The Analysis of Deep Learning Based Vehicle Classification, Tracking and Speed Estimation Systems

Arkadeepto Majumder, Chaitanya Anand
Mentor: Prof. Ardhendu Kundu
Department of Computer Science and Engineering, Institute of Engineering & Management

Abstract

Traffic congestion and traffic accidents are becoming social problems with the increase in traffic volume due to rapid advancements of technology and increasing urban population. Computer Vision is a useful tool to get information regarding such traffic incidents, however, vehicle detection and tracking based on camera is highly dependent on the environmental factors and technological factors and thus they can be inefficient. Therefore, these varying environmental factors and technological limitations have given way to deep learning based algorithms used for object detections that are employed in vehicle classification and tracking, along with speed detection in order to classify and track vehicles in complex and diverse environments. The rapid increase of computational capability of modern devices have lead to a widespread use of these deep learning techniques such as CNNs, RNNs, Siamese networks, and object detection algorithms like YOLO and Faster R-CNN, etc to enhance traffic safety and efficiency

1 Introduction

As the urban population increases exponentially coupled with rapid technological advancements, the number of vehicles has increased significantly and the capacity of existing transportation systems are stretched thin causing severe traffic congestions in many highly populated cities with inefficient transport systems [1]. This situation is not one that can be solved by simply building more highways and subways, as not only the resources for such infrastructural development are scarce, but at the same time, these steps have been proven to be moot in solving this problem of traffic management [2].

An effective alternative to deal with traffic congestion is a traffic management and monitoring system that is a fully automated system to collect traffic data such as the number of vehicles, type of vehicles and vehicle speed. The collection of these data is important for such traffic management system to perform its functions which is to perform analysis on the traffic to better utilize the roadway systems and facilitate the safety of transportation [3]. The goal of such automated traffic surveillance system is to remove the need for human labour for simple vision tasks that can be performed efficiently by computer systems.

Currently, various data collection and processing technologies in the field of Internet of Things, a lot of traffic monitoring cameras have been placed at congestion prone areas of intersections, highways, etc [4]. However, most of the monitoring systems still work in the tradition way of collecting raw video data and transmitting and storing it. This video data is then analysed manually that presents a significant problem due to the sheer volume of the data that need to be processed and analysed [5]. To tackle this inefficiency, there are various machine learning algorithms that have been developed to extract such data from the video data without manual intervention.

With great progress of in the field of machine learning and artificial intelligence(AI), various methods were proposed to tackle the problem of vehicle tracking and classification. The traditional vehicle detection and classification models were mainly based on the following methods: i) Scale Invariant Feature Transform (SIFT) for feature matching and extraction; ii) Gaussian Mixture model for moving vehicle detection; iii) License Plate extraction method; iv) Histogram of Oriented Gradient(HOG) and Support Vector Machine(SVM) [6]. Modern methods of tracking and classification are mostly based of deep learning techniques and models such as Convolutional Neural Networks (CNNs), Recurrent

Convolutional Neural Networks (RCNNs), You Only Look Once (YOLO), etc. This survey aims at analysing the existing methods and models used to achieve the goal of tracking and classification of vehicles.

2 Literature Survey

2.1 Traditional Methods (Computer Vision, GMMs, SVMs)

Tarun Kumar et al.[7] propose an elementary but efficient approach for detection and speed estimation for moving vehicles using a single point CCTV camera using image processing. The proposed approach is implemented by using OpenCV and JavaCV with MySQL being used as database. Although the calibration of camera is strict to suit this model of detection, this flaw can be easily handled through modern tools and models. Regardless, the innovation of this approach lies in the selection of Region of Interest for vehicle detection. The accuracy of detection through this proposed method was found to be 87.7% with the maximum accuracy found to be 98.3%.

Chen et al. focus on effectively capturing a car image from video footage cite14. The authors adopt the background Gaussian Mixture Model (GMM) and the shadow removal algorithm [8] to reduce the negative impacts on vehicle classification caused by shadow, camera vibration, illumination changes, etc. The Kalman filter [9] is used for vehicle tracking and SVM is used to perform vehicle classification. Experiments were performed with real video footage obtained from cameras deployed in Kingston upon Thames, UK. Vehicles were classified into five categories, i.e., motorcycles, cars, vans, buses, and unknown vehicles. The classification accuracy for these vehicle types was 94.6%. We utilised a Gaussian Mixture model in *scikit-learn* library for image segmentation of a frame in sample video. In the appendix (see Appendix 8.1), you can find the code used for generating the results.

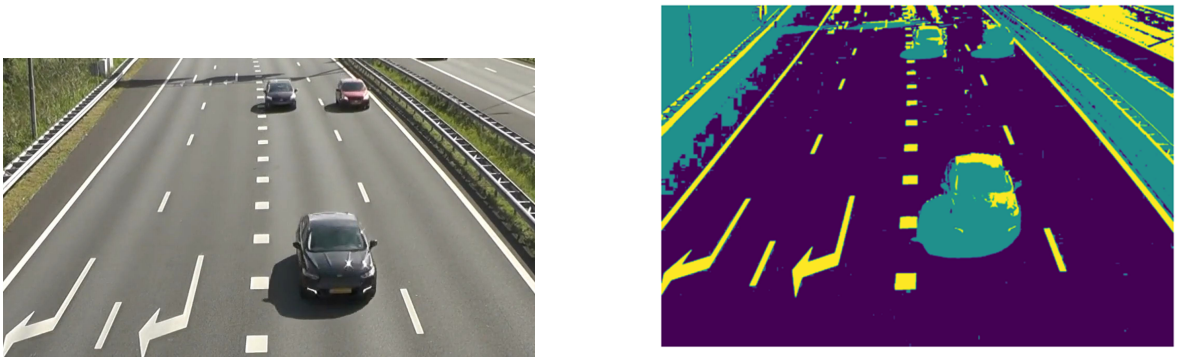


Figure 1: Object Segmentation using Gaussian Mixture Algorithm Model.

2.2 Deep Learning Algorithms (CNNs, RCNNs)

Minglan Sheng et al. [6] suggests one approach is to use convolutional neural networks (CNNs) for feature extraction and classification. CNNs have been shown to be effective in identifying vehicle types from different angles and scenes. However, there are still challenges in accurately identifying vehicles in real traffic situations, such as low image definition, deformation due to large aspect ratios, and imbalanced training datasets. To address these challenges, researchers have explored different techniques, including the use of depth learning for vehicle positioning and classification. Other methods have focused on specific features, such as the outline, windshield, rear-view mirror, and license plate, to classify vehicle types. These traditional methods, however, have limitations in terms of recognition accuracy. Therefore, there is a need for further research to improve the accuracy and robustness of vehicle detection and classification methods.

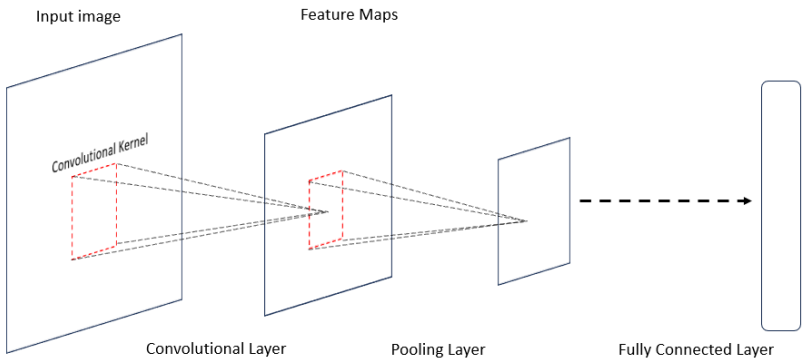


Figure 2: Architecture of a general CNN model.

However, region based CNNs were computationally expensive as originally developed, their cost has been drastically reduced with the introduction of new methods such as Fast R-CNN [10]. Shaoqing Ren et al. [11] proposes a method to unify the fast R-CNNs and RCNs into a fully–convolutional network (FCN) through alternative training scheme between fine-tuning for the region proposal task and then fine-tuning for the object detection, while keeping the proposals fixed. This scheme is used to build the unified convolutional network which was tested against the PASCAL VOC detection benchmarks [7] and it resulted in improvement in region proposal quality and thus object detection accuracy.

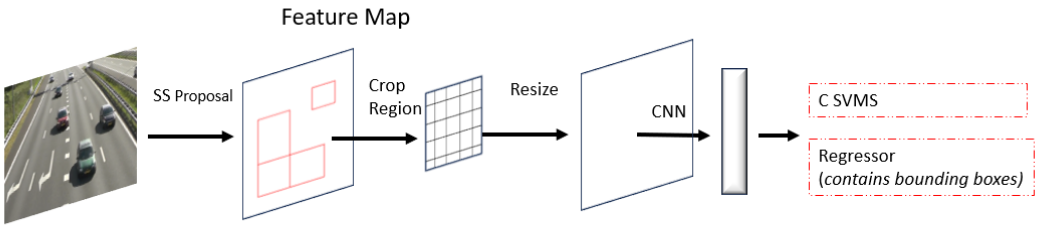


Figure 3: Basic Steps involved in Object Detection using RCNN Model.

A major challenge for applying machine learning algorithms for automated background filter in image processing and feature extraction is that different parts of the vehivle are treated without distinction, which degrades their performance. Zhao et al. focus on this problem that potentially misses the key part of a car image [12]. Their work is motivated by the human vision system that distinguishes the key parts of an image from the background, which is called the multiglimpse and visual attention mechanism. The key idea of their work is thus to exploit the visual attention mechanism to generate a focused image first and provide the image as input to CNN for more accurate vehicle classification. They performed experiments to classify a vehicle into five types, sedans, vans, trucks, SUVs, and coaches, and achieved the classification accuracy of 97.9%.

This [13] paper proposes a framework for predicting driver behavior in dilemma zones at signalized intersections using machine learning techniques. The study collected vehicle attribute data at the onset of the yellow indication and developed prediction models using Support Vector Machine (SVM) and Artificial Neural Network (ANN). The models achieved high prediction accuracies and were evaluated based on various performance measures. The study found that the start and end points of the dilemma

zone were influenced by approaching speeds and varied by time of day. The proposed framework shows potential for improving intersection signal operations and reducing red-light violations. The findings of this study contribute to the existing research on driver behavior in the dilemma zone and highlight the need for further research to enhance traffic safety and signal timing strategies.

2.3 Modern Tools and Models (Tensorflow, DeepSORT, YOLO)

Modern object detection models can be broadly classified on the basis of the number of passes that are made in order to detect objects in the frame. There are one-pass models and two-pass models. Two-pass models are more computationally expensive than their one-pass counterparts. Modern two-pass models include R-CNN, Fast R-CNN, Faster R-CNN and RCFN. Modern one-pass models include YOLO, YOLOv2 and SSD.

Girshick et al. proposed Fast R-CNN[14] to address the limitations of RCNN . Fast R-CNN computed a feature map for the whole image and extracted fixed length region features on the feature map. But unlike R-CNN and other two-pass models like SPP-net [15, 16], Fast R-CNN used ROI Pooling layer to extract region features. In Fast RCNN, the feature extraction, region classification and bounding box regression steps can all be optimized end-to-end, without extra cache space to store features (unlike SPP Net). Fast R-CNN achieved a much better detection accuracy than R-CNN and SPP-net, and had a better training and inference speed. [17] Currently, there are huge number of detector variants based on Faster R-CNN for different usage [18, 19, 20, 21]

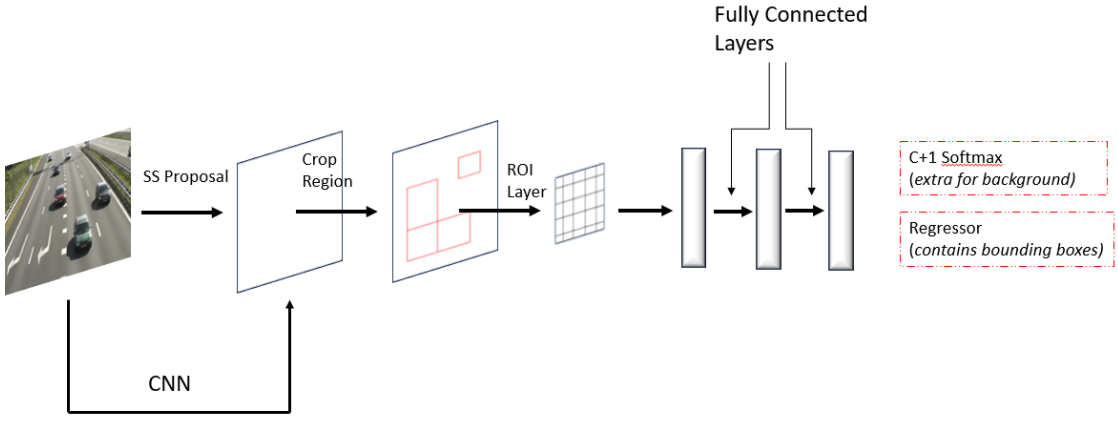


Figure 4: Architecture of Fast R-CNN.

Despite the progress in learning detectors, the proposal generation step still relied on traditional methods such as Selective Search [22] or Edge Boxes [23], which were based on low-level visual cues and could not be learned in a data-driven manner. To address this issue, Faster R-CNN [11] was developed which relied on a novel proposal generator: Region Proposal Network (RPN). This proposal generator could be learned via supervised learning methods. RPN is a fully convolutional network which takes an image of arbitrary size and generates a set of object proposals on each position of the feature map. The network slid over the feature map using an n $\times$ n sliding window, and generated a feature vector for each position. The feature vector was then fed into two sibling output branches, object classification layer (which classified whether the proposal was an object or not) and bounding box regression layer. These results were then fed into the final layer for the actual object classification and bounding box localization.

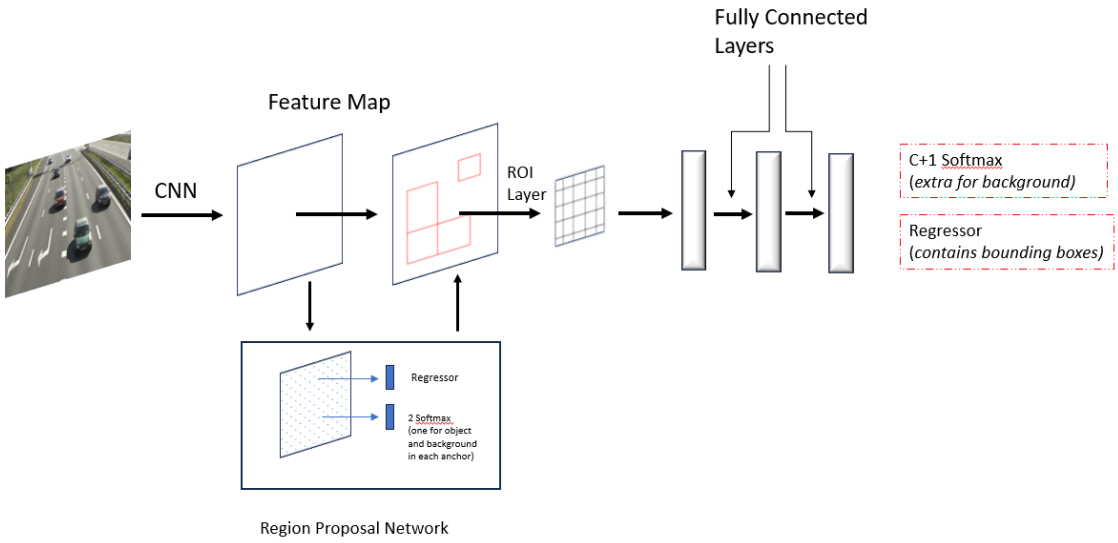


Figure 5: Architecture of Faster R-CNN.

Redmon et al. [24] developed a real-time detector called YOLO (You Only Look Once). YOLO considered object detection as a regression problem and spatially divided the whole image into fixed number of grid cells (e.g. using a 7×7 grid). Each cell was considered as a proposal to detect the presence of one or more objects. In the original implementation, each cell was considered to contain the center of (upto) two objects. For each cell, a prediction was made which comprised the following information: whether that location had an object, the bounding box coordinates and size (width and height), and the class of the object. [25] However, YOLO faced some challenges: (i) it could detect upto only two objects at a given location, which made it difficult to detect small objects and crowded objects [24]. (ii) only the last feature map was used for prediction, which was not suitable for predicting objects at multiple scales and aspect ratios.

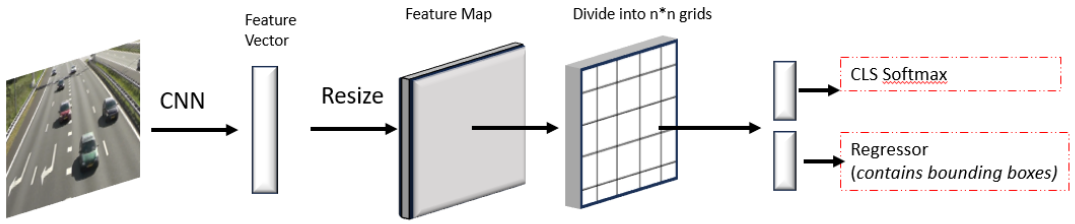


Figure 6: Architecture of YOLO model.

Moving away from variations of the convolutional neural networks (CNNs), Muhammad Azhad bin Zuraimi and Fadhlan Hafizhelmi Kamaru Zaman [19] propose a vehicle detection and tracking system using Tensorflow library with DeepSORT algorithm based on YOLOv4 model to perform Real time traffic surveillance with accuracy of 82.08% with the mean average precision of 0.5 with performance speed as 40 fps using GTX 1660ti. With modern CUDA enabled graphical processors or GPUs available through cloud platforms such as Google Colab, this metric could be further improved upon using the proposed methods.

Cheng-Jian Lin, Shiou-Yun Jeng, and Hong-Wei Lioa [26] have presented a real-time traffic monitoring system that addresses the challenges of accurate vehicle detection and classification in traffic

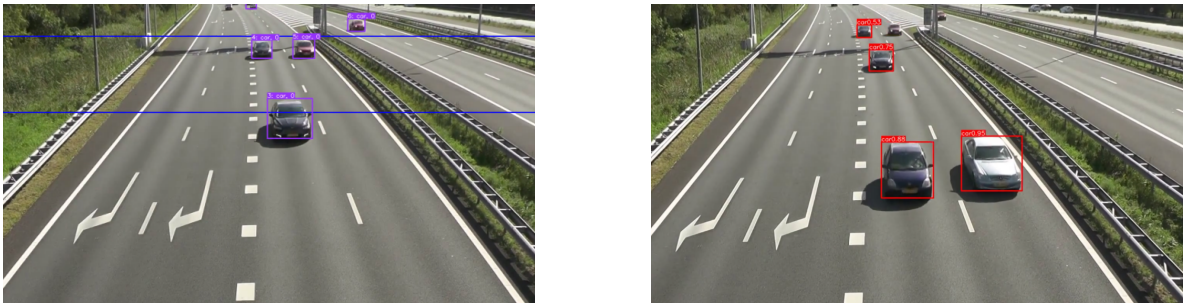
flows, particularly dealing with total occlusions. They employ convolutional neural networks, specifically utilizing the You Only Look Once (YOLO) algorithm. Their system, which integrates a virtual detection zone, Gaussian mixture model (GMM), and YOLO, enhances vehicle counting and classification efficiency. Additionally, the method estimates vehicle speed based on distance and time traveled. Experimental results using the MAVD and GRAM-RTM datasets demonstrate the effectiveness of the proposed approach. The highest classification accuracy achieved with YOLOv4 was 98.91% in MAVD and 99.5% in GRAM-RTM datasets. The method also showed strong performance in different environmental conditions (daytime, night time, and rainy day). Moreover, the average absolute percentage error for vehicle speed estimation was approximately 7.6

3 Implementation

The project was implemented using a Python script that utilizes the OpenCV library and the YOLO (You Only Look Once) object detection framework to analyze a video stream, identify vehicles, and calculate their speeds based on their movement between two predefined lines in the video frame. The YOLO model is loaded with pre-trained weights and configuration files, and the COCO dataset is used for class labels. The video stream is obtained from a .avi file , and the output is saved to an MP4 file with obtained results.

The script processes each frame of the video, detects vehicles using YOLO, and draws bounding boxes around them. Two horizontal lines are drawn on the video frame to represent positions where vehicle speeds will be calculated. The script tracks the time when each vehicle crosses the first line, and when it crosses the second line, it calculates the vehicle's speed based on the known real-life distance between the two lines. The calculated speed is then displayed on the video frame along with the bounding box and label for each detected vehicle.

The script employs multi-object detection, non-maximum suppression to filter out overlapping bounding boxes, and frame-by-frame processing. The calculated speeds and visual annotations are included in the output video. The processing continues until the user presses 'q' to exit the video playback. This script is useful for traffic monitoring applications, providing a practical example of computer vision and object detection in a real-world context.



(a) DeepSORT (b) YOLO-NAS

Figure 7: Object detection and segmentation

4 Methodology

Our system utilizes computer vision and object tracking techniques to calculate and display speed of moving vehicles. The system takes a video feed as input and for each frame an object detection deep learning model is applied to identify vehicles within the frame.

There are several object detection models like R-CNN, FAST R-CNN and YOLO. After training the models on COCO Annotated Vehicle dataset we can object their accuracy metrics. These weights and configuration files of the pretrained model are loaded into OpenCV which uses now-trained models to detect objects in real-time. Each model was trained to provide outputs in different formats so after

careful parsing we can obtain the bounding box and confidence scores. In our current version we have updated the yoloV4 model to yolo-NAS [27] which offers a streamlined output and parsing system.

Following detection, our project employs an object detection program. This algorithm assigns a unique identifier to each vehicle throughout the sequence of video frames, allowing for the tracking of individual vehicles as they move across the frame., Deep SORT extracts features from each detected bounding box using a separate CNN to distinguish between objects of the same type [28, 29]. Deep SORT maintains a state using a Kalman Filter for each object. The state typically includes the object's location, velocity, acceleration, and other dynamic parameters. The Kalman Filter predicts the new state (position and velocity) of the object based on its previously estimated state [30] which each new input and after each detection that has been associated with a track, the Kalman Filter updates its state with the new measurement. This step refines the predictions using the actual observed detection, thereby improving the accuracy of the tracker [31].

We plan to detect whether a slow-moving vehicle is causing congestion or not by determining:

- **Velocity of Buses**: Determine the speed of buses by analyzing their movement across frames. A significant drop in speed compared to vehicles in earlier time frames may indicate congestion.
- **Surrounding Vehicle Density**: Analyze the density of traffic near the bus. High vehicle density with slow movement can signal a traffic jam.

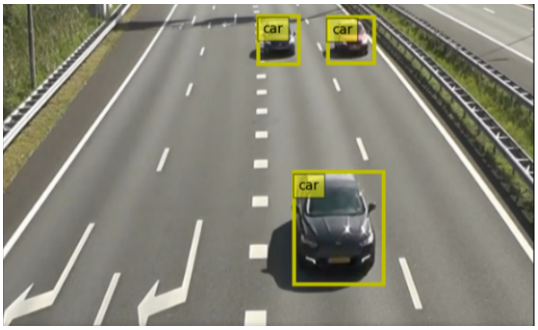
5 Results

The following results were obtained after using 3 different models to detect and segment objects in a sample video. The models used were - Faster-RCNN algorithm using the VGG16 model, YOLOv4 and YOLO-NAS.

5.1 Faster-RCNN:

The faster-RCNN model, being a two-pass algorithm was more accurate than its counterparts. However it was significantly slower in obtaining the results compared the other two models as they are one-pass algorithms

(a) Raw image

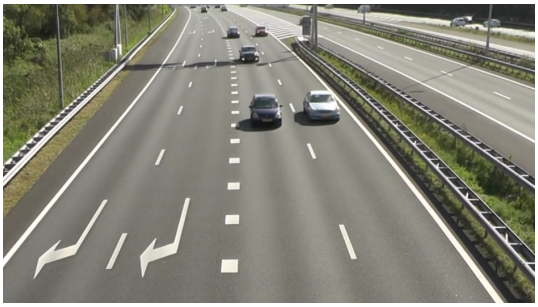


(b) Result obtained from faster-RCNN model

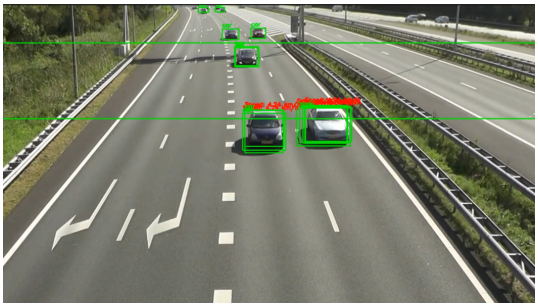
Figure 8: Results of faster-RCNN

5.2 YOLOv4:

YOLOv4 model [32] obtained results with lower latency compared to faster-RCNN model, but it fell short of the latest YOLO-NAS model. Additionally, it struggled with tracking of objects through continuous frames, evidence of which can be seen in the adjoined figure.



(a) Raw image

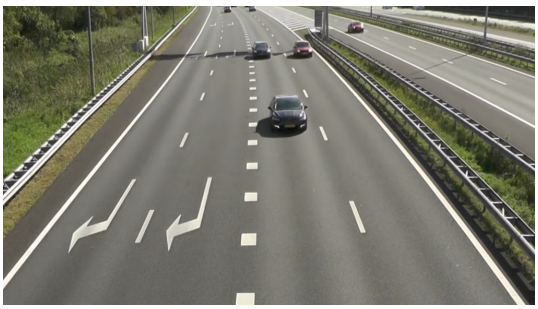


(b) Result obtained from YOLOv4 model

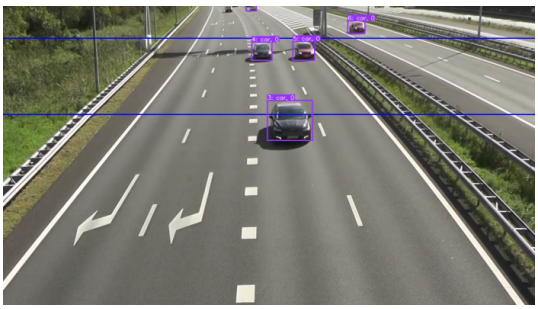
Figure 9: Results of YOLOv4

5.3 YOLO-NAS:

This model [33] provided results with the lowest latency and was successful in tracking objects through individual frames. Along with detection, segmentation and tracking, calculation of speed of vehicles was done with the help of this model. The following result was obtained when the same video was given as input to the model.



(a) Raw image



(b) Result obtained from YOLO-NAS model

Figure 10: Results of YOLO-NAS

Model	**Latency (ms)**	**mAP 0.5:0.95**	**reference**
Faster-RCNN (VGG16)	27	0.73	[11]
YOLOv4	8.7	0.435	[32]
YOLOv8	7.1	0.509	[34]
YOLO-NAS	5.8	0.523	[33]

Table 1: Comparison of different models based on latency and mAP 0.5:0.95

6 Future Prospects and Challenges

We have witnessed significant development of vehicle classification systems in the past decade. Thanks to recent advances in sensing, machine learning, and wireless communication technologies, the classification accuracy has improved greatly at a significantly reduced cost. However, these emerging vehicle classification systems have left a number of open questions. In this section, we discuss these challenges and several future research directions.

More and more vehicle classification systems depend on machine learning techniques. To achieve high classification accuracy, however, a huge amount of data should be collected to train and create an effective classification model. Especially, the manual labelling process for training the classification model requires a significant amount of time and efforts. It also requires extra efforts for obtaining

the ground truth data. Although we have seen that many classification systems achieve very high classification accuracy, achieving near 100% classification accuracy especially for a large number of vehicle types is still a very challenging task. One possible reason for the difficulty lies in the fact that most solutions rely on a single type of sensor for vehicle classification. If different type of sensors could be used together to collect this data, the task would become much easier because collecting video data is dependent upon physical and environmental factors and thus could be complemented through the use of devices such as infrared sensors [9] which could be used for better detection during night or dark periods of the day.

Another major hurdle that prevents us from employing these high accuracy systems such as YOLO into general traffic surveillance is the hardware requirements to perform real time object detection and tracking. Even though there have been continuous improvements in the latency and mAP scores of modern models[27], it is still not feasible to implement these Although, the advent of modern large language models (LLMs) is significantly accelerating the growth of GPU capabilities. The increased demand from LLMs ensures a continuous cycle of improvement, positioning GPUs as critical enablers of future AI breakthroughs.

The advent of self-driving cars [35] have led to opening of a whole new avenue with regards to the management of traffic systems. Autonomous Traffic Management (ATM) [36] is one of the most vigorously pursued areas of research in the ITS now a days which promise to reduce traffic problems through a well-connected and coordinated infrastructure. For this, AVs and their connected environment need to manage through dynamically changing traffic situations intelligently, and road conditions [37]

7 Conclusion

We presented a review of the modern traffic surveillance systems, focusing on the key functionality of vehicle detection, tracking and speed calculation systems. By categorizing the vehicle classification systems into 3 categories based on the tools used by authors in implementing the system, mainly the traditional methods (SVMs, GMMs, SIFT, Computer Vision), the Deep Learning Techniques (CNNs, RCNNs, RPMs, ANNs) and the modern methods that provide an inbuilt set of tools to aid in classification and detection such as TensorFlow, Colab, YOLO, etc. Our project was implemented using a Python script that utilizes the OpenCV library and the YOLO (You Only Look Once) object detection framework with deep SORT to analyze a video stream, identify vehicles, and calculate their speeds based on their movement between two predefined lines in the video frame. We also discussed some of the limitations of these proposed solutions as well as some future research directions.

References

[1] M. Won, T. Park, and S. H. Son, "Toward mitigating phantom jam using vehicle-to-vehicle communication," *IEEE Trans. Intell. Transp*, vol. 18, pp. 1313–1324, May 2017.

[2] M. T. Masood, "Transportation problems in developing countries pakistan: a case-in-point," *International Journal of Business and Management*, vol. 6, no. 11, p. 256, 2011.

[3] A. Arinaldi, J. A. Pradana, and A. A. Gurusinga, "Detection and classification of vehicles for traffic video analytics," in *Procedia Computer Science*, vol. 144, 2018, pp. 259–268.

[4] H. Lu, Z. Sun, and W. Qu, ": Big data and its applications in urban intelligent transportation system," *J. Transp. Syst. Eng. Inf. Technol*, vol. 15, no. 5, 2015.

[5] R. Ravish and S. R. Swamy, "Intelligent traffic management: A review of challenges, solutions, and future perspectives," *Transport and Telecommunication Journal*, vol. 22, no. 2, pp. 163–182, 2021.

[6] M. Sheng, C. Liu, Q. Zhang, L. Lou, and Y. Zheng, "Vehicle detection and classification using convolutional neural networks," in *2018 IEEE 7th Data Driven Control and Learning Systems Conference (DDCLS)*, 2018.

[7] T. Kumar and D. S. Kushwaha, "An efficient approach for detection and speed estimation of moving vehicles," in *Procedia Computer Science*, vol. 89, 2016, pp. 726–731.

[8] Z. Chen, N. Pears, M. Freeman, and J. Austin, "Background subtraction in video using recursive mixture models spatio-temporal filtering and shadow removal," in *Proc. International Symposium on Visual Computing*, 2009, pp. 1141–1150.

[9] M. Won, "Intelligent traffic monitoring systems for vehicle classification: A survey," *IEEE Access*, vol. 8, pp. 73 340–73 358, 2020.

[10] R. Girshick, "Fast r-cnn," *arXiv:1504.08083*, 2015.

[11] S. Ren and et al., "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in Neural Information Processing Systems*, vol. 28, 2015.

[12] D. Zhao, Y. Chen, and L. Lv, "Deep reinforcement learning with visual attention for vehicle classification," *IEEE Transactions on Cognitive and Developmental Systems*, vol. 9, no. 4, pp. 356–367, Dec. 2017.

[13] M. Rahman, M.-W. Kang, and P. Biswas, "Predicting time-varying, speed-varying dilemma zones using machine learning and continuous vehicle tracking," *Transportation Research Part C: Emerging Technologies*, 2021.

[14] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[15] J. Kleban, X. Xie, and W.-Y. Ma, "Spatial pyramid mining for logo detection in natural scenes," in *2008 IEEE International Conference on Multimedia and Expo*. IEEE, 2008, pp. 1077–1080.

[16] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 37, no. 9, pp. 1904–1916, 2015.

[17] X. Wu, D. Sahoo, and S. C. Hoi, "Recent advances in deep learning for object detection," *Neurocomputing*, vol. 396, pp. 39–64, 2020.

[18] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2117–2125.

[19] Z. Cai and N. Vasconcelos, "Cascade r-cnn: Delving into high quality object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 6154–6162.

[20] T. Kong, A. Yao, Y. Chen, and F. Sun, "Hypernet: Towards accurate region proposal generation and joint object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 845–853.

[21] S. Bell, C. L. Zitnick, K. Bala, and R. Girshick, "Inside-outside net: Detecting objects in context with skip pooling and recurrent neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2874–2883.

[22] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders, "Selective search for object recognition," *International journal of computer vision*, vol. 104, pp. 154–171, 2013.

[23] C. L. Zitnick and P. Dollár, "Edge boxes: Locating object proposals from edges," in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*. Springer, 2014, pp. 391–405.

[24] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[25] J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7263–7271.

[26] T.-H. M. et al., "A real-time vehicle counting, speed estimation, and classification system based on virtual detection zone and yolo," *Mathematical Problems in Engineering*, vol. 2021, p. 1577614, 2021.

[27] J. Terven and D. Cordova-Esparza, "A comprehensive review of yolo: From yolov1 to yolov8 and beyond," *arXiv preprint arXiv:2304.00501*, 2023.

[28] X. Hou, Y. Wang, and L.-P. Chau, "Vehicle tracking using deep sort with low confidence track filtering," in *2019 16th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. IEEE, 2019, pp. 1–6.

[29] A. Pujara and M. Bhamare, "Deepsort: real time & multi-object detection and tracking with yolo and tensorflow," in *2022 International Conference on Augmented Intelligence and Sustainable Systems (ICAISS)*. IEEE, 2022, pp. 456–460.

[30] Y. Kim, H. Bang *et al.*, "Introduction to kalman filter and its applications," *Introduction and Implementations of the Kalman Filter*, vol. 1, pp. 1–16, 2018.

[31] G. Welch, G. Bishop *et al.*, "An introduction to the kalman filter," 1995.

[32] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," 2020.

[33] S. Aharon, Louis-Dupont, Ofri Masad, K. Yurkova, Lotem Fridman, Lkdci, E. Khvedchenya, R. Rubin, N. Bagrov, B. Tymchenko, T. Keren, A. Zhilko, and Eran-Deci, "Super-gradients," 2021. [Online]. Available: https://zenodo.org/record/7789328

[34] G. Jocher, A. Chaurasia, and J. Qiu, "Ultralytics yolov8," 2023. [Online]. Available: https://github.com/ultralytics/ultralytics

[35] A. Mushtaq, I. U. Haq, M. U. Imtiaz, A. Khan, and O. Shafiq, "Traffic flow management of autonomous vehicles using deep reinforcement learning and smart rerouting," *IEEE Access*, vol. 9, pp. 51 005–51 019, 2021.

[36] S. E. Hamdani and N. Benamar, "Autonomous traffic management: Open issues and new directions," in *Proc. International Conference on Selected Topics in Mobile and Wireless Networking (MoWNeT)*, Jun. 2018, pp. 1–5.

[37] M. V. Rajasekhar and A. K. Jaswal, "Autonomous vehicles: The future of automobiles," *2015 IEEE International Transportation Electrification Conference (ITEC)*, pp. 1–6, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:8766010

8 Appendix

8.1 Models utilised and implemented

8.1.1 Guassian Mixture Model

Here is the code used for the guassian mixture model employment:

```
import numpy as np
import cv2
from sklearn.mixture import GaussianMixture
import matplotlib.pyplot as plt

image_path = '/content/img1.jpg'
image = cv2.imread(image_path)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Preprocess the image: Flatten the image to (num_pixels, 3) where 3 corresponds to RGB channels
pixels = image.reshape(-1, 3)

# Fit Gaussian Mixture Model
n_components = 3  # Number of segments
gmm = GaussianMixture(n_components=n_components, random_state=0)
gmm.fit(pixels)

# Predict the label of each pixel
labels = gmm.predict(pixels)

# Reshape the labels back to the original image shape
segmented_image = labels.reshape(image.shape[0], image.shape[1])

# Display the segmented image
plt.imshow(segmented_image, cmap='viridis')
plt.axis('off')
plt.show()
```

8.2 Evaluation methods employed

8.2.1 YOLOv8

Here is the code used to evaluate the pre-trained yolov8 model.

```
from ultralytics import YOLO

# Load a model
model = YOLO("yolov8n.pt")  # load an official model
# model = YOLO("path/to/best.pt")  # load a custom model

# Validate the model
metrics = model.val()  # no arguments needed, dataset and settings remembered
metrics.box.map  # map50-95
metrics.box.map50  # map50
metrics.box.map75  # map75
metrics.box.maps  # a list contains map50-95 of each category
```

8.2.2 YOLOv4

Here is the code used to evaluate the pre-trained yolov4 model.

```
from ultralytics import YOLO

# Load a model
model = YOLO("yolov4nu.pt")  # load an official model
# model = YOLO("path/to/best.pt")  # load a custom model

# Validate the model
metrics = model.val()  # no arguments needed, dataset and settings remembered
metrics.box.map  # map50-95
metrics.box.map50  # map50
metrics.box.map75  # map75
metrics.box.maps  # a list contains map50-95 of each category
```

8.2.3 Faster-RCNN

Here is the code used to determine the evaluation metrics of pre-trained faster-rcnn model.

```
!pip install autogluon
from autogluon.multimodal import MultiModalPredictor
checkpoint_name = "faster_rcnn_r50_fpn_2x_coco"
num_gpus = -1  # use all GPUs
predictor = MultiModalPredictor(
    hyperparameters={
        "model.mmdet_image.checkpoint_name": checkpoint_name,
        "env.num_gpus": num_gpus,
    },
    problem_type="object_detection",
)
test_path = "/kaggle/input/coco-2017-dataset/coco2017/annotations/instances_val2017.json"
predictor.evaluate(test_path)
```

8.2.4 Custom evaluation metrics

Here is the code for the most popular evaluation metrics of object detection models.

```
def calculate_ap(precisions, recalls):
precisions = [0] + precisions + [0]
recalls = [0] + recalls + [1]

for i in range(len(precisions) - 1, 0, -1):
    precisions[i - 1] = max(precisions[i - 1], precisions[i])

indices = [i for i in range(len(recalls) - 1) if recalls[i + 1] != recalls[i]]
ap = sum((recalls[i + 1] - recalls[i]) * precisions[i + 1] for i in indices)
return ap

def calculate_map(pred_boxes, gt_boxes, num_classes, iou_threshold=0.5):
ap_sum = 0.0
for cls in range(num_classes):
    cls_pred_boxes = [box for box in pred_boxes if box[4] == cls]
```

```
    cls_gt_boxes = [box for box in gt_boxes if box[4] == cls]

    precisions, recalls = [], []
    for threshold in range(0, 101, 10):
        threshold /= 100.0
        precision, recall = calculate_precision_recall(cls_pred_boxes, cls_gt_boxes, threshold)
        precisions.append(precision)
        recalls.append(recall)

    ap = calculate_ap(precisions, recalls)
    ap_sum += ap

mAP = ap_sum / num_classes
return mAP

def calculate_iou(boxA, boxB):
# Coordinates of the intersection rectangle
xA = max(boxA[0], boxB[0])
yA = max(boxA[1], boxB[1])
xB = min(boxA[2], boxB[2])
yB = min(boxA[3], boxB[3])

# Area of the intersection rectangle
interArea = max(0, xB - xA + 1) * max(0, yB - yA + 1)

# Areas of the prediction and ground-truth rectangles
boxAArea = (boxA[2] - boxA[0] + 1) * (boxA[3] - boxA[1] + 1)
boxBArea = (boxB[2] - boxB[0] + 1) * (boxB[3] - boxB[1] + 1)

# IoU calculation
iou = interArea / float(boxAArea + boxBArea - interArea)

return iou
```