

Spring 5-22-2019

Improving Steering Ability of an Autopilot in a Fully Autonomous Car

Shivanku Mahna
San Jose State University

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_projects

Part of the [Artificial Intelligence and Robotics Commons](#)

Recommended Citation

Mahna, Shivanku, "Improving Steering Ability of an Autopilot in a Fully Autonomous Car" (2019). *Master's Projects*. 716.
DOI: <https://doi.org/10.31979/etd.b9hs-3fce>
https://scholarworks.sjsu.edu/etd_projects/716

This Master's Project is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Projects by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

The Designated Project Committee Approves the Project Titled
Improving Steering Ability of an Autopilot in a Fully Autonomous Car

By

Shivanku Mahna

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

Spring 2019

Dr. Robert Chun, Department of Computer Science

Dr. Philip Heller, Department of Computer Science

Ms. Pooja Prashar, Santa Clara University

Improving Steering Ability of an Autopilot in a Fully Autonomous Car

A Thesis

Presented to

Dr. Robert Chun

Department of Computer Science

San José State University

In Partial Fulfillment

Of the Requirements for the Class

CS 298

By

Shivanku Mahna

May 2019

ACKNOWLEDGEMENT

I would like to thank Dr. Robert Chun for his continuous guidance during the course of my thesis. I would also like to thank him for showing the faith in my abilities and giving me complete freedom to do the project of my liking and interest. I would also like to extend my gratitude towards my Committee members, Dr. Philip Heller and Ms. Pooja Prashar, both of whom have been really kind and helpful in lending me their patient ears and giving value feedback on what I have been trying to achieve through this project. Lastly, I would also like to extend a note of thanks to my artificial intelligence professor, Dr. Gayathri Namashivayam, whose excellent style of teaching helped in inculcating a deep interest for Machine Learning in me and hence paved a path for this project.

ABSTRACT

The world we live in is developing at a really rapid pace and along with it is developing the technology that we use. We have clearly come a long way from calling a car modern because it had a touch screen infotainment system to calling it modern because it drives on its own. The progress has been so rapid that it demands for us to analyze this and try to improvise a small part of this journey. With the same thought in mind, this project focuses on improvising the steering ability of an autonomous car. In order to make more sense of what is an autonomous car and what all goes on inside the working of a car that runs on its own, the thesis will be divided into a part of it explaining the theory and a part of it explaining the logic and results achieved as a part of the experiments performed on the data. It will show how an autonomous car's CNN model can keep on becoming better as we keep on feeding more and more data into it. It will also show how a CNN model is a generalized model which can not only be used for steering a car but also to control and predict the speed of the car or the breaking ability of the car.

Table of Contents

I. Introduction.....	1
II. Autonomous Vehicles.....	1
III. Levels of Autonomous Car.....	2
IV. Components of Autonomous Car.....	4
4.1 LIDAR	5
4.2 RADAR	6
4.3 Camera	7
4.4 GPS and IMU	8
4.5 Sensors	9
4.6 GPU	10
4.6.1 Unified Shaders	11
4.6.2 Vulkan API	13
V. Tasks in Autonomous Car	14
5.1 Localization.....	14
5.2 Object Recognition	15
5.3 Object Tracking	15
5.4 Network Architecture	16
5.5 Data Selection.....	18

5.6 Augmentation	18
5.7 Simulation.....	18
5.8 Evaluation.....	19
VI. Current Research & Implementation of Autonomous Car	20
VII. My Implementation of Autonomous Car	22
7.1 Converting to Greyscale	23
7.2 Gaussian Blue	24
7.3 Detecting Edge in an Image	25
7.4 Masking Edge in an Image	26
7.5 Performing Hough Transformation in Image	27
VIII. Tools and Dataset	29
8.1 Python Libraries	29
8.2 C++ Header Files	31
8.3 Caffe	31
8.4 Dataset	32
8.4.1 Udacity Dataset	32
8.4.2 Self-created Dataset	33
IX. Experiments & Results	37
9.1 Pre-processing the Data	37
9.2 Experiments	40

9.2.1 Training & Testing on Self-Created Dataset	45
9.2.2 Training & Testing on Subset of Udacity's Dataset	42
9.2.3 Training & Testing on Complete Udacity's Dataset	44
9.2.4 Testing again on Self-created Dataset	45
9.3 Addition Experiments	47
X. CONCLUSION	49
XI. Future Works	51
References	52

I. INTRODUCTION

The world we live in is progressing at a very fast pace and so is the technology that we use. We have come a long way in terms of the technology we have inside a car and the technology we use to make a car. From being in an era of having mandatory airbags for the protection of driver, we have come to a stage where driverless vehicles have become a reality. This leap was made possible only because of the availability of advancements in the world of Radars, ability to create 3D mapping of our surrounding at an astonishing rate of 90 frames per second and due to technological developments, which enabled us to better utilize the data which we are able to collect from our surrounding environment. To better appreciate and understand the value of this development, it is very important to understand what the very basic concepts like what does autonomous mean and when do we call a car as autonomous.

II. Autonomous Vehicle

We call a system as autonomous when it has the ability to self-govern and provide satisfactory performance even under various uncertainties in the environment. It also refers to the ability to compensate for system failures without any external intervention. A single functionality being made to work on its own in a system is referred to as automation. But, when multiple automations are made to function together due on the commands of an artificial brain made of neural network, it is known as an autonomous object.

Similar to this, a car is referred to as autonomous when it has the capability of sensing its environment and navigating from source to destination without any human input. What makes an autonomous car truly autonomous is the fact that it requires zero human intervention and has the ability of handling immediate response situations such as applying immediate breaks, changing lanes to name a few. The natural question which comes to mind is how does such a car function. The following section aims at explaining the same while also explaining the different levels of autonomous cars.

III. Levels of Autonomous Car

Level 0

We call a car as Level 0 automated when it has absolutely no automation. The only amount of automation in such a vehicle is the system warning lights or alarms. Vehicles in such a category require constant monitoring from the human driver.

Level 1

Cars classified in this level of autonomous are also known as “Hands On” Cars. Such cars have humans and automated systems sharing the control of the vehicle. An example of can be the feature of “Adaptive Cruise Control” in which the human controls the steering while the system manages the speed of the vehicle. Another example of such a feature can be a “Parking Assist” in which the human controls the speed but the system controls the steering.

Level 2

Cars classified in this category are also known as “Hands Off” cars. Such cars typically have the system having full control of the vehicle for certain conditions or situations, but the human driver is still required to monitor automated driving being done by the system even in those reliable conditions as the system might fail to respond to certain kind of emergencies and hence a human might be needed to step in and take control of the vehicle.

Level 3

There are certain vehicles being produced which can behave as almost fully autonomous in certain conditions. When the vehicle is in those specified conditions, the vehicle’s system can control the car and even handle emergency situations, making it robust and allowing the driver to relax. However, the driver might be required to intervene if the vehicle exits the conditions in which it can act fully autonomous. Hence the driver must make sure that the vehicle is functioning in the specified zone before leaving it unattended. An example of such a car can be the new 2018 Audi A8. The latest Audi can drive automatically in any traffic conditions till its speed reaches 60 Kmph. Hence such a vehicle is fully autonomous from 0 to 60 Kmph but will require the driver to intervene and pay attention once the car crosses 60 Kmph mark.

Level 4

Such category of vehicles are similar to Level 3 vehicles, with the only difference being that they are allowed to behave fully autonomous only in certain geographical areas or in certain traffic conditions. For example, some vehicles classified as Level 4 are only allowed to behave as fully

autonomous in geographical areas where the human population is significantly lesser than the other areas. Such vehicles can be considered as being more robust than a Level 3 car but not being robust enough to be classified as a Level 5 car.

Level 5

Vehicles classified as Level 5 are regarded as truly and fully autonomous cars. No human intervention is required to handle any emergency situation and the vehicle is smart enough to sense the environment and take decisions based on the analysis of the surroundings and the data being collected using various sensors. Such vehicles work under all traffic and geographical conditions. Example of such a vehicle can be Google's Waymo.

IV. Components of Autonomous Car

To make a vehicle capable of making important decisions on its own, it needs to be supplied with extremely accurate data and processing power, all of which, when happening in parallel, can help the car take advanced decisions without the intervention of any human, even in the most adverse conditions. In order to make it all possible, there are a lot of components which go into making a fully autonomous car. While every component has some pros and cons, amalgamation of data received from all the components helps the system to make all the necessary calls. Taking the environment data as the input, the vehicle should be able to localize itself, detect the objects, classify them and further process this data to make navigating decisions in real-time. Since the decisions on the data collected from the sensors are to be made in real-time, the vehicle should consist of an extremely powerful and energy efficient computing device. The different components which make it

possible are Li-DAR, RADAR, Cameras, Sensors, GPS/IMU and GPUs. The different components are explained in detail in the sub sections below.

4.1 LiDAR

LiDAR is a Light Detection and Ranging System which is capable of bouncing back the laser Light off the objects in order to make it easier to detect their shape, size and location with respect to the car. This is done by first emitting a light pulse and noting the time at which it was emitted. As soon as the light pulse reaches an object and gets reflected, the reflection of pulse is detected. Once the reflection reaches back the LiDAR, its wavelength is recorded, and the time is calculated. Using both time and wavelength, the distance of that particular object from the car is calculated. While this might seem like a lot of steps, but all this happens in 1/10th of a second and hence we are able to quickly sense the surroundings of the car. As a result, we are able to obtain a 3D map of the surrounding of the car.

LiDAR architecture consist of three things: -

- Transmitter: It is responsible for transmitting the light pulse from the car.
- Receiver: It is responsible for receiving the light pulse.
- Data Acquisition and Control System: It is responsible for doing all the calculations on the data and outputting the results. It is the same system which notes the time at which the transmitter transmits the light pulse and the same system notes the time at which the light pulse reflects and reaches back to the receiver.

Figure 4.1 below shows how the 3 components in the LiDAR architecture are connected.

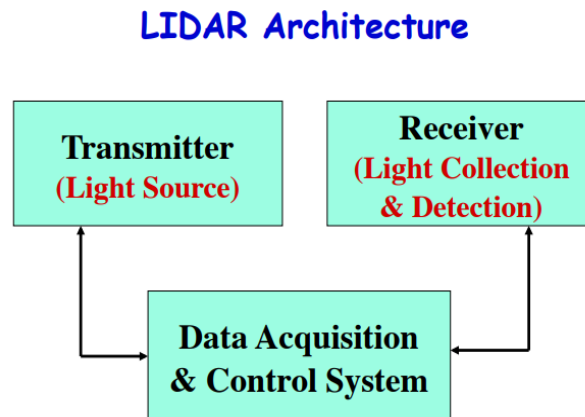


Figure 4.1: LiDAR Architecture

4.2 RADAR

Radars are responsible for transmitting the radio or micro waves. Radars also work pretty similar to LiDAR but the major difference in both LiDAR and Radars is that LiDAR transmits Light waves whereas Radars transmit Radio waves. Radio waves emitting from a radar are used to gauge the range, angle and the velocity of different objects around an autonomous vehicle. Radars are able to do so by having the radio waves reflect from the different objects present in its path and then doing calculations on that data to get various parameters about the surrounding of the car.

There are 3 types of radars which are used in an autonomous car. They are as follows:

- Long range Radar (76 to 81 MHz) - Has range up to 200 m
- Example of use of a Long-Range Radar in action can be that of Adaptive Cruise Control in a car which helps in driving the car automatically till the car reaches a certain defined speed.

- Medium range radar (76 to 81 MHz) -Has range up to 60 m
- Example of use of a Medium-Range Radar in action can be that of Lane change assist. Another example can be that of a Blind spot detector as well as Emergency brake assist in a car.

- Short range radar (24 to 26 GHz) - Has range from 0.2-30 m
- Example of use of a Short-Range Radar in action can be that of Rear collision warning and Parking assist in a car.

4.3 CAMERA

A fully autonomous car has multiple Cameras on its body. It has a camera in the front, a camera in the back and multiple small fish-view cameras to provide a deeper and wider angle shot of the surroundings. Cameras help in providing a 360-degree view of the surroundings of the vehicle. Also, it helps in reassuring that whatever prediction we have done using LiDAR and RADAR, is accurate or not. It is also useful in recognizing road signs and lane markings, using which an autonomous steering in a Level 5 autonomous vehicle functions. Since the car has multiple cameras, all the cameras provide overlapping images to the software, which are then processed using Image processing algorithm to identify different objects and form a 3D image out of the 2D images so obtained. The images also help in determining the depth of field, peripheral movement and the dimensionality of the objects.

4.4 GPS and IMU

GPS, like in any other vehicle, helps in determining the Global position as well as helps in Localization, which is explained later in this project report. It also helps in reporting inertial updates about our position to the other cars so that they know about our position with respect to their vehicle and also helps in determining the Global position estimates. But, as with every positive comes a negative, GPS has a really slow updation rate and hence alone cannot be useful for a situation in which an autonomous car has to make a decision within seconds. Therefore, autonomous cars are fitted with another device called as IMU. IMU helps in providing real time position updates and helps in speeding up the process of decision making in an autonomous car. But, the caveat with IMU is that they cannot be used to really long periods of time. Hence a combination of GPS and IMU help in obtaining the desired information with ease, while not trading off accuracy for speed.

4.5 Sensors

While everything mentioned in the project report before this can also be categorised as a Sensor, there are some sensors which cater to special requirements and perform special functionalities in an autonomous car. The following are the sensors which are really important for efficient and accurate functioning of a fully automated car:

- Wheel Speed Sensor – It helps in making sure that the vehicle, when driving on its own, doesn't accelerate too much on the curves and saves the car from toppling over.

- **Emergency Braking** – It helps in proving the functionality of emergency breaking to the car in the events of immediate response requirement.
- **Pedestrian Detection** – It helps in detecting if there are any pedestrians which are crossing using the Zebra crossing and if the car needs to come to a halt and need to wait for them to cross over and then start off again.
- **Collision Avoidance** – It helps in making sure that we don't collide with the car ahead of us or the car behind us during driving forward or in reverse. It makes sure to indicate the autonomous system about the proximity of the cars with respect to the vehicle, so that the system can take the required steps in time and avoid any untoward incident.
- **Rear Collision Avoidance** – It helps in making sure that whenever the car is trying to reverse or trying to park in reverse, it is steering clear of any vehicle/person/animal behind it.
- **Blind Spot Detection** – As the name suggests, it helps in determining people or vehicles which might not be visible to Cameras or even a LiDARs. Hence blind spot sensors make sure to alert the system and help the system in changing lanes or avoid close by traffic and drive in a safer way.
- **Cross Traffic Alerts** - It helps in providing real time updates about the traffic to the car system so that the system can plan its route in accordance with the traffic update and can avoid the

highly congested routes. It highly depends on the GPS, IMU and cloud data regarding the location of the traffic and vehicles.

4.6 GPU

GPU is the brain of a fully autonomous vehicle which enables all the sensor data to be processed at real time and helps in generating useful deductions out of that data. The reason for selecting a GPU for this job and not a CPU is that a GPU works 33 times faster than a CPU and processed data 150 times as fast as a CPU does. Hence, in a situation where every millisecond counts and we have to process data and get results as soon as possible, GPU is the right choice. Amongst several possible options available, companies are relying on NVIDIA GPUs due to their exceptional 20 TFLOPS single as well as double precision as compared to 7.5 TFLOPS in their nearest rival AMD.

GPUs process data in the form of the architecture shown in Figure 4.2 below. The GPU architecture consists of several stages, with every stage having its own significance and unique importance for rendering final image as the output on the screen. Modeling 3D information to a screen full of pixels is known as Rendering Pipeline. It is the sequence of steps that the GPU takes to render the scene.

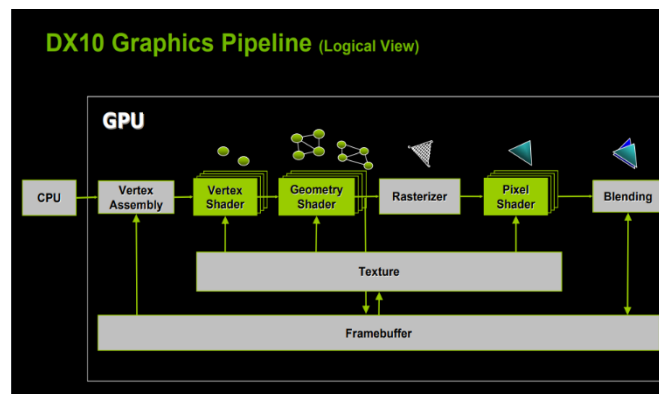


Figure 4.2 Architecture of a GPU

While GPUs are really fast in processing the data, if we want to maximize their performance and obtain even faster speeds, then we need to improvise on their current architecture. There are two possible ways of obtaining even better performances from a GPU. Both the methods are explained below.

4.6.1 Unified shader:

A GPU consists of two processors which are known as Vertex processor and Fragment processor.

When a vertex processor in a GPU is used, there are times when the fragment processor can be sitting idle and vice versa. Hence, this points to a situation in which we are not utilizing the resources to its fullest potential and hence it lead to the birth of the concept of Unified shaders.

A unified shader is capable of performing both vertex as well as fragment shading, as and when required. As a result, we ended up having a single unified architecture which can perform the same amount of work in half the time with 0 idle time. The example showing benefits of having a unified shader are shown in figures below.

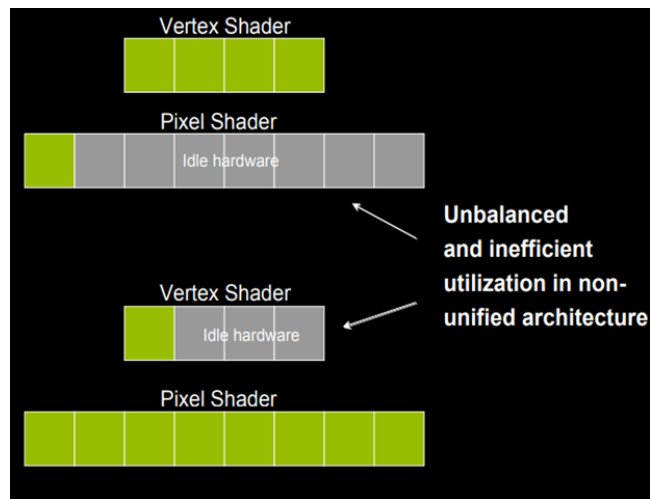


Figure 4.6.1: Separate Vertex and Fragment shaders

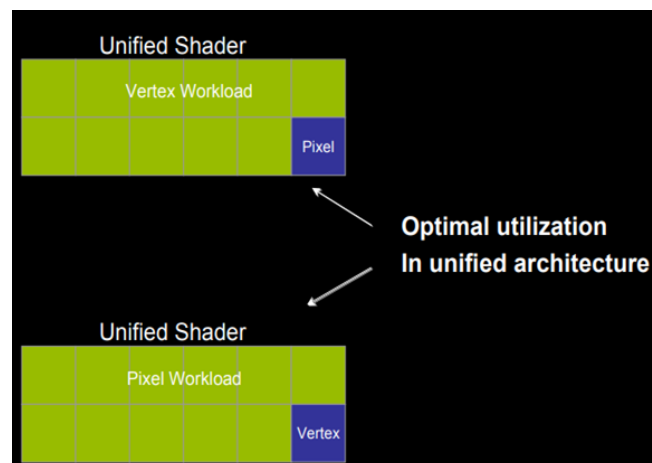


Figure 4.6.2: Unified shaders

In figure 4.6.1, we can see a vertex shader being able to handle a maximum load of 4 and being fully utilized whereas a fragment shader being capable of handling a load of 8 is being underutilized, resulting in inefficient utilization of the architecture. Now comparing that to having a unified shader model shown in figure 4.6.2, in which the whole available space of 12 units can

act either as a vertex shader or a pixel shader, as and when required, making a case for optimal usage of the resources.

But, as is with every other good thing, unified shaders also have their own set of downsides and they are:

- a) Since the shaders are programmable, we are giving too much power in hands of programmer.
- b) A badly written shader can slow down the whole GPU and affect its performance.

4.6.2 Vulkan API:

API's are the interface for effective communication between hardware and the software. It enables software to utilize all the newest features of a graphics card and use it to its maximum potential. Vulkan API is useful because it delivers notable performance improvements by reducing the driver overheads and improving multi-threaded CPU usage. Hence it also helps in maximizing the performance which can be obtained out of a Graphics card.

V. Tasks in Autonomous Car

The different tasks in an autonomous car are divided into three major stages called as Sensing, Perception and Decision. Sensing consists of collecting of data using all the sensors which are located in an autonomous car and then passing on the data to the perception stage.

The different parts of Perception stage are localization, object recognition and object tracking, which are explained in the following sub sections below.

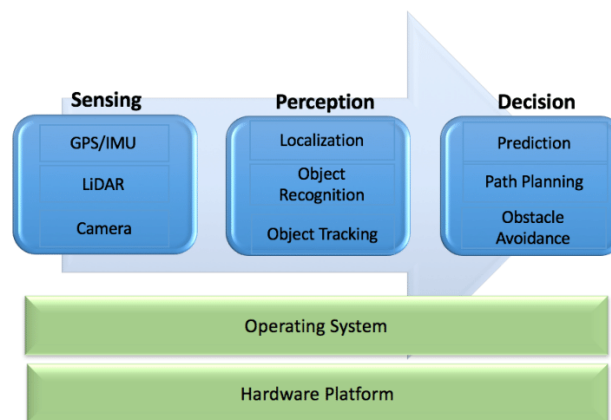


Figure 6.1 Different Stages of tasks in autonomous car

5.1 Localization

It refers to knowing “where am I” in the global map. It consists of sensor-fusion process in which similar kind of data from all the sensors is analyzed to deduce useful results and generate a 3D ground map. For example, Camera data, LiDAR data and Radar Data is matched, and a map of the surrounding environment is created. It uses particle filter method for real-time localization.

5.2 Object Recognition

Convolutional Neural Networks are responsible for object detection and recognition in a fully autonomous car. It is a type of Deep Neural network consisting of multiple layers such as Convolution layer, Activation layer, Pooling layer and Connected layer.

5.3 Object Tracking

It captures the surrounding Information using sensors. The sensors activate the actuators which generate the order to activate the final components.

Also, it is because of object tracking and sensors that an autonomous car is able to detect the traffic light. It uses an actinometer, which detects the intensity of radiations of light. Since lights of different colors radiate with different intensity, hence red, green and yellow light is easily identified by the sensors.

Decision making requires prediction, path planning and obstacle avoidance. All these three things are done using CNNs through a series of steps which are explained in the sections below.

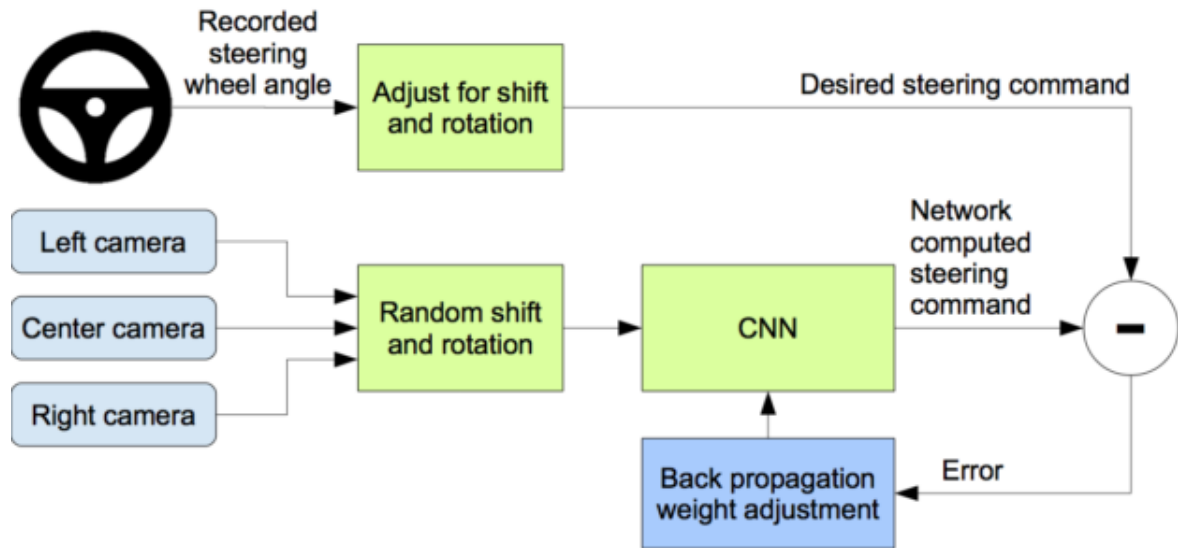


Figure 6.2: Training of a Neural Network

5.4 Network Architecture

A network of a self driving car is made up of CNN. CNN is really useful tool because we can simply train a CNN model on some examples and it has the ability of learning those functions or behaviors which result in those examples. CNN finds its biggest application in the field of image recognition and image processing, where in, we can extract the features of an image and feed the 2D nature of the image to the convolution. Since we are using CNN and not pattern recognition, relatively fewer parameters are needed and hence it makes the entire process very smooth.

This is done as follows:

1. We try to minimize the MSE (Mean squared error) by training the weights and bias of our network. The loss or error can be calculated as the difference between the steering rotation degree values of an actual person versus the model trying to steer the car. The lower the value, the better the model.

2. For a lot of companies like google, their model consists of 9 layers, with 5 layers being convoluted and 3 layers being fully connected layers. The input image is not directly worked upon. Instead, a lot of filters are applied on the image to remove the extra data from it. It is done by splitting the input image into smaller planes and passing it into the network.

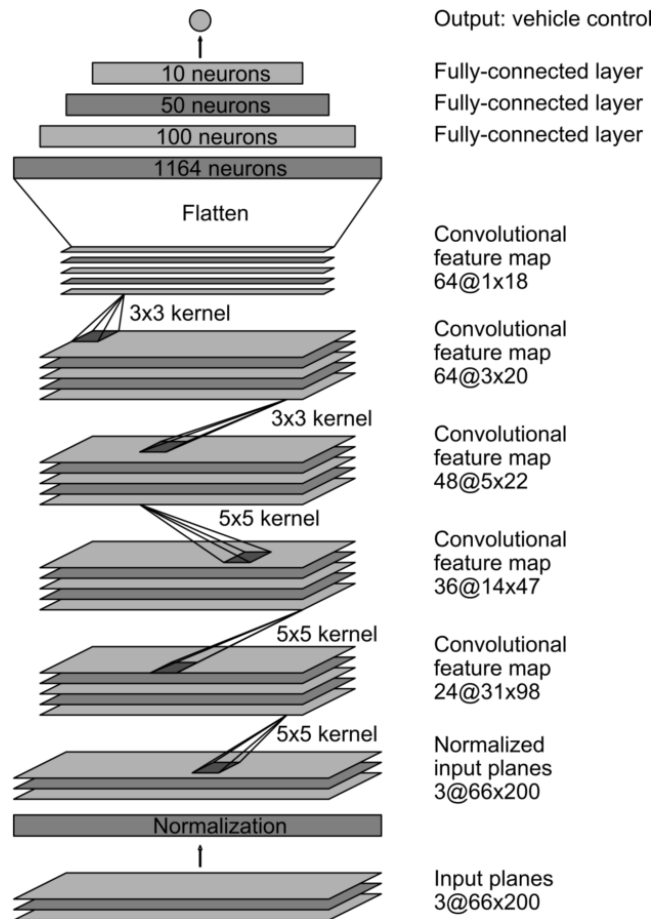


Figure 6.3 Architecture of a CNN used in an Autonomous Car

5.5 Data Selection

In order to train a CNN, we need high quality data, which is labelled correctly. This is made available with the help of the sensors which are attached to a self-driving car. Objects such as LIDAR, RADAR, Camera help in providing a detailed description of the surroundings of a car. The data obtained from all these resources is then synced with the steering values, acceleration values and breaking values for those moments in time. The data is also labelled to depict the type of weather in which the data was collected to make it extremely realistic.

While selecting the data to train the model, we want to train the model for so that it can initially learn to go in a straight lane. However, we want to be careful of not introduce bias for straight driving because ultimately, we also want our model to recognize frames which represent roads with curves and know what to do in those situations. We try and convert the video into images by sampling it at a small rate of approximately 10-12 frames per second.

5.6 Augmentation

The next and most important part of the architecture of a CNN model is the fact that we try and train our model in such a way that we add artificial shifts and rotations into it. This is done so that our model knows what to do in a situation when it has landed up in a poor position and now wants to recover from that poor orientation. This is done by randomly choosing data from a normal distribution and then teaching the system what to do. The randomization part is kept so that the system, with more time, data, and experience learns on its own, what to do when.

5.7 Simulation

Simulation is showing how the machine learning model will control the car and what action it will take for that particular frame of image. In order to make complete sense of this, a video of CNN

controlling the vehicle is overlaid on top of the original video of the road. Doing so enables us to compare the data of actual steering rotation values with the ones obtained using CNN model and then conclude about how good or bad was our model in terms of accurately predicting the results.

5.8 Evaluation

Evaluation of a model is a two step process. Once, the model is evaluated using simulation and it is done till we get 100 % accuracy and 0 Loss value. This is a very major requirement as a mistake of even 0.1 % Loss value can be extremely lethal in real life conditions. Hence, only when a model is getting excellent values in simulation, it is passed on for performing a road test on an empty road.

In order to measure the results of a road test of a model, a value called as Autonomy is calculated. It is the amount of human interventions which were required during the whole process. The formula for calculating it is as given below:

$$\text{autonomy} = \left(1 - \frac{\# \text{ of interventions} \cdot 6[\text{seconds}]}{\text{elapsed time}[\text{seconds}]}\right) \cdot 100$$

If the values is found to low, then the model is trained again using even more data and then the whole 2 step process is done once again.

VI. Current Research and Implementations of Autonomous Cars

The research and development in the field of autonomous cars has come a long way since its inception. Today, there are multiple ways of obtaining an autonomous car, depending upon what level of autonomusness one needs in their car. Different companies are trying to implement different logics. While some are using a wide variety of sensors, Cameras, Lidars and Radars for providing quality data about the surroundings of a vehicle, some are only relying on in-car sensor data and Cameras located outside the car to make it autonomous. Companies such as Google can be seen as taking the former approach with their Waymo offering whereas companies such as TESLA can be seen as following the latter approach with their cars being full autonomous.

Even though it is an obvious logic that these companies would probably never reveal how they have built their car because of leaking their potential plan to their rivals, they surely do want others to make progress in this field, so that the overall development in this field continues forever. Hence every company releases their white paper, in which they simply outline extremely basic details about how they have implemented their logic of building the car and what their vision of an autonomous car looks like. While these papers do not contain any piece of code, it definitely gives you a sense of direction in which you can work and try to obtain something which the authors of the white paper also did.

One such company is Nvidia, who is pretty heavily invested into building of autonomous cars. Owing to the fact that its supremely powerful Tesla GPU is used by autonomous cars such a Google Waymo as well as some of the other competitors, it becomes extremely important for Nvidia to release a white paper, so that more and more companies can get an idea of how they think an

autonomous car can be implemented. This is a move for the overall benefit for Nvidia, because ultimately if more companies will try and build a self-driving car, they would probably be looking for an extremely powerful GPU and hence this whole circle will lead to more people buying into Nvidia Tesla.

Every company listed above is using a deep neural network, which is made from a multiple layer, with each layer being used to train the model for a specific important feature and hence improving the overall performance of the model. Also, every company is trying to use image processing and extract important features from an image so that it can use those features towards building the model.

Since no company has revealed how they have built their models, what is the level of accuracy they are getting for their models or how they are using the sensor data; working on developing an autopilot for a self driving car seems a pretty daunting task with absolutely no information out there in this regard.

Keeping all the possible loopholes in mind and also keeping in mind that I might not have access to multiple sensors, Lidar or RADAR data, I opted to try and implement Nvidia's research paper, while trying better their accuracy of 98%. I strive to achieve 100 % accuracy in steering a car. Another reason for choosing Nvidia's white paper is the fact that their implementation resonates pretty well with what TESLA is doing; depending on only 1 main camera and rest only on the sensor data being received from the car and at a non- commercial level, implementing a setup, which is only dependent on a Camera isn't too difficult.

Hence, it would be a really interesting thesis to try and implement this, so that someone trying to implement the same after me knows exactly if this approach works or it doesn't.

VII. Developing an autonomous model with enhanced steering ability

In order to implement Nvidia's white paper, it is very important to understand the key features which are being talked about in that paper. As a result, I tried to inculcate all those key features in my implementation.

After having a lot of brain storming with my faculty of Artificial Intelligence, Dr. Gayathri Namashivayam and having a period of 6 months to research the literature and do some small experiments to have certain hit and trial values, I decided to implement a model, which has 5 Convolutional layers, with 4 Fully connected layers, having output of every layer being fed as the input into the other layer.

The whole accuracy of the model depends on how well you are able to find values of W and b in the equation:

$$Wp + b = q$$

where:

p is the set of images being fed into the model

q is the resultant amount we need to steer the car (in degrees).

In order to have a realistic representation, I need to perform efficient image processing and should be able to extract key features out of the images. Once I have those key features, I need to highlight the key features which can be used to learn on by the fully connected convoluted layers in the model.

To be able to successfully extract key features of an image, I wanted to acquire knowledge of how to convert an image into greyscale, how to be able to perform a Gaussian Blur on an image, how to detect mask edges in an image, how to detect Canny edges in an image and how hough lines can be converted into lane line output. Let's look at each of the process separately.



Figure 7.1 Original Image which has been fed into the model

7.1 Converting to Greyscale – In order to convert an image to greyscale, we need to downgrade its color gamut from 3 dimensions to a single dimension of grey color, i.e., changing its RGB values to suit the formula $0.21 R + 0.72 G + 0.07 B$.



Figure 7.2 Greyscale converted Image

7.2 Gaussian Blur – Once we obtain a Greyscale image, we apply Gaussian Blur to remove the noise from an image. It is done by taking average values of one pixel in regard to its neighbors based on a Gaussian curve of weights being applied while doing the conversion. As a result, we can eliminate high frequency data from the image and reduce noise component in the image.



Figure 7.3 Gaussian Blurred Image

7.3 Detecting Edges in an Image -

After we have obtained an image which is greyscale and also doesn't have any noise, the next step is to detect all the edges in an image. An edge can be thought of as detecting all the boundaries of objects in the image so that we can then recognize the lane in which we are supposed to drive.

In order to apply the mask edging filter, we simply calculate the pixel values of the left and the right neighbor of a pixel and calculate the pixel gradient. Once we have pixel gradient, we simply set the current pixel to that value. On performing this process on all the pixels one by one, we obtain an output which looks like figure 7.4 shown below.



Figure 7.4 Output after applying Edge Filter

Once we obtain this intermediate result, we then normalize our results to obtain an even more important feature set. This is done by simply selecting only those gradient values which are beyond a threshold value and then setting those values to a white RGB colour and setting the values below it to black RGB colour.

As a result, we will be able to black out even more edges which are not relevant for us in the current image and thus, it will produce an image which looks like that obtained in figure 8.5 below.



Figure 7.5 Final output of Edge detection Phase

7.4 Masking Unnecessary Edges in an Image -

The next step in the whole process is to mask the un-necessary edges using a masking technique and only leave behind the edges which are relevant to the current frame. This has been done in the

project using masking technique given by Erik et. al [7]. The results obtained due to application of the technique are shown in figure 7.6.

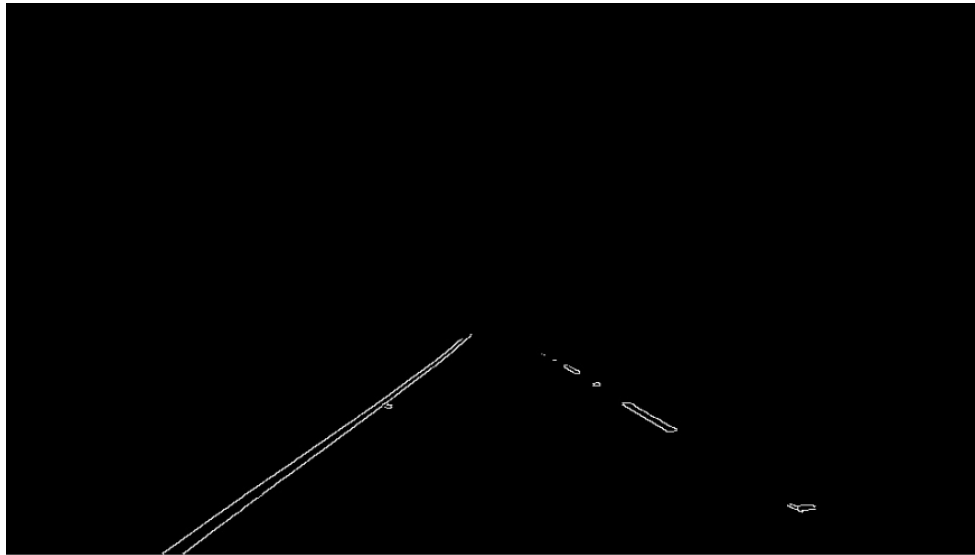


Figure 7.6 Masking all irrelevant edges

7.5 Performing Hough Transformation on Image -

The next important step in the whole process is to perform Hough transformation on the image so obtained. The aim behind doing that is to obtain an image in which we have straight lines instead of broken white-dashed lines. The straight lines will represent the lanes and will give the model an idea about lanes little better. Hence it is a very important aspect of the whole process.

It is done by calculating the slope and intercept of the line for every point in the image and performing hit and trial method for those values. We finally need to select the value which can be

best suited to represent the lanes for the image in question. As a result, we obtain the result which can be seen in Figure 7.7 below.

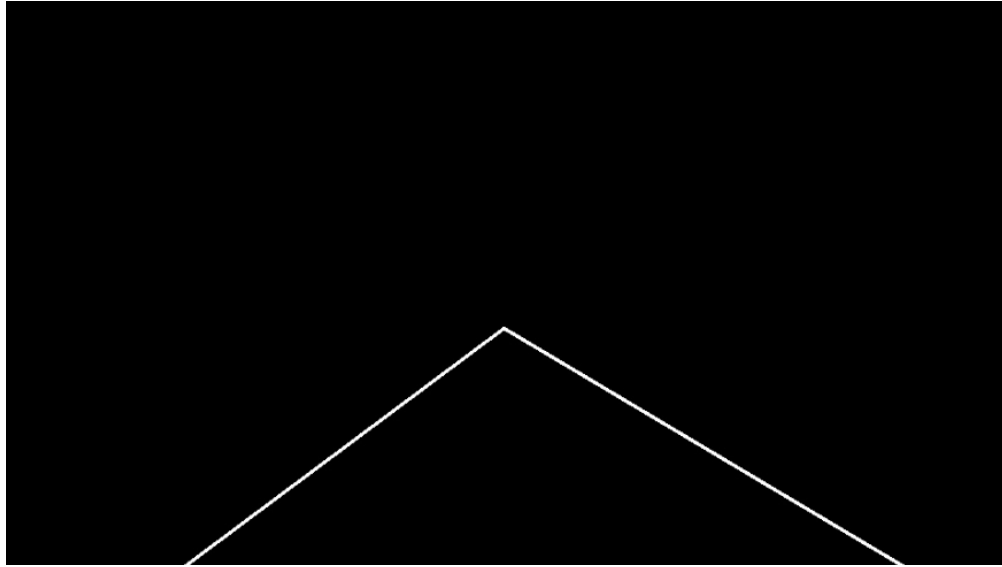


Figure 7.7 Hough Transformation

Figure 7.8 below shows how this obtained Hough Transformation looks on top of the original Greyscale Image, to give a prospect of how through that image, using a couple of filters and algorithms, we could eliminate everything else in the image and obtain only the data which was of prime importance to us.



Figure 7.8 Greyscale Image along with Hough Transformation

VIII. Tools and Datasets

In order to develop a model which behaves like a fully autonomous car and has the capacity to self-steer, break as well as to accelerate, a bunch of programming languages will be required. Python will be used to do the deep learning related programming and C++ will be required to program Arduino board and to perform some low-level programming. Also, a couple of datasets will be needed in order to build the system. All the required tools are explained below:

8.1 Python Libraries

TensorFlow - In order to develop a model which is based on deep learning and has a neural network, we will be required to use Google's Neural network-based library for python called as TensorFlow. It helps in customizing each layer of a multi-layer perceptron and also helps in

achieving the desired model by having each layer work on a different feature of prime importance to us.

Scientific Computing - It is an open source ecosystem of libraries which can be used for mathematical, scientific and technical computing in python. It consists of a couple of packages which are found in-built in python's library and are used to enable a person to perform hard computations with ease. It consists of the following:

- Numpy - It is the most basic package for numerical computation and is mostly used for defining arrays and for performing operations on 2D matrices.
- Scipy Library - It consists of a collection of algorithms which can be used in a lot of domains such as signal processing, statistics, optimization tasks etc.
- Matplotlib - It helps in plotting the results obtained in a 2D/3D manner and hence is a really important tool in terms of being able to visualize the results that one obtains by applying all the algorithms in Scipy library.
- Scikit-image – It consists of a collection of algorithms which can be used for image processing and for extracting useful features from an image. It is one of the most important libraries because having Scikit enables us to have a model with higher accuracy rate.
- Scikit Learn – It is a collection of algorithms and tool which helps us in implementing Machine Learning based algorithms on our dataset. It has a complete guide which explains as to how to use each algorithm using this library and has an extensive set of examples included in the guide.
- Sympy – It is a python library which is used for using symbolic mathematics as well as computer algebra, which is required when dealing with absolutely low-level data such as data

being produced by a steering wheel of a car as well as by the breaks and accelerators every Nano second.

Random - It is an in-built library in python which helps in obtaining random numbers. It behaves similar to what a rand() function does in Java/C++.

CV2 - It is a python library which is meant for real time computer vision related computation and can be used to perform calculations in tandem with Scipy library. It helps in enhancing the abilities of python by making it perform computing on data coming in at real time.

OS - It is an in-built library which helps in enabling a us to use Operating system dependent functionality in our programs. It is OS independent and allows to use similar functions and interface on Windows, Linux as well as Mac.

8.2 C++ Header Files

We will need the usual C++ header files such as vector, string, memory, thread, utility, stdio, algorithm, fcntl, stdint, errno to name a few.

8.3 CAFFE

It is an open source deep learning framework written in C++. It is open source, under a BSD license. It is written in C++, with a Python interface

8.4 Dataset

The accuracy of a model largely depends on the type and quality of dataset you feed into it. Hence a good quality of time was spent in this project in searching for datasets which are of extremely high quality. Also, realizing what kind of data is required is also very important for the success of a model. Since having a model give steering instructions to a car involves it seeing a real-world image, we need to feed it image data. Also, since the real-world scenario is like a continuous video, hence we need images which are taken by dividing the video at certain continuous intervals. One of the most important factors while having such a data becomes the fact that the data should be labelled and should have clear description of what all data was achieved during that particular frame of moment. Also, having such data in bulk is another important factor because the more data we feed a model, the more accurate it becomes with time. There are 2 such datasets of images which were found with rigorous efforts, both of which have been used for this project. The datasets are explained below in detail:

8.4.1 Udacity Dataset - Udacity has a dataset of more than 230 GB of images. The images are obtained as a result of dividing a footage of 10+ hours into frames. The images which are a part of the dataset are of extremely high quality. They also labelled the dataset with all the sensor data at the instance of that image such as steering angle, the acceleration, the speed, the breaking etc. Due to the large nature of dataset, Udacity hosts the dataset on its cloud server and gives a torrent link to download the dataset on our machines. Udacity also uses this dataset in order to provide the nano degree and have people train and code a model which is a simulator of a fully autonomous car.

Driving Data

Date	Lighting Conditions	Duration	Compressed Size	Uncompressed	Direct Download	Torrent
09/29/2016	Sunny	00:12:40	25G	40G	HTTP	Torrent
10/03/2016	Overcast	00:58:53	124G	183G	HTTP	Torrent
10/10/2016	Sunny	03:20:02	21G	23.3G		Torrent
10/20/2016	Sunny	03:30:00	30G	40G		Torrent

Figure 8.1: Udacity's Dataset Link on their GitHub Page

8.4.2 Self-Created Dataset - While searching for high quality datasets, I could not find a lot of high-quality datasets being easily available and it can be due to the fact that Autonomous Cars is a hot topic and companies don't want to give away their secret ingredients which they might be using in their implementation. As a result, in order to make sure that the model which was being built was free of bias, I was pushed against the wall to build my own dataset. In order to have my dataset, I would have required two things, Images and steering values to go with the images. The images part was easy. But getting steering values was where a lot of research was required. After a lot of efforts, I concluded that sensors in a car produce a lot of data, which is consistently checked for errors in the car. This data can be extracted using the debugging plug in a car. Hence, I used an OBD cable and attached it to the can Bus on an Arduino Shield board to extract the data from the debugging plug of the car. I used the fact that we can get a car's data through its debugging plug to my benefit. Since the data I obtained was coded by some serial number and did not have data in terms of the degree by which the steering was being turned, I came across multiple implementations of methods to convert this data into degrees by which the steering of a car is turned.

0	CAN BUS Shield init ok!								
1	86	7	108	255	160	7	4		
2	234	0	0	0	0	0	0	0	45
3	142	8	228	52					
4	142	8	222	44					
5	60	255	234	255	232	0	0	4	40
6	145	1	0	0	161	161	0	127	31
7	145	128	0	8	14	3	0	0	43
8	155	159	53	238	91	37			
9	60	255	232	255	232	0	0	4	57
10	145	1	0	0	161	161	0	127	46

Figure 8.2: Raw Data Obtained from Arduino Shield Card

The next thing which was required was to capture a video while capturing the car data so that I can work using this combination of data. I shot a video of roads around Santa Clara by using a duct tape on the wind shield of my friend's car and then recorded a video of the roads. While doing that, I also captured the values being generated by the Arduino board. The length of the video captured was roughly equal to 25 minutes. In the end, I split the video frame by frame into 63,824 images and named each one as 1, 2, ... 63824 by using a python script. Then I clubbed the image name with the data for that instance generated by the car so that I can easily use that data later on.

1	0.jpg	0.000000,2019-01-01	17:09:44:912
2	1.jpg	0.000000,2019-01-01	17:09:44:972
3	2.jpg	0.000000,2019-07-01	17:09:45:11
4	3.jpg	0.000000,2019-01-01	17:09:45:76
5	4.jpg	0.000000,2019-01-01	17:09:45:105
6	5.jpg	0.000000,2019-01-01	17:09:45:145
7	6.jpg	0.000000,2019-01-01	17:09:45:205
8	7.jpg	0.000000,2019-01-01	17:09:45:246
9	8.jpg	0.000000,2019-01-01	17:09:45:301
10	9.jpg	0.000000,2019-01-01	17:09:45:341
11	10.jpg	0.000000,2019-01-01	17:09:45:407
12	11.jpg	0.000000,2019-01-01	17:09:45:467
13	12.jpg	0.000000,2019-01-01	17:09:45:507
14	13.jpg	0.000000,2019-01-01	17:09:45:537
15	14.jpg	0.000000,2019-01-01	17:09:45:575
16	15.jpg	0.000000,2019-01-01	17:09:45:634
17	16.jpg	0.000000,2019-01-01	17:09:45:672
18	17.jpg	0.000000,2019-01-01	17:09:45:732
19	18.jpg	0.000000,2019-01-01	17:09:45:770
20	19.jpg	0.000000,2019-01-01	17:09:45:839
21	20.jpg	0.000000,2019-01-01	17:09:45:867
22	21.jpg	0.000000,2019-01-01	17:09:45:936
23	22.jpg	0.000000,2019-01-01	17:09:45:966
24	23.jpg	0.000000,2019-01-01	17:09:46:37
25	24.jpg	0.000000,2019-01-01	17:09:46:67
26	25.jpg	0.000000,2019-01-01	17:09:46:137
27	26.jpg	0.000000,2019-01-01	17:09:46:166
28	27.jpg	0.000000,2019-01-01	17:09:46:233
29	28.jpg	0.000000,2019-01-01	17:09:46:293
30	29.jpg	0.000000,2019-01-01	17:09:46:330
31	30.jpg	0.000000,2019-01-01	17:09:46:398
32	31.jpg	0.000000,2019-01-01	17:09:46:426
33	32.jpg	-3.530000,2019-01-01	17:09:46:464
34	33.jpg	-3.930000,2019-01-01	17:09:46:493
35	34.jpg	-5.240000,2019-01-01	17:09:46:564
36	35.jpg	-5.340000,2019-01-01	17:09:46:621
37	36.jpg	-4.840000,2019-01-01	17:09:46:661

Figure 8.3: Sample Snippet of how data looked



Figure 8.4: A Sample of how the dataset looked

It is important to have more than a single type of dataset as you can then check your model against different kinds of datasets and be absolutely sure about its accuracy in different situations as compared to ones in which you have trained the model.

IX. Experiments and Results

Before beginning the experiments and analyzing the results being obtained as a result of running the datasets, it is really important to pre-process the datasets which we have to work with to make sure that we use the dataset to the maximum effect and get the results which we should actually be obtaining. This was done for both the datasets as explained in the section below.

9.1 Pre-processing the data:

9.1.1 For Self-Created Dataset: A large chunk of pre-processing for this dataset was done while creating this dataset. The only bit of pre-processing which was done later was the fact that the data being produced by the Can Bus had to be clubbed with the image number for which it was relevant. This was done by using python scripts for automating the process of joining both the fields in a new text file and removing everything else from the data. The data sheet looked as the shown below after pre-processing the data.

1	0.jpg	0.000000
2	1.jpg	0.000000
3	2.jpg	0.000000
4	3.jpg	0.000000
5	4.jpg	0.000000
6	5.jpg	0.000000
7	6.jpg	0.000000
8	7.jpg	0.000000
9	8.jpg	0.000000
10	9.jpg	0.000000
11	10.jpg	0.000000
12	11.jpg	0.000000
13	12.jpg	0.000000
14	13.jpg	0.000000
15	14.jpg	0.000000
16	15.jpg	0.000000
17	16.jpg	0.000000
18	17.jpg	0.000000
19	18.jpg	0.000000
20	19.jpg	0.000000
21	20.jpg	0.000000
22	21.jpg	0.000000
23	22.jpg	0.000000
24	23.jpg	0.000000
25	24.jpg	0.000000
26	25.jpg	0.000000
27	26.jpg	0.000000
28	27.jpg	0.000000
29	28.jpg	0.000000
30	29.jpg	0.000000
31	30.jpg	0.000000
32	31.jpg	0.000000
33	32.jpg	-3.530000
34	33.jpg	-3.930000
35	34.jpg	-5.240000
36	35.jpg	-5.340000
37	36.jpg	-4.840000

Figure 9.1 Self-created Dataset sheet after pre-processing

9.1.2 For Udacity's Dataset: The pre-processing part in this dataset was quite large as the dataset had photos which were named by their date and time stamps. As a result, it was required to rename them in such a way that the first Image is called as 1, 2 and so on. Also, it was required to rename the data file, wherein, all the data against the photo name, gets renamed to the new name of that photo. Since the size of the dataset was 230 GB, even using automated python scripts to perform the change, it took more than 2 hours to perform the same.

B	C	D	E	F
			steering_angle	
0	0.jpg		-0.3736651061	0.jpg -0.373665106110275
1	1.jpg		-0.06539628841	1.jpg -0.0653962884098291
2	2.jpg		-0.1607354414	2.jpg -0.160735441371799
3	3.jpg		0.3178960579	3.jpg 0.317896057851613
4	4.jpg		0.196493084	4.jpg 0.19649308398366
5	5.jpg		-0.3413863465	5.jpg -0.341386346518993
6	6.jpg		-0.02395744976	6.jpg -0.0239574497565627
7	7.jpg		-0.2538138354	7.jpg -0.253813835419714
8	8.jpg		0.3020051625	8.jpg 0.302005162462592
9	9.jpg		-0.3826719603	9.jpg -0.382671960256994
10	10.jpg		-0.04453581125	10.jpg -0.0445358112454414
11	11.jpg		0.2953192173	11.jpg 0.295319217257202
12	12.jpg		-0.1876285043	12.jpg -0.187628504261374
13	13.jpg		-0.2350884967	13.jpg -0.235088496655226
14	14.jpg		-0.3244785026	14.jpg -0.324478502571583
15	15.jpg		-0.2597559279	15.jpg -0.259755927883089
16	16.jpg		-0.1359966669	16.jpg -0.135996666923165
17	17.jpg		-0.2464172781	17.jpg -0.246417278051376
18	18.jpg		-0.004179532453	18.jpg -0.00417953245341779
19	19.jpg		-0.07171424292	19.jpg -0.0717142429202795
20	20.jpg		-0.0109193271	20.jpg -0.0109193271026015
21	21.jpg		-0.2904880041	21.jpg -0.290488004125655
22	22.jpg		0.2807980975	22.jpg 0.280798097513616
23	23.jpg		-0.007306088321	23.jpg -0.00730608832091095
24	24.jpg		0.05400142949	24.jpg 0.054001429490745
25	25.jpg		-0.03611099049	25.jpg -0.0361109904944897
26	26.jpg		0.09188717287	26.jpg 0.091887172870338

Figure 9.2 Udacity Data sheet after renaming the images

After doing so, the data in F column was copied in a new text file and separated with a white space using Python script. As a result, the final dataset sheet obtained was as shown below.

0.jpg	-0.373665106110275
1.jpg	-0.0653962884098291
2.jpg	-0.160735441371799
3.jpg	0.317896057851613
4.jpg	0.19649308398366
5.jpg	-0.341386346518993
6.jpg	-0.0239574497565627
7.jpg	-0.253813835419714
8.jpg	0.302005162462592
9.jpg	-0.382671960256994
10.jpg	-0.0445358112454414
11.jpg	0.295319217257202
12.jpg	-0.187628504261374
13.jpg	-0.235088496655226
14.jpg	-0.324478502571583
15.jpg	-0.259755927883089
16.jpg	-0.135996666923165
17.jpg	-0.246417278051376
18.jpg	-0.00417953245341779
19.jpg	-0.0717142429202795
20.jpg	-0.0109193271026015
21.jpg	-0.290488004125655
22.jpg	0.280798097513616
23.jpg	-0.00730608832091095
24.jpg	0.054001429490745

Figure 9.3 Data sheet for Udacity's Dataset

9.2 Experiments:

To gauge the accuracy of the model, it was trained and tested using a varied combination of training and testing data. The results obtained have been explained along with the different datasets taken in the respective experiments are listed below:

9.2.1 Training and Testing on Self-Created Dataset:

Firstly, the model was trained using self-created dataset. It was run for 30 epochs, with each epoch consisting of more than a few dozen of steps.

```
Please switch to tf.summary.FileWriter. The interface and behavior is the same;
this is just a rename.
Epoch: 0, Step: 0, Loss: 6.7098
WARNING:tensorflow:*****
WARNING:tensorflow:TensorFlow's V1 checkpoint format has been deprecated.
WARNING:tensorflow:Consider switching to the more efficient V2 format:
WARNING:tensorflow:  `tf.train.Saver(write_version=tf.train.SaverDef.V2)`
WARNING:tensorflow:now on by default.
WARNING:tensorflow:*****
Epoch: 0, Step: 10, Loss: 6.43693
Epoch: 0, Step: 20, Loss: 6.1716
Epoch: 0, Step: 30, Loss: 6.41206
Epoch: 0, Step: 40, Loss: 6.23232
Epoch: 0, Step: 50, Loss: 6.08394
Epoch: 0, Step: 60, Loss: 6.18194
Epoch: 0, Step: 70, Loss: 5.97316
Epoch: 0, Step: 80, Loss: 6.04343
Epoch: 0, Step: 90, Loss: 5.9292
Epoch: 0, Step: 100, Loss: 5.87816
WARNING:tensorflow:*****
WARNING:tensorflow:TensorFlow's V1 checkpoint format has been deprecated.
WARNING:tensorflow:Consider switching to the more efficient V2 format:
WARNING:tensorflow:  `tf.train.Saver(write_version=tf.train.SaverDef.V2)`
WARNING:tensorflow:now on by default.
```

Figure 9.4 Running 30 Epochs on Self-created Dataset

Loss defines the difference between the Steering angle which was generated by the Model – the angle with which steering was actually steered. Hence the whole process started with having 6.43693 degree of loss values, which was extremely large. The whole training process took around 9 hours to fully train the model on this dataset. While running the training dataset, Tensorboard was used to obtain a plot of how the “Loss” value is getting affected as we go through steps in our

training dataset. The plot so obtained, with X axis showing the Steps taken and Y axis showing the Loss value over time is shown below.

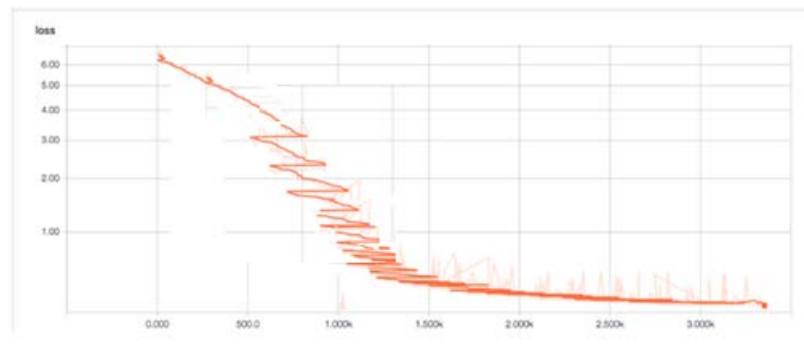


Figure 9.5 Loss vs Step Plot for 30 after Training Model on complete dataset

From the plot it is clearly visible that while it started with a Loss value of around 6.43693, after taking more than 3 thousand steps, the value reduced drastically to 0.1211, which is very close to 0.

When the model was tested against a subset of the same dataset, it showed almost 0 error rate. But, to check the model for overfitting the data points being fed into its system, a few different combinations were also fed into the same model.

9.2.2 Training and Testing on a small subset of Udacity Dataset:

In this experiment, the model which was trained in the previous experiment was tested against the dataset which was made available by Udacity. This was done in order to check

how the model behaves when unknown and unseen data is thrown against it. The results obtained were as shown below in figure 9.6.

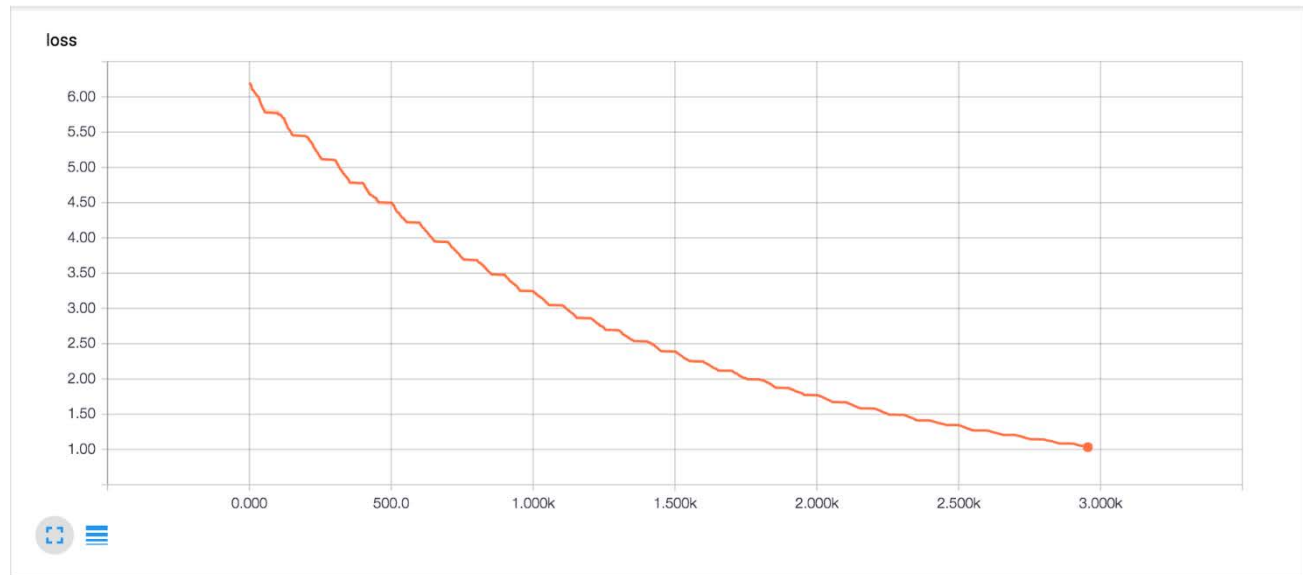


Figure 9.6 Loss vs Steps plot for subset of Udacity's dataset

As the Loss versus steps graph shows, the model performed pretty bad on this dataset. Even after training the model through all the examples, it showed a Loss of 1degree, which is very big. Hence it was required to do some analysis of these results in-order to correct them and have a model which has a more generalized behavior. Thus, two reasons were identified for such a behavior of the model, which are listed as follows:

1. The dataset created by Udacity and the dataset created by me are completely different as Udacity's Video was a bit more zoomed in as compared to mine. Since I used a normal webcam whereas Udacity might have used a full-blown autonomous car Camera, hence the model, which was trained on Zoomed out images, was bound to perform bad on zoomed in images.

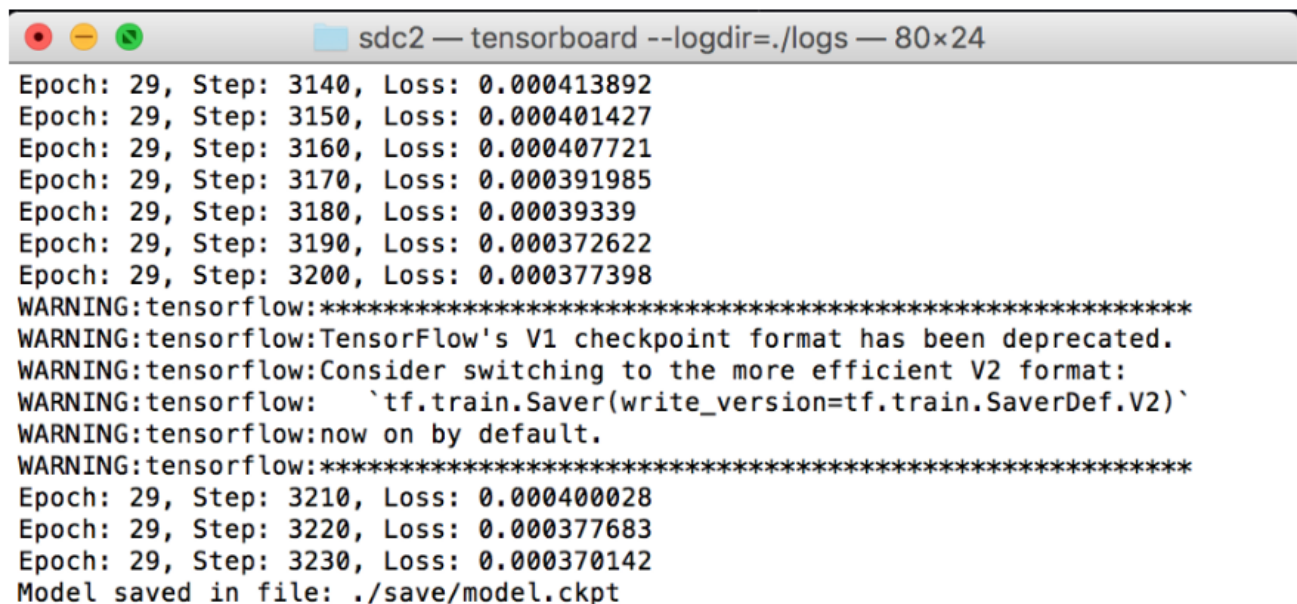
2. The size of the dataset used in this experiment was appreciably smaller than anything which was fed to the system before. It was around 9 times smaller than the data set which I had created. Hence due to lack of data points, it might have given the results which it gave. As a result, a newer experiment was required with all of the data of 230 GB images.

9.2.3 Testing and Training on complete Udacity's Dataset:

The model was trained against the huge dataset provided by Udacity and the following results were obtained when the model was trained against 30 epochs, with each epoch containing more than a few dozens of steps.

Starting Loss Value: 6.14783

Ending Loss Value: 0.000370



```
sdc2 — tensorboard --logdir=./logs — 80x24
Epoch: 29, Step: 3140, Loss: 0.000413892
Epoch: 29, Step: 3150, Loss: 0.000401427
Epoch: 29, Step: 3160, Loss: 0.000407721
Epoch: 29, Step: 3170, Loss: 0.000391985
Epoch: 29, Step: 3180, Loss: 0.00039339
Epoch: 29, Step: 3190, Loss: 0.000372622
Epoch: 29, Step: 3200, Loss: 0.000377398
WARNING:tensorflow:*****
WARNING:tensorflow:TensorFlow's V1 checkpoint format has been deprecated.
WARNING:tensorflow:Consider switching to the more efficient V2 format:
WARNING:tensorflow:  `tf.train.Saver(write_version=tf.train.SaverDef.V2)`
WARNING:tensorflow:now on by default.
WARNING:tensorflow:*****
Epoch: 29, Step: 3210, Loss: 0.000400028
Epoch: 29, Step: 3220, Loss: 0.000377683
Epoch: 29, Step: 3230, Loss: 0.000370142
Model saved in file: ./save/model.ckpt
```

Figure 9.9 Final values obtained after completing 30 epochs

The graph of Loss versus steps obtained for this experiment was as shown below in figure 9.10.

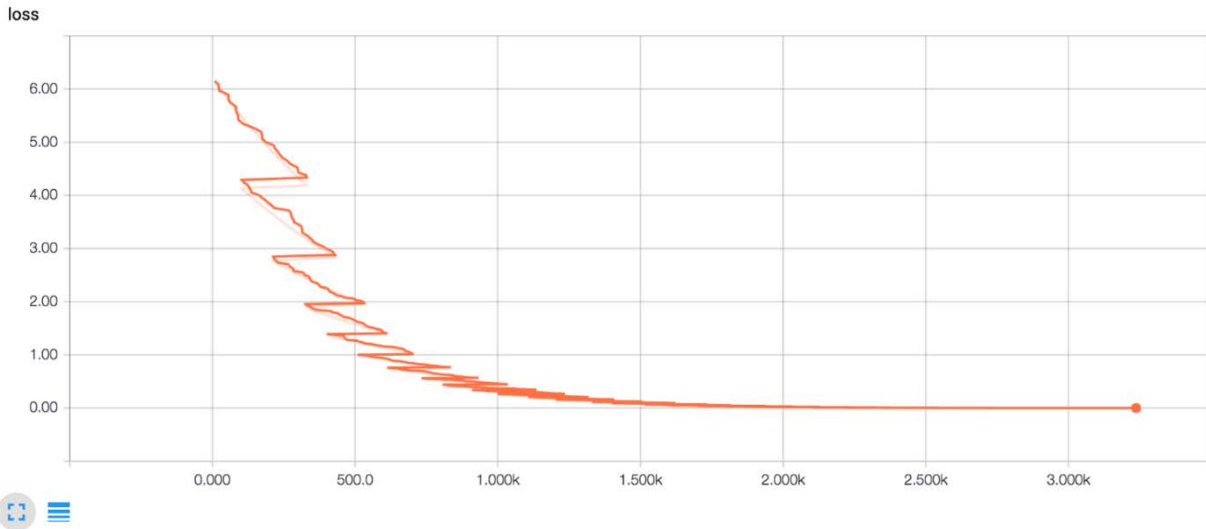


Figure 9.10 Loss versus Steps Graph for Complete Udacity Dataset

Since having 9000 data points in the previous experiment generated a model which gave a Loss value of approximately 1 whereas on increasing the data points to more than 200000, the error rate dropped to even less than 0.000. This implies that such a model can be made more and more accurate with time, by feeding more data into it. Once, it has been trained on a large amount of data, it should be able to handle similar images with ease, with a Loss rate of approximately 0 (100% accuracy).

9.2.4 Testing again on Self-created dataset:

On testing the model on the images of self-created dataset, I could achieve ~100% accuracy, with just 1 frame going wrong out of the 63 thousand. I had a steering wheel image and made it rotate on the basis of the amount of angle the model wants the steering to steer. I rendered this into a full video and overlaid this on top of the original video of my self-created dataset. As a result, we could clearly see that due to training the model on similar dataset as well as a dataset with more than 2 hundred thousand data points, it resulted in a model which is highly accurate. A single screenshot of what the whole video looked like is shown below in figure 9.11.



Figure 9.11 Screenshot of overlaid video of model-controlled steering on top of self-created dataset video

Hence, it can be concluded that the model so obtained is extremely accurate. Also, since the data being generated by a camera will be similar in terms of the amount of zoom and the type of images being captured, hence the concept overfitting would not apply that much on this model.

9.3 Additional Experiments:

For Acceleration:

A similar experiment can be performed on the acceleration data in order to check if the model is a generalized one and can perform on any data being fed into it.

In order to do the same, we need to pre-process the data by making a text file consisting of all the JGP names and along with those, the acceleration values for those particular frames. The same can be obtained using python scripts and in the exact similar manner as it was done for Steering control data.

On performing similar experiments with the acceleration data, it was observed that it also follows the same pattern, with data initially giving a very high loss value of 6.3 but after 30 epochs and training over massive dataset of 230 GB data, it got reduced to a minor value of 0.3. Even though this was not included in the scope of this project, a small sample experiment was performed to check if the model so built was a generalized one and, if on changing the type of data, it behaves in the same manner, then the model can be deemed as being highly generalized. Since model did perform really well on predicting the acceleration for future images on being trained on some of the images, and its accuracy went on increasing with more data and time, shows that the mode is Generalized.

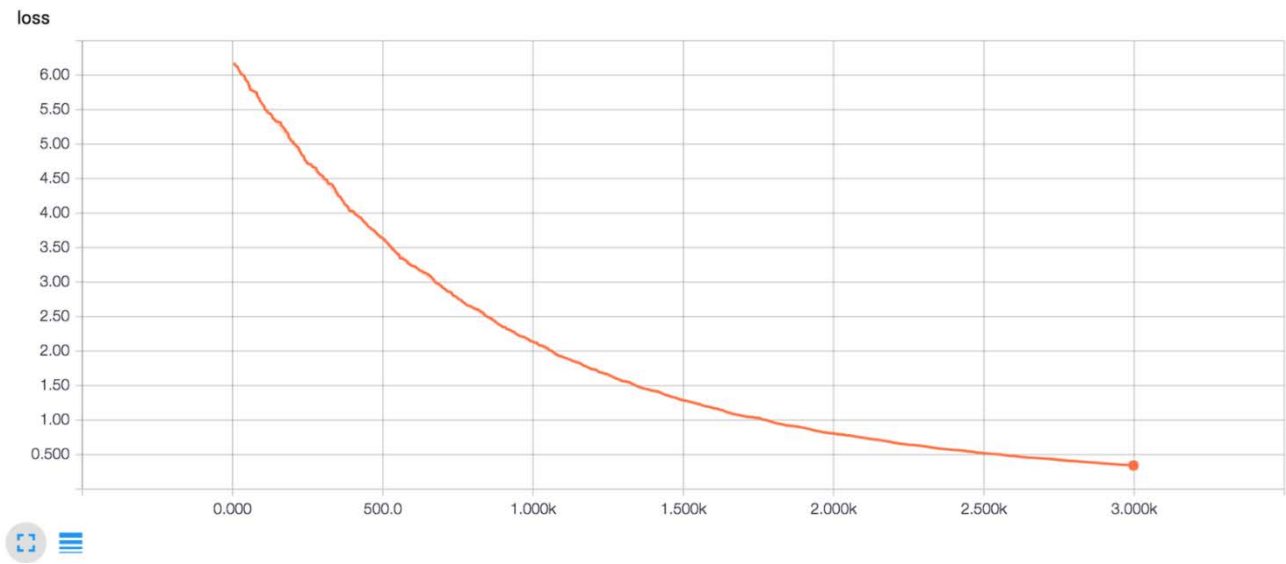


Figure 9.13 Loss vs Steps Curve for Acceleration data in Udacity Dataset

10. CONCLUSION

From the experiments performed above and the results obtained, we can clearly see that a model tends to become more and more accurate along with time. Also, the more data we feed to a model, the higher is its accuracy rate in predicting the correct steering angle for the steering wheel in the car. Also, since the model resulted in approximately 100% accuracy, we can safely assume that our feature extraction logic worked pretty accurately.

The results also showed that the angle and amount of zoom with which the camera is processing the photos also affects the results we obtain in a large manner. A model which is trained on images which are zoomed out, might not fair very well in a situation in which a camera is presenting to it images which are zoomed in, as it was not trained to handle that kind of data. Also, it can be concluded that, a model can be trained on multiple types of data, like in this case, the model was trained on zoomed in as well as zoomed out images. The model, along with time, becomes extremely proficient in guessing the steering angle for the input image. However, in an ideal situation such a thing is not desired, because we will only have a camera produce a particular kind of data, either zoomed in or zoomed out and that would not change over time. Thus, we can make our model suitable for a specific kind of data and image type and it should not be a problem.

Also, it can be seen that the model thus formed is a generalized model and is applicable on acceleration data as well as braking data. Our results clearly show that as we keep on feeding the images and acceleration data into the model, it keeps on learning how to predict

the acceleration values and gets extremely better once we are done with training the model with large amount of training data.

Lastly, it can be concluded that the most important aspect in the whole process is the model and how accurately the neural network processes the features which you have extracted. Hence, this means that the feature extraction process also holds a very key role in the whole process because extracting poor features can lead to convolutional layers getting trained on bad data and then predicting bad steering angles. Thus, the most important step is to pre-process the images and extract useful data out of every image so that we train the layers only on relevant data. Also, using correct values of weights and bias is also very critical. Taking cues from the research papers read, I was able to come to some figures which would have worked for my model. Hence, it is really important to thoroughly cover the previous research, so that we don't end up wasting time selecting poor weight and bias values.

In the end it can be concluded that making a fully autonomous system which can be fitted inside a car is not a tedious process. The tedious process is that of training the model with relevant data and having correct feature extraction algorithms in place, along with perfect weight and bias values; all of which can then be put to work towards training our deep neural network which will end up guessing how to drive a car on its own with perfection.

11. FUTURE WORKS

From the above experiments, it is clear that if we carefully pick the number of convoluted layers, and apply correct weights, then we can get approximately 100% accuracy. Since the advancement in the field of Machine Learning keeps on changing every day, hence in future works, a different deep learning model can also be tried on the same data, to see if it results in exact 100% accuracy. Also, there are several different kinds of values which can be taken for weight and bias in the convolutional layer. Trying out experiments with different values can also potentially lead to better results. The image processing technique used in this project can also be improvised in the future. As a result, we might get a more accurate features set to highlight in our convolutional layers, thus increasing the accuracy of prediction of our model.

Besides this, one can also try a different pattern of the image processing and object recognition technique, in which probably Gaussian Blur is done before converting an image into greyscale, thus leading to a different variety of results altogether.

Performing experiments on all the above-mentioned models using different kinds of datasets can give results which will be really interesting to analyze. Another part of the future work can be having more datasets to work with because currently, not many datasets are available to work on such kind of a project as every company wants to keep their efforts, logic, formula and data as a secret before their product becomes a success. Thus, having to test our model on wide variety of data can also be a thing to look forward to in the future.

REFERENCES

- [1] M. Daily, S. Medasani, R. Behringer and M. Trivedi, "Self-Driving Cars," in *Computer*, vol. 50, no. 12, pp. 18-23, December 2017.
- [2] S. Urooj, I. Feroz and N. Ahmad, "Systematic literature review on user interfaces of autonomous cars: Liabilities and responsibilities," *2018 International Conference on Advancements in Computational Sciences (ICACS)*, Lahore, 2018, pp. 1-10.
- [3] C. Pozna and C. Antonya, "Issues about autonomous cars," *2016 IEEE 11th International Symposium on Applied Computational Intelligence and Informatics (SACI)*, Timisoara, 2016, pp. 13-18.
- [4] K. Bimbraw, "Autonomous cars: Past, present and future a review of the developments in the last century, the present scenario and the expected future of autonomous vehicle technology," *2015 12th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, Colmar, 2015, pp. 191-198.
- [5] Carlos Reaño and Federico Silla, "Performance Evaluation of the NVIDIA Pascal GPU Architecture: Early Experiences", 2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems.

- [6] S. Tariq, Hyunsoo Choi, C. M. Wasig and Heemin Park, "Controlled parking for self-driving cars," *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Budapest, 2016, pp. 001861-001865.
- [7] Erik Lindholm, John Nickolls, Stuart Oberman and John Montrym "NVIDIA Tesla: A Unified Graphics and Computing Architecture", *IEEE Micro* Volume: 28, Issue: 2, March-April 2008.
- [8] T. Banerjee, S. Bose, A. Chakraborty, T. Samadder, B. Kumar and T. K. Rana, "Self driving cars: A peep into the future," *2017 8th Annual Industrial Automation and Electromechanical Engineering Conference (IEMECON)*, Bangkok, 2017, pp. 33-38.
- [9] Kelly Heather, "Self-Driving Cars now Legal in California", *CNN*. Retrieved 11 October 2013, October 2012.
- [10] D. Bajpayee and J. Mathur, "A comparative study about autonomous vehicle," *2015 International Conference on Innovations in Information, Embedded and Communication Systems (ICIECS)*, Coimbatore, 2015, pp. 1-6.
- [11] G. Lugano, "Virtual assistants and self-driving cars," *2017 15th International Conference on ITS Telecommunications (ITST)*, Warsaw, 2017, pp. 1-5.
- [12] M. V. Rajasekhar and A. K. Jaswal, "Autonomous vehicles: The future of automobiles," *2015 IEEE International Transportation Electrification Conference (ITEC)*, Chennai, 2015, pp.1-6.
- [13] <https://spectrum.ieee.org/cars-that-think/transportation/self-driving/have-selfdriving-cars-stopped-getting-better>
- [14] <https://spectrum.ieee.org/cars-that-think/transportation/self-driving/mit-experts-selfdriving-cars-wont-need-accurate-digital-maps>
- [15] <https://spectrum.ieee.org/transportation/self-driving/creating-driving-tests-for-selfdriving-cars>

- [16] <http://www.nvidia.com/object/volta-architecture-whitepaper.html>
- [17] <https://cacm.acm.org/news/229872-rethinking-autonomous-vehicles/fulltext>
- [18] <https://cacm.acm.org/magazines/2015/8/189836-the-moral-challenges-of-driverless-cars/abstract>
- [19] <https://www.ieee.org/about/news/2012/5september-2-2012.html>
- [20] <https://spectrum.ieee.org/cars-that-think/transportation/self-driving/exposing-the-power-vampires-in-self-driving-cars>
- [21] <https://spectrum.ieee.org/transportation/self-driving>
- [22] <https://spectrum.ieee.org/transportation/self-driving/the-big-problem-with-selfdriving-cars-is-people>
- [23] S. Kumar, L. Shi, S. Gil, N. Ahmed, D. Katabi, and Daniela, "CarSpeak: A Content-Centric Network for Autonomous Driving," in ACM SIGCOMM, Aug. 2012.
- [24] Policy on Automated Vehicles, NHTSA,
http://www.nhtsa.gov/staticfiles/rulemaking/pdf/Automated_Vehicles_Policy.pdf
- [25] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S.Kammel, J. Kolter, D. Langer, O. Pink, V. Pratt, M.Sokolsky, G. Stanek, D. Stavens, A. Teichman, M. Werling, and S. Thrun. Towards fully autonomous driving: Systems and algorithms. In Proceedings of IEEE Intelligent Vehicles Symposium 2011
- [26] A. Teichman, J. Levinson, and S. Thrun, "Towards 3d object recognition via classification of arbitrary object tracks," in 2011 IEEE International Conference on Robotics and Automation. IEEE, May 2011
- [27] <https://www.techrepublic.com/article/autonomous-driving-levels-0-to-5-understanding-the-differences>

[28] Nvidia blog post on Xavier,<https://blogs.nvidia.com/blog/2016/09/28/gtc-europe-keynote>

[29] Nvidia, “End to End Deep Learning In Autonomous Vehicle”, Nvidia Blog