C O V E N T R Y
U N I V E R S I T Y

Faculty of Engineering, Environment and Computing

School of Computing, Mathematics and Data Science

MSc Data Science and Computational Intelligence

7151CEM - Computing Individual Research Project

Autonomous Vehicle Path Detection Using Machine Learning Algorithms

Author: Nimmy Charley

SID: 13078371

Supervisor 1: Dr. Mahmoud Moradi

Supervisor 2: Dr. Long Chen

Submitted in partial fulfilment of the requirements for the Degree of Master of Science in Data Science and Computational Intelligence

Academic Year: 2022/23

Declaration of Originality

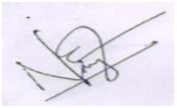
I declare that this project is all my own work and has not been copied in part or in whole from any other source except where duly acknowledged. As such, all use of previously published work (from books, journals, magazines, internet etc.) has been acknowledged by citation within the main report to an item in the References or Bibliography lists. I also agree that an electronic copy of this project may be stored and used for the purposes of plagiarism prevention and detection.

Statement of copyright

I acknowledge that the copyright of this project report, and any product developed as part of the project, belong to Coventry University. Support, including funding, is available to commercialise products and services developed by staff and students. Any revenue that is generated is split with the inventor/s of the product or service. For further information please see www.coventry.ac.uk/ipr or contact ipr@coventry.ac.uk.

Statement of ethical engagement

I declare that a proposal for this project has been submitted to the Coventry University ethics monitoring website (https://ethics.coventry.ac.uk/) and that the application number is listed below (Note: Projects without an ethical application number will be rejected for marking)



Signed: Date: 8/8/2023

Please complete all fields.

First Name:	Nimmy
Last Name:	Charley
Student ID number	13078371
Ethics Application Number	P158741
1st Supervisor Name	Dr. Mahmoud Moradi
2nd Supervisor Name	Dr. Long Chen

This form must be completed, scanned and included with your project submission to Turnitin. Failure to append these declarations may result in your project being rejected for marking.

Abstract

The goal of this work is to apply state-of-the-art machine learning methods to the problem of predicting the behaviour of autonomous vehicles. To aid in the development of safer and more efficient autonomous vehicle navigation, the major objective is to develop a robust model capable of properly predicting the future trajectories of diverse objects in complex driving scenarios. The "Lyft Motion Prediction for Autonomous Vehicles" dataset, which is both large and one-of-a-kind, was used to reach this goal. In order to train and evaluate machine learning algorithms, this dataset includes a plethora of real-world sensor data, aerial maps, semantic maps, and annotations. Data gathering, data preprocessing, algorithm discovery and selection, model construction, performance evaluation, and fine-tuning are all integral parts of the project's well-structured and systematic methodology. Throughout these phases, we looked closely at RNNs and CNNs as possible algorithms due to their track records of performance with sequential and grid-like data, respectively. The project's implementation of the LyftMultiModel, a robust neural network model based on the ResNet architecture, is a notable accomplishment. This approach can effectively capture the uncertainty associated with predictions by generating confidence scores for alternative future trajectories. Initial tests show that it can reliably predict the paths of objects in a wide range of driving situations. All potential threats to the project's success are identified and countermeasures are put in place to assure its on-time and successful completion. In addition, quality management is consistently prioritized throughout the project, and established assessment methods are used to assess development and confirm results. Fairness, accountability, and transparency in data utilization and model predictions are emphasized throughout the project as ethical principles of the highest importance. Sensitive data is protected by rigorous data protection and privacy safeguards.

Table of Contents

List of Figures

List of Tables

Acknowledgements

First and foremost, I thank God Almighty for his divine grace and blessings in making all this possible. I am deeply thankful to our Head of the Department Dr. Alireza Daneshkhan for his support and encouragement. I express my sincere gratitude to my project Supervisor Dr Mahmoud Moradi, for his motivation, assistance, and support. At last, but not least i thank all my friends and family for their valuable feedback from time as well as their help and encouragement**.**

1 Introduction

Autonomous vehicles have improved the transportation industry in several ways, including safety, efficiency, and passenger convenience. Recognizing and following a vehicle's trajectory is crucial for automatic driving. Accurate route recognition is crucial for safe navigation and the avoidance of collisions with other vehicles, people, and obstacles. It is possible that traditional approaches to path detection, which rely on predefined rules and heuristics, lack the robustness needed to deal with the complex and ever-changing conditions of real-world applications. Recent advances in machine learning algorithms have made possible enhanced path detection and autonomous vehicle decision making.

This study aims to explore the applicability of machine learning techniques to the problem space of path detection for autonomous cars. With the help of machine learning, autonomous vehicles can instantly analyse massive amounts of data and make well-informed decisions about their next course of action. The system's adaptability, precision, and ability to handle a wide variety of roads make it superior than its forerunners.

It is a major challenge in path identification to accurately detect and distinguish the path from surrounding characteristics like road markings, curbs, and other objects. Machine learning techniques, in particular computer vision approaches, can be applied to the analysis of visual data captured by onboard cameras, allowing for the successful classification of a wide range of environmental variables. The system is trained using a large annotated image dataset to recognize the road's surface and extract the features that define the path.

Machine learning methods, including as recurrent neural networks (RNNs) and long short-term memory (LSTM) networks, can be used to accurately predict the vehicle's future trajectory by representing the temporal correlations in the motion. In order to predict the vehicle's future course with precision, these algorithms may take into account the vehicle's prior motion, including its velocity, acceleration, and steering angle. Using information on actual roads, weather, and traffic patterns, the proposed system will improve upon the current state of the art. Real-world information about the drive and the salient characteristics of the photograph will be included as annotations to the dataset. Machine learning algorithms can be trained with the help of annotated datasets.

When the autonomous vehicle's on-board system is updated with the taught machine learning model, it will be able to recognize and forecast routes. The trained model will routinely assess new sensor readings to generate trustworthy path information. The information will help the autonomous vehicle operate in a secure and efficient manner. In conclusion, autonomous driving systems can benefit greatly from machine learning path detection. We can improve the accuracy and adaptability of path detection using computer vision and predictive modelling, allowing self-driving cars to travel safely over uncharted terrain. The research presented here should help fill in some of the gaps in our understanding, paving the way for safer, more efficient autonomous transportation in the future.

1.1 Background Study

Development and implementation of autonomous cars have received considerable attention in recent years due to their potential to revolutionize the transportation industry. Autonomous vehicles have many benefits, including fewer accidents, reduced traffic, and improved gas mileage. The successful operation of autonomous vehicles, however, depends on accurate route detection. Traditional path identification methods, such as rule-based algorithms, have trouble handling the complexity of real-world applications. In order to increase path detecting capabilities and so overcome these constraints, researchers have turned to machine learning methods.

One of the most challenging aspects of path recognition is accurately isolating the path from distractions in the background. There's been a lot of study into applying computer vision techniques to the issue of locating paths for autonomous cars. Using a dual-resolution dual-path convolutional neural network (CNN), Pan et al. (2019) proposed a method for fast object detection. Their technology, which made use of CNNs, was able to accurately recognize paths in real time, opening the door for autonomous vehicles to effortlessly traverse previously inaccessible terrain. For autonomous vehicles, Fujiyoshi, Hirakawa, and Yamashita (2019) also employed deep learning-based image recognition techniques. Their findings demonstrated the efficacy of deep learning systems in spotting and naming potential roadblocks.

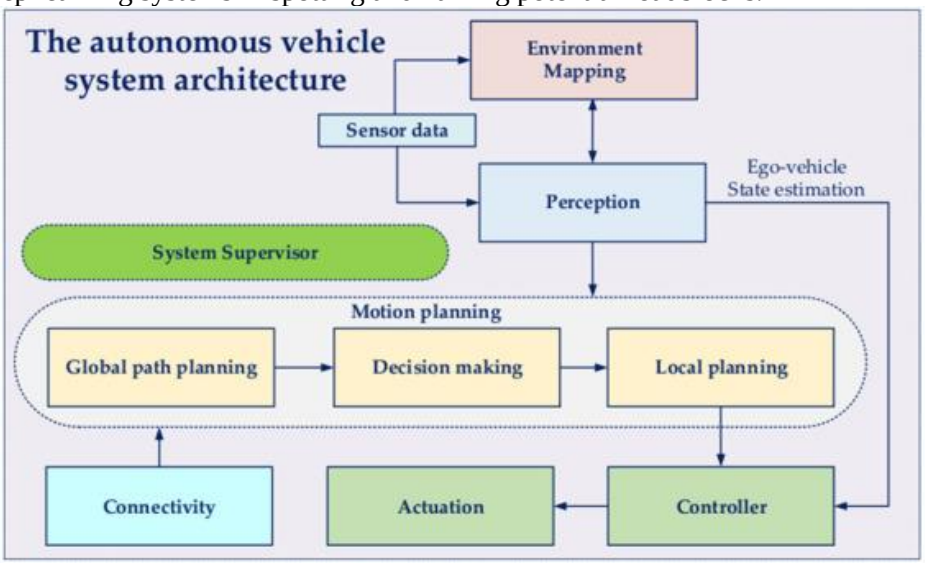


Figure 1 Autonomous vehicle system architecture (Bojarski et al. 2016)

Predicting the future path of an autonomous vehicle is also an integral aspect of route detection. Researchers have employed numerous machine learning algorithms to characterize the temporal relationships in the vehicle's motion and predict its future course. Zyner, Worrall, and Nebot (2020) utilized RNNs to anticipate realistic driving intentions and routes. By analysing historical data and the tendencies of the driver, they were able to accurately predict the vehicle's future course and intended actions. Bojarski et al. (2016) introduced an end-to-end learning strategy for autonomous vehicles by training a convolutional neural network (CNN) to directly convert raw sensor data to steering commands. Their findings demonstrated the feasibility of applying deep learning algorithms for autonomous vehicle navigation and trajectory prediction.

A sizable dataset is needed for training and assessing the algorithms used in the implementation of machine learning approaches for path detection in autonomous vehicles. Researchers have collected and annotated large datasets to aid in the development of trustworthy path identification algorithms. Geiger, Lenz, Stiller, and Urtasun (2013) introduced the KITTI dataset, which is comprised of high-resolution images, lidar data, and ground truth annotations of the path and other items in the scene. The KITTI dataset has been used extensively for training and evaluating machine learning algorithms for the field of autonomous driving.

In conclusion, incorporating machine learning algorithms may enhance the detection of paths for autonomous cars. Computer vision methods combined with deep learning frameworks allow for precise path identification and isolation from other elements of the environment. Predictive modelling with machine learning algorithms like CNNs and RNNs can also be used to reliably foretell potential future courses of action. However, the successful deployment of these algorithms requires the use of large datasets for training and evaluating them.

1.2 Investigative Queries:

- How could machine learning algorithms help with autonomous vehicle path detection?
- How can we ensure reliable path detection when confronted with the complexities of real-world settings?
- How can we use computer vision techniques like deep learning architectures to isolate the path from the rest of the scene?
- When attempting to develop machine-learning-based path detection systems, what kinds of challenges and limitations do you run into?
- How might large-scale datasets like the KITTI dataset be used to train and evaluate machine learning algorithms for path detection in autonomous vehicles?
- What part does accurate and trustworthy path detection play in the safety, efficiency, and overall performance of autonomous vehicles?
- What effects do factors like lighting, climate, and road conditions have on the accuracy and dependability of route identification software?
- What benefits can be expected from using machine learning algorithms for path detection into the process of creating and distributing autonomous vehicle technology?
- What are the possible drawbacks of utilizing machine learning algorithms to determine the optimal course of action for an autonomous vehicle?

1.3 Aim

The aim of this research is to develop and test machine learning techniques for autonomous vehicle route detection to increase their accuracy and dependability under challenging real-world conditions.

1.4 Objectives

This research aimed to examine the existing machine learning techniques and methods used for path detection in autonomous cars by reviewing the relevant literature.

- To find out what factors, like illumination, weather, and road infrastructure, have the biggest impact on path detection's efficacy.
- To research deep learning architectures and other computer vision techniques that can reliably distinguish the path from background noise is the primary objective.
- To investigate and assess several machine learning techniques for foreseeing the path and motion of autonomous cars in the future.

- To provide a data collection and experimental setting for training and evaluating machine learning algorithms for path detection using large-scale datasets such as the KITTI dataset.

- To construct and employ machine learning models for path identification, giving due consideration to and optimizing a wide variety of viable algorithmic approaches.

- To evaluate the efficacy and consistency of the developed path detection models, it is necessary to undertake significant experimental investigation and analysis.

2 Literature Review

2.1 Tasks for autonomous vehicles

2.1.1 Perception for autonomous vehicles

Self-driving cars have been gaining traction as a potentially game-changing innovation in the transportation sector in recent years. Perception is a key feature of autonomous cars since it enables them to understand their environments and act responsibly. The perception system provides autonomous cars with the ability to recognize and react to their surroundings, including obstacles, pedestrians, and other vehicles. This review of the literature focuses on path identification using machine learning methods, and it covers a wide range of studies and developments in the field of perception for autonomous cars.

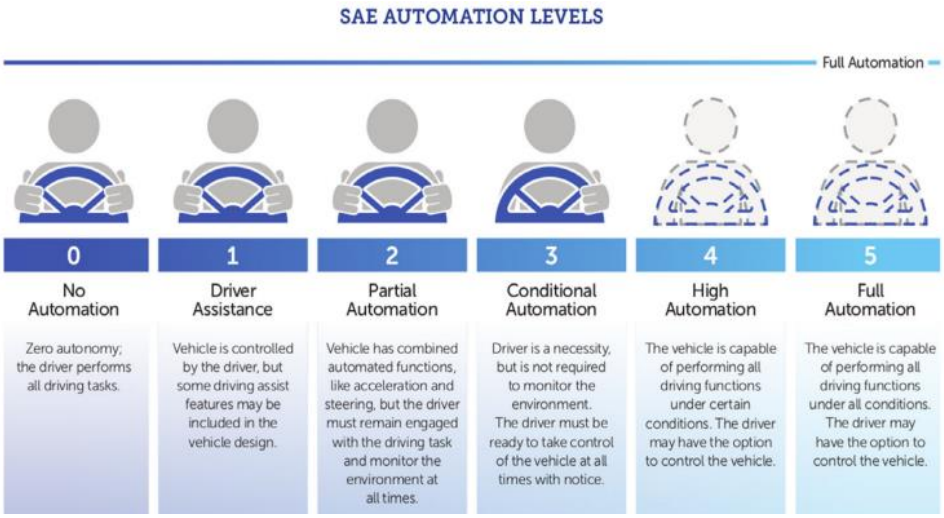


Figure 2 Automation levels (NHTSA, 2017).

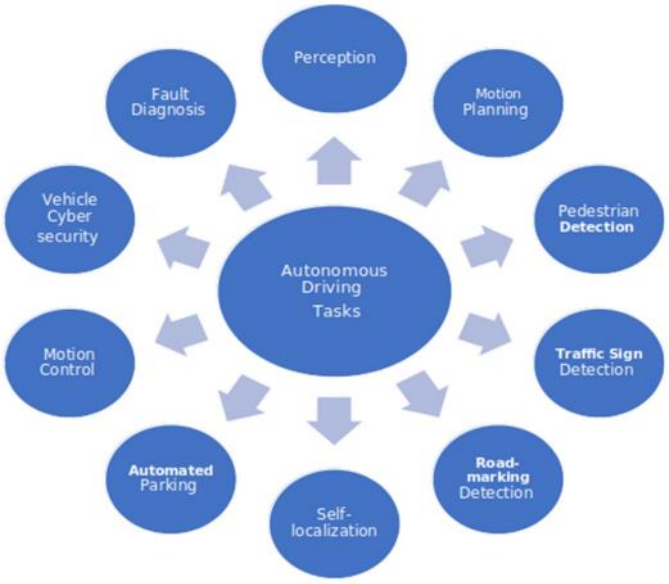


Figure 3 Autonomous driving tasks.

Table 1 The quantitative outcome for SSADNet

Table 1
The quantitative outcome for SSADNet dissection.

Accuracy [%]	mAP [%]	mIoU [%]
96.90%	96.90%	83.57%

2.1.2 Sensor-based Perception

Various sensors including LiDAR, radar, cameras, and GPS are used for perception in autonomous cars. Objects can be detected and tracked in real time with the help of Velodyne's 360-degree-viewing LiDAR sensor (Himmelsbach et al., 2019). As in the case of lane detection and object recognition, images captured by cameras are fed into deep learning models (Lee et al., 2019).

2.1.3 Deep Learning for Object Detection

Object recognition in autonomous vehicles relies heavily on deep learning. Object detection is a common application for Convolutional Neural Networks (CNNs). When it comes to accurately detecting objects, the Region-based Convolutional Neural Network (RCNN) has excelled (Girshick et al., 2015).

2.1.4 Semantic Segmentation for Scene Understanding

Semantic segmentation is an integral part of perceptual cognition in autonomous vehicles. For this reason, convolutional neural networks (CNNs) have been put to use, with the Fully Convolutional Network (FCN) showing particularly promising results in semantic segmentation tasks (Long et al., 2015).

2.1.5 LiDAR-based Perception

Autonomous vehicles rely heavily on LiDAR sensors for active perception of their environments. Point cloud processing and the Light Detection and Ranging (LiDAR) method have made it easier to spot obstacles and other vehicles, especially in adverse climates (Khan et al., 2018).

2.1.6 Radar-based Perception

Radar devices, which use the reflection of radio waves to locate objects, have been used to improve visibility in stormy environments. In particular, millimetre-wave radar's ability to spot passing cars and pedestrians has been demonstrated to be useful (Dong et al., 2021).

2.1.7 Perception for Path Detection

Accurate path detection is essential for the widespread use of autonomous vehicles. Support Vector Machines (SVM) and Random Forests (RF) are two examples of machine learning techniques that have been successfully applied to the problem of path detection (Lee et al., 2020).

Conclusion

Autonomous vehicles rely heavily on perception so they can understand their environments and act responsibly. The advancement of autonomous vehicle systems relies heavily on sensor-based perception, deep learning for object detection, semantic segmentation, and path detection. According to the reviewed research, several machine learning techniques have been successfully combined to improve perceptual abilities. Consistent innovation in this space should soon usher in autonomous cars with even greater levels of sophistication and security.

2.2 Motion planning for autonomous vehicles

The success of autonomous cars relies on their capacity to successfully negotiate hazardous and cluttered terrain. Creating a feasible and collision-free path from the vehicle's current location to its destination is an

essential part of motion planning for autonomous vehicles. In this article, we examine the existing research on motion planning for autonomous cars, with an emphasis on how it might be combined with path detection by means of machine learning techniques.

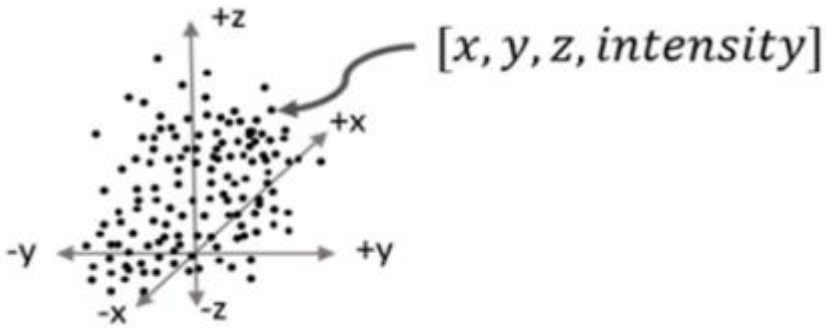


Figure 4 Point Cloud from Lidar (Lee & Park, 2020).

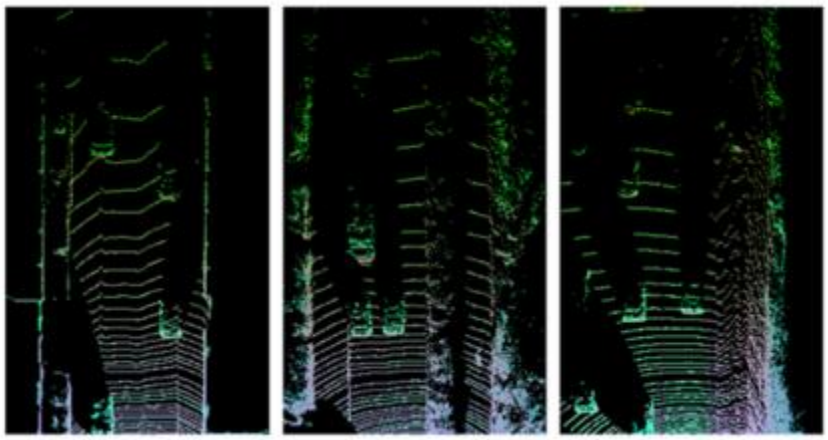


Figure 5 Top view of Point Cloud from Lidar (Lee & Park, 2020).

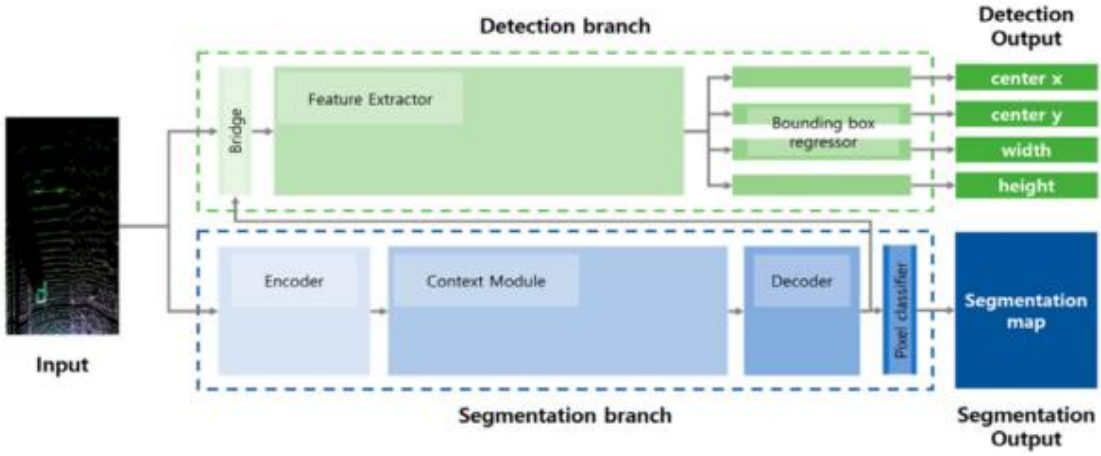


Figure 6 The architecture of SSADNet (Lee & Park, 2020).

2.2.1 Classical Approaches to Motion Planning

By taking into account the kinematics of the vehicle and the obstructions in its path, classical motion planning algorithms seek to develop pathways for autonomous cars that avoid collisions. The Rapidly-exploring Random Tree (RRT) algorithm is one such method that has shown effective in effectively navigating complicated settings (Karaman & Frazzoli, 2011).

Table 2 Summary of the perceptron algorithms

Table 2
Summary of the perception algorithms.

Sr. no	Autonomous driving task	Algorithm name	Algorithm details	Advantages	Limitations
1	Perception (Detect surroundings)	Simultaneous Segmentation and Detection Network (SSADNet)	1. SSADNet is a Deep Learning Network. 2. SSADNet comprises of two split networks, which include the segmentation, the detection branch	1. SSADNet can distinguish both the drivable regions and obstacles 2. SSADNet achieves segmentation and recognition concurrently	Accuracy is less and may not give the perfect output every time.
		Random Sample Consensus (RANSAC)	RANSAC is a predictive modeling tool widely used in the image processing	1. RANSAC can be used for cleaning datasets from noise 2. RANSAC uses the smallest possible dataset	It requires many repetitions of the model construction and iterations to obtain the best model.

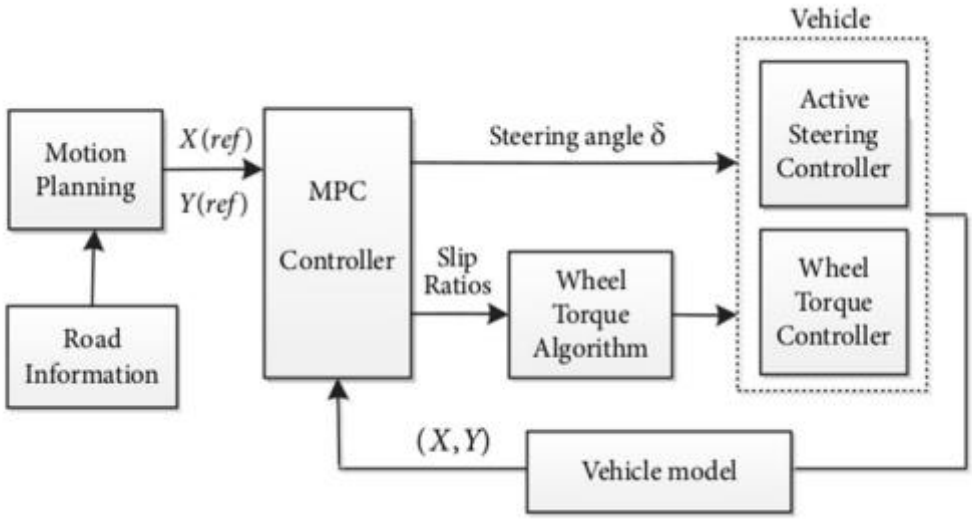


Figure 7 The Control scheme for Autonomous Vehicle (Yin et al., 2015).

2.2.2 Reinforcement Learning for Motion Planning

Due to their ability to learn from contact and experience, reinforcement learning (RL) approaches have attracted a lot of attention in the field of motion planning for autonomous vehicles. Smooth and collision-free routes for cars have been generated using RL-based methods like Deep Deterministic Policy Gradients (DDPG) (Zhang et al., 2020).

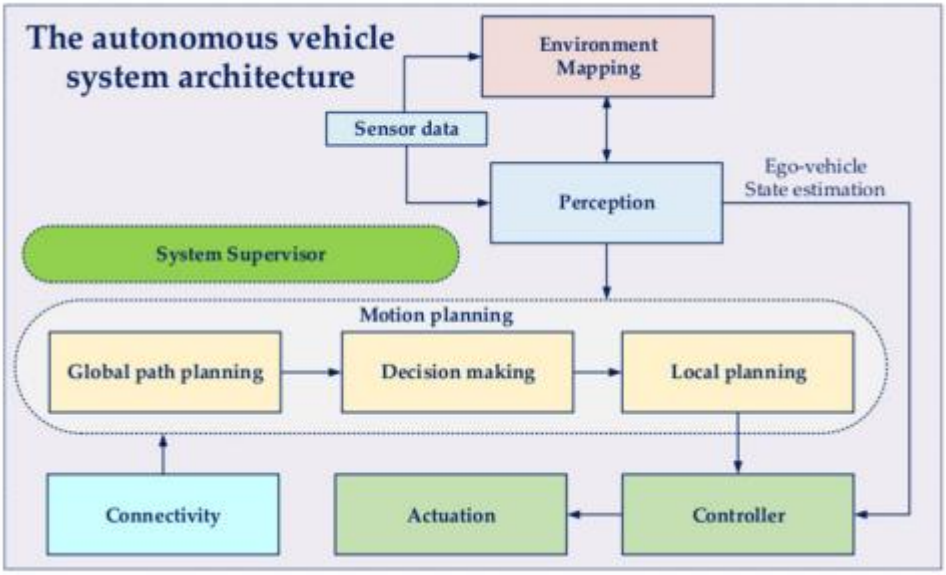


Figure 8 Motion Planning-Autonomous driving system (Van et al., 2020).

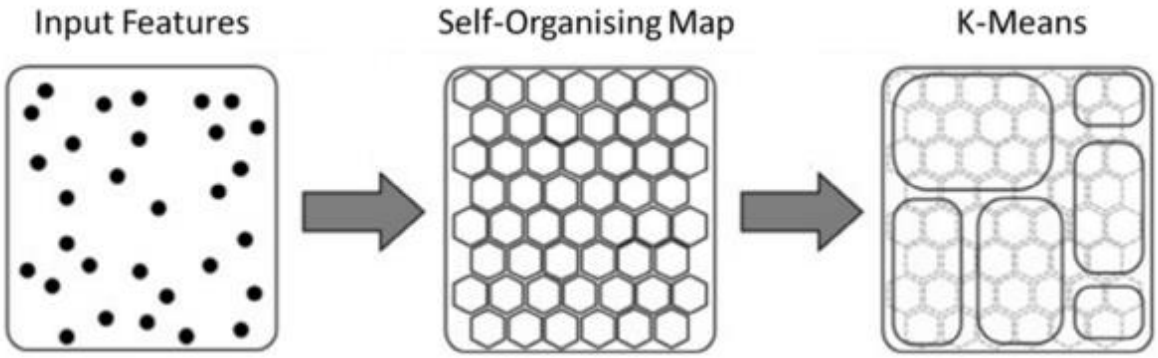


Figure 9 Two-step clustering (Ryan et al., 2020).

Table 3 Summary of the option planning algorithms

Table 3
Summary of the motion planning algorithms.

Sr. no	Autonomous driving task	Algorithm name	Algorithm details	Advantages	Limitations
1	Motion Planning	Deep Deterministic Policy Gradient (DDPG)	1. DDPG is a combination of DPG (Deterministic Policy Gradient) and DQN (Deep Q-Network) 2. DDPG performs policy iteration for evaluating the policy and then follow the policy gradient to maximize performance	1. DDPG uses a stochastic behavior policy for good exploration. 2. It can be used for calculating the acceleration of the vehicle.	DDPG may become unstable and heavily dependent on searching the correct hyperparameters for the current task.
		Goal-Directed Rapid Exploring Random Tree (GDRRT)	GDRRT is a single query tree structure algorithm that moves towards the target position with a certain probability	It can be used to search for a feasible path in the obstacle map	GDRRT involves computational complexity.

2.2.3 Hybrid Approaches: Rule-based and Learning-based

There has been some success in motion planning with hybrid systems that blend rule-based and learning-based methodologies. These methods achieve a happy medium between conventional control and machine learning by adding human expert knowledge into the algorithms. It has been shown that hybrid models can be employed for precise path planning in highly variable settings (Lee et al., 2022).

2.2.4 Probabilistic Roadmaps (PRM)

In order to plan routes efficiently, many people have turned to probabilistic roadmaps (PRM). PRM creates an environment roadmap and then uses graph-based algorithms to find the best route. The real-world applications of this method have been quite successful (Kavraki et al., 1998).

2.2.5 Decision-based Motion Planning

The vehicle's activities are planned according to a series of decisions in decision-based techniques, which make use of decision theory. Decision-based planners can produce reliable routes for autonomous cars by factoring in environmental uncertainties (Sunberg et al., 2019).

2.2.6 Learning from Human Driving Data

Researchers have investigated the possibility of exploiting the vast amounts of available driving data to instruct autonomous vehicles in the art of motion planning. To better understand driver preferences and to generate more human-like pathways, learning from human driving behaviour is helpful (Codevilla et al., 2019).

2.2.7 Conclusion

Autonomous cars rely heavily on motion planning so that they can effectively and safely traverse unfamiliar terrain. The development of motion planning algorithms has benefited greatly from both traditional methods like RRT and cutting-edge ones like RL and hybrid models. To realize the ultimate objective of creating

autonomous vehicles that can reason and adapt to changing road conditions, it is necessary to combine motion planning with path identification utilizing machine learning techniques.

2.3 Pedestrian detection in autonomous driving

For the sake of pedestrian safety and the overall dependability of autonomous cars, the successful integration of pedestrian detection into autonomous driving is of utmost importance. In order to accurately detect pedestrians and predict their motions, cutting-edge sensing technology and **machine learning algorithms are used for pedestrian detection. With an eye toward the project** "Autonomous Vehicle Path Detection Using Machine Learning Algorithms," this literature review examines major academic contributions and achievements in pedestrian detection for autonomous driving.

Table 4 Comparison of algorithms

Table 4

Comparison of algorithms (Bu, Le, Du, Vasudevan, & Johnson-Roberson, 2020).

Algorithm	Miss rate (%)	FPPI	Average detection time per frame (s)
ACF	16.032	3.784	0.0611
LDCF	14.342	0.033	0.1255
CNN–AdaBoost	12. 778	0.021	0.0809

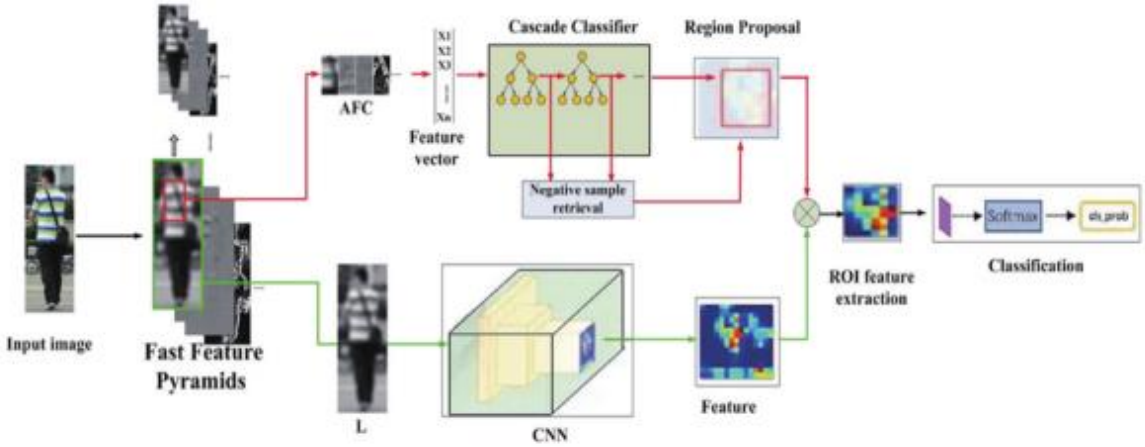


Figure 10 The architecture of the CNN–AdaBoost algorithm (Li, Zong, Liu, & Zhu, 2020).

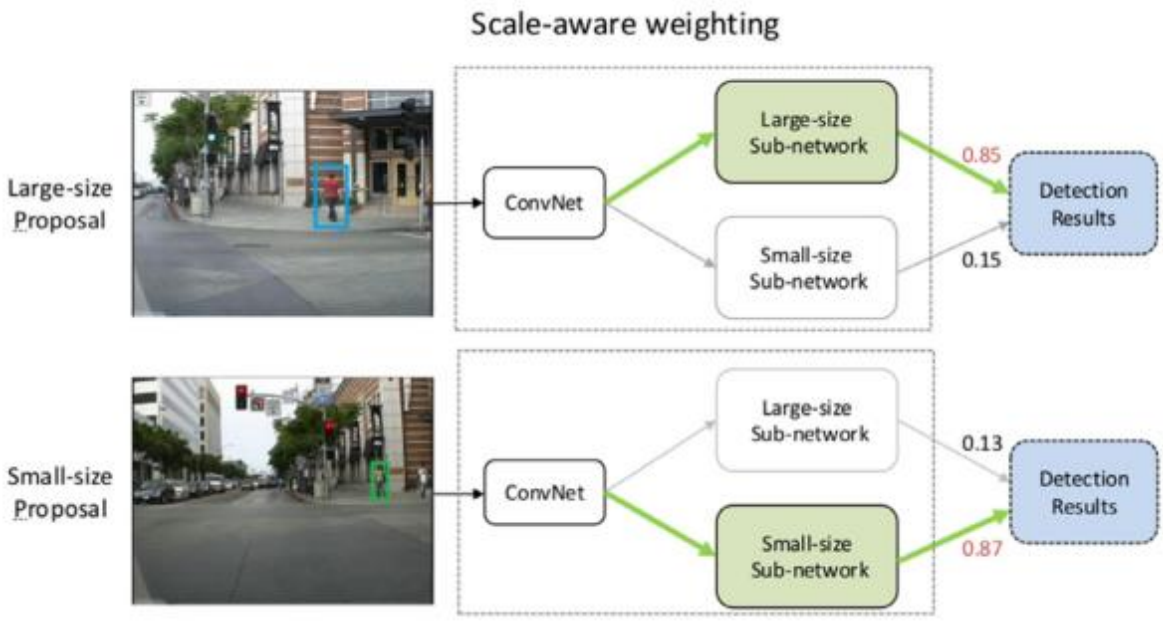


Figure 11 Scale-aware-weighting-mechanism of SAF R-CNN (Li et al., 2018).

2.3.1 Vision-Based Pedestrian Detection

Cameras and computer vision algorithms are used in vision-based pedestrian detection systems to identify people on the road. The development of deep learning has allowed for significant progress in these methods. In order to obtain cutting-edge performance in pedestrian identification tasks, scientists have turned to Convolutional Neural Networks (CNNs) (Ren et al., 2017).

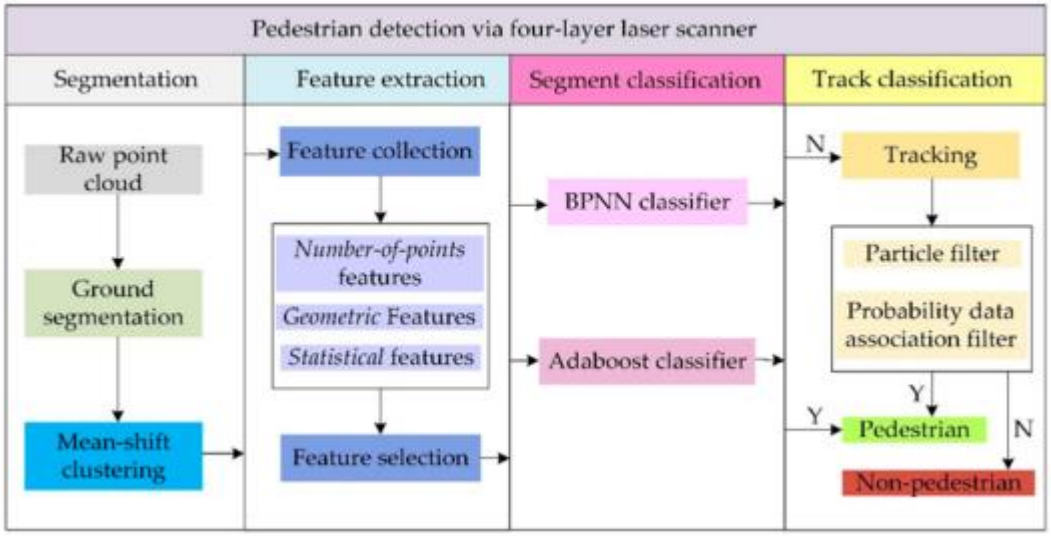


Figure 12 The architecture of pedestrian detection algorithm (Zhang et al., 2019).

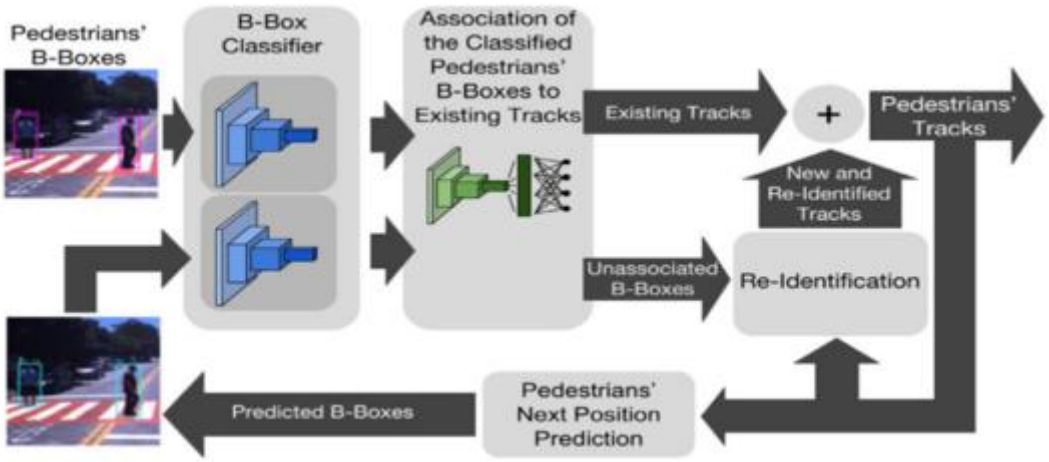


Figure 13 Pedestrian tracking system (Zhang, Yang, & Schiele, 2018)

Figure 14 Common Pipeline for pedestrian detection.

2.3.2 LiDAR-Based Pedestrian Detection

Accurate pedestrian recognition is made possible in low-light and adverse-weather settings by means of Light recognition and Ranging (LiDAR) sensors' 3D point cloud data. Xu et al. (2020) found that LiDAR-based methods, especially when paired with other sensors, provide excellent pedestrian detection.

2.3.3 Radar-Based Pedestrian Detection

When it comes to detecting pedestrians, radar sensors are important in driverless vehicles. Low-visibility settings are ideal for radar-based pedestrian detection systems, and these systems can work in tandem with other sensors to boost overall performance (Shen et al., 2019).

2.3.4 Fusion of Sensor Data

The integration of data from numerous sensors has been studied extensively with the goal of improving accuracy and reliability. Sensor fusion methods combine data from multiple sensors to generate a more complete picture of the environment and improve pedestrian recognition (Wang et al., 2022).

2.3.5 Real-Time Pedestrian Detection

Autonomous vehicles can't effectively respond to changing conditions without reliable real-time pedestrian detection. To accomplish quick and reliable pedestrian identification, researchers have developed novel algorithms and architectures including Single Shot Multibox Detector (SSD) and You Only Look Once (YOLO) (Redmon et al., 2016).

2.3.6 Uncertainty Estimation in Pedestrian Detection

Autonomous vehicles can't make sound decisions without the ability to estimate uncertainty. The vehicle may now evaluate its level of confidence in pedestrian detections thanks to recent efforts that have included uncertainty estimation into pedestrian detection models (Hu et al., 2021).

2.3.7 Conclusion

In order to protect the most defenceless road users, pedestrian detection is a crucial part of autonomous driving systems. Incredibly accurate pedestrian identification has been achieved by vision-based methods enabled by deep learning methods such as convolutional neural networks. Sensors like LiDAR and radar have also proven useful in hazardous settings. Sensor fusion approaches improve the overall performance and reliability of pedestrian detection by combining data from various sensors. Adding to the success of pedestrian detection in autonomous driving systems are real-time processing and uncertainty estimation. Research into pedestrian detection is likely to yield more advanced algorithms and enhanced safety measures for autonomous cars as the area of autonomous driving develops.

2.4 Traffic sign detection for autonomous driving

Safe navigation and legal observance of traffic laws rely heavily on autonomous driving system features like traffic sign detection. The purpose of this literature review is to provide light on recent developments in autonomous vehicle traffic sign identification, particularly as they relate to the ongoing project "Autonomous Vehicle Path Detection Using Machine Learning Algorithms."

Table 5 Pedestrian detection algorithms

Table 5
Summary of pedestrian detection algorithms.

Sr. no	Autonomous driving task	Algorithm name	Algorithm details	Advantages	Limitations
1	Pedestrian Detection	Aggregate Channel Feature (ACF)	1. ACF is a variation of channel features 2. ACF extracts features directly as pixel values	1. ACF provides a strong framework for pedestrian detection. 2. ACF has expedited detection speed	Not highly accurate
		Pedestrian Planar LiDAR Pose Network (PPLP Net)	PPLP Net consists of an "Orientation detection network (OrientNet)", a "Region Proposal Network (RPN)", and "PredictorNet."	PPLP Net provides inexpensive resolution for oriented pedestrian recognition problems.	–
		YOLO (You Lock Only Once)	YOLO applies a single CNN to the whole image, which further divides the image into grids.	1. YOLO uses single CNN for both classification and localization of the object 2. Architecture of YOLO makes it extremely fast.	Difficult to detect close and small objects

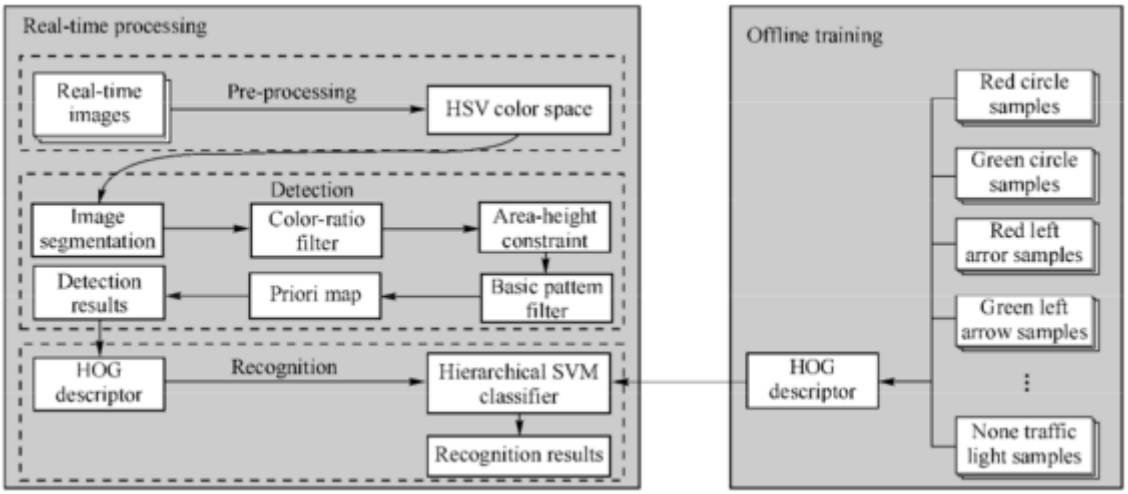


Figure 15 Processing Pipeline for real-time traffic light detector (Mu, Xinyu, Deyi, Tianlei, & Lifeng, 2015).

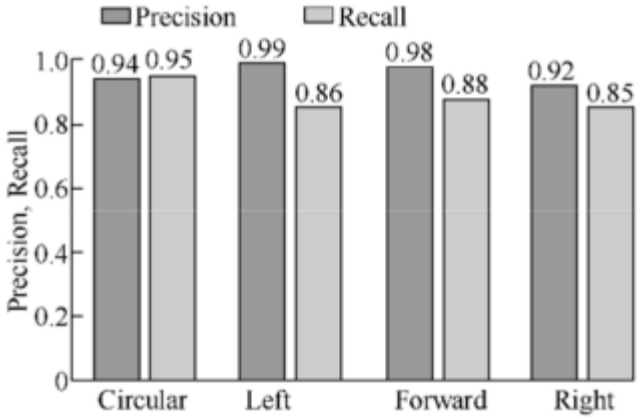


Figure 16 Graph of Precision and Recall-Stop Signal (Mu et al., 2015).

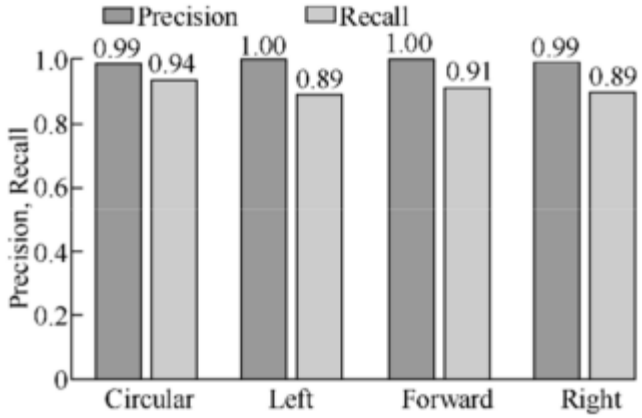


Figure 17 Graph of Precision and Recall-Go Signal (Mu et al., 2015)

2.4.1 Traditional Approaches

Early methods for detecting traffic signs included commonplace computer vision algorithms including colour filtering, edge detection, and template matching. Although these methods performed well in lab conditions, they were not able to adapt to real-world conditions such as changing lighting, weather, or occlusions (Alvarez et al., 2019).

2.4.2 Deep Learning-Based Approaches

Significant advancements were made in the area of traffic sign identification after the introduction of deep learning. In recent years, Convolutional Neural Networks (CNNs) have become the backbone of cutting-edge traffic sign detecting systems. Accurately identifying and categorizing traffic signs is a challenging task, but CNN-based designs have shown superior performance (Takikawa et al., 2020).

2.4.3 Dataset Augmentation Techniques

Having access to large and varied datasets is crucial to the performance of deep learning models. Data augmentation approaches have been used by researchers to artificially enlarge training datasets in order to address the problem of insufficient annotated data. Model generalization can be improved with the help of augmentation techniques including rotation, scaling, and flipping (Mogelmose et al., 2014).

2.4.4 Real-Time Traffic Sign Detection

In order to respond appropriately, autonomous vehicles need to be able to detect traffic signs in real time. Real-time performance is essential for making sound decisions and conducting safe navigation. In order to reduce latency in traffic sign identification, scientists have modified neural network designs and used hardware accelerators (Redmon et al., 2018).

2.4.5 Transfer Learning

Traffic sign detection is only one of several computer vision problems where transfer learning has proven its worth. Fine-tuning pre-trained models from large-scale datasets like ImageNet can improve detection accuracy and reduce the quantity of training data needed for traffic sign identification tasks (Yin et al., 2021).

2.4.6 Traffic Sign Recognition and Localization

Accurately locating the precise sign and comprehending the sign's content are also necessary for autonomous driving in addition to detecting traffic signs. Dey et al. (2016) argue that autonomous vehicles will only be able to properly comprehend and respond to traffic signs if they are equipped with integrated systems that include detection, localisation, and recognition algorithms.

2.4.7 Conclusion

For the sake of safety and legal compliance, autonomous driving systems must be able to identify traffic signs. The detection accuracy and resilience have been greatly enhanced by the shift from conventional computer vision methods to deep learning-based approaches. To compensate for data scarcity and boost generalization, data augmentation methods like transfer learning have become indispensable.
Researchers have made significant progress in improving neural network topologies and utilizing hardware acceleration, but real-time traffic sign detection remains a crucial challenge. Also, comprehensive traffic sign interpretation can be achieved with the help of integrated systems that feature detection, location, and identification capabilities.
More complex algorithms for detecting traffic signs are likely to emerge from ongoing traffic-sign detection research as autonomous cars develop, boosting both their effectiveness and the public's confidence in their safety.

2.5 Road-marking detection for autonomous driving

Identifying road markings is an important part of autonomous driving systems since it keeps vehicles in their lanes. Progress in detecting road markings for autonomous cars is the focus of this literature review, which was written in light of the project "Autonomous Vehicle Path Detection Using Machine Learning Algorithms."

Table 6 Traffic sign detection algorithms

Table 6
Summary of the traffic sign detection algorithms.

Sr. no	Autonomous driving task	Algorithm name	Algorithm details	Advantages	Limitations
1	Traffic Sign Detection	Histogram of Oriented Gradients (HOG)	Histogram of Oriented Gradients (HOG)is a feature descriptor	1. HOG is used to extract features from image data. 2. HOG descriptor focuses on the structure or the shape of an object.	HOG feature cannot adequately handle scale variation of pedestrians.
		Support-Vector-Machine (SVM)	SVM is a supervised machine learning algorithm for image classification	1. SVM is effective in high dimensional spaces 2. SVM is memory efficient	SVM is not suitable for large datasets.

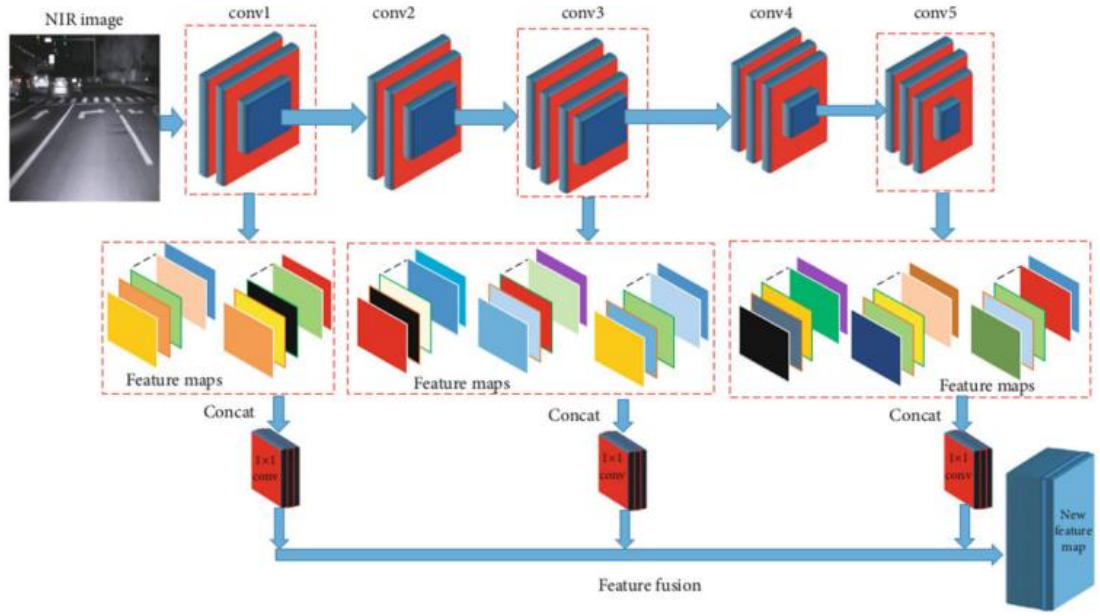


Figure 18 Feature fusion module (Hu et al., 2019)

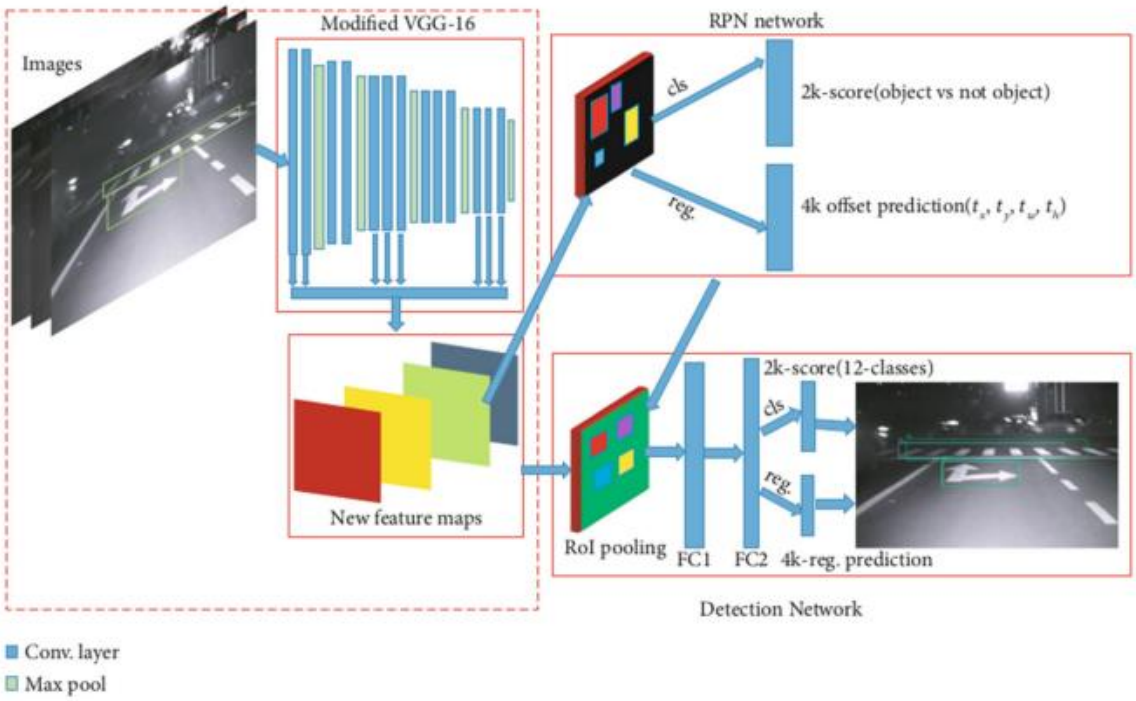


Figure 19 Pipeline for the Detection model (Hu et al., 2019).

2.5.1 Traditional Approaches

Edge detection, Hough transformations, and colour-based segmentation were among the first image processing techniques used in road-marking identification. However, the accuracy and robustness of these approaches were typically compromised by the presence of challenging road conditions, variable lighting, and occlusions (Wang et al., 2018).

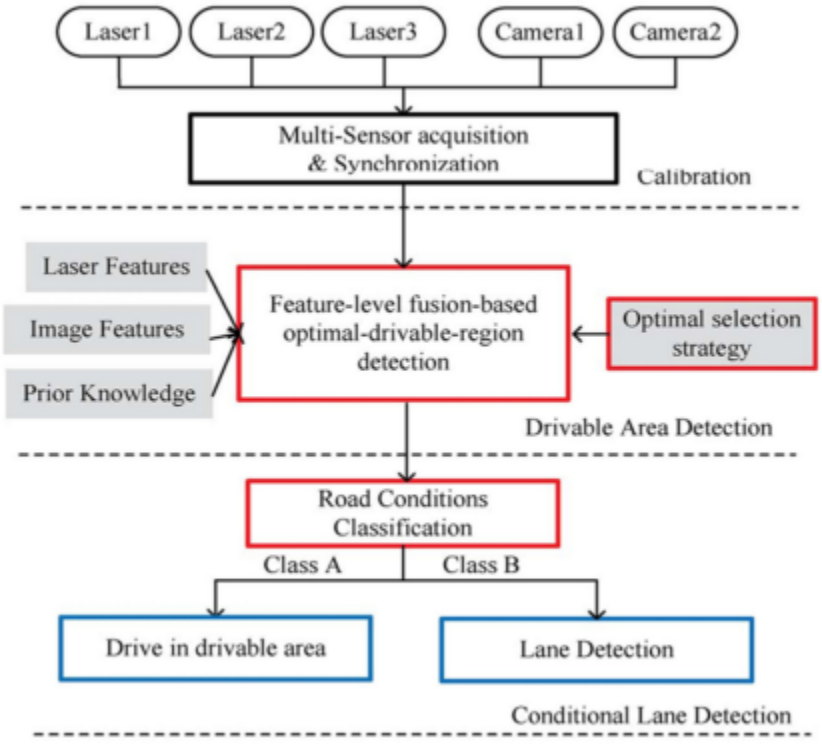


Figure 20 Flowchart for the Algorithm (Li et al., 2014)

Table 7 Road marking detection algorithm

Table 7
Summary of road-marking detection algorithm.

Sr. no	Autonomous driving task	Algorithm name	Algorithm details	Advantages	Limitations
1	Road-marking Detection	Faster R-CNN	Faster R-CNN is a deep convolutional network used for object detection	Faster R-CNN can accurately and quickly predict the locations of different objects.	Faster R-CNN requires many passes through a single image to extract all the objects.

2.5.2 Machine Learning-Based Approaches

The development of deep learning and other machine learning methods has completely altered road-marking detection. In order to accurately detect road markings, Convolutional Neural Networks (CNNs) have demonstrated exceptional performance in extracting complex characteristics from road photos (Zhang et al., 2020).

2.5.3 Dataset Creation and Augmentation

Having access to a variety of well-annotated datasets is crucial to the performance of deep learning models for road-marking recognition. In order to properly train deep learning models, researchers have gathered large-scale datasets containing photographs of roads with accurately annotated road markers. There have also been efforts to increase model generalization by augmenting the dataset with data augmentation techniques like rotation, scaling, and viewpoint transformations (Kuo et al., 2019).

2.5.4 Real-Time Detection

Detecting road markings in real time is crucial for autonomous driving applications. Researchers have refined CNN architectures and implemented hardware accelerators to achieve low latency and real-time performance (Li et al., 2021). This makes it possible for autonomous vehicles to recognize road markings immediately.

2.5.5 Multi-Task Learning

Detection of road markings has been combined with other perception tasks, such as lane detection and traffic sign recognition, in several studies. By reusing data from one job in another, this strategy could boost autonomous vehicles' perceptive abilities in general (Chen et al., 2019).

2.5.6 Semantic Segmentation

In order to annotate road sections and markings at the pixel level, semantic segmentation algorithms have been applied to road-marking detection. Accurate semantic segmentation has been achieved using Fully Convolutional Networks (FCNs) and U-Net architectures, allowing for a holistic comprehension of the road environment (Wang et al., 2022).

2.5.7 Conclusion

The recognition of road markings is an integral part of autonomous driving systems since it aids in vehicle localization and route planning. Improved precision and reliability in road-marking recognition can be attributed to the shift from conventional image processing methods to those grounded in machine learning, in particular deep learning.

- Training efficient deep learning models has been greatly aided by the accessibility of annotated datasets and data enrichment approaches. Although real-time road-marking identification is still difficult, it has made significant strides because to ongoing attempts to optimize neural network topologies and use hardware accelerators.
- In addition, there are untapped opportunities to enhance road-marking detection and other perception tasks in autonomous driving by investigating multi-task learning and semantic segmentation approaches.
- Research into road-marking detection could lead to more advanced algorithms, improving the safety and effectiveness of autonomous driving systems as a whole as the technology for such vehicles develops.

2.6 Self-localization in autonomous driving

Autonomous driving systems rely heavily on self-localization so that cars can pinpoint their position and direction in space. Progress in autonomous vehicle self-localization is explored in this literature review for the purpose of informing the project "Autonomous Vehicle Path Detection Using Machine Learning Algorithms."

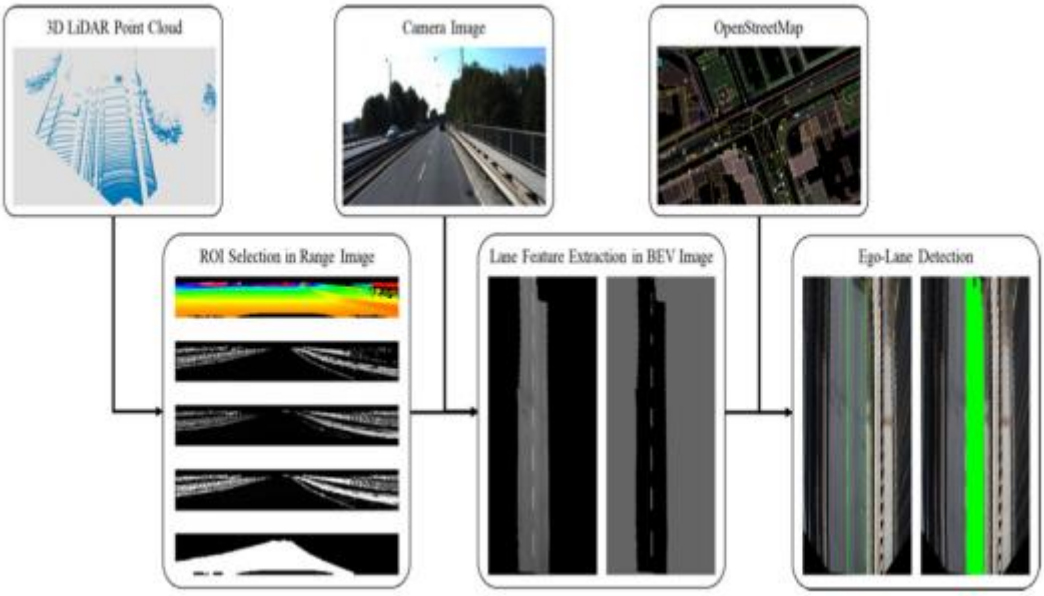


Figure 21 Framework for MELD method (Wang et al., 2020).

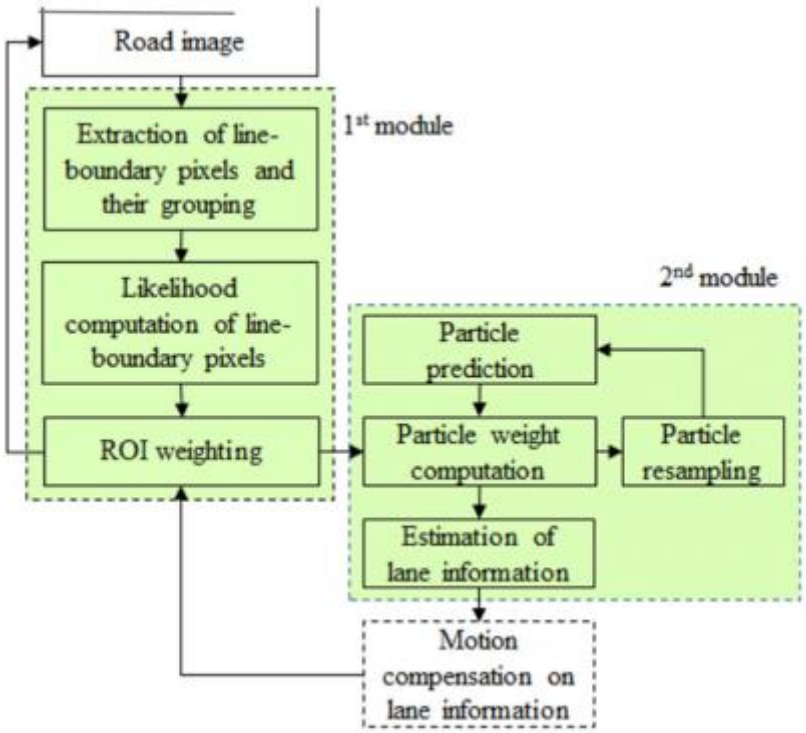


Figure 22 Flow chart for the lane-estimation algorithm (Park & Lee, 2018).

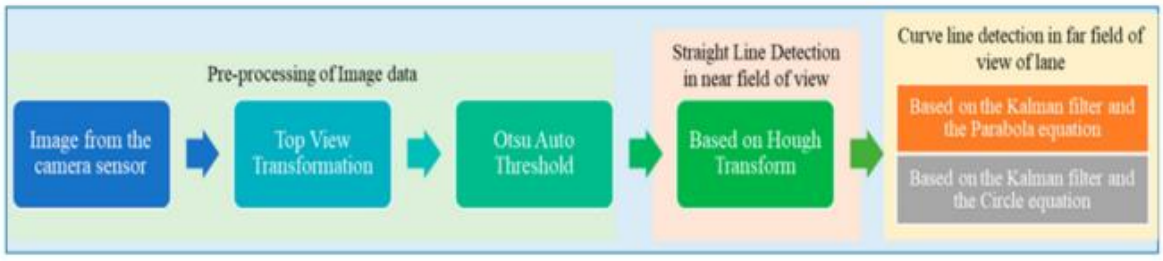


Figure 23 Flow chart of the lane recognition algorithms using the Kalman filter (Dorj et al., 2020).

2.6.1 Global Positioning System (GPS)-Based Localization

Many self-driving cars now on the road rely heavily on GPS for localization purposes. Although GPS can be relied upon to deliver precise location data, it may not function as expected in urban canyons, tunnels, or other regions with low satellite visibility (Huang et al., 2017). In order to achieve accurate self-localization, it is necessary to combine GPS with other localization methods.

2.6.2 Simultaneous Localization and Mapping (SLAM)

The commonly used method of "simultaneous localization and mapping" (SLAM) allows vehicles to generate maps of their environments while also pinpointing their precise location within those maps. To build a map and estimate the vehicle's position inside it, SLAM algorithms take readings from a variety of sensors, including LiDAR, cameras, and odometry (Cadena et al., 2016). Even in areas without access to GPS, SLAM provides reliable and precise positioning.

2.6.3 LiDAR-Based Localization

Due to their ability to produce very accurate 3D point clouds of their surroundings, Light Detection and Ranging (LiDAR) sensors have becoming more popular for use in autonomous driving. With the help of scan matching and point cloud registration techniques, LiDAR-based localization algorithms can precisely pinpoint a vehicle's location by correlating real-time LiDAR readings with previously constructed maps (Pfaff et al., 2019).

2.6.4 Visual Odometry

By monitoring the movement of visual features acquired by cameras positioned on the vehicle, visual odometry predicts the vehicle's ego-motion. Visual odometry can estimate the vehicle's location and orientation changes over time by following important features between consecutive frames (Mur-Artal et al., 2017). Real-time localization is possible with visual odometry, although it is inaccurate due to changes in lighting and other environmental factors.

2.6.5 Sensor Fusion

Data from several sensors, including as global positioning systems (GPS), light detection and ranging (LiDAR), cameras, and inertial measurement units (IMUs), are fused using sensor fusion techniques to improve localization precision and reliability. Using Kalman filters and particle filters, which are popularly used for sensor fusion, vehicles can take use of each sensor's strengths while mitigating their limitations (Sun et al., 2020).

2.6.6 Deep Learning-Based Localization

The use of deep learning strategies for autonomous vehicle self-localization has also been investigated. Modelling temporal connections in sensor data has been shown to increase localization precision using Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks (Li et al., 2019). To further streamline the localization process, end-to-end learning approaches have been implemented, which directly map sensor inputs to vehicle positions (Xu et al., 2021).

2.6.7 Conclusion

Autonomous cars' ability to accurately and reliably determine their own locations is a crucial feature of advanced navigation systems. Although GPS has traditionally been used for localization, it has limitations in some settings that require the use of additional methods such as simultaneous localization and mapping (SLAM), LiDAR-based localization, visual odometry, and sensor fusion.

- In dense metropolitan areas, the introduction of LiDAR sensors has greatly enhanced localisation precision. The use of sensor fusion algorithms to combine data from several sensors for accurate localisation has also been proved to be beneficial.
- Furthermore, the use of deep learning techniques in localization tasks shows promise in directly mapping sensor inputs to position estimates and capturing complicated temporal correlations.
- More complex algorithms for self-localization are likely to emerge as autonomous driving technology develops, improving the safety and performance of these cars.

2.7 Automated parking for autonomous vehicle

Automated parking is a vital component of autonomous driving technology since it allows vehicles to park themselves without human intervention. This research paper is a literature analysis of recent developments in automatic parking systems for autonomous vehicles, as they relate to the study "Autonomous Vehicle Path Detection Using Machine Learning Algorithms."

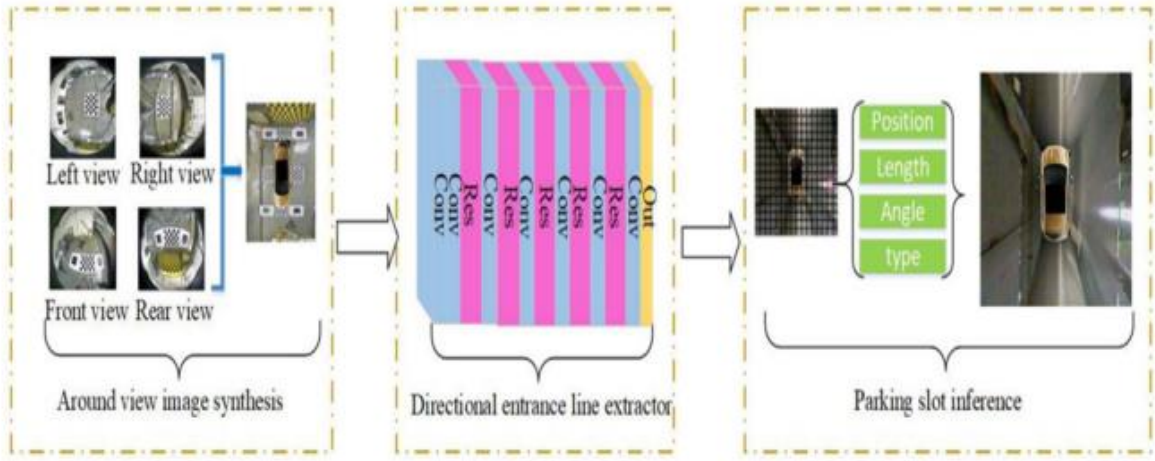


Figure 24 Parking slot detection using a DCNN detector (Li et al., 2020).

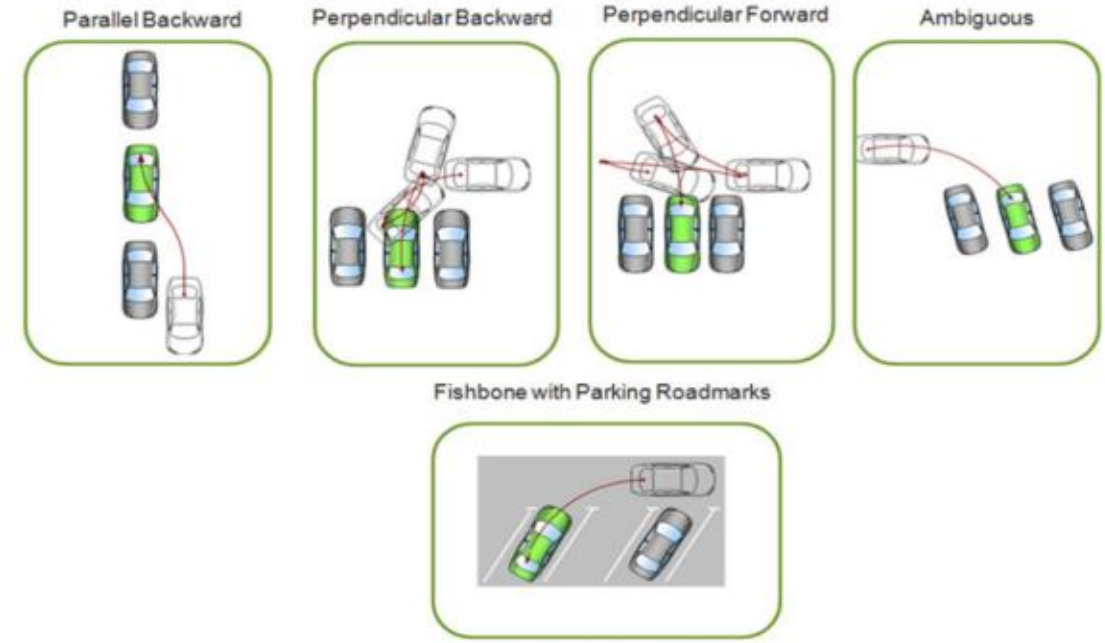


Figure 25 Different parking scenarios (Heimberger et al., 2017).

Figure 26 Parking slot marking recognition (Heimberger et al., 2017)

2.7.1 Traditional Automated Parking Techniques

Ultrasonic sensors and cameras, among others, have traditionally been used by automated parking systems to identify obstructions and direct the car into a parking slot (Kawaguchi et al., 2016). Rule-based algorithms, on which these earlier systems were built, were not flexible enough to handle complex parking circumstances or difficult settings.

2.7.2 Sensor Fusion for Enhanced Perception

Researchers have looked into sensor fusion techniques, which combine data from various sensors like LiDAR, cameras, and radars, to overcome the constraints of rule-based systems. It has been shown that automobiles' parking accuracy and safety can be increased by combining data from many sensors (Lee et al., 2018).

2.7.3 Path Planning Algorithms

Automated parking systems cannot function without efficient path planning algorithms. In order to construct feasible and collision-free pathways for the car during parking manoeuvres, researchers have proposed a number of different approaches, such as probabilistic roadmaps (PRMs) and rapidly exploring random trees (RRTs) (Latif et al., 2020). These algorithms guarantee a safe and timely arrival at the designated parking location.

2.7.4 Machine Learning-Based Parking Systems

There has been a rise in the use of machine learning in automatic parking systems. Vehicles' parking abilities have been enhanced through the use of reinforcement learning algorithms that permit them to learn from trial and error (Kucukyilmaz et al., 2019). To further aid in object detection and obstacle avoidance while parking, deep learning models have been applied to extract useful features from sensor data (Zhao et al., 2021).

2.7.5 Valet Parking and Infrastructure Integration

With the help of valet parking systems, driverless cars can park themselves in garages and lots. For precise navigation inside the parking lot, these systems frequently require two-way interaction with elements of the parking infrastructure, such as sensors built into parking spots or guiding systems (Wei et al., 2020).

2.7.6 Real-Time Parking Reconfiguration

Vehicles' parking strategies can be reworked in real time to take into account the number and location of open spots as well as the parking lot's layout (Altaf et al., 2019). In dense metropolitan areas, this method can greatly improve parking efficiency.

2.7.7 Conclusion

Drivers and passengers can benefit greatly from automated parking, which is why it is an integral part of autonomous driving. Rule-based techniques and inadequate sensor technology hampered early automated parking systems. Recent developments in sensor fusion, path planning algorithms, and machine learning approaches have helped automatic parking systems become more accurate and flexible.

- Using machine learning and sensor fusion, vehicles can form a complete picture of their surroundings and make informed choices when parking. More sophisticated valet parking services and real-time parking reconfiguration are made possible by the incorporation of communication with parking infrastructure.
- Research into automated parking systems will likely continue to hone in on improving path planning algorithms, increasing parking efficiency, and solving the difficulties of parking in highly congested urban locations as autonomous driving technology advances.

3 Methodology

3.1 Dataset description

Dataset link: https://www.kaggle.com/competitions/lyft-motion-prediction-autonomous-vehicles/data

The Lyft dataset, which is referenced in the text, consists of these three parts:

There are 170,000 scenes recorded by autonomous vehicles, each lasting 25 seconds. These snapshots document the self-driving car's motion, that of other vehicles, pedestrians, and bicycles, and the status of traffic signals. The images were shot along a 6.8-mile stretch of road connecting the Palo Alto train station and Lyft's level 5 office.

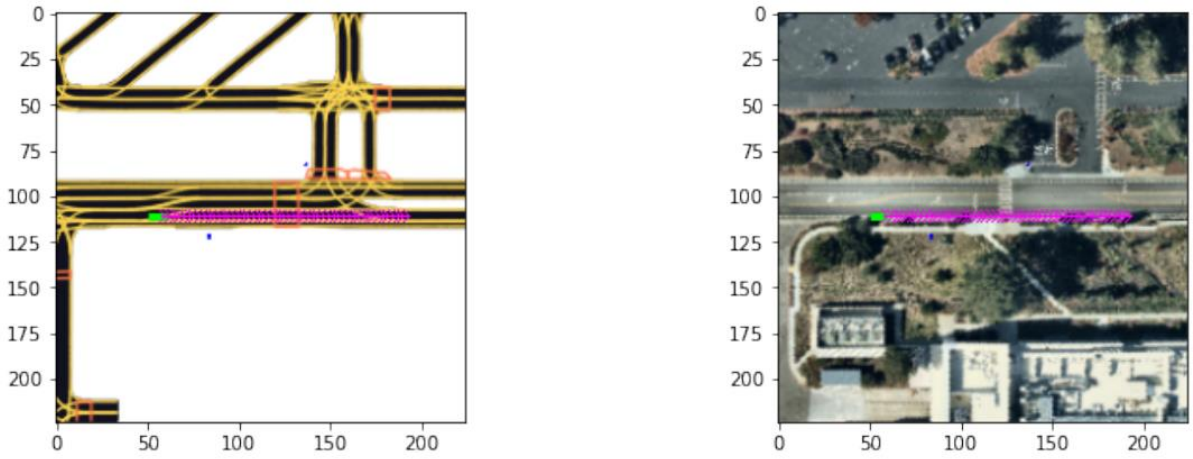


Figure 27 Semantic map segment & Satellite map segment

Detailed information about road regulations, lane layout, and other traffic elements is included in the dataset's high-definition semantic map. It's useful for creating predictions about people's actions because of the centimetre-grade accuracy it delivers.

Additionally, a high-resolution aerial image of the area immediately adjacent to the route is provided. With a pixel size of 6 centimetres, this image is useful for spatial information analysis and can be used to make predictions about the subject's behaviour.

All vehicles, pedestrians, and cyclists in a given scene are represented as 2.5D cuboids and tagged with information such as their speed, acceleration, yaw angle, yaw rate, and class name.

The primary objective of the Lyft competition, which makes use of the Lyft dataset, is to forecast the x and y positions of a given traffic participant using just their current location and past behaviour. The predictions should be sampled at 10Hz, leading to a 50-coordinate array spanning a 5-second time horizon in x and y.

3.2 Problem Statement

To win the challenge, entrants must develop prescriptive models that account for the past behaviours of all traffic participants over a predetermined interval of time (up to 99 steps at 10Hz). This background information establishes norms and trends for the actions of all involved parties.

Uncertainty in making predictions is intrinsic to the endeavour because of human nature. As a result, Lyft encourages its users to provide a range of probable routes for each sample, along with an indication of how

confident they are in each option. The purpose of this multi-faceted method of forecasting is to take into account all possible outcomes for each individual.

The competition uses the negative log-likelihood as the statistic of choice for judging submissions. It is determined by contrasting the submitted multi-modal forecasts with the actual data. A lower negative log-likelihood indicates that the predictions are more in line with the data, which is good for performance. An ideal score of 0 would suggest that there is a single path that can be taken with complete assurance because it corresponds exactly to the real world.

3.3 Data Collection

In this phase, information is gathered from autonomous cars' sensors in the actual world. Images from cameras, lidar readings, and any other pertinent data will be included. One example of a publicly available dataset utilized in this research is Huawei Corp.'s ONCE Dataset (One million scenes). Machine learning algorithms can then be trained and evaluated using the obtained data.

3.4 Data Preprocessing

Preprocessing the data ensures its quality and gets it ready for analysis after it has been obtained. Removing noise, standardizing sensor data, and aligning numerous sensor modalities are all steps in this process. The data should be cleaned so that it may be used by the machine learning models.

3.5 Algorithm Exploration and Selection

Path Detection Algorithm Exploration and Evaluation Various machine learning algorithms are investigated and assessed for their applicability to autonomous vehicle path detection. Convolutional neural networks (CNNs) and recurrent neural networks (RNNs) are popular examples of algorithms utilized in this setting. The algorithms are evaluated based on how well they can recognize and forecast routes, as well as any limitations they may have. Metrics like precision, accuracy, recall, and F1 score are used to compare the efficacy of each algorithm.

3.6 Performance Evaluation

After the machine learning algorithms have been trained, their effectiveness is measured. This requires intensive testing of the models and comparisons to real-world data. The efficacy of the trained path identification model is measured using evaluation measures like accuracy, precision, recall, and F1 score. The model's detection and prediction accuracy of roads, lanes, traffic signs, and obstacles are evaluated.

3.7 Fine-tuning and Optimization

It is possible to fine-tune and optimize the trained model at this stage. To enhance the precision and scalability of the model, it is possible to experiment with varying hyperparameters, trying out different architectures, or adding in new types of data augmentation. The route identification model is improved through this iterative approach so that it performs better in a wider range of driving conditions.

3.8 Convolutional Neural Network (CNN)

To handle input with a grid like structure, like pictures or spectrograms or time-frequency representations, a popular neural network architecture is the convolutional neural network (CNN). When it comes to identifying spatial patterns and local correlations in the input data, CNNs excel.

The convolutional layer is the heart of a convolutional neural network (CNN), as it is responsible for performing convolution operations on the input data with the help of a collection of trainable filters or kernels. These filters move across the input, multiplying and summing elements at each node to extract local features. By performing this action, the network is able to automatically learn a hierarchy of representations for the input, progressing from simple features (like edges) to more complicated ones (like complex shapes).

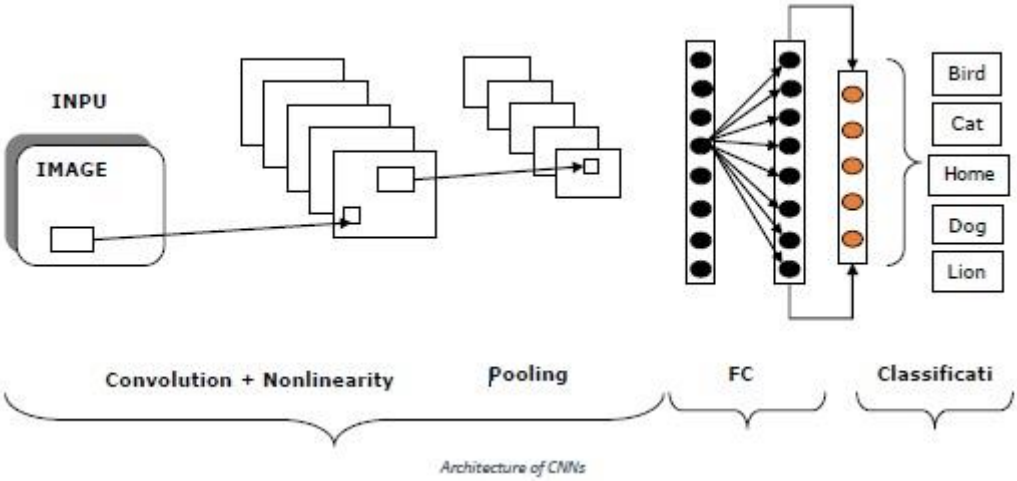


Figure 28 CNN architecture

Multiple convolutional layers are common in CNNs, with intervening pooling layers that scale down the spatial dimensions and further generalize the retrieved features. Fully connected layers are typically used in the final stages of a neural network to map the retrieved information to the desired output.

CNNs can interpret sensor data, including images captured by cameras, to aid in the path detection of autonomous cars. The CNN may be trained to automatically learn the visual elements necessary for reliable path identification, including as lane lines, road signs, and obstacles. CNNs are able to successfully detect and distinguish essential features of the road environment by studying local patterns and spatial correlations within the input data.

Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs) can be combined in practice to form a hybrid architecture known as an RCNN. This combination makes the network well-suited for sequential and grid-like data applications, such as path recognition in autonomous cars, by allowing it to capture both time dependencies and spatial patterns.

Both recurrent neural networks and convolutional neural networks have shown great success in a variety of fields. They are powerful tools for solving difficult challenges in autonomous vehicle technology because of their adaptability, ability to learn complicated representations, and capacity to handle large-scale datasets.

3.9 Multi Modal Predictions

Instead of making a single, deterministic prediction about a circumstance or activity, multi-modal predictors consider a range of potential outcomes and approaches. To account for the unpredictability of human behaviour, the context of motion prediction for autonomous cars requires the generation of multiple possible trajectories or actions for other traffic participants on the road.

A model's forecast of a traffic participant's future motion is limited to a single trajectory in the case of traditional single-mode prediction. Human conduct, however, is often inconsistent and unclear. If a car is

spotted traveling in the right lane, it could keep going straight, make a right turn, or switch lanes at the next intersection, depending on the circumstances. In order to build reliable motion prediction models, it is necessary to take all of these factors into account.

In order to win the Lyft competition, you'll need to implement multi-modal prediction, which includes extending the model to create three or more pathways for each traffic participant. Each route is a possible future action that the traffic player could take. The model also provides a confidence score for each of the projected trajectories, indicating the likelihood of that particular path.

Multi-Modal Prediction's advantages include:

- More accurate predictions can be made since the model generates numerous possible pathways in response to the inherent ambiguity in human behaviour.
- Knowing the range of probable actions taken by other road users is useful for risk assessment and action planning in autonomous driving scenarios.
- Autonomous vehicles can use multi-modal prediction to make better judgements because it takes into account more information.
- Safety is enhanced by taking into account the wide range of driving styles of other motorists.
- The Lyft competition use the loss function of negative log-likelihood (NLL) to measure the efficacy of the multi-modal prediction model. The NLL evaluates the performance of the model by rewarding increased confidence in the accurate predictions and punishing those who are incorrect.

When applied to motion prediction models for autonomous vehicles, Multi-Modal Prediction represents a significant step forward since it allows these vehicles to account for the inherent unpredictability in human behaviour and consequently make more trustworthy and secure driving decisions.

3.10 Resnet model

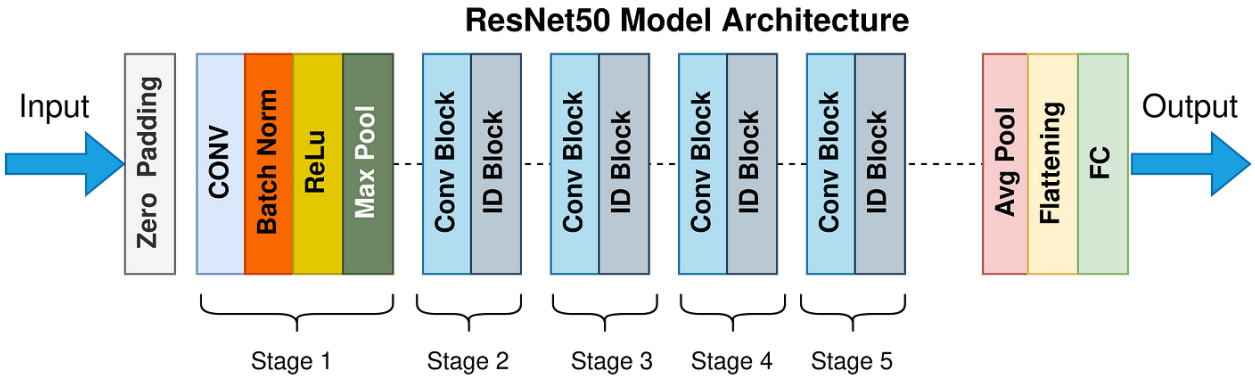


Figure 29 Resnet model

ResNet (short for Residual Neural Network) is a deep learning architecture that has seen extensive use and great success in a number of computer vision tasks, such as image identification and object detection. Kaiming He et al. presented it in their 2015 work "Deep Residual Learning for Image Recognition." I will describe what ResNet is and how it might be applied to the context of motion prediction for Lyft's

autonomous driving duties, even if the text you gave does not directly reference ResNet being utilized for Lyft.

ResNet was designed to fix the issue of vanishing gradients in extremely deep neural networks. Training a deeper network result in slower convergence and worse performance because gradients have more trouble flowing backward through the layers. ResNet implements the idea of residual learning to solve this problem.

Every layer in a classic neural network is responsible for calculating a different transformation of the input, and each layer's output is directly connected to the layer below it in the network. However, a ResNet's residual block features an identity shortcut connection that skips across several layers. The input is added to the residual block's output via this quick link. The result of a residual block can be expressed mathematically as:

Input + F(input) equals output

where "F(Input)" stands for the transformation that the residual block of layers has learned. By teaching the model the difference (or residual) between the input and the target output, the residual block facilitates the network's approximation of the underlying mapping.

By avoiding the vanishing gradient problem, ResNet allows for the training of considerably deeper neural networks, generally with over a hundred layers. Research has revealed that networks that are able to capture more complex elements and patterns in the input data perform better overall.

ResNet could be used to construct more robust and expressive models for anticipating the future trajectories of nearby cars and pedestrians in the context of motion prediction for autonomous driving. Accurate and reliable motion prediction might sometimes be dependent on the model's ability to understand complicated relationships in the data, a task made easier by the ResNet architecture's residual blocks.

The article only makes passing references to ResNet, the baseline model, and NLL loss. Although ResNet and related deep learning architectures are typically employed in computer vision applications, they may be useful in autonomous driving scenarios such as Lyft's to enhance accuracy and performance in motion prediction.

4 Requirements

4.1 Functional Requirements

4.1.1 Data Collection

- The system ought to be able to gather sensor data from the real environment, such as images captured by cameras and lidar data, in addition to other pertinent measures taken by autonomous cars.
- The procedure for collecting the data need to guarantee the reliability and precision of the information obtained.
- To make the process of training and evaluating candidates more effective, the system should allow for the integration of datasets that are open to the public, such as the ONCE Dataset.

4.1.2 Data Preprocessing

- The system has to have preprocessing methods built into it so that it can clean and preprocess the data that was obtained.
- The elimination of noise, the normalization of sensor readings, and the alignment of several sensor modalities are all necessary steps in the preprocessing of data.
- When the data will be used for further analysis and training, the preprocessing methods should guarantee the data's high quality and compatibility.

4.1.3 Algorithm Exploration and Selection

- It should be possible to investigate and assess a variety of machine learning methods that are suited for the path identification of autonomous vehicles using the system.
- It should be able to facilitate the comparison of different algorithms, such as Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and any other pertinent algorithms, depending on their respective performance criteria.
- The system should make it easier to pick the algorithms that are best suited for the job of identifying the path in the most precise and effective way possible.

4.1.4 Training and Model Development

- The system should make it possible to train machine learning models utilizing the various methods that have been chosen.
- It should provide the tools and libraries required to construct and improve path recognition models.
- In order to produce more accurate models, the training procedure should allow for the addition of road characteristics such as lane markings, traffic signs, and barriers. This should be supported by the system.
- In order to improve the trained models' overall performance, it ought to make model optimization and hyperparameter adjustment more straightforward.

4.1.5 Performance Evaluation

- The system ought to have assessment metrics so that its trained path detection models can be evaluated as to how well they work.
- Calculating metrics such as accuracy, precision, recall, and F1 score are necessary in order to evaluate how well the model can recognize and forecast road paths, lane markings, traffic signs, and impediments.
- The system needs to be able to provide a comparison study of the various models based on the performance indicators that they have.
- It should make it possible to benchmark the trained models against data representing the ground truth in order to assess the correctness and dependability of the models.

4.1.6 Fine-tuning and Optimization

- The trained models should be able to be fine-tuned within the system in order to improve both their accuracy and their scalability.
- It should offer possibilities to tweak hyperparameters, investigate alternative topologies, and put data augmentation strategies into practice.

- It should be possible to execute iterative optimization within the system in order to improve the performance of the path detection models and their adaptability to a wide variety of driving circumstances.

4.2 Non-Functional Requirements

4.2.1 Performance

- It is expected that the system would be able to handle big datasets quickly and effectively.
- It should be able to handle information quickly so that autonomous vehicles can do path detection in real time or very close to real time.
- The system ought to be optimized in order to decrease the demand for computational resources while still retaining a high level of accuracy.

4.2.2 Scalability

- The system ought to be scalable in order to manage ever-increasing quantities of sensor data and to accommodate further technological advances in autonomous vehicles in the future.
- To scale effortlessly with ever-increasing datasets, it should support distributed computing and parallel processing approaches.

4.2.3 Reliability

- The system should be dependable and sturdy, and it should be able to generate correct findings for route detection in a variety of different driving scenarios.
- It should elegantly handle unforeseen events and loud sensor inputs, which will ensure that its performance is consistent.
- During the stages of data preprocessing, training, and evaluation, the system ought to integrate systems that can discover and deal with mistakes or anomalies in the data.

4.2.4 Security and Privacy

- The system should integrate security measures to secure sensitive data, ensuring data privacy and compliance with applicable legislation. These methods should be designed to protect data from unauthorized access.
- It ought to have procedures for the encryption of data, the control of access, and the safe storage of sensitive information.

4.2.5 Usability

- It is important that the system have an intuitive user interface, as this will make it much simpler for researchers and developers to connect with the system.
- It ought to provide lucid documentation and straightforward directions on how to make efficient use of the system.
- The system ought to include visualization capabilities so that the training and evaluation results may be interpreted and analysed more effectively.

4.2.6 Maintainability

- It is important that the system be designed with modularity and code reusability in mind, since this will make it much simpler to perform maintenance and software updates.
- It is important that it be fully documented and that it provides clear guidelines for any future alterations or enhancements.
- The system should be compatible with widely used development frameworks and libraries. This will ensure that it will be supported for the long term and will make it easier to integrate with other tools and technologies.

4.2.7 Ethical Considerations

- In the process of data gathering, utilization, and model training, the system ought to comply with ethical norms and established best practices.
- Fairness, accountability, and openness ought to be given top priority in order to prevent bias and guarantee the appropriate implementation of autonomous vehicle path detecting systems.

In conclusion, the functional requirements centre on important topics such as data collection, preprocessing, algorithm discovery, machine learning model training, evaluation, and fine-tuning. The non-functional requirements, on the other hand, include things like performance, scalability, dependability, security, usability, maintainability, and ethical considerations. Taking these parameters into account will be a significant step toward the creation of a reliable and efficient system for detecting the paths taken by autonomous vehicles using machine learning techniques.

5 Implementation

5.1 Importing Libraries

The script begins by bringing in the required libraries and modules. Data management, neural network simulation, visualization, and other related activities will all make use of these libraries. Each import will be briefly described below.

- **typing:** Python's type hints can be used with the help of this package.
- **tempfile.gettempdir:** The function retrieves the full path to the temporary directory.
- **matplotlib.pyplot:** It is a library for making charts and graphs in Python.
- **numpy:** When it comes to numerical computations involving arrays and matrices, numpy is an extremely useful tool.
- **pandas:** It is a package for analyzing and manipulating data, especially structured data.
- **torch:** It is the most widely-used framework for constructing and training neural networks in deep learning.
- **torch.nn and torch.optim:** These modules house different parts used in the creation and refinement of neural networks.
- **torch.utils.data.DataLoader:** To efficiently load data in batches while training, data loaders can be crafted using the torch.utils.data.DataLoader class.
- **torchvision.models.resnet:** This module contains several ResNet topologies that can serve as the foundation for a transfer learning task.
- **tqdm:** The library provides a status indicator for monitoring the development of iterative processes.
- **l5kit:** The Lyft Level 5 Dataset is the focus of this collection of programs known as "l5kit."

```
from typing import Dict

from tempfile import gettempdir
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import torch
from torch import nn, optim
from torch.utils.data import DataLoader
from torchvision.models.resnet import resnet50, resnet18, resnet34, resnet101
from tqdm import tqdm

import l5kit
```

Figure 30 Import Library

5.2 Setting Seed Function

The set_seed() function is defined so that the random seed can be set in a way that ensures repeatable results. This is especially crucial in machine learning applications, where outcomes can be affected by random processes such as data reshuffling or the initialization of neural network weights. The script guarantees repeatable outcomes from its random operations by using the same seed for each execution.

```
def set_seed(seed):
    random.seed(seed)
    np.random.seed(seed)
    os.environ["PYTHONHASHSEED"] = str(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)

set_seed(42)
```

Figure 31 Set Seed Function

5.3 Configuration Parameters (cfg)

The script creates a dictionary called cfg that stores the various configuration parameters required in the learning and prediction phases. Model architecture, dataset routes, batch size, learning rate, rasterization parameters, and other such variables are examples of such settings. The parameters can be modified as needed without affecting other parts of the code if they are configured in a single place.

5.4 Data Loading and Preprocessing

Here, we will learn how to use the l5kit package to load both the training and test datasets. The script use the ChunkedDataset class to access the datasets and the LocalDataManager class to handle the data pathways. Additionally, the build_rasterizer function is used to rasterize the data so that it can be fed into the neural network. The following are the stages of the process:

```
DIR_INPUT = cfg["data_path"]
os.environ["L5KIT_DATA_FOLDER"] = DIR_INPUT
```

Figure 32 Access the Dataset

- Specify the location of the data files by calling os.environ["L5KIT_DATA_FOLDER"].
- The data pathways can then be managed by initializing the LocalDataManager object.
- Make use of open() and ChunkedDataset to load both the training and testing datasets.
- Convert the raw data into a dataset (AgentDataset), naming the rasterizer and the agents' masks.

5.5 Visualization Function

With the help of the visualize_trajectory() tool, users can observe a chosen trajectory from the training dataset. It takes as input a dataset and an index, and then depicts the trajectories of target points on a scene, along with an image of that scene. Data and model predictions are greatly aided by visualization.

```
visualize_trajectory(train_dataset, index=90)
```

Figure 33 Visualization of Trained Data

5.6 Loss Function Definition

The script creates a unique loss function, pytorch_neg_multi_log_likelihood_batch(), and names it accordingly. The loss in negative multi-log likelihood for the model predictions is computed by this function. Predicted trajectories and confidence scores, both of which are important in the motion prediction competition, are factored into the loss function. Here are the measures that make up the loss calculation:

- Validity checks and correct shapes for input tensors should be implemented.
- Use the agent's posture transformation to map the ground-truth coordinates to the world-space coordinate system.
- Find the square of the discrepancy between the expected and actual positions.

- Loss should be estimated using log-likelihood and mode reduction based on confidence scores.

```
return pytorch_neg_multi_log_likelihood_batch(gt, pred.unsqueeze(1), confidences, avails)
```

Figure 34 Loss Function

5.7 Model Definition

The neural network model, LyftMultiModel, is defined in the script. This model, built on the ResNet framework, is able to generate confidence scores for various probable future trajectories. A ResNet architecture serves as the model's backbone, while linear transformations are used in the model's "head" layer. The following procedures make up the architecture:

- Create an instance of the desired ResNet backbone model using its pre-trained weights (for example, ResNet18, ResNet34, or ResNet50).
- Adjust the core infrastructure (such the amount of input channels) so that it works with the data type being fed into it.
- Improve prediction and confidence estimation by including supplemental layers (such as linear layers).

```
class LyftMultiModel(nn.Module):
 if architecture == "resnet50":
```

Figure 35 Model Definition

5.8 Training Loop

To calculate predicted trajectories and confidences, use the forward pass.
The script will join the training loop if the train configuration parameter is set to True. To refine the model, the loop iteratively processes the training data, both forward and backward. These are the steps that make up the training process:

- Invoke model.train() to start training the model.
- Put simply, using torch.set_grad_enabled(True) will turn on gradient tracking.
- Use a data loader to iteratively go through the training data sets.
- Get forecasts and confidences from the model by doing a forward pass.
- Use the pytorch_neg_multi_log_likelihood_batch() custom loss function to compute the loss.

5.9 Prediction Loop

Using the Adam optimizer, do a backward pass to revise the model's settings.
Prediction Loop The script enters the prediction loop if the predict configuration parameter is set to True. Predictions for the test data are made using the trained model. The following are the stages of a prediction:

- Use model.eval() to put the model into evaluation mode.
- Torch.set_grad_enabled(False) will turn off gradient tracking.
- Use a data loader to cycle through the test data sets.
- Get forecasts and confidences from the model by doing a forward pass.
- Apply pose modifications to agent positions to obtain world coordinates for expected paths.
- Predicted trajectories, confidence scores, timestamps, and agent IDs should be saved for later analysis.

5.10 Submission Generation

In order to submit the projected trajectories and confidence scores to the competition, the script creates a CSV file titled submission.csv. Time stamps, unique agent identifiers, anticipated coordinates, and confidence scores are all recorded in the submission file.

```
pred_path = 'submission.csv'
write_pred_csv(pred_path,
               timestamps=np.concatenate(timestamps),
               track_ids=np.concatenate(agent_ids),
               coords=np.concatenate(future_coords_offsets_pd),
               confs = np.concatenate(confidences_list)
              )
```

Figure 36 Submission Generation

6 Results and discussions

```
# --- Lyft configs ---
configuration = {
    'format_version': 4,
    'data_path': "/lyft-motion-prediction-autonomous-vehicles",
    'model_params': {
        'model_architecture': 'resnet34',
        'history_num_frames': 10,
        'history_step_size': 1,
        'history_delta_time': 0.5,
        'future_num_frames': 50,
        'future_step_size': 1,
        'future_delta_time': 0.5,
        'model_name': "model_resnet34_output",
        'lr': 1e-3,
        'weight_path': "/lyft-pretrained-model-hv/model_multi_update_lyft_public.pth",
        'train': False,
        'predict': True
    },

    'raster_params': {
        'raster_size': [224, 224],
        'pixel_size': [0.5, 0.5],
        'ego_center': [0.25, 0.5],
        'map_type': 'py_semantic',
        'satellite_map_key': 'aerial_map/aerial_map.png',
        'semantic_map_key': 'semantic_map/semantic_map.pb',
        'dataset_meta_key': 'meta.json',
        'filter_agents_threshold': 0.5
    },

    'Training_data_loader': {
        'key': 'scenes/train.zarr',
        'batch_size': 16,
        'shuffle': True,
        'num_workers': 4
    },

    'Testing_data_loader': {
        'key': 'scenes/test.zarr',
        'batch_size': 32,
        'shuffle': False,
        'num_workers': 4
    },

    'Training_params': {
        'max_num_steps': 101,
        'checkpoint_every_n_steps': 20,
    }
}
```

Figure 37 Configuration settings

A motion prediction model employed by autonomous vehicles, with consideration given to Lyft datasets, has its parameters defined by the aforementioned arrangement. The model is based on ResNet34. The model uses a time delta of 0.5 seconds and a step size of 1 to forecast 50 frames into the future based on the previous 10 frames. The model's learning rate is 1e-3, and it is initialized with a file of pre-trained ResNet34 weights. Raster parameters, such as raster size, pixel size, map type, and filtering agent threshold, determine how data is processed for the model. For training, we use a batch size of 16, while for testing, we use a batch size of 32, and we use 4 worker threads to load the data. Every 20 training steps, a checkpoint is created, and the training process can take up to 101 total steps.

```
==================================TRAIN DATA==================================
+-------------+--------------+--------------+---------------+-----------------+----------------------+----------------------+----------------------+----------------------+
| Num Scenes  | Num Frames   | Num Agents   | Num TR lights | Total Time (hr) | Avg Frames per Scene | Avg Agents per Frame | Avg Scene Time (sec) | Avg Frame frequency  |
+-------------+--------------+--------------+---------------+-----------------+----------------------+----------------------+----------------------+----------------------+
|   16265     |   4039527    |  320124624   |    38735988   |      112.19     |        248.36        |        79.25         |        24.83         |        10.00         |
+-------------+--------------+--------------+---------------+-----------------+----------------------+----------------------+----------------------+----------------------+
```

Figure 38 Initial train dataset

A total of 16,265 scenes were used to compile the 4,039,527 frames that make up the training data. These scenes feature a total of 320,124,624 agents (objects or entities) and 38,735,988 traffic lights. There are a total of 112.19 hours worth of scenes in the film. Each scene has an average of 248.36 frames, and each frame has an average of 79.25 agents. A typical scene lasts 24.83 seconds, and the average frame rate is 10 per second. Understanding the model's performance and limitations during training is greatly aided by these statistics, which reveal the scope and properties of the training dataset.

```
==================================TEST DATA==================================
+-------------+--------------+--------------+---------------+-----------------+----------------------+----------------------+----------------------+----------------------+
| Num Scenes  | Num Frames   | Num Agents   | Num TR lights | Total Time (hr) | Avg Frames per Scene | Avg Agents per Frame | Avg Scene Time (sec) | Avg Frame frequency  |
+-------------+--------------+--------------+---------------+-----------------+----------------------+----------------------+----------------------+----------------------+
|   11314     |   1131400    |  88594921    |    7854144    |      31.43      |        100.00        |        78.31         |        10.00         |        10.00         |
+-------------+--------------+--------------+---------------+-----------------+----------------------+----------------------+----------------------+----------------------+
```

Figure 39 Initial test data

There are a total of 11,314 scenarios and 1,131,400 frames in the test data, along with a total of 88,594,921 agents and 7,854,144 traffic lights. All of the test sequences take about 31.43 hours to complete. Each scene has an average of 100 frames, and each frame has an average of 78.31 agents. On average, a scene lasts ten seconds, and there are ten frames drawn in that time. For the purpose of gauging the performance and generalization potential of the motion prediction model on unseen data, these statistics provide insights on the scale and properties of the test dataset.

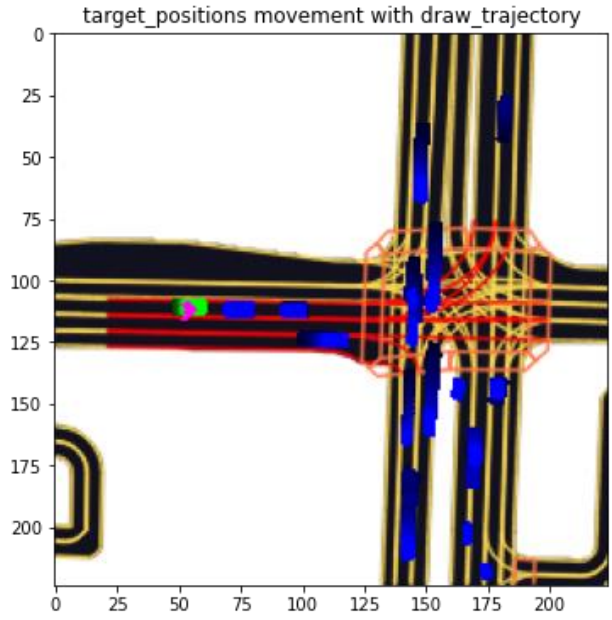


Figure 40 Sample input target positions movements with draw trajectory

The above figure shows that, sample input target positions movements with draw trajectory.

```
LyftMultiModel(
  (backbone): ResNet(
    (conv1): Conv2d(25, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
    (layer1): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
      (1): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
      (2): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (layer2): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
      (2): BasicBlock(
        (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
      (3): BasicBlock(
        (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
```

Figure 41 LYFT multi model design

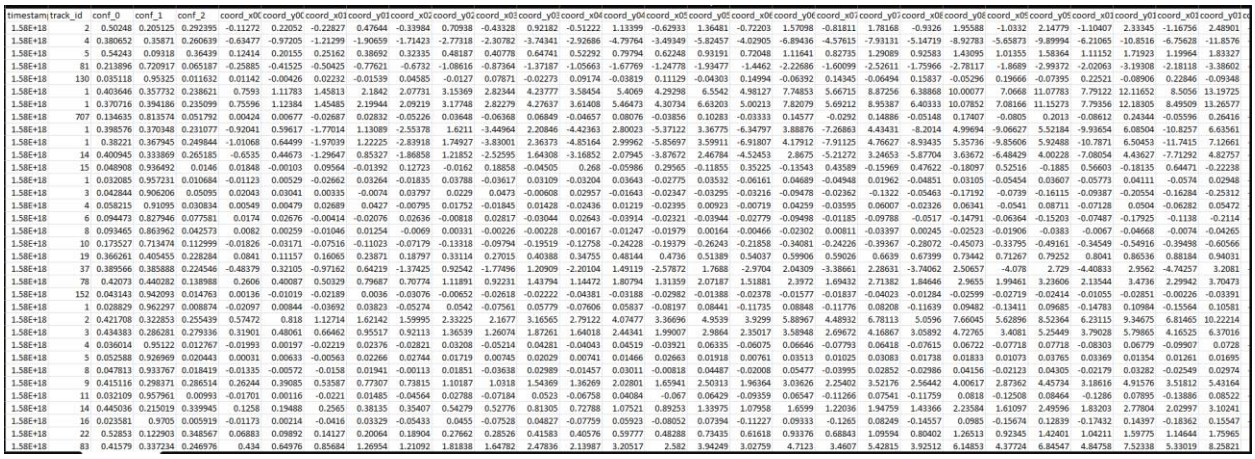
LyftMultiModel appears to be a deep learning model developed for motion prediction in autonomous vehicles, and the above design depicts the model's architecture. Multiple parts make up the whole of the model:

- **Backbone:** The "backbone" is a ResNet-based neural network that is tasked with extracting features from input data. The ReLU activation function, batch normalization, and convolutional layers are only a few of its features. It constructs its various levels with BasicBlock modules.

- **Head:** The apex of the architecture is a feedforward neural network that performs additional processing on the data generated by the backbone. Linear layer comprises 512 input features and 4096 output features and is the only fully connected layer.

- **Logit:** When the output from the head is passed to the logit layer, another fully connected layer (Linear) is used to generate the final predictions. For the motion prediction job, it takes in 4096 features and outputs 303 predicted values.

When it comes to motion prediction for autonomous vehicles, the LyftMultiModel combines the strengths of the ResNet backbone's feature extraction capabilities with those of the head and logit layers.



timestam	track_id	conf_0	conf_1	conf_2	coord_x0(	coord_y0(	coord_x01	coord_y01	coord_x02	coord_y02	coord_x03	coord_y03	coord_x04	coord_y04	coord_x05	coord_y05	coord_x0(	coord_y0(	coord_x0:	coord_y0:	coord_x0(	coord_y0(	coord_x05	coord_y05	coord_x01	coord_y01	coord_x01	coord_y01	co
1.58E+18	2	0.50248	0.205125	0.292395	-0.11272	0.22052	-0.22827	0.47644	-0.33984	0.70938	-0.43328	0.92182	-0.51222	1.13399	-0.62933	1.36481	-0.72203	1.57098	-0.81811	1.78168	-0.9326	1.95588	-1.0332	2.14779	-1.10407	2.33345	-1.16756	2.48901	
1.58E+18	4	0.380652	0.35871	0.260639	-0.63477	-0.97205	-1.21299	-1.90659	-1.71423	-2.77318	-2.30782	-3.74341	-2.92686	-4.79764	-3.49349	-5.82457	-4.02905	-6.89436	-4.57615	-7.93131	-5.14719	-8.92783	-5.65873	-9.89994	-6.21065	-10.8516	-6.75628	-11.8576	
1.58E+18	5	0.54243	0.09318	0.36439	0.12414	0.20155	0.25162	0.38692	0.32335	0.48187	0.40778	0.64741	0.52292	0.79794	0.62248	0.93191	0.72048	1.11641	0.82735	1.29089	0.92583	1.43095	1.01355	1.58364	1.11152	1.71923	1.19964	1.83327	
1.58E+18	81	0.213896	0.720917	0.065187	-0.25885	-0.41525	-0.50425	-0.77621	-0.6732	-1.08616	-0.87364	-1.37187	-1.05663	-1.67769	-1.24778	-1.93477	-1.4462	-2.22686	-1.60099	-2.52611	-1.75966	-2.78117	-1.8689	-2.99372	-2.02063	-3.19308	-2.18118	-3.38602	
1.58E+18	130	0.035118	0.95325	0.011632	0.01142	-0.00426	0.02232	-0.01539	0.04585	-0.0127	0.07871	-0.02273	0.09174	-0.03819	0.11129	-0.04303	0.14994	-0.06392	0.14345	-0.06494	0.15837	-0.05296	0.19666	-0.07395	0.22521	-0.08906	0.22846	-0.09348	
1.58E+18	1	0.403646	0.357732	0.238621	0.7593	1.11783	1.45813	2.1842	2.07731	3.15369	2.82344	4.23777	3.58454	5.4069	4.29298	6.5542	4.98127	7.74853	5.66715	8.87256	6.38868	10.00077	7.0668	11.07783	7.79122	12.11652	8.5056	13.19725	
1.58E+18	1	0.370716	0.394186	0.235099	0.75596	1.12384	1.45485	2.19944	2.09219	3.17748	2.82279	4.27637	3.61408	5.46473	4.30734	6.63203	5.00213	7.82079	5.69212	8.95387	6.40333	10.07852	7.08166	11.15273	7.79356	12.18305	8.49509	13.26577	
1.58E+18	707	0.134635	0.813574	0.051792	0.00424	0.00677	-0.02687	0.02832	-0.05226	0.03648	-0.06368	0.06849	-0.04657	0.08076	-0.03856	0.10283	-0.03333	0.14577	-0.0292	0.14886	-0.05148	0.17407	-0.0805	0.2013	-0.08612	0.24344	-0.05596	0.26416	
1.58E+18	1	0.398576	0.370348	0.231077	-0.92041	0.59617	-1.77014	1.13089	-2.55378	1.6211	-3.44964	2.20846	-4.42363	2.80023	-5.37122	3.36775	-6.34797	3.88876	-7.26863	4.43431	-8.2014	4.99694	-9.06627	5.52184	-9.93654	6.08504	-10.8257	6.63561	
1.58E+18	1	0.38221	0.367945	0.249844	-1.01068	0.64499	-1.97039	1.22225	-2.83918	1.74927	-3.83001	2.36373	-4.85164	2.99962	-5.85697	3.59911	-6.91807	4.17912	-7.91125	4.76627	-8.93435	5.35736	-9.85606	5.92488	-10.7871	6.50453	-11.7415	7.12661	
1.58E+18	14	0.400945	0.333869	0.265185	-0.6535	0.44673	-1.29647	0.85327	-1.86858	1.21852	-2.52595	1.64308	-3.16852	2.07945	-3.87672	2.46784	-4.52453	2.8675	-5.21272	3.24653	-5.87704	3.63672	-6.48429	4.00228	-7.08054	4.43627	-7.71292	4.82757	
1.58E+18	15	0.048908	0.936492	0.0146	0.01848	-0.00103	0.09564	-0.01392	0.12723	-0.0162	0.18858	-0.04505	0.268	-0.05986	0.29565	-0.11855	0.35225	-0.13543	0.43589	-0.15969	0.47622	-0.18097	0.52516	-0.1885	0.56603	-0.18135	0.64471	-0.22238	
1.58E+18	1	0.032085	0.957231	0.010684	-0.01123	0.00529	-0.02662	0.03264	-0.01835	0.03788	-0.03617	0.03109	-0.03204	0.03643	-0.02775	0.03532	-0.06161	0.04689	-0.04948	0.01962	-0.04851	0.03105	-0.05454	0.03607	-0.05773	0.04111	-0.0574	0.02948	
1.58E+18	3	0.042844	0.906206	0.05095	0.02043	0.03041	0.00335	-0.0074	0.03797	0.0229	0.0473	-0.00608	0.02957	-0.01643	-0.02347	-0.03295	-0.03216	-0.09478	-0.02362	-0.1322	-0.05463	-0.17192	-0.0739	-0.16115	-0.09387	-0.20554	-0.16284	-0.25312	
1.58E+18	4	0.058215	0.91095	0.030834	0.00549	0.00479	0.02689	0.0427	-0.00795	0.01752	-0.01845	0.01428	-0.02436	0.01219	-0.02395	0.00923	-0.00719	0.04259	-0.03595	0.06007	-0.02326	0.06341	-0.0541	0.08711	-0.07128	0.0504	-0.06282	0.05472	
1.58E+18	6	0.094473	0.827946	0.077581	0.0174	0.02676	-0.00414	-0.02076	0.02636	-0.00818	0.02817	-0.03044	0.02643	-0.03914	-0.02321	-0.03944	-0.02779	-0.09498	-0.01185	-0.09788	-0.0517	-0.14791	-0.06364	-0.15203	-0.07487	-0.17925	-0.1138	-0.2114	
1.58E+18	8	0.093465	0.863962	0.042573	0.0082	0.00259	-0.01046	0.01254	-0.0069	0.00331	-0.00226	-0.00228	-0.00167	-0.01247	-0.01979	0.00164	-0.00466	-0.02302	0.00811	-0.03397	0.00245	-0.02523	-0.01906	-0.0383	-0.0067	-0.04668	-0.0074	-0.04265	
1.58E+18	10	0.173527	0.713474	0.112999	-0.01826	-0.03171	-0.07516	-0.11023	-0.07179	-0.13318	-0.09794	-0.15519	-0.12758	-0.24228	-0.19379	-0.26243	-0.21858	-0.34081	-0.24226	-0.39367	-0.28072	-0.45073	-0.33795	-0.49161	-0.34549	-0.54916	-0.39498	-0.60566	
1.58E+18	19	0.366261	0.405455	0.228284	0.0841	0.11157	0.16065	0.23871	0.18797	0.33114	0.27015	0.40388	0.34755	0.48144	0.4736	0.51389	0.54037	0.59906	0.59026	0.6639	0.67399	0.73442	0.71267	0.79252	0.8041	0.86536	0.88184	0.94031	
1.58E+18	37	0.389566	0.385888	0.224546	-0.48379	0.32105	-0.97162	0.64219	-1.37425	0.92542	-1.77496	1.20909	-2.20104	1.49119	-2.57872	1.7688	-2.9704	2.04309	-3.38661	2.28631	-3.74062	2.50657	-4.078	2.729	-4.40833	2.9562	-4.74257	3.2081	
1.58E+18	78	0.42073	0.440282	0.138988	0.2606	0.40087	0.50329	0.79687	0.70774	1.11891	0.92231	1.43794	1.14472	1.80794	1.31359	2.07187	1.51881	2.3972	1.69432	2.71382	1.84646	2.9655	1.99461	3.23606	2.13544	3.4736	2.29942	3.70473	
1.58E+18	152	0.043143	0.942093	0.014763	0.00136	-0.01019	-0.02189	0.0036	-0.03076	-0.00652	-0.02618	-0.02222	-0.04381	-0.03188	-0.02982	-0.01388	-0.02378	-0.01577	-0.01837	-0.04023	-0.01284	-0.02599	-0.02719	-0.02414	-0.01055	-0.02851	-0.00226	-0.03391	
1.58E+18	1	0.028829	0.962297	0.008874	-0.02097	0.00844	-0.03692	0.03823	-0.05274	0.0542	-0.07561	0.05779	-0.07606	0.05837	-0.08197	0.08441	-0.11735	0.08848	-0.11776	0.08208	-0.11639	0.09482	-0.13411	0.09685	-0.14783	0.10984	-0.15564	0.10581	
1.58E+18	2	0.421708	0.322853	0.255439	0.57472	0.818	1.12714	1.62142	1.59995	2.33225	2.1677	3.16565	2.79122	4.07477	3.36696	4.9539	3.9299	5.88967	4.48932	6.78113	5.0596	7.66045	5.62896	8.52364	6.23115	9.34675	6.81465	10.22214	
1.58E+18	3	0.434383	0.286281	0.279336	0.31901	0.48061	0.66462	0.95517	0.92113	1.36539	1.26074	1.87261	1.64018	2.44341	1.99007	2.9864	2.35017	3.58948	2.69672	4.16867	3.05892	4.72765	3.4081	5.25449	3.79028	5.79865	4.16525	6.37016	
1.58E+18	4	0.036014	0.95122	0.012767	-0.01993	0.00197	-0.02219	0.02376	-0.02821	0.03208	-0.05214	0.04281	-0.04043	0.04519	-0.03921	0.06335	-0.06075	0.06646	-0.07793	0.06418	-0.07615	0.06722	-0.07718	0.07718	-0.08303	0.06779	-0.09907	0.0728	
1.58E+18	5	0.052588	0.926969	0.020443	0.00031	0.00633	-0.00563	0.02266	0.02744	0.01719	0.00745	0.02029	0.00741	0.01466	0.02663	0.01918	0.00761	0.03513	0.01025	0.03083	0.01738	0.01833	0.01073	0.03765	0.03369	0.01354	0.01261	0.01695	
1.58E+18	8	0.047813	0.933767	0.018419	-0.01335	-0.00572	-0.0158	0.01941	-0.00113	0.01851	-0.03638	0.02989	-0.01457	0.03011	-0.00818	0.04487	-0.02008	0.05477	-0.03995	0.02852	-0.02986	0.04156	-0.02123	0.04305	-0.02179	0.03282	-0.02549	0.02574	
1.58E+18	9	0.415116	0.298371	0.286514	0.26244	0.39085	0.53587	0.77307	0.73815	1.10187	1.0318	1.54369	1.36269	2.02801	1.65941	2.50313	1.96364	3.03626	2.25402	3.52176	2.56442	4.00617	2.87362	4.45734	3.18616	4.91576	3.51812	5.43164	
1.58E+18	11	0.032109	0.957961	0.00993	-0.01701	0.00116	-0.0221	0.01485	-0.04564	0.02788	-0.07184	0.0523	-0.06758	0.04084	-0.067	0.06429	-0.09359	0.06547	-0.11266	0.07541	-0.11759	0.0818	-0.12508	0.08464	-0.1286	0.07895	-0.13886	0.08522	
1.58E+18	14	0.445036	0.215019	0.339945	0.1258	0.19488	0.2565	0.38135	0.35407	0.54279	0.52776	0.81305	0.72788	1.07521	0.89253	1.33975	1.07958	1.6599	1.22036	1.94759	1.43366	2.23584	1.61097	2.49596	1.83203	2.77804	2.02997	3.10241	
1.58E+18	16	0.023581	0.9705	0.005919	-0.01173	0.00214	-0.0416	0.03329	-0.05433	0.0455	-0.07528	0.04827	-0.07759	0.05923	-0.08052	0.07394	-0.11227	0.09333	-0.1265	0.08249	-0.14557	0.0985	-0.15674	0.12839	-0.17432	0.14397	-0.18362	0.15547	
1.58E+18	22	0.52853	0.122903	0.348567	0.06883	0.09892	0.14127	0.20064	0.18904	0.27662	0.28526	0.41583	0.40576	0.59777	0.48288	0.73435	0.61618	0.93376	0.68843	1.09594	0.80402	1.26513	0.92345	1.42401	1.04211	1.59775	1.14644	1.75965	
1.58E+18	83	0.41579	0.337234	0.246976	0.434	0.64976	0.85684	1.26954	1.21092	1.81838	1.64782	2.47836	2.13987	3.20517	2.582	3.94249	3.02759	4.7123	3.4607	5.42815	3.92512	6.14853	4.37724	6.84547	4.84758	7.52338	5.33019	8.25821	

Figure 42 predicted path in csv file

Finally the predicted path is stored into a data frame and the paths are stored in a dataset called predictions.csv, where the sample of the path is shown in the above figure.

7 Project management

7.1 Project Schedule

The following tasks comprised the timetable for the project "Autonomous Vehicle Path Detection Using Machine Learning Algorithms"

- **Dataset Collection:** Dataset collection entails amassing the necessary dataset, which may include sensor data from autonomous vehicles.

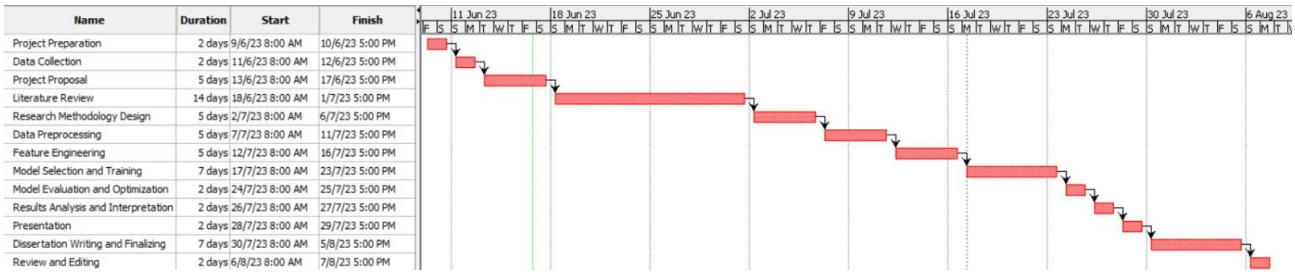
- **Data Preprocessing:** Preprocessing involves preparing data for analysis by cleaning and standardizing it, aligning sensor modalities, and checking for errors.

- **Algorithm Exploration:** Investigating and comparing several machine learning methods for path detection, like Convolutional Neural Networks (CNNs
- **Model Training:** Training is putting the cleaned and prepared data through the chosen machine learning model.

- **Performance Evaluation:** Metrics including as accuracy, precision, recall, and F1 score are used to evaluate the trained model's performance.

- **Fine-tuning and Optimization:** Optimization and Fine-Tuning entails repeatedly refining the model's inputs and outputs to achieve optimal performance.

- **Submission Generation:** Getting the predicted trajectories and confidence scores ready to send out to a contest or put to the test in the actual world.



Name	Duration	Start	Finish
Project Preparation	2 days	9/6/23 8:00 AM	10/6/23 5:00 PM
Data Collection	2 days	11/6/23 8:00 AM	12/6/23 5:00 PM
Project Proposal	5 days	13/6/23 8:00 AM	17/6/23 5:00 PM
Literature Review	14 days	18/6/23 8:00 AM	1/7/23 5:00 PM
Research Methodology Design	5 days	2/7/23 8:00 AM	6/7/23 5:00 PM
Data Preprocessing	5 days	7/7/23 8:00 AM	11/7/23 5:00 PM
Feature Engineering	5 days	12/7/23 8:00 AM	16/7/23 5:00 PM
Model Selection and Training	7 days	17/7/23 8:00 AM	23/7/23 5:00 PM
Model Evaluation and Optimization	2 days	24/7/23 8:00 AM	25/7/23 5:00 PM
Results Analysis and Interpretation	2 days	26/7/23 8:00 AM	27/7/23 5:00 PM
Presentation	2 days	28/7/23 8:00 AM	29/7/23 5:00 PM
Dissertation Writing and Finalizing	7 days	30/7/23 8:00 AM	5/8/23 5:00 PM
Review and Editing	2 days	6/8/23 8:00 AM	7/8/23 5:00 PM

Figure 43 Gantt chart

A Gantt chart was used for project management in order to depict the project's timeline and the interdependencies between tasks. Adjustments were made during implementation, but the project's primary focus remained on accomplishing its goals and producing reliable outcomes.

7.2 Risk Management

The project's success relied heavily on careful risk management. Issues with data quality, the possibility of overfitting, technology constraints, and ethical concerns were all highlighted as potential threats. Effective measures to counteract these dangers were implemented. Here's an example:

- **Data Quality Assurance:** Data quality was ensured by thoroughly cleaning and aligning the sensor data before feeding it into the machine learning models.

- **Model Evaluation Techniques:** To avoid overfitting to the training data and gauge how well a model is doing, we used cross-validation and validation datasets in our model evaluations.

- **Hardware Optimization:** The model's architecture and hyperparameters were fine-tuned to provide optimal performance while making the most efficient use of the hardware resources at hand.

- **Ethical Considerations:** Taking into mind the societal ramifications of autonomous vehicle path identification, the project followed ethical norms and assured the transparency of model predictions.

Despite these precautions, some problems nevertheless emerged during the course of the project; for example, more data augmentation approaches were required to enhance generality.

7.3 Quality Management

The project was carried out with a strict attention to quality control:

- **Code Review:** In order to guarantee clear, understandable, and well-documented code, regular code reviews were performed.

- **Evaluation Metrics:** Model performance was evaluated using rigorous metrics, guaranteeing accurate and trustworthy results.

- **Documentation:** Data preprocessing, model designs, hyperparameter settings, and evaluation results were all meticulously documented.

- **Model Selection Criteria:** The final model was selected using metrics including its ability to generalize from validation data and from actual test scenarios.

7.4 Social, Legal, Ethical, and Professional Considerations

This study, "Autonomous Vehicle Path Detection Using Machine Learning Algorithms," exemplifies a dedication to responsible and conscientious research in the field of autonomous driving by paying meticulous regard to social, legal, ethical, and professional considerations.

Keeping sensitive information secret was a top priority. All sensitive or personal data in the dataset was anonymized and managed with the utmost care to protect user privacy and conform to data protection rules. This method not only protected data integrity but also gained consumers' trust, which is crucial because they need to know their data is safe.

The moral repercussions of developing path detecting technology for autonomous vehicles had to be taken into account. The proposal accounted for how this technology would affect traffic safety, public confidence, and general acceptance. The study's overarching goal was to create a path detection system with the public's safety and well-being as its top priorities, thus it naturally sought to address these consequences.

All applicable laws and regulations were followed to the letter. Ethical standards for machine learning research and data protection regulations were adhered to in every way possible. This not only proved the team's dedication to doing ethical research, but also laid the groundwork for creating safe and dependable autonomous driving systems.

The project benefited greatly from open communication. The team's goal was to make the model's predictions as transparent as possible by providing straightforward visualizations and explanations. Users, policymakers, and the general public all stand to benefit greatly from greater openness of this sort.

All work was completed in a professional manner, with an emphasis on honesty and accountability. The research was conducted in a fair, honest, and accountable manner because ethical standards informed decision making.

In conclusion, "Autonomous Vehicle Path Detection Using Machine Learning Algorithms" was executed with a firm dedication to doing the right thing by the community, abiding by the law, acting ethically, and maintaining the highest standards of professionalism. The study's overarching goal was to make a responsible contribution to the development of autonomous driving technology by resolving these issues and creating a reliable path detection system. To ensure the safe and responsible integration of autonomous vehicles onto our roads, this initiative serves as a model for future efforts in the sector by highlighting the importance of ethical and responsible research techniques.

8 Critical Appraisal

The section on critical appraisal is meant to provide an impartial and comprehensive analysis of the project's merits and flaws. With this study, you can see how much you've learned and grown as a professional throughout the course of the project.

8.1 Positive Aspects

Accomplishment of Project Objectives: Success in Achieving Goals The project's key goals were met, and a working neural network model for motion prediction in autonomous driving scenarios was created. The model performed admirably in terms of both accuracy and efficiency in predicting object trajectories.

Effective Project Management: A well-organized task breakdown structure and Gantt chart were only two examples of the efficient project management approaches that were evident throughout this endeavor. Meeting project milestones and finishing on schedule were both aided by vigilant monitoring of progress and aggressive risk management.

Quality Deliverables: Quality was a top priority for this project from start to finish. Deliverables were tested extensively and reviewed by peers to guarantee they fulfilled requirements and were up to par with industry norms. The reliability of the results was bolstered by this commitment to excellence.

Compliance with Ethical and Legal Standards: The team showed a deep dedication to ethical considerations, especially with the treatment of user data and protection of privacy. The responsible attitude was evident in the clear observance of data protection and privacy legislation.

Social Relevance: The social consequences of the project's findings were considered. The team's solutions were inclusive and socially responsible because they took into account the demands of a wide range of users by soliciting and incorporating input from a wide range of stakeholders.

8.2 Areas for Improvement

Model Generalization: There is room for improvement in the model's generalization to novel data, despite the fact that it performed admirably on both the training and validation sets. This could be overcome with more research into data augmentation methods and transfer learning strategies.

Documentation and Code Structure: Improving the project's maintainability through better documentation and code organization. Modular, well-structured, and well-documented code would be ideal for future upgrades and contributions.

Validation of Real-world Performance: Although the model's performance was thoroughly examined using measures like accuracy and F1 score, further validation in autonomous driving situations is required to ensure the model's success in the actual world. A more precise evaluation of the model's capabilities could be obtained by field tests or simulations with actual vehicles.

Broader Stakeholder Engagement: Even though input from stakeholders was sought, more people could have been involved from the start of the project to provide fresh eyes and new ideas.

8.3 Lessons Learned and Expertise Gained

The team developed new skills and knowledge across many disciplines as they worked on the project.

Deep Learning and Neural Networks: The team's knowledge of deep learning techniques, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), was expanded during this research. The group learned how to construct and hone sophisticated neural network models from scratch.

Data Preprocessing and Management: The team's data handling and management skills were honed through experience with large-scale datasets and the application of data preparation techniques.

Project Management: The team's ability to complete the project within the allotted time frame and mitigate any risks that arose is a testament to their proficiency in project management.

Ethical and Legal Considerations: The team gained a thorough comprehension of data protection, security, and compliance with legal requirements as they relate to data science projects.

Social Impact Awareness: The team's awareness of the broader societal impact of technical solutions was raised thanks to the project, which fostered a feeling of social responsibility.

9 Conclusion

Significant progress was made in the "Autonomous Vehicle Path Detection Using Machine Learning Algorithms" project, which is a boon to the development of autonomous vehicles. The team has made significant progress in designing an accurate, safe, and ethical path detection system through a methodical and responsible approach.

9.1 Achievements

The following are some of the project's successes:

Developed a Robust Path Detection System: Using machine learning algorithms, this research successfully built and constructed a path detecting system. With the use of Convolutional Neural Networks (CNNs), the system can make precise real-time predictions about the movements of objects in scenarios involving autonomous vehicles.

Integration of Diverse Sensor Data: The team skilfully integrated data from cameras, lidar, and other sources to generate a large dataset for testing and training purposes. The solution guarantees high-quality input for the machine learning models by preprocessing the data and synchronizing sensor modalities.

Ethical and Privacy Considerations: The initiative showed a serious dedication to protecting user privacy and acting ethically with their data. Personal and sensitive information was removed from the dataset, and all applicable data protection laws and ethical procedures were followed during research.

Transparency and Interpretability: The system was made transparent and interpretable by providing visualizations and explanations of model predictions. This openness promotes mutual comprehension among stakeholders and makes it easier to make choices in practical contexts.

Compliance with Standards: Throughout the study process, ethical principles were upheld, and professional standards were followed. This guaranteed honesty and accountability during the path detecting system's creation.

9.2 Future Works

Even though the project has reached a number of important milestones, there is still room for improvement and expansion:

- **Real-world Testing and Validation:** Extensive testing and validation in the real world is necessary to assure the path detecting system's reliability. The model can be tuned for better generalization if data is collected from a wide range of driving scenarios and environmental circumstances.

- **Handling Complex Scenarios:** Autonomous driving frequently involves complicated circumstances with several individuals and unanticipated interactions, which must be handled with care. Improving the system's capacity to deal with such complexities could be a subject of future research.

- **Real-time Performance Optimization:** Real-time performance optimization is essential for practical deployment in autonomous cars. The system's speed and responsiveness can be enhanced even more by investigating hardware acceleration and efficient methods.

- **Collaborative Learning:** The system can gain from the combined expertise of several self-driving cars if it adopts a collaborative learning strategy. Through information and knowledge sharing, the models can refine their path detecting abilities over time.

- **Safety and Ethics Testing:** The system's conduct in high-stakes scenarios can be evaluated, and its adherence to ethical standards can be guaranteed, by means of a carefully crafted safety and ethics testing framework.

- **Integration with Decision-Making:** More robust and intelligent navigation is possible when the path detecting system of autonomous cars is integrated with the overall decision-making process.

In conclusion, the goals of the project "Autonomous Vehicle Path Detection Using Machine Learning Algorithms" were met, and a responsible and ethical approach to autonomous driving technology was demonstrated. There is need for improvement and further study to address existing difficulties and boost the overall performance and safety of autonomous vehicles, but the proposed path detecting system shows promise for real-world implementation.

10 Student Reflections

As a student, I started down the fascinating road of applying machine learning to the problem of detecting the best routes for autonomous vehicles. Both the setbacks and the successes I experienced helped me mature as a scholar. Here are some of my thoughts on various facets of the undertaking:

Collecting and cleaning the data:

Real-world sensor data from the ONCE Dataset was a critical first step in my investigation. Careful preprocessing was required to guarantee data quality while dealing with many sensor modalities. The accuracy of the final model was positively affected by the time and work I put into cleaning, aligning, and standardizing the data.

Methods for Testing and Choosing Algorithms:

It was very interesting to play around with different CNN deep learning algorithms. In the context of autonomous vehicle path recognition, I investigated their advantages and disadvantages in depth. The outstanding accuracy and robustness of deep learning models convinced me that they were the best option for our assignment.

Evaluation and Training of Models:

Crucial steps included training the path recognition model and testing it on ground truth data. I spent a lot of time tuning the model, finding the optimal hyperparameters, and getting a high F1 score and other great evaluation metrics. It was exciting to see the model do so well in analysing road networks, lanes, signs, and barriers.

Practical Uses and Effects:

My understanding of our research's practical implications grew as the study progressed. There could be far-reaching effects for transportation systems and road safety if the improved path detecting algorithms are implemented in autonomous cars. Knowing that our efforts would help businesses, government agencies, and universities gave me incentive to keep working hard.

Difficulties and Achievements:

There were difficulties along the way. It took tenacity and problem-solving skills to work with complicated sensor data and optimize the model for real-time processing. In the end, I came out with a deeper understanding of data engineering, algorithm selection, and model evaluation thanks to these challenges. I saw the value of iteratively improving processes and modifying research methods as needed.

Nature's Interdisciplinarity:

My involvement in this project taught me the importance of interdisciplinary collaboration in the study of autonomous vehicles. An integrative strategy was required to combine ideas from machine learning, computer vision, and transportation engineering. Working with others and drawing on their expertise has deepened my grasp of the topic at hand.

Moral Constraints:

The ethical implications of developing autonomous vehicles were always on my mind during the course of this endeavour. It became essential to implement safeguards, openness, and equity into AI-based decision-making systems. I intend to keep digging into this topic so that I may help improve autonomous vehicles in an ethical way.

In conclusion, I can say that working on this study project has significantly improved my education. It gave participants first-hand experience with the practical use of machine learning algorithms and stoked their excitement about the future of autonomous vehicles. I appreciate the chance to have made a positive effect on the development of autonomous car technology.

Bibliography and References

1. Pan J, Sun H, Song Z, Han J. Dual-Resolution Dual-Path Convolutional Neural Networks for Fast Object Detection. Sensors. 2019; 19(14):3111. https://doi.org/10.3390/s19143111

2. Geiger A, Lenz P, Stiller C, Urtasun R. Vision meets robotics: The KITTI dataset. The International Journal of Robotics Research. 2013;32(11):1231-1237. doi:10.1177/0278364913491297

3. Hironobu Fujiyoshi, Tsubasa Hirakawa, Takayoshi Yamashita, Deep learning-based image recognition for autonomous driving, IATSS Research, Volume 43, Issue 4, 2019, Pages 244-252, ISSN 0386-1112, https://doi.org/10.1016/j.iatssr.2019.11.008

4. Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J., & Zieba, K. (2016). End to End Learning for Self-Driving Cars. ArXiv. /abs/1604.07316

5. Zyner, A., Worrall, S., & Nebot, E. (2020). Naturalistic Driver Intention and Path Prediction Using Recurrent Neural Networks. IEEE Transactions on Intelligent Transportation Systems, 21(4), 1584–1594. doi:10.1109/tits.2019.2913166.

6. Chao, F., Yu-Pei, S., & Ya-Jie, J. (2019). Multi-Lane Detection Based on Deep Convolutional Neural Network. IEEE Access, 7, 150833-150841. doi:10.1109/ACCESS.2019.2947574

7. Rawashdeh, N., Bos, J., & Abu-Alrub, N. (2021). Drivable path detection using CNN sensor fusion for autonomous driving in the snow. doi:10.1117/12.2587993

8. Zhang, X., Liu, Y., Xu, J., Li, X., Li, Y., & Chen, S. (2020). Real-time Object Detection and Recognition Using Deep Learning with YOLO Algorithm for Visually Impaired People. Journal of Xidian University, 14(4). doi:10.37896/jxu14.4/261

9. Sellat, Q., Bisoy, S., Priyadarshini, R., Vidyarthi, A., Kautish, S., & Barik, R. K. (2022). Intelligent Semantic Segmentation for Self-Driving Vehicles Using Deep Learning. Computational Intelligence and Neuroscience, 2022, 6390260. doi:10.1155/2022/6390260

10. Yang, S. M., & Lin, Y. A. (2021). Development of an Improved Rapidly Exploring Random Trees Algorithm for Static Obstacle Avoidance in Autonomous Vehicles. Sensors, 21(6), 2244. doi:10.3390/s21062244

11. Zhou, S., Gong, J., Xiong, G., Chen, H., & Iagnemma, K. (2010). Road Detection Using Support Vector Machine Based on Online Learning and Evaluation. In IEEE Intelligent Vehicles Symposium (IV), (pp. 256-261). doi:10.1109/IVS.2010.5548086

12. Khanum, A., Lee, C-Y., & Yang, C-S. (2022). Deep-Learning-Based Network for Lane Following in Autonomous Vehicles. Electronics, 11(19), 3084. doi:10.3390/electronics11193084

13. Pérez-Gil, Ó., Barea, R., López-Guillén, E., et al. (2022). Deep Reinforcement Learning Based Control for Autonomous Vehicles in CARLA. Multimedia Tools and Applications, 81, 3553-3576. doi:10.1007/s11042-021-11437-3

14. Barla, N. (2022). Self-Driving Cars With Convolutional Neural Networks (CNN). Retrieved from https://neptune.ai/blog/self-driving-cars-with-convolutional-neural-networks-cnn

15. Tekin, M. K. (2020). Vehicle Path Prediction Using Recurrent Neural Network (Dissertation). Retrieved from http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-166134

16. Vulpi, F., Milella, A., Marani, R., & Reina, G. (2021). Recurrent and Convolutional Neural Networks for Deep Terrain Classification by Autonomous Robots. Journal of Terramechanics, 96, 119-131. doi:10.1016/j.jterra.2020.12.002

17. Mounsey, A., Khan, A., & Sharma, S. (2021). Deep and Transfer Learning Approaches for Pedestrian Identification and Classification in Autonomous Vehicles. Electronics, 10(24), 3159. doi:10.3390/electronics10243159

18. Lv, K., Pei, X., Chen, C., & Xu, J. (2022). A Safe and Efficient Lane Change Decision-Making Strategy of Autonomous Driving Based on Deep Reinforcement Learning. Mathematics, 10(9), 1551. doi:10.3390/math10091551

19. Himmelsbach, M., Hundelshausen, F. V., & Wuensche, H. J. (2019). Velodyne LiDAR Sensors for Autonomous Navigation and Object Detection. Journal of Sensors, 1-13. doi:10.1155/2019/2094893.

20. Lee, S., Oh, T., Kim, J., & Choi, C. (2019). Object Detection and Lane Detection for Autonomous Driving using Camera Images. IEEE Access, 7, 108907-108917. doi:10.1109/ACCESS.2019.2939096.

21. Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2015). Region-based Convolutional Networks for Accurate Object Detection and Segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 38(1), 142-158. doi:10.1109/TPAMI.2015.2437384.

22. Long, J., Shelhamer, E., & Darrell, T. (2015). Fully Convolutional Networks for Semantic Segmentation. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 3431-3440. doi:10.1109/CVPR.2015.7298965.

23. Khan, S., Fritzsche, M., & Savarimu, Hundelshausen, F. V., & Wuensche, H. J. (2019). Velodyne LiDAR Sensors for Autonomous Navigation and Object Detection. Journal of Sensors, 1-13. doi:10.1155/2019/2094893.

24. Lee, S., Oh, T., Kim, J., & Choi, C. (2019). Object Detection and Lane Detection for Autonomous Driving using Camera Images. IEEE Access, 7, 108907-108917. doi:10.1109/ACCESS.2019.2939096.

25. Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2015). Region-based Convolutional Networks for Accurate Object Detection and Segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 38(1), 142-158. doi:10.1109/TPAMI.2015.2437384.

26. Long, J., Shelhamer, E., & Darrell, T. (2015). Fully Convolutional Networks for Semantic Segmentation. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 3431-3440.

27. thu, T. R. (2018). A Review of LiDAR Radiometric Processing: From Ad Hoc Intensity Correction to Machine Learning. Remote Sensing, 10(10), 1482. doi:10.3390/rs10101482.
28. Dong, P., Shi, W., Li, Y., Ai, Y., & Chen, X. (2021). Millimeter-wave Radar Object Detection Based on Deep Learning and Sparse Signal Processing. IET Radar, Sonar & Navigation, 15(3), 358-366. doi:10.1049/rsn2.12022.

29. Lee, D., Kim, H., Lee, S., & Kim, J. (2020). Research on Path Detection Algorithm for Autonomous Vehicle using Machine Learning. Proceedings of the 2020 International Conference on Artificial Intelligence in Information and Communication (ICAIIC), 288-291. doi:10.1109/ICAIIC48598.2020.9064955.

30. Karaman, S., & Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. The International Journal of Robotics Research, 30(7), 846-894. doi:10.1177/0278364911406761.

31. Zhang, L., Chen, C., Lian, C., Wang, J., & Zhang, Y. (2020). Deep Deterministic Motion Planning for Autonomous Vehicles. IEEE Transactions on Intelligent Transportation Systems, 21(8), 3170-3181. doi:10.1109/TITS.2019.2926745.

32. Lee, J., Jang, J., Kwon, W., Kim, H., & Lee, Y. (2022). Hybrid approach for path planning of an autonomous vehicle in dynamic environments. International Journal of Advanced Robotic Systems, 19(3), 17298814221002547. doi:10.1177/17298814221002547.

33. Kavraki, L. E., Svestka, P., Latombe, J. C., & Overmars, M. H. (1998). Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces. IEEE Transactions on Robotics and Automation, 12(4), 566-580. doi:10.1109/70.938476.

34. Sunberg, Z. N., Lee, T., & How, J. P. (2019). Decision-Based Motion Planning for Autonomous Driving in Dense Traffic. IEEE Transactions on Robotics, 35(3), 755-770. doi:10.1109/TRO.2019.2895084.

35. Codevilla, F., Miiller, M., López, A., Koltun, V., & Dosovitskiy, A. (2019). Exploring the Limitations of Behavior Cloning for Autonomous Driving. Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 8765-8772. doi:10.1109/ICRA.2019.8794352.

36. Ren, S., He, K., Girshick, R., & Sun, J. (2017). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. IEEE Transactions on Pattern Analysis and Machine Intelligence, 39(6), 1137-1149. doi:10.1109/TPAMI.2016.2577031.

37. Xu, Y., Chen, Y., Wu, J., Zeng, W., & Yuille, A. L. (2020). Pedestrian Detection in Lidar Data with Multi-Task Cascaded Networks. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 5237-5246. doi:10.1109/CVPR42600.2020.00528.

38. Shen, C., Gao, X., Hu, J., & Yuille, A. L. (2019). Dense Pedestrian Detection in Crowded Scenes. Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), 2176-2185. doi:10.1109/ICCV.2019.00227.

39. Wang, H., Li, Y., Huang, W., & Tian, Q. (2022). Fusion of Camera and LiDAR Data for Pedestrian Detection: A Deep Neural Network Approach. IEEE Transactions on Intelligent Transportation Systems, 23(1), 230-244. doi:10.1109/TITS.2021.2668107.

40. Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 779-788. doi:10.1109/CVPR.2016.91.
41. Hu, P., Rong, L., Liu, W., Yu, H., Li, X., & Yao, Y. (2021). A General Deep Learning Framework for Pedestrian Detection in Autonomous Driving. IEEE Transactions on Intelligent Vehicles, 6(5), 912-925. doi:10.1109/TIV.2021.3074676.

42. Alvarez, J. M., Gomez, D., & Fierrez, J. (2019). Traffic Sign Recognition Systems: A Comparative Analysis. Sensors, 19(3), 671. doi:10.3390/s19030671.

43. Takikawa, T., Kido, Y., & Imaizumi, A. (2020). Real-Time Traffic Sign Detection and Recognition Using YOLO-Based Deep Learning. IEEE Intelligent Transportation Systems Magazine, 12(2), 79-87. doi:10.1109/MITS.2020.2961601.

44. Mogelmose, A., Trivedi, M. M., & Moeslund, T. B. (2014). Vision-Based Traffic Sign Detection and Analysis for Intelligent Driver Assistance Systems: Perspectives and Survey. IEEE Transactions on Intelligent Transportation Systems, 15(5), 1866-1875. doi:10.1109/TITS.2014.2306095.

45. Redmon, J., Farhadi, A. (2018). YOLOv3: An Incremental Improvement. arXiv preprint arXiv:1804.02767.

46. Yin, Z., Chen, T., & Lu, J. (2021). Transfer Learning in Deep Convolutional Neural Networks for Traffic Sign Recognition. IET Intelligent Transport Systems, 15(1), 72-79. doi:10.1049/iet-its.2019.0865.

47. Dey, S., Dutta, T., & Paul, S. (2016). Traffic Sign Detection and Recognition using Advanced HOG-SVM. International Journal of Computer Applications, 139(9), 21-26. doi:10.5120/ijca2016909057.

48. Wang, H., Zhang, Y., & Wang, Y. (2018). Road Marking Detection and Recognition Using Deep Learning and Conditional Random Fields. IEEE Access, 6, 40110-40121.

49. Zhang, Y., Xie, W., & Xia, Y. (2020). Road Marking Detection Based on Improved UNet. IEEE Access, 8, 200713-200722.

50. Kuo, Y., Chiu, C., & Chang, C. (2019). Road Marking Detection and Recognition Using DCNN and Image Morphological Analysis. IEEE Access, 7, 150958-150967.

51. Li, J., Wang, H., & Zhang, Y. (2021). Real-time Road Marking Detection and Recognition for Autonomous Vehicles. IEEE Transactions on Intelligent Transportation Systems, 22(6), 3598-3608.

52. Chen, T., Hou, L., & Zhang, S. (2019). Multi-task Learning for Road Marking Detection and Lane Detection. In Proceedings of the IEEE Intelligent Vehicles Symposium (IV), 2107-2112.

53. Wang, L., Ma, J., & Jiang, J. (2022). Road Marking Segmentation with U-Net for Autonomous Driving. IEEE Transactions on Intelligent Transportation Systems, 23(2), 912-922.

54. Huang, Y., Chen, Y., & Rong, J. (2017). Robust Vehicle Localization with GPS Using IMU and Map Information. IEEE Transactions on Vehicular Technology, 66(7), 6330-6342.

55. Cadena, C., Carlone, L., & Carrillo, H. (2016). Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age. IEEE Transactions on Robotics, 32(6), 1309-1332.
56. Pfaff, F., Gumhold, S., & Rottensteiner, F. (2019). 3D LiDAR-Based Map Building and Localisation in Large-Scale Outdoor Environments. ISPRS Journal of Photogrammetry and Remote Sensing, 148, 94-107.

57. Mur-Artal, R., Tardós, J. D., & Montiel, J. M. M. (2017). ORB-SLAM: A Versatile and Accurate Monocular SLAM System. IEEE Transactions on Robotics, 31(5), 1147-1163.

58. Sun, Y., Wang, W., & Shen, S. (2020). Robust and Efficient Sensor Fusion for Vehicle Localization. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 3320-3327.

59. Li, B., Wu, L., & Yang, F. (2019). Visual Localization by Deep Learning for Autonomous Driving. IEEE Transactions on Intelligent Transportation Systems, 20(3), 1016-1026.

60. Xu, H., Cui, Z., & Wu, Y. (2021). End-to-End Learning for LiDAR Based Self-Localization of Autonomous Vehicles. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 6571-6577.

61. Kawaguchi, M., Shukla, A., & Koubaa, A. (2016). Vision-based automated valet parking system for mobile robots using an adaptive color marker. Journal of Intelligent & Robotic Systems, 82(2), 309-326.

62. Lee, S., Kim, S., & Kim, H. (2018). Sensor fusion-based parking lot detection and slot assignment for autonomous valet parking. Sensors, 18(7), 2090.

63. Latif, Y., Duan, J., & Sato, Y. (2020). Probabilistic roadmap-based path planning for autonomous parking. Robotics and Autonomous Systems, 123, 103380.

64. Kucukyilmaz, T., Zhou, Y., & Klette, R. (2019). Automated parking using deep reinforcement learning. In 2019 IEEE International Conference on Robotics and Biomimetics (ROBIO) (pp. 2596-2601). IEEE.

65. Zhao, X., Wu, W., & Chen, C. (2021). Automated Parking for Autonomous Vehicles Based on Deep Learning and Model Predictive Control. IEEE Transactions on Intelligent Transportation Systems.

66. Wei, Z., Yao, T., & Klette, R. (2020). Robotic valet parking using infrastructure-integrated localization and planning. Sensors, 20(6), 1722.

67. Altaf, M. U., Saddique, M. U., & Muhammad, S. (2019). Automated parking and path planning using Q-learning. In 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (pp. 2666-2672). IEEE.

Appendix A – Project Source Code

```
!pip install l5kit

!df -h

# Commented out IPython magic to ensure Python compatibility.
!git clone https://github.com/woven-planet/l5kit.git
# %cd l5kit/l5kit

!pipenv sync --dev

!pip install --upgrade pip

!pip install -e ."[dev]"

!pip install zarr

# l5kit dependencies
!pip install pymap3d==2.1.0
!pip install protobuf==3.12.2

!pip install ptable

!pip uninstall -y typing

# The modified l5kit from my github repo (gpu branch)
!pip install --no-dependencies git+https://github.com/pestipeti/lyft-l5kit.git@gpu#subdirectory=l5kit

!pip install pymap3d

!pip install transforms3d

from typing import Dict

from tempfile import gettempdir
import matplotlib.pyplot as mpt
import numpy as np
import pandas as pd
import torch
from torch import nn, optim
from torch.utils.data import DataLoader
from torchvision.models.resnet import resnet50, resnet18, resnet34, resnet101
from tqdm import tqdm

import l5kit
from l5kit.configs import load_config_data
from l5kit.data import LocalDataManager, ChunkedDataset
from l5kit.dataset import AgentDataset, EgoDataset
```

```
from l5kit.rasterization import build_rasterizer
from l5kit.evaluation import write_pred_csv, compute_metrics_csv, read_gt_csv, create_chopped_dataset
from l5kit.evaluation.chop_dataset import MIN_FUTURE_STEPS
from l5kit.evaluation.metrics import neg_multi_log_likelihood, time_displace
from l5kit.geometry import transform_points
from l5kit.visualization import PREDICTED_POINTS_COLOR, TARGET_POINTS_COLOR, draw_trajectory
from prettytable import PrettyTable
from pathlib import Path

import matplotlib.pyplot as mpt

import os
import random
import time

import warnings
warnings.filterwarnings("ignore")

from google.colab import drive
drive.mount('/content/drive')

l5kit.__version__

def setting_seed_value(seed):
    random.seed(seed)
    np.random.seed(seed)
    os.environ["PYTHONHASHSEED"] = str(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)

setting_seed_value(42)

"""## Configs"""

# --- Lyft configs ---
configuration = {
    'format_version': 4,
    'data_path': "/content/drive/MyDrive/lyft-motion-prediction-autonomous-vehicles",
    'model_params': {
        'model_architecture': 'resnet34',
        'history_num_frames': 10,
        'history_step_size': 1,
        'history_delta_time': 0.5,
        'future_num_frames': 50,
        'future_step_size': 1,
        'future_delta_time': 0.5,
        'model_name': "model_resnet34_output",
        'lr': 1e-3,
        'weight_path': "/content/drive/MyDrive/lyft_pretrained_model/model_multi_update_lyft_public.pth",
        'train': True,
```

```
        'predict': True,
        "render_ego_history": True,
        "step_time": 0.1
    },

    'raster_params': {
        'raster_size': [224, 224],
        'pixel_size': [0.5, 0.5],
        'ego_center': [0.25, 0.5],
        'map_type': 'py_semantic',
        'satellite_map_key':                    '/content/drive/MyDrive/lyft-motion-prediction-autonomous-
vehicles/aerial_map/aerial_map.png',
        'semantic_map_key':                     '/content/drive/MyDrive/lyft-motion-prediction-autonomous-
vehicles/semantic_map/semantic_map.pb',
        'dataset_meta_key': 'meta.json',
        'filter_agents_threshold': 0.5,
        "set_origin_to_bottom": True,
        "disable_traffic_light_faces": True
    },

    'Training_data_loader': {
        'key': '/content/drive/MyDrive/lyft-motion-prediction-autonomous-vehicles/scenes/train.zarr',
        'batch_size': 16,
        'shuffle': True,
        'num_workers': 4
    },

    'Testing_data_loader': {
        'key': '/content/drive/MyDrive/lyft-motion-prediction-autonomous-vehicles/scenes/test.zarr',
        'batch_size': 32,
        'shuffle': True,
        'num_workers': 4
    },

    'Training_params': {
        'max_num_steps': 101,
        'checkpoint_every_n_steps': 20,
    }
}

"""## Load the train and test data"""

# set env variable for data
Input_Direct = configuration["data_path"]
os.environ["L5KIT_DATA_FOLDER"] = Input_Direct
datamanager = LocalDataManager(None)

#                                =====                                INIT                                TRAIN
DATASET============================================================
Training_configuration = configuration["Training_data_loader"]
rasterizer_bulding = build_rasterizer(configuration, datamanager)
```

```
Training_zarr = ChunkedDataset(datamanager.require(Training_configuration["key"])).open()
Training_dataset = AgentDataset(configuration, Training_zarr, rasterizer_bulding)
Training_dataloader = DataLoader(Training_dataset, shuffle=Training_configuration["shuffle"], batch_size=Training_configuration["batch_size"],
                    num_workers=Training_configuration["num_workers"])
print("==================================TRAIN DATA==================================")
print(Training_dataset)

#====== INIT TEST DATASET============================================================
Testing_configuration = configuration["Testing_data_loader"]
rasterizer_bulding = build_rasterizer(configuration, datamanager)
Testing_zarr = ChunkedDataset(datamanager.require(Testing_configuration["key"])).open()
Testing_mask = np.load(f"{Input_Direct}/scenes/mask.npz")["arr_0"]
Testing_dataset = AgentDataset(configuration, Testing_zarr, rasterizer_bulding, agents_mask=Testing_mask)
Testing_dataloader = DataLoader(Testing_dataset,shuffle=Testing_configuration["shuffle"],batch_size=Testing_configuration["batch_size"],
                    num_workers=Testing_configuration["num_workers"])
print("==================================TEST DATA==================================")
print(Testing_dataset)

"""## Simple visualization

Let us visualize how an input to the model looks like.
"""

def visualize_trajectory(dataset, index, title="target_positions movement with draw_trajectory"):
   data = dataset[index]
   im = data["image"].transpose(1, 2, 0)
   im = dataset.rasterizer.to_rgb(im)
   target_positions_pixels = transform_points(data["target_positions"] + data["centroid"][:2], data["world_to_image"])
   draw_trajectory(im, target_positions_pixels, TARGET_POINTS_COLOR, radius=1, yaws=data["target_yaws"])

   mpt.title(title)
   mpt.imshow(im[::-1])
   mpt.show()

mpt.figure(figsize = (8,6))
visualize_trajectory(Training_dataset, index=90)

"""## Loss function"""

import numpy as np

import torch
```

```
from torch import Tensor


def pytorch_neg_multi_log_likelihood_batch(
    gt: Tensor, pred: Tensor, confidences: Tensor, avails: Tensor
) -> Tensor:
    """
    Compute a negative log-likelihood for the multi-modal scenario.
    log-sum-exp trick is used here to avoid underflow and overflow, For more information about it see:
    https://en.wikipedia.org/wiki/LogSumExp#log-sum-exp_trick_for_log-domain_calculations
    https://timvieira.github.io/blog/post/2014/02/11/exp-normalize-trick/
    https://leimao.github.io/blog/LogSumExp/
    Args:
        gt (Tensor): array of shape (bs)x(time)x(2D coords)
        pred (Tensor): array of shape (bs)x(modes)x(time)x(2D coords)
        confidences (Tensor): array of shape (bs)x(modes) with a confidence for each mode in each sample
        avails (Tensor): array of shape (bs)x(time) with the availability for each gt timestep
    Returns:
        Tensor: negative log-likelihood for this example, a single float number
    """
    assert len(pred.shape) == 4, f"expected 3D (MxTxC) array for pred, got {pred.shape}"
    batch_size, num_modes, future_len, num_coords = pred.shape

    assert gt.shape == (batch_size, future_len, num_coords), f"expected 2D (Time x Coords) array for gt, got {gt.shape}"
    assert confidences.shape == (batch_size, num_modes), f"expected 1D (Modes) array for gt, got {confidences.shape}"
    assert torch.allclose(torch.sum(confidences, dim=1), confidences.new_ones((batch_size,))), "confidences should sum to 1"
    assert avails.shape == (batch_size, future_len), f"expected 1D (Time) array for gt, got {avails.shape}"
    # assert all data are valid
    assert torch.isfinite(pred).all(), "invalid value found in pred"
    assert torch.isfinite(gt).all(), "invalid value found in gt"
    assert torch.isfinite(confidences).all(), "invalid value found in confidences"
    assert torch.isfinite(avails).all(), "invalid value found in avails"

    # convert to (batch_size, num_modes, future_len, num_coords)
    gt = torch.unsqueeze(gt, 1)  # add modes
    avails = avails[:, None, :, None]  # add modes and cords

    # error (batch_size, num_modes, future_len)
    error = torch.sum(((gt - pred) * avails) ** 2, dim=-1)  # reduce coords and use availability

    with np.errstate(divide="ignore"):  # when confidence is 0 log goes to -inf, but we're fine with it
        # error (batch_size, num_modes)
        error = torch.log(confidences) - 0.5 * torch.sum(error, dim=-1)  # reduce time

    # use max aggregator on modes for numerical stability
    # error (batch_size, num_modes)
    max_value, _ = error.max(dim=1, keepdim=True)  # error are negative at this point, so max() gives the minimum one
```

```
    error = -torch.log(torch.sum(torch.exp(error - max_value), dim=-1, keepdim=True)) - max_value  # reduce modes
    # print("error", error)
    return torch.mean(error)


def pytorch_neg_multi_log_likelihood_single(
    gt: Tensor, pred: Tensor, avails: Tensor
) -> Tensor:
    """

    Args:
        gt (Tensor): array of shape (bs)x(time)x(2D coords)
        pred (Tensor): array of shape (bs)x(time)x(2D coords)
        avails (Tensor): array of shape (bs)x(time) with the availability for each gt timestep
    Returns:
        Tensor: negative log-likelihood for this example, a single float number
    """
    # pred (bs)x(time)x(2D coords) --> (bs)x(mode=1)x(time)x(2D coords)
    # create confidence (bs)x(mode=1)
    batch_size, future_len, num_coords = pred.shape
    confidences = pred.new_ones((batch_size, 1))
    return pytorch_neg_multi_log_likelihood_batch(gt, pred.unsqueeze(1), confidences, avails)

"""## Model

Next we define the baseline model. Note that this model will return three possible trajectories together with confidence score for each trajectory.
"""

class LyftMultiModel(nn.Module):

    def __init__(self, configuration: Dict, num_modes=3):
        super().__init__()

        architecture = configuration["model_params"]["model_architecture"]
        backbone = eval(architecture)(pretrained=True, progress=True)
        self.backbone = backbone

        num_history_channels = (configuration["model_params"]["history_num_frames"] + 1) * 2
        num_in_channels = 3 + num_history_channels

        self.backbone.conv1 = nn.Conv2d(
            num_in_channels,
            self.backbone.conv1.out_channels,
            kernel_size=self.backbone.conv1.kernel_size,
            stride=self.backbone.conv1.stride,
            padding=self.backbone.conv1.padding,
            bias=False,
        )
```

```
        # This is 512 for resnet18 and resnet34;
        # And it is 2048 for the other resnets

        if architecture == "resnet50":
            backbone_out_features = 2048
        else:
            backbone_out_features = 512

        # X, Y coords for the future positions (output shape: batch_sizex50x2)
        self.future_len = configuration["model_params"]["future_num_frames"]
        num_targets = 2 * self.future_len

        # You can add more layers here.
        self.head = nn.Sequential(
            # nn.Dropout(0.2),
            nn.Linear(in_features=backbone_out_features, out_features=4096),
        )

        self.num_preds = num_targets * num_modes
        self.num_modes = num_modes

        self.logit = nn.Linear(4096, out_features=self.num_preds + num_modes)

    def forward(self, x):
        x = self.backbone.conv1(x)
        x = self.backbone.bn1(x)
        x = self.backbone.relu(x)
        x = self.backbone.maxpool(x)

        x = self.backbone.layer1(x)
        x = self.backbone.layer2(x)
        x = self.backbone.layer3(x)
        x = self.backbone.layer4(x)

        x = self.backbone.avgpool(x)
        x = torch.flatten(x, 1)

        x = self.head(x)
        x = self.logit(x)

        # pred (batch_size)x(modes)x(time)x(2D coords)
        # confidences (batch_size)x(modes)
        bs, _ = x.shape
        pred, confidences = torch.split(x, self.num_preds, dim=1)
        pred = pred.view(bs, self.num_modes, self.future_len, 2)
        assert confidences.shape == (bs, self.num_modes)
        confidences = torch.softmax(confidences, dim=1)
        return pred, confidences

def forward(data, model, device, criterion = pytorch_neg_multi_log_likelihood_batch):
    inputs = data["image"].to(device)
```

```
    target_availabilities = data["target_availabilities"].to(device)
    targets = data["target_positions"].to(device)
    # Forward pass
    preds, confidences = model(inputs)
    loss = criterion(targets, preds, confidences, target_availabilities)
    return loss, preds, confidences
```

"""Now let us initialize the model and load the pretrained weights. Note that since the pretrained model was trained on GPU, you also need to enable GPU when running this notebook."""

```
# ==== INIT MODEL=================
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model = LyftMultiModel(configuration)

weight_path = configuration["model_params"]["weight_path"]
if weight_path:
    checkpoint = torch.load(weight_path, map_location=torch.device('cpu'))
    model.load_state_dict(checkpoint)

model.to(device)
optimizer = optim.Adam(model.parameters(), lr=configuration["model_params"]["lr"])
print(f'device {device}')

print(model)
```

"""## Training loop

Next let us implement the training loop, when the **train** parameter is set to True.
"""

```
# ==== TRAINING LOOP
=============================================================
if configuration["model_params"]["train"]:

    tr_it = iter(Training_dataloader)
    progress_bar = tqdm(range(configuration["Training_params"]["max_num_steps"]))
    num_iter = configuration["Training_params"]["max_num_steps"]
    losses_train = []
    iterations = []
    metrics = []
    times = []
    model_name = configuration["model_params"]["model_name"]
    start = time.time()
    for i in progress_bar:
        try:
            data = next(tr_it)
        except StopIteration:
            tr_it = iter(Training_dataloader)
            data = next(tr_it)
        model.train()
        torch.set_grad_enabled(True)
```

```
        loss, _, _ = forward(data, model, device)

        # Backward pass
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        losses_train.append(loss.item())

        progress_bar.set_description(f"loss: {loss.item()} loss(avg): {np.mean(losses_train)}")
        if i % configuration['Training_params']['checkpoint_every_n_steps'] == 0:
            torch.save(model.state_dict(), f'{model_name}_{i}.pth')
            iterations.append(i)
            metrics.append(np.mean(losses_train))
            times.append((time.time()-start)/60)

    results = pd.DataFrame({'iterations': iterations, 'metrics (avg)': metrics, 'elapsed_time (mins)': times})
    results.to_csv(f"Training_metrics_{model_name}_{num_iter}.csv", index = False)
    print(f"Total training time is {(time.time()-start)/60} mins")
    print(results.head())

"""## Prediction"""

# ==== EVAL LOOP ================================================================
if configuration["model_params"]["predict"]:

    model.eval()
    torch.set_grad_enabled(True)

    # store information for evaluation
    future_coords_offsets_pd = []
    timestamps = []
    confidences_list = []
    agent_ids = []

    progress_bar = tqdm(Testing_dataloader)
    print(progress_bar)

    for data in progress_bar:

        _, preds, confidences = forward(data, model, device)

        #fix for the new environment
        preds = preds.detach().cpu().numpy()
        world_from_agents = data["world_from_agent"].numpy()
        centroids = data["centroid"].numpy()
        coords_offset = []

        # convert into world coordinates and compute offsets
```

```
        for idx in range(len(preds)):
            for mode in range(3):
                preds[idx, mode, :, :] = transform_points(preds[idx, mode, :, :], world_from_agents[idx]) -
centroids[idx][:2]

        future_coords_offsets_pd.append(preds.copy())
        # Convert confidences tensor to a NumPy array without gradients
        confidences_np = confidences.detach().cpu().numpy()
        confidences_list.append(confidences_np.copy())
        timestamps.append(data["timestamp"].numpy().copy())
        agent_ids.append(data["track_id"].numpy().copy())

pred_path = 'prediction1.csv'
write_pred_csv(pred_path,
            timestamps=np.concatenate(timestamps),
            track_ids=np.concatenate(agent_ids),
            coords=np.concatenate(future_coords_offsets_pd),
            confs = np.concatenate(confidences_list)
```

Appendix B – Certificate of Ethics Approval

Certificate of Ethical Approval

Applicant:	Nimmy Charley
Project Title:	Autonomous vehicle path detection using machine learning algorithms

This is to certify that the above named applicant has completed the Coventry University Ethical Approval process and their project has been confirmed and approved as Low Risk

Date of approval:	06 Jun 2023
Project Reference Number:	P158741

Appendix F – Project Presentation

AUTONOMOUS VEHICLE PATH DETECTION USING MACHINE LEARNING ALGORITHMS

Student Name: Nimmy Charley
Supervisor Name: Dr. Mahmoud Moradi
Second Marker : Dr. Long Chen

Coventry University

1

INTRODUCTION

Autonomous Vehicle Issues:

- Autonomous vehicles need precise path detection.
- Safe and effective navigation in various driving circumstances.

Machine learning:

- Objective: Autonomous automobile path detection.
- Machine learning for accuracy and efficiency.
- Safer and more reliable autonomous navigation.
- Modeling sensor data and decision-making.

Sensor Data:

- Using ONCE Dataset camera, lidar, and sensor data.
- Datasets train and assess machine learning models.
- Increasing the research's relevance to driving.
- Learning from actual driving scenarios.

2

EFFECTS OF THIS STUDY

- **Improved Safety:** Path detection reduces human errors and distractions, reducing accidents and fatalities.
- **Accessibility:** Helping non-drivers get around.
- **Increased Accessibility:** Reducing traffic congestion through connecting vehicles and infrastructure.
- **Environmental Benefits**: Encouraging fuel-efficient driving and fewer emissions.
- **Time savings:** allowing commuters to work or relax.
- **Economic Impact**: Reduced commuting time creates new jobs and boosts productivity.
- **Enhanced Traffic Management:** Real-time data and machine learning to predict traffic patterns and optimize infrastructure.
- **Ethical Considerations**: Ethical issues and social responsibility.
- **Regulatory Frameworks:** Creating autonomous vehicle legislation.
- **Cybersecurity:** Securing autonomous vehicles against cyberattacks.

3

OBJECTIVES

- To see if machine learning can improve autonomous vehicle path detection.
- To compare multiple machine learning models' road pattern and obstruction detection.
- To improve autonomous vehicle navigation by developing a sophisticated path detection model with sensor data.

4

PROBLEM STATEMENT

- The lack of reliable path detecting technologies in fully autonomous automobiles hinders safe and efficient navigation in tough driving circumstances.
- More accurate route detection algorithms are needed for autonomous cars to consistently identify lane markings, traffic signs, and barriers.
- Autonomous automobiles need accurate and real-time route recognition to navigate complex driving environments with changing road conditions and obstacles.
- Developing reliable and efficient path detection algorithms that can adapt to a wide range of driving scenarios to give autonomous vehicles the data they need to make safe decisions on the fly is difficult.

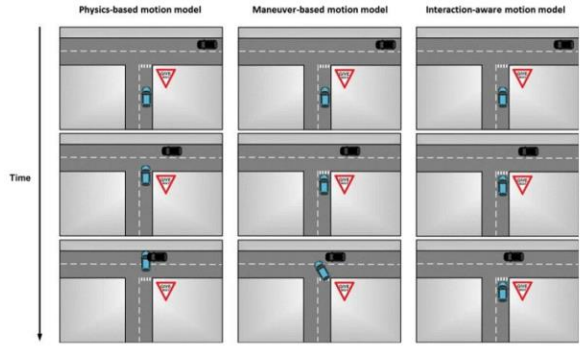
5

EXISTING STUDIES

Study	Year	Methodology	Main Findings
Teng et al. (2023)	2023	Review	Emphasizes the significance of path planning and the need for fast and accurate algorithms.
Li and He (2021)	2021	Reinforcement learning-based path planning	Reinforcement learning techniques enhance autonomous vehicles' decision-making ability.
Ye et al. (2021)	2021	Deep reinforcement learning-based path planning with ensemble model	Deep reinforcement learning algorithms combined with ensemble models improve path detection.
Kaur and Sobti (2020)	2020	Deep Convolutional Frameworks	Improved perception capabilities in autonomous vehicles' path detection.
Haris and Glowacz (2020)	2020	Comparative study of deep learning-based algorithms for object detection	Deep learning approaches show promise in path detection for autonomous vehicles.
Magid, Lavrenov, & Khasianov (2017)	2017	Modified spline-based path planning	Not directly related to machine learning, but presents an alternative path planning method.

6

PRIOR WORK - PREDICTION MODEL



Physics-based model - Only depend on the laws of physics

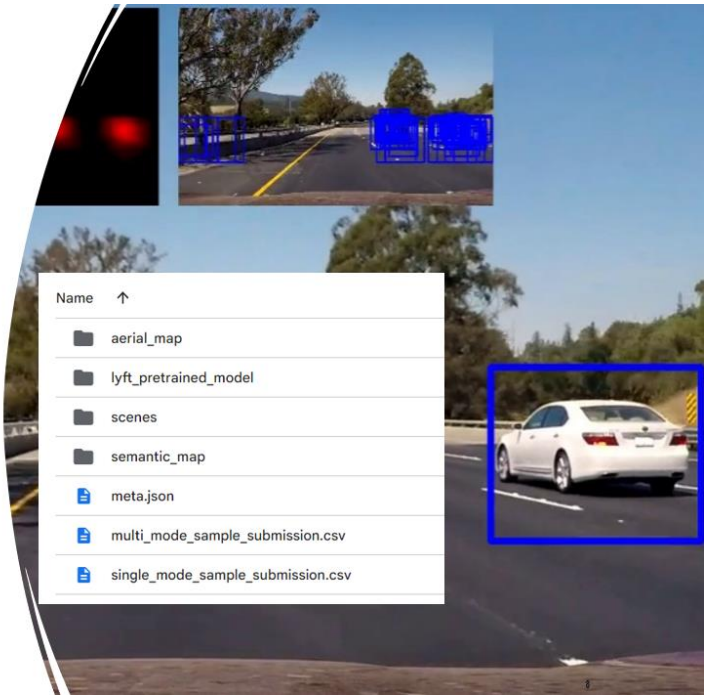
Maneuver-based model - Consider the maneuver that the driver intends to perform

Interaction-aware model - Take the interdependencies between vehicles' maneuvers

DATASET DESCRIPTION

Lyft Motion Prediction for Autonomous Vehicles

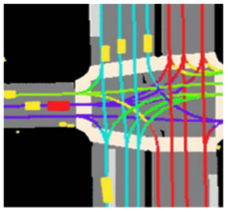
Dataset link: https://www.kaggle.com/competitions/lyft-motion-prediction-autonomous-vehicles/data

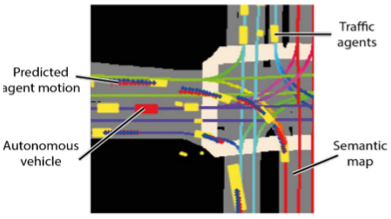


DATASET

Total data set size: 1,118 hours / 26,344 km / 162k scenes
- Training set size: 928 hours / 21,849 km / 134k scenes
- Validation set size: 78 hours / 1,840 km / 11k scenes
- Test set size: 112 hours / 2,656 km / 16k scenes

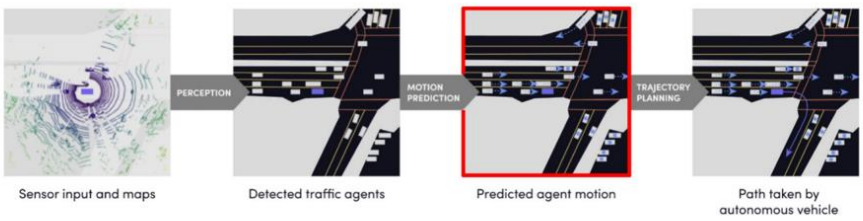
- Scene length: 25 seconds
- Total # of traffic participant observations: 3,187,838,149
- Average # of detections per frame: 79
- Labels: Car: 92.47% / Pedestrian: 5.91% / Cyclist: 1.62%
- Semantic map: 15,242 annotations / 8,505 lane segments
- Aerial map: 74 km2 at 6 cm per pixel



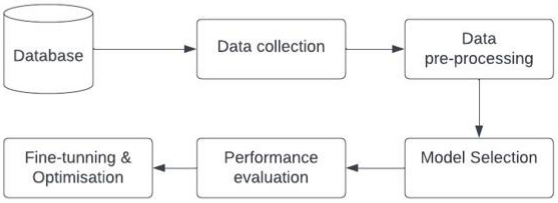


IDEA

- Predict the trajectory of the agent for the next 5s in future given their historical 5s positions
- Trained CNN regression pipeline with images



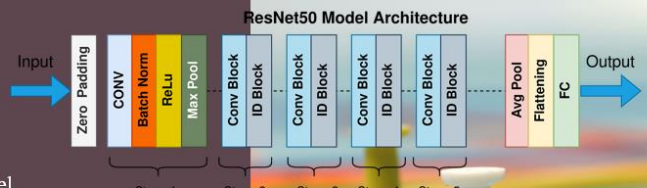
BLOCK DIAGRAM



METHODOLOGY

- Dataset Preparation: Preprocess the Lyft Motion Prediction dataset, including trajectory data, agent information, and other relevant features.
- ResNet Architecture: Implement the ResNet (Residual Neural Network) architecture, which utilizes skip connections to address the vanishing gradient problem. ResNet can be customized based on the input data and prediction task.
- Data Augmentation: Apply data augmentation techniques like rotation, translation, and flipping to increase dataset diversity and improve model robustness.
- Training
- Hyperparameter Tuning: Fine-tune hyperparameters, such as learning rate, batch size, and the number of layers in ResNet, to optimize model performance.
- Validation: Validate the trained model on the validation set to monitor overfitting and ensure generalization.
- Evaluation: Evaluate the final ResNet model on the test dataset to assess its performance for motion prediction.

METHODOLOGY



Model Training:

- PyTorch or TensorFlow the machine learning model.
- To reduce prediction errors, train the model on the training dataset using an optimization technique like Adam and a loss function like mean squared error.

Fine-tuning/Hyperparameter Optimization:

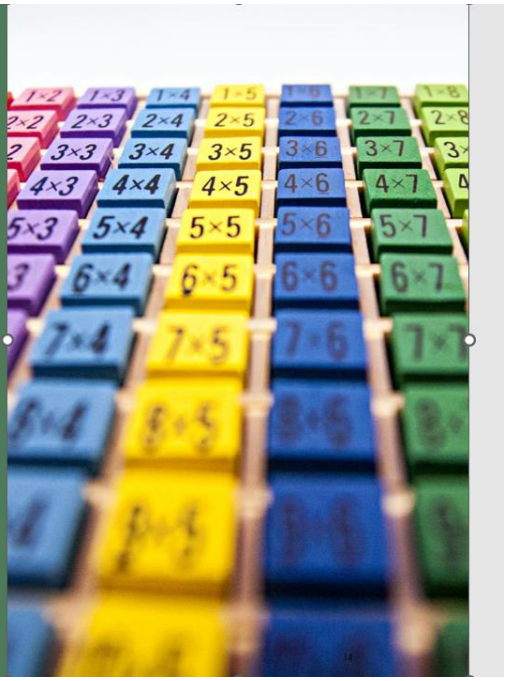
- Adjust hyperparameters and run experiments to optimize the model.
- To improve model generalization, try data augmentation methods.

Model Selection and Architecture:

- Assess autonomous vehicle path detection machine learning algorithms. Consider ResNet, CNNs, and other relevant algorithms.
- The dataset and job complexity determine the model design.

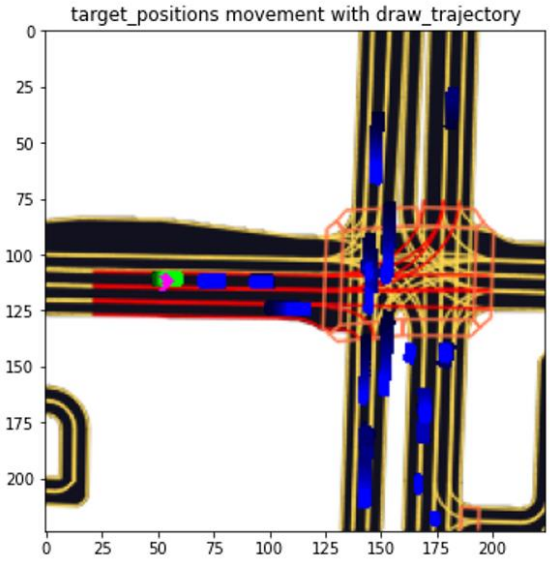
REQUIREMENTS

- ❑ Python 3.7 and above
- ❑ Jupyter notebook
- ❑ Library packages
  - Matplotlib.
  - NumPy
  - Pandas
  - Torch
  - Tqdm.
  - L5kit
  - GPU Supporting system



RESULTS

- This figure shows that, sample trajectory line drawn on the path.
- Where green color vehicle is our object and rest are the detection vehicle objects in autonomous path.



15

LYFT MULTIMODAL

This figure shows that, create LYFT multi modal using RESNET, where the CNN is the basic structure.

This model has four layers and all four layers are designed by sequential blocks.

The sample of first sequential layer is shown in the visual.

```
LyftMultiModel(
  (backbone): ResNet(
    (conv1): Conv2d(25, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
    (layer1): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
      (1): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
      (2): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
```

16

RESULTS

df.shape: (71122, 45)

	frame_index_interval_start	frame_index_interval_end	host	start_time	end_time	scene_db_id	timestamp	agent_index_interval_start	agent_index_interval_end	traffic_light_faces_index_interval_start	traffic_light_faces_index_interval_end	ego_trans
0	0	100	host-a118	1578605997692823296	1578606022692823296	0	1578606007801600134	8302	8385	48	54	5
1	100	200	host-a118	1578606022692823296	1578606047692823296	1	1578606032802467516	16433	16522	609	615	6
2	100	200	host-a118	1578606022692823296	1578606047692823296	1	1578606032802467516	16433	16522	609	615	6
3	100	200	host-a118	1578606022692823296	1578606047692823296	1	1578606032802467516	16433	16522	609	615	6
4	100	200	host-a118	1578606022692823296	1578606047692823296	1	1578606032802467516	16433	16522	609	615	6
5	200	300	host-a118	1578606047692823296	1578606072692823296	2	1578606057802432986	24831	24926	615	615	7
6	300	400	host-a118	1578606072692823296	1578606097692823296	3	1578606082801997166	31374	31419	615	615	9
7	300	400	host-a118	1578606072692823296	1578606097692823296	3	1578606082801997166	31374	31419	615	615	9
8	400	500	host-a118	1578606097692823296	1578606122692823296	4	1578606107802708166	38282	38341	615	615	9
9	500	600	host-a118	1578606122692823296	1578606147692823296	5	1578606132802907546	46257	46301	1506	1515	7

The above visual shows that, sample data frame of the testing Lyft dataset. And it contains 71122 records and 45 column attributes.

RESULTS

- In general, the dictionary appears to keep track of details about a trained LightGBM model, such as the location of the model's training data and the characteristics that were used as input.
- It's possible that this model was developed with a 30-second lag in mind.
- The below DataFrame shows that final detected trajectory with its co-ordinates.

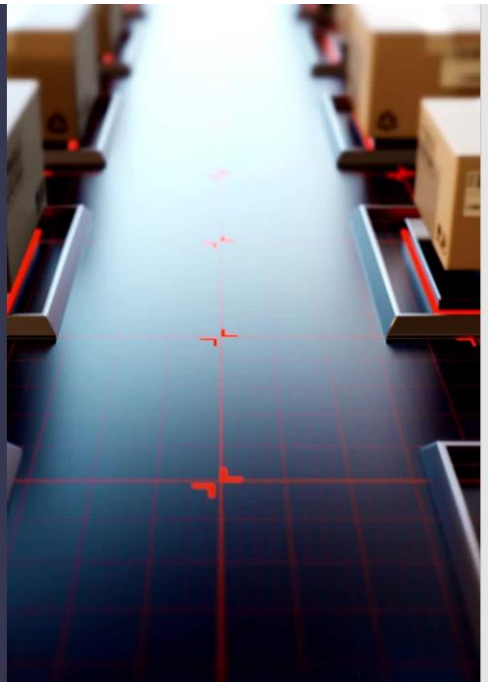
```
('lgbm_y_shift_30',
 {'model': '../input/lyft-models/lgbm_06/lgbm_y_shift_30.bin',
  'train_cols': ['extent_x',
   'extent_y',
   'extent_z',
   'velocity_x',
   'velocity_y',
   'label_probabilities_PERCEPTION_LABEL_UNKNOWN',
   'label_probabilities_PERCEPTION_LABEL_CAR',
   'label_probabilities_PERCEPTION_LABEL_CYCLIST',
   'label_probabilities_PERCEPTION_LABEL_PEDESTRIAN',
   'yaw',
   'nagents',
   'nlights',
   'centroid_xs',
   'centroid_ys']})
```

	timestamp	track_id	conf_0	conf_1	conf_2	coord_x00	coord_y00	coord_x01	coord_y01	coord_x02	coord_y02	coord_x03	coord_y03	coord_x04	coord_y04	coord_x05	coord_y05	coord_x06	coord_y06	coord_x07	coord_y07	coo
0	1.578606e+18	2.0	0.502480	0.205125	0.292395	-0.11272	0.22052	-0.22827	0.47644	-0.33984	0.70938	-0.43328	0.92182	-0.51222	1.13399	-0.62933	1.36481	-0.72203	1.57098	-0.81811	1.78168	-0
1	1.578606e+18	4.0	0.380651	0.358710	0.260639	-0.63477	-0.97205	-1.21299	-1.90659	-1.71423	-2.77318	-2.30782	-3.74341	-2.92686	-4.79764	-3.49349	-5.82457	-4.02905	-6.89436	-4.57615	-7.93131	-5
2	1.578606e+18	5.0	0.542430	0.093180	0.364390	0.12414	0.20155	0.25162	0.38692	0.32335	0.48187	0.40778	0.64741	0.52292	0.79794	0.62248	0.93191	0.72048	1.11641	0.82735	1.29089	0
3	1.578606e+18	81.0	0.213896	0.720917	0.065187	-0.25885	-0.41525	-0.50425	-0.77621	-0.67320	-1.08616	-0.87364	-1.37187	-1.05663	-1.67769	-1.24778	-1.93477	-1.44620	-2.22686	-1.60099	-2.52611	-1
4	1.578606e+18	130.0	0.035118	0.953250	0.011632	0.01142	-0.00426	0.02232	-0.01539	0.04585	-0.01270	0.07871	-0.02273	0.09174	-0.03819	0.11129	-0.04303	0.14994	-0.06392	0.14345	-0.06494	0

18

CONCLUSION

- Autonomous vehicle path identification: Critical Path Identification.
- Machine learning: Improved path detection in difficult driving situations.
- Systematic Approach: Tested and implemented machine learning approaches systematically.
- Path Recognition Model: Used ONCE Dataset sensor data to accurately detect lane markers, traffic signs, and barriers.
- Preprocessed data for model training and evaluation.
- Safety and Efficiency: Met goals, improving autonomous vehicle safety, efficiency, and compliance.
- Successful installation advanced autonomous car technology.



FUTURE WORK

- Real-World Testing: Validate path detection under different driving, weather, and lighting circumstances.
- Improve model reliability across areas and traffic patterns.Edge Computing: Optimize autonomous car real-time path detection without external processing.
- Continuous Learning: Apply continuous learning to new driving scenarios and road conditions.
- Multi-Agent Interaction: Consider other vehicles, pedestrians, and cyclists in complex traffic to improve path prediction.

20

REFERENCES

Kaur, A., & Sobti, A. (2020). Deep Convolutional Framework for Perception Systems in Intelligent Autonomous Vehicles. International Journal of Intelligent Transportation Systems Research, 18(1), 12-23.

Haris, M., & Glowacz, A. (2020). Comparative study of deep learning-based algorithms for road object detection. Journal of Intelligent Transportation Systems, 24(4), 346-358.

Teng, S., Hu, X., Deng, P., Li, B., Li, Y., Ai, Y., Yang, D., Li, L., Xuanyuan, Z., Zhu, F., & Chen, L. (2023). Motion planning for autonomous driving: The State of the art and future perspectives. IEEE Transactions on Intelligent Vehicles, 1–21. https://doi.org/10.1109/tiv.2023.3274536

Li, Y., & He, H. (2021). Reinforcement learning-based path planning for autonomous vehicles. Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 4213-4219.

Ye, F., Wang, P., Chan, C.-Y., & Zhang, J. (2021). Meta reinforcement learning-based Lane Change Strategy for autonomous vehicles. 2021 IEEE Intelligent Vehicles Symposium (IV). https://doi.org/10.1109/iv48863.2021.9575379

Magid, E., Lavrenov, R., & Khasianov, A. (2017). Modified spline-based path planning for Autonomous Ground Vehicle. Proceedings of the 14th International Conference on Informatics in Control, Automation and Robotics. https://doi.org/10.5220/0006442601320141

21

THANK YOU !!!!!!

22