our outputs
            with torch.no_grad():
                for data, target in test_epoch:
                    data, target = (
                        Variable(data.to(args.device)),
                        Variable(target.to(args.device)),
                    )

                    # calculate outputs by running images through the network
                    output = model(data)
                    loss = model.loss(output, target)
                    test_losses = np.append(test_losses, loss.item())

                    # the class with the highest conf is what we choose as prediction
                    _, pred = torch.max(output, 1)

                    for label, prediction in zip(target, pred):
                        if label == prediction:
```

```
                    correct_pred[train_dataset.label_encoding[
                        label]] += 1
                total_pred[train_dataset.label_encoding[label]] += 1

            pred = output.data.cpu().numpy()  # [0]
            pred = softmax(pred)
            gt = target.data.cpu().numpy()  # [0]
            test_acc = np.append(test_acc, accuracy(gt, pred))

            test_epoch.set_description(f"Test")
            test_epoch.set_postfix(
                loss=loss.item(), acc=test_acc[-1], mean_loss=np.mean(test_losses),
mean_acc=np.mean(test_acc))

        writer.add_scalar('Loss/test', np.mean(test_losses), e)
        writer.add_scalar('Accuracy/test', np.mean(test_acc), e)
        state = {
            'config': config,
            'state_dict': model.state_dict(),
        }
        if np.mean(test_acc) > best_acc:
            best_acc = np.mean(test_acc)
            torch.save(state,
                       f"{args.save_dir}/model_best.pth")

        torch.save(state,
                   f"{args.save_dir}/model_epoch{e}_acc{round(np.mean(test_acc),2)}.pth")
        for classname, correct_count in correct_pred.items():
            t_accuracy = 100 * float(correct_count) / total_pred[classname]
            # print(f'Accuracy for class: {classname:5s} is {t_accuracy:.1f} %')
            writer.add_scalar(f"Accuracy/{classname}", t_accuracy, e)

    print('Finished Training')
    print(f'Saving the model on {args.save_dir}')


if __name__ == "__main__":
    args = parse_arguments()
    main(args)
```

4. predict.py

```
import random
import os
import argparse
from cv2 import cv2
from model import Classifier
from matplotlib import pyplot as plt


def parse_arguments():
    arg = argparse.ArgumentParser()
    arg.add_argument('--source', '-s', type=str, default='data/EuroSAT/2750',
                     help="give main source directory")
    arg.add_argument('--device', '-d', default='cuda',
                     type=str, choices=['cuda', 'cpu'])
    arg.add_argument('--model_path', '-m', type=str, default='saved_models/model_best.pth',
                     help="give saved model path")
    arg.add_argument('--display', action='store_true')
    arg.add_argument('--colab', action='store_true')
    arg.add_argument('--save_path', '-sa', type=str,
                     default='predict_results/')

    return vars(arg.parse_args())




def display(img, gt, pred, is_colab, save_path):
    if gt == pred:
        text = f"Correct. Pred: {pred}"

    else:
        text = f"Incorrect. GT: {gt}, Pred: {pred}"

    if is_colab:
        plt.imshow(img)
        plt.title(text)
        plt.savefig(f'{save_path}/{gt}.png')
    else:
        cv2.imshow(f'{text}', img)
        cv2.waitKey(0)

def enhance_image(img):

import numpy as np
# Convert BGR (OpenCV) to RGB
```

```
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Apply Gaussian blur
img_blur = cv2.GaussianBlur(img_rgb, (3, 3), 0)

# Sharpening kernel
kernel = np.array([[0, -1, 0],
                   [-1, 5, -1],
                   [0, -1, 0]])
sharpened = cv2.filter2D(img_blur, -1, kernel)

# Resize to 224x224 (EfficientNet-B0 input)
resized = cv2.resize(sharpened, (224, 224))
return resized


 if __name__ == "__main__":
    kwargs = parse_arguments()
    device = kwargs.pop('device')
    source = kwargs.pop('source')
    model_path = kwargs.pop('model_path')
    is_display = kwargs.pop('display')
    is_colab = kwargs.pop('colab')
    save_path = kwargs.pop('save_path')
    random.seed(42)
    if not os.path.exists(save_path):
        os.makedirs(save_path, exist_ok=True)

    model = Classifier()
    model = model.from_pretrained(model_path).to(device)

    category_list = os.listdir(source)
    for category in category_list:
        category_path = os.path.join(source, category)
        category_img_list = os.listdir(category_path)
        random_selected = random.choice(category_img_list)

         img = enhance_image(img)
        result = model.predict(img)
        max_proba_result = max(result[0], key=result[0].get)

        print("--"*20)
        print(f"Ground truth: {category}")
        print(f"Predicted: {max_proba_result}")
        print(
            f"Result: {'Correct' if category == max_proba_result else 'Incorrect'}")

        if is_display:
            display(img, category, max_proba_result, is_colab, save_path)
```

5. User Interface Implementation

5.1. fileUpload.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Satellite Image Classifier</title>
</head>
<body>
  <div class="container">
    <h1>Satellite Image Classifier</h1>
    <form id="uploadForm">
      <input type="file" id="imageInput" name="image" accept="image/*" required />
      <button type="submit">Upload & Classify</button>
    </form>
    <div id="result">
      <p id="classification"></p>
      <img id="classifiedImage" src="" alt="" style="display: none;">
    </div>
  </div>


  <script>
    const uploadForm = document.getElementById('uploadForm');
    const resultDiv = document.getElementById('result');
    const classificationText = document.getElementById('classification');
    const classifiedImage = document.getElementById('classifiedImage');

    uploadForm.addEventListener('submit', async (e) => {
      e.preventDefault();

      const imageInput = document.getElementById('imageInput');
      const file = imageInput.files[0];

      if (!file) {
        classificationText.textContent = 'Please select an image.';
        return;
      }

      const formData = new FormData();
      formData.append('image', file);

      try {
```

```
                const response = await fetch('http://localhost:3000/upload', {
                    method: 'POST',
                    body: formData,
                });

                if (!response.ok) {
                    throw new Error('Error uploading the image');
                }

                const data = await response.json();
                classificationText.innerHTML = `<strong>Classification:</strong>
${data.classification} <br> <strong>Confidence:</strong> ${data.confidence}`;
                classifiedImage.src = data.imageUrl;
                classifiedImage.style.display = 'block';
            } catch (error) {
                console.error(error);
                classificationText.textContent = 'An error occurred while classifying the image.';
            }
        });
    </script>
</body>
</html>
```