

COMPUTER VISION PROJECT

Intelligent Traffic Monitoring System: A Comparative Study of Classical and Deep Learning Approaches Group No- 11



Name :-

Aryan Kumar M23CSA510

Abhishek Sahu M23CSA504

Rohan Frederick M23CSA525

Chandra Mohan Singh Negi M23CSA512

Rahul Singh M23CSA522

**Git : <https://github.com/Aryank47/Intelligent-traffic-monitoring-system.git>
Deployment Link : <https://intelligent-traffic-monitoring-system.streamlit.app/>**

Table of Contents

1. Introduction.....	1
2.Role of Each User.....	3
3. Dataset Description	3
4. Approach 1: HOG + SVM (Classical Machine Learning).....	4
4.2 Brief Overview	5
4.3 Architecture.....	6
5 Approach 2:- Haar Cascade Classifier (Classical Method).....	9
5.1 Brief Overview.....	10
5.2 Architecture	11
6.Approach 3 :-R-CNN (Deep Learning).....	12.
6.1 Brief Overview.....	12
6.2 System Flow and Acrhitecture	14
7. Overall All Implemenatation Details.....	16
8.Evaluation Metrics	16
9 .Visual Results	17
10.Comparison	18
11. Conclusion	19
12. References.....	20

Intelligent Traffic Monitoring System: A Comparative Study of Classical and Deep Learning Approaches

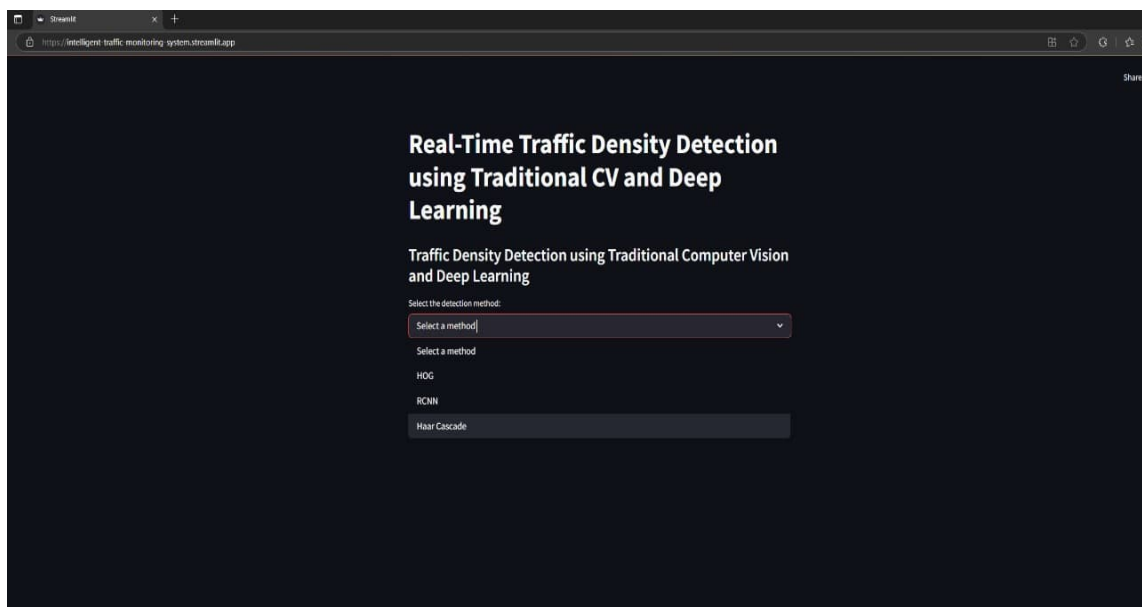
1. Objective

This report presents the design, implementation, and comparison of three vehicle detection and traffic density estimation systems. The objectives are:

- To implement a lightweight and explainable traffic monitoring system using HOG + SVM.
- To implement alternative techniques like HAAR - Cascade
- To explore a deep learning-based approach using Region-based Convolutional Neural Networks (R-CNN).
- To analyze performance trade-offs between the two.

2. Introduction

Urban congestion is a persistent issue that demands intelligent solutions. Vision-based systems provide an effective, scalable alternative to traditional traffic monitoring. This study compares a classical ML pipeline (HOG + SVM) and a modern deep learning pipeline (R-CNN) on the same task. Techniques such as Haar Cascade were also explored.



Role of each user:-

Aryan Kumar: was primarily responsible for designing and implementing the vehicle detection pipeline. He developed the HOG feature extraction method, optimized the sliding window approach for image-based detection, and implemented the non-maximum suppression algorithm to improve detection accuracy. He also ensured seamless integration of pre-processing steps like image resizing and contrast enhancement for better feature extraction. Furthermore, Aryan contributed to refining the overall detection logic and ensuring the robustness of the single-image detection workflow.

Chandra Mohan Singh Negi: led the efforts in dataset preparation, model training, and video stream processing. He implemented the background subtraction technique for real-time vehicle detection in videos and handled the data preprocessing pipeline, including grayscale conversion and feature scaling. Additionally, he developed the visualization components using Streamlit, ensuring an interactive user interface for both image and video modes. Chandra also contributed to performance optimization and validation of the trained SVM model.

Abhishek Sahu: Developed and deployed a vehicle detection system using Haar Cascade classifiers in OpenCV. Managed the full pipeline including dataset collection, annotation, .vec file creation, and cascade training with OpenCV tools. Tuned parameters to improve detection accuracy and reduce false positives. Integrated the trained model into a real-time application to detect and track vehicles in video streams, with emphasis on performance and reliability. Automated parts of the workflow to streamline dataset preparation and training.

Rohan Frederick: Worked on vehicle detection system using the UA-DETRAC dataset, converting TXT annotations for compatibility with PyTorch detection models. Trained a Faster R-CNN model, then explored further optimization using TorchScript, AMP, and Quantization-Aware Training (QAT). Addressed performance bottlenecks, scripted models for inference.

Rahul Singh: Explored and analysed SSDLite320_MobileNetV3_Large lightweight model for resolving performance bottleneck. Integrated the code with overall streamlit application. The

model evaluation script IoU, precision, and recall was built and executed. Also worked on architectural differences between Fast R-CNN and our final MobileNet-based model.

3. Dataset Description

- **Source:** Video From Kaggle
- **Dataset Composition:**
 - Positive Samples: Cars, trucks, vans.
 - Negative Samples: Roads, vegetation, buildings.
- **Annotations:** For R-CNN

4. Approach 1: HOG + SVM (Classical Machine Learning)

4.1 Brief Overview

Histogram of Oriented Gradients (HOG) is a feature extraction technique that captures edge and shape information by computing gradient orientations in localized image regions. It is especially effective for detecting objects like vehicles or pedestrians.

Support Vector Machine (SVM) is a supervised classification algorithm that finds the optimal boundary (hyperplane) between different classes—in this case, vehicles vs. non-vehicles.

Together: HOG extracts features from an image, and SVM classifies those features to detect the presence of vehicles.

4.2 System Overview

- Feature extraction: HOG.
- Classifier: Linear SVM.
- Detection: Multi-scale sliding window (static images), background subtraction (video).

4.3 Implementation Workflow

1. **Preprocessing:** CLAHE for contrast enhancement.
2. **Feature Extraction:** HOG descriptors.
3. **Model Training:** Linear SVM trained using scikit-learn.
4. **Detection:** Sliding-window on scaled images.
5. **Traffic Density Estimation:**

- Low: < 3 vehicles.
- Medium: 3–5.
- High: ≥ 6 .

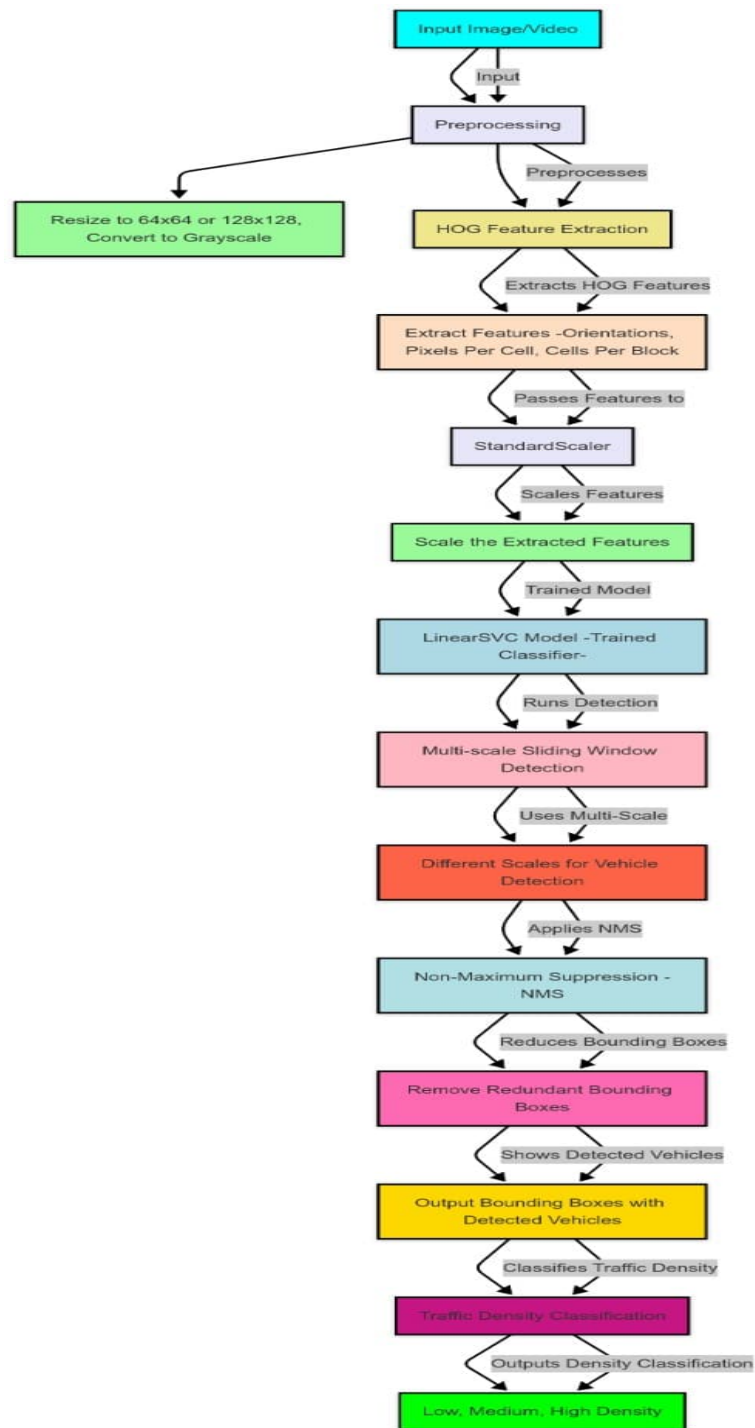
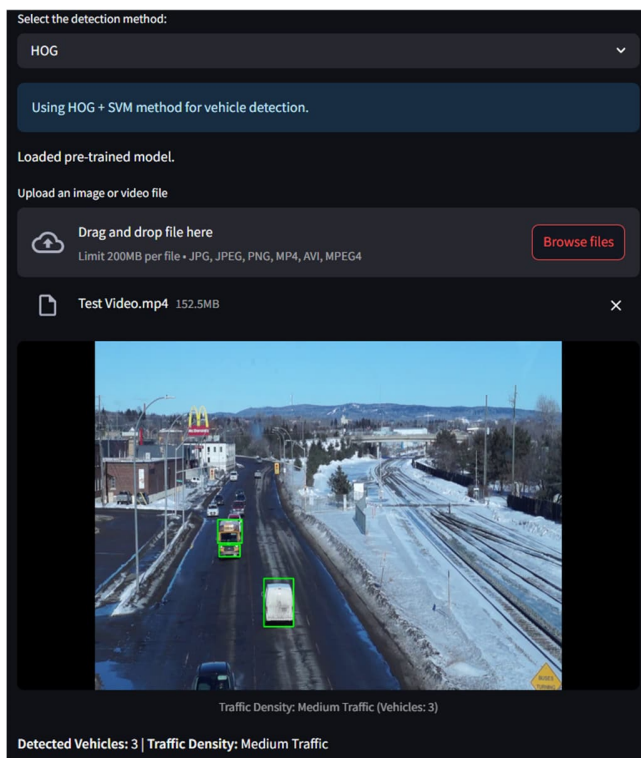
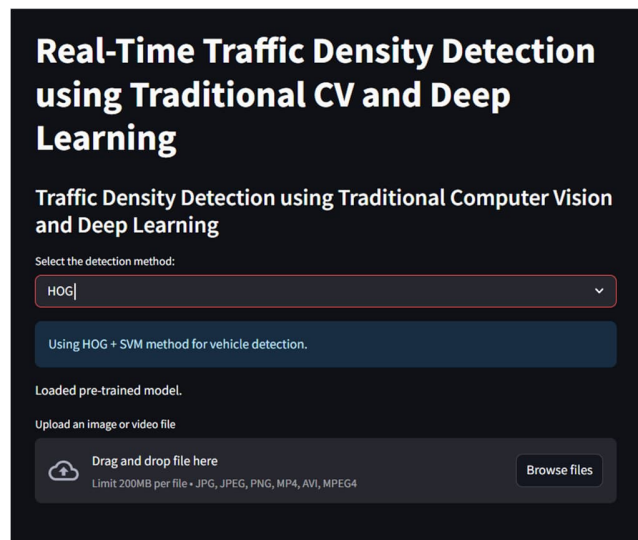




Fig:- Detection method by HOG+SVM for pedestrian detection, vehicle detection

4.4 Streamlit Interface



- Image/video upload.
- Real-time bounding box visualization.
- Traffic density status.

4.6 Advantages & Limitations

-  Interpretable and fast.
-  High false positive rate on cluttered scenes.

5. Approach 2: Haar Cascade Classifier (Classical Method)

Overview

A machine learning-based approach where a cascade function is trained from many positive and negative images to detect objects like vehicles.

Pipeline

1. **Training:** Pre-trained Haar features (rectangle features) using AdaBoost algorithm.
2. **Detection:** The classifier scans the input frame and identifies areas of interest based on feature matching.

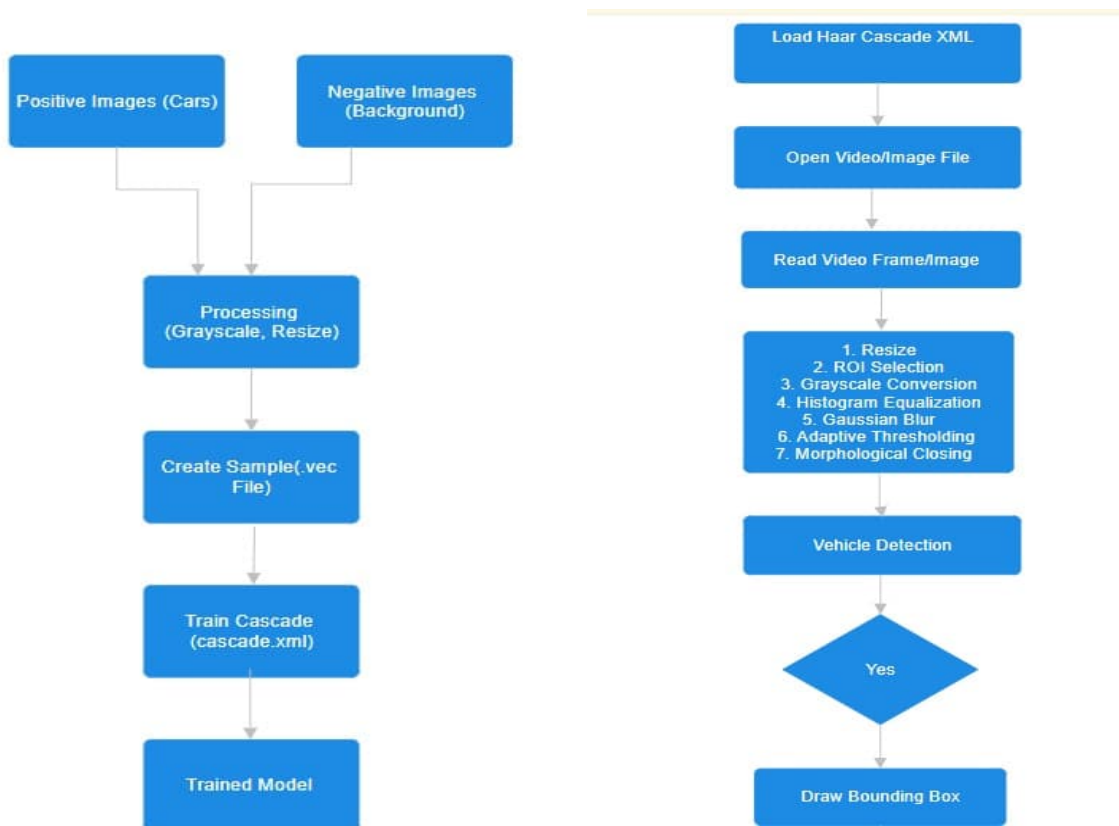


Figure 2:- Architecture Diagram Of Haar Cascade Classifier

Key Concepts of Haar Cascade Classifier:

1. Haar-like Features:

- Haar features are similar to convolution filters and are used to detect edges, lines, and textures.
- **Common feature types:**
 - Edge features: e.g., a white rectangle next to a black one.

- Line features: e.g., three rectangles in a row.
- These features are applied over sub-regions of the image to extract information.

2. Integral Image:

- To speed up the feature computation, the image is transformed into an integral image (also called summed-area table).
- The integral image allows rapid calculation of the sum of pixel values in a rectangular area, regardless of size.

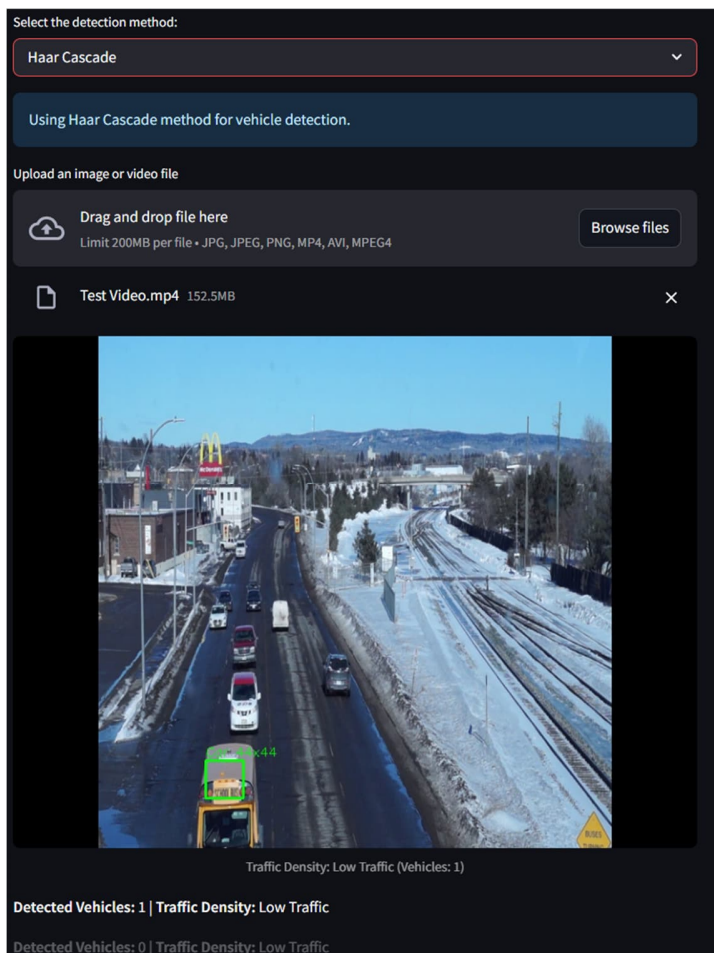
3. Adaboost Learning Algorithm:

- Not all Haar features are useful—Adaboost selects the most relevant features and combines them into a strong classifier.
- It also assigns higher weights to misclassified samples during training.

4. Cascade of Classifiers:

- Rather than applying all features to every region, a cascade of classifiers is used:
 - Early stages quickly eliminate non-object regions.
 - Later stages apply more complex classifiers to refine detection.
- This cascade structure improves speed without much loss in accuracy.

5.1 Streamlit Interface

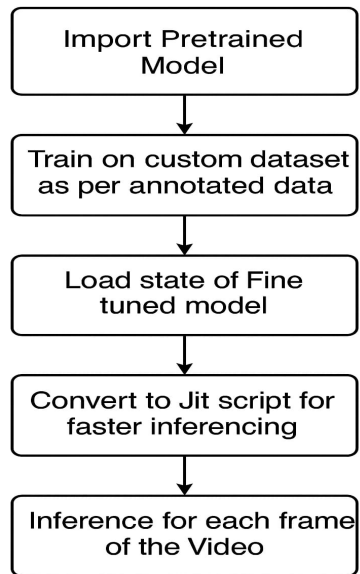


6. Approach 3 :-R-CNN (Deep Learning)

6.1 Brief Overview

R-CNN (Region-based Convolutional Neural Network) is a deep learning object detection framework. For this project FastRCNN was implemented and fine tuned on UA-DETRAC dataset available on - <https://www.kaggle.com/dtrnngc/ua-detrac-dataset>. However due to latency issues a student model **fasterrcnn_mobilenet** was fine tuned instead and used for inferencing. An SSDLite model was tried with extensive training but failed to provide good enough accuracy even though it was much faster to inference.

6.2 System flow



6.3 Training Details

- Pretrained model (transfer learning) fine-tuned on custom dataset
- Binary class used for vehicle identification 0 – background, 1 - vehicle
- Roboflow used for data annotation and subsequently used for training
- All training done for RCNN on PyTorch
- Mobile net model (145 MB footprint) preferred over Fast RCNN (315 MB footprint) due to latency issues

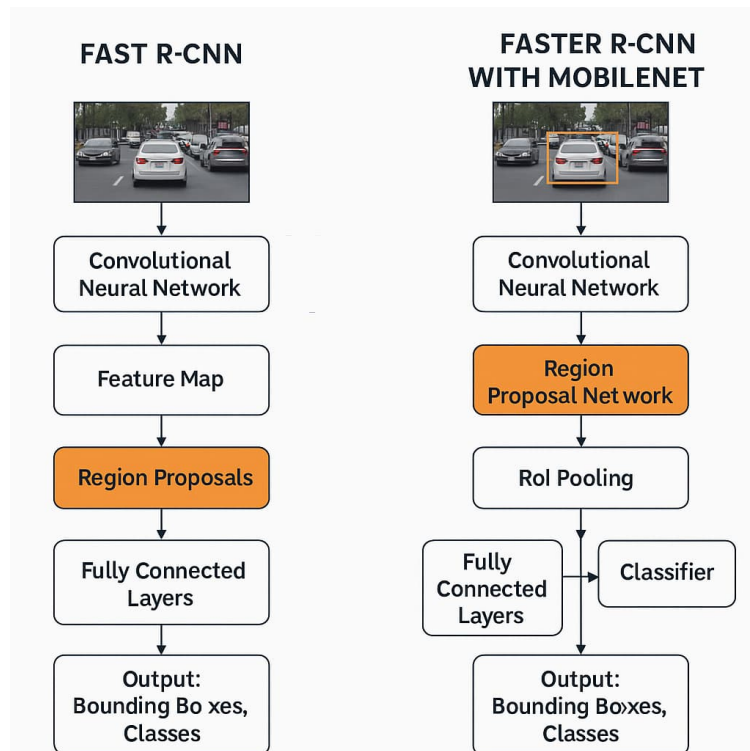


Fig:- Comparison for faster R-CNN with mobile net model.

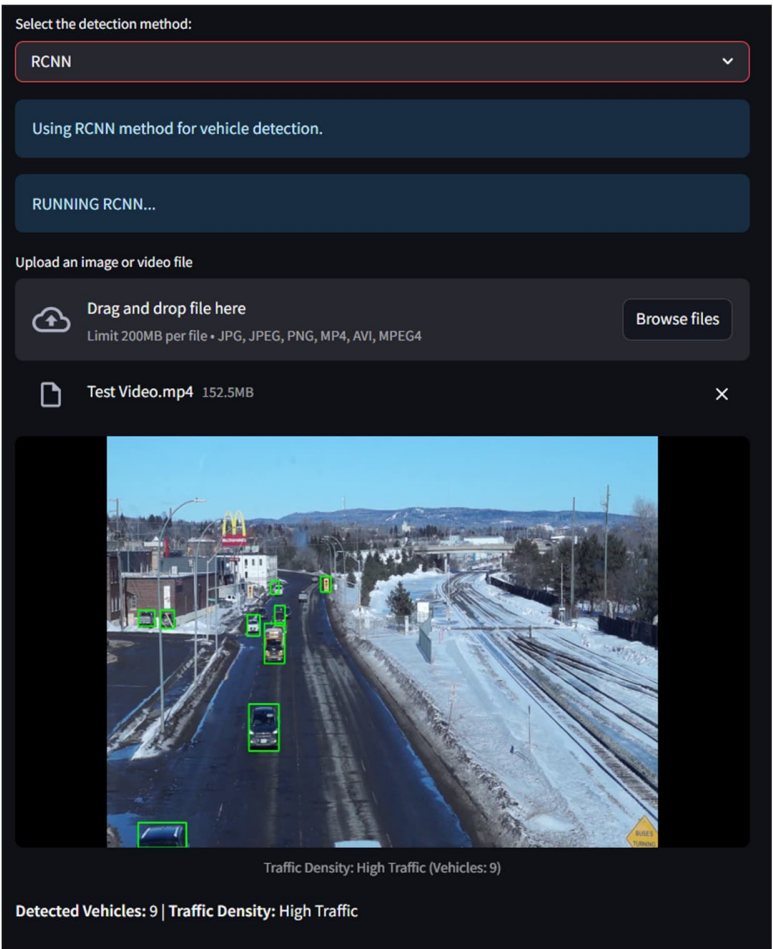
6.4 Known issues:-

- Non moving i.e. static vehicles also detected
- Eventhough mobile net is faster but still slow for a good realtime user friendly response
- Distant vehicles detected as an aggregate for the current point in time traffic count

6.5 Traffic Density Estimation

- Count number of bounding boxes per frame.
- Low: < 3 vehicles.
- Medium: 3–5.
- High: ≥ 6 .

6.6 Streamlit Interface



7. Over all Implementation Details

Component	HOG + SVM	Haar Cascade Classifier	R-CNN
Feature Extraction	Histogram of Oriented Gradients	Haar-like Features (Rectangular filters)	CNN (ResNet, VGG)
Classifier	Linear SVM	AdaBoost (Cascade of Weak Classifiers)	Fully connected Neural Network
Input Type	Grayscale image patches	Grayscale images	RGB frames
Region Proposal	Sliding Window	Cascade-based Region Scanning	Selective Search
Framework	scikit-learn, OpenCV	OpenCV	TensorFlow, PyTorch

8. Evaluation Metrics

Metric	HOG + SVM	Haar Cascade Classifier	R-CNN
Accuracy	95.47%	81.20%	93.88%
Precision	96.44%	82.00%	93.70%
Recall	94.29%	86.70%	99.00%
F1 Score	95.35%	85.80%	96.84%

9. Visual Results

HOG + SVM

- Bounding boxes over vehicles.
- False positives near complex backgrounds.

HAAR – Cascade

- Bounding boxes over vehicles with size (e.g. 43X43).
- False positives near backgrounds and sometime not detecting the vehicle

R-CNN

- Tighter, more accurate boxes.
- Detects partially occluded or small vehicles better.

10. Comparison

Criteria	HOG + SVM	Haar Cascade Classifier	R-CNN
Simplicity	High	Very High	Moderate
Accuracy	Moderate–High	Moderate	Very High
Hardware Requirements	Low	Very Low	GPU Preferred
Training Time	Short	Very Short	Long
Real-Time Capability	Limited	High	Feasible (on GPU)
Interpretability	High	Moderate	Low

11. Limitations

- **HOG + SVM:**
 - Struggles with occlusion and lighting.
 - Sliding window increases inference time.
- **Haar Cascade Classifier:**
 - High false positives in complex scenes.
 - Less accurate for occluded or small vehicles.
- **R-CNN:**
 - Requires large, annotated dataset.
 - Slower compared to modern detectors (e.g., YOLO, SSD).

12. Future Enhancements

- Replace R-CNN with Faster R-CNN
- Use Kalman Filters or Deep SORT for vehicle tracking.
- Implement At Streamlit and VSCODE , collab
- Add license plate recognition module.

13. Conclusion This report demonstrates the trade-offs between a traditional and deep learning-based vehicle detection pipeline. While HOG + SVM offers simplicity and transparency, R-CNN provides superior accuracy and robustness. Depending on deployment constraints (hardware, real-time needs, accuracy), either approach can be adopted or hybridized for a production-ready intelligent traffic monitoring solution.

13. References

1. Dalal & Triggs, 2005. "Histograms of Oriented Gradients."
2. Girshick et al., 2014. "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation."
3. Udacity Self-Driving Car Dataset.
4. KITTI Vision Benchmark Suite.
5. OpenCV, scikit-learn, TensorFlow, PyTorch official documentation