Traffic Sign Detection Report

1. Introduction

1.1 Problem statement:

1.1.1. Machine Learning Problem:

This project aims at detecting and classifying traffic signs in a video or a camera feed using different variants of the YOLO (You Only Look Once) machine learning model. The goal is to create a system that can recognize various traffic signs—such as stop signs, speed limits etc. under diverse real-life conditions, including changes in lighting, weather, and camera angles.

1.1.2. Importance:

Traffic sign recognition plays an important role in road safety and transportation sector. By addressing this issue:

- Both drivers - human and automated, can make swift and informed decisions, which reduces the occurrence of accidents.
- Autonomous vehicle systems can earn capability of safe navigation, adhering to traffic rules with precision.
- Urban traffic flow can be optimized, leading to fewer delays and safer roads for everyone.

1.2. Business value:

1.2.1. Impact on industry:

The ability to accurately detect traffic signs has various implications:

- For the automotive industry: Traffic sign detection enhances autonomous vehicles and driver-assistance systems, providing a safer and more reliable product.
- For society: Safer roads lead to fewer accidents which can eventually lead to lives saved. Additionally, traffic management becomes more efficient, leading to reduced congestion and potentially lower emissions.

- For researchers and developers: This project sheds light on the computational demands of training YOLO models without GPUs, offering benchmarks that guide informed decisions on resource allocation and hardware requirements, making advanced detection systems more accessible.

1.2.2. Stakeholders and benefits to them:

Several groups stand to gain from advancements in traffic sign detection:

- Car manufacturers: Incorporating robust detection models improves the vehicle's safety features, reliability and appeal.
- Transportation agencies: Real-time sign detection supports adaptive traffic systems which leads to an improved and smoother traffic control.
- Insurance providers: Fewer accidents reduce the amount of filed claims which benefits both providers and policyholders.
- Drivers and pedestrians: Reduced human error while driving creates safer roads for everyone.

2. Methodology:

2.1. Literature review:

2.1.1. Summary:

Traffic sign detection remains to be an area of interest, especially with the rise of driverless or autonomous driving systems, and researchers have explored a variety of methodologies over the years. Broadly, these approaches can be divided into traditional detection techniques and modern deep learning-based solutions.

Traditional Traffic Sign Detection Techniques:

The early traffic sign detection systems, dating as far back to the 1970s, relied on manually extracted features, such as color and shape, to detect and classify signs. Methods like color segmentation and template matching were adopted in this domain. For example, Akatsuka[1] applied color segmentation and template matching to detect traffic signs, and Fleyeh [2] extended this idea by converting images to IHLS color space and using segmentation to extract features. However, these methods faced significant limitations in real-world scenarios due to environmental factors like lighting, weather, and occlusion.

Advanced methods like feature detection and multi-scale analysis, were later introduced, achieving moderate success. However, traditional methods still fell short in terms of accuracy and speed, especially in dynamic environments[3].

Deep Learning-Based Traffic Sign Detection:

The advent of deep learning, particularly Convolutional Neural Networks (CNNs), revolutionized traffic sign detection by bringing deeper architectures and better feature representation, significantly outperforming traditional methods. After 2013, CNNs started to become the standard for object detection tasks. Some examples of these CNNs include OverFeat [4], which uses a scalable sliding window and a greedy merge procedure to produce bounding boxes and scores.

Detection algorithms were further refined into two categories:

1. Two-Stage Detection Models: These include R-CNN, Fast R-CNN, and Faster R-CNN, which first identify regions of interest and then classify objects. While highly accurate, these models are computationally expensive, in both memory and time and processing a test image takes about 47 seconds[4]. Hence, making them unsuitable for real-time applications.
2. One-Stage Detection Models: YOLO (You Only Look Once)[6] emerged as a game-changer by framing detection as a single regression task. Early versions, such as YOLOv1 and YOLOv2, prioritized speed but struggled with small object detection. YOLOv3 and YOLOv4 addressed these challenges by improving accuracy and handling smaller targets better, making them ideal for real-world use cases.

Recent advancements in YOLO models, particularly YOLOv8[5][6], have focused on incorporating cutting-edge techniques to further enhance detection capabilities by addressing interactions between non-adjacent layers making it particularly suited for real-time applications in dynamic environments.

2.1.2. Gaps to address:

The gaps that this project aims to address include:
Computational resourcing:

- This project provides insights into the computational challenges of training YOLO models without GPU acceleration, a scenario often overlooked in existing literature.
- By documenting the training time on a CPU-only system, the project offers practical benchmarks for researchers and developers with limited access to high-performance hardware.
- Readers can gain a clear understanding of the time and resource requirements for CPU-based YOLO model training, helping them make informed decisions about hardware investments or alternative approaches.

Scalability:

- Many models require extensive computational resources, making them impractical for edge devices like car cameras.

2.2. Model selection:

2.1.1. Choice of selected model:

The YOLO models were selected for this project as they are well known for their performance in object detection and classification tasks. Unlike traditional models that sequentially perform detection and classification in separate steps, YOLO integrates these steps, enabling faster and more accurate detection. Its architecture has been optimized for both speed and precision, making it ideal for real-time applications like traffic sign recognition.

We experimented with three different YOLO models and implemented the most compact iterations of each model. Below are the key features of each model:

YOLOv8n:

- Advanced Backbone and Neck Architectures: YOLOv8 employs state-of-the-art backbone and neck architectures, resulting in improved feature extraction and object detection performance.
- Anchor-free Split Ultralytics Head: YOLOv8 adopts an anchor-free split Ultralytics head, which contributes to better accuracy and a more efficient detection process compared to anchor-based approaches.
- Optimized Accuracy-Speed Tradeoff: With a focus on maintaining an optimal balance between accuracy and speed, YOLOv8 is suitable for real-time object detection tasks in diverse application areas.
- Variety of Pre-trained Models: YOLOv8 offers a range of pre-trained models to cater to various tasks and performance requirements, making it easier to find the right model for your specific use case.

YOLOv9t:

- Information Bottleneck Principle: It reveals a fundamental challenge in deep learning: as data passes through successive layers of a network, the potential for information loss increases.
- Reversible Functions: The concept of Reversible Functions is another cornerstone of YOLOv9's design. A function is deemed reversible if it can be inverted without any loss of information

- Programmable Gradient Information (PGI): PGI is a novel concept introduced in YOLOv9 to combat the information bottleneck problem, ensuring the preservation of essential data across deep network layers.
- Generalized Efficient Layer Aggregation Network (GELAN): Its design allows for flexible integration of various computational blocks, making YOLOv9 adaptable to a wide range of applications without sacrificing speed or accuracy.

YOLOv10n:

- NMS-Free Training: Utilizes consistent dual assignments to eliminate the need for NMS, reducing inference latency.
- Holistic Model Design: Comprehensive optimization of various components from both efficiency and accuracy perspectives, including lightweight classification heads, spatial-channel decoupled down sampling, and rank-guided block design.
- Enhanced Model Capabilities: Incorporates large-kernel convolutions and partial self-attention modules to improve performance without significant computational cost.

Each of these models were trained for 25 epochs and the results are as below:

Model	**TrainTime**	**Precision**	**Recall**	**mAP50**
YOLOv8n	12.748 hrs	0.94	0.92	0.96
YOLOv9t	17.598 hrs	0.94	0.89	0.95
YOLOv10n	20.762 hrs	0.92	0.86	0.94

Comparing the results, YOLOv8n proves to be the optimal choice for the project with the lowest training time and the highest recall of 0.92, combined with a high precision of 0.94. It also showcases an excellent mAP50 score which reflects the fact that the model is positioning the bounding boxes around the detected traffic signs closer to the ground truth bounding boxes.

2.2.2. Expected advantages:

The approach of experimenting with different YOLO models and selecting the optimal one for the task offers several advantages, in the context of training observations and results, and real-time traffic sign detection and classification:

- Comprehensive Exploration of State-of-the-Art Models: By evaluating YOLOv8n, YOLOv9t, and YOLOv10n, the approach ensures that various advanced object detection techniques and architectures are considered.

- Real-Time Applicability: YOLO models are renowned for their real-time detection capabilities, crucial for traffic sign detection applications like autonomous driving or smart traffic management. The integration of detection and classification in a single pipeline eliminates the inefficiencies of traditional step-by-step methods, making YOLO an ideal choice.
- Selection Based on Practical Metrics: Metrics like training time, precision, recall, and mAP50 were used to evaluate the models:
  - Precision ensures minimal false positives, critical for avoiding incorrect traffic sign classifications.
  - Recall ensures minimal false negatives, crucial for capturing all traffic signs in real-time.
  - mAP50 reflects the quality of bounding box placements, ensuring accurate detection of sign boundaries.
- An overview for prospective readers: This methodology can give prospective readers such as students an overview about the training time consumed by each model based on the system being used.

2.3. Data Preparation:

2.3.1 Data collection, augmentation, and preprocessing steps:

The dataset is sourced from Kaggle and contains images of different traffic signs with various lighting conditions, backgrounds, distance and angles. Each image has its label along with the coordinates of bounding boxes for traffic signs.

Train set- 3531 images (before augmentation)

Test set- 639 images

Validation set- 802 images

<u>Augmentation and Preprocessing Steps:</u>

To enhance the dataset and improve model robustness, the following augmentation techniques were applied using the "albumentations" library.

- Brightness and Contrast Adjustments: Simulated varying lighting conditions.
- Scaling, Rotation, and Cropping: Mitigates perspective distortion and camera movement.
- Hue-Saturation Alteration: Mimics color shifts under different weather conditions.
- Resizing images: Resizing images to 640x640 pixels to match the YOLOv8 model requirements.

This augmentation further doubled the amount of training data (7060 images).

2.3.2. Dataset split for training, validation, and testing:

The dataset was readily available divided into three folders:
Train set- 3531 images (before augmentation), 7060 images (after augmentation)
Test set- 639 images
Validation set- 802 images
This split ensured a balanced representation of traffic sign classes across all subsets.

2.4. Training and Fine-tuning :

2.4.1. Training process and fine tuning:

System Information:
Computer- MacBook M3 Pro (16 GB RAM)
GPU- Turned off

The GPU was not utilized during the training process, providing prospective readers with a realistic estimate of the time required for training when relying solely on CPU resources without GPU acceleration.

The YOLO models were fine-tuned using transfer learning. Originally trained on the COCO dataset, the model already came with pre-loaded weights. The models were then trained on the entire train set over 25 epochs. The batch size was set to -1, which allows the model to automatically set a bat h size based on the available system resources. The checkpoints were saved for the best-performing model based on validation metrics.

2.4.2. Hyperparameters and tuning strategy:

YOLO implements most of the hyperparameters automatically to streamline the training process. Some of the parameters are as follows:

- Learning Rate: Set to an initial value of 0.01, gradually decayed to stabilize training.
- Batch Size: Adjusted based on available memory, optimized at 16 for computational efficiency.
- Confidence Threshold: Tuned to balance precision and recall, targeting a value of 0.5.
- IoU Threshold: Set at 0.7 to improve detection overlap accuracy.
- Optimizer Selection: The “optimizer=auto” dynamically selects hyperparameters (lr0, momentum) and chose AdamW for training.

The tuning strategy also involved monitoring metrics like mAP-50 (mean Average Precision) and loss values during training to iteratively refine hyperparameters.

2.5. Integration with Downstream Pipeline:

2.5.1. Utilization of model's output in a downstream application:

Inferences can be run on a live feed from a vehicle camera using the trained model. The outputs generated can be used for the following purposes:

- Alert the driver in case of traffic rules infringement.
- Applied to an automated car to detect traffic signs and road speed limits to response accordingly.

The outputs can trigger actions such as reducing speed for speed limit signs or halting for stop signs, ensuring compliance with traffic laws.

2.5.2. Details of the expected workflow and how the solution adds value:

The end-to-end workflow includes:

1. Capturing real-time camera feeds or batch-processed images.
2. Running the trained YOLOv8 model to detect traffic signs.
3. Sending detections to decision-making modules or visualization dashboards.
4. Action taken by the authoritative entity (driver/self-driving car CPU).

This workflow adds value by enabling:

- Real-time Decision Support: Enhancing driver safety and vehicle autonomy.
- Data-Driven Traffic Insights: Providing valuable analytics for urban traffic management systems.
- Cost-Effective Implementation: Deploying lightweight models such as YOLOv8n suitable for on-device inference reduces infrastructure overhead.

3. Implementation and Results:

3.1. Code and Repository

3.1.1. Code was uploaded on GitHub: https://github.com/romilp3058/TME_6015

3.1.2. Instructions for setup, training, and inference:

- The coding environment used was Jupyter Notebook. Blocks of code were executed in separate cells to simplify debugging during implementation.
- The code can be executed cell by cell.
- The only requirement to be followed is to place the dataset in the same directory as the code exists.

3.2. Model Performance

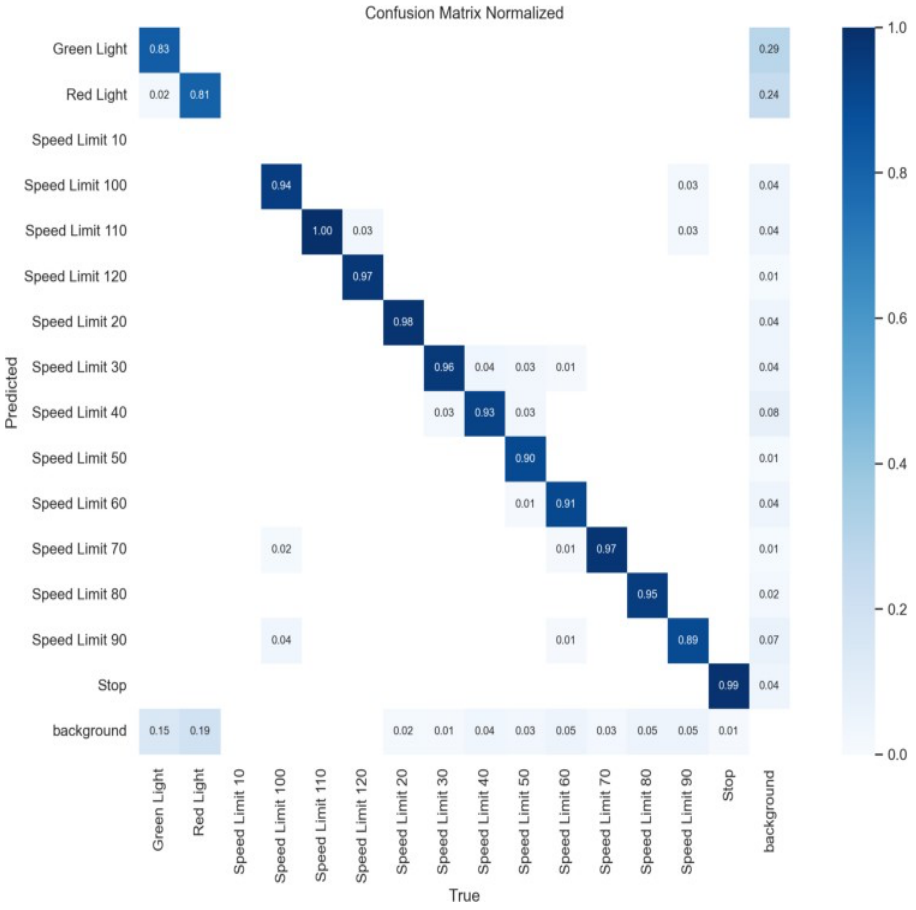
3.2.1. Results:

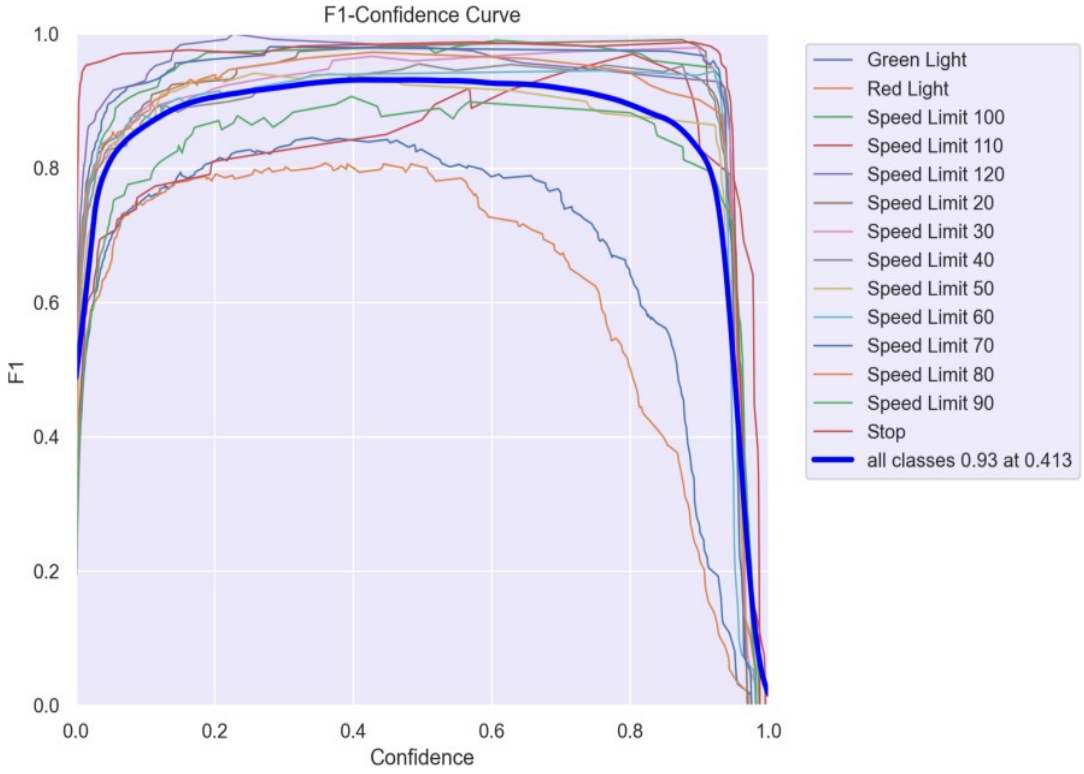
Results achieved during training and inference are as follows:

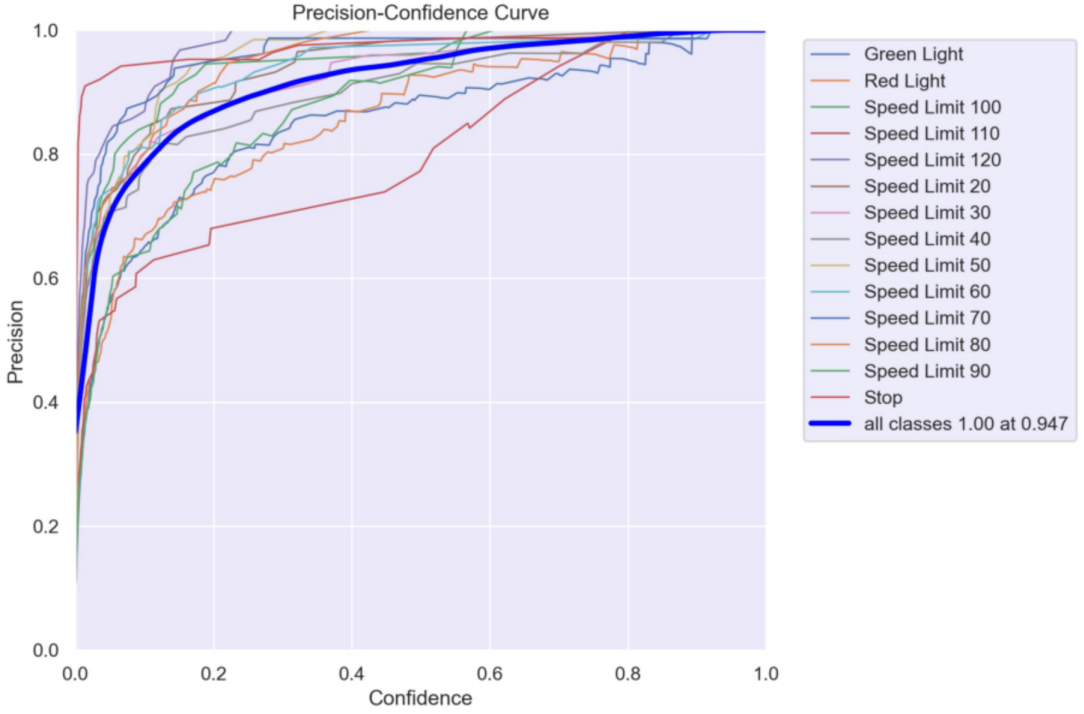
Model	TrainTime	Precision	Recall	mAP50
YOLOv8n	12.748 hrs	0.94	0.92	0.96
YOLOv9t	17.598 hrs	0.94	0.89	0.95
YOLOv10n	20.762 hrs	0.92	0.86	0.94

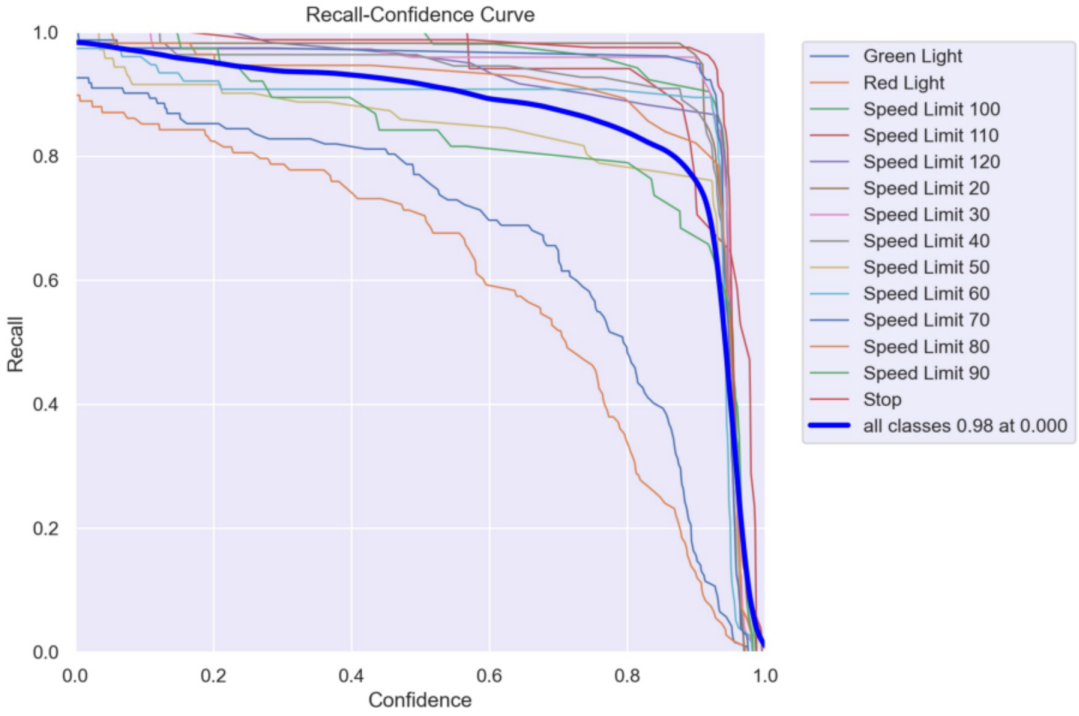
Accuracy was not used as a metric because the project deals with object detection. The goal is to not only classify the object correctly, but also identify the correct position of the identified object.

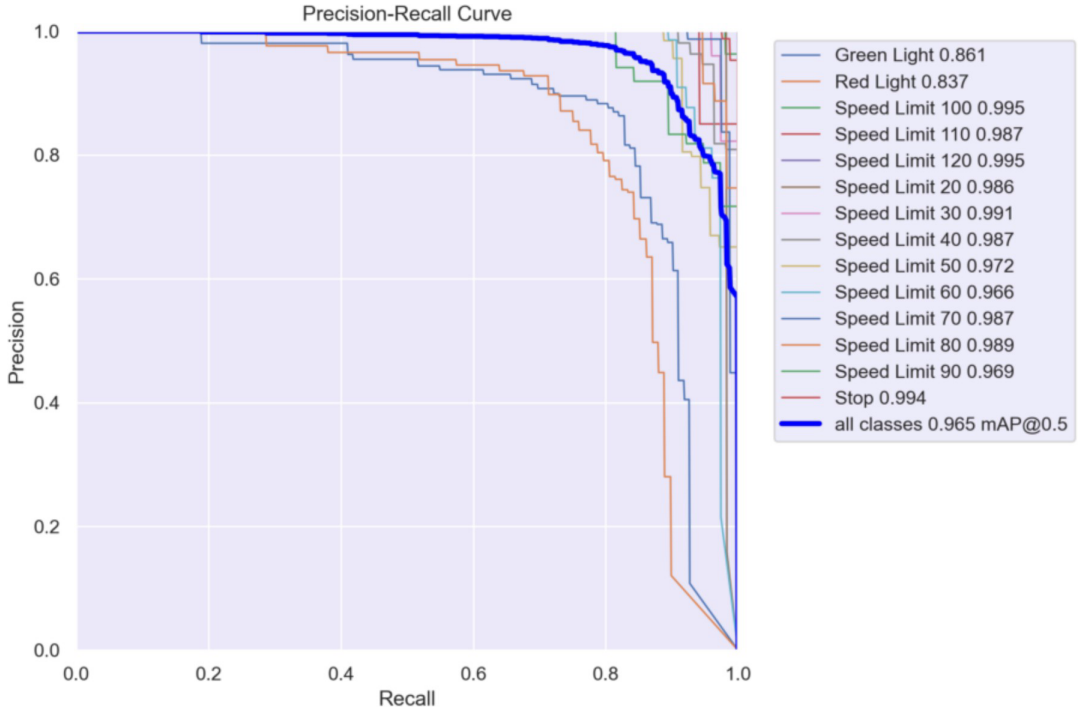
3.2.2. Provide visualizations such as confusion matrices, ROC curves, or sample outputs:




F1-Confidence Curve
F1
Confidence
0.0
0.2
0.4
0.6
0.8
1.0
Green Light
Red Light
Speed Limit 100
Speed Limit 110
Speed Limit 120
Speed Limit 20
Speed Limit 30
Speed Limit 40
Speed Limit 50
Speed Limit 60
Speed Limit 70
Speed Limit 80
Speed Limit 90
Stop
all classes 0.93 at 0.413


Precision-Confidence Curve
Precision
Confidence
0.0
0.2
0.4
0.6
0.8
1.0
Green Light
Red Light
Speed Limit 100
Speed Limit 110
Speed Limit 120
Speed Limit 20
Speed Limit 30
Speed Limit 40
Speed Limit 50
Speed Limit 60
Speed Limit 70
Speed Limit 80
Speed Limit 90
Stop
all classes 1.00 at 0.947


Recall-Confidence Curve
Recall
Confidence
0.0
0.2
0.4
0.6
0.8
1.0
Green Light
Red Light
Speed Limit 100
Speed Limit 110
Speed Limit 120
Speed Limit 20
Speed Limit 30
Speed Limit 40
Speed Limit 50
Speed Limit 60
Speed Limit 70
Speed Limit 80
Speed Limit 90
Stop
all classes 0.98 at 0.000


Precision-Recall Curve
Precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0
Green Light 0.861
Red Light 0.837
Speed Limit 100 0.995
Speed Limit 110 0.987
Speed Limit 120 0.995
Speed Limit 20 0.986
Speed Limit 30 0.991
Speed Limit 40 0.987
Speed Limit 50 0.972
Speed Limit 60 0.966
Speed Limit 70 0.987
Speed Limit 80 0.989
Speed Limit 90 0.969
Stop 0.994
all classes 0.965 mAP@0.5

3.3. Downstream Application Integration:

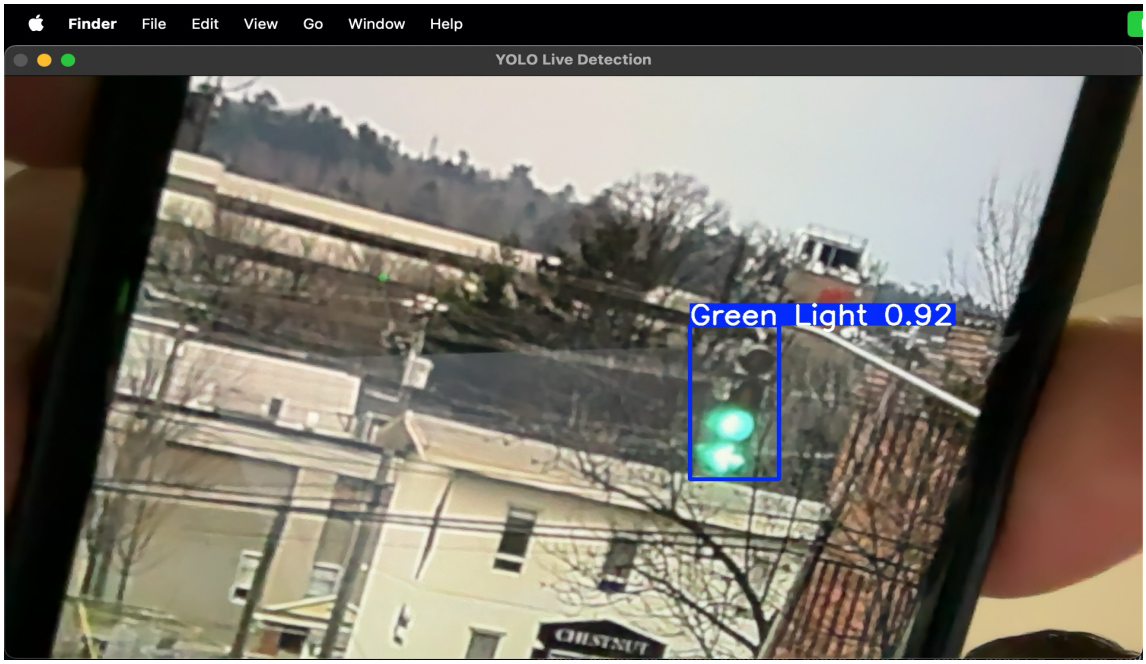
3.3.1. Demonstration of integration with a downstream application or process:

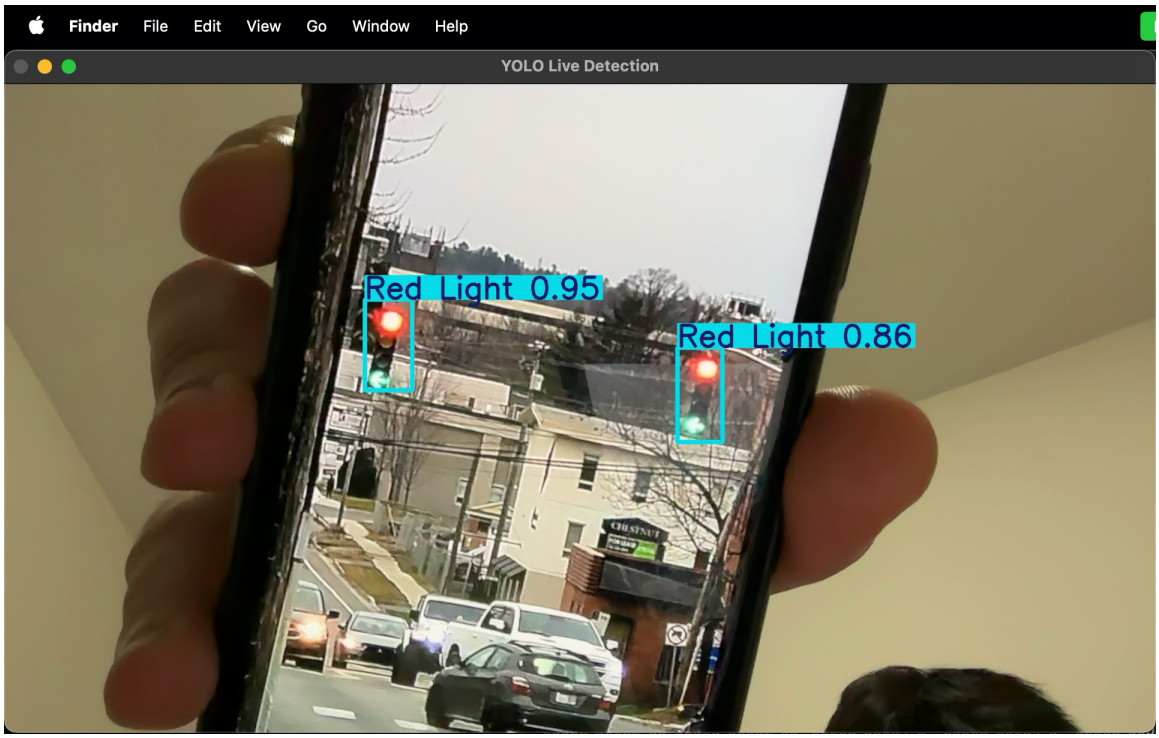
```
[5]: from IPython.display import Video

     # Get the path of the saved video
     output_video_path = "runs/detect/predict2/stop_video.mp4"

     # Display the video in Jupyter Notebook
     Video(output_video_path, embed=True, width=960, height=540)
```

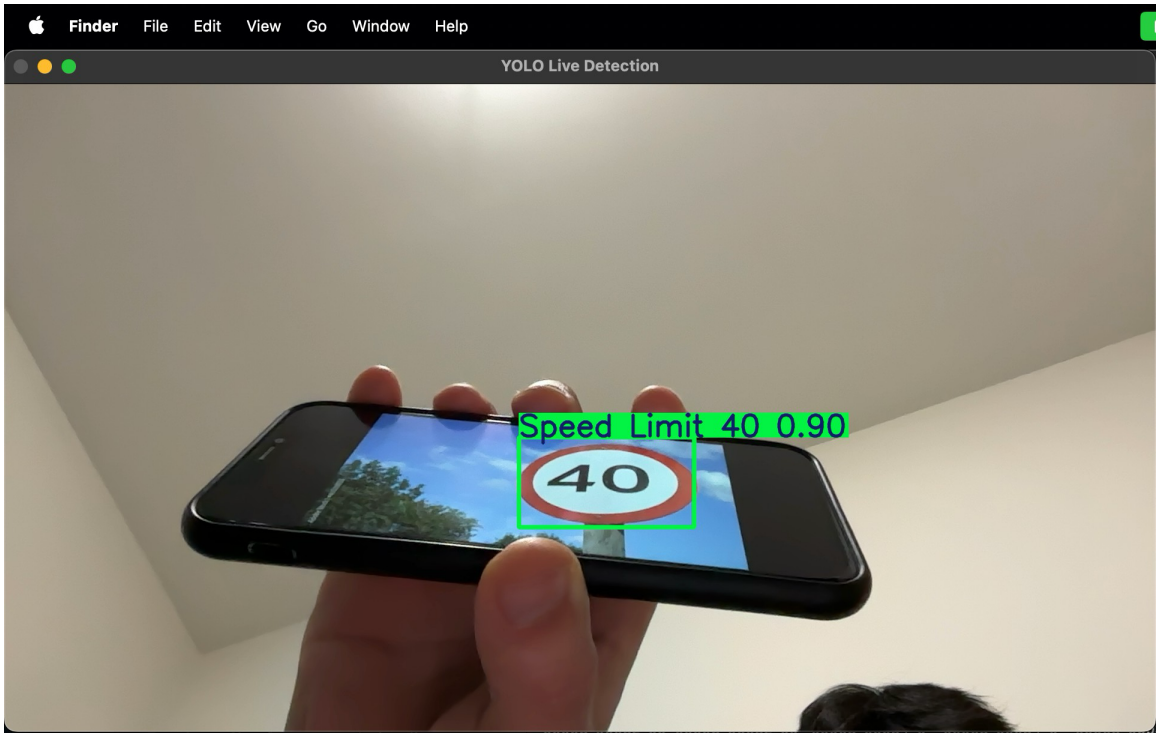
[5]:

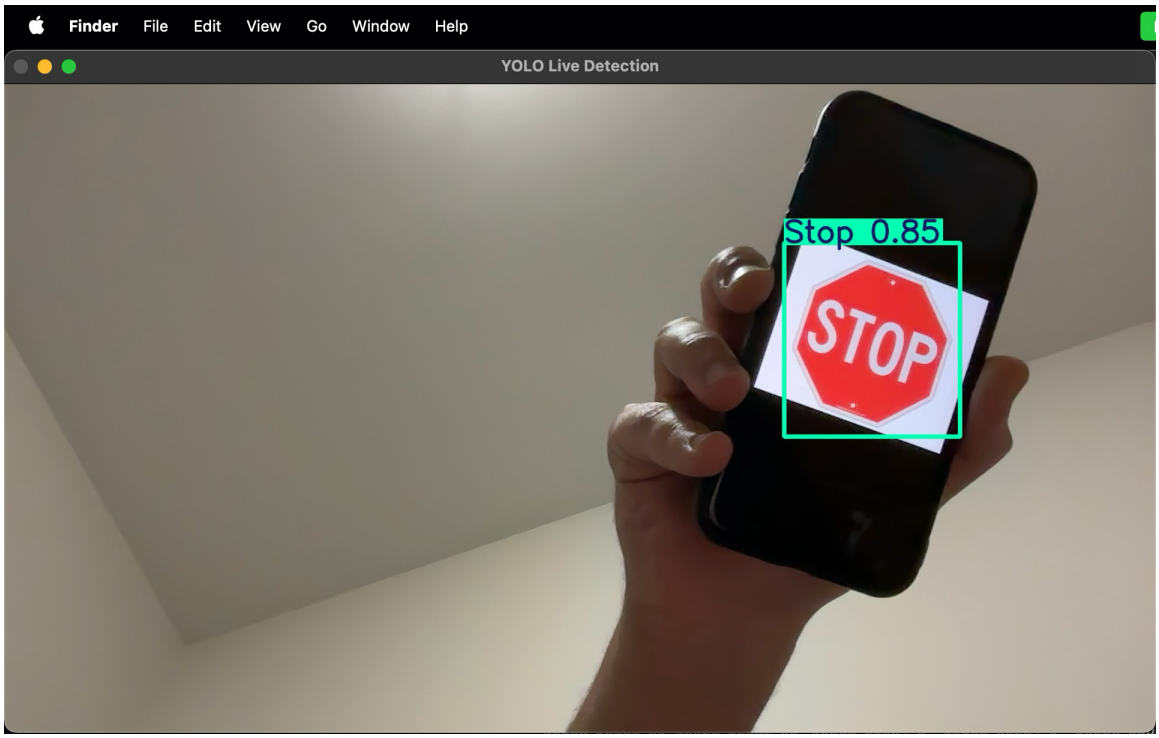
3.3.2. Use of model's output to provide actionable insights and solve a problem:

Below are the screenshots taken when running inferences on a live camera feed through a webcam.




Finder File Edit View Go Window Help
YOLO Live Detection
Red_Light 0.95
Red_Light 0.86


Finder File Edit View Go Window Help
YOLO Live Detection
Speed_Limit_40 0.90
40



The images indicate how the model can detect and classify traffic signs in a live video feed. Mentioned below are some of the ways how these outputs can be utilized:

- Alert the driver in case of traffic rules infringement.
- Applied to an automated car to detect traffic signs and road speed limits to response accordingly.

4. Discussion and Conclusion:

4.1. Interpretation of Results:

As we can observe from the confusion matrix, the performs well across all classes as evident through the strong diagonal values with Speed Limit 110 and Stop Sign achieving near perfect or perfect classification. However, the model gets confused between green lights and red lights.

The F1 confidence curve provides the following insights:

- The overall F1 score across all classes peaks at 0.93 with a confidence threshold of approximately 0.413 (indicated by the thick blue line).
- Most classes exhibit consistently high F1 scores across a wide range of confidence thresholds, indicating strong performance.

- Certain classes, such as Red Light and Green Light, show a decline in F1 scores at higher confidence thresholds, suggesting these classes are more sensitive to confidence tuning.
- The model performs uniformly well for most classes, with slight variations among traffic signs and speed limits.

4.2. Limitations and Future Work:

4.2.1. Acknowledge any limitations of the current approach:

While the project delivered successful results, the following limitations were realized:

- Class Imbalance: Classes underrepresented in the dataset reduced the performance for some of the traffic signs.
- Hardware Constraints: While YOLOv8 is optimized to be efficient, its deployment on resource-constrained edge devices may still pose a challenge.
- Static Dataset: The dataset did not have temporal data that could enable the evaluation of the model on dynamic, video-based scenarios.

4.2.2. Suggest potential improvements or areas for future research:

To address these limitations and advance the field, the following enhancements are proposed:

- Dataset Enrichment: Incorporating data from diverse environments will expand occurrence of rare traffic signs.
- Temporal Analysis: Extend the model to process sequential video frames, enabling it to account for motion and continuity, which are critical for real-time applications.
- Transferable Learning: Investigate the applicability of the model across regions with different traffic sign standards by fine-tuning it on regional datasets.

5. References:

[1] Akatsuka, H., C Imai, S. (1987). Road signposts recognition system (No. 870239). SAE Technical Paper.

[2] Fleyeh, H. (2004, December). Color detection and segmentation for road and traffic signs. In IEEE Conference on Cybernetics and Intelligent Systems, 2004. (Vol. 2, pp. 809-814). IEEE.

[3] W. Yang and W. Zhang, "Real-time Traffic Signs Detection Based on YOLO Network Model," 2020 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), Chongqing, China, 2020, pp. 354-357, doi: 10.1109/CyberC49757.2020.00066.

[4] Wang, C. Y., C Cheng-Yue, R. (2016). Traffic sign detection using you only look once framework. Standford, Tech. Rep.

[5] Huang, Z., Li, L., Krizek, G. C., C Sun, L. (2023). Research on traffic sign detection based on improved YOLOv8. Journal of Computer and Communications, 11(7), 226-232.

[6] https://docs.ultralytics.com/models/