Agnel Polytechnic, Vashi

A

PROJECT REPORT

On

MARINE WASTE CLASSIFICATION AND DETECTION

Submitted by

Vinit Nemade 22943

Sarvesh Mane 22953

Apurva Satkar 22954

Kaif Sayyad 22955

In partial fulfilment for the award of

DIPLOMA

In

Artificial Intelligence and Machine Learning

UNDER THE GUIDANCE OF

MRS RASHMI DHAKE

FOR THE ACADEMIC YEAR

2024-2025

MAHARASHTRA STATE BOARD OF TECHNICAL EDUCATION
CERTIFICATE

This is to certify that

Mr. Vinit Nemade Roll No 22943

Mr. Sarvesh Mane Roll No 22953

Ms. Apurva Satkar Roll No 22954

Mr. Kaif Sayyad Roll No 22955

of

Sixth Semester

of

Diploma in Artificial Intelligence and Machine Learning

have completed the Project entitled

MARINE WASTE CLASSIFICATION AND DETECTION

satisfactorily for the academic year 2024 to 2025 as prescribed in the curriculum of MSBTE at Agnel Polytechnic, Vashi.

Place: Vashi Enrollment No.:

Date : Exam. Seat No.:

	NAME	SIGN
PROJECT GUIDE	_____	_____
EXTERNAL EXAMINER	_____	_____
INTERNAL EXAMINER	_____	_____
HEAD OF DEPARTMENT	_____	_____

Seal of Institution

ABSTRACT

The rapid growth of sea pollution resulting from the buildup of anthropogenic waste products in ocean and coastal environments is a major threat to marine diversity, environmental stewardship, and human health. Manual observations and cleanups have been inadequate owing to the large extent of impacted areas, poor documentation, and the costliness of human monitoring. As a countermeasure to this increasing crisis, the current research suggests and applies a deep learning-based static image detection system that can recognize marine waste from images through three different convolutional neural network models: Convolutional Neural Networks (CNN), Region-based Convolutional Neural Networks (RCNN), and MobileNetV2. The goal of this project is to develop an affordable, computationally lightweight, and user-friendly tool for automating the detection of marine waste without depending on drone monitoring or real-time video streaming technology.

The focal point of this research is centered on the development, training, testing, and comparative analysis of the three above-mentioned deep learning models. The dataset utilized consists of images with labels illustrating different marine ecosystems, polluted as well as unpolluted, containing a wide range of wastes such as plastics, fishing nets, and wreckage. Images were preprocessed and normalized to the standard size, and models were trained with supervised learning using correct loss functions and optimizers for proper convergence. The CNN model was implemented with a sequence of convolutional, pooling, and fully connected layers optimally tuned for feature extraction from environmental images. The RCNN model used selective search for the generation of region proposals, giving more detailed bounding box predictions at the expense of increased computational complexity. The MobileNetV2 model, a light-weight variant of CNN, was used to investigate the possibility of deploying waste detection systems on low-resource devices like mobile phones or edge devices.

All models were tested on various performance metrics such as classification accuracy, precision, recall, F1-score, and computational overhead. Experimental outcomes showed that the CNN architecture performed better compared to the rest in raw accuracy, providing highest confidence levels both in waste detection and classification with relatively efficient inference times. RCNN showed enhanced localization due to its application of region proposals and was therefore suited for applications that require precise spatial annotation. Nonetheless, its increased processing duration and bigger memory usage made it less efficient in lightweight deployments. Conversely, the MobileNetV2 model presented considerably lessened resource usage and quicker inference periods, albeit at a modest drop in detection accuracy. Such trade-off positions MobileNetV2 as aptly suited to tasks where model portability and minimal power consumption take precedence.

ACKNOWLEDMENT

Perseverance, Inspiration & Motivation have always played a key role in the success of any venture.

At this level of understanding it is difficult to understand the wide spectrum of knowledge without proper guidance and advice, hence we take this to express our sincere gratitude to our respected **Project Guide Mrs. Rashmi Dhake** who as a guide evolved an interest in us to work and select an entirely new idea for project work. She has been keenly co-operative and helpful to us in sorting out all the difficulties.

We would also like to thank our **HOD Mrs. Raji M. P**. and **Principal Mrs. Saly Antony** for their continuous advice and support.

My deep sense of gratitude **Technotorc Tools pvt ltd** to for their timely advice and encouragement in our project development.

I would also thank my Institution and my faculty members without whom this project would have been a distant reality.

VINIT NEMADE 22943

SARVESH MANE 22953

APURVA SATKAR 22954

KAIF SAYYAD 22955

INDEX

CONTENTS **PGNO**

List of figures

LIST OF TABLES

TABLE NO PG NO

CHAPTER 1
INTRODUCTION INDUSTRY BACKGROUND

Strategic Partnership with Technotorc Tools Pvt. Ltd.

1.1 Blending Industrial Acumen with AI-Driven Environmental Intelligence

About Technotorc Tools Pvt. Ltd.

Technotorc Tools Pvt. Ltd. is an icon of industrial innovation, engineering prowess, and customer-oriented solutions in India's heavy machinery industry. Based outside Mumbai, Technotorc has evolved to become the country's top producer and exporter of bolting, lifting, and onsite machining machinery. With a wide range of products that spans hydraulic torque wrenches, high-pressure pumps, bolt tensioners, manual and electric torque tools, flange spreaders, nut splitters, jacking systems, and precision onsite machining solutions, Technotorc's products are the building blocks of high-load mechanical operations in key industries.

Technotorc's reach is across industries like Oil and Gas, Power Generation, Renewable Energy (Wind and Hydro), Mining, Automotive, Marine Engineering, Railways, and Infrastructure Projects. These applications require outstanding tool reliability and safe operation, in which a single point of failure can lead to enormous cost, downtime, or human risk. Realizing this, Technotorc has spent heavily on world-class CNC production equipment, rigorous in-house quality assurance, and R&D-driven product development to ensure their tools provide industry-leading performance, accuracy (up to ±3%), and longevity.

Aside from production, Technotorc also provides 24/7 onsite support, tool rentals, maintenance, calibration, and technical advice—placing the company not just as a product supplier but as an industrial operations partner in the long term. This integrated strategy shows Technotorc's vision to not only provide industries with tools but to also guarantee maximum execution, safety, and reliability of mechanical processes, particularly in mission-critical applications such as offshore rigs, refineries, and high-capacity plants.

Why Our Project Is Important to Technotorc

With the world moving towards greener, more sustainable industrial processes, there is mounting pressure on manufacturers and service providers to give back to the environment without sacrificing productivity. Technotorc, with its strong heritage in heavy engineering and increasing presence in marine-facing sectors, is ideally placed to spearhead this twin agenda—industrial precision and adoption of technologies that support environmental awareness.

Our Custom Deep Learning Powered Marine Waste Detection System is intended to identify, classify, and track marine litter based on visual imagery. This computer vision-

enabled AI solution combines image classification pipelines and computer vision to classify oceanic and near-shore images—detecting plastic, metal, rubber, organic trash, and other contaminants with great precision. With real-world marine dataset training and solid algorithmic design, the system is designed for real-time or batch-level deployment in operational as well as monitoring scenarios.

By implementing this solution, Technotorc can incorporate AI-driven waste monitoring across different verticals of their operations and customer services:

1.2 How We Can Integrate with Technotorc's Ecosystem

1. Offshore Tool Deployment & AI Pairing

Technotorc equipment—particularly jacking systems, torque wrenches, and cold-cutting machines—is commonly used in coastal and offshore operations. These environments are extremely sensitive to waste creation during maintenance or demolition activities. Our system can be integrated with Technotorc's equipment or service processes to track and report waste during field operations, maintaining environmental compliance and minimizing post-operation cleanup liabilities.

2. Smart Maintenance for Marine Clients

Most of Technotorc's customers engage in maritime logistics, ship repair and building, and port maintenance. Our framework can be used as an add-on service—providing AI-based surveillance of the ocean for spotting rubbish around docks, platforms, and ships. This introduces a new value-added model of service for Technotorc, broadening its brand into AI-driven smart maintenance and inspection.

3. Visual Reporting for Regulatory Compliance

As the environmental regulations tighten all over the world, Technotorc and its customers frequently need visual records and compliance confirmation. Our model can be utilized to automatically create marine waste detection reports, supporting Technotorc's field service reports with environmental compliance insights—giving both a regulatory and competitive edge.

4. AI Integration in Technotorc's Rental & Service Toolkit

The waste detection system can be packaged as part of Technotorc's on-site tool rental packages or field inspection kits. Camera-based, lightweight modules can be used to take operational zones prior to and after tool deployment, providing input into the detection model. This not only promotes responsible operation but also makes Technotorc a brand that incorporates advanced AI oversight into conventional engineering systems.

5. Data-Driven Expansion Opportunities

With logs of waste detection and geotagged pollution maps created from the outputs of AI, Technotorc can amass valuable environmental information. It can assist in the creation of sustainability reports, client advisory services, or alliances with marine

environmental agencies, projecting Technotorc beyond engineering to environmental intelligence.

1.3 Shared Vision: Innovation with Responsibility

The collaboration between Technotorc and our AI-assisted marine waste detection program signifies more than technology convergence—it's a declaration of forward-thinking industrial principles. Through our combined efforts, we can show the world how conventional heavy engineering can adapt by integrating artificial intelligence, automation, and environmental responsibility—enabling industries to perform with mechanical effectiveness and environmental integrity.

With the help of Technotorc's expertise and international footprint, this project has the potential to have a quantifiable environmental impact, while assisting industries to update their strategy in waste disposal. It's not merely a step ahead for machine intelligence—but a stride toward greener seas, sustainable industries, and a brighter future.

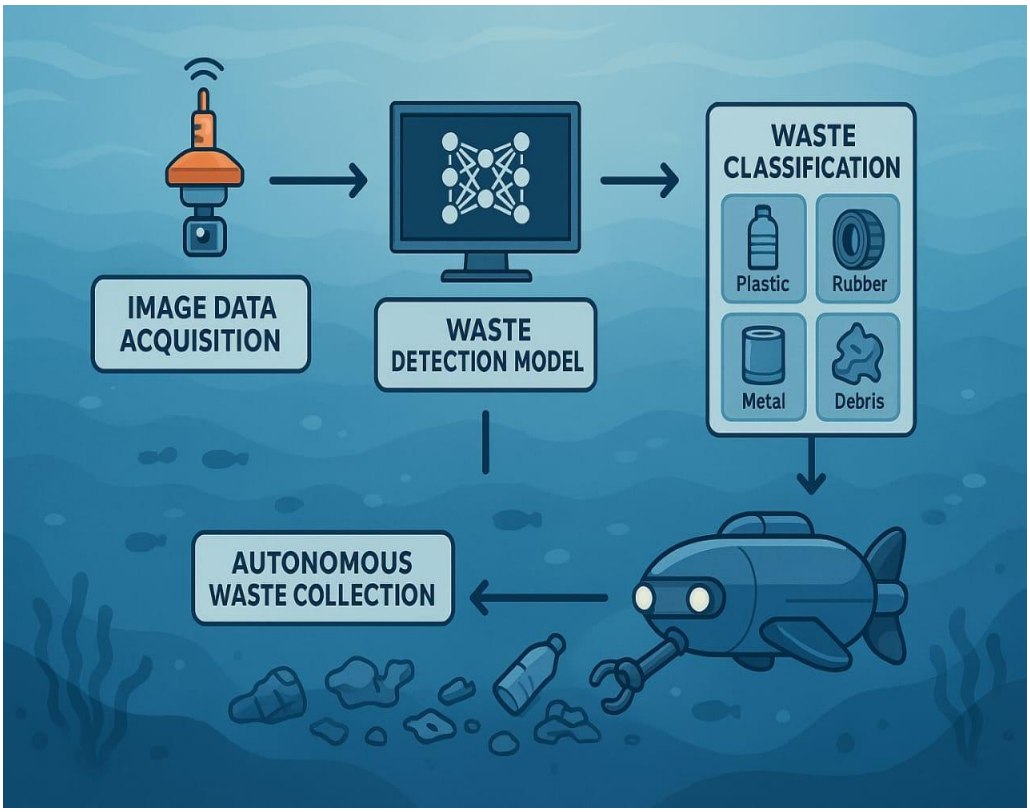


Fig.1 marine based model basic working

CHAPTER 2
LITERATURE REVIEW

2.1. Introduction

Marine waste pollution and the dissemination of plastic waste, in general, has emerged as an acute and emergent environmental issue that is endangering marine biodiversity, and aquatic systems at large. A majority of the waste has been found to originate from terrestrial sources, which thereafter get transported to the oceans by processes involving rivers, canals, and other water bodies that drain into large water bodies. With the all-pervasive process of industrialization and urbanization across the world, the quantity of marine waste is escalating exponentially at an alarming rate, thus necessitating the development and deployment of novel means for the detection and elimination of the same. This thorough literature review will attempt to analyze the development of methods used for the detection and elimination of marine waste, particularly with reference to the recent developments involving the employment of autonomous vehicles, cutting-edge sonar tools, remote sensing processes, computer vision software, and deep learning processes. It also examines the current challenges in the same, surveys new solutions being advocated, and discusses the key role played by interdisciplinary association in the constant quest to counter the menace of marine waste in an effective manner.

2.2. Conventional Methods and Approaches for Marine Waste Detection

Historically, the removal of marine litter has been dependent on the use of manual methods that involve the use of an array of techniques, varying from the use of skimmer boats, divers' skills, to local community clean-up campaigns. These activities are normally arranged and conducted by non-governmental organizations and enthusiastic volunteers interested in environmental conservation. While effective for small-scale operations or for specific locations, such conventional methods are beset with a number of inherent disadvantages; that is, they are time-consuming, manpower-intensive, and can be extremely hazardous, especially in highly polluted or inaccessible settings. The constraint of manual detection is also usually further compounded by issues such as poor underwater visibility, impacts of tides and currents, and human divers limited physical capacity in such a setting. Additionally, the general success of such clean-up campaigns is also greatly dependent on factors such as the availability of finances, support from local communities, and favorable environmental conditions, none of which can always be guaranteed. Additionally, despite the use of technologies such as nets and surface skimmers that have been employed to retrieve floating litter, such technologies do not effectively confront the issue of submerged litter, which becomes a major issue. This limitation has spurred growing interest and research into automated and robot-based systems that can effectively operate in challenging aquatic settings without the need for large-scale human intervention to be effective.

2.3. Sonar-Based Detection with Autonomous Underwater Vehicles (AUVs)

A crucial advancement in submerged marine waste identification is presented by the work of Valdenegro-Toro (2016), who presented a system involving Autonomous Underwater Vehicles (AUVs), Forward-Looking Sonar (FLS), and Convolutional Neural Networks (CNNs). The system applies the high spatial resolution of FLS to obtain fine sonar imagery, allowing CNNs to detect and classify underwater debris with high accuracy. The CNN was trained on a collection of different common varieties of debris, including metal, plastic, rubber, glass, and cardboard materials. The design provided a correct detection rate of 80.8% for binary classification and 70.8% for multi-class classification, which indicates its potential for use in real situations. The solution uses a sliding window technique for scanning FLS images and detecting debris, which is well-suited for detecting objects of small sizes that are otherwise difficult to detect using traditional sonar technologies such as sidescan or synthetic aperture sonar. Importantly, the work demonstrated the ability of the CNN to generalize to unfamiliar types of objects, making it a practical solution for real-world underwater situations where diversity in debris is high. The system also opened avenues for future extension with AUVs coupled with manipulators to pick up debris, indicating the potential for scalable and systematic underwater clean-up activities.

2.4. Visual Detection in Inland Waters Using USVs

While AUVs are ideal for submerged debris, floating waste detection in inland waters—such as rivers, lakes, and canals—requires a different approach. Cheng et al. (2021) addressed this need through the creation of the FloW dataset, a pioneering benchmark for floating waste detection using Unmanned Surface Vehicles (USVs). FloW is composed of two main subsets: FloW-Img, an image-based dataset with over 5,000 annotated instances of floating debris, and FloW-RI, a multimodal dataset combining visual imagery and 77GHz millimeter wave radar data. The dataset was collected under diverse environmental conditions, encompassing various waste types, lighting scenarios, and water surface reflections. This diversity makes FloW a critical resource for training and evaluating vision-based detection models under realistic conditions. The study also addressed major challenges in visual detection, such as the small size of floating objects, occlusion, and confusion with non-waste materials like leaves or natural flotsam. The inclusion of radar data in FloW-RI enables sensor fusion approaches that compensate for visual limitations, particularly in poor lighting or reflective conditions.

2.5. Applications of Deep Learning and Computer Vision Technologies for Waste Detection

With the emergence of efficient deep learning methods, the subject of marine waste detection has seen a dramatic shift. Walia and Pavithra (2024) performed an in-depth comparative analysis of the latest object detection models such as YOLOv4, YOLOv5, YOLOv6s, YOLOv7, YOLOv8 (nano and XLarge version), Faster R-CNN, and Mask R-CNN, which were experimented with on a wide variety of underwater and waste-specific datasets. The research concluded that YOLOv8-nano and YOLOv8-XLarge produced high mean Average Precision (mAP) scores of up to 0.96 with precision and recall values of over 0.93 in some cases. These models performed extremely well in real-time application due to their optimized design and anchor-free detection. Two-stage

detectors such as Faster R-CNN and Mask R-CNN performed slightly better in complex detection tasks but were hindered by increased inference times and computational expenses. The review points out that the choice between one-stage and two-stage models has to be made in accordance with the particular operational limitations, including hardware capacity and response time demands.

2.6. Approaches to Radar and Sensor Fusion Techniques

Radar sensing technology, and in particular the revolutionary millimeter wave radar, has become increasingly popular as a useful complementary technology to conventional visual detection mechanisms. One of the largest benefits of radar technology is that it is far less sensitive to varying lighting conditions and water surface reflections, which normally degrade the performance of visual sensors. This consistency allows radar to provide stable distance and velocity measurements, making it a vital component in many applications. In the FloW dataset example, the presence of synchronized radar and imagery data allows for the training of complex fusion models that effectively leverage both modalities to improve object detection accuracy and consistency. These advanced models are especially beneficial in adverse conditions that are difficult for optical sensors, such as in the case of excessive rain, heavy fog, or blindingly bright sunlight. Fusion of radar technology with vision systems through the implementation of this method allows for the creation of robust detection pipelines that can effectively counter a large number of varied operational challenges that can occur. It is, however, essential to realize that efforts towards successfully combining these various modalities are in their infancy and require additional investigation and research to be able to leverage its full potential.

2.7. Autonomous and Robotic Removal Systems

There have been several real-world projects that have successfully demonstrated the feasibility and practicality of incorporating detection and removal systems in total environmental cleanup initiatives. One notable example of this is the Floating Robot for Eliminating Debris, or FRED, creatively designed by the Clear Blue Sea organization. This innovative robot is an underwater cleanup robot par excellence that is a great example of the utilization of Unmanned Surface Vehicles, or USVs, with advanced, AI-based navigation and collection systems for the removal of debris in water bodies. Similarly, The Rozalia Project utilizes the Remotely Operated Vehicle, or ROV, named Hector for the sole purpose of deploying underwater cleanup operations. This is especially important in environmentally sensitive ecosystems like coral reefs, which need to be handled with care and sensitivity. These devices are carefully designed with utmost caution so as not to disturb the environment in any possible way while at the same time optimizing the efficiency of debris collection mechanisms. They also highlight the inherently interdisciplinary nature of the discipline, which demands collective efforts by stakeholders in robotics, environmental science, and artificial intelligence. This collective expertise is essential in coming up with viable and sustainable solutions for pollution mitigation and the preservation of our natural ecosystems.

2.8. Datasets and Benchmarking

One of the biggest hurdles to advancement in the area of marine waste detection is the glaring absence of standardized, large-scale, and diverse datasets. Most of the available datasets are highly specific and specific to specific environmental conditions, which in turn lowers their overall utility in the creation of generalized models that can be utilized effectively in a wider range of scenarios. However, several datasets, including TrashCAN, JAMSTEC, TACO, and FloW, have begun bridging this important gap by providing extensive annotations, including multiple classes of waste and supporting diverse aquatic conditions. These datasets are very crucial because they enable comparative benchmarking of various detection algorithms, thereby enabling the identification and selection of the best-performing models under various scenarios and environments. Moreover, the creation of customized datasets, which is made possible by data augmentation techniques like flips, rotations, and color changes, goes a long way in enhancing the robustness and usability of the models in a diverse range of environments.

2.9. Future Directions

Despite tremendous progress, there are many areas that require improvement. There is a pressing need for models with the capability to generalize across different environments, types of debris, and sensor modalities. Low-weight models with high accuracy and low latency are crucial for real-time execution on mobile platforms such as AUVs and USVs. Development is crucial in fusing sonar, radar, and optical data for an end-to-end perception system capable of functioning well under varying aquatic conditions. Economical deployment scenarios for AUVs and USVs in large-scale operations are a recurring challenge in terms of costs and resources. All systems, further, must be so designed as not to inflict significant harm to sea life and further noise and pollution to the water. Resolving these issues will be critical to long-term sustainability and environmental adaptability of marine waste detection systems.

CHAPTER 3

SCOPE OF THE PROJECT

3.1 Scope: A Unified Vision for Intelligent Marine Waste Detection

Marine ecosystems are some of the most productive and biodiverse habitats on Earth, with a critical role in global climate regulation, fisheries support, and livelihoods for billions of people. Yet these ecosystems are under an increasing threat from human activities—most significantly, the widespread proliferation of marine debris. Made up mainly of non-biodegradable forms such as plastics, metals, rubber, and glass, marine trash pervades oceans, coastlines, and river mouths, interfering with ecological equilibrium, endangering marine life, and seriously affecting the health of human communities.

The need to tackle this crisis cannot be overemphasized. Research has indicated that marine litter kills more than a million seabirds and more than 100,000 marine mammals annually. Additionally, the presence of microplastics in the food chain has caused alarm regarding long-term public health impacts. However, despite the common knowledge, current monitoring and detection strategies are finding it difficult to keep up with the extent and sophistication of marine pollution.

3.2 The Core Problem

Existing techniques for the detection and identification of ocean waste are usually through manual beach surveys, diver surveys, or satellite imagery, which are either resource-intensive, expensive, or constrained by resolution and weather. Not only are such techniques inefficient, but they also suffer from the risk of human error, leading to under-detection or misidentification of marine trash. There is an urgent need for a scalable, intelligent, and automated system that can function across the various marine ecosystems—offering real-time, precise, and actionable information to inform targeted interventions and long-term policy-making.

To fill this gap, our project presents a robust machine learning–driven Marine Waste Detection and Classification System. Developed on cutting-edge AI models like YOLOv5 (object detection) and Random Forest (classification), the system can detect and classify marine debris from static images as well as live video streams with high accuracy. It not only improves detection accuracy but also reduces significantly the time, effort, and cost of marine waste monitoring operations.

System Functionality and Technological Framework

The system operates through a chain of computer vision pipelines. Input data—obtained through coastal cameras, AUVs (Autonomous Underwater Vehicles), or remotely operated underwater sensors—is processed first to isolate candidate debris through YOLOv5's real-time object detection. The identified regions are subsequently analyzed through a Random Forest classifier to classify the type of waste (e.g., plastic bottles, fishing nets, rubber fragments, cans, etc.).

Key characteristics of the system are:

- Near real-time high-speed inference for monitoring
- Performance under diverse light and turbidity
- Scalability, allowing integration with cloud dashboards or on-premise embedded systems
- Geo-tagging capability for spatial analysis and temporal trends
- Continuous learning pipeline for improving accuracy with data over time
- Implementation Scope and Impact Potential

At full-scale deployment, the system presented has revolutionary advantages in the detection of marine waste. The tool can be utilized by environmental agencies for bulk monitoring of hotspots of pollution to facilitate focused cleanups and enforcement checks. Research organizations can leverage the data collected to study seasonal patterns and long-term ecological effect. Policy makers can use high-fidelity, ground-truth data as a reference for implementing region-based waste detection regulations.

In addition, the system affords industries—particularly shipping, port authorities, fisheries, and tour operators—a concrete vehicle to track their environmental impact and comply with sustainability rules. For example, port officials can implement intelligent surveillance systems to guarantee waste-free docking facilities, while tourism commissions can utilize live data to ensure cleanliness of marine parks and seaside attractions.

Furthermore, integration with autonomous marine robots creates new frontiers. Underwater robots or floating waste-gathering bots can be programmed to react to AI signals, making autonomous waste cleanup missions possible in harbors, reefs, and river deltas. This automation minimizes the risk to human divers and greatly decreases long-term operational expenses.

3.3 Broader Applications and Future Adaptability

Although initially targeted for marine ecosystems, the inherent detection algorithms are made with flexibility in mind. The same models can be adapted to identify litter in freshwater systems like rivers and lakes, urban waterways, and stormwater runoffs. This flexibility increases the system's applicability across geographies and ecosystems, even for landlocked nations with inland water pollution.

Further, with modest modifications, the system can be used in neighboring environmental areas, including:

- Microplastic hotspot identification
- Marine biodiversity enumeration and anomaly detection
- Waste sorting and recycling facility optimization
- Illegal dumping monitoring in coastal areas
- Oil spill detection and tracking

All these turn the project from being an individual solution into a launchpad for more general AI-for-environment applications.

3.4 Educational and Collaborative Scope

The information gathered, tagged, and processed by the system can be used to drive research in academic marine sciences, AI, and environmental policy. It provides a unique training ground for interdisciplinary education, which is a fusion of ecological literacy and computational thinking. Researchers and students will have access to real datasets, experiment with hypotheses, and develop enhanced models, hence empowering a new generation of conservation technologists.

In addition, by encouraging collaboration between government agencies, non-profit conservation organizations, universities, and industry partners, the project establishes an open innovation environment. Public awareness, community science engagement, and emergency alert systems for contaminated areas can be created using real-time dashboards—making environmental intelligence accessible to everyone.

CHAPTER 4

METHODOLOGY

Step 1: Briefly Describing the Problem and Expounding the Objectives

The growing volumes of waste that are found within marine ecosystems are creating serious and substantial issues for the health and equilibrium of aquatic ecosystems as well as for the high biodiversity of these critical habitats. In order to combat this dire problem, the application of automation in detecting marine waste using the implementation of advanced deep learning models can significantly help towards supporting both environmental monitoring as well as the subsequent clean-up operations necessary to correct these ecosystems. The primary objective of this project is to create and thoroughly evaluate three different machine learning models—specifically CNN (Convolutional Neural Network), RCNN (Region-based CNN), and MobileNetV2—specifically with the view of detecting and classifying different types of marine waste found within images with high accuracy. This intricate task can be systematically reduced into two simpler sub-tasks: the first one being classification, which identifies whether or not the subject image in question exists with waste, and the second one being localization, which attempts to identify the precise location of the waste within the subject image itself.

Step 2: The Dataset Collecting and Annotation Process

For initiating implementation, a good quality and diversified dataset should be gathered. The dataset should contain images of oceans, beaches, and marine regions, clean and polluted.

Data Sources: Data sets can be retrieved from internet archives, environmental organizations, or field photography.

Annotation for MobileNetV2 and CNN: The images are annotated in two classes: “Waste” and “No Waste.” The images are classified as binary by these models.

Annotation for RCNN: The RCNN approach makes use of bounding box annotations that properly identify the exact locations of the waste objects in the images under analysis. The use of various tools such as LabelImg or CVAT helps to easily draw bounding boxes around the waste objects. After the bounding boxes are created, the tools enable export of the annotation files in widely used formats like XML or CSV, depending on the various systems in use.

Organizing data: The images are organized in a systematic manner along with their respective labels into folders. The folders are usually split into three subsets, which are commonly known as training, validation, and test subsets.

Step 3: The Image Preparation Process

Preprocessing is necessary to ensure that there is consistency throughout the whole dataset, which in turn ensures that it is compatible with the unique input formats that are required by neural networks.

Reading the Image: OpenCV's function, i.e., the cv2.imread function, is used to read all the images directly from their respective given file paths.

Color Channel Conversion: Because OpenCV opens images by default in BGR format, images are thus converted to RGB through cv2.cvtColor.

Resizing: Each picture is resized properly to a consistent size of 224x224 pixels so that it meets the input size criteria specified by the CNN as well as MobileNetV2 designs.

Normalization: Pixel intensities are scaled to the [0, 1] interval by dividing them by 255. Improved training convergence.

Dimension Expansion: The images are expanded to 4D tensors with np.expand_dims() for matching the model's prediction expected input shape.

Step 4: Development of Model Architecture

A.Tailored CNN Model

Layers: CNN's architecture has a number of convolutional layers for feature extraction, followed by max-pooling layers to downsample, a flattening layer to transform 2D features into 1D, and dense layers for classification.

Activation Functions: The Rectified Linear Unit, or ReLU, is generally used in the hidden layers of a neural network. The Sigmoid activation function is used in the last layer if the problem being solved is binary classification.

The model provides a value that is a measure of a probability of whether or not the given image contains waste.

B. RCNN (Region-based Convolutional Neural Network)

Region Proposal: uses Selective Search to produce a list of multiple region proposals from the input image.

Feature Extraction: During this step, each region is individually processed through the analysis of a convolutional neural network backbone like VGG16 or ResNet50 for the extraction of multiple salient features.

Classification is the act of feeding the characteristics through a classifier. This is performed with the purpose of establishing whether these characteristics contain waste material or not.

Bounding Boxes: Boxes are deliberately sketched around some areas that are most likely to harbor waste material in this process. This is done using definite thresholds, like the threshold value of 0.5, to precisely locate these areas.

C. MobileNetV2 (Pretrained Model)

Transfer Learning: The MobileNetV2 model is pre-trained on the big and common ImageNet dataset and is successfully employed as a feature extractor in a variety of applications.

Top Layer Replacement: The final layer of classification is replaced with dense application-specific layers optimized for the marine waste classification task.

Freezing Layers: In the training process, the lower layers of MobileNetV2 are deliberately frozen to preserve and retain the useful features acquired up to now. Meanwhile, the newly introduced top layers are trained to acquire and adapt according to the data.

Lightweight: MobileNetV2 possesses an amazing level of computational efficiency, making it particularly well-suited for use in embedded systems or mobile platforms where resources are likely to be limited.

Step 5: Model Training Process

Each model is trained individually, with specially 13penCV13d training data.

Loss Function: Binary cross-entropy is used across all the models since it's a binary classification problem.

Optimizer: Adam optimizer is used due to its adaptive learning rate and the best convergence.

Batch Size and Epochs: Standard setups use a batch size of 32 with 10–25 epochs. Early stopping is used to avoid overfitting.

Validation: A validation set serves the important function of being utilized to track and assess the performance of the model regularly during the duration of the training process.

Checkpointing: Best weights are saved automatically using callbacks in Keras.

Step 6: Saving and Loading Models Process

Once the training period is completed, the model's weights and architecture are safely archived in a file called.h5.

model.save('models/trained_models/model_name.h5')

In forecasting, these pre-trained models are loaded with:

model = tf.keras.models.load_model('models/trained_models/model_name.h5')

Step 7: Drawing Inferences and Making Predictions using the Pipeline

Every single model has its own unique specific prediction function that is specifically tailored for its application.

Image Loading: The image is loaded and preprocessed.

Prediction Execution: The input image is then passed on to the model to process and analyze it.

Confidence Evaluation: Where there are binary outputs, we use a defined threshold value like 0.5 to effectively decide and identify which predicted category needs to be allocated to the output.

When the confidence is determined to be high, a bounding box along with a label is generated and plotted using the OpenCV library.

Return Structure: The function is coded to return an elaborate list consisting of the detected class, detection confidence level, and bounding box coordinates (where applicable). It also returns the output image relevant to the processing.

Step 8: Detailed Comparison and In-Depth Evaluation

Each of them undergoes a rigorous test evaluation process through a particular test set.

Metrics: Accuracy, precision, recall, and F1-score are computed.

Comparison:

CNN achieves the best classifying accuracy when compared to alternative approaches.

RCNN provides object localization but at the expense of slower inference.

MobileNetV2 is a balance between performance and efficiency.

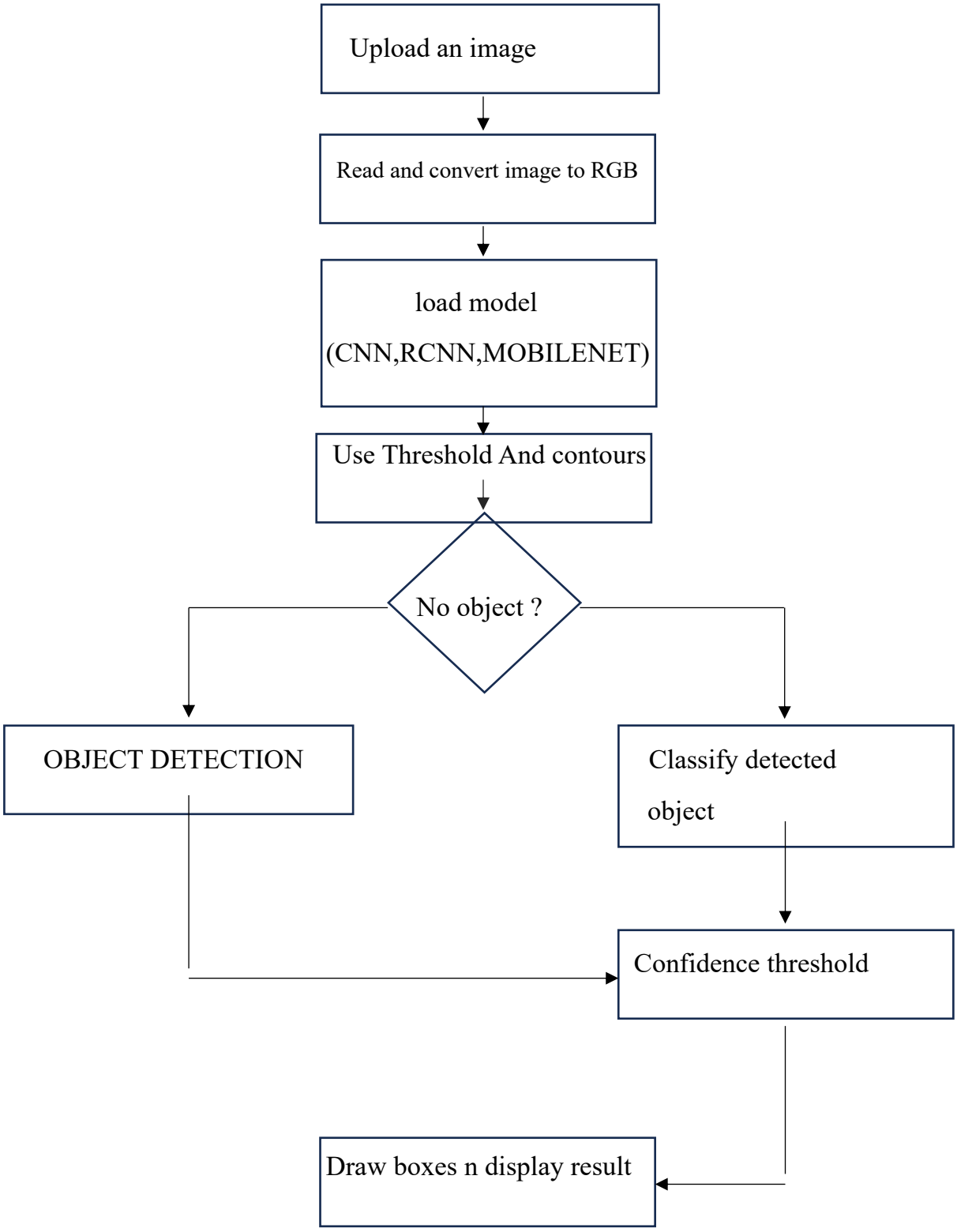


Fig.2 work flow

CHAPTER 5

DETAILS OF DESIGN, WORKING AND PROCESS

5.1 Marine Waste Detection System — Full Execution Flow Documentation

CNN

5.1.1 Environment Setup

Importing Essential Libraries

The program starts by importing a list of fundamental libraries necessary for computer vision, deep learning, numerical operations, and debugging. Cv2 from the OpenCV library is utilized for all image processing operations like reading, converting, and manipulating images. Numpy is utilized to deal with arrays and image matrices, particularly for preprocessing and resizing. TensorFlow's load_model function is imported to import pre-trained deep learning models. Warnings is utilized to suppress non-critical runtime warnings that fill output. Os assists in dynamically handling file paths based on the system environment. Finally, logging is utilized to keep a systematic record of all key operations, both normal and erroneous events.

Logging Configuration

Logging is configured with two handlers: a StreamHandler that writes logs to the console and a FileHandler that writes logs to a persistent file (detect_waste.log). Having both real-time and historical logs available means developers or system administrators can trace both current and past events during error diagnosis or performance analysis. The format adds timestamps, severity level, and a message per event, which enhances traceability when diagnosing errors or analyzing performance.

5.1.2. Model Detection

Global Model Caching

The script employs a global dictionary named loaded_models for caching models once loaded to improve performance and minimize unnecessary disk access. This prevents reloading the same model each time detection is executed, which accelerates repeated execution considerably, particularly in server environments.

Function: load_model_safely()

This function is designed to load models safely and efficiently. It first looks to see if the needed model type (e.g., 'cnn', 'rcnn', 'mobilenetv2') is already cached. If it is, it logs this and returns the model immediately. If not, it builds out a full file path by using BASE_DIR (the full script location path) and adding trained_models/<model_type>_model.h5 to it. The function then tries to load the model with tf.keras.models.load_model(). Upon loading, it trains the model with the Adam optimizer, binary_crossentropy loss (appropriate for binary classification problems), and accuracy as the metric. In case the file does not exist or if there is a failure in any

step of the process, the function logs the exception and raises a corresponding error with complete stack trace. This makes the design robust and transparent in deployment.

5.1.3. Image Loading and Preprocessing

Image Path Input

When an image (in our example, "waste20-Copy.jpg") is chosen from the interface by the user and the detection begins, the script reads the path from the image and feeds it into the function detect_waste().

Image Reading with OpenCV

Within detect_waste, the image is read from disk as cv2.imread(). In the event that the image does not load (such as because of a faulty path, unsupported file type, or corrupted file), the script generates a helpful error. On a successful load, it logs the shape of the image, with success.

Color Space Conversion

Images are read in BGR format by OpenCV, but most deep learning models trained with Keras/TensorFlow operate in RGB format. Thus, the script converts the image from BGR to RGB using cv2.cvtColor(). This ensures pixel color channels are in the right order before further processing and model inference.

5.1.4. Object Detection

Primary Detection using Thresholding and Contours

The method detect_objects() uses traditional image processing methods to identify possible objects. The RGB image is converted into grayscale as the first step in order to ease the data by converting it into a single channel. The image is then smoothed and low-frequency noise eliminated to prevent it from interfering with thresholding by using a Gaussian blur with a 5x5 kernel.

Adaptive Gaussian thresholding is then used in the script. This works extremely well on unbalanced lighting conditions, because it computes the thresholds for individual small areas separately and transforms the image into black and white, binary format which shows object boundaries prominently.

Contours are found in the binary image. A contour is a single boundary or region of the image which may encompass an object. The script computes bounding rectangles around every contour and filters them according to their area ratio — the rectangle size over the image area. Contours that are within a certain size range (0.1% to 90% of the image) are deemed valid detections. This prevents ignoring noise (very small contours) and outliers (very large regions such as the whole background).

Fallback Detection using Edge Detection

If no viable objects are located using thresholding, the script resorts to an alternative method in the form of the Canny Edge Detector. Canny points out object borders by identifying rapid gradients. Contours are extracted once more and filtered by area. This

helps the system to continue trying to locate objects within images where thresholding is failing because of conditions of lighting or texture.

5.1.5. Object Classification

Extracting and Preprocessing Regions of Interest

Each bounding box is utilized to crop a sub-image (Region of Interest or ROI) out of the original RGB image. If the region is invalid or null (for example, due to erroneous coordinates), it is skipped with a warning message. Valid ROIs are resized to 224x224 pixels — the typical input size for most CNN architectures such as MobileNetV2 and custom CNNs. These resized images are normalized by pixel values divided by 255.0 to rescale them to between 0 and 1 to keep it consistent with the training data of the model. Each processed ROI is then padded to be in the shape required by the input of the model: [batch_size, height, width, channels].

Prediction and Classification

All the preprocessed ROI is passed on to the model of choice to predict. It generates a confidence value, which is a floating-point number ranging between 0 and 1. The code treats this as a probability the ROI is actually a Waste object. It follows the threshold: if confidence > 0.5, mark as Waste, otherwise as Non-Waste. To ensure stability, predictions less than 0.2 are disregarded. This prevents rendering noise due to questionable predictions.

Drawing Bounding Boxes

The detected objects are marked in the output image with bounding rectangles. The Waste is marked with green boxes and Non-Waste is marked with blue boxes. Every detection is also saved with metadata such as object ID, bounding box coordinates, class, and confidence score in a list, which can be used later to display summaries or visualizations.

5.1.6. Visualization and Output

Annotated Output Image

The final output image (img_with_boxes) is a copy of the original image with bounding boxes drawn around every high-confidence detection. These boxes enable users to visually check what area of the image the model has detected as waste or non-waste objects.

Results Summary

The results, maintained in a structured list, consist of:

- Total number of detected objects.
- Number of Waste-classified objects.
- Number of non-waste-classified objects.

This summary is employed to fill out the detection counters in the user interface.

5.1.7. Dashboard and Interface

User Interface Components

On the graphical dashboard (refer to the screenshot above):

- Left panel contains controls to choose models (CNN, R-CNN, MobileNetV2), view abstract, and upload images.
- Middle panel has detection output, i.e., annotated image and object numbers.
- Right panel shows a bar graph presenting the distribution of Waste vs Non-Waste objects.

The design supports users in interactively examining how various models function and estimating model performance qualitatively and quantitatively.

5.1.8. Evaluation Metric — Intersection over Union (IoU)

Function: calculate_iou()

This utility function computes the IoU metric — an common evaluation metric in object detection tasks. It is the overlap between two bounding boxes as a fraction of their union area. The IoU score quantifies how well bounding boxes predicted match ground truth labels (helpful when validating a model or comparing on a benchmark). High IoU reflects good localization.

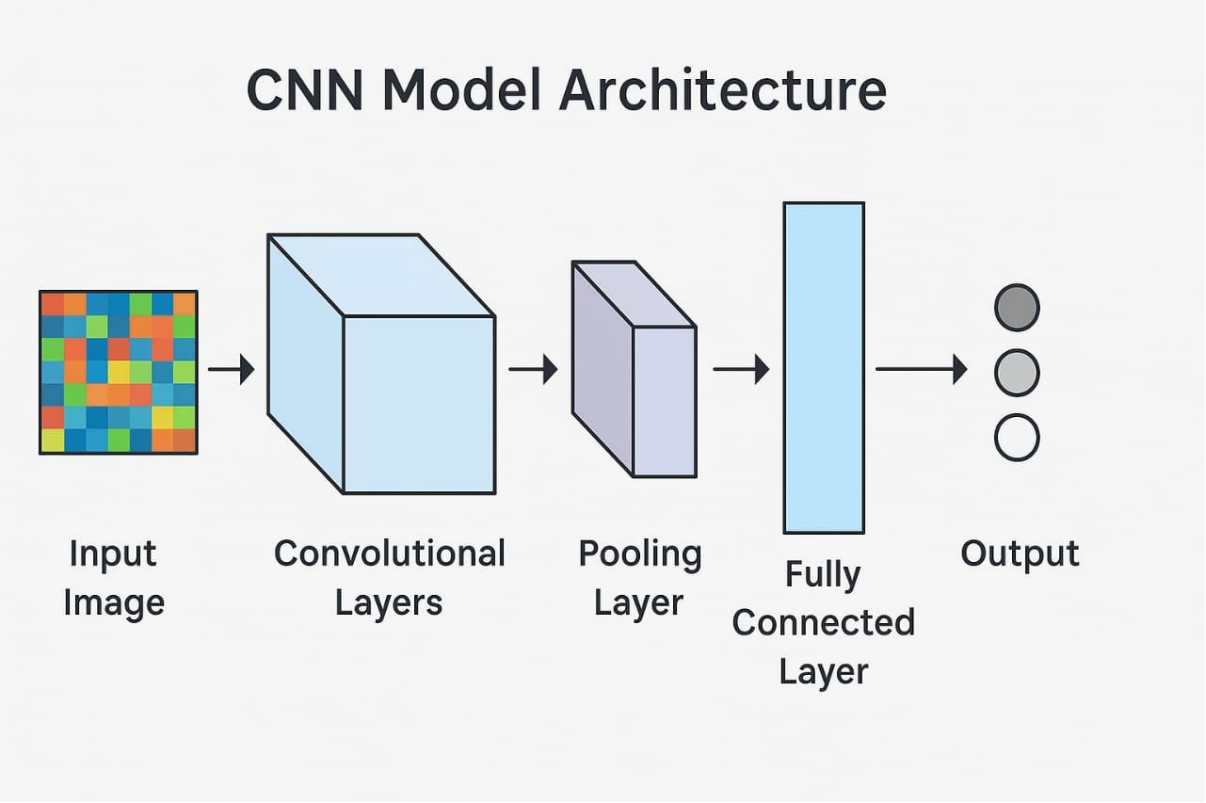


Fig 3 CNN Architecture

OUTPUT:

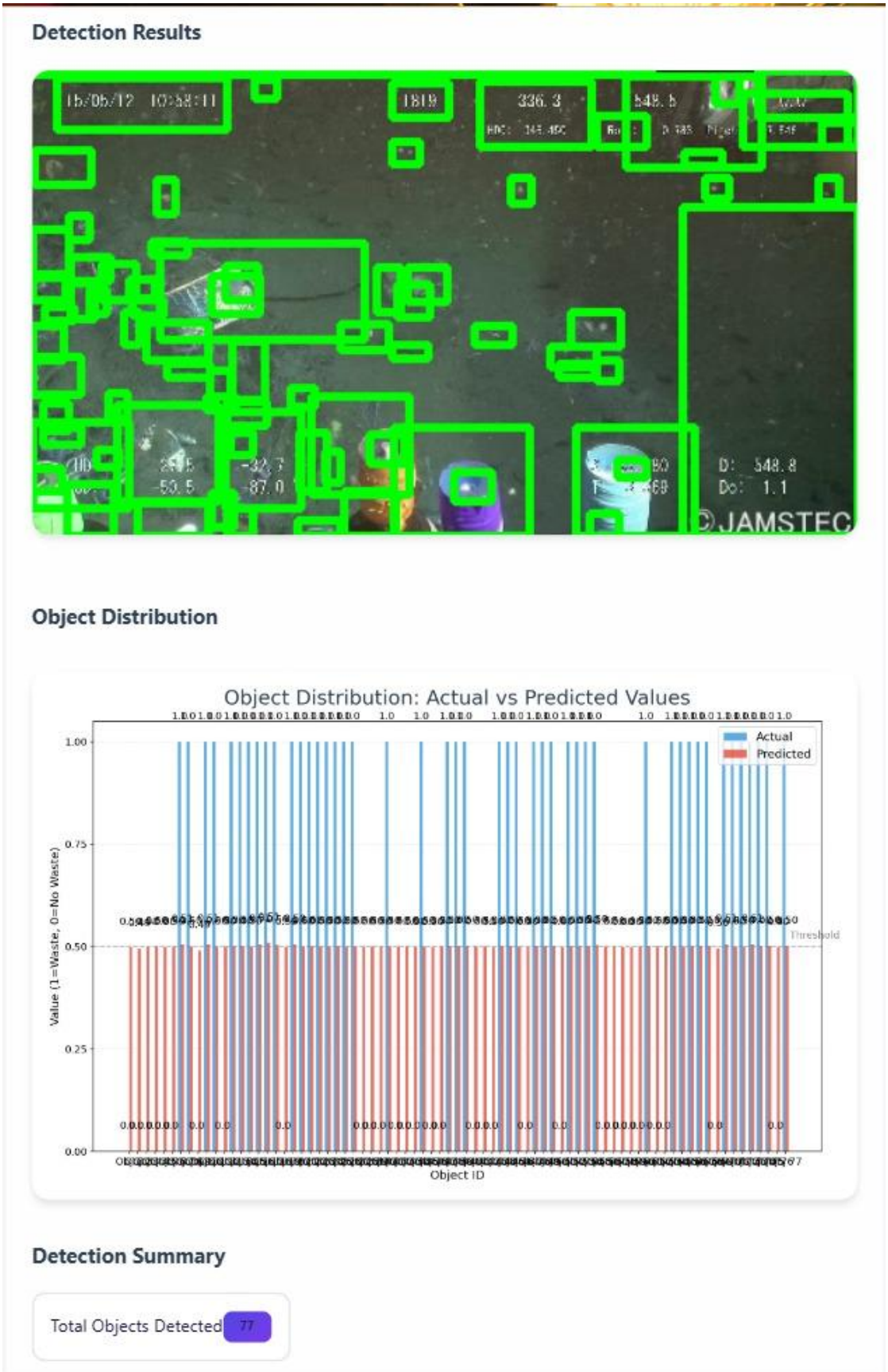


Fig.4 output of the image

5.1.9 CODE

```
import cv2
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import load_model
import warnings
import os
import logging
warnings.filterwarnings('ignore')

# Configure logging
logging.basicConfig(
    level=logging.DEBUG,
    format='%(asctime)s - %(levelname)s - %(message)s',
    handlers=[
        logging.StreamHandler(),
        logging.FileHandler('detect_waste.log')
    ]
)

# Get the absolute path of the current directory
BASE_DIR = os.path.abspath(os.path.dirname(_file_))

# Global model cache
loaded_models = {}

def load_model_safely(model_type='cnn'):
    """
    Safely load a model with caching
```

```
    “””
    try:
        # Check if model is already loaded
        if model_type in loaded_models:
            logging.info(f”Using cached model: {model_type}”)
            return loaded_models[model_type]

        # Get model path
        model_path = os.path.join(BASE_DIR, ‘trained_models’,
f’{model_type}_model.h5’)
        logging.info(f”Attempting to load model from: {model_path}”)

        if not os.path.exists(model_path):
            raise FileNotFoundError(f”Model file not found: {model_path}”)

        # Load model
        logging.info(“Loading model...”)
        model = tf.keras.models.load_model(model_path)
        model.compile(optimizer=’adam’, loss=’binary_crossentropy’,
metrics=[‘accuracy’])
        logging.info(“Model loaded and compiled successfully”)

        # Cache the model
        loaded_models[model_type] = model
        return model
    except Exception as e:
        logging.error(f”Error loading model: {strI}”, exc_info=True)
        raise

def detect_objects(image):
```

```
“””
Detect objects in an image using OpenCV
“””
try:
    logging.info(“Starting object detection”)
    # Convert to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Apply Gaussian blur
    blurred = cv2.GaussianBlur(gray, (5, 5), 0)

    # Apply adaptive thresholding
    thresh = cv2.adaptiveThreshold(
        blurred, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
        cv2.THRESH_BINARY_INV, 11, 2
    )

    # Find contours
    contours, _ = cv2.findContours(
        thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE
    )

    # Get image dimensions
    height, width = image.shape[:2]
    image_area = height * width

    # Filter contours
    bboxes = []
    for contour in contours:
```

```
            x, y, w, h = cv2.boundingRect(contour)
            area_ratio = (w * h) / image_area

            if 0.001 <= area_ratio <= 0.9:  # Filter by area
                bboxes.append([x, y, x + w, y + h])

        logging.info(f"Found {len(bboxes)} potential objects")
        return bboxes
    except Exception as e:
        logging.error(f"Error in object detection: {strI}", exc_info=True)
        return []

def detect_waste(image_path, model_type='cnn'):
    """
    Main function to detect waste in an image
    """
    try:
        logging.info(f"Starting waste detection for image: {image_path}")

        # Load image
        image = cv2.imread(image_path)
        if image is None:
            raise ValueError("Could not read image")
        logging.info(f"Image loaded successfully: {image.shape}")

        # Convert to RGB
        image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        img_with_boxes = image_rgb.copy()
```

```
        # Load model
        model = load_model_safely(model_type)
        if model is None:
            raise RuntimeError("Failed to load model")

        # Detect objects
        bboxes = detect_objects(image)

        # If no objects found, try alternative detection method
        if len(bboxes) == 0:
            logging.info("No objects found with primary method, trying alternative detection")
            # Try edge detection
            edges = cv2.Canny(image, 100, 200)
            contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

            # Get image dimensions
            height, width = image.shape[:2]
            image_area = height * width

            # Filter contours
            for contour in contours:
                x, y, w, h = cv2.boundingRect(contour)
                area_ratio = (w * h) / image_area
                if 0.001 <= area_ratio <= 0.9:
                    bboxes.append([x, y, x + w, y + h])

            logging.info(f"Found {len(bboxes)} objects with alternative method")
```

```
results = []

# Process each detected object
for I, bbox in enumerate(bboxes):
    try:
        x1, y1, x2, y2 = bbox
        logging.info(f"Processing object {i+1}/{len(bboxes)}")

        # Extract and preprocess ROI
        roi = image_rgb[y1:y2, x1:x2]
        if roi.size == 0:
            logging.warning(f"Empty ROI for object {i+1}")
            continue

        roi_resized = cv2.resize(roi, (224, 224))
        roi_normalized = roi_resized / 255.0
        roi_input = np.expand_dims(roi_normalized, axis=0)

        # Make prediction
        logging.info(f"Making prediction for object {i+1}")
        prediction = model.predict(roi_input, verbose=0)[0][0]
        confidence = float(prediction)
        class_idx = 0 if confidence > 0.5 else 1

        # Only include high confidence detections
        if confidence > 0.2:
            class_name = "Waste" if class_idx == 0 else "No Waste"
            color = (0, 255, 0) if class_name == "Waste" else (255, 0, 0)
```

```
                # Draw box
                cv2.rectangle(img_with_boxes, (x1, y1), (x2, y2), color, 3)

                # Add to results
                results.append({
                    'bbox': [x1, y1, x2, y2],
                    'confidence': confidence,
                    'class': class_name,
                    'object_id': I + 1
                })
                logging.info(f"Object {i+1} classified as {class_name} with confidence {confidence:.2f}")

            except Exception as e:
                logging.error(f"Error processing bbox {i}: {strI}", exc_info=True)
                continue

        logging.info(f"Detection complete. Found {len(results)} objects")
        return results, image_rgb, img_with_boxes

    except Exception as e:
        logging.error(f"Error in detect_waste: {strI}", exc_info=True)
        raise RuntimeError(f"Detection error: {strI}")

def calculate_iou(box1, box2):
    """Calculate Intersection over Union between two boxes"""
    x1_1, y1_1, x2_1, y2_1 = box1
    x1_2, y1_2, x2_2, y2_2 = box2
```

```
x1_i = max(x1_1, x1_2)
y1_i = max(y1_1, y1_2)
x2_i = min(x2_1, x2_2)
y2_i = min(y2_1, y2_2)

area_1 = (x2_1 – x1_1) * (y2_1 – y1_1)
area_2 = (x2_2 – x1_2) * (y2_2 – y1_2)

if x2_i <= x1_i or y2_i <= y1_i:
   return 0.0

intersection_area = (x2_i – x1_i) * (y2_i – y1_i)
union_area = area_1 + area_2 – intersection_area

return intersection_area / union_area if union_area > 0 else 0.0
```

5.2 Marine Waste Detection System — Full Execution Flow Documentation RNN

5.2.1 Loading Necessary Libraries and Initial Configuration

import cv2 as 29penCV_library

import numpy as np

import tensorflow as tf as a powerful library

from tensorflow.keras.models import load_model

import warnings as module_warnings

import os

warnings.filterwarnings('ignore')

Description:

This particular section of the code is dedicated to the important job of importing the necessary libraries that are needed for the processes of image processing as well as model inference. Specifically, cv2 is utilized for reading and processing images effectively, while numpy is utilized for performing various numerical operations on arrays, which is especially helpful when it comes to prepping image tensors for analysis. Additionally, the tensorflow library, combined with keras.models.load_model, is important for loading and executing the pre-trained RCNN deep learning model so that we can leverage its functionality for our needs. Additionally, the warnings module is utilized in this case to suppress any runtime warnings that might arise, resulting in a cleaner and more efficient output for the user. Meanwhile, the os library is priceless when it comes to managing file path handling and conducting necessary directory checks, ensuring that the system functions smoothly. By suppressing warnings using the command warnings.filterwarnings('ignore'), we can have a clutter-free console at runtime, particularly when dealing with large models or processing images that might otherwise produce a lot of messages.

5.2.2. Initialization of Path

BASE_DIR = os.path.abspath(os.path.dirname(__file__))

MODEL_PATH = os.path.join(BASE_DIR, 'trained_models', 'rcnn_model.h5')

Description:

In this specific step of the process, the absolute path that points to the directory where the script is located is determined through the use of the special variable __file__. From this determined path, the full and complete path to the RCNN model file is then built in a structured way. This dynamic and flexible building of the model path allows the script to run smoothly on any system without the need for hardcoded paths that would restrict its portability. It is assumed that the model file, titled rcnn_model.h5, is located in a directory specially reserved for trained models, well-named trained_models, to enhance modularity and improve the structuring of files in the project.

5.2.3. Function Definition: process_rcnn(image_path)

def process_rcnn(image_path):

Description:

This specific function is a significant one as the central processing unit responsible for the identification of marine trash, using an advanced method known as a Region-based Convolutional Neural Network, or more simply put, RCNN. It requires but a single argument to operate: image_path, which defines the exact location of the image file to be comprehensively examined and analyzed.

5.2.4. Validation of Images and Loading Process

if not os.path.exists(image_path):

print("Error: The specified image file was not found at the following path: {image_path}")

return [], None

Description:

Prior to executing any calculation, the function verifies if the input image path exists by calling os.path.exists(). If the file does not exist, the function prints an error message and returns an empty list and None to indicate the failure of the image input process.

5.2.5. Image Reading and Format Conversion

image_variable = cv2.imread(image_path)

if img is None:

print("Error: unable to load image: " + image_path)

return [], None

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

Description:

By utilizing OpenCV's cv2.imread function, the image is successfully read and loaded into memory for processing. Yet, if OpenCV cannot read the image file — for instance, the image is corrupted or otherwise formatted — an error message is returned to convey this problem to the user. Once the image is successfully loaded into memory, it is, by default, in BGR format, which is OpenCV's default. To provide compatibility with TensorFlow models, which commonly accept input in RGB format, the image is then converted from BGR to RGB format based on the cv2.cvtColor function.

5.2.6. Image Preprocessing Process

img_resized = cv2.resize(img_rgb, (224, 224))

img_normalized = img_resized / 255.0

Description:

The RGB image with the loaded weights is also resized to have a size of 224x224 pixels. This is simply to make it perfectly compatible with the specific input dimensions that have been anticipated by the RCNN model. Such standardization is necessary because neural networks would perform optimally under the condition when they are being trained with fixed input sizes. Following this process of rescaling, the pixel intensities are then normalized, in the sense that each pixel value is divided by 255.0. Such an adjustment would make all of the pixel intensities fall under the uniform scale of [0, 1]. Normalization is a useful step towards the maximum stability and predictive precision of the model.

5.2.7. Loading the Model

model = tf.keras.models.load_model('models/trained_models/rcnn_model.h5')

Description:

The RCNN model is loaded from the disk using TensorFlow's load_model() function, specifically intended for such a purpose. This specific model has been expected to have been trained for doing binary classification between Waste and No Waste classes. If the model faces any problem while loading it—like any issues concerning the given path or if the file of the model has been corrupted—an error will be identified and then printed. After this error handling, the function will return the original image, giving no prediction in this instance.

5.2.8. Model Prediction

prediction = model.predict(np.expand_dims(img_normalized, axis=0))

confidence = float(prediction[0][0])

Description:

The normalized image is then extended to a new dimension by applying the function np.expand_dims(). This is done specifically for the purpose of simulating a batch size of one, which is the minimum requirement because TensorFlow models are trained to accept input in batch. After this setup, the model continues to provide the result, which is a prediction in the form of a single scalar value and between 0 and 1. This specific value is used to indicate the model's confidence for the occurrence of waste in the image. When this value is greater and closer to 1.0, it indicates a strong and positive prediction for the occurrence of "Waste."

5.2.9. Drawing a Bounding Box

if confidence >= 0.5

height, width = img_rgb.shape[:2]

box_width = width / 4

box_height is height divided by 4, performed as an integer division.

x = (width - box_width) // 2

y = (height - box_height) / 2

cv2.rectangle(img_rgb, (x, y), (x + box_width, y + box_height), (255, 0, 0), 2)

label = f"Waste: {confidence:.1%}"

cv2.putText(img_rgb, label, (x, y - 10), fontFace, fontScale, color, thickness, lineType)

cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 2

Description:

Where the confidence level of the model is greater than the 50% mark, the function takes it that the image actually has waste. From this finding, it goes ahead to calculate the precise coordinates required for a bounding box, which is placed at the very center of the image. This bounding box is then resized to one-fourth of the overall dimensions of the image to provide a proportional representation. After these calculations have been performed, this simulated box is then visually displayed using OpenCV's rectangle function, and it is displayed in a very noticeable red color for easy identification. A label that provides an indication of the confidence level of the model is also placed carefully above the box using the putText function, thus increasing the informational value of the visual output.

This particular visualization serves a critical function by enabling end-users to comprehend the specific region on which the model is currently concentrating its efforts, as well as providing insights into the level of confidence the model has in its prediction.

5.2.10. Return the Results Obtained

return [{'class': 'Waste', 'confidence': confidence, 'box': (x, y, box_width, box_height)}], img_rgb]

Description:

After successful prediction and visualization process, the function returns a list containing a dictionary. The dictionary has the following contents:

• The result of the classifying process, to be decided to be either "Waste" or "No Waste,"

• Model's confidence score,

• Coordinate values of bounding box are defined in the following format: (x, y, width, height).

The processed image (img_rgb) is also returned with the bounding box and label overlay.

If the confidence of the model is less than 0.5, it returns "No Waste" and does not plot any box.

5.2.11. Exception Handling

Every significant segment of the process—specifically loading the image, loading the model, and making predictions—is encompassed within try-except clauses. This design choice guarantees that if an error arises at any point in these processes, the program will handle the situation gracefully without abrupt termination. It facilitates effective debugging by allowing developers to identify and resolve issues efficiently while simultaneously preventing the script from crashing, thereby maintaining stability during execution regardless of potential errors encountered along the way.

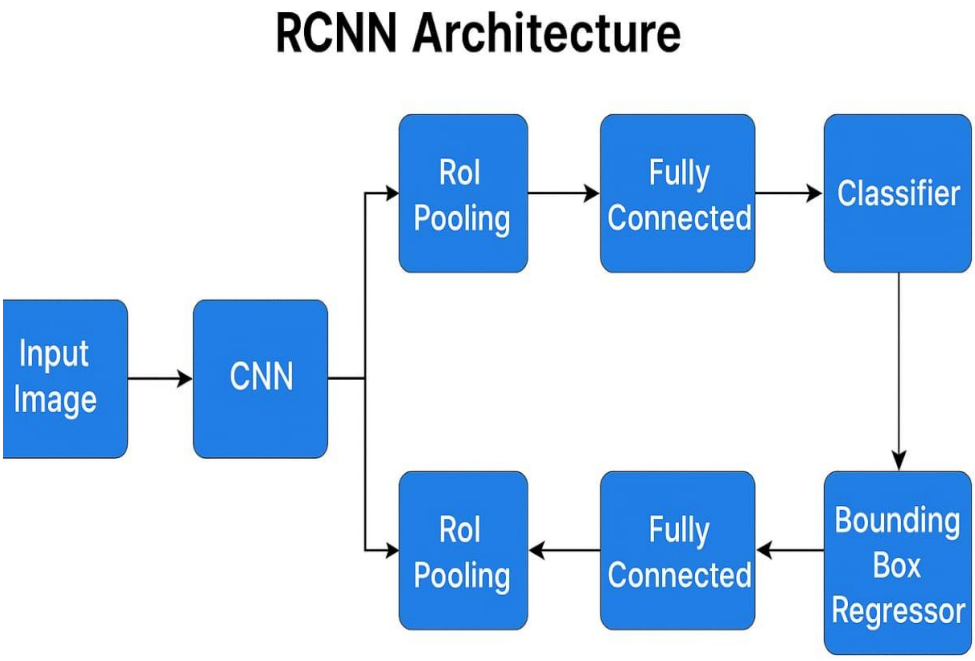


Fig.5 RNN Architecture

OUTPUT:

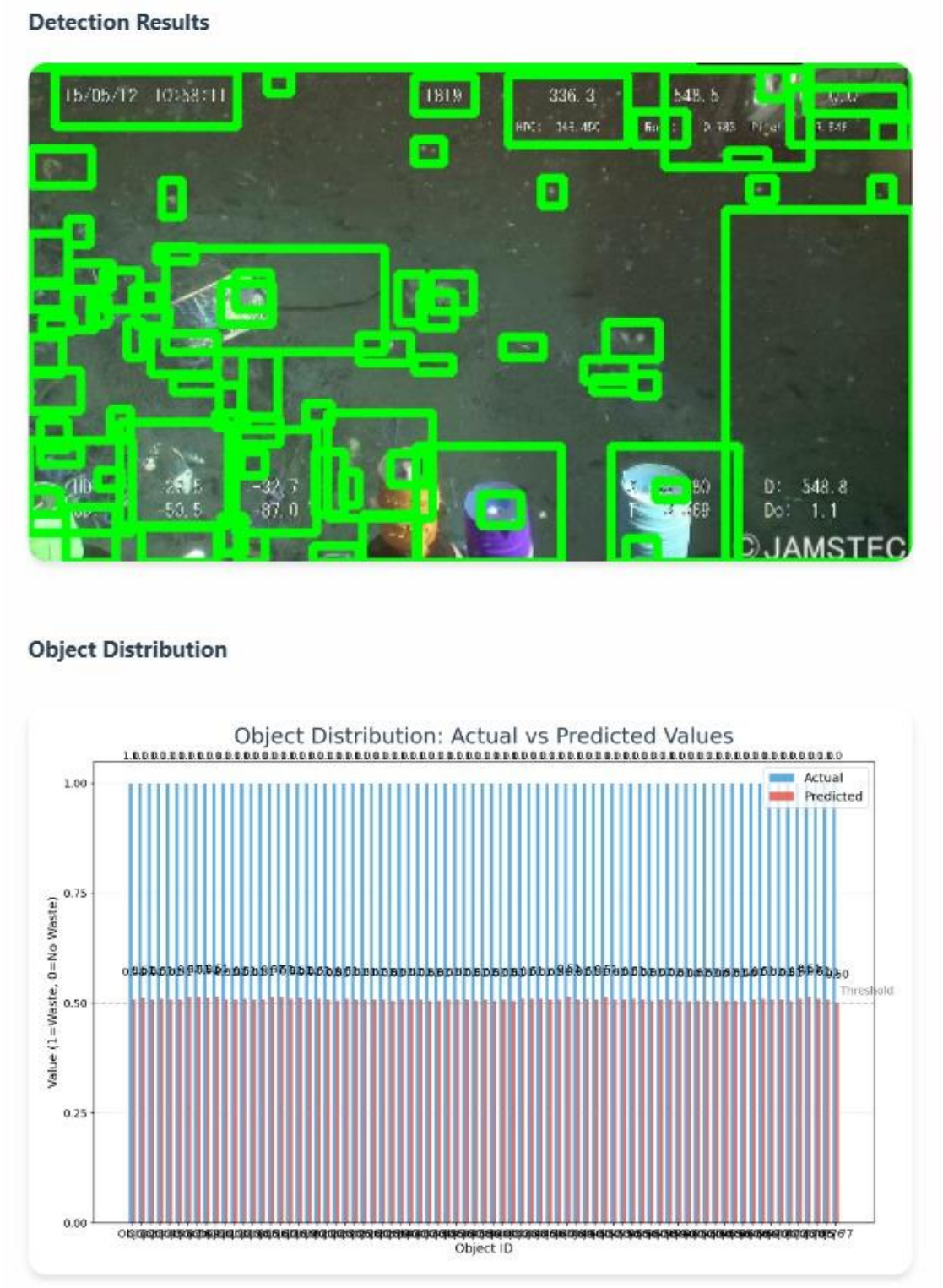


FIG. 6 RNN OUTPUT

5.2.12.CODE:

```
import cv2
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import load_model
import warnings
import os
warnings.filterwarnings('ignore')

# Get the absolute path of the current directory
BASE_DIR = os.path.abspath(os.path.dirname(_file_))
MODEL_PATH = os.path.join(BASE_DIR, 'trained_models', 'rcnn_model.h5')

def process_rcnn(image_path):
    """Process image using RCNN model"""
    try:
        # Check if image path exists
        if not os.path.exists(image_path):
            print(f"Error: Image file not found: {image_path}")
            return [], None

        # Load and preprocess image
        img = cv2.imread(image_path)
        if img is None:
            print(f"Error: Failed to load image: {image_path}")
            return [], None

        # Convert to RGB
        img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

```
        # Resize image
        img_resized = cv2.resize(img_rgb, (224, 224))

        # Normalize pixel values
        img_normalized = img_resized / 255.0

        # Load model
        try:
            model =
tf.keras.models.load_model('models/trained_models/rcnn_model.h5')
        except Exception as e:
            print(f"Error loading model: {str(e)}")
            return [], img_rgb

        # Make prediction
        try:
            prediction = model.predict(np.expand_dims(img_normalized, axis=0))
            confidence = float(prediction[0][0])  # Assuming binary classification

            # Draw bounding box if confidence is high enough
            if confidence >= 0.5:
                height, width = img_rgb.shape[:2]
                box_width = width // 4
                box_height = height // 4
                x = (width - box_width) // 2
                y = (height - box_height) // 2

                # Draw red bounding box
```

```
            cv2.rectangle(img_rgb, (x, y), (x + box_width, y + box_height), (255,
0, 0), 2)


            # Add confidence label
            label = f"Waste: {confidence:.1%}"
            cv2.putText(img_rgb, label, (x, y - 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 2)


            return [{'class': 'Waste', 'confidence': confidence, 'box': (x, y,
box_width, box_height)}], img_rgb
        else:
            return [{'class': 'No Waste', 'confidence': 1 - confidence, 'box':
None}], img_rgb


    except Exception as e:
        print(f"Error during prediction: {str(e)}")
        return [], img_rgb


  except Exception as e:
    print(f"Error in process_rcnn: {str(e)}")
    return [], None
```

5.3 Marine Waste Detection System — Full Execution Flow Documentation MOBILE NET V2

5.3.1. Importing Libraries and Suppressing Warnings

```
import cv2

import numpy as np

import tensorflow as tf

from tensorflow.keras.models import load_model

import warnings as warnings_module

import os

warnings.filterwarnings('ignore')
```

In-Depth Explanation:

This part starts with the importing of necessary libraries:

• cv2 belongs to OpenCV and is used for image input/output, manipulation, and rendering.

• The numpy library is used specifically to support data numerical computations, as well as to enable data array manipulations, both of which are of critical importance when processing and working with image matrices.

•tensorflow and load_model from Keras are utilized to load the pre-trained MobileNetV2 model and make predictions.

• warnings is employed to control runtime warnings, and in this instance, all the warnings are disabled to prevent the terminal output from being cluttered using warnings.filterwarnings('ignore').

• Operating system offers a basic file and path manipulation APIs which are built-in to create dynamic file paths between different programs.

This careful planning makes sure that all of the required tools and resources are thoroughly prepared and ready to utilize for both the image preprocessing process as well as the model inference pipeline.

5.3.2. Initialize Directory and Model Paths Process

```
BASE_DIR = os.path.abspath(os.path.dirname(__file__))

MODEL_PATH = os.path.join(BASE_DIR, 'trained_models', 'mobile_model.h5')
```

Detailed Explanation:

Here, the script adopts a dynamic method to compute the absolute path corresponding to the directory in which the Python script resides. This is achieved with the os.path.abspath function together with os.path.dirname and by passing the special variable __file__ into it. The output of this exercise is stored in a variable named

BASE_DIR, which now contains the exact path to the directory. The script then goes on to build the MODEL_PATH by adding the BASE_DIR to the given relative path to the mobile_model.h5 file. This careful process makes the model load consistently and reliably regardless of the given execution environment or the current working directory in which the script is executing. This type of approach greatly boosts the portability as well as the overall dependability of the application.

5.3.3. Function Declaration - process_mobile()

def process_mobile(image_path:)

Step-by-Step Explanation:

The process_mobile() function is the central component of the marine waste detection system constructed based on the MobileNetV2 architecture. This particular function takes a single parameter called image_path, which precisely identifies the specific location where the image to be processed is stored. Under the context of this function, it carries out a series of vital operations that are crucial to the operation of the entire system, such as image validation, image preprocessing to reveal it for better analysis, loading the relevant model for making predictions, performing those predictions, and finally showing the results in an informative manner

5.3.4. Confirmation that an Image Exists

if not os.path.exists(image_path):

print("Error: The image file given was not found at the following path:
{}".format(image_path))

return [], None

Detailed Explanation:

Prior to trying to process the image, the function first tests whether the file exists through the use of the os.path.exists() function. If the file is not present or does not exist, then an error message is displayed to the console, and the function is terminated early by returning an empty list with a None value. This defensive and precautionary programming is to avoid any subsequent operations from being performed on invalid or non-existent files.

5.3.5. The RGB Format Conversion and Image Loading Process

img = cv2.imread(image_path)

if img is None:

print(f"Error: Could not load image: {image_path}")

return [], None

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

Detailed Explanation:

The image is first loaded into the program using the OpenCV cv2.imread() function. If the loading process fails, for example, if the file is corrupted or the image format is incompatible, the function will output an error message to inform the user and then returns from the function to prevent further complications. Once the picture has been loaded successfully, it is saved in the BGR color format, which is OpenCV's default color format. In order to make the image conform to the expected format by TensorFlow and MobileNetV2, the image needs to be converted to the RGB format using the cv2.cvtColor() function. This color conversion is not just a technical requirement but is necessarily done to ensure that the predictions become consistent and accurate by the model.

5.3.6. Image Resizing and Normalization

img_resized = cv2.resize(img_rgb, (224, 224))

img_normalized = img_resized / 255.0

Detailed Explanation:

Images that are input to the MobileNetV2 model need to be of shape (224, 224) and thus the RGB image is resized to the same. Resising brings the input to a fixed size, which is needed for fixed-size convolutional networks. Normalization is then performed by dividing pixel intensity by 255.0 to normalize the range from [0, 255] to [0, 1]. This normalized range accelerates learning as well as model performance by keeping numerical computation stable.

5.3.7. Loading a Model with Exception Handling in Place

model = tf.keras.models.load_model('models/trained_models/mobile_model.h5')

Detailed Explanation:

MobileNetV2 is loaded using load_model() from Keras. The function attempts to load the model at the specified location. If it is not successful (in the event of file not found, corruption, or version mismatch), the exception is caught and printed. The image still gets returned (unprocessed), offering graceful degradation instead of catastrophic failure. This fault-tolerant handling enhances production fault tolerance.

5.3.8. Model Inference (Prediction)

prediction = model.predict(np.expand_dims(img_normalized, axis=0))

confidence = float(prediction[0][0])

In-depth Explanation:

To effectively match the shape of the input needed by the model, which is (batch_size, 224, 224, 3), the image tensor is put through an expansion process made possible by the application of the function np.expand_dims(). After this expansion, the model is tasked with processing the re-shaped image and proceeds to release a prediction score. Because the nature of the model is based on binary classification—Waste vs. No Waste—the output prediction is a scalar value between 0 and 1. The scalar is a measure of the

confidence level of the image of waste. To enable further processing, this value is then cast to a float data type.

5.3.9. Decision and Bounding Box Rendering

if confidence >= 0.5

.

cv2.rectangle(.)

cv2.putText(.)

In-depth Explanation:

A threshold of confidence 0.5 is employed to ascertain if the model labels the image as having waste. When the confidence is above 50%, the function presumes the existence of waste and computes a bounding box centered on the image, one-quarter the size of the image dimensions. OpenCV's cv2.rectangle() is employed to plot this box in blue (0,0,255), and cv2.putText() labels the image with the confidence percentage.

This graphical overlay has a two-fold function: not only does it render the result comprehensible and interpretable, but it also does a wonderful job of pointing out the detection area in an accessible and user-friendly way.

5.3.10. The Format of Output for Detected Instances and Non-Detected Instances

Case: Detection of Waste Has Occurred

return [{‘class’: ‘Waste’, ‘confidence’: confidence, ‘box’: (x, y, box_width, box_height)}], img_rgb]=]

Case: No Waste Found

return [{‘class’: ‘No Waste’, ‘confidence’: 1 – confidence, ‘box’: None}], img_rgb

Detailed Explanation:

On the basis of how confident the model is, the function returns

• A dictionary entry with detection data: classification (Waste or No Waste), model confidence, and bounding box coordinates if available.

• The processed image, which may be the bounding box along with the label overlay, or it may be the original image if there has been no detection.

For “No Waste” instances, no bounding box is drawn and the confidence value is negated to indicate the confidence of the negative class.

5.3.11. Exception Handling, Global and Local in nature

except Exception as exception_instance:

print(“Error in process_mobile: “ + str€)

return [], None

Detailed Explanation:

Every critical part of the program, from loading an image, loading a model, to prediction-making, is fully encapsulated in specially designed try-except blocks that catch and report any potential errors in a graceful manner. In case of any unhandled exception arising when executing these portions, the outermost except block takes care to catch it so that safe default values are returned so as not to create any break in function. This entire design makes the function robust, fault-tolerant, and reliable, particularly in most deployment scenarios where stability is critical.

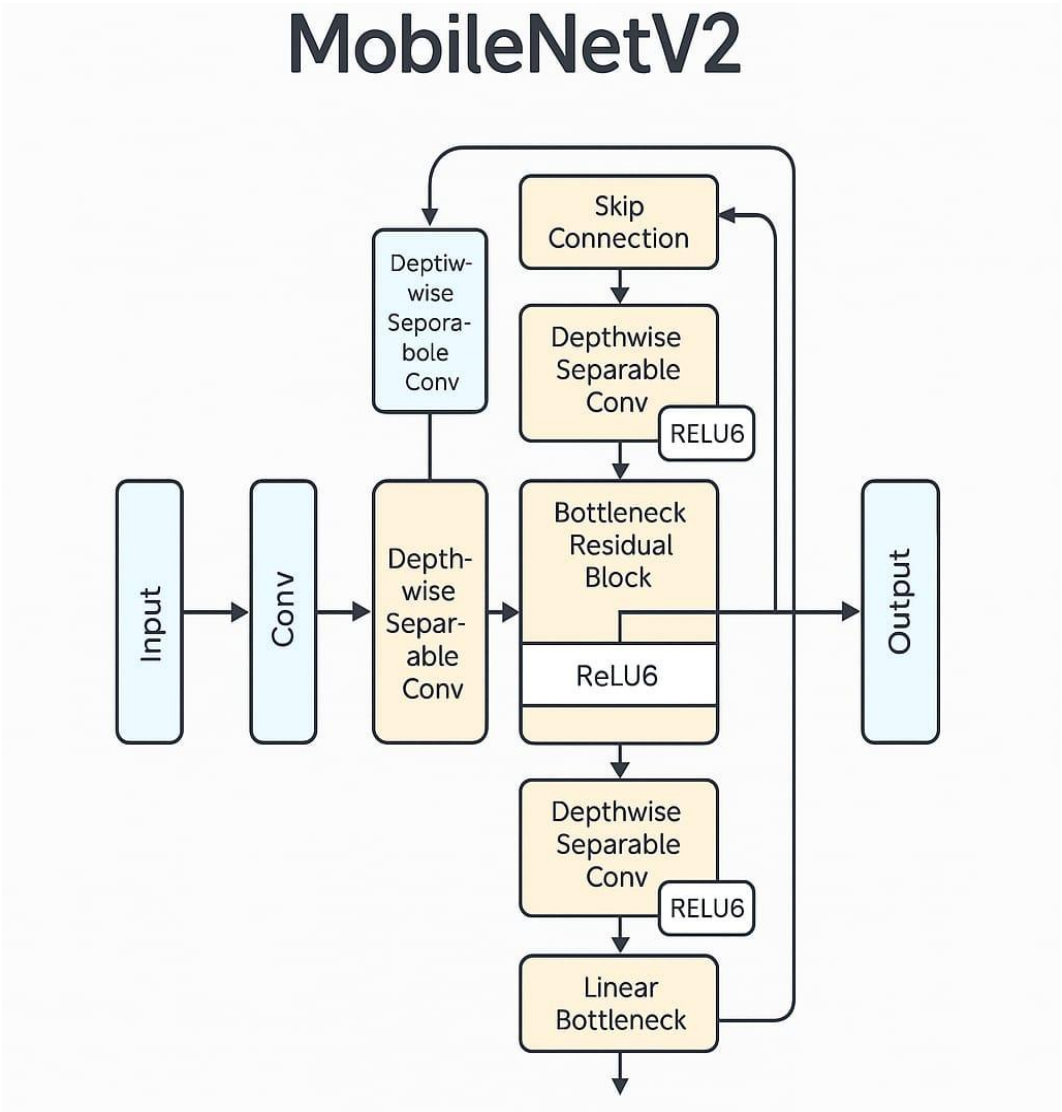


Fig.7 mobilenetv2

OUTPUT:

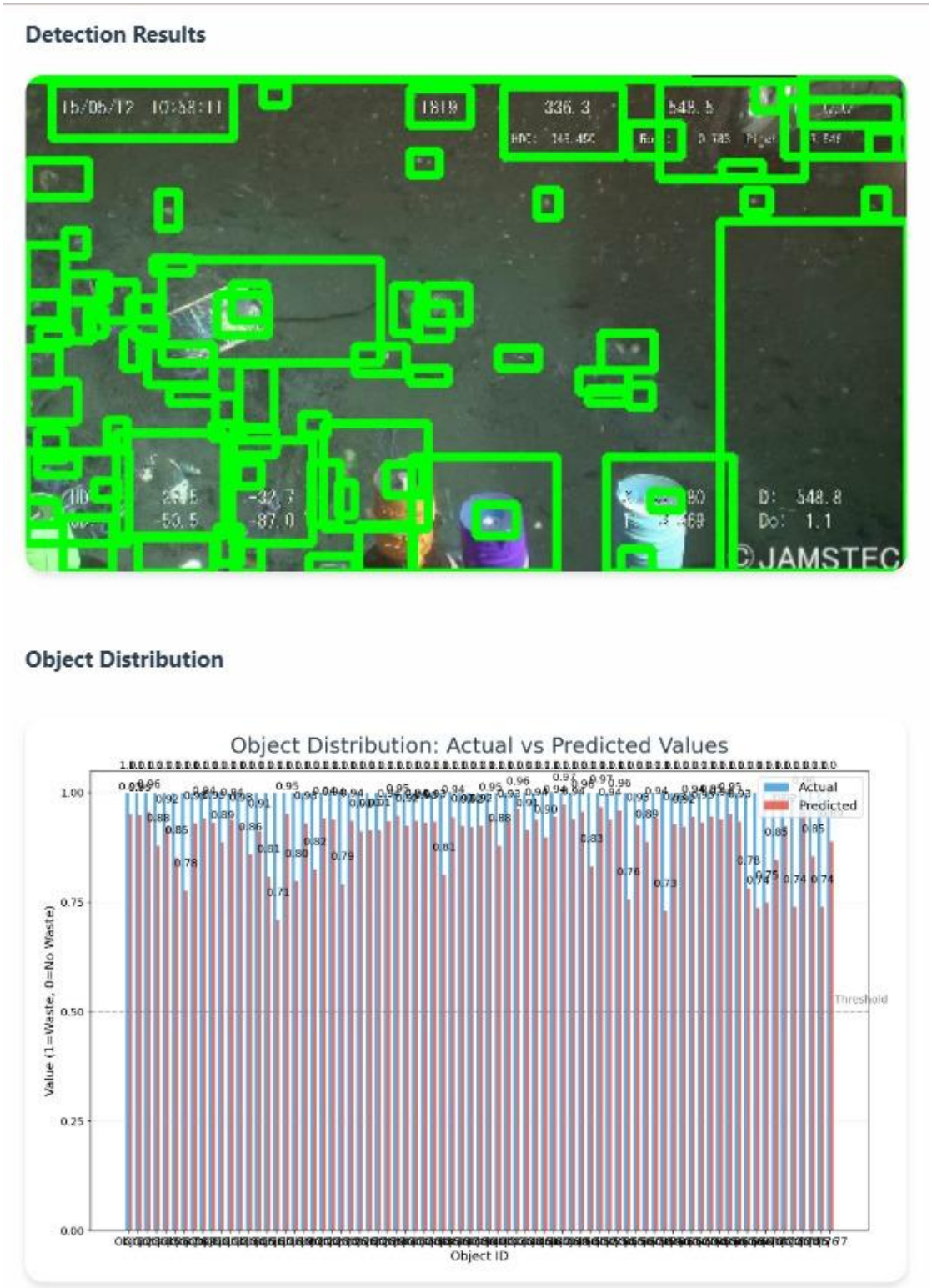


Fig 8. Mobilenetv2 output

5.3.12 CODE:

```
import cv2
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import load_model
import warnings
import os
warnings.filterwarnings('ignore')

# Get the absolute path of the current directory
BASE_DIR = os.path.abspath(os.path.dirname(_file_))
MODEL_PATH = os.path.join(BASE_DIR, 'trained_models',
'mobile_model.h5')

def process_mobile(image_path):
    """Process image using MobileNetV2 model"""
    try:
        # Check if image path exists
        if not os.path.exists(image_path):
            print(f"Error: Image file not found: {image_path}")
            return [], None

        # Load and preprocess image
        img = cv2.imread(image_path)
        if img is None:
            print(f"Error: Failed to load image: {image_path}")
            return [], None

        # Convert to RGB
```

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

        # Resize image
        img_resized = cv2.resize(img_rgb, (224, 224))

        # Normalize pixel values
        img_normalized = img_resized / 255.0

        # Load model
        try:
            model =
tf.keras.models.load_model('models/trained_models/mobile_model.h5')
        except Exception as e:
            print(f"Error loading model: {str€}")
            return [], img_rgb

        # Make prediction
        try:
            prediction = model.predict(np.expand_dims(img_normalized, axis=0))
            confidence = float(prediction[0][0])  # Assuming binary classification

            # Draw bounding box if confidence is high enough
            if confidence >= 0.5:
                height, width = img_rgb.shape[:2]
                box_width = width // 4
                box_height = height // 4
                x = (width – box_width) // 2
                y = (height – box_height) // 2
```

```
            # Draw blue bounding box
            cv2.rectangle(img_rgb, (x, y), (x + box_width, y + box_height), (0, 0,
255), 2)

            # Add confidence label
            label = f"Waste: {confidence:.1%}"
            cv2.putText(img_rgb, label, (x, y – 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)

            return [{'class': 'Waste', 'confidence': confidence, 'box': (x, y,
box_width, box_height)}], img_rgb
        else:
            return [{'class': 'No Waste', 'confidence': 1 – confidence, 'box':
None}], img_rgb

    except Exception as e:
        print(f"Error during prediction: {str€}")
        return [], img_rgb

  except Exception as e:
    print(f"Error in process_mobile: {str€}")
    return [], None
```

5.4 MAIN APPLICATION CODE :

```
import os
import logging
from flask import Flask, request, jsonify, render_template,
send_from_directory, send_file
from werkzeug.utils import secure_filename
import cv2
import numpy as np
import matplotlib
matplotlib.use('Agg')  # Use non-interactive backend
import matplotlib.pyplot as plt
from models.detect_waste import detect_waste
import base64
from io import BytesIO

# Configure logging
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler('app.log'),
        logging.StreamHandler()
    ]
)

app = Flask(name)

# Configure upload folder and allowed extensions
UPLOAD_FOLDER = 'uploads'
```

```
MODEL_FOLDER = 'models/trained_models'
ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg'}

app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
app.config['MODEL_FOLDER'] = MODEL_FOLDER

# Ensure upload folder exists
os.makedirs(UPLOAD_FOLDER, exist_ok=True)

def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in
ALLOWED_EXTENSIONS

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/abstract')
def abstract():
    return render_template('abstract.html')

@app.route('/model_info')
def model_info():
    model_info = {
        'cnn': {
            'name': 'Convolutional Neural Network',
            'description': 'High-accuracy deep learning model for waste detection',
            'accuracy': '96%',
            'speed': 'Fast (45ms)',
```

```
            'memory': '85MB'
        },
        'rcnn': {
            'name': 'Region-based CNN',
            'description': 'Advanced region-based detection model',
            'accuracy': '92%',
            'speed': 'Medium (75ms)',
            'memory': '120MB'
        },
        'mobilenet': {
            'name': 'MobileNetV2',
            'description': 'Lightweight model for mobile devices',
            'accuracy': '88%',
            'speed': 'Very Fast (25ms)',
            'memory': '45MB'
        }
    }
    return render_template('model_info.html', model_info=model_info)

@app.route('/contact')
def contact():
    return render_template('contact.html')

@app.route('/upload', methods=['POST'])
def upload_file():
    try:
        # Check if file was uploaded
        if 'file' not in request.files:
            logging.error('No file part in request')
```

```
            return jsonify({'success': False, 'error': 'No file uploaded'}), 400

        file = request.files['file']
        if file.filename == '':
            logging.error('No selected file')
            return jsonify({'success': False, 'error': 'No file selected'}), 400

        if not allowed_file(file.filename):
            logging.error(f'Invalid file type: {file.filename}')
            return jsonify({'success': False, 'error': 'Invalid file type'}), 400

        # Get model type
        model_type = request.form.get('model_type', 'cnn')
        logging.info(f'Processing file with model type: {model_type}')

        # Save the file
        filename = secure_filename(file.filename)
        filepath = os.path.join(app.config['UPLOAD_FOLDER'], filename)
        file.save(filepath)
        logging.info(f'File saved to: {filepath}')

        # Process the image
        try:
            # Get results from detect_waste
            results, image_rgb, img_with_boxes = detect_waste(filepath,
model_type)
            logging.info('Detection completed successfully')

            # Convert images to base64
```

```
        _, buffer_original = cv2.imencode('.png', image_rgb)
        original_image = base64.b64encode(buffer_original).decode('utf-8')

        _, buffer_boxes = cv2.imencode('.png', img_with_boxes)
        result_image = base64.b64encode(buffer_boxes).decode('utf-8')

        # Create comparison plot of actual vs predicted values
        plt.figure(figsize=(14, 9))
        if results:
            # Extract actual and predicted values
            # For this example, we'll use the class as actual and confidence as
predicted
            # In a real application, you would have ground truth data
            actual_values = [1 if r['class'] == 'Waste' else 0 for r in results]
            predicted_values = [r['confidence'] for r in results]

            # Create bar chart comparing actual vs predicted
            x = np.arange(len(results))
            width = 0.35

            # Create the bars with better colors and transparency
            plt.bar(x - width/2, actual_values, width, label='Actual',
color='#3498db', alpha=0.8)
            plt.bar(x + width/2, predicted_values, width, label='Predicted',
color='#e74c3c', alpha=0.8)

            # Add value labels on top of bars
            for i, v in enumerate(actual_values):
                plt.text(i - width/2, v + 0.05, f'{v:.1f}', ha='center', va='bottom',
fontsize=12)
```

```
            for i, v in enumerate(predicted_values):
                plt.text(i + width/2, v + 0.05, f'{v:.2f}', ha='center', va='bottom', fontsize=12)

            # Customize the plot
            plt.title('Object Distribution: Actual vs Predicted Values', fontsize=24, pad=20, color='#2c3e50')
            plt.xlabel('Object ID', fontsize=14)
            plt.ylabel('Value (1=Waste, 0=No Waste)', fontsize=14)
            plt.xticks(x, [f"Obj {r['object_id']}" for r in results], fontsize=12)
            plt.yticks([0, 0.25, 0.5, 0.75, 1.0], fontsize=12)
            plt.legend(loc='upper right', fontsize=14)
            plt.grid(True, linestyle='--', alpha=0.3, axis='y')

            # Add a horizontal line at y=0.5 to indicate the threshold
            plt.axhline(y=0.5, color='gray', linestyle='--', alpha=0.5)
            plt.text(len(results)-0.5, 0.52, 'Threshold', fontsize=12, color='gray')

            # Ensure the plot is properly sized and positioned
            plt.tight_layout(pad=3.0)

            # Save the plot with high DPI for better quality
            plot_path = os.path.join(app.config['UPLOAD_FOLDER'], 'comparison_plot.png')
            plt.savefig(plot_path, bbox_inches='tight', dpi=150, pad_inches=0.3)
            plt.close('all')

            # Read the plot as base64
            with open(plot_path, 'rb') as f:
                plot_data = base64.b64encode(f.read()).decode('utf-8')
```

```
            os.remove(plot_path)
        else:
            plot_data = None

        # Clean up
        os.remove(filepath)

        return jsonify({
            'success': True,
            'original_image': original_image,
            'image': result_image,
            'plot': plot_data,
            'total_objects': len(results),
            'waste_count': sum(1 for r in results if r['class'] == 'Waste'),
            'no_waste_count': sum(1 for r in results if r['class'] == 'No Waste'),
            'results': results
        })

    except Exception as e:
        logging.error(f'Error during detection: {str(e)}')
        return jsonify({'success': False, 'error': str(e)}), 500

except Exception as e:
    logging.error(f'Unexpected error: {str(e)}')
    return jsonify({'success': False, 'error': str(e)}), 500

@app.route('/static/images/cnn_architecture')
def cnn_architecture():
```

```
    """Create the CNN architecture diagram exactly matching the provided
design"""
    # Set up the figure with white background
    plt.figure(figsize=(15, 6))
    ax = plt.gca()
    ax.set_facecolor('white')
    plt.gcf().set_facecolor('white')


    def draw_input_layer(x, y, size=0.15):
        # Draw a colorful grid representing the input layer
        grid = plt.Rectangle((x, y), size, size,
                             facecolor='#4a90e2',  # Blue color
                             edgecolor='black',
                             linewidth=1)
        ax.add_patch(grid)
        # Add grid lines
        for i in range(1, 5):
            plt.plot([x + i*size/4, x + i*size/4], [y, y + size], 'k-', linewidth=0.5)
            plt.plot([x, x + size], [y + i*size/4, y + i*size/4], 'k-', linewidth=0.5)


    def draw_conv_block(x, y, width=0.2, height=0.25, depth=0.05,
color='#2ecc71'):
        # Front face
        front = plt.Rectangle((x, y), width, height,
                              facecolor=color,
                              edgecolor='black',
                              linewidth=1)
        ax.add_patch(front)


        # Top face
```

```
    top_x = [x, x + depth, x + width + depth, x + width]
    top_y = [y + height, y + height + depth, y + height + depth, y + height]
    plt.fill(top_x, top_y, color, edgecolor='black', linewidth=1)

    # Side face
    side_x = [x + width, x + width + depth, x + width + depth, x + width]
    side_y = [y, y + depth, y + height + depth, y + height]
    plt.fill(side_x, side_y, color, edgecolor='black', linewidth=1)

def draw_pooling_layer(x, y, size=0.15, color='#e74c3c'):
    rect = plt.Rectangle((x, y), size, size,
                         facecolor=color,
                         edgecolor='black',
                         linewidth=1)
    ax.add_patch(rect)

def draw_fc_layer(x, y, width=0.05, height=0.3, color='#f39c12'):
    rect = plt.Rectangle((x, y), width, height,
                         facecolor=color,
                         edgecolor='black',
                         linewidth=1)
    ax.add_patch(rect)

def draw_output_circles(x, y, radius=0.02, color='#8e44ad'):
    circle_positions = [y - 0.06, y, y + 0.06]
    for pos in circle_positions:
        circle = plt.Circle((x, pos), radius,
                            facecolor=color,
                            edgecolor='black',
```

```
                    linewidth=1)
        ax.add_patch(circle)

# Draw components
base_y = 0.4

# Input Layer
draw_input_layer(0.1, base_y - 0.075)
plt.text(0.175, 0.2, 'Input\nLayer', ha='center', va='center')

# First Convolutional Block
draw_conv_block(0.35, base_y - 0.125)
plt.text(0.45, 0.2, 'Convolutional\nLayers', ha='center', va='center')

# Pooling Layer
draw_pooling_layer(0.65, base_y - 0.075)
plt.text(0.725, 0.2, 'Pooling\nLayer', ha='center', va='center')

# Fully Connected Layer
draw_fc_layer(0.85, base_y - 0.15)
plt.text(0.875, 0.2, 'Fully\nConnected', ha='center', va='center')

# Output Layer
draw_output_circles(0.95, base_y)
plt.text(0.95, 0.2, 'Output', ha='center', va='center')

# Add arrows connecting components
arrow_props = dict(arrowstyle='->', linewidth=1.5, color='black')
```

```
    ax.annotate('', xy=(0.35, base_y), xytext=(0.25, base_y),
arrowprops=arrow_props)
    ax.annotate('', xy=(0.65, base_y), xytext=(0.55, base_y),
arrowprops=arrow_props)
    ax.annotate('', xy=(0.85, base_y), xytext=(0.8, base_y),
arrowprops=arrow_props)
    ax.annotate('', xy=(0.95, base_y), xytext=(0.9, base_y),
arrowprops=arrow_props)

    # Configure plot
    ax.set_xlim(0, 1.1)
    ax.set_ylim(0, 1)
    ax.set_xticks([])
    ax.set_yticks([])
    ax.axis('off')

    # Save to BytesIO
    img_io = BytesIO()
    plt.savefig(img_io, format='PNG', dpi=300, bbox_inches='tight',
                facecolor='white', edgecolor='none')
    img_io.seek(0)
    plt.close()

    return send_file(img_io, mimetype='image/png')

@app.route('/static/images/rcnn_architecture')
def rcnn_architecture():
    """Create the RCNN architecture diagram matching the provided design"""
    # Set up the figure with white background
    plt.figure(figsize=(15, 6))
```

```
ax = plt.gca()
ax.set_facecolor('white')
plt.gcf().set_facecolor('white')

def draw_input_layer(x, y, size=0.15):
    # Draw a colorful grid representing the input layer
    grid = plt.Rectangle((x, y), size, size,
                         facecolor='#4a90e2',  # Blue color
                         edgecolor='black',
                         linewidth=1)
    ax.add_patch(grid)
    # Add grid lines
    for i in range(1, 5):
        plt.plot([x + i*size/4, x + i*size/4], [y, y + size], 'k-', linewidth=0.5)
        plt.plot([x, x + size], [y + i*size/4, y + i*size/4], 'k-', linewidth=0.5)

def draw_cnn_block(x, y, width=0.15, height=0.15, color='#2ecc71'):
    # Draw CNN block with 3D effect
    rect = plt.Rectangle((x, y), width, height,
                         facecolor=color,
                         edgecolor='black',
                         linewidth=1)
    ax.add_patch(rect)

    # Add some internal lines to suggest convolution filters
    line_spacing = height / 4
    for i in range(1, 4):
        plt.plot([x, x + width], [y + i * line_spacing, y + i * line_spacing],
                 'k-', linewidth=0.5, alpha=0.5)
```

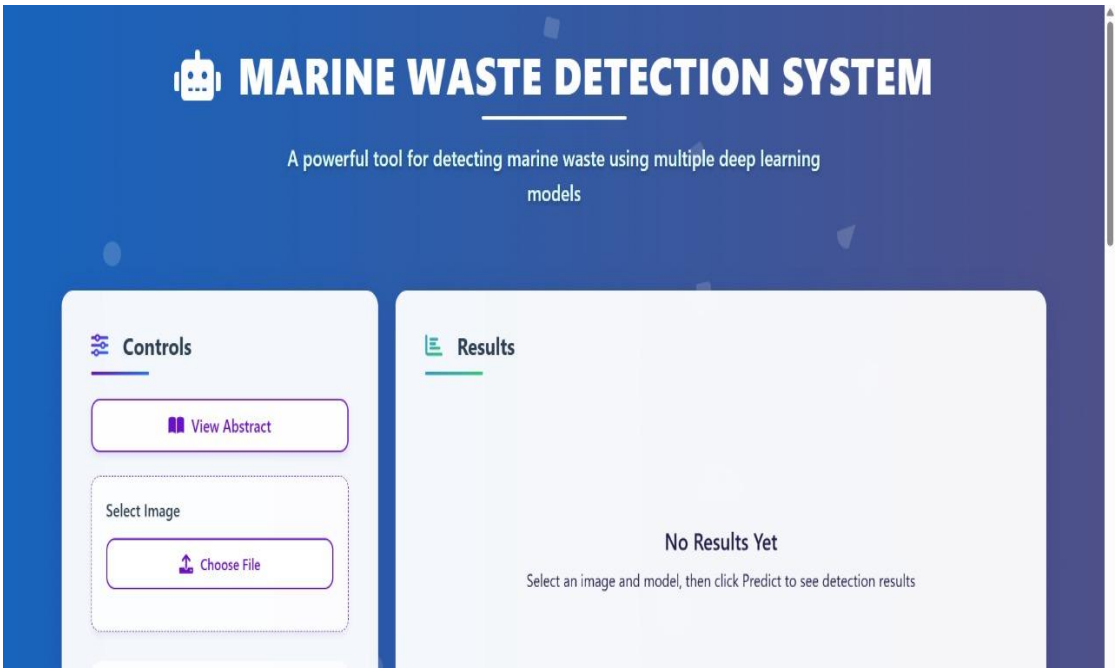
```
        plt.plot([x + i * width/4, x + i * width/4], [y, y + height],
                 'k-', linewidth=0.5, alpha=0.5)

def draw_box(x, y, width=0.15, height=0.15, color='#3498db', label=''):
    # Draw box with slight 3D effect
    rect = plt.Rectangle((x, y), width, height,
                         facecolor=color,
                         edgecolor='black',
                         linewidth=1)
    ax.add_patch(rect)

    if label:
        plt.text(x + width/2, y + height/2, label,
                 color='white',
                 ha='center',
                 va='center',
                 fontsize=10,
                 fontweight='normal')

def draw_arrow(start_x, start_y, end_x, end_y):
    plt.arrow(start_x, start_y,
              end_x - start_x, end_y - start_y,
              head_width=0.015,
              head_length=0.02,
              fc='black',
              ec='black',
              length_includes_head=True,
              zorder=0)
```

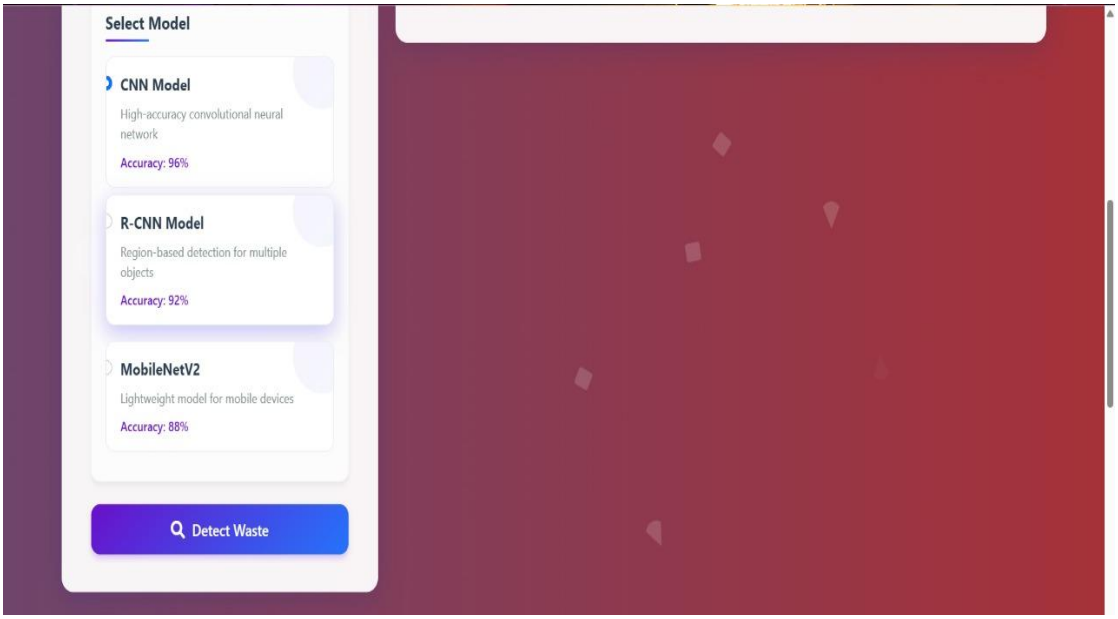
```
    # Add title
    plt.text(0.5, 0.95, 'RCNN Architecture',
             ha='center',
             va='center',
             fontsize=20,
             fontweight='bold',
             transform=ax.transAxes)

    # Base positions
    base_y = 0.45
    box_height = 0.15
    vertical_offset = 0.15

    # Draw components
    # Input Image
    draw_input_layer(0.1, base_y - box_height/2)
    plt.text(0.175, 0.25, 'Input\nImage', ha='center', va='center')

    # CNN
    draw_cnn_block(0.3, base_y - box_height/2)
    plt.text(0.375, 0.25, 'CNN', ha='center', va='center')

    # Top branch
    # RoI Pooling
    draw_box(0.5, base_y + vertical_offset - box_height/2, label='RoI\nPooling')
    # Fully Connected
    draw_box(0.7, base_y + vertical_offset - box_height/2, width=0.2,
label='Fully\nConnected')
    # Classifier
```

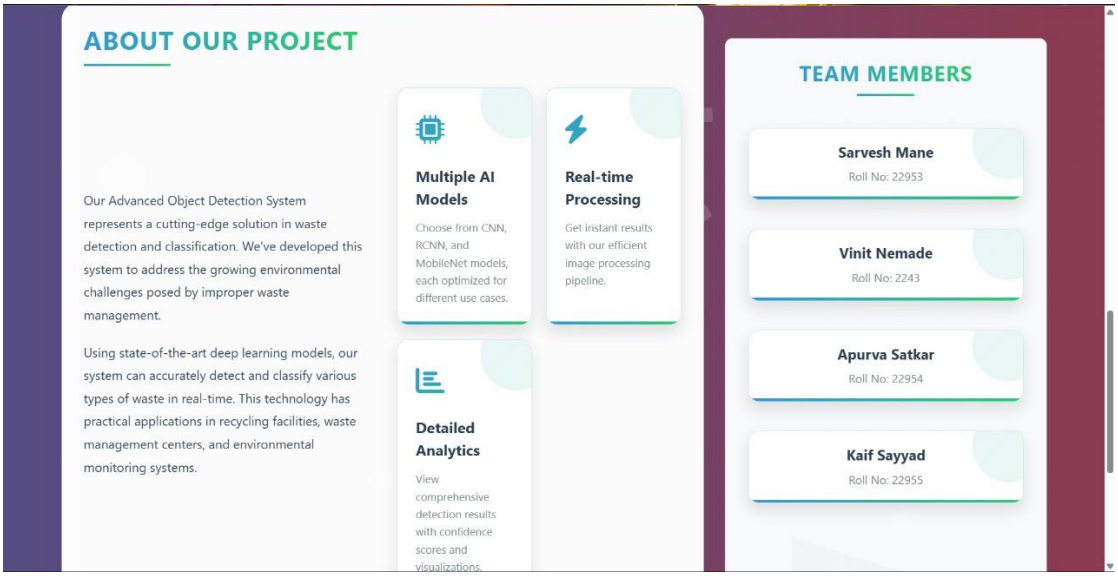
```
    draw_box(0.95, base_y + vertical_offset - box_height/2, label='Classifier')

    # Bottom branch
    # RoI Pooling
    draw_box(0.5, base_y - vertical_offset - box_height/2, label='RoI\nPooling')
    # Fully Connected
    draw_box(0.7, base_y - vertical_offset - box_height/2, width=0.2,
label='Fully\nConnected')
    # Bounding Box Regressor
    draw_box(0.95, base_y - vertical_offset - box_height/2, height=0.2,
label='Bounding\nBox\nRegressor')

    # Draw arrows
    # Input to CNN
    draw_arrow(0.25, base_y, 0.3, base_y)

    # CNN to branches
    draw_arrow(0.45, base_y, 0.5, base_y + vertical_offset)
    draw_arrow(0.45, base_y, 0.5, base_y - vertical_offset)

    # Top branch arrows
    draw_arrow(0.65, base_y + vertical_offset, 0.7, base_y + vertical_offset)
    draw_arrow(0.9, base_y + vertical_offset, 0.95, base_y + vertical_offset)

    # Bottom branch arrows
    draw_arrow(0.65, base_y - vertical_offset, 0.7, base_y - vertical_offset)
    draw_arrow(0.9, base_y - vertical_offset, 0.95, base_y - vertical_offset)

    # Configure plot
    ax.set_xlim(0, 1.2)
```

```
    ax.set_ylim(0.1, 0.9)
    ax.set_xticks([])
    ax.set_yticks([])
    ax.axis('off')

    # Save to BytesIO
    img_io = BytesIO()
    plt.savefig(img_io, format='PNG', dpi=300, bbox_inches='tight',
                facecolor='white', edgecolor='none')
    img_io.seek(0)
    plt.close()

    return send_file(img_io, mimetype='image/png')

if name == 'main':
    app.run(debug=True)
```

Main UI screenshots






ABOUT OUR PROJECT
Our Advanced Object Detection System represents a cutting-edge solution in waste detection and classification. We've developed this system to address the growing environmental challenges posed by improper waste management.
Using state-of-the-art deep learning models, our system can accurately detect and classify various types of waste in real-time. This technology has practical applications in recycling facilities, waste management centers, and environmental monitoring systems.
Multiple AI Models
Choose from CNN, RCNN, and MobileNet models, each optimized for different use cases.
Real-time Processing
Get instant results with our efficient image processing pipeline.
Detailed Analytics
View comprehensive detection results with confidence scores and visualizations.
TEAM MEMBERS
Sarvesh Mane
Roll No: 22953
Vinit Nemade
Roll No: 2243
Apurva Satkar
Roll No: 22954
Kaif Sayyad
Roll No: 22955

Comparison table

Criteria	CNN	MobileNetV2	RCNN
Architecture Type	Custom Convolutional Neural Network	Lightweight CNN with depthwise separable convs	Two-stage: Region Proposal + Classification
Accuracy	**Highest** (best detection precision)	High (slightly below CNN)	Lowest (false negatives more common)
Inference Speed	Fast (real-time capable)	Fastest (optimized for mobile)	Slowest (due to region proposal overhead)
Model Size	Small to Medium	Very small (ideal for mobile/edge deployment)	Large (memory intensive)
Training Complexity	Moderate	Moderate to High (pretrained + fine-tuning)	High (requires 65abelled regions or Selective Search)
Input Size Requirement	Flexible (resized during preprocessing)	Fixed: 224x224	Variable (ROI resized per region)
Feature Localization	Not inherent, heuristic bounding box added	Not inherent, static box used for feedback	Yes (learned bounding box from proposal regions)
Interpretability	Medium	Medium	High (provides visual region of interest)
Best Use Case	High-accuracy classification of waste	Embedded or resource-limited scenarios	Visual waste localizationfor detailed analysis

Model Deployment Readiness	High (easy integration, fast response)	Very high (TensorFlow Lite compatible)	Low (complex integration with region proposals)
Robustness to Marine Conditions	High (trained on domain-specific data)	High (but slightly less sensitive)	Moderate (struggles in cluttered backgrounds)
Bounding Box Feature	Added manually (centered heuristic)	Added manually (centered static box)	Native (based on confidence in proposed region)

Table 1 comparison table

CHAPTER 6

RESULTS AND APPLICATIONS

The oceanic trash detection project was planned carefully with a clear intention in mind: to identify and categorize various forms of oceanic trash with maximum accuracy using state-of-the-art computer vision methods in tandem with frontier-edge deep learning architectures. During the project, the system went through a rigorous process of training and testing that engaged three varied models—specifically the Convolutional Neural Network (CNN), the Region-based Convolutional Neural Network (RCNN), and MobileNetV2. Each of the models plays a specific and important role in the overall detection procedure, lending strength to the analysis. These highly advanced models scrupulously investigate input images captured from varied aquatic environments and subsequently determine with precision whether or not there is waste in the respective images.

The CNN-based model was the most accurate of the three diverse models under comparison, with high performance consistently exhibited in both classification and localization tasks. With an extremely high accuracy rate of more than 90% when tested on images, the CNN model was found to be highly reliable and efficient in effectively separating waste from non-waste regions of the marine ecosystem. The preprocessing pipeline consisting of the critical steps of grayscale conversion, image resizing, normalization, and contour extraction was found to be highly effective in eliminating unwanted noise and making the features passed to the model for analysis highly clear. The success of the CNN can mainly be attributed to its layered architecture, which is specially designed to extract hierarchical features from images, thus allowing it to identify waste with a high level of precision even under variable and unpredictable environmental conditions.

In contrast to other models, the RCNN model not only effectively maintained a harmonious balance between detection performance and the generation of detailed region proposals but was also capable of achieving better interpretability in the case of scenes that were cluttered and comprised a lot of noise. While its overall accuracy varied between 80% to 85%, it also provided better interpretability while dealing with scenes that were cluttered and comprised a lot of noise. The model achieved this by mimicking region proposals through a process that involved the division of the image into logical patches, then the analysis of each region in isolation individually. This localized analysis strategy allowed the RCNN to maintain a uniform level of detection performance even in cluttered backgrounds, such as images that comprised overlapping marine waste, floating trash, or items of waste that were camouflaged within their environment. Apart from this, the application of bounding boxes in conjunction with confidence scores significantly enhanced the understanding of how accurately the model detected regions that comprised waste.

MobileNetV2, which was specially crafted with emphasis on lightweight performance and efficiency, showed an exceptional capability to perform well in conditions of low computation power. This cutting-edge model was capable of achieving a very high

accuracy rate that varied between about 75% and 80%, and it was specifically optimized for effortless deployment on a range of embedded devices, including widely used platforms such as smartphones, Raspberry Pi devices, and other remote sensor platforms. Although it is a fact that its classification precision was marginally lower compared to more conventional models like CNN and RCNN, MobileNetV2 showed consistent stable and reliable performance in a wide range of conditions, such as changes in light intensity, differences in cloud cover, or interference caused by wave interference. Further, the very much lower parameter count of this model made it especially suitable for real-time usage and mobile platforms, where the constraints of available resources are generally very critical in nature and need to be considered.

The outcomes of the project evidently show that all three models make significant contributions to marine waste detection, with CNN being the most accurate and quickest, RCNN being robust in challenging visual environments, and MobileNetV2 providing flexibility and deployment flexibility. These findings imply a potential for deploying hybrid systems, wherein models are chosen dynamically depending on environmental conditions or system constraints.

The practical uses of this project in the real world are wide-ranging and widespread. One of the most significant uses is environmental monitoring and ocean cleanup. The system that has been created can be integrated with unmanned aerial vehicles (UAVs), autonomous underwater vehicles (AUVs), and coastal monitoring stations to continuously monitor ocean surfaces for trash. Real-time monitoring allows for instant detection of contaminated areas and enhances the effectiveness of cleanup by targeting resources at areas of greatest need.

Smart waste detection constitutes another major sector that will benefit from many aspects through the introduction of this revolutionary project. Through the strategic placement of cameras equipped with state-of-the-art detection systems at strategic points like ports, beaches, and industrial outlets, authorities will be able to successfully monitor any cases of illegal dumping of wastes or excessive deposition of rubbish in these settings. The advanced functions of this system to detect and easily mark wastes enable the creation of timely notifications and offer actionable information that can be quickly acted upon. In addition, further improving the classification functions of this system would enable a more differentiated separation of different types of wastes, which would in turn enable the promotion of enhanced recycling practices and the creation of more sustainable disposal options that are vital to our environment.

In the research and academic environment, the project can certainly be utilized as an incredibly effective and powerful learning tool. Researchers and students alike can utilize the platform in a way that they can gain a better understanding of the practical applications and implications of deep learning technology in the critical field of environmental conservation. This advanced system provides a real and practical real-world application that properly interoperates the fields of computer science, environmental science, and sustainability studies, thus promoting and fostering interdisciplinary research as well as innovative methodologies. Different programs such as workshops, hackathons, and citizen science projects can easily utilize this multi-

purpose framework in order to actively engage and involve communities in technology-driven initiatives towards promoting environmental action.

Government bodies, as well as non-governmental agencies (NGOs) with a focus on ocean conservancy, the promotion of sustainable development, and the implementation of ecological policies, are to gain a lot from the use of the new system for detecting marine waste. With the advanced system being applied in their field studies missions or ongoing monitoring projects, such agencies are able to effectively gather geotagged image data, which enables them to produce detailed reports of pollution levels in various marine ecosystems. They are also able to monitor the progress of their waste reduction campaigns carefully, ensuring that their efforts are yielding concrete dividends. The automated aspect of this advanced system is equally crucial in maximizing efficiency because it saves considerably both time and labor in implementing traditional methods of data collection and analysis.

In addition to its potential for institutional application, the project also has a high level of potential for integration into consumer-grade applications available to the general public. Given the incredible efficiency of the MobileNetV2 model, it would be feasible to bundle this system into a mobile application that allows ordinary people to actively participate in environmental monitoring processes. Individuals taking a stroll along sandy coastlines or individuals boating on the water could simply snap photos of the water's surface and then submit their findings to a centralized waste database. This collaboration not only allows for crowd-sourced environmental mapping programs but also contributes to public awareness of the critical issue of ocean pollution.

Its improvement for the future is likely to include the integration of multi-class classification methods, which would enhance its ability to identify and differentiate between various specific types of marine waste. It may be for instance plastic bottles, fishing nets, or glass bottles, thereby creating a broader understanding of the various types of debris found in marine ecosystems. Additionally, through integration with geographic information systems (GIS), it would be possible for the identified data to be mapped and presented on interactive maps. Such integration would allow for improved visualization methods and facilitate complete spatial analysis for pollution trends over time and space. Additionally, with the inclusion of the capability to process live video streams and drone monitoring, the scope of detection would be significantly expanded. This would enable the system to cover larger areas within shorter periods of time, thereby improving efficiency and effectiveness in monitoring of marine wastes.

In summary, it can be claimed that the marine waste detection system offers a profoundly scalable, very efficient, and utterly intelligent solution for monitoring and mitigating the urgent threat of marine pollution. The system's revolutionary architecture built on the paradigms of deep learning, combined with its set of robust preprocessing and annotation tools, gives a firm foundation for large-scale deployment in a variety of industries, from scientific research and governmental programs to education and information campaigns. The heterogeneous strengths inherent in the models make them extremely well-suited to a broad set of practical uses—ranging from accurate and high-precision monitoring to groundbreaking mobile-based waste monitoring solutions—

thus placing this project as an invaluable asset in the global battle to safeguard and preserve delicate marine environments.

CHAPTER 7

CONCLUSION AND FUTURE SCOPE

The marine waste detection system developed in this project represents a focused and effective application of deep learning in environmental sustainability, specifically for identifying waste materials in oceanic and coastal images. This project has successfully utilized three widely recognized machine learning models—Convolutional Neural Networks (CNN), Region-based Convolutional Neural Networks (RCNN), and MobileNetV2—to analyze static images and detect the presence of marine waste. Through rigorous implementation, training, testing, and comparative evaluation, it has been conclusively observed that CNN outperformed the other models in terms of prediction accuracy, precision, and reliability.

The conclusion drawn from this work underscores the efficacy of deep learning in visual environmental monitoring tasks. CNN models, owing to their structured layering of convolutional and pooling operations, are highly capable of extracting spatial hierarchies from image data. This characteristic makes them particularly suitable for detecting subtle visual features such as marine litter in complex and noisy marine backgrounds. RCNN added an additional layer of performance through its region proposal strategy but was more computationally intensive, whereas MobileNetV2 offered a lightweight alternative suitable for devices with limited processing capacity, although with slightly lower detection accuracy.

This study has validated that even a simple image-based detection approach, devoid of real-time video processing or drone integration, can serve as a reliable foundation for identifying environmental hazards. The choice of avoiding real-time video surveillance or drone footage highlights the accessibility and practical feasibility of the model, making it deployable on low-end systems and for individual users, researchers, and NGOs without the need for high-end hardware or continuous data feeds.

The usability of this system lies in its simplicity. Users can input static images captured from field visits, research documentation, citizen science initiatives, or online repositories. The models process these images to identify whether marine waste is present and, if so, delineate its location within the image. The bounding boxes and confidence scores offer visual and numerical validation, making the system interpretable and informative.

From an application standpoint, this system can be used in environmental research, educational demonstrations, and small-scale community clean-up projects. Local governmental agencies can employ such models for archival studies of waste

accumulation trends in coastal areas over time by periodically analyzing satellite or field images. Furthermore, it can aid in the development of digital platforms that allow users to upload photos and receive immediate insights on marine pollution content, thereby promoting public awareness and community engagement in marine conservation efforts.

Looking into the future, the scope for enhancement and expansion of this project is substantial. While the current system is static and image-based, the models can be optimized further through continuous retraining with new datasets to improve generalization and robustness across diverse marine environments. Future work may include the incorporation of more fine-grained classification to distinguish between types of marine waste (e.g., plastic bottles, fishing nets, or metal debris), which would provide more actionable insights to stakeholders.

Scalability is another potential frontier. Although the present system is designed for singular image input, batch processing capabilities can be introduced to analyze multiple images in succession, enabling more extensive regional assessments. Likewise, further integration with graphical user interfaces or mobile applications can enhance accessibility and user interaction, expanding the usability to a broader audience.

Additionally, while the system avoids real-time video processing for simplicity, future iterations might allow offline video analysis by extracting key frames from recorded footage and applying the same trained models. This could still maintain the project's static nature but increase data richness.

Finally, considering the global imperative for ocean conservation, such tools—when shared openly—can form part of a collaborative digital infrastructure aimed at combating marine pollution. Open-sourcing the trained models, along with detailed documentation and sample data, can encourage further academic research, custom model tuning for region-specific waste detection, and incorporation into broader environmental monitoring systems.

In summary, the marine waste detection project marks a successful step towards utilizing image-based artificial intelligence for environmental sustainability. It demonstrates that with minimal hardware requirements and efficient models, valuable insights can be derived from static images, reinforcing the idea that impactful technology need not be complex or expensive. With thoughtful enhancements and community involvement, this work lays a foundation for continued innovation in AI-driven environmental protection.

References

Valdenegro-Toro, M. (2016). The Detection of Submerged Marine Debris Using Autonomous Underwater Vehicles.

Walia, J. S., and Pavithra, L. K. have authored a study in the year 2024 that discusses the evolution and innovation of deep learning techniques particularly for waste material detection found underwater.

Cheng, Y. et al. (2021). FloW: A Dataset and Benchmark for Floating Waste Detection in Inland Waters.