TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

THAPATHALI CAMPUS

A Minor Project Report

On

Municipal Plastic Detection and Classification in Real Time Using YOLOv9 and Custom CNN model

Submitted by:

Jesis Upadhayaya	(22716)
Kamal Shrestha	(22717)
Pabin Khanal	(22726)
Prajwal Chaudhary	(22727)

Submitted To:

Department of Electronics and Computer Engineering

Thapathali Campus

Kathmandu, Nepal

May, 2025

TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

THAPATHALI CAMPUS

A Minor Project Report

On

Municipal Plastic Detection and Classification in Real Time Using YOLOv9 and Custom CNN model

Submitted by:

Jesis Upadhayaya	(22716)
Kamal Shrestha	(22717)
Pabin Khanal	(22726)
Prajwal Chaudhary	(22727)

Submitted To:

Department of Electronics and Computer Engineering

Thapathali Campus

Kathmandu, Nepal

In partial fulfillment for the award of the Bachelor's Degree in Computer Engineering.

Under the Supervision of

Er. Anup Shrestha

May, 2025

DECLARATION

We hereby declare that the report of the project entitled **"Municipal Plastic Detection and Classification in Real Time Using YOLOv9 and Custom CNN model"** which is being submitted to the **Department of Electronics and Computer Engineering, IOE, Thapathali Campus**, in the partial fulfillment of the requirements for the award of the Degree of Bachelor of Engineering in **Computer Engineering**, is a bonafide report of the work carried out by us. The materials contained in this report have not been submitted to any University or Institution for the award of any degree, we are the only author of this complete work, and no sources other than the listed here have been used in this work.

Jesis Upadhayaya (THA078BCT017) _____

Kamal Shrestha (THA078BCT018) _____

Pabin Khanal (THA078BCT027) _____

Prajwal Chaudhary (THA078BCT028) _____

Date: May, 2025

CERTIFICATE OF APPROVAL

The undersigned certify that they have read and recommended to the **Department of Electronics and Computer Engineering, IOE, Thapathali Campus**, a minor project work entitled **"Municipal Plastic Detection and Classification in Real Time Using YOLOv9 and Custom CNN model"** submitted by **Jesis Upadhyaya, Kamal Shrestha, Pabin Khanal and Prajwal Chaudhary** in partial fulfillment for the award of Bachelor's Degree in Computer Engineering. The Project was carried out under special supervision and within the time frame prescribed by the syllabus.

We found the students to be hardworking, skilled and ready to undertake any related work in their field of study and hence we recommend the award of partial fulfillment of Bachelor's degree of Electronics and Communication Engineering.

_____

Project Supervisor
Er. Anup Shrestha
Department of Electronics and Computer Engineering, Thapathali Campus

_____

External Examiner
Er. Gopal Karn
Assistant Professor, Department of Electronics and Computer Engineering, Pulchowk Campus

_____

Project Coordinator
Er. Saroj Shakya
Department of Electronics and Computer Engineering, Thapathali Campus

_____

Er. Umesh Kanta Ghimire
Head of Department
Department of Electronics and Computer Engineering, Thapathali Campus

May, 2025

COPYRIGHT

ACKNOWLEDGEMENT

This project is prepared in partial fulfillment of the requirement for the bachelor's degree in Computer Engineering. We owe our deepest gratitude to the Department of Electronics and Computer Engineering IOE, Thapathali Campus for allowing us to work on a minor project as a part of our syllabus. We would also like to express our gratitude to our supervisor, Er. Anup Shrestha, for his guidance.

The experience of working on this project will surely enrich our technical knowledge and also give us an experience of working on a project and also develop our team work skills to a great extent. We would also like to express our heartfelt thanks to Suwarna Lingden Sir for his continuous motivation and insightful perspectives, which greatly contributed to the success of this project. Additionally, we are grateful to our senior brothers, Prashant Raj Bista and Om Prakash Sharma, for their help and support. The experience gained from working on this project has significantly enhanced our technical knowledge and provided us with valuable insights into project development. Furthermore, it has fostered our teamwork and collaboration skills, which will be beneficial for our future endeavors.

Jesis Upadhayaya (THA078BCT017)
Kamal Shrestha (THA078BCT018)
Pabin Khanal (THA078BCT027)
Prajwal Chaudhary (THA078BCT028)

ABSTRACT

Plastic waste management is a global challenge, with more than 380 million tons of plastic produced annually with municipal plastic waste that includes a substantial portion of global pollution. Manual sorting of municipal plastic waste is difficult and expensive, necessitating the development of automated methods. Since it is possible to recycle certain types of plastic (PET can be converted into polyester material). Therefore, we must look at ways to separate this waste. The proposed system can detect municipal plastic waste and classify plastic types simultaneously (in real time). The system leverages deep learning models, specifically the YOLOv9 (You Only Look Once) object detection framework for the detection of plastic waste, with a custom-designed Convolutional Neural Network (CNN) for plastic classification. The YOLO model is able to detect plastics from provided images and videos. The project leverages the YOLOv9s model. Detection model obtained Precision 0.95774, Recall 0.97038, Mean Average Precision mAP@50 0.97891 and mAP@50-95 0.90924. Validation losses are minimized (Box Loss: 0.38921, Class Loss: 0.24097, dfl Loss: 0.97034), showcasing strong learning. The CNN model is able to separate plastics into categories such as PET, PP, HDPE, and PS commonly found in household waste with test accuracy of 98.4%.

Keywords: CNN, Deep Learning, Plastic waste management, Recycling, Sustainable waste management.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

CNN	Convolutional Neural Network
HDPE	High Density Polyethylene
PET	Polyethylene Terephthalate
PP	Polypropylene
PS	Polystyrene
ReLU	Rectified Linear Unit
SiLU	Sigmoid Weighted Linear Unit
SSD	Single-Shot Detector
YOLO	You Only Look Once
UHT	Ultra High Temperature
MS COCO	Microsoft Common Objects in Context
WSGI	Web Server Gateway Interface
ONNX	Open Neural Network Exchange

1. INTRODUCTION

1.1. Background

Plastics are much more complex than most people realize. Waste management, particularly municipal plastic waste, has become a major environmental challenge. Municipal plastic waste management has emerged as a significant environmental challenge in recent years. In 2024, the world is expected to generate 220 million tons of plastic waste, with 70 million tons ending up in nature. Municipal plastic waste, which includes household and commercial plastic products, contributes significantly to urban pollution. As plastics do not biodegrade, they accumulate in landfills and oceans, breaking down into harmful microplastics. The need for efficient and scalable recycling solutions has become critical.

One of the main reasons we have such high levels of wastage today is the ineffective and often careless way that materials are handled. That's where the 3R Initiative steps in, promoting the ideals of reducing, reusing, and recycling on a global scale. The goal is to build a society that gets the most out of its resources while generating as little wastage as possible. Waste reduction starts with making intelligent choices i.e. buying only what's needed and foregoing those that will end up in the landfill in a short while. Reusing gives products a longer life by using them more than once instead of throwing them away. Recycling transforms already used material into a new product, either during the manufacturing process or after consumers are through with them. Recycling at the consumer level entails processing the materials in such a way that they can be reused, generally with some change in their form. For recycling to take place efficiently, household waste has to be sorted out properly for e.g. plastics, metal, organic waste, paper, and glass have to be segregated. Yet not everything within these groups can be recycled, and so the more advanced technology needs to come into play in order to sift them out and identify what can actually be re-used. This is especially important with plastics, where slight differences in type will affect whether or not they will be recyclable.

This project aims to address this issue by developing a deep learning-based automated system for plastic waste management using YOLOv9, a powerful object detection model, while a custom-designed Convolutional Neural Network (CNN) model is used

for classifying different types of plastics based on their visual features. YOLOv9 is chosen for its real time processing capabilities and high accuracy in detecting objects within images. Custom-designed Convolutional Neural Network (CNN) is used for the categorization of different types of plastic based on visual features. The system utilizes deep learning and image processing techniques to classify plastic waste such as PET, HDPE, PP and PS.

1.2. Motivation

Our motivation comes from a shared responsibility to contribute to the well -being of our environment, specifically in the management of municipal plastic waste. Inadequate elimination of plastic waste leads to serious environmental consequences, including obstructed drainage systems, increased landfills and contamination of natural water bodies. As aspiring engineers, we recognize the urgent need to address this problem with technological advances.

The vast potential of deep learning and computer vision inspires us to develop an innovative solution that automates the municipal classification of plastic waste. By taking advantage of the detection and classification promoted by AI, our goal is to improve waste management efficiency, reduce manual classification dependence and minimize plastic pollution. This project is not just a technological advance, but it is about creating a tangible impact on society by promoting sustainable practices in urban waste management.

By aligning avant-garde technology with environmental preservation, we strive to develop a system that enables municipalities, improves recycling efficiency and fosters a cleaner and healthier world for coming generations.

1.3. Objectives

- To develop an automated system for accurate detection of municipal plastic waste and classify its type in real time.
- To enhance recycling efficiency by implementing deep neural networks.

1.4. Problem Statement

Current methods of sorting urban plastic waste are inefficient, expensive and difficult to scalable. Traditional techniques, such as manual sorting and basic visual classification using cameras, lack the accuracy needed for effective waste management. More advanced methods, including scanning triangulation with 3D cameras, spectral imaging by near infrared (NIR) and hyperspectral imaging (HSI) and medium infrared (MIR) spectroscopy, provide improved accuracy, but remain disproportionately expensive and require specialized equipment. In addition, these methods often include crushing waste materials, making it more difficult to classify small particles or mixed plastics.

As a result, the level of contamination in recycling streams remains high, reduces the quality of recycled materials and limits the efficiency of waste management processes. The municipal waste sorting facilities seek to cope with the growing volume of plastic waste and emphasize the need for an efficient, cost -effective and scalable solution.

The aim of this project is to develop a controlled AI system that is able to detect and classify urban plastic waste in real time without requiring complicated pre -processing or grinding. Using deep learning models, this system seeks to increase recycling efficiency, reduce contamination and contribute to more sustainable and efficient urban waste management procedures.

1.5. Scope of Project

1.5.1. Project Scopes

This project focuses on developing a deep learning solution for automated detection and the classification of municipal plastic waste using YOLOV9 and a personalized CNN model. The main objectives include the design and training of a model that can efficiently identify and classify common municipal plastics (PET, HDPE, PP and PS) in various real world environments. The system will be implemented mainly in the last waste management stage, focusing on waste segregation to guarantee high precision, efficiency and real -time processing.

- Plastic classification: The system will precisely classify PET plastics, HDPE, PP and PS commonly found in municipal waste, improving recycling processes.

- Real time classification of municipal plastic waste: The model will work in real time, which makes it appropriate for classification applications of industrial and domestic waste in a Municipality.

- Integration with existing waste management systems: the project will explore ways to integrate the model with the existing waste classification infrastructure.

- Minimization of human intervention: The system will focus on reducing manual classification dependence, decreasing pollution in recycling current and improving the general efficiency of the municipal management of plastic waste.

1.5.2. Project Applications

- Municipal Waste Management: As our project is able to distinguish recyclable plastics from commonly used plastic i.e municipal plastics making easier waste management.

- Plastic Recycling Plants: The system can be deployed in recycling facilities to automate the sorting process, improving the efficiency and accuracy of plastic separation.

- Environmental Impact Reduction: By improving recycling efficiency, the system will help reduce plastic pollution, contributing to cleaner environments and sustainable waste management practices.

- Industrial Waste Sorting: Large-scale industries that generate significant plastic waste can use this system for automated sorting, reducing manual labor and contamination in the recycling process.

1.6. Report Organization

The report begins with an Introduction section providing background information, objectives, and scopes. Following this, the Literature Review presents a synthesis of existing research relevant to the project. The Requirement Analysis section delves into project requirements, including hardware and software needs along with feasibility study. System Architecture and Methodology elucidates the system's design, including block diagrams, flowcharts, and theoretical algorithm explanations. Implementation Details cover practical aspects like dataset collections, model details . Results and Analysis detail project outcomes, encompassing YOLO and CNN model performances along with the results of the integrated pipeline. Future Enhancements propose potential improvements or extensions. The Conclusion offers a summary of findings and recommendations for future work, while Appendices contain supplementary materials supporting the main content of the report.

1.7. Project Limitations

In distinguishing between Plastic and Glass: There could be difficulty in distinguishing between transparent plastic and glass using CNN-based models. We can use some mechanical ways here like glass sinks in water. First of all, we need to sort out plastic and glass in level 1. In level 2 of sorting, we use the above mechanical way for differentiating between the two.

- Environmental Variability: The model's performance might vary based on external factors like lighting, image resolution, or the angle of plastic waste in real-world conditions.
- Generalization to All Plastic Types: While the system will focus on the most common types of plastic, some lesser-known plastics or complex mixed plastic waste might pose challenges for accurate classification.

This project aims to tackle these challenges while providing a scalable solution for Municipal plastic waste management, focusing on efficiency, accuracy, and environmental sustainability.

2. LITERATURE REVIEW

Deep learning neural networks have multiple implications in domains such as object detection and classification. The performance of deep learning models depends on their architecture and use of proper datasets. We have reviewed the following works relating to the subject as follows.

[1] Rahman et al. designed an automated waste management system that integrates deep learning with IoT technology. In this system, deep learning models classify waste, while IoT handles the practical disposal processes. Their study also compared various CNN models to identify the most efficient one, emphasizing the importance of optimizing for both accuracy and computational efficiency.

[2] Hassan et al. (2020) investigated the use of advanced data augmentation techniques to improve dataset diversity. By employing methods like synthetic image generation and adaptive augmentation, they enhanced model generalizability, especially for underrepresented plastic types such as PP and PS. These techniques help mitigate the common issue of dataset imbalance in waste classification tasks.

[3] Deep learning models, such as YOLO, ResNet, and SSD, have revolutionized waste detection by automating traditionally manual tasks. Models trained on datasets like TrashNet and TACO have demonstrated strong performance, with YOLOv5 and YOLOv8 excelling in real-time detection and handling challenges like occlusion and varying illumination. However, gaps persist, particularly the need for more comprehensive datasets and refined feature extraction techniques. Advances in segmentation and feature extraction are helping bridge these gaps, paving the way for practical waste management applications.

[4] A state-of-the-art review published by IEEE addressed the complexities in plastic waste detection. It stressed the need for algorithms that combine real-world data with advanced techniques, such as transfer learning, to improve model performance and scalability. The study argued for the integration of adaptive systems that can better respond to dynamic environmental conditions.

[5] Recent work in plastic waste management has shifted toward using deep learning and machine vision to overcome the limitations of traditional methods like near-infrared spectroscopy (NIRS), which struggles with plastics like PET and PET-G. For example, YOLOv8 has achieved over 91.7% accuracy by leveraging visual features such as texture and shape. These systems often combine automated sorting mechanisms, such as pneumatic sorting, to enhance efficiency, though challenges like lighting variability and dataset limitations remain. Nonetheless, advancements in deep learning and dataset quality provide scalable solutions for improving plastic recycling rates.

[6] Ramos, Lopes, and Mendonça reviewed the use of machine learning in waste classification, emphasizing the effectiveness of CNN-based models, particularly YOLO and SSD. They suggested that the development of benchmark datasets would be crucial for advancing the field and facilitating comparisons across different approaches.

[7] Guezouli et al. (2022) developed a system utilizing CNNs to detect and classify plastic waste, aiming to enhance recycling efficiency. It is trained on a vast dataset, achieving an impressive accuracy rate of 97%. This shows that there is a potential for neural networks to aid in plastic segmentation and recycling. However, the architecture they have used is rather simple and leaves room for exploration of better architectures. Our approach is to create a more lightweight model in order to assist real-time classification of plastics.

[8] "A Deep Learning Approach to Manage and Reduce Plastic Waste in the Oceans" by El zaar et al. (2022) explains the utilization of deep learning techniques to identify plastic waste from images in order to curb ocean pollution. The authors have applied two transfer learning techniques using CNNs: using pre-trained CNNs as a feature extractor with an SVM classifier and fine-tuning pre-trained CNNs. These approaches are validated on challenging object discovery datasets and texture classifying datasets. The paper illustrates that deep learning may be utilized in the task of managing plastic waste but without particular measures of performance and straightforward information about datasets.

3. REQUIREMENT ANALYSIS

3.1. Hardware requirements

We used both google colab and kaggle to train our model. While starting the training process we used google colab for training but due to limited runtime for T4 GPU we shifted our training process to kaggle which handed us more GPU runtime i.e. 12 hrs a day enabling longer training sessions.

Google colab

Table 3-1: Google Colab's Hardware Specification

Hardware Components	**Details**
CPU	Intel Xeon 2-core 2.00 GHz
GPU Model	NVIDIA Tesla T4
Number of GPUs	1
CUDA Cores Per GPU	2560
Clock Speed (GPU)	585 MHz (Base) / 875 MHz (Boost)
VRAM (GPU)	15 GB
Memory Bandwidth(GPU)	300 GB/sec
RAM	12.7 GB
Storage	112.6 GB SSD

Kaggle

Table 3-2: Kaggle's Hardware specification

Hardware Components	Details
CPU	Intel Xeon 2-core 2.2GHz
GPU Model	NVIDIA Tesla T4, P100
Number of GPUs	1,1
CUDA Cores per GPUs (P100, T4)	3584, 320
Clock speed of GPUs (P100, T4)	1.30 GHz (base) / 1.48 GHz (boost) 1.60 GHz (base) / 1.77 GHz (boost)
VRAM (GPU)	16 GB HBM2, 16 GB GDDR6
Memory Bandwidth (GPU)	720 GB/s, 320 GBs
RAM	16 GB
Storage (working directory)	20 GB

The Kaggle T4 GPU offers significant computational power, making it well- suited for handling the ferocious calculations involved in training large- scale deep literacy models. This GPU, grounded on the NVIDIA Tesla armature, provides ample memory and processing capabilities to efficiently train our discovery models(YOLOv9) and Bracket model for sufficient ages. By exercising the Kaggle T4 GPU, we can accelerate the training process, reduce training times, and achieve optimal performance for this design.

3.2. Software requirements

3.2.1. TensorFlow

TensorFlow is an open- source deep literacy library developed by Google. It offers a wide range of tools, libraries, and community support for structure and training machine literacy models. TensorFlow is employed in this work to make and train a customized CNN model to classify plastics. It offers inflexibility in neural network design and GPU acceleration, making it accessible to speed up the training process.

3.2.2. Python

Python is a widely used, high-level, open-source programming language, especially in machine learning, deep learning, and data science. It is used as the core language for the entire system, doing everything from data preprocessing and training the models to integrating YOLOv9 and the custom CNN. Its simplicity and wide support from libraries make it ideal for developing end-to-end AI applications.

3.2.3. Roboflow

Roboflow is an end-to-end platform for managing computer vision datasets. It provides tools for data annotation, augmentation, and preprocessing. Roboflow is used for annotating plastic waste images with bounding boxes (for YOLOv9 training) and organizing the dataset into classes (for CNN training). It also helps in augmenting the dataset to improve model generalization.

3.2.4. Google Colab

Google Colab is a cloud-based computing platform that offers free GPU and TPU access, making it ideal for deep learning model training. It is compatible with Jupyter notebooks and well integrated with Google Drive, making data storage and sharing convenient. Colab is utilized in this project for training both YOLOv9 and CNN models, leveraging its GPU for faster training and collaboration features for team-based working.

3.2.5. Kaggle

Kaggle is an online community and platform for data science and machine learning. It provides a rich repository of datasets and a collaborative environment that helped us in training our model. It offers powerful tools and resources for data exploration, model training, and evaluation, which were crucial for the success of our project.

3.2.6. Git/GitHub

Git is a version control system, and GitHub is a cloud-based platform for hosting and sharing code repositories. Git/GitHub is used for version control and collaboration among team members. It ensures that all code changes are tracked, and the project can be easily shared and updated.

3.2.7. Ultralytics

Ultralytics is a library mainly working on implementations around the YOLO object detection models. It serves users and developers in training, benchmarking, and deploying state-of-the-art computer vision models using a well-thought-through architecture with the most minimal of setups. Powered by PyTorch, it enables users to plug in their model, select and configure apps as needed, deploy on any device, cloud, or codec directly from a provided configuration menu option.

3.2.8. Flask

Flask framework is a lightweight WSGI web application. It is employed for making the web server and processing HTTP responses and requests. Flask offers us the basis for our web application and enables us to build routes, render templates, and process user input. It is necessary to serve the web interface where users can upload images and see inference results.

3.2.9. Requests

Requests is a simple HTTP library for Python. It is used for making HTTP requests. Requests provides a simple and efficient way to make HTTP requests, which can be useful for interacting with external APIs and services.

3.2.10. OpenCV

OpenCV is a computer vision library that contains a lot of functionalities for image and video processing. It's utilized here to read, process, and display image and video inputs. Some of the OpenCV functions utilized are reading images, resizing, and drawing bounding boxes. It serves as the key handler for visual data and presenting the inference results in an effective way.

3.2.11. Matplotlib

Matplotlib is a plotting library for Python. It is used to create visualizations, such as displaying cropped images with labels. Matplotlib allows us to create visualizations of the inference results, such as displaying cropped images with their predicted labels. This is important for presenting the results in a user-friendly manner.

3.2.12. NumPy

NumPy is a library for numerical computing in Python. It is used for array operations and mathematical functions. NumPy provides efficient array operations and mathematical functions that are essential for preprocessing images and handling numerical data during model inference.

3.2.13. Pandas

Pandas is a data manipulation and analysis library. It is used for handling and processing data in tabular form. Pandas provides powerful data structures and functions for manipulating and analyzing data, which are useful for managing and processing the results of the inference.

3.3. Feasibility Study

This study explores the feasibility of building a system to detect and classify plastic waste using YOLOv9 for object detection and our customized CNN for classification. The goal is to identify four types of plastic i.e. HDPE, PET, PP, and PS using a camera. The system will highlight detected plastics with bounding boxes, displaying their type and confidence levels. Designed as a college minor project, the focus is on achieving accuracy, efficiency, and educational value.

3.3.1. Technical Feasibility

The system leverages two powerful AI models: YOLOv9 for object detection and our CNN for classification. The YOLO model is well documented, widely used, and optimized for tasks like this. Implementation relies on Python programming and open source tools like TensorFlow, paired with a GPU enabled system.Training data can be sourced from public repositories or created by collecting and labeling images, ensuring a robust dataset for accurate predictions.

3.3.2. Operational Feasibility

Once deployed, the system is designed to run with minimal manual input, making it ideal for scenarios like waste management. By combining detection and classification into one workflow, it ensures smooth operation and precise identification of plastic types. This approach simplifies categorization tasks and adds real-world applicability to the project.

3.3.3. Time Feasibility

The expected goals for the designated period are met successfully. Data annotation and preprocessing were completed in the first two weeks. The rest of the time was invested in designing, training and the models. The remaining tasks include improving the performance of the models, integration of the two models in a single pipeline for seamless detection and classification and deployment of the system in a usable form.

4. SYSTEM ARCHITECTURE AND METHODOLOGY

4.1. System Block Diagram

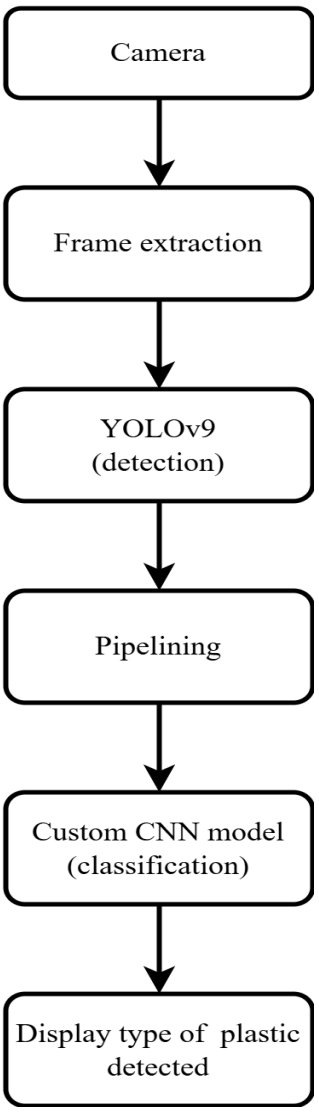


Figure 4-1: System Block Diagram

Our system comprises of a camera and two models for plastic detection and classification i.e. YOLOv9 and a custom CNN model. The proposed system takes real-time images as input from the camera. The input from the camera is passed on to the YOLOv9 frame by frame. The output is received which contains bounding boxes for individual plastics detected in the image. The exact bounding boxes(ROIs) are cropped and resized to meet input requirements of our CNN model. It classifies the images into individual plastic types. The outputs from both models are then combined to get the final output.

4.2. Flowchart

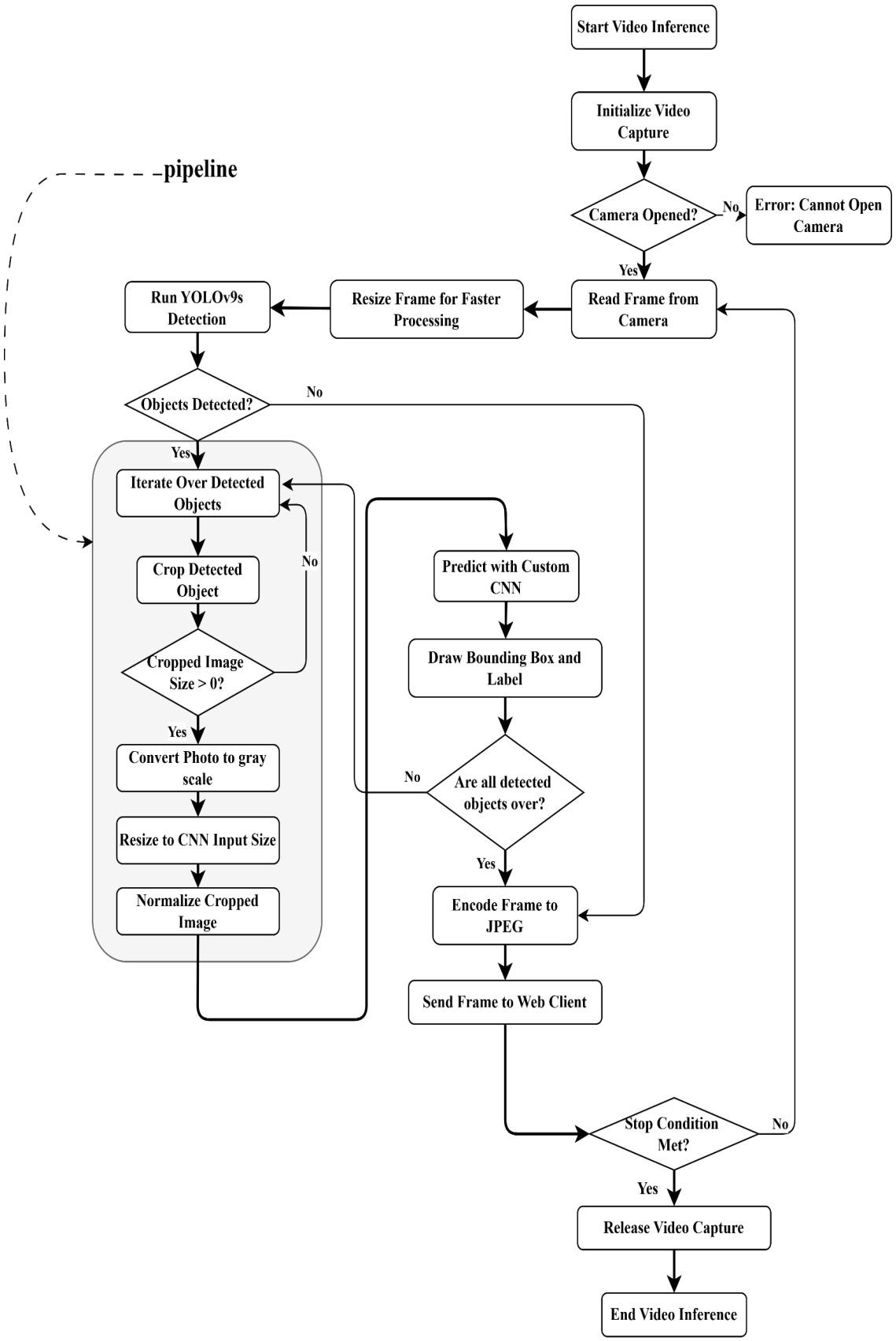


Figure 4-2: Video inference flowchart

4.2.1. Video Inference

The video inference process is initiated by taking live frames with a camera. It views each frame in live time to detect and classify objects in each frame and displays results to the user. Initially, an OpenCV video capture object is created. This app utilizes the camera to initiate reading frames. All frames get resized to simplify processing by the computer system. The resized frame is passed through the YOLO object recognition system to detect and classify objects in the frame. Objects detected get clipped from the frame, and each clipped photo is processed and tagged by a Convolutional Neural Network (CNN) system. Results, in the form of bounding around objects and class names, get superimposed over the original frame along with the confidence score. This frame is saved and transmitted to the web client to be viewed. This process continues until stopped by the user. Following this, the video capture object is destroyed, and processing is complete.

Pipeline:

The pipeline connecting YOLO and CNN is a fundamental component of the video inference process. YOLO is responsible for detecting and localizing objects within each frame. It produces a series of bounding boxes with coordinates of detected objects. A corresponding region is cropped from the original frame for each detected object. The cropped image is converted to the grayscale color space according to the expectations of the CNN model. Next, the cropped image is resized to the required input size for the CNN model. Afterward, the pixel values of the cropped image are normalized to get aligned with the training data. The preprocessed image is finally fed into the CNN model for class prediction. Annotating the original frame with bounding boxes and labels based on the predicted class label and confidence score ensures that the strengths of both YOLO (real-time object detection) and our Custom CNN (accurate classification) provide accurate yet efficient results in the video inference.

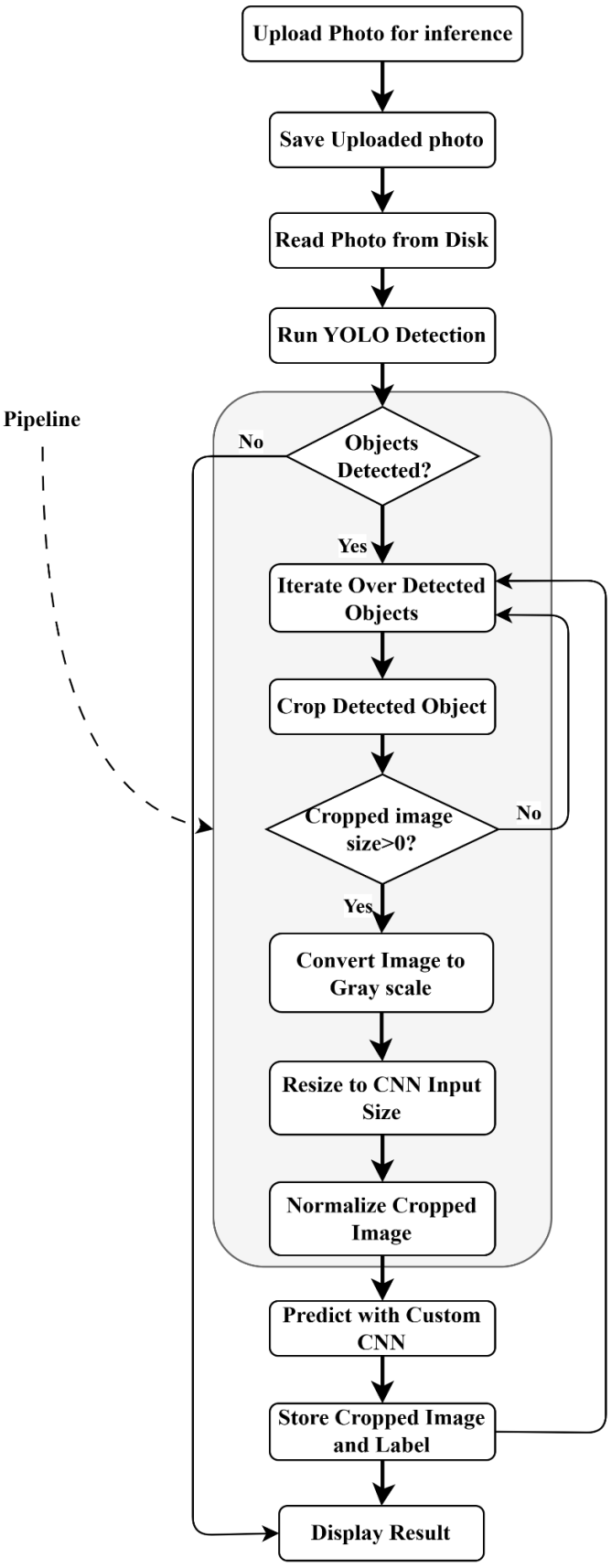


Figure 4-3:Photo Inference Flowchart

4.2.2. Photo Inference

To do photo inference, an input image must be uploaded and classified through the detection of objects within it; processed image files are returned to the user as results. A user uploads a photo for inference. This uploaded photo is written onto the disk for persistence and easier retrieval. Afterward, this saved photo is read from the disk.The saved photo is read using CV2 read and passed on to YOLO object detection model for object detection and localization within the image. If YOLO detects any object, this process iterates over every detected object.

The pipeline between YOLO and CNN aims to leverage the strengths of both modules in the execution of accurate object detection and classification. YOLO is employed first i.e. the input image is first processed for object detection using YOLO. YOLO processes the whole image and outputs the bounding boxes such that the coordinates of all the detected objects are positively identified. The bounding boxes serve as bases for cropping from the original image. Each cropped-out image is converted to grayscale color format and then resized to 224*224 size to fit the input of CNN. To ensure compatibility between the input and training data, pixel values are normalized. The preprocessed images are fed into a CNN for class prediction. The predicted class labels with confidence scores are used to annotate the original image via bounding boxes. The cropped image is also retained along with the predicted label. If no objects were detected by YOLO, then the blank result is displayed. The results of the cropped images and predicted labels were finally displayed to the user.

4.2.3. Image processing from camera

To process an image from the camera for YOLO input, the captured frame is first resized to match the dimensions required by the YOLO model. The image is then converted from BGR format, commonly used by OpenCV, to RGB format, which is the standard input format for YOLO models. These preprocessing steps ensure that the image is properly formatted and optimized for inference by the YOLO network.

4.2.4. Object Detection

Object detection is a key computer vision task that deals with the dual problems of object recognition in visual content and their exact locations. As opposed to

straightforward classification tasks that just identify if items are present in an image, object detection is concerned with the precise localization of every object by drawing bounding boxes around them. Prominent frameworks like R-CNN, Fast R-CNN, Faster R-CNN, and YOLO use convolutional neural networks (CNNs) to classify objects while utilizing regression techniques to precisely forecast the coordinates of all detected objects.

4.2.5. Image Classification

Image classification is the foundation of the majority of computer vision tasks. The aim is to assign an image to a single label from a finite set of categories. CNN-based models are widely applied to this task, as they excel at picking up spatial and pixel-level features.These models tend to be trained on huge datasets such as ImageNet, where images can be classified into numerous classes such as objects like mice, pencils, keyboards, and animals. YOLO models are distinct in that they are real-time object detection models that classify and detect objects in a single pass of the image. This is much better compared to previous multi-stage approaches, which makes YOLO extremely fast and efficient.

YOLO enables the simultaneous detection and classification of objects in real-time applications. YOLO employs a deep neural network that takes the entire image in a single pass, enabling the simultaneous detection of all the objects in the scene. The approach is to divide the image into a grid of cells, where each cell in the grid predicts multiple bounding boxes and corresponding confidence scores. YOLO also calculates class probabilities, which enable the model to decide the objects it detects.

YOLO works on the principle of using one convolutional neural network which divides the input image into a grid layout that is structured. Each grid cell is responsible for predicting a specified number of bounding boxes as well as the probability for all object classes. These predictions enable the system to effectively and efficiently calculate both the localization and classification of various objects.

4.3. YOLOv9

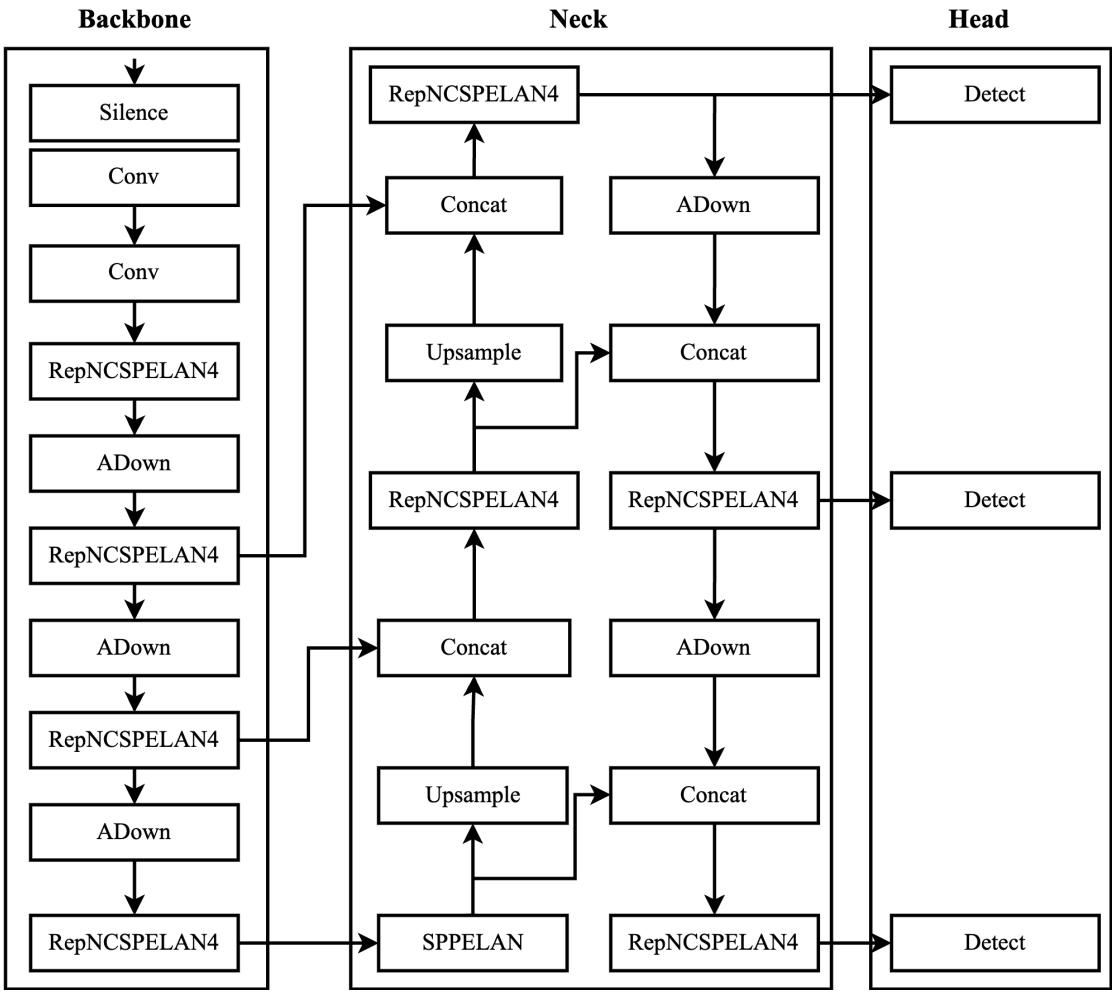


Figure 4-4: YOLOv9 Architecture

The architecture consists of three primary parts: the backbone, neck, and head. The backbone, typically a Convolutional Neural Network (CNN), is used to extract hierarchical feature maps from the input image capturing low, mid, and high-level features. These feature maps are then passed to the neck, which aggregates them using structures like the Feature Pyramid Network (FPN). This combined representation is forwarded to the head, which handles both object classification and bounding box prediction. The head can be either a single-stage dense predictor like YOLO or SSD, or a two-stage sparse detector such as those from the R-CNN family.

4.3.1. Backbone

The backbone is responsible for extracting features from the images. It employs convolutional layers, specialized blocks like RepNCSPELAN4 and ADown (Adaptive downsampling) systematically expanding the receptive field while maintaining computational efficiency.

Convolutional Layer:

The convolutional layer is fundamental in the YOLOv9 backbone for downsampling feature maps and extracting higher-level features. This layer operates on the input image, reduces spatial dimension and captures global information from the input image.

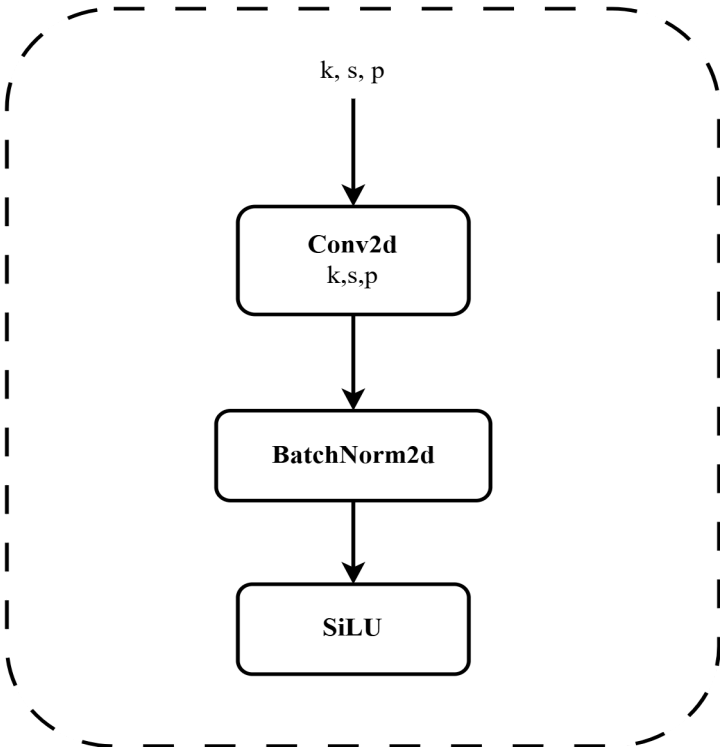


Figure 4-5: Block Diagram of Convolution Layer

Kernel Size (k): A kernel is a filter matrix that extracts local features from the image. The kernel slides over the image and calculates dot product of the pixels within the kernel window with the kernel resulting in a feature map. A kernel of size 3x3 means each filter covers 3x3 pixel area in the input image.

Stride (s): A stride of 2 indicates the filter moves 2 pixels at a time across the input. This effectively reduces the spatial dimensions of the output feature map by half, contributing to downsampling and expanding the receptive field with fewer computations.

Padding (p): Padding of 1 adds a single pixel border around the input, allowing the convolution operation to include edges of the input image, thereby preserving spatial dimensions after the convolution.

RepNCSPELAN4:

The network utilizes RepNCSPELAN4 blocks i.e. repeated layers that include NCSPELAN units (Normalized Cross Stage Partial with Efficient Large Kernel Attention). These blocks are designed to improve gradient propagation and enhance feature fusion throughout the deeper layers of the network.

It splits the input into two parts, processes each part separately with multiple RepNCSP blocks and then concatenates the outputs before passing them through the final convolutional layer. CSP connections facilitate information flow between different stages of the network. ELAN enhances feature representation through attention mechanisms. RepNCSPELAN4 enhances feature representation by combining efficient feature extraction with attention mechanisms. It enables the network to capture and emphasize important features relevant to object detection tasks. Figure 4-6 is the illustration of RepNCSPELN4 block.

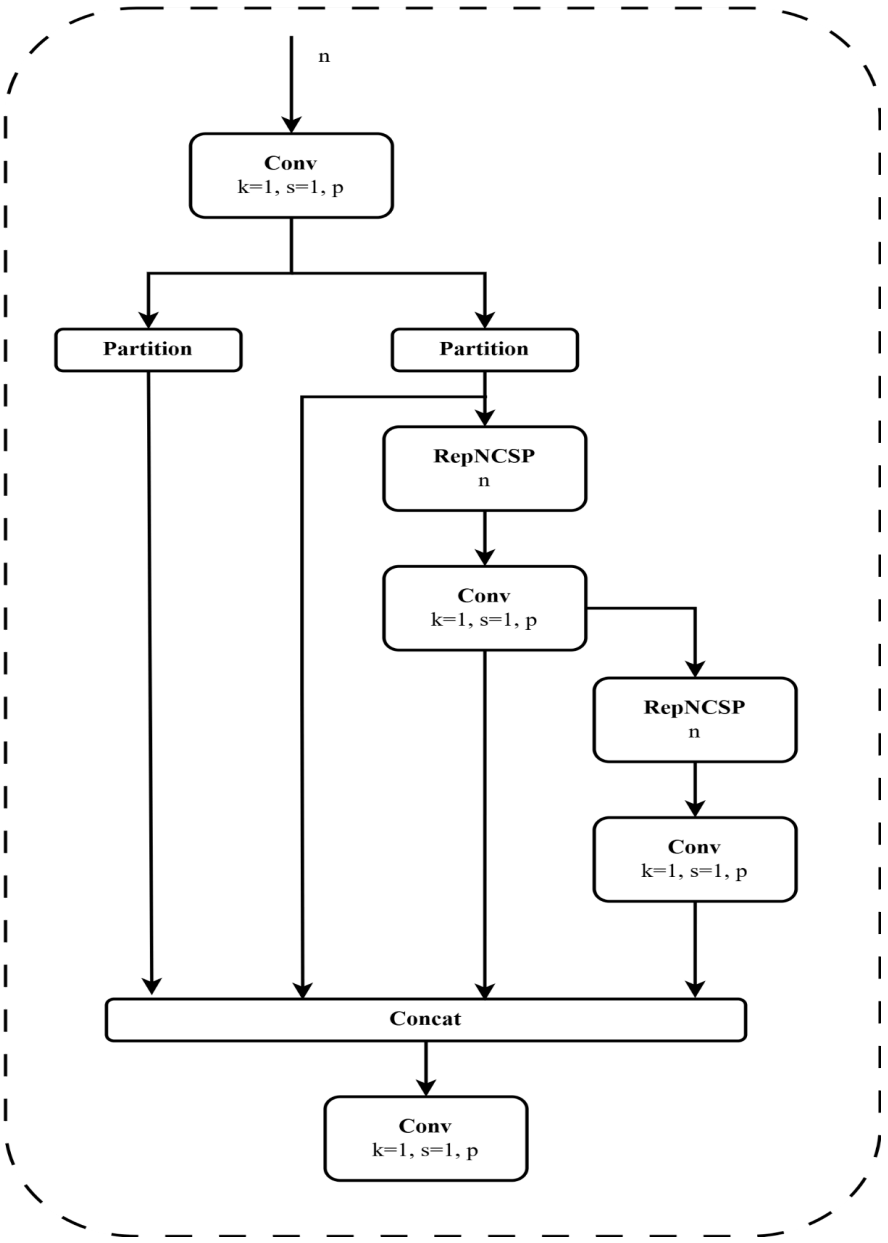


Figure 4-6: Block Diagram of RepNCSPELAN4

ADown:

This block employs both average pooling and max pooling for downsampling, which reduces computational complexity since pooling layers are parameter-free.

The ADown structure is asymmetrical downsampling block: it takes the input tensor, applies average pooling, then splits the result. One path applies max pooling followed by a 1×3 convolution, while the other performs a 3×1 convolution. The outputs of

both paths are concatenated to form the final result. Figure 4-7 is the illustration of ADown.

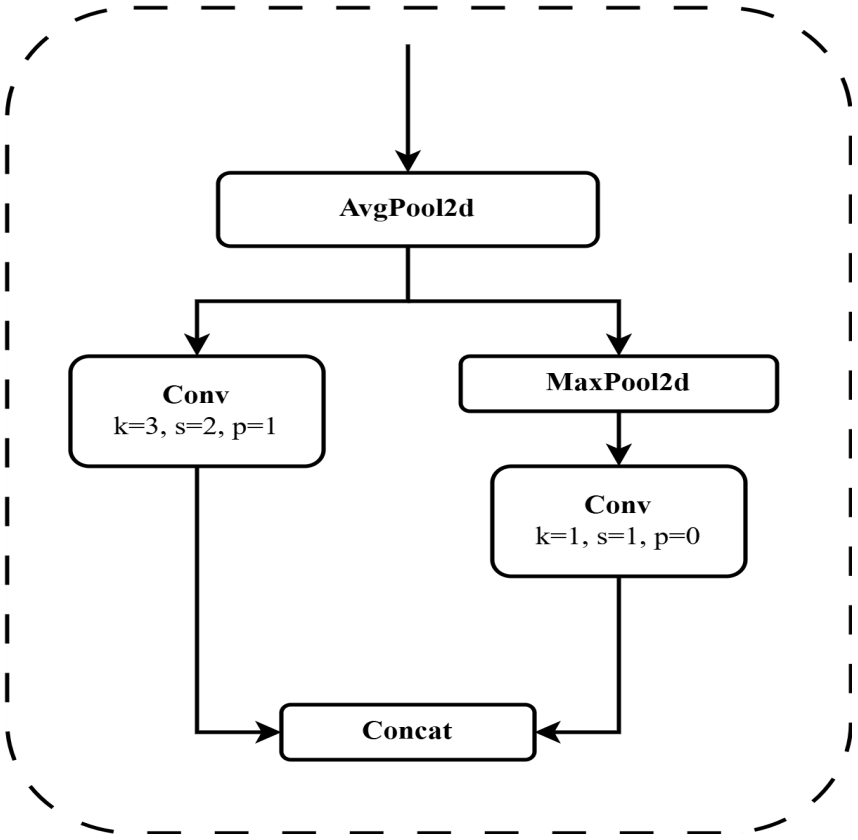


Figure 4-7: Block Diagram of ADown

4.3.2. Neck

The neck is an essential stage in the YOLO architecture where feature refinement, aggregation, and multi-scale processing occur. It allows the model to handle objects at various scales, provides better feature function for localization, and improves the model's overall accuracy and robustness. The neck's complete job is to enhance the information before it reaches the head, where final predictions are made, ensuring better detection performance.

In later stages, the architecture incorporates upsampling and concatenation steps to fuse features from multiple levels of the backbone. This multi-scale feature fusion

enhances the richness of the representation, making the model more robust in detecting objects across a range of sizes.

Upsampling and Concatenation:

The Neck begins with upsampling operations, which increase the spatial resolution of the feature maps coming from the Backbone's deeper layers, where the resolution is lower due to previous pooling or striding operations. The upsampled feature maps are then concatenated with higher-resolution feature maps from earlier stages of the Backbone. Concatenation merges the semantically rich, deeper features with the spatially rich, shallower features, allowing the network to consider both detailed texture and broader contextual information simultaneously.

- **Feature Pyramid Network(FPN)**

  FPN is a technique designed to address the challenge of detecting objects at multiple scales. In object detection, different objects appear at different sizes, so having multi-scale features is crucial. The backbone of a network might provide features at varying scales, but they may not be combined efficiently.

  FPN creates a pyramid of feature maps, each with different resolutions. This helps capture features for different sizes of objects. The top-level (high-resolution) feature maps capture fine details while the bottom levels (low-resolution) capture contextual and global information. In this way, features from all levels of abstraction are fused resulting in richer and more detailed feature maps for detection.

- **Path Aggregation Network(PANet)**

  PANet is another technique that further enhances the FPN's capabilities by focusing on improving the flow of information between different layers. It aims to address the challenge of localized feature learning, especially when an object is far from the image's center or is partially visible.

PANet improves the model's ability to locate objects accurately, especially in crowded scenes or images with multiple objects at various sizes. It enhances the model's localization power by aggregating features in multiple ways.

4.3.3. Head

YOLOv9's detection module is divided into two specific parts: the classification head and the regression head. The classification head is responsible for assigning class probabilities, while the regression head predicts the coordinates of bounding boxes. Both heads consist of two convolutional layers followed by a 2D convolution layer to output final predictions.

BBox Loss and Cls Loss: These represent the loss functions used for training the network. "BBox Loss" refers to the loss associated with the bounding box predictions, ensuring they match the ground truth as closely as possible. "Cls Loss" refers to the classification loss, which guides the network in accurately predicting the class of each object.

IoU+DFL and BCE: These are specific types of loss functions. IoU (Intersection over Union) + DFL (Distribution Focal Loss) is used for bounding box regression, optimizing the predicted boxes to align with the ground truth. BCE (Binary Cross-Entropy) is used for the classification task, to distinguish between the object classes.

In a running YOLO model, the head takes the feature maps and outputs a set of predictions per grid cell:

- Bounding box coordinates (x, y, width, height)
- Objectness score (the likelihood that a box contains an object)
- Class probabilities (the likelihood of the object belonging to each of the classes the model is trained on)

Each set of predictions is made at different scales corresponding to different layers within the head, allowing the model to detect objects of various sizes. The head is the

final decision-making component of the YOLO architecture, where the actual detections are output based on the learned features.

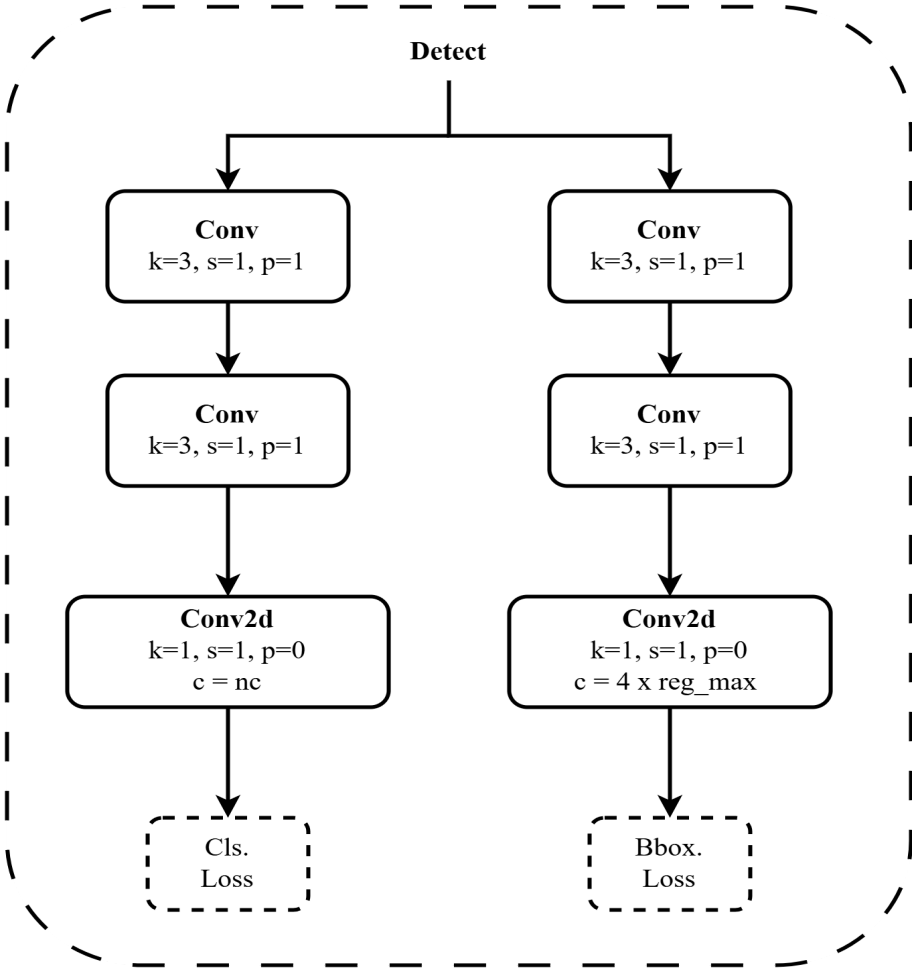


Figure 4-8: Block Diagram of Detect

Object Localization

In computer vision, object localization is all about figuring out the specific area in an image where an object is located. It extends beyond simple image classification by adding a component that predicts the position of the object typically through four values that describe the size and placement of a bounding box. This process is especially important in object detection, where the goal isn't just to recognize what's in the image, but also to figure out exactly where the object is located.

Non-Max Suppression (NMS)

After generating the bounding boxes and their associated class probabilities, the next step is to clean up the results using post-processing methods. One commonly used technique is Non Max Suppression (NMS). This method helps reduce clutter by removing overlapping boxes that point to the same object. Essentially, it retains the most confident prediction while discarding others that overlap too much with it.

Vector Generalization

Vector generalization is a method utilized in the YOLO algorithm to manage the complexity arising from high-dimensional outputs. The YOLO algorithm generates a tensor containing the coordinates of bounding boxes, a score indicating the presence of an object, and the probabilities for each class.. To ease processing, this high-dimensional tensor is transformed into a vector. The vector is then passed through a softmax function, which transforms the class scores into probability values.

In the end, the model outputs a vector for each grid cell that holds the bounding box position, how likely it is that an object is present, and the chances of it belonging to each class.. YOLOv9 represents an updated iteration of the YOLO object detection framework, featuring enhancements that improve its capability to locate and identify objects in images. Two important additions in YOLOv9 are PGI (Positional Gradient Integration) and the GELAN Architecture.

PGI improves how YOLOv9 understands object positions. It uses positional data to locate objects better, focusing on both the object and its surroundings. GELAN is a flexible design that adapts to different tasks and ensures efficiency. It features adaptive layers that adjusts itself for specific tasks, feature retention that keeps critical object details and scalability that works well on various devices, from small to large systems.

Key Highlights of YOLOv9

Information Bottleneck Principle: One of the intrinsic issues of deep learning is the gradual loss of useful information as the data travels through layers of a neural network. YOLOv9 alleviates this issue by using Programmable Gradient Information

(PGI), a method that is designed to maintain useful features throughout multiple layers. Maintaining important data throughout the network enhances gradient quality, which leads to faster convergence and enhanced model performance.

Reversible Functions: The function is referred to as reversible if it can be reversed without any loss of the initial information. YOLOv9 integrates reversible functions into its structure to help retain information, especially in the deeper layers where data is more prone to being lost. This design decision aids in preserving valuable features necessary for accurate object detection, thereby enhancing the overall performance of the model..

4.4. Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a type of deep learning model primarily designed to identify patterns, particularly in visual data. It plays a crucial role in fields like image analysis and object recognition. The core idea behind CNNs is the process of convolution, where a function is applied to extract meaningful features from the input data, typically an image helping the system understand and make predictions based on it. This begins at the convolutional layer, where filters are applied to scan the image and gather feature information.

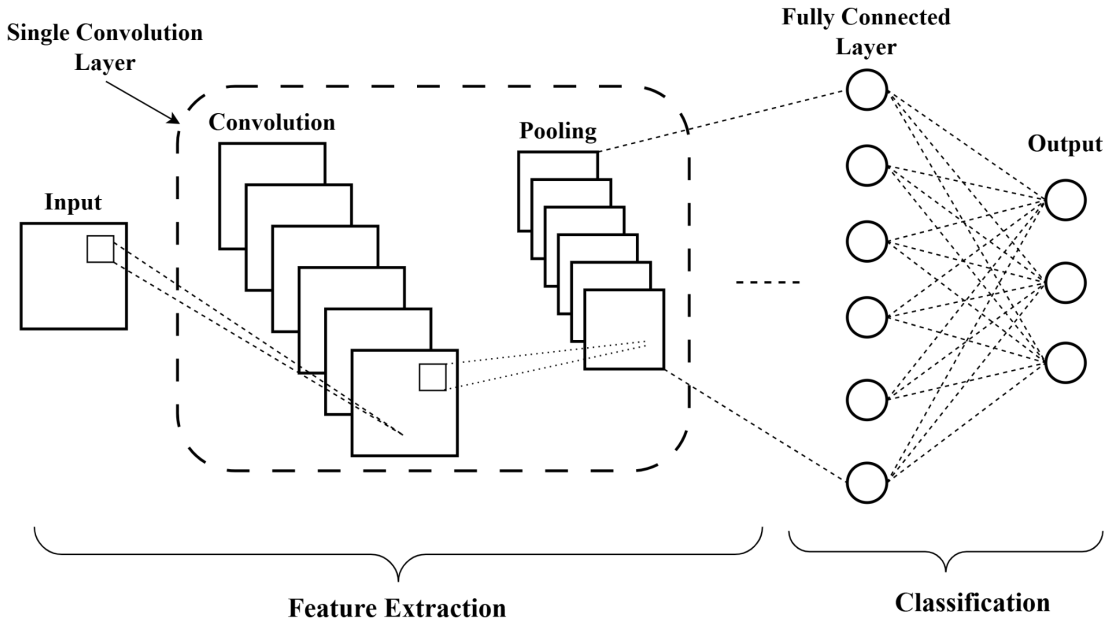


Figure 4-9: Simple CNN Architecture

CNNs are composed of several layers, including an input layer, one or more convolution and pooling layers, and fully connected layers near the end. Convolutional layers apply filters to extract essential patterns while minimizing computational complexity. The final fully connected layers then use the learned features to make accurate predictions. Throughout training, the network updates its filter weights using optimization techniques like backpropagation and gradient descent.

4.4.1. Input Layer

This layer serves as the entry point for data. In the case of images, it receives pixel values as input. The number of neurons here matches the total number of features, which is equal to the number of pixels in the image.

4.4.2. Convolutional Layer

This is the main layer where feature extraction takes place. It uses an input image, a set of filters, and produces feature maps as output. Images are typically processed in three dimensions—height, width, and depth (depth being the color channels, such as RGB). Filters, also called kernels, slide over small sections of the image to detect specific features like edges or textures. This sliding process is known as convolution.

Each filter is a small matrix of weights often 3x3 in size that performs dot product operations on overlapping parts of the image. These values are stored in a new matrix known as the feature map (also called the activation map). The filter moves over the image with a defined step size, or stride, repeating the process until the entire image has been processed. After each convolution, a nonlinear activation function like ReLU is applied to help the network capture complex patterns.

The use of the same filter weights across the image, known as parameter sharing, helps reduce complexity and improve efficiency. While these weights are adjusted during training, certain architectural settings must be defined beforehand. These includes:

- **Number of filters:**

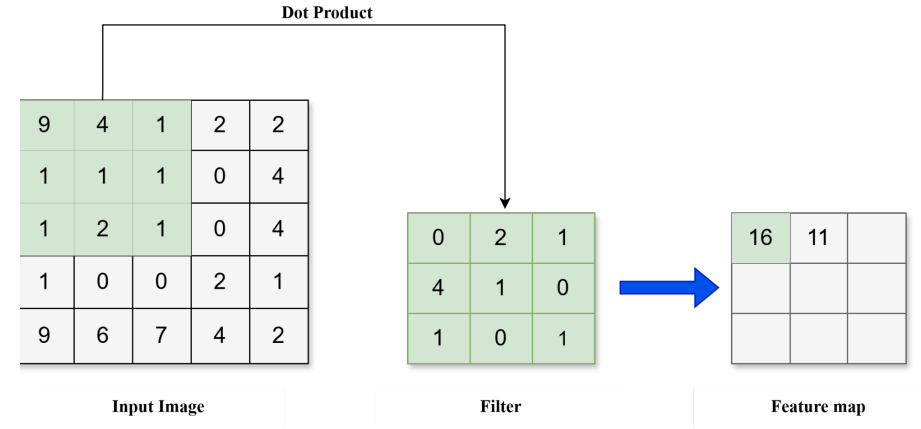
  This impacts the depth of the output. For example, using three different filters will produce three distinct feature maps, resulting in a depth of three for that layer.

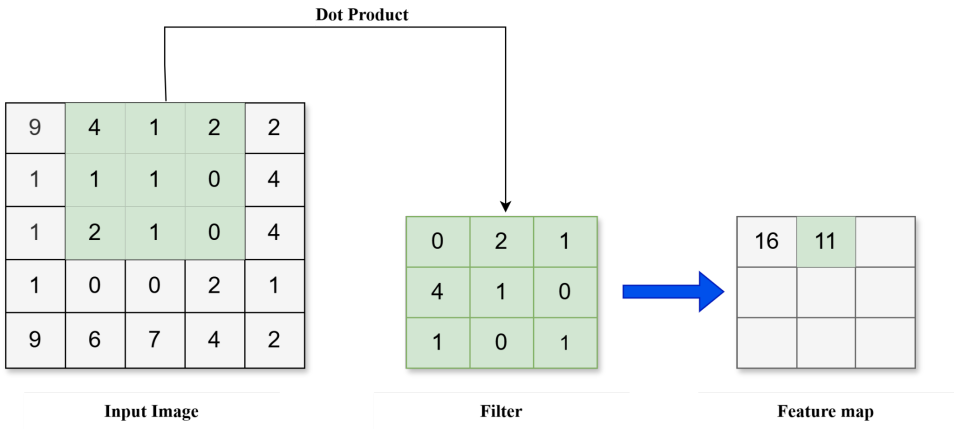
- **Stride**

  It is the distance or the number of pixels that the kernel moves over the input matrix. While the stride values of two or greater is rare, a larger stride yields a smaller output.

- **Zero-padding**

  It is usually used when the filters do not fit the input image. This sets all elements that fall outside the input matrix to zero, producing a larger or equally sized output.





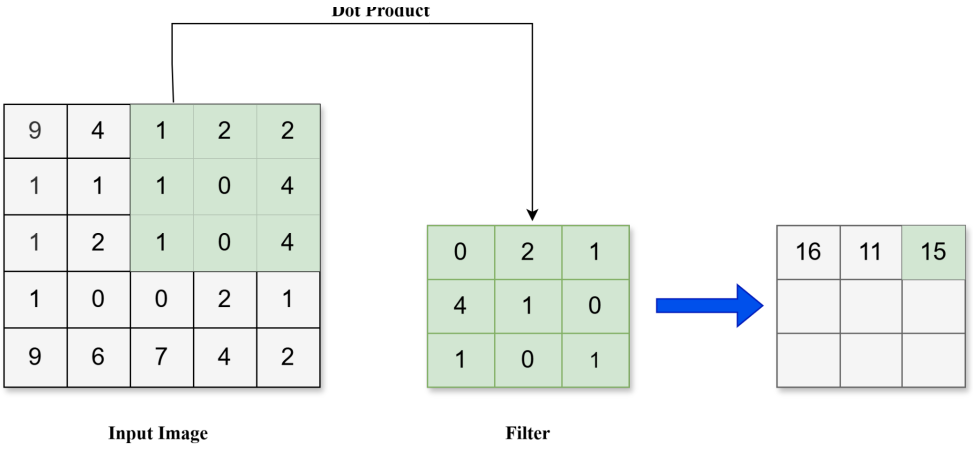


Figure 4-10: Kernel Convolution in CNN

4.4.3. Pooling Layer

Pooling layers, often referred to as downsampling layers, are used to reduce the size of the input data and, in turn, the number of parameters in the model. Like convolution layers, pooling scans across the input using a small window, but the key difference is that it doesn't use any learnable weights. Instead, it applies a simple function such as taking the maximum or the average within each region it covers. Although this process results in some loss of detail, it brings several advantages to a CNN: it lowers computational load, speeds up training, and helps prevent overfitting. There are two main types of pooling:

Max pooling

As the filter slides over the input data, it picks out the highest value from each small region it covers. This means only the most prominent feature like the brightest spot or strongest edge gets passed to the next layer.

Average pooling

Instead of picking the strongest value, this method takes the average of all values within the filter's window. It gives a more balanced summary of the region, keeping general patterns rather than sharp details.

Global Average Pooling (GAP)

Global Average Pooling provides a single average value for a feature map. It essentially converts features maps into a vector of the length of the number of feature maps. It is used in CNNs as a substitute for dense layers. Unlike dense layers it can work with a variable input size. This also greatly reduces the computational overhead as there are no trainable parameters as in fully connected layers.

4.4.4. Dense Layer (Fully connected layer)

In a fully connected layer, every node is linked to every node in the layer before it. This dense connection allows the network to combine all the features learned so far and make a final decision. While earlier layers like convolution and pooling focus on extracting and condensing features using functions like ReLU, the fully connected layer typically uses a softmax function. This activation turns the output into a set of probabilities, helping the model decide which class the input most likely belongs to.

4.5. Our Custom CNN Architecture

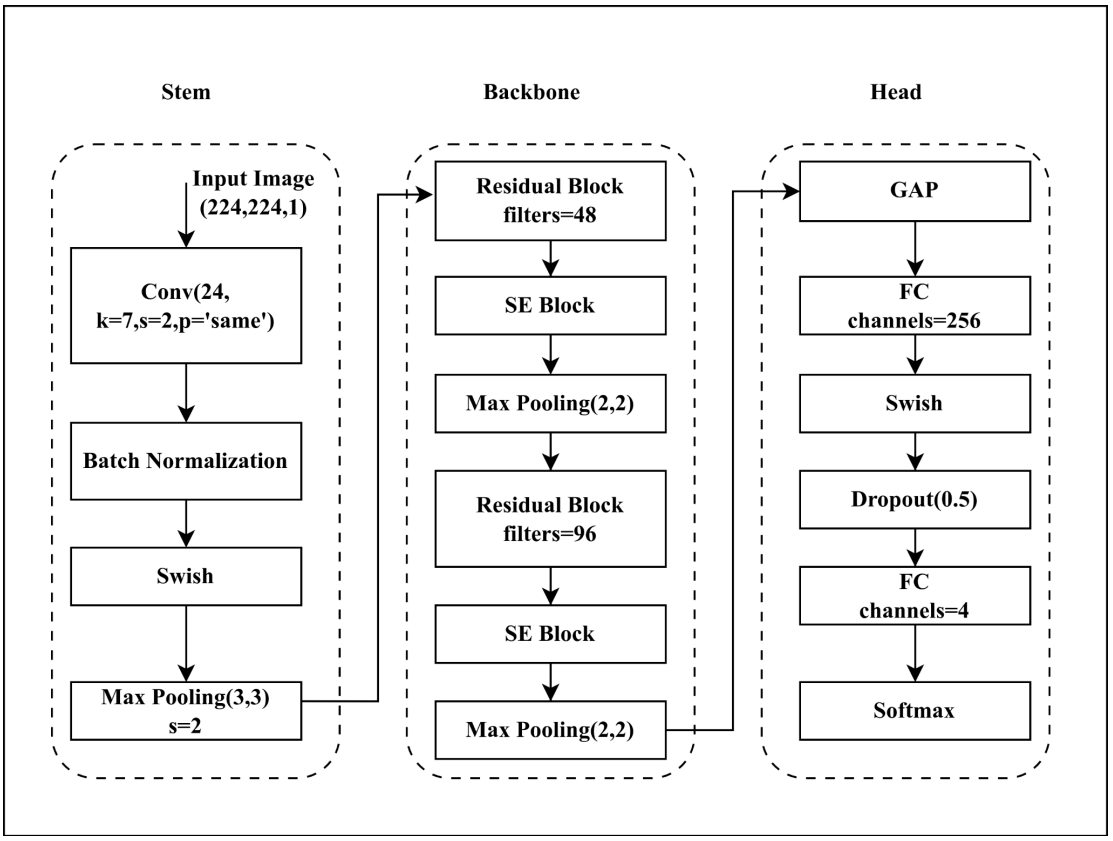


Figure 4-11: Residual Network based Custom CNN Architecture

Stem: The stem is responsible for extracting low level features. It makes use of a single convolutional layer followed by batch normalization function and swish activation. The output is max-pooled to reduce spatial dimension.

Backbone: The backbones makes use of residual and squeeze-and-excitation blocks to learn deeper features. Residual network helps with reducing the vanishing gradient problem and retaining valuable information. Squeeze-and-Excitation block improves features extraction by learning channel-wise attention.

Head: The head processes the extracted features and provides classification output. It makes use of global average pooling for spatial invariance as the features are averaged to a single value. Fully connected layers then learn to mix and emphasize on the most relevant features for classification. Finally the logits are converted to probabilities for classification.

4.5.1. Residual Block

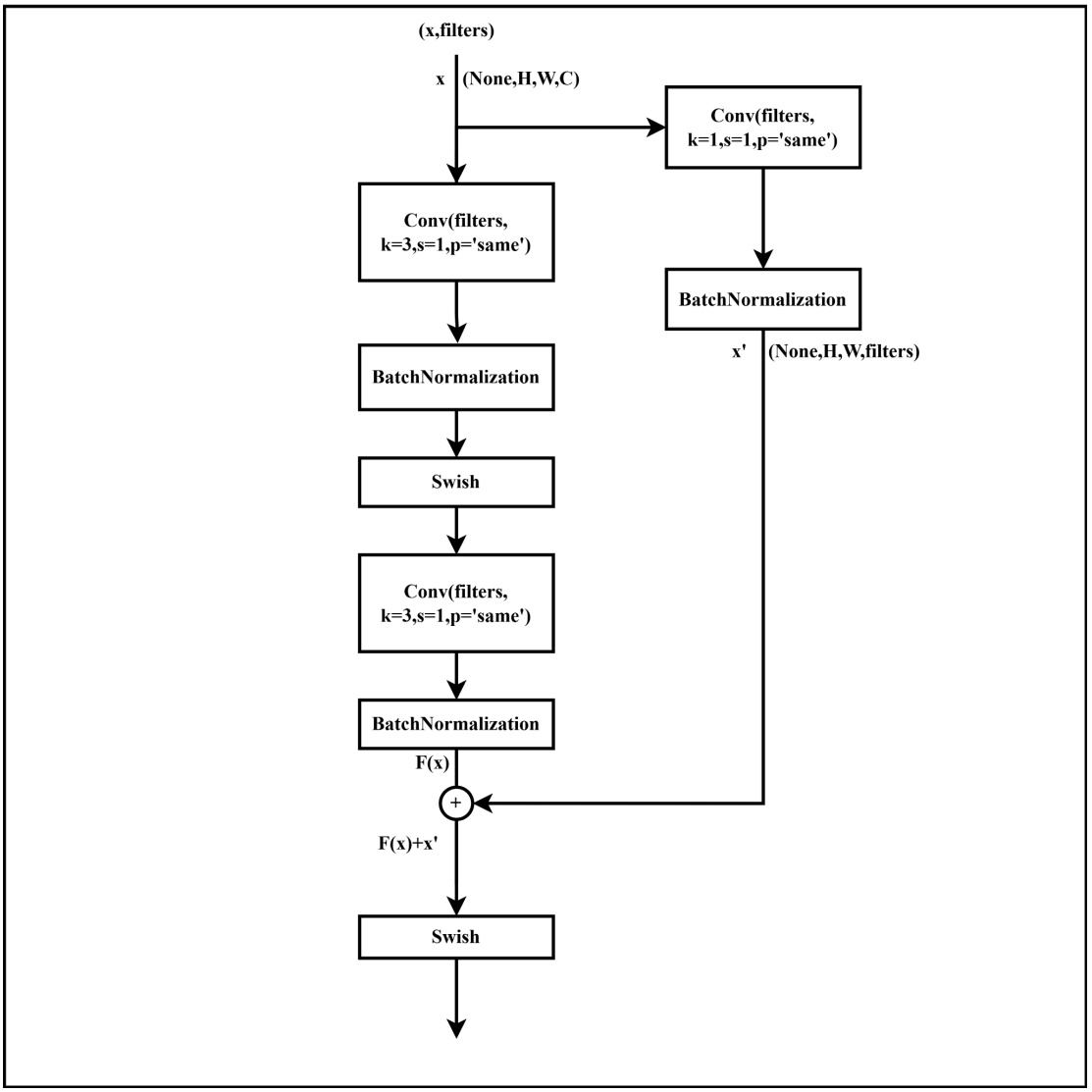


Figure 4-12: Residual Block

Residual blocks are introduced mainly in deeper neural networks to solve the vanishing gradient problem. The key idea behind residual networks is some part of the input is left unoperated for better information retention while passing into further layers.

Here, some of the output is processed and some of it is passed directly to the output. This helps with information retention and better gradient flow. It uses two convolution layers with kernel size 3x3 for feature extraction. A 1x1 kernel is used in the skip (shortcut) connection to adjust the number of channels. The processed output is added

to the output from skip connection, thus adjusting channels is needed to maintain consistency.

4.5.2. Squeeze and Excitation Block

The SE block performs three main operations on the feature maps provided to the block, squeeze, excitation and reshaping. Addition of this block in CNN networks improves the classification performance of the networks without adding any computational overhead.

Squeeze: The squeeze block makes use of global average pooling to reduce the feature maps to a single representative value. If the input feature maps are *H x W x C* they will get transformed into a vector *1 x 1 x C.*

Excitation: The vector obtained after squeezing goes through two subsequent small fully connected layers. One layer reduces the dimension and another expands this. This operation helps highlight the most important features and suppress the ones that are less important.

Reshaping and Scaling: The original feature maps are now multiplied using the obtained vector after the excitation operation. This makes sure that only the most important features are highlighted and less important ones are ignored.

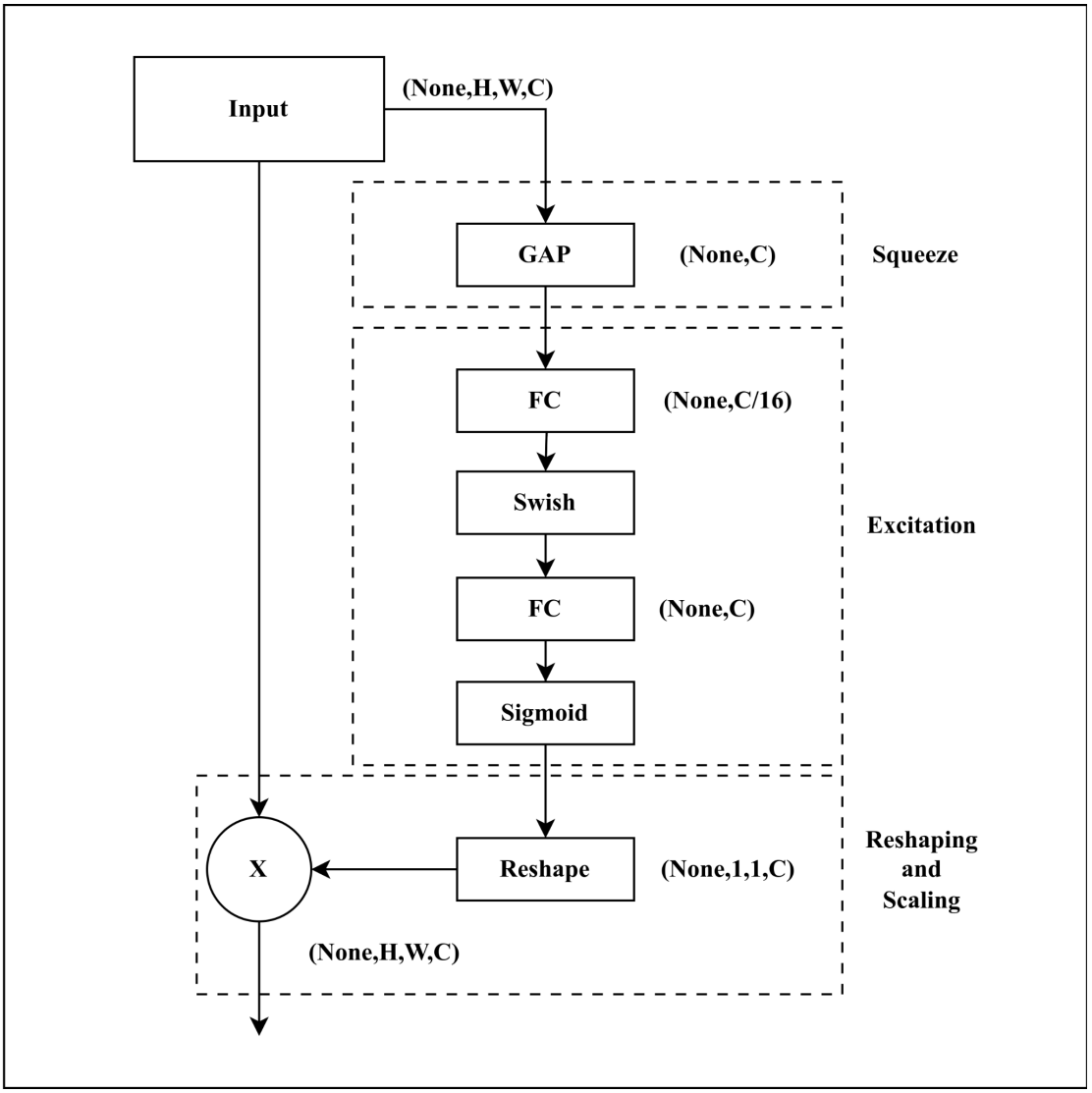


Figure 4-13: Squeeze and Excitation block

4.6. Activation Functions

Sigmoid Activation Function

The sigmoid function transforms its input into a range of (0,1). It maps extremely large or small inputs to its extremities i.e. 1 or 0. It is given by,

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{4-1}$$

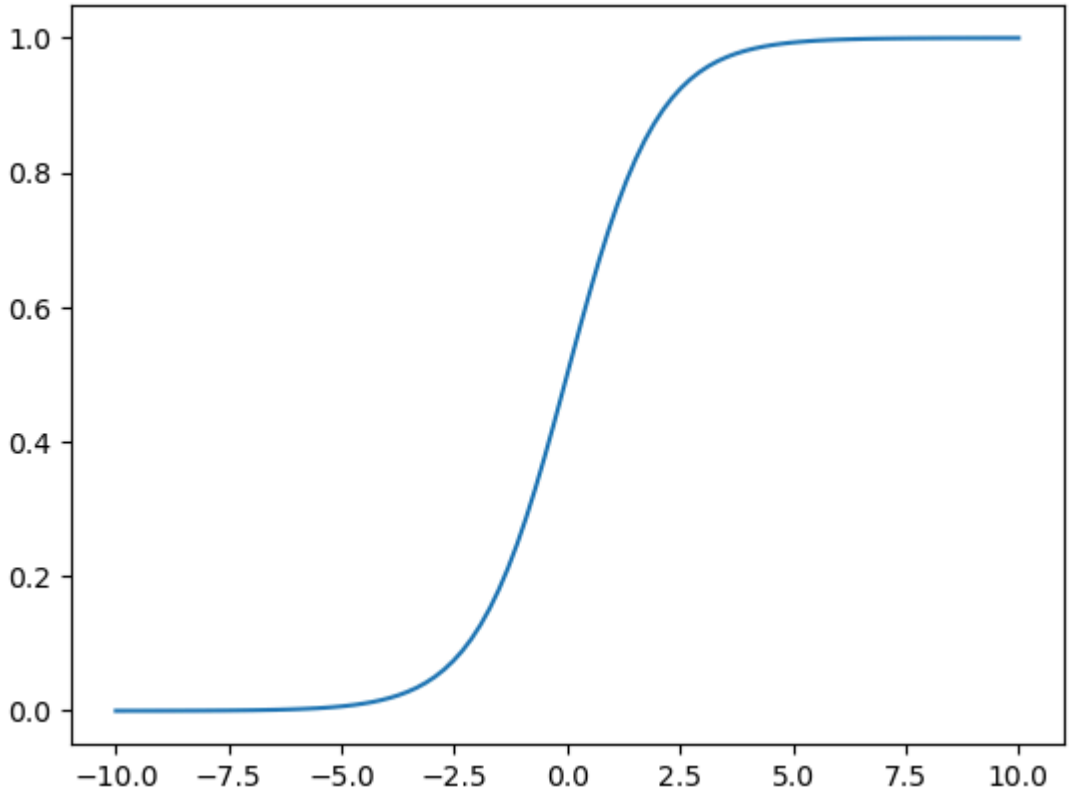


Figure 4-14: Sigmoid Activation Function

And its derivative is given by,

$$\frac{d}{dx}\sigma(x) = \sigma(x)(1 - \sigma(x)) \tag{4-2}$$

We can see that its derivative will be zero if the input is extremely large or small. This problem is called the vanishing gradient problem.

SiLU(Swish-1) Activation Function

The swish function also known as Swish-1 is given as,

$$\text{SiLU}(x) = x.\sigma(x) \tag{4-3}$$

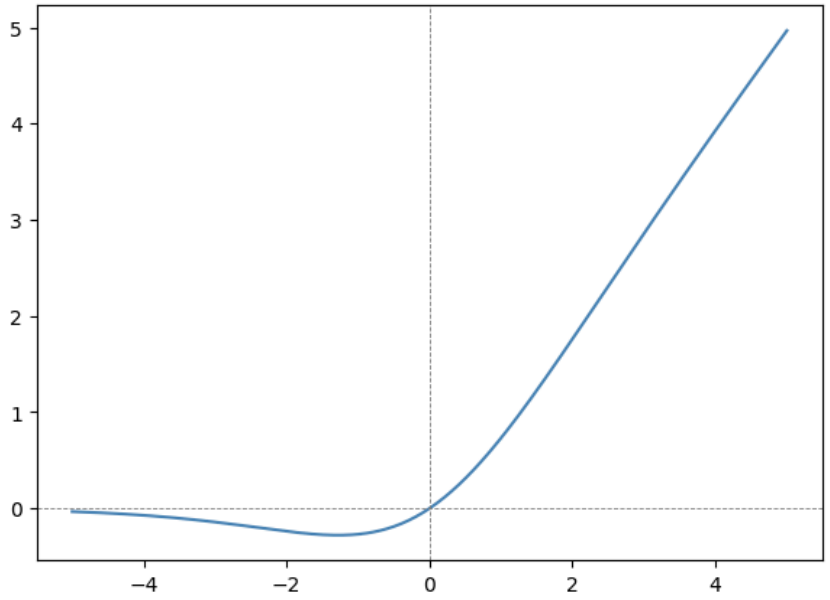


Figure 4-15: SiLU Activation Function

It is better to ReLU when it comes to the neuron dying problem but does not entirely mitigate it. It gives small negative outputs for negative inputs close to zero. It is used in state-of-the-art models like MobileNet and YOLO.

Softmax

The softmax function is the generalization of the sigmoid function to multiple dimensions. For a *n*-dimensional input, it provides a *n*-dimensional output representing a probability distribution. It is given by,

$$\text{Softmax}\left(x_i\right) = \frac{e^{x_i}}{\sum_k e^{x_k}} \tag{4-4}$$

For a randomly generated *10*-dimensional input of [0.0427, 0.1277, 0.0200, 0.0819, 0.0571, 0.0943, 0.1194, 0.0491, 0.3157, 0.0921], the probability distribution obtained by applying softmax on them can be seen below:

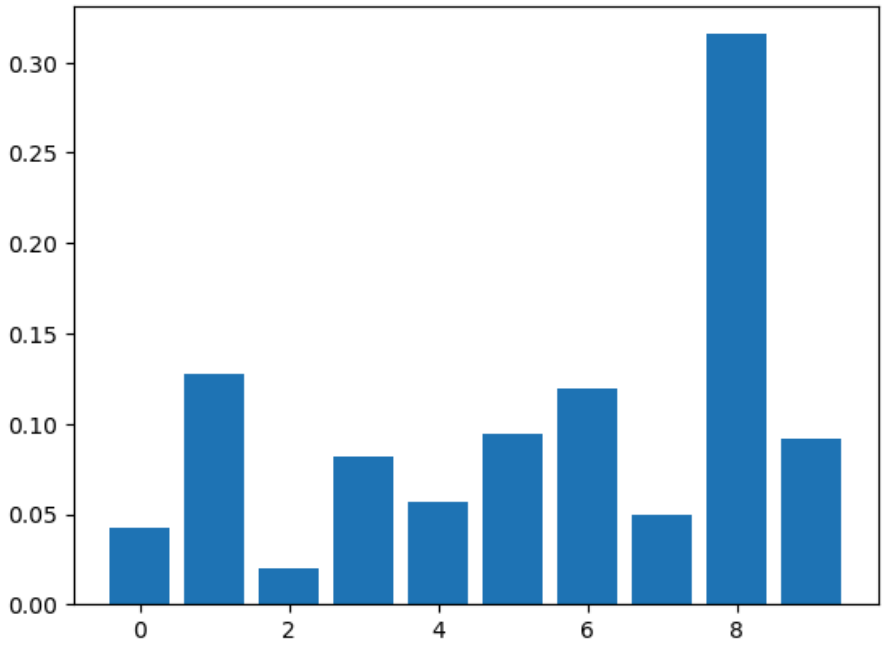


Figure 4-16: Example of Softmax Function

Its derivative is given by,

$$\frac{d}{dx_j}\text{Softmax}(x_i) = \begin{cases} \text{Softmax}(x_i) - \text{Softmax}(x_j), & if\ i = j \\ -\text{Softmax}(x_j)\text{Softmax}(x_i), & if\ i \neq j \end{cases} \tag{4-5}$$

4.7. Loss Function

Cross Entropy Loss

The cross-entropy loss measures the difference between a predicted probability distribution and true distribution. It is given by,

$$L_{CE}(y, \hat{y}) = -\frac{1}{N}\sum_{n=1}^{N} y_n log\hat{y}_n \tag{4-6}$$

Its derivative is given by,

$$\frac{\delta}{\delta\hat{y}_n}L_{CE}(y, \hat{y}) = -\frac{1}{N}\frac{y_n}{\hat{y}_n} \tag{4-7}$$

4.8. Losses in YOLO

YOLO models provide various losses at each epoch while training and validating. The three main losses to evaluate performance of the model are:

Box Loss

This represents the error in the bounding box predictions. It measures how well the model predicts the location and size of the bounding boxes relative to the ground truth. A common approach is to use a regression loss like Mean Squared Error (MSE) or Intersection over Union (IoU)-based losses such as CIoU or DIoU loss.

$$L_{CIoU} = 1 - IoU + \frac{\rho^2(center_{pred}, center_{true})}{c^2} + \alpha v \quad (4\text{-}8)$$

where,
IoU is the Intersection over Union
ρ(·)is Euclidean distance between predicted and true boxes
c is diagonal length of the smallest box covering both the predicted and true boxes
α is a trade-off parameter, and v is a measure of consistency in aspect ratio

Value Range: The box loss can range from 0 to any positive number, where 0 represents perfect predictions. However, due to normalization and averaging, typical values might be seen between 0 and 1 or 0 and a few tens, depending on the size of the dataset and the variance in object sizes.

Lower box loss values represent better alignment of the predicted boxes with the ground truth.

Classification Loss (cls loss)

This measures the error in predicting object classes. It often uses Binary Cross-Entropy Loss for multi-class classification problems.

$$L_{cls} = -\sum_{i=1}^{c} y_i \log(p_i) + (1 - y_i)\log(1 - p_i) \quad (4\text{-}9)$$

where,

y_i is the binary indicator (0 or 1) if class label i is the correct classification

p_i is the predicted probability of the sample being of class i

C is the total number of classes

Value Range: Classification loss ranges from 0 to a positive number, with 0 indicating perfect classification. In practice, especially when softmax activation is used, the values typically range between 0 and the number of classes.

Lower classification loss values mean the predicted class probabilities are closer to the one-hot encoded ground truth labels.

Distribution Focal Loss (dfl loss):

This is used to address the imbalance between easy and hard examples in the training dataset, focusing more on incorrectly predicted instances. It's an adaptation of the Focal Loss, which adds a factor to the standard Cross-Entropy Loss that down-weights the loss assigned to well-classified examples

$$L_{dfl} = -\sum_{i=1}^{c}(1 - p_i^{\gamma})\log(p_i) \quad (4\text{-}10)$$

where,

p_i is the predicted probability for the class i

γ is the focusing parameter, > 0, which adjusts the rate of down-weighting

Value Range: Similar to classification loss, DFL can range from 0 upwards, but typically it will start higher and should decrease as the model learns to focus on the hard examples.

Lower DFL values indicate that the model is becoming more adept at handling hard-to-classify examples, improving overall robustness.

4.9. Optimizer

Adam(Add a momentum)

The Adam optimizer uses the first and second moments of the gradient to adapt the learning rate for each weight of the neural network.

The exponentially weighted moving average of the gradients and squared gradients is given as,

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\frac{\delta L}{\delta w_t}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\frac{\delta L}{\delta w_t})^2$$

Initializing $v_0 = s_0 = 0$ causes bias towards smaller values. This can be addressed by normalizing the terms. The normalized state variables are,

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

$$\hat{v}_t = \frac{m_t}{1-\beta_2^t}$$

Then, the update can be performed as,

$$w_t = w_{t-1} - \eta \frac{1}{\sqrt{\hat{v}_t}+\epsilon}\hat{m}_t \quad (4\text{-}11)$$

4.10. Evaluation

After training and integration of our model we will evaluate our model on different metrics such mAP, F1-score, recall to ensure our system performance.

Object Detection: Performance Evaluation Metrics

Intersection over Union (IoU):

$$IoU = \frac{area\ of\ intersection}{area\ of\ union} \quad (4\text{-}12)$$

IoU measures the overlap between two bounding boxes:

1. Ground Truth Box: The actual location of an object.
2. Predicted Box: The box predicted by the mode

The Intersection over Union (IoU) is a standard measurement for evaluating how accurate object detection models are at detecting objects. It measures the extent of overlap between the predicted bounding box (P) and the relevant ground truth box (G). IoU calculation is done by dividing the area of intersection of P and G by the total area covered by both boxes. This value ranges from 0 to 1, where 0 represents that there is no overlap and 1 represents perfect alignment. Higher IoU means more accurate object detection.

Average Precision (AP) is a performance metric commonly used in object detection tasks, including models like YOLO, to evaluate how well the model detects objects across different confidence thresholds. It captures how well the model balances accuracy and completeness in its predictions by turning the entire precision-recall curve into one easy-to-understand score.

Precision: It indicates how many of the predicted bounding boxes are correct.

$$\text{Precision} = \frac{True\ Positive(TP)}{True\ Positive(TP) + False\ Positive(\ FP)} \quad (4\text{-}13)$$

Recall: It measures how many of the actual objects were correctly detected.

$$\text{Recall} = \frac{True\ Positive(TP)}{True\ Positive(TP) + False\ Negative(\ FN)} \quad (4\text{-}14)$$

F1 Score: F1-Score is the harmonic mean of Precision and Recall, which a balance between them.

$$\text{F1 Score} = \frac{2\ x\ Precision\ x\ Recall}{Precision + Recall} \quad (4\text{-}15)$$

Precision-Recall Curve: The precision-recall curve plots precision on the y-axis and recall on the x-axis for different confidence thresholds.Lowering the confidence threshold usually increases recall but decreases precision.

Average Precision: AP is the area under the precision-recall curve.

$$\text{Average Precision} = \int_{r=0}^{1} p(r)dr \quad (4\text{-}16)$$

4.11. UML Diagrams

4.11.1. Use Case Diagram

A use case diagram provides a visual representation of the interaction of the actors with the system to accomplish a particular goal. It is a high level overview of the system functionality and the ways in which a user can interact with the system. It shows what a user/system can do without including the implementation details.

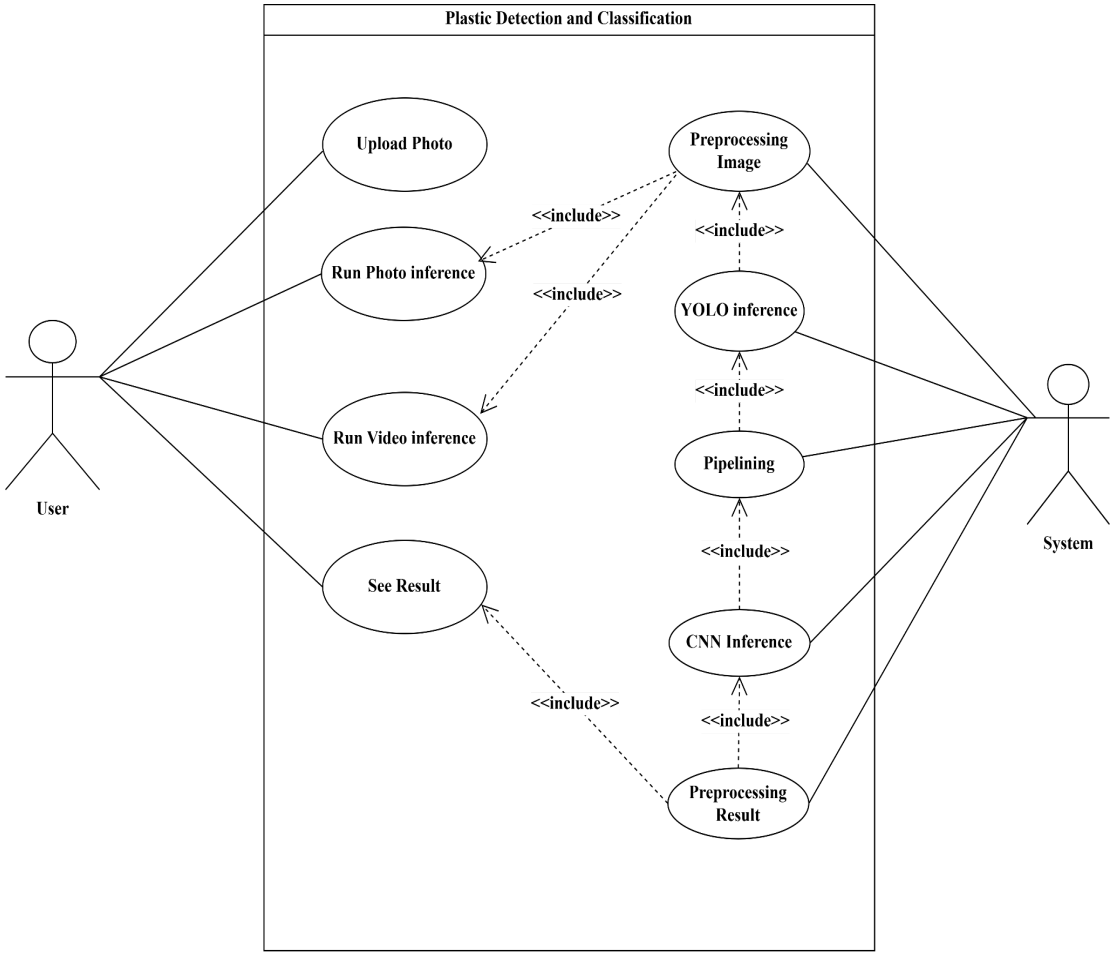


Figure 4-17: Use Case Diagram

User Entity

A user entity uses the Plastic Detection and Classification system. The user is able to upload a photo of plastic images and run photo inference or can run live video inference. The user needs to trigger the Upload Photo button or run video inference button and the user gets the result from the system.

System Entity

A system entity is the vital entity in the Plastic Detection and Classification system. which is actually responsible for detecting and classifying plastic types. It is triggered by the user, the image uploaded or webcam feed is preprocessed and the result is generated by the system. The output is converted to appropriate format before giving it to the user as result.

4.11.2. Sequence Diagram

A sequence diagram is a diagram used to illustrate the flow of messages between various component objects of a system in a sequential order of their interaction.

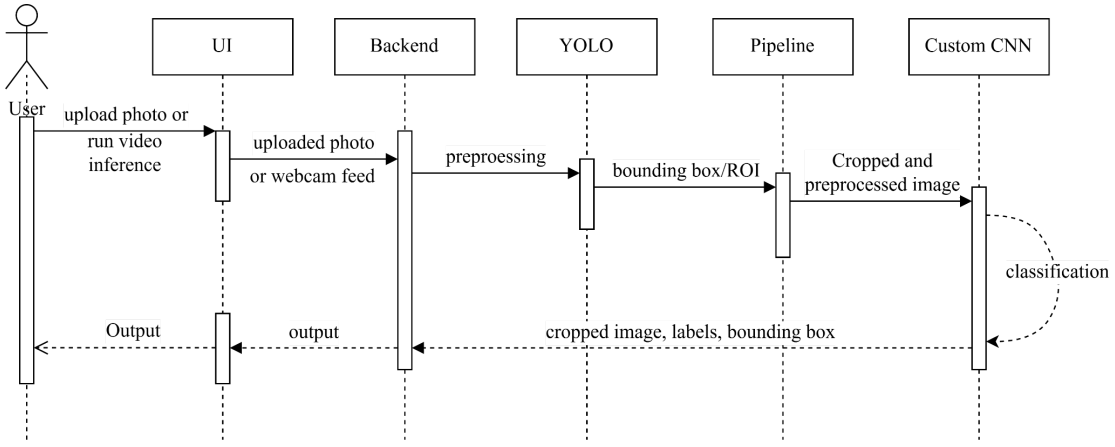


Figure 4-18: Sequence diagram

The figure above shows the sequence diagram for the Plastic Detection and Classification web app. A user uploads an image of plastic or press Start video inference button using the web UI. The image is forwarded to the backend where preprocessing tasks are done before passing it to YOLO. afterward YOLO extracts bounding box/ROI from the input and passes to the pipeline where again necessary preprocessing steps like cropping, converting to grayscale, resizing and normalizing are done before passing to CNN model. Our Custom CNN model classifies the plastic image into the category where it belongs and returns back cropped image, classified labels and bounding box to the backend. The final output is post-processed at backend and then forwarded to the UI from where the user can observe the output.

5. IMPLEMENTATION DETAILS

5.1. Dataset Collection

The dataset used for plastic detection and classification is primarily sourced from WaDaBa(Waste Database), Kaggle open source datasets and some from Roboflow.

5.1.1. WaDaBa Dataset

The WaDaBa datasets, developed by researchers J. Bobulski and J. Piatkowski, serves as the foundation for a dataset of plastic waste detection and classification which includes high-quality images for machine learning tasks. The original size of the images is 1920*1277 pixels with resolution of 300 dpi. The colour palette is RGB 24 bits and the file format is JPG. The name of the photographed object is encoded as follows:

0004 a01b05c2d0e1f0g1h1.jpg

Where,

[a] - a number denoting the plastic type;

[b] - colour;

[c] - type of light;

[d] - the deformation level;

[e] - the level of a dirt;

[f] - presence of a screw cap or a lid;

[g] - presence of a ring - characteristic for the bottles with a screw caps;

[h] - order number of a random position of a photographed object.

The values for each parameter and their description are provided in Table 5-1.

The dataset contains images of plastics from four different classes namely HDPE, PET, PP, PS. There are a total of 3,960 images; 2,200 images of PET, 600 images of HDPE class, 640 images of PP class, and 520 images of PS class of plastic.

Figure5-1: WaDaBa Sample Images

5.1.2. Kaggle Dataset

From Kaggle we got approximately 3,200 images of PET and HDPE plastic bottles, along with various bottle caps, captured in a real-world setting on a conveyor belt. The images were taken in an industrial environment, where the bottles and caps are moving along the belt.

5.1.3. Dataset from Roboflow Universe

Additionally, images of plastics of the same classes with varying backgrounds were collected from the internet and added to the training dataset to improve recognition accuracy. This dataset consisted of about 4,018 images annotated into four different classes (Aluminium can, HDPE plastic, PET plastic, and UHT box. Out of these we removed annotations (Aluminium can and UHT box) from the dataset and annotated them as Null class declaring that there are no relevant images in them. And then we merged this dataset into our project database.

Figure5-2: Annotated Dataset from Roboflow universe

5.1.4. Final Prepared Dataset for YOLOv9

After merging all the above dataset including the manually annotated WaDaBa dataset, we got our final dataset of 19,578 images. Other than the above three dataset we added around 9,077 random images of different objects like dog, cat, glasses, different scenery, cars, humans etc. so that our model can generalize that not every object it sees must be an object. The average image quality was at an average 0.41 mp. The final dataset had 29,424 annotations in class plastic.

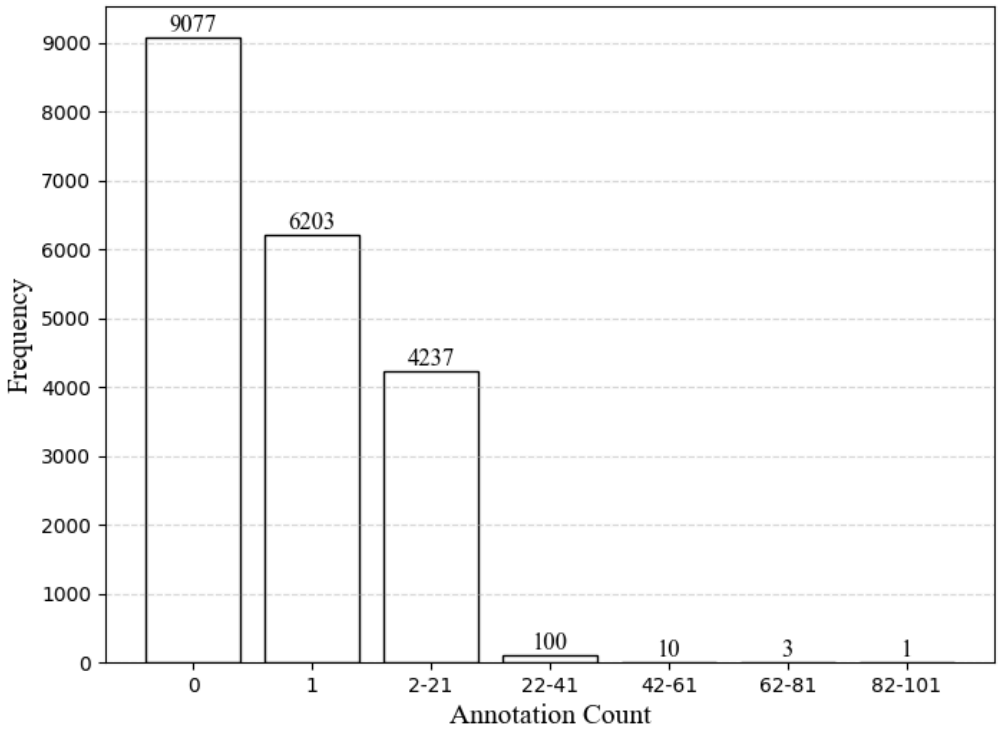


Figure 5-3: Annotation Distribution Across Images

5.1.5. Final dataset prepared for custom CNN

For our CNN model, we combined approximately 7,600 images from Kaggle and Wadaba, including PET (3,618 images), HDPE (2,925 images), PP (560 images), and PS (520 images) and then augmented these to around 11,000 images in each class to improve class balance and model robustness.After that we trained the model but better results weren't being obtained .So we again searched the roboflow and found around 10,000 and 12,000 images for PET and HDPE respectively .For PP and PS, we

scraped a few hundred images from Google using Selenium WebDriver. After that, we augmented the dataset to reach 28,000 images for each.Then,we cropped the images to focus solely on the plastic, excluding the entire background, to ensure the model learns features that are most relevant for effective plastic classification.We splitted our dataset into three categories: train, valid and test with percentage share of 70%, 20% and 10%.

5.2. Image Annotation

The images needed to be annotated before feeding the YOLO model for training. We used Roboflow for extensive annotation of 3,960 raw images from the WaDaBa dataset. Annotating involves creation of bounding boxes to surround the object in the image as shown in Figure 5-4. Then the object class is provided which in our cases is plastic. We are then able to obtain a yaml file that contains information about the position of the object within the image in textual form.

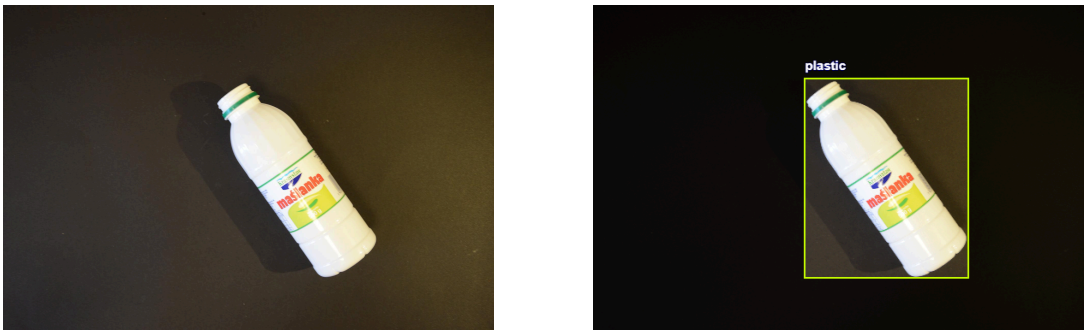


Figure 5-4: Annotation of raw images

The yaml file thus obtained had following data:

class_id	x_center	y_center	width	height
0	0.5984375	0.5390625	0.3421875	0.61875

Where,

- class_id: The object belongs to class 0 in the dataset which is named as plastic.

- x_center: The normalized value of distance of the center of the bounding box on x-axis.
- y_center: The normalized value of distance of the center of the bounding box on y-axis.
- width: Normalized width of the bounding box.
- height: Normalized height of the bounding box.

Since the above pixels are normalized, they can be converted into actual pixel values by using the following formulas.

$$x\_center\_pixel = x\_center * image\_width$$

$$y\_center\_pixel = y\_center * image\_height$$

$$width\_pixel = width * image\ \_width$$

$$height\_pixel = height * image\_height$$

5.3. Image Augmentation

The dataset we have is not uniform i.e. there are unequal data for each class. We performed augmentation on the dataset to further improve the diversity and equality of the dataset. We used Roboflow to apply augmentations on these images including vertical and horizontal flip, saturation changes between -25% and +25% , brightness changes between -15% and 15%, changes in the exposure in the range of +14% and -14% and random noise was introduced in up to 1.5%2 of the pixels.

For the CNN model,we applied data augmentation using ImageDataGenerator.Since we're converting the dataset to grayscale, we focused on geometric and intensity-based augmentations, avoiding color-based transformations. We applied rotation (±45°), width and height shifts (20%), and shear transformations to make the model more resilient to different orientations and perspectives. Additionally, zooming (0.8 - 1.2) and horizontal and vertical flipping were used to account for variations in object size and positioning. To handle varying lighting conditions, we incorporated

brightness adjustments (0.7 - 1.3). To ensure transformations don't cause artifacts, we used fill mode = 'nearest', which fills missing pixels by copying the nearest valid pixel. These augmentations help reduce overfitting, improve generalization, and enable the model to classify plastic waste more accurately in different environments.

5.4. Dataset Splitting

We splitted our dataset into three categories: train, valid and test with percentage share of 70%, 20% and 10% during annotation and merging, which made around 21,200 train, 6,076 valid and 2,921 test images for YOLOv9 belonging to single class plastic. Similarly, the dataset for the CNN was split into train, validation, and test sets in a ratio of 7:2:1. In the train, validation, and test sets, the number of samples were 80,396, 22,972, and 11,488, respectively each set containing images categorized into four classes (HDPE, PET, PP and PS).

Figure 5-5: Dataset splitting

5.5. YOLOv9 Training

For YOLOv9 training we had to choose one among the variants. So we all decided to train all the available variants and compare their metrics to see which model aligns perfectly with our requirement. The available variants were YOLOv9t, YOLOv9s, YOLOv9m, YOLOv9c, and YOLOv9e which had the following configurations and performance metrics when trained on MS COCO dataset

Table 5-1: YOLOv9 variants configuration and performance metrics

Model	Test Size	APval	AP50val	AP75val	Param	FLOPs
YOLOv9t	640	38.3%	53.1%	41.3%		7.7G

YOLOv9s	640	46.8%	63.4%	50.7%	7.1M	26.4G
YOLOv9m	640	51.4%	68.1%	56.1%	20.0M	76.3G
YOlOv9c	640	53.0%	70.2%	57.8%	25.3M	102.1G
YOLOv9e	640	55.6%	72.8%	60.6%	57.3M	189.0G

Out of these five variants we trained four variants i.e YOLOv9t, YOLOv9s, YOLOv9m and YOLOv9c on the final dataset that we have prepared for YOLO. We trained all these models using the ultralytics framework and the training Hyperparameters were as follows:

Table 5-2: Hyperparameters for YOLOv9 model training

S.N	**Hyperparameters**	**Values**
1	lr0	0.002
2	imgsz	640
3	Models	YOLOv9t.pt, YOLOv9s.pt, YOLOv9m.pt, YOLOv9c.pt
4	optimizer	AdamW (auto)
5	momentum	0.9
6	batch size	16
7	Weight decay	0.0005

5.6. CNN Model Training

In our plastic classification project, we designed a Convolutional Neural Network (CNN) using TensorFlow to categorize images into four classes: PET, HDPE, PP, and PS. Overall,we have 112,000 images and train the model in batches of 16 images.

Our model architecture integrates residual blocks to improve gradient flow and Squeeze-and-Excitation (SE) blocks to enhance feature importance. We begin with a 7×7 convolutional stem with 24 filters, followed by Swish activation and batch normalization for stable training. The backbone consists of stacked residual and SE blocks with increasing filters (48 → 96) and MaxPooling layers to extract hierarchical features.

For classification, we apply global average pooling, followed by a dense layer with Swish activation and L2 regularization (0.001) to prevent overfitting. We also introduce dropout (0.5) for improved generalization. The final softmax layer predicts the probability distribution over the four plastic categories.

To optimize training, we use the Adam optimizer with a dynamic learning rate schedule, along with categorical cross-entropy loss and label smoothing (0.1) to improve model calibration. This enhanced architecture aims to achieve higher precision and recall, ensuring accurate classification of plastic waste.

Table 5-3: Hyperparameters for Latest CNN model

S.N	Hyperparameters	Description	Value
1	num_classes	Number of output classes	4 (HDPE, PET, PP, PS)
2	input_shape	Shape of input images (height, width, channels)	(224,224,1)
3	batch_size	Number of samples processed before updating model weights	16

4	epochs	Number of times the model sees the entire dataset during training	100
5	Batch Normalization	Normalizes activations for stable training	Applied after each Conv Layer`
6	Dropout Rate	Drops neurons randomly to prevent overfitting	0.5
7	Optimizer	Algorithm to update model weights during training	Adam
8	Learning Rate	Step size for updating weights	Initial 0.001 and decrease by the factor of 0.9 every 10,000 steps
9	Loss Function	Measures difference between predicted and actual labels	Categorical Crossentropy
10	Label smoothing	Softens labels to reduce model overconfidence	0.1
11	L2 Regularization	Penalizes large weights to prevent overfitting and encourage generalization	0.001

5.7. Frontend development

For frontend development we have used Html, TailwindCss and Javascript . Html is used as the building block, tailwind for styling and javascript for handling form validation and drag and drop file uploads. We created three different pages index.html, inference. html, and video_inference. html. The web pages have forms for image uploads and results of inference and for showing live video. Index page is the main interface for users to upload photos for inference and start video inference. It contains a form to upload photos and a button to start photo inference. Inference page displays the result of combined inference of YOLO and CNN models. The result of

this inference consists of cropped images of detected objects along with their classification labels. Video inference page displays live video feed from webcam superimposed with bounding box and classification labels along with the combined inference.

5.8. Backend Development

The backend is created using flask which is a lightweight WSGI web application framework for python. This backend is capable of handling HTTP requests, processing of images and video frames, and performing object detection and classification using YOLOv9s and our custom CNN models. The main features of backend includes file upload handling, combined inference on uploaded images and real time inference. The backend ensures that the application can handle user requests efficiently, process images and video frames accurately, and provide real-time feedback to the users.

6. RESULTS AND ANALYSIS

6.1. Selecting One Variant of YOLOv9

After the preparation of the final dataset for yolo we trained four variants of versions of YOLOv9 i.e. YOLOv9t, YOLOv9s, YOLOv9m and YOLOv9c for 10 epochs. significant improvements were seen on the precision, recall, map@50, map@50-95 and loss metrics which we can observe in the following graphs. The training results are as follows:

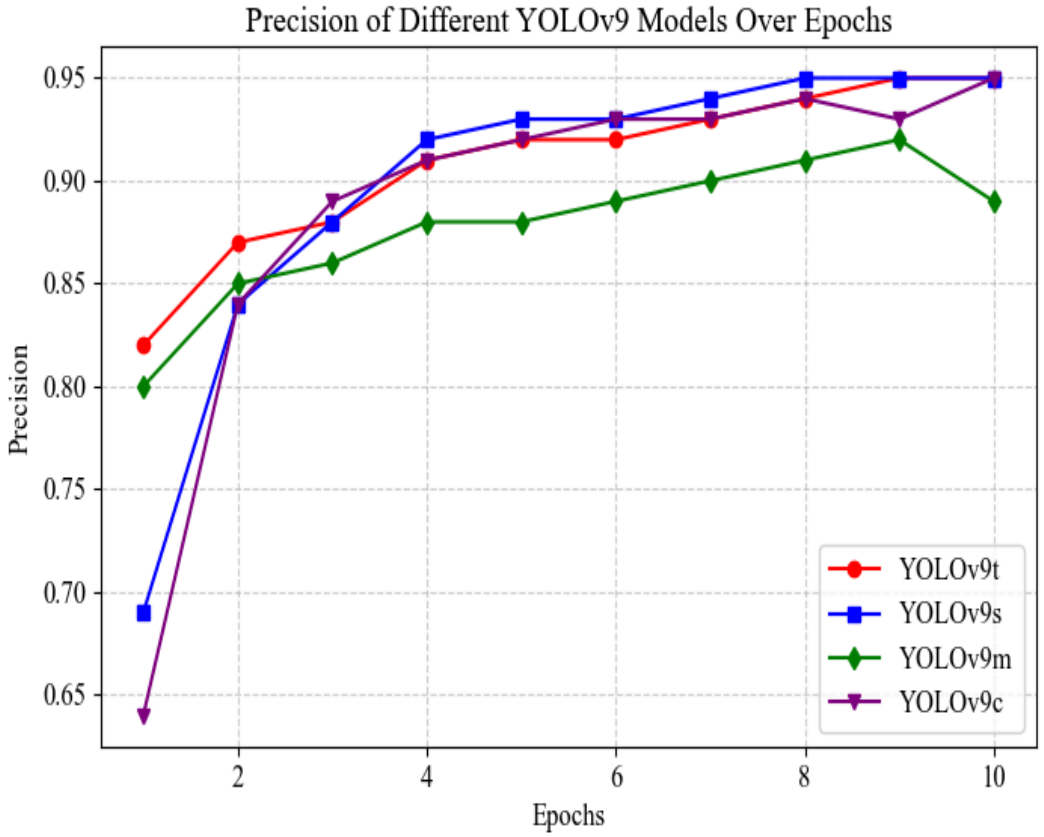


Figure 6-1: Precision vs Epoch Comparison of Different YOLOv9 models

Precision vs epoch: The precision graph indicates the proportion of true positives over the sum of true positives and false positives. The precision rises quickly and levels off close to 1, suggesting a high accuracy in the model's positive predictions across the batches. All four YOLOv9 models show an improvement in recall over the epochs, indicating that they are learning and becoming more capable of identifying all relevant objects **YOLOv9t**, **YOLOv9s**, and **YOLOv9c** achieve the highest precision

values (0.95) by the tenth epoch, making them the best-performing models among the four.

Recall vs epoch: The recall graph shows the proportion of true positives over the sum of true positives and false negatives. The rapid ascent to a plateau near 1 indicates that the model is capable of correctly identifying most of the positive examples. **YOLOv9t**, **YOLOv9s**, and **YOLOv9c** achieve the highest recall values (0.94-0.95) by the tenth epoch, making them the best-performing models among the four. YOLOv9s and YOLOv9c show significant improvements in the early epochs, suggesting that they have a faster learning rate compared to YOLOv9t and YOLOv9m.

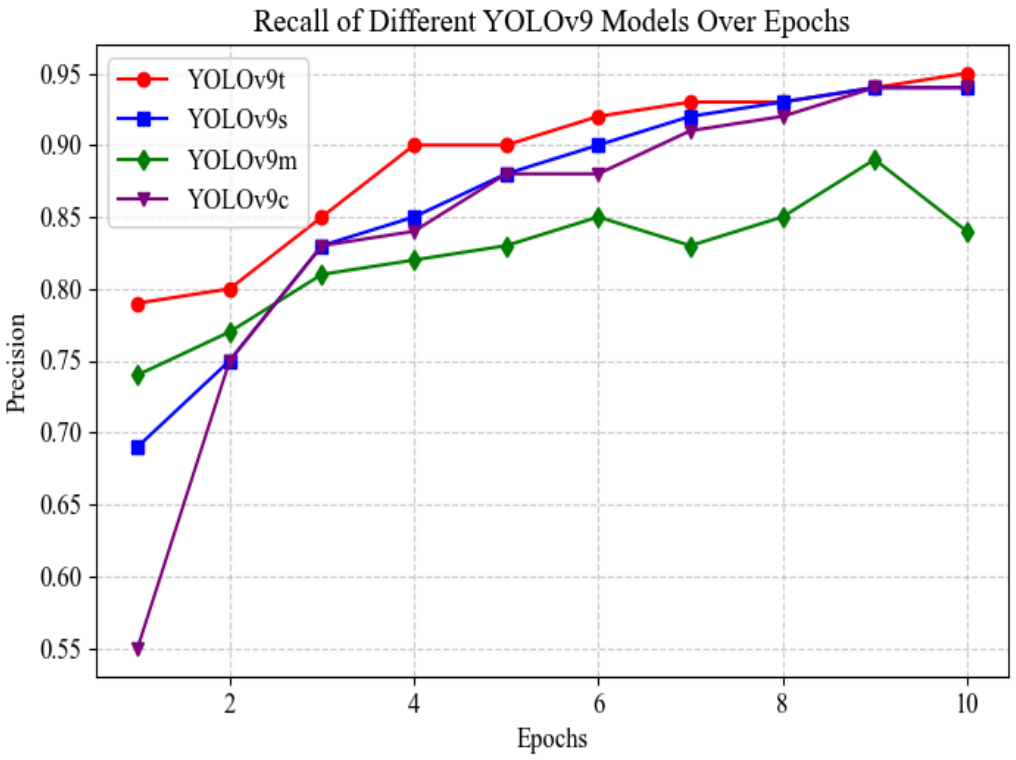


Figure 6-2: Recall vs Epoch Comparison of Different YOLOv9 models

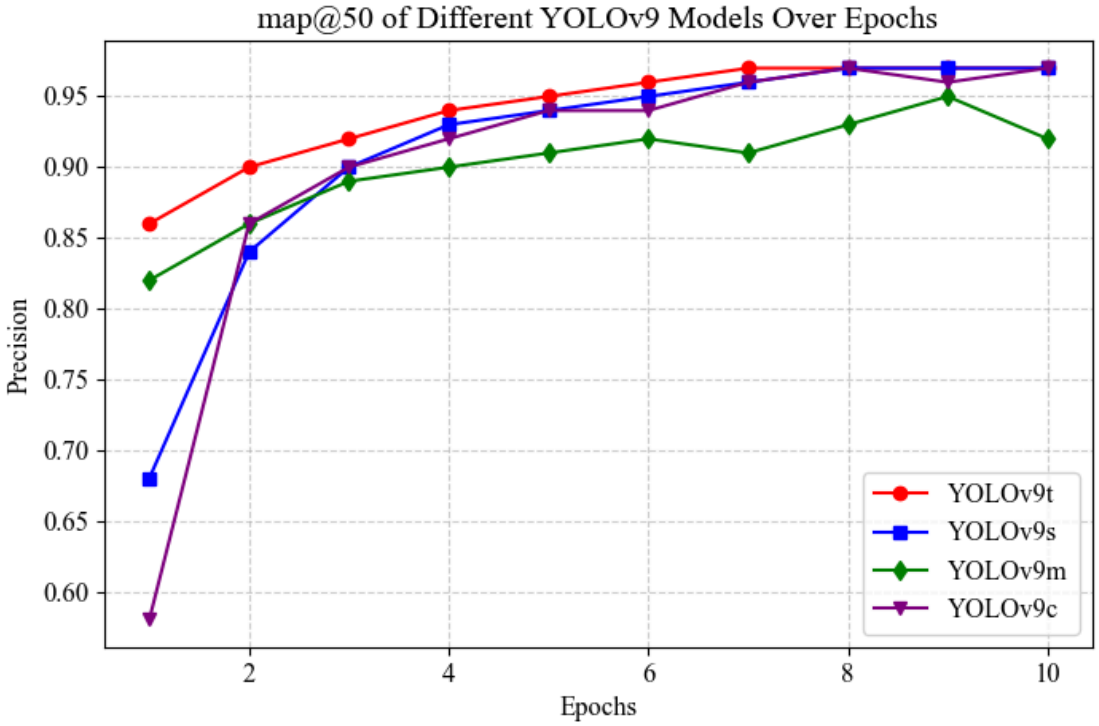


Figure 6-3: map@50 of comparison of YOLOv9 models

map@50 vs epoch: This graph shows the mean average precision averaged at an intersection over union (IoU) threshold of 0.5. mAP@50 is a metric that measures the accuracy of a model's object detection and used to evaluate the model's performance. All four YOLOv9 models show an improvement in map@50 over the epochs, indicating that they are learning and becoming more accurate in detecting objects. **YOLOv9t**, **YOLOv9s**, and **YOLOv9c** achieve the highest precision values (0.97) by the tenth epoch, making them the best-performing models among the four. YOLOv9s and YOLOv9c show significant improvements in the early epochs, suggesting that they have a faster learning rate compared to YOLOv9t and YOLOv9m.

map@50-95 vs epoch: This graph shows the mean Average Precision averaged over multiple IoU thresholds from 0.5 to 0.95. All four YOLOv9 models show an improvement in map@50-95 over the epochs, indicating that they are learning and becoming more accurate in detecting objects. **YOLOv9s** achieves the highest map@50-95 value (0.89) by the tenth epoch, making it the best-performing model among the four. YOLOv9s and YOLOv9c show significant improvements in the early

epochs, suggesting that they have a faster learning rate compared to YOLOv9t and YOLOv9m.

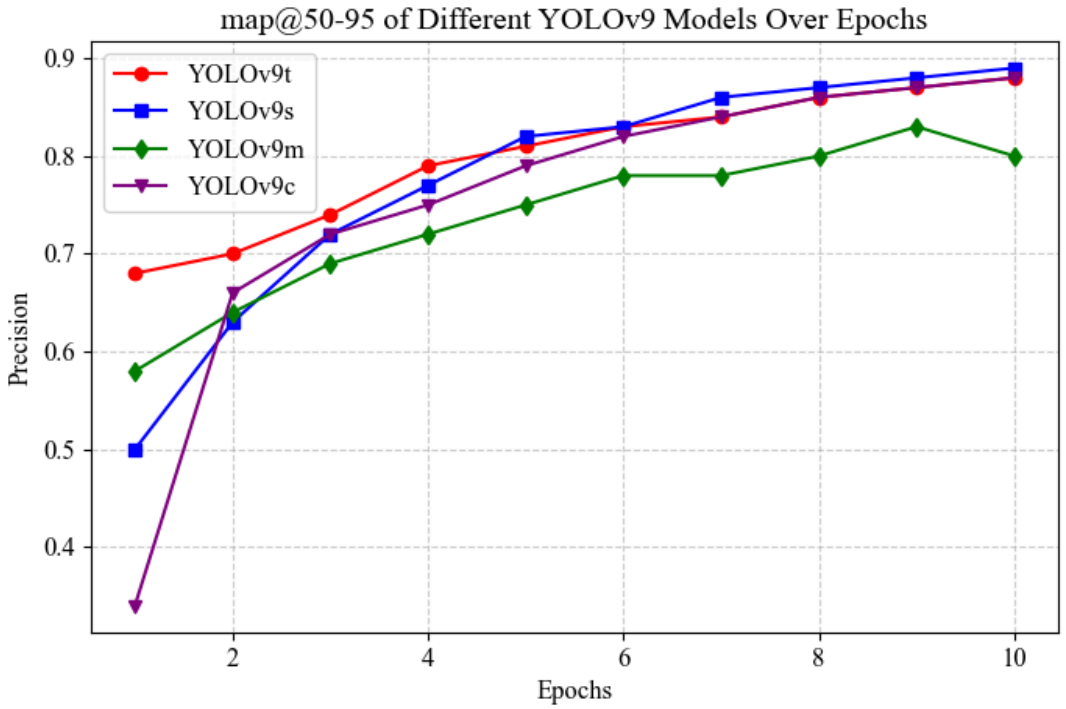


Figure 6-4: map@50-95 comparison of YOLOv9 models

Selection: After comparing the four models on precision, recall, map@50, map@50-80 and observing validation losses, trained on the same dataset we came to a conclusion that YOLOv9s was best suited for our project. So we selected **YOLOv9s** as our object detection model as it has in all performance metrics and has a great blend of high accuracy and lower inference time and was well suited for real time object detection.

6.2. Training Result of YOLOv9s

After the selection of YOLOv9s we trained it for 60 epochs with early stopping with patience level set to 7 in order to prevent overfitting on train dataset. It took us around 11 hours for our model to complete the training process. The results obtained after training are as follows:

6.2.1. Train/Val Loss

Train/Val Box Loss: These graphs show the loss associated with bounding box predictions. Both training and validation losses decrease sharply in the initial epochs, indicating rapid learning. The trend stabilizes as the model converges, reflected by the flattening of the curve. This suggests that the model becomes proficient in predicting the location and size of the bounding boxes over time.

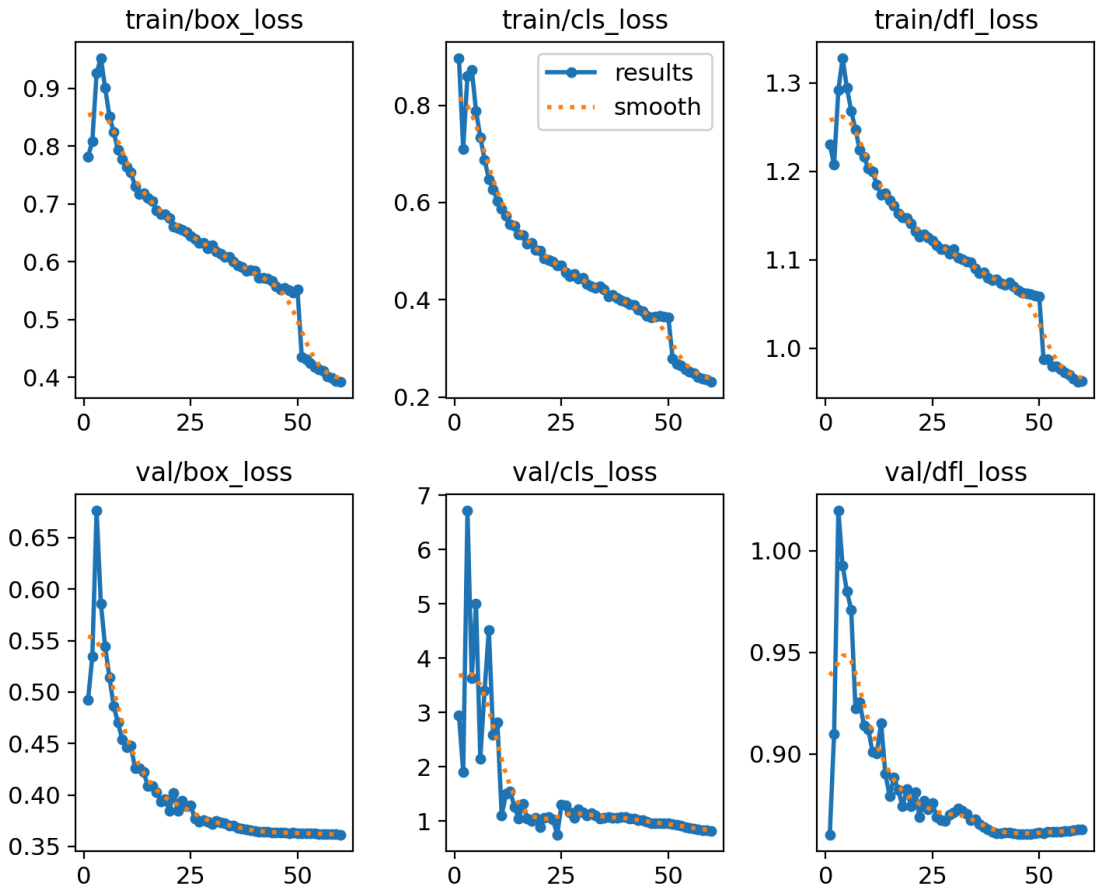


Figure 6-5: Losses of YOLOv9s model

Train/Val Classification Loss: These represent the error in the predicted class probabilities during the training and validation. A sharp decline followed by a stable plateau is seen, which implies the model is effectively learning to classify the different classes it's been trained on.

Train/Val Distribution Focal Loss (DFL): These graphs display the loss used to address class imbalance, focusing training on hard-to-classify examples. The trend of

decrease and then stabilization in both the training and validation phases suggests that the model is improving in handling difficult cases over the course of training.

6.2.2. Precision, Recall and mAP Per epoch

Precision: Indicates how accurately the model identifies plastic without false positives. The increasing trend suggests improved detection accuracy.

Recall: Measures the model's ability to detect all relevant plastic instances. The steady rise shows fewer missed detections over time.

mAP50: Represents the mean average precision at a 0.5 IoU threshold, reflecting the model's ability to make correct detections. The high values indicate strong detection performance.

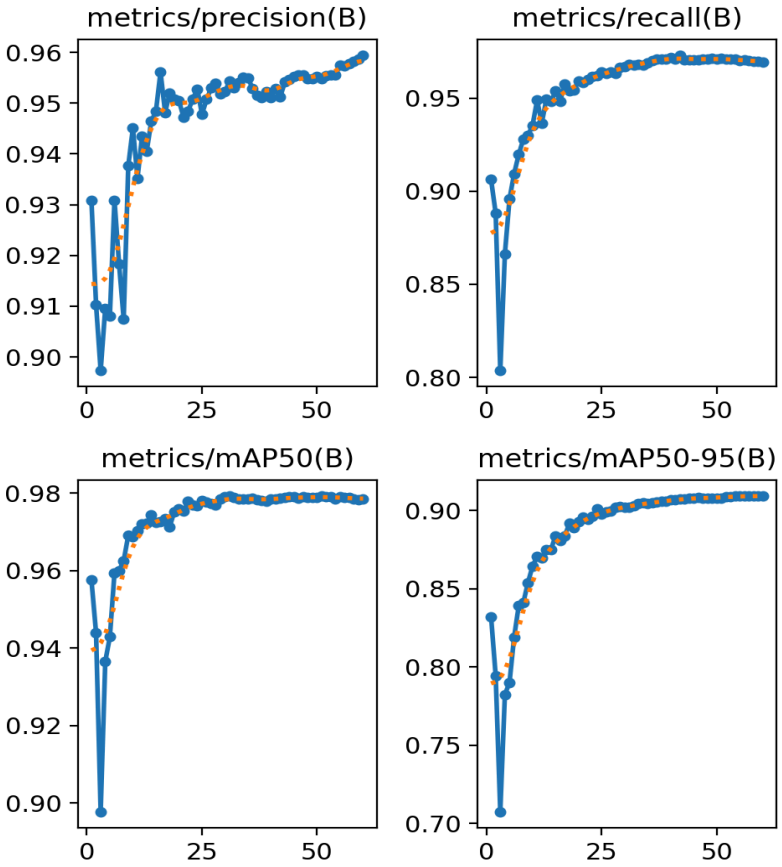


Figure 6-6: Graphs of performance metrics of YOLOv9s

mAP50-95: Evaluates precision across multiple IoU thresholds (0.5 to 0.95), providing a comprehensive view of performance. The upward trend confirms consistent improvements.

The graphs show steady improvements across all metrics, indicating that the model is learning effectively. Precision and recall curves stabilize at high values, while mAP metrics confirm strong detection capabilities. Overall, the training is progressing well, and the model is performing as expected.

6.2.3. Precision Confidence

A Precision-Confidence graph shows how precision changes as the confidence threshold varies in a classification model. It helps evaluate the trade-off between making fewer but more accurate predictions versus taking more risks with lower confidence.

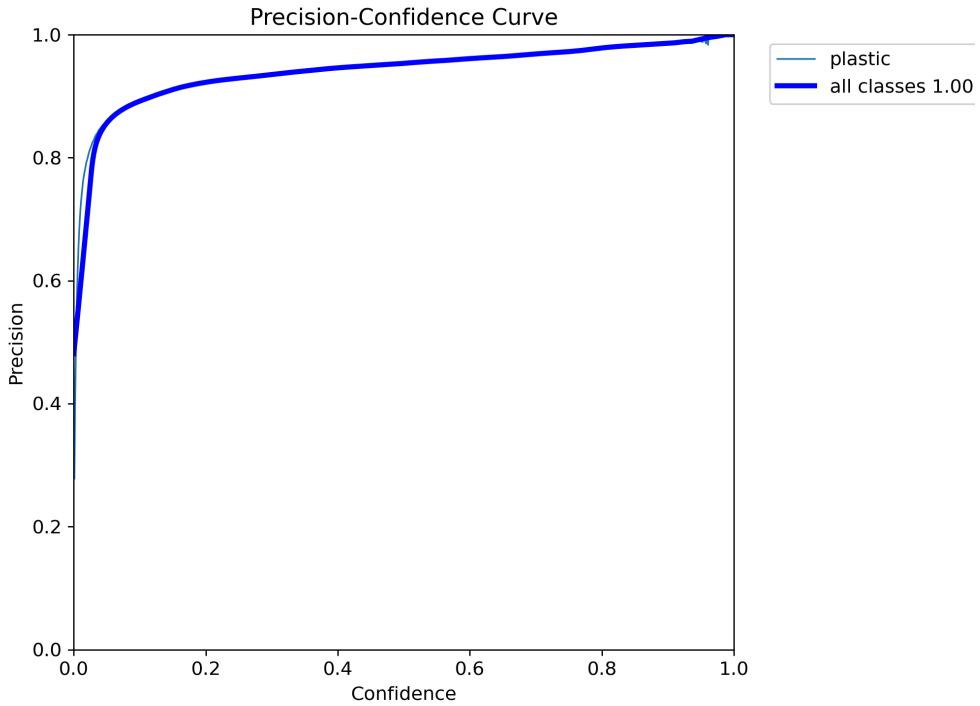


Figure 6-7: Precision Confidence curve of YOLOv9s

The curve starts low but quickly rises and stabilizes near 1.0, indicating that as confidence increases, the model maintains high precision. This suggests strong

reliability in detections at higher confidence thresholds, with minimal false positives across different confidence levels.

6.2.4. Recall Confidence

A Recall-Confidence graph shows how recall changes as the confidence threshold varies in a classification model. It helps assess the trade-off between capturing more true positives versus setting a higher confidence threshold

The curve stays close to 1.0 for most confidence values but drops sharply after 0.8, indicating that a very high confidence threshold may lead to missed detections, ultimately reducing recall.

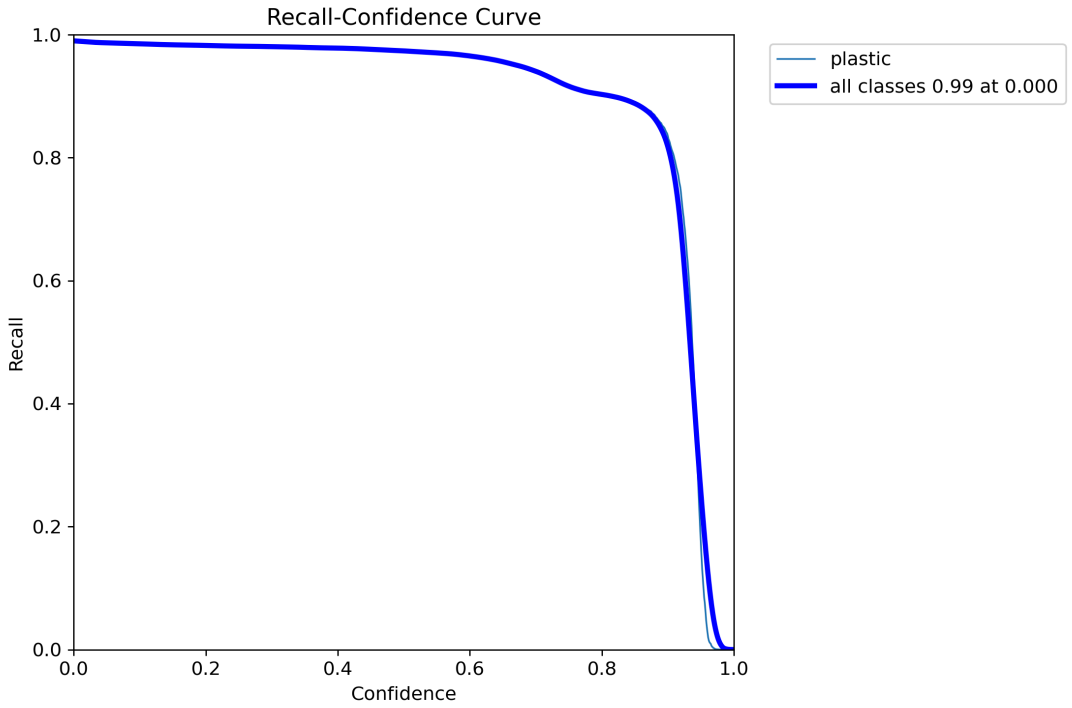


Figure 6-8: Recall Confidence curve of YOLOv9s

6.2.5. Precision Recall

The Precision-Recall (PR) curve is a way to measure how well a classification model makes accurate predictions. It shows the balance between precision (how many of the predicted positives are actually correct) and recall (how many actual positives the model successfully identified) at different confidence levels. A well-performing model has a curve that stays close to the top right, meaning it maintains high accuracy while catching most positive cases. In our case, the curve is nearly perfect, with a

mean Average Precision (mAP@0.5) of 0.979, indicating that the model does an excellent job of detecting plastic objects.

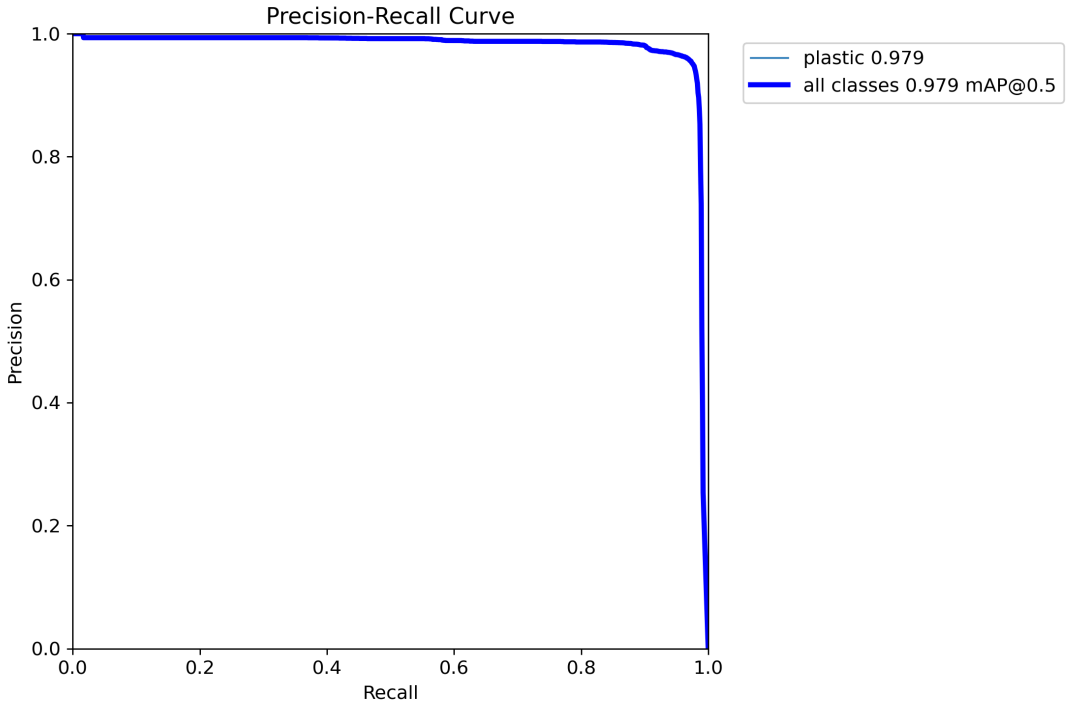


Figure 6-9: Precision Recall Graph

6.2.6. F1 Confidence Score

The F1-Confidence Curve shows how the F1-score, which balances precision and recall, varies across different confidence thresholds in a classification model. It helps identify the confidence level where the model performs best by maintaining a strong trade-off between precision and recall. In this case, the F1-score remains high over a broad range, peaking at **0.96 at a confidence level of 0.546**, indicating that the model is highly effective. The curve starts lower at very low confidence values, rises quickly, stays stable, and then drops sharply at very high confidence thresholds. This analysis is useful for setting optimal confidence levels.

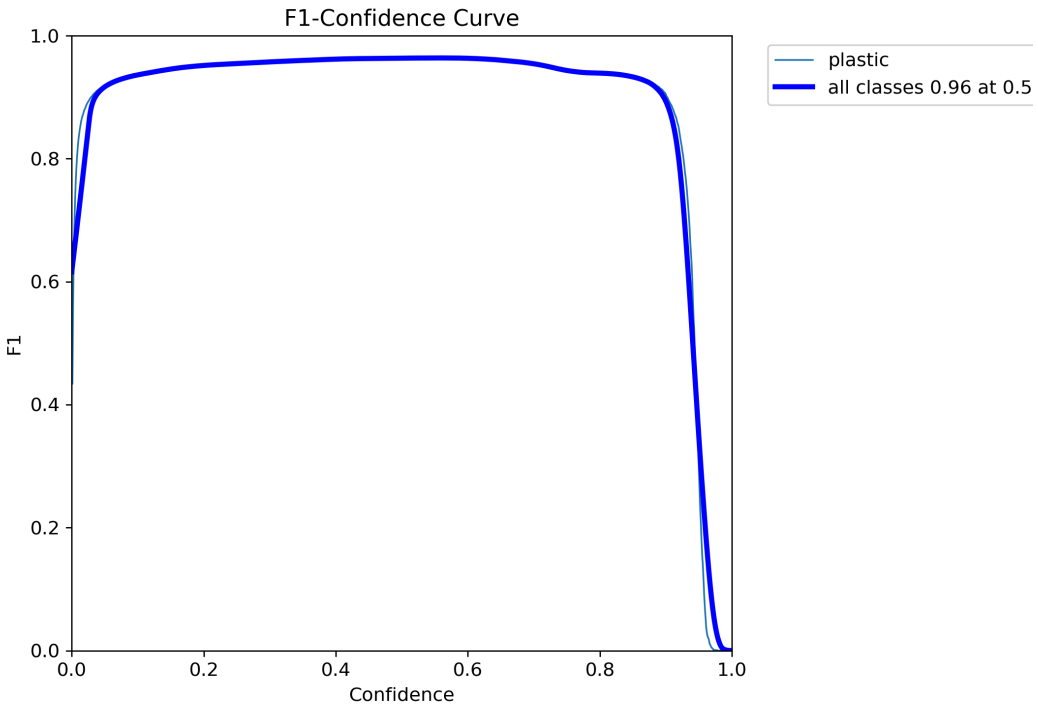


Figure 6-10: F1-confidence curve of YOLOv9s

6.2.7. Confusion Matrix

A confusion matrix is a table used to assess the performance of a classification model by comparing predicted labels to actual labels. It helps in identifying patterns of correct classifications and misclassifications, providing insights into where the model performs well and where it struggles. The confusion matrix depicts that the trained yolo model 8257 TP (True positives), 716 FP (False Positives) and 435 FN (FalseNegatives).

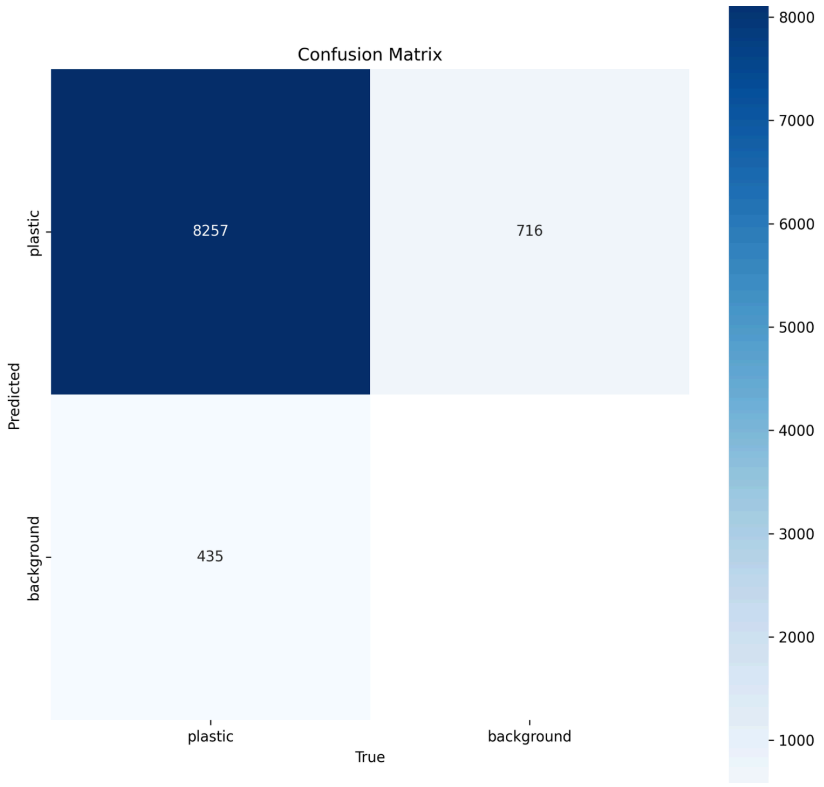


Figure 6-11: Confusion matrix of YOLOv9s

6.2.8. YOLOv9s Evaluation Metrics

Table 6-1: Evaluation metrics of YOLOv9s

Metrics	**Values**
Precision	0.95774
Recall	0.97038
f1-score	0.96402
map@50	0.97892
map@50-95	0.90924

6.3. CNN Training result

6.3.1. Train/Val Loss:

These graphs represent the error in the predicted class probabilities during training and validation. The red dashed curve, representing training loss, shows a smooth, consistent decline over the epochs, indicating that the model is effectively learning to classify plastics as it adjusts its internal parameters. In contrast, the blue solid curve for validation loss starts with some fluctuations, likely due to variability in the validation data or noise, but gradually decreases and levels off after about 40–50 epochs. This convergence between the training and validation loss suggests that the model is generalizing well, meaning it is not just memorizing the training data but is also capable of performing accurately on new, unseen plastic classification examples.

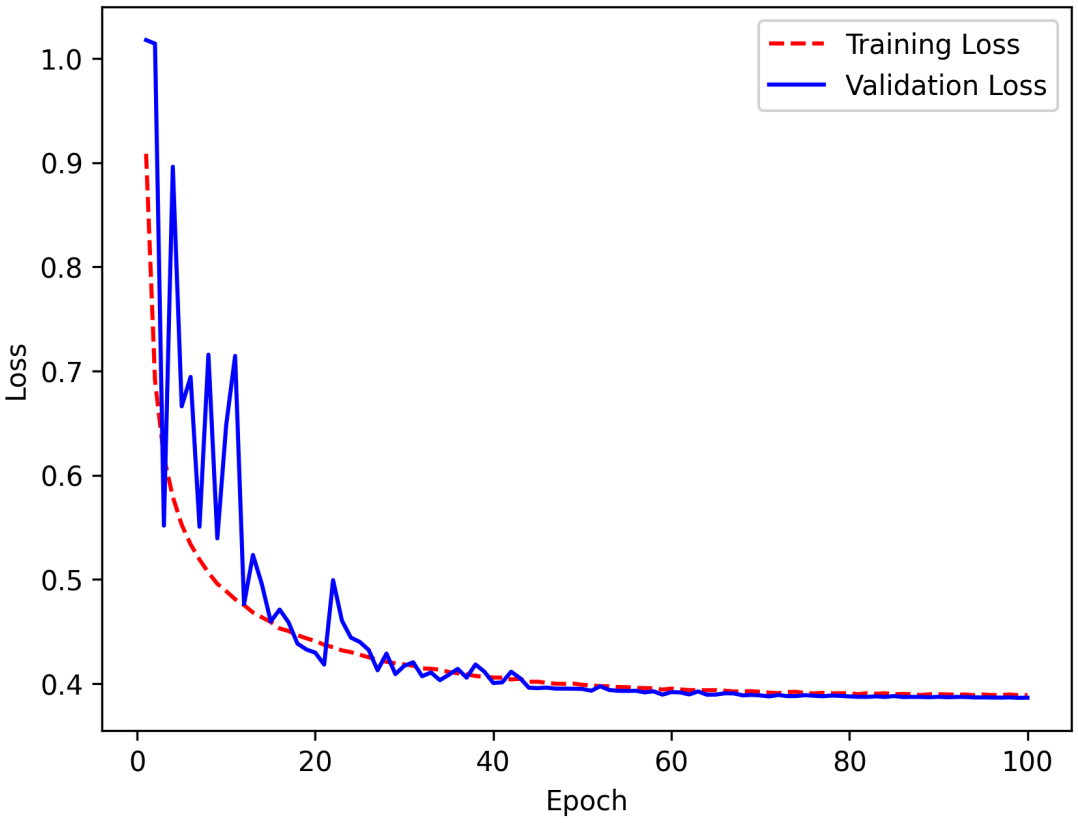


Figure 6-12: Epoch vs Loss graph for CNN model

6.3.2. Precision Confidence:

A Precision-Confidence graph shows how precision changes as the confidence threshold varies in a classification model. It helps evaluate the trade-off between making fewer but more accurate predictions versus taking more risks with lower confidence.

The graph shows how precision changes with different confidence thresholds in the plastic classification model. At very low thresholds, precision is low because the model makes many incorrect predictions. However, as the threshold increases, precision rises sharply, meaning the model becomes more selective and reduces false positives. Around a threshold of 0.4 and beyond, precision stabilizes near 1.0 for all four plastic types, indicating that when the model does make a prediction, it is highly likely to be correct. The consistency across HDPE, PET, PP, and PS suggests balanced performance across classes, with the macro-average precision closely following individual class curves. Overall, the model achieves strong precision at moderate thresholds, making it reliable for minimizing misclassifications.

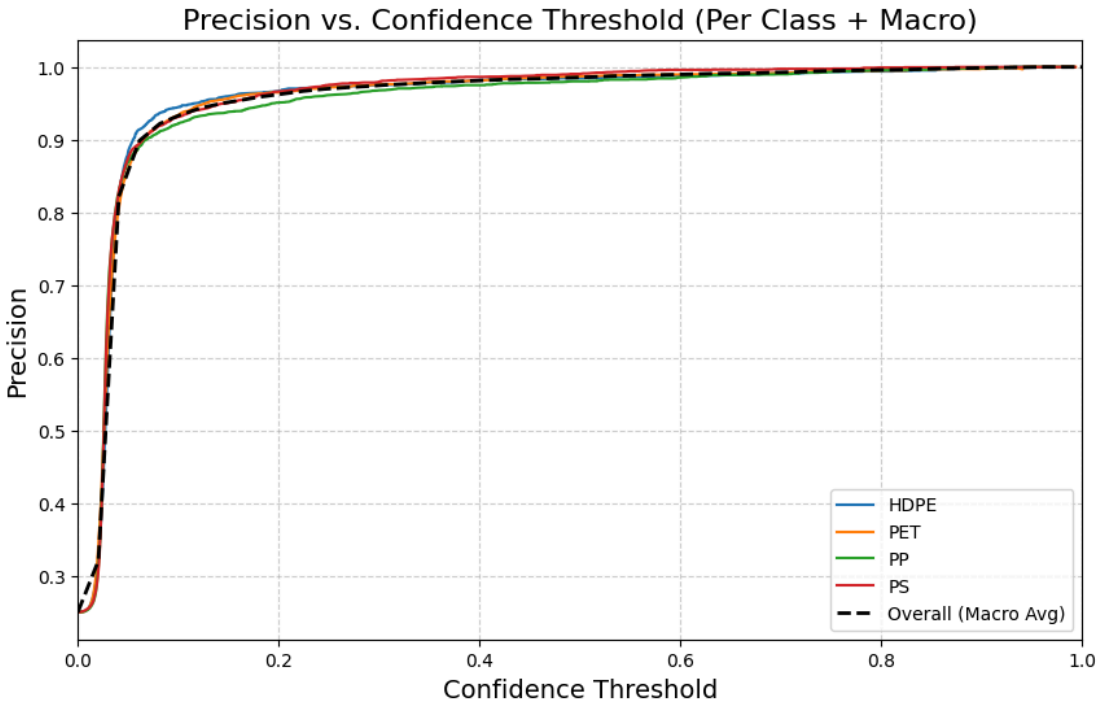


Figure 6-13:Precision Confidence Graph

6.3.3. Recall Confidence:

A Recall-Confidence graph shows how recall changes as the confidence threshold varies in a classification model. It helps assess the trade-off between capturing more true positives versus setting a higher confidence threshold.

The graph shows how recall changes with different confidence thresholds in the plastic classification model. At very low thresholds, recall is high (close to 1.0) because the model makes nearly all possible predictions, ensuring that most correct instances are captured. However, as the threshold increases, recall begins to decrease gradually, meaning the model becomes more selective and starts filtering out lower-confidence predictions.

Around a threshold of 0.8 and beyond, recall drops sharply. This indicates that while the model is highly confident in the predictions it does make, it is also discarding many correct classifications. The consistency across HDPE, PET, PP, and PS suggests balanced recall performance across classes, with the macro-average recall closely following individual class curves.Overall, the model achieves strong recall at lower thresholds, making it reliable for capturing most instances.

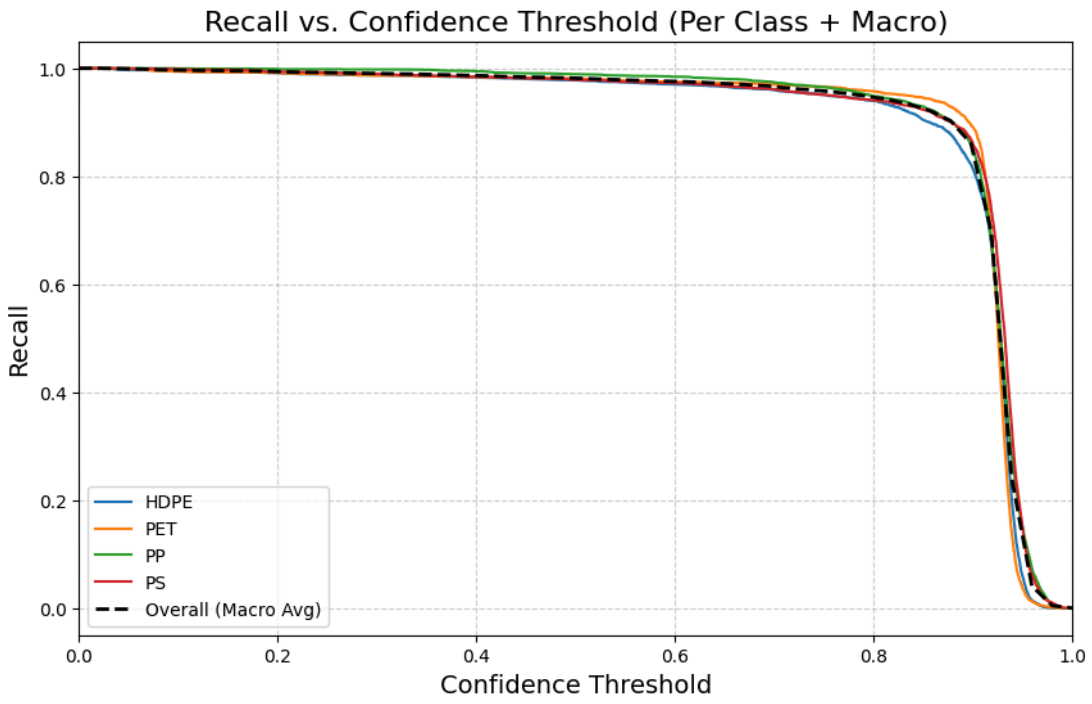


Figure 6-14:Recall Confidence Graph

6.3.4. F1 Score:

The below “F1-Confidence Curve” shows the relationship between the F1 score and the confidence threshold for the CNN model trained to classify a plastic.. The F1 score is a balanced measure of a model's precision and recall.

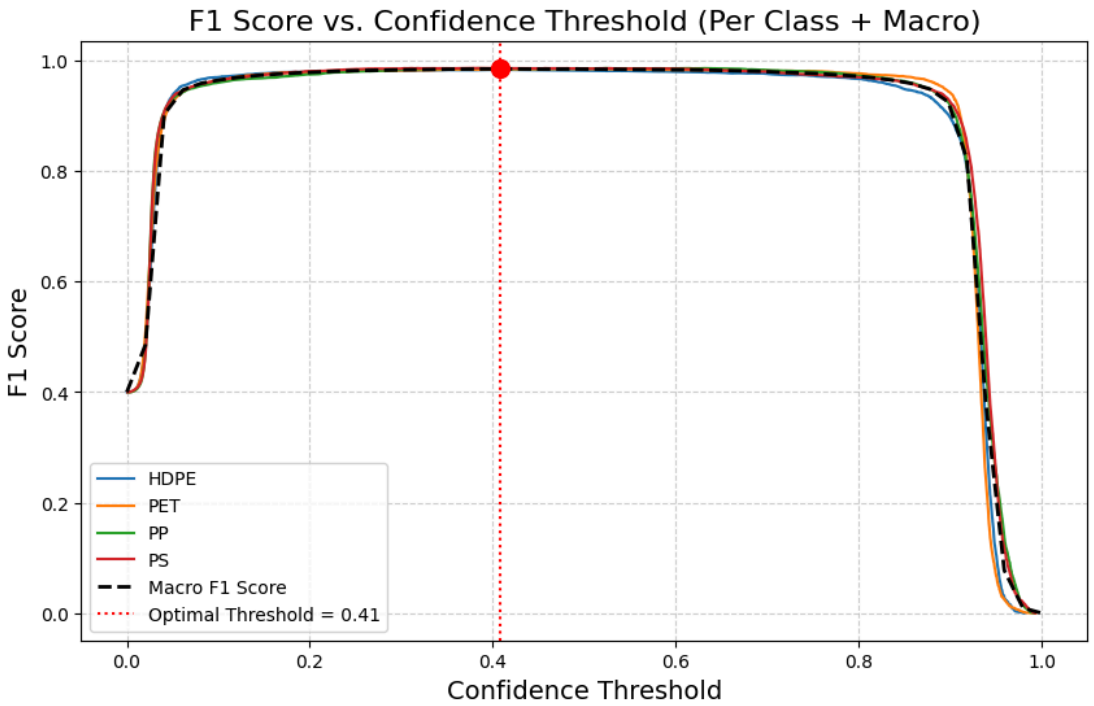


Figure 6-15: F1 Score vs Confidence for all classes

The graph illustrates how the F1 Score changes with different confidence thresholds for the plastic classification model. At very low thresholds, the model makes many incorrect predictions, leading to lower F1 Scores. As the threshold increases, the scores rise quickly and stabilize near 1.0, indicating a strong balance between precision and recall. However, when the threshold becomes too high, the model becomes overly cautious, leading to a drop in the F1 Score as fewer predictions are made. The dashed line represents the overall F1 Score across all classes, showing consistent performance. The dotted line marks the optimal threshold at 0.41, where the model achieves the best balance between making confident predictions and maintaining accuracy. With the scores reaching high values across all classes, the model performs well, though further refinements—such as fine-tuning the threshold for each class or analyzing misclassified samples—could further enhance its accuracy.

6.3.5. Confusion Matrix

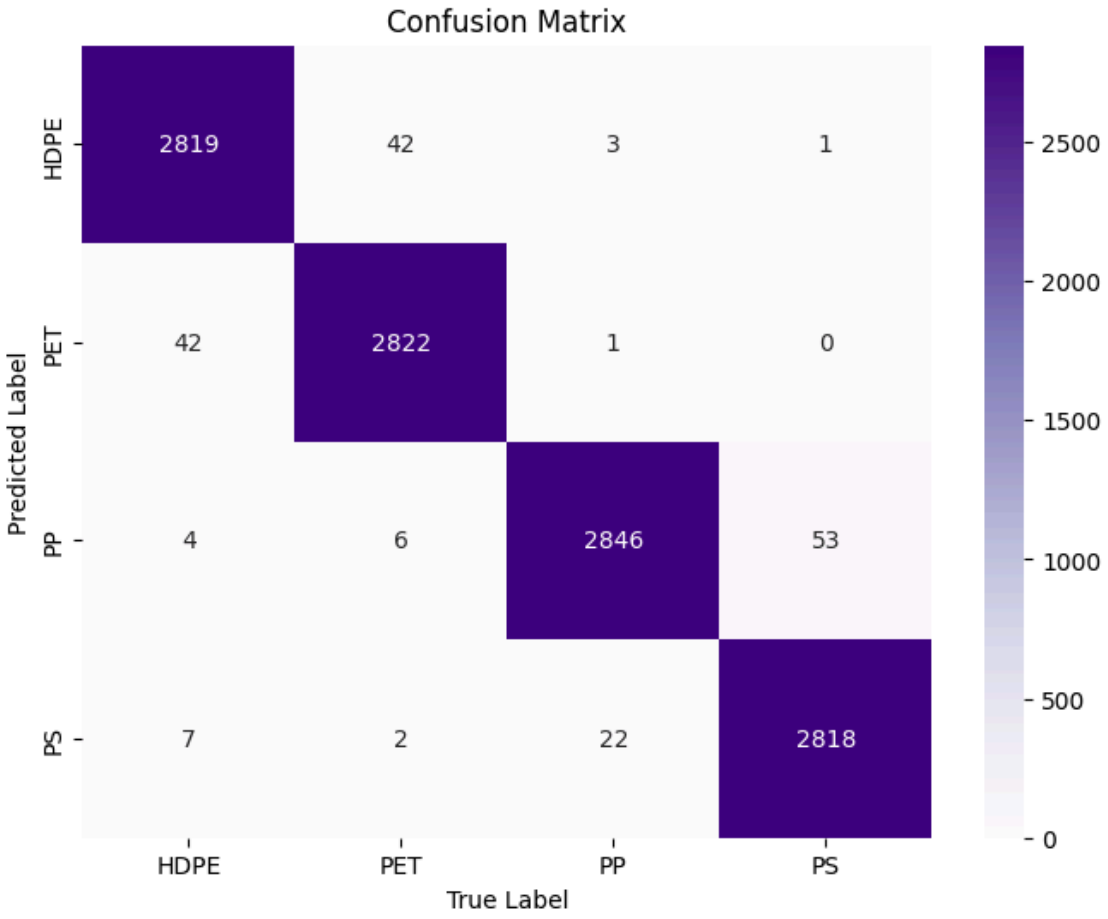


Figure 6-16: Confusion matrix for CNN Model

The confusion matrix shows that the model performs well in classifying HDPE, PET, PP, and PS, with most predictions aligning correctly. HDPE, PET, PP, and PS have high accuracy, with over 2,800 samples correctly classified in each category, indicating that the model effectively differentiates between them. However, some misclassifications highlight areas for improvement. PET is occasionally misclassified as HDPE (42 cases), likely due to visual similarities, while PP is most frequently misclassified as PS (53 cases), suggesting overlap in their distinguishing features. Despite these errors, overall misclassification rates are low, and the model maintains strong performance. To further enhance accuracy, refining the dataset or improving feature extraction could help address the PP-PS confusion.

6.4. Output of YOLO

After the training process was completed we obtained our best weight saved in the best.pt file and then we ran an inference on our yolov9t model on random images from Google and we obtained the following results. The initial results were satisfactory but it struggled a bit while detecting multiple objects in a single frame. The cause for this problem must be lack of sufficient image data containing **multiple objects** with **real life lighting conditions** and **insufficient training period.**

Figure 6-17: Inference on an Image having Single Object

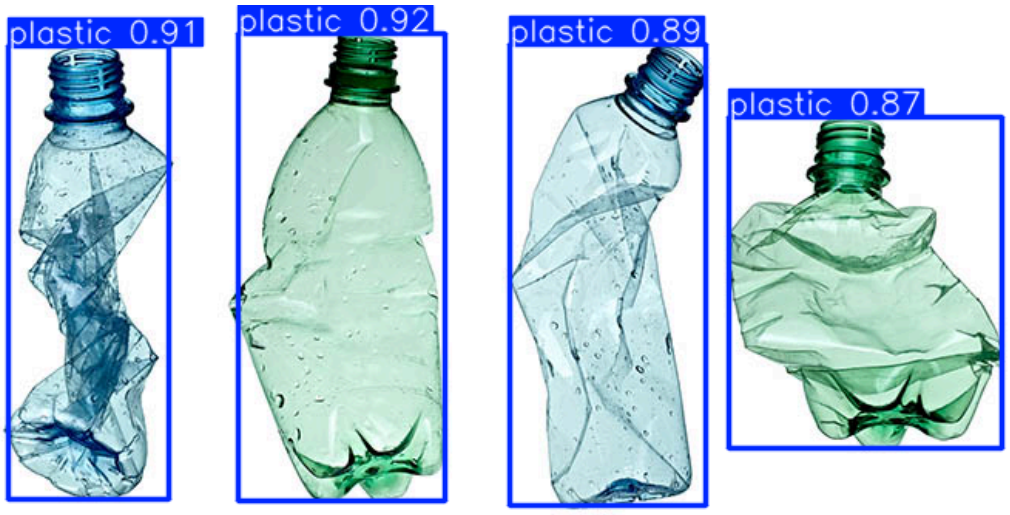


Figure 6-18: Inference on Image containing multiple objects

Figure 6-19: YOLOv9 Detecting Multiple Objects in Real Lighting Conditions

6.5. Output of Custom CNN Model

This is the output of our CNN Model choosing 2 random images from each class and running inference on them.

Figure 6-20:CNN’s inference on images from different classes

6.6. User Interface and Integrated Output

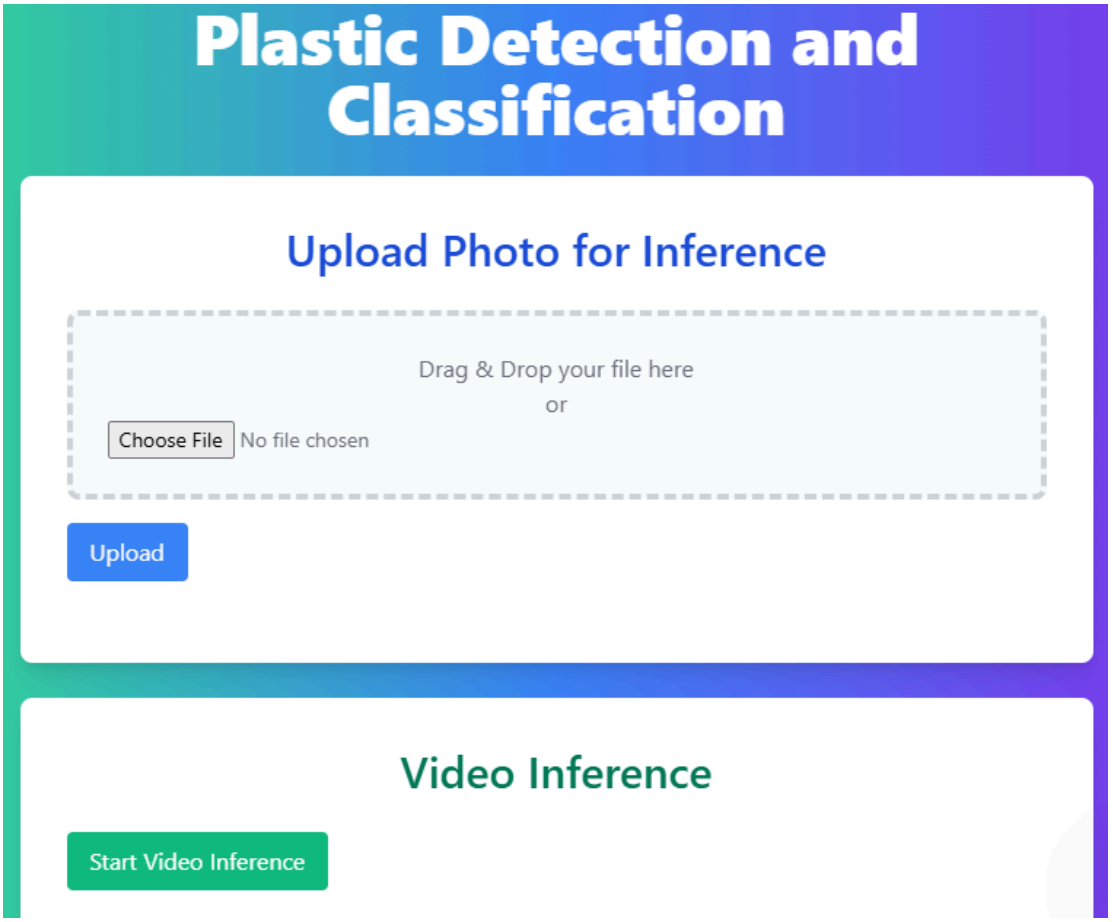


Figure 6-21: Web User Interface

This is the first page of the web user interface. It has options for uploading an image for photo inference and a button to start live video inference.

Here in figure 6-21 we can see a test image is being uploaded to the UI for photo inference and its preview is shown as well. The result of this inference is shown in figure 6-22 where we can observe that our integrated model inference is able to differentiate between HDPE and PET plastics .

Figure 6-23 shows a glimpse of live video inference run through a webcam. It's clear that it is able to classify HDPE and PET plastics with confidence scores 0.73 and 0.92 respectively. The figure 6-22 and 6-23 shows our custom CNN capabilities for plastic classification. We achieved a video inference rate of 2 to 3 frames per second (FPS)

on an Intel Core i5 10th Gen CPU, depending on the number of objects to classify. We seek to optimize this performance further.

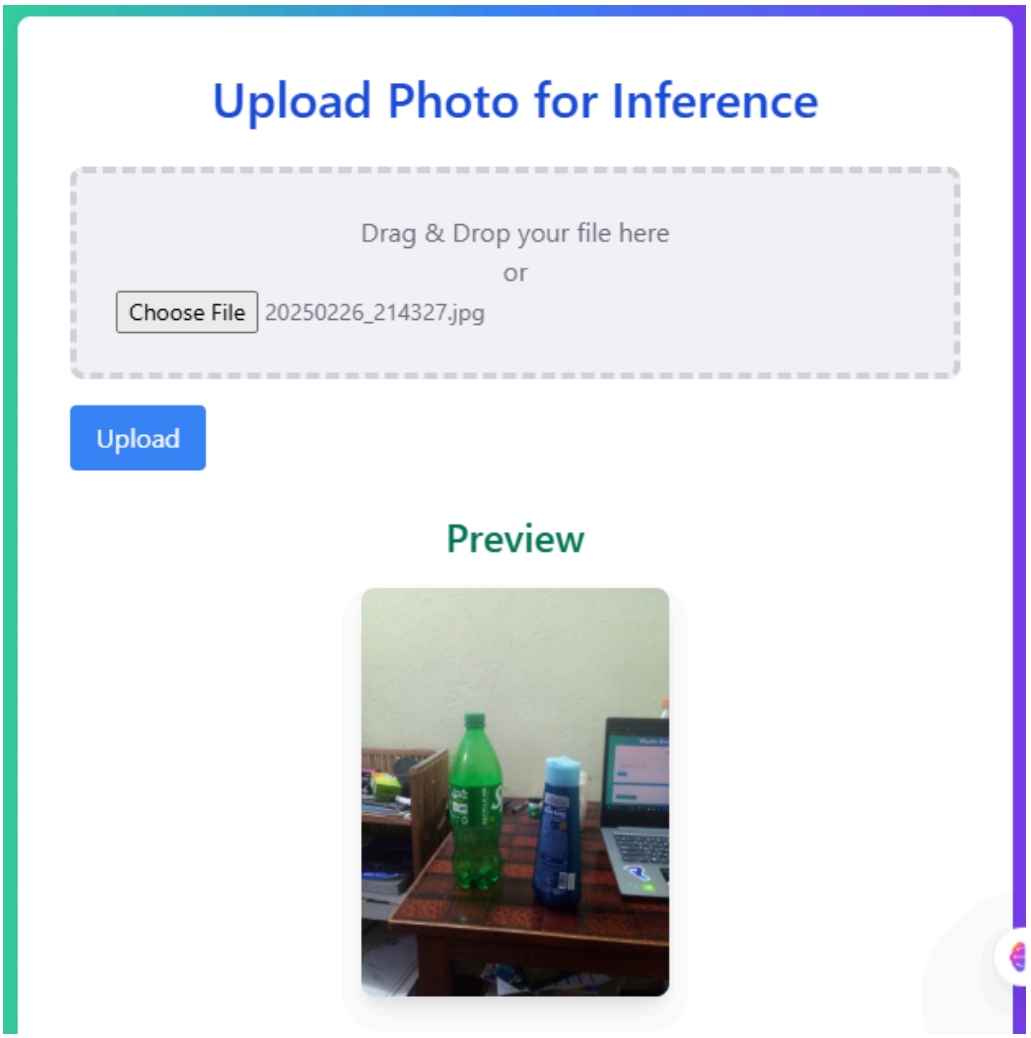


Figure 6-22: Uploading Image for Inference

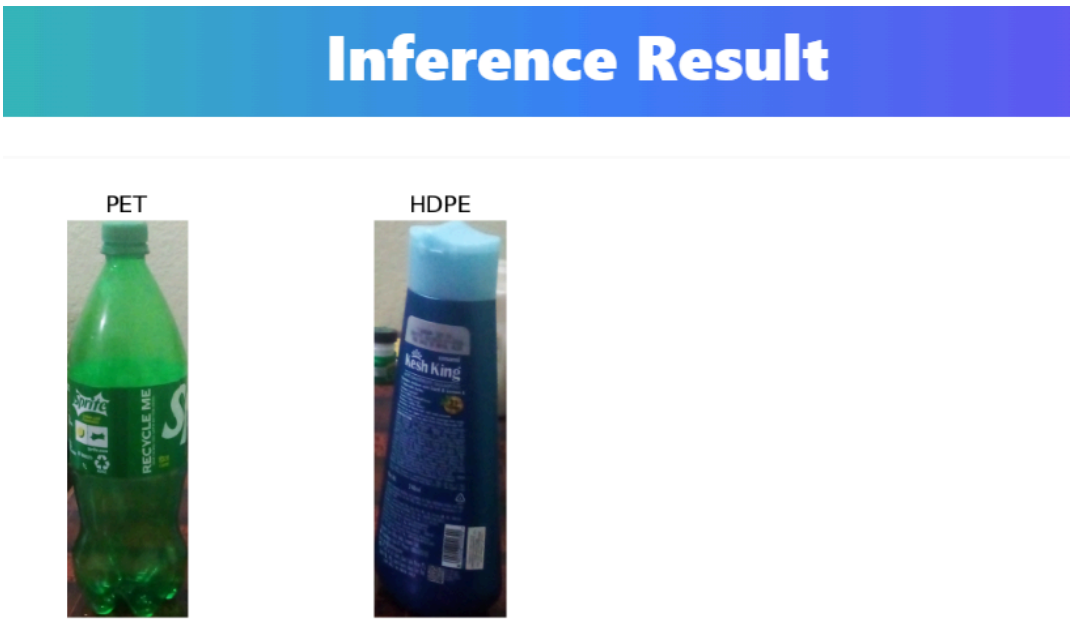


Figure 6-23: Photo inference result

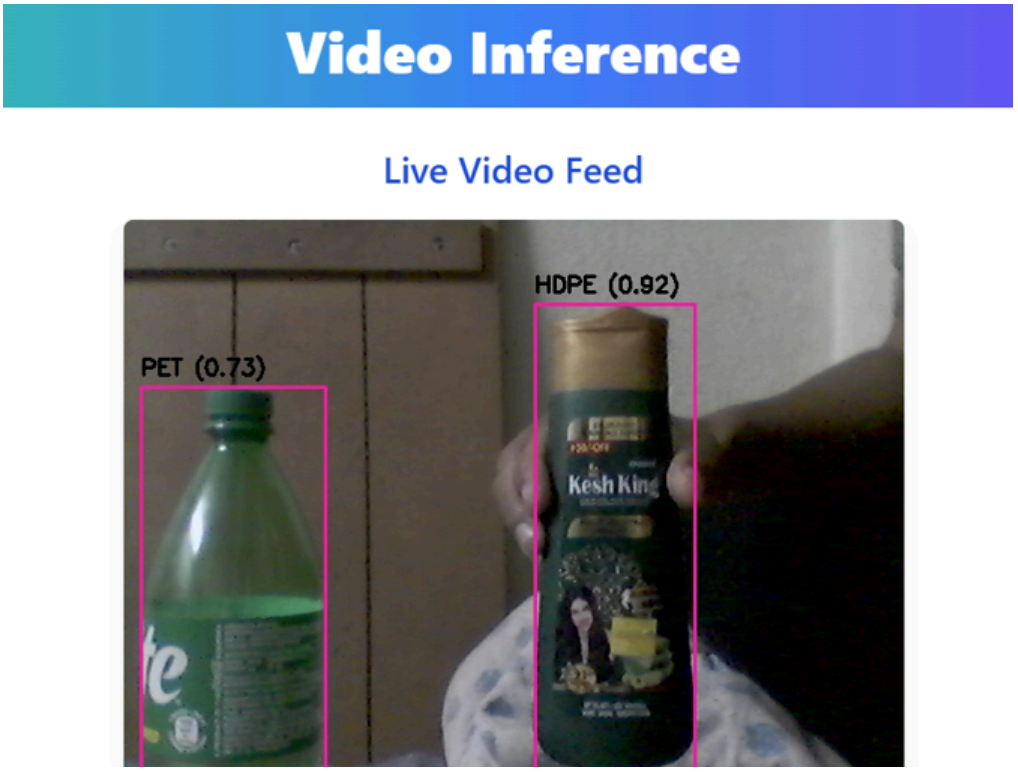


Figure 6-24: A glimpse of Video inference

7. FUTURE ENHANCEMENTS

7.1. Dataset Expansion

Benchmark dataset is very important for efficient and successful plastic detection and classification. With a better dataset we can achieve better classification results. In the current models the dataset used is largely limited . There is a possibility to include a wide range of commonly seen plastics in municipal waste to further generalize the model's capability. With the development of benchmark plastic waste dataset waste management will be much more economical, convenient and automated.

7.2. Improving Our Custom CNN Architecture

Although our custom CNN model is able to distinguish between plastic types its performance is not as we expected. But we can work on refining our CNN's architecture to improve feature extraction and make it more robust, efficient for plastic classification. There is a room for more experiments in the model's architecture to achieve benchmarking results.

7.3. Improving Web Application

We have developed a minimalistic UI and backend to run on our local computer. It just has a basic design and features right now. More improvements can be done on the frontend as well as backend for optimized performance and memory usage. The web application can be hosted on the cloud so that real time plastic detection and classification can be performed on edge devices delivering smooth performance.

7.4. Enhance Real Time performance

The real time performance of our integrated model can be enhanced by converting into ONNX, TensorRT which will enable optimized execution across various hardware platforms with reduced inference latency. Additionally model quantization and pruning can be done to reduce model size and speed up inference without losing accuracy. Such an optimized model can be used with Raspberry Pi, Coral TPU which are low power devices making the system more practical for real world applications.

8. CONCLUSION

The objective “To develop an automated system for accurate detection of plastic waste and classify its type in real time” has been completed however the performance of the model is affected by different lighting conditions. Similarly, the objective “To enhance recycling efficiency by implementing deep neural networks” is fulfilled as the system is able to predict commonly found recyclable plastics types with a respectable accuracy.

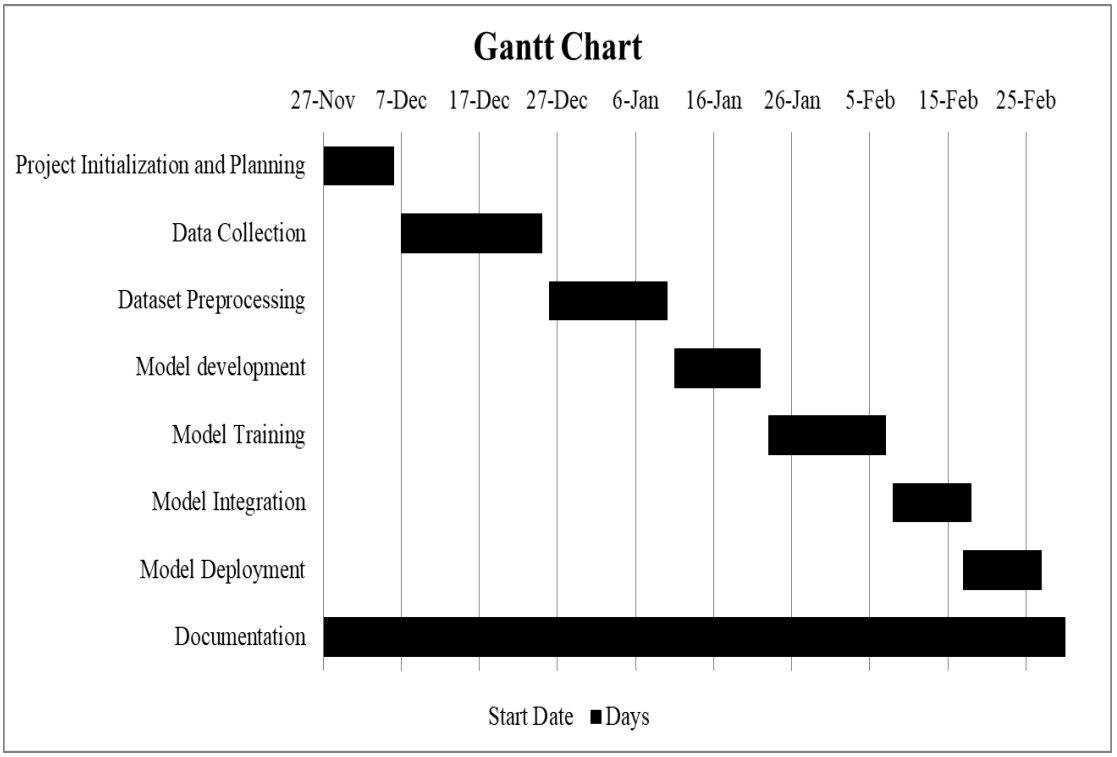
After working on this project we came to a conclusion that most of the commonly found plastic waste can be classified on the the basis of visual characteristics like edges, shape and textures provided that integrated model approach is used just like YOLO for detection and CNN for classification. The proper working of this approach relies on benchmark waste dataset and more advanced feature extraction by CNN.

This project provided us with a great learning experience, enhancing our understanding of deep neural networks especially convolutional neural networks. This has equipped us with a combination of skills in machine learning i.e. dataset preparation, model development, feature extraction, etc.

9. APPENDICES

Appendix A: Project Timeline

Table 8-1: Gantt Chart



Appendix B: Code Snippets

```
def residual_block(x, filters):
    shortcut = x

    if x.shape[-1] != filters:
        shortcut = Conv2D(filters, (1, 1),
padding='same')(shortcut)
        shortcut = BatchNormalization()(shortcut)
    x = Conv2D(filters, (3, 3), padding='same')(x)
    x = BatchNormalization()(x)
    x = Activation('swish')(x)
    x = Conv2D(filters, (3, 3), padding='same')(x)
    x = BatchNormalization()(x)
    x = Add()([shortcut, x])
    return Activation('swish')(x)

def se_block(x, ratio=16):
    channels = x.shape[-1]
    se = GlobalAveragePooling2D()(x)
    se = Dense(channels // ratio, activation='swish')(se)
    se = Dense(channels, activation='sigmoid')(se)
    se = Reshape((1, 1, channels))(se)
    return Multiply()([x, se])

def create_grayscale_cnn(input_shape, num_classes):
    inputs = Input(shape=input_shape)

    # Stem (Initial Convolution)
    x = Conv2D(24, (7, 7), strides=2, padding='same')(inputs)
    x = BatchNormalization()(x)
    x = Activation('swish')(x)
    x = MaxPooling2D((3, 3), strides=2)(x)

    # Backbone (Feature Extraction)
    x = residual_block(x, 48)
    x = se_block(x)
    x = MaxPooling2D((2, 2))(x)

    x = residual_block(x, 96)
    x = se_block(x)
    x = MaxPooling2D((2, 2))(x)
```

```
    # Head (Classification Layer)
    x = GlobalAveragePooling2D()(x)
    x = Dense(256, activation='swish',
kernel_regularizer=l2(0.001))(x)
    x = Dropout(0.5)(x)
    outputs = Dense(num_classes, activation='softmax')(x)
    model = Model(inputs, outputs)

    model.compile(optimizer =
tf.keras.optimizers.Adam(learning_rate=lr_schedule),
loss=tf.keras.losses.CategoricalCrossentropy(label_smoothing=
0.1), metrics=['accuracy'])
    return model
```

Figure 8-1: Notebook code for defining CNN Model

```
#installing Ultralytics library
!pip install ultralytics

#importing YOLO model
from ultralytics import YOLO
model = YOLO("yolov9s.pt")

# Downloading our final dataset from Roboflow
!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="CM3o4LDQzu8QMJSzfoyY")
project =
rf.workspace("prajwal-lsapy").project("final-dataset-igcpo")
version = project.version(1)
dataset = version.download("yolov9")

#Training YOLO
results = model.train(data='/content/dataset/data.yaml',
epochs=20, imgsz =640, batch = 32, save = True, save_period =
1)
```

Figure 8-2: Notebook code for YOLOv9 training

```
@app.route('/inference/<filename>')
def inference(filename):
    filepath = os.path.join(app.config['UPLOAD_FOLDER'],
filename)
    original_img = cv2.imread(filepath)
    original_img_rgb = cv2.cvtColor(original_img,
cv2.COLOR_BGR2RGB)

    # YOLO inference
    yolo_result = yolo_model(source=filepath, conf=0.5)
    cropped_images = []
    labels = []

    # Iterate over detected objects
    for detection in yolo_result[0].boxes:
        x1, y1, x2, y2 = map(int, detection.xyxy[0])
        cropped_img = original_img_rgb[y1:y2, x1:x2]

        # Convert to grayscale
        cropped_img_gray = cv2.cvtColor(cropped_img,
cv2.COLOR_RGB2GRAY)

        img = cv2.resize(cropped_img_gray, (224, 224))
        img_array = img / 255.0
        img_array = np.expand_dims(img_array, axis=-1)
        img_array = np.expand_dims(img_array, axis=0)

        cnn_prediction = cnn_model.predict(img_array)
        predicted_class_index = np.argmax(cnn_prediction)
        predicted_label = class_labels[predicted_class_index]

        cropped_images.append(cropped_img)
        labels.append(predicted_label)

    # Display cropped images in a grid with labels
    num_columns = 4
    num_rows = max(1, (len(cropped_images) + num_columns - 1)
// num_columns)
    fig, axes = plt.subplots(num_rows, num_columns,
figsize=(15, 5 * num_rows))
    plt.subplots_adjust(hspace=0.5)  # Add vertical space
between rows
```

```
    for i, (img, label) in enumerate(zip(cropped_images,
labels)):
        row = i // num_columns
        col = i % num_columns
        ax = axes[row, col] if num_rows > 1 else axes[col]
        ax.imshow(img)
        ax.set_title(label)
        ax.axis('off')

    # Hide any unused subplots
    for j in range(len(cropped_images), num_rows *
num_columns):
        row = j // num_columns
        col = j % num_columns
        ax = axes[row, col] if num_rows > 1 else axes[col]
        ax.axis('off')

    output_filepath =
os.path.join(app.config['UPLOAD_FOLDER'], 'output.png')
    plt.savefig(output_filepath)
    plt.close()

    return render_template('inference.html',
filename='output.png')
```

Figure 8-3: Backend Code for Photo Reference

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

def generate_video_frames():
    cap = cv2.VideoCapture(0)
    if not cap.isOpened():
        return "Error opening video stream"

    while cap.isOpened():
        success, frame = cap.read()
        if not success:
            break
```

```
        # Resize frame for faster processing
        frame_resized = cv2.resize(frame, (640, 640))

        # Run YOLO detection
        results = yolo_model(frame_resized, conf=0.5)[0]

        for (x_min, y_min, x_max, y_max) in
results.boxes.xyxy:
            x_min, y_min, x_max, y_max = map(int, [x_min,
y_min, x_max, y_max])

            # Scale coordinates back to original frame size
            x_min = int(x_min * frame.shape[1] /
frame_resized.shape[1])
            y_min = int(y_min * frame.shape[0] /
frame_resized.shape[0])
            x_max = int(x_max * frame.shape[1] /
frame_resized.shape[1])
            y_max = int(y_max * frame.shape[0] /
frame_resized.shape[0])

            # Crop detected object
            cropped_image = frame[y_min:y_max, x_min:x_max]
            if cropped_image.size == 0:
                continue

            # Convert to grayscale
            cropped_image_gray = cv2.cvtColor(cropped_image,
cv2.COLOR_BGR2GRAY)

            # Resize to CNN input size
            cropped_image_gray =
cv2.resize(cropped_image_gray, (224, 224))
            cropped_image_gray = cropped_image_gray / 255.0
            cropped_image_gray =
np.expand_dims(cropped_image_gray, axis=-1)  # Add channel
dimension
            cropped_image_gray =
np.expand_dims(cropped_image_gray, axis=0)  # Add batch
dimension
```

```
            # Predict with CNN
            prediction =
cnn_model.predict(cropped_image_gray)
            class_idx = np.argmax(prediction)
            class_name = class_labels[class_idx]
            confidence = prediction[0][class_idx]

            # Draw bounding box and label
            cv2.rectangle(frame, (x_min, y_min), (x_max,
y_max), (160, 32, 240), 2)
            label = f"{class_name} ({confidence:.2f})"
            cv2.putText(frame, label, (x_min, y_min - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 0), 2)

        ret, buffer = cv2.imencode('.jpg', frame)
        frame = buffer.tobytes()

        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + frame +
b'\r\n')

    cap.release()

@app.route('/video_feed')
def video_feed():
    return Response(generate_video_frames(),
mimetype='multipart/x-mixed-replace; boundary=frame')

@app.route('/start_video_inference', methods=['POST'])
def start_video_inference():
    return redirect(url_for('video_inference'))
```

Figure 8-4: Backend Code for Webcam Reference

Appendix C: YOLO Components

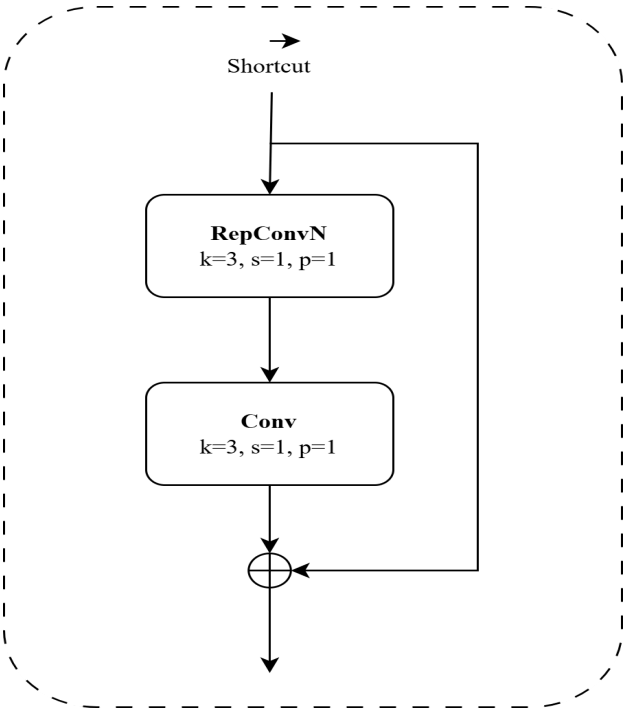


Figure 8-5: RepNBottleneck(Shortcut = True)

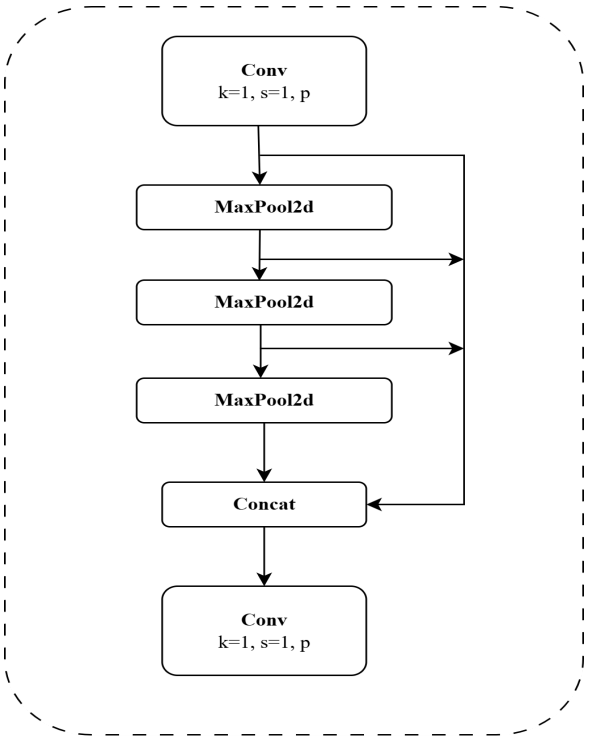


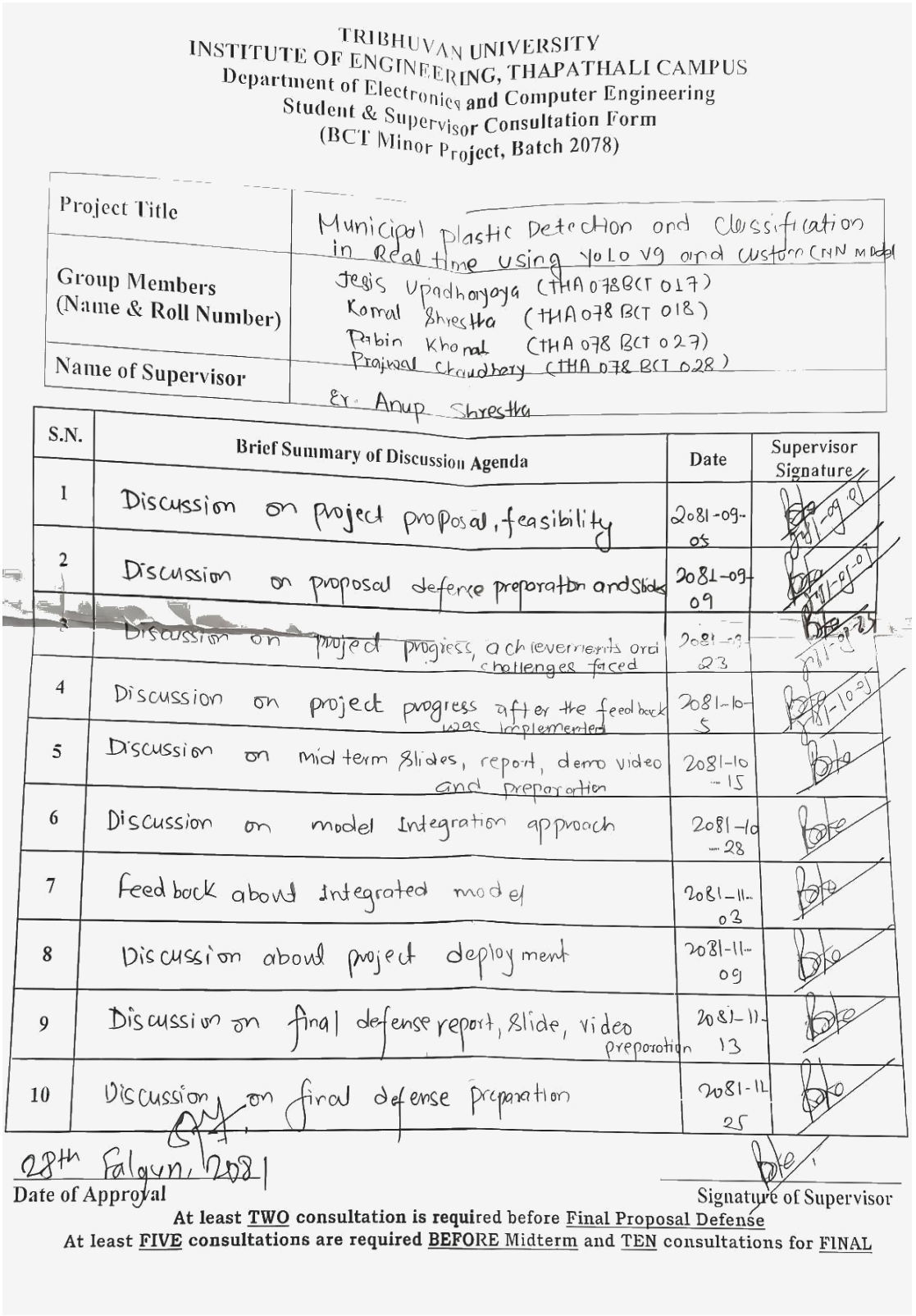
Figure 8-6: SPPELAN Block

Appendix D: Dataset Details

Table 8-2: Dataset Statistics

Number of set \ code	PET	PE-HD	PP	PS
1	5			
2	2	1		2
3	3		2	
4	2		1	1
5	2	2	1	
6	4			1
7	2			3
8	2	1	2	
9	3			2
10	4		1	
11	4		1	
12	3			2
13	1	4		
14	4		1	
15	3	1		1
16	1	3	1	
17	4		1	
18	4		1	
19	3		1	1
20	1	3	1	
Total	57	15	14	13

Appendix E: Student Supervisor Consultation Form



TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING, THAPATHALI CAMPUS
Department of Electronics and Computer Engineering
Student & Supervisor Consultation Form
(BCT Minor Project, Batch 2078)

Project Title	Municipal plastic Detection and Classification in Real time using YoLo v9 and custom CNN Model
Group Members (Name & Roll Number)	Jesis Upadhayaya (THA078BCT017) Komal Shrestha (THA078BCT018) Rabin Khanal (THA078BCT027) Prajwal Chaudhary (THA078BCT028)
Name of Supervisor	Er. Anup Shrestha

S.N.	Brief Summary of Discussion Agenda	Date	Supervisor Signature
1	Discussion on project proposal, feasibility	2081-09-05	
2	Discussion on proposal defense preparation and Slides	2081-09-09	
3	Discussion on project progress, achievements and challenges faced	2081-09-23	
4	Discussion on project progress after the feedback was implemented	2081-10-5	
5	Discussion on mid term slides, report, demo video and preparation	2081-10-15	
6	Discussion on model integration approach	2081-10-28	
7	Feedback about integrated model	2081-11-03	
8	Discussion about project deployment	2081-11-09	
9	Discussion on final defense report, slide, video preparation	2081-11-13	
10	Discussion on final defense preparation	2081-11-25	

28th Falgun, 2081
Date of Approval

Signature of Supervisor

At least TWO consultation is required before Final Proposal Defense
At least FIVE consultations are required BEFORE Midterm and TEN consultations for FINAL

Figure 8-7: Student Supervisor Consultation Form

Appendix F: Plagiarism Report

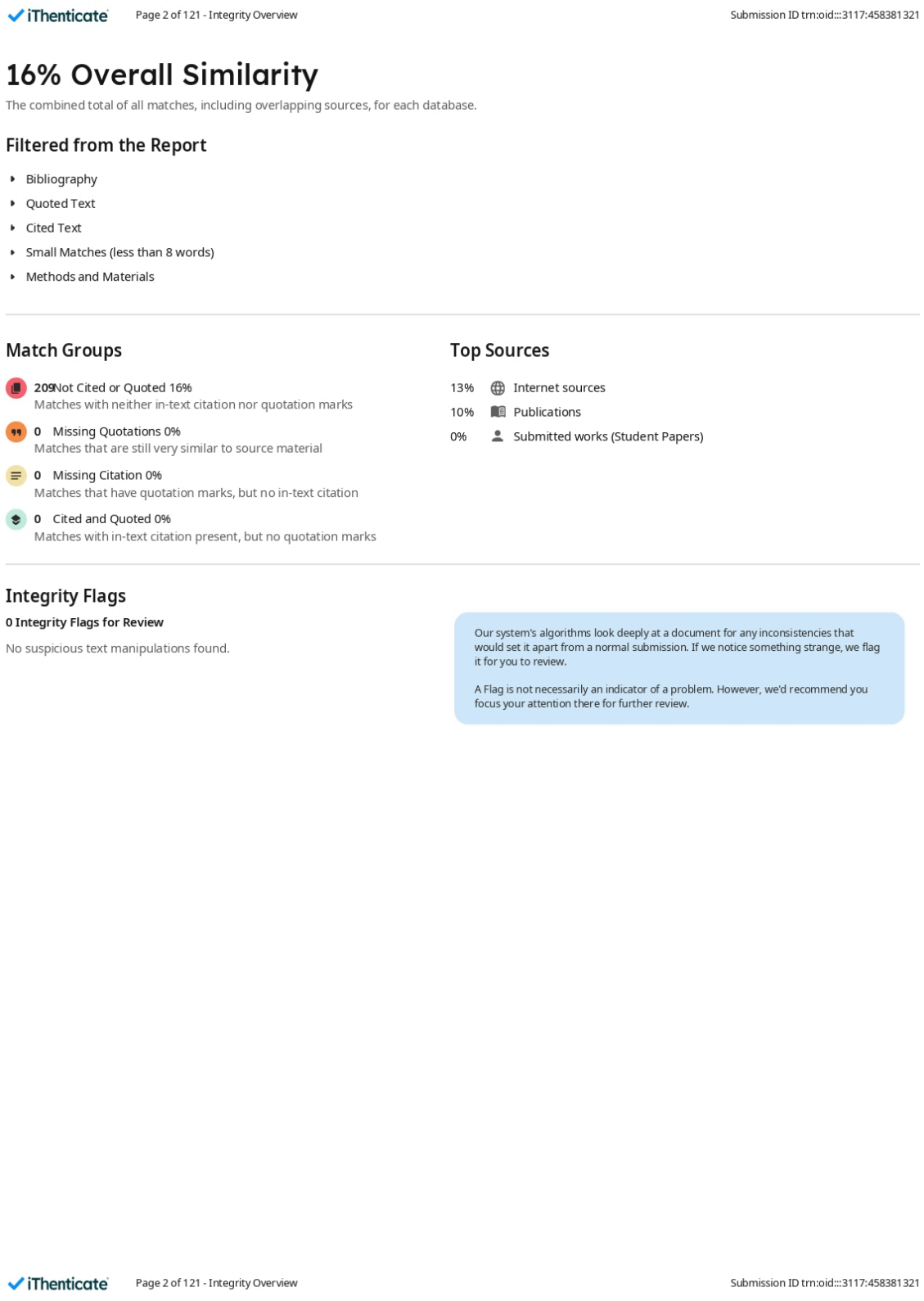


Figure 8-8: Summary Page of Plagiarism Report

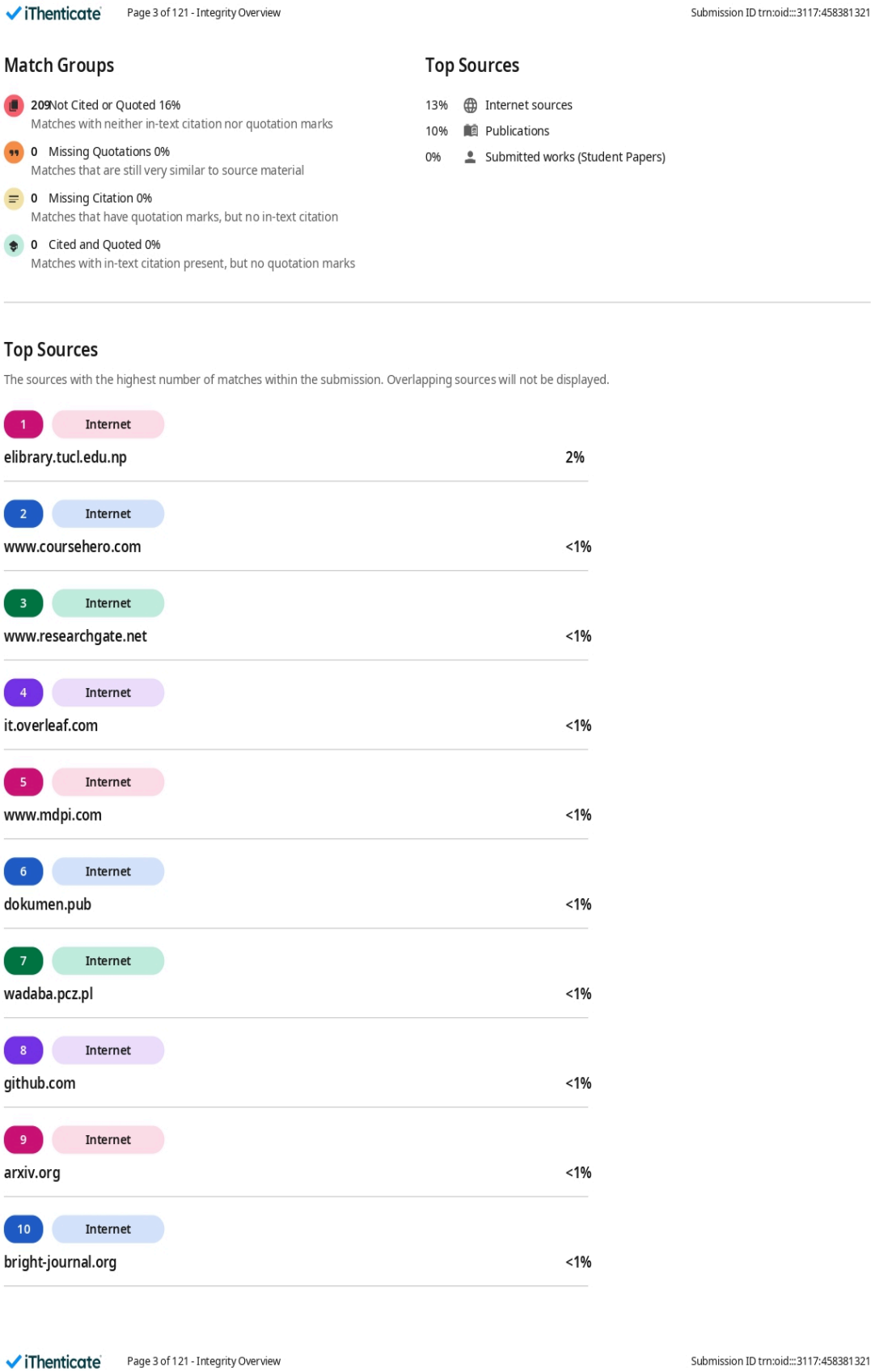


Figure 8-9: Plagiarism Report Showing Top 10 Sources

REFERENCES

[1] W. Rahman, R. Islam, A. Hasan, N. Bithi, M. Hasan, and M. M. Rahman, “Intelligent waste management system using deep learning with IoT,” Journal of King Saud University - Computer and Information Sciences, Sep. 2020.

[2] M. Hassan et al., “Advanced data augmentation techniques for improving dataset diversity in plastic waste classification,” *Journal of Computer Vision and Machine Learning*, 2020.

[3] E. Gothai, R. Thamilselvan, P. Natesan, M. Keerthivasan, K. Kabinesh, and D. K. Ruban, “Plastic waste classification using CNN for supporting 3 R's principle,” in 2022 *International Conference on Computer Communication and Informatics (ICCCI*), Coimbatore, India, pp. 01–07, 2022.

[4] O. Tamin, “Machine learning for plastic waste detection: State-of-the-art, challenges, and solutions,” IEEE, 2023. [Online]. Available: https://ieeexplore.ieee.org/document/9990703

[5] J. Choi, B. Lim, and Y. Yoo, “Advancing plastic waste classification and recycling efficiency: Integrating image sensors and deep learning algorithms,” *Applied Sciences*, vol. 13, 2023.

[6] E. Ramos, A. G. Lopes, and F. Mendonça, “Application of machine learning in plastic waste detection and classification: A systematic review,” Processes, vol. 12, no. 8, 2024. [Online]. Available: https://www.mdpi.com/2227-9717/12/8/1632

[7] L. Guezouli, G. Aridj, and L. Saher, “CNN-based plastic waste detection system,” *BOHR International Journal of Smart Computing and Information Technology*, vol. 4, pp. 79–85, 04 2023.

[8] A. El Zaar, A. Aoulalay, N. Benaya, A. El Mhouti, M. Massar, and A. El Allati, "A Deep Learning Approach to Manage and Reduce Plastic Waste in the Oceans," E3S Web of Conferences, vol. 337, p. 00065, 2022. [Online]. https://www.e3s-conferences.org/articles/e3sconf/pdf/2022/03/e3sconf_icegc2022_00065.pdf

[9] A. Zhang, Z. Lipton, M. Li, and A. Smola, “Dive into deep learning,” 06 2021.

[10] J. Bobulski and J. Piatkowski, “Pet waste classification method and plastic waste database - wadaba,” 01 2018, pp. 57–64.

[11] J. Bobulski and M. Kubanek, “Deep learning for plastic waste classification system,” *Applied Computational Intelligence and Soft Computin*g, vol. 2021, 05 2021.

[12] C.-Y. Wang, I.-H. Yeh, and H.-Y. M. Liao**,** "YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information," in *Computer Vision – ECCV 2024*, A. Leonardis, E. Ricci, S. Roth, O. Russakovsky, T. Sattler, and G. Varol, Eds. Cham: Springer Nature Switzerland, 2025, pp. 1–21. Available: https://github.com/WongKinYiu/yolov9.