International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

# INTELLIGENT TRAFFIC MANAGEMENT SYSTEM

***Rishikesh C[1], Vishnu N[2], Kamala Kannan G[3], Mr. B. Muthukrishna Vinayagam[4]***

[1] UG - Computer Science and Engineering, Kamaraj College of Engineering and Technology, Virudhunagar, Tamil Nadu
[2] UG - Computer Science and Engineering, Kamaraj College of Engineering and Technology, Virudhunagar, Tamil Nadu
[3] UG - Computer Science and Engineering, Kamaraj College of Engineering and Technology, Virudhunagar, Tamil Nadu
[4] Assistant Professor, Computer Science and Engineering, Kamaraj College of Engineering and Technology, Virudhunagar, Tamil Nadu
**[1]21ucs080@kamarajengg.edu.in , [2]21ucs098@kamarajengg.edu.in , [3]21ucs047@kamarajengg.edu.in ,**
**[4]muthukrishnavinayagamcse@kamarajengg.edu.in**

**ABSTRACT :**

Managing city traffic is becoming harder with more people moving to urban areas and more cars on the roads. Traditional traffic systems, which rely on fixed schedules and simple sensors, struggle to keep up with the fast-changing and complicated nature of modern traffic. These systems often react too slowly, causing ongoing problems like traffic jams, delays, and safety concerns. Although earlier methods like adaptive traffic lights and basic vehicle detection have helped, they are limited in handling large-scale traffic and reacting quickly in real-time. This paper introduces an Intelligent Traffic Management System (ITMS) that uses advanced technologies such as computer vision, machine learning, and real-time data processing. The ITMS monitors traffic through live camera feeds and sensors, identifying vehicles and pedestrians to adjust traffic lights as needed. By improving traffic flow, enhancing safety, and providing helpful insights for traffic managers, the ITMS offers a smart, adaptable solution for better managing city traffic and making urban living safer and smoother.

**Keywords** Traffic management, Smart traffic system, Real-time signal control, Computer Vision, Machine Learning, Traffic flow improvement, Urban mobility, Traffic safety, Congestion reduction.

## I. INTRODUCTION

### *1. Automated Traffic Management Systems (ATMS)*

The rapid urbanization of metropolitan areas has brought about significant challenges in transportation systems, including increased traffic congestion, longer commute times, higher accident rates, and greater environmental pollution. Traditional traffic management solutions, such as fixed timers and manual interventions, are no longer sufficient to handle the dynamic and complex traffic patterns in modern cities. As a result, **Automated Traffic Management Systems (ATMS)** have emerged as an effective solution to address these issues.
ATMS utilizes real-time data and advanced technologies to manage and optimize traffic flow, improve safety, and reduce congestion. The integration of **Artificial Intelligence (AI)**, **Machine Learning (ML)**, and **Computer Vision (CV)** has significantly enhanced the capabilities of traffic management systems. These systems can analyze real-time data from traffic cameras, sensors, and other monitoring devices to make dynamic decisions, such as adjusting traffic light timings, rerouting traffic, or controlling pedestrian signals.
Automated traffic management systems aim to create more intelligent and responsive traffic environments, optimizing the movement of vehicles, pedestrians, and public transportation while reducing delays and environmental impact.

### *2. Convolutional Neural Networks (CNNs)*

**Convolutional Neural Networks (CNNs)** are a class of deep learning algorithms widely used for processing structured grid data, particularly images. CNNs have revolutionized the field of computer vision due to their ability to automatically learn hierarchical features from raw pixel data, eliminating the need for manual feature extraction. This makes CNNs particularly effective in tasks such as image classification, object recognition, and image segmentation.
In the context of automated traffic systems, CNNs are employed to detect and classify various objects within traffic scenes, such as vehicles, pedestrians, traffic signs, and signals. The ability of CNNs to extract features and learn patterns from images enables them to perform high-accuracy object detection and recognition in real-time, even under diverse and challenging conditions like varying lighting, weather, or occlusions.
CNNs are a key enabler of intelligent traffic management, helping systems process vast amounts of image and video data from cameras and sensors, making real-time decisions about traffic control and safety.

### *3. YOLO Algorithm*

The **YOLO (You Only Look Once)** algorithm is a state-of-the-art deep learning model for real-time object detection. YOLO's approach is fundamentally different from traditional object detection methods, as it simultaneously predicts multiple bounding boxes and class probabilities in a single pass over an image, making it exceptionally fast and efficient. This capability is particularly well-suited for applications that require real-time performance, such as traffic management systems.

YOLO divides an image into a grid and, for each grid cell, it predicts the presence of objects and their bounding boxes. It also classifies each object based on its category, such as vehicle, pedestrian, or traffic signal. The key advantage of YOLO lies in its speed and accuracy, making it capable of detecting multiple objects (e.g., cars, trucks, buses, pedestrians) within a single frame of a video feed or image. This is crucial for dynamic traffic environments, where vehicles and pedestrians are constantly moving.

For intelligent traffic management systems, YOLO plays a vital role in monitoring and detecting vehicles, pedestrians, and other important traffic elements, which then influence the system's decision-making process, such as adjusting traffic signal timings or controlling traffic flow based on real-time conditions.

### *4. Pygame for Traffic Simulation*

**Pygame** is a Python library widely used for developing 2D games and simulations. While its primary use is in game development, Pygame's versatility allows it to be applied to simulate various real-world scenarios, including traffic flow. In the context of automated traffic systems, Pygame is utilized to create dynamic, interactive simulations of traffic environments.

Through Pygame, developers can visualize various traffic elements such as **traffic lights**, **vehicles**, **pedestrians**, and **traffic rules** within a simulated environment. These simulations are useful for testing and visualizing traffic management algorithms, providing a platform where different traffic scenarios can be replicated, analyzed, and optimized. Pygame allows the simulation of traffic congestion, the interaction between different vehicles, and the real-time changes in traffic signal timings, which are essential for evaluating the performance of a traffic management system before real-world deployment.

Additionally, Pygame can integrate with real-time object detection models like YOLO, enabling the visualization of detected objects and the subsequent decision-making processes in a realistic environment. This helps in testing the responsiveness and accuracy of traffic management strategies under varying conditions.

### *5. Docker for Containerization*

As automated traffic management systems become more complex, ensuring portability, scalability, and ease of deployment becomes increasingly important. **Docker**, a platform for containerization, offers an effective solution by allowing developers to package applications and all their dependencies into a single container, ensuring consistent execution across different environments.

Using Docker, the entire **Intelligent Traffic Management System (ITMS)**—which includes object detection models like YOLO, traffic simulation systems like Pygame, and traffic signal control logic—can be encapsulated into isolated, self-contained environments. This makes the system portable across various platforms and devices, such as **edge devices**, **cloud environments**, or **on-premise servers**, without the need to worry about environment discrepancies.

Docker also simplifies deployment by enabling the application to run consistently across different development, testing, and production environments. Furthermore, Docker's support for **scalable deployments** through tools like **Kubernetes** allows the traffic management system to scale across a city or region, handling large volumes of real-time data and making intelligent decisions at the city level.

Docker also helps in versioning and maintaining consistency in the development process, enabling easy updates and rollbacks of traffic management systems. This level of flexibility and reliability is critical in creating automated traffic systems that can adapt to growing urban challenges.

In summary, the integration of **YOLO**, **Pygame**, and **Docker** in the development of **automated traffic systems** provides a powerful combination of real-time object detection, dynamic simulation, and scalable deployment. Together, these technologies offer an innovative solution to the challenges posed by urban traffic management, enabling cities to build more efficient, responsive, and intelligent transportation networks. This research paper explores the potential and synergies of these technologies, aiming to advance the field of automated traffic control and improve the flow of traffic in increasingly crowded urban environments.

## II. LITERATURE SURVEY

1. L. F. P. Oliveira, L. T. Manera, and P. D. G. Luz [1] proposed a centralized traffic light controller system with wireless communication to improve urban traffic management. The study highlighted the limitations of locally programmed traffic systems and demonstrated the feasibility of IoT-based solutions for smart cities.

2. Nicole Diaz, Jorge Guerra, and Juan Nicola [2] developed an autonomous traffic light control system that dynamically adjusts signal timings using IoT technology, including a Raspberry Pi and PIR sensors, to optimize urban traffic flow. The system supports future enhancements such as camera integration for improved traffic management.

3. Khaled Zaatouri and Tahar Ezzedine [3] proposed a self-adaptive traffic light control system using YOLO (You Only Look Once) for real-time traffic flow analysis. By optimizing signal phases based on queue lengths and waiting times, the system minimizes delays and enables efficient deep learning on embedded controllers via Transfer Learning.

4. Cheng-jian and Jyun-yu Jhang [4] developed an intelligent traffic-monitoring system using YOLO and convolutional fuzzy neural networks (CFNN) for real-time vehicle classification and counting. The system achieved high accuracy, including 90.45% on the Beijing dataset and 99% on the GRAM-RTM dataset, with detection rates exceeding 30 frames per second on the AGX platform.

5. Shuai Yang and Bin Wu [5] introduced a parallel approach for large-scale video data analysis using Spark. The study presented methods for feature extraction, clustering, and bag of features, demonstrating efficiency in video action detection and near-duplicate retrieval tasks.

6. Yash Desai, Yashowardhan Rungta, and Parth Reshamwala [6] proposed an automatic traffic management and surveillance system to reduce congestion and improve emergency response. The system integrates traffic signal management with a surveillance feature.

7. Thi-Thu-Trang Do, Tai-Huy Ha, and Kyungbaek Kim [7] designed a big data platform for real-time video surveillance, combining Kafka, Spark Structured Streaming, and tools like OpenCV and YOLO for intelligent object detection. A web interface supports interactive video analysis, demonstrating the platform's efficiency.

8. Seda Kul, Isabek Tashiev, Ali Sentas, and Ahmet Sayar [10] introduced an event-based microservices framework using Apache Kafka Streams for real-time vehicle detection based on type, color, and speed. The system enables efficient retrieval of video chunks, publishes vehicle attributes as events, and supports real-time querying without re-processing.

9. Jay Kreps, Neha Narkhede, and Jun Rao [11] presented Kafka, a distributed messaging system for high-volume log data processing. Kafka supports both offline and online message consumption with superior scalability and efficiency, processing hundreds of gigabytes of data daily in production environments.

10. Eman Shaikh, Yasmeen Alufaisan, Iman Mohiuddin, and Irum Nahvi [12] reviewed Apache Spark, a unified big data processing engine known for in-memory computation, batch and stream processing, and multithreading capabilities. The study highlighted Spark's architecture, ecosystem, and applications in emerging technologies.

11. Chapte Viresh M., Bhole Rahul Hiraman, and Karve Abhijeet C. [13] explored the use of Apache Kafka in big data stream processing, emphasizing its scalability, reliability, and high throughput for real-time data streaming and analytics.

12. Mihir M. Gandhi, Devansh S. Solanki, Rutwij S. Daptardar, and Nirmala Shinde Baloorkar [14] proposed a smart traffic control system using AI and live camera feeds to calculate real-time traffic density. The system dynamically adjusts traffic lights based on vehicle density, reducing congestion, pollution, and transit delays.

## III. PROPOSED SYSTEM

The development of an intelligent traffic management system (ITMS) involves optimizing traffic flow using real-time data from object detection algorithms like YOLO (You Only Look Once) for vehicle and pedestrian detection and Pygame for simulation and visualization. Docker containers are used to encapsulate the entire application, ensuring reproducibility, scalability, and ease of deployment. This methodology explains the steps required to integrate YOLO, Pygame, and Docker to create a robust, scalable traffic management system.
For each frame, YOLO identifies the objects and provides their class (e.g., car, pedestrian) and coordinates (bounding box).

### *3.1. Data Collection and Preprocessing for YOLO*

- **Traffic Dataset Collection:**
  - Obtain or create a dataset that includes images or videos of traffic scenarios (vehicles, pedestrians, traffic lights, etc.). Use publicly available datasets such as **KITTI**, **Cityscapes**, or **UA-DETRAC** for vehicle detection, or collect custom footage from cameras at intersections.
- **Preprocessing:**
  - Resize images to the appropriate input size for YOLO (e.g., 416x416 or 608x608).
  - Normalize pixel values (e.g., divide by 255).
  - Annotate the dataset with bounding boxes for vehicles, pedestrians, and any other relevant objects (e.g., traffic signs) in the YOLO format (class, x_center, y_center, width, height).

### *3.2. Training YOLO for Object Detection*

- **Model Selection:**
  - Choose a suitable YOLO version (e.g., YOLOv3, YOLOv4, or YOLOv5) based on accuracy and speed requirements.
  - Fine-tune a pre-trained YOLO model on the traffic dataset if you're working with limited data, or train from scratch for a more custom solution.
- **Training the Model:**

  - Use **Darknet** (the original framework for YOLO) or **PyTorch** for training the model.
  - Configure the model's hyperparameters (e.g., learning rate, batch size) and train on the dataset. Use metrics like **mAP (mean Average Precision)** to evaluate the model's performance.
- **Deployment of YOLO in the Docker Container:**
  - Include YOLO in the Docker container by installing necessary dependencies (e.g., OpenCV, PyTorch, TensorFlow, or Darknet).
  - Create a Python script or service inside the Docker container that performs inference on video streams or images captured from a camera or a simulation.

### *3.3. Traffic Simulation Using Pygame*

- **Simulation Setup:**
  Create a Pygame simulation that includes:
    - A 2D visualization of traffic lanes and intersections.
    - Moving vehicles that follow basic motion rules (acceleration, deceleration, lane changes).
    - Pedestrians that walk or wait at designated crossings.
    - Traffic lights that change based on traffic flow and pedestrian detection.
- **Vehicle and Pedestrian Behavior:**
  - Vehicles should move according to predefined rules (e.g., they stop at red lights, speed up on green, or slow down in traffic).
  - Pedestrians should wait for green lights or crosswalks to be active before crossing the street.
- **Interaction with YOLO:**
  - Integrate YOLO with Pygame: YOLO's object detection system will detect vehicles in the simulation, adjusting traffic light timings based on vehicle count.
  - Detect pedestrians in the simulation and trigger the traffic lights to accommodate pedestrian crossings.
  - For each frame in the simulation, pass the image through YOLO for detection and use this data to adjust traffic flow in Pygame (e.g., change light durations).

### *3.4. Dockerization of the System*

Creating the Docker Environment:
Install Dependencies:
Install Python, OpenCV, PyTorch (or TensorFlow), Pygame, and any other necessary libraries in the Docker container.
Install YOLO dependencies (e.g., Darknet or other YOLO implementations).
Dockerfile Creation:
Create a Dockerfile that defines the environment:
Dockerfile
Copy code

```
# Use a base image with Python and required libraries
FROM python:3.8-slim

# Install necessary system packages
RUN apt-get update && apt-get install -y \
    python3-opencv \
    libsm6 libxext6 libxrender-dev \
    && rm -rf /var/lib/apt/lists/*

# Install required Python libraries
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy the application code to the container
COPY . /app
WORKDIR /app

# Expose the port for visualization (if necessary)
EXPOSE 8080

# Run the Python script for simulation
CMD ["python", "traffic_management.py"]
```

Build and Run the Docker Container:

Build the Docker image:
bash
Copy code
```
docker build -t traffic-management-system .
```
Run the container:
bash
Copy code
```
docker run --rm -it --gpus all traffic-management-system
```
This command runs the container with GPU support if YOLO requires it (for real-time processing).

### *3.5. Testing and Deployment in Docker*

- **System Testing:**
  Test the integrated system within the container:
  - Ensure YOLO is detecting vehicles and pedestrians accurately.
  - Verify that the traffic signals in Pygame change based on the detected data.
  - Evaluate the performance of the simulation, including frame rates, accuracy, and real-time responsiveness.
- **Scaling the System:**
  - If deploying to multiple intersections or larger simulations, consider using **Docker Compose** to manage multiple containers (e.g., one for the object detection service and another for simulation).
  - Scale the system horizontally by deploying it on multiple machines using Docker Swarm or Kubernetes.

## YOLO Performance

**Accuracy:**
  - Evaluate YOLO’s precision and recall in detecting vehicles and pedestrians.
  - Test YOLO on various traffic scenarios (e.g., dense traffic, nighttime conditions) to assess robustness.

### *4.2. Traffic Management Efficiency*

- **Traffic Flow:**
  - Measure the average wait time at intersections.
  - Calculate vehicle throughput during peak and off-peak hours.
  - Analyze how dynamically the traffic lights adapt to changes in traffic flow and pedestrian presence.
- **Real-time Simulation Performance:**
  - Measure the frame rate of the Pygame simulation to ensure smooth operation. Aim for at least 30 frames per second (FPS) for fluid interaction.

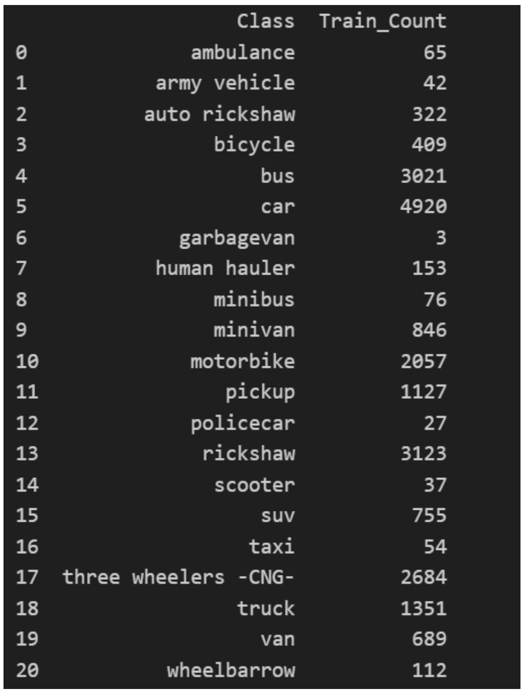
### *4.3. Docker Performance*

**Container Efficiency:**
  - Ensure the containerized system runs efficiently, utilizing system resources (CPU, GPU, RAM) appropriately.
  - Assess scalability by running multiple instances of the system and ensuring that the container can handle larger loads with minimal latency.

## 5. Challenges and Future Improvements

- **Model Accuracy in Varied Conditions:** YOLO might struggle under poor weather conditions, or with occlusions. Use data augmentation techniques and fine-tuning to improve robustness.
- **Real-time Performance in Large-Scale Simulations:** As the system scales to more intersections, optimizing traffic signal control and ensuring real-time performance may require more efficient algorithms and parallel computing.
- **Integration with Real Traffic Systems:** Transitioning from simulation to real-world deployment will require interfacing with actual traffic sensors and cameras

## IV. RESULT AND ANALYSIS



```
                    Class  Train_Count
0               ambulance           65
1            army vehicle           42
2           auto rickshaw          322
3                 bicycle          409
4                     bus         3021
5                     car         4920
6               garbagevan           3
7            human hauler          153
8                 minibus           76
9                 minivan          846
10              motorbike         2057
11                 pickup         1127
12              policecar           27
13               rickshaw         3123
14                scooter           37
15                    suv          755
16                   taxi           54
17  three wheelers -CNG-         2684
18                  truck         1351
19                    van          689
20            wheelbarrow          112
```

**Fig.1. Classes Representation**

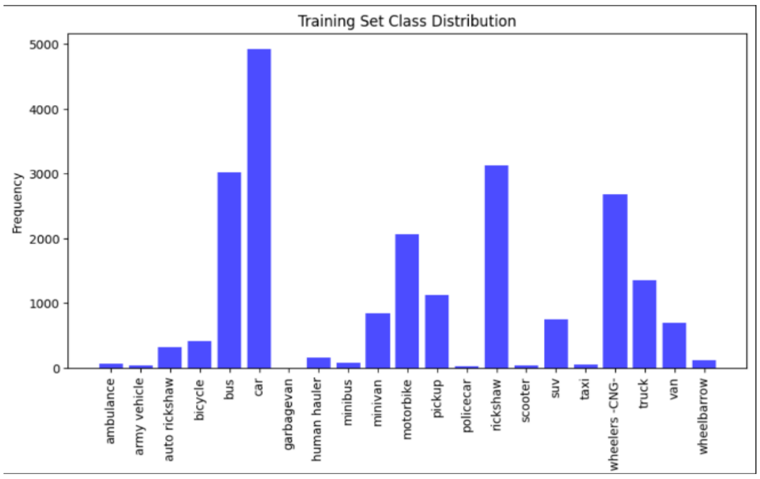


**Fig.2. Representation of Training Data**

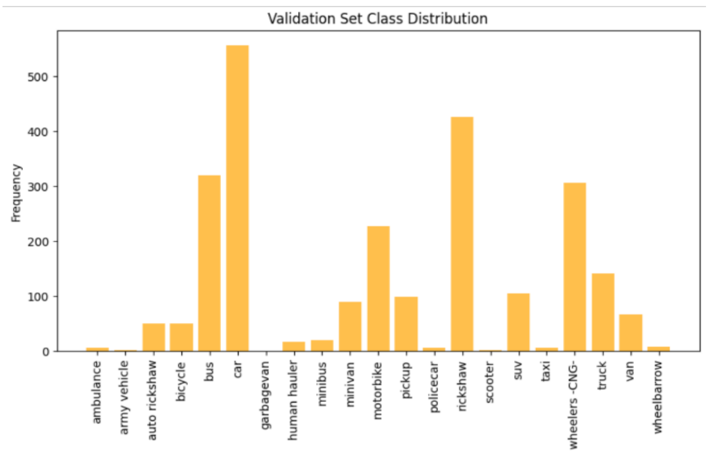


**Fig.3. Representation of Validation Data**

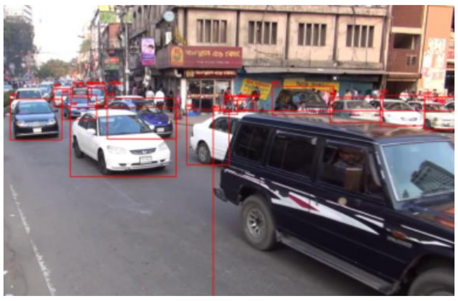
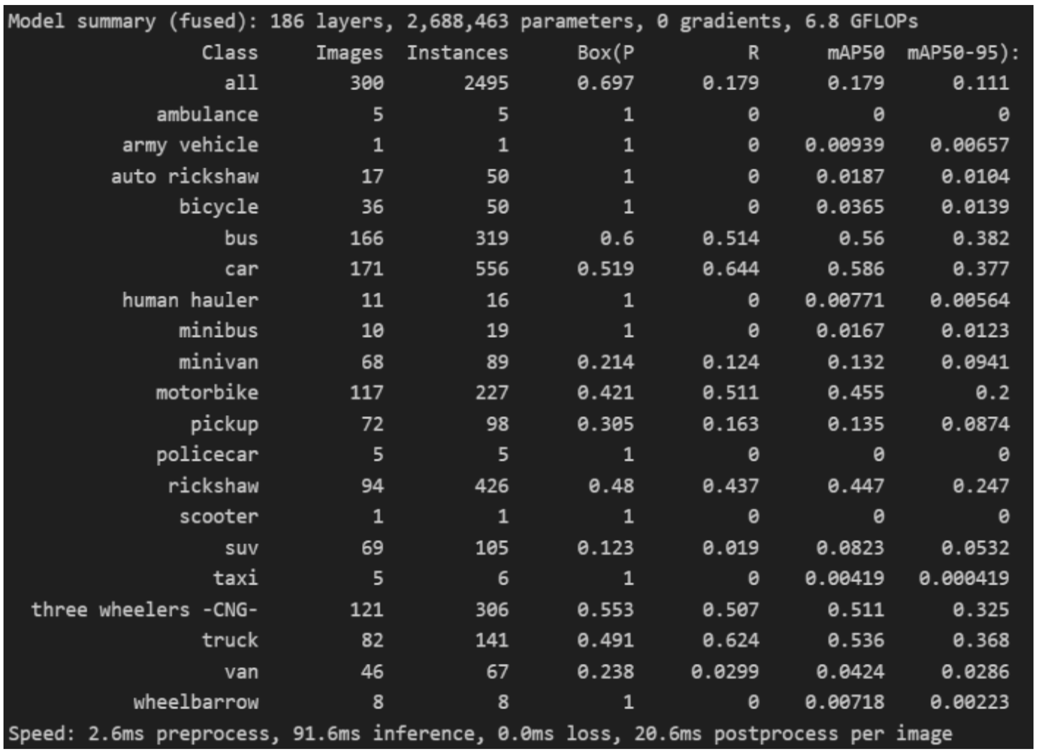
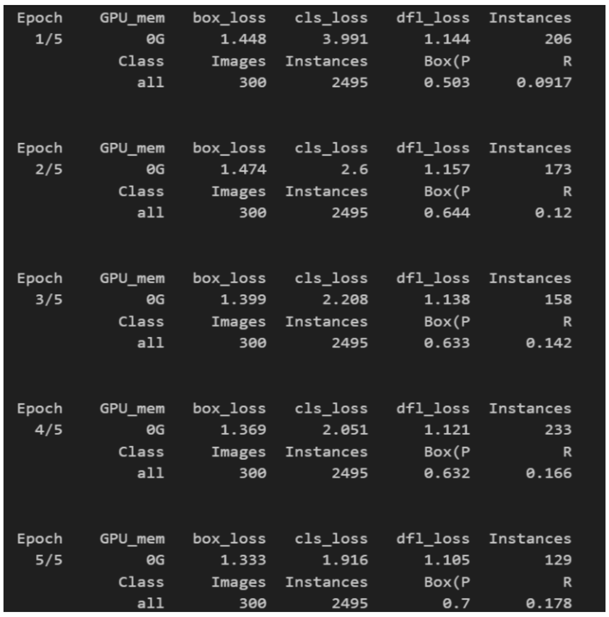


**Fig.4. Sample Validation Image**



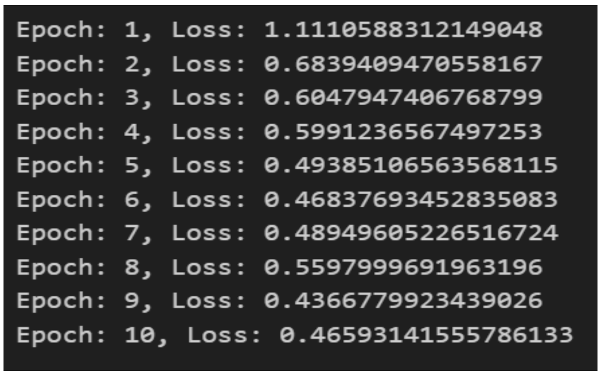
```
Model summary (fused): 186 layers, 2,688,463 parameters, 0 gradients, 6.8 GFLOPs
                   Class     Images  Instances      Box(P          R      mAP50  mAP50-95):
                     all        300       2495      0.697      0.179      0.179      0.111
               ambulance          5          5          1          0          0          0
            army vehicle          1          1          1          0    0.00939    0.00657
           auto rickshaw         17         50          1          0     0.0187     0.0104
                 bicycle         36         50          1          0     0.0365     0.0139
                     bus        166        319        0.6      0.514       0.56      0.382
                     car        171        556      0.519      0.644      0.586      0.377
            human hauler         11         16          1          0    0.00771    0.00564
                 minibus         10         19          1          0     0.0167     0.0123
                 minivan         68         89      0.214      0.124      0.132     0.0941
               motorbike        117        227      0.421      0.511      0.455        0.2
                  pickup         72         98      0.305      0.163      0.135     0.0874
               policecar          5          5          1          0          0          0
                rickshaw         94        426       0.48      0.437      0.447      0.247
                 scooter          1          1          1          0          0          0
                     suv         69        105      0.123      0.019     0.0823     0.0532
                    taxi          5          6          1          0    0.00419   0.000419
   three wheelers -CNG-        121        306      0.553      0.507      0.511      0.325
                   truck         82        141      0.491      0.624      0.536      0.368
                     van         46         67      0.238     0.0299     0.0424     0.0286
             wheelbarrow          8          8          1          0    0.00718    0.00223
Speed: 2.6ms preprocess, 91.6ms inference, 0.0ms loss, 20.6ms postprocess per image
```

**Fig.5. Training and Testing Data**



```
Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances
  1/5         0G      1.448      3.991      1.144        206
               Class     Images  Instances      Box(P          R
                 all        300       2495      0.503     0.0917


Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances
  2/5         0G      1.474        2.6      1.157        173
               Class     Images  Instances      Box(P          R
                 all        300       2495      0.644       0.12


Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances
  3/5         0G      1.399      2.208      1.138        158
               Class     Images  Instances      Box(P          R
                 all        300       2495      0.633      0.142


Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances
  4/5         0G      1.369      2.051      1.121        233
               Class     Images  Instances      Box(P          R
                 all        300       2495      0.632      0.166


Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances
  5/5         0G      1.333      1.916      1.105        129
               Class     Images  Instances      Box(P          R
                 all        300       2495        0.7      0.178
```

**Fig.6. Analysis Images Using YOLO**



```
Epoch: 1, Loss: 1.1110588312149048
Epoch: 2, Loss: 0.6839409470558167
Epoch: 3, Loss: 0.6047947406768799
Epoch: 4, Loss: 0.5991236567497253
Epoch: 5, Loss: 0.49385106563568115
Epoch: 6, Loss: 0.46837693452835083
Epoch: 7, Loss: 0.48949605226516724
Epoch: 8, Loss: 0.5597999691963196
Epoch: 9, Loss: 0.4366779923439026
Epoch: 10, Loss: 0.46593141555786133
```

**Fig.7. Analysis Images using CNN**

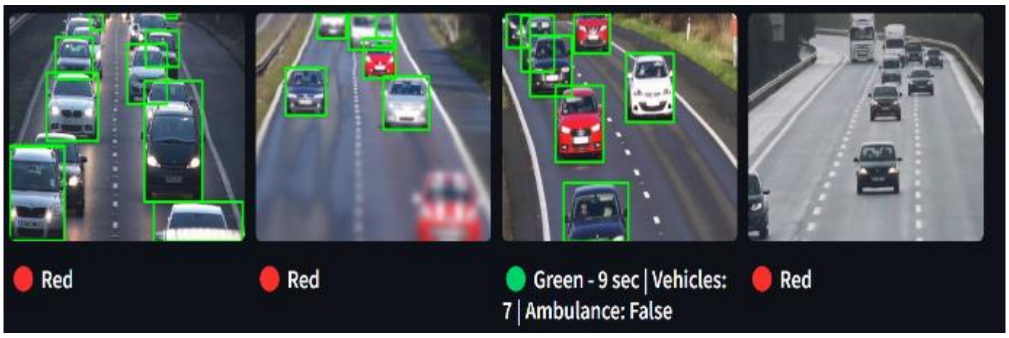


**Fig.8. Time updating for less number of vehicles**

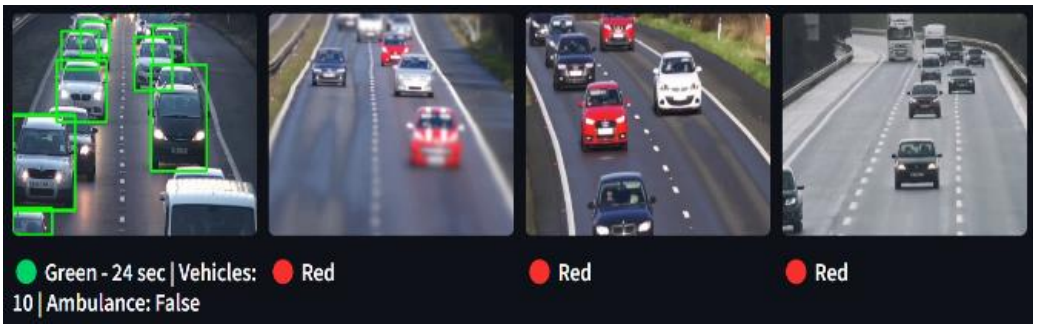


**Fig.9. Time updating for more number of vehicles**

## V. CONCLUSION

This methodology outlines how to integrate **YOLO** for object detection, **Pygame** for traffic simulation, and **Docker** for containerization to develop a robust Intelligent Traffic Management System (ITMS). The use of **Docker** ensures that the system is portable, scalable, and reproducible, making it easier to deploy across various platforms. By leveraging **YOLO**, the system performs real-time object detection, such as identifying vehicles and pedestrians in urban environments. **Pygame** provides the simulation aspect, where traffic flow, signal adjustments, and pedestrian movements are visualized and managed. Together, these components work seamlessly to optimize traffic flow, reduce congestion, and improve safety.

However, for real-world applicability and scalability, there is a need for more advanced system features and the ability to scale across a wide network of intersections and devices. This is where **orchestration** in a distributed network comes into play. The following sections describe the future scope of this intelligent traffic management system, focusing on how orchestration can enable the deployment and coordination of multiple ITMS components in a distributed network.

## REFERENCES

**[1]** Y. Cui and D. Lei, "Design of Highway Intelligent Transportation System Based on the Internet of Things and Artificial Intelligence," (2023), IEEE Access, vol. 11, pp. 46653-46664.

**[2]** M. Driss Laanaoui, M. Lachgar, H. Mohamed, H. Hamid, S. Gracia Villar and I. Ashraf, "Enhancing Urban Traffic Management Through Real-Time Anomaly Detection and Load Balancing," (2024), IEEE Access, vol. 12, pp. 63683-63700.

**[3]** T. Syum Gebre, L. Beni, E. Tsehaye Wasehun and F. Elikem Dorbu, "AI-Integrated Traffic Information System: A Synergistic Approach of Physics Informed Neural Network and GPT-4 for Traffic Estimation and Real-Time Assistance," (2024), IEEE Access, vol. 12, pp. 65869-65882.

**[4]** R. Manasw V et al., "Traffic Light Detection for Information Systems and Telecommunications Using CNN," ICMPC, 2023.

**[5]** S. M. Mortazavi et al., "Smart Control of Traffic Lights Based on Traffic Density in a Multi-Intersection Network Using Q-Learning," Springer, 2023.

**[6]** S. Kul et al., "Event-Based Microservices with Apache Kafka Streams: A Real-Time Vehicle Detection System Based on Type, Color, and Speed Attributes," IEEE, 2021.

**[7]** K. Zaatouri et al., "A Self-Adaptive Traffic Light Control System Based on YOLO," IEEE, 2018.

**[8]** C. Jian et al., "Intelligent Traffic-Monitoring System Based on YOLO and Convolutional Fuzzy Neural Networks," IEEE, 2022.

**[9]** S. Yang and B. Wu, "Large-Scale Video Data Analysis Based on Spark," IEEE, 2015.

**[10]** Y. Desai et al., "Automatic Traffic Management and Surveillance System," IEEE, 2020.

**[11]** T. T. T. Do et al., "A Big Data Platform for Real-Time Video Surveillance,"PTI, 2022.

**[12]** L. F. P. Oliveira et al., "Smart Traffic Light Controller System," IEEE, 2019.

**[13]** N. Diaz et al., "Smart Traffic Light Control System," IEEE, 2018.

**[14]** J. Kreps et al., "Kafka: A Distributed Messaging System for Log Processing," Corpus, 2011.