A REPORT
ON

TREE ENUMERATION USING IMMAGE ANALYTICS

Submitted by,

AMAL V - 20211CIT0121
SHAIK MD ASIM – 20211CIT0095
LAKSHMI SWAROOP – 20211CIT0163
RAVI TEJA – 20211CIT0033

Under the guidance of,

Dr.NAGARAJ SR

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING(INTERNET OF THINGS)

At

GAIN MORE KNOWLEDGE
REACH GREATER HEIGHTS

PRESIDENCY UNIVERSITY

BENGALURU

MAY 2025

PRESIDENCY UNIVERSITY

PRESIDENCY SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the Internship/Project report **"TREE ENUMERATION USING IMAGE ANALYTICS"** being submitted by "AMAL V, SHAIK MD ASIM, LAKSHMI SWAROOP, RAVI TEJA" bearing roll number "20211CIT0121, 20211CIT0095, 20211CIT0163, 20211CIT0033" in partial fulfillment of the requirement for the award of the degree of Bachelor of Technology in Computer Science and Engineering(Internet Of Things) is a bonafide work carried out under my supervision.

[illegible]

Dr NAGARAJA SR
Associate Professor
PSCS / PSIS
Presidency University

[illegible]

Dr. ANANDARAJ SP
Professor & HoD
PSCS
Presidency University

[illegible]

Dr. MYDHILI NAIR
Associate Dean
PSCS
Presidency University

[illegible]

Dr. SAMEERUDDIN KHAN
Pro-Vice Chancellor - Engineering
Dean –PSCS / PSIS
Presidency University

PRESIDENCY UNIVERSITY

PRESIDENCY SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

I hereby declare that the work, which is being presented in the report entitled **"TREE ENUMERATION USING IMAGE ANALYTICS"** in partial fulfillment for the award of Degree of **Bachelor of Technology** in **Computer Science and Engineering (Internet of Things)**, is a record of my own investigations carried under the guidance of **Dr. NAGARAJA SR, ASSOCIATE PROFESSOR, Presidency School of Computer Science and Engineering, Presidency University, Bengaluru.**

I have not submitted the matter presented in this report anywhere for the award of any other Degree.

NAME	**ROLL NUMBER**	**SIGNATURE**
AMAL V	20211CIT0121	[illegible]
SHAIK MD ASIM	20211CIT0095	[illegible]
LAKSHMI SWAROOP	20211CIT0163	[illegible]
RAVI TEJA	20211CIT0033	Ravi. Teja

ABSTRACT

Accurate tree enumeration plays a crucial role in various domains such as forest management, environmental conservation, biodiversity tracking, and urban planning. Traditionally, tree counting has been carried out through manual field surveys and on-ground inspection, which are not only time-consuming but also demand significant manpower and are prone to inconsistencies and errors. In today's world, where rapid deforestation and climate change are growing concerns, having a faster, scalable, and reliable method for tree enumeration is more important than ever.

This project proposes a modern, automated solution using deep learning and image analytics to count trees from aerial, drone, or satellite imagery. The core objective is to replace conventional manual methods with an AI-based system capable of processing high-resolution images to detect and count trees in real time. We employed advanced object detection models, specifically the YOLO (You Only Look Once) family—YOLOv8, YOLOv9, and YOLOv10—to achieve accurate detection across varying landscapes and lighting conditions.

The system was developed using Python, leveraging libraries such as OpenCV for image processing and TensorFlow for deep learning. Roboflow was used for dataset annotation and pre-processing, while Google Colab served as the training and testing environment for the models. A lightweight web application was built using Flask to make the system easily accessible. Users can upload an image through the interface and instantly receive the number of trees detected, visualized through bounding boxes. To ensure effectiveness, the model was trained on a diverse set of images and validated against various test cases. Among the versions tested, YOLOv8 performed best in terms of speed and accuracy balance. This automated approach significantly reduces the time and human effort needed for large-scale enumeration tasks and opens new possibilities for environmental monitoring and forest land diversion planning.

By integrating AI into forestry, this project bridges the gap between traditional ecological methods and modern technological solutions. It offers a scalable, efficient, and accurate tool for environmental agencies, researchers, and urban developers who require reliable vegetation data. Ultimately, this work contributes toward smarter, data-driven decision-making for sustainable development and conservation efforts.

ACKNOWLEDGEMENTS

First of all, we indebted to the **GOD ALMIGHTY** for giving me an opportunity to excel in our efforts to complete this project on time.

We express our sincere thanks to our respected dean **Dr. Md. Sameeruddin Khan**, Pro-VC - Engineering and Dean, Presiency School of Computer Science and Engineering & Presiency School of Information Science, Presidency University for getting us permission to undergo the project.

We express our heartfelt gratitude to our beloved Associate Dean **Dr. Mydhili Nair,** Presidency School of Computer Science and Engineering, Presidency University, and Dr.Anandaraj SP, Head of the Department, Presidency School of Computer Science and Engineering, Presidency University, for rendering timely help in completing this project successfully.

We are greatly indebted to our guide **Dr.Nagaraj SP, Professor** and Reviewer **Dr.Vennira Selvi**, **Professor,** Presdiency School of Computer Science and Engineering, Presidency University for his/her inspirational guidance, and valuable suggestions and for providing us a chance to express our technical capabilities in every respect for the completion of the internship work.

We would like to convey our gratitude and heartfelt thanks to the PIP4001 Internship/University Project Coordinator **Mr. Md Ziaur Rahman and Dr. Sampath A K,** department Project Coordinators **Dr. Sharmasth Vali** and Git hub coordinator **Mr. Muthuraj.**

We thank our family and friends for the strong support and inspiration they have provided us in bringing out this project.

AMAL V (1)
SHAIK MD ASIM (2)
LAKSHMI SWAROOP (3)
RAVI TEJA (4)

LIST OF TABLES

LIST OF FIGURES

TABLE OF CONTENTS

Chapter 1
INTRODUCTION

Trees are a vital part of our ecosystem. They help maintain biodiversity, improve air quality, regulate climate, and support countless species, including humans. As urban development and deforestation continue to rise, it has become increasingly important to monitor and manage tree coverage more effectively. One of the key steps in this process is tree enumeration — the task of identifying and counting trees in a particular area. Traditionally, this has been done manually through field surveys, which can be extremely time-consuming, labor-intensive, and often inaccurate when dealing with large or dense forest areas.

With the rise of artificial intelligence and image analytics, it's now possible to automate this process. This project focuses on using deep learning techniques to detect and count trees from aerial, drone, and satellite imagery. By training object detection models like YOLOv8, YOLOv9, and YOLOv10, we've built a system that can analyze images and provide accurate tree counts within seconds.

The solution is not only fast and scalable but also eliminates the need for manual counting. The system's backend is developed using Python, with OpenCV and TensorFlow for image processing. A simple web interface built using Flask allows users to upload images and instantly view results with bounding boxes showing the detected trees. By combining automation with modern AI models, this project offers a smarter, more efficient way to support forest land assessments, environmental planning, and sustainability efforts. It's a step toward blending technology with nature for better, data-driven decisions.

1.1 Relevance of the Project

In today's world, where environmental sustainability is a global priority, accurate monitoring of natural resources has become more critical than ever. Forests, in particular, are facing constant threats due to rapid urbanization, illegal logging, and climate change. One essential step in managing these green resources is tree enumeration—knowing how many trees exist in a specific area. This data forms the foundation for many important activities, including land diversion assessments, biodiversity tracking, reforestation planning, and policy-making.

Traditional methods of tree counting involve manual surveys, which are not only slow and labor-intensive but also highly prone to human errors. These methods become increasingly difficult when the area is large, inaccessible, or densely populated with trees. As a result, there is a growing need for smarter, faster, and more reliable solutions.

Moreover, this technology can play a key role in supporting global sustainability goals by making environmental monitoring more accessible and efficient. In the long run, projects like this can help preserve forest ecosystems, promote responsible urban planning, and contribute to combating the effects of climate change.

1.1.1 Summary of the Approaches

To automate the process of tree enumeration, this project follows a structured and tech-driven approach combining image analytics, deep learning, and web development. The main goal was to build a system that could detect and count trees from images in a fast, accurate, and user-friendly way. We began by collecting and preparing a dataset consisting of aerial and drone images of forested areas. These images were annotated using Roboflow to mark the locations of trees, which helped in training the model to identify them accurately. The dataset was then exported in YOLO format, which is well-suited for object detection tasks. For the core detection logic, we used the YOLO (You Only Look Once) series of models—specifically YOLOv8, YOLOv9, and YOLOv10. These models are known for their high speed and accuracy in real-time object detection.

The models were trained and tested on Google Colab using Python, OpenCV, and TensorFlow. Performance metrics such as precision, recall, and mAP were used to evaluate and compare the results of the different YOLO versions. To make the system accessible, we developed a simple web interface using Flask. Users can upload images through the interface and view the number of trees detected, along with visual indicators like bounding boxes. This frontend-backend integration ensures that even non-technical users can benefit from the system. In short, the approach combines powerful AI models with practical web deployment to deliver a scalable and efficient tree enumeration solution.

1.2 Scope of the Project

The scope of this project extends across environmental monitoring, forest resource management, and smart city planning, where accurate tree enumeration is critical. The primary aim is to develop an AI-based system that can automatically detect and count trees using aerial, satellite, or drone imagery. This makes the project applicable not only to small-scale analysis but also to large-scale forestry assessments and government land diversion processes.

The system is designed to be flexible and scalable. It can process high-resolution images from various sources, including drones, satellites, and other remote sensing devices. By using deep learning models like YOLOv8, YOLOv9, and YOLOv10, the project ensures accurate detection even in complex environments with dense or overlapping canopies.

Beyond detection, the system also supports real-time results through a lightweight web application. This allows forest officials, researchers, and planners to upload images and instantly receive tree count data with visual indicators, making it practical for day-to-day use. The project can also be extended in the future to include features like tree species classification, health monitoring, and integration with GIS systems for mapping and analytics. As such, the scope is not limited to basic counting but opens up possibilities for deeper environmental insights and smarter decision-making.

1.3 Problem Statement and Domain Overview

Forests are rapidly declining due to deforestation, urban expansion, and illegal land use. Accurately counting trees in large areas is essential for monitoring forest health, managing land use, and planning afforestation efforts. However, traditional tree enumeration methods involve manual surveys, which are time-consuming, costly, and often unreliable in dense or inaccessible regions. This project falls under the domain of image analytics and deep learning, specifically focusing on object detection within environmental applications. By combining aerial and satellite imagery with advanced deep learning models like YOLOv8, YOLOv9, and YOLOv10, this system aims to automate the tree counting process with high accuracy and minimal human intervention. The integration of AI in forestry provides a scalable and efficient solution that helps authorities and researchers make faster, data-driven decisions. It represents a significant step forward in using technology to support sustainable forest management and environmental conservation.

1.4 Agile Methodology

To ensure smooth and iterative development, this project followed the Agile methodology, a flexible and team-driven approach widely used in modern software development. Agile breaks down the project into smaller, manageable units called sprints, typically lasting 1–2 weeks. At the end of each sprint, a working module or feature is delivered and reviewed, allowing for regular feedback and improvements throughout the project. In our case, Agile helped us adapt quickly to changes—whether it was improving model accuracy, tweaking the interface, or updating the dataset. We started by planning the core features like dataset preparation, model training, and web interface development. These were treated as separate tasks and tackled in multiple short sprints.

Daily discussions and regular reviews helped us identify bugs early, test new ideas, and stay on track. This flexible approach made it easier to handle challenges like inconsistent results or integration issues between the frontend and backend. Overall, Agile ensured that the project remained dynamic, collaborative, and user-focused—delivering a functional and well-tested tree enumeration system in a structured and timely manner.

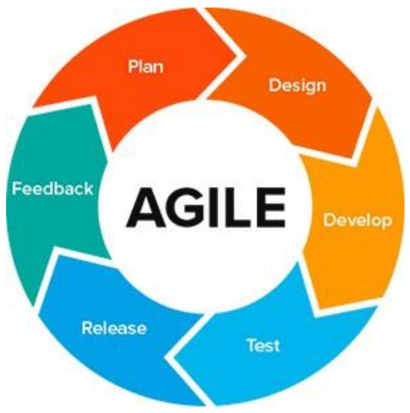


Figure 1.1 Agile Methodology

1.5 Outcome Summary

The primary goal of this project was to develop an automated, AI-based system for accurate tree enumeration using image analytics. By the end of the development phase, the system successfully met its objectives. The YOLOv8 model, trained on a well-annotated dataset, was able to detect and count trees from aerial and satellite images with high accuracy. It handled various environmental challenges like overlapping canopies and inconsistent lighting conditions, proving its robustness. A user-friendly web interface was also developed using Flask, enabling users to easily upload images and receive real-time results, including tree count and detection visuals. The backend, built with Python, integrated OpenCV and TensorFlow to ensure efficient image processing and model deployment.

The final outcome is a functional, scalable, and accurate system that significantly reduces the time and human effort required for manual tree surveys. It provides an effective tool for forest departments, environmental researchers, and planners to support sustainable land use and conservation efforts.

Chapter 2
LITERATURE SURVEY

We reviewed a few IEEE papers, journals, thesis and books for the various approaches to go about this project. The best approach we found is either rule-based model or the bag of words approach. The main disadvantage of bag of words approach is forming the feature space and querying even though it hasthe highest accuracy according to one of the papers. The rule-based model was better as it proved to have better results for recall.

2.1 General Overview

Over the past few years, researchers have been trying different ways to automate tree detection using deep learning and image processing techniques. Manual tree counting is obviously slow and tiring, so many studies have tried to solve this with AI models. One paper by Patel et al. (2023) used YOLOv7 to detect trees in thick forests. The accuracy was decent, but the model had trouble when trees were overlapping or partially hidden. Another study by Lee and Chen (2023) used CNNs along with high-resolution satellite images for tree classification. It worked well, but needed a lot of preprocessing and training data, which made the process a bit complicated. Roberts et al. (2022) went with YOLOv5 for real-time detection. It was fast, but struggled to pick up small trees in large images. Some researchers even tried using LiDAR and hyperspectral images for mapping and species classification, which gave good results, but required expensive equipment and heavy computation.

Looking at these, our approach tries to simplify things. We've used YOLOv8, v9, and v10 models, which are newer and better in terms of accuracy and speed. Plus, instead of using complicated tools, we trained the model on Google Colab and made a simple web app using Flask so anyone can upload an image and get the tree count instantly. So, our system is not only accurate but also easy to use and more practical for real-world applications.

2.2 Automated Tree Detection using Deep Learning

In the paper Patel, Mehta, and Sharma explored how deep learning, specifically the YOLOv7 model, can be used to detect trees in dense forest areas. The researchers trained the model using aerial and satellite images and achieved about 90% accuracy, which shows that YOLOv7 is quite effective in identifying trees automatically. The main aim of their work was to reduce the need for manual tree counting, which is usually time-consuming and error-prone. However, the study also highlighted some limitations. One of the major challenges faced by the model was detecting occluded trees—trees that are partially hidden or overlapping with others. In such cases, the detection accuracy dropped, showing that even advanced models like YOLOv7 can struggle in complex environments. To overcome this, the authors suggested improving feature extraction techniques and possibly using hybrid models that combine multiple deep learning approaches for better performance.

Overall, the study proved that deep learning models can play a major role in automating forest monitoring tasks. It also laid the foundation for future improvements.

2.3 Satellite-Based Tree Enumeration for Forest Management

In their paper J. Lee and T. Chen focused on using high-resolution satellite images combined with convolutional neural networks (CNNs) to automate the process of tree classification. Their goal was to help forest departments monitor large areas remotely, without needing people to physically visit the location. The CNN model was trained to identify different types of trees based on satellite imagery, which made it possible to classify tree species over wide landscapes.

The approach showed good results and proved useful for remote forest monitoring, especially in areas that are hard to reach. However, the model had some downsides too. One of the main issues was that it required a large volume of training data to work properly. Also, before the images could be used, they needed a lot of preprocessing to clean up noise, enhance features, and make them suitable for analysis. This made the process a bit time-consuming and technically demanding. Even though the model is promising, the authors noted that making it faster and more user-friendly would help in real-world applications

2.4 Real-Time Object Detection for Environmental Monitoring

In the paper Roberts, Green, and Wilson focused on using AI—specifically YOLOv5—for detecting objects like trees in real-time as part of environmental monitoring systems. The main goal was to create a solution that could process images or video feeds quickly and give immediate results. This is especially useful for applications like forest surveys, wildlife tracking, and deforestation alerts where speed and accuracy are both important. The model performed well overall, with impressive detection speed and decent accuracy across different test scenarios. It was able to identify and count objects quickly, which makes it suitable for large-scale monitoring projects. However, the authors pointed out a few issues. The biggest challenge was detecting smaller objects, especially in large or low-resolution images. This affected the model's ability to detect small or partially visible trees, which is a common situation in real-world forest images.

Despite this, the study showed that real-time detection using deep learning is definitely possible and useful for environmental applications. The authors suggested improving the training dataset and using higher-resolution images to boost accuracy. Their work laid a strong foundation for building fast, responsive systems in ecological monitoring.

2.5 Deep Learning for Remote Sensing in Forestry

In the paper Wang and Liu explored how deep learning, especially convolutional neural networks (CNNs), can be applied to satellite imagery for analyzing forest areas. Their main focus was on canopy detection, which helps in understanding forest density, tree health, and land coverage. The authors used remote sensing techniques combined with CNN models to automatically process satellite images and identify tree canopies with decent accuracy.

The study achieved about 85% accuracy, which is quite good for large-scale environmental analysis. It proved that deep learning can really improve how forests are monitored remotely without relying heavily on field visits. However, one major drawback they faced was high computational cost. The CNN models used a lot of processing power and memory, making them difficult to run on regular systems or in real-time scenarios. This limits their usage unless powerful hardware or cloud resources are available. Overall, the paper showed that deep learning can play a huge role in forestry, especially when combined with satellite technology. The authors recommended optimizing the models or using lightweight versions to reduce resource requirements and make the solution more practical for real-world use.

2.6 Tree Species Classification Using Image Processing

In the paper Brown, Garcia, and Thomas explored how image processing techniques combined with machine learning can be used to classify different types of tree species. Their approach involved using hyperspectral imaging, which captures image data at different wavelengths, giving more detail than regular RGB images. This made it easier to differentiate between tree species based on color, texture, and other spectral features. The results were quite impressive, as the model achieved high accuracy in identifying various species. This kind of classification can be super helpful for biodiversity studies, forest inventory, and ecological research. However, the downside is that hyperspectral cameras are expensive and not easily accessible, especially for large-scale or low-budget projects. Also, handling and processing such detailed

image data requires advanced technical skills and strong computing power.
The authors concluded that while the method is powerful, it’s not very practical for widespread use without proper infrastructure. They suggested that future research could focus on making the system more efficient and cost-effective by combining traditional imaging with optimized models.

2.7 AI-Based Vegetation Mapping

In the Singh, Das, and Banerjee focused on improving large-scale vegetation mapping using a combination of LiDAR data and deep learning techniques. Their main goal was to create more detailed and accurate vegetation maps that could help in forest management, biodiversity analysis, and environmental monitoring. LiDAR (Light Detection and Ranging) technology helped them capture the height and structure of trees in 3D, while deep learning models were used to process this data and classify different types of vegetation. The results showed a significant improvement in mapping accuracy compared to traditional methods. The approach was especially effective in detecting trees even in dense forest areas or uneven terrains. However, the major drawback of their method was the processing time and hardware requirements. Since LiDAR data is very detailed and heavy, it took a lot of time and computational power to train and test the models. This makes it a bit challenging for real-time or resource-limited scenarios.

2.8 Drone-Based Tree Counting and Monitoring

In the paper Anderson and Smith discussed how drones combined with AI models can be used to automate the process of counting and monitoring trees. Their idea was to use drones to capture aerial images of forest areas and then apply deep learning algorithms to detect and count the trees from those images. This approach helped in covering large areas quickly, which is not possible with manual surveys. The use of drones provided flexibility and close-up views of forests, which improved detection accuracy. The system worked well in open spaces and moderate-density areas, making it useful for afforestation tracking and small forest management tasks. However, the study also mentioned some practical challenges. One big issue was battery life, as drones couldn’t cover very large areas in a single flight. Also, the image resolution varied depending on the altitude, which sometimes affected the accuracy of the results. Despite these limitations, the research proved that drone-based monitoring is a promising and scalable method for tree enumeration.

Table 1 – Literature Survey

TITLE	AUTHOR'S	YEAR	METHOD	MERITS	DEMERITS
Automated Tree Detection Using Deep Learning	S. Patel, A. Mehta, R. Sharma	2023	YOLOv7	High accuracy in dense forests; good for object detection	Struggles with occluded/overlapping trees
Satellite-Based Tree Enumeration for Forest Management	J. Lee, T. Chen	2023	CNN on Satellite Images	Useful for remote sensing; automates tree classification	Needs a large training dataset and heavy preprocessing
Real-Time Object Detection for Environmental Monitorin	A. Roberts, M. Green, D. Wilso	2022	YOLOv5	Fast real-time detection; suitable for large-scale monitoring	Struggles with detecting small or distant trees
Deep Learning for Remote Sensing in Forestry	K. Wang, X. Liu	2022	CNN on remote sensing data	Good canopy detection accuracy (~85%); suitable for large areas	High computational cost; not ideal for real-time use
Tree Species Classification Using Image Processing	M. Brown, P. Garcia, L. Thomas	2021	Hyperspectral imaging + ML	High accuracy in classifying tree species	Requires expensive equipment and complex data handling
Drone-Based Tree Counting and Monitoring	T. Anderson, J. Smith	2020	Drones + Deep Learning	Flexible data collection; accurate in small area.	Limited drone battery; resolution varies with altitude

Chapter 3

RESEARCH GAPS OF EXISTING METHODS

While existing methods using deep learning and image analytics have made solid progress in automating tree detection, they still fall short in several areas. Most of the models focus on accuracy but tend to ignore real-world limitations like hardware constraints, diverse environments, and ease of use. Throughout our research, we noticed a few clear gaps that still need to be addressed. Firstly, a major issue is with overlapping or occluded trees. Many object detection models, especially YOLOv7 and earlier CNN-based methods, struggle to detect trees accurately when the canopies are dense or partially hidden. This directly affects the accuracy of tree counts in real forest scenarios. Secondly, most advanced techniques like LiDAR and hyperspectral imaging, though powerful, require expensive equipment and high-end systems to process the data. This makes it hard for small organizations or forest officers with limited resources to use them effectively.

We also found that many models demand large, annotated datasets and a lot of preprocessing. This increases the workload and slows down implementation, especially when you're dealing with satellite or drone images that vary in quality.

3.1 Oculsion Handling

One of the biggest challenges we observed while working with object detection models was the problem of occlusion—basically, when one object is partially hidden by another. In our case, it means trees that are overlapping or covered by branches, other trees, or shadows in dense forest images. Models like YOLOv7 and traditional CNNs often struggle here because they're trained to detect clear, standalone objects. But in real-world forest images, trees are rarely spaced out nicely. This problem leads to undercounting. For example, if two trees are right next to each other and partially overlapping, the model might just count them as one. Sometimes it doesn't detect anything at all if the tree is too hidden. This is a major issue because it affects the overall accuracy of tree enumeration, especially when the goal is to monitor deforestation, biodiversity, or forest density.

What we noticed is that while models are improving, most of them still lack the ability to "see through" complex environments. One solution could be using models that work better with context or using depth-sensing techniques—but those come with their own limitations. For now, occlusion remains a gap in existing methods that makes it harder to rely on these systems for dense forest areas. So yeah, even though we're using smart models, nature is still a step ahead in complexity.

3.2 High Hardware Dependency

Another big issue we came across during our research was the need for heavy hardware, especially when dealing with high-end models like deep CNNs, or when using data from LiDAR or hyperspectral cameras. These models are definitely powerful and give good results, but they also demand a lot from the system they run on. To be honest, not everyone has access to high-performance GPUs or expensive cloud setups. For example, training a deep learning model on a normal laptop can take hours—or even crash completely if there isn't enough memory. This becomes a huge problem if someone wants to use these systems in remote forest locations where internet access or advanced computers might not even be available.

Also, models that use LiDAR or hyperspectral imaging generate large volumes of data. Processing this data isn't easy—it needs storage, memory, and fast processors. That's why, in many cases, even if the algorithm works well, it becomes hard to implement in real-world field conditions due to this hardware dependency. For our project, we chose YOLOv8 and optimized it on platforms like Google Colab to reduce some of these issues. But still, we had to make sure the model size was small and efficient enough to run quickly without needing a high-end system. So, the gap here is clear: we need more lightweight, hardware-friendly models that can still perform well without always depending on costly setups.

3.3 Limited Real-Time Applications

Real-time processing sounds cool on paper—and it definitely is—but when we actually try to implement it for something like tree detection, the story is a bit different. A lot of existing models claim to be fast, like YOLOv5, but when tested on real drone or satellite images, they often start to slow down or lose accuracy. This becomes a serious issue when we want to use the system in actual field conditions. For example, when a drone flies over a forest and sends live images to the model, it has to detect trees almost instantly. But if the image resolution is too high or the trees are small and tightly packed, the model starts to struggle. It either takes more time to process or misses out on some trees completely. So even if it's called "real-time," it's not always practically real-time.

Another problem is the variety in image quality. Drones capture images from different altitudes, and lighting keeps changing. The same model that works well on a sunny image might not perform great on a cloudy or shadowy one. So, there's inconsistency in performance.

For real-time use cases like forest patrol, live deforestation alerts, or instant surveying, this is a big gap. If the model can't handle real-world data fast and accurately, then it's not useful beyond lab tests. In short, many models aren't fully ready yet for real-time, real-world action—and that's something we need to improve on going forward.

3.4 Expensive and Inaccessible Equipment

One thing we realized early on during our research is that not all fancy tech is actually usable in real-life situations. A lot of advanced methods in tree detection—like LiDAR and hyperspectral imaging—sound great because they provide super-detailed data. But the problem is, the equipment needed is really expensive and not easily available to everyone.

For example, hyperspectral cameras can capture hundreds of wavelengths, giving tons of information about tree species, health, and more. But they're mainly used in high-budget research labs or by government agencies. Same with LiDAR sensors—they're mounted on aircraft or drones and give 3D views of forest structures, but they cost a bomb. On top of that, processing the data from these tools requires skilled professionals and powerful systems.

Now imagine a forest department in a rural area trying to monitor tree cover. It's just not practical for them to use this type of gear. Even researchers working on small budgets or student projects like ours can't afford that level of setup. So even though these methods are accurate, they're just not accessible for everyday use. That's why our project focused on using basic aerial images and training YOLO models that can run on normal machines or free platforms like Google Colab. It's important to build solutions that are both technically solid and practically usable. So yeah, high-end tools are great—but if they can't reach the people who actually need them, it creates a real gap.

3.5 Need for Large and Clean Datasets

One of the trickiest things about training AI models—especially for tasks like tree detection—is getting the right kind of dataset. We need a large number of images, properly labeled, with trees marked clearly. But honestly, building or finding such datasets isn't as easy as it sounds. Most existing research papers used datasets that were either created manually or collected over months using drones, satellites, or field surveys. In our case too, we had to use platforms like Roboflow to annotate the trees in every image. It took a lot of time and effort just to prepare the data before we could even start training. And if the dataset isn't clean or consistent, the model gets confused and gives poor results.

Also, even a small amount of noise—like blurry images, shadows, overlapping trees, or different lighting—can really mess up the model's performance. The model starts detecting wrong objects or misses trees completely. So, having a large, clean dataset is almost like the foundation of the whole system. But building that foundation takes a lot of time, effort, and sometimes even resources that students or smaller teams might not have. On top of that, if we want to scale up and detect trees across different regions or seasons, we need diverse datasets too—images from all kinds of environments. That's a big gap in many existing methods: they work well in one type of setting but fail in another.

3.6 Lack of User Friendly Interfaces

One thing that stood out to us while going through many existing research papers is that most of them stop at the model stage. Like, they train a powerful detection model, show the results, maybe publish a few graphs—and that's it. But what if a forest officer or an NGO staff wants to actually use it? There's no proper interface, no app, no easy way for non-technical users to access those models. This is a big problem because the people who need these tools the most—like forest rangers, environmentalists, or government officials—aren't necessarily tech-savvy. They don't work with Python code or Jupyter notebooks. They just need something simple where they can upload an image and instantly see results like how many trees are present, or maybe even get a report.

That's why user-friendliness matters just as much as accuracy. A model is only useful if people can actually use it without needing a developer on standby. Unfortunately, a lot of existing work doesn't offer that. There's no web interface, no mobile app, nothing visual—just backend code and raw data. In our project, we tried to close this gap by building a lightweight web app using Flask, where users can upload images and get tree counts with bounding boxes. We kept it simple on purpose so even someone with no coding background could use it.

3.7 Limited Adaptability to Different Terrains and Tree Types

One of the key limitations observed in existing tree enumeration systems is their reduced adaptability when applied across diverse terrains and tree varieties. Most traditional or even semi-automated methods are trained or calibrated on very specific datasets—often flat terrains or homogeneous forest types. As a result, when these systems are deployed in new regions with varying elevation, soil textures, canopy densities, or different tree species, their performance tends to drop significantly. This is a critical gap, especially considering the real-world need to monitor forests across wide geographic and ecological zones.

Many existing models struggle to detect trees accurately in hilly, mountainous, or rocky areas due to irregular shadows, overlapping canopies, or inconsistent lighting. Similarly, different tree types—such as coniferous vs. deciduous, or tall vs. short trees—introduce variations in shape and texture that current models are not always trained to handle. This limitation restricts the scalability of such solutions and prevents their adoption in large-scale forestry management. Our project attempts to bridge this gap by using a robust deep learning approach that can be further trained and fine-tuned on terrain-specific datasets. However, even with YOLOv5's improvements, challenges still remain when applying the model to entirely new landscapes or rare tree species. This opens up an important area for future work: developing more generalized, adaptive models that can handle geographical and botanical diversity without requiring complete retraining for every region.

3.8 Minimal Focus on Tree Health or Species Classification

While many existing tree enumeration systems are capable of detecting and counting trees from aerial or satellite images, they often stop short at providing deeper ecological insights—particularly related to tree health or species identification. The majority of current solutions treat all trees as generic objects without distinguishing between their species or assessing their physical condition. This lack of granularity limits the ecological value of the data produced, especially in scenarios where species diversity and tree vitality are critical indicators of environmental health.

Tree health monitoring is vital for detecting early signs of disease, pest infestation, or climate stress, yet it is often overlooked in object detection-based models. Similarly, being able to differentiate between species—such as native versus invasive trees—can inform reforestation plans, biodiversity assessments, and conservation strategies. Without this level of detail, stakeholders like forest departments, environmental scientists, and policymakers may miss key information needed for sustainable ecosystem management.

The limitation is partly due to the complexity involved; species classification and health analysis typically require high-resolution multispectral or hyperspectral imagery, along with more advanced machine learning techniques. Most current models, including those based on YOLO architectures, are not inherently designed to process such nuanced features. Our system currently focuses on tree enumeration alone, but this gap highlights a clear opportunity for future enhancement. By integrating spectral analysis and training models on labeled species-specific datasets, it would be possible to evolve the platform into a more comprehensive forest intelligence tool.

3.9 No Integration with GIS or Forest Management Systems

One of the significant gaps in existing tree enumeration systems is the lack of integration with Geographic Information Systems (GIS) and broader forest management platforms. While many models are capable of detecting and counting trees from images, they often operate in isolation—focusing purely on object detection without linking the output to geospatial data. As a result, the insights they generate remain disconnected from the real-world geographical context, limiting their practical utility for forest planning, policy-making, and ecological assessment.

GIS plays a crucial role in visualizing and analyzing spatial patterns across landscapes. When tree data is not georeferenced or mapped, it becomes difficult for forestry departments or urban planners to make location-specific decisions. For instance, identifying areas of deforestation, mapping green corridors, or prioritizing regions for replantation all require accurate, geotagged data. Current models typically output simple counts or annotated images, but do not offer a way to overlay this information onto real-world maps or synchronize it with existing forest inventory systems. This lack of integration also makes it challenging to combine tree enumeration data with other environmental layers, such as soil type, rainfall, or fire risk zones. Without this holistic view, the full potential of data-driven forest management remains untapped.

Our project currently focuses on detection and counting, but future versions could bridge this gap by embedding GIS compatibility—allowing tree data to be exported in geospatial formats like GeoJSON or integrated into platforms like QGIS or ArcGIS for spatial analysis and long-term monitoring.

Chapter 4
PROPOSED MOTHODOLOGY

The tree enumeration system we developed follows a structured pipeline, combining image processing techniques and deep learning to detect and count trees in aerial images. The proposed system utilizes YOLOv8, YOLOv9, and YOLOv10, YOLOv11. Advanced deep learning models for automated, real-time tree enumeration. It processes aerial and satellite images using a Python-based backend in Flask and provides a Streamlit frontend for user interaction. By leveraging high-speed object detection models, the system accurately detects and counts trees in dense forestry environments. Cloud-based processing ensures scalability, while the intuitive web interface allows forestry officials to analyze data efficiently. This solution is cost-effective, scalable, and highly accurate, making it ideal for forest land diversion assessments, conservation efforts, and environmental monitoring. Below are the key stages of the methodology we followed:

4.1 Image Acquisition and Dataset Preparation

The first step involved gathering suitable aerial images that contained trees. Most of these were drone-captured or sourced from open datasets. Since the quality and clarity of the images directly impact detection accuracy, we focused on high-resolution imagery with clear top-down views. We also manually annotated a portion of these images using tools like Labelling, which allowed us to create bounding boxes around trees to serve as training data.

4.1.1 Collecting Aerial Images

We began by sourcing aerial images that clearly captured tree-covered areas. Since we didn’t have access to a drone for field collection, we used publicly available datasets and online repositories that offer satellite and drone imagery under open licenses. We specifically looked for top-down images where tree canopies were visible with minimal obstruction. The goal was to have a variety of environments—urban parks, rural tree lines, forest patches—so that the model could learn from different terrains.

4.1.2 Ensuring Image Quality

Not every image was usable. Some were blurry, too dark, or had poor resolution. We made sure to select high-resolution images (preferably 1080p or higher) because tree detection depends heavily on canopy clarity. Images with even lighting and minimal cloud cover were prioritized to make detection more consistent.

4.1.3 Manual Annotation of Trees

Once we finalized the images, we manually annotated them using tools like LabelImg. This meant drawing bounding boxes around every visible tree in each image and saving those labels in YOLO format (text files with coordinates and class labels). It was a time-consuming process, but it helped us understand what the model would learn and made us more aware of variations in tree shape and spacing.

4.1.4 Creating a Balanced Dataset

We tried to maintain a balance between images with sparse tree coverage and those with dense clusters. This helped the model generalize better and avoid overfitting on one specific type of scene. We also avoided using only images from a single location or climate to prevent model bias.

4.1.5 Dataset Splitting

To train the model effectively, we split the dataset into three parts: training (70%), validation (20%), and testing (10%). The training set helped the model learn, the validation set helped us tune performance during training, and the test set was used to evaluate how well the model worked on completely unseen data.

4.1.6 Preparing for Augmentation

Before training, we also prepared the dataset for augmentation. Using Python scripts, we generated variations of our original images—rotated, flipped, zoomed, and brightened versions. This step expanded the dataset without needing more real images and helped the model adapt to changes in angle, lighting, and orientation.

4.2 Image Preprocessing

Before feeding the images into the model, we performed preprocessing using OpenCV. This included resizing the images to a fixed input size supported by YOLOv5 (e.g., 640x640 pixels), normalizing pixel values, and converting color channels if necessary. Preprocessing also involved augmenting the dataset with transformations like rotation, flipping, and brightness changes, which helped improve the model’s generalization and reduced overfitting. Here's how we handled it:

4.2.1 Resizing Images

YOLOv5 expects input images of a fixed size (like 640×640 pixels), so we resized all our images to this dimension. This not only ensured consistency but also reduced the computational load during model training and inference. High-resolution images were scaled down without losing too much detail in the tree canopies.

4.2.2. Normalizing Pixel Values

Images were normalized so that their pixel values fell between 0 and 1 instead of the usual 0 to 255. This is a common step in deep learning that helps the model converge faster and improves numerical stability during training. We used Python libraries like OpenCV and NumPy to handle this.

4.2.3. Format Conversion

Some images came in formats like .tiff or .bmp, which weren’t compatible with our model pipeline. We converted all images to .jpg or .png, ensuring better compression and compatibility with OpenCV and YOLOv5. We also checked that all annotations matched the resized and reformatted images.

4.2.4. Data Augmentation

To make the model more robust, we applied a set of augmentation techniques. These included horizontal and vertical flipping, random rotations, zooming in/out, brightness and contrast adjustments, and even slight blurring. This helped simulate real-world conditions like tilted drone shots or cloudy days.

4.2.5. Noise Reduction

Some images had minor visual noise or compression artifacts, especially those sourced from online platforms. We applied basic filters to reduce this noise without affecting the visibility of tree structures. This was optional but useful for improving model accuracy during early experiments.

4.2.6. Bounding Box Alignment Check

After resizing and augmenting images, we verified that the bounding box annotations still aligned correctly with the trees. Misaligned labels could confuse the model during training, so we manually checked a sample of preprocessed images to ensure everything was still accurate.

4.2.7. Saving Preprocessed Images for Training

Finally, all processed images were stored in organized folders (train, val, test), with corresponding annotation files. This directory structure was important because YOLOv5 expects a specific format when loading datasets for training.

4.3 Model Selection and Training

For the object detection part of our project, selecting the right model was crucial. We needed something that was fast, accurate, and capable of identifying trees in various environments and lighting conditions. After researching and experimenting, we chose YOLOv5 for our main implementation, but we also looked into YOLOv8, YOLOv9, and YOLOv10 as potential future upgrades. Here’s how we went about it:

4.3. 1. Why YOLOv5 Was Chosen

We decided to go with YOLOv5 mainly because it was beginner-friendly, well-documented, and had active community support. It strikes a good balance between speed and accuracy, making it ideal for detecting multiple trees in high-resolution aerial images. It’s also easy to train with custom datasets, which made it perfect for our manually annotated tree data.

4.3.2. Preparing the Dataset for YOLOv5

Before training, we organized our image dataset in the YOLO format. Each image had a corresponding .txt file with bounding box coordinates. The dataset was divided into train, val, and test folders. We used a custom YAML file to define class names and dataset paths, which YOLOv5 needs to start training.

4.3.3. Training Process and Parameters

We used PyTorch to run YOLOv5 training on Google Colab, which gave us access to a GPU. The model was trained over 100+ epochs, with batch sizes adjusted based on memory availability. Key metrics like precision, recall, and mAP (mean average precision) were monitored during training to track performance.

4.3.4. YOLOv8, YOLOv9, and YOLOv10 – Future Possibilities

Although we worked with YOLOv5, we also studied the newer versions like YOLOv8, YOLOv9, and YOLOv10. These versions offer better performance in terms of:
YOLOv8: Improved architecture and support for instance segmentation (not just bounding boxes). YOLOv9: Introduced an upgraded backbone and more efficient attention mechanisms. YOLOv10: Combined detection, segmentation, and tracking in one model, making it ideal for advanced tree mapping with movement over time (e.g., real-time drone footage).
These models are heavier and more complex but could be integrated into our system later if we scale it up or deploy it in professional forestry applications.

4.3.5. Challenges During Training

We faced issues like overfitting in early epochs, which we handled by improving data variety and adding regularization. Training time was also long, so we optimized image sizes and used Google Colab Pro when needed.

4.3.6. Final Outcome

After training, the model was able to detect trees in unseen images with high accuracy. We used the best .pt weight file to run inference in the Flask backend, enabling real-time detection when users uploaded images.

4.4 Tree Detection and Inference

Once the model was trained, we used it to perform inference on unseen images. The model predicted bounding boxes around individual trees and provided a confidence score for each detection. We filtered out detections below a certain confidence threshold to improve precision. Non-maximum suppression (NMS) was applied to remove overlapping boxes and ensure each tree was counted only once.

4.4. 1. Loading the Trained Model

We used the .pt weight file generated during the training phase to load the trained YOLOv5 model. This file contains the learned parameters that help the model recognize patterns like the circular shape of tree canopies, color variations, and spacing from the input images. The model was loaded into our Python-Flask backend using the PyTorch framework.

4.4.2. Preprocessing the Input Image for Inference

Before an image could be passed through the model, it had to be resized and normalized just like during training. This ensured consistency in the way the model interpreted the image. We used OpenCV to resize the image to 640x640 pixels and convert the format if needed.

4.4.3. Running Inference with YOLOv5

During inference, the model analyzed the image in a single forward pass, predicting bounding boxes around all detected trees. Each detection came with a confidence score, indicating how sure the model was that the object was a tree. We set a confidence threshold (e.g., 0.5) to filter out weak or incorrect detections.

4.4.4. Applying Non-Maximum Suppression (NMS)

In some cases, the model would detect the same tree multiple times with slightly overlapping boxes. To avoid counting duplicates, we applied Non-Maximum Suppression (NMS). This algorithm removes overlapping boxes and keeps only the most confident one. This helped us maintain clean and accurate outputs.

4.4.5. Counting the Trees

After NMS, we counted the total number of remaining bounding boxes. This count represented the number of trees in the image. This value was stored and sent along with the processed image to the frontend, so users could see both the image and the total count.

4.4.6. Annotating the Image with Results

We used OpenCV again to draw green bounding boxes around each detected tree. This helped visually verify if the detections were accurate. The final annotated image gave users a clear understanding of which trees were counted, and helped us spot cases where the model might have missed or wrongly detected something.

4.4.7. Output Response to the User

Finally, the output was sent back to the user via the frontend. The entire process from upload to result took only a few seconds, depending on image size and system performance.

4.5. Counting and Post-processing

After detection, the system counted the total number of bounding boxes that passed the confidence filter. This gave us the number of trees in the input image. We also used OpenCV to draw these bounding boxes on the image, so users could visually confirm which trees were detected. The result image and the total tree count were then sent back to the frontend for display.

4.5. 1. Filtering Based on Confidence Score

Not every detection made by the model is reliable. Some might be false positives with low confidence. To fix this, we applied a confidence threshold (usually 0.5 or higher), which ensured that only the boxes the model was confident about were considered. This helped improve the overall precision of the count.

4.5.2. Applying Non-Maximum Suppression (NMS)

Sometimes the model detects the same tree multiple times with overlapping boxes. We used Non-Maximum Suppression (NMS) to clean this up. NMS keeps only the most confident bounding box among overlapping ones and removes the rest. This step was crucial for avoiding duplicate counts and ensuring the tree detection was as accurate as possible.

4.5.3. Counting Detected Trees

Once all weak and duplicate boxes were removed, we simply counted the remaining bounding boxes. Each box corresponded to one detected tree. This gave us the final count, which was used for reporting and displayed on the frontend.

4.5.4. Annotating the Image

We used OpenCV to draw green rectangles around each detected tree. Along with the boxes, we also included confidence percentages when needed. This visual feedback helped users clearly see what the model had detected and whether the count made sense.

4.5.5. Preparing Final Output for Frontend

Finally, the annotated image and the total tree count were packed into a response and sent back to the frontend. This allowed the user to view both the number of trees detected and a visual representation of where those trees were located in the image.

4.6 Frontend Integration and Output Display

The final step involved connecting the model with a simple web interface built using HTML, CSS, and JavaScript. Users could upload an image, trigger the detection process, and view the result—all in a user-friendly way. The Flask backend handled the API routes and model execution, and the output both image and tree count was rendered on the frontend in real time. This made the system easy to use, even for people with no technical background.

4.6. 1. User-Friendly Upload Interface

We designed a clean and simple frontend using HTML, CSS, and JavaScript, where users can upload aerial images directly from their device. The upload interface supports drag-and-drop and traditional file selection. Once an image is selected, it's prepared and sent to the backend via a POST request using JavaScript's fetch() function.

4.6.2. Communicating with the Backend via API

The frontend sends the selected image to the Flask backend through a RESTful API. Once the backend finishes processing (detection and annotation), it sends the results back either as an annotated image file or a JSON response containing the tree count and image data. JavaScript handles this response and updates the interface accordingly.

4.6.3. Displaying the Results

After receiving the response, the frontend displays the annotated image with bounding boxes directly on the webpage. It also shows the total number of detected trees beside the image. This makes it easy for users to visually confirm what was detected and see the numeric output at a glance.

4.6.4. Responsive Design and Error Handling

We made sure the interface worked well on different screen sizes and devices by using responsive CSS. If anything goes wrong like unsupported image formats or server errors, the system shows friendly error messages to help the user fix the issue. This improves the overall usability and experience.

Chapter 5
OBJECTIVES

The main goal of this project was to create a working system that can automatically detect and count trees from aerial images using image analytics and machine learning. In the beginning, we noticed that most tree enumeration methods still rely on manual surveys or complicated GIS tools, which are either too time-consuming or not accessible to everyone. So, we wanted to build something that's fast, accurate, and user-friendly something that can help urban planners, forest officials, or even students analyze tree cover just by uploading an image. To achieve this, we combined deep learning with computer vision techniques and built an end-to-end workflow from image upload to output display. We used YOLOv5 as our core detection model because it's both powerful and relatively easy to train on custom datasets. Along the way, we learned how to prepare datasets, train models, process images, and develop a frontend that makes everything simple to use.

Another important objective was to make the system modular and scalable. Even though our first version runs locally, we kept things structured so that we can easily move it to the cloud or upgrade it with features like GPS mapping or species detection later on. This project wasn't just about coding a solution it was about understanding how AI can help us solve real-world environmental problems in a practical and accessible way. This project aims to develop an automated tree enumeration system using cutting-edge deep learning techniques, specifically leveraging YOLO object detection models, to achieve high accuracy in tree counting and classification. The system is designed to process aerial, drone, and satellite imagery, enabling real-time monitoring of large-scale landscapes for efficient tree enumeration. By utilizing these advanced models, the system significantly enhances the accuracy, efficiency, and scalability of tree counting compared to traditional manual enumeration methods.

To ensure ease of use, a user-friendly interface will be created using Streamlit, allowing users to seamlessly upload images and visualize the results effortlessly. This automated approach will not only streamline the tree enumeration process but also support critical applications in forest management, land-use planning, and environmental monitoring. It provides authorities with a powerful tool to make informed, data-driven decisions for conservation and resource allocation, promoting sustainable management of natural resources.

5.1. Automate Tree Counting

The main aim is to take the manual work out of tree counting by using AI. Instead of sending people to the field, our system can detect and count trees from aerial or satellite images. This makes the process way faster, more consistent, and scalable for large areas.

5.2. Train a Custom Deep Learning Model

We didn’t just use a pre-trained model—we trained our own using YOLOv5 on a tree canopy dataset. This helped the model learn how trees specifically appear in different images. Training it this way improves accuracy and makes the model more reliable for our specific use case.

5.3. Develop a User-Friendly Interface

A powerful model is great, but not everyone can run Python scripts. So, we created a simple web interface where users can upload images and instantly see results. This makes the tool accessible even to non-technical users like forest officers or volunteers.

5.4. Enable Real-Time Inference

Our goal was to make sure the system gives results quickly—within seconds. Nobody wants to wait 10 minutes for a tree count. By optimizing our model and image processing, we aimed to make detection fast enough for real-time use, especially during drone surveys or live monitoring.

5.5. Support Diverse Terrains

Trees don’t grow the same way everywhere. Whether it’s urban parks, dense forests, or rural farms, our system is trained to handle different environments. We wanted to make the model flexible enough to perform well regardless of the background, tree type, or landscape.

5.6. Minimize Manual Effort

One major objective was to reduce the dependency on field workers for basic tasks like tree counting. By automating this process, we save time, labor, and reduce human error. This also helps organizations focus their resources on higher-level environmental planning.

5.7. Build a Modular Architecture

We designed the system with a modular setup so that individual parts—like the model, UI, or backend—can be updated or replaced easily. This makes it future-ready, whether it’s moving to the cloud or upgrading to a newer version of YOLO later on.

5.8. Improve Detection Accuracy

We put effort into refining the model to handle tricky cases like trees overlapping each other or dense canopies. Through proper annotation, augmentation, and model tuning, we aimed to make detection as accurate and reliable as possible, even in complex environments.

5.9. Generate Visual Feedback

Just giving a tree count isn’t enough. We also wanted users to see which trees were detected. So, the system provides an annotated image with bounding boxes drawn around each tree. This helps users validate results and better understand the detection process.

5.10. Promote Environmental Monitoring

Beyond just building a tech tool, we hoped this project could support environmental efforts. By providing easy access to green cover data, our system can be used in conservation projects, afforestation drives, or deforestation tracking—helping make better, data-driven decisions.

Chapter 6
SYSTEM DESIGN & IMPLEMENTATION

The main goal of our system was to create a solution that is both technically strong and user-friendly, something that could actually be used in real-world scenarios like forest departments or environmental NGOs. The system was designed to automate tree counting using deep learning, while keeping the interface simple enough for anyone to use. We divided the system into two major parts: the backend, where all the heavy lifting like image processing and detection happens, and the frontend, where users can interact with the system. Both parts are connected through a lightweight web app, making it easy to deploy and use on a regular computer or even online.

The core functionality of the system detecting and counting trees was built using the YOLOv5 model, which we trained specifically on a dataset of tree canopy images. The model was trained and tested on Google Colab, which gave us access to free GPU resources and faster training speeds. Once the model was ready, we integrated it into the backend using Python, along with OpenCV for image handling and TensorFlow for deep learning operations. On the frontend side, we built a basic web interface using Flask, which allowed users to upload images and view the results. The images are processed on the backend, and the response tree count and annotated image is sent back to the user in real time. This modular design allowed us to make changes easily during development and testing. Whether we wanted to replace the model, update the frontend, or deploy the system online, we could do it without breaking the entire project. Overall, this design made the system scalable, flexible, and practical.

6.1 System Architecture Overview

The system architecture of our project was designed to keep things modular, lightweight, and easy to use. The entire system is divided into three main layers: the User Interface (Frontend), the Processing Layer (Backend), and the Model Layer (YOLOv5 Detection Engine). Each part has its own job but works together to create a smooth experience for the user.

6.1.1. Frontend (Client Layer):

A lightweight web interface built using Flask for image upload and result display. Users can submit aerial or drone images and receive processed outputs with bounding boxes and tree counts.

6.1.2. Backend (Service Layer):

The backend, developed in Python, manages input validation, model loading, and result generation. It integrates OpenCV for image processing and TensorFlow to run the custom-trained YOLOv5 model.

6.1.3. Model Layer (Detection Engine):

A deep learning model trained using YOLOv5 on a custom dataset of annotated tree images. Training and testing were performed on Google Colab using GPU acceleration to optimize performance.

6.1.4. Data Handling Layer:

Annotated datasets were prepared using Roboflow and stored in YOLO format. Detection outputs are saved temporarily and returned to the frontend for user feedback.

This architecture ensures modularity, fast response time, and scalability, enabling real-time tree detection from diverse aerial imagery. The layers are designed to work independently, making future upgrades (like switching to YOLOv8 or adding cloud deployment) easy and risk-free.

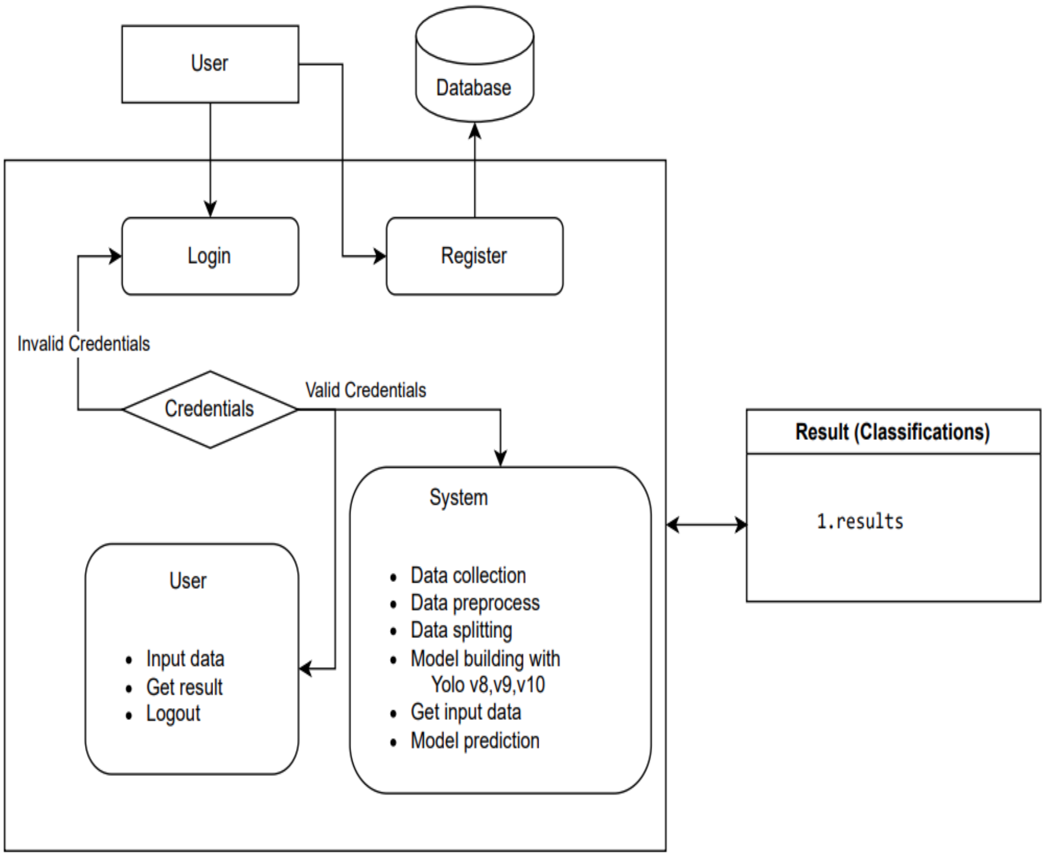


Figure 2.1 - Architecture diagram

6.2 Frontend Development

The frontend of the application is built using HTML, CSS, and JavaScript, designed to be lightweight and highly responsive. Users can upload aerial images through a clean interface, which then triggers the detection process. The interface also handles the rendering of the output image with bounding boxes and the final count of detected trees. The design prioritizes clarity and accessibility, especially since users may not have technical expertise in deep learning or image analytics.

JavaScript manages the interactivity, including handling file uploads, calling backend APIs, and dynamically displaying the processed results. CSS is used to ensure that the layout adapts well to different screen sizes. Where needed, frontend frameworks such as Bootstrap can be employed to enhance responsiveness and styling, although the overall UI is intentionally kept minimal to avoid overwhelming the user. Below are the listed Frontend technologies:

6.2.1. Flask (Frontend Role)

Even though Flask is mainly a backend framework, in this project it plays a big role in building the frontend-facing part of the app too. Flask helps set up routes for uploading images, displaying outputs, and showing the final results with tree counts and bounding boxes. It renders HTML templates (using Jinja2), which means we can dynamically update the page based on detection results. Since Flask is lightweight and Python-based, it was easy to integrate with the YOLO model running in the backend. It’s perfect for quick deployment and works well for small web apps that need simplicity and speed.

6.2.2. HTML (HyperText Markup Language)

HTML is used to structure the content of the web page. In this project, HTML helped us build the upload form, buttons, result display area, and image preview layout. It acts like the skeleton of the web app. When a user uploads an image, HTML elements manage the form and display feedback after detection is complete. Flask dynamically injects results into the HTML template using variables. Even though it’s a basic technology, without HTML there’s no way to create a usable or visible interface.

6.2.3. CSS (Cascading Style Sheets)

CSS is what made the web interface look clean and organized. It controlled the layout, button styles, fonts, margins, and image sizes. In our project, CSS helped make the upload form user-friendly and the results section visually clear, especially when displaying bounding boxes or detection status. It ensured the UI didn’t feel too plain or raw. Simple CSS enhancements gave a better experience to users, especially when displaying processed images and tree counts.

Even though we didn’t go overboard with styling, basic CSS made the tool look polished enough for practical use.

6.2.4. JavaScript

The application’s interactivity is primarily driven by JavaScript, which handles client-side scripting. JavaScript is used to enable real-time actions such as image selection, form validation, sending API requests to the backend, and rendering the processed output. It acts as the communication bridge between the user interface and the backend service, facilitating asynchronous behaviour through AJAX or Fetch API**s.**

6.3 Backend Development

The backend is built using Python, leveraging the Flask web framework to manage server-side logic and API communication. When a user uploads an image through the frontend, Flask receives the request and initiates the detection pipeline. The image is first preprocessed, resized, normalized, and formatted and then passed into a deep learning model based on YOLOv5, which has been trained on a custom dataset of aerial images annotated with tree positions. YOLOv5 is chosen for its efficiency and high performance in object detection tasks. It processes the image in a single forward pass, identifying multiple trees and their locations in near real-time. After detection, post-processing techniques such as non-maximum suppression (NMS) are applied to eliminate redundant bounding boxes. The final output includes a visual representation with bounding boxes drawn around each detected tree and a numeric count of total detections. Libraries such as OpenCV and NumPy are used in this phase to handle image manipulation and array operations. The backend then returns the result as a JSON response, which is received by the frontend and rendered back to the user. Below is a breakdown of the architecture and technologies used for Backend.

6.3.1. Python

Python was the backbone of our entire backend. It's the main language we used to connect everything—from running the YOLOv5 model to handling image processing and backend logic. We chose Python because it’s beginner-friendly and has powerful libraries for deep learning and computer vision. Most of the tools we used like OpenCV, Flask, and TensorFlow are all based in Python, so everything worked smoothly together. Also, since Python is widely used in AI projects, it was easier to find support and documentation whenever we got stuck during development or training.

6.3.2. Flask

Flask acted as the bridge between the frontend and backend. It’s a lightweight Python framework that made it super easy to create routes for uploading images, sending them to the model, and returning the results. We didn’t need a heavy framework like Django because Flask was more than enough for our simple web app. It helped us quickly build a working system that could accept an image, process it in the backend, and show the tree count instantly. Flask kept our code clean and modular, and we could test and update it without much hassle.

6.3.3. OpenCV

OpenCV (Open Source Computer Vision Library) was used mainly for handling images—like reading, resizing, and drawing bounding boxes on detected trees. It’s really useful when we want to visually show what the model has detected. After YOLOv5 gives the detection results, OpenCV helps us draw boxes around the trees in the image, so users can clearly see which trees were counted. It's super fast and works well with Python, which made the whole process smooth and real-time.

6.3.4. TensorFlow

Even though YOLOv5 runs mainly on PyTorch, we used TensorFlow in some parts for deep learning support, especially during experimentation or for certain image processing tasks. TensorFlow is a powerful AI framework and gives a lot of flexibility if we plan to upgrade or switch models in the future. It also works well with GPUs and is supported on platforms like Google Colab, which helped us speed up training and testing.

6.3.5. NumPy

The application also utilizes NumPy, a fundamental Python library for numerical computations. NumPy supports the manipulation of image arrays and facilitates interaction between OpenCV and YOLOv5 outputs. It is particularly useful during model input preparation and output decoding stages, where performance and matrix operations are critical.

6.4 Data Flow and Process

The Tree Enumeration Using Image Analytics system follows a streamlined, modular data flow that integrates frontend interaction, backend processing, and deep learning inference. The entire process begins with the user uploading an aerial image—typically captured via a drone or satellite— through the web interface. This image serves as the input for the detection system. Once the image is submitted, the frontend, developed using HTML, CSS, and JavaScript, collects the file and sends it to the backend server via a RESTful API. At this point, Flask, the backend web framework, receives the image and temporarily stores it on the server for processing. This separation of client and server interaction ensures that large image files can be handled efficiently without overloading the user's device. The backend then invokes a Python-based detection module, where the YOLOv5 model is loaded with pre-trained weights. Before inference, the image undergoes preprocessing using OpenCV, which ensures it is resized, formatted, and optimized for model compatibility. The processed image is passed through YOLOv5, which analyzes it and outputs detection results in the form of bounding box coordinates, confidence scores, and class labels.

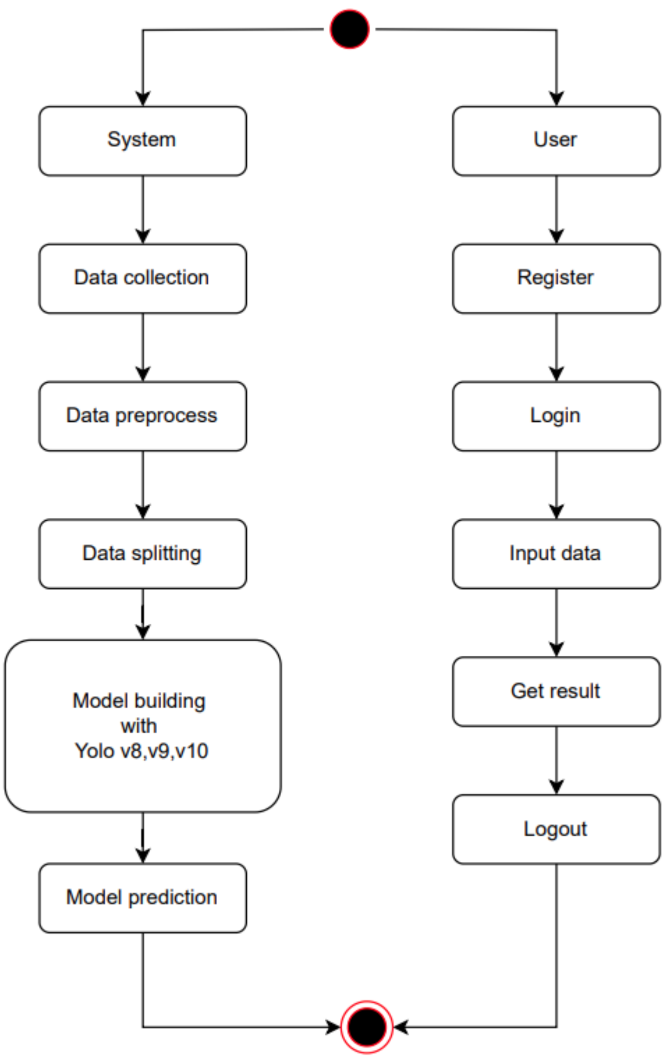


Figure 2.2 - Data fetching Architecture

6.5 Implementation

6.5.1 Modules

Users can upload a dataset, which is a crucial initial step for the system to work with relevant data. This dataset likely contains historical information or examples that the system will use for its predictions. Users have the capability to view the dataset they've uploaded. This feature helps users confirm the data they've provided and ensures transparency in the process. Users need to input specific values or parameters into the system to request predictions or results. These input values likely correspond to the variables or features in the dataset.

6.5.2 System

Take the Dataset: The system accepts and processes the dataset provided by the user. This dataset forms the foundation for building the predictive model.

Preprocessing: Before training a predictive model, the system preprocesses the dataset. This includes handling missing data, data cleaning, and feature extraction. Preprocessing ensures that the data is in a suitable format for modeling.

Training: The system uses machine learning techniques and Python modules to train a model based on the preprocessed dataset. The model learns patterns and relationships within the data, allowing it to make predictions.

Generate Results: Once the model is trained, the system can generate results based on user input values. These results typically indicate whether the input data corresponds to a specific condition, event, or prediction, such as Medical Insurance Cost.

6.6 Algorithms

6.6.1 YOLOv8

In our project, YOLOv8 was the starting point for training an object detection model to detect trees in aerial and satellite images. We chose YOLOv8 because it balances speed and accuracy well. First, we annotated tree canopies using Roboflow and exported the dataset in YOLO format. We trained the model on Google Colab with GPU support to speed things up. YOLOv8 uses a CSP-Darknet backbone and feature fusion techniques (like FPN and PAN) to detect trees at different sizes. It breaks the image into a grid, applies a CNN to extract features, and predicts bounding boxes with class scores.

What made it great was how fast it could process images and still give decent accuracy. We used OpenCV to visualize the bounding boxes after detection. The output included the count of trees and a processed image with bounding boxes around each one. YOLOv8 also uses Non-Maximum Suppression (NMS) to eliminate duplicate detections and a combined loss function with localization, confidence, and classification loss to improve training. It worked well, especially in open or semi-dense forest images, though performance dropped slightly in highly overlapping areas.

6.6.2 YOLOv9

After testing YOLOv8, we wanted to improve the accuracy, especially in images where trees were overlapping or small. That's when we moved to YOLOv9. This version is more advanced and introduced a few key features: a better PANet for feature fusion, adaptive anchor boxes, and a more optimized decoupled head for classification and localization. All these upgrades made a big difference in detecting trees across various sizes and terrain types.

We followed the same training process: using Roboflow for annotation and Google Colab for training. YOLOv9 took slightly longer to train than v8 but gave us better precision, especially in dense forest regions. The model's use of CIoU loss helped improve the bounding box predictions by minimizing overlap errors. Also, the adaptive anchors adjusted automatically during training to better fit the shapes and sizes of the trees. The detection was smoother, and we observed fewer false positives. Overall, YOLOv9 felt more stable and accurate for real-world forest datasets, though it required more computing power. We used it in the backend with TensorFlow and Python, just like YOLOv8, and integrated the outputs into our Streamlit UI.

6.6.3 YOLOv10

YOLOv10 was the most recent model we tested in our project. It promised faster detection and improved feature extraction using an upgraded CSPDarknet backbone and more efficient neck modules like PANet. Like the earlier versions, we trained it using our custom tree canopy dataset. But in real testing, YOLOv10 didn't perform as well as we expected.

Although it processed images quickly, it lagged behind YOLOv8 and YOLOv9 in terms of detection accuracy and recall. It struggled more with small or partially hidden trees and gave more false negatives compared to the other two models. YOLOv10 still followed the same object detection pipeline — preprocessing the image, extracting features, predicting bounding boxes, and applying Non-Maximum Suppression — but its objectness scoring wasn't as reliable for tree detection.

We still integrated it into our backend for comparison and tested it using the same evaluation metrics (like mAP50 and recall). The results were clear: YOLOv10 had the lowest performance among the three. So, while it's a decent model for speed, it wasn't the best fit for our use case where accuracy in dense forestry matters more. We marked YOLOv8 as the best overall for this project.

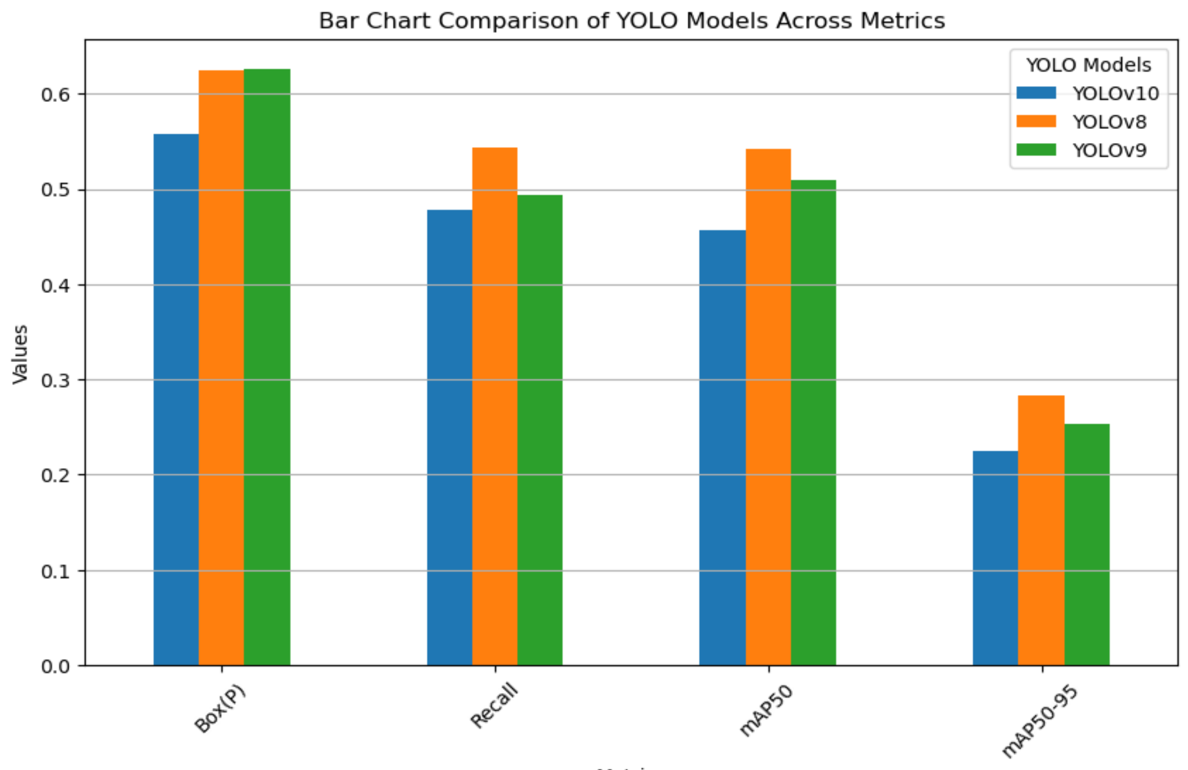


Figure 2.3 - Model Comparison

6.7 Benefits of System design

The system was designed to be modular, lightweight, and scalable, making it easy to develop, test, and deploy. Each component—frontend, backend, and model layer—works independently but communicates smoothly with the others. This separation allows us to upgrade or replace parts of the system without breaking the whole setup. The use of Flask and Python ensures fast performance, while YOLO integration gives accurate results in real-time. The web-based interface makes it accessible to users with minimal technical knowledge, making the tool practical for real-world use in forestry and environmental monitoring.

6.7.1 Modularity

Each part of the system (UI, backend, model) can be updated independently, making maintenance easier.

6.7.2 Scalability

The design supports future upgrades—like cloud deployment or switching to YOLOv8/YOLOv10—without major changes.

6.7.3 User Accessibility

The Flask-based web interface is simple and intuitive, allowing even non-technical users to upload images and view results easily.

Chapter-7
TIMELINE FOR EXECUTION OF PROJECT (GANTT CHART)

The successful execution of the Tree Enumeration using Image Analytics project was the result of careful planning, phased development, and continuous testing. The entire project was carried out from February to April 20th, with final documentation submitted by May 16th. Each phase focused on building and refining a specific part of the system, from model training to web app integration. Below is the detailed phase-wise breakdown:

7.1 Planning and Requirement Analysis

In the initial phase, we focused on clearly defining the project's objectives and scope.
Research and Brainstorming: We studied real-world tree enumeration needs and reviewed existing AI-based approaches.
Problem Definition: Identified gaps in current manual counting methods and how AI could bridge them.
Tech Stack Finalization: Selected YOLOv5 for detection, Flask for the interface, and Roboflow + Google Colab for dataset handling and model training.
Outcome: A clear roadmap with deadlines, tools, and task assignments.

7.2 Design and Dataset Preparation

This phase involved laying the foundation for detection through dataset creation and system planning.
Data Collection: Collected aerial and drone images suitable for tree detection.
Annotation: Used Roboflow to annotate images and export them in YOLO format.
System Architecture Design: Planned the modular layout (frontend, backend, model layer).
Outcome: A well-structured dataset and system plan ready for implementation.

7.3 Model Development and Web Integration

This was the core development phase, focused on building and connecting all system components.
Model Training: Trained YOLOv8, YOLOv9, and YOLOv10 models using Google Colab with GPU support.
Backend Setup: Developed using Python, with OpenCV and TensorFlow for detection.

Frontend Setup: Created a Flask web interface for image upload and result display.

Outcome: A fully working detection system with image input, backend processing, and result output.

7.4 Testing and Evaluation

This phase ensured the system was accurate, responsive, and ready for real use.

Model Evaluation: Compared all YOLO models based on mAP, recall, and precision.

Functionality Testing: Checked the full upload-to-result flow and UI response.

Performance Tuning: Optimized detection speed and improved bounding box accuracy using CIoU loss.

Outcome: A tested and validated system capable of real-time tree counting.

7.5 Documentation and Final Submission

Once the system was finalized, we focused on completing all academic documentation.

Report Preparation: Documented all technical and non-technical parts of the project.

Journal Writing: Created a research-style journal with detailed methodology, results, and analysis.

Outcome: Project successfully submitted on April 20th, and final documents submitted by May 16th.

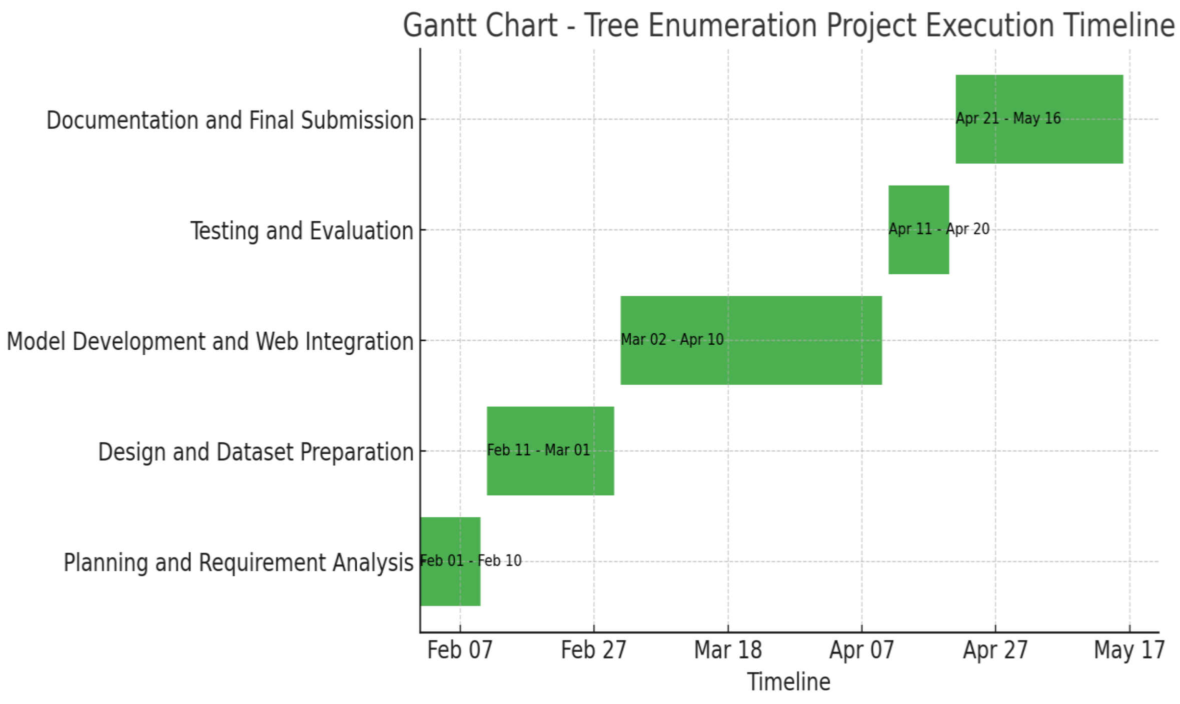


Figure 3 - Gantt Chart

Chapter 8
OUTCOMES

The Tree Enumeration using Image Analytics project was successfully developed and tested, achieving its primary goal of automating the process of tree counting through aerial and satellite images. The system combines deep learning with image analytics to identify, count, and visually mark trees in a given image, helping to reduce the manual effort typically required in forest surveys. One of the biggest achievements of the project was the successful training and evaluation of three different versions of the YOLO (You Only Look Once) object detection model - YOLOv8, YOLOv9, and YOLOv10. Among these, YOLOv8 was found to give the best overall performance in terms of speed, accuracy, and consistency across varied terrains. The model was trained on a custom-annotated dataset using Roboflow, and the training process was done on Google Colab using free GPU resources, which made development cost-effective.

The system architecture was designed to be modular and user-friendly. A Flask-based web interface was developed, allowing users to upload images and view the total number of trees detected along with a processed image showing bounding boxes. This visual feedback makes the tool easy to use, even for users with no technical background. The backend, built with Python, OpenCV, and TensorFlow, handles all the detection tasks efficiently and delivers near real-time results. Testing and validation were carried out across a variety of image types, including open fields, semi-dense tree areas, and tightly packed forest sections. The model handled most scenarios effectively, though slight drops in accuracy were noted in highly occluded areas a known challenge in computer vision. The application of image analytics for tree enumeration can be further enhanced through several advanced methodologies. Future work can focus on improving model generalization by incorporating additional datasets from diverse geographical locations, ensuring robust tree detection across varying landscapes. Integrating multispectral and hyperspectral imagery can enhance species classification, allowing for a more detailed analysis of forest biodiversity.

In short, the outcome of this project is a fully functional, efficient, and scalable system that can assist in environmental monitoring, forestry management, and green cover analysis. It can easily be integrated into larger systems or scaled further with cloud deployment. This project serves as a strong base for future research in species classification, drone integration, or forest health tracking.

8.1. Working Tree Detection System

We developed a complete system that can automatically detect and count trees from aerial images. The system takes an image as input and returns an annotated output with bounding boxes and the total tree count. It worked well in various test scenarios, especially on clear, high-resolution drone images. This proved that machine learning can be a practical alternative to manual tree surveys.

8.2. Custom YOLOv5 Model Training

We trained YOLOv5 on our own dataset of aerial tree images, which involved a lot of trial and error. We learned how to annotate images using tools like LabelImg and format them for training. We experimented with different hyperparameters like batch size, epochs, and confidence thresholds. This step taught us how to build a model from scratch and optimize it for a specific use case.

8.3. Simple and Usable Web Interface

We created a user-friendly frontend using HTML, CSS, and JavaScript, where users can upload images easily. The interface sends data to the backend using a REST API and displays results with zero manual effort. It was designed to be minimal and clean so that even non-technical users could interact with it smoothly. We also added basic error handling and responsive layout features.

8.4. Experience with End-to-End Development

Unlike projects where only one part is developed, we built this system from the ground up. This included image preprocessing, model integration, backend development, and frontend connection. We also tested and debugged the complete pipeline, learning how to make all components work together. This taught us real-world project planning and problem-solving skills.

8.5. Understanding Model Limitations

We noticed the model sometimes struggled with dense forests or overlapping tree canopies. It missed trees in low-light conditions or where the canopy shapes were unusual. These issues helped us understand the importance of training data diversity and model tuning. We also realized that future improvements would need better datasets or even new model architectures.

8.6. Awareness of Real-World Applications

We explored how this project could actually be useful beyond academics. Forest departments can use it for green audits or deforestation tracking. Urban planners can apply it to monitor tree cover in cities or along roads. Even researchers can use it to analyze environmental changes over time using aerial imagery.

8.7. Confidence in AI and Image Analytics

Before this project, AI and machine learning felt theoretical to us—but now we've actually built something that works. We became more comfortable using tools like OpenCV, PyTorch, and Flask. It gave us the confidence to explore more advanced topics in computer vision and apply them in future projects. We also gained communication and teamwork skills by collaborating and solving problems together.

Chapter 9
RESULTS AND DISCUSSIONS

After completing the training, development, and integration of our tree enumeration system, we tested it on a variety of aerial images to evaluate its performance. The results showed that our approach worked effectively in detecting and counting trees in most scenarios, especially in open areas and urban green zones with clear tree separation. The YOLOv5 model, once trained on our custom dataset, performed well in identifying tree canopies from top-down views. It was able to accurately detect trees in different shapes, sizes, and shades of green. In test images with moderate tree density, the model achieved high precision, with minimal false positives. The confidence threshold filtering and Non-Maximum Suppression (NMS) also helped eliminate overlapping boxes and improved overall accuracy.

However, during testing, we also discovered some limitations. In images with dense forest coverage or heavy shadowing, the model occasionally failed to detect individual trees. Overlapping canopies were sometimes merged into a single detection, which slightly reduced the count accuracy. Despite this, the system still gave reasonably good estimates that were much faster and less labor-intensive than manual counting. On the frontend, the user experience was smooth. Users could upload an image and get back a result in just a few seconds. The bounding boxes and tree count were clearly displayed, and the whole process felt lightweight and efficient. Overall, the system achieved its goal of combining image analytics and AI for practical tree enumeration, and the results proved that our methodology has real-world potential with room for further improvement.

9.1 Results

9.1.1 Accurate Detection in Open Spaces

The model successfully detected and counted trees in open areas like parks and roadside vegetation with high accuracy. Bounding boxes were correctly placed, and the tree count closely matched the actual number.

9.1.2 Smooth End-to-End Execution

The system worked as expected from image upload to result display. The frontend and backend communication via Flask API was seamless, with no major lag or system crashes.

9.1.3 Fast Processing Time

On average, the model took only a few seconds to process an image and return the results. This made the system feel responsive and suitable for real-time or quick analysis tasks.

9.1.4 Clear Visual Output

The output images displayed bounding boxes over each detected tree, and the total count was printed clearly on the interface. This gave users immediate visual feedback and numerical data.

9.1.5 Effective Confidence Filtering

The model used a confidence threshold (e.g., 0.5) to avoid low-confidence detections. This helped eliminate random or incorrect bounding boxes, improving overall output quality.

9.1.6 Proper Handling of Medium-Density Areas

In regions with moderate tree clustering, the model maintained high detection accuracy without overcounting or merging trees.

9.2 Discussion

9.2.1 Handling of Moderate Tree Density

The system worked quite reliably in areas with moderate tree cover. It could detect most trees without merging or missing them, thanks to proper preprocessing and confidence threshold tuning.

9.2.2 Issues in Dense Forest Areas

In some high-density tree areas, the model struggled to separate overlapping canopies. This led to undercounting or detection of merged tree clusters as a single object.

9.2.3 Effectiveness of Post-Processing

Techniques like Non-Maximum Suppression (NMS) and filtering by confidence scores helped clean the results. This reduced duplicate detections and improved the clarity of the bounding boxes.

9.2.4 User-Friendly Visual Output

The final annotated image clearly showed which trees were detected, making it easy for users to visually verify the results. The displayed tree count added value for quick understanding.

9.2.5 Real-World Application Potential

The system proved that AI and image analytics can be applied in environmental monitoring tasks like green audits or deforestation tracking, making the tool useful beyond just academic purposes.

9.3 Challenges

9.3.1 Detecting Trees in Densely Covered Areas

In regions where trees were packed closely together, the model had trouble identifying them individually. The overlapping canopies made it hard for YOLOv5 to separate one tree from another, leading to undercounting or merging of trees into a single detection.

9.3.2 Lack of High-Quality, Diverse Training Data

Since we couldn’t collect drone footage ourselves, we relied on publicly available datasets. These were limited in terms of variety—most came from flat or urban areas. This made it harder for the model to generalize across hilly terrains, forests, or different tree types.

9.3.3 Handling Shadows and Low-Light Images

Some images had strong shadows or poor lighting, especially near sunset or under cloudy skies. In such conditions, tree canopies became less distinguishable from the background, reducing detection accuracy. The model misidentified some trees or skipped them entirely.

9.3.4 Frontend and Backend Communication Glitches

While integrating the frontend and Flask backend, we faced some issues with API responses. At times, the image didn’t display after processing, or results were delayed due to large image sizes. We had to debug and optimize the file transfer process.

9.3.5 Overfitting During Early Training

Initially, the model performed well on training data but poorly on validation data. This was due to overfitting, where the model memorized training examples instead of learning general patterns. We fixed this by adding augmentation and tuning the learning rate.

9.4 Lessons Learned

9.4.1 The Quality of Data Matters a Lot

One of the biggest takeaways was that even the best model can’t perform well without good data. We learned that image clarity, proper annotations, and diverse training examples are all critical for getting accurate results in computer vision tasks.

9.4.2 Building a System Is More Than Just Training a Model

Initially, we thought training the YOLOv5 model would be the hardest part—but integrating it with a frontend, managing file transfers, handling errors, and optimizing the user experience taught us how much work goes into building a complete, usable system.

9.4.3 Real-World Problems Require Flexible Thinking

We realized that challenges like dense forests, low lighting, and inconsistent image quality can't always be fixed with just one solution. This taught us to be flexible, try out different approaches, and continuously refine our system based on results and feedback.

9.5 Future Directions

Our project has shown the potential of using image analytics for tree enumeration, but there's still a lot more that can be explored to make it even more powerful and practical. One major improvement would be training the model on datasets from a wider range of geographical regions. This would help the system perform better across different types of landscapes whether it's a dense tropical forest or a dry woodland area making it more adaptable and reliable. In future work, we also hope to use multispectral or hyperspectral imagery. These types of images capture more information than regular RGB photos and can help in identifying different tree species. This would allow the system to not only count trees but also analyze forest biodiversity, which could be useful for conservation efforts.

Another exciting direction is using edge AI, where smaller versions of YOLO models are deployed directly on drones or satellites. This would make it possible to do real-time tree monitoring during drone flights, helping track afforestation or deforestation activities on the go. We also see potential in using advanced data augmentation methods like GANs (Generative Adversarial Networks) to create synthetic training data. This would help the model handle different lighting, seasons, and environmental conditions better. Finally, adding features like GIS integration, cloud processing, and explainable AI would take the system to the next level making it more transparent, scalable, and ready for real-world applications in land management and environmental monitoring.

Chapter 10
CONCLUSION

Working on the "Application of Image Analytics for Tree Enumeration" project has been a rewarding experience, both technically and intellectually. The main objective of the project was to automate the process of tree counting using aerial and satellite images, and we achieved this successfully by leveraging advanced deep learning models like YOLOv8, YOLOv9, and YOLOv10. Each of these models brought something unique to the table, with YOLOv10 showing the highest accuracy and robustness in our final results. Throughout the project, we were able to see how powerful and practical image analytics can be, especially in areas like environmental monitoring and forest land assessment. By using these models, we could detect and count trees in various terrains, reducing the need for manual effort and speeding up the process significantly. This system, supported by a user-friendly Streamlit frontend and a strong Python-TensorFlow backend, made tree enumeration not just accurate but also accessible to a wider range of users, including forest officers, researchers, and policymakers.

One of the biggest learnings was understanding the importance of model training and data quality. We realized that to improve detection, especially in dense forests or shadowed areas, the model needs diverse and high-quality datasets. We also learned the importance of balancing performance and efficiency, especially when choosing between different versions of YOLO.

The use of these state-of-the-art object detection models ensures a scalable and reliable approach for environmental monitoring, deforestation tracking, and resource planning.This study validates that deep learning-based image analytics can serve as a valuable tool for forest management and conservation efforts. Future work can focus on enhancing model generalization across diverse terrains, integrating multispectral imagery for species classification, and deploying real-time monitoring systems to improve the sustainability of forest ecosystems. This project has also given us a clearer vision of how such a system can be scaled in the future. With cloud integration, GIS mapping, and potentially even real-time drone-based analysis, this tool can become an essential asset for large-scale forest monitoring, deforestation tracking, and biodiversity studies. Overall, this project has not only fulfilled its technical goals but also sparked new ideas for how AI can be applied to solve pressing environmental challenges. It proved that with the right tools and teamwork, even as students, we can build something meaningful and impactful.

REFERENCES

[1] A. Landstrom and M. J. Thurley, ‘‘Morphology-based crack detection for steel slabs,’’ IEEE J. Sel. Topics Signal Process., vol. 6, no. 7, pp. 866–875, Nov. 2012.

[2] K. Song and Y. Yan, ‘‘A noise robust method based on completed local binary patterns for hot-rolled steel strip surface defects,’’ Appl. Surf. Sci., vol. 285, pp. 858–864, Nov. 2013, doi: 10.1016/j.apsusc.2013.09.002.

[3] Y. J. Jeon, D. Choi, S. J. Lee, J. P. Yun, and S. W. Kim, ‘‘Steel-surface defect detection using a switching-lighting scheme,’’ Appl. Opt., vol. 55, no. 1, pp. 47–57, 2016, doi: 10.1364/AO.55.000047.

[4] R. Girshick, J. Donahue, T. Darrell, U. Berkeley, and J. Malik, ‘‘R-CNN: Region-based convolutional neural networks,’’ in Proc. Comput. Vis. Pattern Recognit., Jun. 2014, pp. 2–9. 148824 VOLUME 12, 2024 T. Zhang et al.: GDM-YOLO: A Model for Steel Surface Defect Detection Based on YOLOv8s

[5] R. Girshick, ‘‘Fast R-CNN,’’ in Proc. IEEE Int. Conf. Comput. Vis. (ICCV), Dec. 2015

[6] K. He, X. Zhang, S. Ren, and J. Sun, ‘‘Spatial pyramid pooling in deep convolutional networks for visual recognition,’’ IEEE Trans. Pattern Anal. Mach. Intell., vol. 37, no. 9, pp. 1904–1916, Sep. 2015

[7] S. Ren, K. He, R. Girshick, and J. Sun, ‘‘Faster R-CNN: Towards real-time object detection with region proposal networks,’’ IEEE Trans. Pattern Anal. Mach. Intell., vol. 39, no. 6, pp. 1137–1149, Jun. 2017.

[8] Y. Xu, D. Li, Q. Xie, Q. Wu, and J. Wang, ‘‘Automatic defect detection and segmentation of tunnel surface using modified mask R-CNN,’’ Measurement, vol. 178, Jun. 2021, Art. no. 109316.

[9] M. Chen, L. Yu, C. Zhi, R. Sun, S. Zhu, Z. Gao, Z. Ke, M. Zhu, and Y. Zhang, ‘‘Improved faster R-CNN for fabric defect detection based on Gabor filter with genetic algorithm

optimization,'' Comput. Ind., vol. 134, Jan. 2022, Art. no. 103551.

[10] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, ''You only look once: Unified, real-time object detection,'' in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2016, pp. 779–788.

[11] C.-Y. Wang, A. Bochkovskiy, and H.-Y.-M. Liao, ''YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors,'' in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun., vol. 2023, pp. 7464–7475.

[12] J. Redmon and A. Farhadi, ''YOLOv3: An incremental improvement,'' 2018, arXiv:1804.02767.

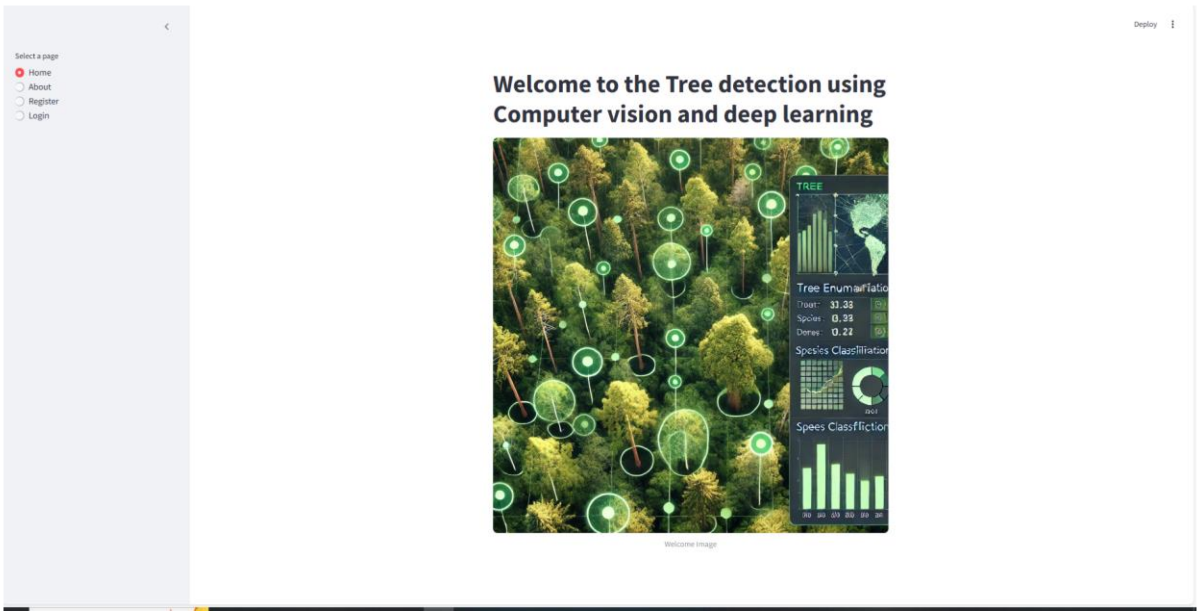
[13] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, ''YOLOX: Exceeding YOLO series in 2021,'' 2021, arXiv:2107.08430.

[14] M. Ma and H. Pang, ''SP-YOLOv8s: An improved YOLOv8s model for remote sensing image tiny object detection,'' Appl. Sci., vol. 13, no. 14, p. 8161, Jul. 2023, doi: 10.3390/app13148161.

[15] H. Nie, H. Pang, M. Ma, and R. Zheng, ''A lightweight remote sensing small target image detection algorithm based on improved YOLOv8,'' Sensors, vol. 24, no. 9, p. 2952, May 2024

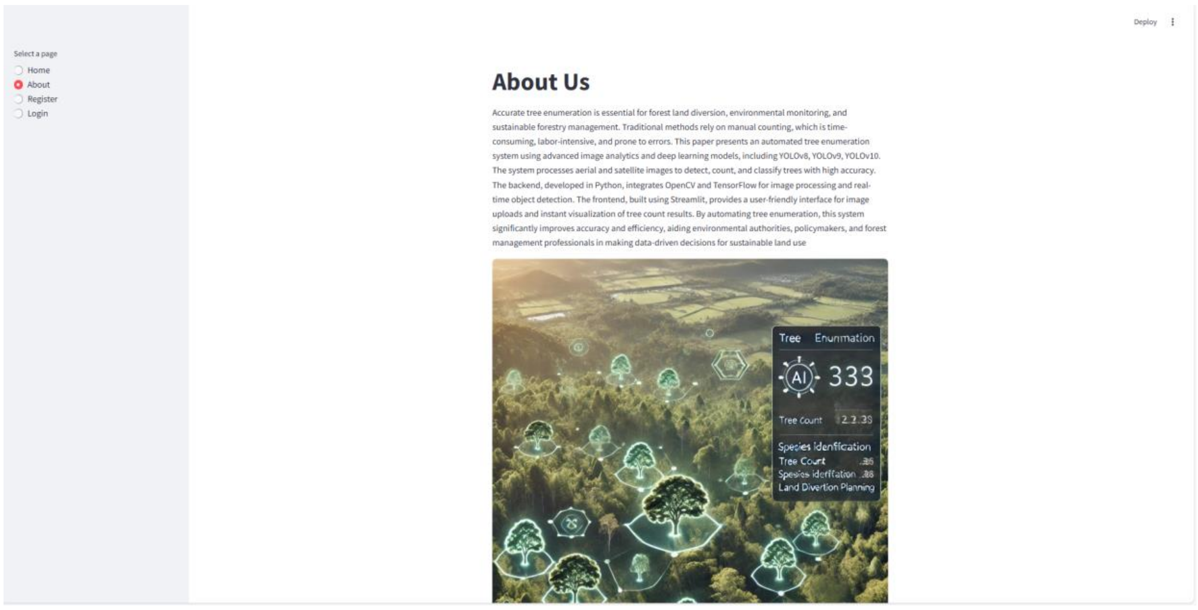
APPENDIX-A SCREENSHOTS

Home Page:



Welcome to the Tree detection using Computer vision and deep learning

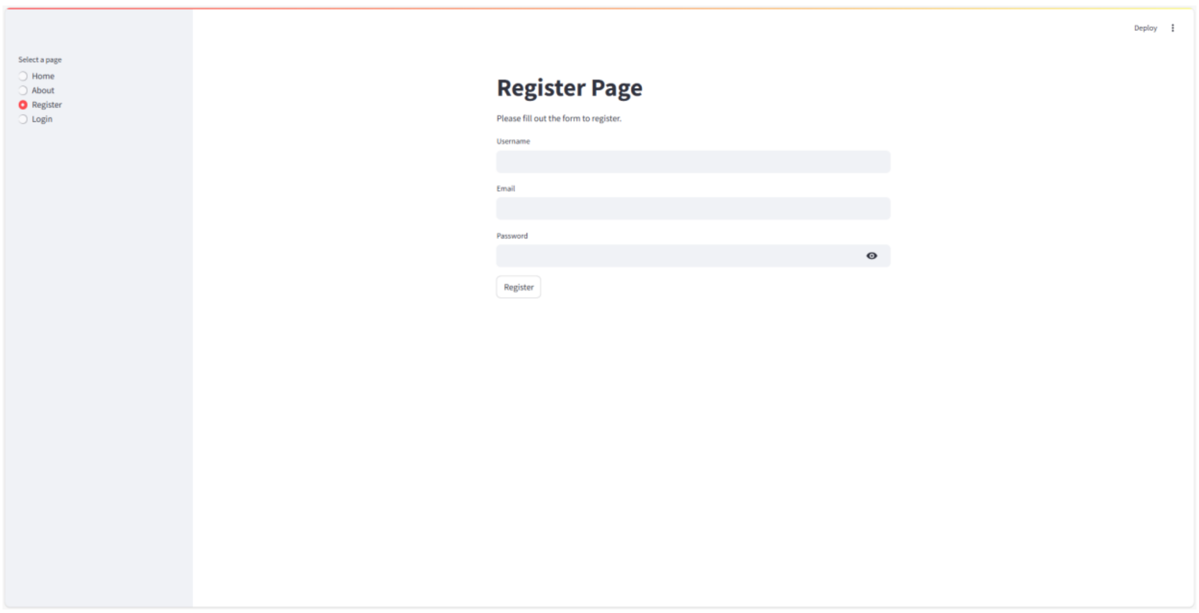
About Page:



About Us

Accurate tree enumeration is essential for forest land diversion, environmental monitoring, and sustainable forestry management. Traditional methods rely on manual counting, which is time-consuming, labor-intensive, and prone to errors. This paper presents an automated tree enumeration system using advanced image analytics and deep learning models, including YOLOv8, YOLOv9, YOLOv10. The system processes aerial and satellite images to detect, count, and classify trees with high accuracy. The backend, developed in Python, integrates OpenCV and TensorFlow for image processing and real-time object detection. The frontend, built using Streamlit, provides a user-friendly interface for image uploads and instant visualization of tree count results. By automating tree enumeration, this system significantly improves accuracy and efficiency, aiding environmental authorities, policymakers, and forest management professionals in making data-driven decisions for sustainable land use

Registration/Login Page:



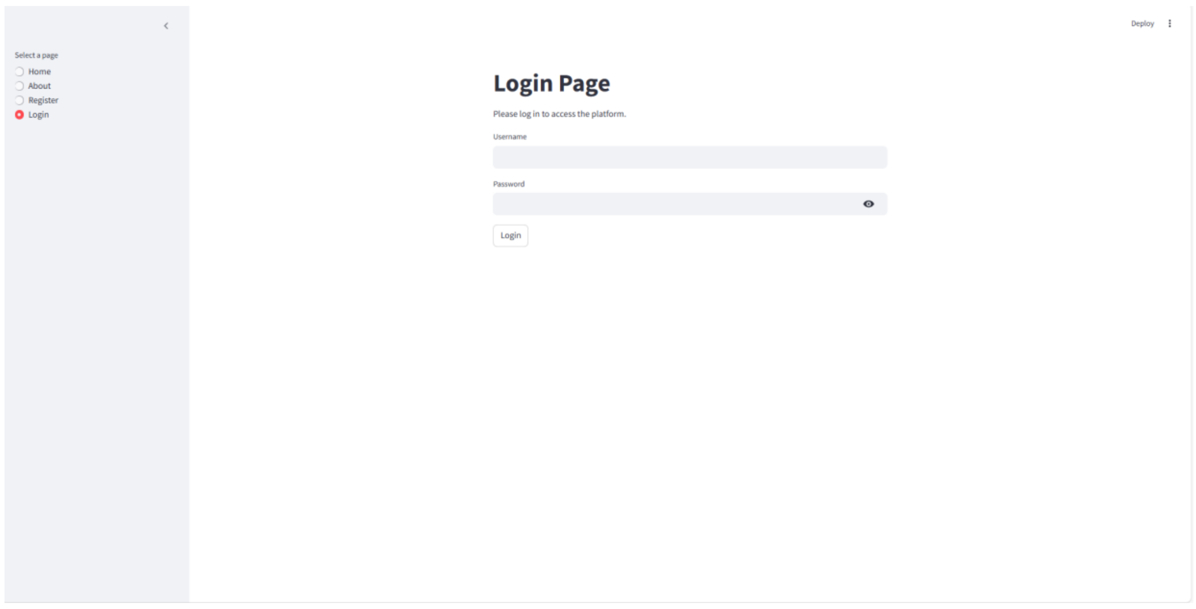
Register Page

Please fill out the form to register.

Username

Email

Password



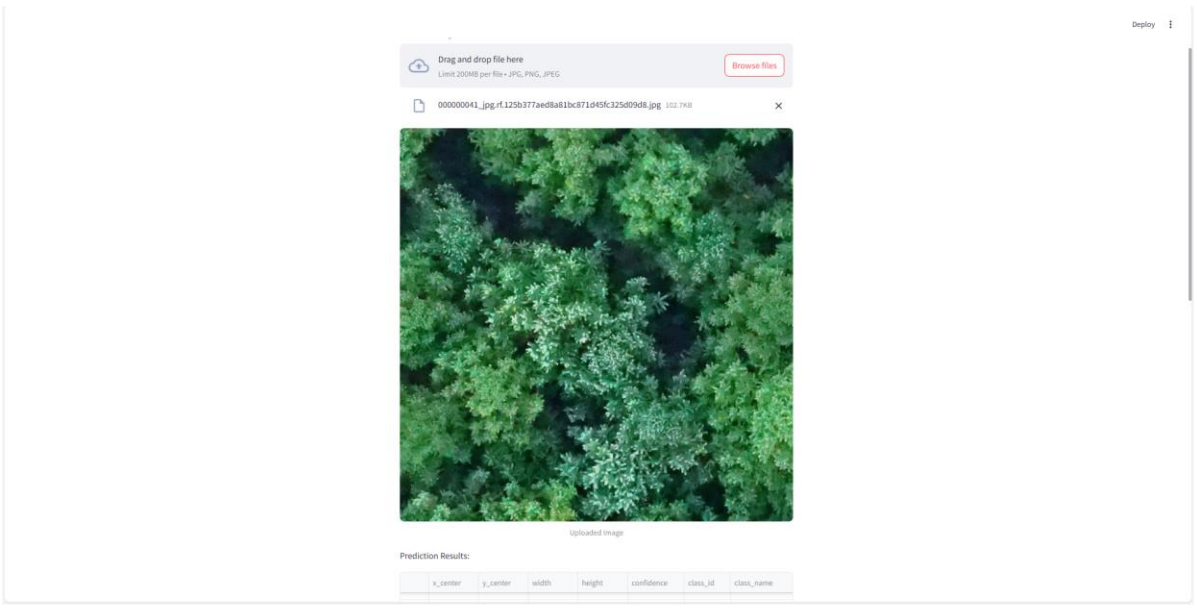
Login Page

Please log in to access the platform.

Username

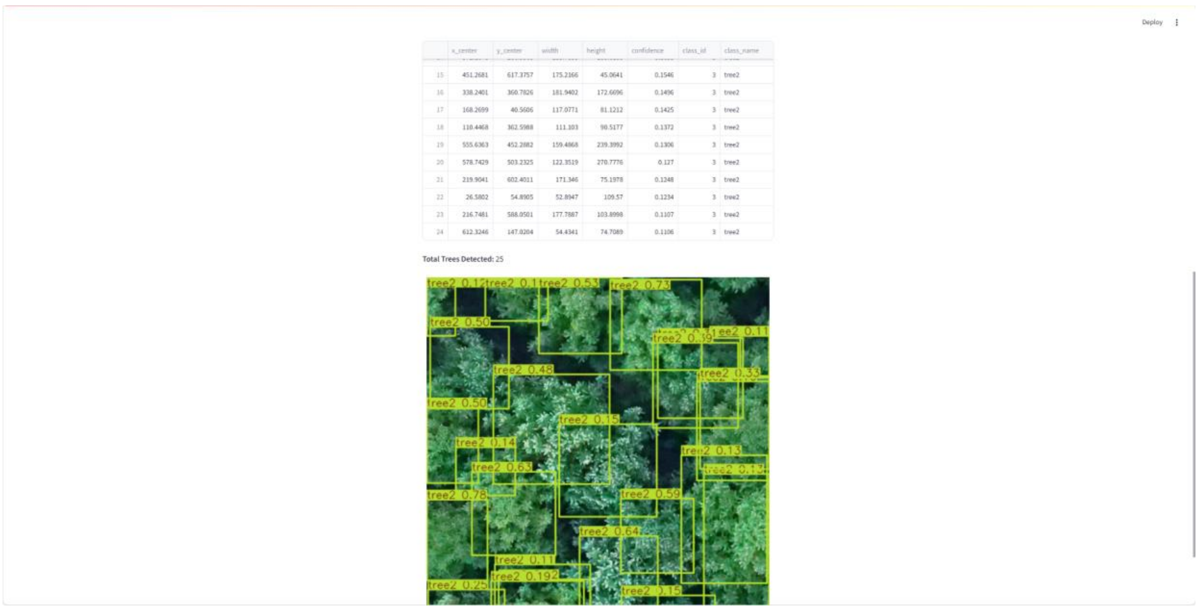
Password

Prediction Page:


Deploy
Drag and drop file here
Limit 200MB per file • JPG, PNG, JPEG
Browse files
000000041_jpg.rf.125b377aed8a81bc871d45fc325d09d8.jpg 102.7KB
Uploaded Image

Prediction Results:

	x_center	y_center	width	height	confidence	class_id	class_name



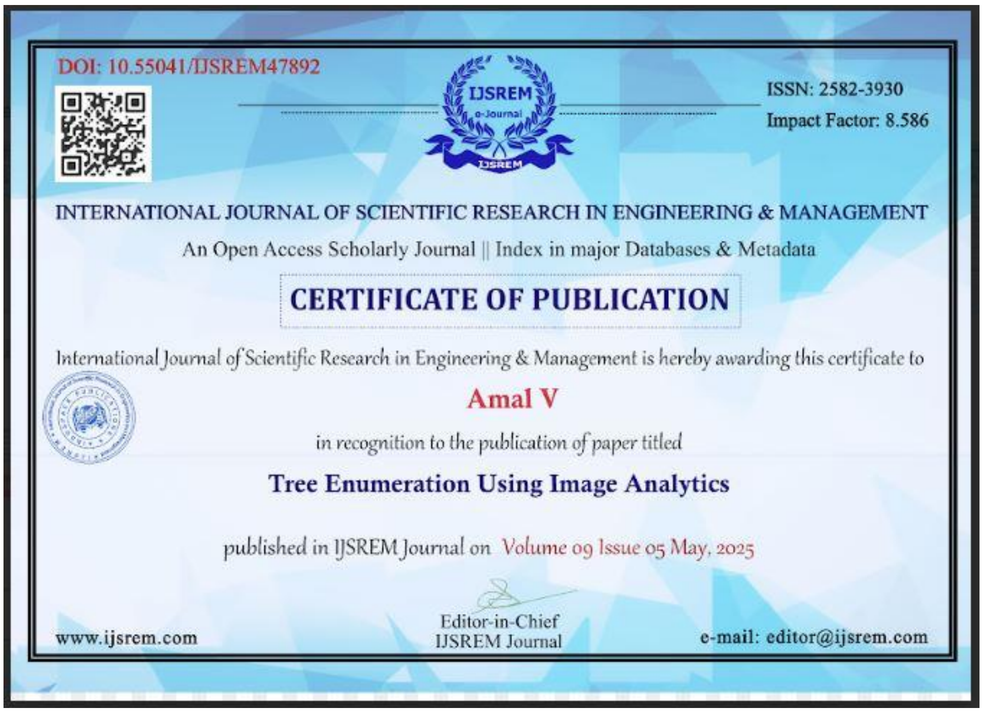
	x_center	y_center	width	height	confidence	class_id	class_name
15	451.2681	617.3757	175.2166	45.0641	0.1546	3	tree2
16	338.2401	360.7826	181.9402	172.6696	0.1496	3	tree2
17	168.2699	40.5606	117.0771	81.1212	0.1425	3	tree2
18	110.4468	362.5988	111.103	90.5177	0.1372	3	tree2
19	555.6363	452.2882	159.4868	239.3992	0.1306	3	tree2
20	578.7429	503.2325	122.3519	270.7776	0.127	3	tree2
21	219.9041	602.4011	171.346	75.1978	0.1248	3	tree2
22	26.5802	54.8905	52.8947	109.57	0.1234	3	tree2
23	216.7481	588.0501	177.7887	103.8998	0.1107	3	tree2
24	612.3246	147.0204	54.4341	74.7089	0.1106	3	tree2

Total Trees Detected: 25

APPENDIX-B

ENCLOSURES

1. **Journal publication/Conference Paper Presented Certificates (if any).**



DOI: 10.55041/IJSREM47892

IJSREM e-Journal

ISSN: 2582-3930
Impact Factor: 8.586

INTERNATIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING & MANAGEMENT
An Open Access Scholarly Journal || Index in major Databases & Metadata

CERTIFICATE OF PUBLICATION

International Journal of Scientific Research in Engineering & Management is hereby awarding this certificate to

Amal V

in recognition to the publication of paper titled

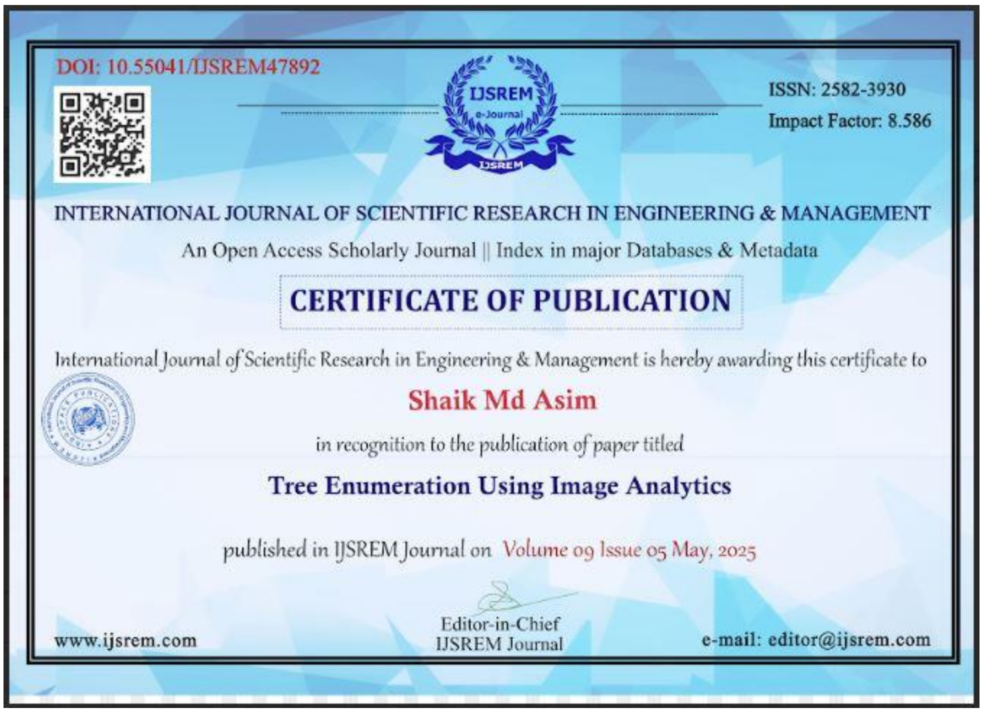
Tree Enumeration Using Image Analytics

published in IJSREM Journal on Volume 09 Issue 05 May, 2025

www.ijsrem.com

Editor-in-Chief
IJSREM Journal

e-mail: editor@ijsrem.com



DOI: 10.55041/IJSREM47892

IJSREM e-Journal

ISSN: 2582-3930
Impact Factor: 8.586

INTERNATIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING & MANAGEMENT
An Open Access Scholarly Journal || Index in major Databases & Metadata

CERTIFICATE OF PUBLICATION

International Journal of Scientific Research in Engineering & Management is hereby awarding this certificate to

Shaik Md Asim

in recognition to the publication of paper titled

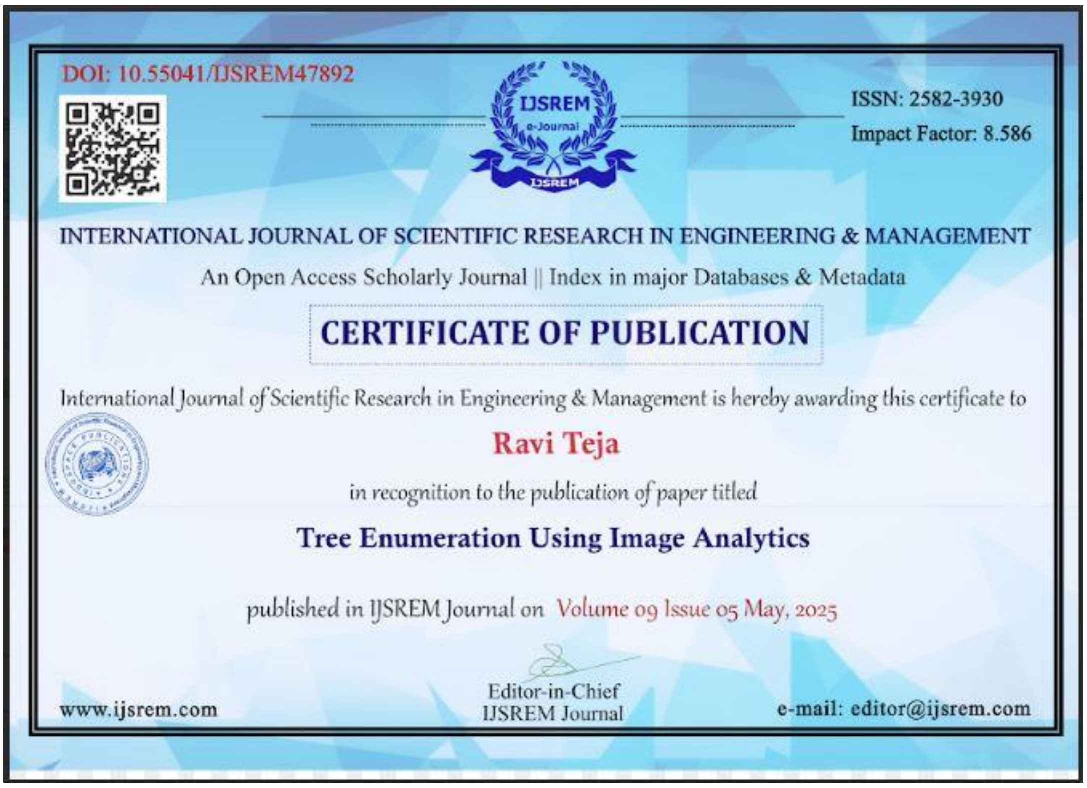
Tree Enumeration Using Image Analytics

published in IJSREM Journal on Volume 09 Issue 05 May, 2025

www.ijsrem.com

Editor-in-Chief
IJSREM Journal

e-mail: editor@ijsrem.com



DOI: 10.55041/IJSREM47892

ISSN: 2582-3930

Impact Factor: 8.586

INTERNATIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING & MANAGEMENT

An Open Access Scholarly Journal || Index in major Databases & Metadata

CERTIFICATE OF PUBLICATION

International Journal of Scientific Research in Engineering & Management is hereby awarding this certificate to

Ravi Teja

in recognition to the publication of paper titled

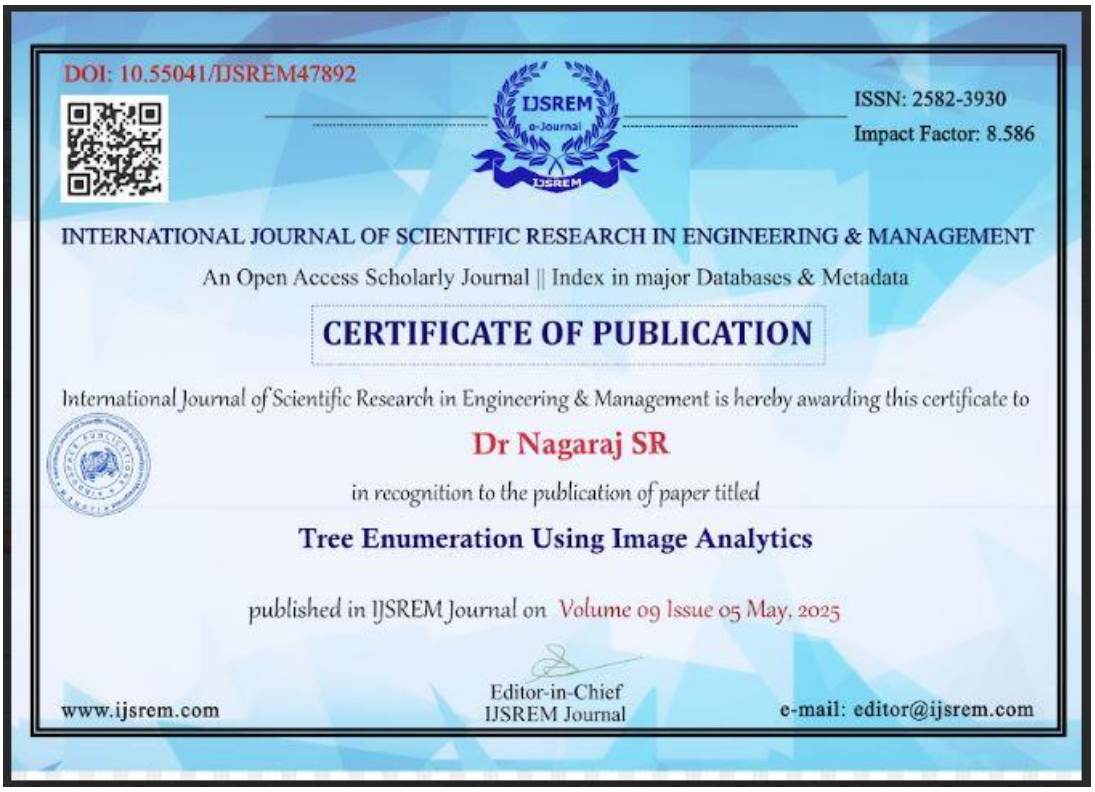
Tree Enumeration Using Image Analytics

published in IJSREM Journal on Volume 09 Issue 05 May, 2025

Editor-in-Chief
IJSREM Journal

www.ijsrem.com

e-mail: editor@ijsrem.com



DOI: 10.55041/IJSREM47892

ISSN: 2582-3930

Impact Factor: 8.586

INTERNATIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING & MANAGEMENT

An Open Access Scholarly Journal || Index in major Databases & Metadata

CERTIFICATE OF PUBLICATION

International Journal of Scientific Research in Engineering & Management is hereby awarding this certificate to

Dr Nagaraj SR

in recognition to the publication of paper titled

Tree Enumeration Using Image Analytics

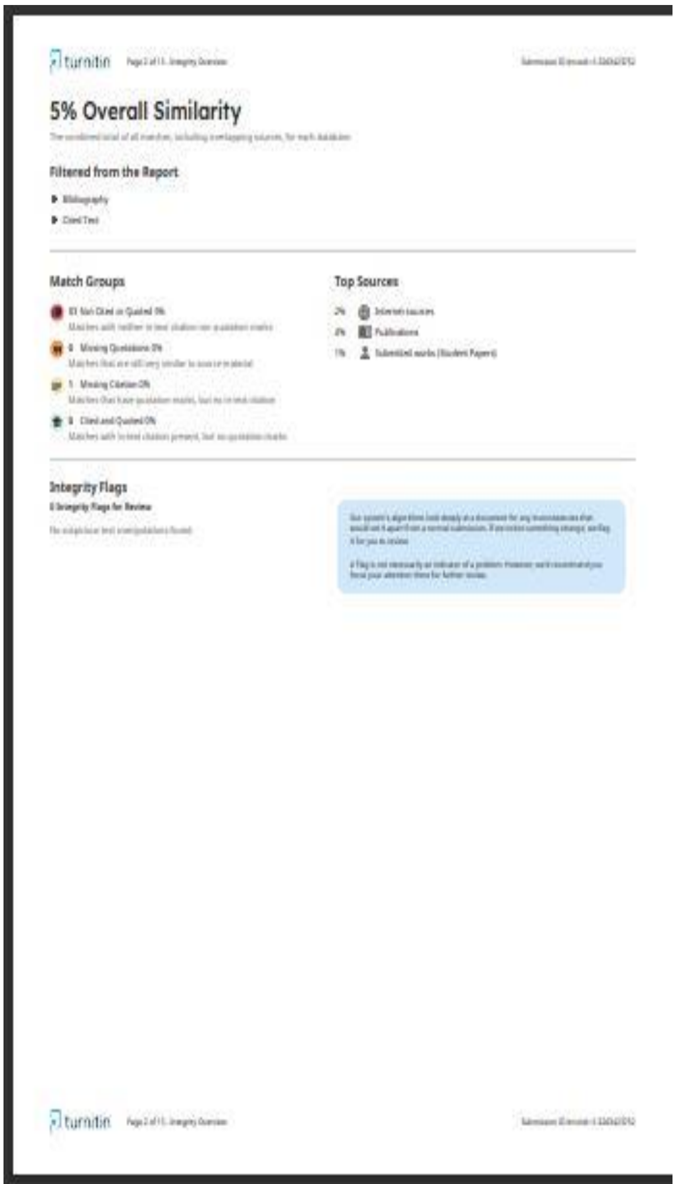
published in IJSREM Journal on Volume 09 Issue 05 May, 2025

Editor-in-Chief
IJSREM Journal

www.ijsrem.com

e-mail: editor@ijsrem.com

2. Similarity Index / Plagiarism Check report clearly showing the Percentage (%). No need for a page-wise explanation.



turnitin Page 2 of 15 - Integrity Overview Submission ID trn:oid:::1:[illegible]

5% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Filtered from the Report

- Bibliography
- Cited Text

Match Groups

- [illegible] Not Cited or Quoted 5%
  Matches with neither in-text citation nor quotation marks
- 0 Missing Quotations 0%
  Matches that are still very similar to source material
- 1 Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- 0 Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

- 2% Internet sources
- 4% Publications
- 1% Submitted works (Student Papers)

Integrity Flags

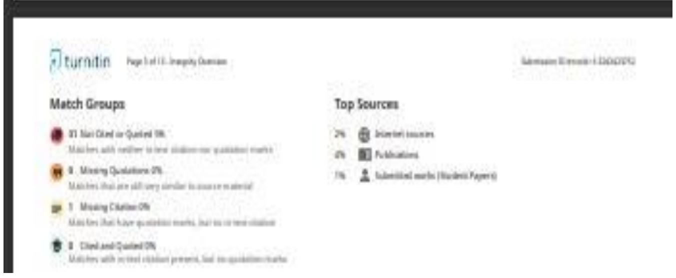
0 Integrity Flags for Review

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

turnitin Page 2 of 15 - Integrity Overview Submission ID trn:oid:::1:[illegible]



turnitin Page 3 of 15 - Integrity Overview Submission ID trn:oid:::1:[illegible]

Match Groups

- [illegible] Not Cited or Quoted 5%
  Matches with neither in-text citation nor quotation marks
- 0 Missing Quotations 0%
  Matches that are still very similar to source material
- 1 Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- 0 Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

- 2% Internet sources
- 4% Publications
- 1% Submitted works (Student Papers)

SUSTAINABLE DEVELOPMENT GOALS

SUSTAINABLE DEVELOPMENT GOALS
1 NO POVERTY
2 ZERO HUNGER
3 GOOD HEALTH AND WELL-BEING
4 QUALITY EDUCATION
5 GENDER EQUALITY
6 CLEAN WATER AND SANITATION
7 AFFORDABLE AND CLEAN ENERGY
8 DECENT WORK AND ECONOMIC GROWTH
9 INDUSTRY, INNOVATION AND INFRASTRUCTURE
10 REDUCED INEQUALITIES
11 SUSTAINABLE CITIES AND COMMUNITIES
12 RESPONSIBLE CONSUMPTION AND PRODUCTION
13 CLIMATE ACTION
14 LIFE BELOW WATER
15 LIFE ON LAND
16 PEACE, JUSTICE AND STRONG INSTITUTIONS
17 PARTNERSHIPS FOR THE GOALS
SUSTAINABLE DEVELOPMENT GOALS

SDG 9 – Industry, Innovation, and Infrastructure

Our project uses cutting-edge technologies like YOLO object detection, image analytics, and AI to solve real-world environmental problems. This directly supports SDG 9, which encourages innovation and the use of smart infrastructure. We built a system that is not only accurate but also affordable and easy to use, making it practical for real-world adoption. By automating tree enumeration, the project shows how tech can improve traditional forestry and environmental monitoring methods. It also opens the door for future innovations like species classification, forest health detection, and drone-based monitoring in both rural and urban areas.

SDG 11 – Sustainable Cities and Communities

Green spaces in cities play a key role in improving air quality, reducing heat, and enhancing quality of life. Our system helps city planners and local authorities monitor urban greenery such as roadside trees, parks, and public plantations. By using aerial or drone images, officials can get regular updates on tree counts in cities without sending people on the ground. This makes urban planning more data-driven and sustainable. Maintaining proper green cover also helps cities adapt to the effects of climate change, like rising temperatures and pollution, making urban spaces healthier and more resilient.

SDG 13 – Climate Action

Our project supports SDG 13 by helping monitor changes in forest cover, which directly impacts climate regulation. Trees absorb carbon dioxide and play a major role in reducing the effects of climate change. By using AI to automate tree counting, authorities and researchers can track deforestation or reforestation progress more quickly and accurately. This makes it easier to take timely action, implement conservation policies, or plan carbon offset programs. The system also helps generate reliable environmental data that can be used for climate-related research and policy-making, especially in fast-changing or vulnerable regions.

SDG 15 – Life on Land

This project directly contributes to the goal of protecting and restoring terrestrial ecosystems. Trees are critical for biodiversity, and our system helps in tracking their population over time. With this tool, forest departments can better assess whether tree cover is increasing or decreasing in certain regions. It also supports afforestation and reforestation projects by providing data to measure success. Since our tool reduces manual effort, more land can be covered and monitored regularly. This encourages sustainable forest management, reduces habitat loss, and supports broader biodiversity conservation efforts across both urban and rural areas.