PROGRESS REPORT: AUTONOMOUS CAR USING CNN

Rudra Dey
Student# 1010124866
rudra.dey@mail.utoronto.ca

Pravin Kalaivannan
Student# 1010141295
pravin.kalaivannan@mail.utoronto.ca

Aadavan Vasudevan
Student# 1010101514
aadavan.vasudevan@mail.utoronto.ca

Abishan Baheerathan
Student# 1010218756
abishan.baheerathan@mail.utoronto.ca

ABSTRACT

This document presents our team's progress in developing an autonomous car. The project description contains minor changes that we made and explains why deep learning is necessary for this project. The team's responsibilities, work completed, and a future timeline are listed, which also provides solutions to redundancies. The data processing section discusses how the team collects data from the simulator to create a dataset and how that data is modified to use on the CNN model. This document compares a baseline model to our primary model and shows the quantitative results that we collected.
Total Pages: 9

1 PROJECT DESCRIPTION

This project involves training a simulated autonomous car using a convolutional neural network (CNN) and a forward-facing camera. The goal of this project is to build a model that can predict steering commands based on visual input from the camera. This should allow the car to drive on its own without the need for additional sensors.

Our team's motivation behind the project was to explore a simplified, small-scale version of autonomous driving systems, using deep learning. Furthermore, traditional rule-based approaches are limited by their inability to predict the vast number of driving scenarios that can occur. Deep learning offers the ability to learn from data and generalize across different visual conditions. This made it a suitable choice for handling the complex input from a moving car. The project also provided insight into the effectiveness of vision-only autonomous systems and the role of deep learning in embedded robotics.

2 INDIVIDUAL CONTRIBUTIONS AND RESPONSIBILITIES

Our team worked collaboratively throughout the project with clear task divisions and weekly milestones. All communication was handled through Discord, where we maintained an active group chat with frequent updates and responses. We held two scheduled meetings every week and used GitHub to track code progress, manage issues, and handle version control across all components. The team has a shared Google Drive where documents are added for collaboration.

2.1 INDIVIDUAL CONTRIBUTIONS

Each team member was assigned specific responsibilities:

- **Aadavan** handled the baseline rule-based model used for comparison, authored the ethical considerations section, and participated in model evaluation. Additionally, helped build the CNN architecture.

- **Rudra** led the software and simulator setup (camera system), collected the training dataset, and worked on preprocessing the image data. Helped with model training and tuning.

- **Pravin** contributed to background research on similar projects and helped build the CNN architecture. Additionally, helped create the dataset using CARLA along with a training regime.

- **Abishan** supported data preprocessing, designed the training setup (optimizer, loss function, hyperparameters), created the project's risk register, and contributed to model training.

2.2 Timeline of Completed Tasks

We maintained an internal task schedule which helped the team stay on track. Table 1 summarizes the tasks completed by each team member, their completion dates, and the outcomes of each task.

Task	**Team Members**	**Completion Date**	**Result**
Brainstorm Project Ideas	All Members	June 9, 2025	Team decided to work on a simulated autonomous car
Model Selection	All Members	June 10, 2025	Team decided on using a CNN model
Project Proposal (Draft & Final)	All Members	June 12–13, 2025	Completed a Project Proposal document
Data Collection & Setup	Rudra, Aadavan, Pravin	June 20, 2025	Team successfully set up CARLA simulator and collected data
Data Preprocessing	Rudra, Abishan	June 27, 2025	Resized and normalized the image
Baseline Model Implementation	Aadavan	June 30, 2025	Implemented ridge regression algorithm to create the baseline model
Architecture & Model Building	Pravin, Aadavan	July 4, 2025	Created a CNN architecture, to apply transfer learning
Training Regime Development	Abishan, Pravin	July 8, 2025	Selecting appropriate training parameters
Model Training & Tuning	Abishan, Rudra	July 10, 2025	Trained model through a training dataset, adjusted overfitting, and quantified results
Progress Report	All Members	July 11, 2025	Summarized work done so far, completed a progress report document

Table 1: Timeline of all completed tasks

2.3 Future Task Timeline and Responsibilities

Table 2 outlines the tasks to be completed in the next phase of the project.

Task	Team Member / Deadline	Description	Solutions to Redundancies
Model Evaluation & Testing	Aadavan, Pravin (July 25, 2025)	Test trained model with live input, identify weaknesses, confusion matrix.	The model should be tested under the same conditions to ensure consistency.
Final Report (Draft)	Everyone (Aug 1, 2025)	Write first full draft including abstract, results, ethics, background, etc.	If an assigned section has not been started 2 days before the deadline, it will be redistributed.
Final Report (Final)	Everyone (Aug 8, 2025)	Submit the polished report with citations and final results.	Everyone is required to edit their section and must read through the entire document once.
Record Presentation	Everyone (Aug 12, 2025)	Record team sections, align visuals with speech.	Everyone will record their assigned section and upload to the shared drive to ensure all clips are collected.
Edit Presentation Video	Rudra, Pravin (Aug 13, 2025)	Merge clips, finalize cuts, add transitions, captions, or diagrams.	Rudra will be the primary editor; Pravin will take over if necessary.
Final Deliverable	Rudra (Aug 15, 2025)	Submit final code, data, report, and video via the course portal.	Rudra will be responsible for submitting all final work and verifying that it is satisfactory.

Table 2: Future timeline with redundancies

Each task was documented, versioned, and reviewed by at least one other team member. This structure helped us manage project risks, distribute workload evenly, and ensure consistent progress through each phase.

3 Data Processing

Data was collected from the CARLA simulator, including images and corresponding steering values. We then normalized the images allowing us to train our CNN. Further, data augmentation was applied to increase the size and diversity of the dataset, leading to better generalization of the model.

3.1 Data Collection and Cleaning

In CARLA, some vehicles have autonomous functionality, which we used to collect data. We attached an RGB Camera Sensor to the car to record 10 frames per second (FPS) as the car travels around a designated map. In addition to the images, data for the steering values and turn signals of the car were collected allowing for use as ground truth labels for training the model. This data collection was done on different maps with different weather conditions to ensure a diverse dataset. We collected samples on three maps, in which the car collected data for 30 minutes, resulting in 54000 total images.

To clean the data, we removed any images that were all black, duplicates or not the correct resolution of 320x240 leaving us with 53457 images. The data is stored as numpy arrays, zipped and uploaded to a shared Google Colab file, ensuring access for all members.

Procedure:

1. Spawn in a vehicle
2. Attach an RGB Camera Sensor (front-facing) to the dash of the car.
3. Collect image data (10 fps) along with steering data and turn signals.
4. Clean the data by removing corrupted images and duplicates.
5. Save the data as numpy arrays for easy access and processing.

3.2 Data Preprocessing

All collected images were resized to a resolution of 160x120 pixels as shown in Figure 1, with the pixel values normalized to a range of [0, 1]. This was done to ensure consistency in the input data, allowing the model to learn effectively from the images.

Figure 1: Resized 3x160x120 example images from dataset

3.3 Data Distribution

We decided to do an 80/10/10 split for the training, validation and test sets respectively. This was done to ensure that we have a large set of data for training, while also having enough data for validation and testing. Table 3 shows the distribution of the data.

Dataset Split	**Total Images**
Training Set	42766
Validation Set	5345
Test Set	5346
Total Images	53457

Table 3: Dataset Distribution

3.4 Data Agumentation

Data augmentation was applied to the training set to increase the size and diversity of the dataset. We applied two techniques to generalize the model and force it to learn critical features.

- **Greyscale Conversion:** All training images were converted to greyscale reducing the the model's reliance on colour-based features, encouraging learning of more general visual features/road patterns. This will allow the model to improve, and work in different lighting conditions or if the camera is distorted.

- **Cutout Augmentation:** A rectangular region of each training image was masked out to simulate real-world situations (e.g. raindrop or debris on the camera lens). This encourages the model to learn relevant features from different parts of the image making it more robust overall.

These augmentations lead to the training set being increased to 128298 images, with each image having two copies with each augmentations applied respectively as shown in Figure 2.

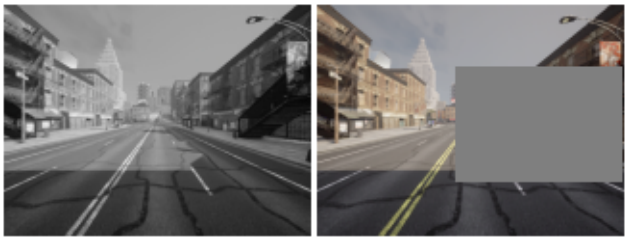

Figure 2: Augmented image examples (greyscale and cutout)

Dataset Split	Total Images
Training Set	42766 x 3 = 128298
Validation Set	5345
Test Set	5346
Total Images	138989

Table 4: Dataset Distribution after Training Set Augmentation

3.5 Plan for Testing on Unseen Data

In order to evaluate the model's performance, we will be testing it on unseen data which we reserved 10% of the dataset. Given how the data was collected sequentially across different maps and weather conditions, the test set will inherently contain frames that the model has not encountered before. This will provide the model with different roads and turns to encounter and we can see how well it generalizes to new situations.

Additionally, to further test the model's generalization, we can load the model in CARLA and make it drive around a new map where data was not collected from, and see how well the model performs in real-time. This will allow us to visualize the model's performance and see how well it can adapt to new environments.

3.6 Challenges Faced and Solutions

During the data collection process, we faced several challenges and overcame them with the following solutions:

- **Slow Initial Data Collection:** The simulator originally ran in real time, meaning that collecting data for 30 minitues would take that long in real time. To solve this issue, we sped up the simulation, allowing us to collect data at a faster rate and collect more data in a shorter time frame.
- **Turn Signal Data Collection:** Collecting turn signal data was initially problematic, because the vehicle's indicators were not being used by the automated vehicle in CARLA. We tried to resolve this through manual activation based on steering angle, but this proved to be inaccurate and inconsistent. We resolved this by enabling the CARLA Traffic Manager's automatic lights, allowing turn signals to activate based on the vehicle's navigation. In turn, this allows us to have accurate turn signal data for the training set.

4 Baseline Model

Using a Ridge Regression Algorithm we created a baseline model that predicts steering angles for a self-driving car from grayscale camera images and turn signal inputs. This baseline model serves as a simple, interpretable baseline such that we can compare our more complex primary neural network model later. To collect data, we used a simulator called CARLA, to obtain camera images, turn signal inputs, and our ground truth label for the steering angles.

4.1 Ridge Regression with Image Features

In the model outlined in Figure 3, the grayscale images (of shape 160x120) were flattened into 1D feature vectors and normalized. The features themselves represent the visual input of the car's front-facing camera. Additionally, left and right turning signals were captured as an additional feature; combining this with our flattened grayscale images we were left with a numpy array of shape $N \times 19201$, where N represents the number of images, and 19201 are the number of feature column vectors. The Ridge Regression algorithm was selected because it penalized large coefficients (L_2 norm), allowing for a generalized model. A series of models were trained with different regularization strengths (α), and the performance was evaluated using Mean Squared Error (MSE) and R^2 Score on a held-out 20% test set.

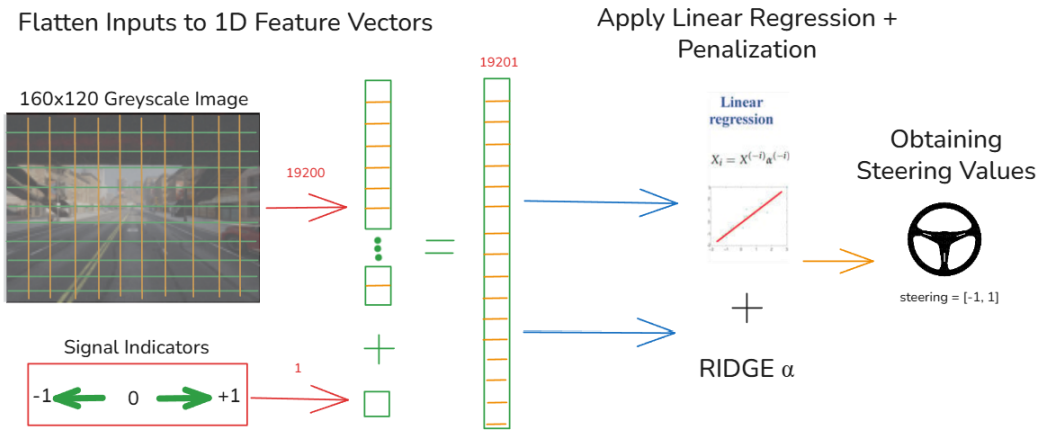

Figure 3: Basic Architecture of Steering Angle Ridge Regrsession Model

4.2 Minimal Tuning of α

The only hyperparameter needing to be tuned was the regularization strength for the Ridge Regression Algorithm. This was manually tuned, and thus no validation set was used. A small set of candidate values for α was chosen: $\alpha \epsilon \{1, 10, 100, 250, 1000, 8000\}$ With each value of α, the model was trained and tested with a dataset of 15 minutes of simulated driving. By obtaining the average MSE and the average R^2 score, our fourth model of $\alpha = 100$ came out to be the best.

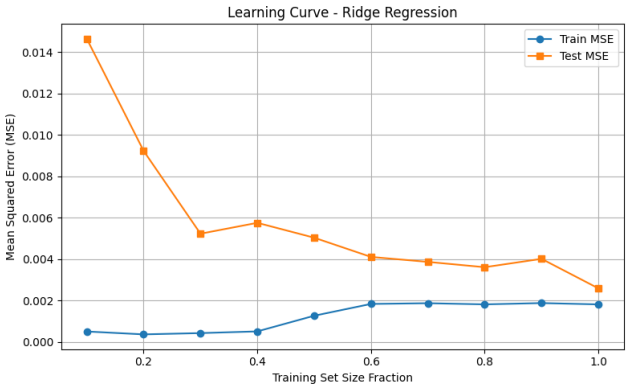

Figure 4: This is the learning curve for "Model 4" a Ridge Regression Model with $\alpha = 100$

4.3 Overfitting and Model Quality

As shown in Figure 4, the results suggest some overfitting due to the model's tendency to perform much better on training data than unseen data. This is reasonable since the training data frames, are all consecutive in nature, and the dataset used isn't extremely large either. In addition, as the training size increases the gap between Test and Train MSEs narrows, suggesting that with more data the model will generalize better. The final test MSE is (0.0025) which is quite low, meaning the model performs well overall after training on the data.

4.4 Qualitative Observations

When running the steering angle prediction model in the CARLA simulator, the model exhibited promising behaviour in terms of autonomous driving capabilities, however with large limitations. The car was able to drive freely, using the model's predictions from the front-facing grayscale camera images and turn signals provided by the user. In Figure 5 we can see the model taking a left turn in an intersection.

- **Turn Signal Response:** The car consistently turned in the correct direction when prompted, showing that the model incorporated turn signal input effectively.
- **Obstacle Avoidance:** The vehicle generally avoided obstacles but occasionally clipped walls, suggesting basic spatial awareness but limited precision.
- **Lane-Keeping:** The car had trouble staying within lane boundaries, often drifting, especially during turns or on complex roads.
- **Map Generalization:** Despite limitations, the car performed reasonably well across different map layouts, showing decent control over turns.

Figure 5: This is the demo for "Model 4" in CARLA

5 Primary Model

5.0.1 Transfer Learning with ResNet

We opted to use transfer learning to take advantage of powerful visual feature extraction from pretrained models, in this case, we used ResNet-18. This model was trained on ImageNet, and it has general-purpose feature extraction for RGB images. Rather than training a convolutional neural network from scratch (which would require large amounts of labelled data and time). To use ResNet-18 we had removed its final classification layer which allowed us to reuse the learned convolutional layers for predicting the steering angles from the images themselves. By freezing the pretrained weights the model benefits from previous feature extraction and uses it towards our dataset.

5.0.2 Architecture Design

The model architecture consists of two main components:

1. Feature Extractor (ResNet-18)
   - Input: RGB image resized to 224x224
   - Output: 512-dimensional feature vector
   - Final fully connected layer removed (cnn.fc = nn.Identity())
2. Prediction layer
   - The extracted 512-dimensional image features are concatenated with a scalar turn signal input (left=-1, straight=0, right=1), forming a 513-dimensional input vector.
   - This combined vector is passed through a small fully connected network:
     - Linear(513 $\rightarrow$ 128) + ReLU activation
     - Linear(128 $\rightarrow$ 1) for the final steering angle prediction

5.0.3 Training Methodology

The model was trained using the following methodology:

- Loss Function: Mean Squared Error (MSE) between predicted and ground truth steering angles.
- Optimizer: Adam optimizer with a learning rate of 1e-4, chosen for its robust adaptive learning capabilities.
- Batch Size: 32
- Epochs: 10
- Train/Validation Split: 80/20

5.1 Model Performance

Figure 6 shows the learning curve for the CNN model using ResNet18.

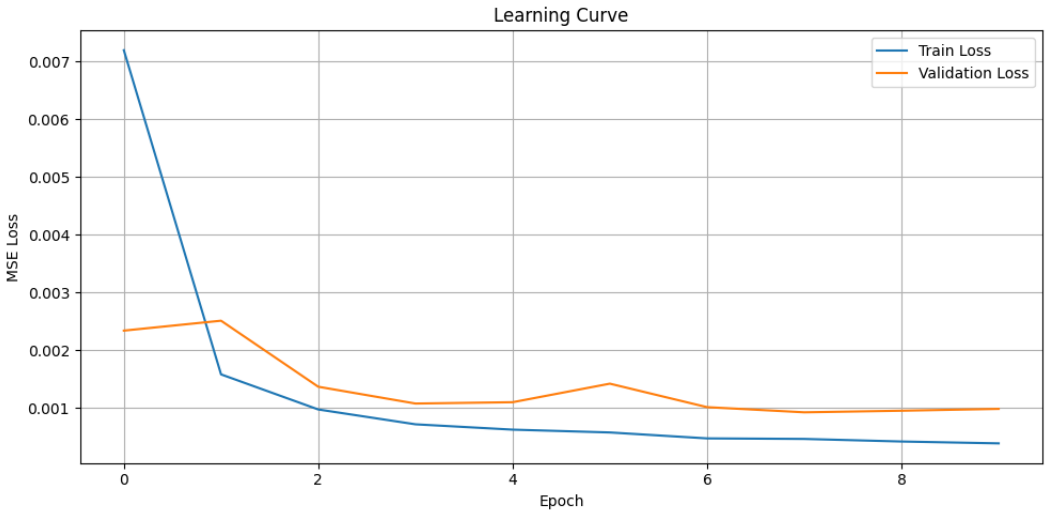

Figure 6: Learning Curve for CNN model using ResNet18 for 10 epochs

5.1.1 Quantitative Results

As shown in Figure 6, the CNN model shows a significant improvement over the baseline model. The training loss began at around 0.007 and decreased sharply to around 0.0015 by epoch 1, then steadily decreased to around 0.0005 by the final epoch. This indicates that the model is learning effectively from the training data. We also see that the validation loss started near 0.0024 and over the 10 epochs decreased to 0.001, showing good generalization with minimal overfitting. The small gaps between the training and validation loss after epoch 1, show that the model is performing well on both training and unseen data.

5.1.2 Qualitative Results

When running the CNN model in the CARLA simulator, the car exhibited significantly improved driving behaviour compared to the baseline model. The car was able to navigate complex turns and maintain lane discipline more effectively as seen in Figure 7. The model was able to make lane changes and stay within its respective lane. Similar to the baseline, the model struggles when it comes to large turns, but it does a better job staying away from walls and other obstacles.

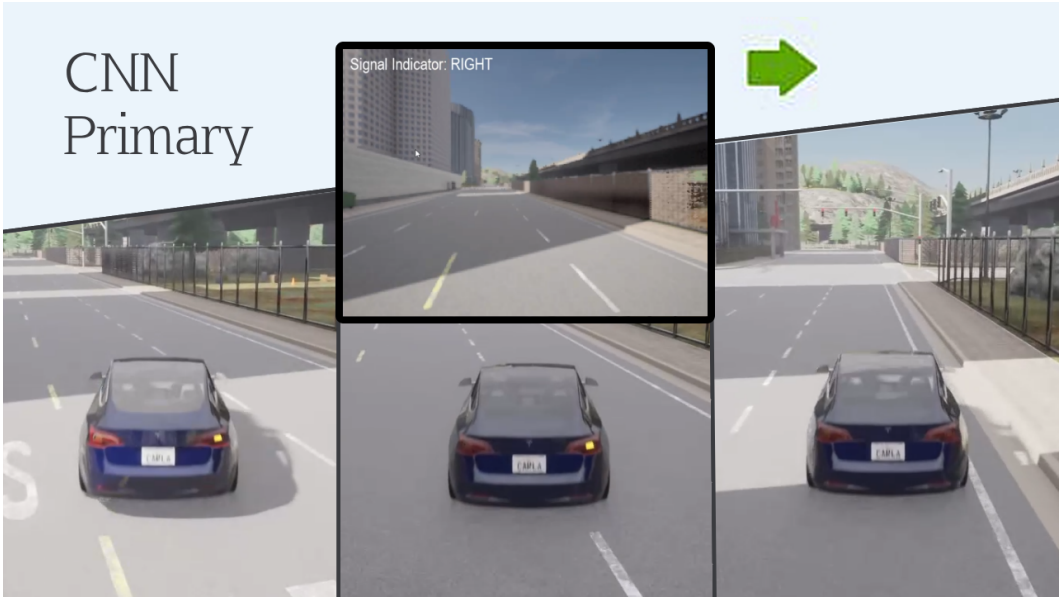

Figure 7: Primary Model Lane Switching

5.1.3 Challenges and Observations

One key challenge was in the integration of image and turn signal data in a single model. This required tensor dimension reshaping and matching in PyTorch correctly, specifically in concatenating CNN-extracted features with the scalar turn signal input. Mistakes here led to early runtime errors that had to be painstakingly debugged. There was also the problem of generalization tracking. As training loss kept improving, validation loss began to plateau from epoch 6 onward, indicating the possibility of overfitting. This indicates the need for early stopping or regularization in subsequent models.

One key finding is that the CNN model performs much better than the baseline model, especially on examples where there are lane markings. The CNN learned to recognize helpful spatial features from the driving environment, resulting in smoother and more accurate steering and improved lane-keeping behaviour at test time.

6 Conclusion

Our ResNet18 model outperformed the ridge regression baseline, keeping the car in its lane more reliably and cutting validation loss by a noticeable margin. To build on this, we'll introduce stronger data augmentation like MixUp or CutMix so the network learns to ignore odd artifacts and lighting changes. We also want to experiment with newer backbones such as EfficientNetV2 or Vision Transformers and try combining several models in an ensemble to cover each other's weaknesses. Finally, more hyperparameter tuning based on the dataset collected from CARLA should help the model pick up driving-specific features. These updates should lead to smoother steering on tight turns and better handling of roads the model hasn't seen before.