# CSE 5312: Traffic Identification in CARLA using Deep Learning

Matthew Caro, Kevin Huang, Ankith Nagabandi

February 9, 2026

# Contents

# 1. Abstract

Self-driving systems are on the rise in the world and they are rapidly being integrated into everyday objects like cars. However, not all of them are able to perform in a real-time system where failure can lead to heavy damages and injuries. This project proposes and presents an IoT structured Deep Learning architecture, designed to perform object detection and classification quickly and accurately in a real-time system. The proposed system is designed and trained using data procured from CARLA, a software that can simulate various driving situations. With this data, we achieved an overall best 78.5% accuracy. This paper details the methods we took to accomplish this and how we can use this to make driving and the roads safer with Artificial Intelligence and Deep Learning.

# 2. Introduction

CARLA is a popular vehicle simulation software, a "Digital Twin" of a City Environment. Similarly to the real world, we can simulate situations that include traffic lights, road sensors, and connected vehicles, allowing continuous data recording. With this software, the user can control all the factors on the road and then save these features for later use.

Some of the general objectives are developing and generating a diverse traffic dataset and simulating traffic scenarios through normal flow, accident-induced bottlenecks, and road closures. This can be accomplished by using CARLA.

Using the data generated by CARLA, we can then train a predictive AI model to properly identify the different features of the environment and react to them.

# 3. Projective Motivations and Objectives

## 3.1 Objectives

- Generate a diverse traffic dataset in CARLA, generating script scenarios such as normal flow, accident-induced bottlenecks, and road closures.
- Develop and train a predictive model (e.g., GNN or CNN) that takes the recent traffic history and current visuals as input and accurately forecasts future traffic states (vehicle positions and velocities).
- Implement machine learning models that accurately identify and classify surrounding objects.
- Implement a proactive vehicle driving system within CARLA that uses the model's predictions to make real-time decisions (e.g., moving forward, slow down, or stop).
- Evaluate the models accuracy by comparing key performance metrics .

## 3.2 Expected Outcome

In the end, we can expect to have a functional proof-of-concept demonstrating a closed-loop system where a simulation generates data, an AI model that learns from it, and the model's intelligence is used to optimize driving in the simulation itself. The final deliverable will include the trained model, that should be able to intake images live and properly identify different objects.

# 4. High-level Design

First, using CARLA, we can generate data simulating real world scenarios. Then, using the data generated by CARLA, we can train a predictive AI model to test proactive traffic management strategies in real-time in the simulation. Early research suggests that some of the best models to implement are Graph Neural Networks (GNNs) and Convolutional Neural Network.s (CNNs). We can also validate and test these models using more data generated in CARLA.

## 5. Hardware and Software Requirements

- Python3 to attach scripts to the simulation environment and run scenarios in real-time
- CARLA
  - Windows or Ubuntu
  - 135 GB of storage space to build CARLA and run Unreal Engine
  - Dedicated GPU with at least 6 GB VRAM

## 6. Deliverables and Timeline

- Week 6- Project Proposal
- Week 7- Set Up Environment and Divide Tasks
- Week 8- Attempted to Generate Data
- Week 9- Continued to try for CARLA
- Week 10- Start Development of CNN
- Week 11- Finish CNN and Start GNN
- Week 12- Finish GNN and prepare presentation
- Week 13- Final Presentation / Start Final Report
- Week 14- Finish and Revise Final Report

## 7. Results and Model Analysis

For our project, we looked at two main models for object classification. Our models included a CNN (specifically used ResNet50), GNN as well as a combined CNN and GNN.

### 7.1 Data Collection

To start, let's talk about our attempt to collect data and how we did it. For the process of our data collection, we found a GitHub repository containing Jupyter Notebooks with scripts to generate and collect data. This script would connect to the CARLA simulator and start a scenario. It would generate the user's car on a map along with other vehicles and passengers. It would set everything on an autopilot mode and have it run. At the same time, the data collector would record the frames from the short scenario and store them locally. We modified the script to help generate the scenario with random weather and a random vehicle. The weather scenarios that were available includes sunny, clear night, rainy day, rainy night, and a few more.

Unfortunately, the software was quite limited and had very specific requirements. For example, there were a variety of weather scenarios, but snowy weather scenarios was not available. In addition, CARLA can only run on Windows or Linux and we all had MacBooks for our daily use and Windows on personal desktops that we aren't near all the time. CARLA has very specific requirements such as the Python version must match the CARLA version as closely as possible. As a result, we spent several hours trying to find the right combination of Python version and CARLA version. Following that, we had to find how to allow the script to connect to CARLA. The software had to run concurrently with the script and the script requires direct connection with it. Here, we had to create an inbound rule at a specific port for CARLA to enter and the script would connect.

Next, we also had to find the right version for many of the packages it requires such as Pandas and NumPy. There was also a carla library that was required and initially, we had no further problems after set up to run it. However, after we tried to start up the script again, the script refused to connect. We attempted resetting up everything in

a conda environment to start with a clean slate with no success. As a result, we moved away from generating our dataset and moved to using pre-labeled datasets we found on Kaggle. Later, after further discussion with another group, we found that when installing the carla library, it defaults to a specific version. However, that version was incompatible with the software and thus, unable to run. Due to timing and simplicity sake, we decided to continue with the Kaggle datasets.

## 7.2 Dataset 1

In this dataset we are focused on object detection tasks. It contains images captured from CARLA with bounding boxes labeling the detected objects. They are also annotated in three different formats- Pascal VOC Format (XML files containing labels) and YOLO. Some of the object classes includes vehicles (cars, trucks), bikes, motorbikes, traffic lights, and traffic signs.

| Category | Details |
|---|---|
| Classes | Bike, Motorbike, Traffic Light, Traffic Sign, Vehicle |
| Label Format | YOLO Labels |
| Train Samples | 2556 (58.9%) |
| Test Samples | 1839 (41.8%) |

Table 1: Dataset I Summary for Object Classification

### 7.2.1 CNN

To work with dataset 1, we implemented a Convolutional Neural Network (CNN) capable of detecting and classifying several object types. These included Bikes, Motorbikes, Traffic Lights, Traffic Signs, and various other vehicles commonly found in urban environments.

As shown in Figure 1, each original image is paired with its corresponding output image containing bounding boxes. Bounding boxes serve as a visual tool for locating objects within an image by outlining the object and a small portion of the surrounding area. This approach is especially valuable in complex scenes, such as busy streets with multiple vehicles and traffic elements. The model can differentiate between objects by drawing boxes in distinct colors and assigning confidence scores, making the results easy to interpret.

Overall, the model performed reasonably well given the size and diversity of the dataset. It was generally able to identify and classify objects even when they appeared smaller or farther away in the frame. This provided a solid foundation for understanding how our pipeline handles object detection before moving on to larger and more varied datasets.


Figure 1: Labeling - DS1

The performance of CNN I shows a strong overall accuracy of 99%, which suggests that the model learned the dataset extremely well. Most classes—particularly bike, traffic-light, and vehicle—achieve very high precision, recall, and F1-scores. This is reflected in the confusion matrix, where almost all samples fall cleanly along the diagonal.

However, the motorbike class stands as a weakness. Its precision is 0.385 and F1-score 0.556, meaning the model struggles to correctly identify motorbikes and often confuses them with other vehicle types. This is also visible in the confusion matrix, where the motorbike column and row have noticeable off-diagonal activity. The reason behind the issue of the classification of motorbikes could be a smaller sample size was included within the dataset, or even the classification model could have been confused with the normal bike. Other than that, the other objects seem to have incredible success with their precision, recall and F1 Score, individually.

The averages (precision 0.876, F1 0.898) reflect this imbalance—despite high overall accuracy, the model is not uniformly strong across all categories.

If there were more samples included in this dataset, then I believe we would have consistent success, and even 100% accuracy if there weren't any issues with the motorbike.

Within the 9 samples that were given, we can see that there is 100% success with the classification of those objects, according to figure 3.

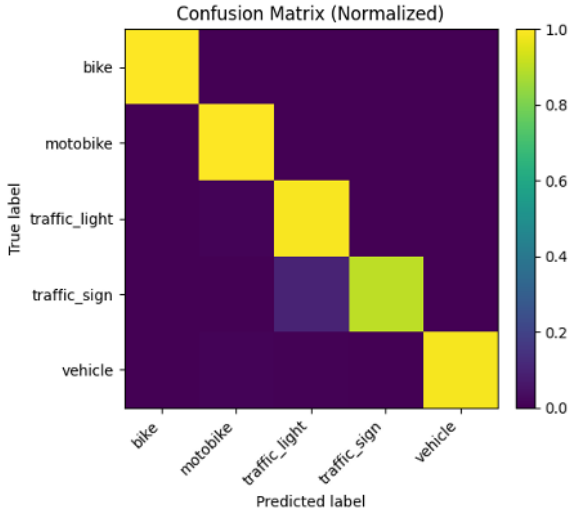


Figure 2: *
(a) CNN - DS1

```
Sample 00: true=vehicle, pred=vehicle
Sample 01: true=vehicle, pred=vehicle
Sample 02: true=bike, pred=bike
Sample 03: true=vehicle, pred=vehicle
Sample 04: true=traffic_light, pred=traffic_light
Sample 05: true=traffic_light, pred=traffic_light
Sample 06: true=traffic_light, pred=traffic_light
Sample 07: true=traffic_light, pred=traffic_light
Sample 08: true=bike, pred=bike
```

Figure 3: *
(b) CNN - DS1

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| bike | 1.000 | 1.000 | 1.000 | 30 |
| motobike | 0.385 | 1.000 | 0.556 | 10 |
| traffic_light | 0.999 | 0.991 | 0.995 | 1586 |
| traffic_sign | 1.000 | 0.900 | 0.947 | 10 |
| vehicle | 0.995 | 0.985 | 0.990 | 203 |
| accuracy | | | 0.990 | 1839 |
| macro avg | 0.876 | 0.975 | 0.898 | 1839 |
| weighted avg | 0.995 | 0.990 | 0.992 | 1839 |

Figure 4: *
(c) CNN - DS1

Figure 5: CNN Results for Dataset 1

### 7.2.2 GNN

For our GNN, we focused on using the first dataset. The GNN attempted to learn and find the correlations, but the performance is highly imbalanced, class-dependent, and is strongly affected by the dataset design. It performs relatively well identifying traffic lights and to some extent, traffic signs. These classes have the highest recall, precision, and F1 scores, suggesting they are consistent and have distinguishable features. In comparison, it does not perform as well on pedestrians, cyclists, and in some cases, vehicles. They have extremely low recall and F1 scores and tend to have heavy confusion with other classes. This indicates poor feature separation for these objects, weak or noisy graph connectivity, an imbalanced dataset, and the node features do not capture the visual characteristics well.

The confusion matrix shows a degree of systematic misclassification where vehicles are often misclassified, pedestrians and cyclists are often confused with each other, traffic lights are detected extremely well, and traffic signs have mixed results. This means the GNN is unable to capture enough information to distinguish pedestrians, cyclists, and vehicles, but has captured strong cues for traffic control objects.

There is a good chance that the model is overfitting on some classes, but some classes are extremely small in comparison, showing major class imbalances.

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| vehicle | 0.894 | 0.470 | 0.616 | 1172 |
| pedestrian | 0.351 | 0.940 | 0.511 | 151 |
| cyclist | 0.401 | 0.827 | 0.540 | 81 |
| traffic_light | 0.734 | 0.827 | 0.778 | 1057 |
| traffic_sign | 0.517 | 0.968 | 0.674 | 95 |
| accuracy | | | 0.675 | 2556 |
| macro avg | 0.580 | 0.807 | 0.624 | 2556 |
| weighted avg | 0.767 | 0.675 | 0.677 | 2556 |

Figure 6: *
(a) GNN - Training

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| vehicle | 0.833 | 0.172 | 0.286 | 203 |
| pedestrian | 0.179 | 0.900 | 0.298 | 30 |
| cyclist | 0.250 | 0.600 | 0.353 | 10 |
| traffic_light | 0.905 | 0.915 | 0.910 | 1586 |
| traffic_sign | 0.500 | 0.900 | 0.643 | 10 |
| accuracy | | | 0.831 | 1839 |
| macro avg | 0.533 | 0.697 | 0.498 | 1839 |
| weighted avg | 0.879 | 0.831 | 0.826 | 1839 |

Figure 7: *
(b) GNN - Testing

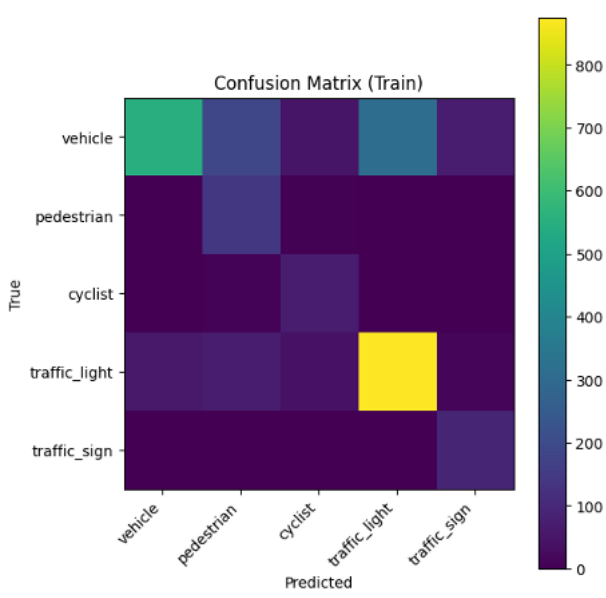


Figure 8: *
(c) GNN - DS1

```
Graph idx: 0
Image path: carla-object-detection-dataset\images\train\Town01_001440.png
Num nodes: 1
True labels:         [4]
Predicted labels:    [4]
Recommended action: GO
------------------------------------------------------------
Graph idx: 546
Image path: carla-object-detection-dataset\images\train\Town03_017580.png
Num nodes: 5
True labels:         [1, 2, 0, 3, 0]
Predicted labels:    [2, 2, 2, 2, 2]
Recommended action: SLOW
------------------------------------------------------------
Graph idx: 134
Image path: carla-object-detection-dataset\images\train\Town01_012480.png
Num nodes: 1
True labels:         [3]
Predicted labels:    [3]
Recommended action: GO
------------------------------------------------------------
Graph idx: 124
Image path: carla-object-detection-dataset\images\train\Town01_011820.png
Num nodes: 2
True labels:         [0, 3]
...
Num nodes: 1
True labels:         [0]
Predicted labels:    [0]
Recommended action: GO
```

Figure 9: *
(d) GNN - DS1

Figure 10: GNN Results for Dataset 1

## 7.3 Dataset 2

In this second dataset we are using, there are thirteen primary objects that were labeled- Traffic Signs, Buildings, Fences, Other, Pedestrians, Poles, Road Lines, Road, Sidewalks, Vegetation, Cars, Walls, and a few unlabeled data. Each image is annotated with dense per-pixel semantic labels, meaning every pixel in the image is labeled with a semantic class. Unlike the previous dataset, we are using dense segmentation. Each class is labeled with a different color to help with the classification.For this dataset, we worked with around 109,000 images, a huge upgrade compared to dataset I.

| Category | Details |
|---|---|
| Classes | Traffic Sign, Building, Fence, Road Line, Car, Wall, Unlabeled |
| Label Format | RGB Color Code |
| Train Samples | 76828 (70%) |
| Test Samples | 32927 (30%) |

Table 2: Dataset II Summary for Object Classification

### 7.3.1 CNN

We also evaluated our CNN on a second dataset. However, this time around for classification, instead of bounding boxes, we relied RGB codes to identify the 13 different classes, as it had almost tripled in size.

In Figure 11 , we can see that in the original image, there are a variety of objects, including a person on a bike, a car, traffic lights, poles, and vegetation. We can see that the labels on the right, are multiple distinct colors representing different objects.

When we trained and tested the CNN on this dataset, we achieved an overall accuracy of 78.5%. From the results in Figure 12, we can see that while performance varies across categories, the model demonstrates solid classification ability for many of the major object types

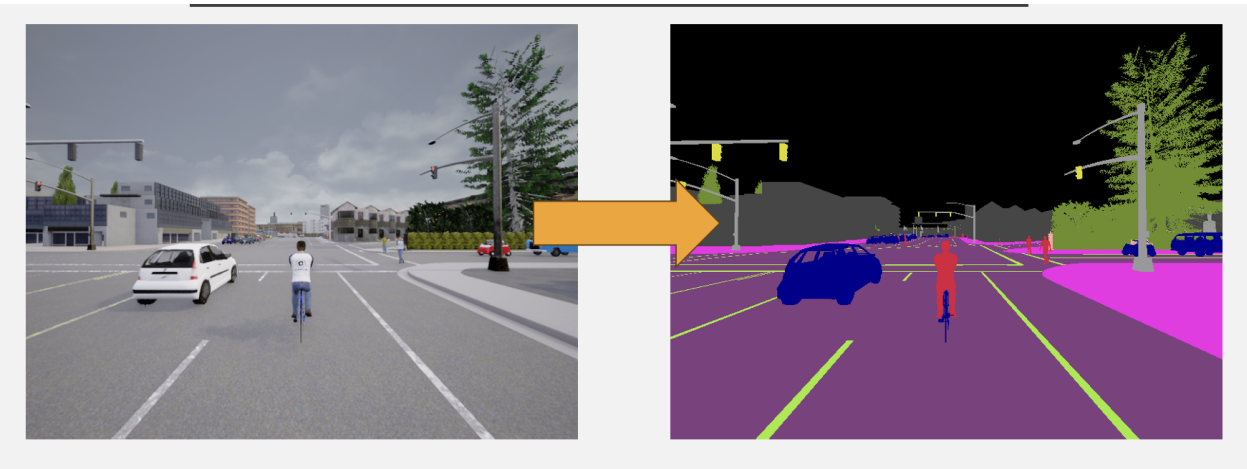

Figure 11: Labeling - DS2

The CNN trained on Dataset 2 shows moderate but the performance seems to be uneven across the classes, reflected by an overall accuracy of 78.5% across more than 32,000 test samples. With a significantly larger and more diverse set of classes compared to Dataset 1, the model demonstrates reasonable generalization but also highlights several challenging categories.

The higher performing classes include the Traffic Sign, Building, Fence, Vegetation, and Wall achieve precision and recall values above 0.85 for at least one metric, indicating that the model can reliably distinguish structurally distinct and consistently shaped objects. These patterns are confirmed in the confusion matrix, where these classes form strong diagonal clusters with minimal misclassification.

However, several classes show notably weaker performance. Road, Car, Pedestrian, and especially Road Line have reduced F1-scores, with Road Line dropping as low as 0.450. These categories tend to have high intra-class variability and visual ambiguity, making them more challenging for a simple CNN. For example, Road and Sidewalk share similar textures and colors, while Road Line is often thin, low-resolution, or partially occluded—factors that contribute to the spread of misclassifications visible in the confusion matrix.

The macro and weighted averages (F1 around 0.79–0.78) reflect this imbalance, showing that while many classes perform well, the model struggles with the more visually subtle or context-dependent categories.

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Traffic Sign | 0.941 | 0.905 | 0.922 | 2015 |
| Building | 0.896 | 0.920 | 0.908 | 3207 |
| Fence | 0.932 | 0.826 | 0.876 | 800 |
| Other | 0.760 | 0.864 | 0.809 | 2656 |
| Pedestrian | 0.718 | 0.772 | 0.744 | 3117 |
| Pole | 0.870 | 0.850 | 0.860 | 3161 |
| Road Line | 0.791 | 0.315 | 0.450 | 3207 |
| Road | 0.540 | 0.866 | 0.665 | 3206 |
| Sidewalk | 0.828 | 0.784 | 0.805 | 3207 |
| Vegetation | 0.921 | 0.858 | 0.888 | 2786 |
| Car | 0.728 | 0.789 | 0.757 | 3209 |
| Wall | 0.901 | 0.767 | 0.829 | 2356 |
| accuracy | | | 0.785 | 32927 |
| macro avg | 0.819 | 0.793 | 0.793 | 32927 |
| weighted avg | 0.804 | 0.785 | 0.780 | 32927 |

Figure 12: *
(a) CNN - DS2

```
Sample 00: true=Traffic Sign, pred=Traffic Sign
Sample 01: true=Car, pred=Road Line
Sample 02: true=Pedestrian, pred=Pedestrian
Sample 03: true=Other, pred=Other
Sample 04: true=Pedestrian, pred=Pedestrian
Sample 05: true=Road, pred=Road
Sample 06: true=Vegetation, pred=Vegetation
Sample 07: true=Building, pred=Building
Sample 08: true=Other, pred=Other
```

Figure 13: *
(b) CNN - DS2

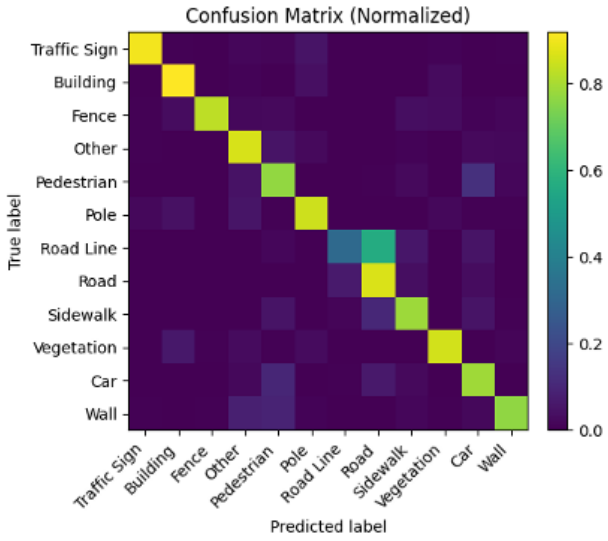


Figure 14: *
(c) CNN - DS2

Figure 15: CNN Results for Dataset 2

### 7.3.2 CNN/GNN Combo

The GNN/CNN hybrid model applied to Dataset 2 shows a noticeable drop in performance compared to the standard CNN. The overall accuracy falls to 64.9 percent, with macro and weighted F1-scores around 0.65 and 0.64, indicating that the model struggled with several classes across the dataset.

The class-level metrics highlight this variation. Based on what cna be observed in Figure 16, a few categories—such as Pedestrian, Fence, and Wall seem to maintain moderate performance with F1-scores above 0.70. This suggests that the GNN was able to extract useful relational information for objects with clear edges or distinct structural patterns. However, many other classes, including Road Line, Road, Car, and Pole, score much lower, largely because these categories are visually subtle, heavily context-dependent, or often overlap with neighboring objects in the scene.

The confusion matrix in Figure 17 reinforces this. Although diagonal elements are still visible—showing the model does identify correct labels for many samples—there is significantly more off-diagonal scattering compared to the CNN results. This means the model frequently misclassified objects that share visual similarity or spatial proximity. For example, Road and Sidewalk show high levels of confusion, as do Car and Road, and Pole with several neighboring classes.

Overall, these results suggest that the GNN/CNN combination introduced additional complexity without providing a strong boost in representation learning for this type of pixel-based image classification. Rather than improving accuracy, the added relational modeling made the task harder, possibly because the dataset's features are better captured through deep convolutional patterns than through graph-based structural reasoning.

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Traffic Sign | 0.599 | 0.889 | 0.716 | 2009 |
| Building | 0.746 | 0.710 | 0.728 | 3200 |
| Fence | 0.620 | 0.958 | 0.753 | 780 |
| Other | 0.701 | 0.791 | 0.743 | 2637 |
| Pedestrian | 0.840 | 0.512 | 0.637 | 3109 |
| Pole | 0.747 | 0.541 | 0.628 | 3157 |
| Road Line | 0.473 | 0.676 | 0.556 | 3200 |
| Road | 0.492 | 0.243 | 0.325 | 3200 |
| Sidewalk | 0.681 | 0.747 | 0.713 | 3200 |
| Vegetation | 0.667 | 0.737 | 0.700 | 2749 |
| Car | 0.579 | 0.664 | 0.619 | 3211 |
| Wall | 0.785 | 0.678 | 0.727 | 2359 |
| accuracy | | | 0.649 | 32811 |
| macro avg | 0.661 | 0.679 | 0.654 | 32811 |
| weighted avg | 0.662 | 0.649 | 0.640 | 32811 |

Figure 16: *
(a) GNN/CNN - DS2

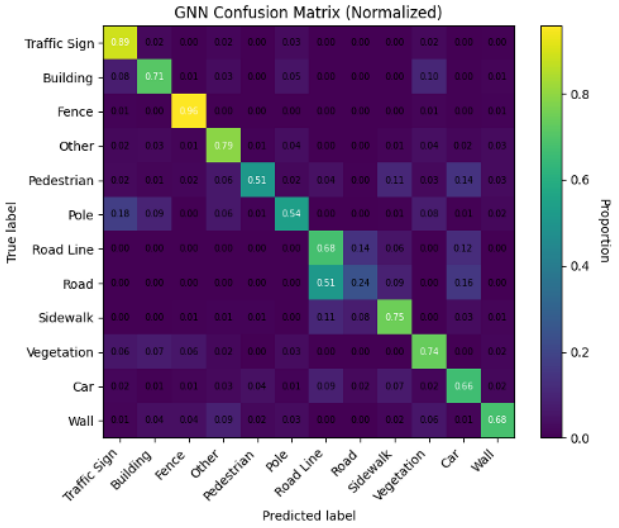


Figure 17: *
(b) GNN/CNN - DS2

Figure 18: GNN/CNN Results for DS 2

## 8. Applications, Use Cases and Future Works/Improvements

With the rise of many self-driving cars and systems, object detection systems grow increasingly important. This is also important for many safety systems on these vehicles. Being able to quickly and accurately detect and classify objects is important in making good timely decisions. This is a requirement in real-time systems where a failure to make these timely decisions can result in heavy costs such as damages and injuries.

If we had more time, we can attempt to connect our system with CARLA to act as part of it's self-driving, autopilot system. Using that, we can use it as a baseline for performance before eventually implementing it into an actual real-life system than can safely and accurately perform in a variety of situations and environments.

## 9. Challenges

There were a couple challenges that we have faced throughout the project. One of the major challenges involved the limitations of CARLA and the specific requirements to run it. We spent several hours trying to figure out which CARLA version matched with which Python version with multiple upgrades and downgrades to try and figure it out. Once, when we paused the data collection script, it refused to connect to CARLA again. We attempted reinstalling everything from scratch and doing everything in a conda environment, but none of it was working. We also did not have the time to label the data ourselves since the output images or frames lacked the labels. In the end, we switched to the Kaggle datasets to train our models.

A potential reason for the script not being able to connect to CARLA may have been due to one of the recent Windows 11 updates where developers have been finding issues with connecting to localhost. The script connects to the CARLA simulator through localhost on port 2000, but it is unable to. The other reason for the script not being able to connect is that the CARLA library's version is different from the CARLA simulator's version and it has very strict rules on versioning to connect.

Another important challenge/constraint that had to be taken into account was time. This was a consistent challenge throughout the project in relation to both CARLA and the machine learning models. With the issues regarding CARLA and data generation, time was an important factor for us to move to a Kaggle dataset instead. After two weeks of attempting to properly use CARLA to generate our dataset we found that any additional time spent without starting the models would be detrimental to our final product so instead we chose to pivot to utilizing our Kaggle datasets.

When it comes to our machine learning models time was also an important factor to consider. While dataset one didn't have too large of a dataset, dataset two did. While training our final models for dataset two training took approximately 7 hours. This meant that if there were something wrong in our code or our attempted optimizations didn't improve our model as expected we would need to wait another 7 hours to run a new version of our model. This meant that we needed to consistently work on the models throughout the semester to properly optimize our solution.

## 10. Project Management

| Team Member | Task |
|---|---|
| Matt | Develop, train, and optimize machine learning models |
| Kevin | Ran Carla and generate dataset |
| Ankith | Researched and found Carla Kaggle datasets |

Table 3: Tasks

## 11. Conclusion and Summary

This project aims to make part of a real-time system. Self-driving cars and systems are becoming a larger part of our lives and it is important that these systems are able to make executive decisions quickly and accurately. We

trained a few models such as a CNN and a CNN + GNN combo to help us identify various objects in an image or frame. While we did not get to applying these models to a system like the CARLA simulator, we got some decent results in object detection and classification. If we were to continue working on this, we hope to apply this project to some real-life systems.

## 12. References


[1] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. *CARLA: An Open Urban Driving Simulator*. Proceedings of the 1st Annual Conference on Robot Learning, pp. 1–16, 2017.

[2] R. Hamad, "What is LSTM? Introduction to long short-term memory," *Medium*, Dec. 3, 2023. [Online]. Available: https://medium.com/@rebeen.jaff/what-is-lstm-introduction-to-long-short-term-memory-66bd3855b9ce

[3] J. Noble, "What is a GNN (graph neural network)?," *IBM*, Feb. 19, 2025. [Online]. Available: https://www.ibm.com/think/topics/graph-neural-network

[4] S. Suraj, "Carla Object Detection Dataset," *Kaggle*, 2023. [Online]. Available: https://www.kaggle.com/datasets/suraj520/carla-object-detection-dataset

[5] A. Zorzetto, "CARLA Densely Annotated Driving Dataset," *Kaggle*, 2025. [Online]. Available: https://www.kaggle.com/datasets/albertozorzetto/carla-densely-annotated-driving-dataset