ENPM690 Final Project Report

End to End Learning for Autonomous Vehicles

Parthasarathy Vasanthi,Charan Karthikeyan	115522428
Jagadesh Nischal,Nagireddi	116190923

4th May 2020

Contents

1 Abstract

Self-driving cars promise to be the biggest change in the automotive and transport industry in recent times. Almost all the major car manufacturers are now involved in developing their own version of self-driving cars. Autonomous technology will be a huge industry and promises to improve efficiency and safety of the transportation sector.

The goal for this project is to train an autonomous driving vehicle using the Udacity Unity Simulator. This project was inspired by a research paper published by Nvidia [2] which uses the basis of Behaviour Cloning, where the network is trained using the initial inputs from the user and the autonomous driving behaviour is recreated using the Udacity simulator. A convolutional neural network will be the network implemented for this project. The end goal of the project is for the algorithm to mimic the human driving behaviour. The car needs to steer itself while maintaining lanes, handling slopes and turns and avoid going off road.

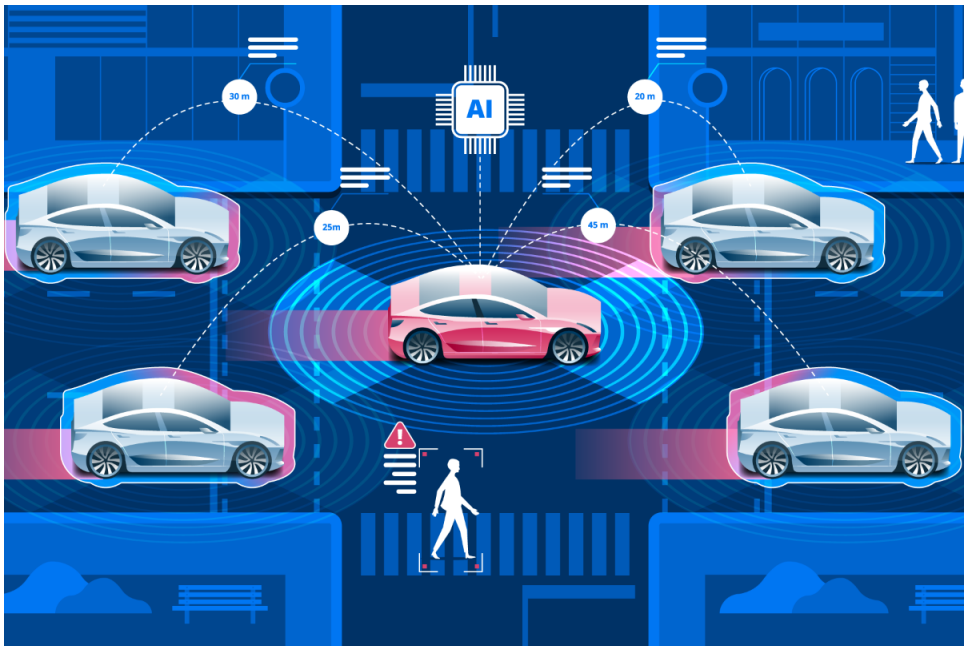
2 Introduction

Although autonomy has been in demand for quite a while, it is still a hotly researched topic and in spite of years of research, we have not still reached a safe full-autonomy system. It is a complex problem with no specific answer. We have chosen this topic to combine the incredible technology of Deep learning which has recently emerged to be a practical approach to various problems due to the advent of high-performance GPU developments in the past decade. The self driving problem is predominantly solved using classical non-Machine Learning methods so we wanted to apply the merits of deep learning to the self driving problem to achieve simple and safe autonomy. Some of the merits of going fully autonomous are:

- Brings down driver-related costs.
- reduce driver stress.
- Effective parking and reduction of parking space needs in crowded or congested areas.
- Frees up traveller time and reduces traffic.
- Mitigates accidents.
- Encourages car-pooling.
- Overall increase in speed due to vehicles being connected and in sync.

3 Background/Related Work

The team comprises two robotics students who have previously worked on Computer vision, Path planning, Machine Learning and Control systems. Having worked with navigation and automation of various robots, we chose this topic as it would involve combining the knowledge gained by various fields of robotics and Artificial Intelligence. We worked on implementing the lane detection and localisation of the vehicle through computer vision using Hough line transforms and Visual Odometry through a monocular camera. We have also implemented various projects in control systems and path planning for robotics. Although we have worked with navigation modules previously, we never implemented Deep Learning for the purpose of navigating a vehicle. We wanted to implement Deep learning on this problem to identify the merits and demerits of this process and also to understand the process of converting a problem which is predominantly approached in the classical methods to a process where the Models learns the pattern without the human supervision/intervention. So, we started reading more about self-driving using Deep learning and found this research paper [2]. The results were quite promising. Next, we wanted to create our own network which could be able to drive the vehicle and not just give visual outputs. We found this great open-source simulator [2] which was perfect for our application. We started working on Convolutional Neural Networks starting with object classification on CIFAR-10 dataset and then we implemented various networks such as ResNet, ResNeXt and DenseNet. These helped us understand the usage, applications and limitations of using CNNs. The main limitation with Convolutional Neural Networks and Deep Learning in general is lack of understanding behind the decision or output given by the network. It acts like a black box which takes in input and gives an output without having a clear answer as to how it approached towards the output. In Spite of Deep Learning being such a powerful tool, we find it hard to understand its inner workings and that can lead to problems in situations where feedback/reason behind an action is necessary.



4 Approach

The problem of autonomous driving can be approached in both classical and Machine Learning/Deep Learning means. Classical methods involve combining Computer Vision, Control Systems and Path Planning modules while our machine learning method involves using Deep Learning using Visual and Sensor data.

4.1 Classical Approach

Classical approach is the most adopted approach in the current scenario and is more mainstream when compared to the machine learning approach. Most of the current Research and developments are done in the classical approach method. The methods and explanation of the modules of the classical approach is as follows.

4.1.1 Lane Detection (Computer Vision)

Lane Detection is an essential part of an autonomous system, they are a subsidiary of the Vision module and play a pivotal role in the functioning and safety of the vehicle. The lane detection module takes in the images and synthesizes the images to highlight the lanes on the road, plot them and implement equations to find the curvature of the lane to feed as a secondary measurement to the vehicle for cross-referencing to improve its safety and have a fail safe mechanism in-case of a failure of the localization module.

Figure 1: Lane Detection

4.1.2 Localization

For any autonomous system without localizing itself, the system is unable to get to and from a certain place in the world coordinates. The same applies for an autonomous vehicle as well. In the case of our autonomous vehicle, the most common system in use is a fusion of IMU and GPS to accurately locate itself in terms of the world coordinates. This helps the vehicle to act in its required fashion by distinguishing the scenario in the current world frame and acting according to them. The vision system acts along with the localization module and helps the vehicle understand it's location and helps it proceed to the desired goal location.

4.1.3 Path Planning

Path Planning consists of planning and acting for all the modules in the vehicle. An actor is the one that implements the decided output and is in direct interaction with the external factors and provides feedback based on its execution. Planning module takes in all the inputs from various modules of the vehicle and plans to get from point A to Point B which is the intended destination. The Planning module is further classified based on the type of planning done by them, they are Global Planning, Motion Planning and Local Planning[3]. Global planning does the high level route planning from point A to point B , the motion planning takes in the route from the Global plan and converts them into short term goals the vehicle has to achieve to get to the destination. This helps the planning algorithm to break down the complexity of the overall route. The local planning gets the short term goal and generates the path to reach from the current position of the vehicle to the short term goal. The local planner takes in the various inputs from its surroundings via the various sensory modules and formulates a traversal path for the vehicle to follow along and hence the local planner is responsible for how the vehicle behaves.

In our case the module of planning is not changed much and only the part where the decision on how the inputs are used to make a decision is done and action to take is decided by the machine learning algorithms and methods.

4.1.4 Brief Summary of the Classical Approach

To give a brief few line on how the system progresses. First, the Vision module detects the lane markings and position of the car using the Localization module to localise it. Then, the Planning module generates the path for the vehicle. Next, the Control system takes over the speed profile and speed limitations and also aids in the local and global planner movements.

This approach is prone to errors and not quite robust. There are chances of the algorithm throwing a false positive which can lead to unpredictable behaviour by the vehicle unless other safety measures are put in place.

4.2 Implemented Approach

The implemented approach is different from that of the classical approach and the is up and coming field of Research on Autonomous Vehicle. They use machine learning techniques to mimic the driving behaviour of the driver on various scenarios. The modules associated with the implemented approach is explained in the upcoming sections.

4.2.1 Machine Learning/Deep Learning

his approach is not part of the classical planning method for decision making and is predominantly used in the field of image classification for the vision module. In our method we are trying to achieve the same result that is done through classical planning but trying to get a more robust and simple to implement process. The method of achieving this is called Behavioural cloning. Behavioural cloning means that the characteristics of a certain behaviour or their thought process allows machines to learn and achieve the same behaviour as that of the human. This is the machine learning method used to achieve the decision making in our vehicle. The algorithm used to achieve this is called End-to-End Learning[2]. Here we combine the controls, path planning and the vision system all into one Convolutional Neural Network and that is used to train on the human inputs to mimic the human way of driving the vehicle.

When compared to explicit discretization and decomposition of the self driving car behaviour, as in, dividing the problem as lane detection via the Vision system, path planning and control, this deep learning system approach will be able to optimize all the intermediate steps of learning simultaneously. This aids in achieving a better system along with a simpler one to implement. We can achieve better performance as the system can optimise for its own internal parameters and not human described parameters. These parameters might be difficult for a human to interpret but they are the ones deemed by the system to optimize and efficiently learn the problem.

4.2.2 End-to-End Learning

End to end learning is a machine learning algorithm developed by Nvidia for behavioral cloning. The algorithm takes in inputs from the sensors and images on the vehicle at the given time and processes those images in correlation with the steering, throttle, speed and brake to achieve the required decision. The model trains to minimize the error between the model output and human input.

The model is trained on the error between the human inputs for the sensor data i.e. steering angle, throttle, brake and speed and the algorithm output for these parameters and the algorithm trains by using the images as the training data to minimise for the sensor error.

Humans in general tend to have a bias and dividing the problem into multiple smaller problems such as lane detection, control systems, planning, etc.,. might not be the best way to approach the

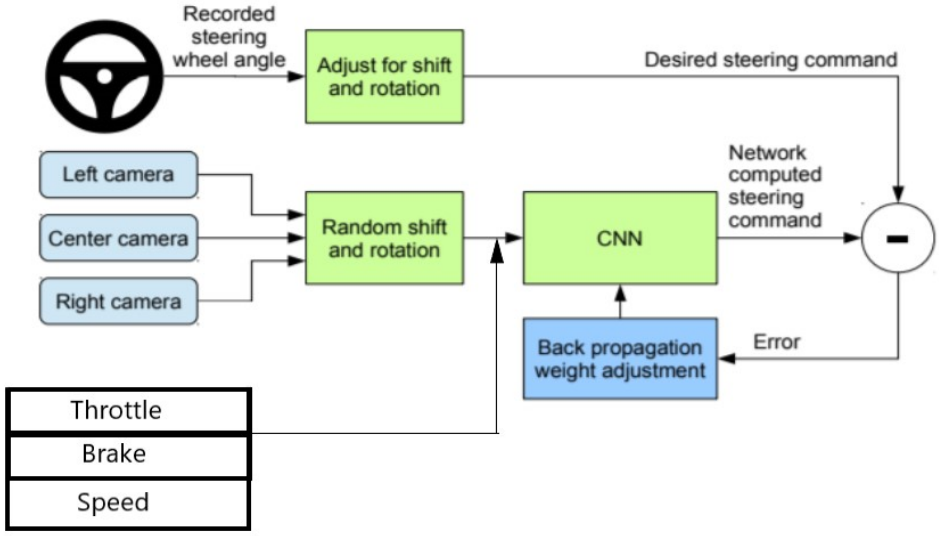


Figure 2: Model Architecture

problem of autonomy. In end to end learning, the model can optimise on internal parameters it finds are more beneficial to efficiently solve the problem and it learns accordingly. This is a huge merit for end to end learning. The training is done using Convolutional neural network with GPU based CUDA support, written in python with the PyTorch libraries and Tensorboard loggers.

5 Simulator Environment

The simulator used here is the open-source Udacity's Self-Driving Car Simulator built on Unity Engine. The simulator was built to enable training and testing of Deep Learning methods on self driving vehicles.

The simulator has two tracks to train and test the car on. The tracks vary in their complexity and terrain. The simulator car is equipped with three cameras mounted on the bonnet of the car. Video is captured simultaneously along with sensor data such as steering angle, throttle, brake and speed. For the sake of independence of the system from the car geometry, we read the steering angle as 1/r where r is the turning radius of the vehicle (in meters). '1/r' is used instead of 'r' helps in avoiding singularity while the vehicle moves in a straight line. The saved data consists of pictures taken from each of the three cameras along with the sensor data at that specific time point. It is stored in a data log file which is in csv format. The rows represent the input data points and the columns represent the number of data points that have been recorded.

The First track (Refer Figure 4) is the more simpler one out of the two with mild turns and no

Figure 3: Simulator Home Screen

inclinations but it has varying road textures. The vehicle is thought how to tackle the change in the track textures and adapt to drive without making any erratic or sudden behaviors.

Figure 4: Track 1

The Second track (Refer Figure 5) is more complicated and has multiple sharp twists, turns and steep inclined slopes. The track also has multiple textures and parts of the track covered in shadows. This is also a two lane road,over the single lane Track 1.

Figure 5: Track 2

6 Data Processing

6.1 Image Operations

- The input images are cropped to remove the bonnet of the car and also the sky as these are irrelevant features for the training operation.
- The images are resized (66x200) and changed from RGB to YUV colour space.
- The input images are then normalized to avoid any saturation and also for better gradient outputs.

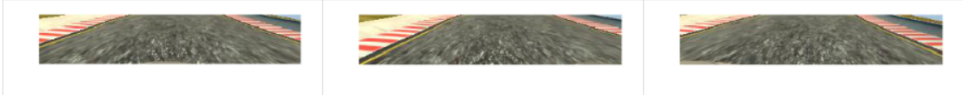


Figure 6: Cropped Images of the Left, Center and Right Cameras

6.2 Image Augmentation

- We drive the car in both clockwise and anti clockwise paths in both the tracks to generalise better and generate more data.

- For all left camera images, the steering sensor angle has been altered by '+0.2'.
- For all right camera images, the steering sensor angle has been altered by '-0.2'.
- We flip images horizontally and add to the training data.
- We randomly translate the image (horizontal manner) with steering angle adjustment.
- Randomly change the image brightness to lighter the darker areas and make a uniform brightness distribution.
- Usage of the left and right camera images is useful to train the model during the recovery driving scenarios
- Separately record recovery training data by intentionally driving the vehicle off lane and recover back to the lane.
- Horizontal translation for images is quite useful for difficult curve handling.

7 Implementation

7.1 Network Architecture

The network for our project is adopted from Nvidia's End to End Learning model, with minor changes in the shape of the data and the normalization methods. The network consists of a Normalization layer followed by five Convolutional layers. The first two convolutional networks have a search pattern of 5x5 kernels and the next three Convolutional layers have a 3x3 search kernels. The convolution network is followed by a flatten layer to convert the two dimensional convolutional layer into a single vector to feed to the fully connected layers to follow. The Flatten layer connects the five Convolutional Layers to four Fully connected layers. The four fully connected layers have 100,50,10 and 1 as the number of neural layers to pass through for each fully connected layer. This gives the final prediction of the given data as a single weight vector. The image of the architecture is shown below.

The generated weight vector is saved after the training phase is over and saved to a h5 file for future prediction (Autonomous phase) and reference.

7.2 Training Data Generation

The simulator has a Training mode where the human can operate the vehicle manually using either the keyboard, mouse or a joystick. Here we have the option of recording the human driving through the captured images of the cameras and the sensor measurements. The training data is recorded in the folder of your choice as an image folder along with a '.csv' file. The csv file is a tabulated data

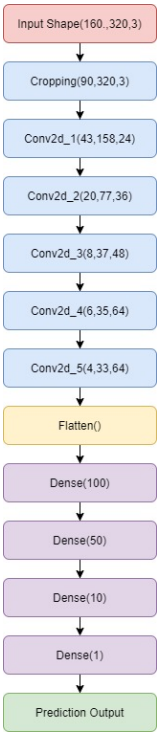


Figure 7: Network Architecture

log of the image paths and the sensor measurements taken during the 'record' Training mode. This is the file we use to train our network.

7.3 Training the Model

The training of the model is done in batches and is shuffled each epoch before being fed to the network. Since the network works on making decisions and can only be tested on the simulator, unlike other image classification networks where we have a set of test data to run the model against, we save the model and run it against the real time images from the simulator and test its effectiveness. When running the model after each training epoch on the simulator, we can see that the driving decision of the vehicle improves and the errors such as driving off the lane, crashing,,swerving improves or entering or exiting a curve goes down. After a certain number of epochs we can see that the model fares well on the simulator with the vehicle trying to stay in the middle of the road as much as possible and in some places the car handles better than a normal person would. In certain models we can also see how the model over fits by the way the vehicle makes decisions such as extreme swerving due to overcompensation and under compensation and over fitting to a certain texture. This also shows and replicates exactly as the training data was generated along with its erratic behaviour. It fits itself to one single scenario and does not contain a generality to produce dependable decisions in a different scenario. The output from each single epoch is saved as a model in a folder as a h5 format to read and call during testing on the simulator.

7.4 Testing the Model

To test the model, we plot graphs for Training and Validation loss to understand the model learning. Also, the simulator has the option to test the trained model on. To use this mode, activate the autonomous mode on the main screen of the simulator along with the choice of the track. Here we run the drive.py file in a terminal along with the saved model. The vehicle should drive on it's own using the saved model as the driver for the vehicle.

8 Results

The results are split up into statistical and observational results.

8.1 Training, Validation and Test

We split the image data-set into a training set and a validation set in to evaluate performance at each epoch. We do the testing using the simulator environment.

The evaluation criteria for training is MSE (Mean Square Error) as the loss for the measurement of how similar the model predicts compared to the human input behaviour in terms of steering angles. We have used Adam Optimiser with a learning rate of 0.0001. The default value of Learning rate was quite high which led to the improvement of validation loss stopping quite early.

This code checks for GPU and runs pytorch cuda if GPU is detected, else, the training takes place on the CPU. In our case, The training took approximately 12 minutes per epoch when trained on a Nvidia GTX 1060 6GB GPU for a training set of around 200 training points (3 images each). We found Track 1 to converge at 5 epochs while track 2 converges at around 20 epochs. Next, the parameters for training the network are:

- Train Size fraction
- Drop out Probability
- Number of Epochs
- Samples per Epoch
- Batch Size
- Learning rate

All of these values can be changed before training the model. The default values when unaltered by the user are as follows: Train size fraction= 0.8; drop out probability=0.5; number of epochs=10;

samples per epoch= 20000; batch size= 40; learning rate= 0.0001.We have experimented with varying values of drop out probability and number of Epochs. Below are the resulting plots from our project.

Results observed from Track 1:

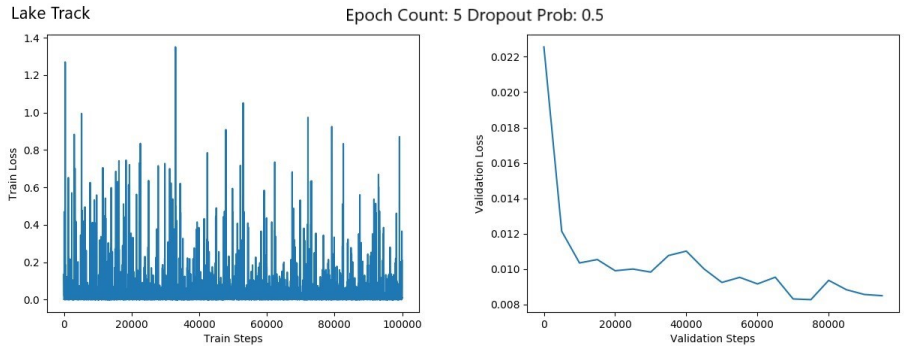


Figure 8: Output from 5 Epochs with 0.5 Dropout

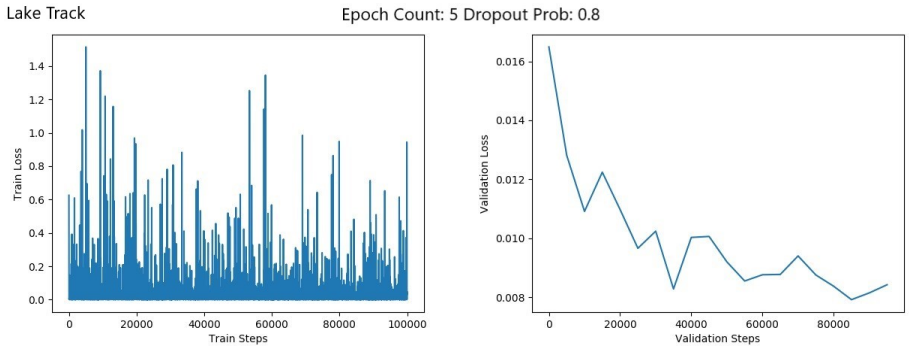


Figure 9: Output from 5 Epochs with 0.8 Dropout

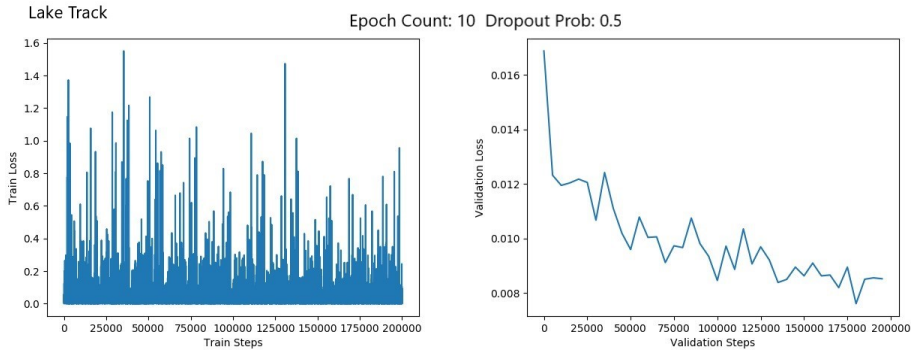


Figure 10: Output from 10 Epochs with 0.5 Dropout

From the given graphs in Figure 8,9,10,11, the first 4 graphs (Epoch 5, Fig 8,9) denote the model which was a Good fit through the driving behaviour from the simulator Autonomous mode for the

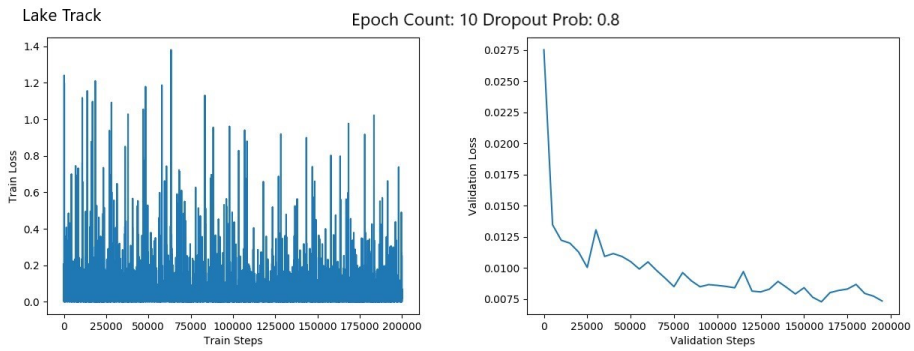


Figure 11: Output from 10 Epochs with 0.8 Dropout

Track 1 (Lake Side track). We can observe a good drop of validation loss with Drop out probability at 0.5. The loss was seen to decrease faster compared to Drop out probability at 0.8. The Train loss was quite identical in both these cases. Coming to the next 4 graphs (Epoch 10, Fig 10,11) which denote an Overfit model according to the driving behaviour as tested on the simulator Autonomous mode. Here, contrastingly, the validation error is dropping at a much faster rate for Drop out Probability at 0.8 than for the 0.5 condition while the training loss remains quite similar. In the case of 10th epoch, the model is trained beyond requirement and tends to 'memorise' the training data leading to an Overfit driving behaviour.

Results from Track 2(Jungle Track):

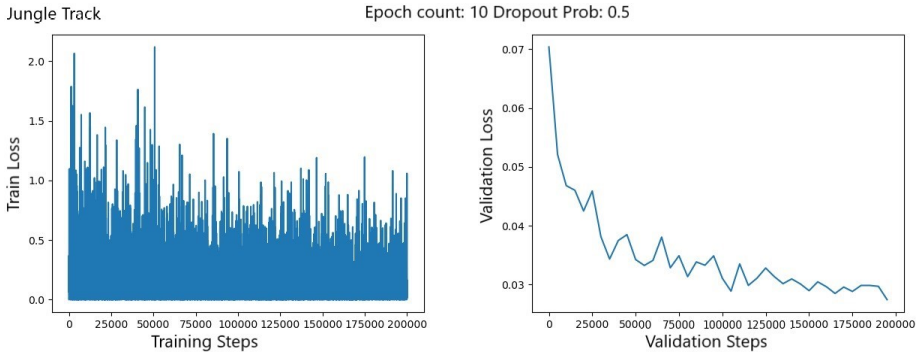


Figure 12: Output from 10 Epochs with 0.5 Dropout

From the graphs given in Figure 12,13,14,15, the first 4 graphs (Epoch 10, Fig 12,13) denote the model which was an Underfit fit through the driving behaviour from the simulator Autonomous mode for the Track 2 (Jungle track). We can observe a good drop of validation loss with both Drop out probabilities of 0.8 and 0.5. The loss is a tiny bit quicker for Drop out probability at 0.5 compared to Drop out probability at 0.8. The Train loss was quite identical in both these cases.

Coming to the next 4 graphs (Epoch 20, Fig 14,15) which denote a Good fit model according to the driving behaviour as tested on the simulator Autonomous mode. Here, the validation error is dropping at a much faster rate for Drop out Probability at 0.8 than for the 0.5 condition while the training loss remains quite similar. In the case of the 20th epoch, the model is trained just right and

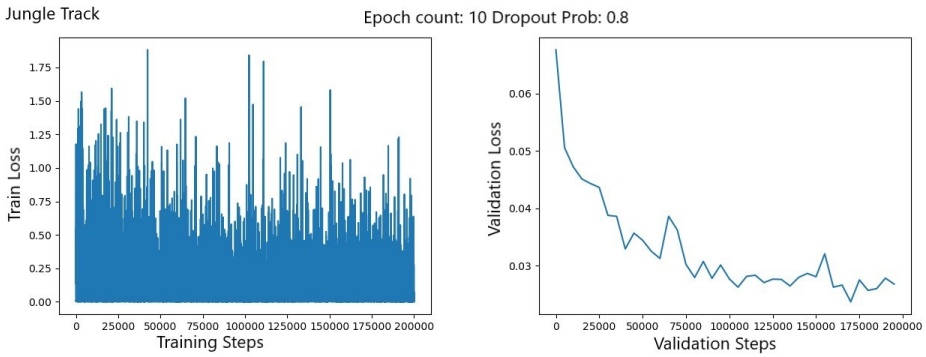


Figure 13: Output from 10 Epochs with 0.8 Dropout

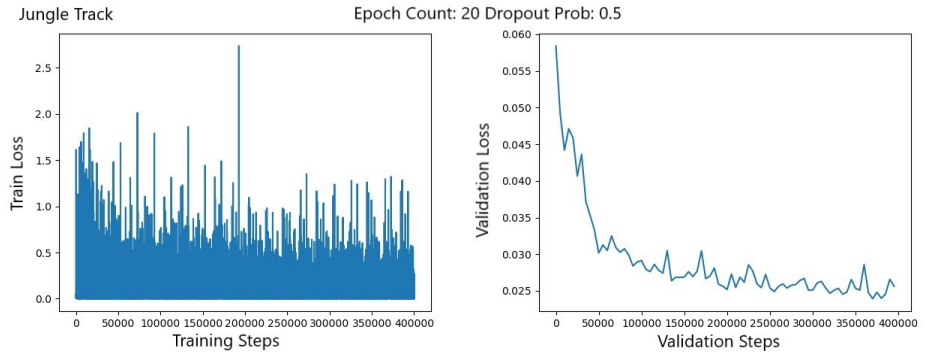


Figure 14: Output from 20 Epochs with 0.5 Dropout

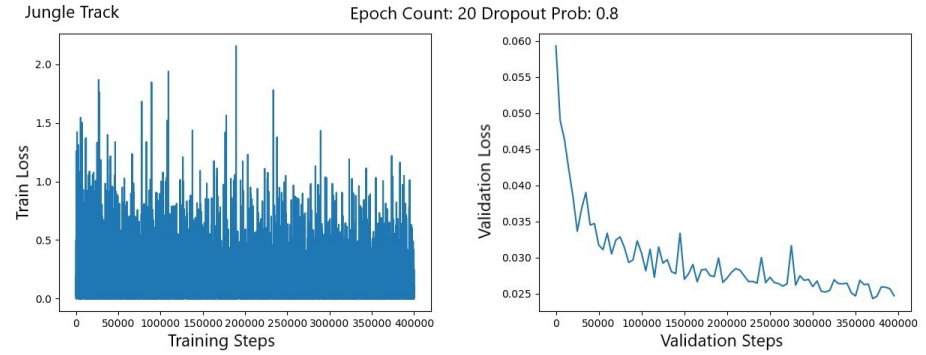


Figure 15: Output from 20 Epochs with 0.8 Dropout

the model can generalize well.

In our Project, graphs do not convey all the information. For instance, with autonomous vehicles, the driving can vary depending on the environmental changes but the car might still be maintaining its lane which is ultimately the goal for the project. Another factor to consider while looking for a good model output would be the smoothness at which the vehicle drives. This plays a huge role in the comfort of the passengers as a jerky speed profile can lead to an uncomfortable ride. So, we not only took the graph outputs as the test of our models, we also tested the model back on the simulator via the 'Autonomous' mode to see how the car responds to real time application.

So to test the network, we test the model output on the Simulator in 'Autonomous' mode. Here the saved model weights are used as a feed-forward network to predict the output for Steering angle, Throttle, Brake and Speed and these values are sent in real time back to the simulator software to steer the vehicle, Here the vehicle cameras send the images to the network and the output is the sensor data which controls the car in the simulator environment. The analysis of the testing on the simulator is explained in the next section of the report.

9 Analysis

The analysis has been split into two parts based on the track.

9.1 Track 1 (Lake Side Track)

Track 1 has mild turns and no inclinations but it has varying road textures. The vehicle needs to understand these changing road textures and identify these as valid locations for driving. This track was trained over 10 epochs. At epoch 1, the model was 'Under fit' with the vehicle unable to maintain its lane and went into the lane nearby. At epoch 5, the model was a 'Good fit'. The car was driving smoothly with turns taken in a non-jerky manner and the car never left its lane. At epoch 10, we found the model to be an 'Over fit' as the car showed over compensation towards going to the left lane even on straight roads. Also, the turns were not as smooth as epoch 5 and there was visible over compensation by the car to stick to the lane and the model could not generalize well. Reason for the vehicle trying to move towards the left lane was due to the training data being generated was with the car driving close to the left lane. If our model can generalize well, it would avoid trying to go towards the left side of the road and would only compensate towards the sides while taking a turn in order to stay in lane. In epoch 5, we can observe a generalized driving behaviour while in epoch 10, the model was overfitting and 'memorizing' as it was continuously trying to move towards the left side of the road.
Video Link : Track 1 Demo

9.2 Track 2 (Jungle Track)

Track 2 is a more complicated environment comprising parts of the road covered with shadows, inclinations and steep turns. Also, there are two lanes here compared to one for track 1. The model was trained on 20 epochs while the car was driven on the lane marking between the two lanes. At epoch 1, the model was 'Underfit' with the vehicle unable to maintain its lane and went and hit the road dividing object instantly as it could not deal with the shadow on the road right at the start of the track. At epoch 10, the model was still an 'Underfit'. The car was driving well with turns and inclinations handled with low speed but the car crashed into the obstacle outside the lane while taking a turn. At epoch 20, we found the model to be the 'Good fit' as the car showed good lane

keeping behaviour while handling turning and inclinations with care and the car was not going off-lane like the previous epochs. This was the desired output needed from the model.
Video Link : Track 2 Demo

10 Conclusion

At the end of this project, we have come to see that we could successfully train an autonomous vehicle to clone the behaviour of the human through a non-complex Convolutional Neural Network and using relatively small amounts of training data and computational power. We found this Machine Learning approach to the problem of self driving cars to be a better solution compared to the classical approach of discretizing the problem into multiple small problems and combining all the systems to work in sync. We found the Machine Learning approach to be quite a robust way to solve the problem of self driving vehicles.

11 Future Work

In this project, the model was trained and tested on a simulated track with empty roads and no obstacles on the lanes. This is far from the reality. The future work would be to add more real world obstacles and objects to make the model respond to the environment. Also, the vehicle needs to understand the traffic rules and signs. This would be the next logical step to build on top of the existing model. Also, the vehicle was found to be quite jittery and wobbly at certain times. This would need to be smoothed out to reduce the jerky feeling to the passengers in the car. Also a more efficient program would help in reducing the training time. A more powerful hardware would improve real time performance of the self-driving car as the system can process the input of images and sensors faster and react better to the environment.

12 Bibliography

[1] Car simulator:(https://github.com/udacity/self-driving-car-sim)

[2] Research Paper: Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, End to End Learning for Self-Driving Cars (https://arxiv.org/abs/1604.07316)

[3] On-Road Motion Planning(https://ri.cmu.edu/pub$_{f}iles$/2012/10/$ICIRA2012.pdf$)

[4] Waymo autonomous research(https://sites.google.com/view/waymo-learn-to-drive)

[5] Nvidia DL Research (https://devblogs.nvidia.com/deep-learning-self-driving-cars/)

13 Code Repository

The code for the above project is attached in the link given.
Github Link :End-to-End Learning for Self Driving Car