FINAL REPORT

AI Based Detection of Roadside Rubbish for Smart City Waste Management

Tran Duc Anh Dang - 103995439 | Rupayan Banerjee - 103538229 | Vivek Saini - 103828056 | Lochana Hettiarachchi - 103517367 | Akindu Wikramarachchi - 104084367

Studio Session: 1-3 | Group: 2 | Theme: 1

Abstract

This project explores the application of artificial intelligence to detect and classify roadside rubbish in urban environments. With the increasing need to maintain clean and safe public spaces, city councils face challenges in monitoring illegal dumping. This study developed an AI model using YOLO (You Only Look Once) object detection algorithms to automate the identification of common rubbish items such as electrical goods, furniture, toys, and bags. Using a dataset of annotated roadside images, the project involved data augmentation, model training, and evaluation to enhance the model's accuracy and adaptability in real world scenarios. We trained multiple YOLO variations (YOLO11n and YOLO11s) across original and augmented datasets, analyzing their performance on metrics such as precision, recall, and mean Average Precision (mAP).

A key output of this project is an AI demonstrator that provides users with an interactive interface to upload images or videos, select a model, and receive real time detection feedback. This system offers city councils a practical tool to support faster response to waste management issues, integrating AI insights with a user friendly experience. The results demonstrate how AI can effectively assist in addressing urban cleanliness challenges, with future applications extending to other Smart City initiatives. This project showcases the potential for AI to transform waste management, offering a scalable, automated approach to urban maintenance.

Table of Contents

Introduction

Background and Motivation

As cities grow, keeping public areas clean and well-maintained becomes more important for the safety, health, and comfort of residents. Using AI technology, we can now monitor urban spaces more effectively and quickly identify issues like illegal dumping of rubbish on roadsides. This project aims to use AI to support city councils and waste management teams by detecting roadside rubbish automatically, which could help them clean up faster and keep streets safer and more appealing.

The main motivation for this project is to apply AI to a real world problem: keeping cities clean by identifying roadside rubbish. AI can help streamline the process, making it easier for city workers to know where rubbish has been dumped and what types of items are there. This could help city councils and maintenance teams plan better and respond faster, improving the environment for everyone.

Project Objectives

The main goal of this project is to develop an AI model that can detect and identify dumped rubbish on roadsides from images. This model will answer two main questions:

1. **What types of roadside issues can it find?**

   The model should be able to detect and label rubbish.

2. **What specific items are in the rubbish?**

   It should identify familiar objects within the 'electrical goods', 'mattress', 'furniture', 'toy', 'bag', 'clothes', 'electrical', 'chair'.

Completing this project will benefit both the project team and the users. For us, it builds technical skills in AI areas like image processing, data labelling, and model training. For city planners and engineers, the AI model could provide a new way to monitor roadside rubbish and plan clean-up efforts, supporting a cleaner and safer city environment.

Summary of Outcomes

This project uses a dataset of roadside images provided by city councils, specifically focusing on the “rubbish” dataset. After training the model, both without and with extra data processing (augmentation), the results show it can accurately detect and classify different types of rubbish, giving a confidence score for each item it identifies. This outcome highlights the model’s potential to support Smart City projects by helping city councils detect and respond to littering issues more efficiently.

Dataset

Data Source

The dataset for this project consists of roadside images provided by city councils, specifically focused on identifying dumped rubbish in urban areas. This dataset, organized in a folder labeled "rubbish," includes images of various types of discarded items, such as 'electrical goods', 'mattress', 'furniture', 'toy', 'bag', 'clothes', 'electrical', 'chair'. The images are used to train our AI model to recognize different types of rubbish accurately, providing essential data for detecting illegal dumping.

The dataset exists in two versions: an original set without any data augmentation and an augmented version. The augmented version enhances the model's ability to adapt to real world conditions by simulating different environments and lighting conditions.

Data Processing

To prepare the dataset for training, several steps were conducted:

1. **Data Cleanup**: We reviewed the dataset to ensure quality and relevance, discarding any low-quality or irrelevant images. Only images that clearly depicted roadside rubbish were included for annotation.
2. **Data Annotation with LabelMe**: Annotation was a key step in preparing the dataset. We used the LabelMe tool to manually label objects within the rubbish piles. In this step, bounding boxes were drawn around familiar objects like mattresses, bags, and other discarded items to teach the model how to identify them accurately. The labels were saved in JSON format, compatible with YOLO for easy conversion into the required format. LabelMe allowed precise annotations that facilitated accurate object detection and classification **(dataset.py).**
3. **Augmentation**: To create an augmented dataset, we applied various transformations to add diversity, simulate real-world variability, and improve model robustness. This augmentation pipeline, implemented in **augment_dataset.py**, included:
    - **Horizontal Flip**: Simulates variation in object orientation.
    - **Brightness and Contrast Adjustments**: Represents different lighting conditions.
    - **Rotation and Scaling**: Adds robustness against slight image variations.
    - **Blur and Noise**: Mimics environmental noise or slight motion blur.
    - **Color Adjustments and Distortions**: Simulates natural color changes and distortions.
4. These augmentations were applied selectively with specified probabilities, ensuring that the model was exposed to a wide range of scenarios. The pipeline also maintained bounding box alignment with transformed images to retain labelling accuracy **(augment_dataset.py)**.
5. **Train-Validation Split**: Using a 90-10 split, the dataset was divided into training and validation sets, ensuring that 10% of the data was reserved for validating the model on unseen images. This split was managed with **train_test_split** from **sklearn**, to create a balanced, randomized division of images between the sets **(dataset.py and augment_dataset.py)**.

A.I Model Development

Feature Engineering

Feature engineering involves refining the data input to the model to improve its learning capability and accuracy. In this project, feature engineering included image specific enhancements and augmentations to help the model learn patterns in roadside rubbish effectively. For example, brightness and contrast adjustments were applied during augmentation to simulate different lighting conditions, ensuring that the model could generalise well across various environments. This preprocessing helped create more realistic training data, especially important for enhancing the model's performance on urban images.

Feature Extraction

In the context of this project, feature extraction is handled by the YOLO model itself, which learns features directly from the input images to identify and classify objects. YOLO's architecture is well suited for extracting visual features such as shape, texture, and size, which are crucial for distinguishing between different types of objects within rubbish piles. These features allow YOLO to detect and label objects accurately within each image, enabling the model to recognize various objects, such as bag, furniture, or other discarded items.

Image Processing

Image processing was a critical part of data preparation and included the following steps:

1. **Resizing**: Images were resized to 640x640 pixels to ensure compatibility with YOLO's input requirements and to standardize the data.
2. **Normalization**: Pixel values were normalized to a standard range, which helped improve the model's convergence during training by ensuring consistent image quality across the dataset.
3. **Data Augmentation**: As discussed in section 2.2, various augmentations, such as flipping, brightness changes, and noise addition, were applied. This step expanded the dataset's diversity and improved model resilience to real world variations. The augmentation process was implemented in the script, applying transformations with controlled probabilities to maintain a balance between the original and transformed images **(augment_dataset.py**).

Train/Test Split

The dataset was split into training and validation sets with a 90-10 ratio, meaning 90% of the images were used for training, while 10% were used to validate the model's performance. This split was chosen to ensure the model had enough data to learn effectively while also having a separate set to evaluate its generalization on unseen images. The split was conducted using **train_test_split** from **sklearn**, providing a balanced and randomized selection of images across the two sets **(dataset.py and augment_dataset.py**).

Training Model

The model training was conducted using two variations of the YOLO model, YOLO11n and YOLO11s, each trained on both the original and augmented versions of the dataset for a total of 150 epochs. This resulted in four distinct models, which were optimized for detecting multiple object types within the images. Key parameters and setup details for the training process included:

- **Model Framework**: The YOLO models were implemented using the **ultralytics** library, allowing for efficient setup and optimization in object detection tasks.
- **Batch Size**: A batch size of **32** was selected to maintain a balance between training speed and model performance. This batch size ensured stable convergence without exceeding memory constraints.
- **Hyperparameter Tuning**: Basic tuning focused on the learning rate and batch size. Additionally, augmentation specific parameters, such as brightness, rotation, and contrast adjustments, were fine tuned to simulate diverse real world conditions and improve model robustness.
- **Training Framework**: YOLO models were configured to save the best performing weights during training for use in the final evaluation. The augmentation settings were managed using **augment_dataset.py** and the training script **train.py** for seamless integration.

Evaluation of A.I Model

The performance of the four trained models YOLO11n and YOLO11s, each trained on original and augmented datasets was evaluated based on key metrics such as precision, recall, mean Average Precision (mAP), and validation losses. These metrics provide a comprehensive assessment of the model's detection, classification, and generalization capabilities.

Summary of Model Performance

Model	Train Box Loss	Train Class Loss	Train DFL Loss	Precision	Recall	mAP50	mAP50-95	Val Box Loss	Val Class Loss	Val DFL Loss
YOLO11n Augment	0.39463	0.25209	0.87084	0.34046	0.19285	0.23865	0.15024	1.88539	2.52548	2.40333
YOLO11n Original	0.45896	0.55118	0.86405	0.73972	0.19834	0.25287	0.18329	1.36516	2.67231	1.62438
YOLO11s Augment	0.29770	0.19625	0.82887	0.26962	0.25356	0.27200	0.17867	1.81857	2.14878	2.34050
YOLO11s Original	0.34352	0.29173	0.82906	0.35938	0.29048	0.29828	0.20837	1.48941	3.01689	1.79776

Visual Comparison of Key Metrics

Precision and Recall Comparison

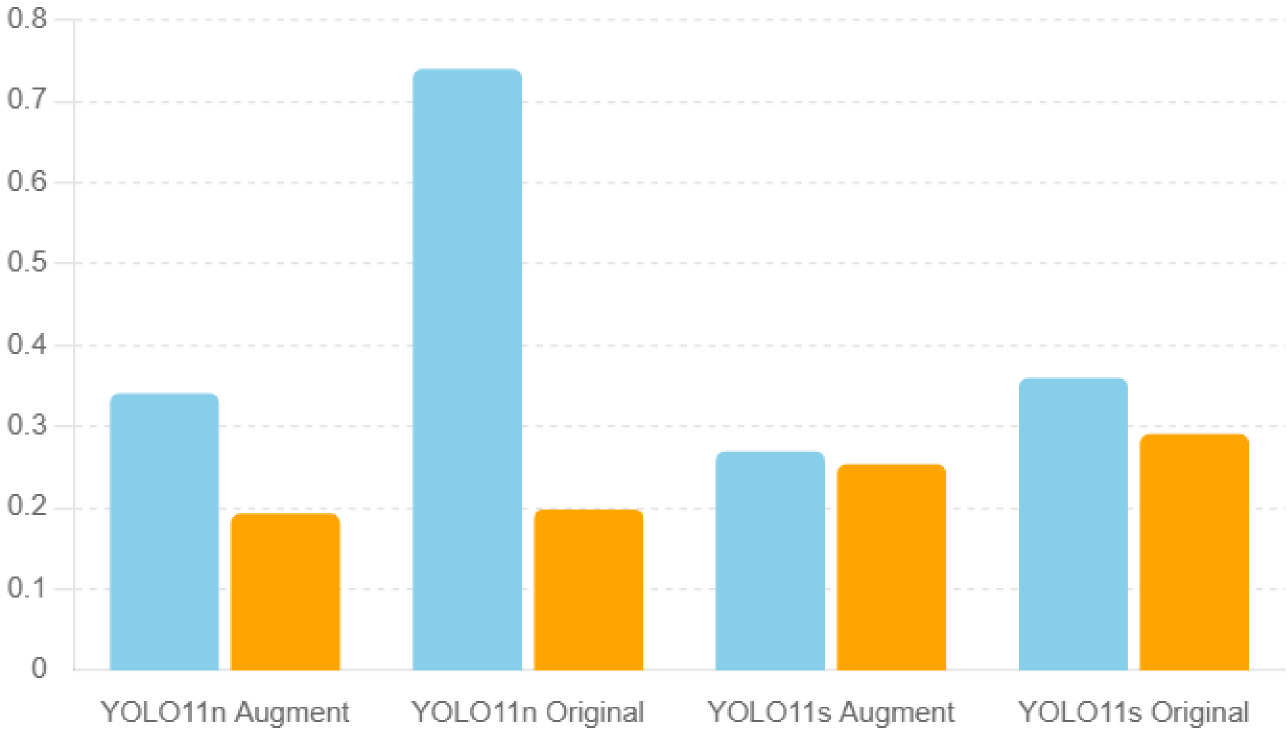


Figure 1: Precision and Recall Comparison

Figure 1 shows the precision and recall values for each model. The highest precision was achieved by YOLO11n Original, while YOLO11s Original achieved the highest recall, demonstrating different strengths across the models. YOLO11n Original was more conservative, achieving high precision, while YOLO11s Original captured more true positives with a higher recall.

Mean Average Precision (mAP) Scores

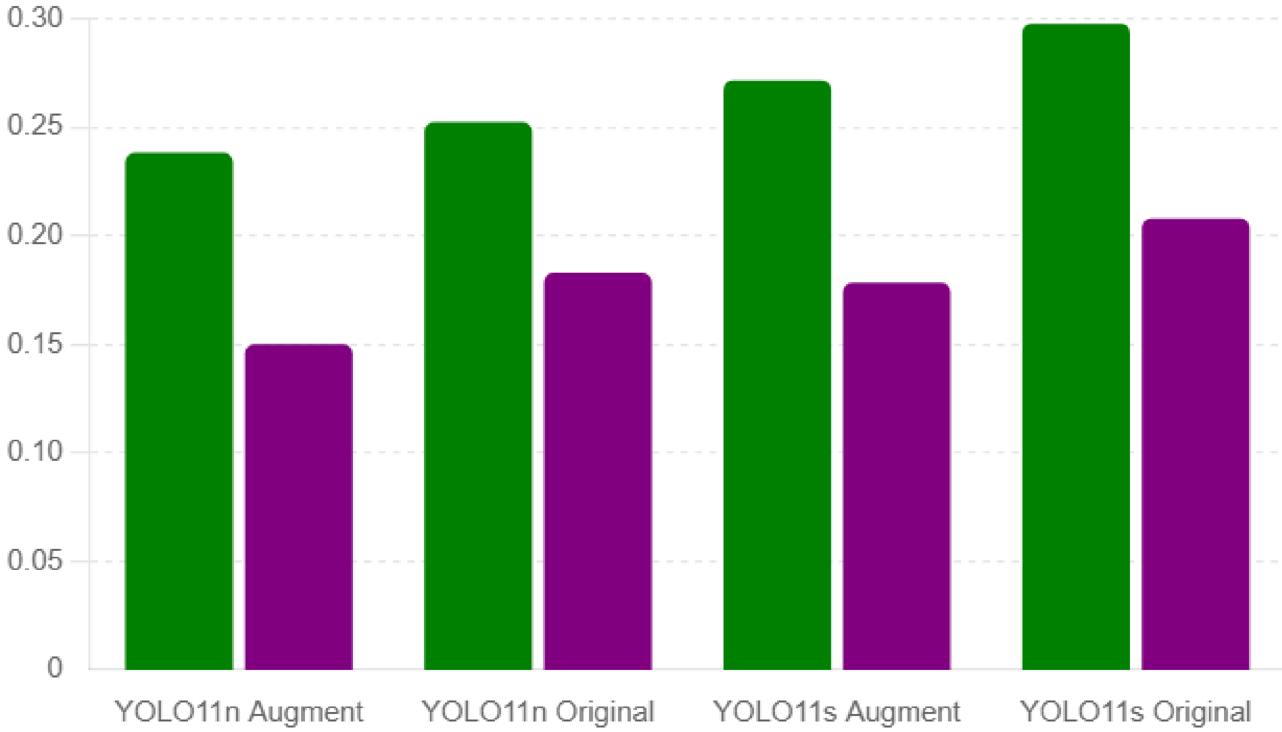


Figure 2: Mean Average Precision (mAP) Scores

Figure 2 compares the mean Average Precision scores at two levels: mAP50 and mAP50-95. YOLO11s Original achieved the highest scores in both mAP50 and mAP50-95, indicating superior detection accuracy and localization capabilities across multiple IoU thresholds. This model is particularly effective for applications requiring precise detection and identification.

Validation Loss Comparison

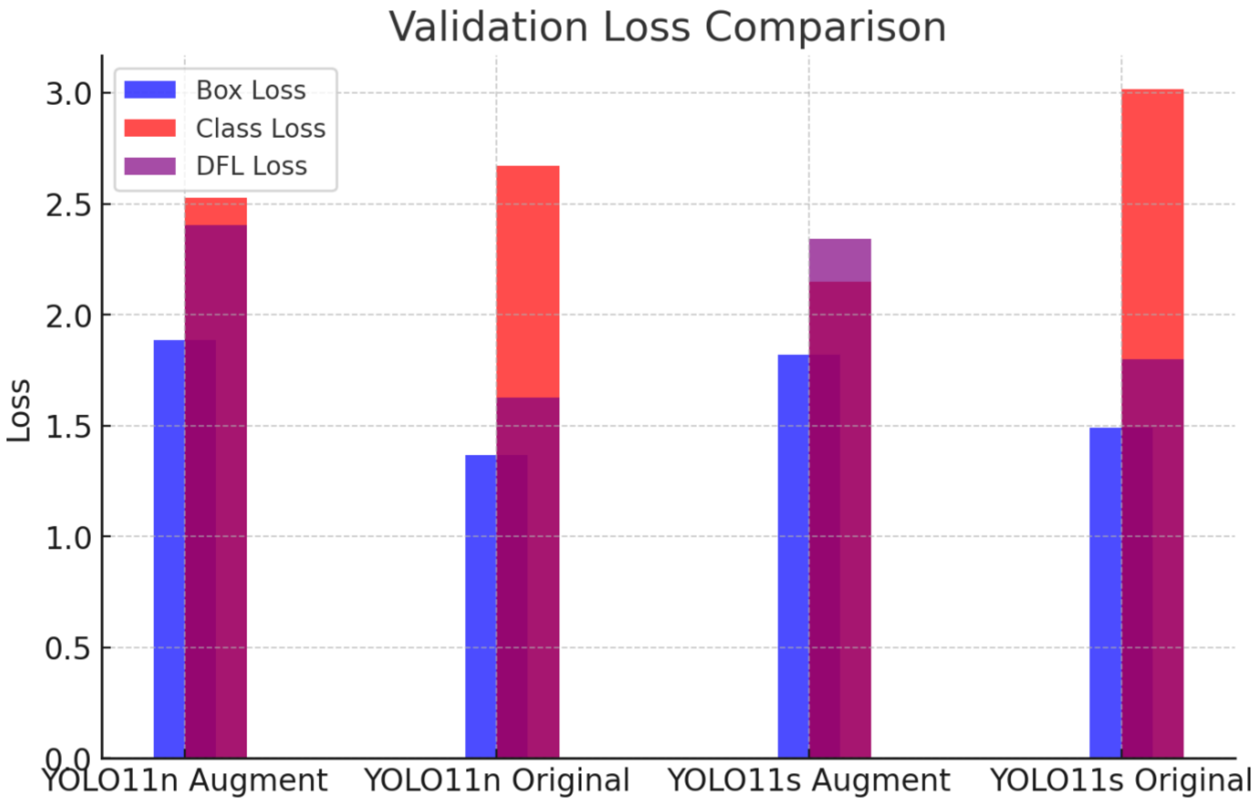


Figure 3: Validation Loss Comparison

The validation losses for box, class, and DFL are shown in *Figure 3*. YOLO11n Original demonstrated the lowest overall validation losses, suggesting robust generalization and minimal overfitting. This is critical for achieving consistent performance on unseen data, making YOLO11n Original a strong candidate for real-world deployment where generalization is vital.

Training Loss Comparison

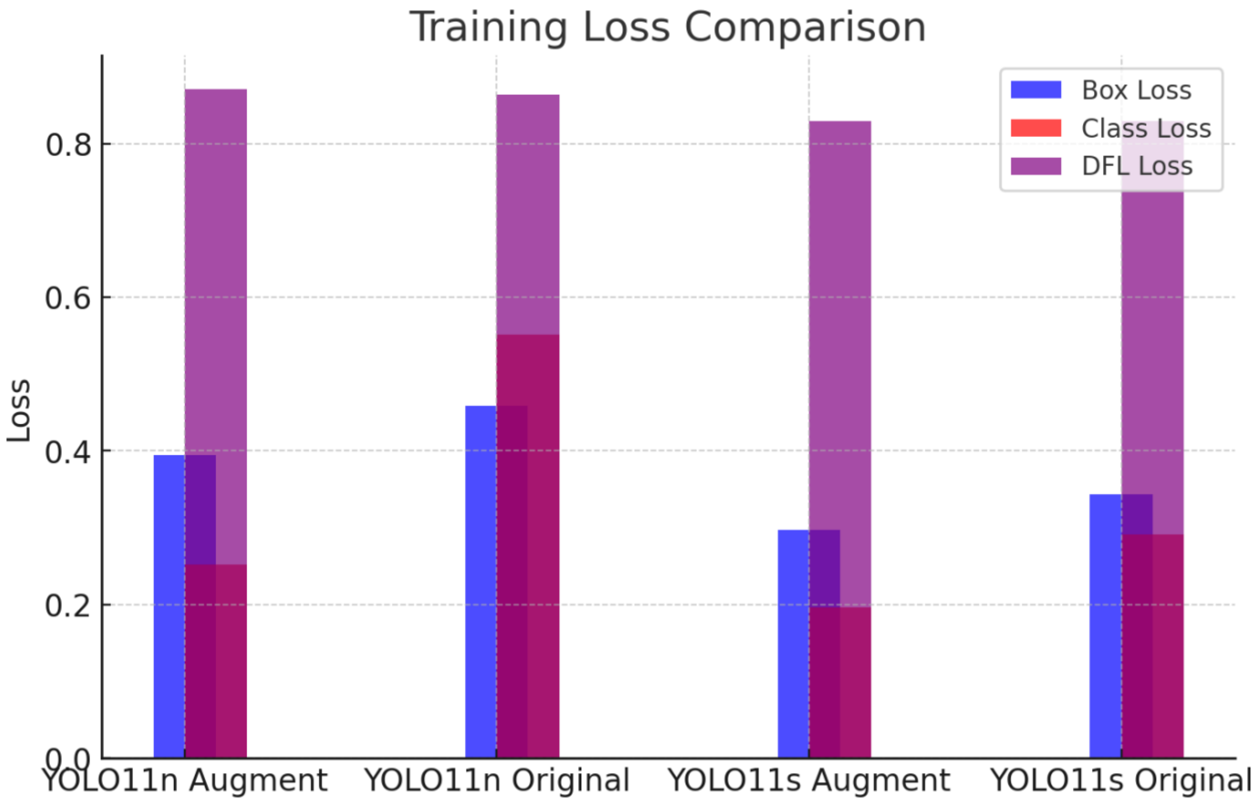


Figure 4: Training Loss Comparison

Figure 4 presents the training losses for each model in terms of box, class, and DFL losses. These losses were tracked across training epochs to evaluate convergence. Each model's training losses indicate the effectiveness of the training process, with YOLO11s Augment demonstrating the lowest training losses overall.

Epoch Wise Performance Analysis

To further analyze model performance, the following plots illustrate the progression of precision, recall, mAP50, and mAP50-95 scores over the training epochs for each model:

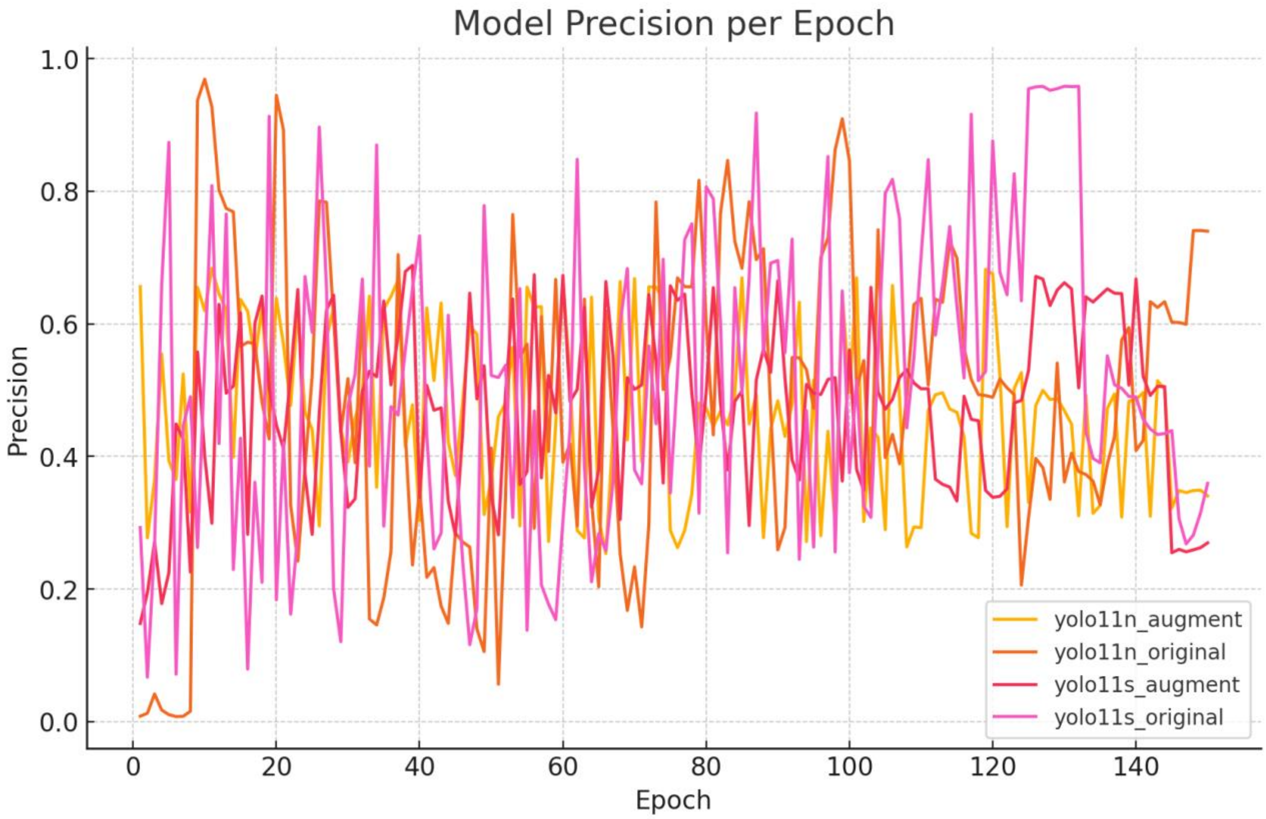


Figure 5: Model Precision per Epoch

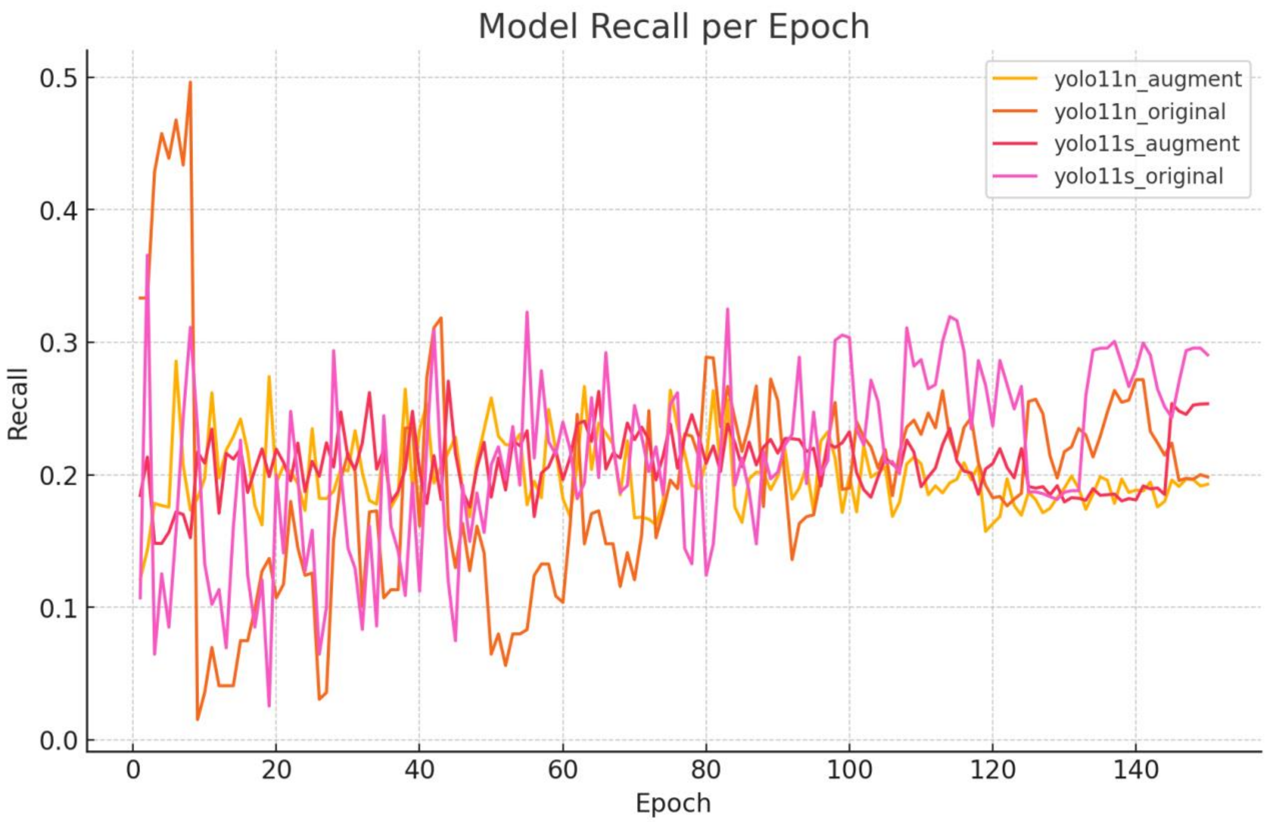


Figure 6: Model Recall per Epoch

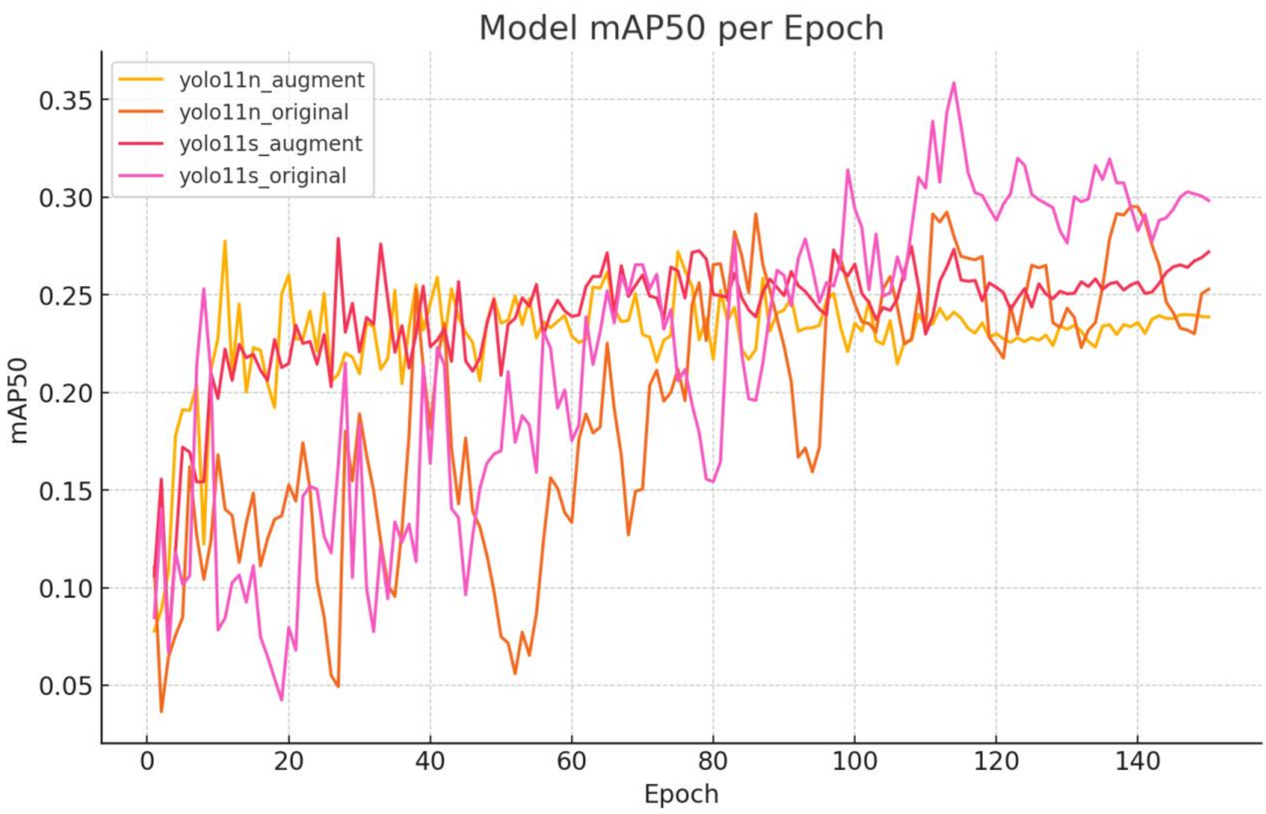


Figure 7: Model MAP50 per Epoch

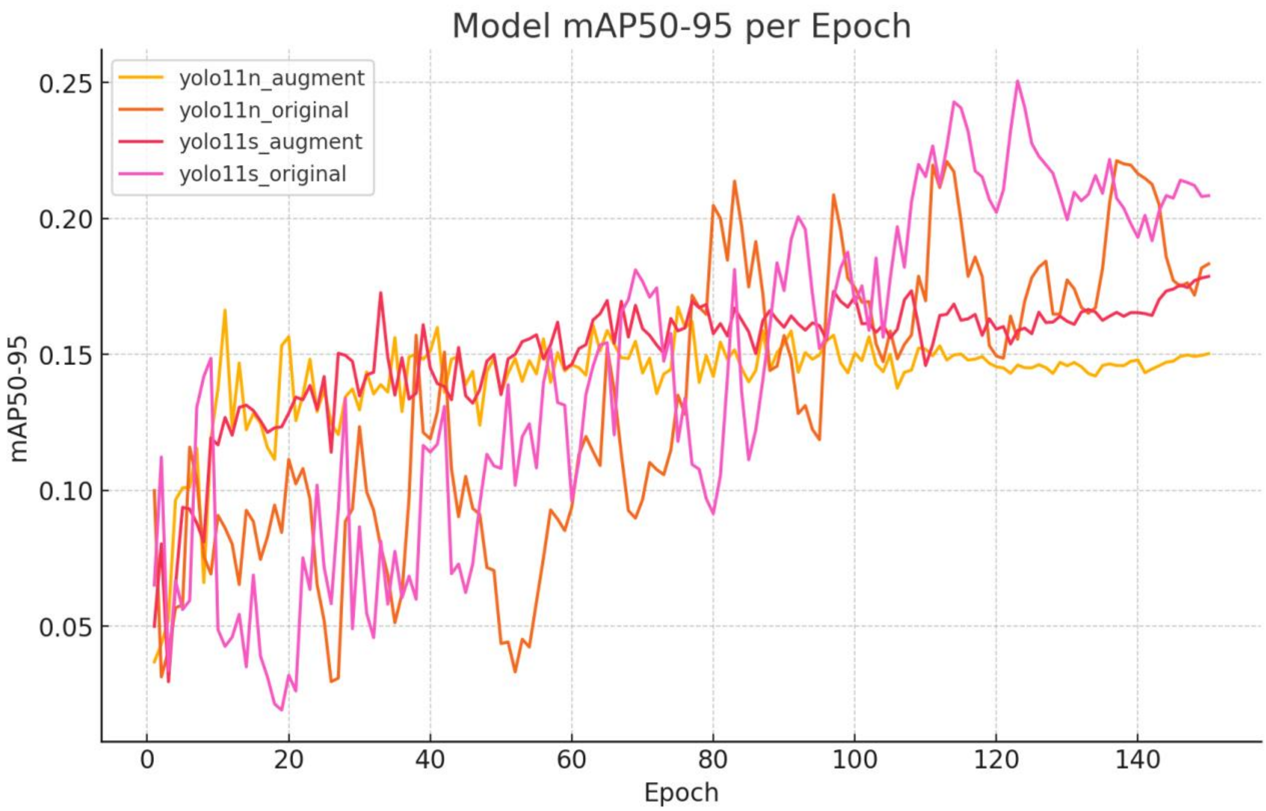


Figure 8: Model MAP50-95 per Epoch

- **Precision per Epoch** (*Figure 5*): Tracks how each model's precision evolves, with YOLO11n Original stabilizing at a higher precision rate.
- **Recall per Epoch** (*Figure 6*): Indicates how effectively each model captures true positives, with YOLO11s Original maintaining higher recall throughout training.
- **mAP50 per Epoch** (*Figure 7*): Reflects detection accuracy at an IoU threshold of 0.5, showing YOLO11s Original achieving the highest mAP50 scores.
- **mAP50-95 per Epoch** (*Figure 8*): Provides an aggregated view of detection performance across varying IoU thresholds, highlighting YOLO11s Original's consistent accuracy across all IoU levels.

These epoch-wise analyses are valuable for understanding each model's convergence behavior and stability during training, aiding in model selection based on specific application needs.

A.I Demonstrator

The AI demonstrator showcases an end-to-end detection system built for processing and analyzing images and videos using YOLO models. This section explains the demonstrator's architecture, development approach, input requirements, and expected outputs, supported by both the backend and frontend components.

Overview of A.I Demonstrator

The demonstrator provides a user-friendly interface, allowing users to:

- Upload media (images or videos)
- Choose a detection model
- Initiate the detection process
- View real-time detection results and interact with an AI assistant for detailed insights.

The architecture combines a Flask backend for processing and a React frontend for user interaction, which together support smooth media handling, model selection, and AI-driven feedback.

Development and Structure

Backend

The backend, implemented in Flask with SocketIO, handles file uploads, model selection, and detection processing. Key components are:

1. **Controller Layer** (**detection.py)**
   Manages API endpoints for:
   - Listing models (**/list_models**)
   - Setting the detection model (**/set_model**)
   - Uploading files (**/upload**)
   - Initiating the detection process (**/start_process**).
2. A lock mechanism ensures thread-safe operations, and resources are efficiently managed with caching and garbage collection.
3. **Model Management** (**manager.py)**
   Dynamically loads YOLO models available in the designated directory, allowing flexible model management and updates. The manager emits training results through WebSocket events, supporting real time updates on model performance.
4. **Detection Service (detection.py)**
   Handles object detection by processing frames and emitting results as annotated images and JSON objects, which are then sent to the frontend for real time display.

Frontend

The React frontend offers a cohesive interface with intuitive components for model selection, file uploads, and viewing results:

1. **Model Selector (ModelSelector.tsx)**
   Displays available models and sends user-selected models to the backend for configuration. The WebSocket is used to request and receive training results from the backend, which are then rendered on the frontend for analysis.
2. **File Upload (FileUpload.tsx)**
   Supports image and video uploads through drag-and-drop or selection. Users can extract specific frames from video files, converting them into images for detection.
3. **Start Process (StartProcess.tsx)**
   Allows users to start the detection process after model selection and file upload. It sends a request

to the backend, triggering the analysis and displaying a loading indicator until the process is complete.

4. **Image Results Display (ImageResults.tsx)**
   Provides real-time display of original and detected frames, with an option to engage the AI assistant. This component connects to the backend via WebSocket to receive and display frame-by-frame detection results. Users can attach images for detailed AI analysis.
5. **Chat for Training and Detection Analysis (GroqTrainingResultsChat.tsx** and **GroqDetectionResultsChat.tsx)**
   Allows users to interact with an AI assistant for insights into detection results. Each chat component supports full screen mode and provides step-by-step explanations, detailed object descriptions, and accuracy assessments based on model performance.

Input Requirements

To perform a detection analysis, users need to:

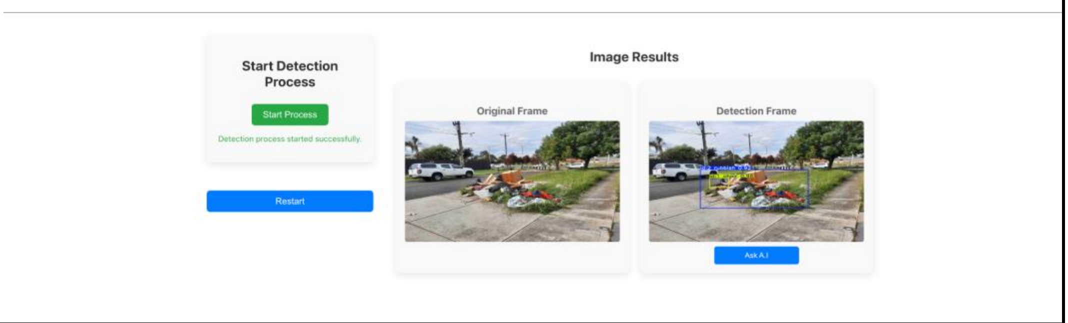
- **Upload an Image or Video**: Uploads are handled by **FileUpload.tsx**, which also allows users to select and extract frames from videos if needed.
- **Select a Model**: **ModelSelector.tsx** provides a list of YOLO models for selection. The chosen model is set up on the backend and optimized for the detection task.
- **Start the Detection Process**: Once a model and file are ready, users initiate the detection process via the Start Process button. The backend then analyzes the uploaded media, emitting results to the frontend in real-time.

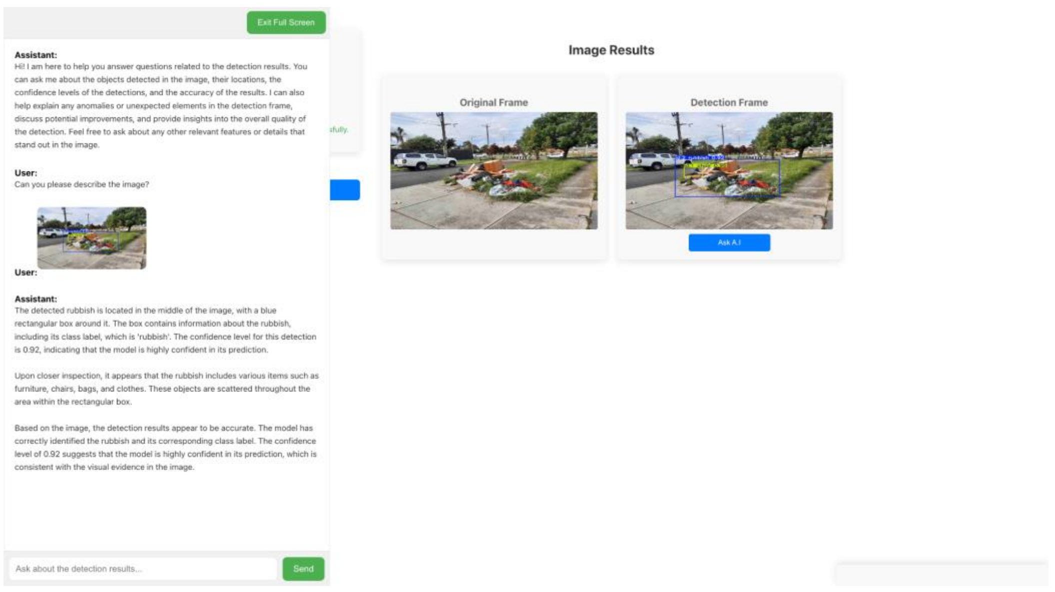
Expected Outputs

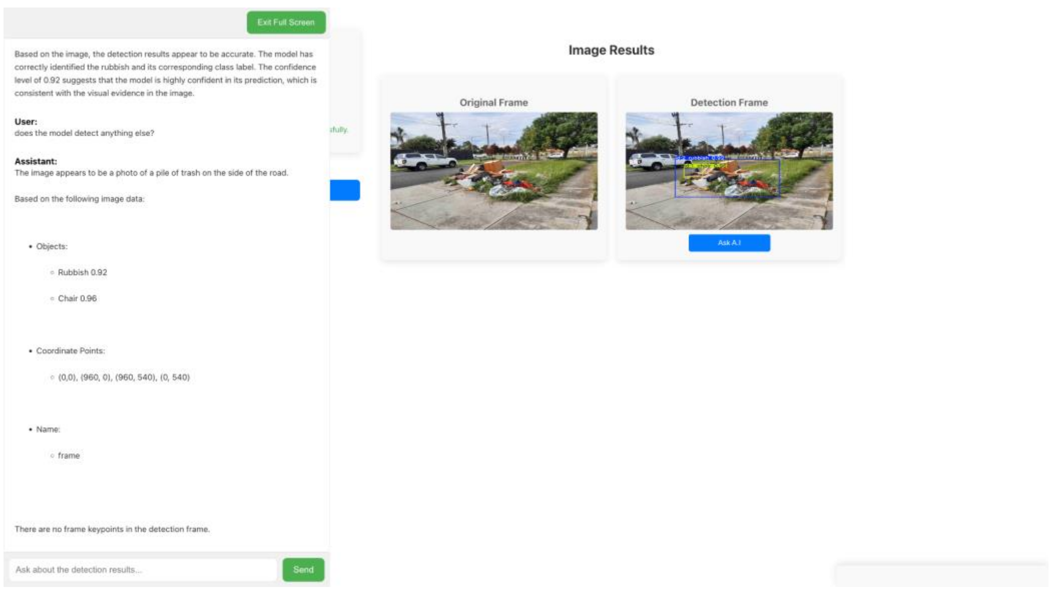
Upon completing the detection process, the demonstrator produces the following outputs:

1. **Real Time Detection Frames**
   The processed frames are displayed on the frontend in real-time. The **ImageResults.tsx** component shows both the original and annotated frames, offering users a clear visual of detected objects.
2. **Detailed AI Analysis**
   Users can engage with the AI assistant for deeper analysis. The chat system (handled by **GroqDetectionResultsChat.tsx** and **GroqTrainingResultsChat.tsx**) interprets detected objects and provides detailed insights. The assistant can respond to queries about specific objects, detection accuracy, and the overall context of the image.
3. **Training Results**
   For selected models, training results such as accuracy metrics, loss curves, and confusion matrices are displayed. The **TrainingResults.tsx** component groups these results into easy to navigate sections, allowing users to visualize training data and assess model performance.

Example Outputs

<table>
<tr>
<td>Yolo Model Detection</td>
<td>


</td>
</tr>
<tr>
<td>AI Analysis on Detection Frame</td>
<td>


</td>
</tr>
<tr>
<td>AI Analysis on Detection Frame</td>
<td>


</td>
</tr>
</table>

Workflow Summary

1. **Upload Media**: Users upload an image or video for analysis.
2. **Model Selection and Training Review**: Users select a model and review its training results to assess suitability for their detection needs.
3. **Start Detection**: Initiates frame-by-frame processing, streaming results to the frontend.
4. **Interact with AI**: Users engage with the AI assistant to receive detailed explanations and insights.
5. **Review Training Results**: Training metrics, confusion matrices, and other performance indicators are available, helping users make informed choices about model performance.

Conclusion

This project successfully developed an AI-based system to help identify and classify dumped rubbish on roadsides, aiming to support city councils in keeping urban spaces cleaner and safer. Using deep learning with YOLO models, we tackled a real-world problem by creating a model that can detect rubbish in images, even under varied lighting and environmental conditions.

The project involved several main steps: preparing and annotating a dataset, training the model on both original and augmented data, evaluating its performance, and deploying it through a user friendly AI demonstrator. Augmentation made the model more adaptable to different real world conditions, and testing multiple YOLO configurations (YOLO11n and YOLO11s) provided insights into model strengths. YOLO11n Original excelled in precision, while YOLO11s Original showed the best balance of accuracy across a range of conditions, making it suitable for detecting different types of rubbish in urban settings.

The AI demonstrator brought the model to life, allowing users to upload images or videos, choose detection models, and get real time feedback on detected objects. Through an interactive frontend, users can also view training results and engage with an AI assistant to better understand the detection results. Together, the backend and frontend create a complete detection system that's accessible and easy to use.

Overall, this project highlights how AI can help address everyday urban issues. By identifying rubbish on roadsides automatically, city councils and maintenance teams could save time and improve response times, ultimately leading to cleaner streets. This project demonstrates the potential of AI in supporting cleaner and safer cities, contributing positively to Smart City initiatives.

References

- Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). "You Only Look Once: Unified, Real-Time Object Detection." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Available at: https://arxiv.org/abs/1506.02640
- Ultralytics (2024) 'Ultralytics YOLO11', *GitHub Repository*. Available at: https://github.com/ultralytics/ultralytics (Accessed: 1 November 2024).
- Kentaro, W. (2024) 'LabelMe: Image Polygonal Annotation with Python', *GitHub Repository*. Available at: https://github.com/wkentaro/labelme (Accessed: 1 November 2024).
- Swinburne University of Technology (2024) 'COS40007 Design Project', *Unit Outline*.

Appendix A

- **Backend Code**: The server-side implementation, including API endpoints and model management, is available at: https://github.com/kingsradio/COS40007-Artificial-Intelligence-for-Engineering/tree/main/group_assignment/code/backend
- **Frontend Code**: The client side interface, facilitating user interactions and displaying detection results, can be accessed at: https://github.com/kingsradio/COS40007-Artificial-Intelligence-for-Engineering/tree/main/group_assignment/code/frontend

- **Dataset and Training Code**: Scripts for data preprocessing, augmentation, and model training are provided here: https://github.com/kingsradio/COS40007-Artificial-Intelligence-for-Engineering/tree/main/group_assignment/code/training
- **Presentation Slides**: The project's presentation materials are available at: https://github.com/kingsradio/COS40007-Artificial-Intelligence-for-Engineering/tree/main/group_assignment/presentation
- **Project Brief**: Detailed project objectives and scope are outlined in the brief: https://github.com/kingsradio/COS40007-Artificial-Intelligence-for-Engineering/tree/main/group_assignment/project_brief
- **Full Dataset**: The annotated dataset, including images and LabelMe annotations, is accessible at: https://github.com/kingsradio/COS40007-Artificial-Intelligence-for-Engineering/tree/main/group_assignment/datasets/data/data/rubbish
- **Training Results**: Outputs from model training sessions, including weights and performance metrics, are documented here: https://github.com/kingsradio/COS40007-Artificial-Intelligence-for-Engineering/tree/main/group_assignment/runs/train
- **Model Weights**: Within the training results directory, each model folder contains a **'weights'** subfolder with '**best.pt**' and '**last.pt**' files, representing the optimal and final model states, respectively.