CC-406 PROJECT-II: Project

Semester-7

Adaptive Traffic Signal Optimization Using Deep Learning for Smarter Traffic Management

submitted to

By

Mit Thaker
Zurin Lakdawala

Under the guidance of

Prof. Hardik Soni

M.Sc. (Integrated) Data Science

Department of AIML & Data Science

School of Emerging Science and Technology

Gujarat University

DECEMBER-2024

DECLARATION

I hereby declare that the study entitled **"Adaptive Traffic Signal Optimization Using Deep Learning for Smarter Traffic Management"** submitted to the Gujarat University, Navarangpura, Ahmedabad (Gujarat) in partial fulfilment of M.Sc. (Int) Data Science degree is the result of investigation done by myself. The material that has been obtained (and used) from other sources has been duly acknowledged in this study. It has not been previously submitted either in part or whole to this or any other university or institution for the award of any degree or diploma.

Place: Ahmedabad — Signature

Date: — Name of Student

Index

Abstract

Traffic congestion in urban areas is a growing challenge, leading to longer travel times, increased pollution, and a lower quality of life. Traditional fixed-timing traffic signals are often ineffective in managing traffic flow, as they don't adjust to real-time traffic conditions. This project introduces an innovative Adaptive Traffic Signal Optimization System that combines YOLOv8 for real-time vehicle detection with a Genetic Algorithm to optimize green light durations.

The YOLOv8 model accurately detects and counts vehicles from live traffic feeds, providing real-time data on vehicle volume in each lane. This information allows the system to adjust signal timings dynamically, ensuring that traffic is managed more efficiently, especially during peak hours or fluctuating traffic conditions.

The Genetic Algorithm is used to determine the optimal green light durations based on the vehicle count, continuously evolving the traffic light timings to reduce waiting times and improve traffic flow. By evaluating different timing strategies, the algorithm ensures that the traffic signals respond to actual road conditions rather than relying on static schedules.

This adaptive approach offers multiple benefits, including:

1. Reduced Congestion: More efficient traffic flow through optimized signal timings.
2. Decreased Fuel Consumption: Reduced idling time leads to lower fuel usage and less pollution.
3. Improved Traffic Efficiency: Faster commute times for drivers and smoother traffic movement.

By integrating AI and optimization techniques, this system helps pave the way for smarter, more sustainable urban traffic management, ultimately leading to reduced congestion, better air quality, and improved commuter experience.

Introduction

With urbanization rapidly increasing across the globe, cities are facing the growing challenge of efficiently managing traffic flow. As populations swell and the number of vehicles on the road rises, traditional fixed-timing traffic signal systems are proving inadequate. These systems operate on a rigid, preset cycle, which doesn't adapt to fluctuating traffic conditions, leading to frequent congestion, longer commute times, and elevated vehicle emissions. This inefficiency not only frustrates commuters but also contributes to pollution and unnecessary fuel consumption, worsening urban air quality.

To address these challenges, Adaptive Traffic Signal Optimization offers a revolutionary solution. This intelligent system dynamically adjusts the timing of traffic signals based on real-time traffic data, improving traffic flow, reducing congestion, and enhancing overall road efficiency. By incorporating cutting-edge technologies, the system ensures that traffic signals are always optimized for current traffic patterns, rather than relying on outdated fixed timings.

At the heart of this project is the YOLOv8 model, a highly advanced deep learning algorithm renowned for its outstanding accuracy and speed in real-time object detection. YOLOv8 is used to monitor and analyze traffic flows, providing real-time vehicle counts from traffic camera feeds. Its ability to detect vehicles with high precision ensures that the system always has up-to-date and reliable data to inform its decisions. This data forms the basis for the subsequent optimization of signal timings.

Once the vehicle counts are captured, they are fed into a Genetic Algorithm, which uses evolutionary principles to calculate the optimal green light durations for each direction. The algorithm simulates multiple solutions and continuously evolves to find the best timing strategy, ensuring that vehicles are cleared efficiently from intersections, minimizing waiting times and preventing congestion.
The system’s adaptive nature offers a host of benefits. First, it ensures that green light durations are tailored to the actual traffic demand at any given time, improving traffic distribution across all directions. This not only helps alleviate congestion but also reduces the time vehicles spend idling at traffic lights, leading to lower fuel consumption. As a result, the system significantly reduces carbon emissions, contributing to cleaner, more sustainable urban environments.

Moreover, this Adaptive Traffic Signal Optimization System enhances the commuting experience by reducing travel times, easing frustration, and ensuring smoother movement of traffic. By combining the power of deep learning for vehicle detection and evolutionary algorithms for optimization, this project represents a step toward smarter, more efficient cities.

The goal of this documentation is to provide a comprehensive overview of the development, implementation, and potential of the Adaptive Traffic Signal Optimization System. It explores how these innovative technologies can work together to improve urban traffic management, reduce environmental impact, and help pave the way for future smart cities that are better equipped to handle the demands of rapid urbanization.

Data-Preprocessing

For this project, the dataset titled "Traffic Vehicles Object Detection" was sourced from Kaggle. This dataset contains a total of 1,016 annotated images, which are carefully split into subsets to ensure optimal training and evaluation of the YOLOv8 model. The dataset is divided into the following subsets:

- **738 Training Images:** These images are used to train the YOLOv8 model, allowing it to learn to detect and classify vehicles from various angles and under different conditions.
- **278 Test Images:** These images are used to evaluate the performance of the trained model and determine its generalization capability on unseen data.
- **185 Validation Images:** These images are used for hyperparameter tuning during the training process. They help ensure that the model doesn't overfit to the training data by providing an unbiased evaluation of model performance at various stages of training.

Each image in the dataset is annotated with one of the following five vehicle classes:

1. **Car:** Includes all types of four-wheeled vehicles such as sedans, hatchbacks, and SUVs.
2. **Two-Wheeler:** Includes motorcycles, scooters, and other two-wheeled vehicles.
3. **Auto:** Refers to autorickshaws, which are a common form of public transport in many regions.
4. **Bus:** Includes large public transport buses used for carrying passengers.
5. **Truck:** Includes large goods-carrying vehicles such as delivery trucks, lorries, and semi-trucks.

The preprocessing of the data was crucial to ensure that the dataset was clean, well-organized, and properly formatted for use with the YOLOv8 object detection model. The following steps were undertaken to prepare the data:

1. Annotation Verification and Correction

The dataset annotations were first thoroughly verified and corrected to ensure their accuracy and consistency. Roboflow was used to assist with annotation verification and correction. This tool helps identify any discrepancies or errors in the labels, ensuring that each image is correctly annotated with the appropriate vehicle class. This step is critical as incorrect annotations can severely impact model performance and lead to poor detection results.

2. Image Resizing and Normalization

The images in the dataset were resized to a consistent dimension of 640x640 pixels. This resizing step is necessary to meet the input size requirements of the YOLOv8 model, which typically expects square images of this size. Additionally, the pixel values of the images were normalized to scale the pixel intensities between 0 and 1. Normalization is an important preprocessing step as it helps the model learn more efficiently by reducing the variance in the input data and ensuring consistent numerical ranges across all images.

3. Dataset Splitting

The dataset was carefully split into three subsets: training, validation, and testing. This split was done with a focus on ensuring that each subset had a balanced representation of all five vehicle classes. This is particularly important because a skewed dataset, where certain vehicle types are overrepresented, can result in a model that is biased towards detecting the overrepresented classes. By maintaining balance across the subsets, the model is better able to learn to detect and classify all vehicle types equally well.

4. Conversion to YAML Format

Once the dataset was preprocessed, it was converted into a .yaml configuration file, which is essential for feeding the data into the YOLOv8 model. The YAML file contains critical information about the dataset, including:

- Path to images: Specifies where the images are located.
- Path to annotations: Specifies where the annotation files are stored.
- Class names: Lists the five vehicle classes in the dataset.
- Training, validation, and test set splits: Indicates how the data is divided into the three subsets.

This YAML file serves as the configuration for YOLOv8, providing the model with the necessary details to load and process the dataset during training.
These preprocessing steps ensured that the dataset was not only clean and balanced, but also compatible with the YOLOv8 framework. Proper data preparation plays a key role in the accuracy and effectiveness of object detection models, as it provides the model with high-quality, well-structured input. With this data, the YOLOv8 model can effectively learn to detect vehicles across a wide range of scenarios, setting the foundation for an efficient and reliable vehicle detection system.

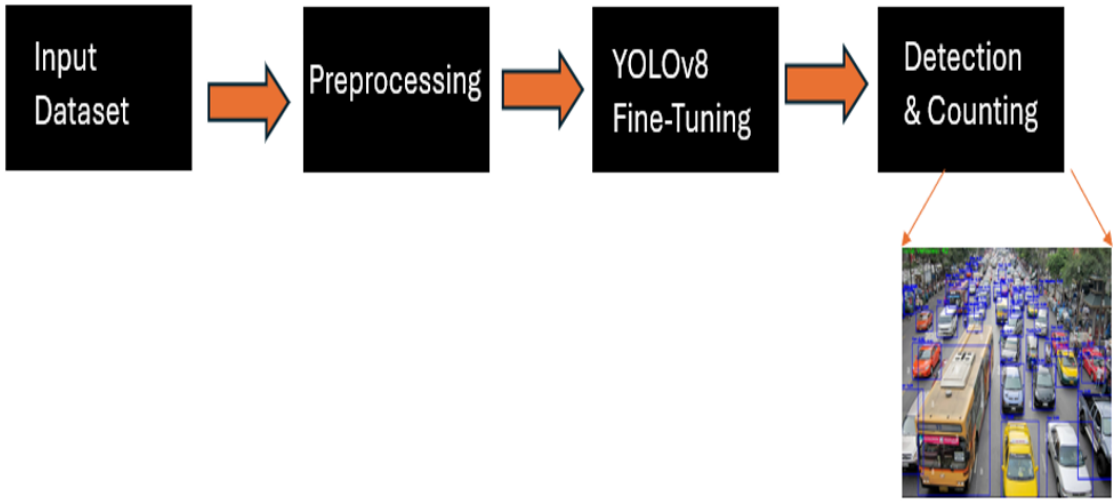
[1]:

```
data = """
train: /kaggle/input/traffic-vehicles-object-detection/Traffic Dataset/images/train
val: /kaggle/input/traffic-vehicles-object-detection/Traffic Dataset/images/val

nc: 7
names: ['Car',Two Wheeler', 'Auto', 'Bus', 'Truck']
"""

path = '/kaggle/working/dataset.yaml'


with open(path, 'w') as file:
    file.write(data)

print(f"dataset.yaml has been saved to: {path}")k
```

```
dataset.yaml has been saved to: /kaggle/working/dataset.yaml
```

Model Development

Fine-Tuning YOLOv8 for Vehicle Detection



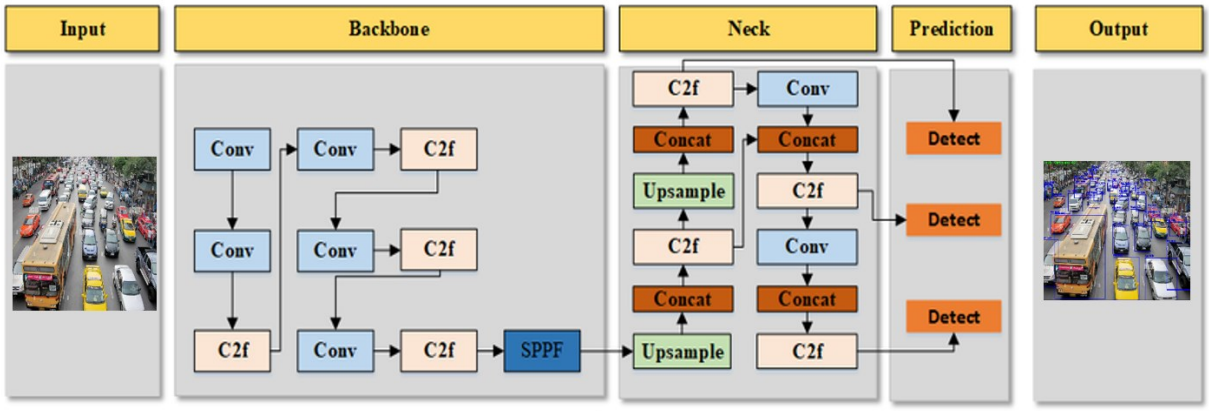
YOLOv8 Overview

YOLO (You Only Look Once) is a state-of-the-art, real-time object detection algorithm known for its high speed, accuracy, and generalization capabilities. The YOLO model has become one of the most popular algorithms for object detection, due to its ability to perform detection and classification in a single pass. This makes YOLO extremely efficient, making it well-suited for real-time applications where both speed and accuracy are crucial.

YOLOv8 is the latest iteration of the YOLO model and builds on the success of its predecessors while introducing several architectural and performance improvements. YOLOv8 uses a **Convolutional Neural Network (CNN)** architecture, allowing it to process entire images at once. This unique architecture enables YOLOv8 to detect and classify multiple objects, such as vehicles, people, and other objects, with high precision. The real-time nature of the model makes it particularly suitable for applications such as traffic surveillance, autonomous driving, and video surveillance, where quick and accurate object detection is necessary.

1. YOLOv8 Architecture

The YOLOv8 model architecture is divided into three main components:



Source :- https://www.researchgate.net/figure/YOLOv8-object-detection-architecture_fig4_377835363

1.Backbone (Feature Extractor)

The **Backbone** is the first component of the YOLOv8 architecture, and it is responsible for feature extraction. This part of the model processes the input image and extracts essential features that will help in detecting objects in the image. It functions as the foundational layer that transforms raw pixel data into useful information for higher layers.

- **Low-level Feature Extraction**: In the initial layers, the backbone captures simple patterns such as edges, textures, and basic shapes. These low-level features are essential for recognizing objects at a granular level.
- **Hierarchical Feature Extraction**: As the network moves deeper, it starts extracting more abstract and high-level features. This helps the model recognize more complex structures, such as the contours of objects and their contextual relationships with the surrounding environment.
- **Common Backbone Architectures**: YOLOv8, like its predecessors, typically uses a pre-trained backbone (like **CSPDarknet** or **EfficientNet**) that has been optimized for feature extraction tasks, allowing the model to generalize well across various datasets.

By using a deep and efficient feature extractor, YOLOv8 can process complex images and understand both fine details (like small objects) and broader context (like object relationships within a scene).

2. Neck

The **Neck** component serves as the intermediary between the Backbone and the Head. Its role is to enhance the features extracted by the Backbone and prepare them for final output by performing operations that fuse information across different levels of the model.

- **Feature Fusion**: The Neck is responsible for combining features from various stages of the Backbone. It uses a technique known as **feature pyramid networks (FPN)** or **path aggregation networks (PAN)** to fuse multi-scale features from both low-level and high-level layers. This allows the model to recognize objects at various sizes, from small to large, and helps in dealing with occlusions and scale variations in the image.
- **Contextual Integration**: By aggregating features from different levels, the Neck ensures that the model has a comprehensive understanding of both local details (e.g., the shape of a vehicle) and global context (e.g., its position in the scene).
- **Improved Object Localization**: The Neck enhances the model's ability to accurately localize objects in an image by ensuring that the feature maps retain both high-resolution details and contextual information.
- The Neck essentially helps the YOLOv8 model bridge the gap between detecting simple patterns and generating the final output, improving the accuracy and robustness of object detection.

3.Head

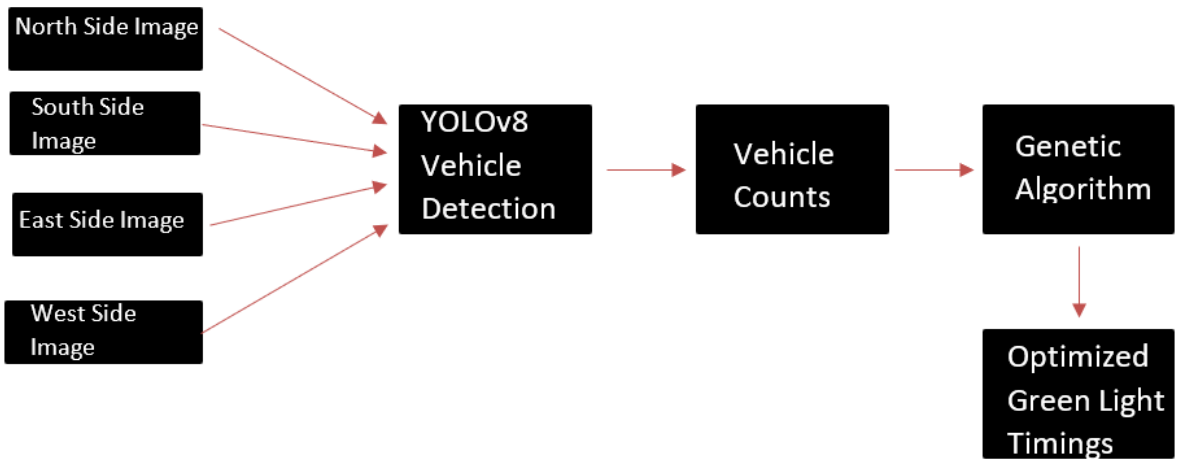
The **Head** is the final component of the YOLOv8 model and generates the ultimate output of the network. This component processes the fused features from the Backbone and Neck and produces the results in the form of object detections.

- **Bounding Box Prediction**: The Head predicts the coordinates of bounding boxes that outline the detected objects in the image. These bounding boxes are defined by their **center coordinates (x, y)**, **width (w)**, and **height (h)**, which allow the model to locate the objects within the image.
- **Class Prediction**: In addition to bounding boxes, the Head assigns a **class label** (e.g., Car, Truck, Bus) to each detected object. YOLOv8 is capable of detecting multiple classes simultaneously, making it versatile for multi-object detection tasks.

- **Confidence Scores**: For each detected object, the Head also generates a **confidence score**, indicating the model's certainty that the predicted bounding box contains a particular object. This score helps in filtering out low-confidence predictions, ensuring that only reliable detections are kept.
- **Object Detection Output**: The final output of the Head is a set of bounding boxes, each associated with a class label and a confidence score. The results are then passed through a **Non-Maximum Suppression (NMS)** algorithm to remove redundant detections and retain only the most accurate predictions.

The Head component essentially transforms the extracted features into actionable results that can be used in applications such as traffic monitoring, autonomous vehicles, and surveillance systems.

Overall System Workflow



Genetic Algorithm for Traffic Signal Optimization

A Genetic Algorithm (GA) is a search heuristic inspired by natural evolution, used for solving optimization and search problems. It mimics the process of natural selection to evolve solutions toward optimality. In traffic signal optimization, GAs is applied to allocate green light times effectively, reducing congestion and delay while considering road capacity and vehicle flow.

Key Components of the Genetic Algorithm

1. Population Initialization

This step creates a population of random green time configurations for each traffic direction while ensuring the total green time adheres to the cycle time limit. The configurations are evaluated and sorted based on their delay values, prioritizing solutions that minimize delay.

2. Fitness Function

The fitness function evaluates the efficiency of traffic signal timings in terms of minimizing delays and penalties.
Variables:

- C: Total cycle time.
- g: Green time for a specific direction.
- x: Vehicle count in a given direction.
- c: Road capacity.
- vehicles: List of vehicle counts in each direction.

Components:

1. **Delay Terms:**
    - **Underutilization Delay**:

$$(1-(g/C))2(1-(g/C))^2(1-(g/C))2$$

    - **Penalty Delay**

$$1-((g/C)\times x)1-((g/C)\times x)1-((g/C)\times x)$$

- **Comprehensive Delay**:

$$d1i = 0.38 * C * a/p$$

- Delay due to underutilized green time and vehicle count.

$$a2 = 173 * (x^2)$$

- Affects delay, with higher vehicle count increasing delay.

$$ri1 = (x - 1) + (x - 1)^2 + (16 * x/c)$$

- Adjusts delay based on vehicle count and road capacity.

$$d2i = a2 * ri1$$

- Delay due to vehicle count and capacity.

2.Penalty Factor: Ensures green times are proportional to vehicle counts:

a. Ensures green times are proportional to vehicle counts.

b.

$$Penalty = \sum |gi/\sum gi - vi/\sum vi| * 100$$

Penalizes unequal distribution of green time relative to vehicle count.

Final Formula:

Total Delay = d1i + d2i + Penalty Factor

This formula evaluates the traffic signal timings, balancing delays and penalties to optimize the green times.

3.Green Time Normalization

The **normalize green times** function adjusts the green light times based on the proportion of vehicles in each direction, relative to the total number of vehicles at all intersections.

Vehicle Proportion:

$$\text{vehicle\_proportion}_i = \frac{v_i}{\sum_{i=1}^{N} v_i}$$

where v_i is the number of vehicles in direction i.

Normalized Green Time:

$$\text{normalized\_green\_time}_i = \text{cycle\_time} \times \text{vehicle\_proportion}_i$$

This adjusts the green time for each direction based on its proportion of vehicles.

Here's a more concise and detailed version of the content:

4. Selection (Roulette Wheel)

In the **Roulette Wheel** method, the probability of selecting a parent solution is proportional to its fitness. Solutions with better performance have a higher chance of being chosen. The roulette wheel metaphor is used, where each solution occupies a segment of the wheel, and the larger the segment (based on fitness), the higher the likelihood of selection. This ensures that optimal solutions are more likely to be passed to the next generation.

5. Crossover

Crossover combines two parent solutions to produce offspring. A random crossover point is chosen, and parts of the green time configurations are swapped between parents. This allows offspring to inherit the strengths of both parents. Crossover creates diverse solutions, exploring new combinations of green times that may optimize traffic flow better.

6. Mutation

Mutation introduces small, random changes to a solution's green time configuration. These changes help avoid premature convergence and explore new possibilities. For example, randomly adjusting the green time for a specific direction introduces variability, which can lead to better solutions over time.

7. Inversion

Inversion reverses a randomly selected segment of the green time configuration. This adds further diversity to the population, preventing repetitive patterns and encouraging the discovery of different, potentially better solutions. By flipping sections of the green time schedule, the algorithm can escape local optima.

8. Genetic Algorithm Loop

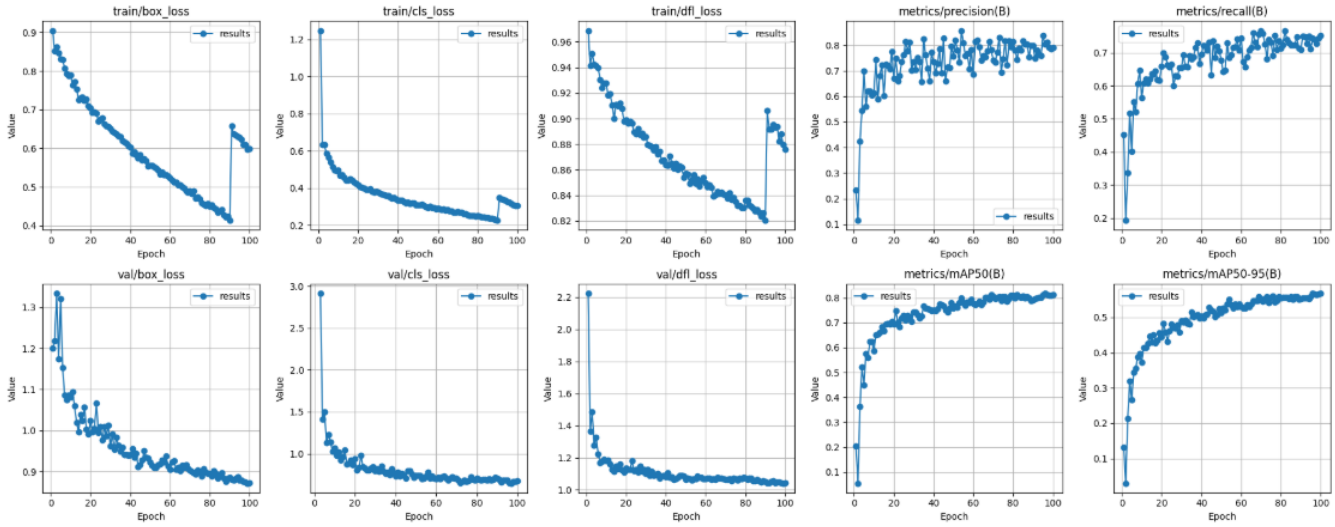
The GA works in iterations, following these steps:

1. **Initialization**: Generate and evaluate a random population of solutions.
2. **Selection**: Choose parents based on their fitness for crossover.
3. **Crossover**: Combine parents to create offspring.
4. **Mutation & Inversion**: Apply small random changes to maintain diversity.
5. **Evaluation**: Assess the fitness of the new solutions.
6. **Repeat**: Continue the process until an optimal or near-optimal solution is found.

Each iteration refines the solutions to improve green time allocation for better traffic management.

Result

Model Training and Performance Analysis Report



Loss Functions:

1. **Training Loss:**
    - **Train/box_loss**: The box loss steadily decreases across epochs, indicating improved localization predictions. However, there is a noticeable fluctuation around epoch 80, suggesting a temporary difficulty in the model's ability to minimize this loss.
    - **Train/cls_loss**: The classification loss also shows a significant decrease throughout the epochs, indicating the model's improving ability to classify the objects.
    - **Train/dfl_loss**: Similar to the box and class losses, the DFL loss decreases steadily, with a slight jump at epoch 80, suggesting a similar training irregularity.

2. **Validation Loss:**
    - **Val/box_loss**: The validation box loss follows a similar pattern to the training loss, showing a steady decline and some fluctuation at around epoch 80.
    - **Val/cls_loss**: The validation classification loss decreases consistently and reaches a much lower value than the training loss, indicating better generalization during validation.
    - **Val/dfl_loss**: There is a sharp decline in validation DFL loss, followed by minor fluctuations, suggesting the model is generalizing well in terms of object localization.

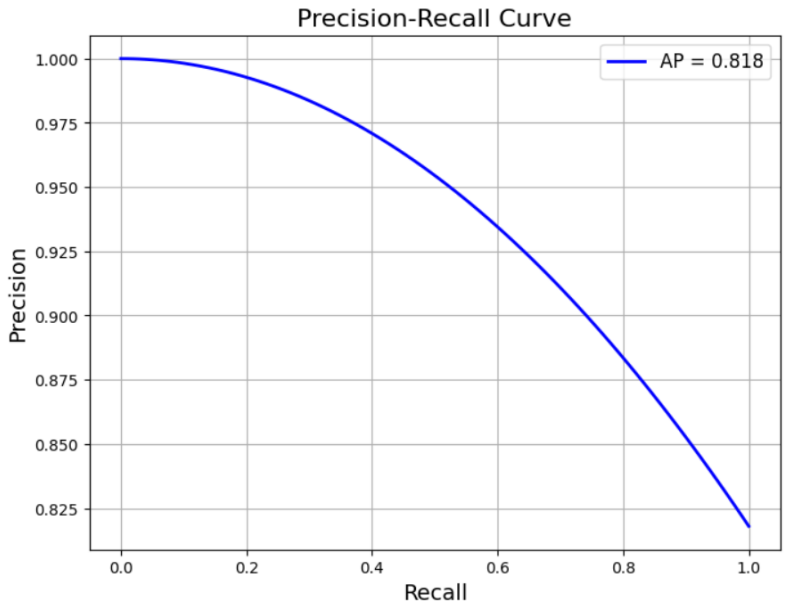
Performance Metrics:

1. **Precision and Recall:**
    - **Precision(B)**: Precision steadily increases throughout the epochs and plateaus, suggesting that the model is improving in terms of correctly identifying the traffic vehicles.
    - **Recall(B)**: Recall also increases consistently, indicating that the model's ability to correctly detect all relevant objects (true positives) is improving over time.

2. **mAP Metrics:**

- **mAP50(B)**: The mean average precision at IoU=0.5 increases significantly, reflecting improved detection performance across epochs.
- **mAP50-95(B)**: The mAP at IoU between 0.5 and 0.95 also shows improvement, though it lags behind mAP50, which is expected since higher IoU thresholds are more demanding for accurate localization.

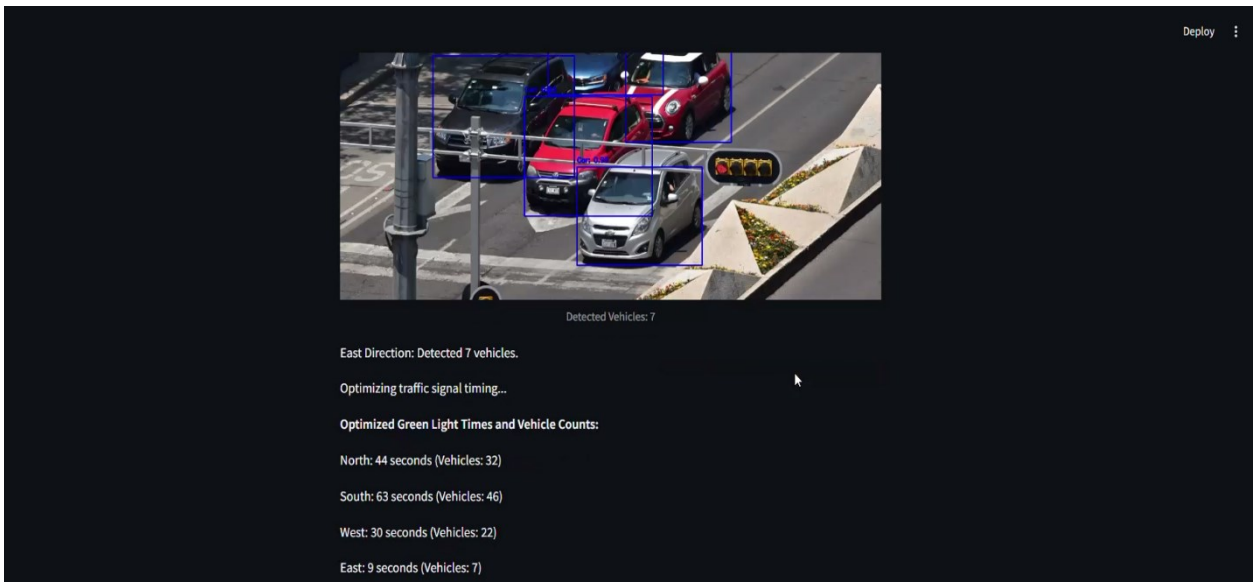
Performance of Model: Precision vs. Recall



- **Precision-Recall Curve** shows the trade-off between precision and recall.
- **Average Precision (AP)** is 0.818, indicating good performance.

- Precision stays above 0.825 as recall increases, showing minimal loss in precision.
- The model effectively balances precision and recall, with a high AP score.
- Overall, the model demonstrates strong detection performance.

Working website output



Conclusion

- The project demonstrates the potential for optimizing urban traffic management using genetic algorithms to adjust traffic signal timings in real-time, improving traffic flow, reducing congestion, and minimizing delays.
- Real-life implementation faces several challenges, including prioritizing emergency vehicles, accurately capturing lane-specific traffic data from various sensors, managing unpredictable traffic patterns, and integrating with existing traffic infrastructure.
- Overcoming these challenges will require advanced sensors, real-time data processing, and effective collaboration between traffic authorities and technology providers.
- The use of innovative technologies such as 5G networks, edge computing, and AI will be essential to enhance the responsiveness and efficiency of the system.
- Despite the challenges, this approach lays a strong foundation for developing smarter, adaptive traffic management systems, which could significantly improve urban mobility, safety, and sustainability.
- This system could become a key component of the smart cities initiative, where infrastructure and technology work together to optimize urban living conditions.

- As urban populations grow and cities become more complex, intelligent traffic management systems will be critical for addressing issues such as congestion, safety, and environmental sustainability.
- The project shows promise and, with further development, could play a major role in the future of urban traffic management.

References

1. Ravichandran, M., Laxmikant, K., & Muthu, A. (2024, March). Smart Traffic Control: Adaptive Signal Management Based on Real-time Lane Detection using YOLOv8. In *2024 International Conference on Automation and Computation (AUTOCOM)* (pp. 174-180). IEEE.

2. Teo, K. T. K., Kow, W. Y., & Chin, Y. K. (2010, September). Optimization of traffic flow within an urban traffic light intersection with genetic algorithm. In *2010 Second International Conference on Computational Intelligence, Modelling and Simulation* (pp. 172-177). IEEE.

3. Farooqi, A. H., Munir, A., & Baig, A. R. (2009). THE: traffic light simulator and optimization using Genetic Algorithm. In *International Conference on Computer Engineering and Applications IPCSIT* (Vol. 2).

4. Hassan, M., Mahin, H. D., Jusoh, M., Al Nafees, A., Paul, A., & Islam, M. A. (2024, August). Vehicle Class Detection and Counting on a Malaysian Road Using YOLOv8 and OpenCV. In *2024 IEEE International Conference on Artificial Intelligence in Engineering and Technology (IICAIET)* (pp. 271-275). IEEE

Code Attachment

https://github.com/Zurinlakdawala91/Adaptive-Traffic-Signal-Optimization-Using-Deep-Learning-for-Smarter-Traffic-Management