Comparison of End-to-End Imitation Learning Algorithms for Autonomous Driving in Urban Environments

Faizan Sana Arain
faizan@uwaterloo.ca
University of Waterloo
Waterloo, ON, Canada

Abstract

As deep neural networks and autonomous learning continue to advance, self driving cars are a promising area of research and are becoming a priority for the automotive sector. This paper presents the design of an end-to-end deep neural network architecture for self-driving vehicles to map raw pixels to control commands. In particular, it builds upon the PilotNet architecture which maps RGB images from a single frontal camera to steering angles. The PilotNet architecture, referred to as the modified PilotNet architecure, is extended to also output the throttle press and brake pressure in addition to the steering angle to allow for both lateral and longitudinal control of the vehicle. The simulation environment used in this case is the open source urban simulator CARLA. It was found that the PilotNet architecture performed well on the simulated environment with and was able to run for an average of 10.2 seconds as compared to the modified PilotNet architecture which would run for an average of 1.1 seconds before crashing. The code can be found at `https://github.com/faizansana/cs686-project`.

Introduction

Over the past several years, significant advancements in machine learning techniques, notably in Deep Neural Networks (DNNs), have made it possible to construct safety-critical systems, particularly Autonomous Vehicles (AVs). The idea of a day when we will no longer need to drive is quite alluring to people. These autonomous cars will enhance road safety in addition to improving the quality of life in general.

According to a report released by the World Health Organization (WHO), more than 1.35 million deaths occur due to road traffic accidents every year (World Health Organization 2018). The report also states that traffic accidents are the eighth most common cause of death amongst humans while being the leading cause of death for children and young adults between the ages of 5 and 29. A study conducted by the National Highway Traffic Safety Administration (NHTSA) in the United States demonstrated that 94% of traffic accidents occur due to human error while only a mere 2% are caused by vehicle failures (Singh 2018). Hence, the development of AVs could severely reduce the deaths due to road accidents since computing systems are not subject to the same fatigue humans experience. Additionally, AVs could also benefit in terms of reduced pollution (Nunno 2021), increased productivity and traffic flow (Atkins 2016), better fuel economy (US Energy Information Administration 2017) as well as car sharing (Ross 2014). In addition to these benefits, AVs also provide attraction to industry leaders due to the rising AV financial market share. According to (Singh and Mutreja 2022), the global AV industry generated \$76.12 billion in 2020 and is forecast to reach over \$2,161 billion by 2030, growing at a compound annual growth rate of 40.1% from 2021 to 2030. Therefore, significant research is being carried out by both industry as well as academia in order to further AV technology.

The Society of Automotive Engineers has defined the levels of autonomy into 6 levels ranging from 0 to 5 (SAE 2021) as seen in Figure 1. Current commercial implementations are limited to level 2 with several companies rolling out level 4 and level 5 AVs in test phase (California Department of Motor Vehicles 2021).

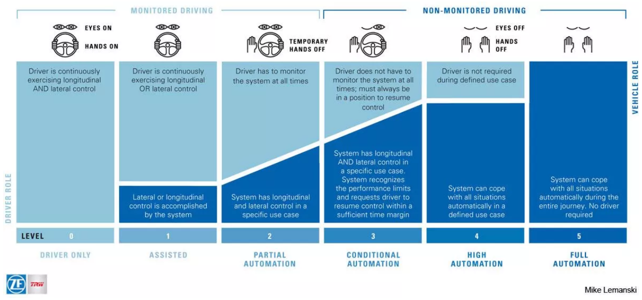

MONITORED DRIVING
NON-MONITORED DRIVING
EYES ON
HANDS ON
TEMPORARY HANDS OFF
EYES OFF
HANDS OFF
Driver is continuously exercising longitudinal AND lateral control
Driver is continuously exercising longitudinal OR lateral control
Driver has to monitor the system at all times
Driver does not have to monitor the system at all times; must always be in a position to resume control
Driver is not required during defined use case
Lateral or longitudinal control is accomplished by the system
System has longitudinal and lateral control in a specific use case
System has longitudinal AND lateral control in a specific use case. System recognizes the performance limits and requests driver to resume control within a sufficient time margin
System can cope with all situations automatically in a defined use case
System can cope with all situations automatically during the entire journey. No driver required
DRIVER ROLE
VEHICLE ROLE
LEVEL
0
1
2
3
4
5
DRIVER ONLY
ASSISTED
PARTIAL AUTOMATION
CONDITIONAL AUTOMATION
HIGH AUTOMATION
FULL AUTOMATION
Mike Lemanski

Figure 1: Levels of automation as defined in (SAE 2021)

There are two main approaches to the AV driving problem: modular and end-to-end pipelines (Figure 2)

Within the modular architecture, which evolved primarily from autonomous mobile robots, AV driving is broken down into five main components: perception, localization and mapping, prediction, planning and decision making, and vehicle control (Van Brummelen et al. 2018). In comparison, an end-to-end architecture, in the context of AV navigation, directly maps sensory data to control commands. The inputs for these architectures are from a high-dimensional fea-

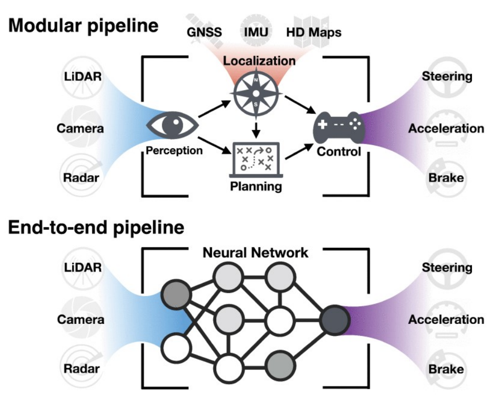


Figure 2: Modular and end-to-end pipelines. Figure courtesy of (Tampuu et al. 2022)

ture space such as RGB images, LiDAR point cloud etc. Although modular pipelines are the widely adopted approach amongst industry, some researchers believe that end-to-end systems can lead to better performance, less complexity as well as smaller systems (Haavaldsen, Aasboe, and Lindseth 2019) and have therefore been getting more attention lately. Current literature on end-to-end systems can be divided into four main categories: behavior cloning (or imitation learning (IL)), deep reinforcement learning (DRL), a combination of the two and using intermediate data representations.

This paper proposes to extend the PilotNet architecture (Bojarski et al. 2016), an end-to-end learning behavioral cloning (BC) or imitation learning (IL) model, in an urban environment on the open source simulator, Cars Learning to Act (CARLA) (Dosovitskiy et al. 2017). The PilotNet architecture will be enhanced to also output the longitudinal controls (throttle and brake pressure) alongside the lateral control (steering angles). In order to test and compare the performance of the algorithms, environment scenarios will be created in CARLA which will be used for both training and testing.

Related Work

In order to achieve end-to-end learning for AVs, BC and DRL are the most popular approaches. Using BC, the driving model is learned in a supervised learning fashion in order to replicate human drivers while DRL approaches use exploration to improve the driving policy from scratch. End-to-end approaches have demonstrated superhuman performance in Atari video games (Mnih et al. 2015), board games such as Go (Silver et al. 2016) and Chess (Silver et al. 2017) as well as competitive multiplayer games such as StarCraft (Vinyals et al. 2019) and Dota 2 (Berner et al. 2019).

IL is a supervised learning approach in which the model is trained to replicate human expert driving behavior. Based on the sensory information captured when the human driver was driving, the model is tuned to perform the same driving maneuvers. Although this works well for simpler tasks such as lane following, it fails to accommodate for rarely occurring, more complex driving scenarios (Bojarski et al. 2016; Pomerleau 1988).

The first implementation of IL for end-to-end navigation was the ALVINN model (Pomerleau 1988) in which a shallow fully connected neural network took RGB and radar images as input to output steering wheel angles. This was able to achieve lane following on public roads. Thereafter, DAVE (Muller et al. 2005) was able to learn obstacle avoidance in a cluttered environment using two cameras. The recent effort by NVIDIA (Bojarski et al. 2016), which improved upon DAVE, involved training a large scale Convolutional Neural Network (CNN), a subset of DNN, in order to steer a commercial vehicle in a variety of driving conditions. (Bojarski et al. 2016) used three front facing cameras with the left and right cameras for providing off-centre shifted perspectives that are utilized to rectify and drift that the car may have. The network is trained to minimise the Mean Squared Error (MSE) between the models' steering command and the command from either the human driver or shifted viewpoint and used the number of human interventions required as an evaluation metric. In order to increase the explainability of the model, (Bojarski et al. 2017) studied the algorithm in (Bojarski et al. 2016) to identify the regions of the input images that are most salient to the network for predicting the steering angles. Hence, (Cultrera et al. 2020) trained a model that receives an input image alongside a high-level command to predict the steering angles. A separate predictive section of the network is used for the four high-level commands $\{$*follow the lane, go straight, turn left, turn right*$\}$, which consists of an attention layer and a Fully Connected Network (FCN) to predict the steering commands. (Hecker, Dai, and Van Gool 2018) proposed that AVs need to have access to more information and therefore use four cameras, with each camera feeding to a network consisting of multiple CNN and Long Short-Term Memory (LSTM) sub-networks. A FCN then fuses the information from all the cameras and the route planner to provide steering angle as well as longitudinal speed. This was seen to outperform (Bojarski et al. 2016). (Codevilla et al. 2018) propose an extension of imitation learning, Conditional Imitation Learning (CIL) in which the network is fed the agent's intention alongside the input observation. This allows for the network to be used for different tasks rather than for one specific subset such as lane following. (Hawke et al. 2020) use CIL to train a network capable of navigating complex urban environments.

In RL, the model learns to maximize the rewards by interacting with the environment which is formulated as a Markov Decision Process (MDP) (Sutton and Barto 2018). Since IL learns from expert behavior, it usually suffers from insufficient exposure to various driving scenarios. On the other hand RL is more immune to this issue. (Riedmiller, Montemerlo, and Dahlkamp 2007) used RL in order to learn steering on a real car just using five variables which describe the state of the car. Deep learning in RL is used to approximate the parameters of the network. For instance, in Q-learning, the Q-function $Q^{\pi}(s, a)$ also known as the state-action value function of a policy π, needs to be approximated since it is generally impractical to represent this Q-function in memory for every combination of s and a. Hence a neural network is used as a function approximator $Q^{\pi}(s,a;\theta) \approx Q^{\pi}(s,a)$ where θ are the parameters of the network. Deep Q-Learning has been used in order to develop an end-to-end model that is based on vision (Müller et al. 2018). (Liang et al. 2018) used Deep Deterministic Policy Gradients (DDPG) (Lillicrap et al. 2015) in order to develop a vision-based algorithm, demonstrating its performance in CARLA. Since an RL policy needs to interact with the environment in order to learn, the learning phase is usually done in simulation tools such as CARLA as well as the GTA V computer game (Zhou, Krähenbühl, and Koltun 2019). (Dosovitskiy et al. 2017) developed and evaluated an asynchronous advantage actor-critic (A3C) DRL method on CARLA. However, they demonstrated that the classical modular pipeline and IL models outperformed the RL based method.

Methodology

In order to compare end-to-end learning methodologies, the PilotNet architecture (Bojarski et al. 2016) will be the baseline that will be implemented. Since the PilotNet architecture only outputs the steering angle, the architecture will be updated to also incorporate throttle as well as brake press. In addition, since the original results were only concerned with relatively straight sections of roads, this paper also studies the effectiveness of the algorithm in urban city environments with traffic lights. The code for this paper is available at `https://github.com/faizansana/cs686-project`.

Algorithms

The algorithms to be employed are as follows.

1. PilotNet Architecture

   The architecture of the neural network can be seen in Figure 3.

   The network has around 27 million collections and 250 thousand parameters. During training done in (Bojarski et al. 2016), three front facing cameras are used and a random shift as well as rotation is employed in order to make the training more robust. The inputs during training are the three frontal cameras as well as the steering angle as seen in Figure 4. However, in this case, the training only involved the input from a single frontal camera.

   The images are fed into the network, which then computes a steering command which is compared to the desired command for the specific image. Based on the error between the predicted and desired steering angles, the weights of the network are updated using backpropagation.

2. Modifed PilotNet Architecture

   The architecture of this network is based off the PilotNet architecture (see Figure 3.

   The following were the main changes made in this algorithm:

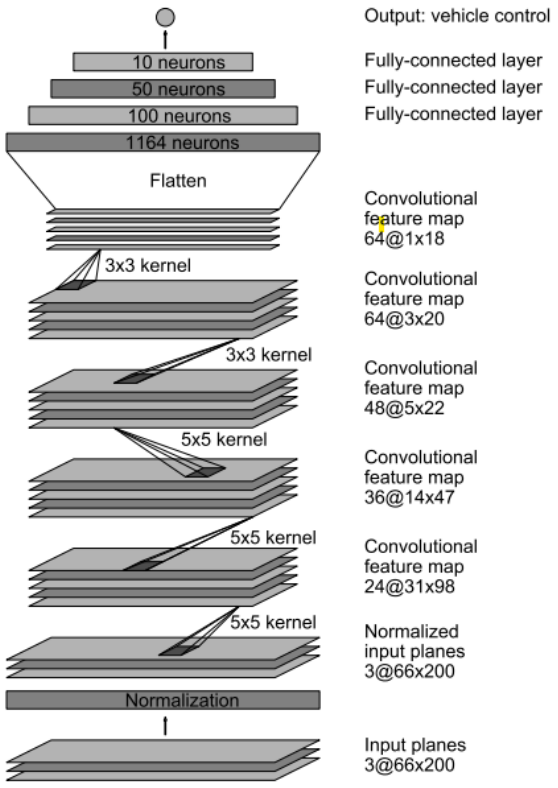


Figure 3: Neural Network architecture of PilotNet. Figure courtesy of (Bojarski et al. 2016).

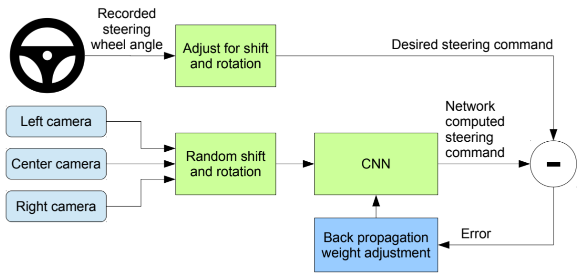

Recorded steering wheel angle
Adjust for shift and rotation
Desired steering command
Left camera
Center camera
Right camera
Random shift and rotation
CNN
Network computed steering command
-
Error
Back propagation weight adjustment

Figure 4: Training procedure employed in (Bojarski et al. 2016) for training PilotNet. Figure courtesy of (Bojarski et al. 2016).

(a) Changes in output layer

As compared to the original architecture shown in Figure 3, the output layer consisted of three neurons representing the steering angle, throttle press and brake pressure. Two different methodologies were experimented with for the output layer.

i. By using a single neuron to represent each of the steering angles, throttle press and brake pressure. The activation function used for these was the linear activation i.e. $f(x) = x$ where f represents the activation function and x represents the aggregation at the neuron.

ii. In this case, in addition to adding the single neuron, another neuron was connected to the output which would multiply the input of the previous neuron with $\arctan$. Hence this was a custom Keras lambda layer with the function $f(x) = \arctan(x)$. This was used since the original paper (Bojarski et al. 2016) mentioned the output representing the radius and using the $\arctan$, the angle can be found.

(b) Dropouts in fully-connected layers

Since overfitting is an issue when training deep neural networks, dropping off nodes during the forward propagation can improve generalization (Srivastava et al. 2014). Hence, a dropout layer with a probability of 10% was used after each fully-connected layer.

Simulation Environment

In order to compare the three algorithms, it is essential that an identical environment is setup for all of them. One of the issues of AV algorithms is that, in general, most researchers construct their own simulation environments which makes it arduous to compare the results. Since these algorithms are based on direct image input, although feature normalization is performed to allow for the algorithm to adapt to a variety of situations, slight distortions in images can make significant differences. Although there are several simulation tools available, the open-source simulator CARLA (Dosovitskiy et al. 2017) is used in this case since it is one of the more popular ones with companies such as NVIDIA supporting the initiative. Since it is also one of the first to be released, it also has good documentation and community support. Furthermore, they have also built an official API in Python with documentation available at `https://carla.readthedocs.io/en/latest/` which is particularly important since the machine learning library, Tensorflow, was to be used in Python.

Although it was decided to use the INTERACTION dataset (Zhan et al. 2019) and incorporate it within the CARLA environment, this was not possible since the dataset was not publicly available and was not received in time from the authors in order to perform tests.

The simulation environment used in this case was Town 10 of the CARLA simulation tool which contains buildings, sidewalks, 4-way intersections, T-intersections as well as traffic lights. A top view of the map is shown in Figure 5.

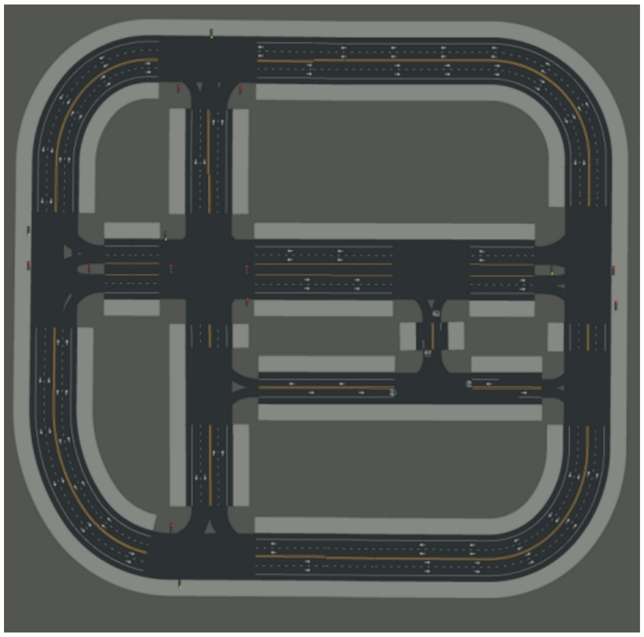


Figure 5: Top View of Town 10 of the CARLA Simulation tool

Generating Data

Since the goal was to run the algorithms to run on live CARLA simulation in Town 10, a subject vehicle was created to which an RGB camera was attached to provide the frontal view. For each collection of data, a random vehicle from the 39 available in CARLA were chosen. The field of view (FOV) of the camera was set to 110° while all other camera parameters were set to default and can be seen in Table 1. The choice of FOV was in order to only see the front of the vehicle based on (Bojarski et al. 2016) while the image size (640×480) was chosen to reduce dataset storage size. The simulation was set to synchronous mode with the Frames per Second (FPS) set to 20 to allow for vehicle control updates to be deterministic. In order to collect data, the vehicle was driven in autopilot mode by the CARLA simulation tool and all the traffic lights were initally set to green to avoid stopping of the vehicle at any intersection. No other road participants such as other vehicles or pedestrians were setup.

Table 1: Camera attributes for the vehicle attached camera. All other values were kept as default.

Attribute	**Value**
Field of view	110
Image Width	640
Image Height	480

In order to store the data, each frame contains the time in seconds, steering angle, throttle and brake pressure. It is important to note that the range of steering angle, throttle and brake pressure in CARLA is $[0, 1]$. An example of three images collected as part of this dataset is shown in Figure 6.

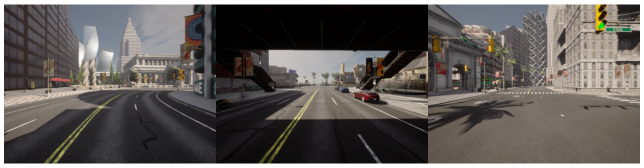


Figure 6: Example of the dataset of three images collected in the to form the dataset. As seen above, it also takes into consideration shadows from buildings, signs and other similar artifacts and also contains a tunnel (center image).

Using the above, 1.5 hours of dataset with 19,332 images and a size of 9 GB were generated and used for training the algorithms. All experiments were run on CARLA version 0.9.13, the latest version at the time of writing.

Training of Algorithms

In order to train the algorithms, the data was split into a training set, validation set and testing set. 60% used for training, 20% used for validation while 20% was used for testing. The data was not shuffled while splitting. The hyperparameters used for training are shown in Table 2.

Table 2: Hyperparameters used to train the model. Most of these values were used from literature to stay as close as possible to the results obtained in (Bojarski et al. 2016)

Hyperparameter	Value
Optimizer	Adam (Kingma and Ba 2014)
Epochs	1000
Steps per epoch	10
Batch size	64
Validation steps	10
Steering angle loss	Mean Squared Error (MSE)
Throttle loss	MSE
Brake pressure loss	MSE
Accuracy metric	Mean Absolute Error (MAE)
Learning Rate	$1e^{-4}$

Although 1000 epochs were initially used, this was later adjusted and is discussed in the results section in detail. The throttle loss and brake pressure loss were only used for the modified PilotNet architecture.

Results

This section describes the data generation process, model training and testing as well as model execution within the simulation. The machine used for all aspects of this project had the following specifications (see Table 3).

Data generation

Figure 7 shows the data generation pipeline in details. After running the CARLA server with Town 10 loaded, the environment was first set into synchronous mode. The vehicle was then spawned on a possible spawn point on the map and the camera parameters were setup and attached to the front of the vehicle. The data was then stored on the file system of the machine.

Table 3: Specifications of the machine used to perform the experiments

Specification	Value
CPU	AMD 2950X 16-core processor
GPU	NVIDIA GeForce RTX 2080 Ti (11GB VRAM)
RAM	64 GB

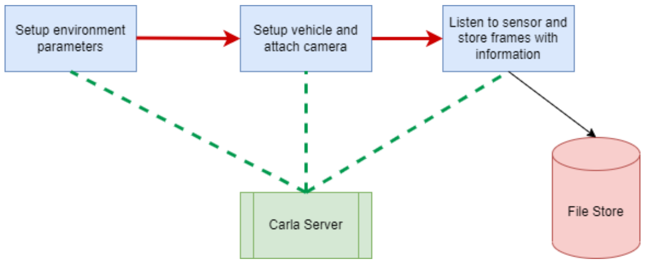

Setup environment parameters
Setup vehicle and attach camera
Listen to sensor and store frames with information
Carla Server
File Store

Figure 7: Data generation pipeline

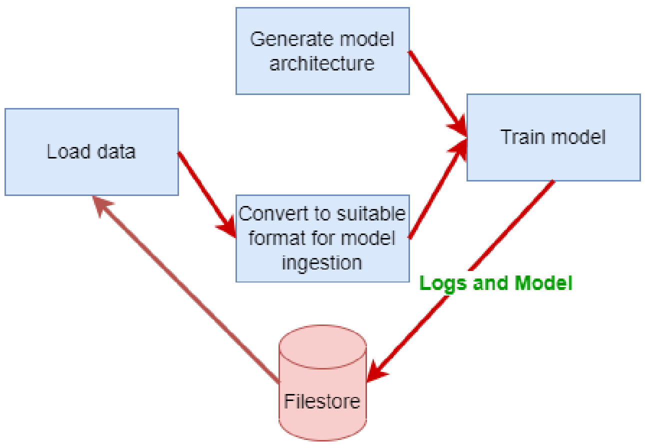
All communication with the CARLA server was done using the official Python API. A significant challenge of data generation would occur when the subject vehicle would get into accidents when CARLA was controlling the vehicle (autopilot was set to true). This would cause the CARLA server to automatically destroy the actor (i.e. the vehicle) and would cause the data generation to file before the time limit was reached. The crashes were also frequent and would occur every minute. This was due to the asynchronous mode wherein the PythonAPI and the server would operate on different clocks and therefore the controller updates, provided by the PythonAPI, were sent at incorrect rates. Hence, the API and server were synced at a rate of 20 FPS corresponding to an update every 5 ms. In addition, since crashes still sometimes occured in synchronous mode, a new vehicle was spawned every time a crash would occur and the simulation would then continue to record the image frames.

Model Training

Figure 8 shows the training pipeline of both networks. The data is first loaded from the file system, converted into the format needed for model ingestion (numpy arrays) and reshaped. For both architectures, the input image shape was $200 \times 66 \times 3$. The outputs were steering angle for the case of PilotNet and steering angle, throttle and brake pressure for the modified PilotNet architecture. The hyparameters used for training are shown in Table 2. Since the images were loaded from file system and then had to be resized into the necessary input shape, the data processing was the bottleneck of the entire pipeline.

Figures 9 and 10 show the training and validation losses for the model architectures on the dataset.

The results of evaluating both models on the test set is shown in Table 4



Generate model architecture
Load data
Train model
Convert to suitable format for model ingestion
Logs and Model
Filestore

Figure 8: Model training pipeline

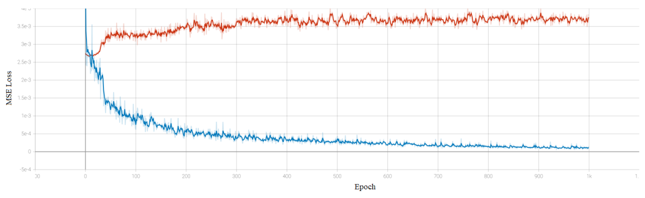


Figure 9: Loss vs Epoch for the PilotNet architecture. The orange represents the validation set while the blue represents the training set.

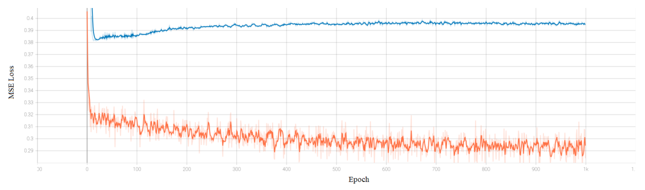


Figure 10: Loss vs Epoch for the modified PilotNet architecture. This loss represents the addition of the steering loss, throttle loss and brake pressure loss. The blue line represents the validation data while the orange represents the training data.

Table 4: Results of evaluating both PilotNet and Modified PilotNet on the test dataset.

Metric	PilotNet	Modified PilotNet
Overall Loss	$4.1324e^{-3}$	0.4120
Steering Angle Loss	$4.1324e^{-3}$	0.1372
Throttle Loss	-	0.1371
Brake Pressure Loss	-	0.1377
Steering Angle MAE	0.04232	0.3032
Throttle MAE	-	0.3017
Brake Pressure MAE	-	0.3023

Since 1000 epochs took significant time and did not seem to have much affect on the validation losses, it was decided to reduce it to 100 epochs. Additionally, since the modified PilotNet architecture still had a comparatively higher loss, it was decided to do the following.

1. Scheduled updating of the learning rate

   Since the validation loss was the metric that was trying to be improved and as seen in both Figures 9 and 10, this was relatively constant. In addition, the training loss also seemed to converge which suggested that the learning rate may be too high for latter epochs. Hence, an exponentially decaying learning rate was used with a decay rate of 0.9 and decay step of 10,000 (see Figure 12) . The learning rate for the 100 epochs can be seen in Figure 11.

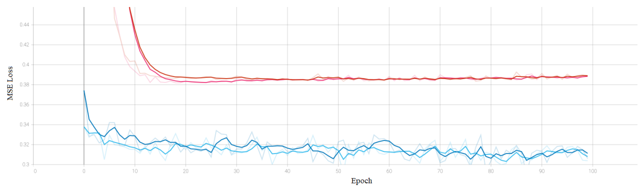


Figure 11: Comparison of epoch loss with a constant learning rate of $1e^{-4}$ as compared to a scheduled learning rate. The dark blue and red represent the loss with a scheduled decay learning rate (training and validation set) while the light blue and pink represent the loss with a constant learning rate of $1e^{-4}$

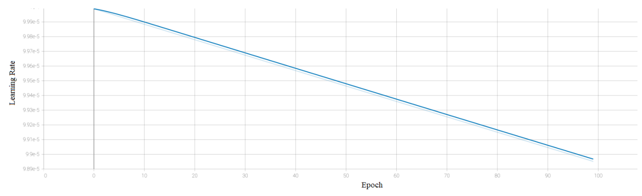


Figure 12: Scheduled learning rate used in Figure 11

2. Addition of another layer

   Initially, as discussed in the methodology section, an $\arctan$ layer was connected to each output neuron. Since this did not have any affect on the loss, this was discarded. Thereafter, a fully connected layer of 1000 neurons with the activation function of ReLU was attached right after the flatten layer. This was to increase the trainable parameters (i.e. the weights) in the network which would introduce more non-linearity within the model. The total trainable parameters of this model was 1,390,047 as compared to 252,247 in the modified PilotNet architecture The training was run for 100 epochs and gave a loss of 0.3925 which was not significantly better than the one seen in Table 4. Hence, this was not investigated further.

All of the graphs were obtained by logging to Tensorboard. In addition, for each run of the model, the one with the lowest validation loss was stored to be used for testing on live simulation rather than using the model from the last epoch.

Model Test on Live Simulation

The best performing models in terms of lowest validation loss for both the modified PilotNet architecture as well as the original PilotNet architecture were tested on live simulation wherein the model predicted in real time and got feedback from simulation. Since no manual control was provided as in (Bojarski et al. 2016), no autonomy metric (see Equation 1) was obtained.

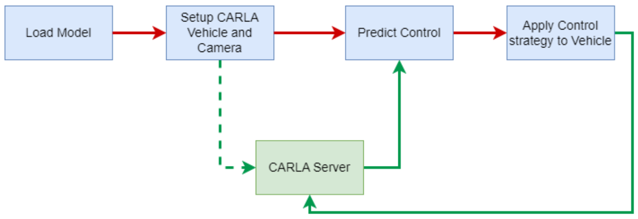

Load Model
Setup CARLA Vehicle and Camera
Predict Control
Apply Control strategy to Vehicle
CARLA Server

Figure 13: Pipeline for real time predictions on the simulation environment

Table 5 shows the collision statistics. This was done for 50 runs, with a random vehicle spawned each time and as soon as a collision would occur, the episode would be terminated and the simulation restarted.

Table 5: Time before a collision would happen. Averaged over 50 runs.

	Time (seconds)
Modified PilotNet Architecture	1.1
PilotNet Architecture	10.2

Discussion

This section analyzes the results that were displayed in the previous section and explains in detail the outcomes, possible reasonings for them as well as the shortcomings.

PilotNet Architecture

As seen in Figure 9, the model converges during training to a training loss of less than $1e^{-4}$ and a validation loss less than $4e^{-3}$. This was expected since the results in (Bojarski et al. 2016) showed that their model performed with an autonomy value of 98% which is defined by the following Equation 1. Since these are regression problems, we used the MSE as the loss and the MAE as the evaluation metrics.

$$autonomy = (1 - \frac{(\text{number of interventions.6 seconds}}{\text{elapsed time in seconds}}).100 \quad (1)$$

For the test set, the model had a loss of $4.1324e^{-4}$ and a MAE of 0.04232. Since the original paper (Bojarski et al. 2016) did not release these results, direct comparison was not possible. However, these values seemed to be satisfactory wherein the model converged and therefore no further optimization in the learning parameters had to be performed.

Modified PilotNet Architecture

As seen in Figure 10, the model converged to a training loss of approximately 0.3 and a validation loss of 0.40. These losses are the combined MSE losses of the steering angle, throttle and brake pressure. All three were also set to use the MSE loss since this was a regression problem. Since the range of the steering angle, throttle and brake pressure are $[0, 1]$, these values did not have to be normalized.

As seen in Table 4, the losses of all three values were relatively similar. However, the steering angle loss was 33 times higher than the PilotNet architecture. Although the loss value was expected to be higher since this was a multi-objective optimization (optimizing for steering angle, throttle loss and brake pressure), the difference was found to be extremely large. Hence it was proposed to investigate the learning hyperparameters as well as modifying the neural network architecture further. The goal was to reach a steering angle loss similar to that of the original PilotNet architecture or within 5 times the value i.e. less than 0.021 MSE.

1. Scheduled updating of the learning rate

   The learning rate was dynamically changed as the number of epochs increased. The learning rate can be seen in Figure 12. As seen in Figure 11, the losses of both trainings, with a constant learning rate of $1e^{-4}$ and the scheduled decayed learning rate, follow very closely to one another. Hence it can be concluded that the decayed learning rate did not improve learning performance.

2. Addition of another layer

   In this case the total number of trainable parameters were increased by over 450%. Although the test set loss did decrease by approximately 4.7%, this decrease was not significant and therefore this was not investigated further.

Since other studies (Dosovitskiy et al. 2017) consider other neural network architectures for the prediction of the steering angle, throttle and brake pressure, it was not used for comparison. No other literature was found which would build upon the PilotNet architecture to predict all three values. Further investigation into modifying the network architecture needs to be done in order to improve the accuracy metrics. Similar to CIL as seen in (Codevilla et al. 2018) a larger fully connected layer could be connected before the output neuron to allow for the preceding layer to optimize for a single objective while the rest of the network optimizes for all three objectives. However, since autonomous driving is a real time task, it is essential that the DNN be able to predict the outputs within a specified time frame which is particularly challenging with deeper neural networks as well as the limited hardware available on board a vehicle.

Testing on Live Simulation

Since there were no metrics developed for this case due to the lack of manual control, this was evaluated based on the collision statistics (Table 5) as well as the visuals of the simulation. It was seen that the trained PilotNet architecture performed relatively well as compared to the modified PilotNet architecture (10.1 seconds compared to 1.1 seconds). This was expected since the PilotNet architecture only outputs the steering angle while the modified PilotNet architecture adapts both lateral (steering angle) as well as longitudinal controls (throttle and brake). However, it was expected that the PilotNet architecture would last for longer runs since (Bojarski et al. 2016) showed a 98% autonomy on a physical road. This was likely due to the fact that the dataset that was used for training only consisted of 11,600 images and only for the situations of two vehicles while the live simulation tests were tested on all 39 vehicles. This likely did not cover the entirety of the town and more data should have been generated and augmented for increasing the training dataset size.

Limitations

Although this study proposed and successfully demonstrated the accuracy of the prediction of steering angles in an end-to-end fashion, the following were some of the shortcomings:

- Other than the subject vehicle, there were no other vehicles that were driving in the environment at the same time. However, there were stationary vehicles that were parked on the sides of the road.
- There were no pedestrians, cyclists or other road users in the environment.
- The algorithms were only trained in clear, sunny weather and were not tested in various weather conditions such as rainy, cloudy etc.
- The algorithm was only tested in the same simulation town and were not tested in towns with varying building structures.
- All traffic lights were set to be green.

Conclusion

This research analyzed an end-to-end BC approach for self driving vehicles and extended the popular, PilotNet architecture to adapt not only lateral controls but also longitudinal controls of the vehicle for urban environments. Although level 3 AVs (see Figure 1) such as the Tesla 3 model are commercially available, level 4 and above is still a challenge particularly within the urban environments due to their complex nature . The modified PilotNet architecture takes RGB image input from a frontal camera and controls the steering angle, throttle as well as the brake pressure. The algorithm was implemented using TensorFlow2. A dataset was generated using the CARLA simulator which was then used to train both the PilotNet as well as modified PilotNet architectures. The networks were then tested in Town 10 of CARLA. It was found that the original PilotNet architecture performed relatively well while the modified PilotNet architecture struggled to adapt both longitudinal and lateral controls.

In order to further improve the accuracy of the modified PilotNet architecture, it is proposed to enhance the training dataset by generating (a) a larger dataset (b) from a variety of different vehicles (c) using a similar training pipeline as in (Bojarski et al. 2016) wherein three cameras were used for the training phase. Additionally, increasing the size of the network by adding a hidden layer for each of the output neurons as seen in (Codevilla et al. 2018) should also be investigated further. Although hyperparameter analysis was done for the learning rate, a detailed hyperparameter analysis for other aspects such as optimizer, batch size and so on needs to be conducted in order to improve training results. Furthermore, after the training of the modified PilotNet architecture is conducted on the current environment, it should be tested on a more dynamic environment with other live road users (such as pedestrians and vehicles).

References

Atkins, W. 2016. Research on the impacts of connected and autonomous vehicles (cavs) on traffic flow. Technical report, Department for Transport London, UK.

Berner, C.; Brockman, G.; Chan, B.; Cheung, V.; Debiak, P.; Dennison, C.; Farhi, D.; Fischer, Q.; Hashme, S.; Hesse, C.; et al. 2019. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*.

Bojarski, M.; Del Testa, D.; Dworakowski, D.; Firner, B.; Flepp, B.; Goyal, P.; Jackel, L. D.; Monfort, M.; Muller, U.; Zhang, J.; et al. 2016. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*.

Bojarski, M.; Yeres, P.; Choromanska, A.; Choromanski, K.; Firner, B.; Jackel, L.; and Muller, U. 2017. Explaining how a deep neural network trained with end-to-end learning steers a car. *arXiv preprint arXiv:1704.07911*.

California Department of Motor Vehicles. 2021. Autonomous Vehicle Testing Permit Holders.

Codevilla, F.; Müller, M.; López, A.; Koltun, V.; and Dosovitskiy, A. 2018. End-to-end driving via conditional imitation learning. In *International Conference on Robotics and Automation (ICRA)*.

Cultrera, L.; Seidenari, L.; Becattini, F.; Pala, P.; and Del Bimbo, A. 2020. Explaining autonomous driving by learning end-to-end visual attention. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 340–341.

Dosovitskiy, A.; Ros, G.; Codevilla, F.; Lopez, A.; and Koltun, V. 2017. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, 1–16.

Haavaldsen, H.; Aasboe, M.; and Lindseth, F. 2019. Autonomous vehicle control: End-to-end learning in simulated urban environments. In *Symposium of the Norwegian AI Society*, 40–51. Springer.

Hawke, J.; Shen, R.; Gurau, C.; Sharma, S.; Reda, D.; Nikolov, N.; Mazur, P.; Micklethwaite, S.; Griffiths, N.; Shah, A.; et al. 2020. Urban driving with conditional imitation learning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 251–257. IEEE.

Hecker, S.; Dai, D.; and Van Gool, L. 2018. End-to-end learning of driving models with surround-view cameras and route planners. In *Proceedings of the european conference on computer vision (eccv)*, 435–453.

Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Liang, X.; Wang, T.; Yang, L.; and Xing, E. 2018. Cirl: Controllable imitative reinforcement learning for vision-based self-driving. In *Proceedings of the European conference on computer vision (ECCV)*, 584–599.

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *nature* 518(7540):529–533.

Muller, U.; Ben, J.; Cosatto, E.; Flepp, B.; and Cun, Y. 2005. Off-road obstacle avoidance through end-to-end learning. *Advances in neural information processing systems* 18.

Müller, M.; Casser, V.; Lahoud, J.; Smith, N.; and Ghanem, B. 2018. Sim4cv: A photo-realistic simulator for computer vision applications. *International Journal of Computer Vision* 126(9):902–919.

Nunno, R. 2021. Autonomous vehicles: State of the technology and potential role as a climate solution. Technical report, Environmental and Energy Study Institute.

Pomerleau, D. A. 1988. Alvinn: An autonomous land vehicle in a neural network. *Advances in neural information processing systems* 1.

Riedmiller, M.; Montemerlo, M.; and Dahlkamp, H. 2007. Learning to drive a real car in 20 minutes. In *2007 Frontiers in the Convergence of Bioscience and Information Technologies*, 645–650. IEEE.

Ross, P. E. 2014. Robot, you can drive my car. *IEEE Spectrum* 51(6):60–90.

SAE. 2021. Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles. Technical report, Society of Automotive Engineers.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of go with deep neural networks and tree search. *nature* 529(7587):484–489.

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; et al. 2017. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*.

Singh, A., and Mutreja, S. 2022. Autonomous vehicle market by level of automation (level 1, level 2, level 3, level 4, and level 5), application (civil, defense, transportation & logistics, and construction), drive type (semi-autonomous and fully autonomous), and vehicle type (passenger car and commercial vehicle): Global opportunity analysis and industry forecast, 2021–2030. Technical report, Allied Market Research.

Singh, S. 2018. Critical reasons for crashes investigated in the national motor vehicle crash causation survey. Technical report, National Highway Traffic Safety Administration.

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research* 15(1):1929–1958.

Sutton, R. S., and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.

Tampuu, A.; Matiisen, T.; Semikin, M.; Fishman, D.; and Muhammad, N. 2022. A survey of end-to-end driving: Architectures and training methods. *IEEE Transactions on Neural Networks and Learning Systems* 33(4):1364–1384.

US Energy Information Administration. 2017. Study of the potential energy consumption impacts of connected and automated vehicles. Technical report, US Department of Energy, Washington DC.

Van Brummelen, J.; O'Brien, M.; Gruyer, D.; and Najjaran, H. 2018. Autonomous vehicle perception: The technology of today and tomorrow. *Transportation research part C: emerging technologies* 89:384–406.

Vinyals, O.; Babuschkin, I.; Czarnecki, W. M.; Mathieu, M.; Dudzik, A.; Chung, J.; Choi, D. H.; Powell, R.; Ewalds, T.; Georgiev, P.; et al. 2019. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature* 575(7782):350–354.

World Health Organization. 2018. Global status report on road safety 2018. Technical report, World Health Organization.

Zhan, W.; Sun, L.; Wang, D.; Shi, H.; Clausse, A.; Naumann, M.; Kümmerle, J.; Königshof, H.; Stiller, C.; de La Fortelle, A.; and Tomizuka, M. 2019. INTERACTION Dataset: An INTERnational, Adversarial and Cooperative moTION Dataset in Interactive Driving Scenarios with Semantic Maps. *arXiv:1910.03088 [cs, eess]*.

Zhou, B.; Krähenbühl, P.; and Koltun, V. 2019. Does computer vision matter for action? *Science Robotics* 4(30):eaaw6661.