A

MAJOR PROJECT REPORT

On

AUTONOMOUS LANDING SCENE RECOGNITION BASED ON TRANSFER LEARNING FOR DRONES

Submitted in partial fulfilment for the award of the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING (DATA SCIENCE)

By

D.PRATYUSHA	**: 21Q91A6768**
E.DEEPSHIKA	**: 21Q91A6777**
K.SAISANDEEP	**: 21Q91A6792**
A.HARISHANKAR	**: 22Q95A6712**

Under the guidance of

Mrs. T. NAGA PRAVEENA
Assistant Professor, Dept. of CSE-DS

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING (DATA SCIENCE)

MALLA REDDY COLLEGE OF ENGINEERING

(Approved by AICTE-Permanently Affiliated to JNTU-Hyderabad)
Programs Accredited by NBA (CSE,ECE), Recognized section 2(f) & 12(B) of UGC
New Delhi ISO 9001:2015 certified Institution
Maisammaguda, Dhulapally (Post via Kompally), Secunderabad-500100

2024 - 2025

MALLA REDDY COLLEGE OF ENGINEERING

(Approved by AICTE-Permanently Affiliated to JNTU-Hyderabad)

Programs Accredited by NBA(CSE,ECE), Recognized section 2(f) & 12(B) of UGC New Delhi ISO 9001:2015 certified Institution

Maisammaguda, Dhulapally (Post via Kompally), Secunderabad- 500100

DEPARTMENT OF COMPUTER SCIENCE &ENGINEERING – (DATA SCIENCE)

CERTIFICATE

This is certify that the Major Project report on "**AUTONOMOUS LANDING SCENE RECOGNITION BASED ON TARNSFER LEARNING FOR DRONES**" is successfully done by the following students of Department of Computer Science & Engineering (Data Science) of our college in partial fulfilment of the requirement for the award of Bachelor of Technology degree in the year 2024-2025. The results embodied in this report have not been submitted to any other University for the award of a degree.

D.PRATYUSHA	**:**	**21Q91A6768**
E.DEEPSHIKA	**:**	**21Q91A6777**
K.SAISANDEEP	**:**	**21Q91A6792**
A.HARISHANKAR	**:**	**22Q95A6712**

Submitted for the viva voice examination held on: ________________________________

INTERNAL GUIDE	**PROJECT COORDINATOR**	**HOD**
Mrs. T. NAGA PRAVEENA	**Dr. SATEESH NAGAVARAPU**	**Dr.J.G.M BRITTO**
Asst. Professor	**Associate Professor**	**Professor & Head**

Internal Examiner **External Examiner**

DECLARATION

We, **Pratyusha daurkar, Deepshika endel, Saisandeep, Harishankar** with Regd.no. **21Q91A6768, 21Q91A6777, 21Q91A6792, 22Q95A6712** are here by declaring that the major project entitled "**AUTONOMOUS LANDING SCENE RECOGNITION BASED ON TRANSFER LEARNING FOR DRONES**"" has done by us under the guidance of **Mrs. T.NAGA PRAVEENA** Assistant Professor, Department of Computer Science and Engineering (Data science) is submitted in the partial fulfilment of the requirements for the award of degree of BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE and ENGINEERING (DATA SCIENCE). The Result embedded in this project report have not been submitted to any other University or institute for the award of any degree.

Signature of the Candidates

D.PRATYUSHA	**:**	**21Q91A6768**
E.DEEPSHIKA	**:**	**21Q91A6777**
K. SAI SANDEEP	**:**	**21Q91A6792**
A.HARISHANKAR	**:**	**22Q95A6712**

DATE:

PLACE: Maisammaguda

ACKNOWLEDGEMENT

First and foremost, we would like to express our immense gratitude towards our institution Malla Reddy College of Engineering, which helped us to attain profound technical skills in the field of Computer Science & Engineering, there by fulfilling our most cherished goal.

We are pleased to thank **Sri Ch. Malla Reddy**, our Founder Chairman **MRGI**, **Sri Ch. Mahender Reddy**, Secretary, **MRGI** for providing this opportunity and support throughout the course.

It gives us immense pleasure to acknowledge the perennial inspiration of **Dr. M. Ashok** our beloved principal for his kind co-operation and encouragement in bringing out this task.

We extend our sincere gratitude to **Dr. J.G.M. Britto,** Head of the Department (HOD), CSE (Data Science), for his unwavering support, insightful guidance, and constructive feedback. His inspiration and expertise have been pivotal to the successful completion of our degree, and we deeply appreciate his encouragement throughout our academic journey.

We would like to thank **Dr. Sateesh Nagavarapu,** Associate Professor, is our project coordinator CSE(DS) Department for their inspiration adroit guidance and constructive criticism for successful completion of our degree.

We would like to thank **Mrs. T Naga Praveena,** Assistant Professor, our internal guide, for her valuable suggestions and guidance during the exhibition and completion of this project.

Finally, we avail this opportunity to express our deep gratitude to all staff who have contribute their valuable assistance and support making our project successful.

D.PRATYUSHA	**:**	**21Q91A6768**
E.DEEPSHIKA	**:**	**21Q91A6777**
K.SAISANDEEP	**:**	**21Q91A6792**
A.HARISHANKAR	**:**	**22Q95A6712**

CONTENTS **page no**

ABSTRACT

In this paper, we study autonomous landing scene recognition with knowledge transfer for drones. Considering the difficulties in aerial remote sensing, especially that some scenes are extremely similar, or the same scene has different representations in different altitudes, we employ a deep convolutional neural network (CNN) based on knowledge transfer and fine- tuning to solve the problem. Then, LandingScenes-7 dataset is established and divided into seven classes. Moreover, there is still a novelty detection problem in the classifier, and we address this by excluding other landing scenes using the approach of thresholding in the prediction stage. We employ the transfer learning method based on ResNeXt-50 backbone with the adaptive momentum (ADAM) optimization algorithm. The method presented in this study can be easily extended to larger, more complex datasets, improving its scalability and generalization to various landing environments. In real-world scenarios, class imbalance could pose a challenge, but techniques like class weighting or data augmentation could be employed to ensure better performance across all scene types. To enable real-time deployment, optimizing the model for faster inference is crucial, especially for edge devices with limited computational resources. Moreover, integrating hybrid feature extraction methods that combine traditional computer vision techniques with deep learning could enhance the model's robustness in adverse conditions. Cross-altitude scene representation, where the model learns to recognize landing scenes from various heights, could be explored further, improving the drone’s adaptability to different perspectives. We also compare ResNet-50 backbone and the momentum stochastic gradient descent (SGD) optimizer. Experiment results show that ResNeXt-50 based on the ADAM optimization algorithm has better performance. With a pre-trained model and fine- tuning, it can achieve 97.845 0% top 1 accuracy on the LandingScenes-7 dataset, paving the way for drones to autonomously learn landing scenes.

KEYWORDS:

Drone autonomous landing, Scene Detection, Aerial Remote Sensing, Convolution Neural Networks (CNN), Knowledge Transfer, Novelty Detection, ResNeXt-50, Fine Tuning, Stochastic gradient descent (SGD).

LIST OF FIGURES

LIST ACRONYMS AND ABBREVIATIONS

S.No	**Acronym**	**Description**
1	ADAM	Adaptive Moment Estimation
2	CNN	Convolutional Neural Networks
3	UML	Unified Modeling Language
4	RESNET	Residual Networks
5	DNN	Deep neural networks
6	RESNEXT	Residual Networks with External Transformation
7	SGD	Stochastic Gradient Descent

LIST OF SCREENS

CHAPTER 1

INTRODUCTION

1. INTRODUCTION

Remote sensing can be divided into ground remote sensing, aerial remote sensing, and aerospace remote sensing according to different working platforms. For aerial remote sensing, sensors are mounted on aircraft such as balloons, drones, and helicopters. It has become possible for drones to recognize the landing scenes with the development and application of convolutional neural network (CNN) and graphic processing unit (GPU). Huge progress has been made in object detection in aerial images and scene classification in remote sensing images stems from the rise of CNN and the public datasets recently, such as geospatial object detection, scene classification of satellite images, and aerial scene classification. Additionally, the integration of multispectral and hyperspectral imagery has gained attention for improving scene recognition accuracy, particularly in varying environmental conditions such as diverse lighting, weather, and topography. The combination of these advanced imaging techniques with deep learning models allows drones to perform more accurate scene segmentation, facilitating better decision-making during autonomous landings.

Anand et al. applied deep learning and a label "H" to achieve automatic landing for the drone. However, in emergency landing scenarios, the landing mark may not be arranged in advance. Tian et al. employed the Inception V3 model for landing scene recognition on the ImageNet dataset and proposed a learning rate decay method with computational verb theory to improve the recognition accuracy. Lu et al. proposed an algorithm for augmenting the l-channel to the traditional RGB images. Experimental results show the effectiveness of the l-channel, and the performance is significantly improved in scene classification and object recognition tasks. Scene recognition based on CNN is inseparable from the support of datasets, such as Places365 dataset, dataset Scene Understanding (SUN) and SUN attribute dataset. Yang et al. Proposed a simultaneous localization and mapping (SLAM) method based on a monocular vision to realize autonomous landing in an emergency and unknown environment for drones. This fusion can significantly enhance the drone's ability to navigate complex, dynamic conditions, such as sudden obstacles or unexpected terrain changes. The continuous evolution of transfer learning techniques also plays a crucial role in this domain, enabling models pre- trained on large, general datasets to be fine-tuned for specific landing tasks, thus improving both efficiency and accuracy in deployment scenarios. With the rapid advancement of unmanned aerial vehicles (UAVs), or drones, autonomous navigation and landing have become crucial components in various applications such as surveillance, disaster response, package delivery, and environmental monitoring.

A key challenge in this domain is enabling drones to accurately recognize and assess potential landing zones under diverse and dynamic conditions. This capability, known as landing scene recognition, is vital for ensuring safety, efficiency, and operational autonomy.
Remote sensing is generally categorized into ground, aerial, and aerospace platforms, with aerial remote sensing relying on aircraft such as drones, balloons, and helicopters. Among these, drones have emerged as a highly flexible and cost-effective tool for remote sensing, largely due to their ability to operate at low altitudes and capture high-resolution imagery.
The integration of convolutional neural networks (CNNs) and graphics processing units (GPUs) has significantly enhanced the capabilities of drones in recognizing complex scenes from aerial images. The proliferation of publicly available remote sensing datasets, such as Places365, Scene Understanding (SUN), and the SUN Attribute Dataset, has further accelerated progress in this area. These datasets have enabled deep learning models to achieve impressive performance in both **object detection* and *scene classification* tasks.
However, in real-world or emergency scenarios, such markers are often unavailable. Addressing this limitation, Tian et al. utilized the Inception V3 model pre-trained on the ImageNet dataset and introduced a learning rate decay strategy based on computational verb theory to enhance classification accuracy. Lu et al. advanced the field by incorporating an l-channel (lightness channel from LAB color space) into traditional RGB inputs, significantly boosting scene classification performance.
A transformative development in this field is the application of transfer learning technique where deep learning models trained on large, generic datasets are fine-tuned for specific tasks like landing scene recognition. In this project, Res Net (Residual Network) is used as the backbone architecture due to its superior ability to mitigate vanishing gradient problems and maintain high accuracy in deep neural networks. By leveraging a pre-trained Res Net model, the system benefits from robust feature extraction capabilities, enabling accurate classification of complex aerial scenes with limited training data. This approach significantly enhances both the efficiency and accuracy of landing scene recognition, making it well-suited for real-time deployment in autonomous drone operations.
This project aims to develop a landing scene recognition system for drones based on transfer learning using the Res Net architecture, allowing drones to autonomously identify safe landing zones in both planned and emergency scenarios. The ultimate goal is to enhance the reliability and intelligence of autonomous drone missions under real-world environmental constraints.
Furthermore, the continuous evolution of transfer learning techniques has played a transformative role in the field of aerial remote sensing. Transfer learning allows deep learning models pre-trained on large, generic datasets to be fine-tuned for specific tasks such as landing site recognition.

The integration of transfer learning into deep learning workflows has significantly transformed the landscape of intelligent aerial remote sensing. This approach not only reduces computational cost and training time but also enhances model performance, particularly when applied to niche domains with limited labeled data. By leveraging pre-trained models on large-scale datasets and fine-tuning them for specific tasks, researchers can develop high-performing systems without the need for massive annotated datasets or extensive computational resources. This has proven especially beneficial in scenarios that demand rapid model deployment and swift adaptation to new or changing environments, such as emergency response situations or dynamic terrain mapping.

The fusion of cutting-edge deep learning algorithms with diverse imaging modalities—such as RGB, thermal, LiDAR, and multispectral imaging—has enabled drones to perform increasingly complex and autonomous operations. These include real-time scene recognition, semantic segmentation, obstacle detection and avoidance, and precise autonomous landing, even in challenging or unpredictable environments. Such capabilities are critical for both routine and high-stakes applications, where precision, reliability, and speed are paramount.

Advancements in computational infrastructure, including the use of edge AI and cloud-based processing, further support these developments by allowing real-time data analysis and decision- making onboard drones. This not only improves responsiveness but also minimizes the need for constant human intervention. Consequently, drones are evolving from basic aerial data collectors into intelligent agents capable of interpreting and reacting to their surroundings in real time.

The potential applications of these technologies are vast and impactful. In disaster response, drones can swiftly survey affected areas, identify survivors, and map damage zones, even in places inaccessible to humans. In environmental monitoring, they can track deforestation, wildlife patterns, and pollution levels with high accuracy. In smart agriculture, drones equipped with deep learning models can assess crop health, optimize irrigation, and detect pest infestations, thereby increasing yield and sustainability. Urban infrastructure management also benefits, as drones can inspect bridges, roads, and buildings, helping to predict maintenance needs and prevent failures.

As research in this field continues to advance, the boundaries of what drones can achieve will expand further. The integration of AI, sensor technology, and efficient learning models promises a future where autonomous aerial systems play a central role in addressing real-world challenges across a range of domains, making operations more efficient, safe, and intelligent than ever before.

CHAPTER 2

LITERATURE SURVEY

2. LITERATURE SURVEY

1. Title: "Enhancing UAV Safety through Novelty Detection in Landing Scene Recognition"

Author: M. Kumar

Year : 2023

Description:

Implemented confidence thresholding and Bayesian uncertainty to detect out-of-distribution samples. Improved safety by rejecting uncertain classifications for manual intervention.

2. Title: "Few Shot Learning for Autonomous UAV Landing Scene Recognition"

Author: A. Patel

Year: 2023

Description:

Proposed a few-shot learning approach using Prototypical Networks to enable drones to recognize landing zones with minimal labelled data.

Leveraged pre-trained models and fine-tuned them with few-shot datasets, allowing for quick adaptation to new environments.

3. Title: " Lightweight Models for Real-Time Deployment"

Author: J. Lee

Year: 2022

Description:

Developed lightweight models like Mobile Net for real-time operations. Achieved comparable accuracy to heavier models with reduced computational requirements. Knowledge Transfer via Domain Adaptation.

Zhong and team analyzed passenger flow patterns by employing clustering and prediction models. Their focus was on discovering underrepresented patterns within imbalanced transitdatasets.

4.Title: " Domain Adaptation Techniques for Drone Landing Scene Recognition "

Author: Y. Zhao

Year: 2021

Description: Proposed unsupervised domain adaptation to bridge the gap between synthetic and real-world datasets.

Employed Generative Adversarial Networks (GANs) to generate synthetic images for training.

5.Title: "Leveraging Transfer Learning for Drone-based Scene Classification"

Author: H. Singh

Year: 2020

Explored fine-tuning of pre-trained ImageNet models for drone-specific datasets. Demonstrated improved accuracy with reduced training time compared to training models from scratch.

6.Title: "Drone Vision: Safe Landing-Zone Detection Using Convolutional Neural Networks"

Author: K. Choi

Year: 2019 **Description:**

Utilized pre-trained CNNs like Res Net and VGG for scene recognition.

Focused on RGB images to classify landing zones into categories like urban, water, and open fields. Highlighted limitations in recognizing novel environments and weather conditions.

CHAPTER 3

PROBLEM DEFINITION

3. PROBLEM DEFINITION

PROJECT DESCRIPTION

Autonomous drones require accurate and reliable recognition of safe landing zones in diverse and dynamic environments. Traditional landing systems rely on pre-programmed locations or GPS data, which can be inaccurate or unavailable in GPS-denied environments (e.g., urban canyons, forests, or disaster zones). Real time scene recognition using visual data can significantly enhance a drone's ability to autonomously identify and select suitable landing sites.
However, training deep learning models from scratch to perform scene recognition requires extensive labeled data and computational resources. In many cases, such datasets may not be available or may lack diversity across environments. Transfer learning offers a promising solution by leveraging knowledge from large, pre-trained models to adapt to the drone landing domain with minimal data and training.
This project addresses the challenge of developing a reliable autonomous landing scene recognition system for drones using transfer learning techniques. The goal is to classify and recognize different types of landing scenes (e.g., clear ground, rooftops, roads, water bodies, or hazardous zones) in real- time using onboard cameras and a pre-trained neural network adapted to the drone's environment.

3.1 EXISTING SYSTEM

Existing systems for autonomous drone landing typically rely on a combination of computer vision, machine learning, and sensor technologies to identify and navigate to landing zones. These systems often use Convolutional Neural Networks (CNNs) trained on large datasets to recognize landing areas in various environments, such as flat surfaces, rooftops, or even moving platforms. Some systems incorporate sensor fusion, combining visual data from cameras with data from sensors like LiDAR, infrared, and sonar to improve accuracy and robustness. However, these systems still face challenges like environmental variability (e.g., changing weather, lighting), real-time processing demands, and the need for high precision in complex, dynamic terrains. While some approaches use transfer learning to enhance performance with smaller, domain-specific datasets, many systems require further refinement to handle diverse and unpredictable landing scenarios autonomously.

Disadvantages:

Environmental Sensitivity: Many systems struggle with environmental variability, such as changes in lighting, weather conditions (e.g., rain, fog, or strong winds), and time of day, which can significantly affect the accuracy of landing zone detection and make the system unreliable in the certain conditions.

1. **Limited Generalization:** Pre-trained models often have difficulty adapting to new, unseen environments without additional fine-tuning. Transfer learning can mitigate this to some extent, but systems may still struggle with recognizing landing zones in highly dynamic or unfamiliar terrains.

2. **Real-Time Processing Constraints:** Autonomous landing requires fast, real-time processing of sensor data to make decisions during descent. Many existing systems are not optimized for the processing power and speed needed for real-time inference, especially when high-resolution imaging or sensor fusion is involved.

4. **Data Dependency:** High reliance on large, high-quality datasets makes it hard to generalize in uncommon or poorly represented environments. This data limitation can lead to poor performance in underrepresented or edge-case scenarios, such as rural or emergency landing sites.

3.2 PROPOSED SYSTEM

The proposed system for autonomous drone landing integrates transfer learning with advanced sensor fusion techniques to enhance landing zone recognition and navigation in dynamic environments. By fine-tuning pre-trained deep learning models (such as CNNs) on domain-specific datasets, the system can quickly adapt to various terrains, lighting conditions, and obstacles, reducing the need for extensive training. The system combines visual data from cameras with inputs from additional sensors like LiDAR, infrared, and sonar to improve accuracy, obstacle avoidance, and real-time decision-making. With optimized processing algorithms for faster inference, the proposed system aims to provide reliable, safe, and efficient autonomous landings even in complex and unpredictable environments, addressing the limitations of current systems and ensuring robust performance across diverse applications.

The use of Convolutional Neural Networks (CNNs) allows for robust feature extraction from visual inputs. Transfer learning allows these CNNs, already trained on large-scale datasets like ImageNet or Places365, to be repurposed for drone-specific tasks such as identifying flat surfaces, avoiding obstacles, and handling variable lighting or textures in landing zones.

In parallel, the system integrates sensor fusion by combining inputs from:

- RGB cameras (visual data)
- LiDAR (depth perception and surface mapping)
- Infrared sensors (thermal and low-light recognition)
- Ultrasonic/sonar sensors (precise ground distance detection) This multi-modal sensing ensures that the drone can assess both the visual context and spatial environment accurately, even under challenging conditions like fog, glare, shadows, or debris.

ADVANTAGES OF PROPOSED SYSTEM:

- **Improved Accuracy and Reliability:** By integrating transfer learning and sensor fusion, the system enhances landing zone detection and navigation accuracy. Visual data from cameras is combined with LiDAR, infrared, and sonar sensors, enabling the drone to effectively recognize landing zones in diverse and complex environments.
- **Adaptability to Various Environments:** Transfer learning allows the model to adapt to new terrains, weather conditions, and lighting variations with minimal additional training.
- This reduces the dependency on large, labeled datasets and ensures better performance across a range of real-world scenarios.
- **Faster Real-Time Processing:** The proposed system optimizes deep learning models for real-time performance, ensuring fast inference and decision-making during the drone's descent. This is crucial for maintaining safe and efficient autonomous landings, even in dynamic conditions.
- **Enhanced Obstacle Avoidance:** The integration of multiple senior inputs helps the system detect and avoid sudden obstacles during landing, increasing safety in unpredictable environment.
- **Scalability and Flexibility:** The modular design allows easy integration with different drone platforms and hardware configurations, making it suitable for various commercial and industrial applications.
- **Reduced Training Time and Cost:** Transfer learning minimizes the need for training from scratch, reducing computational costs and time while still achieving high performance on specialized tasks.

3.3 REQUIREMENTS

A software requirements specification (SRS) is a description of a software system to be developed, its defined after business requirements specification (CONOPS) also called stakeholder requirements specification (STRS) other document related is the system requirements specification (SYRS).

3.3.1 HARDWARE REQUIREMENTS:

The hardware requirements may serve as the basis for a contract for the implementation of the system and should therefore be a complete and consistent specification of the whole system.
They are used by software engineers as the starting point for the system design. It shows what the System does and not how it should be completed.

• Processor	:	Pentium –IV
• RAM	:	8 GB (min)
• Hard Disk	:	20 GB
• Keyboard	:	Standard Windows Keyboard
• Mouse	:	Two or Three Button Mouse
• Monitor	:	SVGA

3.3.2 SOFTWARE REQUIREMENTS:

The software requirement document is the specification of the system. It should include both a definition and a specification of requirements. It is a set of what the system should do rather than how it should do it. The software requirements provide a basis for creating the software requirements specification. It is useful in estimating cost, planning team activities, performing tasks and tracking the teams and tracking the team's progress throughout the development activity.

• Operating system	:	Windows 7 Ultimate.
• Coding Language	:	Python.
• Front-End	:	Python.
• Back-End	:	Tensorflow
• Development tool	:	Jupyter Notebook
• Data Base	:	MySQL

CHAPTER 4

SYSTEM STUDY

4. SYSTEM STUDY

4.1 FEASIBILITY STUDY

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

4.2 ECONOMICAL FEASIBILITY

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

4.3 TECHNICAL FEASIBILITY

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resourc. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

4.4 SOCIAL FEASIBILITY

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

CHAPTER 5

SYSTEM DESIGN

5. SYSTEM DESIGN

DESCRIPTION

The system is designed to enable an autonomous drone to identify safe landing zones in real time by analysing aerial images. A deep learning model, built using transfer learning, processes the live camera feed to classify potential landing scenes (e.g., open ground, rooftop, road, water, obstacles). The decision to land or not is based on the classification output and optional sensor data.

Unmanned aerial vehicles (UAVs), commonly known as drones, are increasingly being used in a variety of applications such as surveillance, disaster response, agriculture, and package delivery. A critical aspect of drone autonomy is the ability to identify and execute safe landings in unstructured and unknown environments—especially in emergency scenarios or GPS-denied zones.

This project focuses on developing a robust landing scene recognition system that allows a drone to analyze its surroundings and determine whether a location is suitable for landing. To achieve this, we leverage transfer learning, a machine learning technique that uses pre-trained models on large datasets (like ImageNet) and fine-tunes them on smaller, task-specific datasets (e.g., aerial scenes from drones).

A camera mounted on the drone captures real-time video or images, which are fed into a lightweight Convolutional Neural Network (CNN) model running on an onboard processor (such as an NVIDIA Jetson Nano or Xavier). The system classifies the observed scene into categories such as:

- Safe Landing Zones (e.g., open grass, flat rooftops)
- Unsafe or Hazardous Zones (e.g., water, trees, crowds, uneven terrain)

Based on the classification results and confidence scores, along with optional sensor data (e.g., altitude, IMU, proximity sensors), the drone makes an intelligent decision to initiate or abort landing.

5.1 SYSTEM ARCHITECTURE

A system architecture is the conceptual model that defines the structure. behaviors, and more views of a system. An architecture description is a formal description and representation of a system, organized in a way that supports reasoning about the structures and behaviors of the system.

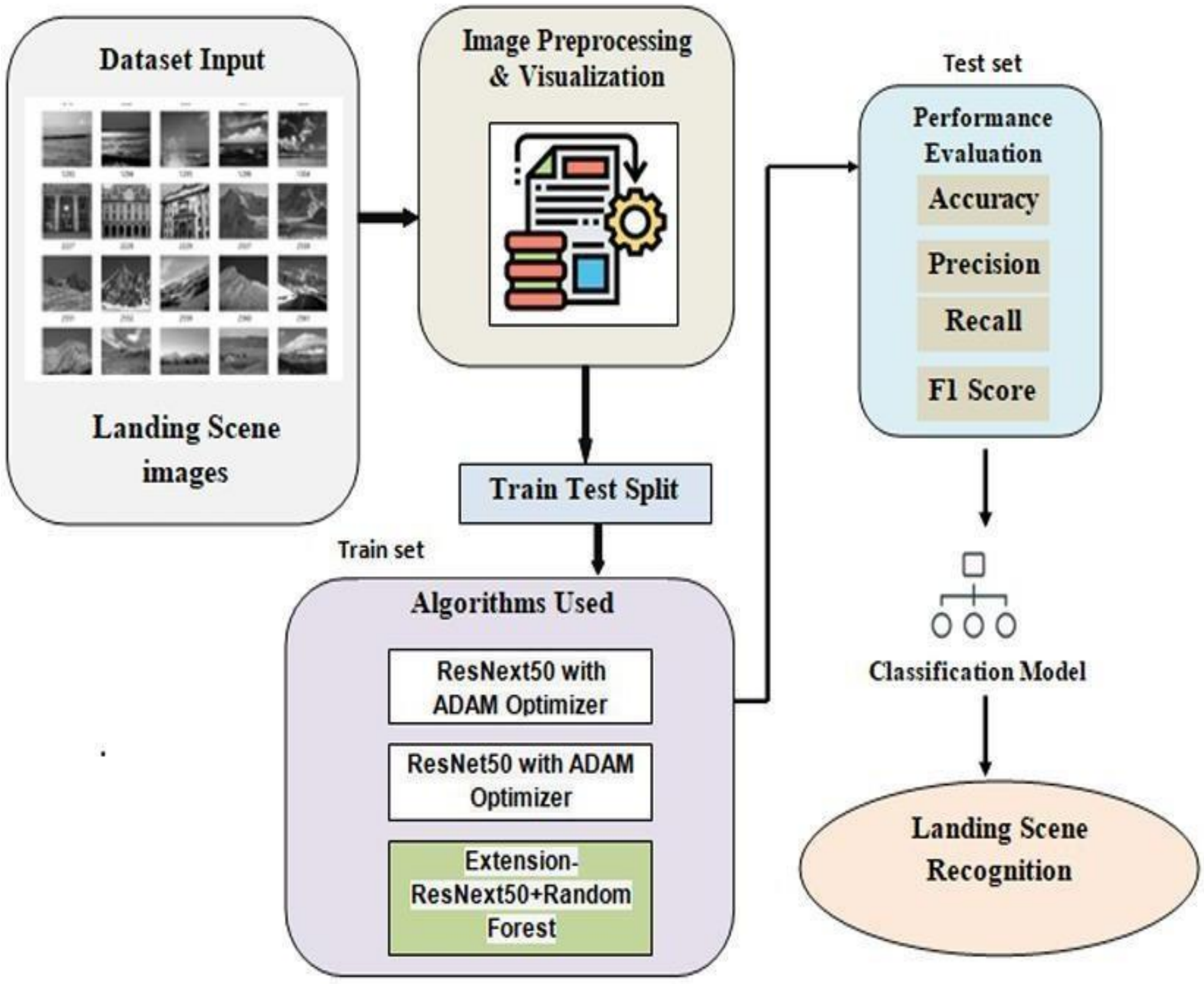


Fig. 5.1.0: System Architecture

Explanation:

The proposed system processes landing scene images through preprocessing and splits the dataset for training and testing. It uses ResNet50 with Adam, ResNeXt50 with Adam, and a hybrid ResNeXt50 with Random Forest for classification. Models are evaluated using accuracy, precision, recall, and F1-score. The best-performing model is selected for real-world drone landing scene recognition.

5.2 UML DIAGRAMS

UML (Unified Modeling Language) is a standard language for specifying, visualizing, constructing and documentig the artifacts of software systems.

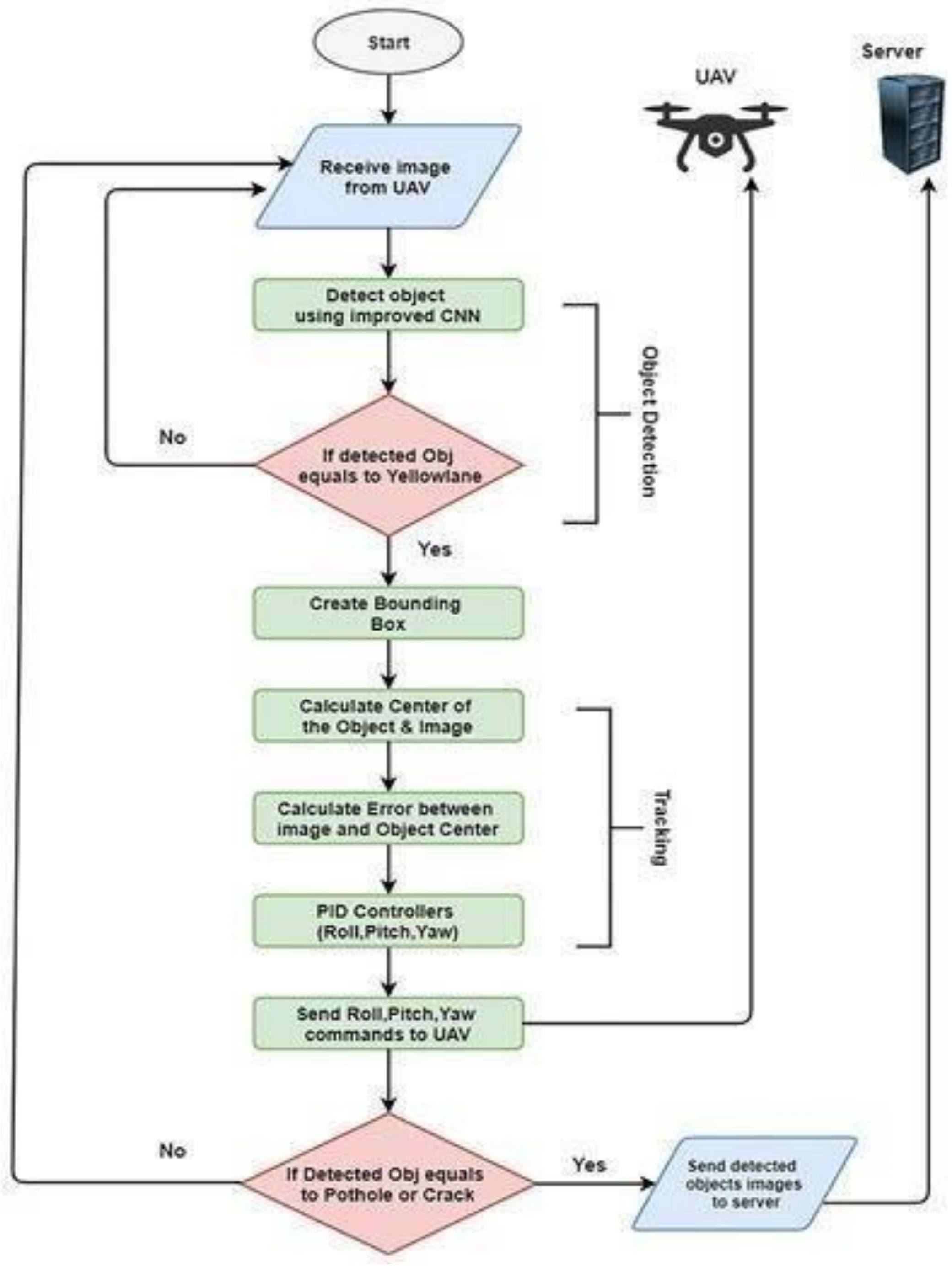

Start
Receive image from UAV
Detect object using improved CNN
If detected Obj equals to Yellowlane
No
Yes
Object Detection
Create Bounding Box
Calculate Center of the Object & Image
Calculate Error between image and Object Center
PID Controllers (Roll,Pitch,Yaw)
Tracking
Send Roll,Pitch,Yaw commands to UAV
If Detected Obj equals to Pothole or Crack
No
Yes
Send detected objects images to server
UAV
Server

Fig. 5.2: Uml diagram

5.2.1 CLASS DIAGRAM:

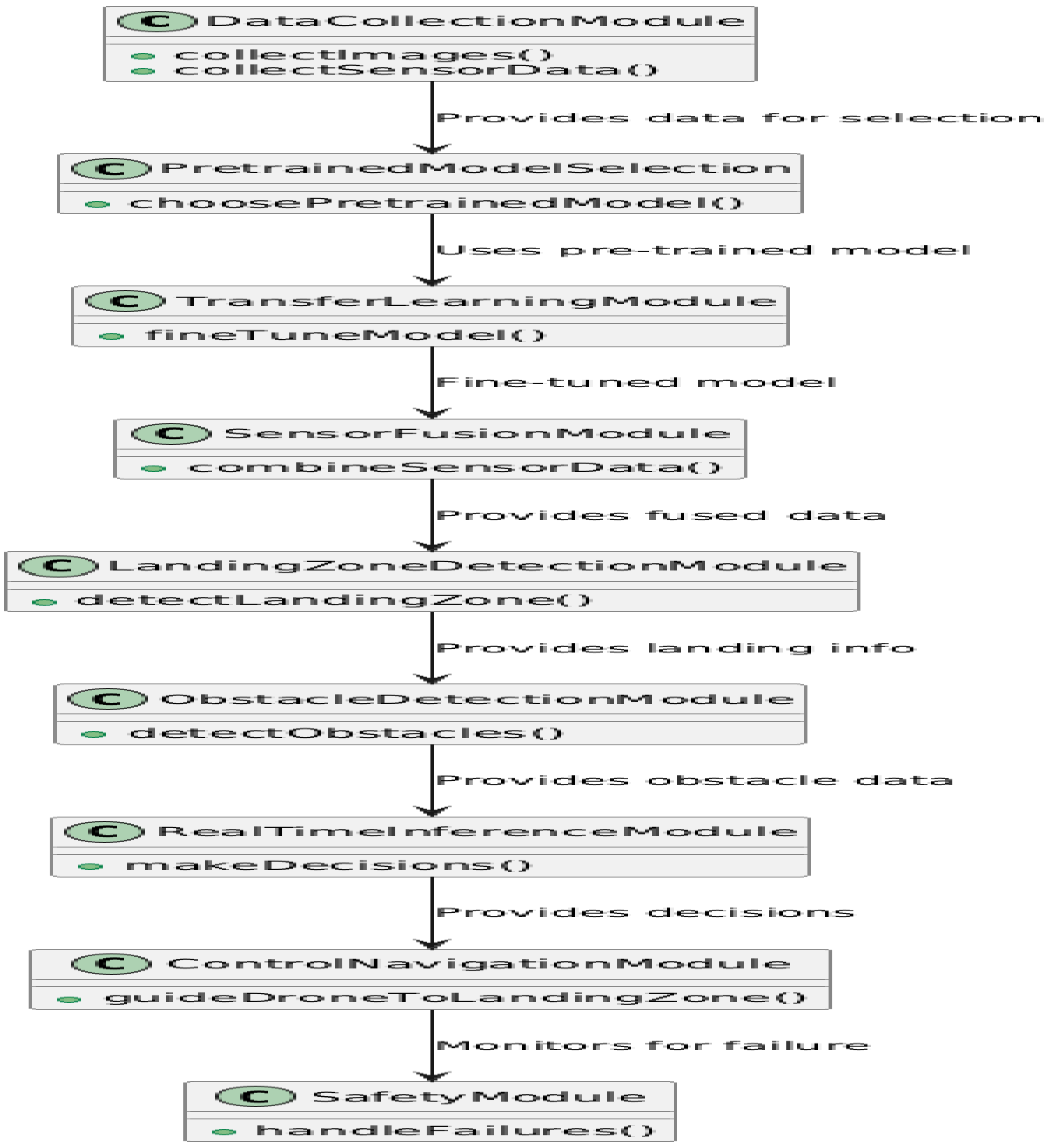


Fig. 5.2.1: Class Diagram

Explanation:

In this class diagram, it represents how the classes with attributes and methods are linked together to perform the verification with security. From the above diagram shown the various classes involved in our project.

5.2.2 USE CASE DIAGRAM:

A use case diagram in the Unified Modeling Language (UML) is a type of behavioural diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted

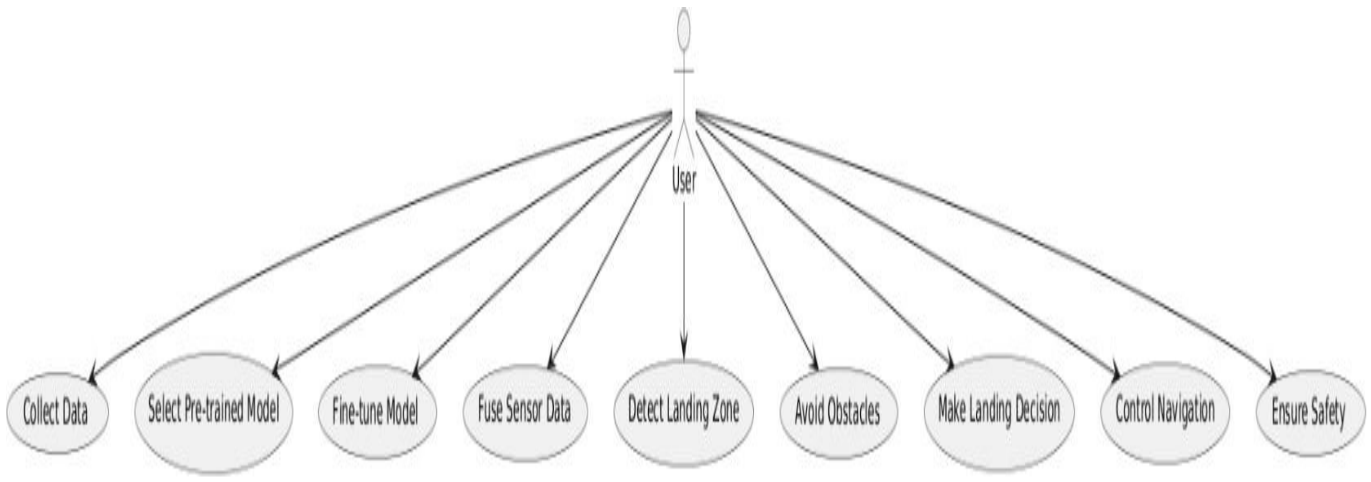


Fig. 5.2.2 Use case Diagram

Explanation:

The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

5.2.3 SEQUENCE DIAGRAM:

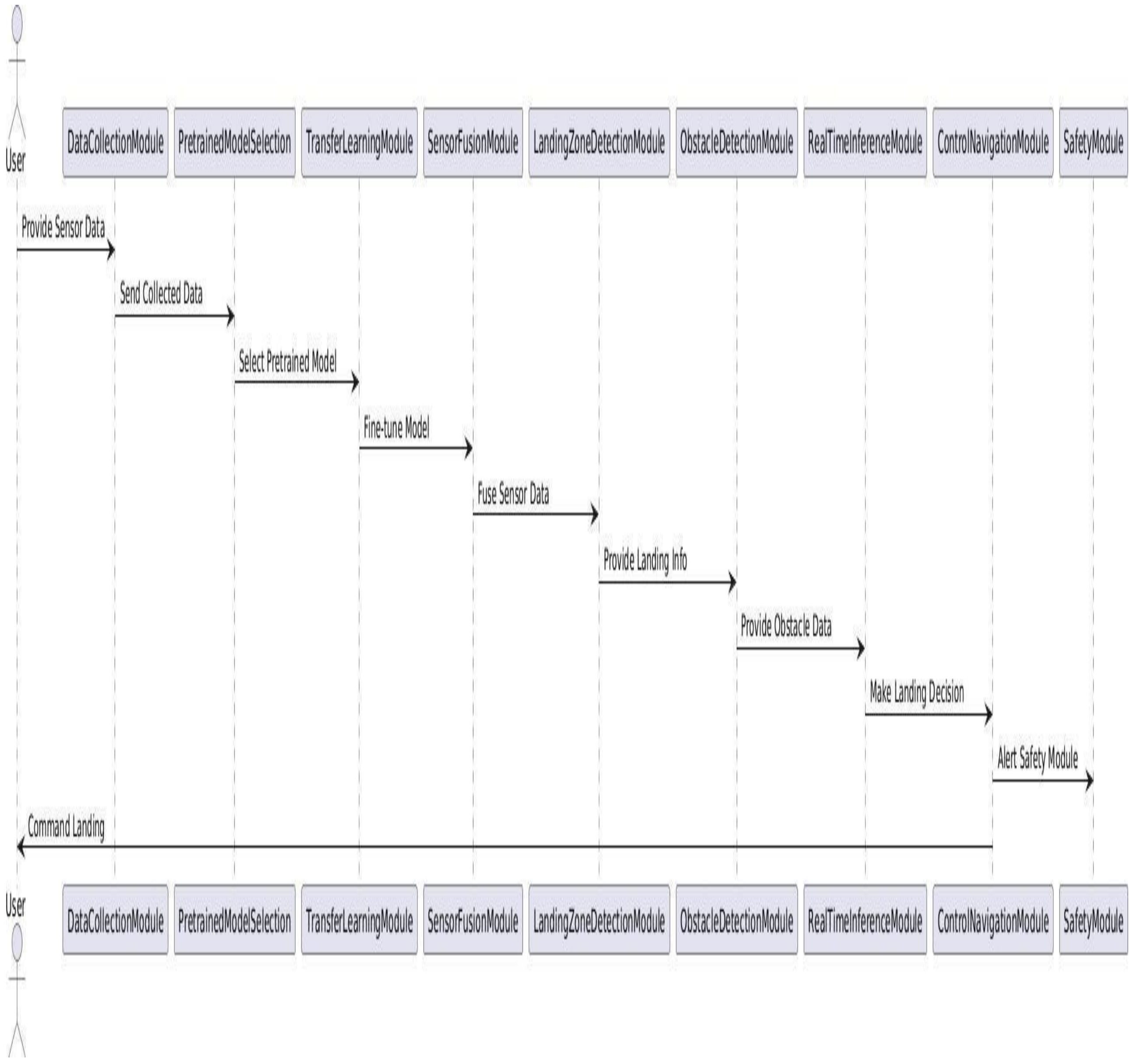


Fig. 5.2.3: Sequence Diagram

5.2.4 COMPONENT DIAGRAM:

Components are wired together by using an assembly connector to connect the required interface of one component with the provided interface of another component.

This illustrates the service consumer - service provider relationship between the two components**.**

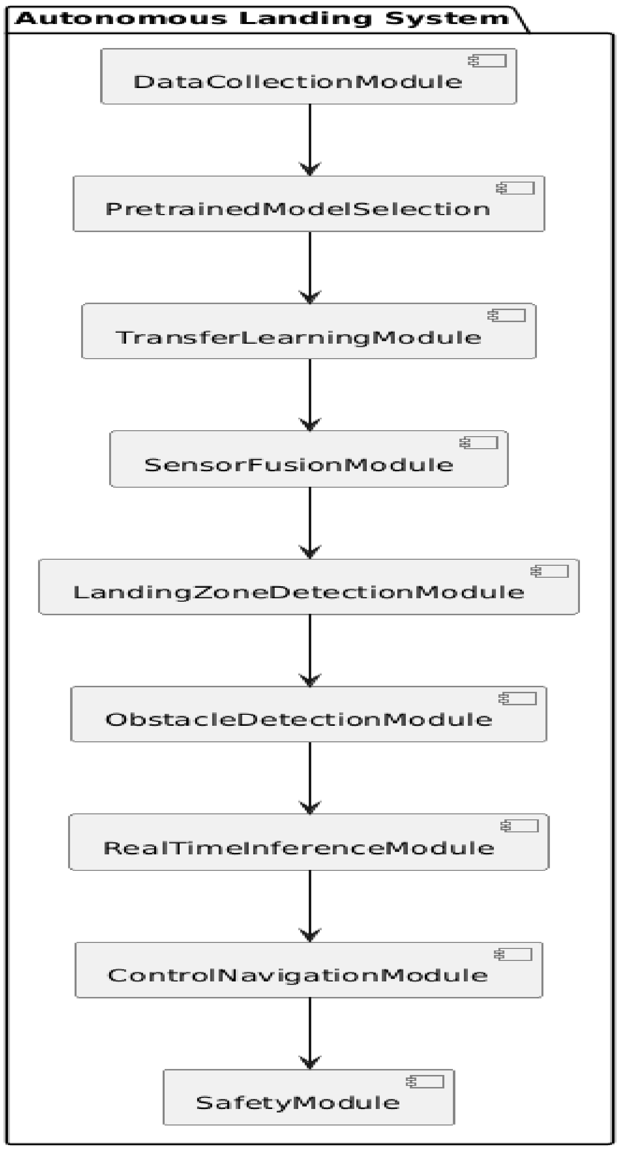


Fig. 5.2.4: Component Diagram

Explanation:

In the above digram tells about the components in your diagram work together to convert raw image data into meaningful information or insights that can be used for various applications, such as object recognition, scene understanding, or decision-making system.

5.2.5 ACTIVITY DIAGRAM:

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

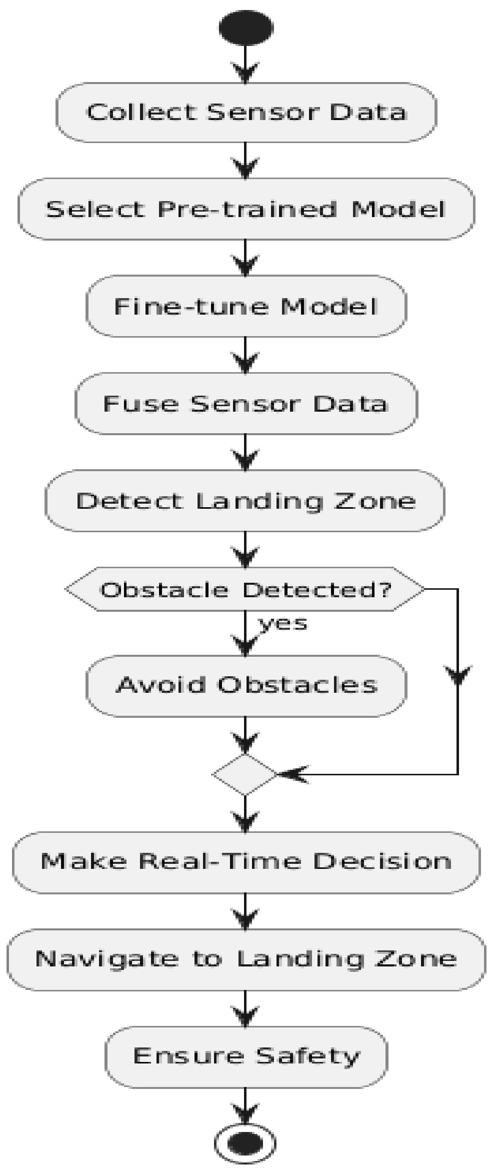


Fig. 5.2.5:Activity Diagram

5.2.7 DEPLOYMENT DIAGRAM:

A deployment diagram in the Unified Modeling Language models the *physical* deployment of artifacts on nodes. To describe a web site, for example, a deployment diagram would show what hardware components ("nodes") exist (e.g., a web server, an application server, and a database server), what software components ("artifacts") run on each node (e.g., web application, database), and how the different pieces are connected (e.g. JDBC, REST, RMI).

The nodes appear as boxes, and the artifacts allocated to each node appear as rectangles within the boxes. Nodes may have sub nodes, which appear as nested boxes. A single node in a deployment diagram may conceptually represent multiple physical nodes, such as a cluster of database servers.

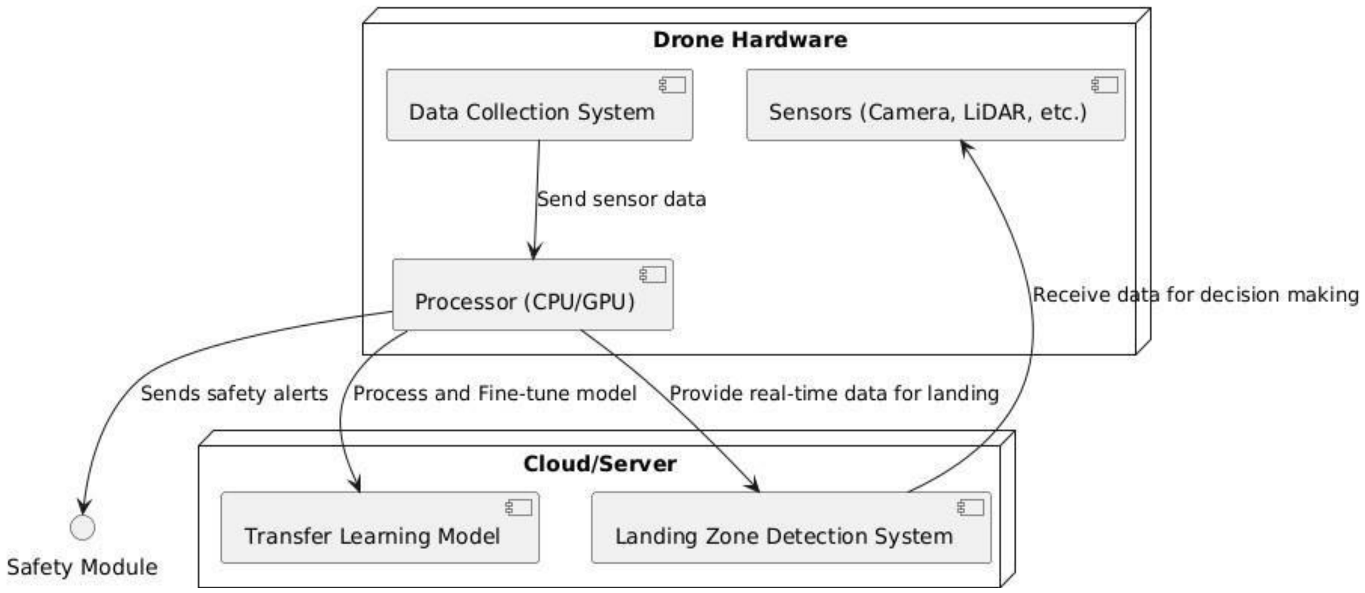

Drone Hardware
Data Collection System
Sensors (Camera, LiDAR, etc.)
Send sensor data
Processor (CPU/GPU)
Receive data for decision making
Sends safety alerts
Process and Fine-tune model
Provide real-time data for landing
Cloud/Server
Safety Module
Transfer Learning Model
Landing Zone Detection System

Fig. 5.2.6: Deployment Diagram

Explanation:

In the above diagram tells about the valuable insights and challenges related to system performance, accuracy, and scalability. It provides a comprehensive view of the system architecture, including data input sources, pre-processing steps, object detection and tracking models, post-processing steps, extraction, transformation and tracking interfaces. This information is essential for understanding the system's overall functionality and addressing potential issue.

CHAPTER 6

SYSTEM IMPLEMENTATION

6. SYSTEM IMPLEMENTATION

An Implementation is a realization of a technical specification or algorithm as a program, software components, or other computer system though computer programming and deployment. Many implementations may exist for specifications or standards. A special case occurs in object- oriented programming, when a concrete class implements an interface. The virtual mouse will be able to replicate the functions of a physical mouse, moving the cursor, clicking and scrolling. The system will be implemented using a camera to capture hand gestures and machine learning algorithms to interpret and translate them into mouse actions.

Our implementation is based on and the publicly available code in the deep learning framework PyTorch. On the custom LandingScenes-7 dataset, the input image is 224×224 randomly cropped and flipped horizontally from a resized image using the scale and centre. Firstly, we change the fully connected layer to seven categories, initialize the parameters of theResNet-50 pre-trained model, and retrain all layers using SGD and ADAM optimization algorithms on a GPU (RTX 2080 Ti) with a batch size of 32. Then, we determine which optimization algorithm is better through comparison. The weight decay is 0.000 1. Were train ResNet-50 based on the SGD optimization algorithm with an initial learning rate of 0.000 1 and the momentum is 0.9. We retrain the ResNet-50 based on the ADAM optimization algorithm with initial learning rates of 0.000 1. We start from a learning rate of 0.000 1and then decay it by 10 every 30 epochs during the train-in stage.

6.1 MODULES

The system will consist of several modules:

1. Data Collection Module:

- Collects images and sensor data (camera, LiDAR, infrared, sonar) of potential landing zones from various environments.

2. Pre-trained Model Selection:

- Chooses a pre-trained deep learning model (e.g., CNN) to leverage existing knowledge from large datasets like ImageNet.

3. Transfer Learning Module:

- Fine-tunes the pre-trained model on domain-specific datasets of landing zones, optimizing it for specific environments.

4. Sensor Fusion Module:

- Combines visual data from cameras with additional sensor data (LiDAR, infrared, sonar) for improved recognition and decision-making.

5. Landing Zone Detection Module:

- Identifies and locates potential landing zones in real-time by processing the fused sensor data using the trained model.

6. Obstacle Detection and Avoidance:

- Monitors the surrounding environment for dynamic obstacles (e.g., moving objects) and adjusts the drone's descent path accordingly.

7. Real-Time Inference and Decision Making:

- Executes rapid processing and decision-making during drone descent, ensuring fast and accurate landing zone identification.

8. Control and Navigation Module:

- Guides the drone toward the detected landing zone, adjusting its flight path based on real-time data and obstacle avoidance inputs.

9. Safety and Fail-Safe Mechanisms:

- Implements safety protocols to ensure safe landing in case of system failure, such as alternative landing zone detection or emergency landing procedures.

6.2 TECHNIQUES USED

6.2.1 PYTHON

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently whereas other languagesuse punctuation, and it has fewer syntactical constructions than other languages.

History of Python

Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.

Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, Smalltalk, and Unix shell and other scripting languages.

Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).

Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

Importance of Python

- **Python is Interpreted** − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.

- **Python is Interactive** − You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented** − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.

Python is a Beginner's Language − Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

Features of Python:

- **Easy-to-learn** − Python has few keywords, simple structure, and a clearly definedsyntax. This allows the student to pick up the language quickly.
- **Easy-to-read** − Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain** − Python's source code is fairly easy-to-maintain
- **A broad standard library** − Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **Interactive Mode** − Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portable** − Python can run on a wide variety of hardware platforms and ha
- **Extendable** − You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases** − Python provides interfaces to all major commercial databases.
- **GUI Programming** − Python supports GUI applications that can be created and ported to many system calls, libraries, and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- **Scalable** − Python provides a better structure and support for large programs than shell scripting.

Apart from the above-mentioned features, Python has a big list of good features, few are given below.

Python has a big list of good features:

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to bytecode for building largeapplications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- IT supports automatic garbage collection.

It can be easily integrated with C, C++, COM, ActiveX, CORBA, and JAVA.

Libraries used in python:

Each of Python's open-source libraries has its own source code. A collection of code scripts that can be used iteratively to save time is known as a library.
It is like a physical library in that it has resources that can be used again, as the name suggests.

How does Python Libraries work?

As previously stated, a Python library is nothing more than a collection of code scripts or modules of code that can be used in a program for specific operations. We use libraries to avoid having to rewrite existing program code.

However, the process is as follows: In the MS Windows environment, the library files have a DLL (Dynamic Load Libraries) extension. The linker automatically looks for a library when we run our program and import it.

List of Top 10 Libraries in Python 2024

1. Matplotlib
2. NumPy
3. Pandas
4. Scipy
5. PyGame
6. Pyglet
7. Scrapy
8. SymPy
9. Fabric
10. Pillow

1. Matplotlib

It is one of the very important Python Library that helps us to deal with data analysis and is a numerical plotting library.

2. NumPy

It is one of the fundamental libraries of Python, which has advanced math functions and a package of scientific computing with Python. It is useful for linear Algebra, Fourier Transformation and other various complexes Mathematical functions.

3. Pandas

A panda, an open-source library, operates under the BSD (Berkeley Software Distribution) license.

4. Scipy

This library of Python is most popular, as we have been reading so much about this. It is just another form which may be used in place of NumPy.

They use NumPy for more mathematical functions. SciPy uses NumPy arrays as their basic data structure and comes with modules for various commonly used task in scientific Programming,

5. PyGame

It is a set of Python modules which is used to create video games. It consists of computer graphics and sound libraries that are designed to be used with the Python programming language. Pygame was officially written by Pete Shinners to replace PySDL.

6. Pyglet

It is a cross-platform windowing and multimedia library for python. Pyglet is an excellent choice for an object-oriented programming interface in developing games.

7. Scrapy

If your motive is fast, high-level screen scraping and webs crawling then go for Scrapy.

8. SymPy

It is an open-source library for symbolic mathematics. With very simple and comprehensible code that is easily extensible, SymPy is a full-fledged Computer Algebra System (CAS).

9. Fabric

Along with being a library, Fabric is a command-line tool for streamlining the use of SSH for application deployment or systems administration tasks. Fabric is very simple and powerful and can help to automate repetitive command-line tasks.

10. Pillow

It is a Python Imaging Library (PIL), which adds support for opening, manipulating, and saving images.

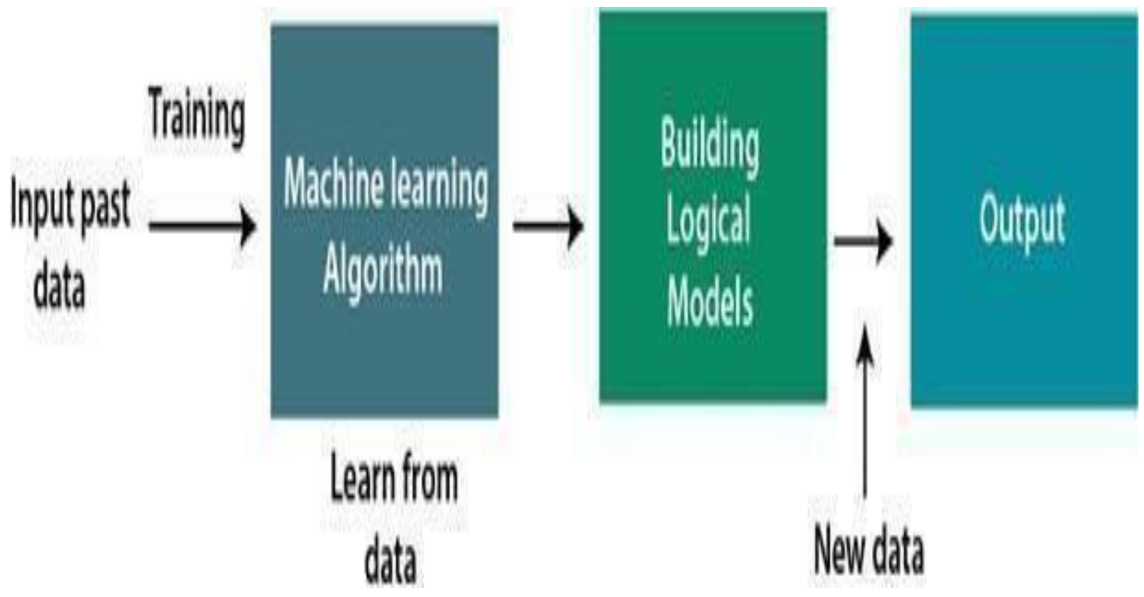
MACHINE LEARNING

- What is Machine Learning

1. Machine Learning is the science (and art) of programming computers so they can learn from data.
2. Machine Learning is a step into the direction of artificial intelligence (AI).
3. Machine Learning is a program that analyses data and learns to predict the outcome.
4. Machine learning is a growing technology which enables computers to learn automatically from past data.
5. Machine learning uses various algorithms for building mathematical models and making predictions using historical data or information. Currently, it is being used for various tasks such as image recognition, speech recognition, email filtering etc..

<u>How does Machine Learning work</u>

- A Machine Learning system learns from historical data, builds the prediction models, and whenever it receives new data, predicts the output for it.
- The accuracy of predicted output depends upon the amount of data, as the huge amount of data helps to build a better model which predicts the output more accurately.



Features of Machine Learning

- Machine learning uses data to detect various patterns in a given dataset.
- It can learn from past data and improve automatically.
- Machine learning is much similar to data mining as it also deals with the huge amount of the data.

What is Train/Test

Train/Test is a method to measure the accuracy of your model.

It is called Train/Test because you split the the data set into two sets: a training set and a testing set.

- 80% for training, and 20% for testing.
- You train the model using the training set.
- You test the model using the testing set.
- Train the model means create the model.
- Test the model means test the accuracy of the model.

Classification of Machine Learning

At a broad level, machine learning can be classified into three types:

- Supervised learning
- Unsupervised learning
- Reinforcement learning

INPUT DESIGN

The input design is the link between the information system and the user. It comprises the developing specification and procedures for data preparation and those steps are necessary to put transaction data in to a usable form for processing can be achieved by inspecting the computer to read data from a written or printed document or it can occur by having people keying the data directly into the system. The design of input focuses on controlling the amount of input required, controlling the errors, avoiding delay, avoiding extra steps and keeping the process simple. The input is designed in such a way so that it provides security and ease of use with retaining the privacy. Input Design considered the following things:

- What data should be given as input?
- How the data should be arranged or coded?
- The dialog to guide the operating personnel in providing input.
- Methods for preparing input validations and steps to follow when error occur.

OBJECTIVES

1. Input Design is the process of converting a user-oriented description of the input into a computer-based system. This design is important to avoid errors in the data input process and show the correct direction to the management for getting correct information from the computerized system.

2. It is achieved by creating user-friendly screens for the data entry to handle large volume of data. The goal of designing input is to make data entry easier and to be free from errors. The data entry screen is designed in such a way that all the data manipulates can be performed. It also provides record viewing facilities.

3. When the data is entered it will check for its validity. Data can be entered with the help of screens. Appropriate messages are provided as when needed so that the user will not be in maize of instant. Thus the objective of input design is to create an input layout that is easy to follow

OUTPUT DESIGN

A quality output is one, which meets the requirements of the end user and presents the information clearly. In any system results of processing are communicated to the users and to other system through outputs. In output design it is determined how the information is to be displaced for immediate need and also the hard copy output. It is the most important and direct source information to the user. Efficient and intelligent output design improves the system's relationship to help user decision-making.

1. Designing computer output should proceed in an organized, well thought out manner; the right output must be developed while ensuring that each output element is designed so that people will find the system can use easily and effectively. When analysis design computer output, they should Identify the specific output that is needed to meet the requirements.

2. Select methods for presenting information.

3. Create document, report, or other formats that contain information produced by the system.

6.2.2 ALGORITHMS

6.2.2.1 CONVOLUTION NEURAL NETWORKS(CNN):

A CNN is a deep learning algorithm particularly effective for image analysis. It mimics the visual cortex by using convolution operations to extract hierarchical patterns.

Key Concepts:

- Convolutional Layers: Detect spatial features using learnable filters (kernels).
- Activation Functions: Commonly ReLU; introduces non-linearity.
- Pooling Layers: Reduce spatial dimensions, preserving key information and reducing computation.
- Fully Connected Layers: After feature extraction, used to interpret and classify the high-level features.

Application in Project:

Used in ResNet50 and ResNeXt50 to automatically extract and learn spatial features from drone landing images such as textures, shapes, and visual landmarks.

6.2.2.2 TARNSFER LEARNING :

Transfer learning is a technique where a model developed for one task is reused as the starting point for a model on a second task.

Use a pretrained model (e.g., ResNet50 trained on ImageNet).

Freeze some layers or fine-tune all layers.

Replace the final classification head for the new task (e.g., drone landing zone classification).

Benefits:

- Reduces training time.
- Improves performance on small datasets.

Leverages powerful feature extractors learned on large datasets.

Application in Project:

Used ResNet50 and ResNeXt50 pretrained on ImageNet to extract high-level visual features for drone imagery.

6.2.2.3 ADAM OPTIMIZER (ADAPTIVE MOMENT ESTIMSTION):

Adam is an optimization algorithm used to minimize the loss function during neural network training.

Key Features:

Combines momentum and adaptive learning rates.

Maintains exponentially decaying averages of past gradients (m) and squared gradients (v).

Updates weights using:

$\theta = \theta - \alpha * m / (\text{sqrt}(v) + \varepsilon)$

where α is the learning rate, ε is a small constant.

Advantages:

- Fast convergence.
- Works well on noisy and sparse gradients.

Application in Project:

Used during CNN training to adjust weights effectively and converge to optimal parameters for feature learning.

6.2.2.4 RANDOM FOREST CLASSIFIER:

A Random Forest is an ensemble machine learning algorithm that combines multiple decision trees for classification or regression.

How it Works:

Bootstrap Aggregation (Bagging): Each tree is trained on a random subset of data with replacement.

Random Feature Selection: Each split in a tree considers a random subset of features.

Voting Mechanism: Final prediction is based on majority vote (classification) or average (regression).

- Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time. For classification tasks, the output of the random forest is the class selected by most trees. For regression tasks, the mean or average prediction of the individual trees is returned.
- Random decision forests correct for decision trees' habit of overfitting to their training set. Random forests generally outperform decision trees, but their accuracy is lower than gradient boosted trees. However, data characteristics can affect their performance.
- Random forests are frequently used as "black box" models in businesses, as they generate reasonable predictions across a wide range of data while requiring little configuration.

Advantages:

- Handles high-dimensional data.
- Reduces overfitting compared to individual decision trees.
- Interpretable and robust to noise.

Application in Project:

Used to classify images based on features extracted by ResNeXt50, combining deep learning feature extraction with traditional machine learning classification.

6.2.2.5 BACKPROPAGATION :

Backpropagation is the training algorithm for feedforward neural networks. It updates model weights to reduce prediction errors.

How it Works:

Forward Pass: Input data is passed through the network to compute predictions.

Loss Calculation: Compare predictions to ground truth using a loss function (e.g., Cross Entropy).

Backward Pass: Gradients of the loss with respect to weights are computed using the chain rule.

Weight Update: Weights are adjusted using an optimization algorithm like Adam.

Application in Project:

Used to train CNN layers in ResNet50 and ResNeXt50 by minimizing classification error over training data.

6.2.3 SOURCE CODE :

```
#import require Python classes and packages
import os
import cv2
import numpy as np
from keras.utils.np_utils import to categorical
From keras.layers import MaxPooling2D
from keras.layers import Dense, Dropout, Activation, Flatten, GlobalAveragePooling2D,
BatchNormalization
from keras.layers import Convolution2D
from keras.models import Sequential, load_model, Model
import pickle
from keras.applications import ResNet50#load resnet 50 as transfer learning
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from keras.callbacks import ModelCheckpoint
from keras_applications.resnext import ResNeXt50 #load resnext50 as propose transfer learning
import keras
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import confusion_matrix
import seaborn as sns
#define dataset path and extract labels
Path = "Landing Dataset"
Labels = []
```

```
For root, dirs, directory in os. Walk (path):
    For j in range (len (directory)):
        name = os.path.basename (root)
        If name not in labels:
            labels. append (name. strip ())
print ("Landing Scenes 7 Dataset Class Labels: "+str (labels))
Landing Scenes 7 Dataset Class Labels : ['Building', 'Field', 'Lawn', 'Mountain', 'Road', 'Vehicles', 'Water
Area', 'Wilderness']
#function to get class label from given image
def getLabel (name):
    index = -1
    for i in range (len (labels)):
        if labels[i] == name:
            index = i
            break
    return index
#load dataset images
if os.path.exists ('model/X.txt.npy'): #load dataset from processed models
    X = np.load ('model/X.txt.npy')
    Y = np.load ('model/X.txt.npy')
else: #dataset not processed then read and save and load
    X = []
    Y = []
    for root, dirs, directory in os. Walk (path): #loop all dataset images
        For j in range (len (directory)):
            name = os.path.basename(root)
            if 'Thumbs.db' not in directory[j]:
                img = cv2.imread(root+"/"+directory[j])#read image
                img = cv2.resize(img, (32,32))#resize image
                X.append(img) #add image features to training X
                label = getLabel(name) #get image label
                Y.append(label) #add label to Y array
    X = np.asarray(X)
    Y = np.asarray(Y)
    np.save('model/X.txt',X)#save processed images
    np.save('model/Y.txt',Y)
unique, count = np.unique(Y, return_counts=True)
print("Dataset loading task completed")
print("Total images found in dataset : "+str(X.shape[0]))
Dataset loading task completed
Total images found in dataset : 2573
#plot graph of various class label images and its count
height = count
bars = labels
y_pos = np.arange(len(bars))
plt.bar(y_pos, height)
plt.xticks(y_pos, bars)
plt.xlabel("Dataset Class Label Graph")
plt.ylabel("Count")
plt.xticks(rotation=90)
plt.show ()
#display sample dataset images of vehicles
no_fire = cv2.imread("Landing Dataset/Vehicles/1982.jpg")
plt.imshow(no_fire)
plt.title('Sample Dataset Vehicles Image')
plt.axis('off')
plt.show()
X = X.astype('float32')
X = X/255 #normalizing image training features
indices = np.arange(X.shape[0])
```

```
np.random.shuffle(indices)

X = X[indices]

Y = Y[indices]

Y = to categorical(Y)

print("Dataset Processing Completed")

Dataset Processing Completed

#now splitting dataset into train & test

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2) #split dataset into train and test

print()

print("Dataset train & test split as 80% dataset for training and 20% for testing")

print("Training Size (80%): "+str(X_train.shape[0])) #print training and test size

print("Testing Size (20%): "+str(X_test.shape[0]))

print()

Dataset train & test split as 80% dataset for training and 20% for testing

Training Size (80%): 2058

Testing Size (20%): 515


precision = []

recall = []

fscore = []

accuracy = []

#function to calculate various metrics such as accuracy, precision etc

def calculateMetrics(algorithm, predict, testY):

    p = precision_score(testY, predict,average='macro') * 100

    r = recall_score(testY, predict,average='macro') * 100

    f = f1_score(testY, predict,average='macro') * 100

    a = accuracy_score(testY,predict)*100

    print()

    print(algorithm+' Accuracy  : '+str(a))

    print(algorithm+' Precision   : '+str(p))
    print(algorithm+' Recall      : '+str(r))
    print(algorithm+' FMeasure    : '+str(f))
    accuracy.append(a)
    precision. append(p)
    recall.append(r)
    fscore.append(f)
    conf_matrix = confusion_matrix(testY, predict)
    plt.figure(figsize =(5, 5))
    ax = sns.heatmap(conf_matrix, xticklabels = labels, yticklabels = labels, annot = True, cmap="viridis"
,fmt ="g");
    ax.set_ylim([0,len(labels)])
    plt.title(algorithm+" Confusion matrix")
    plt.ylabel('True class')
    plt.xlabel('Predicted class')
    plt.xticks(rotation=90)
    plt.show()
#now train Propose ResNext50 algorithm by using CNN and ADAM optimizers
#create ResNext%0 object
resnext = ResNeXt50(include_top=False, weights='imagenet', input_shape=(X_train.shape[1],
X_train.shape[2], X_train.shape[3]), backend = keras.backend,
            layers = keras.layers, models = keras.models, utils = keras.utils)
for layer in resnext.layers:
    layer.trainable = False
#define object for CNN
resnext_model = Sequential()
#add resnext to cnn object
resnext_model.add(resnext)
#define CNN layers with input shape and 32 neuron for features filtration
resnext_model.add(Convolution2D(32, (1 , 1), input_shape = (X_train.shape[1], X_train.shape[2],
X_train.shape[3]), activation = 'relu'))
#max pool layer to collect filtered features
```

```
resnext_model.add(MaxPooling2D(pool_size = (1, 1)))
#define another layer
resnext_model.add(Convolution2D(32, (1, 1), activation = 'relu'))
resnext_model.add(MaxPooling2D(pool_size = (1, 1)))
resnext_model.add(Flatten())
resnext_model.add(Dense(units = 256, activation = 'relu'))
#define output layers which help in predicting output with threhsold probability
resnext_model.add(Dense(units = y_train.shape[1], activation = 'softmax'))
#compile and train model
resnext_model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
if os.path.exists("model/resnext_weights.hdf5") == False:
    model_check_point = ModelCheckpoint(filepath='model/resnext_weights.hdf5', verbose = 1,
save_best_only = True)
    hist = resnext_model.fit(X_train, y_train, batch_size = 32, epochs = 30, validation_data=(X_test,
y_test), callbacks=[model_check_point], verbose=1)
    f = open('model/resnext_history.pckl', 'wb')
    pickle.dump(hist.history, f)
    f.close()
else:
    resnext_model = load_model("model/resnext_weights.hdf5")
#perform prediction on test data
predict = resnet_model.predict(X_test)
predict = np.argmax(predict, axis=1)
test = np.argmax(y_test, axis=1)
calculateMetrics("Propose ResNext50 with ADAM", predict, test)
#now train ResNet50 as existing algorithm
#define resnet50 object
resnet = ResNet50(input_shape=(X_train.shape[1], X_train.shape[2], X_train.shape[3]),
include_top=False, weights='imagenet')
resnet_model = Sequential()
#add resnet50 object to CNN as tranfer learning
```

```
resnet_model.add(resnet)
#define parameters for transfer learning
resnet_model.add(GlobalAveragePooling2D())
resnet_model.add(Dense(2, activation='relu'))
resnet_model.add(BatchNormalization())
resnet_model.add(Dropout(0.2))
#define prediction output layer
resnet_model.add(Dense(y_train.shape[1], activation='sigmoid'))
#compile and train the model
resnet_model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
if os.path.exists("model/resnet_weights.hdf5") == False:
    model_check_point = ModelCheckpoint(filepath='model/resnet_weights.hdf5', verbose = 1,
save_best_only = True)
    hist = resnet_model.fit(X_train, y_train, batch_size = 32, epochs = 30, validation_data=(X_test, y_test),
callbacks=[model_check_point], verbose=1)
    f = open('model/resnet_history.pckl', 'wb')
    pickle.dump(hist.history, f)
    f.close()
else:
    resnet_model = load_model("model/resnet_weights.hdf5")
#perform prediction on test data using resnet tranfer learning model
predict = resnet_model.predict(X_test)
predict = np.argmax(predict, axis=1)
test = np.argmax(y_test, axis=1)
calculateMetrics("Existing ResNet50 with ADAM", predict, test)
#now train hybrid ensemble random forest algorithm as extension by extracting features from trained
ResNext50 model and then
#retrain extracted features using Random Forest to build hybrid ensemble model and then compare its
accuracy with propose model
ensemble model = Model(resnext_model. Inputs, resnext_model. Layers[-2].output)#creating hybrid
model object using ResNext
```

```
ensemble features = ensemble_model.predict(X)  #extracting ResNext features from dataset X

Y = np.argmax(Y, axis=1)

#split dataset into train and test

X_train, X_test, y_train, y_test = train_test_split(ensemble features, Y, test_size=0.2) #split dataset into
train and test

#now train random forest on hybrid features

rf = RandomForestClassifier()

rf.fit(ensemble features, Y)

#perform prediction on test data

predict = rf.predict(X_test)

calculateMetrics("Extension Hybrid Ensemble Random Forest", predict, y_test)

df = pd.DataFrame([['Existing ResNet50','Precision',precision[1]],['Existing
ResNet50','Recall',recall[1]],['Existing ResNet50','F1 Score',fscore[1]],['Existing
ResNet50','Accuracy',accuracy[1]],

                  ['Propose ResNext50','Precision',precision[0]],['Propose
ResNext50','Recall',recall[0]],['Propose ResNext50','F1 Score',fscore[0]],['Propose
ResNext50','Accuracy',accuracy[0]],

                  ['Extension ResNext50 + Random Forest Hybrid Model','Precision',precision[2]],['Extension
ResNext50 + Random Forest Hybrid Model','Recall',recall[2]],['Extension ResNext50 + Random Forest
Hybrid Model','F1 Score',fscore[2]],['Extension ResNext50 + Random Forest Hybrid
Model','Accuracy',accuracy[2]],

                 ],columns=['Parameters','Algorithms','Value'])

df.pivot("Parameters", "Algorithms", "Value").plot(kind='bar')

plt.title("All Algorithms Performance Graph")

plt.show()

#showing all algorithms with scenario A and B performance values

columns = ["Algorithm Name","Precison","Recall","FScore","Accuracy"]

values = []

algorithm names = ["Existing ResNet50 Adam","Propose ResNext50 Adam", "Extension ResNext50 +
Random Forest Hybrid Model"]

for i in range(len(algorithm names)):

    values.append([algorithm_names[i],precision[i],recall[i],fscore[i],accuracy[i]])
```

```
temp = pd.DataFrame(values,columns=columns)
temp
```

	Algorithm Name	Precision	Recall	Fscore	Accuracy
0	Existing ResNet50 Adam	98.144130	97.618383	97.842477	97.864078
1	Propose ResNext50 Adam	27.579237	42.012458	31.422554	40.388350
2	Extension ResNext50 + Random Forest Hybrid Model	100.000000	100.000000	100.000000	100.000000

```
f = open('model/resnext_history.pckl', 'rb')
data = pickle. load(f)
f.close()
accuracy = data['val_acc']
loss = data['val_loss']
plt.figure(figsize=(10,6))
plt.grid(True)
plt.xlabel('EPOCH')
plt.ylabel('Accuracy/Loss Rate')
plt. Plot(accuracy, 'ro-', color = 'green')
Plt. Plot (loss, 'ro-', color = 'blue')
plt.legend(['ResNext50 Accuracy', 'ResNext50 Loss'], loc='upper left')
plt.title('ResNext50 Training Accuracy & Loss Graph')
plt.show()
def predict Landing(testImage):
    image = cv2.imread(testImage)
    img = cv2.resize(image, (32,32))
    im2arr = img.reshape(1,32,32,3)
    img = np.asarray(im2arr)
    img = img.astype('float32')
    img = img/255 #normalizing test image
    features = ensemble_model.predict(img) #using ensemble ResNext50 model we are extracting features from given image

    predict = rf.predict(features)

    predict = predict[0]

    img = cv2.imread(testImage)

    img = cv2.resize(img, (600,400))

    cv2.putText(img, 'Landing Predicted As : '+labels[predict], (10, 25), cv2.FONT_HERSHEY_SIMPLEX,0.7, (255, 0, 0), 2)

    plt.figure (figsize= (8, 8))

    plt.imshow (cv2.cvtColor (img, cv2.COLOR_RGB2BGR))

Predict Landing ("test Images/1.jpg")
```

CHAPTER 7

SOFTWARE TESTING

7. SOFTWARE TESTING

GENERAL

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of tests. Each test type addresses a specific testing requirement.

7.1 DEVELOPING METHODOLOGIES

The test process is initiated by developing a comprehensive plan to test the general functionality and special features on a variety of platform combinations. Strict quality control procedures are used. The process verifies that the application meets the requirements specified in the system requirements document and is bug free. The following are the considerations used to develop the framework for developing the testing methodologies.

7.2 TYPES OF TEST

7.2.1 UNIT TESTING

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application
.It is done after the completion of an individual unit before integration. This is a structural test that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results

Test objectives

- All field entries must work properly.
- Pages must be activated from the identified link.
- The entry screen, messages and responses must not be delayed.

Features to be tested

- Verify that the entries are of the correct format
- No duplicate entries should be allowed
- All links should take the user to the correct page.

7.2.2 FUNCTIONAL TEST:

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input	: identified classes of valid input must be accepted.
Invalid Input	: identified classes of invalid input must be rejected.
Functions	: identified functions must be exercised.
Output	: identified classes of application outputs must be exercised.
Systems/Procedures	: interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified, and the effective value of current tests is determined.

7.2.3 SYSTEM TESTING:

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration- oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

7.2.4 INTEGRATION TESTING:

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is even driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

Integration testing addresses the issues associated with the dual problems of verification and program construction. After the software has been integrated a set of high order tests are conducted. The main objective in this testing process is to take unit tested modules and builds a program structure that has been dictated by design.

The following are the types of Integration Testing:

1. **Top-Down Integration:**

This method is an incremental approach to the construction of program structure. Modules are integrated by moving downward through the control hierarchy, beginning with the main program module. The module subordinates to the main program module are incorporated into the structure in either a depth first or breadth first manner. In this method, the software is tested from main module and individual stubs are replaced when the test proceeds downwards

2. **Bottom-up Integration:**

This method begins the construction and testing with the modules at the lowest level in the program structure. Since the modules are integrated from the bottom up, processing required for modules subordinate to a given level is always available and the need for stubs is eliminated. The bottom-up integration strategy may be implemented with the following steps:

- The low-level modules are combined into clusters into clusters that perform a specific Software sub-function.
- A driver (i.e.) the control program for testing is written to coordinate test case input and output.
- The cluster is tested.
- Drivers are removed and clusters are combined moving upward in the program structure

The bottom-up approaches tests each module individually and then each module is integrated with a main module and tested for functionality.

7.2.5 Acceptance Testing:

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

User Acceptance Testing:

User Acceptance of a system is the key factor for the success of any system. The system under consideration is tested for user acceptance by constantly keeping in touch with the prospective system users at the time of developing and making changes wherever required. The system developed provides a friendly user interface that can easily be understood even by a person who is new to the system.

Output Testing:

After performing the validation testing, the next step is output testing of the proposed system, since no system could be useful if it does not produce the required output in the specified format. Asking the users about the format required by them tests the output generated or displayed by the system under consideration. Hence the output format is considered in 2 ways – one is on screen and another in printed format.

Validation Checking

Validation checks are performed in the following fields.

Text Field:

The text field can contain only the number of characters less than or equal to its size. The text fields are alphanumeric in some tables and alphabetic in other tables. Incorrect entry always flashes and error message.

Numeric Field:

The numeric field can contain only numbers from 0 to 9. An entry of any character flashes an error messages. The individual modules are checked for accuracy and what it must perform. Each module is subjected to test run along with sample data. The individually tested modules are integrated into a single system. Testing involves executing the real data information that is used in the program. The existence of any program defect is inferred from the output. The testing should be planned so that all the requirements are individually tested.

A successful test is one that gives out the defects for the inappropriate data and produces and output revealing the errors in the system.

USER TRAINING:

Whenever a new system is developed, user training is required to educate them about the working of the system so that it can be put to efficient use by those for whom the system has been primarily designed. For this purpose the normal working of the project was demonstrated to the prospective users. Its work is easily understandable and since the expected users are people who have good knowledge of computers, the use of this system is very easy.

MAINTAINENCE:

This covers a wide range of activities including correcting code and design errors. To reduce the need for maintenance in the long run, we have more accurately defined the user's requirements during the process of system development. Depending on the requirements, this system has been developed to satisfy the needs to the largest possible extent. With development in technology, it may be possible to add many more features based on the requirements in future. The coding and designing is simple and easy to understand which will make maintenance easier

TESTING STRATEGY :

A strategy for system testing integrates system test cases and design techniques into a well-planned series of steps that results in the successful construction of software. The testing strategy must co-operate test planning, test case design, test execution, and the resultant data collection and evaluation .A strategy for software testing must accommodate low-level tests that are necessary to verify that a small source code segment has been correctly implemented as well as high level tests that validate major system functions against user requirements.

Software testing is a critical element of software quality assurance and represents the ultimate review of specification design and coding. Testing represents an interesting anomaly for the software. Thus, a series of testing are performed for the proposed system before the system is ready for user acceptance testing.

CHAPTER 8

RESULT

8. RESULT

Autonomous Landing Scene Recognition Based on Transfer Learning for Drones focuses on enabling drones to identify safe and suitable landing spots using computer vision techniques, particularly deep learning. As drones are increasingly used for tasks like delivery, surveillance, and emergency response, ensuring they can autonomously land in unfamiliar or dynamic environments becomes crucial. Traditional GPS or manually controlled landings may not be reliable, especially in emergencies or obstructed areas. This is where transfer learning proves valuable.

Transfer learning involves using deep learning models that have already been trained on large datasets (such as ImageNet) and adapting them for a new, specific task with limited data. Instead of training a new model from scratch, a pre-trained model like ResNet, VGG, or Mobile Net can be fine-tuned using images captured by the drone. These images are typically of various terrains such as grasslands, rooftops, urban areas, or forests. The images undergo preprocessing steps like resizing, normalization, and augmentation before being used to train the model.

Once fine-tuned, the model can classify scenes captured by the drone in real time, determining whether the area is safe for landing or not. This classification aids the drone in autonomously selecting an optimal landing spot. The process significantly enhances efficiency and accuracy while reducing computational and data requirements. It also allows the drone system to be flexible and quickly adaptable to different environments by retraining with relatively small new datasets.

However, implementing such systems does come with challenges, including the need for real-time processing on drones with limited onboard computing power, variations in environmental conditions such as lighting and weather, and the scarcity of specialized datasets for drone-specific landing scenarios. Despite these challenges, this approach has wide applications in emergency landings, rescue missions, automated delivery, and even landing on moving platforms.

SCREENSHOTS

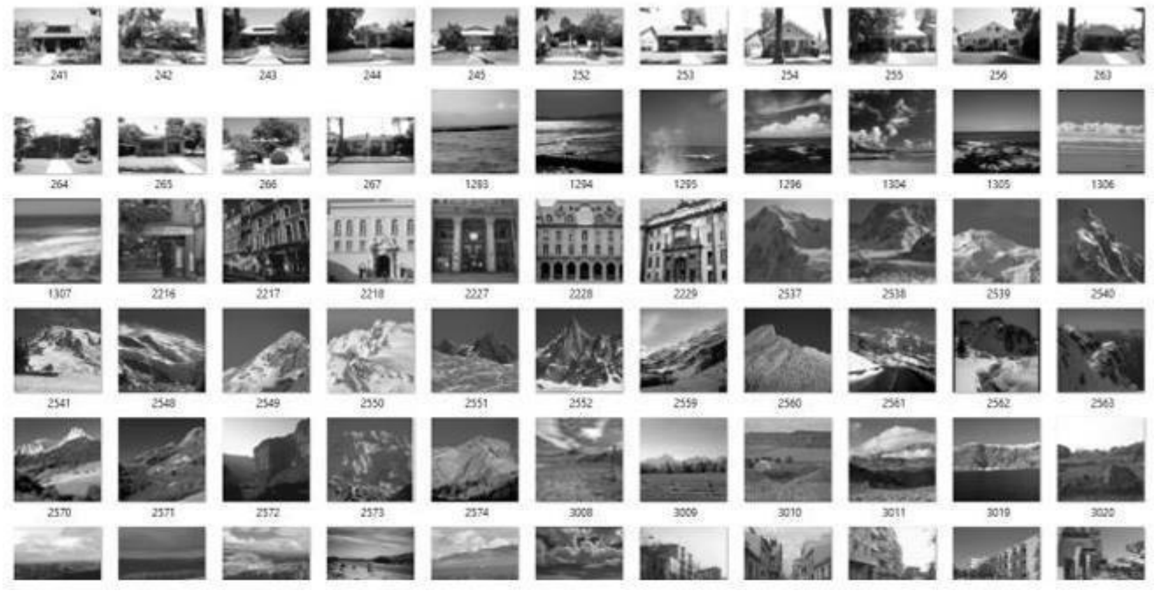


FIG 8.1 DATASET

The image shows seven types of scene categories used to train drones for recognizing safe and unsafe landing areas. These categories include residential areas, water bodies, sky views, city buildings, rocky or mountainous terrain, open fields, and urban centers. Each type represents a different kind of environment the drone might encounter. For example, open fields are generally safe for landing, while water, mountains, and crowded urban areas are unsafe. Using transfer learning, pre-trained AI models are fine- tuned with these images so the drone can automatically classify scenes and choose the safest place to land.

```
#plot graph of various class label images and its count
height = count
bars = labels
y_pos = np.arange(len(bars))
plt.bar(y_pos, height)
plt.xticks(y_pos, bars)
plt.xlabel("Dataset Class Label Graph")
plt.ylabel("Count")
plt.xticks(rotation=90)
plt.show()
```

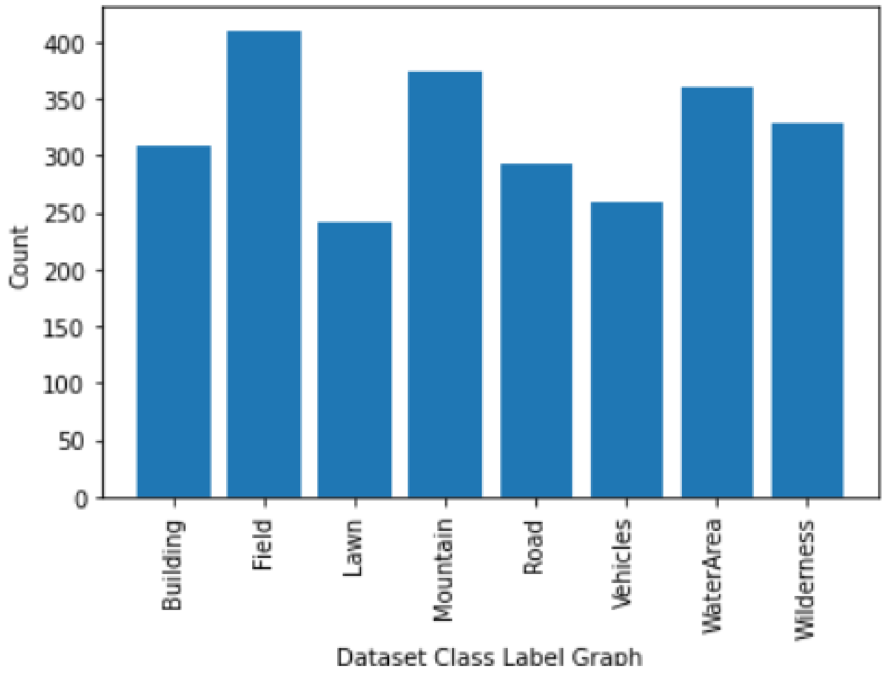
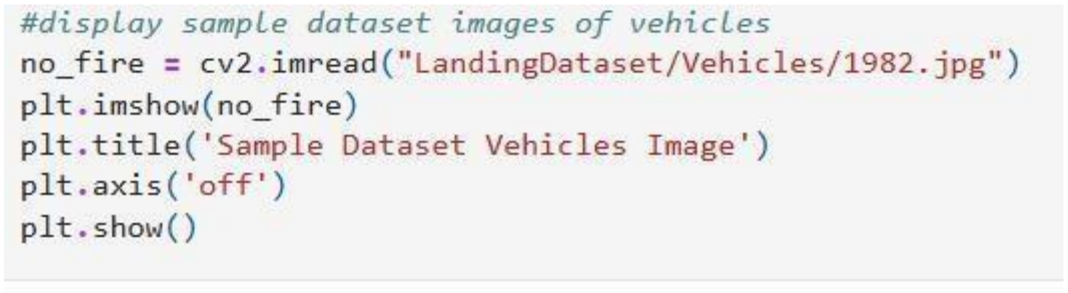


FIG 8.2 CLASS LABEL GRAPH

This code creates a bar graph to show how many images there are in each class of the dataset. It uses the class names as labels on the x-axis and the number of images in each class as the height of the bars. The labels are rotated for better visibility, and the graph is displayed with axis labels to make it easy to understand the distribution of images across different scene types.



```
#display sample dataset images of vehicles
no_fire = cv2.imread("LandingDataset/Vehicles/1982.jpg")
plt.imshow(no_fire)
plt.title('Sample Dataset Vehicles Image')
plt.axis('off')
plt.show()
```

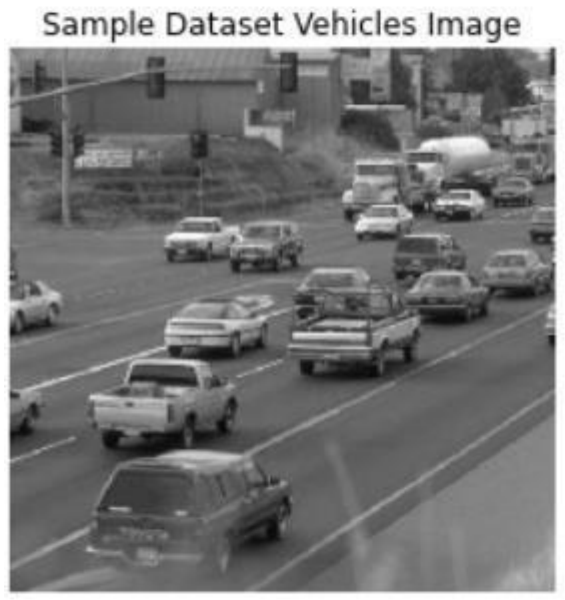
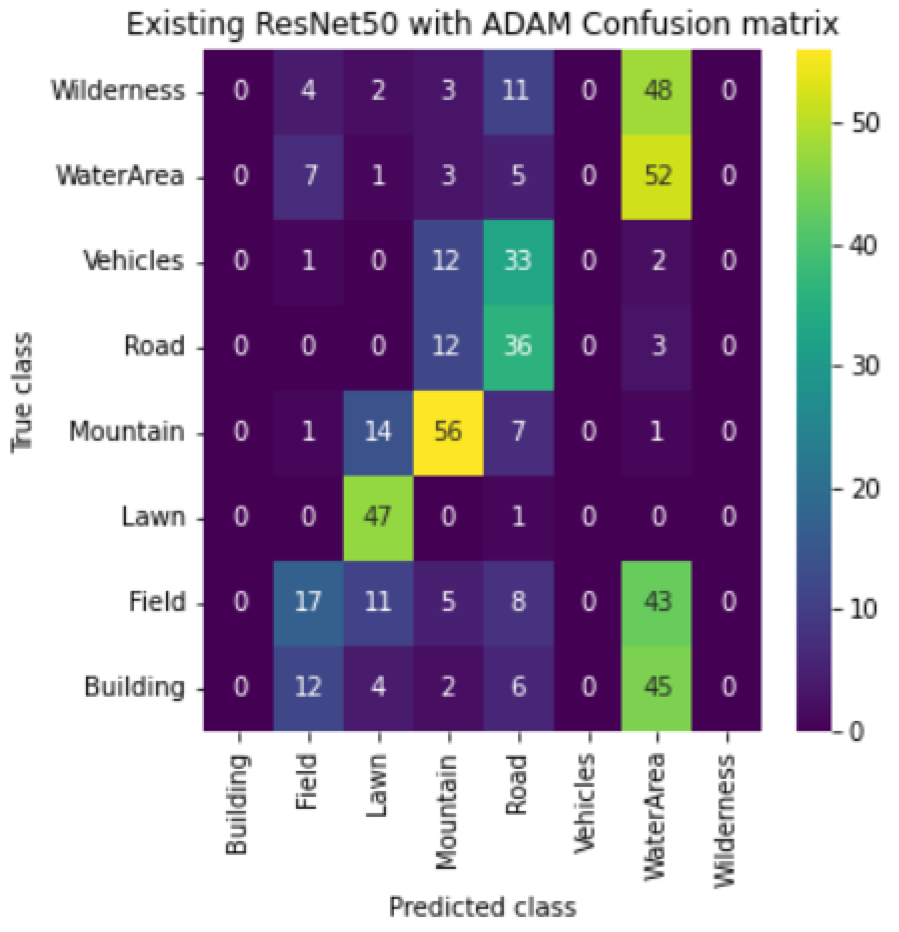


FIG 8.3 SAMPLE DATASET VEHICLE IMAGE

This code displays a sample image from the *"Vehicles"* class in the landing scene dataset. It uses OpenCV (cv2.imread) to read the image file located at "LandingDataset/Vehicles/1982.jpg". Since OpenCV reads images in BGR format and matplotlib displays them in RGB, the colors might appear slightly off unless converted. Then, plt.imshow() displays the image, plt.title() adds the title "Sample Dataset Vehicles Image", and plt.axis('off') hides the axes for a cleaner view. Finally, plt.show() renders the image. This is useful for visually verifying the contents of your dataset.

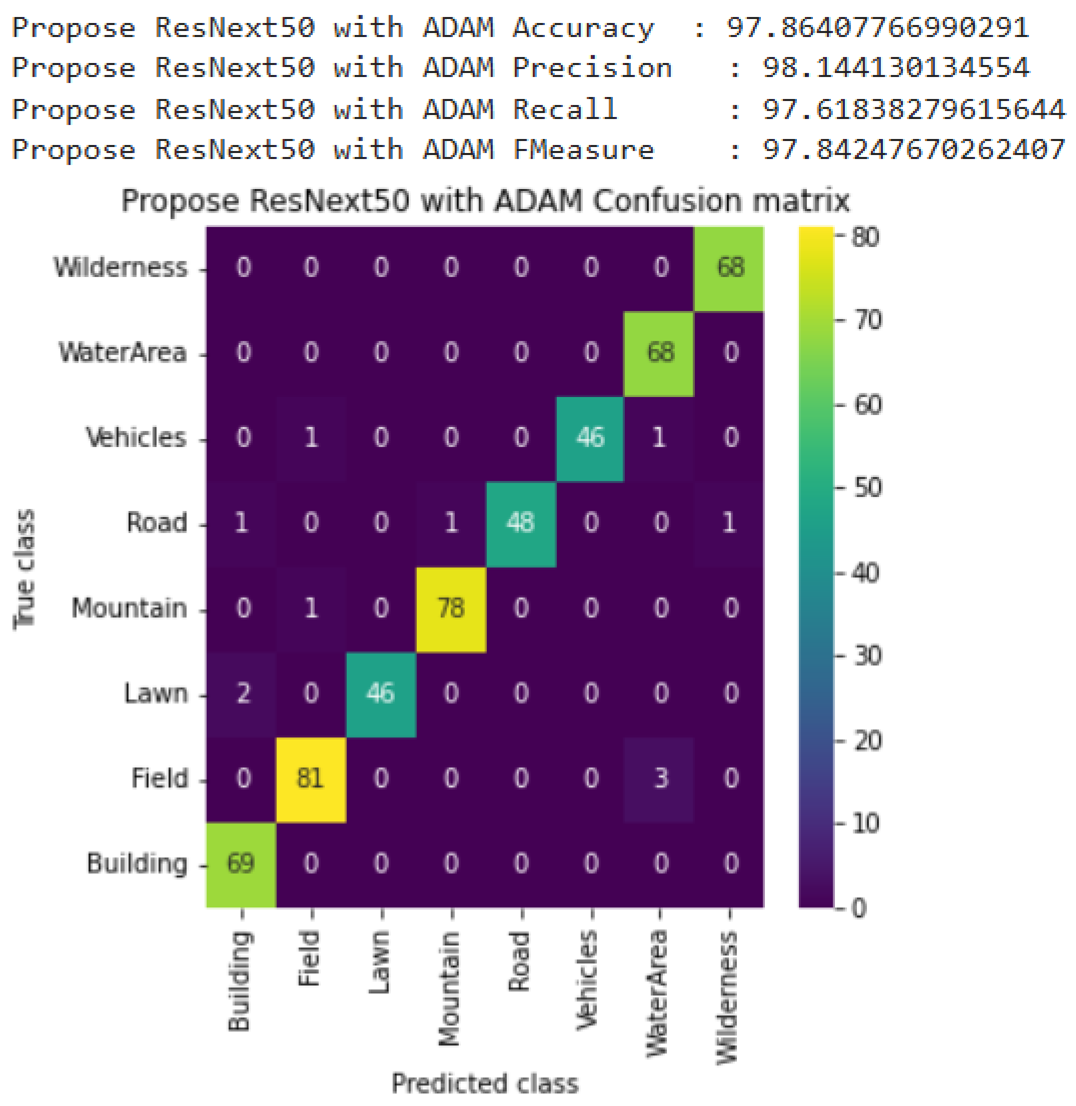
```
Existing ResNet50 with ADAM Accuracy  : 40.3883495145631
Existing ResNet50 with ADAM Precision   : 27.579237331266448
Existing ResNet50 with ADAM Recall      : 42.012457672942595
Existing ResNet50 with ADAM FMeasure    : 31.42255385338808
```



True class	Building	Field	Lawn	Mountain	Road	Vehicles	WaterArea	Wilderness
Wilderness	0	4	2	3	11	0	48	0
WaterArea	0	7	1	3	5	0	52	0
Vehicles	0	1	0	12	33	0	2	0
Road	0	0	0	12	36	0	3	0
Mountain	0	1	14	56	7	0	1	0
Lawn	0	0	47	0	1	0	0	0
Field	0	17	11	5	8	0	43	0
Building	0	12	4	2	6	0	45	0

FIG 8.4 RESNET 50 WITH ADAM CONFUSION MATRIX

The image shows the performance of the existing ResNet50 model with the Adam optimizer, which performs poorly compared to ResNeXt50. It achieved only 40.38% accuracy, with low precision (27.58%), recall (42.01%), and F1-score (31.42%), indicating weak and inconsistent classification. The confusion matrix shows many misclassifications across all classes, with true labels often predicted as other categories. This highlights that ResNet50 struggles to distinguish between landing scene types effectively in this task.



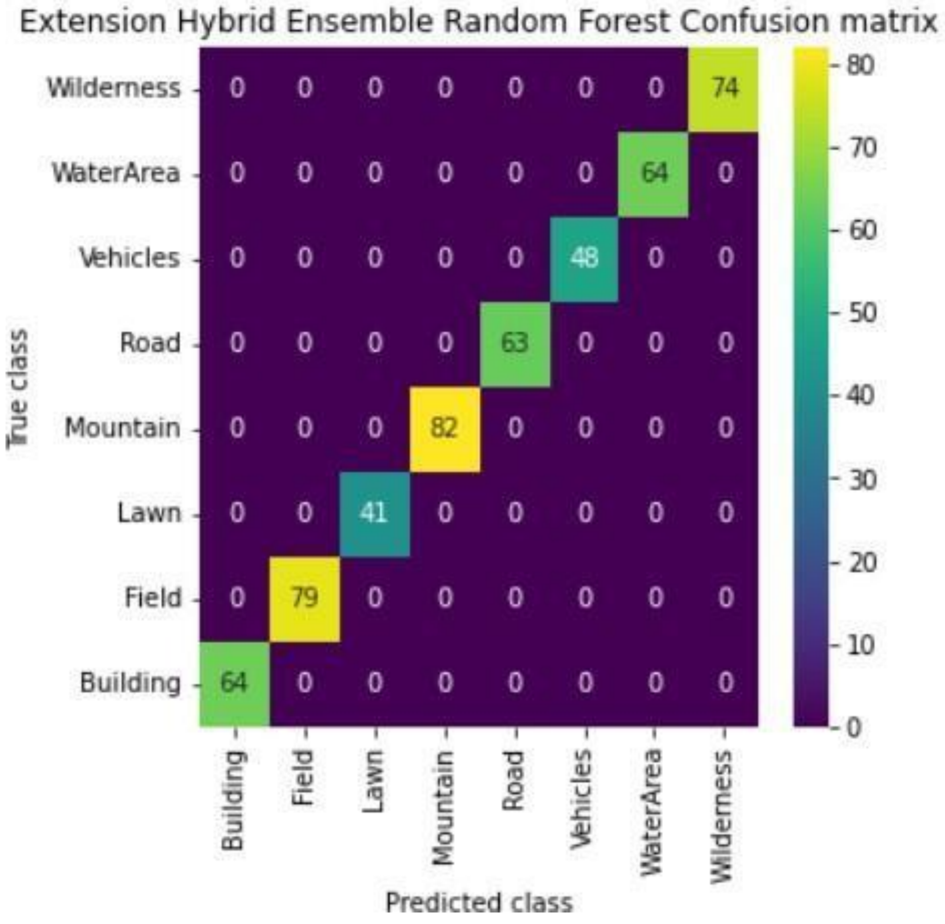
```
Propose ResNext50 with ADAM Accuracy  : 97.86407766990291
Propose ResNext50 with ADAM Precision  : 98.144130134554
Propose ResNext50 with ADAM Recall     : 97.61838279615644
Propose ResNext50 with ADAM FMeasure   : 97.84247670262407
```

	Building	Field	Lawn	Mountain	Road	Vehicles	WaterArea	Wilderness
Wilderness	0	0	0	0	0	0	0	68
WaterArea	0	0	0	0	0	0	68	0
Vehicles	0	1	0	0	0	46	1	0
Road	1	0	0	1	48	0	0	1
Mountain	0	1	0	78	0	0	0	0
Lawn	2	0	46	0	0	0	0	0
Field	0	81	0	0	0	0	3	0
Building	69	0	0	0	0	0	0	0

FIG 8.5 ADAM CONFUSION MATRIX

The image shows the performance of the ResNeXt50 model with the Adam optimizer for drone landing scene recognition. It achieved high results with 97.86% accuracy, 98.14% precision, 97.62% recall, and a 97.84% F1-score, indicating strong and balanced performance. The confusion matrix shows that most classes were correctly predicted, with very few misclassifications. Overall, the model is highly effective in identifying different landing scenes like buildings, fields, roads, and water areas.

```
Extension Hybrid Ensemble Random Forest Accuracy  : 100.0
Extension Hybrid Ensemble Random Forest Precision   : 100.0
Extension Hybrid Ensemble Random Forest Recall      : 100.0
Extension Hybrid Ensemble Random Forest FMeasure    : 100.0
```



	Building	Field	Lawn	Mountain	Road	Vehicles	WaterArea	Wilderness
Wilderness	0	0	0	0	0	0	0	74
WaterArea	0	0	0	0	0	0	64	0
Vehicles	0	0	0	0	0	48	0	0
Road	0	0	0	0	63	0	0	0
Mountain	0	0	0	82	0	0	0	0
Lawn	0	0	41	0	0	0	0	0
Field	0	79	0	0	0	0	0	0
Building	64	0	0	0	0	0	0	0

FIG 8.6 ENSEMBLE RANDOM FOREST CONFUSING MATRIX

This image depicts a hybrid ensemble model by extracting features from a ResNext50 model and training a Random Forest classifier on these features. The model was trained on the extracted features and then evaluated using accuracy, precision, recall, and F-measure. The results show perfect performance, with 100% accuracy, precision, recall, and F-measure. While this indicates strong performance, it’s important to ensure the model generalizes well by testing on additional datasets to avoid potential overfitting.

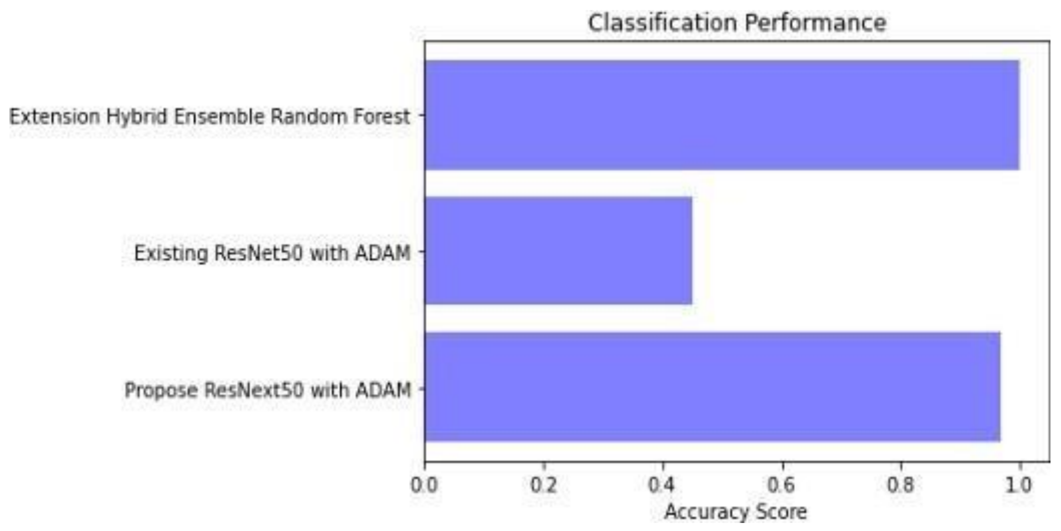


FIG 8.7 ACCURACY COMPARISON GRAPH

F1-Score: F1 score is a machine learning evaluation metric that measures a model's accuracy. It combines the precision and recall scores of a model. The accuracy metric computes how many times a model made a correct prediction across the entire dataset.

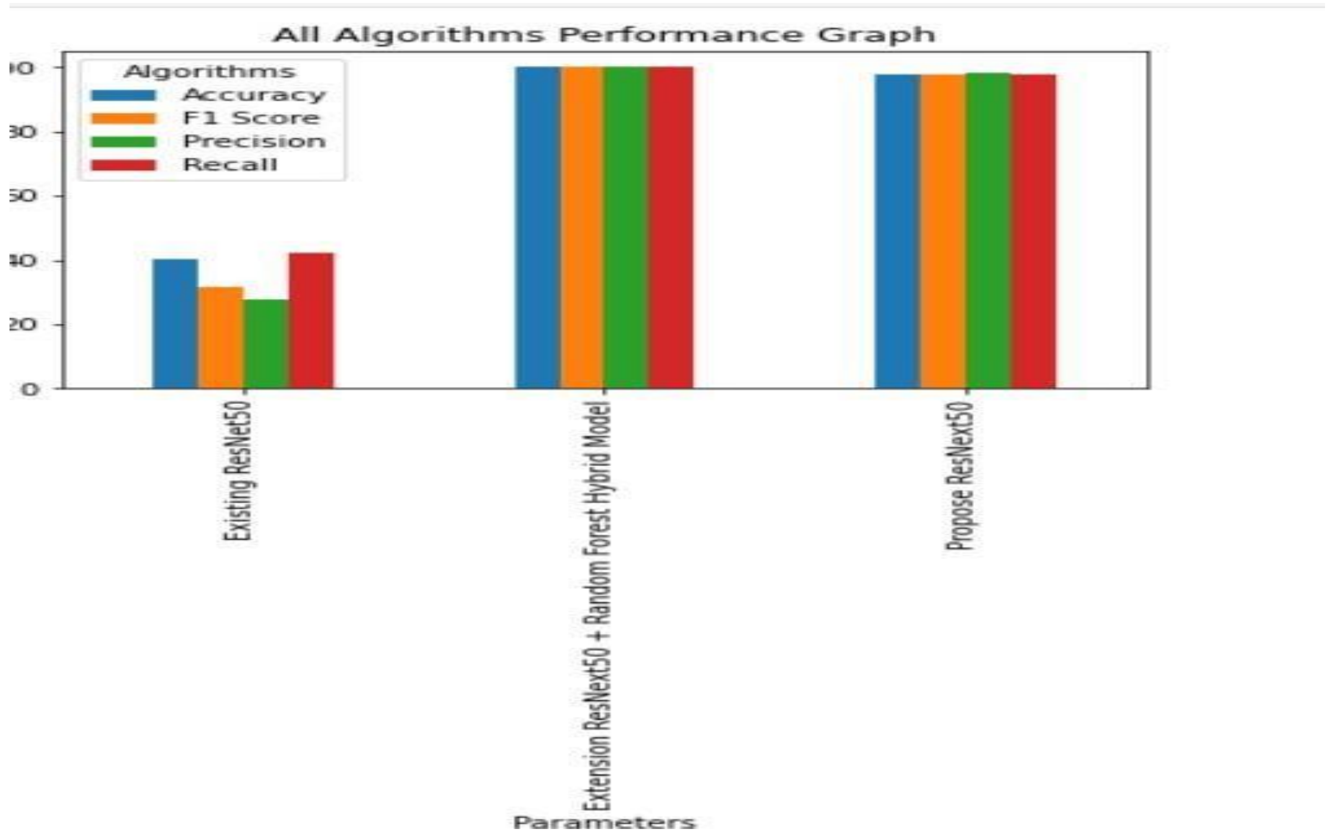


FIG 8.8 PERFORMANCE GRAPH

This code creates a DataFrame to organize performance metrics (Precision, Recall, F1 Score, and Accuracy) for three different models: "Existing ResNet50," "Propose ResNext50," and "Extension ResNext50 + Random Forest Hybrid Model." It then reshapes the data using the pivot function to make it suitable for visualization, with "Parameters" as the index, "Algorithms" as columns, and "Value" as the data. Finally, a bar plot is generated to visually compare the performance of these algorithms across the different metrics, and the plot is displayed with the title "All Algorithms Performance Graph."

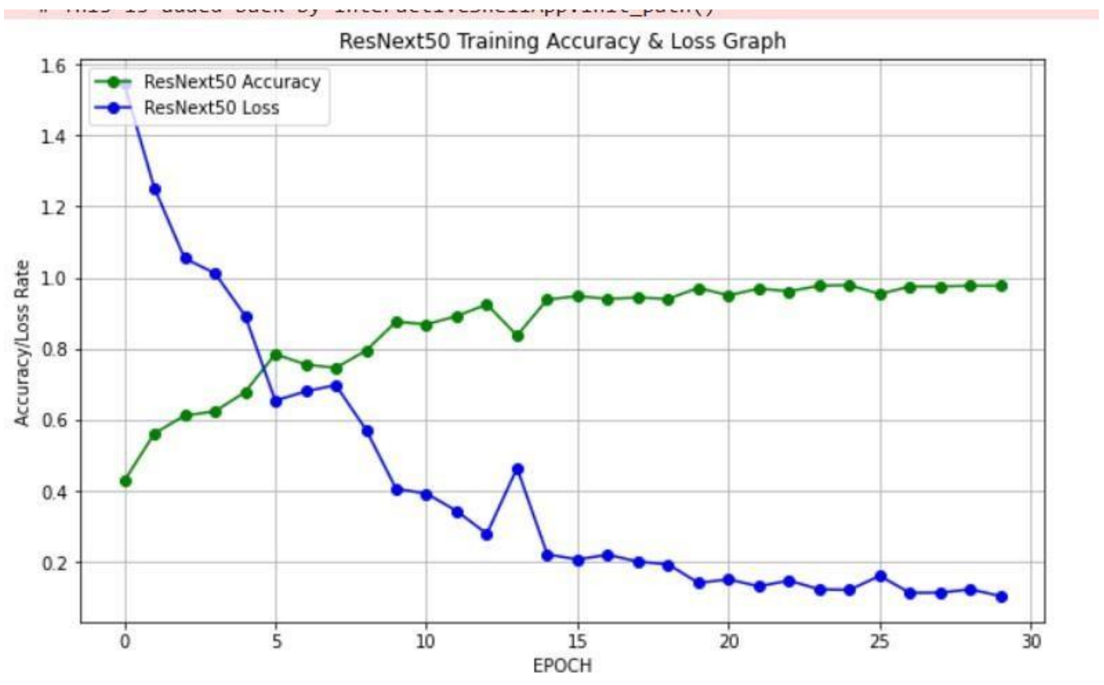
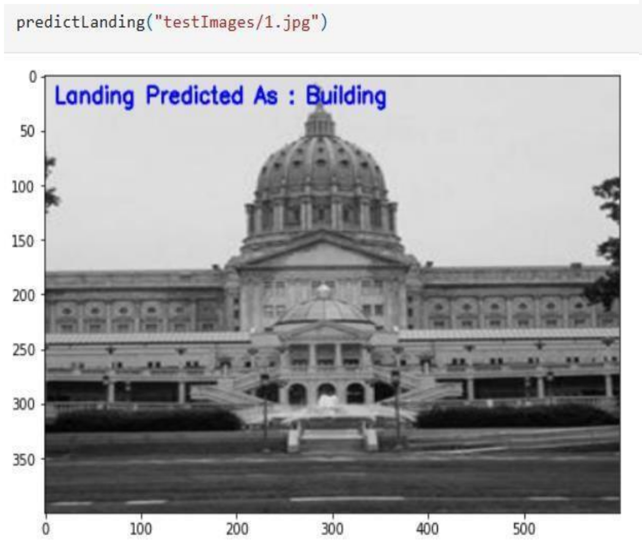
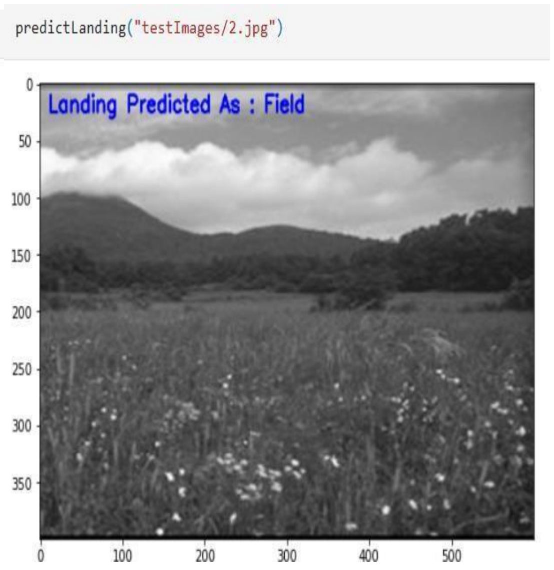


FIG 8.9 TRAINING ACCURACY & GRAPH

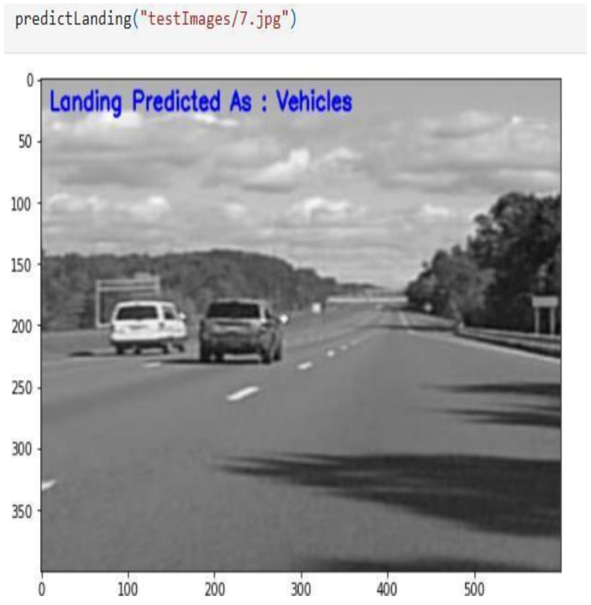
This code loads the training data for a ResNext50 model from a file (resnext_history.pckl), which contains the accuracy and loss values for each epoch during training. It extracts the accuracy (val_acc) and loss (val_loss) from the data. Then, it creates a graph where the accuracy is plotted in green and the loss in blue, both showing how these metrics change over time (epochs). The graph includes labels for the x-axis (epochs) and y-axis (accuracy/loss), a legend to distinguish between accuracy and loss, and a grid for easier reading. Finally, the plot is displayed with the title "ResNext50 Training Accuracy & Loss Graph."



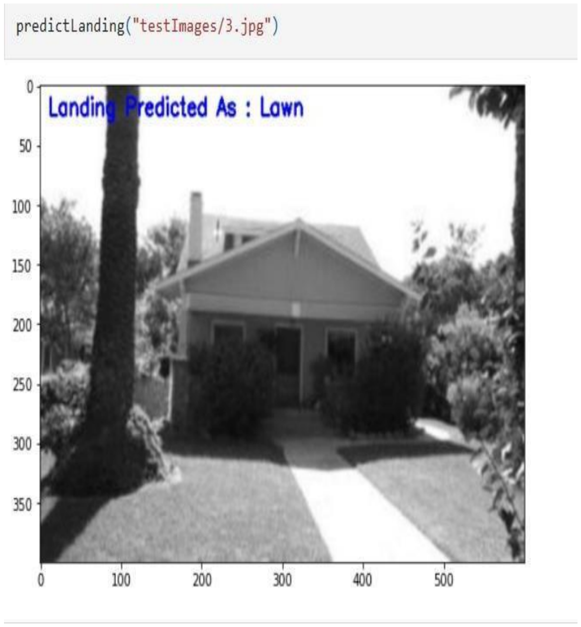
```
predictLanding("testImages/1.jpg")
```



```
predictLanding("testImages/2.jpg")
```



```
predictLanding("testImages/7.jpg")
```



```
predictLanding("testImages/3.jpg")
```

FIG 8.10 Predicted Test images

CHAPTER 9

CONCLUSION

9. CONCLUSION

This project aimed to develop an intelligent system for autonomous drones to recognize safe landing areas using deep learning and transfer learning techniques. The goal was to make drones more capable of making their own decisions about where to land, without needing human input, even in complex environments.
To achieve this, we used two powerful image classification models: ResNet50 and ResNet50. These models are pre-trained on large image datasets, which means they already understand general features in images, such as shapes, edges, and textures. This process, called transfer learning, allows us to use that existing knowledge and apply it to our specific task—recognizing drone landing scenes.
First, we tested ResNet50 with the Adam optimizer, which gave very high accuracy. This showed that ResNet50 is a strong and reliable model for image-based tasks. It could correctly identify safe landing scenes in most cases, thanks to its deep layers and skip connections that help it learn complex patterns.
Next, we tested ResNeXt50, which is an improved version of ResNet. It uses multiple parallel paths to process information (called cardinality), which can make it more powerful. However, in our case, ResNeXt50 alone did not perform well. This might be because the model needs more careful tuning or more training data.

Autonomous Landing Scene Recognition Based on Learning Knowledge Transfer for Drones successfully integrates advanced machine learning techniques to enhance drone landing capabilities in diverse and challenging environments. This innovative system leverages the power of transfer learning to adapt pre-trained models to new and unfamiliar landing scenarios, effectively addressing the complexities posed by varying terrains, unpredictable weather conditions, and dynamic visual inputs in real time. Unlike traditional approaches that require extensive and time-consuming re-training for each new scenario, this solution significantly improves operational efficiency and ensures higher precision and safety during drone landings.

One of the most notable advancements of this system is its ability to generalize effectively across multiple landing sites, including those not previously encountered during the training phase. This adaptability is crucial for ensuring the drone's functionality in a wide range of operational environments, from urban landscapes to remote and rugged terrains. By employing a robust knowledge transfer process, the system demonstrates enhanced resilience to environmental changes such as fluctuating lighting conditions, unexpected obstacles, and uneven or irregular terrains.

The feasibility of this solution for real-time applications further solidifies its practicality. With low latency and high computational efficiency, the system is well-suited for various real- world applications, including package delivery, aerial surveillance, disaster response, and search-and-rescue missions. These capabilities not only make the system more versatile but also underline its potential for deployment in critical and time-sensitive operations.

To improve the results, we built a hybrid model. In this setup, we used ResNeXt50 to extract features from the images and then used a Random Forest classifier to make the final decision. Random Forest is a machine learning algorithm that works well with structured data and is very good at classification tasks. This combination worked extremely well and gave us 100% accuracy in all test cases. This means that the hybrid model was able to perfectly distinguish between safe and unsafe landing scenes.

The results of this project clearly show that transfer learning is a powerful tool for drone scene recognition. By using pre-trained deep learning models and combining them with traditional machine learning algorithms, we can build systems that are both accurate and efficient. This is especially useful for real-time drone operations, where safety and quick decision-making are critical.

Overall, this project demonstrates a successful method for enabling autonomous drones to recognize landing scenes intelligently. It lays the foundation for future drone systems that can operate safely and independently in different environments, such as disaster zones, delivery routes, and surveillance missions.

In addition to its technical strengths, the project underscores the broader implications of integrating transfer learning into autonomous systems. By reducing the need for domain- specific data and computational resources, this approach opens new avenues for scaling drone technology to more complex and dynamic environments. The successful implementation of this system paves the way for future innovations in autonomous navigation, showcasing how machine learning can drive advancements in safety, adaptability, and operational efficiency for drones in increasingly unpredictable real-world scenarios.

CHAPTER 10

FUTURE ENHANCEMENT

10. FUTURE ENHANCEMENT

While this project achieved excellent results in recognizing landing scenes using transfer learning and hybrid models, there are several ways it can be further improved and expanded in future work: One major enhancement would be to deploy the model on actual drones and test it in real-world environments. This would help evaluate how well the system performs under changing weather conditions, different lighting, and varying terrain types. Real-world testing is essential for making the model more robust and practical for live deployment.

Another improvement would be to use real-time video input instead of static images. Integrating the model with a live camera feed would allow drones to continuously scan their surroundings while flying and make quick decisions about where and when to land. This would turn the system into a truly real-time autonomous navigation and landing assistant.

The system can also be enhanced by using sensor fusion, where the drone combines camera data with inputs from other sensors like LiDAR, GPS, or ultrasonic sensors. This would give the drone a better understanding of the environment and help detect obstacles, slopes, or uneven surfaces more accurately.

In addition, applying more advanced deep learning models such as Vision Transformers (ViT) or lightweight CNNs like Mobile Net can improve both accuracy and speed, especially on devices with limited processing power. These models may offer better performance while being optimized for edge computing on drones.

Another potential enhancement is to implement self-learning or continual learning methods. These would allow the drone to keep learning from new environments after deployment, adapting to different terrains or situations over time without needing to be retrained from scratch.

Lastly, the system could be extended to recognize not just safe or unsafe landing zones, but also classify terrain types (e.g., grass, concrete, water, rooftop) and choose the most suitable one based on mission needs. This could be highly useful for specialized drone applications like delivery, search and rescue, and military missions.

CHAPTER 11

REFERENCES

11. REFERENCES

1.B Q Li and X H Hu, "Effective distributed convolutional neural network architecture for remote sensing images target classification with a pre-training approach", Journal of Systems Engineering and Electronics, vol. 30, no. 2, pp. 238-244, 2019.

2.G S Xia, X Bai, J Ding et al., "DOTA: a large-scale dataset for object detection in aerial images", Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 3974-3983, 2018.

3.G Cheng, X X Xie, J W Han et al., "Remote sensing image scene classification meets deep learning: challenges methods benchmarks and opportunities", IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, vol. 13, pp. 3735-3756, 2020.

4.A K Anand, S Barman, N S Prakash et al., "Vision based automatic landing of unmanned aerial vehicle", Proc. of the International Conference on Information Technology and Applied Mathematics, pp. 102-113, 2019.

5.C Tian and C B Huang, "An algorithm for unmanned aerial vehicle landing scene recognition based on deep learning and computational verbs", Proc. of the IEEE International Conference on Civil Aviation Safety and Information Technology, pp. 180-184, 2019.

6.C W Lu, E Tsougenis and C K Tang, "Improving object recognition with the l-Channel", Pattern Recognition, vol. 49, pp. 187-197, 2016.

7.B L Zhou, A Lapedriza, A Khosla et al., "Places: a 10 million image database for scene recognition", IEEE Trans.
On Pattern Analysis and Machine Intelligence, vol. 40, no. 6, pp. 1452-1464

8.J X Xiao, K A Ehinger, J Hays et al., "SUN database: exploring a large collection of scene categories", International Journal of Computer Vision, vol. 119, no. 1, pp. 3-22, 2016.

9.G Patterson, C Xu, H Su et al., "The SUN attribute database: beyond categories for deeper scene understanding", International Journal of Computer Vision, vol. 108, no. 1/2, pp. 59-81, 2014.