Arrow Detection using YOLO Model

Project Overview

This project focuses on the development and deployment of a custom object detection model based on YOLO (You Only Look Once), specifically fine-tuned to detect directional arrows (up, down, left, right) in images and video streams. The model is trained on a custom dataset of arrows and similar symbols and is capable of processing both static images and real-time camera feeds. This documentation provides a comprehensive guide to understanding the model, its training process, evaluation metrics, and usage.

Model Description

YOLO Architecture

YOLO (You Only Look Once) is a state-of-the-art, real-time object detection system. Unlike traditional methods that apply the model to an image at multiple locations and scales, YOLO frames object detection as a single regression problem, directly predicting bounding boxes and class probabilities from the full image in one evaluation.

Arrow Detection Model

In this project, a variant of the YOLO model has been fine-tuned to detect four valid classes of directional arrows:

- **Up Arrow**
- **Down Arrow**
- **Left Arrow**
- **Right Arrow**

These classes are defined in a custom dataset specifically curated for this task. The model has been trained to recognize these arrows in various orientations and contexts, ensuring robust performance across different environments.

Pretrained Models

- **YOLOv9t.pt**: This is the initially intended model, known for its high performance due to the extensive layer depth and parameters. However, due to limitations in computational resources, this model was challenging to train effectively.
- **YOLOc9e.pt**: Due to hardware limitations, a smaller model, YOLOc9e.pt, was used for the final training and deployment. Despite being smaller, this model provided satisfactory results, though the performance could potentially be improved with more powerful hardware using YOLOv9t.pt.

Dataset

Dataset Structure

The dataset comprises images annotated with bounding boxes around directional arrows. These images are categorized into training, validation, and test sets to ensure the model can generalize well to unseen data.

- **Training Set**: Used for training the model, includes a diverse set of arrow images with various orientations and backgrounds.
- **Validation Set**: Used during training to tune hyperparameters and prevent overfitting.
- **Test Set**: Used post-training to evaluate the model's performance.

  Dataset url: https://universe.roboflow.com/mdpindiv-m1xmm/datasetv1-ctr75

Model Training

Training Process

The training process involved the following steps:

1. **Model Initialization**: The YOLO model was initialized with pre-trained weights from a large-scale object detection task (YOLOv9t.pt), providing a solid starting point for fine-tuning on the arrow detection dataset.
2. **Training**: The model was trained over 10 epochs, with each epoch consisting of multiple iterations over the training dataset.
3. **Checkpointing**: The best-performing model weights were saved after each epoch, allowing for the selection of the optimal model at the end of training.

Training Metrics

The model's performance was tracked using several metrics during training:

- **Loss Functions**: The primary metrics used were box_loss, cls_loss, and dfl_loss, which measure the accuracy of the bounding boxes, classification, and distributional focal loss, respectively.
- **Precision and Recall**: Precision and recall were monitored to understand the balance between false positives and false negatives.
- **mAP@0.5 and mAP@0.5:0.95**: These metrics were used to evaluate the mean Average Precision at IoU thresholds of 0.5 and across a range of 0.5 to 0.95.

Training Logs

The following key metrics were observed during the 10 epochs of training:

- **Precision**: Increased from 0.352 to 0.987
- **Recall**: Increased from 0.418 to 0.993
- **mAP@0.5**: Improved significantly, reaching up to 0.994 by the final epoch
- **mAP@0.5:0.95**: Reached a value of 0.815, indicating a high level of accuracy across varying IoU thresholds.

Detailed training logs indicated that the model improved significantly over the course of training, despite the challenges posed by the hardware limitations.

Model Evaluation

Evaluation Metrics

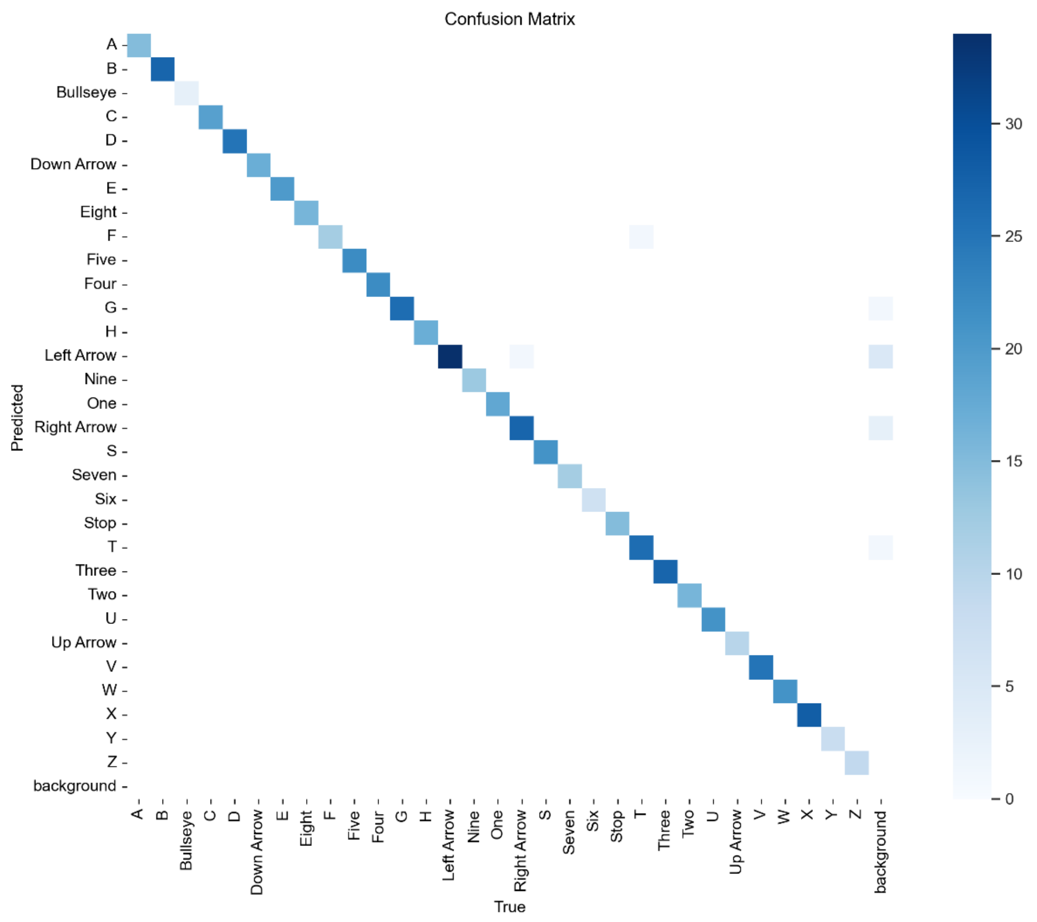
The model was evaluated on the validation set using several important metrics:

- **Precision**: 0.987
- **Recall**: 0.993
- **mAP@0.5**: 0.994
- **mAP@0.5:0.95**: 0.815

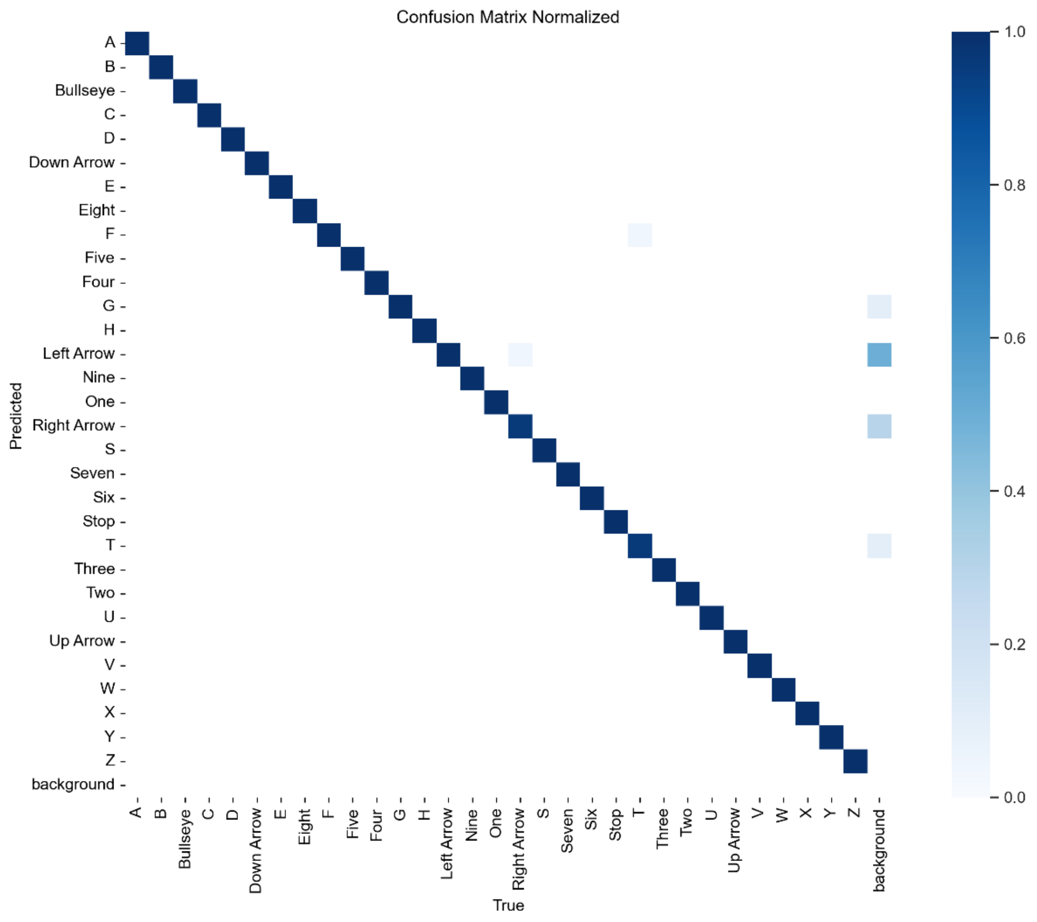
These metrics indicate that the model is highly accurate in detecting arrows, with excellent precision and recall values. The mAP metrics show that the model performs well across a range of intersection-over-union (IoU) thresholds.

Confusion Matrix

A confusion matrix was generated to visualize the performance of the model across different classes. The matrix confirmed that the model could reliably distinguish between different types of arrows with minimal confusion between classes.



- **Normalized Confusion Matrix**: The normalized version of the confusion matrix further highlights the model's ability to accurately classify different arrow types.



Model Testing

Testing on Images

The model was tested on both individual images and a batch of images to evaluate its performance in practical scenarios.

- **Single Image Testing**: The model was able to accurately detect arrows in images with complex backgrounds.
- **Multiple Images Testing**: The model showed consistent performance across a batch of test images, with high accuracy in detecting the correct arrow directions.

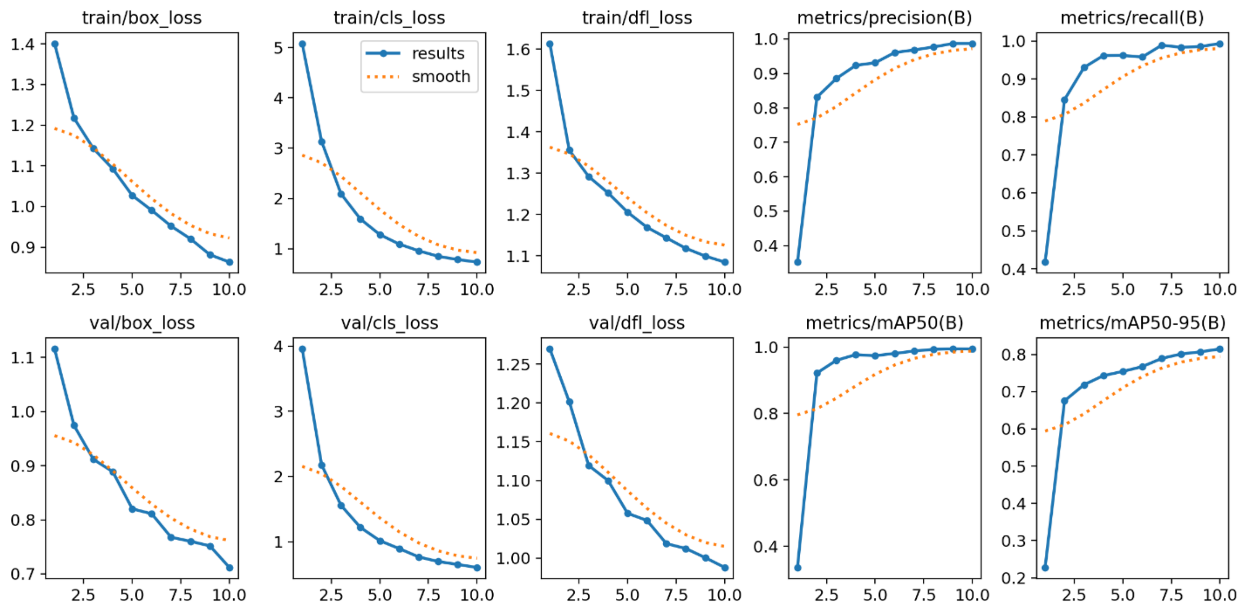
Real-Time Detection

The model was also tested on real-time video feeds to detect arrows as they appear in a live camera stream. Despite the computational limitations, the model performed adequately, with slight delays in inference due to the use of a CPU.

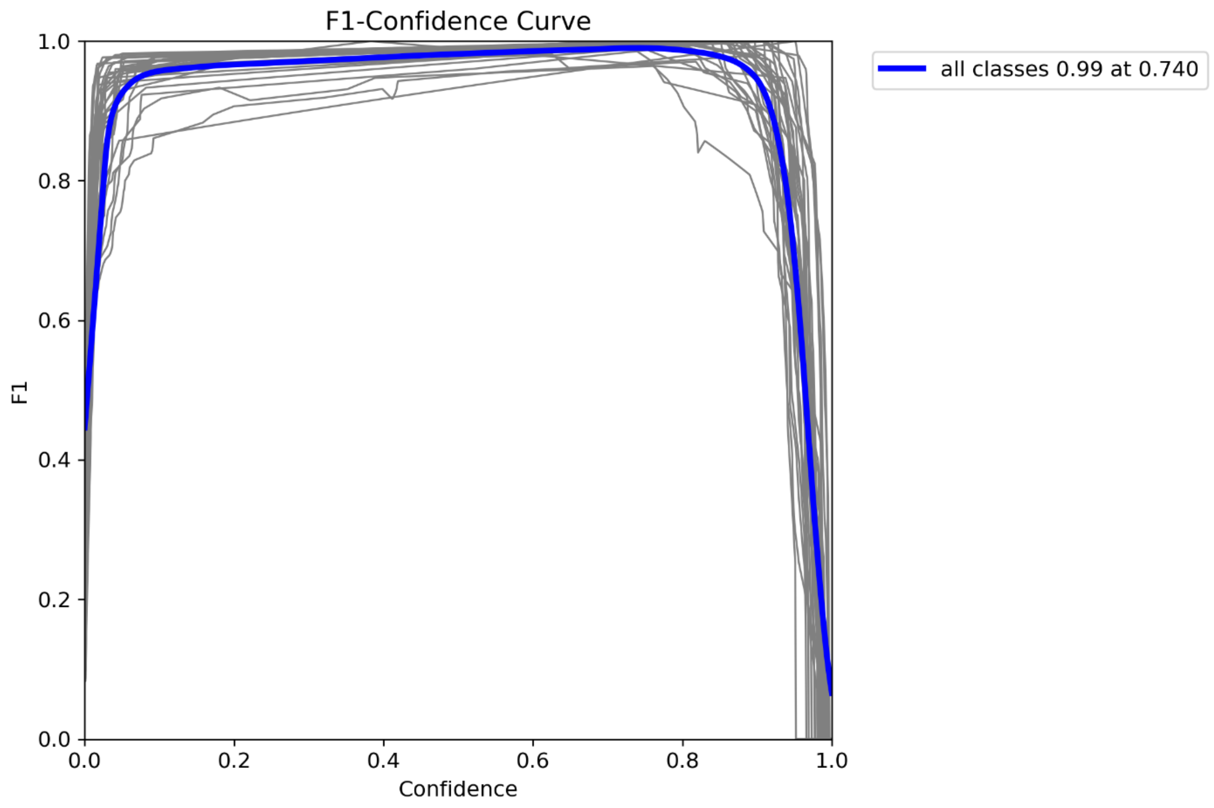
Visualizations

Several visualizations were generated to provide insights into the model's performance:

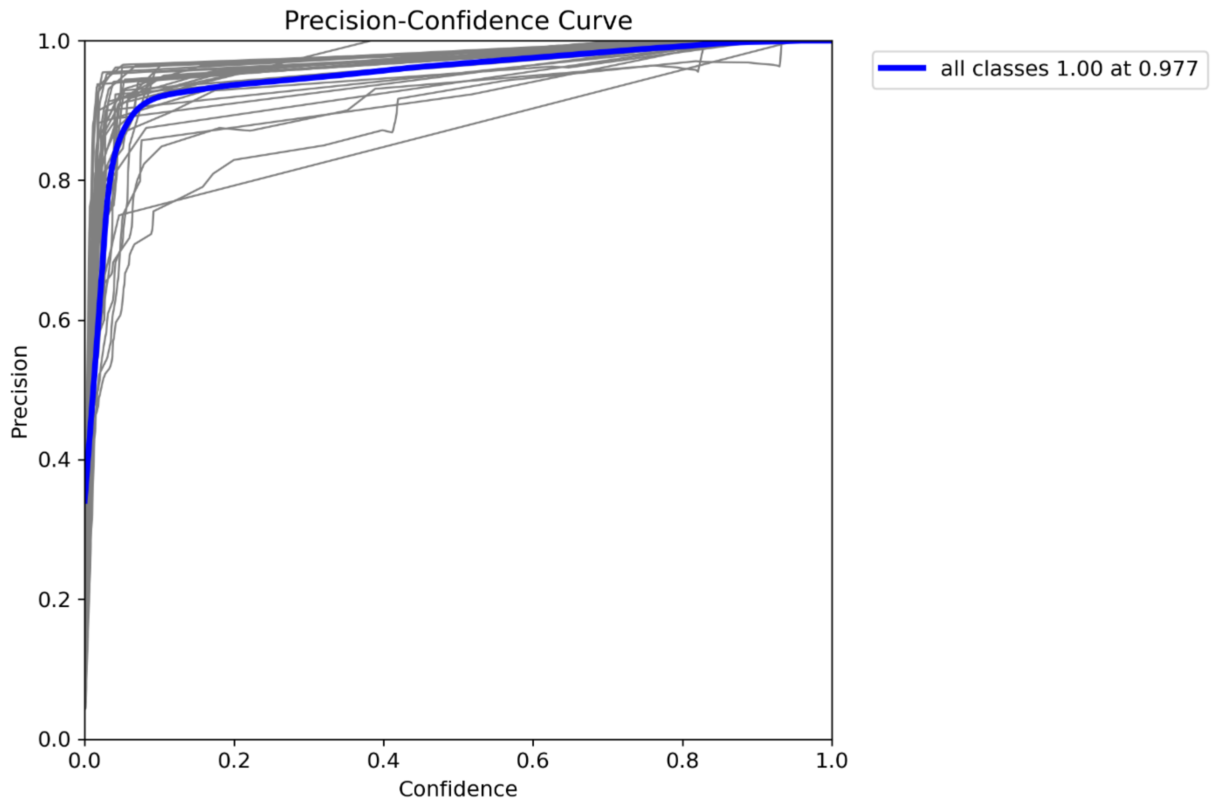
1. **Training Loss Curves**: These curves show the reduction in loss values over the epochs, indicating successful training.



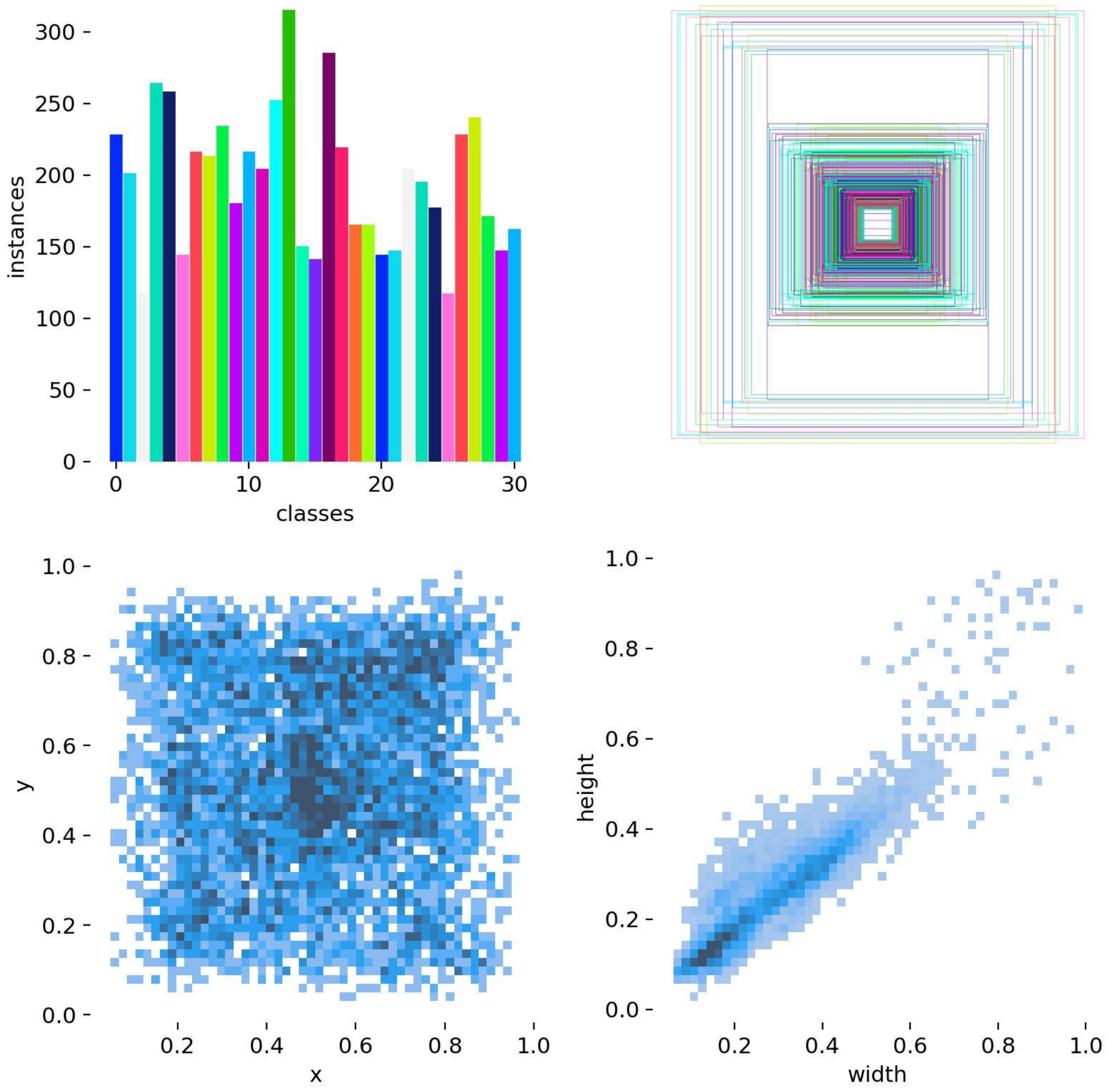
2. **F1-Confidence Curve**: This curve demonstrates the relationship between the confidence threshold and the F1-score, highlighting the optimal balance between precision and recall.



3. **Precision-Confidence Curve**: This curve visualizes the precision of the model at different confidence levels, confirming that the model maintains high precision even at lower confidence thresholds.



4. **Label Distribution and Correlogram**: These plots provide insights into the distribution of labels and the correlation between different bounding box attributes (such as width and height).



Conclusion

This project successfully demonstrates the use of a YOLO model for detecting directional arrows in images and video streams. Despite the constraints posed by limited computational resources, the model achieved high accuracy and performed well in both static and dynamic scenarios. The use of a smaller model, YOLOc9e.pt, allowed for practical deployment while still delivering robust results.

Possible Improvements

Future improvements could include:

- **Using More Powerful Hardware**: Training the larger YOLOv9t.pt model on more powerful hardware could potentially yield even better results.
- **Data Augmentation**: Incorporating more advanced data augmentation techniques could help the model generalize better to different environments.
- **Real-Time Optimization**: Further optimization for real-time performance, particularly on edge devices, could make this model more practical for applications like autonomous vehicles or robotics.