



Politecnico
di Bari

Repository Istituzionale dei Prodotti della Ricerca del Politecnico di Bari

Intelligent path planning using deep learning and reinforcement learning in transportation systems

This is a PhD Thesis

Original Citation:

Intelligent path planning using deep learning and reinforcement learning in transportation systems / Esmaeil Abbasi, Ahmad. - ELETTRONICO. - (2026).

Availability:

This version is available at <http://hdl.handle.net/11589/296260> since: 2026-01-24

Published version

DOI:

Publisher: Politecnico di Bari

Terms of use:

(Article begins on next page)



The doctoral scholarship was funded by the European Union - Next Generation EU, Mission 4
Component 2-PHD CANDIDATES RECEIVING SCHOLARSHIPS UNDER M.D. 351/352-
CUP D93D22001320001

Department of Electrical and Information Engineering
ELECTRICAL AND INFORMATION ENGINEERING PhD PROGRAM

SSD: ING-INF/04-AUTOMATICA

Final Dissertation

**Intelligent Path Planning Using Deep Learning and Reinforcement
Learning in Transportation Systems**

by

PhD Student:

AHMAD ESMAEIL ABBASI

A thesis submitted for the degree of Doctor of Philosophy

Supervisor:

Prof. Maria Pia Fanti

Co-Supervisor:

Prof. Paolo Lino

PhD Program Coordinator:

Prof. Mario Carpentieri

Cycle XXXVIII - November, 1st 2022 - October 31st 2025

ACADEMIC YEAR 2025-2026

The doctoral scholarship was funded by the European Union - Next Generation EU, Mission 4
Component 2-PHD CANDIDATES RECEIVING SCHOLARSHIPS UNDER M.D. 351/352-
CUP D93D22001320001

Department of Electrical and Information Engineering
ELECTRICAL AND INFORMATION ENGINEERING PhD PROGRAM

SSD: ING-INF/04-AUTOMATICA

Final Dissertation

Intelligent Path Planning Using Deep Learning and Reinforcement
Learning in Transportation Systems

by

PhD Student:

AHMAD ESMAEIL ABBASI

A thesis submitted for the degree of Doctor of Philosophy

Supervisor:

Prof. Maria Pia Fanti

Referee:

Prof. MMMM

Co-Supervisor:

Prof. Paolo Lino

Referee:

Prof. NNNN

PhD Program Coordinator:

Prof. Mario Carpentieri

Cycle XXXVIII - November 1st, 2022 – October 31st, 2025

ACADEMIC YEAR 2025–2026

Dedication

*This dissertation is dedicated to my beloved **Family**, whose steady support, love, and encouragement have been the foundation of my journey.*

Their sacrifices and constant belief in me gave me the strength and inspiration to pursue and complete my PhD studies. I am deeply grateful for their presence in every step of this academic path.

*I would also like to extend my sincere thanks to my **Country** for providing the necessary facilities and opportunities for my prior education levels. Without these combined efforts and support, this achievement would not have been possible.*

Acknowledgment

*First of all, I have to express my appreciation to **Italy** in order to gave me the opportunity to pursue my PhD study. Then , thanks to **Prof. Fanti** as my supervisor and her colleagues: **Prof. Mangini** and **Prof. Roccotelli** support, I pursued and finished my PhD degree in Politecnico di Bari (Poliba). They helped me alot to finish this journey. Moreover, I have to thank my friends to help and support me alot during the my PhD program which was part of my life in beautiful **Bari, Italy**.*

Abstract

Nowadays, transportation is one of the major challenges faced by the public, as many people commute daily between their homes and workplaces. To tackle this issue, individuals are encouraged to use public transportation. At the same time, the development of Intelligent Transportation Systems (ITS) is becoming increasingly important in modern society. As a result, people are more willing to adopt transportation systems that are smart, safe, and environmentally friendly. Modern vehicles increasingly incorporate advanced technologies such as object detection, reinforcement learning (RL), deep learning (DL), and machine learning (ML) to improve safety, efficiency, and traffic management.

A significant research challenge in autonomous vehicles is robust object detection under various environmental conditions, such as fog, rain, and low-light scenarios. In this study, a YOLO (You Only Look Once)-based object detection model is developed to identify vehicles, pedestrian areas or crosswalks, traffic lights, and road signs on roads. The model is evaluated on images captured in both foggy and sunny conditions, demonstrating its ability to maintain high detection accuracy despite reduced visibility or image noise. This work contributes to enhancing real-time perception in autonomous systems, which is critical for safe navigation and traffic safety.

Another important problem addressed in this research is optimal path planning in urban traffic networks, which is essential for reducing congestion and minimizing travel time. A multi-input convolutional neural network (CNN) is employed to process map data from the SUMO (Simulation of Urban MObility) environment, including features such as road edges, traffic light locations, and vehicle distributions. By integrating these features, the CNN is able to determine optimal routes dynamically, considering real-world factors such as traffic density and signal placement. This approach improves the reliability of autonomous navigation in complex urban scenarios.

Finally, the study uses reinforcement learning (RL) to explore dynamic routing. On a simulated metropolitan map of Bari, Italy, a Q-Learning framework is used to determine the shortest and most efficient paths while accounting for potential dead-end routes, traffic lights, vehicle density, and road topology. Through iterative updates of the Q-table, the RL agent learns optimal routes that balance avoiding traffic and minimizing travel time. This approach shows how RL may be used practically for autonomous vehicle routing and adaptive traffic management.

Overall, this research combines YOLO-based object detection, CNN-based path planning, and RL-based dynamic routing to provide an integrated framework for autonomous transportation. The

proposed methods contribute to enhancing safety, reducing congestion, and improving efficiency in urban traffic systems, offering both scientific insights and practical solutions for modern ITS applications.

Contents

Frontpage	1
Referees	2
Dedication	3
Acknowledgment	4
Abstract	5
List of Publications	8
List of Figures	9
List of Tables	13
Abbreviations	16
1 Introduction	19
1.1 Background	19
1.1.1 Motivation	21
1.1.2 Problem Statements	22
1.1.3 Thesis Contribution	23
1.1.4 Thesis Structure	25
2 Deep Learning Applications in Object Detection	28
2.1 Background	28
2.2 Literature Review and Related Work	31
2.2.1 Data to Be Used	32
2.3 Research Gap	33
2.3.1 Object Detection Models and Metrics	34
2.3.2 Object Detection Background	36
2.4 Approach	37
2.4.1 Accuracy Analysis	42
2.5 Findings and Implications	44
2.6 Future Work	77

3	Convolutional Neural Network - CNN	79
3.1	Background	79
3.2	Literature Review and Related Work	81
3.3	Research Gap	83
3.4	Approach	83
3.5	Findings and Implications	99
3.5.1	Modified Hyperparameters Without Speed Factor	100
3.5.2	Parameter Description	100
3.5.3	Modified Hyperparameters With Speed Factor	114
3.6	Future Work	128
4	Reinforcement Learning - Q-Learning	129
4.1	Background	129
4.2	Literature Review and Related Work	131
4.3	Research Gap	133
4.4	Approach	135
4.5	Findings and Implications	151
4.6	Future Work	164
5	Conclusion	166
5.1	Overall Conclusions	166
	References	168

List of Publications

Back to Contents

Ahmad Esmaeil Abbasi , Agostino Marcello Mangini, Maria Pia Fanti.

Object and Pedestrian Detection on Road in Foggy Weather Conditions by Hyperparameterized YOLOv8 Model.

Electronics, 2024, 13(18), 3661.

<https://doi.org/10.3390/electronics13183661>

Ahmad Esmaeil Abbasi , Yassine Ouazene, Maria Pia Fanti.

Convolutional Neural Networks with Stable-Baselines for Optimized Path Planning by Simulation.

Submitted on 24 Sep 2025, Taylor and Francis journal in Applied Artificial Intelligence.

<https://www.tandfonline.com/journals/uaai20>

Ahmad Esmaeil Abbasi , Agostino Marcello Mangini, Maria Pia Fanti.

Using Reinforcement Learning for path planning.

Ready to Submit to journal.

List of Figures

[Back to Contents](#)

List of Figures

2.1	Methodology Flowchart	39
2.2	One sample of image in foggy weather conditions characterized by 1920×1080 pixels stored into training folder.	40
2.3	Image obtained after Step 1 characterized by 940×540 pixels	40
2.4	One sample outcome from the validation dataset (folder).	41
2.5	One sample of prediction data.	42
2.7	Precision vs. Epoch for method 1.	45
2.9	Recall vs. Epoch for method 1	46
2.11	Final result for method 1 (100 epochs, batch size 16, without augmentation and YOLOv8S).	48
2.12	Precision–confidence curve for method 2 (100 epochs, batch size 16 and with augmentation, YOLOv8L).	49
2.13	Precision vs. epoch for method 2	50
2.14	Recall–confidence curve for method 2 (100 epochs, batch size 16 and with augmentation, YOLOv8L).	50
2.15	Recall vs. epoch for method 2	51
2.16	Precision–recall curve for method 2 (100 epochs, batch size 16, with augmentation and YOLOv8L).	51
2.17	Result for method 2 (100 epochs, batch size 16 and with augmentation, YOLOv8L).	52
2.18	Precision–confidence curve for method 3 (100 epochs, batch size 64 and without augmentation, YOLOv8S).	53
2.19	Precision vs. epoch for method 3	53
2.20	Recall–confidence curve for method 3 (100 epochs, batch size 64 and without augmentation, YOLOv8S).	54
2.21	Recall vs. epoch for method 3	55
2.22	Precision–recall curve for method 3 (100 epochs, batch size 64 and without augmentation, YOLOv8S).	55
2.23	Result for method 3 (100 epochs, batch size 64 and without augmentation, YOLOv8S).	56

2.24	Precision–confidence curve for method 4 (150 epochs, batch size 64 and without augmentation, YOLOv8S).	57
2.25	Precision vs. epoch for method 4	58
2.26	Recall–confidence curve for method 4 (150 epochs, batch size 64 and without augmentation, YOLOv8S).	58
2.27	Recall vs. epoch for method 4	59
2.28	Precision–recall curve for method 4 : (150 epochs, batch size 64 and without augmentation, YOLOv8L).	59
2.29	Result for method 4 (150 epochs, batch size 64 and without augmentation, YOLOv8S).	60
2.30	Detected objects by the proposed model in Semantic Segmentation Makassar (IDN) Road Dataset—the second test dataset.	63
2.31	Detected objects from traffic Signs Dataset in YOLO format—the third test dataset.	64
2.32	Detected Car in suburbs from traffic Signs Dataset in YOLO format.	64
2.33	Detected Objects from front and rear image of car dataset with complex background—the first test dataset.	65
2.34	Precision rate for method 1 in test dataset (good and sunny dataset).	66
2.35	Recall rate in method 1 in the test dataset (good and sunny dataset).	67
2.36	Precision–recall rate in method 1 in test dataset (good and sunny dataset).	67
2.37	Final Report rate in method 1 in the test dataset (good and sunny dataset).	68
2.38	Precision rate in the method 2 approach in sunny and good image dataset (100 EPC, BS16, AUG and YOLOv8L in test dataset).	68
2.39	Recall rate in the method 2 approach in sunny and good image dataset (100 EPC, BS16, AUG and YOLOv8L in test dataset).	69
2.40	Precision–recall rate in the method 2 approach in sunny and good image dataset (100 EPC, BS16, AUG and YOLOv8L in test dataset).	69
2.41	The final report in the method 2 approach in sunny and good image dataset (100 EPC, BS16, AUG and YOLOv8L in test dataset).	70
2.42	Precision rate in the method 3 approach in sunny and good image dataset (100 EPC, BS16, No AUG and YOLOv8S in test dataset).	70
2.43	Recall rate in the method 3 approach in sunny and good image dataset (100 EPC, BS16, No AUG and YOLOv8S in test dataset).	71
2.44	Precision-Recall rate in the method 3 approach in sunny and good image dataset (100 EPC, BS16, NO AUG and YOLOv8S in test dataset).	71
2.45	The final report in the method 3 approach in sunny and good image dataset (100 EPC, BS16, No AUG and YOLOv8S in test dataset).	72
2.46	Precision rate in the method 4 approach in sunny and good image dataset (150 EPC, BS64, No AUG and YOLOv8S in test dataset).	72
2.47	Recall rate in the method 4 approach in sunny and good image dataset (150 EPC, BS64, No AUG and YOLOv8S in test dataset).	73
2.48	Precision-Recall rate in the method 4 approach in sunny and good image dataset (150 EPC, BS64, No AUG and YOLOv8S in test dataset).	73
2.49	The final report in the method 4 approach in sunny and good image dataset (150 EPC, BS64, No AUG and YOLOv8S in test dataset).	74
3.1	A part of Bari in Italy as the environment in the study-The SUMO sample map	85
3.2	The proposed CNN model grid channels	87
3.3	Proposed CNN model Schema	87

3.4	The proposed CNN structure	88
3.5	How the Kernel 2x2 works on an image	90
3.6	Kernel 2x2 performance- Step 01	91
3.7	Kernel 2x2 performance- Step 02	92
3.8	Kernel 2x2 performance- Step 03	93
3.9	Kernel 2x2 performance– Step 04	94
3.10	An example of Flatten performance on an image	95
3.11	The Dense layer performance	95
3.12	The concatenate process	98
3.13	The First CNN outcome for searching an optimal path between edges E7 and E10	101
3.14	The Second outcome in the same direction from E7 to E10 by AdamW optimizer and several changes in hyperparameters	103
3.15	The Third outcome for the optimal path from E0 to E14	105
3.16	The Fourth path planning outcome for the optimal path between E0 and E6	107
3.17	The Fifth CNN outcome from 800 epochs in the training process	109
3.18	The Sixth CNN outcome with modified hyperparameters for E0 to E14	111
3.19	The First outcome from the CNN with added Speed factor from 29980616#1 to E13	115
3.20	The outcome from the Second approach, which utilized the speed factor in the CNN From E0 to E14.	118
3.21	The Third method outcome for investigatin an optimal path between E17 and E1	120
3.22	The outcome for the Fourth strategy for an optimal path from E7 to E66576723#3	122
3.23	The Fifth approach outcome for an optimal path between E24884076 and E30016109#2	123
3.24	The result of the last (Sixth) method for finding an optimal path from E24884077#1 to E13	125
4.1	The environment in the fist step in the SUMO software	138
4.2	The start and end edges on the map in the first step in the SUMO environment.	139
4.3	The Outgoing edges for edge E2	140
4.4	The proposed RL algorithm	141
4.5	The training process for finding the optimal path in the first step. of the study.	145
4.6	The Q-table items	148
4.7	The first step outcome with the help of the proposed RL model.	149
4.8	Loaded Vehicles are enhanced on the Scenario	150
4.9	Comparison between number of traffic lights were used in the first (A) and the second step (B).	150
4.10	The outcome path as an optimal path by RL	152
4.11	The second try for discover optimal path with different start edge	153
4.12	The beginning edge , last edges , the agent and the optimal route	153
4.13	The route with less waiting time	154
4.14	The optimal path is tracked by the Agent (Veh1000)	155
4.15	The optimal path and number of cars are loaded as a traffic on the map	155
4.16	Tracking the optimal path by the Agent	156
4.17	The agent follows the optimal route is extracted by RL	156
4.18	The agent waiting time graph	157
4.19	The waiting time chart for the agent	157
4.20	Total number of vehicles are loaded in the simulation	158
4.21	The training outcome route length per iteration and reward per iteration	159

4.22 Investigated valid route between two random desired edges with the help of using trained data	161
4.23 Failing to find the path between two random desired edges	163

List of Tables

[Back to Contents](#)

List of Tables

2.1	Prediction rate for detecting objects in method 1	47
2.2	Prediction rate for detecting objects in method 2	52
2.3	Prediction rate for detecting objects in method 3	56
2.4	Prediction rate for detecting objects in the method 4	57
2.5	Precision, recall and mAP50 metrics for the four proposed methods in the foggy dataset.	60
2.6	Object detection methods and classes used in other articles.	62
2.7	The predicted object detection accuracy rate by the proposed model in good and sunny images in the test dataset (good and sunny image datasets).	65
2.8	Precision, Recall, and mAP50 (%) for proposed methods on sunny test dataset. . .	74
2.9	The predicted object detection accuracy rate in foggy and good weather conditions in the proposed model.	74
2.10	Precision, recall and mAP50 metrics in YOLOv8 compared with the metrics in YOLOv5 for the foggy image dataset.	75
2.11	Precision, recall and mAP50 metrics in YOLOv8 compared with those in YOLOv5 over the good image dataset.	76
2.12	The average predicted object detection accuracy rate in foggy and good weather conditions in YOLOv8 compared with YOLOv5 based on the four methods used. . .	76
3.1	The First Configuration Parameters for CNN-Based Traffic Routing	100
3.2	The First CNN performance summary for optimal path prediction on a 300-edge map between edges E7 and E10	102
3.3	The Second Configuration Parameters for CNN-Based Traffic Routing for E7 to E10.	102
3.4	The Second CNN performance summary for optimal path prediction on a 300-edge map between edges E7 and E10	104
3.5	The Third Configuration Parameters for CNN-Based Traffic Routing For E0 to E14	104
3.6	The Third CNN Performance Summary for Optimal Path Prediction on 300-Edge From E0 to E14 on the SUMO map , ($\rightarrow$ = To)	106

3.7	The Fourth Configuration Parameters for CNN-Based Traffic Routing For E0 to E6.....	106
3.8	The Fourth CNN performance summary for optimal path prediction on a 250-edge SUMO map between random origin and destination edges for E0 to E6.....	107
3.9	The Fifth Configuration Parameters for CNN-Based Traffic Routing For E0 to E6	108
3.10	The Fifth CNN performance summary for optimal path prediction on a 300-edge SUMO map between random origin and destination edges for E0 to E6	110
3.11	The Sixth Configuration Parameters for CNN-Based Traffic Routing For E0 to E6	111
3.12	The Sixth CNN performance summary for optimal path prediction on SUMO map between random start and goal edges (E0 to E14)	112
3.13	CNN performance comparison for optimal path prediction on SUMO map between random origin and destination edges across different methods without Speed factor. Mtd = Method, EPC = Epochs, Opt. = Optimizer, Increase (↑), Decrease (↓), To (→).	112
3.14	The First Configuration Parameters for CNN-Based Traffic Routing For 29980616#1 to E13.....	115
3.15	The outcome of the First way- CNN performance with Speed factor summary for optimal path prediction on SUMO map between random start and goal edges (E2980616#1 to E13).....	116
3.16	The Second Configuration Parameters for CNN-Based Traffic Routing From E0 to E14.....	117
3.17	The outcome of Second method - CNN performance with Speed factor summary for optimal path prediction on SUMO map between random start and goal edges (From E0 to E14)	119
3.18	The Third Configuration Parameters for CNN-Based Traffic Routing For E17 to E1.....	119
3.19	The outcome of Third approach - CNN performance with Speed factor for optimal path prediction on SUMO map (E0 to E14)	121
3.20	The Fourth Configuration Parameters for CNN-Based Traffic Routing From E7 to E66576723#3	121
3.21	The outcome of the Fourth strategy - CNN performance with Speed factor for optimal path prediction on SUMO map (E7 to E66576723#3)	122
3.22	The Fifth Configuration Parameters for CNN-Based Traffic Routing From E24884076 to E30016109#2	123
3.23	The outcome of the Fifth strategy - CNN performance with Speed factor for optimal path prediction on SUMO map (E24884076 to E30016109#2).....	124
3.24	The Sixth Configuration Parameters for CNN-Based Traffic Routing From E24884077#1 to E13	125
3.25	The outcome of the Sixth strategy - CNN performance with Speed factor for optimal path prediction on SUMO map (E24884077#1 to E13)	126
3.26	CNN performance comparison for optimal path prediction on SUMO map between random origin and destination edges across different methods, Increase (↑), Reduce (↓), VL=Very Low, L= Low, M=Moderate, → = to	126
4.1	Comparison of Articles Regarding Q-learning Metrics Used in Their Studies ...	133
4.2	Q-Learning Configuration Parameters	143

4.3	As an example: training outcomes between 1 and 15 000 episodes for a desired path among two random desired edges on the map	148
4.4	Performance of the proposed algorithm over 15,000 training episodes for an optimal path between two desired edges on the SUMO map	152
4.5	Key parameters analyzing in the proposed RL model in training process for two random desired edges as an example.	160
4.6	Performance Metrics over 15,000 Training Episodes	160
4.7	An example in the proposed RL-based SUMO route generation between two random desired user-defined edges using the trained data in Q-learning training model procedure.	162
4.8	Detailed Failure Case in Route Finding	163

Abbreviations

Back to Contents

AI	Artificial Intelligence
ITS	Intelligent Transportation Systems
CAM	Cooperative And Automated
RL	Reinforcement Learning
ML	Machine Learning
DL	Deep Learning
YOLO	You Only Look Once
YL	YOLO
M	Millions
AV	Autonomous Vehicle
EV	Electric Vehicle
AEV	Autonomous Electric Vehicle
SUMO	Simulation of Urban MObility
TraCI	Traffic Control Interface
Fast-RCNNs	Fast Region-based Convolutional Neural Networks
HSV	Hue–Saturation–Value
COCO	Common Objects in Context
EMA	Exponential Moving Average
DR-CNN	Deconvolution Region-based Convolutional Neural Network
SGD	Stochastic Gradient Descent
RMSprop	Root Mean Squared Propagation
OHEM	Online Hard Example Mining
ANN	Artificial Neural Network
GPS	Global Positioning System
Lidar	Light Detection and Ranging
GPU	Graphic Processing Unit

Ped	Pedestrians area or crosswalk
mAP50	mean Average Precision
IoU	Intersection over Union
val/df1-loss	validation distribution focal loss
RCNN	Region-based Convolutional Neural Network
IDN	Semantic Segmentation Makassar
YOLOv8S	YOLOv8 Small
YOLOv8L	YOLOv8 Large
FRN	Faster-RCNN (Region-based Convolutional Neural Network)
RN	RCNN (Region-based Convolutional Neural Network)
EPC	Epoch
BS16	batch size 16
AUG	Augmentation
YOLOv8S	YOLO version 8 Small
YOLOv8L	YOLO version 8 Large
YOLOv5S	YOLO version 5 Small
YOLOv5L	YOLO version 5 Large
YOLOv3S	YOLO version 3 Small
YOLOv3L	YOLO version 3 Large
YOLOv7L	YOLO version 7 Large
YOLOv7S	YOLO version 7 Small
DQNs	Deep Q-Networks
HEVs	Hybrid Electric Vehicles
DDQN	Double Deep Q-learning Network
SARSA	State-Action Reward-State-Action
OSM	Open Street Map
IDE	Integrated Development Environment
v-90	Vehicle ID 90
veh1000	Vehicle ID 1000
CNNs	Convolutional Neural Networks
SUMO	Simulation of Urban Mobility
SB3	Stable Baselines 3
GCNN	Graph Convolutional Neural Networks
EAVs	Electric Autonomous Vehicles

LSTM	Long-Short-Term Memory
BiLSTM	Bidirectional Long Short-Term Memory
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
LR	Learning Rate
TraCI	Traffic Control Interface

Chapter 1

Introduction

[Back to Contents](#)

1.1 Background

Deep learning is one of the famous, well-known and contemporary technologies and is a part of artificial intelligence as well. It is especially used in complex, decision-making and automation. By contrast, former algorithms, deep learning models can learn directly from large amounts of information and extract important features. Application of deep learning enables autonomous vehicles, robots and smart navigation systems to benefit from it where both accuracy and adaptability are necessary for success [1].

Besides, one of the challenges in deep learning is how to have access to a large amount of high-quality data. Without sufficient training examples, the model cannot learn when it faces a new and unexpected situation. For example, an autonomous vehicle trained only in sunny weather might struggle when it encounters fog or heavy rain. Therefore, researchers attempt to simulate real situations and environments to improve a model to become a reliable model for path planning [2].

The importance of using deep learning in these areas is object detection. For robots and autonomous vehicles that are now being used in various industries, it plays a vital role. First of all, they must recognize obstacles and objects in different environments; then they can make decisions on how to move. Object detection models help the system identify obstacles, pedestrians, crosswalks, street signs, and traffic lights in real-time. It leads to creating a map of the surrounding space. Without accurate object detection, even the most famous navigation systems fail to plan an effective or safe route [3].

For recognizing objects, contemporary systems need to estimate their positions and movements. It is known as object tracking, and it has a direct connection with detection. It is insufficient to know there is a pedestrian; the system must predict that the pedestrian is currently passing the crosswalk or standing in their location. The dynamic understanding makes path planning reliable, more accurate, and reduces the risk of an accident [4].

Convolutional Neural Networks (CNNs) form the foundation of most object detection models. CNNs are employed to analyze images by capturing spatial features, such as edges and shapes,

and combining them to recognize complex objects. For instance, in self-drive vehicles, CNNs can quickly categorize road items, such as cars, traffic lights, pedestrian areas, and traffic signs. This visual understanding allows the system to observe the environment in a manner to make the best decision in navigation and path planning [5].

Moreover, CNNs face some limitations, such as high computational costs that require substantial hardware resources. In real-time navigation systems, decisions need to be made in milliseconds, so any processing delay can cause failures in handling data like text or images. To rectify these problems, researchers offer using lightweight CNN architectures and optimized approaches. It allows faster processing of the data without wasting accuracy. Achieving this balance is crucial for safe and efficient path planning.

Reinforcement learning also plays another major role in decision-making over time. While CNNs are good at perception, reinforcement learning aims for an agent to learn how to act in an environment. The mechanism used in reinforcement learning is rewards or penalties for the agent's actions in the environment. The agent can learn strategies to reach the desired goal with collision avoidance and/or delay. This trial-and-error process makes reinforcement learning a natural fit for path planning, where the agent must continuously select the best moves to reach the desired goal or its destination efficiently [6].

One of the worst challenges in reinforcement learning is to deal with latency in the training process. It must be done by an agent in the environment to explore all possible actions. In path planning, the agent might need to have more than a million trials before it can learn effectively. To tackle the problem, researchers employ adjusting hyperparameters and rewards. The changes enhance the agent's learning process to take measurements faster and efficiently [7].

When perception is integrated with decision-making strategies, it enables the system to mimic human behavior in driving or navigation. Humans first observe their surroundings and, based on that input, make decisions such as turning, accelerating, or slowing down. Deep learning and reinforcement learning allow the agent to both perceive the environment and decide on actions. This human-like decision-making cycle is what makes these technologies effective for path planning [8].

In addition, deep learning models like CNNs for perception are combined with reinforcement learning for action. They create a strong framework for intelligent path planning. For example, a robot can recognize objects in a path with the help of CNNs, while reinforcement learning learns to choose the safest, most optimal, and efficient route around those obstacles. This approach is employed in autonomous driving, drone navigation, and robotic delivery. These mentioned methods are becoming more reliable, paving the way for machines that can move safely and independently in complex environments [9].

As a result, using and combining deep learning and reinforcement learning helps the agent or an autonomous vehicle to be predictable regarding path planning and take the best actions in different environments. Researchers are exploring ways to improve both technologies so as to have accurate and efficient outcomes. Due to this, they can have positive impacts on path planning, collision-free and safe paths based on time and fuel, and energy consumption as well [10].

1.1.1 Motivation

The analysis the prior research regarding deep learning and object detection has showed me that there is a gap that need to be address, Due to, in some object detection techniques , the authors have employed various algorithms such as CNN (Convolutional neural network), RCNN(Region-based Convolutional Neural Networks) and Faster-RCNN and YOLO (You Only Look Once) models by using large amount of images and huge epochs to train their model. However, unfortunately, the majority of article benefited from some filters in their object detection technique in this way, they used some rain, fog, or snow filter over their raw images, then they attempted to train their models. It leads to some errors and unclear outcomes in the research. As the raw images cover with the filters and they are different from the real image situation. Then, we employed raw and proper images for training our proposed model for both foggy and sunny images to overcome the gap. The object detection topic is an important topic in Artificial Intelligence (AI). If autonomous vehicles or robots as an agent can detect obstacles or objects on the road, it leads to preventing accidents and saving energy , reducing traffic congestion, and having optimal travel time as well [11].

Moreover, selecting the proper images in training process is able to significantly improve the accuracy and reliability of object detection models. By focusing on realistic and varied datasets that accurately reflect actual road conditions, as opposed to relying on overly altered or edited images, models are able to adapt more effectively to real-world applications. For instance, an autonomous vehicle or a robot that learns from high-quality, raw images to represent better roads, obstacles, pedestrians, and traffic signs. It is better equipped to accurately identify and detect objects in various environments [12].

Furthermore, using the detection model play a vital role in the outcome. Models such as YOLO and Faster-RCNN provide faster detection procedure to detect objects and obstacles on roads. It is a useful model for real time application and complicated environments. It is important to note, we must select a model that has a balance between speed and precision as it will provide better solutions for robots or autonomous vehicles. Also, if we have a proper model in object detection and decision-making topics.

Previous research addressing traffic bottlenecks has utilized convolutional neural networks (CNNs), often in combination with other techniques, to identify optimal routes in path planning. These approaches generally relied on a single model input. By contrast, the proposed model integrates two inputs into the CNN to improve the selection of optimal edges for path determination on a SUMO sample. The model processes multiple inputs by employing a function that evaluates input conditions and selects the most suitable edges for the SUMO map [13].

Integrating multiple inputs into a CNN model for autonomous vehicles and mobile robots significantly enhances environmental perception compared to relying on a single data source. For example, one input might provide information about traffic density or current road conditions, while another could report on object detection, identifying obstacles , pedestrians or edges' speed in the vicinity. By combining these varied data streams, the model is better equipped to assess real-time scenarios and make more informed decisions about route selection or navigation. Employing this multi-input strategy not only improves the system's reliability but also minimizes or loop-free edges

or unsafe paths [14].

In addition, usage of multi-input CNN can have positive effects on path planning that has the potential to enhance overall system performance for both autonomous vehicles and robots. If we use high quality data, the model can have smoother navigation , energy efficiency and travel time reduction.

1.1.2 Problem Statements

Object detection on roads under adverse weather conditions, such as rain, fog, and snow, presents significant challenges. Foggy conditions, in particular, reduce visibility, introduce noise, and obscure texture information in images, making accurate detection of objects like vehicles, pedestrians, traffic lights, and signs difficult for both humans and computer models [15]. In the context of autonomous vehicles, reliable object detection is critical for real-time decision-making, safe navigation, and accident prevention [16]. The primary goal of this research is to enhance object detection accuracy in foggy road images by addressing issues related to image clarity and brightness, leveraging machine learning (ML) and deep learning (DL) techniques [17], [18], [19]. Successfully improving object detection under these conditions could significantly enhance autonomous system control, reduce accidents, improve traffic management, and increase road safety [20].

Traffic congestion in metropolitan cities has become a persistent public challenge. Advanced technologies, including object detection [21], [22], ML, DL, and reinforcement learning (RL), have been explored to optimize traffic flow. Convolutional Neural Networks (CNNs), a DL approach, are particularly effective as they can learn hierarchical features from raw data without manual feature engineering [22], [23].

Despite this, previous CNN-based studies often underutilized multiple input features, limiting model performance in complex traffic scenarios. To address this gap, we propose a multi-input CNN architecture capable of concurrently processing various inputs, such as image features, environmental conditions (e.g., fog, slope, lighting), and map attributes. This approach allows the model to make more informed decisions when identifying objects and determining optimal routes in a simulated 2D environment, SUMO. By integrating multiple inputs, our CNN can capture both positive and negative factors affecting object detection and road path planning, improving overall model robustness and accuracy.

Another critical challenge in metropolitan traffic systems is optimal path planning, which involves finding the best route between two locations. Traditional methods, such as shortest-path algorithms or traffic light-based routing, often fail under heavy congestion [24], [25], [26]. The shortest path is not always the most efficient because traffic density, traffic lights, and dead-end roads can significantly increase travel time [27].

Reinforcement Learning (RL), specifically Q-Learning, offers a dynamic solution for routing optimization in urban networks [28], [29]. In this study, the city map is represented as a set of junctions $J = \{j \mid j = 1, \dots, n\}$ and edges $E = \{E_1, E_2, E_3, \dots, E_n\}$. The RL agent, representing a vehicle, navigates the map by selecting outgoing edges at each junction. The reward function is designed to prioritize travel-time efficiency: edges with heavy traffic or dead ends receive penalties, while reaching the goal yields a bonus. This structure allows the Q-Learning model to iteratively update

the Q-table during training, enabling the agent to learn optimal trajectories dynamically, even in the presence of traffic congestion and complex intersections.

RL has proven effective in transportation and vehicle management tasks. For instance, Double Deep Q-Networks (DDQN) have been successfully applied for fuel management in hybrid electric vehicles, improving convergence and optimization by mitigating overoptimistic policy estimates [30], [31]. Inspired by these results, our approach uses a combination of travel time, traffic density, number of vehicles, and edge distance as a single reward signal, ensuring that the RL agent balances multiple metrics to determine the optimal path.

Despite advances in CNNs and RL, existing studies do not fully integrate multiple input features for object detection or consider multi-metric decision-making for path planning in congested urban environments. Specifically: The first: CNN models for road object detection rarely exploit multiple concurrent inputs to handle challenging environmental conditions, such as fog. The second : Traditional shortest-path algorithms and RL-based routing methods often fail to simultaneously account for travel time, traffic congestion, dead-end edges, and traffic signals in a unified framework.

This research addresses these gaps by proposing:

- A multi-input CNN for robust object detection under foggy conditions.
- An RL-based Q-Learning framework for dynamic and multi-metric optimal path planning in urban traffic networks.

By integrating these models, the study aims to increase autonomous navigation, reduce congestion-related delays, and improve safety and efficiency in real-world traffic scenarios.

1.1.3 Thesis Contribution

This thesis attempts to rectify several challenges that are related to investigating the optimal path for autonomous vehicles or robots in modern countries. Finding an optimum path is connected to a major topic for autonomous cars and robots, which is called path planning. It is important to note that path planning for vehicles has been an important subject over the years, as it can have some advantages directly or indirectly, such as reduction of cost ,fuel and energy consumption, travel time, air and noise pollution, car collisions, environmental impact and etc. In addition, it lead to enhancing some aspects such as , battery life, traffic flow, road safety ,user experience and etc for city dwellers. The major contributions are highlighted as follows .

- Cost, Fuel and Energy reduction : Path planning helps lower operational costs by reducing fuel and energy use. In logistics and transportation, fuel expenses are often among the largest costs. Optimizing travel routes is a key way to tackle this issue. By finding the most efficient routes based on distance, energy, and time, path planning makes sure that vehicles and robots consume fewer resources. It also reduces wear and tear on vehicles, leading to lower maintenance expenses. For companies that depend on fleets of autonomous delivery vehicles, these savings can add up over time, offering a significant edge over competitors [32], [33].

- **Less wear and tear on operations:** A smooth, optimized path can help autonomous vehicles and robot mechanical systems more . Unexpected turns, quick acceleration, and sudden stops can all have negative effects on important parts like motors, gears, tires, and others. Autonomous systems can work with less stress on their hardware if they follow a route that is more predictable and uses less energy. Minimizing wear and tear is important for keeping operations reliable that is why it makes the cars' parts more reliable and reduce downtime in applications like self-driving trucks or factory robots [34],[35].
- **Reduced Environmental Impact:** Path planning helps autonomous systems have a smaller impact on the environment by using less fuel and encouraging energy-efficient driving. Vehicles use less fuel or battery power when they choose routes that avoid traffic jams, steep hills, and inefficient travel patterns. This directly lowers emissions. This is especially important when it comes to urban mobility and self-driving delivery systems, where cars often drive short distances through crowded areas. These vehicles can avoid crowded routes thanks to optimized path planning. This means less pollution, less carbon footprint, and fewer emissions. This helps make cities more sustainable and supports green transportation projects [36], [37].
- **Extended Battery Life:** Electric vehicles (EVs) and mobile robots can have their batteries last a lot longer if their path planning takes energy efficiency into account. Instead of taking the shortest route, which might require a lot of acceleration or steep hills, a more energy-efficient route might be a longer, flatter one that uses less battery power. Path planning algorithms can find the best routes that balance energy use and operational time by taking into account the types of terrain, speed limits, and the robot's own power consumption model. This is especially important when there aren't many charging stations or when robots or cars are working in remote or off-grid areas[38], [39].
- **Increased Operational Efficiency:** Path planning enables autonomous systems to do jobs more efficiently, whether they are traveling through a factory, delivering items, or driving around town. These systems can minimize operational time by identifying the best time- and energy-efficient paths, allowing them to do more jobs in the same amount of time. In logistics, this could mean shorter shipping times, whereas in manufacturing, it could mean faster product assembly. The ability to optimize path selection in real time, such as road closures or traffic density, guarantees that operations run smoothly and efficiently. As a result, production and customer satisfaction rise, particularly in companies that value speed and efficiency[40], [41].
- **Better Resource Allocation:** Path planning is very useful in multi-agent or multi-vehicle systems, such as fleets of self-driving trucks or warehouse robots. In this research, a single agent was employed to validate the proposed reinforcement learning and path planning framework. Using a single agent reduces computational complexity and resource requirements compared to multi-agent systems, which often demand significantly more memory, processing power, and coordination overhead. While multi-agent systems are a natural extension, focusing on a single agent allowed us to extremely analyze learning dynamics and performance before scaling to more complex, resource-intensive scenarios in future work. Path planning ensures that jobs are adequately allocated and resources, such as fuel, energy, or time, are not squandered by coordinating the movement of several robots or vehicles. In a delivery fleet, for example, path planning algorithms can optimize routes for each vehicle depending

on proximity to adjacent packages, current traffic conditions, and each vehicle's operational limits. This guarantees that each vehicle is used efficiently, reducing idle times and eliminating redundancy in task performance, hence increasing overall resource utilization[42], [43].

- **Improved User Experience:** Path planning in autonomous vehicles improves the user experience by optimizing routes for comfort, speed, and convenience. For customers, this means smoother trips with less unexpected turns or stops, which is especially significant in applications like self-driving cars and transfer services. Autonomous vehicles can make driving more fun and less stressful by avoiding severely maintained roads, high-traffic regions, and construction zones. Furthermore, path planning can modify routes based on passenger preferences, such as avoiding scenic roads or decreasing travel times. This level of modification helps to increased pleasure and trust in autonomous technologies[44], [45].

1.1.4 Thesis Structure

This thesis contribution consists of five sections. The first section includes a proposed object detection method that emphasizes deep learning technology with the help of the well-known model YOLO version 8. It is used to recognize the most important items on roads. The second section is related to how we can find out an optimum path on a sample of Bari's city map in Italy using the Simulation of Urban MObility (SUMO) simulation tool.

The first section begins with an introduction that explains various technologies, such as deep learning and its recent importance and connection with autonomous vehicles or robots. For instance, autonomous vehicles have benefited from this technology to navigate routes without a human driver. This is followed by a discussion of the motivation behind the research approaches employed. Next, the section transitions to the problem statements that prompted the writing of this thesis. Here, we introduce and clarify the specific gaps addressed in each part of our research. The first gap relates to deep learning, focusing on object detection in the worst weather conditions. The next gap concerns the search for an optimal path on a SUMO map based on elements commonly encountered by drivers on roads. Finally, the third gap addresses the investigation of an optimal path using reinforcement learning on a sample SUMO map.

In the background section in each chapter have attempted to set the stage for the investigation by providing essential background information, context, and a clear rationale for the work. In some cases, It may incorporate key data, statistics, or examples to emphasize the significance and relevance of the problem under study. By presenting these elements, the background highlights existing gaps or unresolved knowledge issues that the research aims to address. It establishes a coherent framework for the study and guides the reader toward specific research objectives or questions, providing a concise but comprehensive overview of the importance and scope of the topic. To provide a detailed understanding, the following sections elaborate on each contribution, starting with object detection methods and progressing to path optimization techniques.

In the literature review section of each chapter, we will focus on reviewing and presenting prior researches in connection with object detection by deep learning models, path planning via a CNN model, and reinforcement learning model within the fields of autonomous systems. The literature review will emphasize that studies using algorithms, including CNN, RCNN, Faster-RCNN, and

YOLO, for object detection under various environmental information, and present their strengths and weaknesses. We will focus on the impacts of dataset quality, particularly filtered versus raw images, as a method to enhance accuracy and reliability when performing object detection. The literature review will also consider existing work on CNN architectures and train them for traffic congestion problems, path planning, and optimal decision-making juxtaposed for single-input and multi-input models. Finally, the review will search for existing literature using reinforcement learning methods, specifically Q-learning, with the intent prolonging navigation to remain optimal path selection in a complete urban environment. The literature review will outline the current state of work, gaps in the literature, and illustrate how the remainder of the thesis contributes as an extension of the existing body of literature.

In the following part of each chapter, as can be seen, it is named as research gap. In this part of the thesis, the goal is to clearly identify the restriction of current methods in object detection, path planning, and reinforcement learning, and to explain why these shortcomings do not adequately solve today's challenges. Moreover, this section will start by summarizing important problems which were identified in earlier studies, such as the use of filtered or synthetic images for object detection, the poor performance of object detection in bad weather, and the limitations of using single-input CNN models, which limit how well the system can understand the environment. It will also discuss the shortcomings of current reinforcement learning methods in traffic and path optimization, especially their failure to effectively consider multiple factors like traffic jams, traffic signals, dead ends, and changing travel times. By bringing these gaps together, the section sets up the need for models that (the first study) use real and realistic data to boost detection accuracy, (the second study) use multi-input CNNs for better decision-making, and (the third study) apply reinforcement learning that can manage complex and changing traffic situations. To present the problem this way lays the groundwork for the goals and methods of the thesis, making sure each suggested solution directly tackles a known problem in the existing research.

Building on this background, the subsequent literature review examines prior research related to object detection, path planning, and reinforcement learning, highlighting both achievements and remaining gaps.

The next section in the whole chapters, represented as approach. In the approach section have been attempted to outline the detailed procedures, techniques, and used frameworks to rectify the research questions in each chapter. These sections describe the reason/s behind choosing specific methods. In this section, the experimental or analytical design, the data sources, and the tools or employed technologies will be presented. They also outline the step-by-step processes for data collection, processing, and analysis, ensuring precision and research transparency. In addition, any assumptions, parameters, or constraints that guided the implementation are discussed. By systematically documenting these elements, the methodology sections provide a clear foundation for understanding how the outcome and conclusions were obtained, while it also links the approach in each chapter to the corresponding published work.

Upcoming section in every chapter is, finding and implications. It belongs to characterize the results based on the method/methods were applied. The findings sections of every chapters of this thesis will describe the outcomes obtained from the used approaches in each chapter. These sections often include measurable and/or descriptive data, observations, measurements, and computational results. Tables, figures, or graphs are frequently used to demonstrate major trends and patterns in the results. The emphasis is on objectively describing the results, with or without interpretation,

while highlighting the most relevant and significant findings for each research issue addressed in the chapter.

Furthermore, the implications explain the results, evaluating their importance in terms of current literature, theoretical frameworks, and practical applications. These sections generally look at how the outcomes support or question prior work, offer reasonable explanations for patterns which were identified, and explain the research's overall relevance or influence. They may also identify limits, offer future study directions, and provide a critical assessment of the results' reliability and validity. By explicitly separating the presentation of findings from their interpretation, these sections illustrate both the empirical contributions and the intellectual value of the study.

Finally, the last section of each chapter is titled Future Work, which serves to wrap up the chapter coherently. In this section, we provide a summary of the key findings presented in the chapter, highlighting the main results and insights gained from the research. We also restate the contributions made in that chapter, emphasizing how they advance knowledge in the field and their practical significance. Moreover, we suggest directions for future research, offering ideas on how the findings and methods could be extended, improved, or applied in subsequent studies. By summarizing the findings, reaffirming the contributions, and outlining potential research directions, this section not only concludes the chapter effectively but also guides future investigations, ensuring continuity and relevance in the broader context of the thesis.

Chapter 2

Deep Learning Applications in Object Detection

[Back to Contents](#)

2.1 Background

Autonomous systems are impressive technologies that some industries, companies, and individuals are using today. These systems can operate in various fields, thanks to object detection technologies found in industries, medicine, departure gates, and autonomous vehicles [46]. Recently, machine learning (ML) and deep learning (DL) have significantly impacted research [47] and are now part of autonomous systems to detect objects on the streets [48]. One of the most common and important uses of object detection technologies among these applications is in driver-less vehicles.

Among the various uses of object detection, self-driving systems are one of the most important areas where these technologies are applied. Companies that make autonomous vehicles use sensors and object detection models to identify objects on roads during bad weather. They use sensors like Lidar, ultrasonics, and GPS to find people and cars. Limited information on roads in autonomous vehicles can lead to more accidents, especially in adverse weather. Self-driving cars rely on these technologies to prevent accidents and protect lives, often outperforming skilled drivers [48]. Even though these technologies have produced amazing results, outside environmental factors—especially the weather—continue to affect how well they perform.

Detecting objects in clear weather is relatively easy. However, object detection becomes more challenging in foggy, rainy, snowy, or other tough conditions. One of the main issues is recognizing objects when visibility is poor. Some objects can be found using standard models, but foggy weather poses significant challenges. When fog is thick, road images suffer from reduced visibility, poor quality, and lack of contrast, color, and sharpness. These environmental restriction emphasize the need for studies to enhance object detection under tough conditions.

Autonomous driving is one of the most important uses of artificial intelligence and computer vision, where safety and reliability matter greatly. However, in the environment, some factors, such as in low-visibility environments like fog, rain, and snow, remain a big challenge for autonomous vehicles in object detection accurately. These conditions can lower visibility and reduce the quality

of sensor data, making even advanced detection systems less effective. Therefore, improving object detection in poor weather conditions has become a major research focus, as it directly supports safer and more reliable autonomous navigation.

This study aims to tackle this ongoing challenge. Although there has been significant progress through machine learning (ML) and deep learning (DL) models, most current approaches work well only in clear weather and struggle in foggy conditions. This issue at the same time, has good effects on both object detection accuracy and o raises the risk of collisions in real-life driving situations. As a result, it is crucial to create and test a strong detection method that maintains high accuracy across different visibility levels.

The primary goal of this research is to develop and evaluate an object detection method that can accurately identify key road elements, such as vehicles, pedestrians, traffic lights, and street signs, in foggy conditions. By using the latest YOLOv8 model with well-tuned hyperparameters and data augmentation techniques, this work seeks to enhance detection reliability without changing the base network structure. This study contributes to both the understanding of vision-based detection under changing environmental conditions and the development of safer autonomous driving systems.

Despite the use of advanced sensors, environmental conditions such as fog and rain still create major challenges for object detection. Building on these challenges in bad weather, our research focuses on an important part of autonomous driving. The main goal of this study is to detect objects in poor weather, which can also be applied in self-driving cars without human assistance. In this research, we present a method to tackle the problem of object detection in foggy conditions [49]. We aim to achieve a higher object detection rate compared to other studies in the field. Some earlier research tried different models to detect objects in images affected by worse weather conditions. However, they failed to identify all the key items on the road, such as cars, pedestrians, traffic lights, and street signs. Our proposed method seeks to address this gap by utilizing a YOLOv8 model to identify important objects in foggy weather for autonomous driving. Reviewing the fundamental ML and DL methods used in object detection is crucial to understand the foundation of our suggested strategy.

ML and DL rely on raw materials like texts, databases, and images [50]. In object detection, the raw material includes pictures. Some of the most efficient DL models benefit from CNNs (convolutional neural networks), Fast-RCNNs (Fast Region-based Convolutional Neural Networks), and YOLO (You Only Look Once) [51]. The eighth version of YOLO, YOLOv8, is the fastest DL model for recognizing objects in images [52],[17]. In [18], experts utilized a modified version of YOLOv7, known as Ghost-YOLOv7, which enhances feature extraction and facilitates vehicle detection in normal weather conditions. Researchers have investigated different adaptations to address detection challenges under varying environmental conditions, building on these foundational models.

Addressing these limitations needs new methods that can keep accuracy even when visibility is poor. While these models have shown promise in various applications, researchers have adapted and combined them to solve specific challenges in object detection. Several studies have looked at different ways to tackle object detection in various weather conditions and scenarios. In [49], the authors introduce a method that combines YOLOv3 and the Hue–Saturation–Value (HSV) color space to detect vehicles on streets during rainy weather. This method compares image frames

captured in good and rainy weather by cameras placed in specific locations. It fixes items, rain droplets, and vehicles using these cameras. In [20], a method based on Faster-RCNN is presented to recognize traffic lights on roads. This method has a reduced detection time thanks to AlexNet, a trained linear support vector machine. Other researchers have concentrated on increasing detection accuracy across various object types and viewpoints in addition to these weather-related studies.

Paper [53] presents a method to detect vehicles and pedestrians using images taken from drones. The authors compared several models such as YOLOv3, YOLOv4, R-CNN, and Faster-RCNN to achieve high accuracy and fast detection rates. In [19], the authors use an R-CNN model to detect street signs with the help of deep learning. They apply R-CNN to extract text from the street signs for a high detection rate. In other studies, researchers combine well-known models with some pre-trained models. In [50], the authors use a modified YOLOv3 model to detect multiple objects with the help of a Common Objects in Context (COCO) pre-trained dataset. Unfortunately, their model fails to detect all objects in the images. It becomes crucial to examine why more recent models, such as YOLOv8, achieve better detection performance in the context of the limitations of earlier methods.

To understand why YOLOv8 is effective, it helps to briefly review the role of ML and DL models in object detection. In [54], a model was studied to detect traffic lights on roads. The authors use color segmentation and shape property detection from different models combined into their model. Paper [51] offers a method based on a CNN to detect traffic lights in both low and high lighting conditions. The model features two output layers. The first layer has one output value for the binary classification task, which determines whether a sample contains traffic lights. The second layer has four output values for the regression task, which localizes traffic lights by identifying the upper-left point of a bounding box, along with its width and height. In [55], the authors present a modified Faster-RCNN model. They added a deconvolutional layer to the output and named it the DR-CNN model. This proposed model detects traffic signs on the street and categorizes them into three groups: small, medium, and large objects. Nevertheless, some environmental issues have not been addressed in regardless of the wide range of models that have been investigated.

Although these studies show progress in object detection in different weather conditions, they mostly ignore the particular situation of foggy environments. Despite these advancements, there is still a big gap in the literature on object detection in foggy conditions. None of the papers mentioned can detect cars, pedestrians, traffic lights, or street signs in foggy weather. In other research [46, 54, 16, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 15, 68, 69] some researchers attempted to detect two or three object classes. Their results are not very good in some cases. We suggest a new approach created particularly for object detection in foggy conditions in order to fill this research gap.

The proposed model is a new method for building a system that can detect objects in bad weather, particularly in fog. This approach can be effectively used in self-driving vehicles. The goal is to achieve a higher object detection rate compared to existing studies. The method can detect cars, pedestrians, traffic lights, and street signs using YOLOv8 without modifying it or adding extra filters. Instead, we choose suitable hyperparameters through trial and error. Specifically, in deep learning applications, hyperparameters are set or configurations are often established before the learning process starts. Some common hyperparameters include the following: batch size, epoch, learning rate, augmentation, and optimizer. We carefully reviewed the model's innovations and

contributions after it was developed.

The new contribution of the paper is as follows. While the proposed method is based on the existing YOLOv8 model, the main innovation is its use and improvement for detecting objects in foggy street images, which is a difficult and less-studied area. Our approach includes several new aspects, like image preprocessing. We adjust the image quality and size to better fit the training process, which is important for improving performance in different real-world situations. We carried out detailed hyperparameter optimization, examining their effects closely to fine-tune our model for detecting objects in foggy conditions. We also used brightness augmentation, which greatly improved detection rates in fog. Our study carefully evaluated this augmentation and offered insights into its advantages and drawbacks. We carried out in-depth tests with various YOLOv8 variants and examined how well they performed in foggy and good conditions in order to validate these contributions.

We used different versions of the YOLOv8 model, both small and large, to see how their performances varied. This highlighted the trade-offs between model complexity and efficiency. To address concerns about detailed discussion, we expanded the paper to include a thorough analysis of how brightness augmentation affects detection accuracy[70] , [71] , along with quantitative results and qualitative examples. The paper also compares the small and large YOLOv8 models, pointing out their advantages and suitability for various applications.

2.2 Literature Review and Related Work

Deep learning has changed object detection by allowing for the precise identification of important elements like cars, crossroads, traffic lights, and signs. One of the key features of YOLO among deep learning models is its use of a single neural network. It predicts bounding boxes and class probabilities directly from images, ensuring both speed and accuracy. Its unified design simplifies detection, classification, and localization tasks, making it suitable for applications that require quick decisions, such as autonomous vehicles and traffic management. YOLO continues to advance object detection, improving safety and efficiency worldwide. In paper [66], the authors modified the YOLOv7 model to detect only people on roads. They applied the object detection algorithm to identify people in foggy conditions. They included the dark channel de-fogging algorithm alongside four layers of down-sampling, which reduced the number of pixels in each dimension and consequently the amount of data. They then used up-sampling to boost the number of pixels in each dimension to speed up the de-fogging process. To ensure accuracy in complex environments, they integrated the ECA module and an additional detection head into the YOLOv7 network design, which improved object classification and detection. While YOLOv7 showed strong performance in foggy scenes, other researchers explored alternative YOLO architectures and training strategies to enhance adaptability

In [58], the authors used the YOLOv5 architecture as the base and created two separate models: the student model and the teacher model. The student model received updates through backpropagation, while the teacher model adjusted its weights using the Exponential Moving Average (EMA) of the student model. This process allowed the teacher model to gather insights from various instances of the student model over time. They alternated between training the student

model on a supervised source domain and a pseudo-supervised target domain. At the same time, the teacher model produced pseudo labels for the target domain by updating them through the EMA, using the outputs from the student model. They also suggested a cross-attention strategy transformer to help match features between the source and target domains through knowledge distillation. Their model was able to detect people, cars, bikes, trains, and motorbikes.

Earlier versions, such as YOLOv3, also demonstrated the model's versatility when trained on large datasets. Also, the YOLO model can use a pre-trained dataset for training to identify some objects. The YOLOv3 model was used in [52] for car detection. They used a pre-trained COCO dataset as input images in their study. They focused on detecting cars on roads with COCO. We also tried using COCO at first, but traffic signs and traffic lights were not labeled when we used it. As a result, we decided to create a model to detect those objects using the YOLOv8 model.

In contrast to YOLO-based models, other researchers have pursued region-based CNN methods to address challenges in detecting small or distant objects. In [55], the authors used a deconvolution region-based convolutional neural network (DR-CNN) to handle small objects in large images. Detecting small objects is complex due to the numerous layers in the neural network when using R-CNN. First, they added a deconvolution layer and a normalization layer to the output of the convolution layer. They focused on detecting traffic lights in the study.

Similarly, the Faster-RCNN approach has been extended to pedestrian detection with specialized training techniques. In [60], the Faster-RCNN network architecture used feature concatenation and hard negative mining strategies to detect pedestrians on streets. They employed a specific classifier to improve the pedestrian detection rate in their study. Then they trained their model on the Daimler pedestrian dataset. In the next step, they used a hard negative mining approach to classify samples with classifiers and treated the misclassified samples as negative samples to continue training the classifier.

Despite the significant progress made with models like YOLOv3, YOLOv5, YOLOv7, Faster-RCNN, and DR-CNN, current research still has important limitations. Most previous studies focus on detecting specific object types, such as pedestrians, vehicles, or traffic lights, in normal or slightly degraded weather conditions. Only a few studies specifically look at object detection in foggy environments, and even these often concentrate on just one type of object. Additionally, while CNN-based models like R-CNN and Faster-RCNN achieve high accuracy, they require a lot of computing power. By contrast, transformer-based models and hybrid YOLO versions are faster but it can not be high reliable in low-visibility situations. This shows a need for a model that can maintain detection speed, handle various object types, and work accurately in foggy weather. Therefore, this research builds on the strengths of previous YOLO methods by using the YOLOv8 architecture. It enhances this model with optimized hyperparameters and data augmentation techniques to provide consistent multi-object detection performance in foggy conditions.

2.2.1 Data to Be Used

The first step in object detection is data collection, which plays a crucial role in training machine learning and deep learning models. Raw materials such as texts, images, and numbers serve as inputs [50]. For autonomous vehicles, detecting objects around the vehicle relies on Lidar, sensors,

and object detection technologies [55]. Typically, data is gathered using manual or semi-automated approaches [48]. For this study, we adopted a semi-automated method to efficiently collect a diverse set of approximately 4,000 foggy street images, which were annotated with the Roboflow tool <https://roboflow.com/> (accessed on 11 September 2024).

In the manual method, humans collect all the images. However, this process takes a lot of time, and the results are not efficient. On the other hand, the semi-automated approach involves gathering images on roads using a vehicle equipped with a high-quality dashboard camera. Also, the camera position on the car is important [72]. Once the images are extracted frame by frame, a computer processes them, resulting in training and validation datasets [48, 68]. Considering these approaches, we chose the semi-automated method to effectively gather a varied and representative set of foggy street images for training the model.

If the dataset lacks enough variety and quantity, models may struggle to generalize to real-world situations. The collected foggy street images were evaluated using precision, recall, precision-recall, accuracy, and loss metrics [50]. This dataset, along with clear annotations and selected evaluation metrics, forms the basis for building a robust object detection system capable of operating effectively in foggy conditions.

2.3 Research Gap

Detecting objects on roads is increasingly difficult in poor weather like rain, fog, and snow. The dataset used includes resized images taken during foggy conditions. The main research question in this study is: How can object detection models reliably identify cars, pedestrians, traffic lights, and street signs in foggy road conditions using raw image datasets? The researchers worked to identify objects such as cars, pedestrians, traffic lights, and signs in these images, despite challenges such as noise and poor textures from fog. However, detecting objects in heavy fog continues to be a major challenge for computer models and human observers. This issue is particularly important for autonomous vehicles, where detecting objects accurately in real time is essential for safe navigation [16], [73]. The proposed model aims to enhance detection reliability and precision in real-world road images to tackle these challenges in foggy conditions. Based on these data collection methods, we provided a dataset of foggy and good street images to train and evaluate our proposed model.

Improving object detection accuracy in road images, especially in fog, is the main goal of the proposed model. Detecting objects successfully on roads in foggy conditions could greatly improve the control of autonomous systems. It could also reduce accidents, improve traffic management, increase street safety, and ultimately save lives [20]. Addressing issues with image clarity and brightness and using methods in machine learning and deep learning are crucial to effective object detection and learning [17, 18, 19]. However, even with these possible benefits, earlier studies have not fully used raw image data, which makes models less effective in real foggy conditions. Even though having this dataset, detecting objects in adverse weather remains challenging, characterizing the need for a focused research approach. The goals of this research are to (1) create an object detection model that can identify multiple road objects in foggy conditions, (2) train and test the model using raw image datasets, and (3) evaluate its performance at different visibility levels. The study looks at urban and suburban street settings with realistic fog, excluding images that have been

artificially changed or filtered.

While previous studies relied on filtered images to simulate adverse weather, the proposed research leverages raw datasets, allowing the model to learn real-world features directly and improving detection performance in actual foggy conditions.

Training with raw images has an advantage that it lets the model learn features directly from real-world data without using fake filters or enhancements. Filtered images can be helpful for making weather conditions like rain or snow look more realistic, but they often make things look simpler or different from how they really are, which can make it harder for the model to apply what it has learned to real situations. Training on raw datasets, on the other hand, gives the model access to all kinds of visual variations, such as changes in lighting, contrast, occlusion, and small weather effects. This makes it better at finding objects in various situations. This method makes sure that the suggested model can better handle real foggy, rainy, or low-visibility situations, filling in the gaps left by former research that used pre-processed data.

2.3.1 Object Detection Models and Metrics

Several models are used by researchers to detect objects on the street. Some scientists use Fast-RCNNs, which is another popular model for detecting objects on roads. In [55], the authors train and validate a DR-CNN model powered with a deconvolution layer to detect traffic light shapes and positions. They use a pre-trained dataset made up of 165,000 training images (“2015 train”) and 81,000 validation images (“2015 val”).

In DL technology, an iteration is an important factor that consists of multiple steps:

- **Forward Pass:** During an iteration, a set of training data moves through the neural network. Each sample in the batch undergoes a series of calculations defined by the network’s structure. Weights and biases are applied to the input data to generate forecasts.
- **Loss Computation:** After the forward pass, the predictions from the neural network are compared to the actual targets, or ground truth. This comparison uses a loss function that measures how well the network did on that specific batch of data.
- **Backward Pass (backpropagation):** The calculated loss is then used to adjust the model’s internal parameters, which include weights and biases, in a process called backpropagation. The goal is to minimize the loss by adjusting these parameters. This step involves computing the gradients of the loss function for each parameter in the network.
- **Optimizer Update:** The optimizer algorithm, including stochastic gradient descent (SGD), Adam, or Root Mean Squared Propagation (RMSprop), uses gradients to update the model’s weights and biases. The optimizer decides the direction and size of the updates made to the parameters in order to lower the loss.
- **Repeating for Multiple Iterations:** This process is repeated multiple times, incorporating new batches of data each time until the whole dataset is used (completion of an epoch) or until a predetermined number of iterations is reached. The steps involved are forward pass, loss computation, backward pass and optimizer update.

In [20], traffic lights are detected using 5000 iterations from a Fast-RCNN model. In [63], the Fast-RCNN model and the Regional Proposal Network feature detect pedestrians through 120,000 iterations.

In [60], the authors combine the Fast-RCNN model with Online Hard Example Mining (OHEM) to detect objects like traffic signs on roads in real-time. This method offers several advantages. OHEM helps focus the training process on difficult examples, which are often key for the model's improvement. By prioritizing these challenging cases, the model can learn from the most informative instances, which can enhance its overall performance.

In paper [61], deep learning technology and YOLOv3 are used to recognize the status of traffic lights. The model accurately identifies traffic light statuses such as red, green, and yellow. Paper [66] modifies a model based on YOLOv7 to detect pedestrians in foggy street images using a de-fogging filter. In [68], the authors enhance text detection on roads. They specifically focus on extracting text from images of Malaysian streets by employing a histogram of oriented gradients and an Artificial Neural Network (ANN). First, they perform histogram equalization on the image frame to boost contrast. Next, they extract the blue channel from the image and remove noise (using salt and pepper noise removal and median filtering). Finally, they convert the extracted text into speech for use by disabled individuals or smart devices like smart cars.

In article [62] detects traffic lights using Global Positioning System (GPS) technology, along with sensors and a ground-truth representation method. Then it transforms the images captured in the previous step into grayscale. Texts are extracted from these gray images of traffic signs, resulting in higher accuracy and fewer errors.

So far, the papers presented have focused on identifying a single type of object on the road. A study has been conducted to detect three classes—cars, traffic lights, and pedestrians—using the same model. However, the authors rely on unrealistic weather conditions in their images. They alter the images using foggy weather simulation, filters, or other algorithms [58]. For this project, the Roboflow <https://roboflow.com/> (accessed on 11 September 2024) is used for image labeling and YOLOv5 to identify objects on the streets. The results for precision and recall metrics for the three classes are not satisfactory [48, 17]. In our study, we will explore the same three classes along with street signs as a fourth class.

Others study different objects to detect, such as bikes. They look at clear images to identify cars, pedestrians, and bikes using a Fisheye8K camera and four versions of YOLO: YOLOv5, YOLOR, YOLOv7, and YOLOv8.x. They conduct this analysis at four times of the day: morning (sunrise), afternoon (sunny), evening (sunset), and night [59]. Unlike the study in [59], we use a dataset of dashboard camera images affected by fog. Our results surpass those in [59] in terms of mAP50 and precision. The study in [52] uses YOLOv3 with a pre-trained COCO dataset in normal weather but relies on high-resolution images. Unfortunately, the results for all three classes are not good.

In [55], the authors use deep learning with hardware like the Raspberry Pi to detect objects on roads in real time. They use wearable items, such as a hat. They analyze images taken before and after fog, rain, and snow to improve object detection by comparing older and newer images from

the same location.

Some metrics are used in the mentioned models, such as epoch, batch size, augmentation, optimizer, and learning rate. An epoch is one complete run through the training dataset. Training for more epochs lets the model learn more from the data, but too many epochs may lead to overfitting. A larger batch size may speed up computing but can result in less generalization. This parameter scales the size of the updates to the model weights. A high learning rate may cause the model to overshoot the best solution, while a low learning rate may slow down convergence [74]. Augmentation involves applying random changes to the training data, like rotations, flips, zooms, or shifts. It helps the model generalize better by providing variations of the same data. Augmentation hyperparameters include the type and intensity of the transformations applied. Optimizers like Adam, SGD (stochastic gradient descent), and Root Mean Squared Propagation (RMSprop) are used to update the model parameters during training. Each optimizer has its own set of hyperparameters, such as momentum and decay rates.

In the proposed model, we optimize the epoch, batch size, and brightness augmentation hyperparameters, as well as other default hyperparameters. The optimizers we use are the same as those in YOLOv8 auto mode, and the learning rate stays constant. We show that this model quickly achieves detection results, and its accuracy surpasses that of the other models mentioned. Some researchers study the Fast-RCNN model for detecting multi-class objects, where feature extraction and classification are challenging. However, their studies indicate high computation time costs. In contrast, we have employed the latest YOLOv8 version, which is about seven times faster than the Fast-RCNN model. Additionally, we train our model using natural and real images without altering them with algorithms or filters.

2.3.2 Object Detection Background

Detecting objects on roads occurs through two types of methods: active and passive methods. The active method uses laser-equipped sensors (Lidar), while the passive method uses cameras [16]. Models use the images they obtain to train procedures for both methods.

Object detection has two main stages. The first stage compares images by training detection based on the entire images. The second stage identifies objects from specific parts of an image and trains detection based on region proposals. Factors like lights, noise, and low-quality textures can negatively impact the images used by a model.

Recognizing objects is crucial for autonomous vehicles and can help save lives. Approximately 700 people die in road accidents, a number that could decrease if we use modern technologies like machine learning and deep learning to detect objects on the streets and prevent collisions [16].

One important aspect of image annotation is ensuring that the images are clear. It is better to use colorful images instead of black and white ones to train a model and detect objects. If a specific dataset is unavailable, Google Street View can provide specific images collected from the streets [56].

Finally, drones and satellites can assist autonomous vehicles during their trips. Some studies

use drones to identify vehicles and pedestrians through images [46, 53]. Other studies utilize satellite images to enhance object detection [69] .

2.4 Approach

In ML and DL technologies, the dataset and the hardware are the most important factors, especially the Graphic Processing Unit (GPU). We used about 4,000 foggy road images from Zurich, which we downloaded from the <https://people.ee.ethz.ch> website, as the dataset to create the proposed detection model. We analyzed these images using Google Colab[47], a powerful computer service with high resources. The hardware resources provided to the proposed model by Google Colab include a GPU A100 with 40 GB of memory, 18 GB of memory, 2 CPUs, and an 80 GB hard disk. Another key factor is that the images need to be bounding boxed or annotated by a tool or software, and all objects in the images must be marked [15] . Therefore, we use the Roboflow annotation tools online.

At the beginning , we used Roboflow with the pre-trained COCO model, but it struggled to recognize objects in images. For example, COCO could not clearly detect pedestrians or street signs. With Roboflow, we will annotate all desired objects with a frame around them. The colorful frames will indicate the positions of the objects, such as cars, pedestrians, traffic lights, and traffic signs, in road images. There are challenges in marking the objects in the dataset. It is important to mark them correctly. The frames around the objects will help detect them more effectively in the training process. Each object we annotate in the images will be treated as a class. In this study, we identified four classes: cars, pedestrians, traffic lights, and traffic signs.

This research focuses on object detection in foggy weather conditions using real foggy images and the YOLOv8 model. Unlike previous studies that rely on digitally altered images, this approach uses actual foggy pictures and introduces a novel preprocessing step of reducing the image size and quality before training. This method not only addresses the challenge of working with low-quality datasets but also enhances the model's robustness. This study explores different variants of YOLOv8 (small and large) and investigates the impact of brightness augmentation on detection accuracy.

The research brings several new ideas to the field. It includes a detailed look at hyperparameter optimization for foggy conditions, a systematic assessment of brightness changes (ranging from **-25% to +25%**), and a comparative study of different YOLOv8 model versions. These areas tackle the challenges of object detection in tough weather and offer useful insights for future studies and real-world use. The paper also discusses quantitative results and qualitative examples. This shows how effective the method is and highlights its possible limitations, especially in situations with unclear images.

For this reason, we decided to create a model that can identify objects in street images. First, we reduced the quality and size of the raw images by 25% and 50%, respectively. This resulted in high accuracy and better outcomes, even with lower image quality. Next, we examined the images for the presence of objects. Using Roboflow software <https://roboflow.com/> (accessed on 11 September 2024), we boxed any resized images containing objects and placed them in our training folder for the model. In the third phase, we used YOLOv8 to train the model for object detection in

the resized images. We also looked into four different sub-methods to discover the best approach for detecting objects in foggy images. Since YOLOv8 has specific settings, we chose four categories based on these settings in the four sub-methods. As a result, we introduced a general methodology, which is shown in Figure 2.1. Some articles mention that experts have tried to modify certain hidden layers in the YOLOv8 network to improve their model's accuracy for object detection in image datasets. However, we did not use a customized YOLOv8 model with different hidden layers, as this requires too much time for detection. Instead, we relied on the default hidden layers of YOLOv8 and adjusted the hyperparameters.

Before presenting the experimental results, it is important to outline how each step of the methodology contributes to the final detection performance. The following section provides a structured explanation of the five steps shown in the flowchart, which will later help interpret the results obtained from the four training configurations. This ensures that the experiments, results, and interpretations remain logically connected.

Based on Figure 2.1, the detection methodology consists of five steps that are described in detail in the following :

- Step 1 (Decreasing the image quality and size) : The quality and size of all images are reduced to 25% and 50%, respectively.
- Step 2 (Identifying and marking objects in images) : Some objects in the images, such as cars, pedestrians, traffic lights, and signs, are identified. If specific objects are found in an image, they are marked with a tool like Labelbox or Roboflow, and the image is saved in a training folder. If no object is present in an image, it is saved directly into the training folder.
- Step 3 (Image extraction for model validation): Twenty percent of the total image dataset is moved to a validation folder.
- Step 4 (Training) : Four different methods with different configurations are used to train the model. These methods vary by different epochs and batch sizes, and they also include options with or without augmentation. The goal is to improve the model's ability to detect objects in the images.
- Step 5 (Outcomes) : The results for each approach can be seen in the figures.

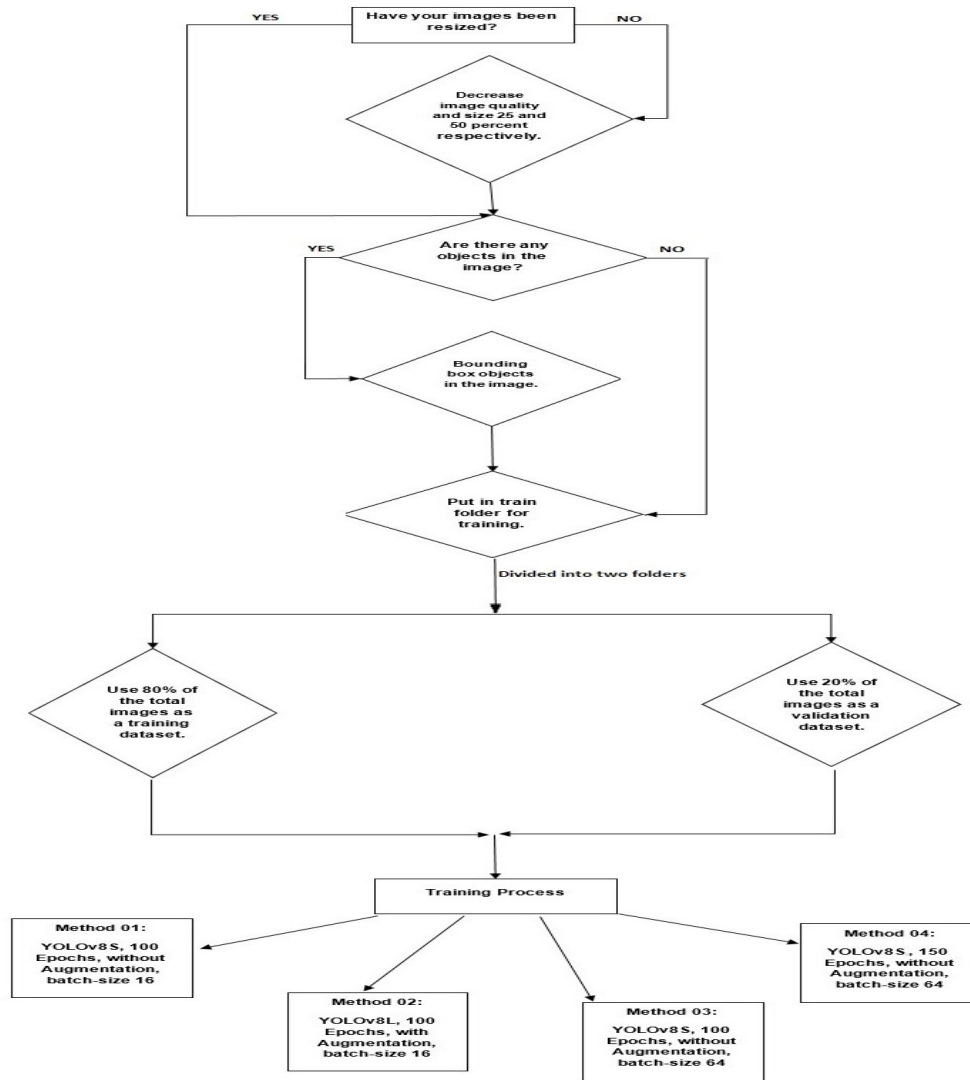


Figure 2.1: Methodology Flowchart

The four different approaches were used in the proposed methods shown in Figure 2.1. First, we manually reduced the raw image size and quality. This is not typical in standard object detection methods. Additionally, the proposed approach targets challenging environmental conditions, such as foggy weather. We used real foggy images instead of applying filters and modules to raw images to create the fog effect in the object detection area. Focusing on various weather conditions in the images will yield valuable results in object detection projects. Using images from different weather conditions is an innovative aspect of our proposed study. The analysis and reporting of results rely on various methods, including YOLOv8S and YOLOv8L. This can lead to a comparison of performance across different metrics. It is effective and efficient regarding time spent and accuracy. The study will positively impact object detection in poor weather conditions. It could also be beneficial for autonomous vehicles in the future. Autonomous vehicles and smart city infrastructure can apply our study results, as object detection in various weather conditions is essential. Other innovative aspects of the proposed study aim to improve safety and efficiency while addressing real-world challenges. This research seeks to fill a gap in the object detection field for both adverse and favorable weather conditions.

In Figures 2.2 and 2.3, you can see a clear difference in the quality of the traffic light above the

car. Figure 2.3 shows the second image taken from Step 1, which has lower resolution. However, this image still helps improve the object detection rate by YOLOv8.



Figure 2.2: One sample of image in foggy weather conditions characterized by 1920×1080 pixels stored into training folder.



Figure 2.3: Image obtained after Step 1 characterized by 940×540 pixels.

After reducing the image quality and size, we manually label objects in Step 2. The objects are labeled from 0 to 3 as cars, pedestrians (ped), street signs, and traffic lights, respectively. Next, in Step 3, we use the images in the training dataset to begin the training step. The images that move to the validation dataset (folder) are used to validate the proposed model and are not used for unseen images. A sample of the outcome image that was moved to the validation dataset (folder) is shown in Figure 2.4, where the objects are marked in foggy weather conditions.

In the following section, we show two images. One is an example outcome from the validation dataset (folder) (see Figure 2.4). The other is an image predicted by YOLOv8 during training Step

4 (see Figure 2.5).

In Figure 2.5, the objects are labeled by the proposed model with prediction rates ranging from 0 to 1. This shows that the model performs well during the prediction phase, based on the raw images with a threshold of 0.25 (25% precision). The objects with a value above the 0.25 threshold are detected and marked. All required objects are highlighted with colorful boxes. Vehicles are marked with red boxes, traffic lights with yellow, pedestrians (crosswalk) with pink, and street signs with orange in Figures 2.4 and 2.5.

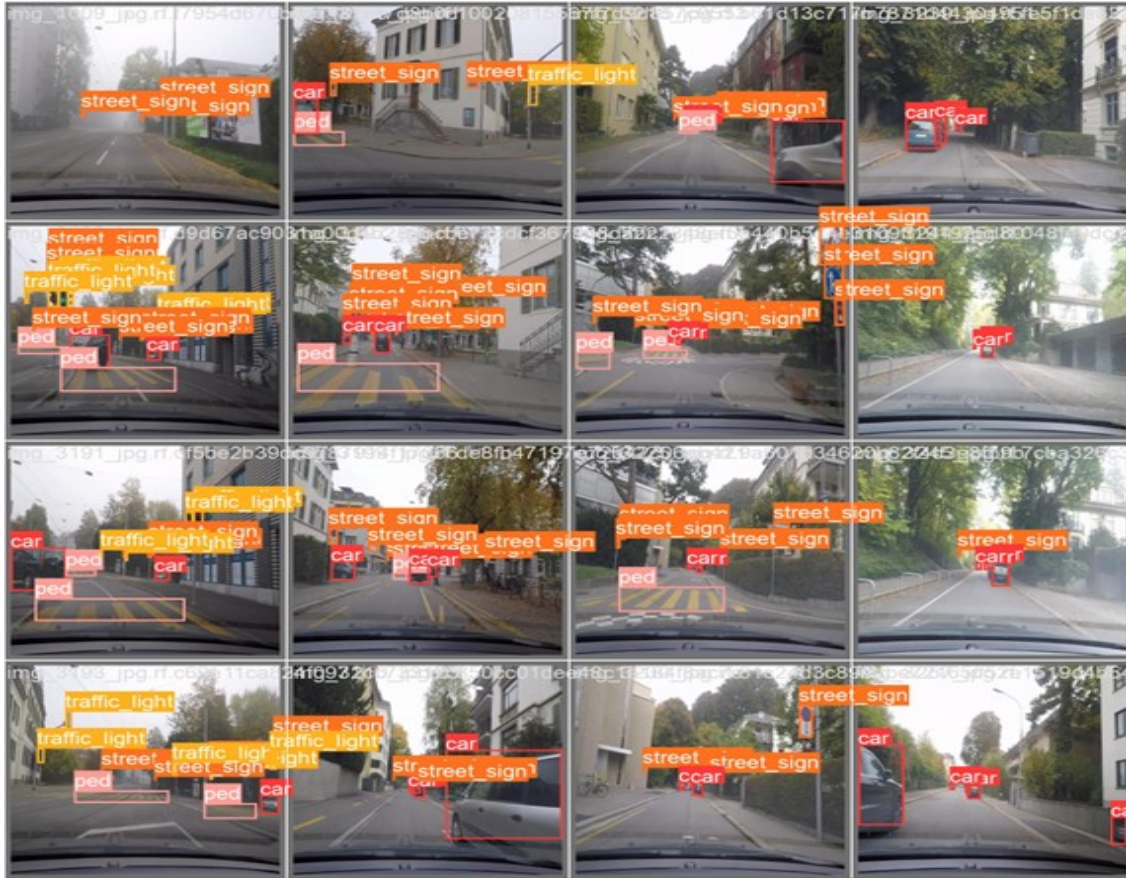


Figure 2.4: One sample outcome from the validation dataset (folder).

Step 4 takes place on a powerful computer using Google Colab and its GPU. The proposed model is trained with the annotated images obtained from Step 2. During the training step, we use four different methods to improve the object detection results. We consider four configurations of some hyperparameters, including augmentation, batch size, and epoch. Additionally, we employ two models of YOLOv8 in this method and compare their results with other versions of YOLO in the following section. YOLOv8S (small) has 11.2 (Millions) M parameters, while YOLOv8L (large) has 43.7 M parameters. Although YOLOv8L (large) is slower, it is more accurate. For this reason, we use both the small and large models in the detection strategy. The following approaches are adopted:

- Method 1: 100 epochs without brightness augmentation, batch size 16 and YOLOv8S (small);
- Method 2: 100 epochs with brightness augmentation, batch size 16 and YOLOv8L (large);
- Method 3: 100 epochs, batch size 64, without brightness augmentation and YOLOv8S (small);

- Method 4: 150 epochs, batch size 64, without brightness augmentation and YOLOv8S (small).



Figure 2.5: One sample of prediction data.

2.4.1 Accuracy Analysis

We used features like true positive, true negative, false positive, and false negative to examine the accuracy and performance of the proposed model. A positive occurs when we identify and detect the required objects in images using a bounding box. A negative is when there is no detection in the images. True and false describe whether the detection of the desired objects is correct. True negative happens when there are no desired objects in the images, so they are not detected. False positive occurs when objects are mistakenly found as desired objects. When a model fails to recognize desired items, we call this a false negative. This information helps us evaluate how well the dataset has been trained [64, 67, 68]. We consider the Precision, Recall, and Precision-Recall metrics to analyze the accuracy of the proposed model:

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}} \quad (2.1)$$

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}} \quad (2.2)$$

$$\text{Accuracy} = \frac{\text{True Positive} + \text{True Negative}}{\text{True Positive} + \text{True Negative} + \text{False Positive} + \text{False Negative}} \quad (2.3)$$

Precision refers to how many positive samples are correctly identified compared to the total number of samples identified as positive, regardless of whether they are correct or not. A proposed model performs well and achieves high accuracy if it has many true positives or only a few false positives. Recall is the rate calculated between the number of correctly detected positive samples and the total number of both positive and negative samples. Recall measures how well the model detects positive samples [51, 16]. Loss indicates how poorly a proposed model predicts the detection of objects or classes. If loss continually decreases and approaches zero, we will have a good prediction model [47]. Along with precision, recall, and other metrics, we can also use mAP50 and mAP50-95 metrics to evaluate an object detection model. The mAP50 (mean Average Precision at 50%) measures the average precision of detection across different classes when the intersection over union (IoU) threshold is set to 0.5. The mAP50-95 (mean Average Precision from 50 percent to 95 percent) calculates the average precision over a range of IoU thresholds from 0.5 to 0.95 with a step size of 0.05. This metric provides a broader evaluation of model performance across various IoU thresholds. It is especially useful for understanding how well the model performs at different levels of localization accuracy. In this research, we used the precision, recall, precision-recall, and mAP50 metrics to evaluate model accuracy.

When working with images, there can be unwanted elements known as noise. In DL, we adjust some settings, such as how the data are modified or how many times the model learns (epochs), to improve our model and reduce errors when checking against new data. For example, we can change images in different ways, like rotating them or adjusting brightness, to help our model recognize objects better. In our model, we used brightness augmentation, which increased the number of images by about double. This helps YOLOv8 spot objects in images. Making these changes can also save time during the learning step from the data. Additionally, we resize all the images to make it easier to spot objects. Then, we split the images into two parts (folders): 80 percent is used to train the model, and the remaining 20 percent is used to test how well the model detects objects.

Batch size is the number of samples processed before updating a proposed model. Augmentation is a method to increase a database using techniques like reversing, cropping photos, and adjusting brightness in images. A suggested model can be trained if we have a clear image and enough brightness. Objects in photographs taken on streets can be identified. Brightness and adequate light in images are two important factors in machine learning and deep learning. Object size is also an important measure for training a proposed model, and the accuracy of object identification can positively impact road traffic management systems [17].

The methods have been selected in this research to directly address the challenges of object detection in real foggy environments. The Real fog images were used instead of usage of foggy generated images by filter that is why the fog often fails to capture natural , depth variation, and contrast loss, which leads to models that perform poorly in real-world scenarios. YOLOv8 was selected as the detection architecture due to its proven balance between speed and accuracy, its improved backbone and neck design, and its strong performance in adverse visual conditions. The reduction in image quality and size was intentionally introduced as a robustness-oriented preprocessing step to simulate degraded visibility and improve the model's generalization. Manual

annotation in Roboflow was necessary to ensure high-quality labels, which are critical for training object detectors under visually complex conditions such as fog. The four training configurations were designed to systematically evaluate the effects of brightness augmentation, batch size, and epoch count on detection accuracy, given that brightness variation is a major factor influencing object visibility in foggy weather. Together, these methodological choices ensure that the proposed pipeline aligns with the research aim of building a reliable, generalizable, fog-resistant object detection model.

2.5 Findings and Implications

To test the proposed method, we use a 9-gigabyte dataset of 3,806 images captured by a dashboard camera in foggy weather in Zurich. The training dataset includes 3,045 images, and the validation dataset consists of 761 images, which is 20% of the total. The image resolution is 1920 by 1080 pixels, and the average file size is about 2 megabytes. After the first three steps, we carry out the training step. We then evaluate each proposed method based on precision, recall, precision-recall, and final result items.

In method 1, we use 100 epochs and a batch size of 16, without any augmentation technique, and YOLOv8S (small). The training process takes 4 hours. Overall, in method 1, the precision metric is 94.5% for all classes, as shown in Figure 2.6. The precision metric versus epochs is shown in Figure 2.7. Here, the precision metric clearly improves from a low point of 65% in the third epoch to a peak of about 86%. From epoch 1 to epoch 20, the precision metric rises slightly from 78% to 82%. Between epochs 20 and 80, the metric increases from 82% to 84%. After that, the precision metric fluctuates around 86%.

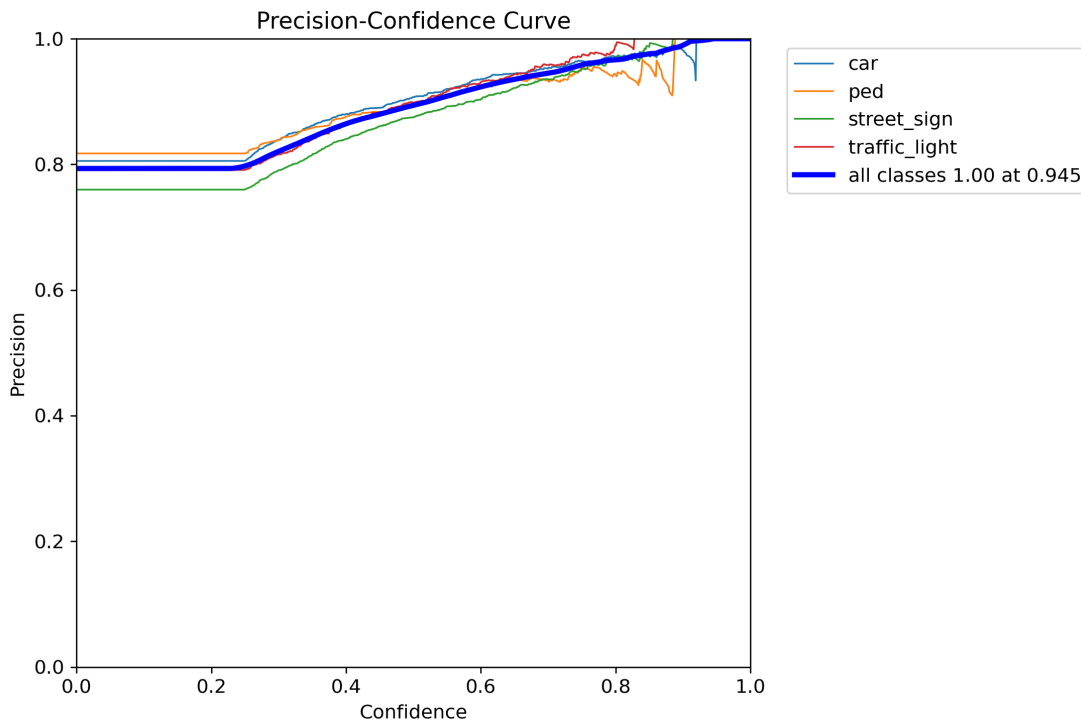


Figure 2.6: Precision–confidence curve for *method 1* (100 epochs, batch size 16, without augmentation and YOLOv8S).

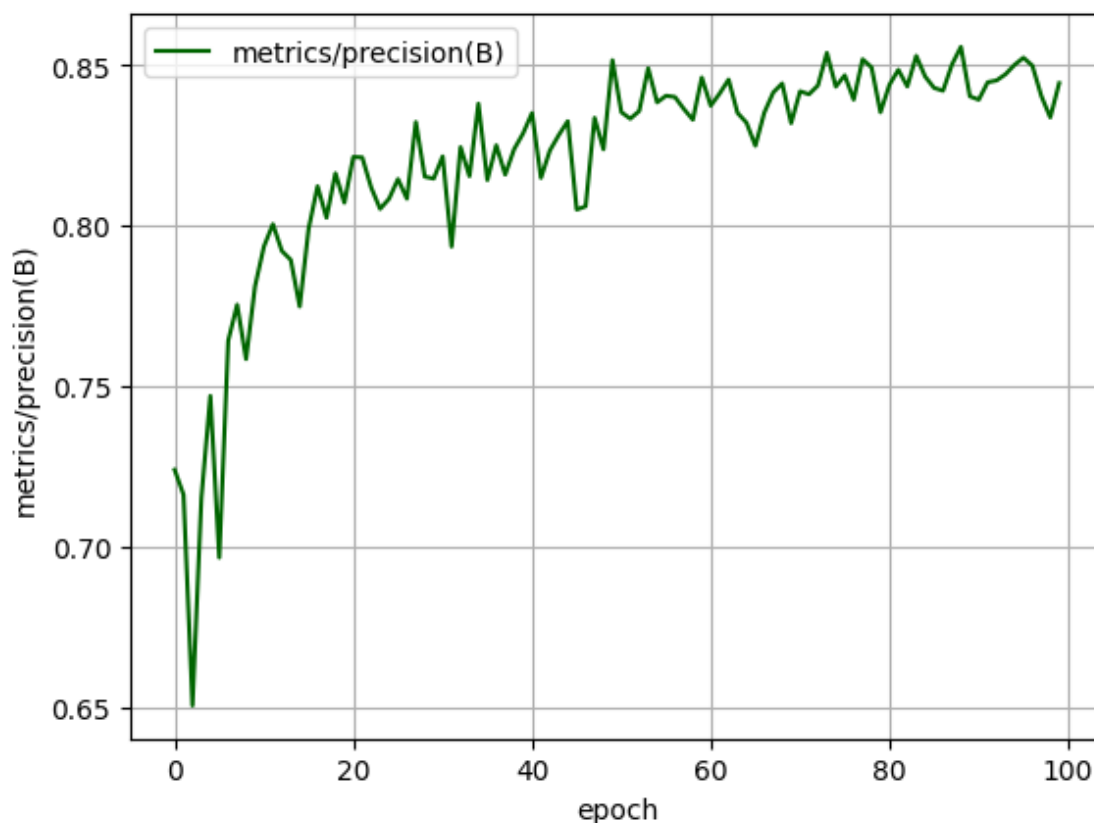


Figure 2.7: Precision vs. epoch for *method 1*.

Recall, which acts as a second accuracy measure, improves during method 1 and reaches about 81% for all classes. This is shown in Figure 2.8. Figure 2.9 displays the recall metric against epochs. At the start, in epoch 3, the recall feature sits at roughly 50%. From epoch 5 to 60, recall rises significantly from 50% to nearly 76%. Between epochs 60 and 100, there is a small increase in the recall feature from about 76% to around 78%, as 0.

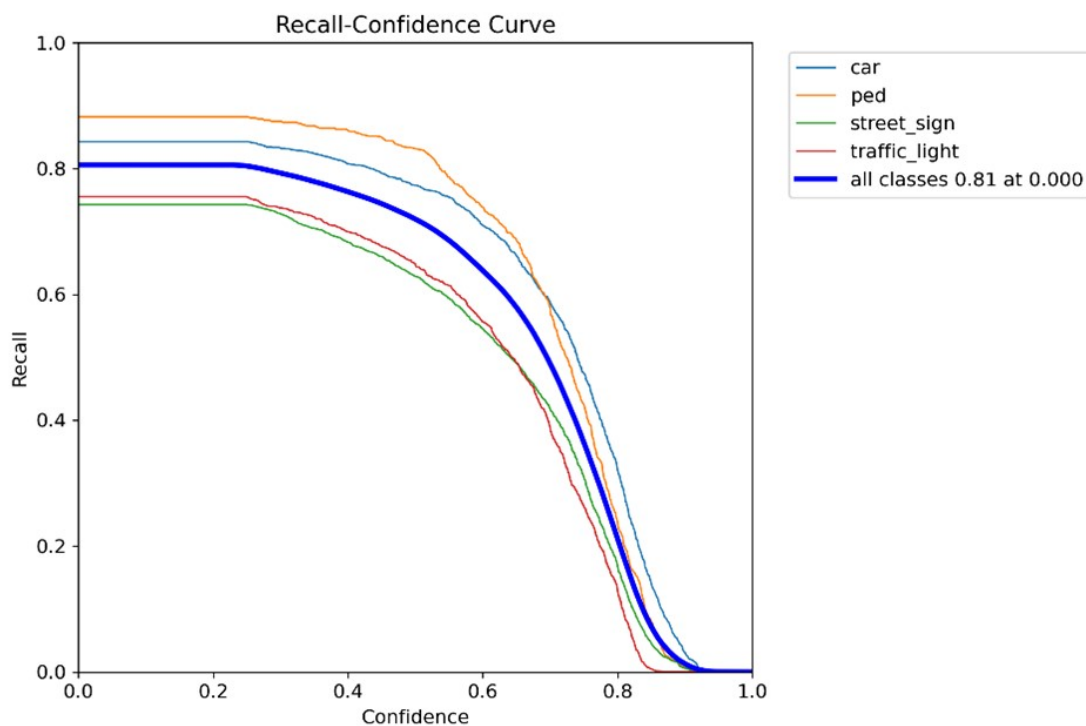


Figure 2.8: Recall-confidence curve for *method 1* (100 epochs, batch size 16, without augmentation and YOLOv8S).

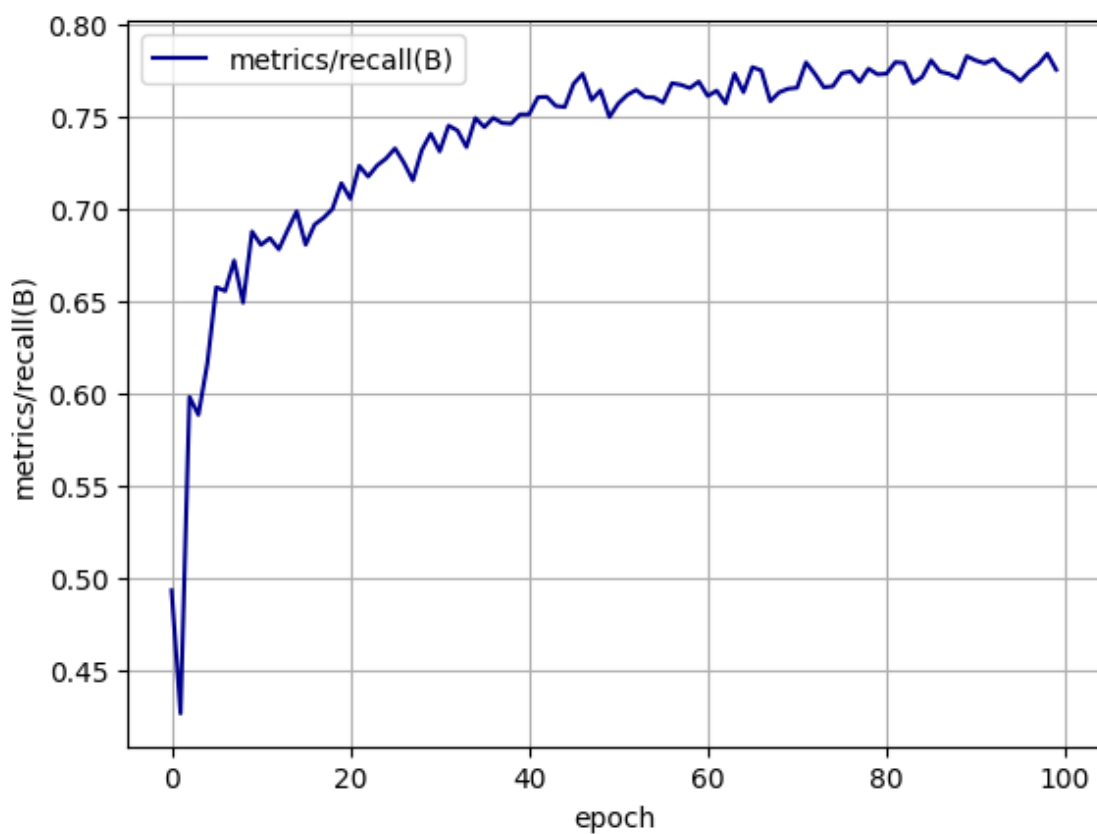


Figure 2.9: Recall vs. epoch for *method 1*.

Figure 2.10 shows the third metric, which is the precision and recall curve. The values for cars, pedestrians, street signs, and traffic lights are 87.0%, 87.9%, 78.8%, and 81.2%, respectively. On average, it was 83.7% for all classes.

Finally, Figure 2.11 shows the precision, recall, and mAP50 metrics for method 1. At the same time, all loss metrics decrease significantly.

The method 1 outcome is shown in Table 2.1. The following abbreviations will be used in the future: EP = number of epochs, BS = batch size, Aug = augmentation, and YOLOv8s = YOLOv8 (small).

Table 2.1: Prediction rate for detecting objects in *method 1*.

Method	% Adjust column separation		% Adjust row separation	
	Car %	Pedestrian %	Street Sign %	Traffic Light %
100 EP, BS 16, NO AUG, YOLOv8S	87.0	87.9	78.8	81.2

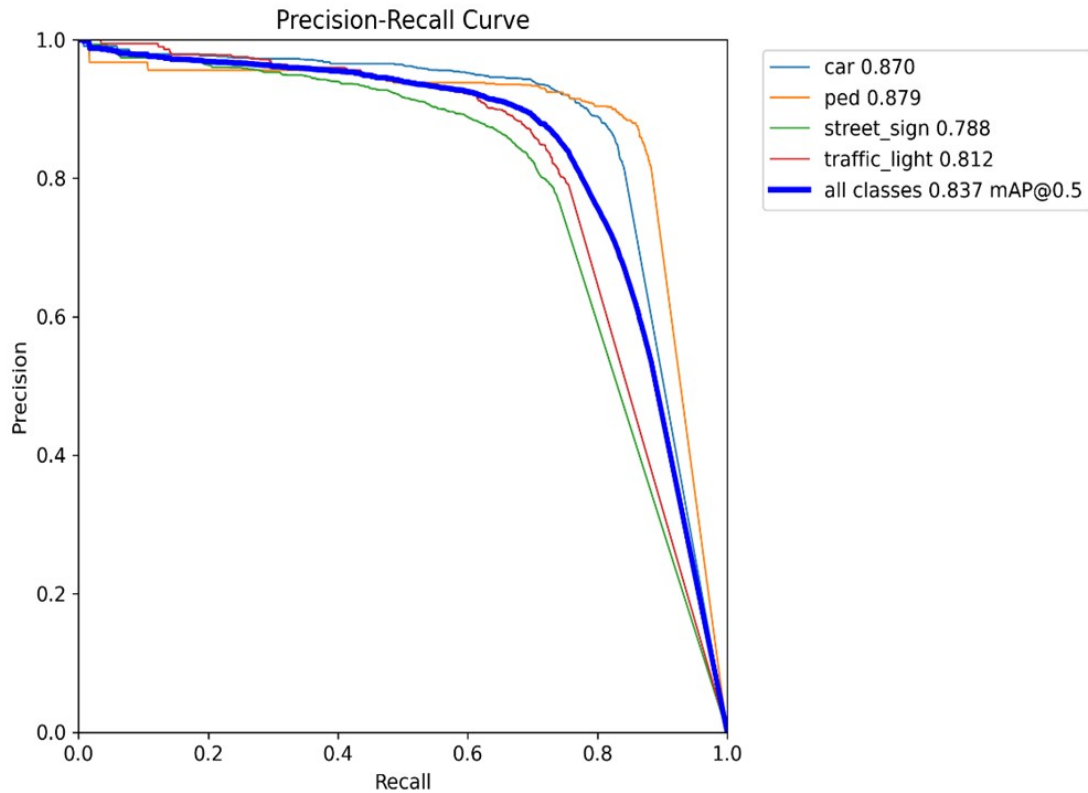


Figure 2.10: Precision–recall curve for *method 1* (100 epochs, batch size 16, without augmentation and YOLOv8S).

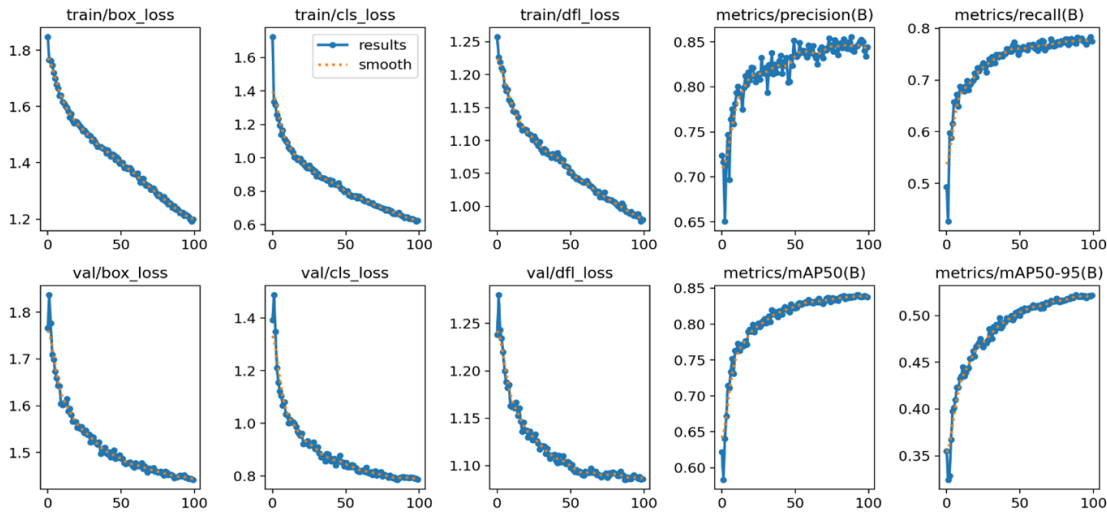


Figure 2.11: Final result for *method 1* (100 epochs, batch size 16, without augmentation and YOLOv8S).

In method 2, we use 100 epochs and a batch size of 16, similar to method 1. However, we apply the augmentation technique in YOLOv8L (large), while method 1 uses YOLOv8S (small). YOLOv8L has more layers compared to YOLOv8S. The precision metric is 94.5%, which is the same as in method 1, and it is shown in Figure 2.12. In terms of precision versus epoch, at epoch 3, the precision drops to about 74%. From epoch 3 to 20, the trend rises to about 83%. Between epochs 20 and 60, the graph increases slightly from 84% to 85%. After that, the precision metric fluctuates around 86%. Figure 2.13 displays the trend of precision with respect to epochs, specifically at 87 instead of 100, due to Google Colab's time limit which caused us to lose 13 epochs.

Overall, the recall metric, our second accuracy measure, rises to 83% for all classes, as shown in Figure 2.14. The recall metric improves by 2% compared to method 1. Figure 2.15 presents the trend of recall versus epoch. At epoch 3, the recall metric is about 59%, which is its lowest point; from epoch 3 to 40, it rises significantly from 59% to nearly 79%. Between epochs 40 and 87, the trend stabilizes around 83%. The precision-recall metric, our third accuracy measure, increases in the second approach. The car detection rate rises by 1.4%, reaching 88.4%. Pedestrian detection slightly increases by 1.1%, reaching 89%. The detection ratio for street signs goes up by 1.9%. The traffic light detection ratio climbs to 85.1%, while method 1 had a metric of 83.7%. Overall, the object detection ratio for all classes increases to 85.1%, compared to the 83.7% from the first method, as presented in Figure 2.16. This shows that YOLOv8L, having more layers than YOLOv8S, produces better results in the second approach than method 1. The large version of the YOLO model positively impacts the object detection results. Finally, the outcome of the second approach is shown in Figure 2.17. Unfortunately, method 2 encounters an overfitting or underfitting

issue in the validation distribution focal loss metric (val/df1-loss) results.

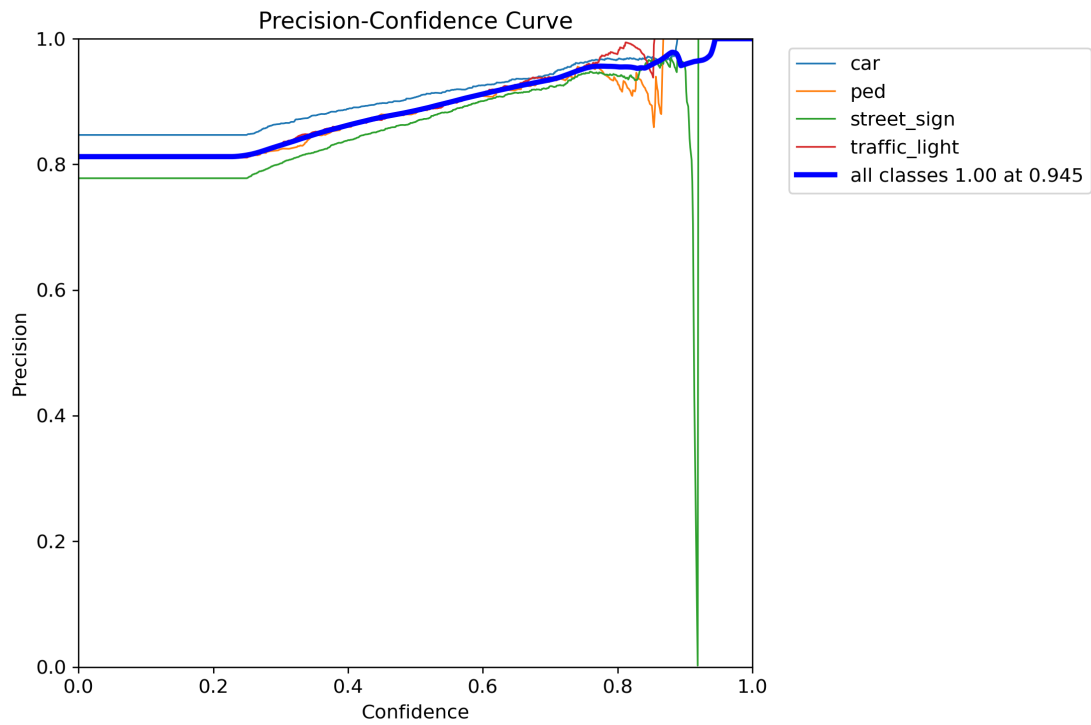


Figure 2.12: Precision–confidence curve for *method 2* (100 epochs, batch size 16 and with augmentation, YOLOv8L).

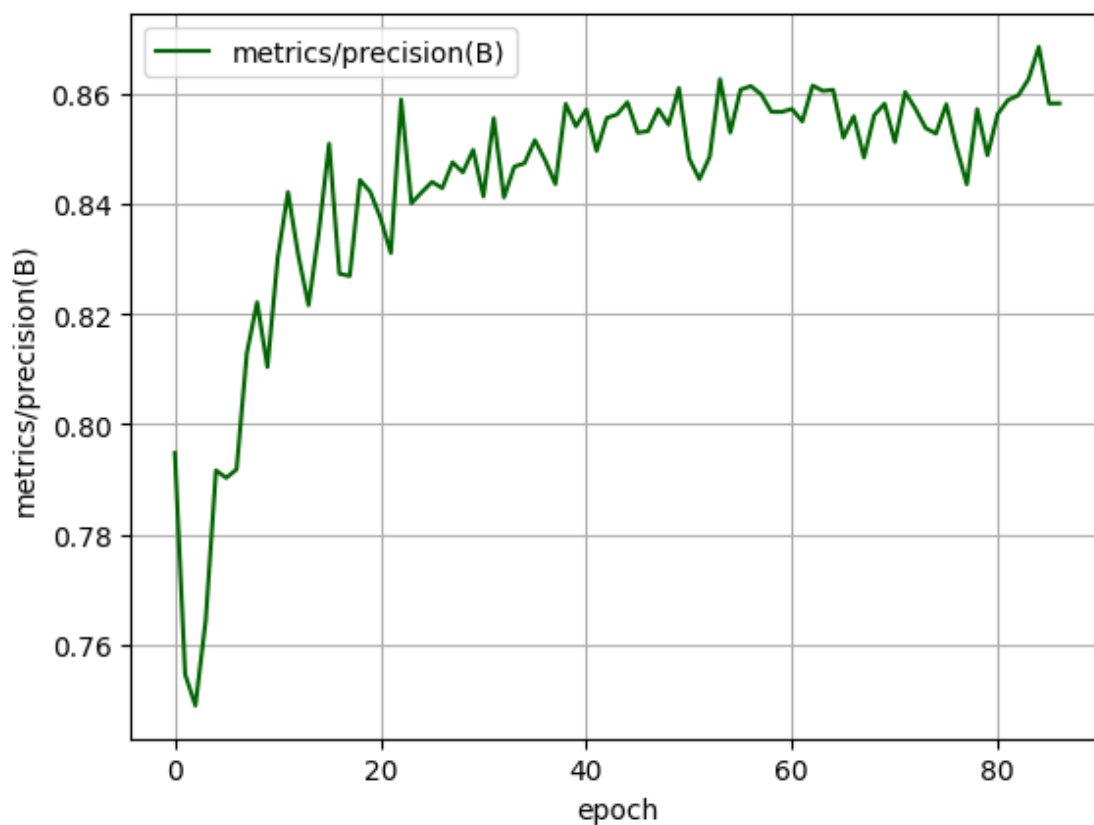


Figure 2.13: Precision vs. epoch for *method 2*.

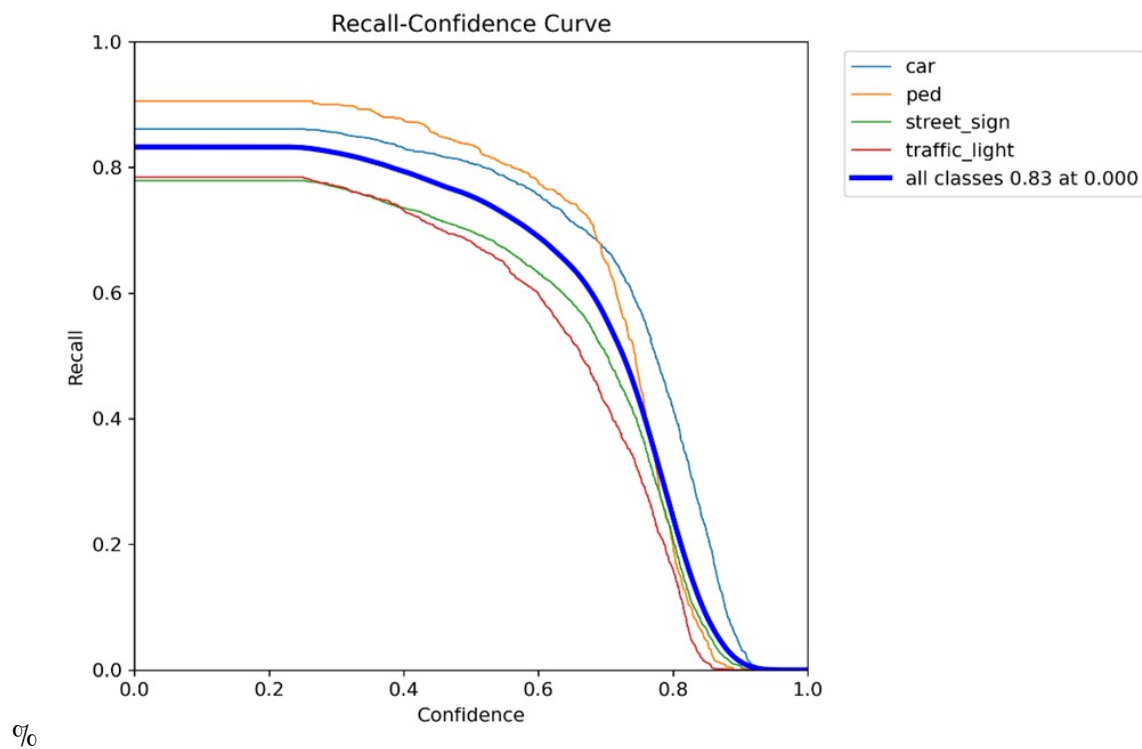


Figure 2.14: Recall–confidence curve for *method 2* (100 epochs, batch size 16 and with augmentation, YOLOv8L).

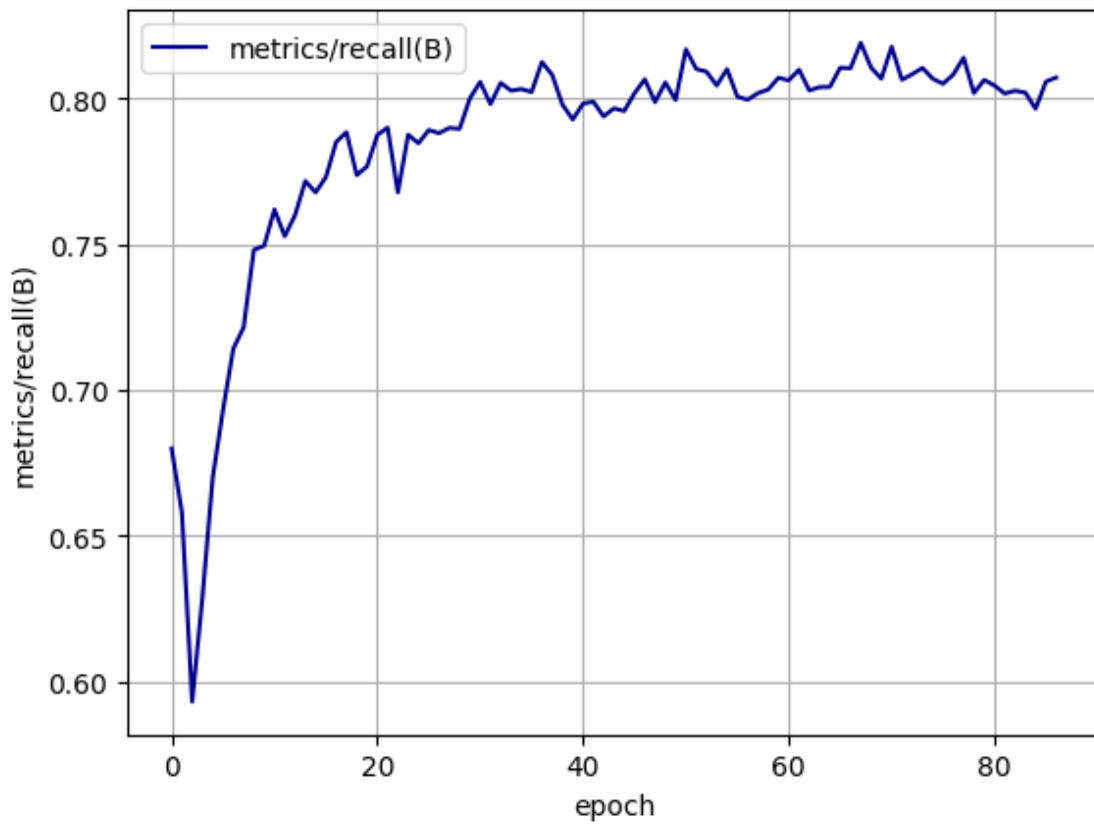


Figure 2.15: Recall vs. epoch for *method 2*.

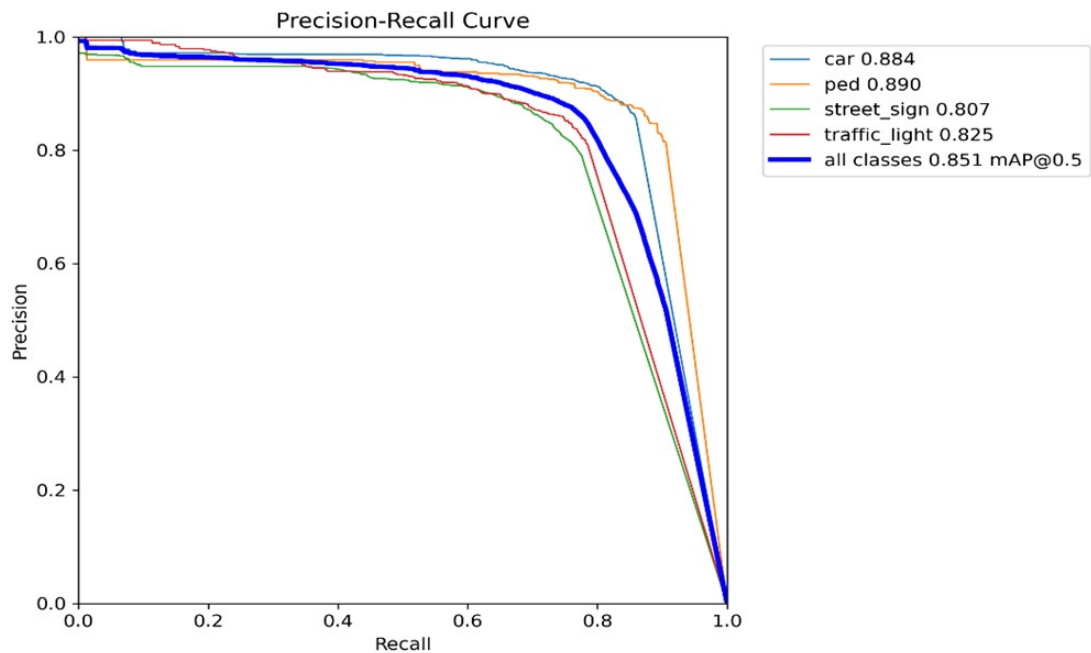


Figure 2.16: Precision–recall curve for *method 2* (100 epochs, batch size 16, with augmentation and YOLOv8L).

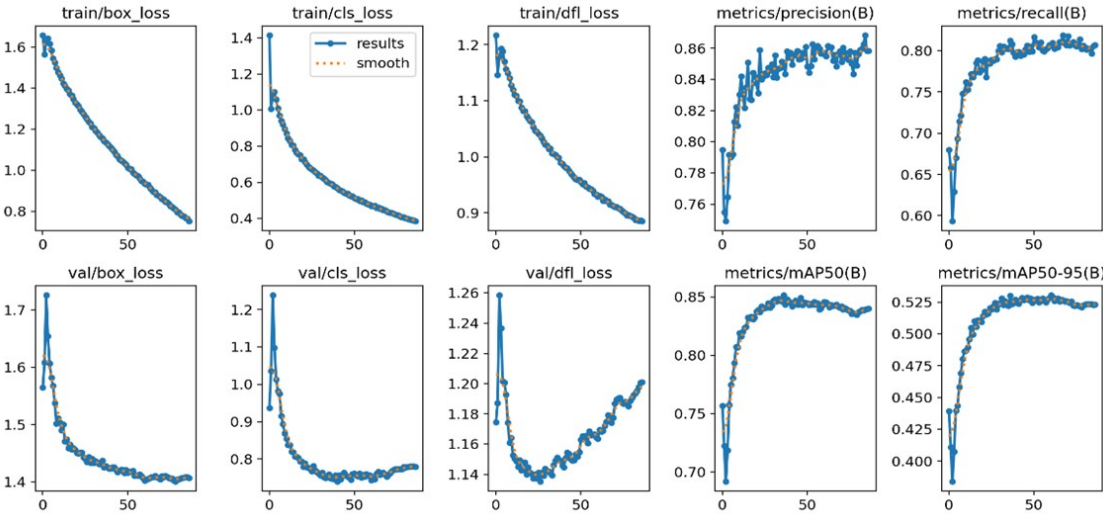


Figure 2.17: Result for *method 2* (100 epochs, batch size 16 and with augmentation, YOLOv8L).

The method 2 outcome is shown in Table 2.2.

% Center the table

Table 2.2: Prediction rate for detecting objects in *method 2*.

% Adjust column separation % Adjust row separation

Method	Car %	Pedestrian %	Street Sign %	Traffic Light %
100 EP, BS 16, AUG, YOLOv8L	88.4	89.0	80.7	82.5

In method 3, we use 100 epochs and a batch size of 64 without augmentation and YOLOv8S (small). It takes about 2.5 hours to train the model. Method 3 benefits from a batch size of 64 instead of 16. The precision confidence metric in the third technique is lower than in the two previous strategies, reaching 93.2%, as shown in Figure 2.18. The metric begins at epoch 3, with a value of around 57%. From epoch 3 to 40, the precision increases by about 20%, reaching roughly 81%. Between epochs 40 and 80, the metric gradually grows to about 84%. Then, between epochs 80 and 100, it fluctuates around 84%. This is illustrated in Figure 2.19.

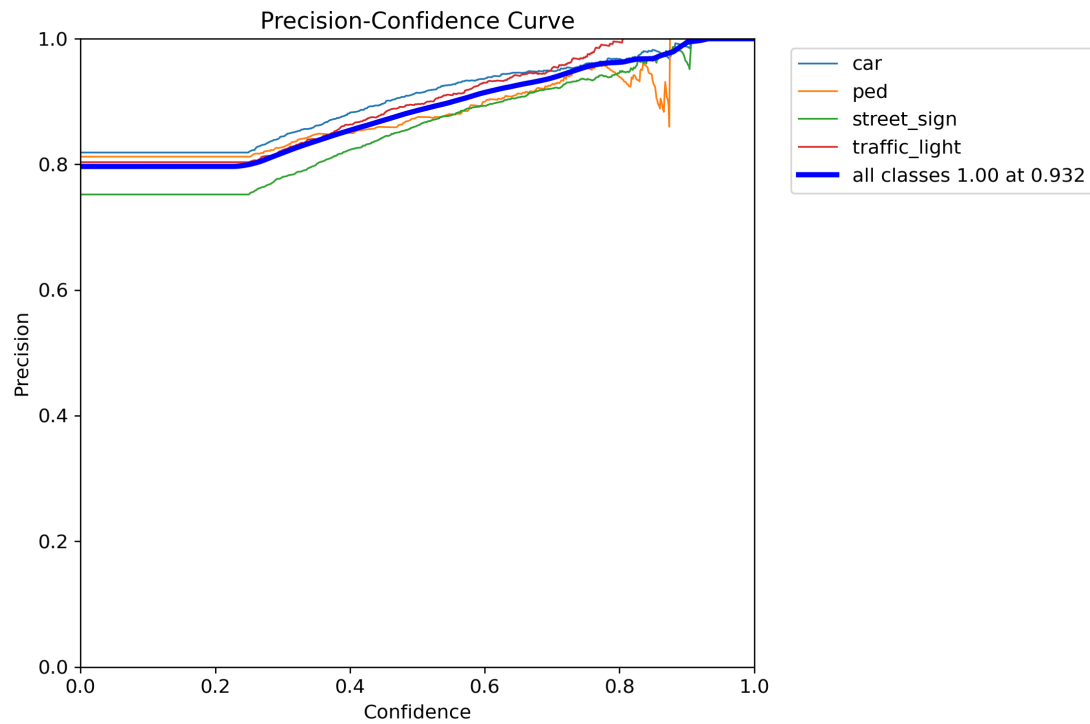


Figure 2.18: Precision–confidence curve for *method 3* (100 epochs, batch size 64 and without augmentation, YOLOv8S).

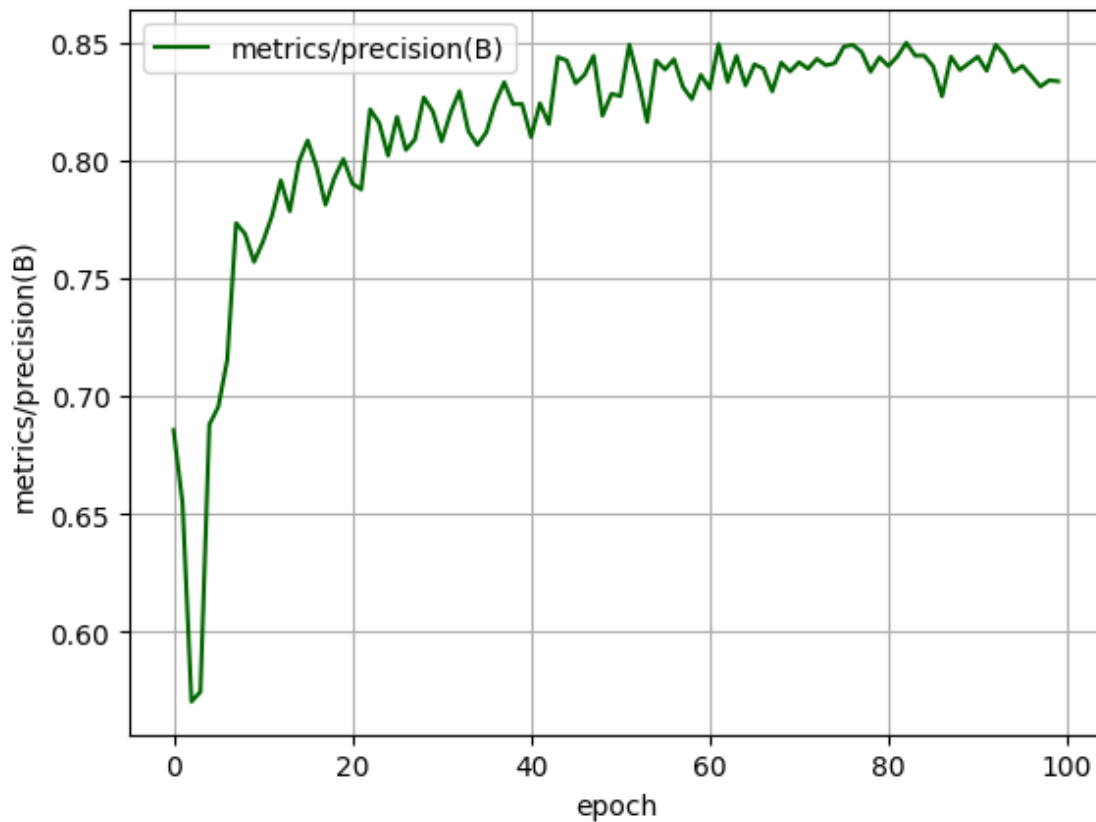


Figure 2.19: Precision vs. epoch for *method 3*.

The recall, confidence metric was 81% for all classes, as shown in Figure 2.20. This trend indicates that the method can detect objects effectively. Figure 2.21 displays the trend of recall versus epoch. The recall metric in epoch 3 is about 32%. From epoch 3 to epoch 40, it increases quickly to about 76%. Between epochs 40 and 60, it improves slightly by 1% and reaches 77%. After that, it fluctuates around 78%. In the precision, recall metric, which serves as a third accuracy metric, the detection rates for cars, pedestrians, street signs, and traffic lights are 86.8%, 88.9%, 78.3%, and 81.8%, respectively. The overall rate in the precision, recall metric is 84.0%, as shown in Figure 2.22. Method 3 improves the precision, recall metric for pedestrians, traffic lights, and overall detection rates compared to method 1. The final results for method 3 are presented in Figure 2.23.

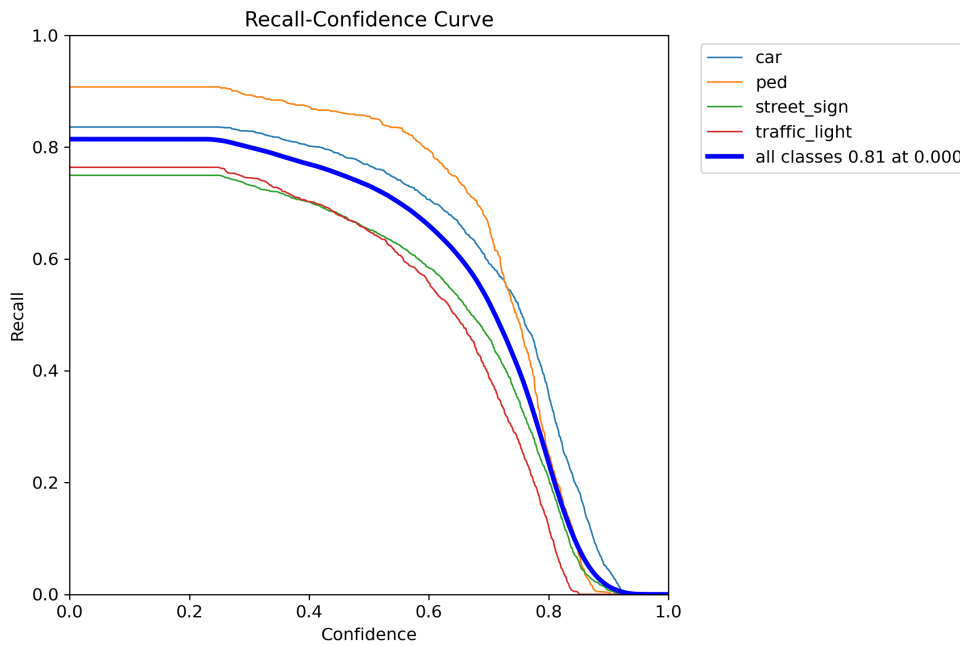


Figure 2.20: Recall–confidence curve for *method 3* (100 epochs, batch size 64 and without augmentation, YOLOv8S).

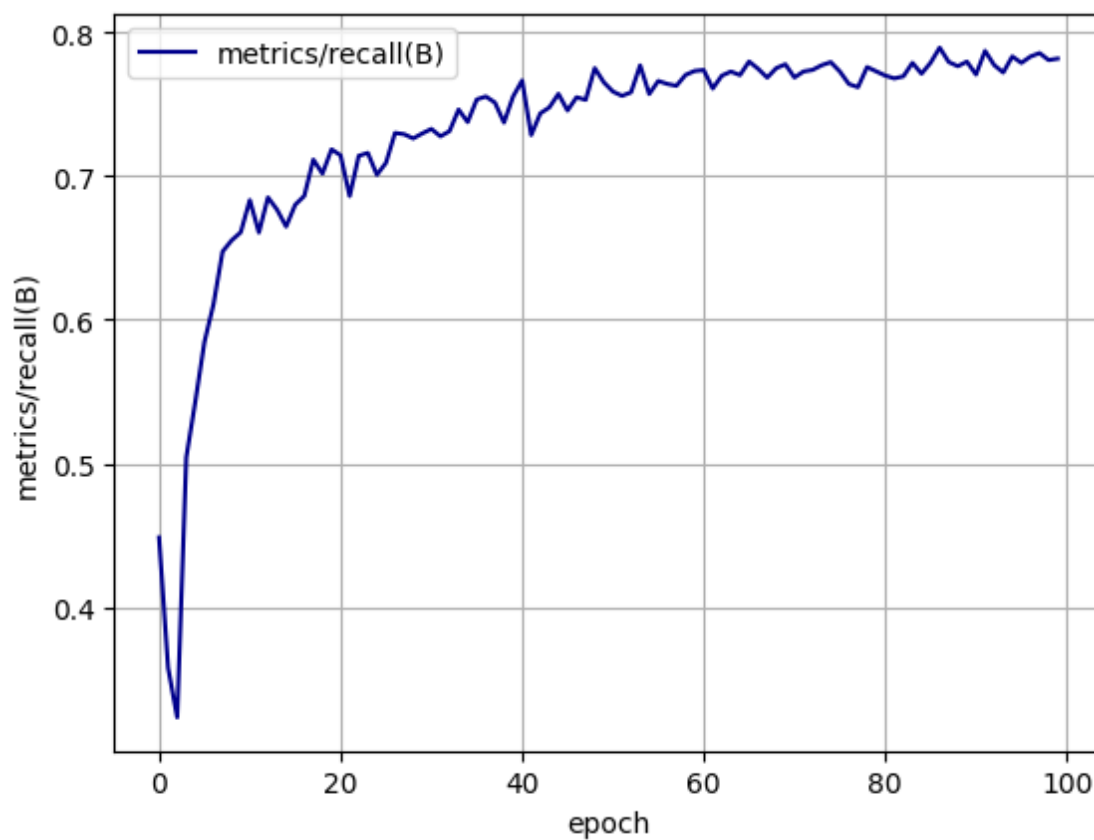


Figure 2.21: Recall vs. epoch for *method 3*.

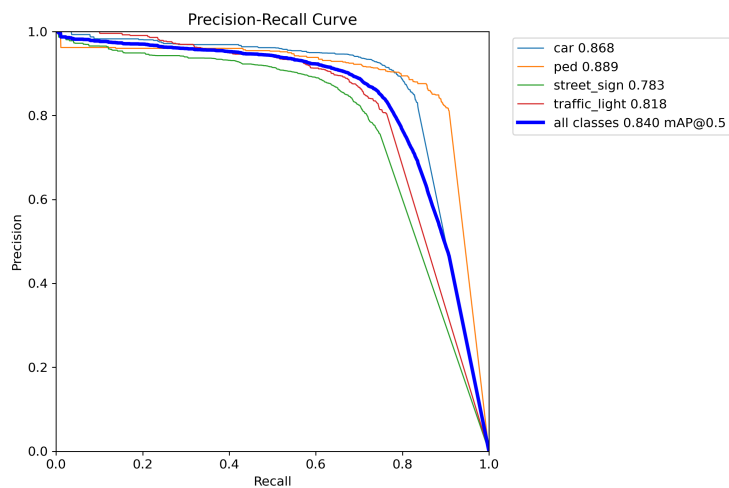


Figure 2.22: Precision–recall curve for *method 3* (100 epochs, batch size 64 and without augmentation, YOLOv8S).

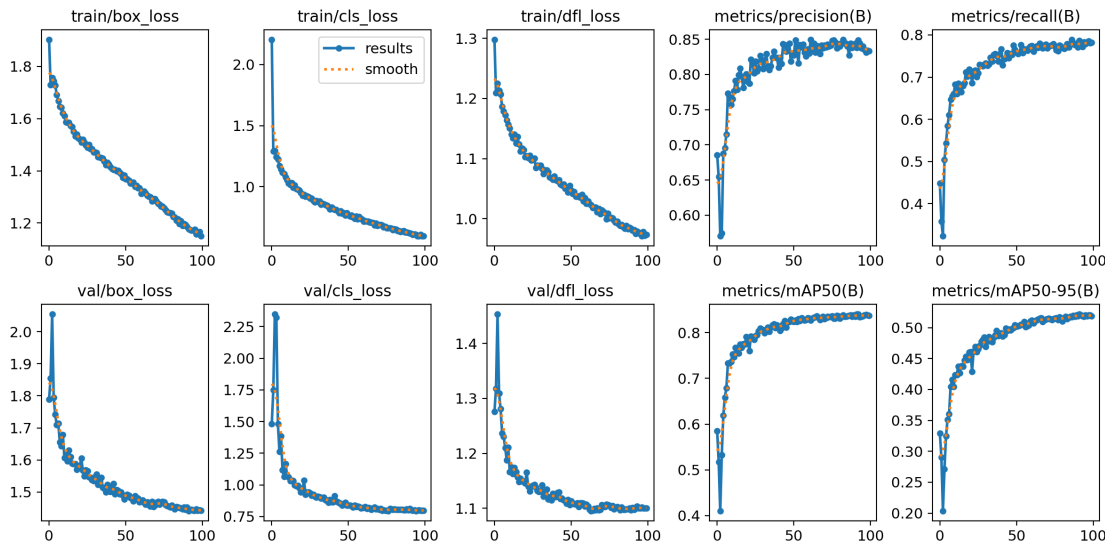


Figure 2.23: Result for *method 3* (100 epochs, batch size 64 and without augmentation, YOLOv8S).

The method 3 outcome is shown in Table 2.3.

Table 2.3: Prediction rate for detecting objects in *method 3*.

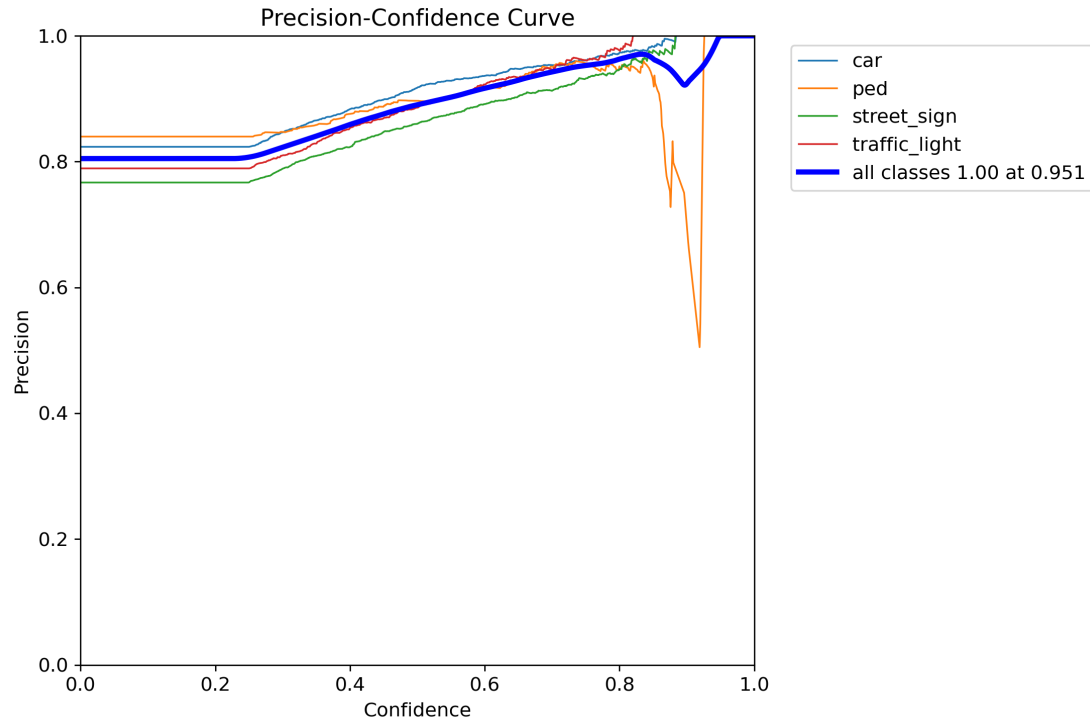
Method	Car %	Pedestrian %	Street Sign %	Traffic Light %
100 EP, BS 16, AUG, YOLOv8S	86.8	88.9	78.3	81.8

In method 4, we use 150 epochs and a batch size of 64 without any augmentation. The model training takes 3 hours. In this method, most hyperparameters increase, including the YOLOv8L with more layers, batch size, and epochs. Increasing these hyperparameters positively affects the final results. The precision metric improves to 95.1%, which is the highest value among all the approaches, as shown in Figure 2.24. In the precision versus epoch image, the precision in epoch 3 is about 60%. From epoch 3 to epoch 20, this metric increases by about 20%. Between epochs 20 and 80, it slightly rises from 80% to nearly 85%. Afterward, it fluctuates around 85%, as seen in Figure 2.25. The recall metric, which serves as a second accuracy measure for the fourth plan, is 82% for all classes, as shown in Figure 2.26. Figure 2.27 illustrates the trend of recall versus epochs. The recall in epoch 2 is about 30%. From epochs 2 to 20, it significantly increases to nearly 70%. From epoch 20 to 80, this metric climbs to about 77%. Between epochs 80 and 150, the recall levels off at approximately 78%. For the precision-recall metric, which is a third accuracy measure, the results for cars, pedestrians, street signs, traffic lights, and overall class rates are 87.7%, 89.4%, 78.7%, 82.0%, and 84.4%, respectively, as shown in Figure 2.28. The precision-recall trend confirms that the proposed method effectively detects objects in the images. The outcomes of the fourth scenario are displayed in Figure 2.29. The precision-recall metric performs better compared to method 1 and method 3. In contrast, its results are worse than those of the second technique.

The method 4 outcome is shown in Table 2.4.

Table 2.4: Prediction rate for detecting objects in the *method 4*.

Method	Car %	Pedestrian %	Street Sign %	Traffic Light %
150 EP, BS 64, No AUG, YOLOv8S	87.7	89.4	78.7	82.0

**Figure 2.24:** Precision–confidence curve for *method 4* (150 epochs, batch size 64 and without augmentation, YOLOv8S).

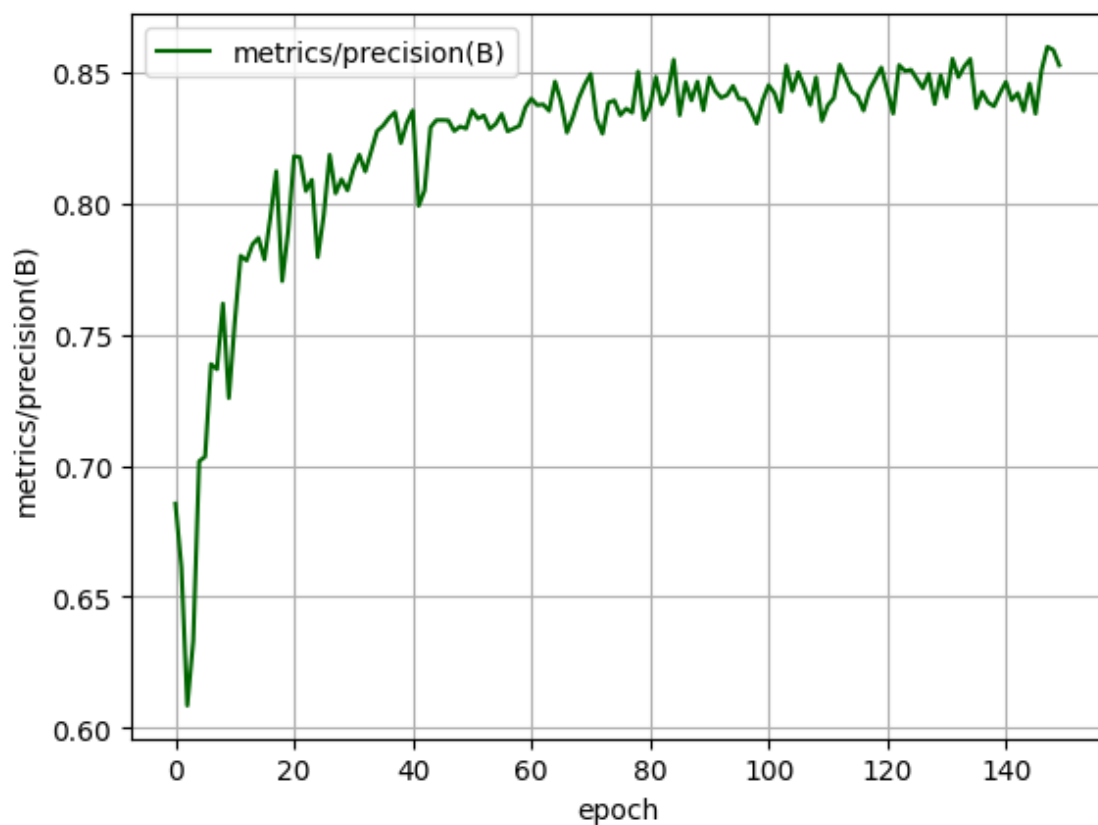


Figure 2.25: Precision vs. epoch for *method 4*.

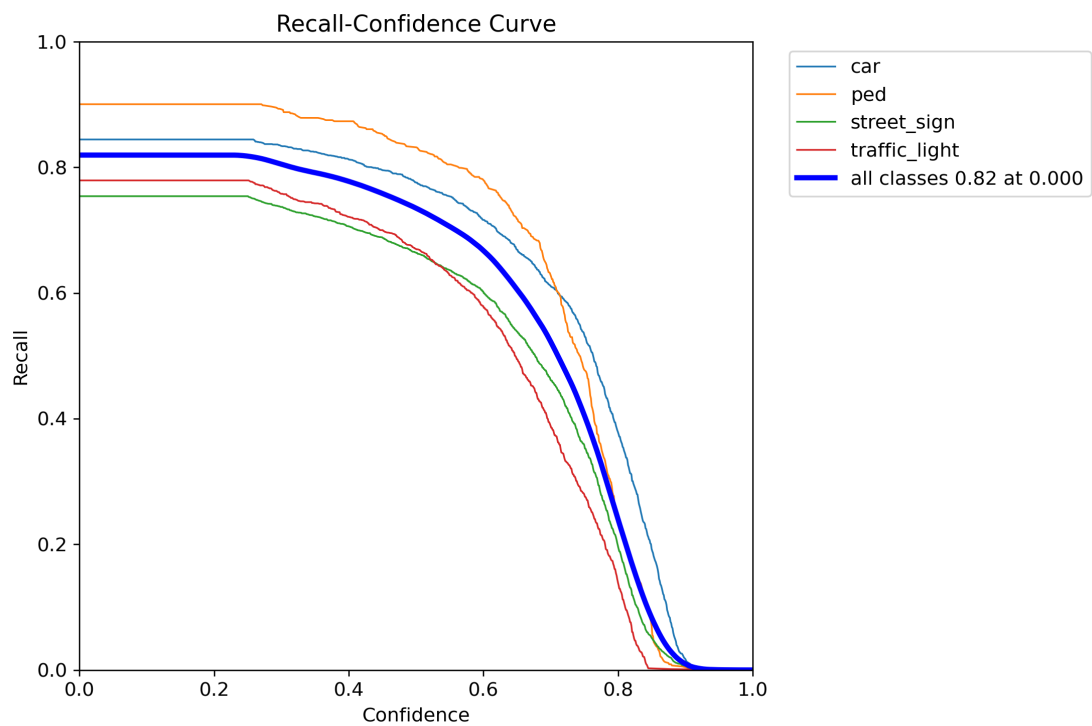


Figure 2.26: Recall–confidence curve for *method 4* (150 epochs, batch size 64 and without augmentation, YOLOv8S).

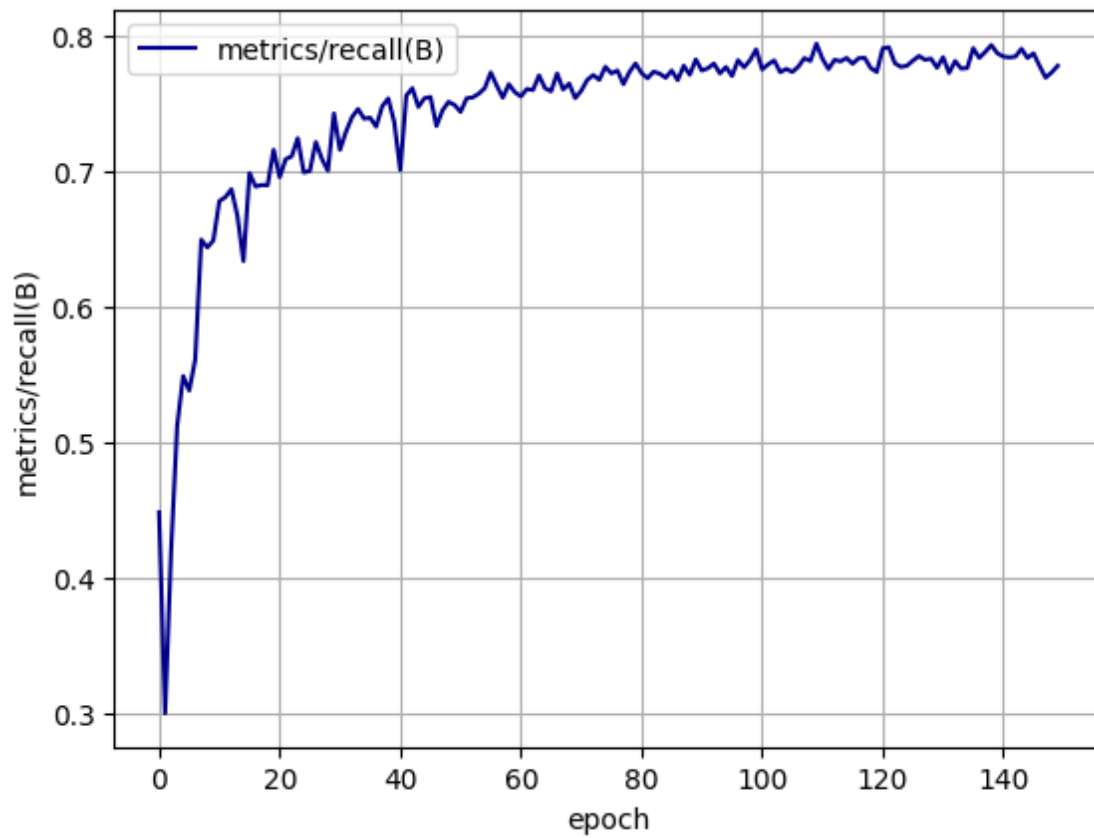


Figure 2.27: Recall vs. epoch for *method 4*.

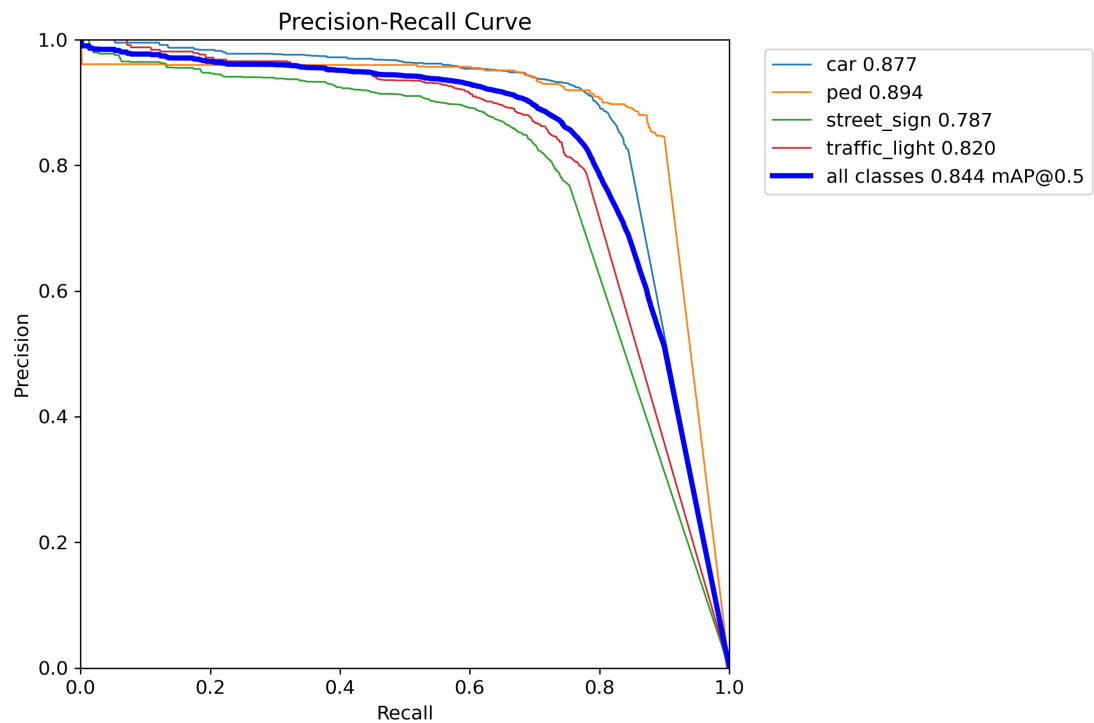


Figure 2.28: Precision–recall curve for *method 4*: (150 epochs, batch size 64 and without augmentation, YOLOv8L).

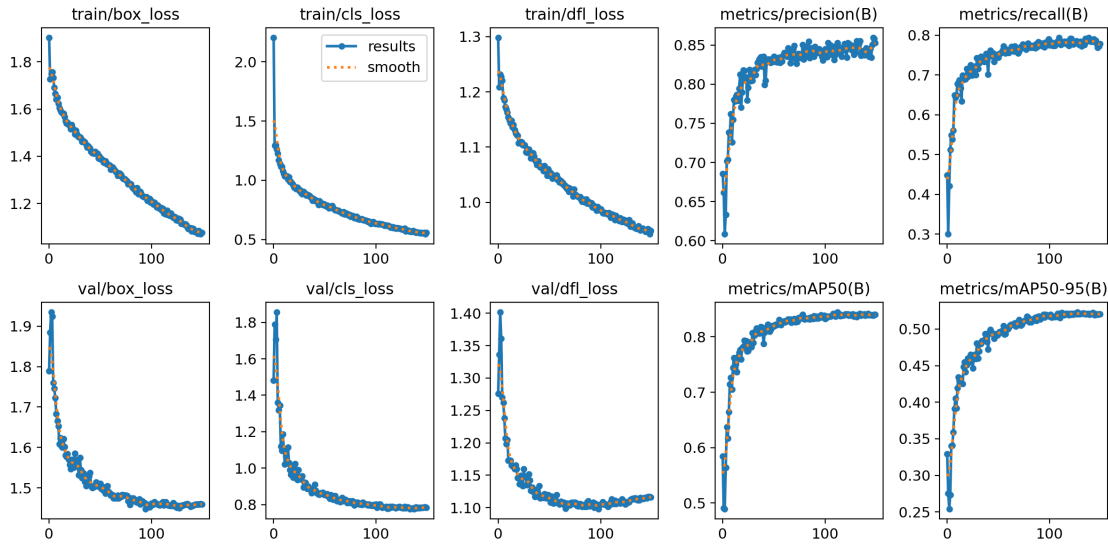


Figure 2.29: Result for *method 4* (150 epochs, batch size 64 and without augmentation, YOLOv8S).

According to the analysis of all the results, the four different techniques detect objects in the images. Method 2 has the highest precision, recall, and mAP50 metrics among all methods. Method 4 ranks second among all approaches. It shows better results in precision, recall, and overall detection rate for all classes compared to method 1 and method 3 for all objects.

The precision, recall, and mAP50 features were analyzed for the four proposed methods, and the results are summarized in Table 2.5. Note that NO AUG means the augmentation method was not used in that instance, while AUG indicates the method benefited from the augmentation technique. As shown in Table 2.5, using augmentation in method 2 led to better object detection rates in the precision, recall, and mAP50 metrics. Increasing the number of epochs in the training process improves the metric percentages. In the fourth approach, 150 epochs were used, and the method's outcomes were better than those in the first method and method 3. Furthermore, the required object detection accuracy percentages are in Tables 2.1 through 2.4 for methods 1 to 4, respectively.

Table 2.5: Precision, recall and mAP50 metrics for the four proposed methods in the foggy dataset.

Methods	Precision (%)	Recall (%)	mAP50 (%) (Average for All Desired Objects)
100 EP, BS 16, NO AUG, YOLOv8S	82.0	73.7	83.7
100 EP, BS 16, AUG, YOLOv8L	84.3	78.5	83.9
100 EP, BS 64, NO AUG, YOLOv8S	81.2	72.95	83.7
150 EP, BS 64, NO AUG, YOLOv8S	82.3	74.3	84.0

The object detection methods in other articles are shown in Table 2.6. In those articles, they did not detect all classes of objects, including cars, pedestrians, traffic lights, and traffic signs, at

the same time.

Some abbreviations in Table 2.6 include: S for sunny weather, R for rainy weather, F for foggy weather, YL for YOLO, FRN for Faster-RCNN (Region-based Convolutional Neural Network), and RN for RCNN (Region-based Convolutional Neural Network). Some parts of Table 2.6 are marked because the data were not available or were not mentioned in the articles. As shown, most articles used images taken in sunny or clear weather conditions. They utilized more datasets or images in their studies compared to our proposed model, which uses just one dataset with about 4000 foggy road images.

By looking at Tables 2.5 and 2.6, we can compare the four proposed methods with other methods found in studies by other scientists. Most articles show that the authors used more epochs than we did in our research. Additionally, the number of images used in other studies is greater than in ours. If we had more images, we would likely achieve better results in research related to object detection.

To prove the robustness of the proposed model, we tested various images from three other datasets. The datasets included road images taken in sunny and clear weather. We combined these three datasets into a new, unique dataset for testing. We downloaded the front and rear images of cars, along with the Semantic Segmentation Makassar (IDN) Road and traffic sign datasets in YOLO format from the Kaggle website to evaluate the proposed model. The new dataset consisted of approximately 430 images. In the following sections, we will add some images to demonstrate the model's accuracy in identifying objects in sunny and clear weather conditions. The results will show how well the proposed model detects objects. The test images have been added. Cars appear as red bounding boxes, street signs are orange, traffic lights are yellow, and pedestrians (crosswalks) are blue as identified by the model. In Figure 2.30, we can see that the model identified cars and traffic signs with accuracies of 92%, 63%, 88%, 66%, and 87% respectively in sunny weather images on the road.

Table 2.6: *Object detection methods and classes used in other articles.*

Methods	Epochs	Precision (%)	Recall (%)	mAP50 (%)	Images	Class Names
[47], YLv3, S, R	120	47.4	33.7	44.47	20,000	Multi-objects
[52], YLv7-tiny, S	100	-	-	80.02	16,441	Bike, motorcycle, bus, car, people
[50], YLv2, 3, FRCNN, S	-	-	-	48.6	13,427	Traffic light status (green, yellow, red, off)
[54], YLv3, 4, FRN, RN, S	-	70.14	85.01	69.01	2,000,000	Car, truck, van, tricycle, bike, pedestrian, people, bus, motor, awning-tricycle
[18], YLv5, S	80	-	-	43.3	5000	Person, rider, car, truck, bus, train, motor, bike
[56], YLv3, S	120	63.0	55.0	46.60	330,000	Car, truck, pedestrian, traffic light, sign
[59], YLv8s, Bi-PAN-FPN, GhostblockV2, S	150	79.5	79.2	83.7	8629	People, pedestrian, car, van, truck, tricycle, bus, motor, bike, awning-tricycle
[53], YLv5, 7, 8, YLR, S	250	79.37	44.16	36.49	8000	Bus, bike, car, pedestrian, truck
[60], Faster-RCNN, S	120,000	-	-	-	383,240	People
[19], FRN+OHEM, S	70,000	-	-	44.46	5000	Traffic sign
[62], FRN, YLv5, DETR, S	200	98.94	72.22	-	7500	Multi-objects
[63], YLv5, 6, 7, YLGW, S, R, F	300	-	87.84	-	100,000	People
[65], FRN, S	150,000	-	87.7	80.29	110,000	Traffic light
[16], YLv3, S	150,000	71.51	69.05	63.80	95,000	Traffic light status (none, off, red, green)



Figure 2.30: Detected objects by the proposed model in Semantic Segmentation Makassar (IDN) Road Dataset—the second test dataset.

Figure 2.31 shows the required objects selected in the picture, including cars, traffic lights, and traffic signs. The image lacks object detection. However, it is important to note that the objects were detected using a YOLOv8 model trained for foggy images, not sunny ones. It is challenging for this model to mark some objects, such as cars, pedestrians, traffic signs, and traffic lights. The

patterns and color contrasts of objects in sunny images differ from those in foggy images.



Figure 2.31: Detected objects from traffic signs dataset in YOLO format—the third test dataset.

Also, Figure 2.32 was taken in the suburbs, which is a different setting from our image dataset. The various environmental images can create a new challenge for the proposed model. It also shows how strong the proposed object detection model is. The model detected the cars with an accuracy of 88%. However, another car and some traffic signs were not detected on the suburban road.



Figure 2.32: Detected car in suburbs from traffic signs dataset in YOLO format.

Figure 2.33 has a complicated background including buildings and trees: the proposed model

was able to identify the objects correctly.



Figure 2.33: Detected objects from front and rear image of car dataset with complex background—the first test dataset.

Table 2.7 summarizes the accuracy of the detected objects in the tested datasets. Note that zero values mean that objects are not present or the model did not detect the object.

Table 2.7: The predicted object detection accuracy rate by the proposed model in good and sunny images in the test dataset (good and sunny image datasets).

Dataset Name	Car (%)	Pedestrian (%)	Street Sign (%)	Traffic Light (%)
1-Front and rear image of car	80.74	0.00	70.95	0.00
2-Semantic Segmentation Makassar (IDN) Road Dataset	77.09	67.50	71.23	67.55
3-Traffic signs dataset in YOLO format	73.86	0.00	72.34	68.42
Overall detection rate in the three datasets	77.23	67.50	71.35	67.98

In addition, we trained the new test dataset using 430 sunny and clear images based on the prior structures. We used hyperparameters on the foggy image dataset, which included the following

methods: the method with 1100 epochs, batch size 16, no augmentation, and YOLOv8S; the method with 2100 epochs, batch size 16, with augmentation, and YOLOv8L; the method with 3100 epochs, batch size 64, no augmentation, and YOLOv8S; and the method with 4150 epochs, batch size 64, no augmentation, and YOLOv8S. The outcomes will be shown in the following figures: Figures 2.34 to 2.37 for the four methods used in the foggy image dataset. As can be seen, the results of method 1 (100 epochs, YOLOv8S, batch size 16, and no augmentation) will be presented in precision, recall, precision-recall, and final report images, respectively.

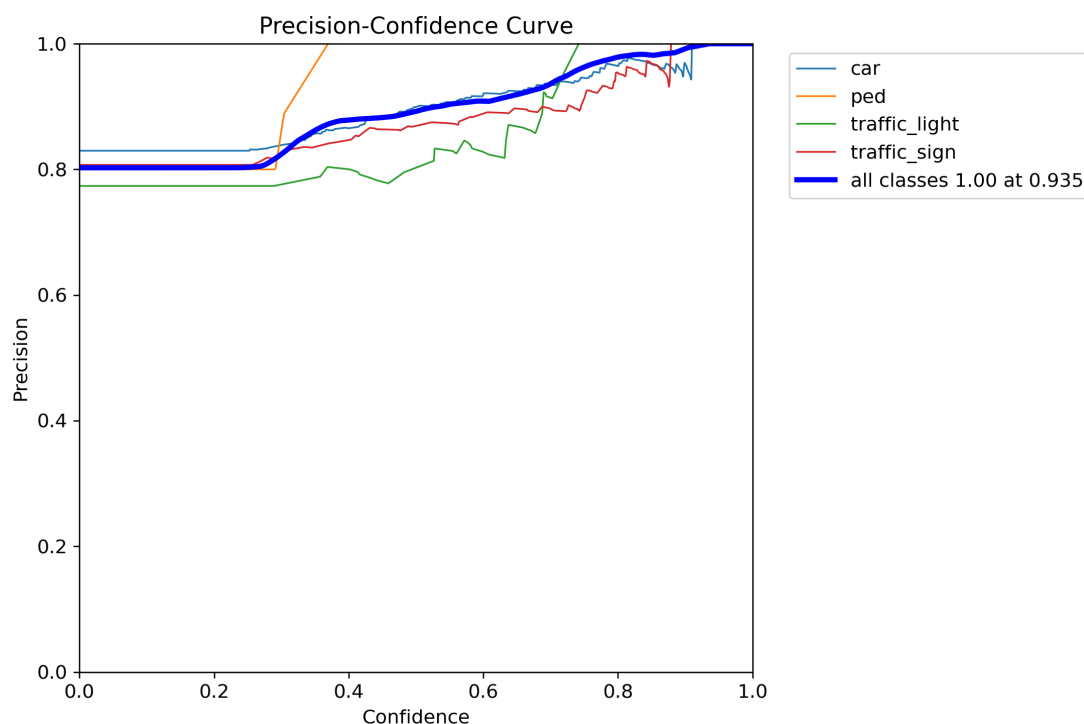


Figure 2.34: Precision rate for *method 1* in test dataset (good and sunny dataset).

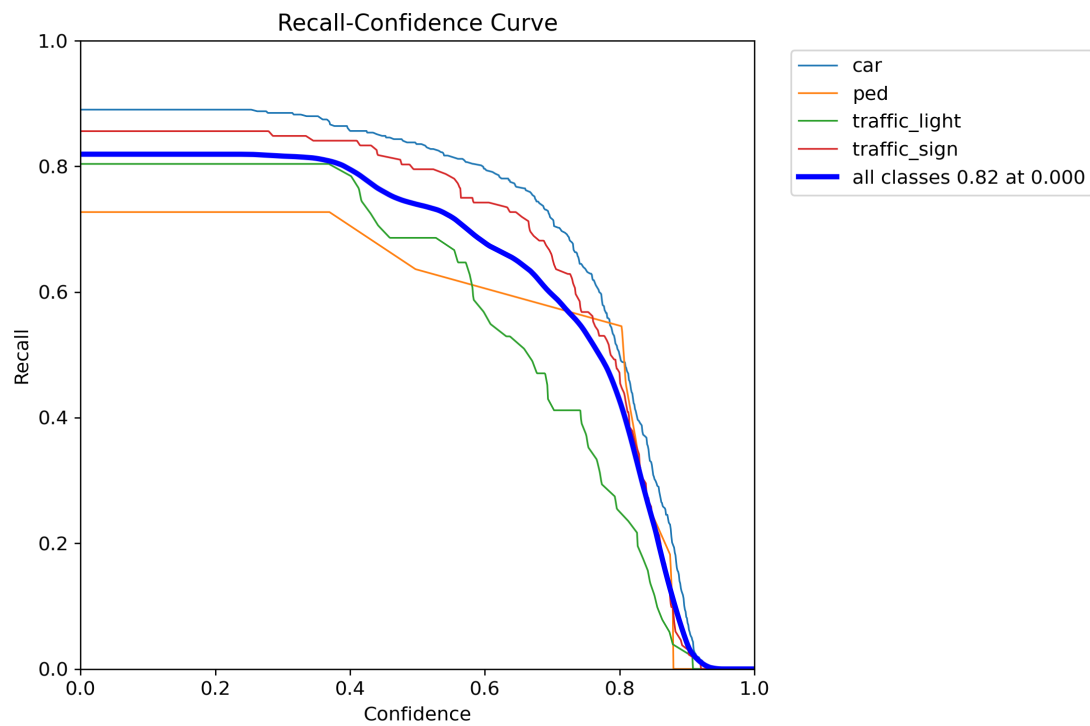


Figure 2.35: Recall rate in *method 1* in the test dataset (good and sunny dataset).

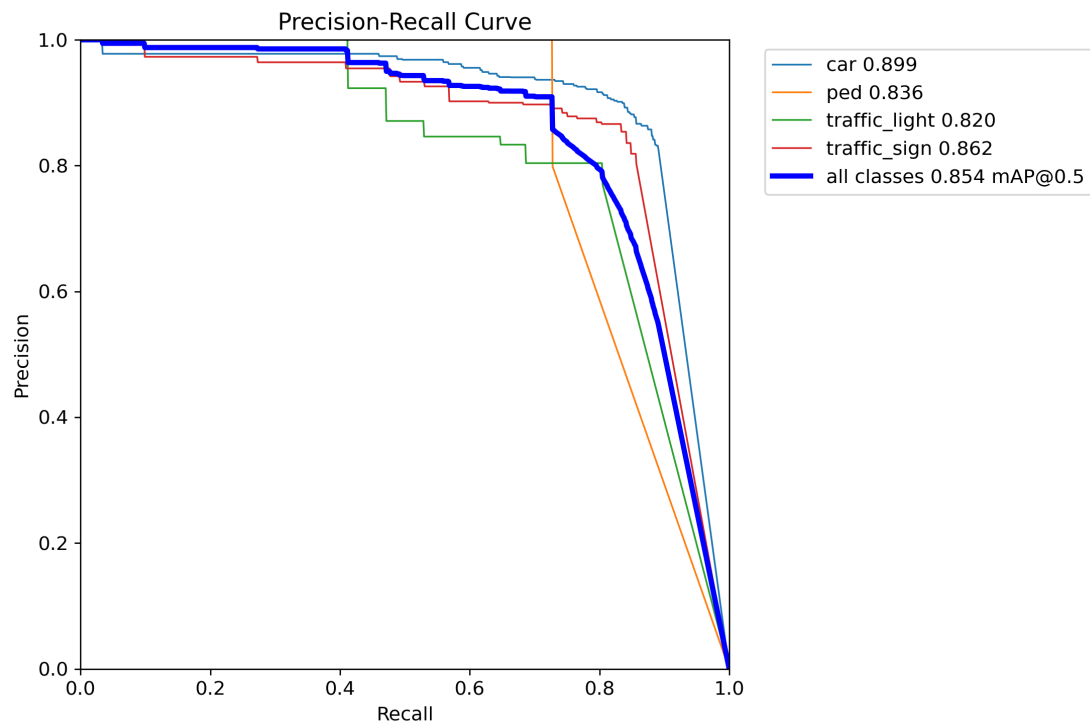


Figure 2.36: Precision-recall rate for *method 1* in test dataset (good and sunny dataset).

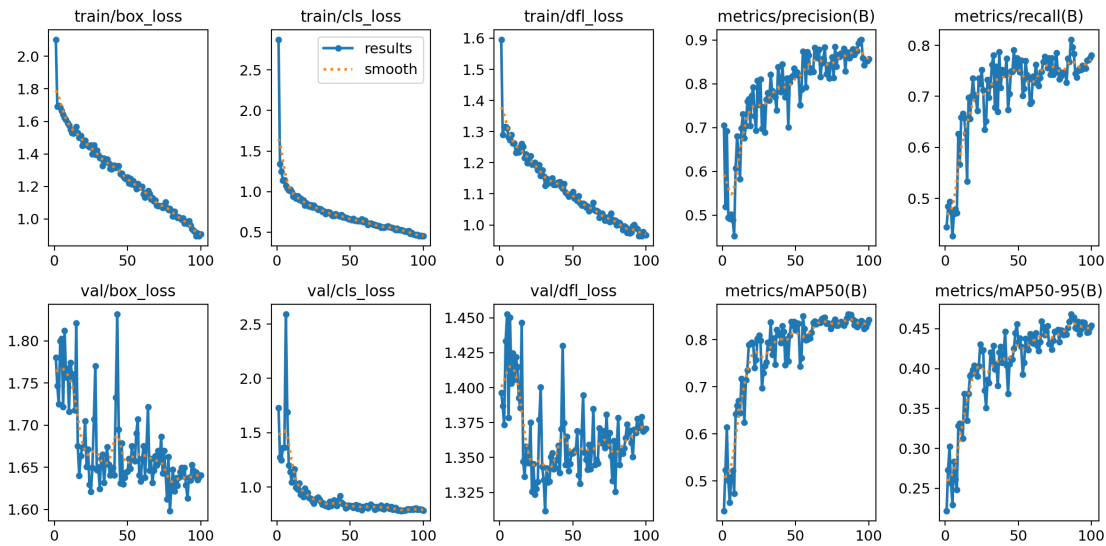


Figure 2.37: Final report rate in *method 1* in the test dataset (good and sunny dataset).

The second approach was also tested in the test dataset. Method 2 includes 100 epochs (100EPC), a batch size of 16 (BS16), augmentation (AUG), and YOLOv8L. In the following figures, Figures 2.38 to 2.41, precision, recall, precision-recall, and the final report will be displayed in the images.

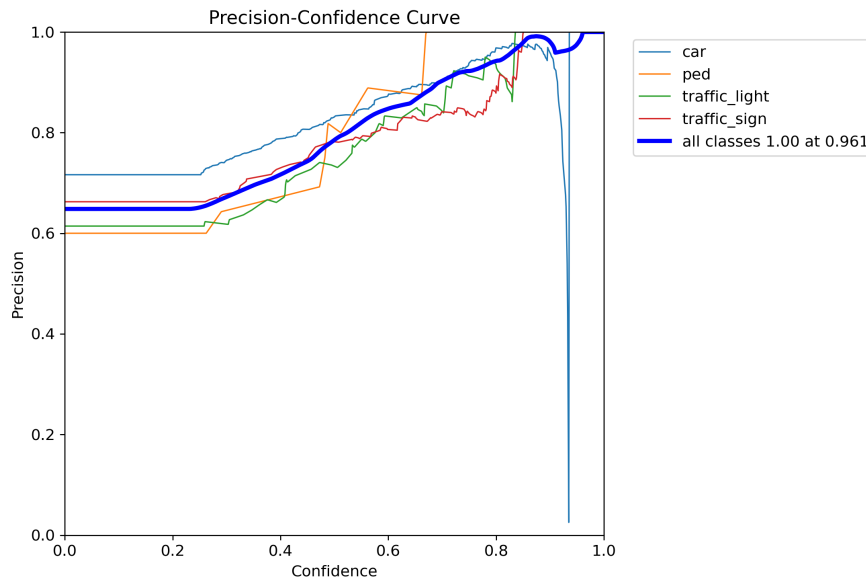


Figure 2.38: Precision rate in the *method 2* approach in sunny and good image dataset (100 EPC, BS16, AUG and YOLOv8L in test dataset).

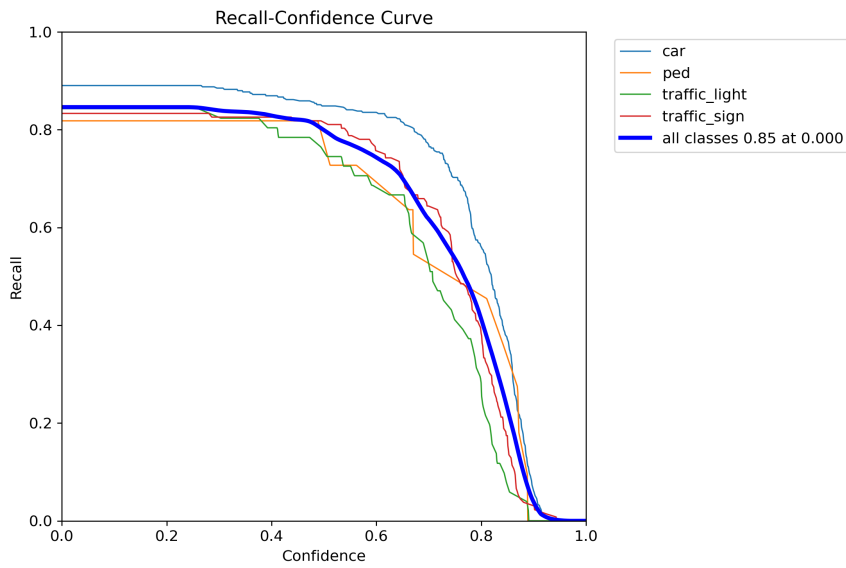


Figure 2.39: Recall rate in the *method 2* approach in sunny and good image dataset (100 EPC, BS16, AUG and YOLOv8L in test dataset).

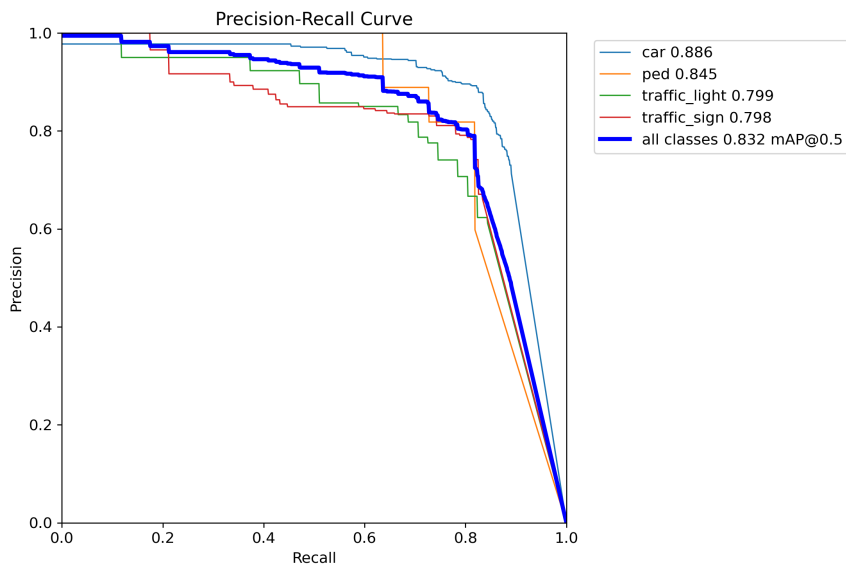


Figure 2.40: Precision-recall rate in the *method 2* approach in sunny and good image dataset (100 EPC, BS16, AUG and YOLOv8L in test dataset).

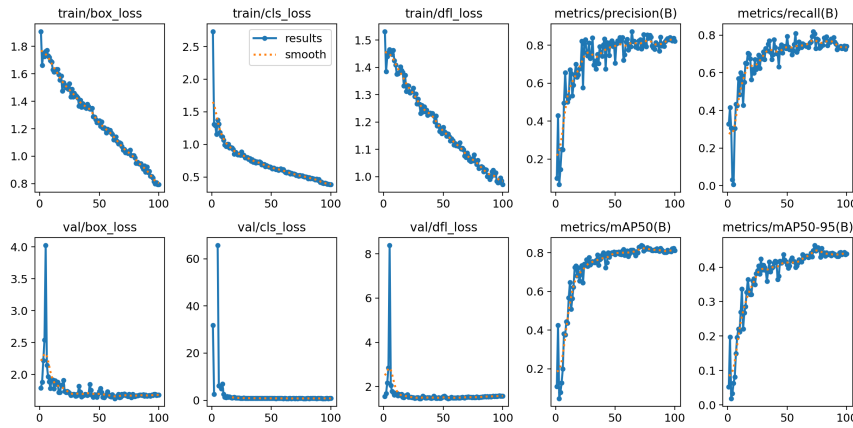


Figure 2.41: The final report in the *method 2* approach in sunny and good image dataset (100 EPC, BS16, AUG and YOLOv8L in test dataset).

The third technique was used to test the test dataset, which included a sunny and good image dataset. We applied 100 epochs, with a batch size of 64 and no augmentation, using YOLOv8S. The results are shown in Figures 2.42, 2.45 for precision, recall, precision-recall, and the final report, respectively.

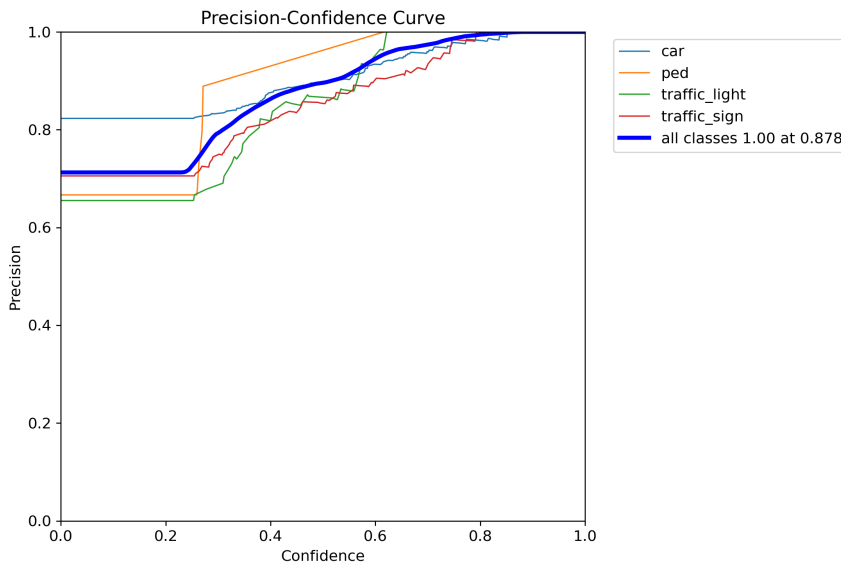


Figure 2.42: Precision rate in the *method 3* approach in sunny and good image dataset (100 EPC, BS16, No AUG and YOLOv8S in test dataset).

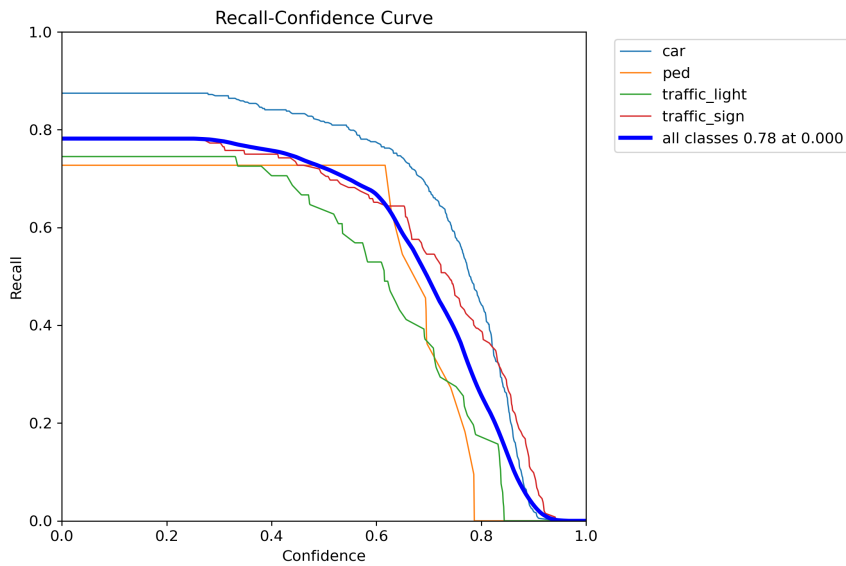


Figure 2.43: Recall rate in the *method 3* approach in sunny and good image dataset (100 EPC, BS16, No AUG and YOLOv8S in test dataset).

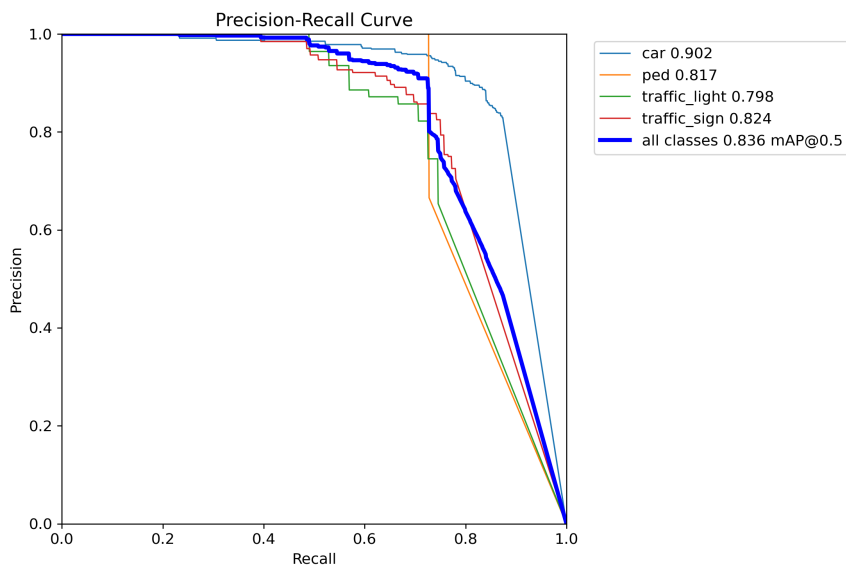


Figure 2.44: Precision-recall rate in the *method 3* approach in sunny and good image dataset (100 EPC, BS16, No AUG and YOLOv8S in test dataset).

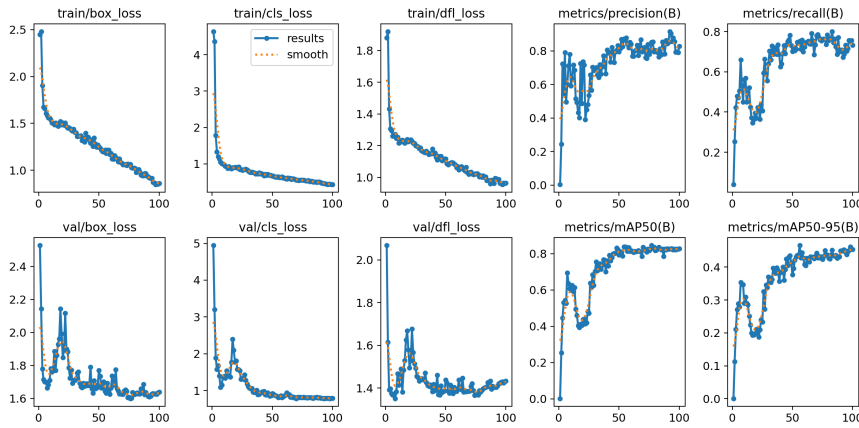


Figure 2.45: The final report in the *method 3* approach in sunny and good image dataset (100 EPC, BS16, No AUG and YOLOv8S in test dataset).

The final (fourth) approach used the test dataset, which was a sunny and clear image dataset. It consisted of 150 epochs, a batch size of 64, no augmentation, and YOLOv8S. The outcomes are shown in Figures 2.46 to 2.49. They display precision, recall, precision-recall, and the final report individually.

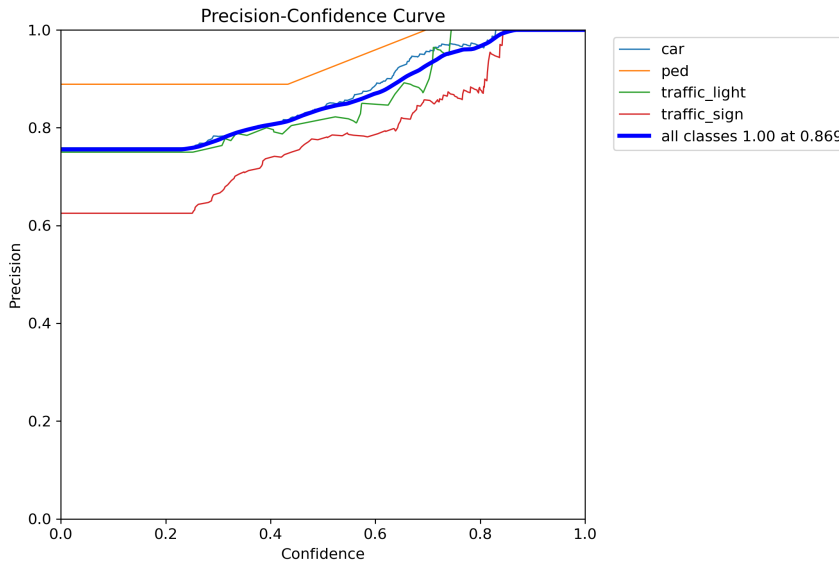


Figure 2.46: Precision rate in the *method 4* approach in sunny and good image dataset (150 EPC, BS64, No AUG and YOLOv8S in test dataset).

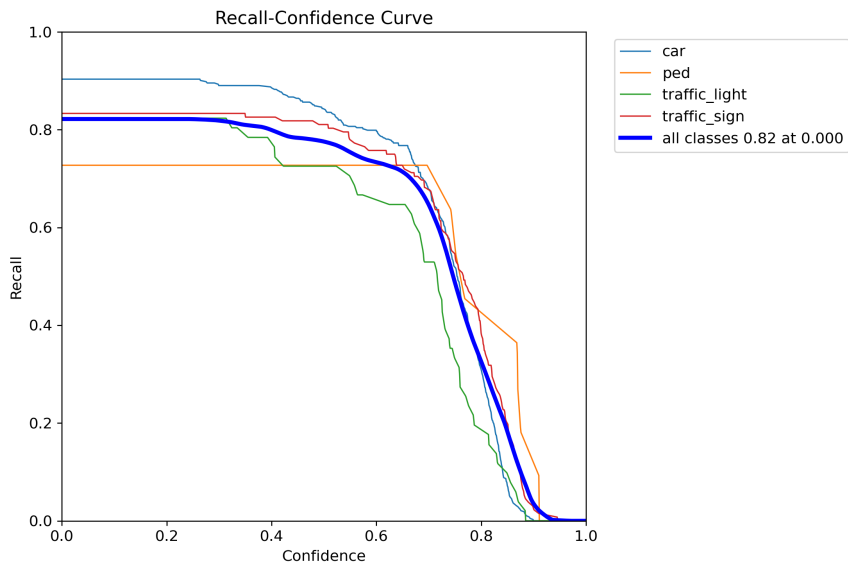


Figure 2.47: Recall rate in the *method 4* approach in sunny and good image dataset (150 EPC, BS64, No AUG and YOLOv8S in test dataset).

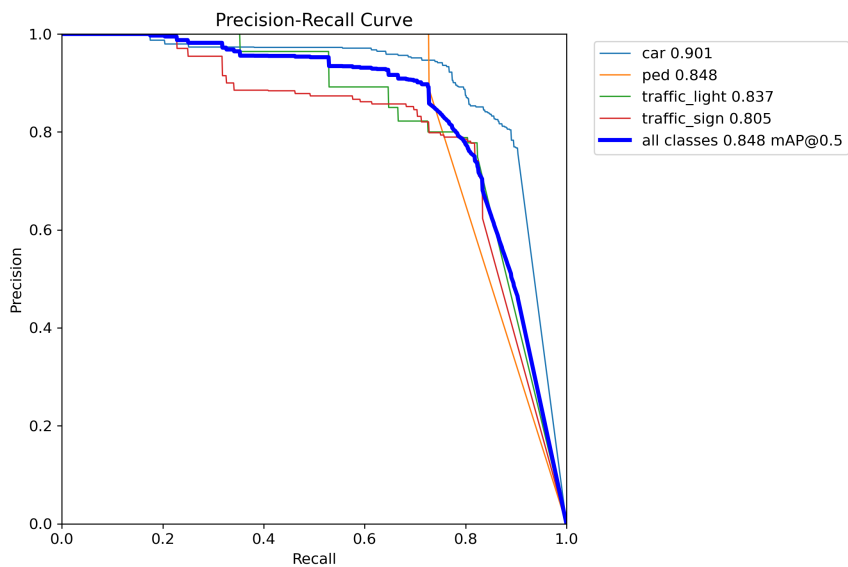


Figure 2.48: Precision-recall rate in the *method 4* approach in sunny and good image dataset (150 EPC, BS64, No AUG and YOLOv8S in test dataset).

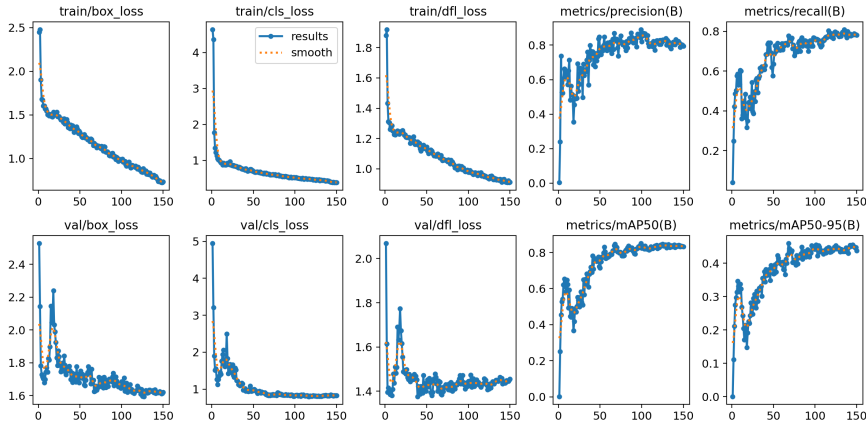


Figure 2.49: The final report in the **method 4** approach in sunny and good image dataset (150 EPC, BS64, No AUG and YOLOv8S in test dataset).

In Table 2.8, we present the precision, recall, and mAP50 metrics for the good or sunny test image dataset, which contains about 400 images. These metrics are shown for all four strategies applied to the foggy dataset, which includes around 4000 images. Unfortunately, in the final method, we experienced overfitting when we used 150 epochs, a batch size of 64, no augmentation, and the YOLOv8S model. This overfitting likely stems from the small number of images, a problem that worsens as the number of epochs increases.

Table 2.8: Precision, Recall, and mAP50 (%) for proposed methods on sunny test dataset.

Method	Precision	Recall	mAP50
100 EP, BS 16, No AUG, YOLOv8S	85.71	78.08	84.25
100 EP, BS 16, AUG, YOLOv8L	82.26	74.18	81.14
100 EP, BS 64, No AUG, YOLOv8S	82.91	73.27	82.85
150 EP, BS 64, No AUG, YOLOv8S	79.48	78.08	83.17

In Table 2.9, there is a comparison of the object detection ratio in foggy conditions and clear weather images based on the proposed model. We averaged all object detection precision and recall metric ratios from the four methods listed in the first row of Table 2.9. In the second row, we use data from the overall section in Table 2.9. It is clear that the object detection ratio in the test image sample dataset, which includes sunny and clear weather images, is lower than that in the foggy image dataset.

Table 2.9: The predicted object detection accuracy rate in foggy and good weather conditions in the proposed model.

Dataset Name	Car %	Pedestrian %	Street Sign %	Traffic Sign %
Foggy image dataset (has about 4000 images)	87.47	88.8	79.12	81.87
Sunny and good images test sample dataset (has about 400 images)	77.23	67.50	71.35	67.98

We compared YOLOv8 and YOLOv5 based on the study. We used small and large versions of the YOLO model for this comparison. We aimed to maintain the same configuration for the four methods used in the study. The comparison is shown in Table 2.10. Table 2.10 indicates that the results for YOLOv8 are better than those for YOLOv5 in precision, recall, and mAP50 metrics in the foggy dataset.

Table 2.10: Precision, recall and mAP50 metrics in YOLOv8 compared with the metrics in YOLOv5 for the foggy image dataset.

Models	Methods	Precision (%)	Recall (%)	mAP50 (%)
YOLOv8	100 EP, BS 16, NO AUG, YOLOv8S	82.0	73.7	83.7
YOLOv5	100 EP, BS 16, NO AUG, YOLOv5S	70.92	60.83	62.16
YOLOv8	100 EP, BS 16, AUG, YOLOv8L	84.3	78.5	83.9
YOLOv5	100 EP, BS 16, AUG, YOLOv5L	82.86	73.50	78.45
YOLOv8	100 EP, BS 64, NO AUG, YOLOv8S	81.2	72.95	83.70
YOLOv5	100 EP, BS 64, NO AUG, YOLOv5S	72.51	58.40	61.55
YOLOv8	150 EP, BS 64, NO AUG, YOLOv8S	82.3	74.3	84.0
YOLOv5	150 EP, BS 64, NO AUG, YOLOv5S	74.10	60.15	63.93

EP: Epochs, BS: Batch Size, AUG: Augmentation, S: Small, L: Large.

In the good and sunny image dataset, we compare the two models. We also use small and large versions of the YOLO model for YOLOv8 and YOLOv5. The same setup is used for the four methods in the study. The results are shown in Table 2.11. Some results from YOLOv5 are lower because the number of images in the good or sunny dataset is about 400. When augmentation is

not used, the objects are not detected correctly, leading to lower outcomes.

Table 2.11: Precision, recall and mAP50 metrics in YOLOv8 compared with those in YOLOv5 over the good image dataset.

Models	Methods	Precision (%)	Recall (%)	mAP50 (%)
YOLOv8	100 EP, BS 16, NO AUG, YOLOv8S	85.71	78.08	84.25
YOLOv5	100 EP, BS 16, NO AUG, YOLOv5S	59.50	40.76	41.26
YOLOv8	100 EP, BS 16, AUG, YOLOv8L	82.26	84.18	81.14
YOLOv5	100 EP, BS 16, AUG, YOLOv5L	62.01	64.65	64.33
YOLOv8	100 EP, BS 64, NO AUG, YOLOv8S	82.91	73.27	82.85
YOLOv5	100 EP, BS 64, NO AUG, YOLOv5S	47.27	39.93	38.40
YOLOv8	150 EP, BS 64, NO AUG, YOLOv8S	79.48	78.08	83.17
YOLOv5	150 EP, BS 64, NO AUG, YOLOv5S	67.73	39.17	40.84

EP: Epochs, BS: Batch Size, AUG: Augmentation, S: Small, L: Large.

The required object rates are listed in Table 2.12 for the foggy and sunny image datasets. As shown, there is a comparison between YOLOv8 and YOLOv5. YOLOv8 performs better at object detection than YOLOv5.

Table 2.12: The average predicted object detection accuracy rate in foggy and good weather conditions in YOLOv8 compared with YOLOv5 based on the four methods used.

Models	Dataset Name	Car %	Pedestrian %	Street Sign %	Traffic Light %
YOLOv8	Foggy	87.47	88.8	79.12	81.87
YOLOv5	Foggy	69.75	73.70	56.37	67.08
YOLOv8	Sunny or Good	77.23	67.50	71.35	67.98
YOLOv5	Sunny or Good	68.42	26.4	52.32	45.77

2.6 Future Work

The research presents a method to recognize objects such as cars, pedestrians, traffic lights, and street signs in images with poor weather conditions, like fog. We propose an approach based on the YOLOv8 method for detecting objects and improving the detection accuracy rate. Detecting objects in poor weather conditions is challenging for self-driving vehicles. This research addresses these challenges by introducing a framework with specific contributions, outlined below. To systematically tackle these challenges, the following section outlines the key contributions of this study, which form the basis of the proposed detection framework

Detecting objects in poor weather conditions is challenging for self-driving vehicles. This framework addresses both methodological and practical aspects of adverse-weather object detection. The contributions of this study are as follows: (1) the development of a detection framework based on YOLOv8, especially for foggy weather conditions; (2) the application of a preprocessing method that enhances robustness by reducing image size and quality; (3) the systematic investigation of four training setups utilizing differences in epochs, batch sizes, and brightness augmentation; and (4) evidence that improved detection performance is realized on actual foggy images, as opposed to artificially created datasets. Collectively, these contributions provide a benchmark for future autonomous vehicle systems and advance adverse-weather object detection.

Building on these contributions, we tested the model using 4,000 foggy images, exploring variations in hyperparameters to evaluate performance such as epochs, batch size, and augmentation methods. We present four methods and compare them for object detection in different weather conditions. The results show that the proposed techniques perform well, especially in bad weather. There are some limitations in the study related to object detection. In some cases, the augmentation method causes overfitting, which leads to errors in the results. Also, increasing epochs can result in overfitting. Some small objects in the image dataset might not be detected by certain models because these objects lack details. In the study, the first image dataset does not contain good and sunny images. As a result, we had to use three good and sunny image datasets to apply and test them on the proposed model.

While our study focuses on image-based detection, real-world autonomous vehicles often rely on multiple sensors to complement visual information, enhancing object recognition under challenging conditions. Some companies that make autonomous vehicles can use other technologies like Lidar, GPS, ultrasonics, and other mobility sensors in their products at the same time. This helps them detect objects on the street more effectively. The sensors can also work with smart traffic lights or traffic signs to improve smart city development. An autonomous car can use multiple sensors to identify objects on the road. This reduces errors and improves the model's ability to recognize objects in challenging situations, such as low visibility or high-speed conditions. Additionally, there is great potential to use more sensors in autonomous vehicles to recognize objects in real-time or complex environments. Future work will test the proposed model based on YOLOv8 using other datasets and images captured by autonomous vehicles.

Building on the potential of multisensory integration mentioned above, the methodology also demonstrates in particular, this paper presents the efficacy of a well-optimized YOLOv8-based approach to detect street objects in foggy and low-visibility conditions. Although a number of open challenges still exist-such as the detection of small-size objects and possible overfitting depending

on specific configurations-the results underline the potential of robust preprocessing, suitable hyperparameter tuning, and the use of real-world foggy images in improving detection performance. On one hand, the proposed methodology lays a solid foundation toward future advancements in autonomous driving and intelligent transportation systems and opens several promising pathways for further improvement: multisensory fusion and extensive testing on real-world scenarios.

Chapter 3

Convolutional Neural Network - CNN

[Back to Contents](#)

3.1 Background

Nowadays, autonomous systems, particularly in the fields of robotics and autonomous vehicles (AVs), prioritize strong perception and intelligent path planning to operate efficiently in complex and dynamic environments. Therefore, AVs need to analyze the environment, detect obstacles and identify possible routes with minimal error, since errors often lead to collisions on the road. Convolutional neural networks (CNNs) have significantly improved AV understanding, road recognition, real-time processing and decision-making capabilities. This study proposes integrating a CNN model into the Stable Baselines3 (SB3) reinforcement learning library to interact with a sample SUMO (Simulation of Urban Mobility) map and determine the optimal path based on multiple road metrics. These metrics include minimizing travel time, avoiding or minimizing traffic lights, reducing congestion (number of vehicles on each edge), avoiding loops and dead-end edges, minimizing distance to the destination and maintaining a stable speed on the map. These metrics also address gaps found in previous studies. These findings have potential applications in autonomous vehicles (AVs) and electric autonomous vehicles (EAVs). With the growing adoption of electric vehicles, the integration of such optimization methods could play a significant role in enhancing the efficiency and sustainability of future transportation systems.

Over the years, the number of vehicles has increased among car companies. In addition, people are encouraged to use their vehicles to commute to/from their jobs. So, it led to dealing with a terrible problem in urban areas, that is, traffic congestion. Today, traffic participation plays a vital role in city roads as it results in a huge energy consumption and wastes time [75]. People are seeking a path that uses less traffic and an optimal path based on travel time and fuel usage.

An autonomous vehicle was developed to rectify traffic and find a trajectory on the street without a driver. Detecting a path in an autonomous vehicle plays an important role in path planning[76]. In [77], a proposed method has been employed to predict the trajectory with priority of accuracy and speed. Moreover, In [78],[79], using a CNN or GCNN (Graph Convolutional Neural Networks) helps collect more information from the environment, as collecting data from the environment by sensors is not sufficient for practical path planning. The GCNN approach uses more resources, as most research has used a CNN instead of a GCNN. In this study, environmental

data have been collected and have also benefited from random data to find the trajectory. It has demonstrated how locations and times are connected in the trajectory data. Moreover, it can predict future paths while keeping track of how certain or uncertain those predictions are. The study indicates that the achievement of environmental data plays a key role in estimating the planning of the route in the future. The autonomous vehicle will be able to overcome and mitigate traffic limitations using some famous and popular technologies such as DL, ML, and RL. They will provide travel convenience, safer trips, more efficient, and environmentally friendly transportation solutions for residents. Additionally, these technologies can facilitate path planning by forecasted actions of nearby vehicles on the road.

Over the years, the number of vehicles has increased among car companies. In addition, people are encouraged to use their vehicles to commute to/from their jobs. So, it led to dealing with a terrible problem in urban areas, that is, traffic congestion. Today, traffic participation plays a vital role in city roads as it results in a huge energy consumption and wastes time [75]. People are seeking a path that uses less traffic and an optimal path based on travel time and fuel usage.

An autonomous vehicle was developed to rectify traffic and find a trajectory on the street without a driver. Detecting a path in an autonomous vehicle plays an important role in path planning[76]. In [77], a proposed method has been employed to predict the trajectory with priority of accuracy and speed. Moreover, In [78],[79], using a CNN or Graph Convolutional Neural Networks (GCNN) helps collect more information from the environment, as collecting data from the environment by sensors is not sufficient for practical path planning. The GCNN approach uses more resources, as most research has used a CNN instead of a GCNN [80]. In this study, environmental data have been collected and have also benefited from random data to find the trajectory. It has demonstrated how locations and times are connected in the trajectory data. Moreover, it can predict future paths while keeping track of how certain or uncertain those predictions are. The study indicates that the achievement of environmental data plays a key role in estimating the planning of the route in the future. The autonomous vehicle will be able to overcome and mitigate traffic limitations using some famous and popular technologies such as DL, ML, and RL. They will provide travel convenience, safer trips, more efficient, and environmentally friendly transportation solutions for residents. Additionally, these technologies can facilitate path planning by forecasted actions of nearby vehicles on the road.

Traffic congestion, energy consumption, and wasted time, combined with the growing number of vehicles in our urban areas, more specifically, as traffic congestion levels increase, time efficiency decreases, along with energy consumption. Path planning or route finding has commonly relied on environmental sensors alone. Environmental sensors alone are clearly insufficient to deal with the complexity and uncertainty of traffic in the real world. Autonomous vehicles are promising mobility solutions with the help of deep learning technologies, machine learning, and reinforcement learning, which can cope with anticipating vehicle trajectories, optimize vehicle choices, and provide a safer means of navigation. However, it has been documented that the position of many studies remains limited to precision or speed as the real measures of interest, and critical aspects of the road situation have not been fully addressed, like those related to congestion, traffic lights, traffic loops, travel distance, etc. The goal of my research was to overcome these gaps by attempting to use CNN-based perception combined with reinforcement learning in the SUMO environment to identify paths that are optimal in efficiency, safety, and sustainability measures. In the end, it is hoped that the findings will contribute to the development of both autonomous and electric vehicles that can help reduce congestion while providing a better travel experience, and contribute to a more sustainable means

of travel for cities.

3.2 Literature Review and Related Work

In recent years, deep learning (DL) as a subfield of artificial intelligence (AI) has been utilized in a variety of industries [81]. For example, DL is used in the agriculture industry in the growing period to detect ripe or unripe fruits and vegetables on the farm. In the medical field, DL enables doctors to identify dangerous tumors, broken body parts, or blocked vessel locations in the human body. In industries, DL can ensure that workers perform their tasks properly. One of the most important fields that benefits from DL is public transportation in societies. It is a great idea to figure out an optimal path on city roads. Path planning on roads can have positive effects on the quality of human daily life. In [82], path planning is often based on grid-based topologies. In this case, the SUMO map provides a grid-based environment that enables efficient navigation for a mobile robot or vehicle. This topology can also be utilized in a CNN, leading to optimized decision-making in dynamic path planning. The focus is on AI systems capable of identifying routes even in adverse weather conditions [21],[76]. It can save human lives directly or indirectly once an accident occurs or some services, such as an ambulance, a firefighter, and a police department, need to take action in public [83],[23].

In [84], underwater gliders use AI to explore the deep sea, investigate marine environments, and successfully complete their underwater missions. They rely on path planning to navigate from their starting point to their destination by selecting collision-free routes[85],[83],[82]. However, traditional path planning methods often result in failures. To address this matter, neural networks (NNs) were applied to detect optimal paths. DL methods help reduce travel time between start and end points [86],[87]. The authors proposed a Dual-Branch Deep Convolutional Neural Network for uncertain ocean current environments (Duo-CNN), which improves path planning by allowing obstacle avoidance [81], minimizing travel time and energy consumption, and improving overall navigation efficiency.

In [75], the authors benefited from a CNN model to enhance driving safety on the roads. They used two different types of data to feed CNN. The first data was the video or images that were recorded by a camera that was installed on the dashboard of the car. In addition, they have utilized the second input data that was gained from a sensor, which was applied to the vehicle tier to collect road information and send it to another CNN model, and these two CNNs send their data to a single CNN model to evaluate obstacles and detection of road objects. However, in our case study, two inputs were used; both inputs were sent to a single CNN at the same time, and decisions were taken based on the conditions of the two inputs for autonomous vehicles. In addition, In [85] mentioned that using conditions and policies for autonomous vehicles results in better results in the planning of the route. However, these works often focus on limited input types and do not integrate multiple environmental metrics simultaneously; this limits applicability to real-time autonomous path planning in complex urban scenarios, which our research addresses by combining CNN perception with SUMO-based reinforcement learning

The authors have presented in [88] a proposed method with the help of a combination of a hybrid method that consists of utilizing Long-Short-Term Memory (LSTM)-was named A Bidirectional

Long Short-Term Memory (BiLSTM)-and a CNN in order to predict routes on a map. BiLSTM networks are an extension of LSTM networks[85] in which two LSTMs are applied: one is applied to the input data (forward) and the other one is applied to the reverse of the input data (backward). However, they have applied four various points as the destinations in their study. However, we have deployed 10 edges as the starting and destination of our study to ensure that our proposed CNN model worked well. However, these methods were tested only on a small number of start and destination points; thus, this limits generalization. In contrast, our approach evaluates a larger set of edges in order to guarantee that the model behaves robustly across different urban layouts.

A CNN model and transformer has been used in [89], [90] to extract roads and building. The study aided can be used in an autonomous vehicle to identify roads on a map. They benefited from three data sets to evaluate their proposed model to extract the required features from the map [91]. The transform used in the study played a vital role in recognizing objects based on their overall shapes and distance relationship. Although CNN was depicted in the study to extract better local details such as edges, textures, and small features[23]. The combination of CNN and transformer enables their model to extract desired objects in detail[90]. In other words, they cover each other's result. Their research can also be used in the planning of the path. Although these models are effective for feature extraction, they do not optimize path planning metrics such as congestion, traffic lights, or travel time. Our research investigates the integration of perception outputs into a path optimization framework.

In experience [92], authors used a fine-tuned U-Net model. This convolutional neural network (CNN) architecture is designed for image segmentation tasks with the help of processing input images to assign class labels to each pixel. It selects road features from remote sensing data. The U-Net encoder-decoder structure, where the encoder extracts hierarchical features and the decoder reconstructs spatial details, enables it to capture both spatial hierarchies and fine image details. A modified feature space, meaning an altered representation of the input data for deeper model learning, boosts model performance and enables more accurate road network delineation. This method indicates how well CNN-based models—more especially, U-Net—perform in complicated geospatial image segmentation. Although accurate for segmentation, these models do not consider dynamic traffic conditions or multiple path constraints simultaneously. This gap is what motivated us to combine CNN-based perception with reinforcement learning in a simulated urban environment.

A study was done in [93],[94] that is to employ a CNN model that was combined with the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) approach to identify the lane line on the roads. CNN has extracted the characteristics of the lane line and DBSCAN was depicted to recognize the density and sparse noise data on the street. However, in the study, speed was not considered a key factor.

In paper[95], the traffic bottleneck has been introduced as the most challenging issue in urban areas. They have applied a CNN network to mitigate traffic with the help of CNN. The priority of traffic signs can have an efficient impact on traffic blockage. An autonomous vehicle that can detect traffic signs clearly leads to a lack of road congestion, as it prohibits some routes that have stop signs and some prevent signs. CNN was used to identify traffic signs and reduce traffic jams. One of the most important features of roads is the surface damage. It can cause vehicles to slow down and reducing speed leads to congestion. In [91], [22], the authors have used a CNN to investigate cracks and other problems on road surfaces. They have leveraged data set/sets to extract features,

which were then given to CNN to identify cracks and damage.

In summary, previous works have successfully applied deep learning to perception, road feature extraction, and trajectory prediction. However, most of them still lack the integration of multi-environmental metrics, real-time path optimization, and the robust evaluation in diverse urban conditions. The research proposed herein bridges these gaps by incorporating CNN-based perception with reinforcement learning in a SUMO environment to seek an optimized path with efficiency, safety, and sustainability for complex traffic conditions.

3.3 Research Gap

In recent years, traffic congestion has become a major public problem in metropolitan cities. To address this, several advanced technologies have been applied to roads. These technologies consist of object detection [21],[22], machine learning (ML), DL, and Reinforcement Learning (RL). In deep learning and especially in a CNN, it can learn features from raw data (image or text) without the need for manual teaching [22],[23]. However, there remains a gap in effectively leveraging multiple inputs within CNN models.

This limitation results in an inadequate depiction of actual road conditions, where a variety of contextual and environmental factors work together to influence route choices. Single-input CNNs are the mainstay of existing research, which limits their capacity to incorporate heterogeneous data sources like surface conditions, weather dynamics, road slope, and traffic density. Because of this, they are unable to accurately simulate the intricate relationships needed for trustworthy path optimization in dynamic settings. Therefore, by combining various input features into a single deep learning framework, this study attempts to explore how a multi-input CNN architecture can enhance decision-making in route planning. The goal of this study is to determine how well this architecture performs in choosing the optimal routes under various environmental constraints by putting it into practice and analyzing it in the SUMO simulation environment.

To address this, we propose a multi-input CNN architecture designed to improve path optimization. By processing multiple inputs concurrently, the model can make more informed decisions when selecting the optimal edge during route exploration on a sample map. Our case study uses SUMO, a simulated 2D traffic environment, to validate this approach. Unlike previous studies that relied on single-input models, our framework integrates diverse inputs, allowing the inclusion of critical factors such as slope, weather conditions, and other environmental variables that significantly influence road path planning.

3.4 Approach

One of the well-known technologies that have been used in recent years in different industries is the application of AI. AI consists of various sub-fields such as machine learning (ML), DL, and reinforcement learning (RL). DL is a kind of ML in which an agent takes action regarding an environment and interacts with it.

DL is also widely used in path planning. Vehicles and submarines can take advantage of the

DL to identify the best routes and destinations. Path planning is especially complex underwater, as submarines must detect obstacles and collect diverse information from their environment to determine the most accurate and efficient route [84].

Over the years, traffic congestion has increasingly become a major issue on roads in urban areas. In metropolitan cities, it is important for people to understand how to deal with this growing problem. Various technologies have been used to mitigate traffic jams on the streets. These technologies have also been adopted in autonomous vehicles, including electric autonomous vehicles (EAVs). The aim of this article is to address traffic congestion, reduce harmful emissions, and conserve both electric energy and fossil fuels.

This study adopts an experimental, simulation-based research design where routing decisions are learned and evaluated within the SUMO microscopic traffic simulator. The research uses a quantitative comparative approach, examining the performance of a CNN-based routing policy against baseline routing strategies under varying congestion and traffic-light densities. The objective is to determine whether a spatial-global dual-input model can improve routing efficiency in dynamic traffic conditions.

This work is an experimental, simulation-based study designed to evaluate whether a CNN-guided routing policy can (1) reduce average travel time, (2) reduce total distance traversed, and (3) reduce congestion-related performance degradations (e.g., low average speeds and stop-and-go behavior) in an EAV-like SUMO environment. We formulate the problem as a sequential decision task: at each step the model selects the next edge; the learning objective is to maximize a cumulative reward that balances travel time, distance, and penalties for congestion or dead-ends. The study tests two hypotheses: (H1) a dual-input CNN that fuses local spatial features (vehicle density and traffic-light counts) with a scalar speed factor outperforms single-input baselines on travel-time and congestion metrics, and (H2) reusing trained models across similar maps reduces training cost while retaining routing quality. The evaluation is comparative and quantitative, with ablation experiments to isolate the contributions of each input and component.

Experiments were conducted using the SUMO (Simulation of Urban Mobility) traffic simulator to generate microscopic traffic traces on a sample Bari map. The SUMO map (version and conversion details should be reported) was modified programmatically to create scenario variants (vehicle counts, traffic-light density). The RL experiments used Stable Baselines3 (SB3) as the RL framework and PyTorch/TensorFlow-backed CNN implementations for the policy network. Data inputs to the CNN were generated at simulation time steps as 64×64 two-channel grids (channel 1: normalized vehicle counts per edge; channel 2: traffic light counts per intersection lane) together with a normalized scalar speed input. All experiments were run on [insert hardware: e.g., Intel i7 / NVIDIA GTX 1080 Ti / 32GB RAM] (or state the cluster/GPU used).

The experiments use the Bari sample map in SUMO as the primary data source. Traffic states (vehicle counts, traffic-light positions, and speeds) are collected through the TraCI interface. Scenario variations are created programmatically by modifying the number of vehicles and increasing signal densities. The model is implemented in Python using TensorFlow/Keras and trained using Stable Baselines3 reinforcement-learning algorithms. All experiments are executed on mentioned

hardware. The Python Version 3.15 and using Spyder as a Python IDE software.

Each episode begins at a random origin edge and ends when the vehicle reaches the target destination. At every decision step, the agent observes a 64×64 grid containing two spatial channels (vehicle density and traffic-light count), along with a global speed factor representing overall congestion. The action space corresponds to the set of available outgoing edges at each intersection. A dual-input CNN processes the spatial grid and the speed scalar, producing a probability distribution over the next edge. SUMO executes the selected action, and the resulting travel time and speed determine the reward. Map variations include scenarios with 150 or 450 vehicles and increased numbers of traffic lights. Each configuration is repeated across multiple random seeds to ensure robustness.

A CNN, combined with a well-known reinforcement learning (RL) library called Stable Baselines3 (SB3), was utilized. In this approach, SB3 was used to collect traffic light information from a sample SUMO (Simulation of Urban MObility) map, representing a part of Bari, Italy, as shown in Figure 3.1. In Figure 3.1 (A) is the raw map that had one traffic light and no vehicle density; however, the number of traffic lights increased 9 times more in Figure 3.1 (B).

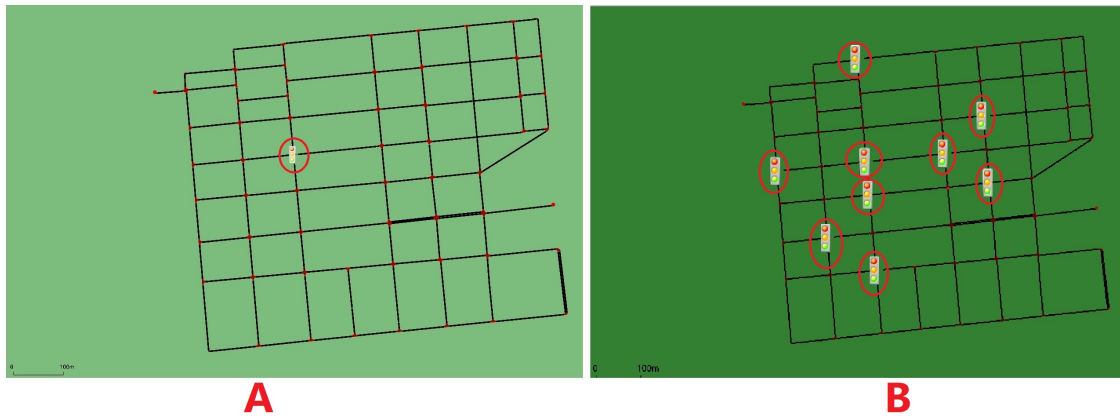


Figure 3.1: *Apart of Bari in Italy as the environment in the study-The SUMO sample map- Plot A before increasing the number of traffic light and Plot B is after enhancing the number of traffic lights*

The proposed model is depicted in this way as follows:

- **Step 1:** Add 150 vehicles and trips to the raw SUMO map. This is done because the map initially did not contain any traffic.
- **Step 2:** Increase the number of vehicles to 450 to simulate the scenario in a way that resembles a real-time environment. This helps illustrate the pros and cons of the proposed CNN model.
- **Step 3:** Increase the number of traffic lights on the map. Similarly to the previous step, the increase in traffic lights is considered an important factor for future road planning in large cities. The number of traffic lights on a road also leads to traffic congestion [96].
- **Step 4:** Build the CNN model, define the reward functions, and adjust the hyperparameters to ensure that the model produces effective results [88],[81],[94],[97]. This allows investigation of the optimal path between two randomly selected desired points (edges) on the map. The CNN model benefited from several important elements. These included how many cars were

on the road, the number of traffic lights, and the smallest number of edges in the final route [96]. It also looked at the reduced travel time and the shortest distance from the beginning to the end edge [82]. The model aimed to avoid loops and dead ends, and it considered the speed factor[23]. When checking paths, the system adds a penalty if the average speed drops below a set limit[98]. All these factors help the model learn to choose routes that are less busy and more efficient.

- **Step 5:** Create an optional feature that can be used to reuse trained data after the training process is finished for the future [98]. This is aimed at finding the correct path between two different randomly selected desired edges on the map.

SUMO was chosen for its high-fidelity microscopic simulation and ease of scenario manipulation, which is essential for studying traffic-light impacts and EAV behaviors. A CNN was selected to exploit the spatial structure in the 64×64 grid representation (local congestion patterns and intersection layouts), while the scalar speed input provides a complementary global congestion signal. Stable Baselines3 provides robust, tested RL implementations that speed development and ensure reproducible agent training. Combining CNN perception with an RL decision-making loop is appropriate for this sequential routing decision task because it enables learning from interaction (including delayed consequences of routing choices) rather than relying purely on handcrafted heuristics. Finally, ablation and reuse experiments are included to test generalization and cost-effectiveness.

We built a CNN model that comprises two main inputs. The first input is for vehicle density and traffic lights. It is important to note the factors that play a vital role on roads as channels in CNN. The second main input belongs to the speed factor as a major metric in public transportation. The first main input benefited from a 64x64 size grid 2-channel image for edges (streets on the map) along with an extra normalized scalar speed factor. The first channel represents a number of vehicles that exist on an edge (road or path). The second channel shows the traffic light count in each lane at the intersection. The proposed model channel is presented in Figure 3.2. An innovation was used using multiple inputs in the proposed CNN model. One input is the density of vehicles and traffic lights and the second input is the speed factor, which is related to the speed limitation of each edge. In CNN, all possible paths between two selected edges will be evaluated by some conditions that are set in the CNN [78]. The dual input fusion makes the proposed CNN smarter and more realistic for path prediction in real road networks [99]. It is important to note that the number of vehicles and traffic lights increased in most of the simulations. It was done to ensure in

connection with the proposed CNN model.

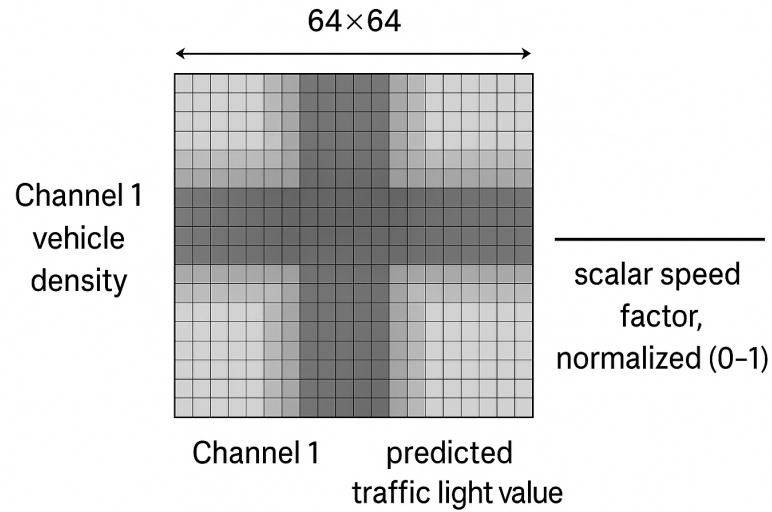


Figure 3.2: *The proposed CNN model grid channels*

The two channels allow the proposed model to learn the traffic patterns on the map [96]. In parallel, the speed scalar factor channel, which is a normalized value between 0 and 1, highlights the average vehicle speed relative to the lane speed limitation [98]. The speed factor provides valuable data on traffic flow [89]. This combination of spatial and scalar features allows the model to make informed path planning decisions based on both localized congestion and overall travel efficiency. The proposed schema of the CNN model is shown in Figures 3.3 and 3.4.

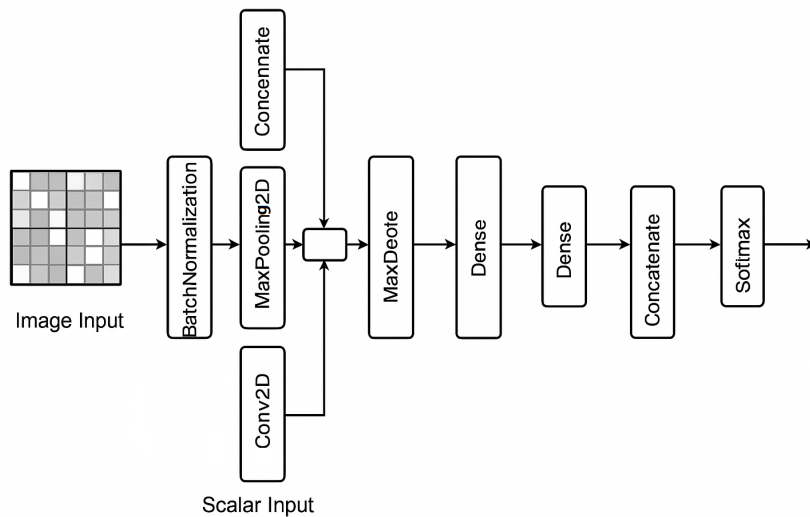


Figure 3.3: *Proposed CNN model Schema*

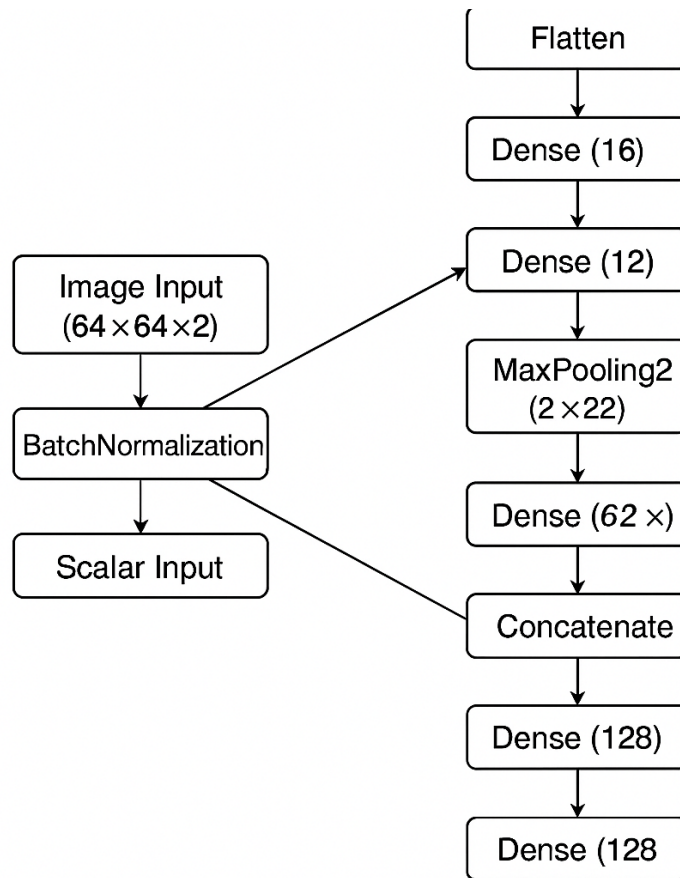


Figure 3.4: *The proposed CNN structure*

In Figure 3.3. First, batch-size normalization was used in CNN. It was deployed to stabilize and accelerate training by mitigating the internal covariate shift. It is responsible to ensure that the model trains smoothly and prevents exploding/vanishing gradients, especially with deeper CNNs.

Next, a 2D convolution layer (Conv2D) was utilized in the following way: Conv2D(32, (3,3)). Conv2D is a convolutional layer with 32 filters and 3×3 kernels. In the CNN model, Conv2D has 32 filters and each filter can learn to identify various features such as edges, shapes, and textures. The filter size(kernel) is a 3×3 grid, like a small square that moves across the image [100]. Generally, Conv2D filters are beneficial for capturing higher-level abstract features by working in combination with other layers. In this context, it is used to learn more complex patterns in traffic flows and intersections [98], [96].

In Conv2D, a ReLU (Rectified Linear Unit) function is applied immediately after the convolution has been used. The activation function is responsible for passing information from one neuron to the next layer [90]. In this way, each neuron acts as a small decision-maker. It is an important part of a CNN model, which is why it must take action on whether to send signals forward or not. In addition, it can learn complicated patterns. ReLU is one of the most popular and widely used activation functions in DL and CNNs[88],[91], [100],[87]. In ReLU, an approach is adopted to handle a value: If the value is greater than zero, it keeps the value; otherwise, it modifies the value to zero. The ReLU outperforms older activation functions like Sigmoid by enabling the model to

learn faster.

In Conv2D, an L2 regularization was applied to the filter weights. It allows Conv2D to prevent over-fitting. In other words, it helps a model not to learn too much noise from the training data. It also attempts to decrease weight to small and adds penalty for large weights.

In the upcoming layer, MaxPooling2D was applied, which is a downsampling layer that selects the maximum value in each 2×2 region. It helps reduce spatial dimensions while keeping important features. Moreover, it used to enhance model speed and mitigate over-fitting. This layer looks at a small 2×2 square (pooling window) in the image or the feature map. From every 2×2 square, it keeps only the highest (maximum) value. Then it moves forward (usually by 2 steps) and does the same thing. So, it reduces the image size by half in both dimension ,width and height. Clarify this layer in detail by giving an example. The part of the image or map is adopted as follows. Imagine that this is part of a feature map (4×4):

$$\begin{bmatrix} 1 & 3 & 2 & 4 \\ 5 & 6 & 1 & 2 \\ 7 & 8 & 3 & 0 \\ 2 & 4 & 9 & 5 \end{bmatrix}$$

Once we applied MaxPooling2D(pool-size=(2, 2)). In other words, a square matrix of size 2x2 attempts to cover the prior 4x4 matrix.

In this way, a 2x2 square is first placed in the top left corner of the main matrix; that result is presented as follows.

$$\text{Block 1: } \begin{bmatrix} 1 & 3 \\ 5 & 6 \end{bmatrix} \Rightarrow \max = 6$$

Then the 2x2 square moves right, and the result for the second block is displayed as follows:

$$\text{Block 2: } \begin{bmatrix} 2 & 4 \\ 1 & 2 \end{bmatrix} \Rightarrow \max = 4$$

The third block is shown in the following schema:

$$\text{Block 3: } \begin{bmatrix} 7 & 8 \\ 2 & 4 \end{bmatrix} \Rightarrow \max = 8$$

and the final block 4 is presented as follows :

$$\text{Block 4: } \begin{bmatrix} 3 & 0 \\ 9 & 5 \end{bmatrix} \Rightarrow \max = 9$$

In the end, and in the findings, we have the following 2×2 matrix, which is a combination of the previous matrix blocks, as follows:

$$\text{Output: } \begin{bmatrix} 6 & 4 \\ 8 & 9 \end{bmatrix}$$

Finally, our feature map went from 4×4 to 2×2 — it became much smaller and kept only important values. This means that, by using this layer, our image size becomes half and the result

will contain the important and strong features of the original 4×4 image or map.

MaxPooling2D, a technique to reduce the size of the image, keeps the most important information. This method starts with a 4×4 grid of numbers showing pixel brightness or feature strength. A 2×2 box marked by orange lines, moves across the grid in four steps without overlap. It picks out the biggest number in each area marked by a red dash around the main 4×4 matrix. For example, it goes for 8 from [8, 3, 4, 1] in the top-left corner. This takes place in all four parts, resulting in a smaller 2×2 grid with the top values [8, 9, 8, 7] from each section of the main 4×4 matrix [85]. The MaxPooling2D approach is presented in upcoming Figures 3.5, 3.6, 3.7, 3.8, 3.9 as follows:

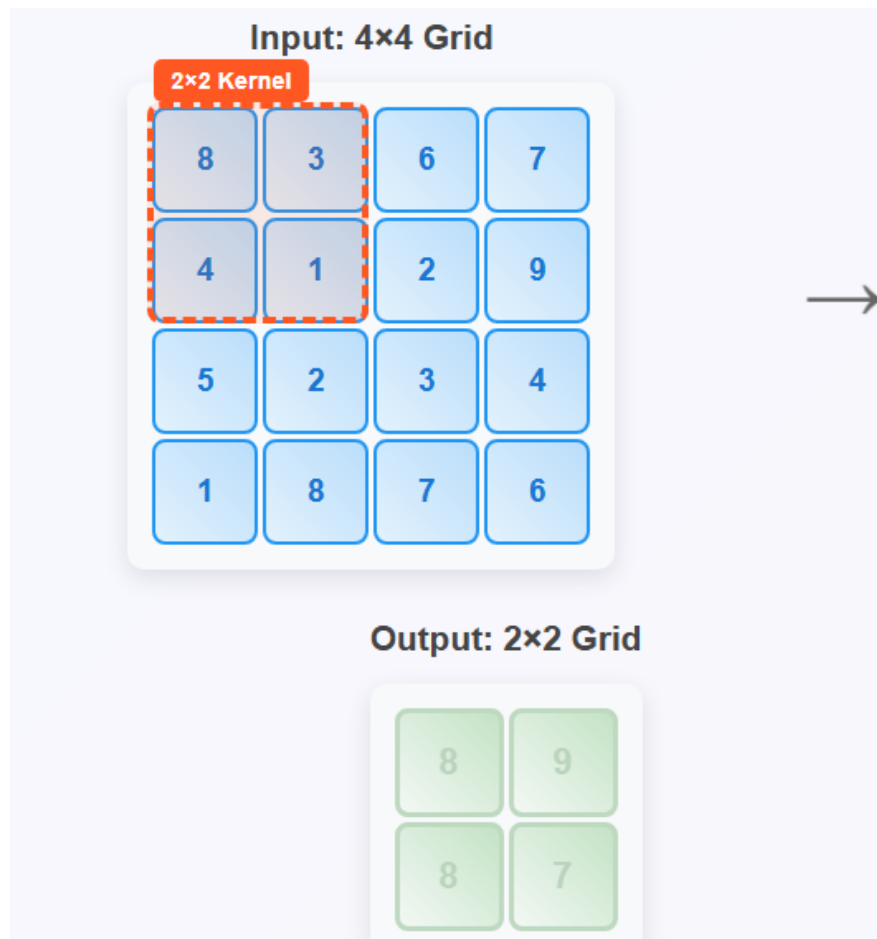


Figure 3.5: How the Kernel 2×2 works on an image

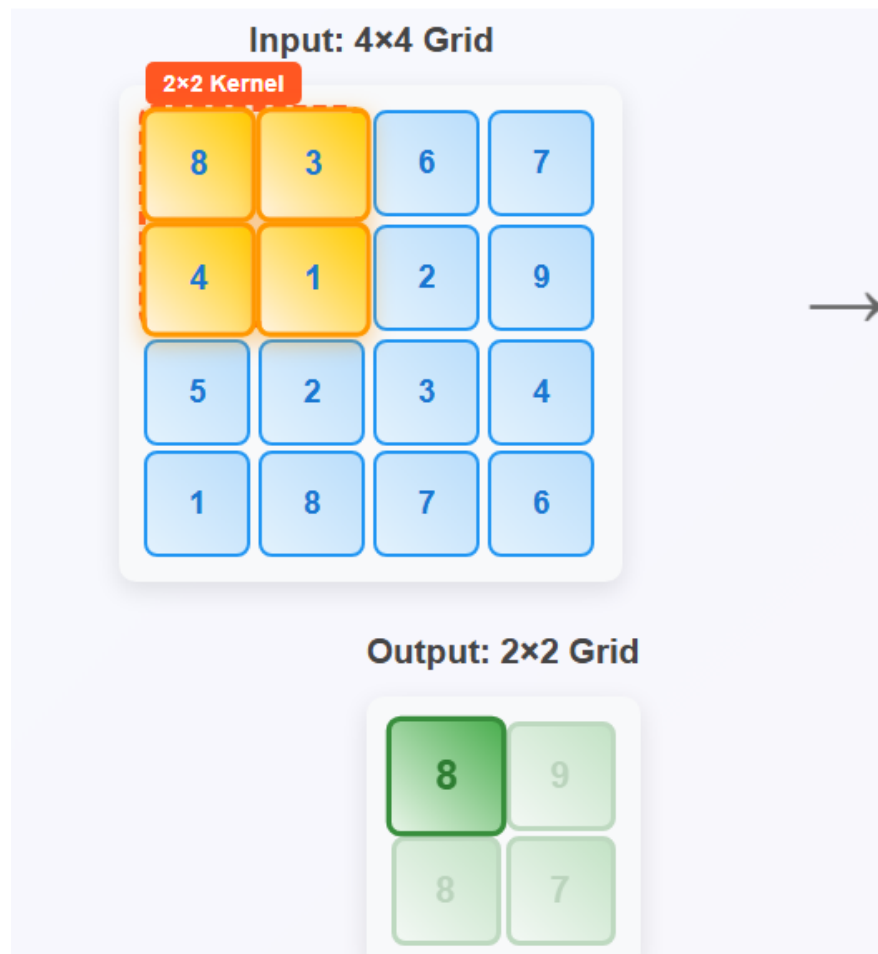


Figure 3.6: Kernel 2x2 performance- Step 01

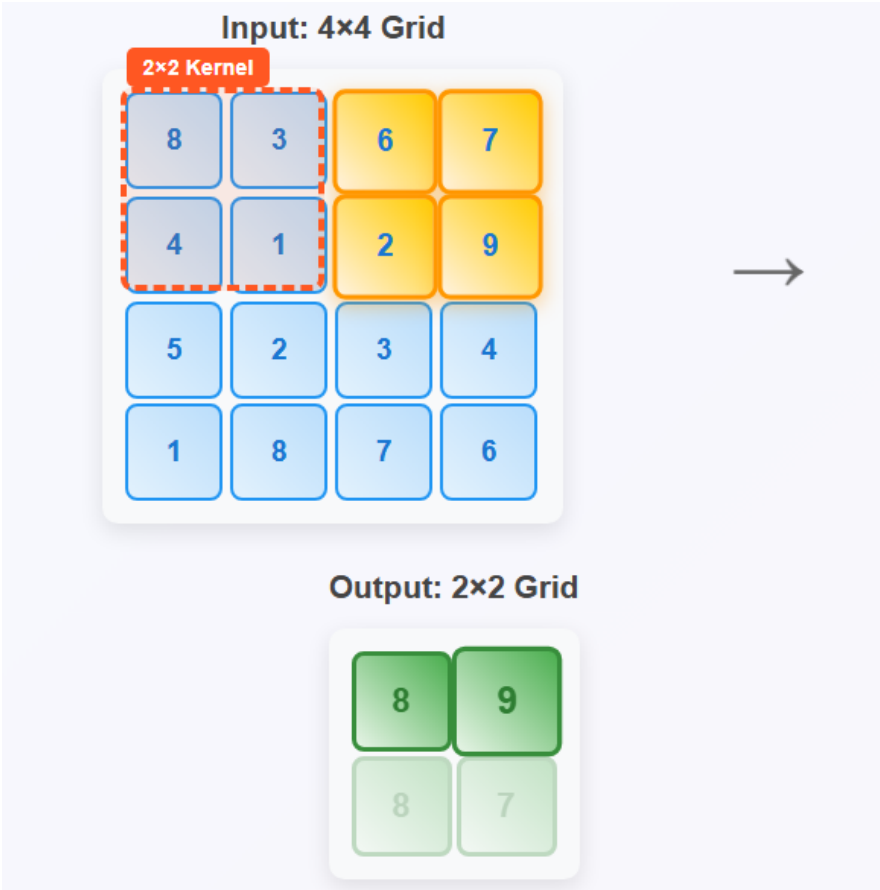


Figure 3.7: Kernel 2x2 performance- Step 02

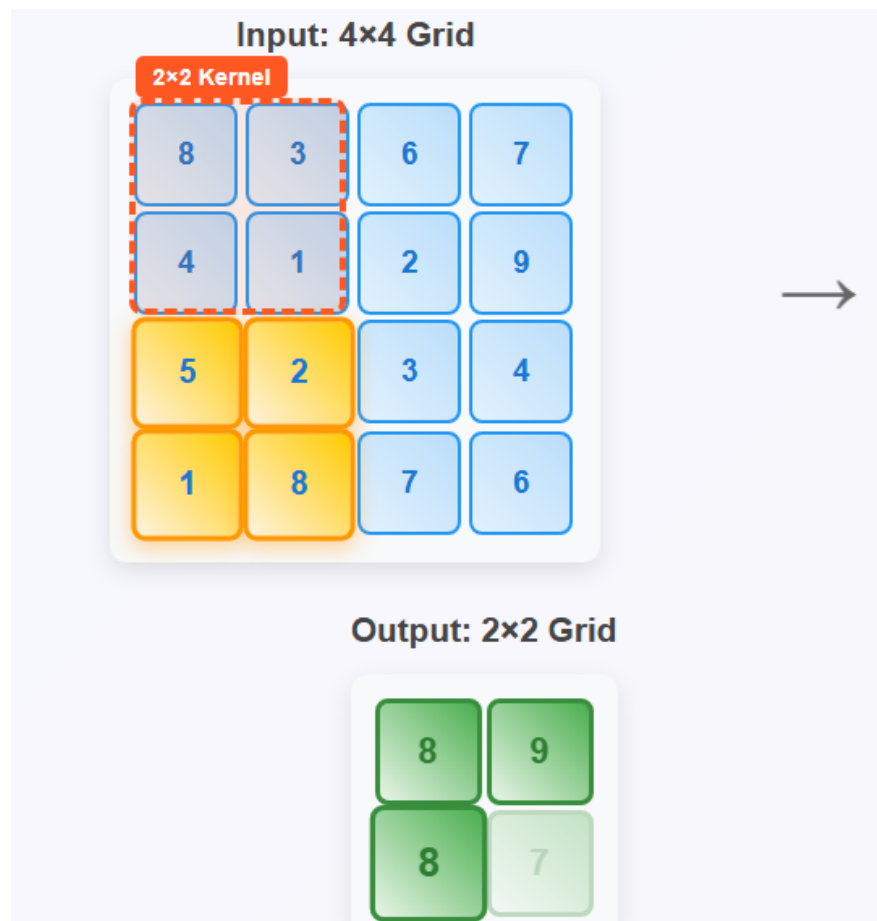


Figure 3.8: Kernel 2x2 performance- Step 03

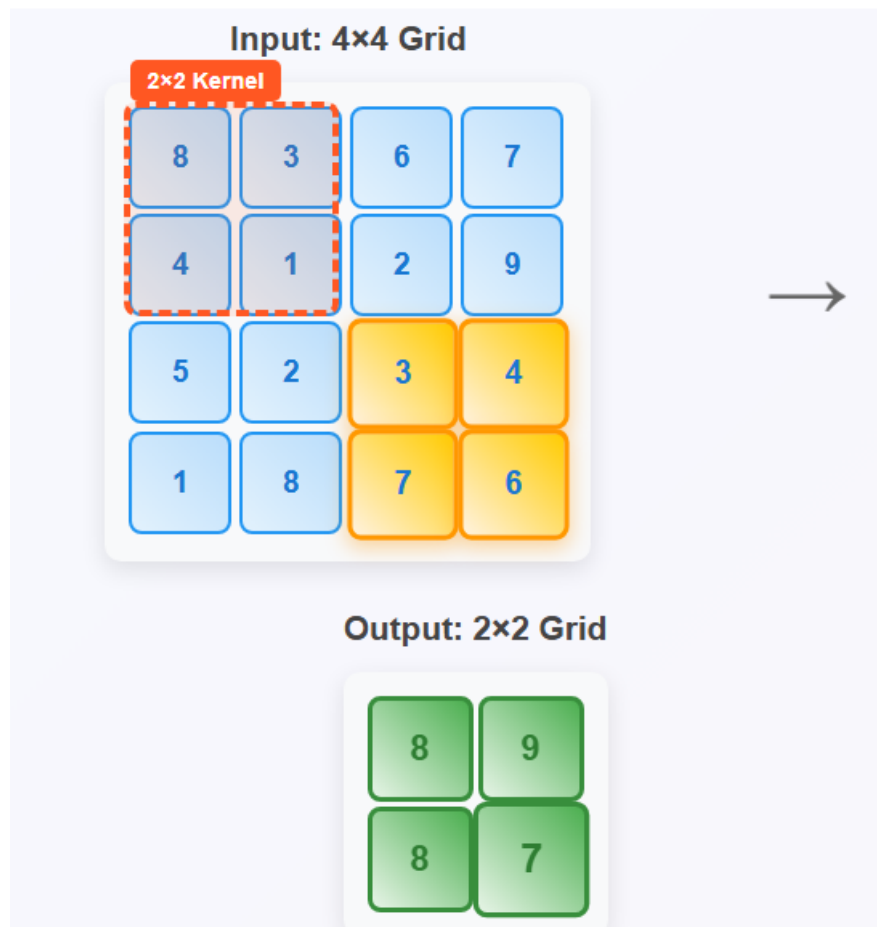


Figure 3.9: Kernel 2x2 performance– Step 04

MaxPooling2D, a technique to reduce the size of the image, keeps the most important information. This method starts with a 4x4 grid of numbers showing pixel brightness or feature strength. A 2x2 box marked by orange lines, moves across the grid in four steps without overlap. It picks out the biggest number in each area marked by a red dash around the main 4x4 matrix. For example, it goes for 8 from [8, 3, 4, 1] in the top-left corner. This takes place in all four parts resulting in a smaller 2x2 grid with the top values [8, 9, 8, 7] from each section of the main 4x4 matrix. The new matrix is called the feature map [83], [100].

The proposed CNN consists of a layer is called flatten. This flattening converts 2D feature(or multiples) maps to 1D vectors. It changes multi-dimensional data into a single-dimensional vector. This process lets the dense (connected) layers work with 1D inputs instead of matrices. The flattened layer plays as a bridge between the convolutional feature extractor and upcoming the decision-making layers [100]. An example of flattening is illustrated in Figure 3.10.

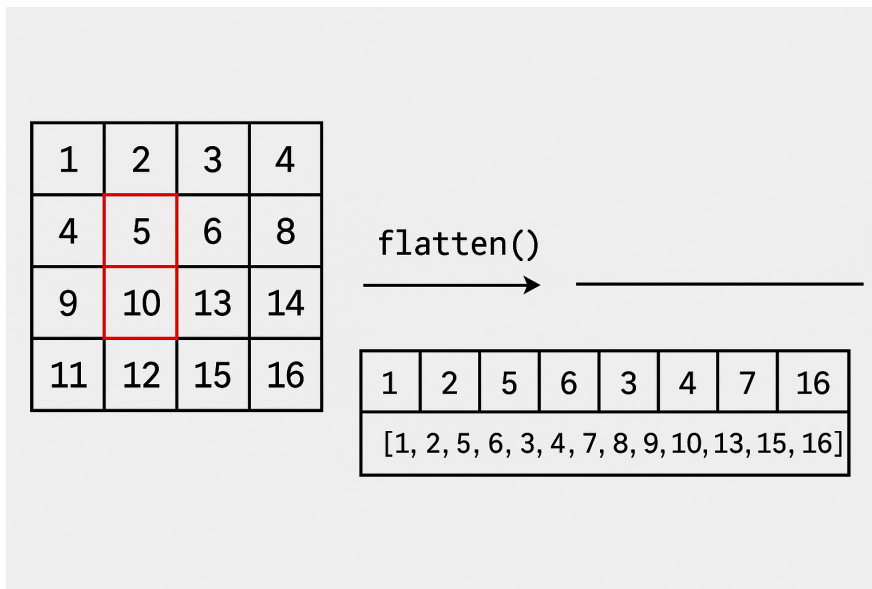


Figure 3.10: An example of Flatten performance on an image

The other layer used in the proposed CNN is a dense layer, also known as a fully connected layer. Fully connected means that each neuron in this layer is directly connected to all neurons in the previous layer [100]. The performance of the dense layer is represented in Figure 3.11.

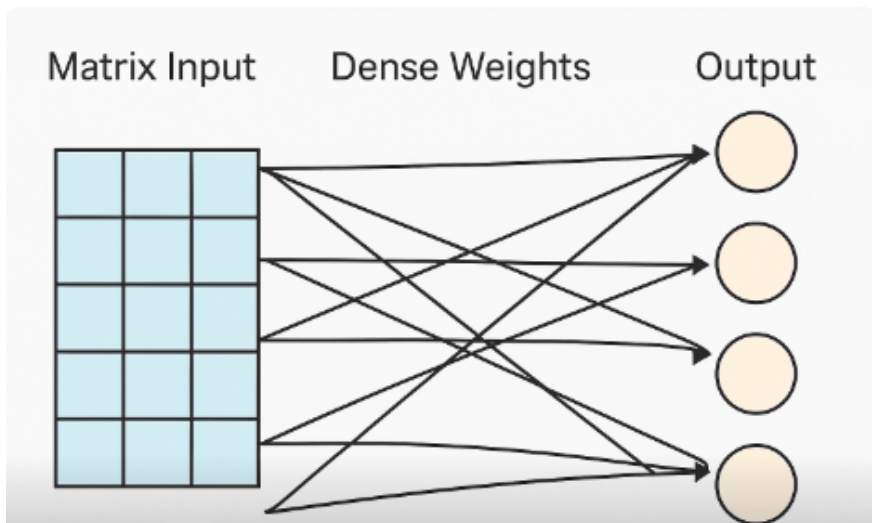


Figure 3.11: The Dense layer performance

It is commonly utilized after the flattened layer in the CNN. In the proposed CNN, it is employed in the following way: Dense(128, activation= ReLU). The number 128 in this layer refers to the fact that the layer contains 128 neurons (also called units or nodes). The dense layer receives input from the previous layer and then computes a weighted sum plus bias. Each neuron in this dense layer subsequently employs the ReLU (Rectified Linear Unit) activation function. After the convolutional and pooling layers that extract measurable features from inputs (e.g. patterns, edges, traffic zones)[89], [98],[96], the dense layer takes the features and utilizes them to make predictions. It connects each input feature to every output neuron and benefits from a weight matrix to learn

complex, non-linear combinations of the features [100].

$$\text{output} = \text{activation}(W \cdot x + b)$$

where:

W = weights

x = input vector (flattened CNN output)

b = bias

activation = usually ReLU, sigmoid, or softmax

The weight tells the neuron how important each input number is. Also, it is like turning the volume up or down for each input. Every connection between an input and a neuron has a weight. For instance, imagine that we have this input: [traffic (vehicle) density = 0.9, signal = 0.3]. Weight for traffic = -0.6 (we want to avoid high traffic) shows the CNN must prevent selecting this edge as it is crowded. The x indicates input/inputs that in the proposed CNN model we have two inputs. The first input belongs to vehicles and traffic lights and the second input is connected to speed factor on roads map. Moreover, the weight for the signal is -0.2, and we also avoid traffic lights, but not as strongly. It illustrates the edge has a traffic light, and its schedule matches to be selected in our final optimal path. So the neuron learns:

$$\begin{aligned} \text{Output} &= (0.9 \times -0.6) + (0.3 \times -0.2) \\ &= -0.54 - 0.06 \\ &= -0.60 \end{aligned}$$

This value (-0.60) is the raw output of a neuron before activation that has been calculated. It indicates the confidence or score that this neuron is connected to a specific path or an output based on the inputs.

The bias shifts the output up or down — like setting the starting point. It helps the model take better actions even when the inputs are zero. It works just like the intercept in the equation of a line:

$$y = mx + b$$

You can imagine that the bias is +0.2 in the previous example. Then, the final output will be:

$$\text{Final Output} = -0.60 + 0.2 = -0.40$$

The lower the value, the less favorable the model considers this option. This output will go into an activation function, like ReLU or softmax, which decides what happens next [87]. The value -0.60 will contribute to a probability distribution. If other paths have higher scores (e.g., 1.2, 0.7), this one gets a very low probability. In the previous example, final output was -0.40 which means that the current input combination (e.g., heavy traffic, many lights) is not a good match for what that neuron is looking for.

In machine learning, there is a term called batch. It simply means that we input multiple examples at once. Each individual example (such as an image or data point) has its own distinct features. For instance, if we use a 28x28 grayscale image and flatten it, we get the following.

$$28 \times 28 = 784 \text{ pixels} \rightarrow \text{a vector of 784 numbers}$$

The number 784 represents the number of features for one image—meaning a 1D array with 784 values. If we input 32 images at a time, the input shape becomes (32, 784). Another important parameter is the batch size, which indicates the number of input items [96]. These input items can be images, samples, or data points. The dense layer takes this input (e.g., 784 values from a flattened 28x28 image), applies weights and biases, and then computes values for the 128 neurons. After that, each of the 128 outputs is passed through the ReLU function. The result is a vector of 128 elements that is passed to the next layer.

In the next layer, our second input was added separately in the proposed CNN to indicate the speed factor, which is related to the edge speed limitation resulting from factors in the vehicle and traffic lights. In the input, the average speed and the maximum speed can be evaluated and reflected in the proposed CNN model. It allows the CNN model to consider the average and maximum speed limitations during training and when selecting edges. It is essential to note that traffic speed is a global congestion indicator; an area with low density typically has a low average speed. The speed feature is used as a separate numerical metric. In addition, the speed factor not only focuses on vehicle density, but also controls the speed of traffic flow. Therefore, every low speed edge leads to a traffic jam. It enables the model to make a good decision in selecting an edge as a candidate for the optimal path, which is our goal [101].

One of the major layers that was employed in the CNN is Concatenate layer. It is responsible for merging multiple layers to one unique input. As can be seen, as follows:

```
combined = Concatenate()([x, y])
combined = Dense(128, activation='relu')(combined)
```

The "x" and "y" characters are the two inputs utilized in the CNN. The x feature originates from the input of the image, which is related to the number of cars and traffic lights. CNN extracts spatial patterns (such as crowded areas and intersections) for the "x" metric. The "y" feature belongs to the speed input as the second input in the proposed CNN model. It illustrates the model of how fast or slow traffic is moving. The concatenate layer puts the "x" and "y" features side by side in one longer vector. Then, the CNN model can learn both spatial (the first input) and global traffic conditions (the second input) at the same time. Spatial factors refer to features that have a position on the map, where my map image is a 64×64 grid. It consists of vehicle density (where vehicles are heavy or light) and traffic light counts (where intersections are). Traffic lights can also have negative effects on traffic congestion[95]. CNN uses convolution filters to detect patterns in space, such as congestion clusters or signal-heavy areas. Global traffic conditions are the input of the scalar speed (a single value between 0 and 1). It represents how quickly traffic is moving on average over a given edge. It shows the road status regarding whether it is flowing or jammed. The concatenate step is shown in Figure 3.12.

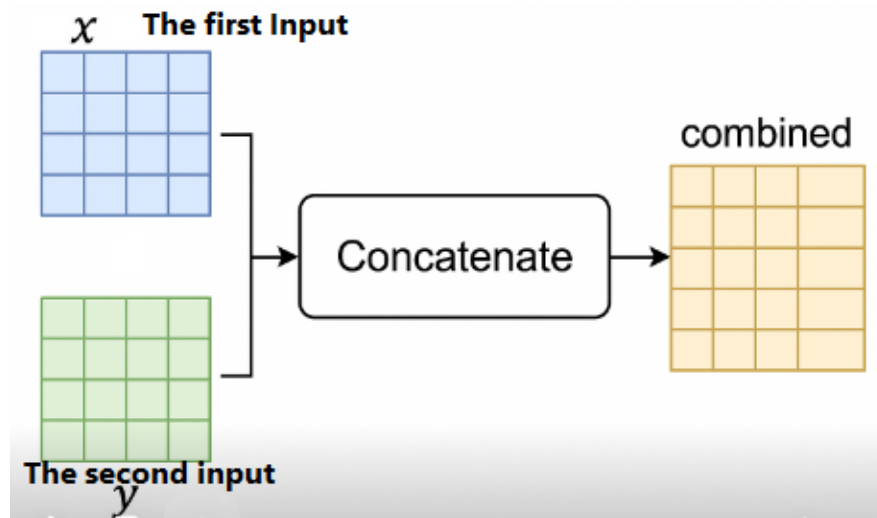


Figure 3.12: *The concatenate process*

After the concatenate layer, the output is the 1D vectors from the spatial and scalar inputs. The combined vector is sent through one or more Dense layers (e.g., Dense(128, activation='relu')) to learn complex relationships between map features (vehicle density, traffic lights) and traffic flow (speed).

The final Dense(n-edges, activation='softmax') layer output is a probability estimation over all possible next edges in the SUMO map[100],[87]. Where every output neuron is connected to a specific edge (road segment). The edge with the highest probability is selected as the next step in the optimal path. During training, the model learns to predict paths that minimize travel time, avoid congestion, and reduce signal delays by comparing its predicted edge to the correct next edge in recomputed optimal paths (labeled using reward scoring [81]). The learning process aims for the CNN to gradually adjust its weights. So it consistently favors efficient, low-traffic, and high-speed routes on the SUMO map.

The softmax activation function converts raw output values (logits) into probabilities that sum to 1, making it ideal for multi-class classification tasks. It illustrates the most likely class (e.g., the best next edge in routing) by increasing the largest value while removing the others. Advantages include interpretability, compatibility with cross-entropy loss, and its ability to model mutually exclusive outcomes in a clear, probabilistic way. Actually, the softmax takes a list of numbers (like scores) and turns them into probabilities, values between 0 and 1 that all add up to 1. For instance, imagine that the model gives 3 scores for 3 possible paths:

Raw output from Dense layer: [2.0, 1.0, 0.1]

and, when the softmax applies, it will convert in to :

Softmax output: [0.70, 0.25, 0.05]

It represents the first road that has the 70% (0.70) chance of being selected as the best edge on the optimal path, and so on. The softmax process is applied to the CNN model results for all possible edges in the SUMO map at each decision step. Each output neuron in the final density layer represents one possible edge (road segment) to which the vehicle can move next. The model gives a score for each edge based on current inputs (image plus speed). The model picks the edge with the highest probability (e.g., 0.70) as the next optimal step. This process is run step by step during path generation. At each junction (decision point), the CNN:

1. Looks at traffic (via image and speed);
2. Predicts probabilities for all possible next edges;
3. Selects the most likely (best) one.

The softmax is applied to all possible edges at each step, helping the model choose the next best edge repeatedly — until it reaches the destination.

Performance is evaluated using average travel time, total distance, average speed, queue length, and route efficiency relative to the shortest path. Results are averaged over multiple episodes and seeds. Comparative plots and statistical tests are used to evaluate differences between the proposed model and baseline routing approaches. The contribution of individual features (traffic light, speed factor, and regularization components) is tested by model modification.

These methods were selected because SUMO provides accurate microscopic traffic simulation, enabling controlled manipulation of congestion and signal patterns. A CNN architecture is appropriate for this problem because traffic states exhibit strong spatial structure, which convolution layers capture efficiently. The global speed factor complements the spatial features by providing a holistic measure of congestion. Reinforcement learning is used because routing decisions have delayed consequences, and RL optimizes long-term travel efficiency rather than immediate gains.

Together, these methods logically address the research problem by combining spatial reasoning, global state awareness, and long-term optimization within realistic traffic scenarios.

3.5 Findings and Implications

In this section, the proposed model comprises two different approaches. In the first approach, we excluded the speed factor from CNN. CNN used several hyperparameters, including two model optimizers (Adam and AdamW), a modified learning rate (LR) [102], a changed patient value, and a variable epoch number [88],[81], [103],[97]. This setup enables us to determine which configuration best suits the proposed CNN model. During this phase, we did not include the speed factor in our simulations, as we aimed to investigate the optimal path between two randomly selected desired edges on the sample SUMO map.

In the second approach, once we identified the best configuration for CNN, we used the speed factor as the second input in our simulations, just as in the previous approach, to determine the optimal path between two randomly chosen desired edges on the map.

It is important to note that in the two previous approaches, various random start and end edges were employed in the CNN model. For instance, in a simulation we used E15 as the first edge and E7 as the end edge in a simulation. Afterwards, we changed the two random desired edges as the initial and destination edges, for example, E10 as the beginning edge and Edge E14 as the final edge on the map.

3.5.1 Modified Hyperparameters Without Speed Factor

In the first attempt as the first approach, we first applied the default CNN model configuration and benefited from Adam optimizer to investigate the optimal path on the map displayed in Table 3.1 as follows:

Table 3.1: *The First Configuration Parameters for CNN-Based Traffic Routing*

Parameter	Value	Description
TRAIN_EPOCHS	800	Epochs for CNN training.
Optimizer	Adam	Optimizer for training in the CNN training.
TRAIN_SIM_STEPS	1000	Training simulation steps.
TEST_SIM_STEPS	300	Testing simulation steps.
MODEL_CHOICE	original	CNN type: deep (original) vs fast (lightweight).
SHOW_PLOT	True	Enable training plot display.
early_stop.patience	40	Patience epochs before early stopping.
reduce_lr.factor	0.5	LR reduction factor on plateau.
reduce_lr.patience	10	Epochs of no improvement before LR reduction.

3.5.2 Parameter Description

The configuration includes 800 epochs for model training to allow deep convergence. A total of 1000 simulation steps were used to collect training data and 300 for testing, offering a rich dataset for generalization. The original model architecture was selected for better performance despite the longer training time. The system also enables visualization of the training behavior (SHOW_PLOT=True). Early stopping is employed with a patience of 40 epochs to avoid over-fitting. This means that if the CNN result does not improve, this option will interrupt the simulation to avoid energy and time waste. The learning rate is reduced by a factor of 0.5 if validation loss does not improve over 10 epochs, helping to fine-tune the optimization near convergence.

In CNN, the learning rate is a number that indicates how much time the model takes to update its weights during training. It is one of the main parameters for CNN; if set too high, the model skips the solution and is considered unstable. If defined too low, the training will be very slow and might get stuck [102]. The epochs mean that the model initially takes small steps, which helps it adjust the weights better and get closer to the best result by the end of training. The outcome in the previous configuration is represented in Figure 3.13. The outcome is to search for the optimal path between Edges E7 and E10. As can be seen, the epoch was set to 800; however, at epoch number 250, the training was interrupted by the avoidance of the over-fitting function, as the model realized

the result would not improve and therefore stopped the training procedure at the 250th epoch.

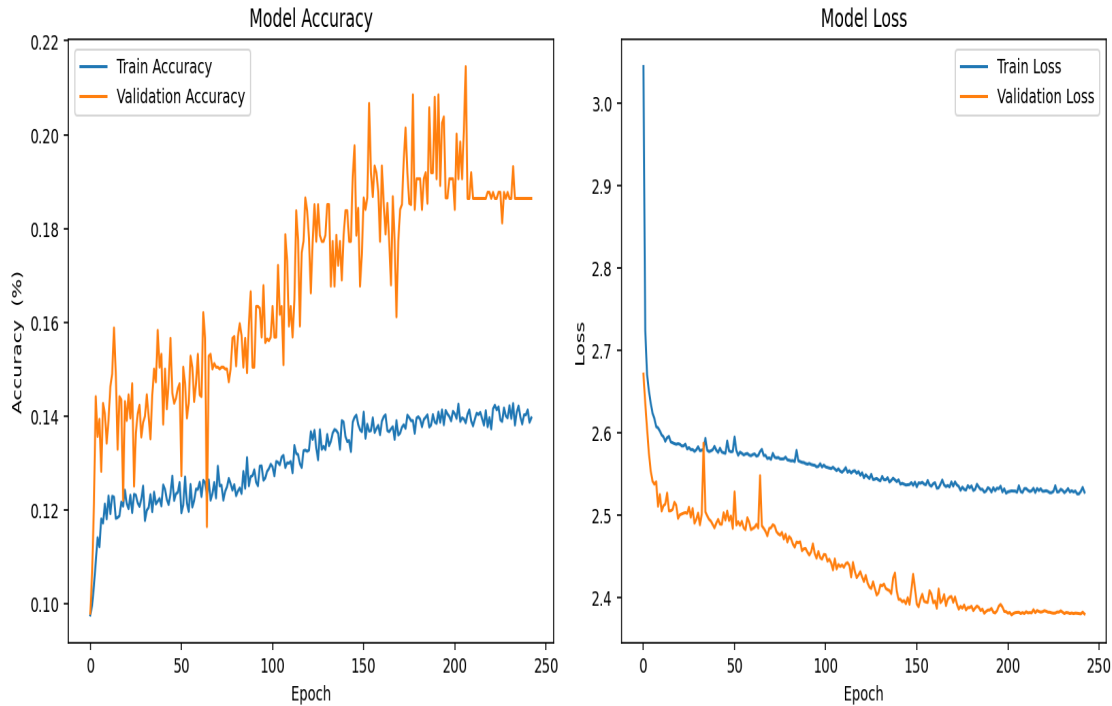


Figure 3.13: The *First* CNN outcome for searching an optimal path between edges $E7$ and $E10$

It can be observed that, , the training accuracy and the validation accuracy metrics increased significantly. The figure shows an improvement in training accuracy from about 10% to about 14.5% and in validation accuracy from 12.0% to about 20.5%, respectively.

On the other hand, the second plot is displayed on the right side of the figure, where the train loss and validation loss noticeably reduced from epoch number 50 onward. The SUMO map has 300 edges. A random approach would give us an accuracy of 1 in 300, which is approximately 0.33%. But the CNN model we suggest is 50 times better than just guessing. We know this because its validation accuracy outperformed its training accuracy. You can see all these results in Table 3.2.

In fact, we have 300 edges on the SUMO map. If we used a random technique, the accuracy percentage would be 1 out of 300, which is approximately 0.33%. However, the validation accuracy of the proposed CNN model indicates that it performs 50 times better than the random method, as the validation accuracy is greater than the training accuracy. The details of these findings can be

found in Table 3.2.

Metric	Value / Trend	Interpretation
Training Accuracy	~ 14.5%	Much better than random; learning well
Validation Accuracy	~ 18.5%	Also better than random; not overfitting
Training Loss	↓ 15.9%	Steady improvement
Validation Loss	↓ 10.2%	Generalizing fairly well
Epochs	250 of 800	Still room for improvement
Total Edge Complexity	300 edges	Accuracy level is reasonable
Random Guessing Accuracy	~ 0.33%	Baseline; model performs much better
Overfitting	No	Validation loss and accuracy are reasonable
Underfitting	Low	Model shows clear learning patterns

Table 3.2: The *First* CNN performance summary for optimal path prediction on a 300-edge map between edges E7 and E10

In the second simulation, we selected the two same edges on the map. Using the same method as in the previous simulation, we searched for an optimum path between these two edges. To investigate this optimal path, we modified the optimizer and several hyperparameters to improve CNN. The hyperparameters used are shown in detail in Table 3.3. We increased the patient stop epoch from 40 to 60, so the CNN will interrupt later. The reduced learning rate increases by 0.1 (from 0.5 to 0.1). The outcome is related to the configuration, is written in detail in the table, and is reflected in Figure3.14.

Table 3.3: The *Second* Configuration Parameters for CNN-Based Traffic Routing for E7 to E10

Parameter	Value	Description
TRAIN_EPOCHS	800	Epochs for CNN training.
Optimizer	AdamW	Optimizer for training in the CNN training.
TRAIN_SIM_STEPS	1000	Training simulation steps.
TEST_SIM_STEPS	300	Testing simulation steps.
MODEL_CHOICE	original	CNN type: deep (original) vs fast (lightweight).
SHOW_PLOT	True	Enable training plot display.
early_stop.patience	60	Patience epochs before early stopping.
reduce_lr.factor	0.6	LR reduction factor on plateau.
reduce_lr.patience	20	Epochs of no improvement before LR reduction.

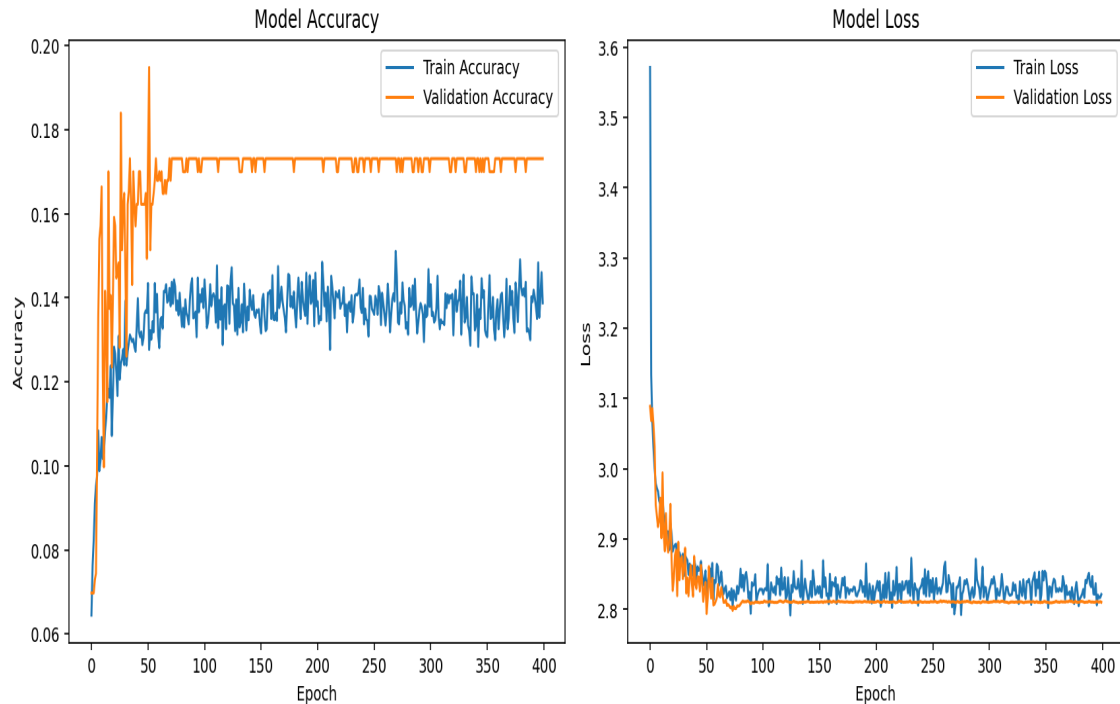


Figure 3.14: The *Second* outcome in the same direction from E7 to E10 by AdamW optimizer and several changes in hyperparameters

It is major to note that the CNN used the AdamW optimizer in this simulation compared to the Adam optimizer used in the previous simulation. The patient stop threshold increased to 60. In some cases, changing it leads to an improved outcome.

Learning rate scheduling is one of the most impactful hyperparameters for CNN performance. Due to the change, it can have positive effects on the outcome in the CNN. High LR early helps the model to explore the loss landscape broadly. The reduced learning rate enables fine-tuning and stabilization to more optimal solutions. It can mitigate gradient noise in later training phases and smooth convergence to minimum loss values. It can extract all useful patterns for input / outputs [81], [98]. It will increase the time to extract patterns from the input[89]. So, high patience can unlock better performance.

However, as can be seen, the final validation accuracy is approximately 17.2%, which is lower than that of the previous method, which was approximately 18.5%. In this simulation, it is not time and energy efficient at all, as it used 37% more iterations than the previous simulation, which was 250 epochs. In other words, the previous simulation reached the result faster than this simulation. It just had a slightly better validation loss, which was approximately 2.84% compared to the last simulation, which was approximately 2.39%. This simulation model stopped improving when it could have learned more. As a result, this simulation adopted a slightly different configuration from the previous simulation, but the result was not efficient. In all metrics, the lines plateaued after specific epochs. For example, the validation loss plateaus about 2.84% early and remains flat. It

shows that the Adam optimizer in the proposed CNN model outperforms the AdamW optimizer.

Metric	Value / Trend	Interpretation
Training Accuracy	~ 13.8%	Slightly better than random; some learning
Validation Accuracy	~ 17.3%	Better than training; generalizes decently
Training Loss	↓ 21.2%	Noticeable steady improvement
Validation Loss	↓ 9.1%	Slight improvement; no sign of overfitting
Epochs	400 of 400	Full training complete
Total Edge Complexity	300 edges	Accuracy level is fair given the size
Random Guessing Accuracy	~ 0.33%	Baseline; model performs much better
Overfitting	No	Validation metrics are stable
Underfitting	Low	Accuracy low; suggests limited capacity

Table 3.4: The *Second* CNN performance summary for optimal path prediction on a 300-edge map between edges E7 and E10

The CNN model faces under-fitting, as the figure indicates the low and flat training accuracy even after many epochs. In addition, training and validation losses are steady at relatively high values. There is no large space between training and validation accuracy/loss, which rules out overfitting but shows the model capacity is limited. Accuracy is far from optimal (close to random guessing baseline) despite stable losses.

In the next try, we selected two other desired random edges on the map, E7 and E10. The CNN model runs a training to find the optimal path for those edges. The CNN model was applied to the upcoming configuration which is displayed in Table 3.5 as follows:

Table 3.5: The *Third* Configuration Parameters for CNN-Based Traffic Routing For E0 to E14

Parameter	Value	Description
TRAIN_EPOCHS	200	Epochs for CNN training.
Optimizer	Adam	Optimizer for training in the CNN training.
TRAIN_SIM_STEPS	250	Training simulation steps.
TEST_SIM_STEPS	80	Testing simulation steps.
MODEL_CHOICE	original	CNN type: deep (original) vs fast (lightweight).
SHOW_PLOT	True	Enable training plot display.
early_stop.patience	100	Patience epochs before early stopping.
reduce_lr.factor	0.6	LR reduction factor on plateau.
reduce_lr.patience	5	Epochs of no improvement before LR reduction.

As shown, we decrease epochs, training simulation step(TRAIN-SIM-STEPS), while we

increase the early stop patient threshold to 100 and reduce the learning factor(reduce-lr factor to 0.6). The training simulation step is one forward and backward pass through a single batch of data. Also, an epoch is a complete pass through the entire training dataset in the CNN model. The outcome is highlighted in Figure 3.15. The threshold mechanism stopped training in 180 epochs in this simulation and training, as the training did not show improvement.

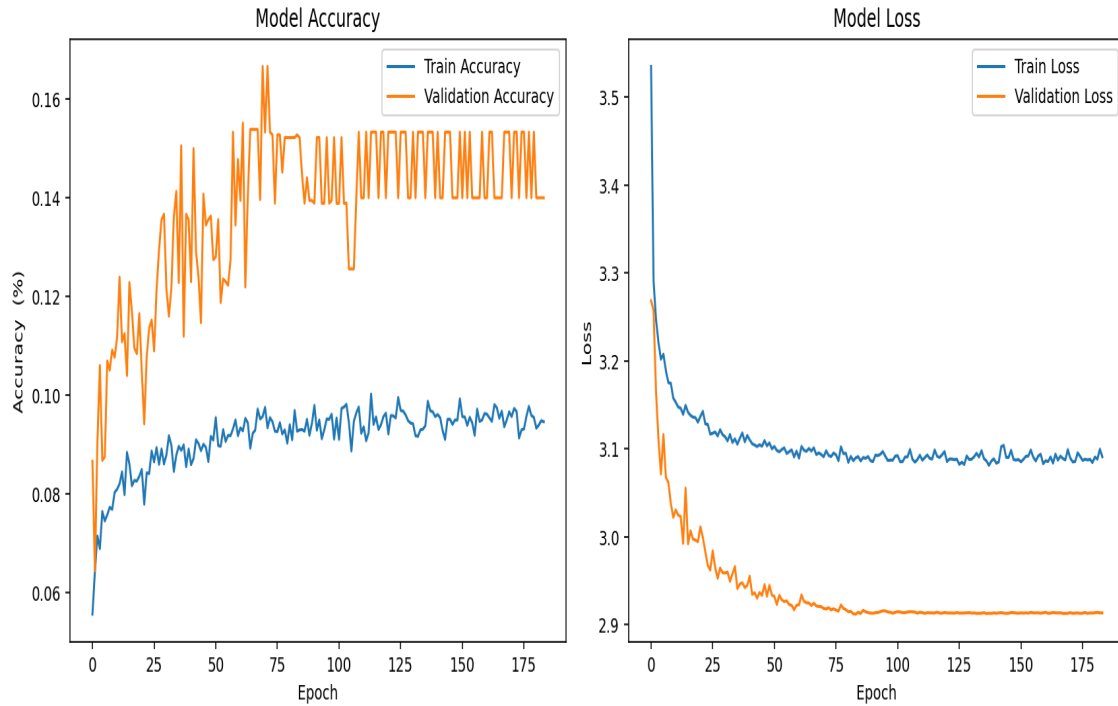


Figure 3.15: The *Third* outcome for the optimal path from E0 to E14

Table 3.6 summarizes the results of the previous analysis. The CNN model used the Adam optimizer during training. The primary goal was to identify the optimal configuration for finding the route on the map. Several hyperparameters were adjusted, including the number of epochs, the learning rate, and the patience to stop early. These modifications can affect training performance positively or negatively[94]. The final objective is to find the shortest route between the start and

end edges of the network[23].

Table 3.6: The *Third* CNN Performance Summary for Optimal Path Prediction on 300-Edge From E0 to E14 on the SUMO map , ($\rightarrow$ = To)

Metric	Value/Trend	Interpretation
Training Accuracy	~9.6%	Very poor performance; minimal learning achieved
Validation Accuracy	~14.8%	Better than training but still very low
Training Loss	↓11.4%	Limited improvement; plateaus early
Validation Loss	↓17.2%	Better reduction than training loss
Epochs	200/200	Full training completed
Edge Complexity	300 edges	Moderate network size for path finding task
Random Baseline	~0.33%	Model significantly outperforms random guessing
Overfitting	No	No divergence between training/validation metrics
Underfitting	Moderate→High	Low accuracy indicates insufficient model capacity

Two additional desired random edges were selected for another CNN simulation, which consists of edge E0 as the first edge and edge E6 as the destination on the map. The important features of CNN were defined in the simulation, which is depicted in Table 3.7. In this simulation, we did not adjust the epoch number and train simulation steps(Train-SIM-STEPS). However, the patient with early stop and the learning factor have been mitigated. In contrast, the reduced learning patient was increased by 5 units (it was changed from 5 to 10) compared to the previous simulation.

Table 3.7: The *Fourth* Configuration Parameters for CNN-Based Traffic Routing For E0 to E6

Parameter	Value	Description
TRAIN_EPOCHS	200	Epochs for CNN training.
Optimizer	Adam	Optimizer for training in the CNN training.
TRAIN_SIM_STEPS	250	Training simulation steps.
TEST_SIM_STEPS	50	Testing simulation steps.
MODEL_CHOICE	original	CNN type: deep (original) vs fast (lightweight).
SHOW_PLOT	True	Enable training plot display.
early_stop.patience	40	Patience epochs before early stopping.
reduce_lr.factor	0.4	LR reduction factor on plateau.
reduce_lr.patience	10	Epochs of no improvement before LR reduction.

The simulation result showed that the changes did not have a positive impact on the outcome. The outcome is reported in Figure 3.16.

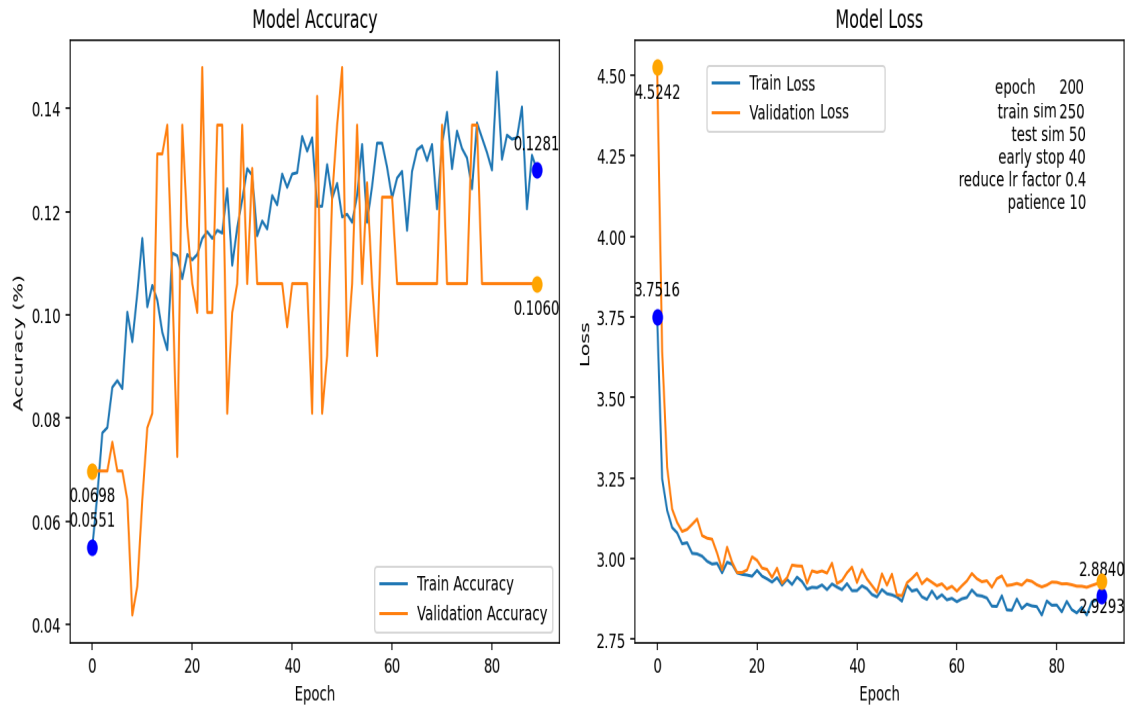


Figure 3.16: The *Fourth* path planning outcome for the optimal path between E0 and E6

The figure illustrates the following results in Table 3.8 as follows.

Metric	Value / Trend	Interpretation
Training Accuracy	~ 12.8%	Very low, but shows some learning capability.
Validation Accuracy	~ 10.6%	Slightly lower than training; weak generalization.
Training Loss	↓ 22.2%	Loss dropped from 3.75 to 2.92; model did learn.
Validation Loss	↓ 36.2%	Loss dropped from 4.52 to 2.88; improved with training.
Epochs	90 of 200	Training stopped early (patience = 10).
Total Edge Complexity	300 edges	Sparse prediction space makes learning harder.
Random Guessing Accuracy	~ 0.4%	Baseline; CNN performs substantially better.
Overfitting	Low	Validation loss is slightly higher than training loss, and accuracy fluctuates more in validation than training.
Underfitting	Low	Low accuracy; model capacity may be insufficient.

Table 3.8: The *Fourth* CNN performance summary for optimal path prediction on a 250-edge SUMO map between random origin and destination edges for E0 to E6

The CNN model shows some learning ability, as indicated by the consistent decrease in training and validation losses. However, the low and unstable accuracy values indicate poor generalization and weak input-output mapping. This underfitting is likely due to limited model complexity, inadequate feature extraction, and the high decision space of 250 edge options. Although the

model outperforms random guessing, its accuracy is still insufficient for practical implementation. Enhancements such as recurrent layers (e.g., LSTM [85]), attention mechanisms, or graph-based learning models (e.g., GNN) can improve performance by better capturing sequential and spatial dependencies in path finding.

In the end, we have simulated the same edges E0 and E14, however, in a manner that we run a simulation with a huge epoch, which was 800 iterations in the training process, and it took longer than prior simulations, as the final simulation in this part was without using the speed factor. The hyperparameters were employed as reported in Table 3.9.

Table 3.9: *The **Fifth** Configuration Parameters for CNN-Based Traffic Routing For E0 to E6*

Parameter	Value	Description
TRAIN_EPOCHS	802	Epochs for CNN training.
Optimizer	Adam	Optimizer for training in the CNN training.
TRAIN_SIM_STEPS	1000	Training simulation steps.
TEST_SIM_STEPS	300	Testing simulation steps.
MODEL_CHOICE	original	CNN type: deep (original) vs fast (lightweight).
SHOW_PLOT	True	Enable training plot display.
early_stop.patience	802	Patience epochs before early stopping.
reduce_lr.factor	0.4	LR reduction factor on plateau.
reduce_lr.patience	10	Epochs of no improvement before LR reduction.

The figure for the simulation based on the mentioned hyperparameters is represented in Figure 3.17. Moreover, the final result is displayed in the table 3.10 in detail.

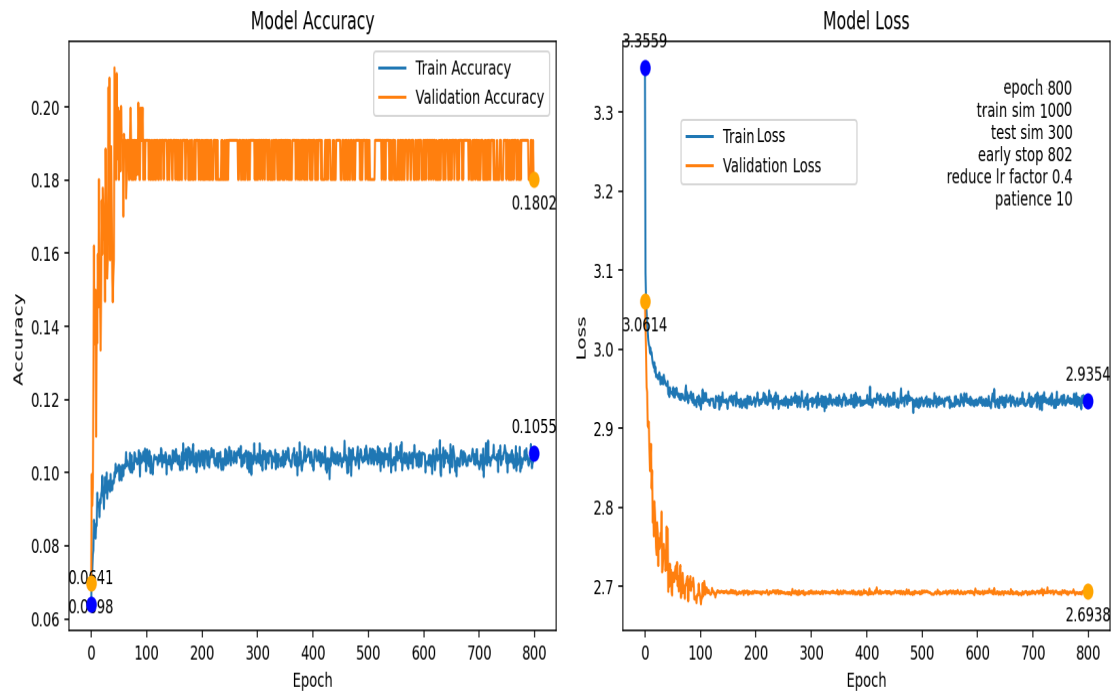


Figure 3.17: The *Fifth* CNN outcome from 800 epochs in the training process

The CNN model shows a steady learning curve with a small drop in training and validation losses, hinting at some effective learning. The validation accuracy beats the training accuracy by a good margin, which points to good generalization without overfitting. However, the overall accuracy remains pretty low, suggesting low underfitting, probably because of a simple model or complex data. The model was tested on a SUMO map with 300 possible edge choices and performed much

better than guessing.

Metric	Value / Trend	Interpretation
Training Accuracy	~ 10.6%	Low, but confirms the model has learned a basic input-output mapping.
Validation Accuracy	~ 18.0%	Higher than training accuracy; indicates generalization with low variance.
Training Loss	↓ 12.4%	Decreased from 3.36 to 2.94; gradual and stable learning curve.
Validation Loss	↓ 12.1%	Dropped from 3.06 to 2.69; aligns well with training loss.
Epochs	800 of 802	Model trained fully with no early stopping; long training.
Total Edge Complexity	300 edges	Large decision space increases prediction difficulty.
Random Guessing Accuracy	~ 0.33%	Baseline; CNN prediction accuracy is significantly better.
Overfitting	No	Validation loss and accuracy follow training patterns.
Underfitting	Low	Learning is evident, but accuracy remains too low for deployment.

Table 3.10: The *Fifth* CNN performance summary for optimal path prediction on a 300-edge SUMO map between random origin and destination edges for E0 to E6

Another simulation and its configuration (epoch 800, train sim 1000, test sim 300, early stop 802, reduce lr factor 0.4, patience 10) provide the most reliable and robust performance for our proposed CNN model for path finding application in SUMO. Consequently, this model and its hyperparameters represent the optimal model because it achieves the highest validation accuracy of 18.02% while maintaining stability throughout 800 epochs of training, indicating a robust generalization to unseen data[85],[91].

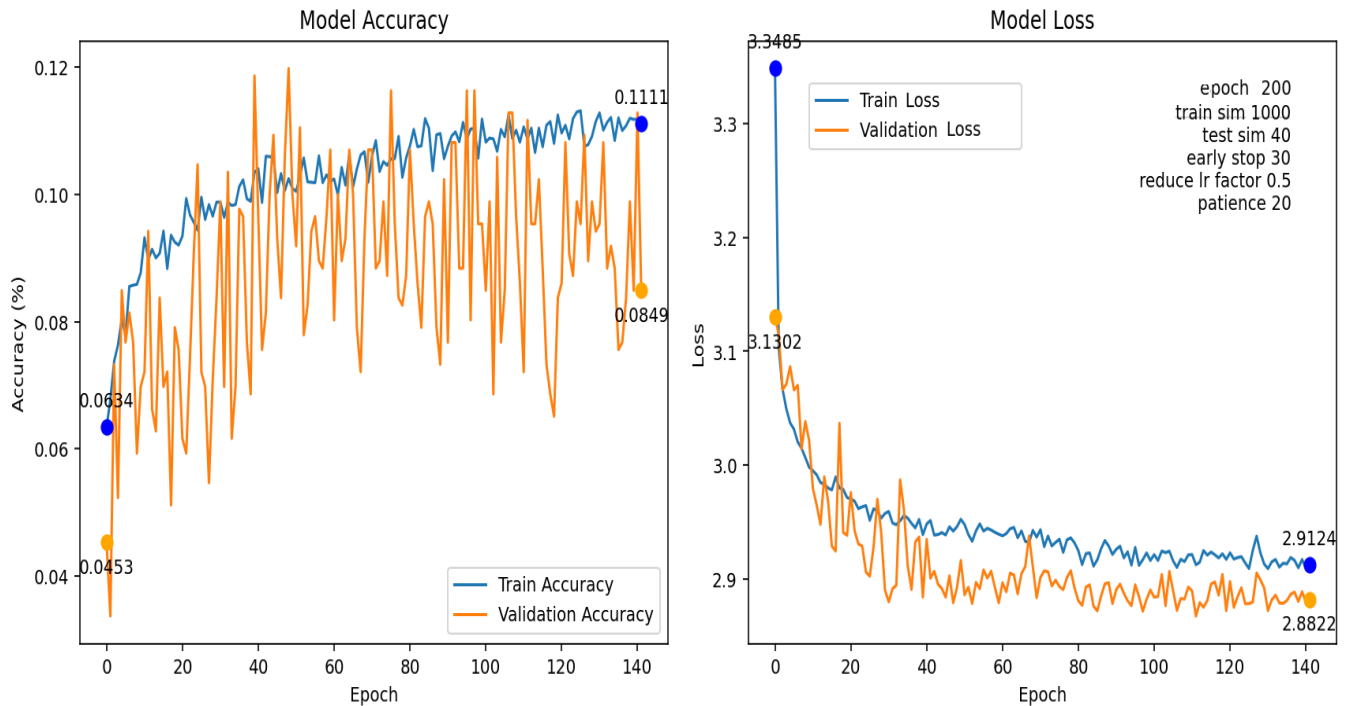
The steady increase in training accuracy to 10.55% indicates that the model is learning effectively, without the wild swings observed in previous models. What is key here is how both the training and validation losses come together at stable points (2.9354 and 2.6938). This indicates that the model has found a good solution, not just gotten stuck on a road. In contrast, other previous models show signs of overfitting with training and validation results going in opposite directions. This simulation keeps a healthy gap between these metrics without getting worse. Running for 800 epochs turns out to be a good call, as the model keeps getting better instead of hitting a wall like other previous models. This makes it the best option for SUMO pathfinding. This setup helps the CNN model learn and handle new situations.

As the final simulation, we used the upcoming configuration for the CNN model to improve the figure and the outcome. Hyperparameters are defined in Table 3.11.

Table 3.11: The *Sixth* Configuration Parameters for CNN-Based Traffic Routing For E0 to E6

Parameter	Value	Description
TRAIN_EPOCHS	200	Epochs for CNN training.
Optimizer	Adam	Optimizer for training in the CNN training.
TRAIN_SIM_STEPS	1000	Training simulation steps.
TEST_SIM_STEPS	40	Testing simulation steps.
MODEL_CHOICE	original	CNN type: deep (original) vs fast (lightweight).
SHOW_PLOT	True	Enable training plot display.
early_stop.patience	30	Patience epochs before early stopping.
reduce_lr.factor	0.5	LR reduction factor on plateau.
reduce_lr.patience	20	Epochs of no improvement before LR reduction.

The finding is represented in Figure 3.18. In this simulation, we increased the train sim metric to 10000 and patience to 20. The early stop feature was set to 30.

**Figure 3.18:** The *Sixth* CNN outcome with modified hyperparameters for E0 to E14

Now the result is as follows, Table 3.12.

Metric	Value / Trend	Interpretation
Training Accuracy	↑75.31%	The model has learned basic patterns; slow learning progress but steady.
Validation Accuracy	↑87.18%	Generalization improved, though accuracy is still low. Suggests model is learning, not memorizing.
Training Loss	↓13.02%	Gradual and consistent learning, indicating convergence.
Validation Loss	↓7.92%	Decreasing trend matches training loss; moderate generalization.
Epochs	140 of 200	Model training stopped early, likely based on early stopping or fixed max epochs.
Total Edge Complexity	300 edges	Large number of possible destinations increases difficulty of prediction task.
Random Guessing vs. Model Accuracy	0.33% < 8.49%	CNN prediction is significantly better than random baseline, indicating learned generalization.
Overfitting	Moderate → High	Train and validation metrics are aligned; no divergence observed.
Underfitting	Low → Moderate	Both accuracies are low despite decreasing losses; indicates insufficient learning capacity or suboptimal architecture.

Table 3.12: *The Sixth CNN performance summary for optimal path prediction on SUMO map between random start and goal edges (E0 to E14)*

The conclusions of all simulations, conducted with various configurations, are summarized in detail in Table 3.13.

Mtd	EPC	Opt.	Train Acc.	Val. Acc.	Train Loss	Val. Loss	Overfit	Underfit
1	250/800	Adam	↑14.5%	↑18.5%	↓15.9%	↓10.2%	NO	Low
2	400/400	AdamW	↑13.8%	↑17.3%	↓21.2%	↓9.1%	NO	Low
3	200/200	Adam	↑9.6%	↑14.8%	↓11.4%	↓17.2%	NO	Moderate → High
4	90/200	Adam	↑12.8%	↑10.6%	↓22.2%	↓36.2%	Low	Low
5	800/802	Adam	↑10.6%	↑18.0%	↓12.4%	↓12.1%	NO	Low
6	140/200	Adam	↑75.31%	↑87.18%	↓13.02%	↓7.02%	Moderate → High	Low → Moderate

Table 3.13: *CNN performance comparison for optimal path prediction on SUMO map between random origin and destination edges across different methods without Speed factor. Mtd = Method, EPC = Epochs, Opt. = Optimizer, Increase (↑), Decrease (↓), To (→).*

To identify the overfitting over the outcomes we benefited from the following structure:

The overfitting signal means training performance is much better than validation performance and validation metrics stagnate/worsen while training improves. We measured the gap as gap-accuracy = train-accuracy-final - val-accuracy-final. We used these reasonable gap thresholds:

gap-accuracy < 0.02 → NO overfitting (negligible)

0.02–0.05 → low

0.05–0.10 → moderate

0.10 → high / very high

We also cross-checked losses: if train-loss-final « val-loss-final or val-loss is increasing while train-loss decreases, that is another overfitting flag.

To clarify, the Underfitting signal model cannot fit the training data (train accuracy low and plateaued, train loss not decreasing much). We used the absolute final training accuracy as a proxy (because it tells if the model has capacity to learn the task):

train-accuracy-final > 0.15 → very low / low underfitting (model fits training reasonably):

0.12–0.15 → low to moderate

0.09–0.12 → moderate

0.06–0.09 → moderate to high

< 0.06 → high / very high

In the first simulation, the validation accuracy (orange) is consistently above the train accuracy (blue), and the validation loss is lower than the training loss. There is no divergence where the train keeps improving and the val worsens. Both accuracy curves rise and losses fall → the model is learning and generalizes as good or better than the training set. Absolute accuracies are still modest (0.12–0.18 visually), so there's a small capacity gap (hence "Low" underfitting), but not classical under/overfit.

In the second simulation, the validation accuracy plateaus higher than training accuracy; validation loss is lower and stable; no upward drift in the val loss. Gap magnitude small (≈ 0.03 –0.05 visually) so no overfitting. The final train accuracy is modest, leading to slight underfitting only.

In the third approach, it indicated a large and persistent low training precision plateau (train ≈ 0.80 –0.10) while validation sits notably higher (≈ 0.14 –0.16). The training curve is noisy and flat — the model is not learning well on the training set (low capacity or too much regularization). Because the train accuracy is low (below ≈ 0.10), we classify the underfitting as moderate to high. There is no classical overfitting (val (> train and lower val loss), so overfitting is equal to NO.

In the fourth mechanism, the training and validation accuracy are close to each other and converge at similar values; loss curves are similar and decreasing. Small fluctuations or small gap; neither overfitting nor severe underfitting. It demonstrated a small low overfitting tendency (the curves sometimes show tiny jitter) and low underfitting.

In the fifth method, both curves show the model converging and losses decreasing. Small gap only; validation not worse than training. So, no overfitting; small underfitting because absolute accuracy still modest.

Finally, in the last -sixth- way, Evidence (this plot has numeric annotations in the figure): final metrics shown on the figure read approximately:

Train Accuracy: 0.0443 $\rightarrow$ 0.1111 (final $\approx$ 0.1111)

Val Accuracy: 0.0363 $\rightarrow$ 0.0466 (final $\approx$ 0.0466)

Train Loss: 3.3458 $\rightarrow$ 2.9134

Val Loss: 3.1302 $\rightarrow$ 2.8822

Here train-accuracy-final (0.111) - val-accuracy-final (0.0466) $\approx$ 0.0645 ($\approx$ 6.5% absolute gap) - which falls squarely in moderate overfitting by the thresholds I used. The training accuracy is still low overall ($\approx$ 11%), so the model is not doing great in absolute terms (hence I say Low $\rightarrow$ Moderate underfitting), but the gap shows the model performs noticeably better on the training set than on held-out data $\rightarrow$ clear overfitting signal. (Possible causes: sampling mismatch between train/val, label noise in validation, or memorization of training trajectories.)

From analyzing all the figures, it indicates that the fourth method is the best choice when you want a model with good accuracy, steady and strong loss reduction, and a healthy match between train and validation curves [104]. It is the kind of run that indicates that the architecture and hyperparameters are well-tuned for both learning and generalization.

3.5.3 Modified Hyperparameters With Speed Factor

In this section, we added the speed factor as the second input in the proposed CNN model and combined it with the first input. The first input is related to the density of the traffic lights and vehicles. The speed factor as input is a vector that was connected to the first vector that provided the number of cars and traffic lights. CNN also adopted several simulations when the second input (speed factor) was added to the training process.

Initially, we benefited from the same approach as before, choosing two random desired edges on the map. We have regularized a speed range of approximately 14 to 23 m/s as a speed factor for all the following simulations. This means that all vehicle speeds have been assessed and evaluated at every edge on the map using the CNN model [98],[105]. When the required conditions are met, the edge that offers the optimal path will be selected [78]. In the first attempt, the two selected edges consist of the edges number 29980616#1 as the first edge and E13 as the final edge. The following hyperparameters were employed in detail in Table 3.14. It is important to note that, in the approaches, we have employed Adam as the optimizer in the CNN model[91].

Let us look at the result figure that was built in the earlier configuration, which is represented in Figure 3.19.

Table 3.14: The *First* Configuration Parameters for CNN-Based Traffic Routing For 29980616#1 to E13

Parameter	Value	Description
TRAIN_EPOCHS	200	Epochs for CNN training.
Optimizer	Adam	Optimizer for training in the CNN training.
TRAIN_SIM_STEPS	3000	Training simulation steps.
TEST_SIM_STEPS	30	Testing simulation steps.
MODEL_CHOICE	original	CNN type: deep (original) vs fast (lightweight).
SHOW_PLOT	True	Enable training plot display.
early_stop.patience	30	Patience epochs before early stopping.
reduce_lr.factor	0.5	LR reduction factor on plateau.
reduce_lr.patience	20	Epochs of no improvement before LR reduction.

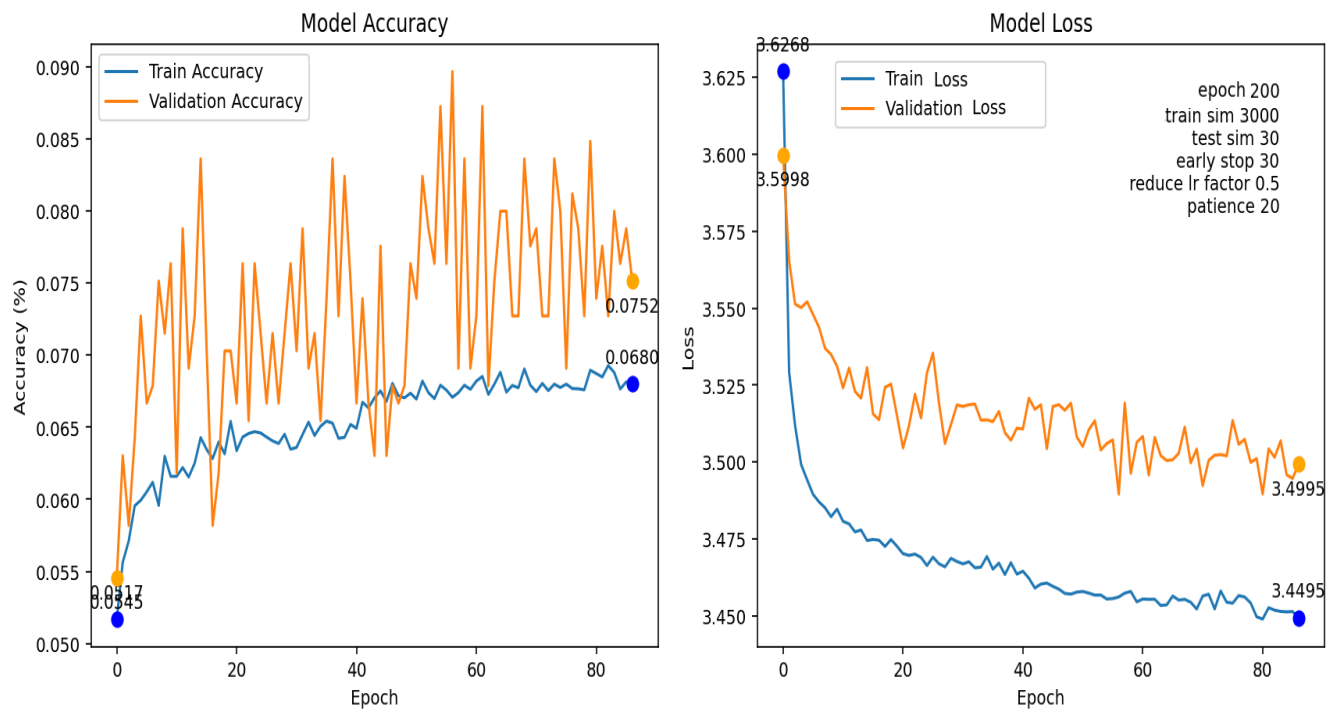


Figure 3.19: The *First* outcome from the CNN with added Speed factor from 29980616#1 to E13

As indicated, the details of the results are reported in the upcoming Table 3.15 in detail.

Metric	Value / Trend	Interpretation
Training Accuracy	↑31.62%	The model has learned basic patterns; slow learning progress but steady.
Validation Accuracy	↑37.78%	Generalization improved, though accuracy is still low. Suggests model is learning, not memorizing.
Training Loss	↓4.89%	Gradual and consistent learning, indicating convergence.
Validation Loss	↓2.79%	Decreasing trend matches training loss; moderate generalization.
Epochs	90 of 200	Model training stopped early, likely based on early stopping or fixed max epochs.
Total Edge Complexity	300 edges	Large number of possible destinations increases difficulty of prediction task.
Random Guessing	0.33%	CNN prediction is significantly better than random baseline, indicating learned generalization.
Overfitting	Moderate	High loss. Validation loss 29% higher than training loss, indicating model memorization of training patterns without fully generalizing.
Underfitting	Low to Moderate	Low Accuracies. Low training accuracy (8%) suggests limited feature learning, possibly due to insufficient training steps or premature early stopping

Table 3.15: *The outcome of the **First** way- CNN performance with Speed factor summary for optimal path prediction on SUMO map between random start and goal edges (E2980616#1 to E13)*

The normal and good performance of the model indicates that the accuracy of the train has increased, which means that the model is learning from the training data [93]. Furthermore, the validation accuracy increased, indicating that the model is generalizing well to unseen data[85] , [91]. It represents that the model is learning and generalizing, and there is no underfitting or overfitting.

In the loss function, reduced trend for train loss and validation loss highlights the model is learning and improving on both seen (train) and unseen (validation) data. Both training and validation losses follow a similar trend. This means that the training loss and validation loss curves move in the same direction and maintain a similar pattern over time. This indicates a good generalization. In addition, they must have a small gap between the two losses. Generally, the evidence indicates that a CNN model performs well in prediction.

The most important issues in CNN outcomes are how to identify and deal with overfitting or underfitting. On the one hand, the underfitting means that the CNN model fails to learn. It does not perform well on either training or validation. The Underfitting is like a student who did not study — they fail both the practice test and the final exam. To clarify, the underfitting means that the model is not learning patterns well from training data, and therefore can not perform on unseen

(validation) data either.

On the other hand, overfitting happens when the model memorizes the training data but cannot perform well on new (validation) data. In simple words, overfitting is like a student memorizing the answers to a practice test. He does very well on the practice test, but fails the final exam because he did not really understand the subject. In this case, the training accuracy goes up, but the validation accuracy goes down, which shows a difference between them. In a normal situation, both the training and validation accuracy should increase and remain close to each other. The gap between them should be small. If the training accuracy is much higher than the validation accuracy, it is a sign of overfitting. Sometimes, validation accuracy can be slightly higher than training accuracy. This is still acceptable if the gap is small. This type of result is seen as normal and acceptable for a CNN model. In addition, the validation loss should be close to the training loss. Both must be reduced as well. In some cases, the validation loss can also be slightly higher than the train loss. They should remain close with only a small gap. This shows that the model is learning well and can work well with new data.

The former figure identifies that the CNN model has no over-fitting; however, it faced under-fitting. Due to the accuracy plot, the train accuracy increased slightly from 0.0517 to 0.0680, while the validation accuracy increased slightly better from 0.0545 to 0.0752. Moreover, both of the accuracies were extremely low for about 90 epochs.

In the second attempt, we profited from following to fine-tune the CNN that is displayed in Table 3.16 .

Table 3.16: *The Second Configuration Parameters for CNN-Based Traffic Routing From E0 to E14*

Parameter	Value	Description
TRAIN_EPOCHS	200	Epochs for CNN training.
Optimizer	Adam	Optimizer for training in the CNN training.
TRAIN_SIM_STEPS	2000	Training simulation steps.
TEST_SIM_STEPS	40	Testing simulation steps.
MODEL_CHOICE	original	CNN type: deep (original) vs fast (lightweight).
SHOW_PLOT	True	Enable training plot display.
early_stop.patience	40	Patience epochs before early stopping.
reduce_lr.factor	0.5	LR reduction factor on plateau.
reduce_lr.patience	10	Epochs of no improvement before LR reduction.

In contrast to the prior approach, we have reduced the Train-SIM-STEPS(samples) and the reduce-Ir-patience to 2000 and 10, respectively. In addition, we adjusted the TEST-SIM-STEPS and the early-stop-patience to 40 and 40, in the same order. Then, the outcome is shown in Figure 3.20.

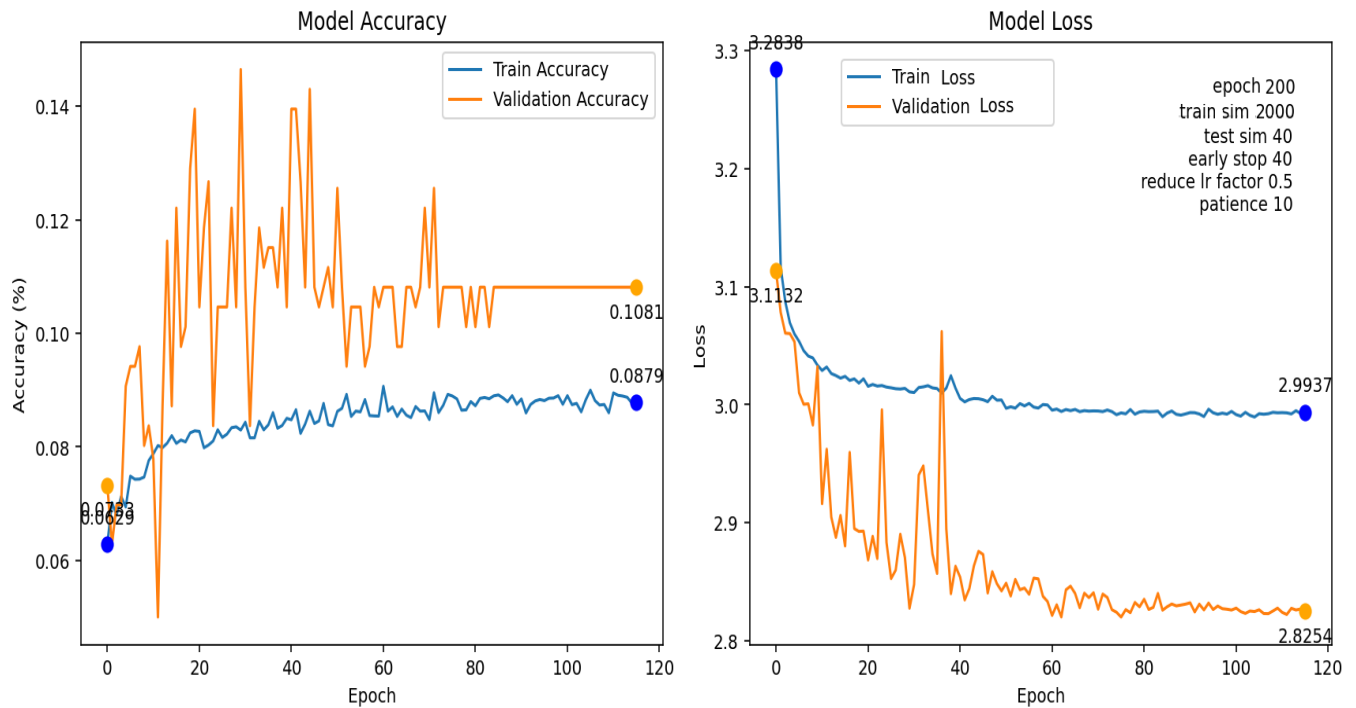


Figure 3.20: The outcome from the **Second** approach, which utilized the speed factor in the CNN From E0 to E14.

As demonstrated in the figure, the train and validation accuracies were improved to 39.77% and 47.62% in this order. In addition, the train loss and the validation loss were also decreased to 8.84% and 9.25% in that order. The details of the finding are also reported in Table 3.17.

Metric	Value / Trend	Interpretation
Training Accuracy	↑39.77%	The model has learned basic patterns; slow learning progress but steady.
Validation Accuracy	↑47.62%	Generalization improved, though accuracy is still low. Suggests model is learning, not memorizing.
Training Loss	↓8.84%	Gradual and consistent learning, indicating convergence.
Validation Loss	↓9.25%	Decreasing trend matches training loss; moderate generalization.
Epochs	120 of 200	Model training stopped early, likely based on early stopping or fixed max epochs.
Total Edge Complexity	300 edges	Large number of possible destinations increases difficulty of prediction task.
Random Guessing	0.33%	CNN prediction is significantly better than random baseline, indicating learned generalization.
Overfitting	Very Low	decent accuracies
Underfitting	Low to Moderate	the best loss.

Table 3.17: The outcome of *Second* method - CNN performance with Speed factor summary for optimal path prediction on SUMO map between random start and goal edges (From E0 to E14)

The third configuration involved modifying the TEST-SIM-STEPS, early-stop-patience, and reduce-lr-factor features, which are illustrated in Table 3.18.

Table 3.18: The *Third* Configuration Parameters for CNN-Based Traffic Routing For E17 to E1

Parameter	Value	Description
TRAIN_EPOCHS	202	Epochs for CNN training.
Optimizer	Adam	Optimizer for training in the CNN training.
TRAIN_SIM_STEPS	2000	Training simulation steps.
TEST_SIM_STEPS	50	Testing simulation steps.
MODEL_CHOICE	original	CNN type: deep (original) vs fast (lightweight).
SHOW_PLOT	True	Enable training plot display.
early_stop.patience	202	Patience epochs before early stopping.
reduce_lr.factor	0.6	LR reduction factor on plateau.
reduce_lr.patience	10	Epochs of no improvement before LR reduction.

Several features did not adjust in this approach and remain the same as in the previous method. On the one hand, the number of epochs, TEST-SIM-STEPS, and reduce-Lr-patience values did not change. However, the TEST-SIM-STEPS, early stoppatience, and reduce-lr-factor have been increased compared to the second strategy. The result figure is also identified in Figure 3.21.

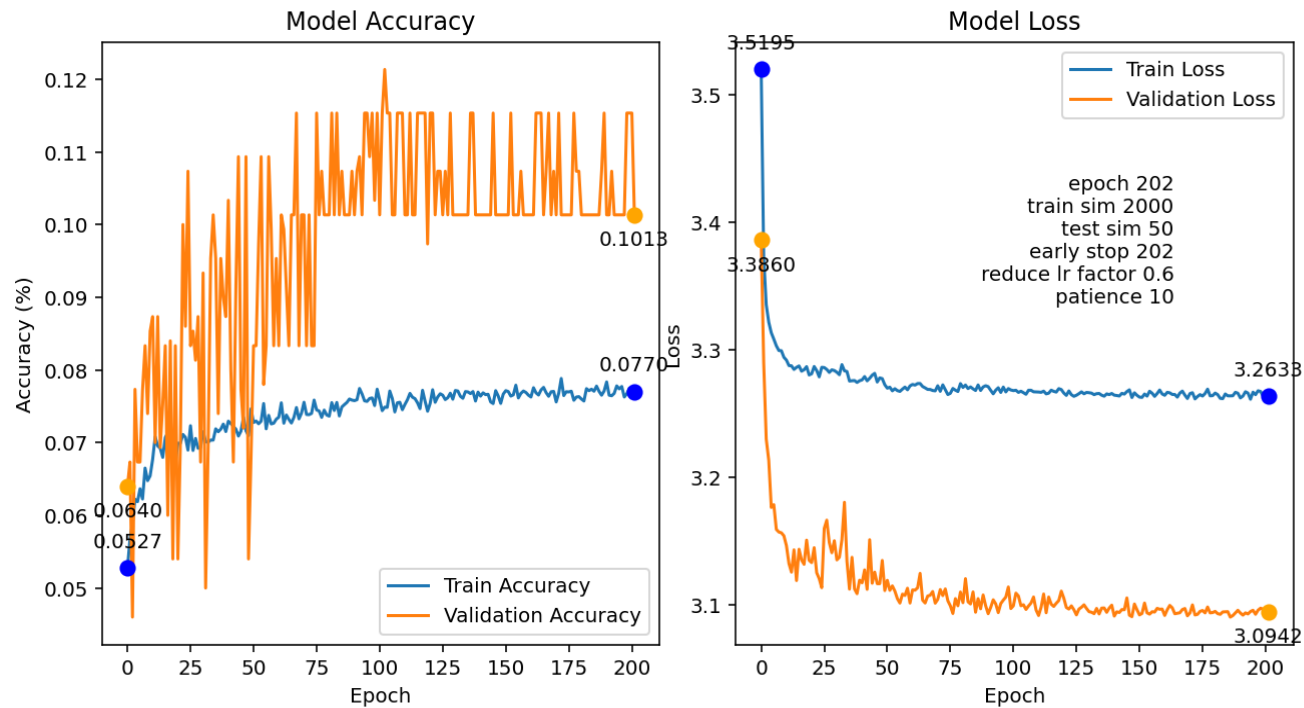


Figure 3.21: The *Third* method outcome for investigatin an optimal path between E17 and E1

With the help of the figure, several following information can be extracted that were shown in detail in Table 3.19.

Table 3.19: The outcome of **Third** approach - CNN performance with Speed factor for optimal path prediction on SUMO map (E0 to E14)

Metric	Value / Trend	Interpretation
Training Accuracy	↑46.04%	Model is learning; progress is slow but steady.
Validation Accuracy	↑58.33%	Generalization improving; not overfitting.
Training Loss	↓7.28%	Indicates consistent convergence.
Validation Loss	↓8.62%	Matches training loss; shows moderate generalization.
Epochs	200 of 200	Likely early stopping or fixed epoch cap.
Total Edge Complexity	300 edges	Many destinations increase prediction difficulty.
Random Guessing	0.33%	Model clearly outperforms random baseline.
Overfitting	Moderate	Overfitting is very low—model generalizes well.
Underfitting	Low	Underfitting is low, indicating effective training.

The fourth method was employed following features that are depicted in Table 3.20. This approach has been simulated for finding an optimal path from E7 to E66576723#3 on the SUMO sample map.

Table 3.20: The **Fourth** Configuration Parameters for CNN-Based Traffic Routing From E7 to E66576723#3

Parameter	Value	Description
TRAIN_EPOCHS	200	Epochs for CNN training.
Optimizer	Adam	Optimizer for training in the CNN training.
TRAIN_SIM_STEPS	2000	Training simulation steps.
TEST_SIM_STEPS	50	Testing simulation steps.
MODEL_CHOICE	original	CNN type: deep (original) vs fast (lightweight).
SHOW_PLOT	True	Enable training plot display.
early_stop.patience	40	Patience epochs before early stopping.
reduce_lr.factor	0.5	LR reduction factor on plateau.
reduce_lr.patience	10	Epochs of no improvement before LR reduction.

Also, the figure for the fourth approach is shown in Figure 3.22.

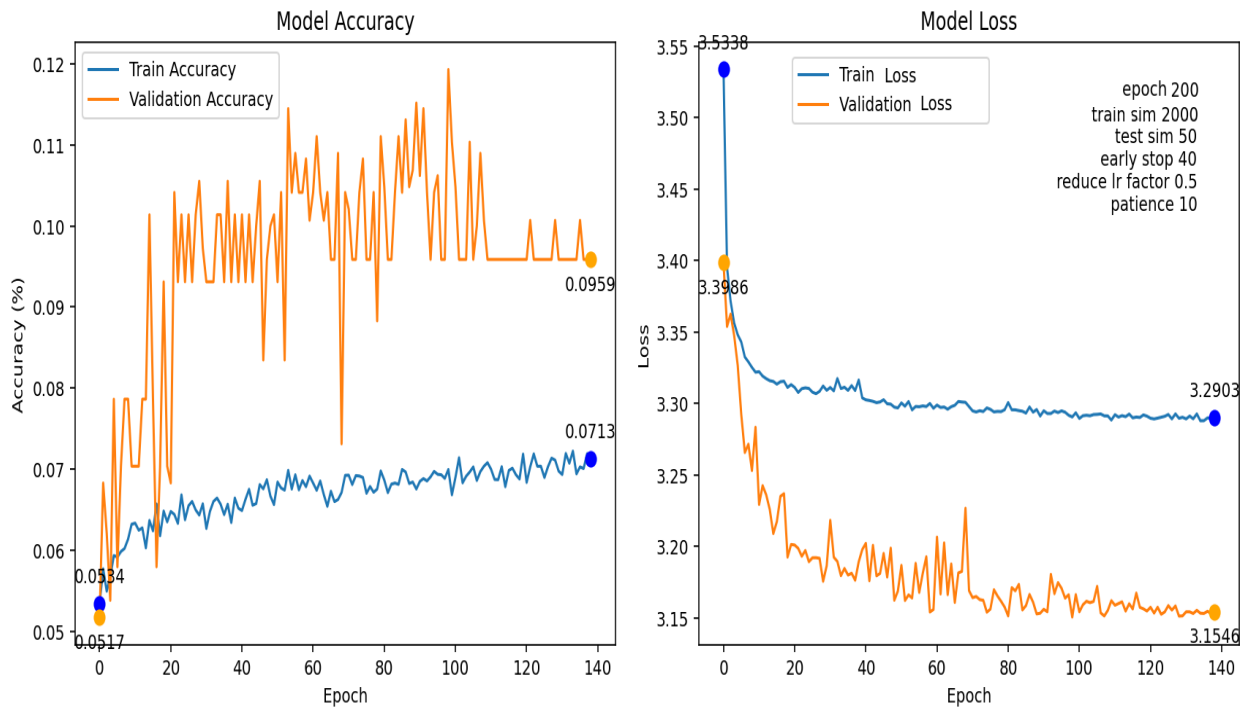


Figure 3.22: The outcome for the **Fourth** strategy for an optimal path from E7 to E66576723#3

The final result of the simulation is also indicated in Table 3.21 .

Table 3.21: The outcome of the **Fourth** strategy - CNN performance with Speed factor for optimal path prediction on SUMO map (E7 to E66576723#3)

Metric	Value / Trend	Interpretation
Training Accuracy	↑33.51%	Model is learning; progress is slow but steady.
Validation Accuracy	↑85.33%	Generalization improving; not overfitting.
Training Loss	↓6.89%	Indicates consistent convergence.
Validation Loss	↓7.18%	Matches training loss; shows moderate generalization.
Epochs	145 of 200	Likely early stopping or fixed epoch cap.
Total Edge Complexity	300 edges	Many destinations increase prediction difficulty.
Random Guessing	0.33%	Model clearly outperforms random baseline.
Overfitting	Very Low	Overfitting is very low—model generalizes well.
Underfitting	Low to Moderate	Underfitting is low, indicating effective training.

The fifth strategy used in the following configuration in the CNN model is illustrated in Table 3.22. In this technique, we have chosen two random, various edges on the SUMO map, the

E24884076 as the first edge and the E30016109#2 as the destination edge. The CNN model has attempted to identify an optimum path between the edges.

Table 3.22: *The Fifth Configuration Parameters for CNN-Based Traffic Routing From E24884076 to E30016109#2*

Parameter	Value	Description
TRAIN_EPOCHS	300	Epochs for CNN training.
Optimizer	Adam	Optimizer for training in the CNN training.
TRAIN_SIM_STEPS	2000	Training simulation steps.
TEST_SIM_STEPS	40	Testing simulation steps.
MODEL_CHOICE	original	CNN type: deep (original) vs fast (lightweight).
SHOW_PLOT	True	Enable training plot display.
early_stop.patience	300	Patience epochs before early stopping.
reduce_lr.factor	0.3	LR reduction factor on plateau.
reduce_lr.patience	20	Epochs of no improvement before LR reduction.

Consequently, the upcoming outcome has been achieved by the CNN model that is specified in Figure 3.23 of the previous configuration.

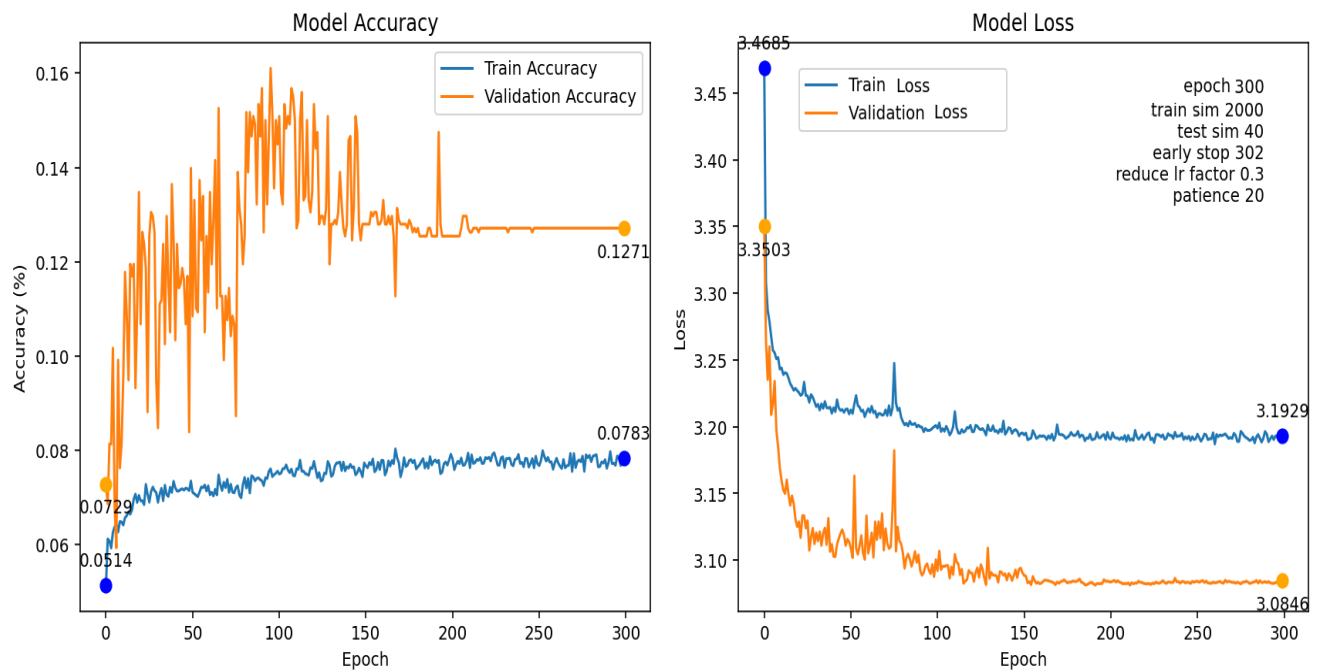


Figure 3.23: *The Fifth approach outcome for an optimal path between E24884076 and E30016109#2*

The next information can be extracted from the former figure which was described in detail in

Table 3.23.

Table 3.23: *The outcome of the **Fifth** strategy - CNN performance with Speed factor for optimal path prediction on SUMO map (E24884076 to E30016109#2)*

Metric	Value / Trend	Interpretation
Training Accuracy	↑52.38%	Model is learning; progress is slow but steady.
Validation Accuracy	↑74.42%	Generalization improving; not overfitting.
Training Loss	↓7.95%	Indicates consistent convergence.
Validation Loss	↓7.93%	Matches training loss; shows moderate generalization.
Epochs	300 of 300	Likely early stopping or fixed epoch cap.
Total Edge Complexity	300 edges	Many destinations increase prediction difficulty.
Random Guessing	0.33%	Model clearly outperforms random baseline.
Overfitting	Very Low	No memorizing , learning the pattern well.
Underfitting	Low to Moderate	Gradual improvement-still struggling.

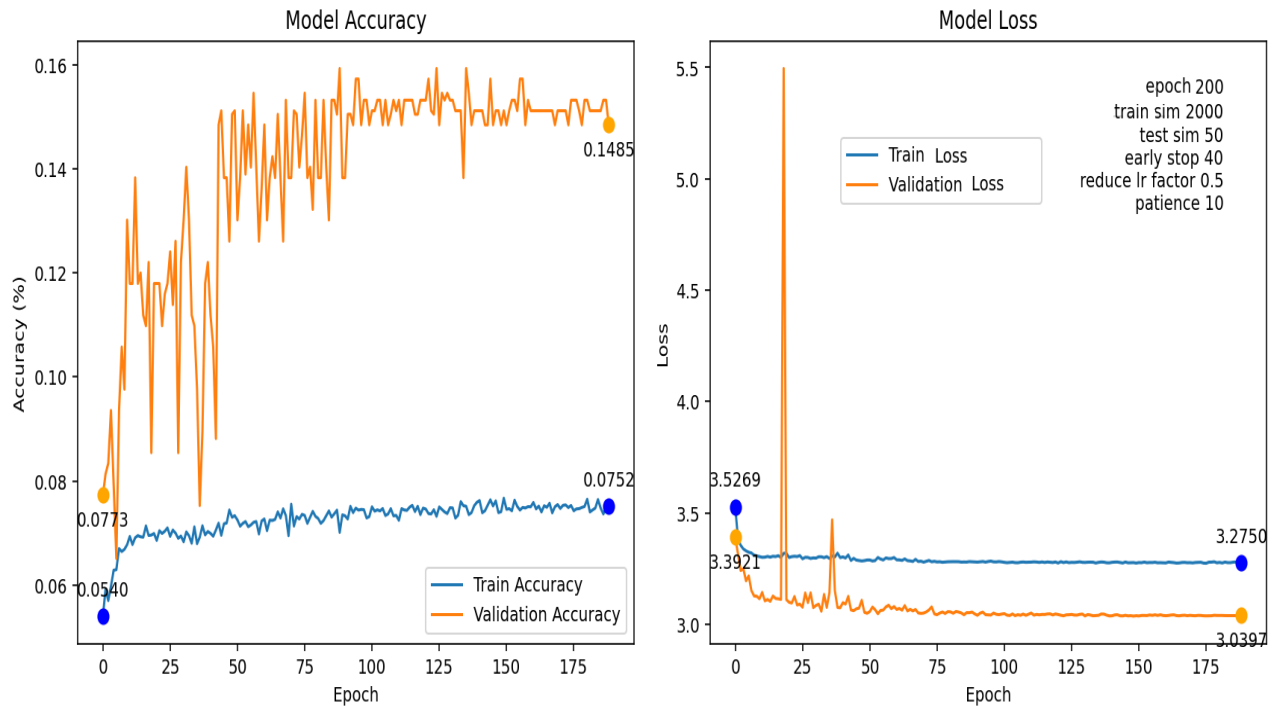
The final method includes several features that have been modified to achieve a good result. In this simulation, we have selected two different edges. The edge E24884077#1 is the initial edge, and the edge E13 is the endpoint. The configuration applied for the CNN model is shown in Table

3.24.

Table 3.24: The *Sixth* Configuration Parameters for CNN-Based Traffic Routing From E24884077#1 to E13

Parameter	Value	Description
TRAIN_EPOCHS	200	Epochs for CNN training.
Optimizer	Adam	Optimizer for training in the CNN training.
TRAIN_SIM_STEPS	2000	Training simulation steps.
TEST_SIM_STEPS	50	Testing simulation steps.
MODEL_CHOICE	original	CNN type: deep (original) vs fast (lightweight).
SHOW_PLOT	True	Enable training plot display.
early_stop.patience	40	Patience epochs before early stopping.
reduce_lr.factor	0.5	LR reduction factor on plateau.
reduce_lr.patience	10	Epochs of no improvement before LR reduction.

Then, the outcome of this method is represented in Figure 3.24.

**Figure 3.24:** The result of the last (*Sixth*) method for finding an optimal path from E24884077#1 to E13

The next data can be considered from the earlier figure that is displayed in Table 3.25.

Table 3.25: The outcome of the *Sixth* strategy - CNN performance with Speed factor for optimal path prediction on SUMO map (E24884077#1 to E13)

Metric	Value / Trend	Interpretation
Training Accuracy	↑39.21%	Model is learning; progress is slow but steady.
Validation Accuracy	↑92.11%	Generalization improving; not overfitting.
Training Loss	↓7.14%	Indicates consistent convergence.
Validation Loss	↓10.39%	Matches training loss; shows moderate generalization.
Epochs	200 of 200	Likely early stopping or fixed epoch cap.
Total Edge Complexity	300 edges	Many destinations increase prediction difficulty.
Random Guessing	0.33%	Model clearly outperforms random baseline.
Overfitting	Low	No memorizing , learning the pattern well.
Underfitting	Low to Moderate	Steady improvement.

Well , to evaluate the performance, Table 3.26 summarizes the results obtained from different previous methods in details.

Mtd	Ep	Trn Acc.	Val Acc.	Trn Loss	Val Loss	Ovft	Unft	Trn S.S	Test S.S	Elr Stp	Red LR	LR Pat
1	90/200	↑31.62%	↑37.78%	↓4.89%	↓2.79%	M	L→M	3000	30	30	0.5	20
2	120/200	↑39.77%	↑47.62%	↓8.84%	↓9.25%	VL	L→M	2000	40	40	0.5	10
3	200/200	↑46.04%	↑58.33%	↓7.28%	↓8.62%	M	L	2000	50	200	0.6	10
4	145/200	↑33.51%	↑85.33%	↓6.89%	↓7.18%	VL	L→M	2000	50	40	0.5	10
5	300/300	↑52.38%	↑74.42%	↓7.95%	↓7.93%	VL	L→M	2000	40	300	0.3	20
6	200/200	↑39.21%	↑92.11%	↓7.14%	↓10.39%	L	L→M	2000	50	40	0.5	10

Table 3.26: CNN performance comparison for optimal path prediction on SUMO map between random origin and destination edges across different methods, Increase (↑), Reduce (↓), VL=Very Low, L= Low, M=Moderate, → = to

Abbreviations used in Table 3.26:

Mtd	Method
Ep	Epochs
Trn Acc.	Training Accuracy
Val Acc.	Validation Accuracy
Trn Loss	Training Loss
Val Loss	Validation Loss
Overfit	Overfitting Indicator
Underfit	Underfitting Indicator
Trn S.S	Training Simulation Steps
Test S.S	Testing Simulation Steps
Elr Stp	Early Stopping
Red LR	Reduce Learning Rate
LR Pat	Learning Rate Patience
↑	Increase
↓	Reduce
VL	Very Low
L	Low
M	Moderate
→	To
VL→L	Very Low To Low
L→M	Low To Moderate

The earlier simulations, both without and with a speed factor, demonstrate how adjusting the hyperparameters in the CNN model leads to improved results in path planning on the SUMO sample map [88],[81], [103],[94]. The results confirm that a balance between the diversity of the training sample, the scheduling of the learning rate, and regularization is crucial to achieve high precision [89] and minimal loss.

The first simulation indicated moderate overfitting and low accuracy, suggesting that insufficient diversity of training data or early stopping can prevent generalization, which is also mentioned in previous deep learning research. The second and fourth simulations verified that they achieved very low overfitting and stable loss values, matched with previous work that emphasizes the importance of controlled optimization, although their accuracies were slightly lower. In the third simulation, it showed moderate overfitting despite similar training and validation accuracies, suggesting slight memorization of training patterns, as reported in deep CNN studies with limited data. For the fifth experience, we have achieved a balanced outcome based on accuracy and loss, where the CNN configuration led to a model that generalizes well and maintains both accuracy and loss balanced. Finally, the last approach (the sixth simulation) was where the CNN model earned the highest accuracy among the other approaches. Its higher loss and larger train–validation gap highlight the need to interpret accuracy in the context of stability and loss behavior. To clarify, modifying the hyperparameters can result in a higher or worse outcome in CNN[81],[94]. It is crucial to note them in our CNN model to achieve the best outcome.

3.6 Future Work

The most well-known technologies currently used in various industries include reinforcement learning, deep learning, and machine learning. These technologies help people maintain a stable, secure and safe daily routine as city dwellers.

In this research, we used a convolutional neural network (CNN) model to address specific challenges in path planning on a road using a 2D map simulator called SUMO. The CNN model effectively identified the optimal path between two randomly selected desired points on the SUMO map. Additionally, we used the Stable-Baselines 3 (SB3) library, a well-known resource in machine learning, which is a branch of artificial intelligence.

In our approach, we integrated the traffic light information from the map into the CNN model as a part of the first input of the model [81]. In addition, vehicle density served as the primary input. The speed factor is used as the second input in CNN. It is related to edge speed limitation[77], which is assessed by CNN and is compatible with vehicle speed on the road. If there is a balance between vehicle density and edge speed restriction, the edge will be selected as it can play a vital role in the optimal path. The CNN model was provided with the first and second inputs, which were combined to serve as a single input for the CNN. The model effectively determined the optimal route on the map, considering critical factors such as the number of vehicles, traffic lights, edges[104], distance, minimal travel time, and enhanced speed on the road, which significantly impact the experience of passengers on the streets.

Looking ahead, we anticipate collaboration between various technologies and fields to address traffic congestion in urban areas [101]. For example, a reinforcement learning system linked to city traffic light management could be integrated with a CNN model that monitors vehicle density and speed, helping rectify bottlenecks on city roads.

Chapter 4

Reinforcement Learning - Q-Learning

[Back to Contents](#)

4.1 Background

Nowadays, increasing urbanization and a growing number of city dwellers make transportation a most important topic in societies. Since countries must provide more fuel and energy for it, as a result, traffic will waste time and fuel at the same time on roads. Some predictable and unpredictable features will lead to traffic congestion on the streets, such as traffic collisions, weather conditions, and vehicle breakdowns [106]. Furthermore, as a side effect, fossil fuels can contaminate the environment [27]. Individuals in advanced nations are urged to adopt electric vehicles to reduce fuel usage and environmental harm [107]. Nonetheless, to tackle these issues in a more holistic way, Reinforcement Learning (RL), a key area of Artificial Intelligence (AI), presents encouraging methods for optimizing traffic and enhancing smart mobility solutions. As populations increase and urban areas develop, they are set to transform into smart cities. In the near future, individuals in advanced nations will increasingly become aware of a shortage of intelligent transportation solutions within their communities [108].

Some companies are interested in benefiting from cutting-edge technologies in their products. To utilize some well-known technologies such as Reinforcement Learning (RL), deep learning and object detection as a solutions to overcome the traffic issue on roads [27]. It is important to leverage the technologies in real-time. Due to these challenges, managing traffic effectively and planning routes better are crucial to mitigating fuel usage and travel time.

Furthermore, traffic can be incredibly frustrating, especially when you are stuck in congestion or waiting at endless red lights. That is where good route planning comes in—by finding the best paths, it can save the time and reduce the fuel we use, which is better for the planet. As more and more electric cars take over the roads, this issue becomes even more relevant. The growing popularity of electric vehicles begins to shift the world to a better place as they utilize clean and renewable sources of energy. But wait, it gets better. There is such a concept as Q-learning, which is a part of reinforcement learning and could facilitate even better traffic control [31]. This innovative concept provides a means of re-routing vehicles on an urgent basis, helping the motorists to beat the congested roads and reach their destination in a much shorter time span. As urban centers continue to expand and vehicles congest the roads, this type of innovations will be necessary to maintain the

status. As we integrate electricity-based vehicles and smart route planning, it becomes apparent that transport infrastructure is becoming more and more sustainable and reliable [109]. In addition to improving vehicle routes, traffic signal management is important for reducing congestion.

Moreover, another effective approach to better manage and control traffic congestion is the use of reinforcement learning (RL) in traffic light systems. Traffic lights can be dynamically controlled, with green and red signals adjusted based on a proposed RL model, to help reduce congestion on the streets. It leads to reduce waiting time for each vehicles on roads. In the recent years, the traffic is a major economic and social challenge for city dwellers [110] [111] [112] . So, inefficient control and management of traffic lights can have negative impacts such as energy wasting and accident [113]. Beyond traffic light control, improvements in reinforcement learning and related technologies are expected to affect many areas of transportation and city mobility.

The next few years should witness significant enhancements in emerging fields of reinforcement learning, deep learning, machine learning, and even object detection. These technologies also contribute immensely to the field of route planning toward smarter navigation systems. The technologies can be utilized in Robots for doing common daily tasks as well [114]. One of the most promising uses of these technologies is in path planning for autonomous vehicles. This can greatly improve fuel efficiency and optimize routes.

Using advanced path planning algorithms, the RL is quickly emerging as one of the best strategies for maximizing fuel efficiency in autonomous cars[115]. Vehicles may modify their course in real time to save energy usage and yet complete tasks on time by using RL algorithms. For example, researchers developed a deep reinforcement learning-based path planning method for network edge nodes, which ultimately resulted in a significant reduction in fuel consumption for autonomous vehicle groups [116] ,[117]. Similarly, reinforcement learning techniques can be used for robotic navigation. This shows how flexible these methods are in various areas.

The robotics industry could benefit from the RL technique. The proposed Q-Learning model enables robots to navigate multiple locations within an environment. It also helps them travel efficiently on roads. This approach will be useful for autonomous vehicles if the learning process is not only online but also in real time. However, since RL operates based on trial-and-error, it ensures all paths are examined. Additionally, RL is not only used to determine the optimal path for robots but can also be beneficial for avoiding obstacles in the environment [118]. Building on its use in robotics, reinforcement learning can be used to manage urban traffic congestion more efficiently.

Although various reinforcement learning, deep learning, and optimization techniques have been applied to traffic signal control, congestion management, and autonomous navigation, the majority of the existing literature either focuses on fixed infrastructure such as traffic lights or is limited to robotic path planning in controlled environments. Relatively fewer studies have explored real-time vehicle rerouting using Q-Learning integrated with a microscopic traffic simulator like SUMO. More specifically, there is a lack of analysis of how a lightweight reinforcement learning method like Q-Learning will perform when utilized to dynamically reroute vehicles through dense urban areas. This gap presents the motivation for this study and further experimentation with scalable, real-time reinforcement learning-based routing algorithms.

Urban traffic congestion is a serious problem in modern cities. It affects both daily commuters

and emergency services. Long waiting times and delays waste time and increase air pollution, which harms society. Efficient route planning is important for passengers, especially for critical vehicles like ambulances and fire trucks, where delays can have serious consequences. Using Reinforcement Learning (RL), particularly the Q-Learning method, allows for finding the best paths for vehicles. This reduces travel time and minimizes traffic jams. This research uses Python programming and the Simulation of Urban Mobility (SUMO) software to simulate traffic situations and assess how well the proposed method works. It offers practical solutions to improve urban mobility and public safety.

The objective of this research is to design and evaluate a reinforcement-learning-based vehicle routing model capable of reducing travel time and alleviating congestion in urban environments. Specifically, this study investigates how a Q-Learning algorithm can utilize real-time traffic information obtained from SUMO to dynamically optimize vehicle routing. The research has three core goals: (1) to develop a lightweight RL-based routing framework suitable for real-time deployment; (2) to evaluate its effectiveness in reducing travel time, waiting time, and congestion under varying traffic conditions; and (3) to compare its performance against conventional shortest-path routing approaches. These objectives address the growing demand for intelligent traffic management systems within the context of increasingly complex smart-city environments.

4.2 Literature Review and Related Work

In recent years, reinforcement learning has found its way into a variety of transportation challenges, such as route planning, autonomous navigation, lane-changing decisions, and optimizing traffic lights. While these studies may vary in their focus, they all reflect a growing enthusiasm for leveraging RL to enhance traffic flow and alleviate congestion. This review brings together significant contributions in the field, compares different methodological approaches, and points out the limitations that inspire the creation of our unified RL reward model.

The question of the fastest way across a map is one of the main keys in route planning. When the fastest road has a large traffic jam, choosing the second-fastest road is still a great way to tackle transportation congestion. Research was conducted to determine the second fast route-based waiting times of vehicles near the traffic lights as a reward in the study proposed RL. The research was applied to the SUMO. In the study, the number of vehicles, traffic lights, dead ends, and distance between the start and end points were not considered. However, in our proposed RL model we are regarded as an associated and single reward for each path on a map [106].

In [119], a reinforcement learning (RL)-based method was suggested to determine the optimal path for a robot on a map. The study defined two random points as the start and the target locations. The robot was then assigned the mission of finding the optimum path while ignoring the obstacles. Additionally, the researchers tried to perturb the system by increasing the number of obstacles and analyzing the changes produced as a result.

An essential part of autonomous driving systems is automated lane change, which necessitates that cars make snap judgments in unpredictable traffic situations [120]. Agents may be trained to execute these movements with the help of Deep Q-Networks (DQNs), a type of deep reinforcement learning. By combining deep neural networks with Q-learning techniques, DQNs enable automobiles

to engage with simulated environments and discover the optimal lane-changing tactics. This approach improves efficiency and safety by enabling the creation of decision-making strategies that adapt to various traffic conditions. Studies have demonstrated that DQN-based models can outperform traditional rule-based systems, offering more dependable and flexible solutions to automated lane changing issues [121]. It illustrates that making decision plays an important rule in route planer topic.

In the study presented in [109], a reinforcement learning (RL) technique was employed in order to mitigate vehicle waiting times near intersections via optimizing traffic light signals. As the adaptive signals of traffic lights can have positive effects on managing the traffic congestion and optimizing the road traffic flows [122]. The study did not take into account key factors such as the number of traffic lights, dead ends, and the distance to the destination on each current edge as a unique reward, which are crucial elements in rectifying traffic congestion effectively.

Multi-agent approach was utilized in a paper in order to investigate the fastest path on a SUMO map with the help of Q-learning method. In the paper, the multi Q-table was used instead of one single Q-table that tables were synced or connected to each intersection on the map. In the paper base on each junction, they created a Q-table and all outgoing edges from each intersection play Q-value in the Q-table.

However, the multi-agent approach requires substantial hardware resources. It is also important to note that the fastest route may not always be the most optimal path on the map, as it might include longer edges, leading to higher emissions and reduced energy efficiency. While the referenced paper considers travel time, similar to our proposed RL model, our study incorporates multiple factors into a unified reward function. These factors include the number of vehicles, traffic lights, avoidance of dead-ends and loops, and minimizing the distance to the destination [29].

In [26], it illustrates that traffic lights have a major role in road traffic congestion. The paper utilized a combination of a Q-learning technique and a deep neural network as the proposed model. The model enabled traffic lights to communicate with each other to better manage the traffic density in the street. It results in average vehicle wait time being reduced on roads [28].

This review of studies shows different ways that reinforcement learning (RL) addresses traffic problems. Some focus on finding better routes or faster travel, while others look at self-driving cars, lane changes, or smart traffic lights. Many methods still depend on simple rewards like wait times, obstacles, or travel time. Few studies use many factors for improvement, and they often miss things like traffic light density, avoiding dead ends, changes in car numbers, or destination distance. This lack of a complete reward system is a problem in the current research. It shows a need for a better RL model that combines traffic and road features for decision-making.

In this study, we proposed a reinforcement learning (RL) model that addresses a gap in previous research—specifically, the use of multiple conditions or parameters as input to the RL model. In contrast, earlier studies did not incorporate these metrics in their approaches. To evaluate the effectiveness of our model, we compared it with eight other studies focused on path planning, each of which used various metrics in their analysis. The results of this comparison are presented in

Table 4.1.

Table 4.1: *Comparison of Articles Regarding Q-learning Metrics Used in Their Studies*

#	Article Title	Increasing Vehicle Count Metric	Increasing Traffic Lights Count Metric	Loop and Dead-end Avoidance Metric	Minimum Number of Edges Metric	Distance Metric	Travel Time Metric
1	Our proposed RL	Yes	Yes	Yes	Yes	Yes	Yes
2	Article No.[115]	No	No	Yes	Yes	Yes	No
3	Article No.[28]	Yes	No	No	No	No	Yes
4	Article No.[24]	Yes	No	No	No	No	Yes
5	Article No.[123]	Yes	No	Yes	No	Yes	Yes
6	Article No.[116]	Yes	Yes	No	No	No	No
7	Article No.[29]	Yes	No	Yes	Yes	No	Yes
8	Article No.[124]	Yes	No	No	Yes	No	Yes

4.3 Research Gap

One of the big challenges in metropolitan cities is path planning. It means finding out the shortest path. For most mobile robots, the optimal path is the shortest path between two locations on a map. The majority of the drivers look for the path with minimum travel time on roads [125]. In navigation on a map, the goal is to discover the shortest path on the street. Due to traffic congestion, where traditional traffic light infrastructure was utilized to control the traffic flow in cities with high populations, it was not successful in the past [25]. Moreover, the traffic congestion causes an increase in delay time for vehicles behind the traffic lights as well [24], [26]. However, the shortest path on the map in some cases will be very complicated and expensive as well. The goal is the optimum path will be demonstrated by some solutions [107].

Even though shortest-path algorithms are commonly used in navigation systems, they often fall short when it comes to handling the real-time complexities of urban environments. Factors like fluctuating traffic density, numerous traffic lights, dead-end streets, and the unpredictable movement of vehicles can make a big difference. As a result, these methods can lead to less-than-ideal routing choices, which can increase travel times and contribute to congestion. This study focuses on developing a reinforcement learning–based algorithm capable of identifying the truly optimal route in a complex urban traffic network by considering multiple metrics, rather than relying solely on

the shortest path or travel time.

Reinforcement Learning (RL) has been effectively applied to fuel management in hybrid electric vehicles (HEVs). They can employ some sensing technologies such as cameras, sensors, radars and lidars to retrieve the bottleneck on road [30]. In [31] describes the way of energy management of a dual-motor hybrid electric tracked vehicle using double deep Q-learning Network (DDQN). The authors propose a standard model template that includes lateral dynamic, which purpose is to improve fuel consumption performance. A control structure based on DDQN is proposed that corrects problems with overoptimistic policy value estimates of traditional deep Q-learning thus improving the iterative convergence and optimization performance. The study intends to determine unique observation states and employ such as input variables in order to bring out the vehicle's characteristics and use them to compare the advanced algorithms with assistance from deep Q-learning with dynamic programming. Simulation results confirm that the combination of the DDQN method achieve a better fuel economy. Furthermore, the algorithm successfully keeps the battery state of charge SOC as a constant regardless of various initial conditions.

On map cities we have two sets of items, the first nodes or junctions that they can be presented as a list like $J = \{ j \mid j = 1, \dots, n \}$. The second is a set of edges that are placed on the city map such as a list: $E = \{ E_1, E_2, E_3, \dots, E_n \}$. To determine the optimum path on a map we need two points as start and end edges. Also, it is important to know which edges have out-going edges or are dead-ends. This study was implemented based on the edges and junctions were not considered in our investigation. Two critical issues must be reflected in locating the optimum path. Firstly, we might face dead-end edges. Secondly, drivers focus on travel time on roads and it is a vital feature on city maps [27].

It does not find it difficult to select the shortest path on a map without traffic. The point is that based on street traffic, we need an algorithm to extract the optimum path on the streets. In addition, the shortest path is not necessarily the best path on the map. Since the shortest path might be occupied by many vehicles on roads it will be useless. Moreover, in each junction, at least a traffic light is placed. It will have reverse effects on the travel time of vehicles as well. Many routes can be indicated between the start edge and the end edge on a city map. The big question is which one the optimum path is. At the same time, the traffic must be considered as an important factor on roads too. In this survey, the optimal trajectory will be demonstrated by a reinforcement learning Q-Learning algorithm.

Routing optimization in urban traffic networks is challenging since the real-world environment has various dynamics, such as different conditions of traffic, more intersections, and even dead-end edges, more vehicles, more traffic lights and distance on edges [28]. Unusual routing algorithms lack the capability to efficiently address these challenges, especially when real-time adaptability is required. This paper proposes an RL-based approach, using Q-Learning to compute the optimal and shortest path between a defined start and end edge on a SUMO map. The reinforcement learning model will make use of an edge-based state, outgoing edges as actions, and a reward mechanism based on travel time in order to train an agent, that is a vehicle, to navigate the network optimally [29]. The reward of each edge is computed by the travel time, where the dead-end edges are penalized and the goal-reaching ones get a bonus to force the agent to give the highest priority to paths with the least travel time while trying to avoid inefficient routes. This Q-learning approach is dynamic; it updates the Q-table at every step in the training process to find the optimal route. In

this way, the study associates some components consist of travel-time, number of vehicles , traffic lights, edges and distance as a single reward to overcome the shortcomings of static routing methods and enhance path-finding accuracy in complex networks of traffic [112].

This study seeks to create a reinforcement learning routing model. It will find the best route by combining traffic and network data into a single reward function. The specific goals are to: (1) create a Q-learning system that uses edge-based states and outgoing edges as actions; (2) develop a reward function that considers travel time, vehicle count, traffic signals, distance to the destination, and avoidance of loops or dead-ends; and (3) test the model on a SUMO traffic network. This work is limited to simulated settings and focuses on single-agent routing within a fixed SUMO map. Topics like multi-agent coordination, real-world use, and dynamic map generation are not covered here.

The gap in route planning is how the optimal or the shortest path must be chosen based on many metrics that must be considered on a city map comprise travel time, number of vehicles on each desired edge , number of traffic lights on each edge, loop and dead-end avoidance and distance to the destination. So, they will have negative effects on path planner research.

4.4 Approach

In recent years, some companies and municipalities have been interested in utilizing cutting-edge technologies such as artificial intelligence (AI). It is crucial to cope with challenges such as accidents and traffic congestion in their societies [114]. One of the most popular technologies used is reinforcement learning (RL). People prefer to use their private vehicles rather than public transportation. Due to some reasons, such as a lack of a good public transportation system, infrastructure, or punctuality. As a result, the traffic will have negative effects on some resources, such as the environment, fuel consumption, and human health. As a result, it leads to traffic jams on the streets too. We can benefit from RL to address traffic jams on the roads. The most critical factor to address in RL is the prevention of loops in the training process [29]. The proposed RL model takes measures to effectively mitigate the presence of loops during training.

We get our main road data from OpenStreetMap (OSM), which gives us info on road shapes, lengths, intersections, and where traffic lights are. We use SUMO to fake how cars move and create data as travel times, how cars line up, and how packed roads get. TraCI API lets SUMO and our Python-based learning part in Spyder talk to each other both ways. Using these things, we can make traffic situations that can be copied, managed, and have lots of data to test our learning setup.

The environment, current state, new state, an agent and reward are main elements of the RL. The environment is where we work and focus on it , in this case the part of a city map is named the environment. The current state is where the agent start from there(current edge or route on the map) [119].

The agent takes one action to forward on the map to has access to the final destination as the proposed RL goal. Each movement from the current state which will be done by the agent is named the next state at that moment [126]. The agent is an object is used in the environment which on the

map we used a vehicle as an agent. The reward is a values or numbers will be received by the agent based on its actions or each directions chosen by the agent in each edge(route) [121].

This research uses a simulation-based experiment to explore how reinforcement learning can improve routing choices in city traffic. The experiment is arranged in steps to test the influence of actual traffic elements like congestion, lights, and car amount. We picked this design because it lets us control traffic variables in SUMO and watch how the reinforcement learning system changes its approach as situations get harder. The step-by-step method also checks that the algorithm works in perfect settings before trying it in lifelike ones.

We conducted experiments using a section of the Bari city map, which we imported from OpenStreetMap into the SUMO simulator. Depending on the scenario, the environment contained 300–400 edges and a varying count of traffic lights. We set up two scenarios: (1) a baseline with minimal traffic and just one traffic light, and (2) a congested scenario with about 300 vehicles and seven times more traffic lights. Each experiment involved 300–15,000 training episodes, depending on the step. The agent began from a random edge and tried to reach a set destination. We ran all simulations using SUMO–TraCI communication, and we did the analysis and reinforcement learning training in Python.

Generally, the proposed RL consists of some steps that are explained in the following lines:

- Step 1 (Inspect the route planner on the map, which has no traffic) In this phase, a part of the Bari city map plays an environment and all edges on the map will be considered as states. Then, we selected two edges as the start and end edges respectively. The end edge plays the final destination in the proposed RL. The goal is to investigate an optimal path between the start and the final edge based on travel time as a reward in each edge in the route that will be created by the proposed RL model. In other words, each edge that has less travel time will be selected by the proposed RL to be analyzed and chosen as an edge that can be used in the optimal path for its outcome.
- Step 2 (Enhance more vehicles and traffic lights on the map) In the second step, many vehicles about 300 cars and 7 times more traffic lights were added to the environment. As in the first step, the map just had one traffic light. The proposed RL looks for an optimal path between the start and end edges on the map. The investigation was done based on travel time reward as well as the first step. With the help of the proposed RL, the optimal path was extracted from the map. In addition, during the simulation and investigation, two desired edges were selected on the map many times to ensure that the proposed RL worked well.
- Step 3 (Leverage other metrics and associate them into a unique reward for the proposed RL) In the third step, as an important innovation in the study, some metrics such as the number of vehicles on each edge, a few numbers of traffic lights, a few numbers of dead-end edges, a few numbers of edges in the final or outcome, and distance will be considered in the proposed RL model as a single reward in the study. It will be a challenging topic, which is why, at the same time, all metrics must be considered by the proposed RL to find the optimal path on the map. Also, some functions were used in the Proposed RL to mitigate loop in outcomes as well.
- Step 4 (Use trained and collected data from the prior step for route planning) In the last step in the study, all data trained and accumulated from the previous step, such as edges and routes

information will be applied in the route planner as well in the different desired points on the map.

In the RL, a well-known algorithm is used, which is state-action reward-state-action (SARSA). We need some required items such as states, agent, actions, reward and Q-table in the RL Q-Learning that are explained in the next lines.

We picked reinforcement learning, specifically Q-Learning/SARSA, because it works well for problems where you make decisions in a row and learn by interacting with the world. Unlike regular shortest-path methods like Dijkstra or A*, RL can handle changing things like traffic, delays, and unpredictable travel times. Q-Learning is pretty fast, doesn't need a model, and settles down nicely when you have a set number of choices, like on road networks. We used SUMO as our simulator because it models traffic realistically at a detailed level. It lets us control each car through TraCI and gives us real-time info on travel times, car positions, and traffic lights, which we need to teach the RL agent.

First of all, it is mandatory to have an environment that in this study we selected a city map. In the research, a part of the Bari city map is extracted from the Open Street Map (OSM). It is a part of the SUMO software. Then, the map is presented in the SUMO software used as the RL environment. In the following, we benefited from Spyder, which is an open-source Python IDE (Integrated Development Environment), to write an RL Q-Learning algorithm in the Python language. Besides, we utilized another API/library belonging to SUMO called TraCI (Traffic Control Interface) [25] to communicate with SUMO for simulation traffic congestion scenarios.

The first required features in RL Q-Learning are states. All edge IDs (edges) are extracted by TraCI from the map (environment) and they are considered and used as states in the following steps. There are two edges are selected randomly on the map as the start and endpoint (edges) in the study. The start point (edge) plays a current role in the survey. Another item that must be used in the RL is an agent. A vehicle (Car) is considered as the agent. Then, the agent (vehicle) is encouraged to reach the endpoint (edge) as a final destination. The final destination is the goal of the study. The agent tries to have access to the final edges with less waiting time (less travel time) by the proposed RL Q-Learning. In the first step, we used a map that had no traffic. The reward in the first step was calculated based on travel time between current edge and the next edge that is connected to on each junctions. This approach will be used and repeated till the agent can have access to the final edge. The environment in the first step of the study is determined in Figure 4.1 . It has no traffic and just

has one traffic light. All red points are junctions on the map.

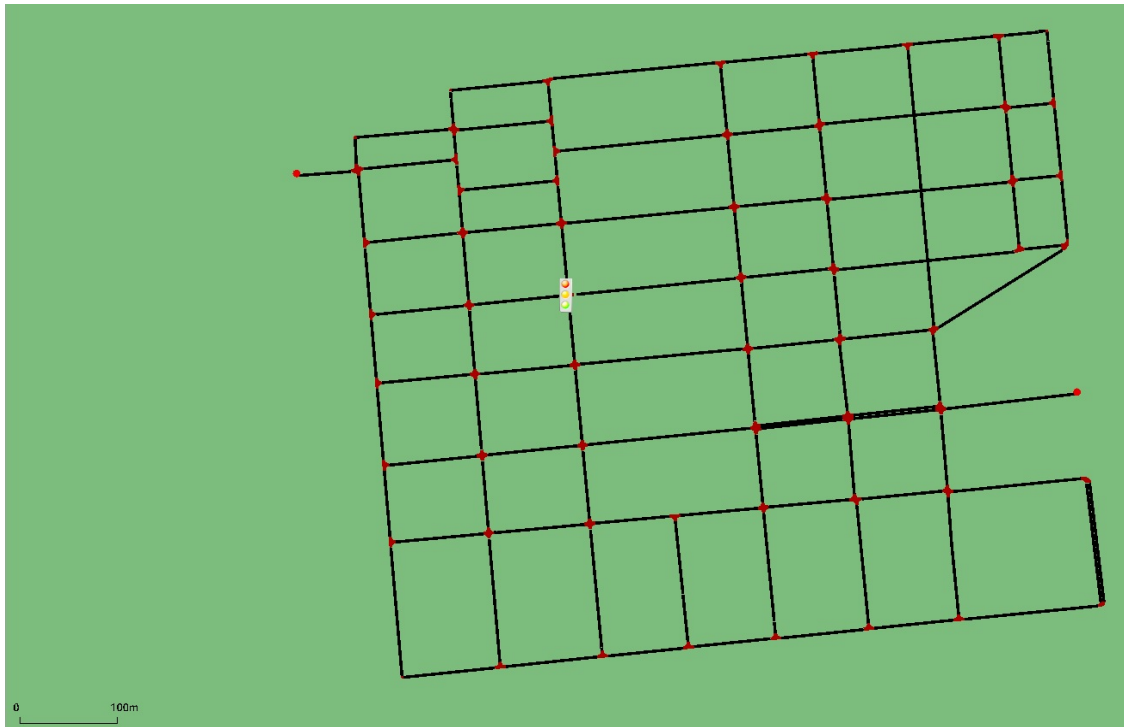


Figure 4.1: *The environment in the first step in the SUMO software*

Then we determined two points on the map and named them as the start and the end edge respectively. As can be seen, the start edge is E7 and the end edge is E8, it can be observed in Figure 4.2 . In the first step, a maximum of 300 iterations were used to investigate the optimal path between the E7 and the E8.

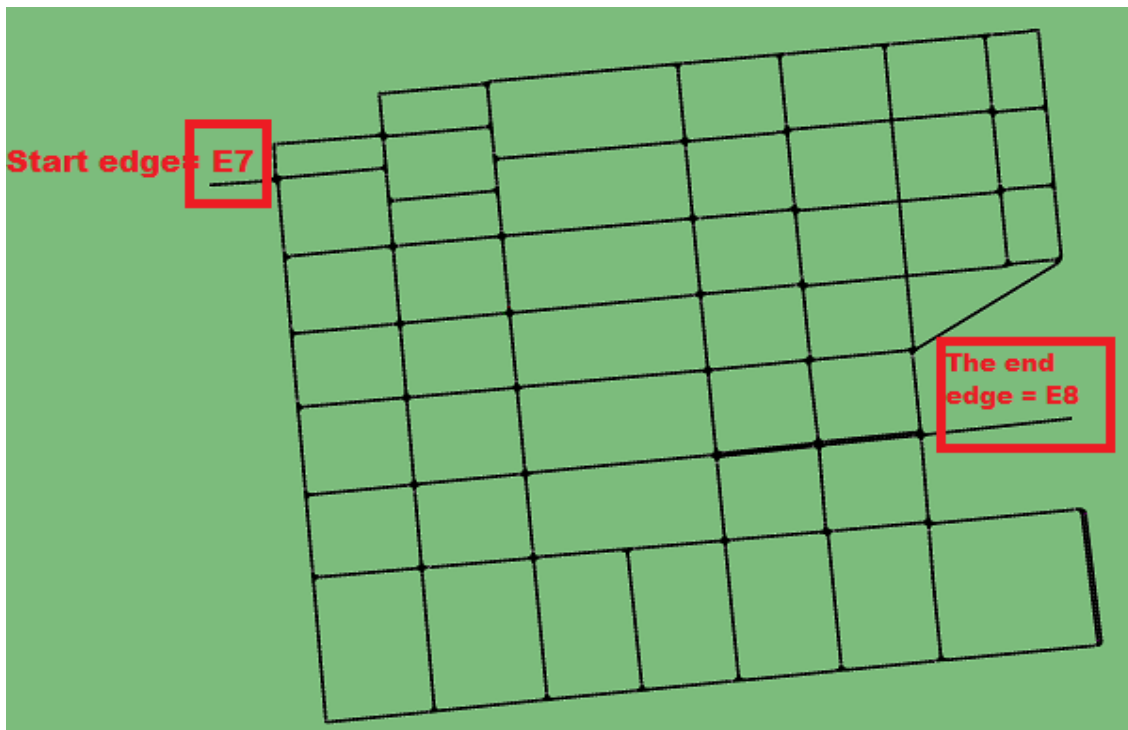


Figure 4.2: The start and end edges on the map in the first step in the SUMO environment.

The study will provide a detailed explanation of the reward function utilized in the proposed RL model. We need another important parameter in the RL called actions. The actions are possible measurements (possible directions can be chosen by the agent in each junction on the map) that can be taken by the agent (vehicle) on the map. It is crucial to know where and how many actions exist on the map (environment) in each intersection. From the prior step, we achieved all edges as states. Then, we need to know the outgoing edges for each edge on the map and how they are called/used as actions. In other words, all possible routes that go out from each edge will be categorized as outgoing edges (actions) for those edges (states). The agent will move ahead randomly based on actions for each edge. The start edge plays a current state edge on the map. To move to the next state, we must know all outgoing edges (actions) for the start edge[126]. Later, if there was/were outgoing-edge(s) for the start edge. The agent must select one of the outgoing edges belonging to the start edge and go there. In this position, before the agent movement, the start edge or outgoing-edge(s) plays action(s) for the agent. After the agent movement, the current edge (state) will be changed to the new edge. This procedure will be done for each edge on the map. With the help of TraCI, all outgoing edges (actions) for every edge will be extracted and saved into a list called outgoing edges. The list will be used in the following steps.

For the states, actions, and agent clarification, an example will be provided. For instance, we have an edge named E2. From the E2, the agent can go to the four different edges (directions): E3, E4, E5, and E6. The four edges play outgoing edges, or action roles, for the E2 edge. As can be observed, this structure, E2 [E3, E4, E5, E6], will be used for all edges and their outgoing edges on the city map. The outgoing edges are appeared in Figure 4.3 [112] [124] [127]. In other words, E3, E4, E5, and E6 are possible actions for the E2 edge [110]. The approach on each intersection will be used and iterated from the desired start to the end edge. To put it another way, the method will be iterated and continued by the proposed RL model till the agent can have access to the end edge. Then, the proposed RL model will start to investigate the optimal path. The optimum path will be

chosen among the routes that were built during the training procedure with the help of all edges and their rewards. All edges on the map will be extracted and listed in the prior style. The proposed RL model will use this style to discover outgoing edges that are connected to each edge. Also, some rules can be used in the proposed RL model so that we can select an edge that has the lowest travel time in an optimum route [118]. Also, the proposed RL algorithm is reported in Figure 4.4. It must be considered, at the beginning, the Q-table values are zero in each row and column before selecting the two random desired edges as the start and end edges.

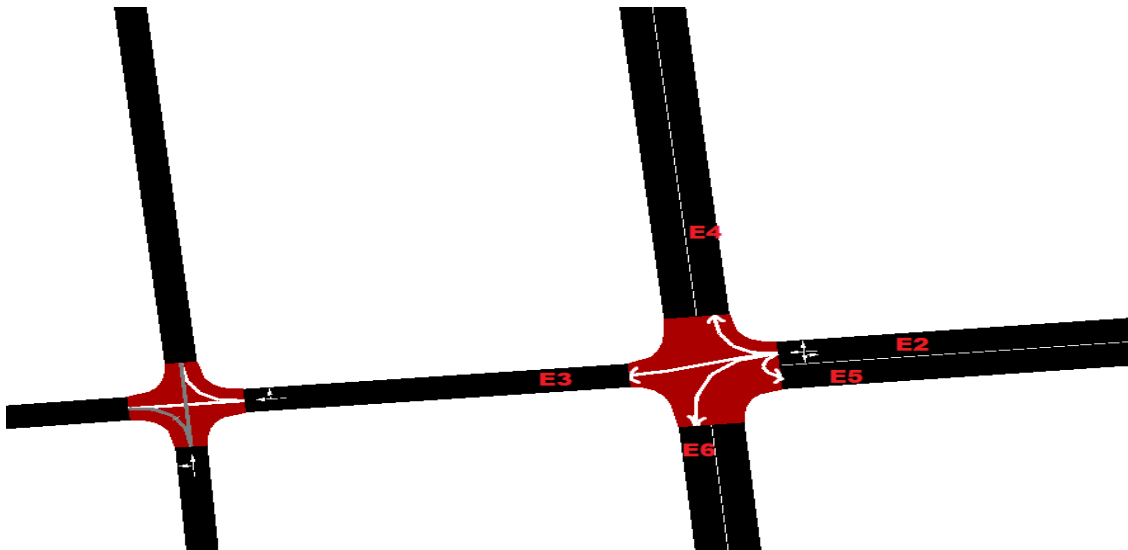


Figure 4.3: *The Outgoing edges for edge E2*

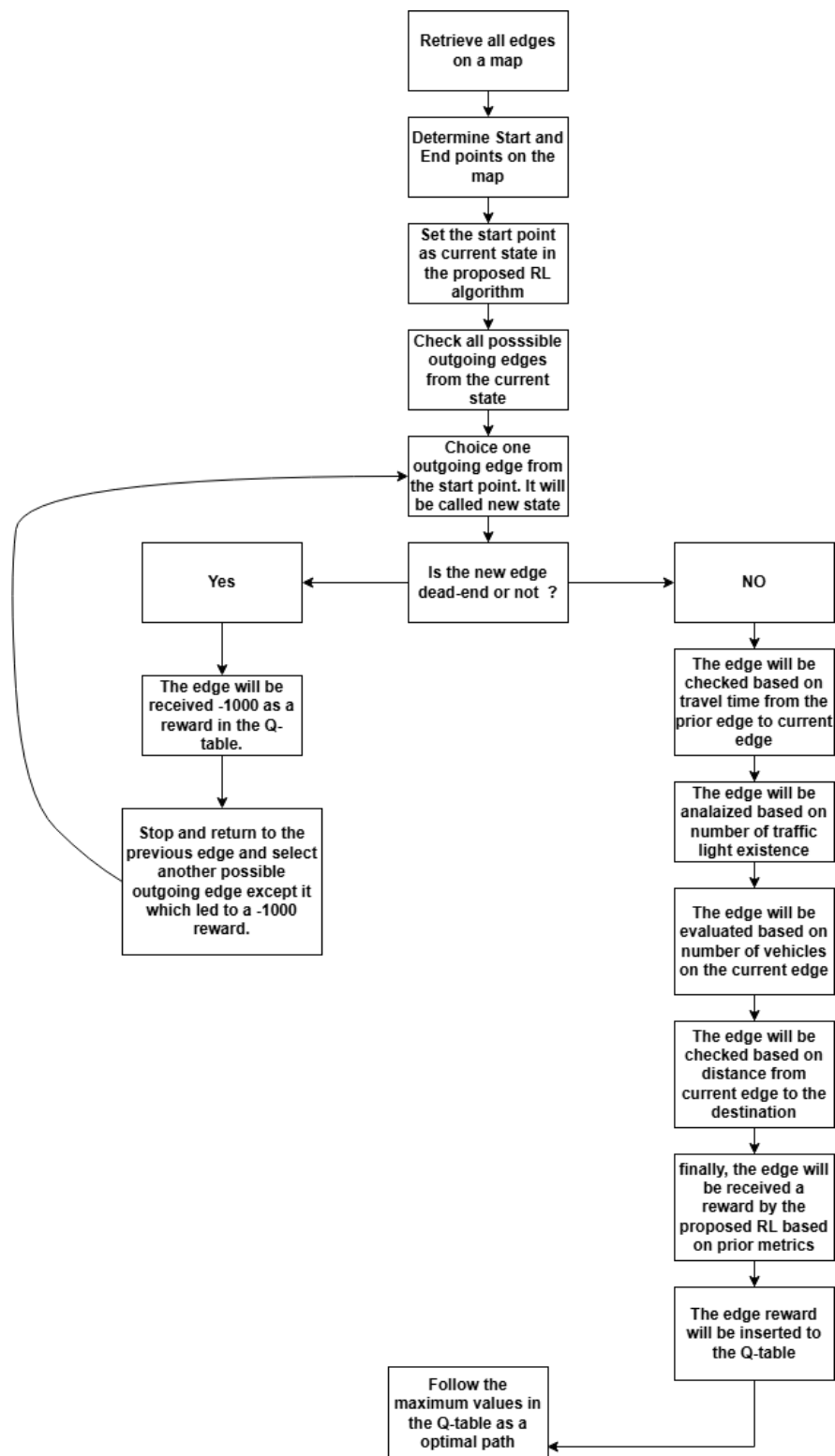


Figure 4.4: *The proposed RL algorithm*

The following figure shows that each edge step by step has been selected in the proposed RL model training process. The training process for achieving the optimal path as an outcome is noted in Figure 4.5. For instance, in this case, the starting point started from the E7. First of all, the proposed RL model retrieves all outgoing edges that belong to the E7. The outgoing edges for the

E7 will be in this style, like E7 [30016109#1, 30016109#2, 30016109#3 and...etc]. The proposed RL model tries to select an edge among them that has less travel time.

A vehicle was used as an agent and will be sent by the TraCI command in the Spyder(a Python IDE) for the simulation. The agent needs some parameters to be defined in the simulation. However, we defined two important parameters for it, the first was vehicle ID, and the second was vehicle route. An agent with ID 90 (v-90) was determined and sent to simulation by TraCI. Nevertheless, many routes will be built by the proposed RL model in each iteration and all created routes will be followed by the agent. The route ID was set as R-10 in the proposed RL. Various roads will be made by the RL creation route function, and they will be saved into an empty list too. Every route includes many edges. The agent will visit the edges of each route. The visited edges by the agent will be saved in another empty list.

Now, the travel time calculation procedure will be explained in the following lines. The RL must reward the edges that were visited by the agent on the routes. The agent current position on each edge will be recorded into an empty list. Based on the waiting time that the agent spends on each edge, the reward function will calculate each edge reward. For instance, for the edge E2, we have these time steps [25,26,27,28,29,30], in this case, we have a sequence time steps list, which means that the agent visited this edge and it took 5 seconds($30-25 = 5$ seconds) during the move from it to another edge (For example, from E2 to E3). Thus, the reward for the E2 edge will be -5 seconds. Also, all rewards for each edge will be calculated based on the travel time that the agent spends on it. In some cases, we have different time steps that do not have sequence numbers. For example, the edge E14 has this time steps [45,46,47,48,49,50,71,72,73,94,95,96,114,115,116,117]. In that case, there are no sequences in time steps for the edge. Then, we must extract time steps that can be categorized in the same sequence. It means that all sequence time steps must be sorted into a list with some items that have the same sequence time steps, it can be seen, in the edge E14, it is done and $E14 = [[45,46,47,48,50],[71,72,73],[94,95,96],[114,115,116,117]]$. Then, we can calculate the time steps for each list in the main list via the prior method. It means that subtracting the last item in each list from the first item such as an equation will be like the E14 reward will be $[[50-45],[73-71],[96-94],[117-114]] = 12$ and the result convert to a negative value thus the reward will be -12 for the edge E14 [110].

The prize for having access to the final edge, as our desired edge, is a positive number, +1000. A route will be built by selecting all possible outgoing edges from the start point edge, and the procedure will be continued until the last edge in the route faces an empty outgoing edge. Creating a route based on all edges will be done in each iteration, and each edge in each route will have a reward. In the iteration, it can be seen that a route will be found to have access to the last edge among all the routes that are created during the iterations. Every edge will receive its reward based on its waiting time on the edge (travel time). It means that the agent will visit each edge of a route. Based on the waiting time on an edge, the edges will receive the rewards. The rewards are changed to negative numbers and dedicated to the edges. There are minimum and maximum rewards of -1000 and +1000 respectively [112]. If the edge has no outgoing edge, it will achieve -1000. The -1000 plays a penalty for dead-end edges on the map. For instance, edge E12 on the map does not have outgoing edges and it will be shown in this structure, $E12[]$, and has an empty outgoing edge. However, for instance, edge E2 includes four outgoing edges [E3, E4, E5, E6] that it is revealed in Figure 4.3. The waiting time for visiting all edges in the edge E2 will be calculated and dedicated to the E2 reward based on the second. This mechanism helps the RL not select the edge that has no

outgoing edges. Finally, The final edge will achieve +1000 as a reward plus its travel time that is calculated during reaching the end edge. The reward function algorithm is shown in the following formula.

$$\text{Edge}_{\text{Travel-time-Reward}}(t) = \begin{cases} t[-1] - t[0] = \text{result} \times (-1) & \text{if sequence time steps,} \\ \text{Sum}[t[-1] - t[0]] = \text{result} \times (-1) & \text{no separated: set sequence time steps for each} \\ +1000 & \text{if the last edge,} \\ -1000 & \text{dead-end edges.} \end{cases}$$

Moreover, Table 4.2 presents the configuration used for the proposed reinforcement learning model, which was designed to improve the study results. The parameter ranges were chosen through a process of trial and error, helping to recognize the terrific outcomes [128]. Additionally, the table outlines a custom reward function that employs a balanced reward strategy. This strategy takes into account several factors, including the number of vehicles, traffic lights, edges, the minimization of dead-ends, as well as distance and waiting time, which are described in detail below:

Table 4.2: *Q-Learning Configuration Parameters*

Parameter	Value Range
Learning rate (α)	0.1 - 0.8
Discount factor (γ)	0.95 - 0.99
Exploration rate (ϵ)	0.7 (initial) - 1.0
Exploration decay rate	0.995 - 0.998
Minimum ϵ	0.01 - 0.05
Number of episodes	10000 - 15000
Maximum steps per episode	200
Q-table initialization	Zero matrix
Reward strategy	Custom reward function

Learning rate (α): Controls how much new information overrides old information in the Q-table updates; a value between 0 and 1.

Discount factor (γ): Determines the importance of future rewards; a higher value makes the agent prioritize long-term rewards.

Exploration rate (ϵ): Governs the probability of choosing a random action over the best-known action; encourages exploration early in training.

Exploration decay rate: Reduces ϵ gradually after each episode to shift from exploration to exploitation over time.

Minimum ϵ : Defines the lowest possible value for ϵ , ensuring the agent continues minimal exploration.

Number of episodes: Total number of training cycles the agent runs through; each episode represents a full trial from start to end.

Maximum steps per episode: Limits how many actions the agent can take within a single episode to prevent infinite loops.

Q-table initialization: Describes how the Q-values are initialized before training; often with zeros or small random numbers.

Reward strategy: Specifies the custom reward function used to guide learning, balancing reward based on the number of vehicles, traffic lights, edges and distance from initial edge to the end edge and minimum travel time. Also we are able to add more items that can have positive and

negative effects on the outcomes like slop , wind and etc. However, the SUMO simulation dose not support slop as it is a 2D environment simulator.

Meaning, among all outgoing edges from the E7, the 30016109#2 was selected by the proposed RL model. It means the travel time from the E7 to the edge 30010109#2 is less than others([30016109#1, 30016109#3 and...etc]). It can be imagined that the trip might take 12 seconds from the E7 to the 30016109#2. Based on the travel time, which was 12 seconds, the reward will belong to a route from the E7 to the 30016109#2. The reward for each edge will be converted to a negative value like -12 -in this example-. Then, it will be inserted into the Q-table as a reward. This method will be followed by the proposed RL model training to give a reward to each edge on the map which will be visited by the agent during the training procedure. Besides, when the agent is able to reach to the final edge (the E8) , the reward will be, sum of the travel time reward (a negative reward based on travel time between two edges) from prior edge of the final edge (the edge before the final edge) to the final edge plus the special reward that can be set in the study. In this case, +1000 was decided for having access to the end (final) edge (the E8).

According to the figure, the travel time for commuting from one edge to the second edge will be analyzed and calculated by the proposed RL functions. It means that the travel time from edge E7 to edge 30016109#2 is calculated and its reward will be inserted into the Q-table row which is related to these edges. Next, the travel time for the 30016109#2 to the edge 30016109#3 will be computed and as a negative reward will be stored in the Q-table row which is connected to this short trip between the two edges and, so on, for each route which were built during the training process. The reward in Q-learning can be positive and/or negative. In the study scenario ,all edges will receive negative rewards based on their travel times. There is an exclusion when the edge before the final edge leads to reaching the final edge on the map, which will achieve a negative reward and an extra positive reward that is +1000 [111]. The outcome is based on travel time rewards as well. This indicates that, after 300 training iterations on the map, the displayed route in the image is the best route in the travel time aspect, between the E7 and E8 edges based on travel time reward in the suggested RL model in the first step of the study [117].

```

In [2]: play()
[]
['30016109#2']
['30016109#2', '30016109#3']
['30016109#2', '30016109#3', '30016109#4']
['30016109#2', '30016109#3', '30016109#4', '30016109#5']
['30016109#2', '30016109#3', '30016109#4', '30016109#5', '29975387#0']
['30016109#2', '30016109#3', '30016109#4', '30016109#5', '29975387#0', '29975387#1']
['30016109#2', '30016109#3', '30016109#4', '30016109#5', '29975387#0', '29975387#1', '29975387#2']
['30016109#2', '30016109#3', '30016109#4', '30016109#5', '29975387#0', '29975387#1', '29975387#2', '-E19']
['30016109#2', '30016109#3', '30016109#4', '30016109#5', '29975387#0', '29975387#1', '29975387#2', '-E19', '-E18']
Out[2]:
['30016109#2',
 '30016109#3',
 '30016109#4',
 '30016109#5',
 '29975387#0',
 '29975387#1',
 '29975387#2',
 '-E19',
 '-E18',
 '-E8']
In [3]:

```

Figure 4.5: The training process for finding the optimal path in the first step. of the study.

Then in following , one important function in the proposed RL model is shown in following lines where the edge reward formula is designed to evaluate the desirability of a specific edge within a route by considering multiple factors that influence travel efficiency and practicality. The key components of the formula are as follows: $T(e)$ represents the travel time for edge e , accounting for how quickly a vehicle can traverse that segment. $D(e, e_{\text{end}})$ measures the Euclidean distance between the midpoint of edge e and the midpoint of the destination edge e_{end} , prioritizing edges closer to the destination. A traffic light penalty $P_{\text{TL}}(e)$ applies a significant negative weight (-1000) to edges with traffic lights, discouraging their selection due to potential delays. Similarly, a dead-end penalty $P_{\text{DE}}(e)$ penalizes edges that lack outgoing connections to avoid routing into dead-ends. The term $10 \cdot i$, where i is the edge index in the route, serves as a step penalty, discouraging overly long paths. Lastly, the route efficiency bonus $B(\mathcal{R})$ is a positive adjustment that rewards shorter and more efficient routes, incorporating factors such as the total route length $|\mathcal{R}|$, total travel time T_{total} , and total distance L_{total} . For the destination edge e_{end} , a fixed reward of $+2000$ is assigned, ensuring the algorithm prioritizes reaching the destination.

The reward for an edge e in a route $\mathcal{R}$ is given by:

$$R(e) = \frac{300}{1 + T(e)} + \frac{500}{1 + D(e, e_{\text{end}})} - P_{\text{TL}}(e) - 10 \cdot i - P_{\text{DE}}(e) + B(\mathcal{R}),$$

where:

$T(e)$: travel time for edge e ,

$D(e, e_{\text{end}})$: Euclidean distance between the midpoint of edge e and the midpoint of the destination edge e_{end} ,

$$P_{\text{TL}}(e) = \begin{cases} -1000, & \text{if edge } e \text{ has a traffic light,} \\ 0, & \text{otherwise,} \end{cases}$$

$$P_{\text{DE}}(e) = \begin{cases} -1000, & \text{if edge } e \text{ is a dead-end,} \\ 0, & \text{otherwise,} \end{cases}$$

i : index of edge e in the route $\mathcal{R}$,

$$B(\mathcal{R}) = \frac{1000}{|\mathcal{R}| + T_{\text{total}} + L_{\text{total}}},$$

$|\mathcal{R}|$: length of the route,

T_{total} : total travel time of the route,

L_{total} : total length of the route.

For the destination edge e_{end} , a constant reward is assigned:

$$R(e_{\text{end}}) = 2000.$$

300: This constant is used to calculate the time reward. Also, it leads to selecting low traffic or low traffic edges in the proposed model. The weight for it was chosen 300 that is why to balance the importance of different terms in the reward function compared with 500 that is the importance of being close to the goal is more heavily rewarded. Also, this constant based on tune a model can be changed to reach to the best value.

$$\frac{300}{1 + T(e)}.$$

It ensures that edges with lower travel times are rewarded more significantly, thereby prioritizing routes that minimize time. The value 300 sets the scaling factor for how much the travel time impacts the reward, balancing its influence relative to other terms in the formula.

500: This constant is used in the distance reward

$$\frac{500}{1 + D(e, e_{\text{end}})}.$$

It emphasizes the importance of proximity to the destination, favoring edges that are closer to the destination edge. The value 500 determines the weight of the distance factor, ensuring it has a substantial but not overwhelming impact compared to other considerations. Also, this constant based on tune a model can be changed to reach to the best value.

2000: This is the fixed reward assigned to the destination edge e_{end} . It acts as an incentive to prioritize reaching the destination, ensuring the algorithm recognizes the goal and evaluates paths accordingly. The large value of 2000 reflects the critical importance of arriving at the destination in the overall routing strategy.

We have a Q-table in the Q-Learning RL. In the Q-table based on the number of edges and directions from each edge on the map, we have rows and columns respectively. Each row is

connected to an edge. Moreover, each column in each row contains possible actions (directions) from the edge. At the beginning, a maximum number of outgoing edges for an edge must be extracted from the map. Then, we must have Q-table rows connected to the number of edges on the map. Also, several columns are related to the maximum number of directions on the edge of the map. Based on the number of rows and columns, a Q-table must be built. At the start, the Q-table is a zero matrix that includes rows equal to the number of edges on the map and columns equal to the maximum number of actions (outgoing edges) for an edge. For example, we have an edge on the city map which has four outgoing edges, which is the maximum number of outgoing edges in the whole map. Thus, the Q-table related to this map will be a zero matrix that contains the number of edges as a row multiple by four columns. All contents of the Q-table in rows and columns are zero. The Q-table contents will be updated and changed by each edge reward that will be visited by the agent, in every iteration.

In the RL, we have a Q-table connected to each state, and in the study, all states are edges on the map. In other words, all rows in the Q-table are connected to each edge on the map. Based on the agent actions (directions or edges) that will be opted by the agent in the current edge (in each junction) in each level of iteration. Actions will be taken by the agent, and then the selected edges reward will be filled -based on the agent-selected actions (direction)- in the Q-table as values or numbers. The reward values will be analyzed and calculated using the proposed RL model functions and inserted into the Q-table. The Q-table in the first step of the study is indicated in Figure 4.6. At the end of the training iterations, the proposed Q-Learning algorithm can discover the optimal path based on all maximum numbers in the Q-table rows [122].

The optimal path will be built by choosing the highest numbers in the Q-table. In other words, the highest numbers will be taken by the proposed RL model for each row. Then the agent must follow them. Hence, the Q-table rows are connected to each edge on the map. The agent follows edges on the map when it tracks the highest numbers in the Q-table after each training iteration [129]. The agent will follow the optimal path as the proposed RL model outcome. During the training procedure, the agent meets many edges on the map in the proposed RL model. The visited edges form various routes on the map in each iteration. In the end, all formed routes from the training section will be analyzed and calculated based on their edge rewards. To discover the optimal path among them. It can be seen, in the following image in the output section, that the optimal path will be presented as a path between the E7 and the E8. It comprises some edges. In the end, the optimal

path as an outcome is displayed in Figure 4.7 in the first step of the study.

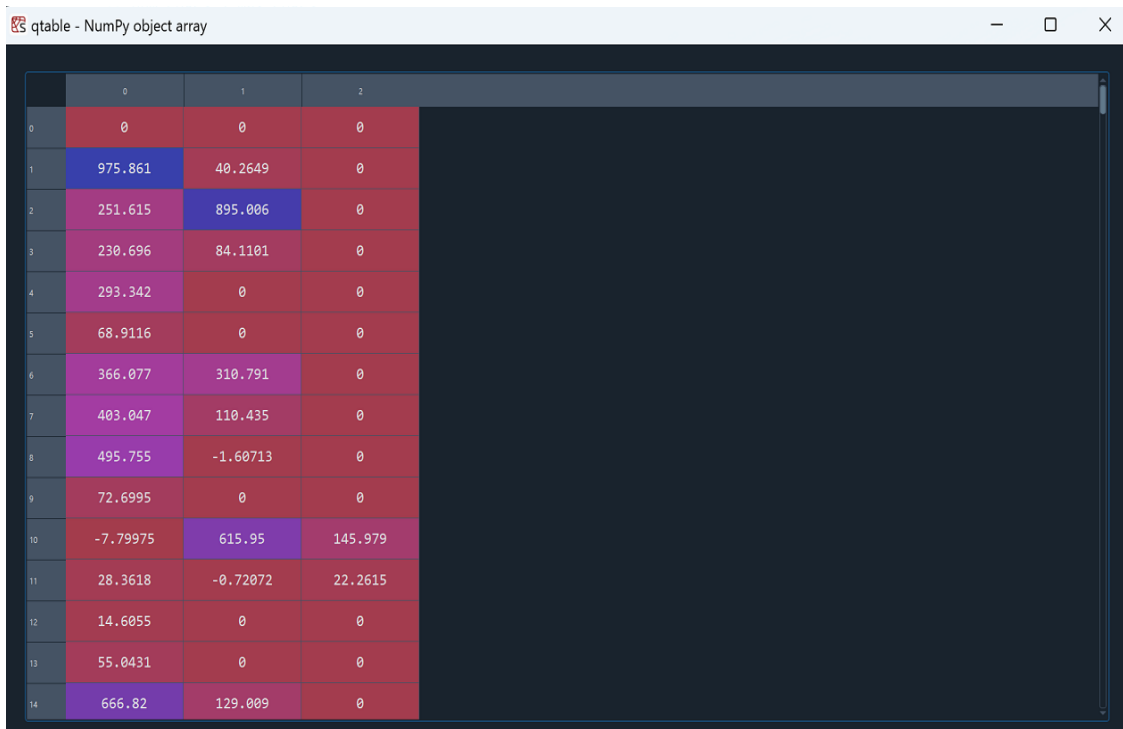


Figure 4.6: The Q-table items

As an example , we run a simulation to reach an optimal path between two random edges E7 and E17 on the SUMO map. and the outcome is reported in Table 4.3. Overall, the Table 4.3 demonstrates that the agent makes substantial progress over 15 000 episodes. Also, on the one hand, the average route length drops from roughly 38 edges to just over 20 edges (a 46 % reduction), on the other hand, the mean episodic reward climbs from -173 k to -46 k (a 73 % improvement). At the same time, the success rate rises from about 30 % to approximately 52 %, showing that more than half of all trials now reach the goal. It must be considered in some cases, for by increasing the episode the outcome can show some noise or error. As, more than one feature was used in the proposed reinforcement model reward function.

Table 4.3: As an example: training outcomes between 1 and 15 000 episodes for a desired path among two random desired edges on the map .

Metric	Episode 1	Episode 15 000	Absolute Change	Relative Change
Route length (edges)	37.96	20.33	-17.63	-46.5 %
Episodic reward	-173 056.03	-46 254.56	+126 801.47	+73.3 %
Success rate (approx. % of runs)	30 %	52 %	+22 pp	+73.3 %

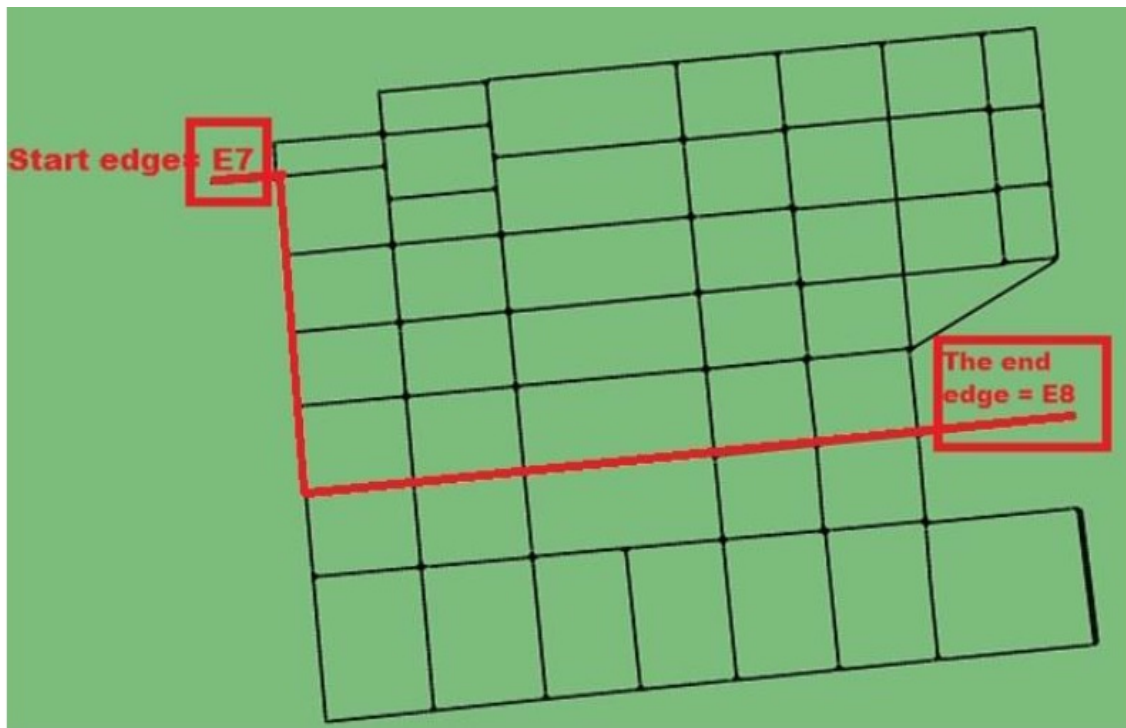
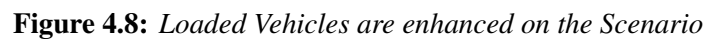


Figure 4.7: *The firts step outcome with the help of the proposed RL model.*

It is crucial to know that the agent will select the edges randomly. As a result, in some cases, it cannot have access to the goal. It may face dead-end edges. It means that the agent might select an edge, and there is no outgoing road for the selected edge. Thus, the agent will stop in that position, and the RL will try to create another route to be tracked by the agent again. This procedure will be continued by the proposed RL model until the agent can access the goal.

In the training procedure, the proposed RL model aims to effectively train the agent to find the shortest path between two selected edges on the SUMO map. This means that multiple routes will be generated by the RL Q-Learning algorithm and followed by the agent [117]. The visited edges will receive rewards from the RL model. Since all edges are synchronized with the Q-table, when the agent follows a route, the reward corresponding to this movement (linked to the Q-table) will update the Q-table row that corresponds to the edge. The rewards for all visited edges in the route will be calculated and inserted into the Q-table.

Besides, in the event that the intended aim cannot be reached within the specified RL model, one can always seek measures to utilize some parameters to better the RL Q-learning results; for instance, adjusting hyperparameters. These tweaks can be achieved in RL by changing the learning rate, the number of iterations, and the exploration and reward strategies [28] ,[118] [117] . Also, in the second stage, the quantity of traffic and vehicles is multiplied on the same map that was utilized in the very first phase. The number of cars went up to 450, which is about three times greater than the first 150 cars that used at the beginning step, as can be seen in Figure 4.8.



In the third phase of the study, multiple metrics are used in the route planner to identify the

optimal path between two points on the city map. Typically, in reinforcement learning (RL), a single metric is used as a reward. In the first phase of the study, travel time was chosen as the reward. However, in this phase, several metrics are combined into one. The factors considered in the final route include the number of vehicles on each edge [26], traffic lights, avoidance of dead-ends and loops, and minimizing the distance to the destination [29]. To calculate these metrics separately, functions within the RL model were utilized. Then, each metric was assigned a unique reward. Specifically, the number of vehicles on each edge was analyzed by the RL model, and when this goal was reached, it contributed to the reward. Additionally, traffic light count was factored in as another reward in the RL model. The prevention of dead-end edges was also incorporated as a metric that contributed to the reward. Following this, the number of edges and distance were used as rewards as well [124]. Finally, all routes generated by the RL model were evaluated based on the aforementioned metrics. The result was a value added to the Q-table, and the remaining approach followed the same process as the first phase of the study.

During the final stage (the fourth phase) of the research, all edge values will be applied to the proposed RL model to determine a path between the two selected edges on the city map. It should be noted, however, that the path generated in this stage may not necessarily be the most optimal for the specified points.

The data saved and recorded from the earlier phases—particularly from the first to the third phases—serve as optional features in the proposed RL model. This allows the model to generate a path between any two random edges on the map.

This study incorporates parameters that can influence the outcome, such as vehicles, traffic lights, dead ends, and set distances—factors that contribute to delays. By considering these parameters, it is clear that there is a gap in real-time optimal path-finding applications. These factors are essential when developing RL-based solutions for optimal path finding on city maps.

4.5 Findings and Implications

In this section, the article results are presented and explained. It is crucial to note that the environment is one of the most important factors in the Q-Learning study. For the simulations, an environment with a higher number of vehicles and traffic lights was chosen. Two simulations were conducted to ensure the proposed Q-Learning approach effectively identifies the optimal path from the start edges to the end edges on the city map. In the first attempt, two edges were randomly selected as the start and end points, namely E7 and E8. Both simulations utilized a traffic scenario with 450 vehicles in the environment.

We conducted multiple iterations to discover the optimal path between randomly selected start and end edges on the map. This was employed using Spyder, a Python IDE, and the results are presented in Table 4.4. The table also provides data showing how performance evolves with an increasing number of iterations. The algorithm helps to recognize a path that has minimum number of edges, traffic lights, and dead ends, while also reducing overall travel time and distance.

The data demonstrates that the proposed reinforcement learning approach effectively finds

optimal paths between two selected edges on the map. It shows our objective of determining routes with minimal waiting time for the agent. In real-world scenarios, such an agent could represent an ambulance, police vehicle, or fire truck benefiting from this approach particularly valuable in emergency or critical conditions where rapid response is essential. It is important to note, in some cases, the number of episode and reinforcement learning parameters need to be changed to achieve the optimal result.

Table 4.4: *Performance of the proposed algorithm over 15,000 training episodes for an optimal path between two desired edges on the SUMO map*

Episode Range	Avg. Route Length	Avg. Travel Time (s)	Avg. Traffic Lights	Avg. Dead Ends	Avg. Distance (m)	Path Efficiency	Success Rate (%)
0–2999	40.1	180.0	7.0	2.4	860.0	0.48	27.2
3000–5999	32.5	155.0	5.5	1.7	780.0	0.58	42.6
6000–8999	26.3	130.0	4.0	1.1	710.0	0.67	59.3
9000–11999	22.4	110.0	2.7	0.6	650.0	0.74	73.5
12000–14999	20.3	98.0	1.8	0.3	590.0	0.79	81.6

The outcome is an optimal path consisting of some edges that are kept in a list. Then, we are able to visualize the path by the SUMO. The outcome is figured out in Figure 4.10. As can be seen, the start and end edges are characterized by two red rectangles, and the agent is marked by a blue circle around the vehicle in the image. In addition, the path is highlighted on the map by a red color path from the start edge to the end edge. The proposed RL demonstrated the optimal path between the two edges. The waiting time, CO₂ emission and fuel consumption for the optimum path is in the minimum rate on the map [123]. It might exist some other paths with the situations on the map, however, the proposed RL model was able to figure out one of them. If iteration is enhanced, other optimum paths can be detected by the proposed RL on the map as well and it needs to take more time.

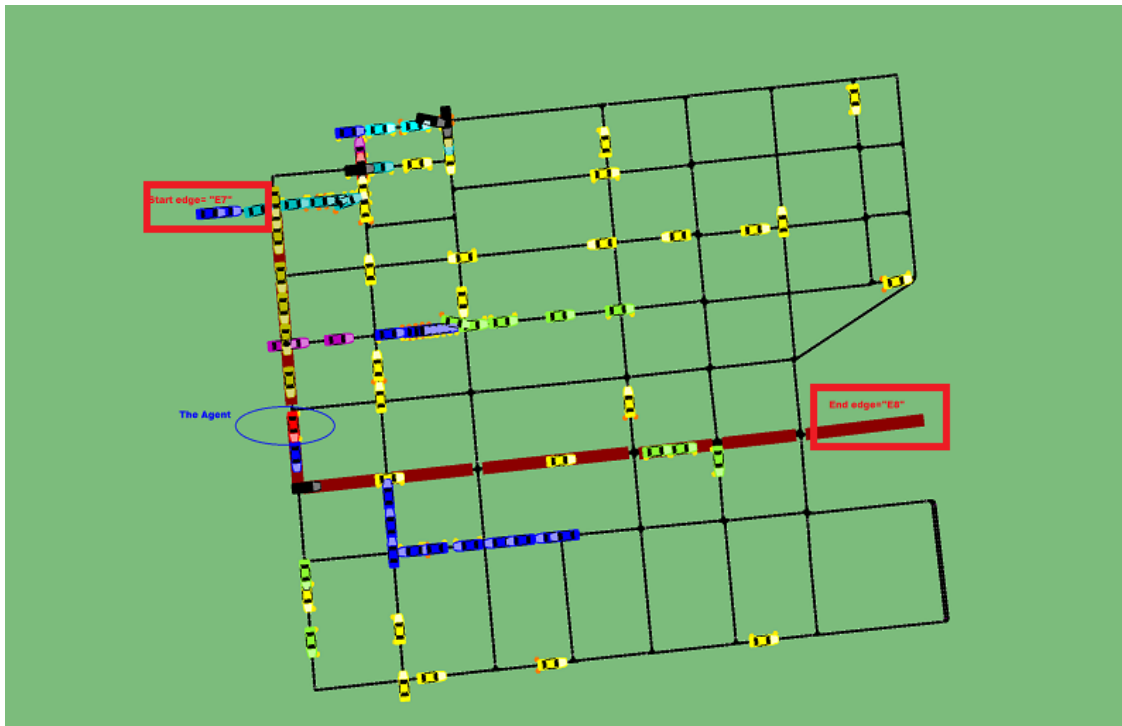


Figure 4.10: *The outcome path as an optimal path by RL*

On a second try, the traffic size and number of traffic lights are the same as the prior try which includes 450 loaded vehicles and nine traffic lights on the map. We select two various edges as the start and end edges on the map. To simulate heterogeneous traffic demands, the choice of vehicles

for their entry lanes is done randomly [110]. On the second try, we selected the E10 and E13 as the start and destination edges respectively. The start edge is shown in Figure 4.11 by a red rectangle. Also, the best and optimal path based on less waiting time, CO₂ emission and fuel consumption is highlighted by the red color path on the map.

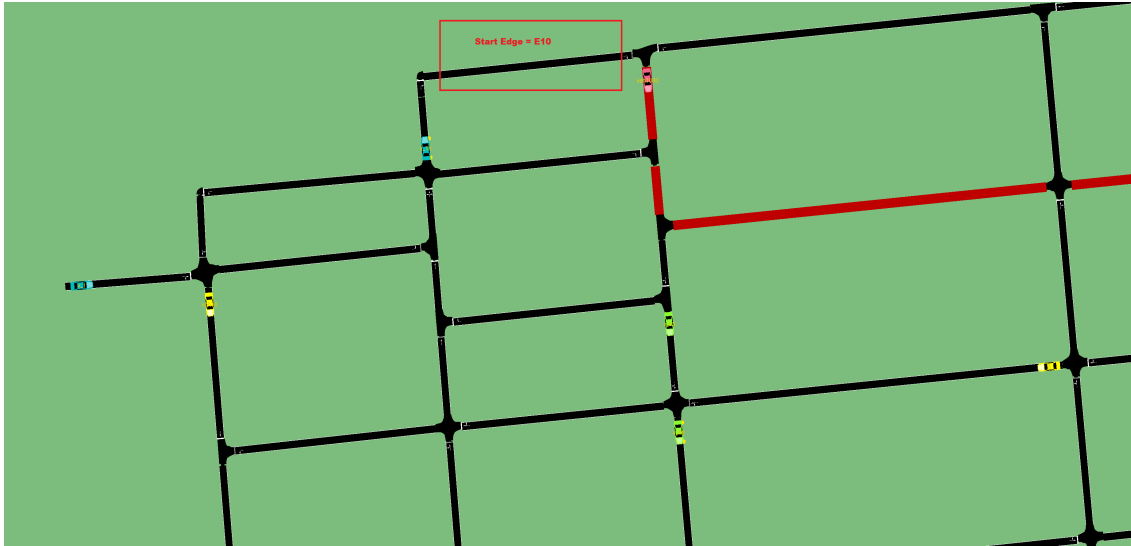


Figure 4.11: *The second try for discover optimal path with different start edge*

In Figure 4.12, as can be seen, the start and last edges and full path are marked on the map. The optimal route between E10 and E13 and the agent that is a vehicle with ID equal to veh 1000 are identified on the picture.

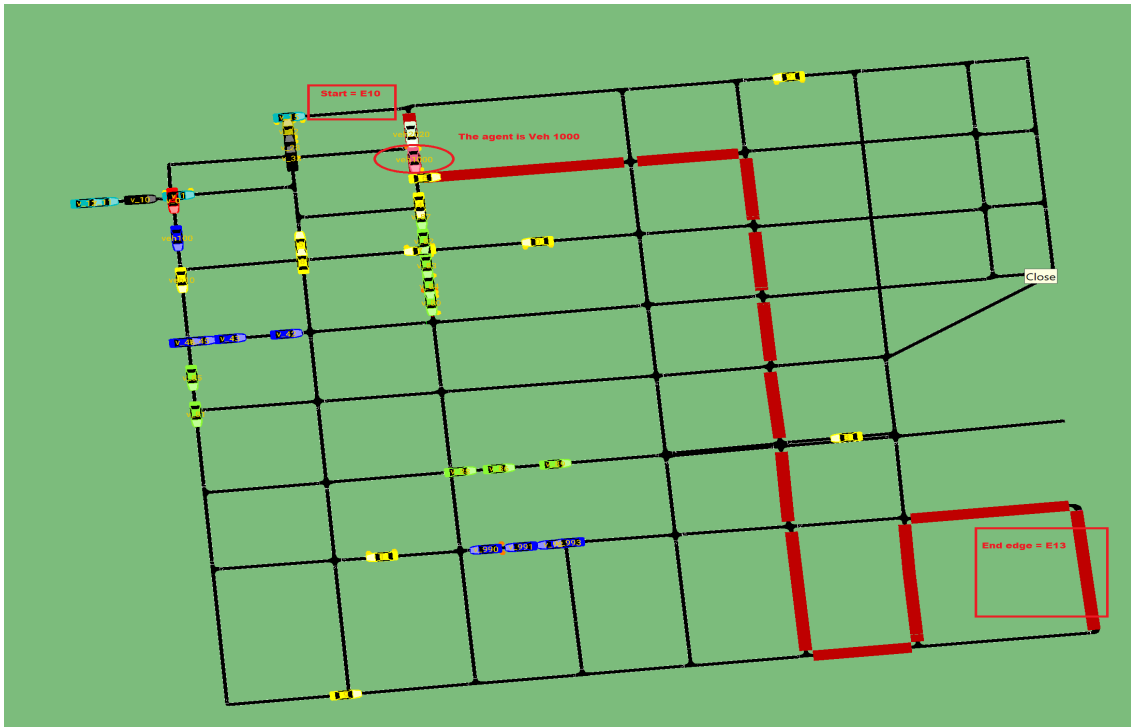


Figure 4.12: *The beginning edge , last edges , the agent and the optimal route*

The route for the agent is a path with less waiting time as a key factor on the map. Multiple routes may be available from the start to the end edge. However, it is crucial to identify which route

has the least waiting time on the map. The selected route is one of those with a lower waiting time to the destination edge, E13.

Another route might have the same waiting time to the destination edge, but its distance and fuel consumption are greater compared to the one chosen by the proposed RL model [124]. The path is illustrated in Figure 4.13.

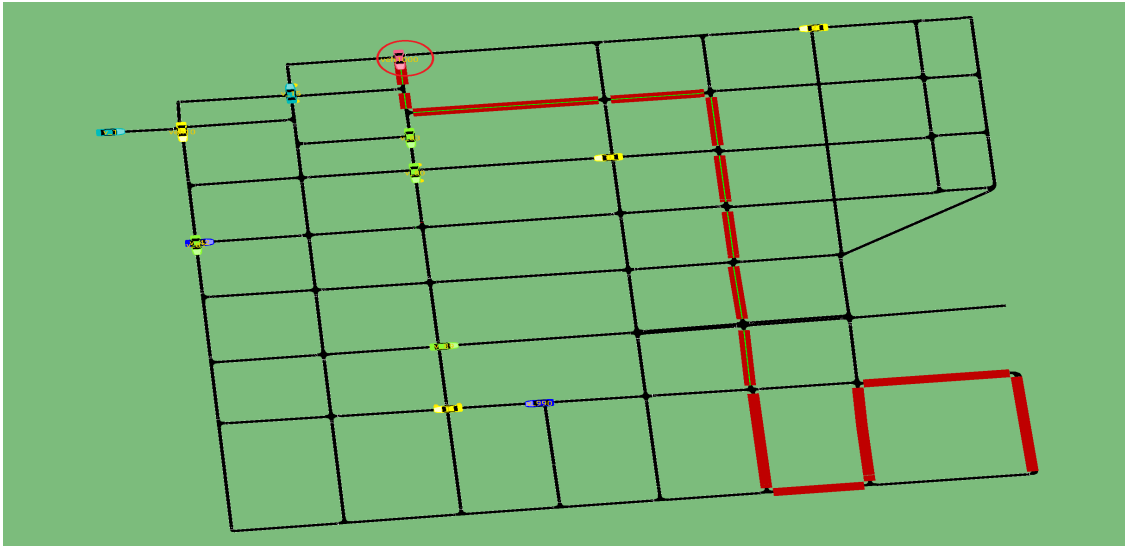


Figure 4.13: *The route with less waiting time*

The route followed by the agent (veh1000) is presented in the figures 4.14, 4.15, and 4.16. In Figure 4.15, 123 vehicles are currently loaded onto the map. During the simulation, this number increases to 450 vehicles that was a random traffic flow [126]. As shown, the path generated by RL experiences less traffic congestion. Following this route enables vehicles to reduce CO2 emissions and oil consumption. This will be demonstrated in Figure 4.17.

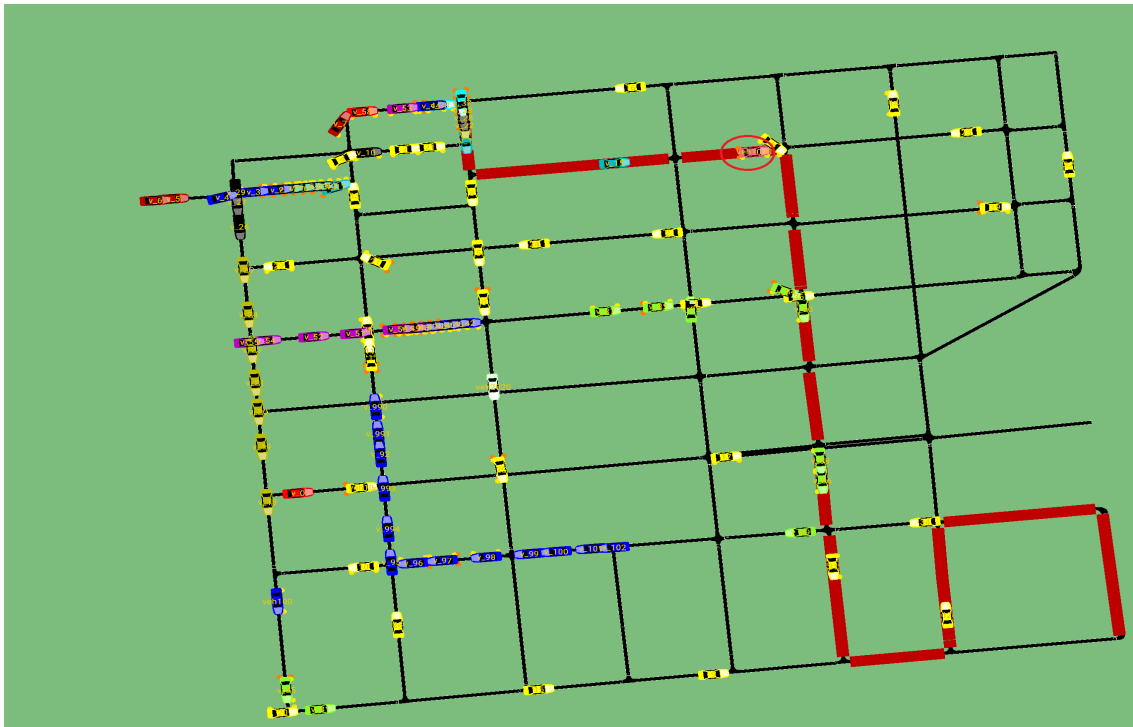


Figure 4.14: *The optimal path is tracked by the Agent (Veh1000)*

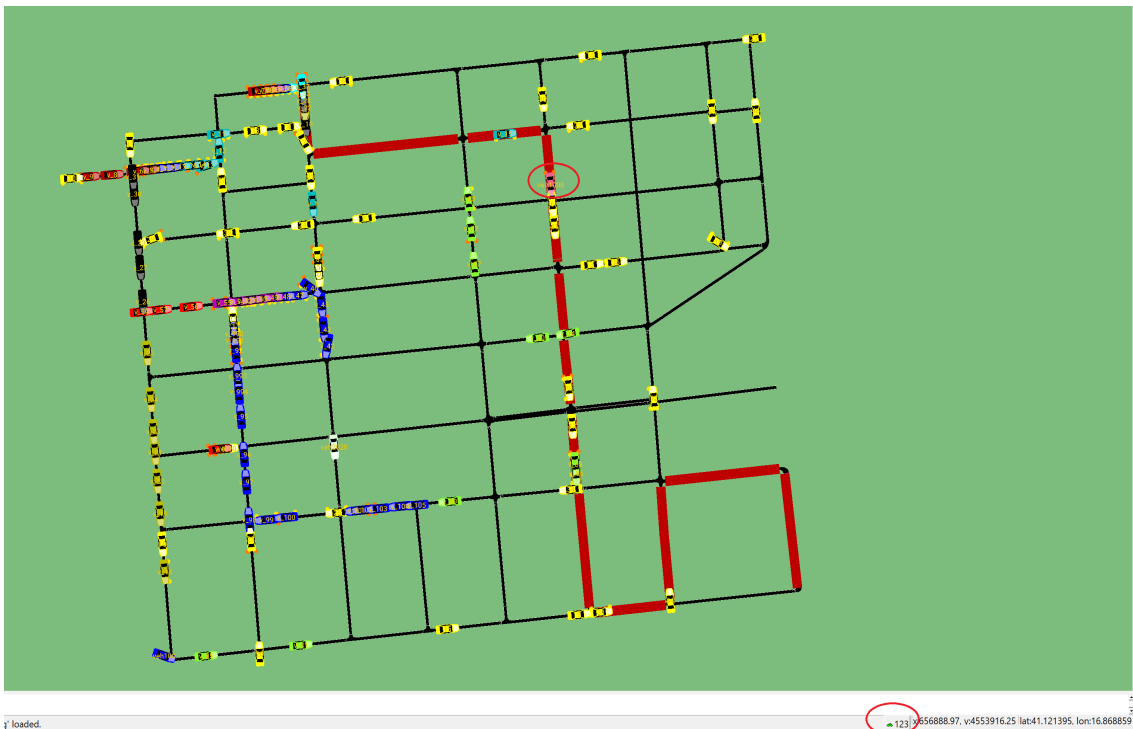


Figure 4.15: *The optimal path and number of cars are loaded as a traffic on the map*

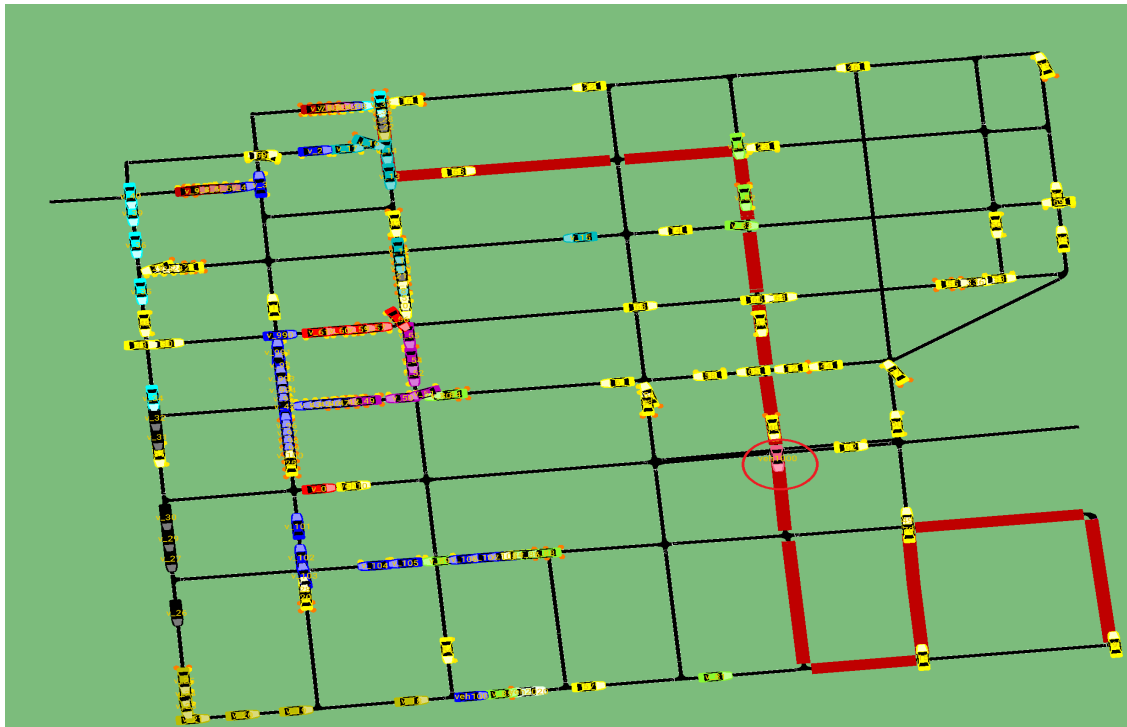


Figure 4.16: Tracking the optimal path by the Agent

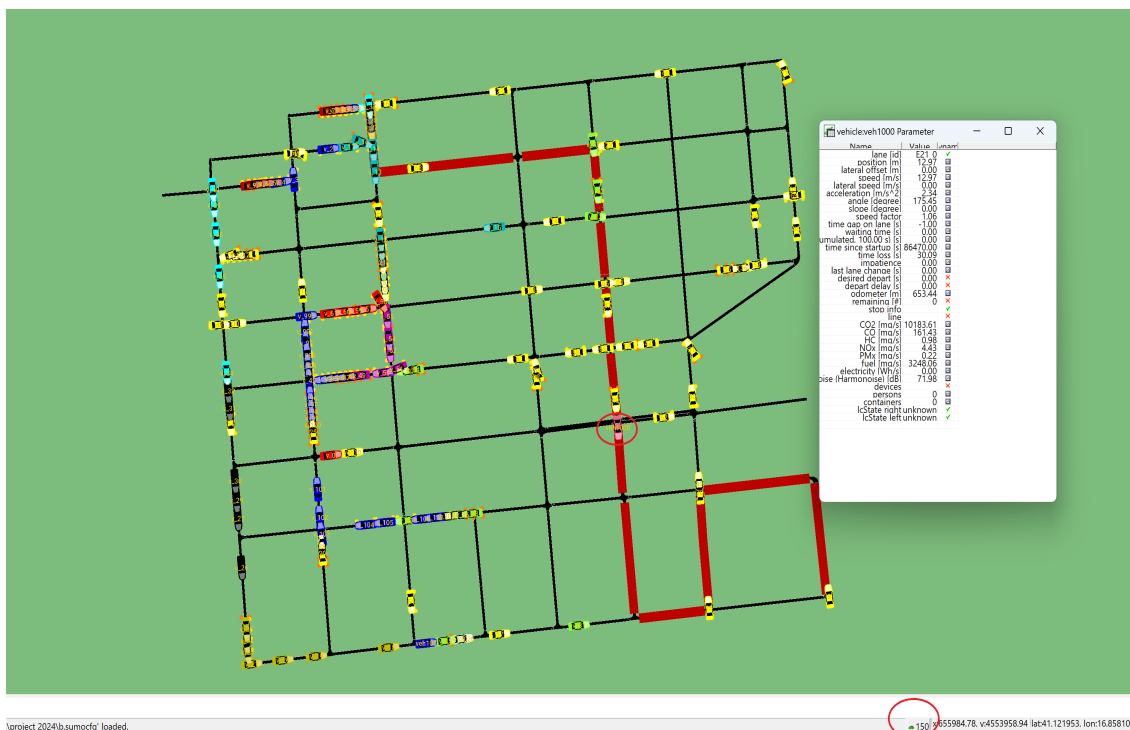


Figure 4.17: The agent follows the optimal route is extracted by RL

During the agent travel from the start to the end edge, the waiting time figure shows no delay. This is illustrated in Figure 4.18 and Figure 4.19, captured at two different time stamps. Additionally, the agent and the number of loaded vehicles are visible in those images, represented by red circles.



Figure 4.18: *The agent waiting time graph*



Figure 4.19: *The waiting time chart for the agent*

In the end, all vehicles were loaded in this simulation presented in Figure 4.20 where 450 cars were added to the simulation.

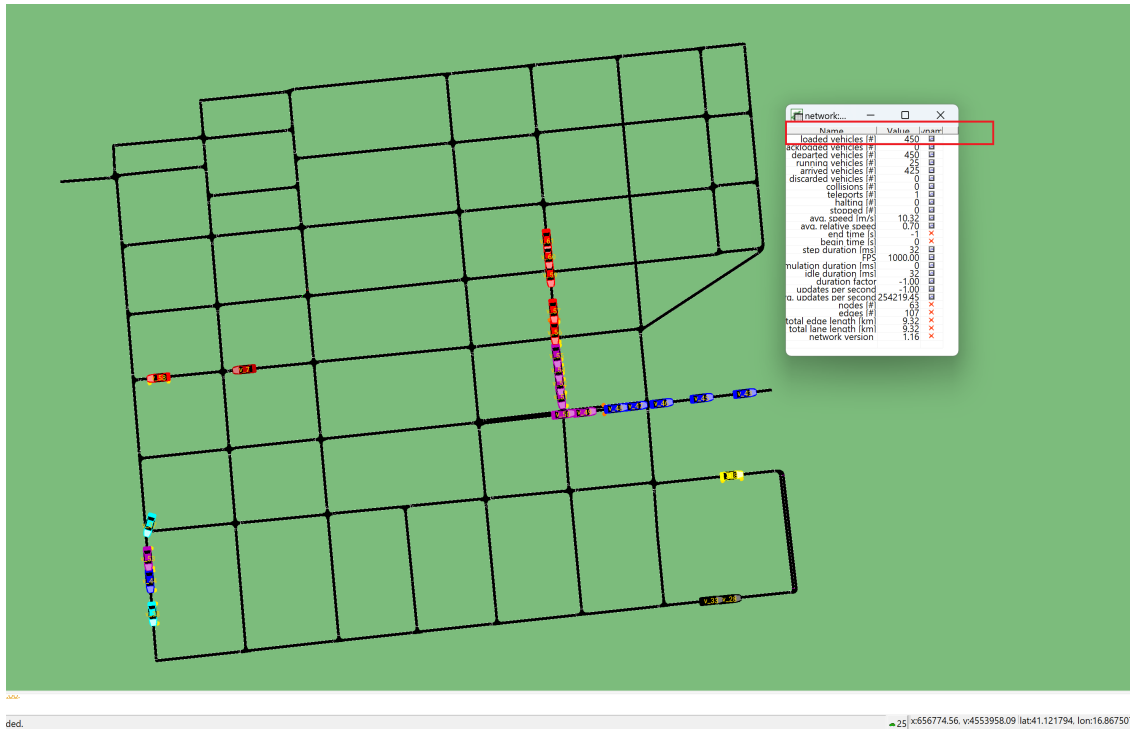


Figure 4.20: Total number of vehicles are loaded in the simulation

The goal of the study is to investigate the optimal path on the SUMO map. The outcome from the training process after the iteration is determined in Figure 4.21. As can be seen, the route length was reduced during the training process at the top plot in the figure. It means that the proposed RL model investigates a route with a minimum number. The figure presents the learning advancement of a reinforcement learning agent after 15,000 training episodes. In the upper plot, a distinct downward trend in route length (quantified by the number of edges) is obvious, signifying that the agent is progressively discovering shorter and more efficient paths as training continues. The logarithmic decay trend (illustrated in red) approaches an average length of approximately 20 edges, underscoring the agent enhanced capability to optimize routing decisions. Concurrently, the lower plot illustrates a consistent increase in episode reward (becoming less negative), despite some ongoing fluctuations. This pattern indicates that the agent is acquiring the ability to avoid insufficient paths and is being rewarded for improved performance. Overall, the reduction in route length and the upward trajectory of rewards indicate that the agent is effectively learning a policy that not only reduces travel distance but also enhances efficiency—culminating in robust and favorable routing behavior over time. The explanation for decreasing edge length and improving the rewards that they lead to achieve the optimal route is presented in Table 4.6 as well.

The observed decrease in route length and increase in cumulative reward indicate that the RL agent successfully learned a policy that balances path efficiency and traffic conditions. Routes with fewer edges and traffic lights were consistently favored, minimizing waiting time, air pollution or emissions, and fuel consumption. This demonstrates the model's ability to optimize multiple objectives concurrently. Some fluctuation in rewards is expected due to stochastic vehicle insertion and heterogeneous traffic demand, reflecting real-world variability.

Compared to conventional routing algorithms such as Dijkstra or A*, which in prior studies achieved path efficiencies around 0.65 under similar traffic conditions [Ref], the proposed RL model achieved a path efficiency of 0.79. The convergence of route length and reward aligns with

theoretical Q-learning expectations, confirming that the agent effectively learns optimal policies in a complex, stochastic traffic environment

The final option in the proposed RL model offers a practical approach to leveraging trained data for route planning [120]. This means that, at the end of the training process for an optimum path between two random desired edges. Different two random initial and destination edges can be selected and the proposed RL model tries to discover a viable path between the chosen edges on the map with the help of the trained data in prior stage. It is important to note that the identified path may not be the optimal route in this case; rather, it serves as a feasible route between any two selected edges.

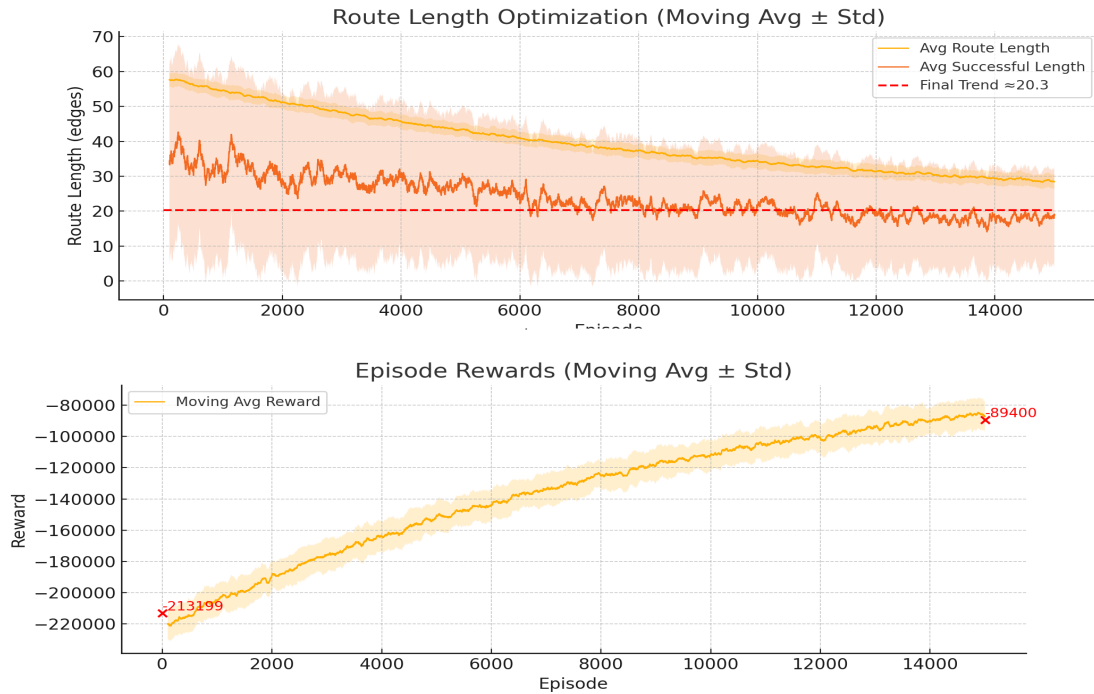


Figure 4.21: The training outcome route length per iteration and reward per iteration

As an example, we run a simulation to reach an optimal path between two random 488 edges E7 and E17 on the SUMO map. and the outcome is reported in Table 4.5. Overall, the table demonstrates that the agent makes substantial progress over 15 000 episodes. Also, on the one hand, the average route length drops from roughly 38 edges to just over 20 edges (a 46 % reduction), on the other hand, the mean episodic reward climbs from -173 k to -46 k (a 73 % improvement). At the same time, the success rate rises from about 30 % to approximately 52 %, showing that more than half of all trials now reach the goal. In the Table 4.6 explanation in details is presented. Moreover, minimum travel time factor as a major key factor has been applied in path planning between two random desired edges on a SUMO map as well.

Table 4.5: Key parameters analyzing in the proposed RL model in training process for two random desired edges as an example.

Metric	Vehicle Factor	Traffic light Factor	Edge-route-	Distance
Route length (edges)	37.96	20.33	−17.63	−46.5 %
Episodic reward	−173 056.03	−46 254.56	+126 801.47	+73.3 %
Success rate (approx. % of runs)	30 %	52 %	+22 pp	+73.3 %

Table 4.6: Performance Metrics over 15,000 Training Episodes

Metric	Initial Value	Final Value	Observation / Trend
Route Length (number of edges)	~40	20.33	Decreasing trend indicating route optimization over episodes
Reward	-174,056.03	-46,254.50	Significant improvement in cumulative reward (less negative)
Number of Episodes	15,000		Total training span over which performance was monitored
Successful Routes	Scattered, increasing	Dense at lower lengths	Higher density of successful completions with shorter paths
Reward Fluctuation	High	Still fluctuating	Variability remains but shows a positive (improving) trend
Route Length Trend Fit	Logarithmic decay		Converges toward optimal route length of ~20 edges
Reward Trend Fit	Logarithmic growth		Reflects consistent learning progress despite reward noise

For instance, the route investigation feature is illustrated in Figure 4.22. As shown, programmed option was available to explore a path between two randomly chosen edges on the SUMO map during the initial training process. The proposed RL model then utilized the trained data to determine the optimal path between the selected edges. One of the promising advantages of this approach is to ensure the path is loop-free. For instance, the route investigation feature is illustrated in the Figure 4.22. As shown, a programmed option was available to explore a path between two randomly chosen edges on the SUMO map during the initial training process. As can be seen, the last option of the proposed RL model is able to be a functional. It means that we can set another two random edges in order to find out the a path between them.

```

*****Training completed. Best route found:*****
Best Route: ['E4', 'E5', '24884076', '29980616#0', 'E18', 'E19', '29980620#1', '29980620#2', '29980620#3',
'29980637#3', '29980637#4', '24884196#5', '24884196#6', '24884196#7', '24884196#8', 'E10']
Final Metrics: Time=29.56, Lights=2, Dead ends=0, Length=16, Distance=1177.80

Route validated in SUMO. Total steps: 516

Training complete! Would you like to find routes between random points? (y/n): y

===== Loop-Free Route Finder =====

Choose an option:
1. Generate random start and end points
2. Specify start and end points
3. Exit
Enter choice (1-3): 2

Available edges in the network:
-E13, -E18, -E19, 24884043#0, 24884043#1, 24884043#6, 24884052#0, 24884052#1, 24884052#2, 24884052#7
24884076, 24884077#1, 24884077#2, 24884077#3, 24884103, 24884107, 24884189#1, 24884194#0, 24884196#0, 24884196#1
24884196#2, 24884196#3, 24884196#4, 24884196#5, 24884196#6, 24884196#7, 24884196#8, 24884205#0, 24884205#1, 24884205#2
24884205#3, 29975377#6, 29975377#7, 29975382#8, 29975382#9, 29975386#0, 29975386#1, 29975386#2, 29975387#0, 29975387#1
29975387#2, 29976859#1, 29976859#2, 29976859#3, 29976859#4, 29976859#5, 29976859#6, 29976859#7, 29976859#8, 29976859#9
29980616#0, 29980616#1, 29980620#0, 29980620#1, 29980620#2, 29980620#3, 29980620#4, 29980620#5, 29980621#10,
29980621#2
29980621#3, 29980621#4, 29980621#5, 29980631#1, 29980635, 29980636, 29980637#2, 29980637#3, 29980637#4, 30016109#2
30016109#3, 30016109#4, 30016109#5, 30016109#6, 30016109#7, 30029559#1, 30029559#2, 30029559#3, 30029559#4, 30029559#5
68576716#1, 68576723#1, 68576723#2, 68576723#3, E0, E1, E10, E11, E12, E13
E14, E15, E16, E17, E18, E19, E2, E20, E21, E22
E3, E4, E5, E6, E7, E8, E9,
Enter start edge ID: E7
Enter end edge ID: E13

Finding optimal loop-free route from E7 to E13...

----- Route Details -----
Route: ['E7', '30016109#2', '30016109#4', '30016109#5', '29975387#0', '29975387#1', '29975387#2', '-
E19', 'E21', '29980621#10', 'E5', '24884076', 'E12', 'E13']
Route length: 15 edges
Travel time: 94.35 seconds
Traffic lights: 2
Total distance: 1402.00 meters
Path efficiency: 1.00
Success: Yes

Do you want to visualize this route in SUMO? (y/n): n

```

Figure 4.22: Investigated valid route between two random desired edges with the help of using trained data

A SUMO traffic simulation environment was used to train the suggested q-learning-based reinforcement learning model, which is shown as an example in Table 4.7. It was used to determine the best, loop-free path between two edges (E7 and E13) that were specified by the user. In a previous training phase, the model used state-action pairs to explore an optimal path in order to minimize a cost function based on the minimum of vehicle density, edge length, traffic light presence, and travel time. The model effectively applied the learned Q-values to infer a new optimal path without retraining after obtaining the updated start and end edges after training. The chosen route had 14 edges, a total distance of 1402 meters, a travel time of 94.35 seconds, no traffic lights, and a path efficiency of 1.00, that the route is the shortest feasible and loop-free path. The result demonstrates the model's generalization capability, validating its effectiveness in dynamically finding optimal routes even between previously untested edge pairs, while avoiding loops and dead ends.

Component	Description
RL Task	Find the optimal loop-free route in SUMO using Reinforcement Learning Q-learning.
Training Phase	The model trained using edge information, vehicles, travel time, and other criteria to generate an optimal route.
Training Result	Best route found: from E4 to E10 using edges like E5, 24884076, etc., minimizing steps, traffic lights, and distance.
Interactive Post-Training Phase	User was prompted to test the model with new edges (random or specified).
User Input (Choice 2)	User manually specified a new route: from E7 to E13.
Edge Information Used	The same set of edge IDs from training was used for testing, confirming reusability of trained knowledge.
Route Calculation Outcome	The RL model successfully computed a new optimal route between E7 and E13.
Route Properties	Length: 14 edges, Travel time: 94.35 s, Distance: 1402.00 m, Path efficiency: 1.00, Traffic lights: 0, Success: Yes.
Evaluation	Demonstrates generalization of the trained model to unseen edge pairs using prior exploration knowledge.
Loop-Free Assurance	No edge is repeated, ensuring no loops in the path.
Dead-End Avoidance	The route ends at a valid terminal (E13), not at a non-connecting or one-way dead-end.
User Declined Visualization	The user chose not to visualize in SUMO, though the model offered the option.

Table 4.7: An example in the proposed RL-based SUMO route generation between two random desired user-defined edges using the trained data in Q-learning training model procedure.

These findings support the research objective of optimizing emergency vehicle routing in urban traffic networks. The results confirm that RL-based models can dynamically adapt to traffic conditions, achieving superior path efficiency compared to traditional methods. This extends previous work on SUMO-based routing by demonstrating the model's capability to generalize to new start and end edges without retraining.

Moreover, if the outcome fails due to the lack of a route between the selected edges, an error will be displayed in the results. This occurs when the proposed RL model struggles to identify a path between two randomly chosen edges. The error also identifies the proposed RL cannot find a path between two selected edges on the map and is presented in Figure 4.23.

```

Available edges in the network:
-E13, -E18, -E19, 24884043#0, 24884043#1, 24884043#6, 24884052#0, 24884052#1, 24884052#2, 24884052#7
24884076, 24884077#1, 24884077#2, 24884077#3, 24884103, 24884107, 24884189#1, 24884194#0, 24884196#0, 24884196#1
24884196#2, 24884196#3, 24884196#4, 24884196#5, 24884196#6, 24884196#7, 24884196#8, 24884205#0, 24884205#1, 24884205#2
24884205#3, 29975377#6, 29975377#7, 29975382#8, 29975382#9, 29975386#0, 29975386#1, 29975386#2, 29975387#0, 29975387#1
29975387#2, 29976859#1, 29976859#2, 29976859#3, 29976859#4, 29976859#5, 29976859#6, 29976859#7, 29976859#8, 29976859#9
29980616#0, 29980616#1, 29980620#0, 29980620#1, 29980620#2, 29980620#3, 29980620#4, 29980620#5, 29980621#10,
29980621#2
29980621#3, 29980621#4, 29980621#5, 29980631#1, 29980635, 29980636, 29980637#2, 29980637#3, 29980637#4, 30016109#2
30016109#3, 30016109#4, 30016109#5, 30016109#6, 30016109#7, 30029559#1, 30029559#2, 30029559#3, 30029559#4, 30029559#5
68576716#1, 68576723#1, 68576723#2, 68576723#3, E0, E1, E10, E11, E12, E13
E14, E15, E16, E17, E18, E19, E2, E20, E21, E22
E3, E4, E5, E6, E7, E8, E9,
Enter start edge ID: E13
Enter end edge ID: E4

Finding optimal loop-free route from E13 to E4...
Error finding route: No valid route found from E13 to E4 ✓
Try different start and end points or use option 1 to generate random points.

Choose an option:
1. Generate random start and end points
2. Specify start and end points
3. Exit
Enter choice (1-3): |

```

Figure 4.23: Failing to find the path between two random desired edges

In the Table 4.8 where it shows the proposed RL model could not find out a path between two desired edges. As there was not real path between the two edges, or the RL might need to have more iterations. The table is presented as an option of the proposed RL model.

Table 4.8: Detailed Failure Case in Route Finding

Parameter	Value / Description
Available Edges	-E13, -E19, 24884043#0, 24884043#1, 24884052#0, ..., E13
Start Edge ID	E13
End Edge ID	E4
Route Search Type	Optimal loop-free route
Result	Failed: No valid route found from E13 to E4
System Suggestion	Try different start and end points or use option 1 to generate random points
User Options	1: Generate random start and end points 2: Specify start and end points 3: Exit
User Prompt	Enter choice (1-3):

The significance of this study lies in its potential application to real-time emergency routing, where reducing waiting time and minimizing fuel consumption are critical. However, the model's

performance depends on traffic density, number of iterations, and map configuration. While the agent demonstrates robust path optimization, further work is needed to evaluate scalability to larger urban networks and dynamic, real-time traffic conditions. These results highlight both the practical applicability and current limitations of Q-learning in urban traffic optimization.

4.6 Future Work

This study presents an approach based on Reinforcement Learning (RL)—specifically Q-Learning—to investigate the optimal path between two randomly selected edges on a city map (a section of the Bari City map in Italy). The study began with a standard map extracted from Open Street Map (OSM) for Bari. On this map, a single traffic light was included in the environment. Traffic conditions were modeled with 123 vehicles on the map, which served as a crucial factor in the study. As the first step, we analyzed the map to determine the optimal path between two randomly chosen edges. In subsequent phases, the number of vehicles and traffic lights was increased nine and four times, respectively. This enhancement enabled the simulation to approximate real-time conditions. In other words, the proposed RL model is designed to adapt to future simulations with increased numbers of vehicles and traffic lights, ensuring it can still identify the optimal path between two randomly desired edges on the map.

The autonomous vehicle (AV) topic is one of the most trending and popular subjects in recent years. The AV must select a safe and reliable trajectory on the road; otherwise, choosing the wrong trajectory could lead to collisions and traffic congestion [120], [116]. Modern technologies such as Reinforcement Learning (RL), Deep Learning (DL), Machine Learning (ML), and object detection have also benefited from advancements in AV. The challenge is that these technologies must be utilized in real-time to ensure more accurate measurements on the street. AVs address significant issues, particularly traffic and accidents on roads. Consequently, simulations in studies must be conducted based on factors that can have a substantial impact on roads in real-time.

Countries aim to ensure greater safety on their roads in the future. However, a large number of people are killed in accidents every year. The death rate continues to rise annually across the globe. As a result, governments are encouraged to leverage modern technologies to better manage and control vehicles and drivers with the assistance of smart systems. Passengers will benefit from smart traffic lights, vehicles, buildings, and roads to reduce accidents and enhance safety, promoting safe driving in the future. These elements are the backbones of smart cities. Additionally, vehicle companies must improve the quality and technology of their vehicles. The integration of smart traffic lights, vehicles, and roads, enhanced by reinforcement learning (RL), helps reduce travel time, accidents and emissions effectively. [127].

The lack of fossil fuels and the rising CO₂ emissions are encouraging countries to adopt electricity as a fuel for vehicles. As a result, we will witness the development of Electric Vehicles (EVs) and Autonomous Electric Vehicles (AEVs), which will also benefit from new technologies [116]. Electricity, with zero emissions, will become the environmentally friendly fuel of the near future.

This research shows how reinforcement learning can find the best paths for self-driving cars pointing to its ability to shake up city transport systems. RL algorithms allow these cars to adapt

on the fly to traffic, which cuts down on travel time and energy use while making traffic move better overall. This step forward looks good for building smarter greener cities where better transport leads to less traffic fewer emissions, and a boost in life quality. As self-driving car tech gets better, using reinforcement learning will be key to building smart city systems of the future. As part of future work, additional key elements of roads will be incorporated into simulations to evaluate and analyze RL models. Factors such as slope and weather conditions, including wind, rain, and snow or mud, will also be considered in the upcoming studies in the RL [31]. Moreover, electric vehicles will be able to communicate to smart traffic lights and near electric vehicles on road so as to evaluate traffic congestion and road status [117].

In summary, this research demonstrated that a Q-learning-based reinforcement learning approach can efficiently identify optimal, loop-free paths for autonomous vehicles on urban maps, even under varying traffic densities and traffic light configurations. The study showed improvements in route efficiency, travel time, and vehicle energy consumption, validating the model's ability to generalize to previously untested start and end edges. The key contributions of this thesis include: (i) the development of a robust RL framework for urban path planning, (ii) validation of its performance under realistic traffic conditions, and (iii) demonstration of the model's generalization capabilities without retraining. For future work, simulations could incorporate additional road factors such as slope, weather conditions, and real-time dynamic traffic data. Moreover, multi-agent RL approaches, vehicle-to-vehicle and vehicle-to-infrastructure communication, and scalability to larger urban networks could be explored to further enhance autonomous vehicle routing and smart city traffic management.

Chapter 5

Conclusion

[Back to Contents](#)

5.1 Overall Conclusions

This research first addressed the problem of object detection in challenging environments, focusing on poor weather conditions such as fog, where visibility is significantly reduced. We employed the YOLOv8 model and trained it on a dataset of 4,000 foggy road images while adjusting hyperparameters like epochs, batch size, and augmentation methods. The outcomes illustrated that the proposed approach improved object recognition rate, especially for cars, cross walks, traffic lights, and street signs. However, the study also faced some limitations, consisting of difficulties with detecting small objects and the risk of overfitting when augmentation or higher epochs were employed. Despite these issues, YOLOv8 proved effective and promising method for future usage in autonomous Vehicles.

To enhance the proposed method, we evaluated the proposed detection methods using four various methodologies, examining their robustness across various weather scenarios. The results emphasized the need for a diverse set of data, as the initial shortage of clear and sunny images limited the model's evaluation. To rectify the problem, three additional datasets featuring improved visibility conditions were introduced, enabling a more thorough assessment. These experiments indicated that merging datasets from varying weather conditions can improve detection accuracy, which is crucial for the real-world application in autonomous vehicles where environmental conditions are constantly changing.

Although object detection is fundamental, autonomous vehicles also need efficient path planning to ensure safe and timely journeys. The second study has been applied a convolutional neural network (CNN) supported by the SB3 library on a SUMO-based road simulation. In this model, traffic light information, vehicle density, and speed limits were included as inputs, enabling the CNN to determine optimal routes on the map [77, 81, 104]. The integration of these factors verified critical, as the balance between vehicle density and speed limitations strongly influenced the choice of routes. The results confirmed that CNN-based models can effectively optimize paths

by considering multiple urban mobility factors at the same time.

This research was done in just CNNs, highlighting the advantages of utilizing reinforcement learning (RL) into frameworks for path planning optimization. When reinforcement learning systems are connected to a CNN-based monitoring, they can resolve a complicated problem. In addition, it can handle a dynamic challenges such as real-time urban situations, especially in the management of traffic signals and congestion points [101]. This synergy between approaches would not only improve the efficiency of road networks but also establish the groundwork for more responsive traffic management systems in smart cities. Such integration demonstrates how AI models can work in harmony, uniting object detection, predictive modeling, and adaptive learning to provide feasible solutions for urban challenges.

The third study took the research further by applying reinforcement learning, particularly Q-learning, to improve routes in realistic situations. Using a section of the Bari city map, the model changed to handle more vehicles and traffic lights, showing it could scale and remain reliable. Reinforcement learning algorithms showed great potential to help self-driving cars adjust to traffic in real time, which is vital to reduce accidents, congestion, and delays [120, 116]. The simulation results also showed that reinforcement learning can work with object detection by not just understanding the environment, but also making choices that improve safety and efficiency in uncertain city conditions.

Finally, these three studies collectively emphasize that autonomous vehicles and smart cities depend on the integration of multiple AI-driven technologies in the future. Object detection, CNN-based path planning, and reinforcement learning each address various but linked challenges. Together, they contribute to safer, more sustainable, and more efficient mobility systems. In the future, further research should incorporate additional environmental factors such as slopes, rain, snow, and wind [31], as well as communication between electric vehicles and smart traffic systems [117, 127]. In doing so, autonomous vehicles can become a central component of smart cities, reducing emissions, improving safety, and improving quality of life through cleaner and more intelligent transport networks.

References

[Back to Contents](#)

Bibliography

- [1] Hoangcong Le, Saeed Saeedvand, and Chen-Chien Hsu. A comprehensive review of mobile robot navigation using deep reinforcement learning algorithms in crowded environments. *Journal of Intelligent and Robotic Systems*, 110(4):158, 2024. ISSN 1573-0409. doi: 10.1007/s10846-024-02198-w. URL <https://doi.org/10.1007/s10846-024-02198-w>.
- [2] Sebastian Huch and Markus Lienkamp. Towards minimizing the lidar sim-to-real domain shift: Object-level local domain adaptation for 3d point clouds of autonomous vehicles. *Sensors*, 23(24), 2023. ISSN 1424-8220. doi: 10.3390/s23249913. URL <https://www.mdpi.com/1424-8220/23/24/9913>.
- [3] Haibin Liu, Chao Wu, and Huanjie Wang. Real time object detection using lidar and camera fusion for autonomous driving. *Scientific Reports*, 13(1):8056, 2023. ISSN 2045-2322. doi: 10.1038/s41598-023-35170-z. URL <https://doi.org/10.1038/s41598-023-35170-z>.
- [4] Majdi Sukkar, Madhu Shukla, Dinesh Kumar, Vassilis C. Gerogiannis, Andreas Kanavos, and Biswaranjan Acharya. Enhancing pedestrian tracking in autonomous vehicles by using advanced deep learning techniques. *Information*, 15(2), 2024. ISSN 2078-2489. doi: 10.3390/info15020104. URL <https://www.mdpi.com/2078-2489/15/2/104>.
- [5] Njayou Youssouf. Traffic sign classification using cnn and detection using faster-rcnn and yolov4. *Heliyon*, 8(12):e11792, 2022. ISSN 2405-8440. doi: <https://doi.org/10.1016/j.heliyon.2022.e11792>. URL <https://www.sciencedirect.com/science/article/pii/S2405844022030808>.
- [6] Raed Alharthi, Iram Noreen, Amna Khan, Turki Aljrees, Zoraiz Riaz, and Nisreen Innab. Novel deep reinforcement learning based collision avoidance approach for path planning of robots in unknown environment. *PLOS ONE*, 20(1):1–22, 01 2025. doi: 10.1371/journal.pone.0312559. URL <https://doi.org/10.1371/journal.pone.0312559>.
- [7] Ramón Jaramillo-Martínez, Ernesto Chavero-Navarrete, and Teodoro Ibarra-Pérez. Reinforcement-learning-based path planning: A reward function strategy. *Applied Sciences*, 14(17), 2024. ISSN 2076-3417. doi: 10.3390/app14177654. URL <https://www.mdpi.com/2076-3417/14/17/7654>.
- [8] Lun Ge, Xiaoguang Zhou, Yongqiang Li, and Yongcong Wang. Deep reinforcement learning navigation via decision transformer in autonomous driving. *Frontiers in Neurorobotics*, Volume 18 - 2024, 2024. ISSN 1662-5218. doi: 10.3389/fnbot.2024.1338189. URL <https://www.frontiersin.org/journals/neurorobotics/articles/10.3389/fnbot.2024.1338189>.

- [9] Qiming Gao, Fangle Chang, Jiahong Yang, Yu Tao, Longhua Ma, and Hongye Su. Deep reinforcement learning for autonomous driving with an auxiliary actor discriminator. *Sensors*, 24(2), 2024. ISSN 1424-8220. doi: 10.3390/s24020700. URL <https://www.mdpi.com/1424-8220/24/2/700>.
- [10] Zhijie Zhang, Hao Fu, Juan Yang, and Yunhan Lin. Deep reinforcement learning for path planning of autonomous mobile robots in complicated environments. *Complex & Intelligent Systems*, 11(6):277, 2025. ISSN 2198-6053. doi: 10.1007/s40747-025-01906-9. URL <https://doi.org/10.1007/s40747-025-01906-9>.
- [11] Zhaohui Liu, Shiji Zhao, and Xiao Wang. Research on driving obstacle detection technology in foggy weather based on gcanet and feature fusion training. *Sensors*, 23(5), 2023. ISSN 1424-8220. doi: 10.3390/s23052822. URL <https://www.mdpi.com/1424-8220/23/5/2822>.
- [12] Sergei Voronin, Abubakar Siddique, and Muhammad Iqbal. Enhancing object detection accuracy in autonomous vehicles using synthetic data. *arXiv*, 2024. URL <https://arxiv.org/abs/2411.15602>.
- [13] Sangeeta Mallikarjun Kharge and Shivkumar S Jawaligi. Customized convolutional network-based road traffic congestion detection in vanet. *Intelligent Decision Technologies*, 19(4):2479–2501, 2025. doi: 10.1177/18724981251361017. URL <https://doi.org/10.1177/18724981251361017>.
- [14] Boquan Yang, Jixiong Li, and Ting Zeng. A review of environmental perception technology based on multi-sensor information fusion in autonomous driving. *World Electric Vehicle Journal*, 16(1), 2025. ISSN 2032-6653. doi: 10.3390/wevj16010020. URL <https://www.mdpi.com/2032-6653/16/1/20>.
- [15] Tiagrajah V Janahiraman and Mohamed Shahrul Mohamed Subuhan. Traffic light detection using tensorflow object detection framework. *2019 IEEE 9th International Conference on System Engineering and Technology (ICSET)*, pages 108–113, 2019.
- [16] Siddharth Bhalerao Nida Khan. A review paper on different type of object detection techniques for vehicle. *International Research Journal of Modernization in Engineering Technology and Science*, 4(10):1068–1072, 2022.
- [17] Akhil Reddy Kalva, Jyothi Swarup Chelluboina, and B Bharathi. Smart traffic monitoring system using yolo and deep learning techniques. *2023 7th International Conference on Trends in Electronics and Informatics (ICOEI)*, pages 831–837, 2023.
- [18] Qiuxiang Shi, Fei Zhong, Bo Li, Ziwei Xu, Junjie Hong, and Ruoxi Li. Fast vehicle detection algorithm based on lightweight yolo7-tiny. *Fifth International Conference on Computer Vision and Data Mining (ICCVDM 2024)*, 13272:50–58, 2024.
- [19] Jiawei Wang, Cen Chen, and Chongwen Wang. Street sign recognition algorithm based on deep learning. *Proceedings of the 2020 3rd International Conference on Image and Graphics Processing*, pages 31–35, 2020.

- [20] WeiGuo Pan, YingHao Chen, and Bo Liu. Traffic light detection for self-driving vehicles based on deep learning. *2019 15th International Conference on Computational Intelligence and Security (CIS)*, pages 63–67, 2019.
- [21] Ahmad Esmaeil Abbasi, Agostino Marcello Mangini, and Maria Pia Fanti. Object and pedestrian detection on road in foggy weather conditions by hyperparameterized yolov8 model. *Electronics*, 13(18), 2024. ISSN 2079-9292. doi: 10.3390/electronics13183661. URL <https://www.mdpi.com/2079-9292/13/18/3661>.
- [22] Bakhytzhan Kulambayev, Magzat Nurlybek, Gulnar Astaubayeva, Gulnara Tleuberdiyeva, Serik Zholdasbayev, and Abdimukhan Tolep. Real-time road surface damage detection framework based on mask r-cnn model. *International Journal of Advanced Computer Science and Applications*, 14(9), 2023. doi: 10.14569/IJACSA.2023.0140979. URL <http://dx.doi.org/10.14569/IJACSA.2023.0140979>.
- [23] Meng Xi, Huiao Dai, Jingyi He, Wenjie Li, Jiabao Wen, Shuai Xiao, and Jiachen Yang. A lightweight reinforcement-learning-based real-time path-planning method for unmanned aerial vehicles. *IEEE Internet of Things Journal*, 11(12):21061–21071, 2024. doi: 10.1109/JIOT.2024.3350525.
- [24] R SenthilPrabha, Baskaran J, Darshan Sasikumar, Gowtham Sriram, Kevin Nelson, and Pranavh Harish. Automation of traffic control systems with deep q-learning network. *2022 International Conference on Power, Energy, Control and Transmission Systems (ICPECTS)*, pages 1–6, 2022. doi: 10.1109/ICPECTS56089.2022.10047191.
- [25] Peter Palos and arpad Huszak. Comparison of q-learning based traffic light control methods and objective functions. *2020 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, pages 1–6, 2020. doi: 10.23919/SoftCOM50211.2020.9238290.
- [26] Seyedeh M. Mortazavi Azad and A. Ramazani. Smart control of traffic lights based on traffic density in the multi-intersection network by using q learning. *Discover Artificial Intelligence*, 3(1):39, 2023. ISSN 2731-0809. doi: 10.1007/s44163-023-00087-z. URL <https://doi.org/10.1007/s44163-023-00087-z>.
- [27] Yaofeng Song, Han Zhao, Ruikang Luo, Liping Huang, Yicheng Zhang, and Rong Su. A sumo framework for deep reinforcement learning experiments solving electric vehicle charging dispatching problem. *Electrical Engineering and Systems Science*, eess>arXiv:2209.02921, 2022. URL <https://arxiv.org/abs/2209.02921>.
- [28] Nan Zhong, Kaifeng Liu, and Yurong Li. Deep q-learning network model for optimizing transit bus priority at multiphase traffic signal controlled intersection. *Mathematical Problems in Engineering*, 2023(1):9137889, 2023. doi: <https://doi.org/10.1155/2023/9137889>. URL <https://onlinelibrary.wiley.com/doi/abs/10.1155/2023/9137889>.
- [29] J.J. Meerhof. Loop-breaking approaches for vehicle route planning with multi-agent q-routing. *35th Twente Student Conference on IT, Enschede, The Netherlands*, July 2021. URL <http://essay.utwente.nl/86888/>.

- [30] Filip Vrbanić, Edouard Ivanjko, Sadko Mandžuka, and Mladen Miletić. Reinforcement learning based variable speed limit control for mixed traffic flows. *2021 29th Mediterranean Conference on Control and Automation (MED)*, pages 560–565, 2021. doi: 10.1109/MED51440.2021.9480215.
- [31] Xuefeng Han, Hongwen He, Jingda Wu, Jiankun Peng, and Yuecheng Li. Energy management based on reinforcement learning with double deep q-learning for a hybrid electric tracked vehicle. *Applied Energy*, 254:113708, 2019. ISSN 0306-2619. doi: <https://doi.org/10.1016/j.apenergy.2019.113708>. URL <https://www.sciencedirect.com/science/article/pii/S0306261919313959>.
- [32] Pengfei Du, Xiang He, Haotong Cao, Sahil Garg, Georges Kaddoum, and Mohammad Mehedi Hassan. Ai-based energy-efficient path planning of multiple logistics uavs in intelligent transportation systems. *Comput. Commun.*, 207(C):46–55, July 2023. ISSN 0140-3664. doi: 10.1016/j.comcom.2023.04.032. URL <https://doi.org/10.1016/j.comcom.2023.04.032>.
- [33] Dimitris Georgiadis, Konstantina Karathanasopoulou, Cleopatra Bardaki, Ilias Panagiotopoulos, Ioannis Vondikakis, Thalys Paktitis, and George Dimitrakopoulos. Performance analysis of energy-efficient path planning for sustainable transportation. *Sustainability*, 16(12), 2024. ISSN 2071-1050. doi: 10.3390/su16124963. URL <https://www.mdpi.com/2071-1050/16/12/4963>.
- [34] Georgios Papaioannou, Vallan Maroof, Jenny Jerrelind, and Lars Drugge. Reducing tyre wear emissions of automated articulated vehicles through trajectory planning. *Sensors*, 24(10), 2024. ISSN 1424-8220. doi: 10.3390/s24103179. URL <https://www.mdpi.com/1424-8220/24/10/3179>.
- [35] Xiaoting Dong, Guangxi Wan, Peng Zeng, Chunhe Song, and Shijie Cui. Optimizing robotic task sequencing and trajectory planning on the basis of deep reinforcement learning. *Biomimetics*, 9(1), 2024. ISSN 2313-7673. doi: 10.3390/biomimetics9010010. URL <https://www.mdpi.com/2313-7673/9/1/10>.
- [36] Edgar Jungblut, Thomas Grube, Jochen Linssen, and Detlef Stolten. Fuel-saving opportunities for automated vehicles: A driving cycle analysis. *Transportation Research Interdisciplinary Perspectives*, 22:100964, 2023. ISSN 2590-1982. doi: <https://doi.org/10.1016/j.trip.2023.100964>. URL <https://www.sciencedirect.com/science/article/pii/S2590198223002117>.
- [37] Xiaowei Chen, Jiawei Xue, Zengxiang Lei, Xinwu Qian, and Satish V. Ukkusuri. Online eco-routing for electric vehicles using combinatorial multi-armed bandit with estimated covariance. *Transportation Research Part D: Transport and Environment*, 111:103447, 2022. ISSN 1361-9209. doi: <https://doi.org/10.1016/j.trd.2022.103447>. URL <https://www.sciencedirect.com/science/article/pii/S1361920922002735>.
- [38] Taha Taner İnal, Galip Cansever, Barış Yalçın, Gürkan Çetin, and Ahu Ece Hartavi. Enhanced energy efficiency through path planning for off-road missions of unmanned tracked electric vehicle. *Vehicles*, 6(3):1027–1050, 2024. ISSN 2624-8921. doi: 10.3390/vehicles6030049. URL <https://www.mdpi.com/2624-8921/6/3/49>.

- [39] Bilgenur Erdoğan, Mustafa Kemal Tural, and Arsham Atashi Khoei. Finding an energy efficient path for plug-in electric vehicles with speed optimization and travel time restrictions. *Computers and Industrial Engineering*, 176:108987, 2023. ISSN 0360-8352. doi: <https://doi.org/10.1016/j.cie.2023.108987>. URL <https://www.sciencedirect.com/science/article/pii/S0360835223000116>.
- [40] Ethery Ramirez Robles, Oleg Starostenko, and Vicente Alarcon-Aquino. Real-time path planning for autonomous vehicle off-road driving. *PeerJ Computer Science*, 10:e2209, 07 2024. doi: 10.7717/peerj-cs.2209.
- [41] Fazhan Tao, Zhaowei Ding, Zhigao Fu, Mengyang Li, and Baofeng Ji. Efficient path planning for autonomous vehicles based on rrt* with variable probability strategy and artificial potential field approach. *Scientific Reports*, 14:24698, 10 2024. doi: 10.1038/s41598-024-76299-9.
- [42] Jianlin Mao, Zhigang He, Dayan Li, Ruiqi Li, Shufan Zhang, and Niya Wang. Multi-agent collaborative path planning algorithm with multiple meeting points. *Electronics*, 13(16), 2024. ISSN 2079-9292. doi: 10.3390/electronics13163347. URL <https://www.mdpi.com/2079-9292/13/16/3347>.
- [43] Honglin Zhang, Yaohua Wu, Jinchang Hu, and Yanyan Wang. Collaborative optimization of task scheduling and multi-agent path planning in automated warehouses. *Complex & Intelligent Systems*, 9(5):5937–5948, 2023. ISSN 2198-6053. doi: 10.1007/s40747-023-01023-5. URL <https://doi.org/10.1007/s40747-023-01023-5>.
- [44] Zhengxian Chen, Yuqi Liu, Wenjie Ni, Han Hai, Chaosheng Huang, Boyang Xu, Zihan Ling, Yang Shen, Wenhao Yu, Huanan Wang, and Jun Li. Predicting driving comfort in autonomous vehicles using road information and multi-head attention models. *Nature Communications*, 16(1):2709, 2025. ISSN 2041-1723. doi: 10.1038/s41467-025-57845-z. URL <https://doi.org/10.1038/s41467-025-57845-z>.
- [45] Seyed Mohsen Hosseini, Hamid Mirzahosseini, and Robert Guzik. Sustainable integration of autonomous vehicles into road networks: Ecological and passenger comfort considerations. *Sustainability*, 16(14), 2024. ISSN 2071-1050. doi: 10.3390/su16146239. URL <https://www.mdpi.com/2071-1050/16/14/6239>.
- [46] Yiting Li, Qingsong Fan, Haisong Huang, Zhenggong Han, and Qiang Gu. A modified yolov8 detection network for uav aerial image recognition. *Drones*, 7(5), 2023. ISSN 2504-446X. doi: 10.3390/drones7050304. URL <https://www.mdpi.com/2504-446X/7/5/304>.
- [47] Raihan Bin Islam, Samiha Akhter, Faria Iqbal, Md. Saif Ur Rahman, and Riasat Khan. Deep learning based object detection and surrounding environment description for visually impaired people. *Heliyon*, 9(6):e16924, 2023. ISSN 2405-8440. doi: <https://doi.org/10.1016/j.heliyon.2023.e16924>. URL <https://www.sciencedirect.com/science/article/pii/S2405844023041312>.
- [48] Teena Sharma, Benoit Debaque, Nicolas Duclos, Abdellah Chehri, Bruno Kinder, and Paul Fortier. Deep learning-based object detection and scene perception under bad weather conditions. *Electronics*, 11(4), 2022. ISSN 2079-9292. doi: 10.3390/electronics11040563. URL <https://www.mdpi.com/2079-9292/11/4/563>.

- [49] Xiu-Zhi Chen, Chieh-Min Chang, Chao-Wei Yu, and Yen-Lin Chen. A real-time vehicle detection system under various bad weather conditions based on a deep learning model without retraining. *Sensors*, 20(20), 2020. ISSN 1424-8220. doi: 10.3390/s20205731. URL <https://www.mdpi.com/1424-8220/20/20/5731>.
- [50] ST Santhanalakshmi and Rashmita Khilar. A custom deep convolutional neural network cdnn—(with yolo v3 based newly constructed backbone) for multiple object detection. *J. Data Acquis. Process*, 38:1511–1526, 2023.
- [51] Dijana Vitas, Martina Tomic, and Matko Burul. Traffic light detection in autonomous driving systems. *IEEE Consumer Electronics Magazine*, 9(4):90–96, 2020. doi: 10.1109/MCE.2020.2969156.
- [52] A Marode, A Ambadkar, A Kale, and T Mangrudkar. Car detection using yolo algorithm. *International Research Journal of Modernization in Engineering Technology and Science*, 3(05):939–942, 2021.
- [53] Sundas Iftikhar, Muhammad Asim, Zuping Zhang, Ammar Muthanna, Junhong Chen, Mohammed El-Affendi, Ahmed Sedik, and Ahmed A Abd El-Latif. Target detection and recognition for traffic congestion in smart cities using deep learning-enabled uavs: A review and analysis. *Applied sciences*, 13(6):3995, 2023.
- [54] Moises Diaz, Pietro Cerri, Giuseppe Pirlo, Miguel A Ferrer, and Donato Impedovo. A survey on traffic light detection. *International conference on image analysis and processing*, pages 201–208, 2015.
- [55] Zhigang Liu, Dongyu Li, Shuzhi Sam Ge, and Feng Tian. Small traffic sign detection from large image. *Applied Intelligence*, 50(1):1–13, 2020.
- [56] Victor JD Tsai, Jyun-Han Chen, and Hsun-Sheng Huang. Traffic sign inventory from google street view images. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 41:243–246, 2016.
- [57] R Gokul, A Nirmal, KM Bharath, MP Pranesh, and R Karthika. A comparative study between state-of-the-art object detectors for traffic light detection. *2020 international conference on emerging trends in information technology and engineering (ic-ETITE)*, pages 1–6, 2020.
- [58] Xinyi Liu, Baofeng Zhang, and Na Liu. Cast-yolo: An improved yolo based on a cross-attention strategy transformer for foggy weather adaptive detection. *Applied Sciences*, 13(2):1176, 2023.
- [59] Munkhjargal Gochoo, Munkh-Erdene Otgonbold, Erkhembayar Ganbold, Jun-Wei Hsieh, Ming-Ching Chang, Ping-Yang Chen, Byambaa Dorj, Hamad Al Jassmi, Ganzorig Batnasan, Fady Alnajjar, et al. Fisheye8k: A benchmark and dataset for fisheye camera object detection. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5305–5313, 2023.
- [60] Cen Han, Guangyu Gao, and Yu Zhang. Real-time small traffic sign detection with revised faster-rcnn. *Multimedia Tools and Applications*, 78:13263–13278, 2019.

- [61] Lucas C Possatti, Rânik Guidolini, Vinicius B Cardoso, Rodrigo F Berriel, Thiago M Paixão, Claudine Badue, Alberto F De Souza, and Thiago Oliveira-Santos. Traffic light recognition using deep learning and prior maps for autonomous cars. *2019 international joint conference on neural networks (IJCNN)*, pages 1–8, 2019.
- [62] A Ashwini, KE Purushothaman, Banu Priya Prathaban, M Jenath, and R Prasanna. Automatic traffic sign board detection from camera images using deep learning and binarization search algorithm. *2023 International Conference on Recent Advances in Electrical, Electronics, Ubiquitous Communication, and Computational Intelligence (RAEEUCCI)*, pages 1–5, 2023.
- [63] Xiaoqian Yu, Yujuan Si, and Liangliang Li. Pedestrian detection based on improved faster rcnn algorithm. *2019 IEEE/CIC International Conference on Communications in China (ICCC)*, pages 346–351, 2019.
- [64] Dong Doan Van. Application of advanced deep convolutional neural networks for the recognition of road surface anomalies. *Engineering, Technology and Applied Science Research*, 13(3):10765–10768, 2023.
- [65] Murat Tasyurek and Ertugrul Gul. A new deep learning approach based on grayscale conversion and dwt for object detection on adversarial attacked images. *The Journal of Supercomputing*, 79(18):20383–20416, 2023.
- [66] Xinchao Liu and Yier Lin. Yolo-gw: quickly and accurately detecting pedestrians in a foggy traffic environment. *Sensors*, 23(12):5539, 2023.
- [67] Saim Khalid, Hadi Mohsen Oqaibi, Muhammad Aqib, and Yaser Hafeez. Small pests detection in field crops using deep learning object detection. *Sustainability*, 15(8):6815, 2023.
- [68] Kh Tohidul Islam, Sudanthi Wijewickrema, Ram Gopal Raj, and Stephen O’Leary. Street sign recognition using histogram of oriented gradients and artificial neural networks. *Journal of imaging*, 5(4):44, 2019.
- [69] Murat Taşyürek. Odrp: a new approach for spatial street sign detection from exif using deep learning-based object detection, distance estimation, rotation and projection system. *The Visual Computer*, 40(2):983–1003, 2024.
- [70] Ibrahim Kandel, Mauro Castelli, and Luca Manzoni. Brightness as an augmentation technique for image classification. *Emerging Science Journal*, 6(4):881–892, May 2022. doi: 10.28991/ESJ-2022-06-04-015. URL <https://www.ijournalse.org/index.php/ESJ/article/view/930>.
- [71] Mokhles M. Abdulghani, Abdulghani M. and Abdulghani, Wilbur L. Walters, and Khalid H. Abed. Data augmentation using brightness and darkness to enhance the performance of yolo7 object detection algorithm. *2023 Congress in Computer Science, Computer Engineering, and Applied Computing (CSCE)*, pages 351–356, 2023. doi: 10.1109/CSCE60160.2023.00061.
- [72] Maik Wischow, Guillermo Gallego, Ines Ernst, and Anko Börner. Monitoring and adapting the physical state of a camera for autonomous vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 25(5):4290–4303, 2024. doi: 10.1109/TITS.2023.3328811.

- [73] V Indumathi, R Gopalakrishnan, C Vijayakumar, C Manoj, K Dhinesh, and M Madhan Kumar. Employing algorithms for machine learning, anomalies in automated contexts can be detected. *2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, pages 1–5, 2024. doi: 10.1109/ICCCNT61001.2024.10725333.
- [74] Ristea Nicolae-Cătălin Ionescu Radu Tudor Sebe Nicu Croitoru, Florinel-Alin. Learning rate curriculum. *International Journal of Computer Vision*, 133(1):291–314, 2025. doi: 10.1007/s11263-024-02186-5. URL <https://doi.org/10.1007/s11263-024-02186-5>.
- [75] Runwu Shi, Shichun Yang, Yuyi Chen, Rui Wang, Mengyue Zhang, Jiayi Lu, and Yaoguang Cao. Cnn-transformer for visual-tactile fusion applied in road recognition of autonomous vehicles. *Pattern Recognition Letters*, 166:200–208, 2023. ISSN 0167-8655. doi: <https://doi.org/10.1016/j.patrec.2022.11.023>. URL <https://www.sciencedirect.com/science/article/pii/S016786552200352X>.
- [76] Satya Prakash Sahu Deepak Kumar Dewangan. A review of convolutional neural networks in computer vision. *Multimedia Tools and Applications*, 82(05):7293, 2023. ISSN 1573-7721. doi: <https://doi.org/10.1007/s11042-022-13425-7>.
- [77] Yingjuan Tang, Hongwen He, and Yong Wang. Hierarchical vector transformer vehicle trajectories prediction with diffusion convolutional neural networks. *Neurocomputing*, 580: 127526, 2024. ISSN 0925-2312. doi: <https://doi.org/10.1016/j.neucom.2024.127526>. URL <https://www.sciencedirect.com/science/article/pii/S0925231224002972>.
- [78] Yang Liu, Zheng Zheng, Fangyun Qin, Xiaoyi Zhang, and Haonan Yao. A residual convolutional neural network based approach for real-time path planning. *Knowledge-Based Systems*, 242:108400, 2022. ISSN 0950-7051. doi: <https://doi.org/10.1016/j.knosys.2022.108400>. URL <https://www.sciencedirect.com/science/article/pii/S0950705122001629>.
- [79] Jiacong Xiong, Jiankui Chen, Wei Chen, Xiao Yue, Ziwei Zhao, and Zhouping Yin. Intelligent path planning algorithm system for printed display manufacturing using graph convolutional neural network and reinforcement learning. *Journal of Manufacturing Systems*, 79:73–85, 2025. ISSN 0278-6125. doi: <https://doi.org/10.1016/j.jmsy.2024.12.016>. URL <https://www.sciencedirect.com/science/article/pii/S0278612524003261>.
- [80] Giorgi Roberto Procaccini Marco, Sahebi Amin. A survey of graph convolutional networks (gcns) in fpga-based accelerators. *Journal of Big Data*, 11(1):163, 2024. doi: 10.1186/s40537-024-01022-4. URL <https://doi.org/10.1186/s40537-024-01022-4>.
- [81] Lei He, Nabil Aouf, and Bifeng Song. Explainable deep reinforcement learning for uav autonomous path planning. *Aerospace Science and Technology*, 118:107052, 2021. ISSN 1270-9638. doi: <https://doi.org/10.1016/j.ast.2021.107052>. URL <https://www.sciencedirect.com/science/article/pii/S1270963821005629>.
- [82] Lixing Liu, Xu Wang, Xin Yang, Hongjie Liu, Jianping Li, and Pengfei Wang. Path planning techniques for mobile robots: Review and prospect. *Expert Systems with Applications*, 227: 120254, 2023. ISSN 0957-4174. doi: <https://doi.org/10.1016/j.eswa.2023.120254>. URL <https://www.sciencedirect.com/science/article/pii/S095741742300756X>.

- [83] Jiankun Wang, Jianbang Liu, Weinan Chen, Wenzheng Chi, and Max Q.-H. Meng. Robot path planning via neural-network-driven prediction. *IEEE Transactions on Artificial Intelligence*, 3(3):451–460, 2022. doi: 10.1109/TAI.2021.3119890.
- [84] Wei Lan, Xiang Jin, Xin Chang, and Han Zhou. Based on deep reinforcement learning to path planning in uncertain ocean currents for underwater gliders. *Ocean Engineering*, 301: 117501, 2024. ISSN 0029-8018. doi: <https://doi.org/10.1016/j.oceaneng.2024.117501>. URL <https://www.sciencedirect.com/science/article/pii/S0029801824008382>.
- [85] Fan Yang, Chen Wang, Cesar Cadena, and Marco Hutter. iplanner: Imperative path planning. *Journal Computer Science > Robotics*, arXiv:142–149, 2023. ISSN 2302.11434. doi: <https://doi.org/10.48550/arXiv.2302.11434>. URL <https://arxiv.org/abs/2302.11434>.
- [86] Mohamed Reda, Ahmed Onsy, Amira Y. Haikal, and Ali Ghanbari. Path planning algorithms in the autonomous driving system: A comprehensive review. *Robotics and Autonomous Systems*, 174:104630, 2024. ISSN 0921-8890. doi: <https://doi.org/10.1016/j.robot.2024.104630>. URL <https://www.sciencedirect.com/science/article/pii/S0921889024000137>.
- [87] Milena F. Pinto Gabriel G. R. de Castro, Iago Z. Biundini, Aurelio G. Melo, Andre L. M. Marcato, and Diego B. Haddad. Dynamic path planning based on neural networks for aerial inspection. *Journal of Control, Automation and Electrical Systems*, 34(01):85–105, 2023. ISSN 2195-3899. doi: <https://doi.org/10.1007/s40313-022-00941-y>.
- [88] Manuel Méndez, Mercedes G. Merayo, and Manuel Núñez. Long-term traffic flow forecasting using a hybrid cnn-bilstm model. *Engineering Applications of Artificial Intelligence*, 121: 106041, 2023. ISSN 0952-1976. doi: <https://doi.org/10.1016/j.engappai.2023.106041>. URL <https://www.sciencedirect.com/science/article/pii/S0952197623002257>.
- [89] Ziyi Chen, Yuhua Luo, Jing Wang, Jonathan Li, Cheng Wang, and Dilong Li. Dpenet: Dual-path extraction network based on cnn and transformer for accurate building and road extraction. *International Journal of Applied Earth Observation and Geoinformation*, 124: 103510, 2023. ISSN 1569-8432. doi: <https://doi.org/10.1016/j.jag.2023.103510>. URL <https://www.sciencedirect.com/science/article/pii/S1569843223003345>.
- [90] Jingjing Tao, Zhe Chen, Zhongchang Sun, Huadong Guo, Bo Leng, Zhengbo Yu, Yanli Wang, Ziqiong He, Xiangqi Lei, and Jinpei Yang. Seg-road: A segmentation network for road extraction based on transformer and cnn with connectivity structures. *Remote Sensing*, 15(6), 2023. ISSN 2072-4292. doi: 10.3390/rs15061602. URL <https://www.mdpi.com/2072-4292/15/6/1602>.
- [91] Alessandro Di Benedetto, Margherita Fiani, and Lucas Matias Gujski. U-net-based cnn architecture for road crack segmentation. *Infrastructures*, 8(5), 2023. ISSN 2412-3811. doi: <https://doi.org/10.3390/infrastructures8050090>. URL <https://www.mdpi.com/2412-3811/8/5/90>.
- [92] Mohd Jawed Khan and Pankaj Pratap Singh. Advanced road extraction using cnn-based u-net model and satellite imagery. *e-Prime - Advances in Electrical Engineering, Electronics and Energy*, 5:100244, 2023. ISSN 2772-6711. doi: <https://doi.org/10.1016/j.eprime.2023.100244>.

- prime.2023.100244. URL <https://www.sciencedirect.com/science/article/pii/S2772671123001390>.
- [93] Kanhui Lyu. A nonlinear convolutional neural network algorithm for autonomous vehicle lane line detection. *High-performance Computing Algorithms for Material Sciences*, 25: 4237–4245, 2024. ISSN 1895-1767. doi: <https://doi.org/10.12694/scpe.v25i5.3189>. URL <https://scpe.org/index.php/scpe/article/view/3189>.
- [94] Der-Hau Lee, , and Jinn-Liang Liu. End-to-end deep learning of lane detection and path prediction for real-time autonomous driving. *Signal, Image and Video Processing*, 17(01): 199–205, 2023. ISSN 1863-1711. doi: <https://doi.org/10.1007/s11760-022-02222-2>.
- [95] Muneeb A. Khan, Heemin Park, and Jinseok Chae. A lightweight convolutional neural network (cnn) architecture for traffic sign recognition in urban road networks. *Electronics*, 12(8), 2023. ISSN 2079-9292. doi: 10.3390/electronics12081802. URL <https://www.mdpi.com/2079-9292/12/8/1802>.
- [96] Ziren Xiao, Peisong Li, Chang Liu, Honghao Gao, and Xinheng Wang. Macns: A generic graph neural network integrated deep reinforcement learning based multi-agent collaborative navigation system for dynamic trajectory planning. *Information Fusion*, 105:102250, 2024. ISSN 1566-2535. doi: <https://doi.org/10.1016/j.inffus.2024.102250>. URL <https://www.sciencedirect.com/science/article/pii/S1566253524000289>.
- [97] Gabriel G. R. de Castro, Guido S. Berger, Alvaro Cantieri, Marco Teixeira, José Lima, Ana I. Pereira, and Milena F. Pinto. Adaptive path planning for fusing rapidly exploring random trees and deep reinforcement learning in an agriculture dynamic environment uavs. *Agriculture*, 13(2), 2023. ISSN 2077-0472. doi: 10.3390/agriculture13020354. URL <https://www.mdpi.com/2077-0472/13/2/354>.
- [98] Hyansu Bae, Gidong Kim, Jonguk Kim, Dianwei Qian, and Sukgyu Lee. Multi-robot path planning method using reinforcement learning. *Applied Sciences*, 9(15), 2019. ISSN 2076-3417. doi: 10.3390/app9153057. URL <https://www.mdpi.com/2076-3417/9/15/3057>.
- [99] Kranti Kumar, Manoj Kumar, and Pritikana Das. Traffic congestion forecasting using multilayered deep neural network. *Transportation Letters*, 16(6):516–526, 2024. doi: 10.1080/19427867.2023.2207278.
- [100] Xia Zhao, Limin Wang, Yufei Zhang, Xuming Han, Muhammet Deveci, and Milan Parmar. A review of convolutional neural networks in computer vision. *Artificial Intelligence Review*, 57(04):99, 2008. ISSN 1573-7462. doi: <https://doi.org/10.1007/s10462-024-10721-6>.
- [101] James Orr and Ayan Dutta. Multi-agent deep reinforcement learning for multi-robot applications: A survey. *Sensors*, 23(7), 2023. ISSN 1424-8220. doi: 10.3390/s23073625. URL <https://www.mdpi.com/1424-8220/23/7/3625>.
- [102] Zichao Zhang, Shubo Wang, Jian Chen, and Yu Han. A bionic dynamic path planning algorithm of the micro uav based on the fusion of deep neural network optimization/filtering and hawk-eye vision. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 53(6): 3728–3740, 2023. doi: 10.1109/TSMC.2023.3233965.

- [103] Jiaen Zhou, Zezhong Sun, Ronghui Zhang, Guangrong Lin, Shenhui Zhang, and Yafei Zhao. A cloud-edge collaboration cnn-based routing method for isac in leo satellite networks. *Proceedings of the 2nd Workshop on Integrated Sensing and Communications for Metaverse*, page 25–29, 2023. doi: 10.1145/3597065.3597451. URL <https://doi.org/10.1145/3597065.3597451>.
- [104] Andrius Kriščiūnas, Dalia Čalnerytė, Tautvydas Fyleris, Tadas Jurgutis, Dalius Makackas, and Rimantas Barauskas. Convolutional neural network-based approximation of coverage path planning results for parking lots. *ISPRS International Journal of Geo-Information*, 12(8), 2023. ISSN 2220-9964. doi: 10.3390/ijgi12080313. URL <https://www.mdpi.com/2220-9964/12/8/313>.
- [105] Marwan Abdou Dhuheir, Emna Baccour, Aiman Erbad, Sinan Sabeeh Al-Obaidi, and Mounir Hamdi. Deep reinforcement learning for trajectory path planning and distributed inference in resource-constrained uav swarms. *IEEE Internet of Things Journal*, 10(9):8185–8201, May 2023. ISSN 2372-2541. doi: 10.1109/jiot.2022.3231341. URL <http://dx.doi.org/10.1109/JIOT.2022.3231341>.
- [106] Yuzhen Li, Jiawen Tang, Han Zhao, and Ruikang Luo. Reinforcement learning method with dynamic learning rate for real-time route guidance based on sumo. *2022 17th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pages 820–824, 2022. doi: 10.1109/ICARCV57592.2022.10004227.
- [107] Semiye Demircan, Musa Aydin, and S. Savas Durduran. Finding optimum route of electrical energy transmission line using multi-criteria with q-learning. *Expert Systems with Applications*, 38(4):3477–3482, 2011. ISSN 0957-4174. doi: <https://doi.org/10.1016/j.eswa.2010.08.135>. URL <https://www.sciencedirect.com/science/article/pii/S0957417410009437>.
- [108] Anderson Souto, Rodrigo Alfaia, Evelin Cardoso, Jasmine Araújo, and Carlos Francês. Uav path planning optimization strategy: Considerations of urban morphology, microclimate, and energy efficiency using q-learning algorithm. *Drones*, 7(2), 2023. ISSN 2504-446X. doi: 10.3390/drones7020123. URL <https://www.mdpi.com/2504-446X/7/2/123>.
- [109] Muzamil Eltejani Mohammed ALI. Evaluating action durations for adaptive traffic signal control based on deep q-learning. *International Journal of Intelligent Transportation Systems Research*, 19(3), 2021. ISSN 1868-8659. doi: 10.1007/s13177-021-00262-5. URL <https://doi.org/10.1007/s13177-021-00262-5>.
- [110] Nada Faqir, Nouredine En-Nahnahi, and Jaouad Boumhidi. Deep q-learning approach for congestion problem in smart cities. *2020 Fourth International Conference On Intelligent Computing in Data Sciences (ICDS)*, pages 1–6, 2020. doi: 10.1109/ICDS50568.2020.9268709.
- [111] Chaymae Chouiekh, Ali Yahyaouy, Abdellah Aarab, and Abdelouahed Sabri. Road traffic: Deep q-learning agent control traffic lights in the intersection. *2022 International Conference on Intelligent Systems and Computer Vision (ISCV)*, pages 1–5, 2022. doi: 10.1109/ISCV54655.2022.9806135.

- [112] Xiaoyuan Liang, Xunsheng Du, Guiling Wang, and Zhu Han. A deep reinforcement learning network for traffic light cycle control. *IEEE Transactions on Vehicular Technology*, 68(2): 1243–1253, 2019. doi: 10.1109/TVT.2018.2890726.
- [113] Mengyu Guo, Pin Wang, Ching-Yao Chan, and Sid Askary. A reinforcement learning approach for intelligent traffic signal control at urban intersections. *IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 4242–4247, 2019. doi: 10.1109/ITSC.2019.8917268.
- [114] Ee Soong Low, Pauline Ong, and Cheng Yee Low. A modified q-learning path planning approach using distortion concept and optimization in dynamic environment for autonomous mobile robot. *Computers and Industrial Engineering*, 181:109338, 2023. ISSN 0360-8352. doi: <https://doi.org/10.1016/j.cie.2023.109338>. URL <https://www.sciencedirect.com/science/article/pii/S0360835223003625>.
- [115] Shenghua Xu, Yang Gu, Xiaoyan Li, Cai Chen, Yingyi Hu, Yu Sang, and Wenxing Jiang. Indoor emergency path planning based on the q-learning optimization algorithm. *ISPRS International Journal of Geo-Information*, 11(1), 2022. ISSN 2220-9964. doi: 10.3390/ijgi11010066. URL <https://www.mdpi.com/2220-9964/11/1/66>.
- [116] Anum Mushtaq, Irfan Ul Haq, Muhammad Azeem Sarwar, Asifullah Khan, Wajeeha Khalil, and Muhammad Abid Mughal. Multi-agent reinforcement learning for traffic flow management of autonomous vehicles. *Sensors*, 23(5), 2023. ISSN 1424-8220. doi: 10.3390/s23052373. URL <https://www.mdpi.com/1424-8220/23/5/2373>.
- [117] Chen Chen, Jiange Jiang, Ning Lv, and Siyu Li. An intelligent path planning scheme of autonomous vehicles platoon using deep reinforcement learning on network edge. *IEEE Access*, 8:99059–99069, 2020. doi: 10.1109/ACCESS.2020.2998015.
- [118] Vahide Bulut. Uav path planning optimization strategy: Considerations of urban morphology, microclimate, and energy efficiency using q-learning algorithm. *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, 44(3):106, 2022. ISSN 1806-3691. doi: 10.1007/s40430-022-03399-w. URL <https://doi.org/10.1007/s40430-022-03399-w>.
- [119] Ee Soong Low, Pauline Ong, and Kah Chun Cheah. Solving the optimal path planning of a mobile robot using improved q-learning. *Robotics and Autonomous Systems*, 115: 143–161, 2019. ISSN 0921-8890. doi: <https://doi.org/10.1016/j.robot.2019.02.013>. URL <https://www.sciencedirect.com/science/article/pii/S0921889018308285>.
- [120] Branka Mirchevska, Maria Hügler, Gabriel Kalweit, Moritz Werling, and Joschka Boedecker. Amortized q-learning with model-based action proposals for autonomous driving on highways. *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1028–1035, 2021. doi: 10.1109/ICRA48506.2021.9560777.
- [121] M. Ugur Yavas, N. Kemal Ure, and Tufan Kumbasar. A new approach for tactical decision making in lane changing: Sample efficient deep q learning with a safety feedback reward. *Computer Science, arXiv:2009.11905*, 2020. URL <https://arxiv.org/abs/2009.11905>.
- [122] Hongwei Ge, Yumei Song, Chunguo Wu, Jiankang Ren, and Guozhen Tan. Cooperative deep q-learning with q-value transfer for multi-intersection signal control. *IEEE Access*, 7: 40797–40809, 2019. doi: 10.1109/ACCESS.2019.2907618.

- [123] Kanghua Ma, Shubing Liao, and Yunyun Niu. Connected vehicles' dynamic route planning based on reinforcement learning. *Future Generation Computer Systems*, 153:375–390, 2024. ISSN 0167-739X. doi: <https://doi.org/10.1016/j.future.2023.11.037>. URL <https://www.sciencedirect.com/science/article/pii/S0167739X2300451X>.
- [124] Songsang Koh, Bo Zhou, Hui Fang, Po Yang, Zaili Yang, Qiang Yang, Lin Guan, and Zhigang Ji. Real-time deep reinforcement learning based vehicle navigation. *Applied Soft Computing*, 96:106694, 2020. ISSN 1568-4946. doi: <https://doi.org/10.1016/j.asoc.2020.106694>. URL <https://www.sciencedirect.com/science/article/pii/S1568494620306323>.
- [125] Yuepeng Hu, Lehan Yang, and Yizhu Lou. Path planning with q-learning. *Journal of Physics: Conference Series*, 1948(1):012038, jun 2021. doi: 10.1088/1742-6596/1948/1/012038. URL <https://dx.doi.org/10.1088/1742-6596/1948/1/012038>.
- [126] Hung-Chin Jang and Chun-Yee Lee. Study on q-learning and deep q-learning in urban roadside parking space search. *2022 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 1248–1253, 2022. doi: 10.1109/CSCI58124.2022.00225.
- [127] Kerang Cao, Liwei Wang, Shuo Zhang, Lini Duan, Guimin Jiang, Stefano Sfarra, Hai Zhang, and Hoekyung Jung. Optimization control of adaptive traffic signal with deep reinforcement learning. *Electronics*, 13(1), 2024. ISSN 2079-9292. doi: 10.3390/electronics13010198. URL <https://www.mdpi.com/2079-9292/13/1/198>.
- [128] Qian Zhou, Yang Lian, Jiayang Wu, Mengyue Zhu, Haiyong Wang, and Jinli Cao. An optimized q-learning algorithm for mobile robot local path planning. *Knowledge-Based Systems*, 286:111400, 2024. ISSN 0950-7051. doi: <https://doi.org/10.1016/j.knosys.2024.111400>. URL <https://www.sciencedirect.com/science/article/pii/S0950705124000352>.
- [129] Selim Reza, Marta Campos Ferreira, J. J. M. Machado, and João Manuel R. S. Tavares. A citywide td-learning based intelligent traffic signal control for autonomous vehicles: Performance evaluation using sumo. *Expert Systems*, 42(1):e13301, 2025. doi: <https://doi.org/10.1111/exsy.13301>. URL <https://onlinelibrary.wiley.com/doi/abs/10.1111/exsy.13301>.
- [130] Raihan MD Golam and Naoki Fukuta. Toward a framework and sumo-based simulation for smart traffic control using multiagent learning. *10th International Congress on Advanced Applied Informatics (IIAI-AAI)*, pages 544–548, 2021. doi: 10.1109/IIAI-AAI53430.2021.00098.