

**Department of Computer Science & Engineering**

Independent University Bangladesh

The Self Driven Car

**Submitted By**

**Madiha Mohammad Ishaque(1710866)**

**Foysal Mahmud Hasan (1730150)**

**Roza Afrin Jamil (1730343)**

**Supervised By**

**Prof. Farruk Ahmed**

**Professor,**

**Dept. of CSE, IU**

**Autumn 2021**

**Abstract:**

The concept of self-driving cars, also known as autonomous cars, evolved from seeing accidents caused by people's reckless driving, which can be exceedingly dangerous at times. The most inconvenient aspect of the commute is traffic. Dense and congested regions might be better handled with self-driving cars that faithfully observe traffic laws. Hence, in this project, a simple self-driving car is created and is implemented by constructing a highly optimized self-driving car that runs only on Deep Learning by the use of Machine Learning and Artificial Intelligence. We will explore an autonomous robotically managed driving car in this study.

Many elements, such as mapping, tracking, and local planning, are used in our project.We can build a car that can show appropriate lane changes, parking, and U-turns all by itself. Obstacle and curb identification methods, a road vehicle tracker, and checking diverse traffic circumstances are some of the different innovations we are using. This will result in a reliable autonomous self-driving vehicle. It will demonstrate correct parking allocation, lane changes, and automated U-turns with ease. We can accomplish this with the car tracker and the obstruction and different curb detecting methods.

In terms of performance, three designs are compared.Through the models performed admirably on the track where they were taught, the major problem was to generalize this behavior to a second simulated circuit. The training set for driving the automobile autonomously on Track 2 included abrupt bends, obstacles, hills, and shadows, whereas the dataset for Track 1 was basic with good road conditions. Image processing and other augmentation techniques were utilized to address this issue, allowing for the extraction of as much information and characteristics from the data as feasible. Finally, the vehicle was able to generalize successfully on Track 2. In the future, the initiative aspires to achieve the same precision on real-time data.

**Attestation**

I understand the nature of plagiarism, and we are aware of the University’s policy on this. I certify that this is an original work by our groupmates. However, following internationally accepted academic guidelines of using others' written work and / or software (in the form of code) in my University project is properly cited if used in any part of this work.

Name: Madiha Mohammad Ishaque Name: Foysal Mahmud Hasan

Student ID: 1710866 Student ID : 1730150

Signature: Signature:

Name: Roza Afrin Jamil

Student ID: 1730343

Signature:

Evaluation Committee

Signature:…..…………………………………………………………………………………

Name:…………………………………………………………………………………………

Supervisor:….…………………………………………………………………………………

Signature:……………………………………………………………………………………

Name:…………………………………………………………………………………………

Internal Examiner:…………………………………………………………………………

Signature:……………………………………………………………………………………

Name:…………………………………………………………………………………………

Internal Examiner:…………………………………………………………………………

Signature:..……………………………………………………………………………………

Name:…………………………………………………………………………………………

Convener:.……………………………………………………………………………………

**Acknowledgements**

The authors would like to convey our heartfelt gratitude to Prof. Farruk Ahmed, my project guide, for his unwavering support and encouragement during this endeavor. I owe him a debt of gratitude for being there for me at every turn, sharing his technical expertise and providing essential feedback and suggestions that helped make this project a success.

The authors would like to express our gratitude to Dr. Tarem Ahmed and Mr. Mohammad Noor Nobi for taking the time to guide and review this research.

The authors would also like to express our gratitude to the officials and staff members of the Computer Science department who assisted me during that time.

Last but not least, the authors want to express our gratitude to our family for their unwavering support and confidence in me throughout my master's program.

**Table of Contents**

**Abstract.................................................................................................................................................1**

**Attestation............................................................................................................................................2**

**Evaluation Committee........................................................................................................................3**

**Acknowledgements.............................................................................................................................4**

**Table of Contents................................................................................................................................5**

**List of Figures......................................................................................................................................7**

**1 Introduction............................................................................................................................10  
  
 1.1 Background and Context......................................................................................10  
  
 1.2 Objectives................................................................................................................11  
  
 1.3 Scope and Limitations..........................................................................................13  
  
 1.4 Outline of Thesis..................................................................................................15**

**2. Chapter 2: Real work..............................................................................................................16  
  
 2.1 Material and Method...........................................................................................16  
  
 2.2 Analysis...................................................................................................................17  
  
  
3. Chapter 3: Udacity Simulator and Dataset…………………………………………….....19**

**4. Chapter 4 Technical Chapter…………………………………………………………...….26  
  
 4. 1 Technologies used...........................................................................................26**

**4.2 Network Architecture....................................................................................39**

**4.3 Augmentation and Image pre-processing..................................................45**

**5 Chapter 5: Experimental Configurations...............................................................................50  
  
  
6. Chapter 6: Results…………………………………………………………………………...51**

**7 Conclusion.....................................................................................................................................61**

**8. Future work…………………………………………………………………………………...62  
  
  
9. References…………………………………………………………………………………….63**

**LIST OF FIGURES**

**Figures Page**

**1. Implementation Architecture................................................................................16**

**2. Configuration Screen................................................................................................19**

**3. Controls Configuration…………………………………………………………20**

**4. First Screen.……………………………………………………………………...20**

**5. Trak 1..........................................................................................................................21**

**6. Track 2........................................................................................................................22**

**7. Autonomous mode...................................................................................................23**

**8. Dataset Sample.........................................................................................................24**

**9. Driving\_log.csv.........................................................................................................25**

**10. Finding lanes.............................................................................................................26**

**11. Returned masked images……………………………………………………….29   
  
 12. German traffic signals………………………………………………………….31  
  
 13. Relu probability results...........................................................................................31  
  
 14. Training dataset distributed……………………………………………………32  
   
 15. RGB Image………………………………………………………………………..32  
  
 16. Grayscale Image………………………………………………………………….33  
  
 17. Equalized Image…………………………………………………………...34  
  
 18. Images after processing…………………………………………………..34  
  
 19. Epoch………………………………………………………………………..35**

**20. Accuracy and loss of model………………………………………………36**

**21. LSTM Cell (Long Short-Term Memory)..................................................37**

**22. GRU (Gated Recurrent Unit) Cell……………………………………...38**

**23. TimeDistributed Dense layer…………………………………………...38**

**24. Time-distributed layers…………………………………………………………39**

**25. Model 1.......................................................................................................................40**

**26. Architecture 1.............................................................................................................41**

**27. Model 2........................................................................................................................42**

**28. Architecture 2.............................................................................................................42**

**29. Model 3…………………………………………………………………………..43**

**30. Architecture 3…………………………………………………………………...44**

**31. Flip Image...................................................................................................................45**

**32. Shift Image Vertical..................................................................................................46**

**33. Shift Image Horizontal.............................................................................................46**

**34. Crop image……………………………………………………………………….46**

**35. Crop image(continued…..………………………………………………………47**

**36.**  **Brightness increased…………………………………………………….47**

**37. Brightness increased(continued)......................................................................47**

**38. Random Shadows......................................................................................................48**

**39. Random Blur..............................................................................................................48**

**40. Random Noise...........................................................................................................49**

**41. Loss over epochs (Architecture\_1)........................................................................53**

**42. Loss over epochs (Architecture\_2)........................................................................55**

**43. Loss over epochs (Architecture\_3).....................................................................57**

**44. Loss over epochs (Graph).....................................................................................57**

**45. Track\_2 screenshot...............................................................................................59**

**46. Generalization Rating Graph..............................................................................59**

**CHAPTER ONE: INTRODUCTION**

A self-driving car project's goal is to create a better autonomous driver. The automobile should be capable of driving itself without drifting off the road, accelerating and braking at the appropriate times. This chapter briefly discusses the project's issue statement as well as the higher-level solution strategy that was applied.

**1.1 Background and Context:**

Road accidents have always been one of the most significant and vital issues including deaths and severe damage, it has become a threat to human life for various reasons due to negligence and irresponsibility of the driver. Considering the government data, it is identified that 94% of the crashes have been due to driver behavior. Car crash cases have risen swiftly as pedestrians get hit by cars, it is estimated that on average 3700 people lose their lives due to road accidents on a daily basis. Although it is not possible to get hold of all the occurring road crashes, it can surely be subsided by a higher percentage with the help of self-driven cars, this is where advanced technology comes in.

This project aims to bring in a major change with the help of AI and advanced technology to be an alternative for diminishing the deaths due to car accidents and other threats. Self-driven cars are the best alternative solution if brought in action and can help reduce driver errors as it includes absolutely no human activity. They can be a great help in reducing uncountable and unnecessary deaths by decreasing dangerous driver behaviors which includes minimizing impaired driving, drugged driving, unbelted vehicle users, speeding issues and any other interruption. Apart from lowering the death levels due to car crashes, self-driving cars can make it a dream come true for the people who are physically disabled, making their lives easier and safer. The self-driven car can not only reduce these issues but it can also cut down the costs of car crashes, which includes reduced medical bills, lost work time, and vehicle repairing, as a result less number of crashes will lessen the cost of insurance. A safe and an accordant distance will be maintained between vehicles, thus congestion will be reduced.

Self-driven cars work by creating and perpetuating a map of their environment, combined with a digital GPS map of the area, depending on a wide formation of sensors, like radar, laser beams, high powered cameras, and sonars. Lasers are used to test the environment =using LIDAR(Light detection and ranging). Detecting traffic signals, reading road signs, tracking down other vehicles and look for pedestrians, automated U-turns, these are the few functions of a self driven car, the inputs are received by the software and it is processed, the path is plotted, that info is sent to the actuators of the vehicle which is responsible for controlling the steering, acceleration, and breaking. Obstacle-avoiding algorithms by using cameras, radar, lidar and ultrasonic sensors are used, and predictive modeling will help the software follow trafficking and overcome barriers. The autonomous car can mitigate parking problems, as they have the potential for saving parking space by minimizing space occupied by self-parking. The autonomous car is capable of driving and navigating on its own, by considering the environmental impacts, avoiding collisions and the ability to self-park.

**1.2 Objectives**

The main focus of the project is to reduce environmental impacts by human driven vehicles including car crashes, deaths, ensuring road safety and providing a hassle free drive with the help of Self-driven cars. As mentioned earlier that on average 3700 die due to car crashes for negligent irresponsible driving, an autonomous car can reduce the driver error as it requires no human driving and is automated using sensors and radars. The autonomous car can help subside costs such as medical and vehicle repairing bills. The goal is to also bring in an improvement in managing traffic as there will be no unnecessary accelerations or decelerations.

It will let people have greater independence, especially disabled people with any disabilities, the self-driven car will allow them to live their life to the fullest. The frequent use of autonomous cars will have a positive impact on mobility for people who are unable or for those who cannot drive, either young or old. Moreover it will also help in improving the user experience by increasing the driver’s propensity to travel.

As autonomous car have their own map depending on their surrounding, it will be able to

detect traffic lights, read road signs, track other vehicles and even look out for pedestrians, with all these functions it completely minimizes the human work, which decreases the negative impact of driving including mental tension while driving in cities, and bodily harm such as muscular cramps and back ache. Improving the impacts on environment and climate is also a motive as autonomous cars can reduce the urban emissions worldwidely of carbon dioxide by 80%, which will result in less pollution, fewer lung disease cases and other health conditions. Parking space issues are very common nowadays so the best solution could be this as it saves up space in parking lots.

The main target is to provide a better, safe, stress-free driving experience without any hassle, not only considering the driving perspective but also taking into consideration the impact on environment and transportation costs, with the help of self-driven car.

**1.3 Scope and Limitations**

Technological advancements are accelerating at an uncontrollable rate. The more a country invests in emerging technology, the better off it will be in the long term. These incentives range from improved infrastructure to increased employment creation. Nowadays, technology's main goal is to reduce the amount of human intervention in day-to-day work.

While self-driving cars offer many great benefits to customers, the safety standards are especially complex, and making these cars available to the general public may pose major hurdles. Some of the scopes to use self-driven cars on the roads are:

1. Nearly No Error

Self-driving cars' highly complex technology allows the onboard computer to do hundreds of calculations each second. These include the distance you are from things, your current speed, the behavior of other automobiles, and your geographic location. These incredibly precise measurements have almost eliminated driving errors for test cars on the road, since the few accidents to yet have occurred while human drivers were in charge.

2. Decongests the Area

Self-driving cars have a great potential for reducing traffic congestion because they are rarely involved in accidents. Not only that, but because self-driving cars can communicate with one another, traffic lights would be obsolete. Better traffic coordination might result in less congestion if people drive slower but make fewer stops.

3. Relieves Parking Issues

Self-driving cars, which do not require a driver, might help reduce parking issues in densely populated regions. For example, a passenger might exit at their destination and the automobile could circle the block until the passenger was ready to exit if no parking was available. Because automobiles can coordinate traffic flow, the impact on traffic congestion is predicted to be minimal.

4. New Design Possibilities

The potential for new automotive designs is enormous since a vehicle might someday operate as a self-guided railway car. Self-driving cars might offer new methods to relax or keep engaged because they don't require sophisticated driving gear. However, the new design possibilities aren't restricted to the interior, and self-driving automobiles may eventually be indistinguishable from conventional vehicles.

5. The Possibility of More Powerful Vehicles

Because self-driving vehicles don't need a driver, technicians may be able to change the location of various mechanical elements on the vehicle. This might lead to more competent and powerful engines in automobiles. Cars might ultimately be capable of significantly greater speeds with fewer driving mistakes.

The limitations of the projects are:

1. Expensive

Self-driving cars are fascinating because they are packed with cutting-edge technology, but all of this technology is presently outrageously expensive. In general, technology becomes less expensive the longer it is available to the general population, thus self-driving cars may one day be affordable to everyone. However, most firms have yet to announce pricing for their self-driving car.

1. The Risk of Technology Going Wrong

Though effective programming allows us to do great feats, there is always the risk of an unanticipated error. Even if a self-driving car initially functions flawlessly, the programming that controls the automobiles might be changed by the car manufacturer with a defective string of code.

1. Weather Conditions

When it snows hard in the winter, the lines that indicate lanes frequently vanish. In blizzards, blinding rain, fog, or other circumstances when visibility is nil, self-driving cars rely on cameras, which are made nearly unusable. Indeed, this has proven to be such a challenge for tech companies developing self-driving cars that Google and several other companies have reported to regulatory authorities in several states that human drivers were forced to take control of their prototype self-driving vehicles during these exact weather conditions.

**1.4 Outline of Thesis:**

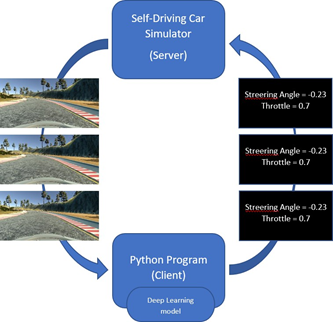
A self-driving car project's goal is to create a better autonomous driver. The automobile should be capable of driving itself without veering off the road, accelerating and braking at the proper times. The aim is to use a model built using deep neural networks to simulate human driving behavior on the simulator. Autonomous driving necessitates mastering the control of steering angle, throttle, and brakes. In the training mode on the track, a behavioral cloning approach is utilized to replicate human driving behavior. That is, in the simulator, a dataset is created by a user-driven automobile in training mode, and the deep neural network model subsequently drives the car in autonomous mode.

The technologies utilized in this project's implementation, as well as the reason for using them, are TensorFlow, a dataflow programming library that is open-source. It's a popular choice for machine learning applications. Python has a number of libraries that may be used in machine learning applications. Several of those libraries have helped to enhance the project's performance. MiniConda Environment, an open source Python distribution that facilitates package management and deployment, is used in this project. It's ideal for processing massive amounts of data. Other than that "Numpy" is a set of high-level arithmetic functions that supports multi-dimensional matrices and arrays. This is used in neural networks to speed up computations over weights (gradients). Second, "scikit-learn" is a Python-based machine learning library that includes a variety of algorithms and Machine Learning function packages. Another option is OpenCV (Open Source Computer Vision Library), which is focused on real-time applications and is intended for computational efficiency. OpenCV is utilized for picture pre-processing and augmentation techniques in this project.

Convolutional Neural Networks (CNN) is used as It excels in extracting hierarchical and geographical information from pictures. It makes use of filters that map portions of an input picture with a given window size to some output. By using all the techniques and with the aid of self-driven automobiles, the major goal is to deliver a better, safe, stress-free driving experience without any trouble, and to solve all other difficulties not only from a driving viewpoint but also from the standpoint of environmental effect and transportation costs.

**CHAPTER TWO: REAL WORK**

**2.1 Material and Method**

****

# Fig 1. Implementation Architecture

Data may be collected by driving the simulator in training mode using a joystick or keyboard, giving "good-driving" behavior input data in the form of a driving log (.csv file) and a collection of pictures. The simulator serves as a server, sending pictures and data to the Python client.

The machine learning model developed with Deep Neural Networks is the client (Python application). Keras is used to create these models (a high-level API over Tensorflow). Keras provides sequential models for constructing a linear network layer stack. In the project, such models are utilized to train over datasets in the second stage

This section describes the technologies utilized in the implementation of this project as well as the motivation for using them.

TensorFlow: This is an open-source dataflow programming library. Machine learning programs make extensive use of it. It's also used for massive computations and as a math library. Keras is a high-level API that leverages TensorFlow as the backend for this project.Keras makes it easier to create models because it is more user-friendly.

Python has a number of libraries that can be used in machine learning projects.

Several of those libraries have helped to improve the project's performance. This section just mentions a few of them. First, there's "Numpy," which provides a library of high-level arithmetic functions to enable multi-dimensional matrices and arrays. This is used in neural networks to speed up computations over weights (gradients). Second, “scikit-learn” is a Python machine learning library that includes a variety of algorithms and Machine Learning function packages. Another option is OpenCV (Open Source Computer Vision Library), which is focused on real-time 4 applications and is geared for computational efficiency.

OpenCV is utilized for image preprocessing and augmentation techniques in this project Anaconda Environment, an open source Python distribution that facilitates package management and deployment, is used in this project. It's ideal for processing big amounts of data. This project was developed on a personal computer with the following configuration:

* CPU : Amd Ryzen 5 3600
* RAM : 16 Gb ddr4 3600 Mhz
* Gpu : Nvidia Gtx 1660 super

**2.2 Analysis**

## 2.2.1 Perception

The perceptron is a very basic type of a neural network that is used to learn binary classifiers supervised. Input values, weights, and a bias, as well as a weighted sum and activation function, make up a perceptron. Perceptrons are divided into two categories: single layer and multilayer.

Single-layer perceptrons can only learn patterns that are linearly separable.

Multilayer - With two or more layers, multilayer perceptrons or feedforward neural networks have increased processing power.

In order to build a linear decision boundary, the Perceptron algorithm learns the weights for the input signals.

A perceptron operates by taking numerical inputs and combining them with weights and a bias. It then multiplies these inputs by the weights assigned to them (this is known as the weighted sum). These items, together with the bias, are then combined. The weighted sum and bias are inputs to the activation function, which returns a final output.

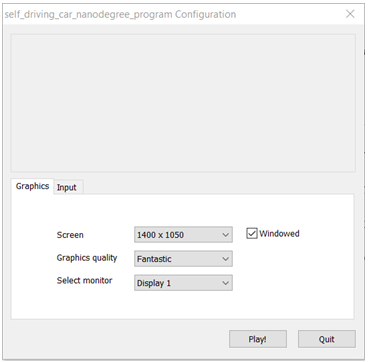
**2.2.2 Convolutional Neural Networks (CNN)**

CNN is a feed-forward neural network computing system that can learn from its incoming data. The network learns by determining a set of weights or filter values that allow the network to model behavior based on the training data. The desired outcome and the outcome generated by a CNN with random weights will differ. This difference (produced error) is returned to the layers of CNN, which adjusts the weights of the neurons, reducing the mistake and allowing us to produce output that is closer to the desired one.

CNN excels at extracting hierarchical and spatial information from images. It makes use of filters to map parts of an input image with a set window size to some output. It then moves the window to other places, covering the entire image with a given stride. Each convolution filter layer thus hierarchically captures the attributes of this input image in a succession of successive layers, collecting details such as lines in the image, forms, and finally full objects in later layers. CNN is an excellent fit for feeding photos from a dataset and classifying them into their appropriate categories.

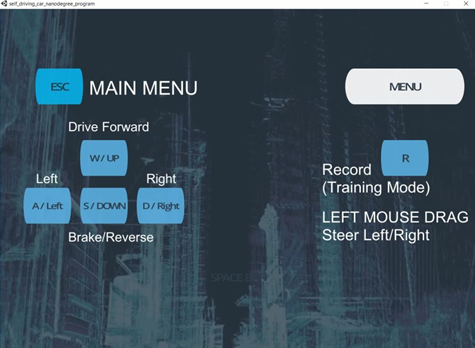
**CHAPTER THREE: Udacity Simulator and Dataset:**

Udacity has created a self-driving car simulator and released it open source so that fans may work on something that is near to real-time. Unity, a video game creation platform, is used to create it. The simulator is highly user friendly and has a changeable resolution and controls configuration.



**Fig 2: Configuration screen**

Depending on the user's preferences and the machine's settings, the graphics and input configurations can be altered. The user enters the simulator user interface by pressing the "Play!" button. You may explore the keyboard controls by going to the Controls tab, which looks a lot like a racing game.

**Fig 3. Controls Configuration**

**Fig 4. First Screen**

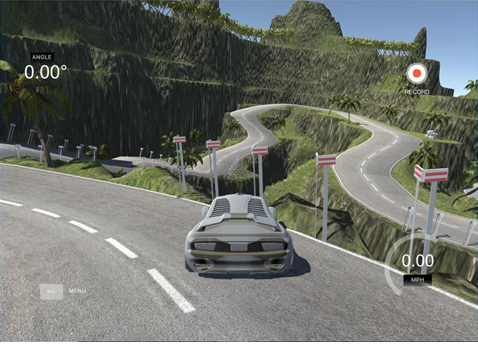
The simulator's first actual screen, and its components are detailed below:

There are two tracks in the simulator. One of them may be classified as basic, while the other can be classified as difficult, as shown in the screenshots. The term "simple" simply refers to the fact that it has fewer curves and is therefore easy to drive on.



**Fig 5. Track 1**

The "challenging" track contains high hills, abrupt twists, and a gloomy atmosphere, making it difficult to drive even by hand.



**Fig 6. Track 2**

In the simulator, there are two driving modes: (1) training mode and (2) autonomous mode.The training mode allows you to record your run and save the training information.

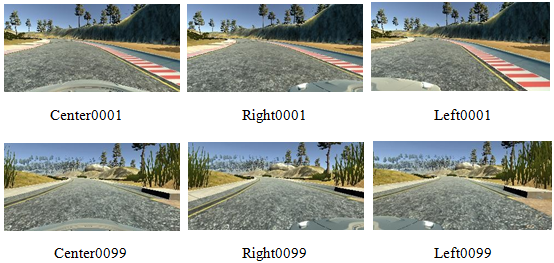
The figure shows a little red indication at the top right of the screen indicating that the automobile is in training mode.  
The autonomous mode may be used to put the models through their paces and check if they can drive on the course without human interference.Furthermore, if you try to use the controls to get the car back on course, it will quickly tell you that it has switched to manual controls.



**Fig 7. Autonomous mode**

The ability to construct your own picture dataset in the simulator makes it simple to work on the topic. The following are some of the reasons why this feature is beneficial:

* The driving aspects of the simulator have been designed to make it appear as though the automobile had three cameras. The three cameras on the front of the automobile, located in the center, right, and left, capture continually while we record in training mode.
* The simulator's driving features have been engineered to make it look as though the car had three cameras. While we record in training mode, the three cameras on the front of the car, placed in the center, right, and left, continuously capture.
* After pressing the record button, the stream of photos is taken, and we may choose where to save the data on the disk. The image collection is labeled in a sophisticated way, with a prefix of center, left, or right indicating which camera recorded the image.
* It also creates a datalog.csv file in addition to the picture dataset. This file holds the picture paths associated with the car's current steering angle, throttle, brakes, and speed.



**Fig 8. Dataset Sample**

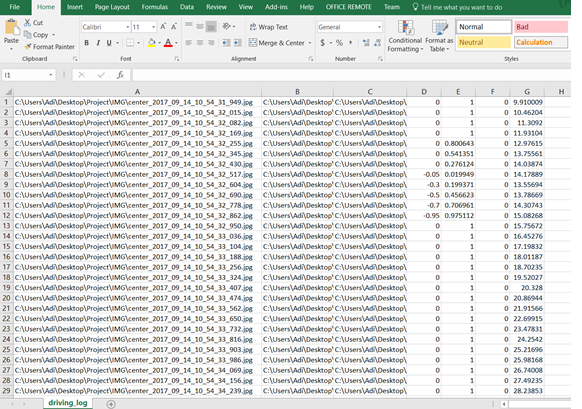
**Column 1, 2, 3:** provides paths to the center, right, and left dataset pictures, respectively.

**Column 4:** includes the steering angle, with a column value of 0 representing straight, a positive value representing a right turn, and a negative value representing a left turn.

**Column 5:** includes the throttle or acceleration at that time

**Column 6:** includes the brakes or slowdown at that time

**Column 7:** includes the vehicle's speed



**Fig 9. driving\_log.csv**

**CHAPTER FOUR: Technical Chapters**

**4.1 Technologies used:**

This section describes the technologies utilized in the implementation of this project, as well as the reason for using them.

1. Finding lane lines:  
   We utilize our sight to decide where to go when driving. The lines on the road that indicate where the lanes are serve as a continual reference for steering the car. Naturally, one of the first things we'd like to accomplish in the development of a self-driving car is use an algorithm to automatically recognize lane lines.

The purpose of this project is to create a pipeline that uses Python and OpenCV to locate lane lines on the road. Consider the following scenario:



**Finding lanes**

Udacity will offer some images and videos for the pipeline to be tested on. Assumptions are made as follows:

* In relation to the road, the camera is always in the same location.
* On the road, there is usually a visible white or yellow line.
* There is no vehicle in front of us.
* We explore a highway situation with ideal weather.

def make\_coordinates(image, line):

slope, intercept = line

y1 = int(image.shape[0])# bottom of the image

y2 = int(y1\*3/5) # slightly lower than the middle

x1 = int((y1 - intercept)/slope)

x2 = int((y2 - intercept)/slope)

return np.array([[x1, y1, x2, y2]])

def average\_slope\_intercept(image, lines):

left\_fit = []

right\_fit = []

if lines is None:

return None

for line in lines:

for x1, y1, x2, y2 in line:

fit = np.polyfit((x1,x2), (y1,y2), 1)

slope = fit[0]

intercept = fit[1]

if slope < 0: # y is reversed in image

left\_fit.append((slope, intercept))

else:

right\_fit.append((slope, intercept))

left\_fit\_average = np.average(left\_fit, axis=0)

right\_fit\_average = np.average(right\_fit, axis=0)

left\_line = make\_coordinates(image, left\_fit\_average)

right\_line = make\_coordinates(image, right\_fit\_average)

averaged\_lines = [left\_line, right\_line]

return np.array([left\_line, right\_line])

def canny(image):

gray = cv2.cvtColor(image, cv2.COLOR\_RGB2GRAY)

kernel=5

blur = cv2.GaussianBlur(gray,(kernel, kernel),0)

canny = cv2.Canny(gray, 50, 150)

return canny

def display\_lines(image,lines):

line\_image = np.zeros\_like(image)

if lines is not None:

for line in lines:

x1, y1, x2, y2 = line.reshape(4)

cv2.line(line\_image,(x1,y1),(x2,y2),(255,0,0),10)

return line\_image

def region\_of\_interest(canny):

height = canny.shape[0]

width = canny.shape[1]

mask = np.zeros\_like(canny)

triangle = np.array([[

(200, height),

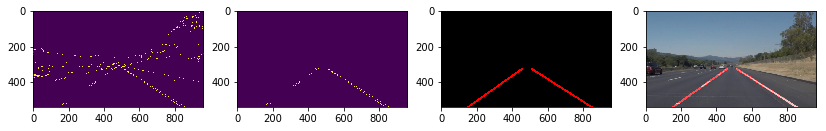
(550, 250),

(1100, height),]], np.int32)

cv2.fillPoly(mask, triangle, 255)

masked\_image = cv2.bitwise\_and(canny, mask)

return masked\_image



**Fig 11: Returned Masked Images**

1. TensorFlow:

TensorFlow is an open-source dataflow programming framework. The new library provides researchers with the tools and resources they need to create and deploy 3D scene understanding models. Machine learning programs make extensive use of it. It's also used for big computations and as a math library. Keras is a high-level API that leverages TensorFlow as the backend for this project. Keras makes it easier to create models since it is more user-friendly. In this project, OpenCV is used for image preprocessing and augmentation techniques.

1. Convolutional Neural Networks (CNN):

CNN is a feed-forward neural network computer system that can learn from its incoming data. The network learns by determining a set of weights or filter values that allow the network to simulate behavior based on the training data. The desired outcome and the outcome obtained by a CNN with random weights will differ. This difference (produced error) is back propagated via the CNN layers, adjusting the weights of the neurons, reducing the mistake and allowing us to create output that is closer to the desired one. Good for capturing hierarchical and spatial data from images. It uses filters that see areas of an input image with a defined window size and map them to an output. Then it moves the window to other areas a certain step and covers the entire image Each layer of convolutional filter thus captures the Properties of this input image hierarchically in a series of subsequent layers, with details such as lines in the image, then shapes, and then entire objects being captured on subsequent layers.

1. Classifying Road Symbols (CRS) :  
   Need a lot of data — thousands of photos — to identify images using a process known as "deep learning." This is because a deep learning neural network takes in each image and learns about it. The more instances you have, the more you can learn from them, and the higher your performance will be.

Fortunately, there is a publicly available dataset of accurately classified German traffic signs. Yes, they are German road signs.

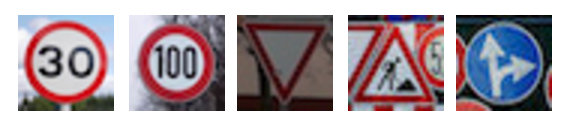
There are 43 different types of traffic signs in the dataset, with image quality ranging from poor to excellent. All of the photographs are in color, and the dataset I used had them all scaled down to 32x32 pixels, which is somewhat grainy. There were approximately 47,000 photos in total.

The specifics of how deep neural networks work are complicated, but here are some key points:

* "Deep" refers to the presence of numerous levels.
* Each layer "learns" to recognize features in the photos it receives.
* The network may identify "features of features" by connecting these layers. The first layer, for example, might identify lines, whereas the second layer groups lines (like triangles). Soon, you'll have something that resembles a yield sign.
* Using the power of mathematics, neural networks understand what features are relevant to classifying the data on their own.

The network improves and gets better at classifying the indicators as more photographs are sent in. My network was 95 percent correct in the end. By the way, the accuracy number will change based on the type of sign. It will be better at identifying speed signs because it has seen more of them than turning signs.

We downloaded five different German traffic signals from the internet to demonstrate this and test my network. The five photos are shown below after being resized to 32x32 pixels



**Fig 12: German Traffic Signals**

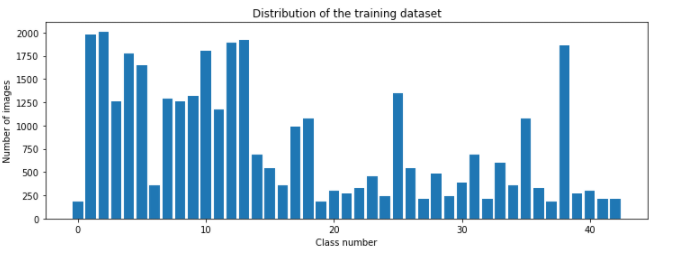
The five photos we downloaded to test my model.

We gave these five photographs to my network and asked it to tell me what kind of sign the pictures reflect and how confident it was in its guess (relu probability). The results are summarized in the table below.



**Fig 13: Relu probability results**

All save the last sign — the "go straight or left" sign — were correctly identified by the network. The fact that the network classified the sign as "stay right," which is at least similar, gives me some reassurance.



**Fig 14: Training dataset distributed**

This is a random RGB image from our dataset .



**Fig 15: RGB image**

This is the code to convert the RGB image to GrayScale

def grayscale(img):

img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

return img

img = grayscale(X\_train[1000])

plt.imshow(img, cmap=plt.get\_cmap("Greys"))

plt.axis('off')

print(img.shape)

This is the grayscale image conversion of our RGB image.



**Fig 16 : Grayscale Image**

This is the code to equalize the image:

def equalize(img):

img = cv2.equalizeHist(img)

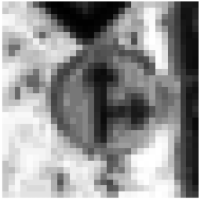
return img

img = equalize(img)

plt.imshow(img, cmap=plt.get\_cmap("Greys"))

plt.axis('off')

print(img.shape)



**Fig 17 : Equalized Image**

This is some random images which are already preprocessed .

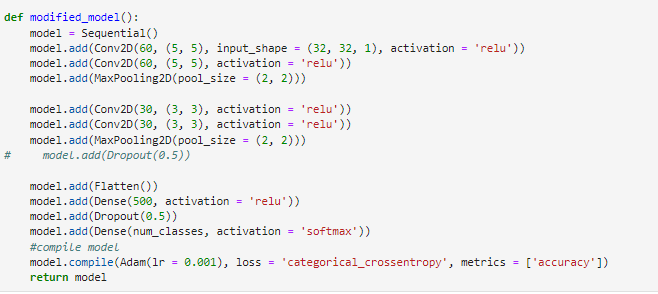


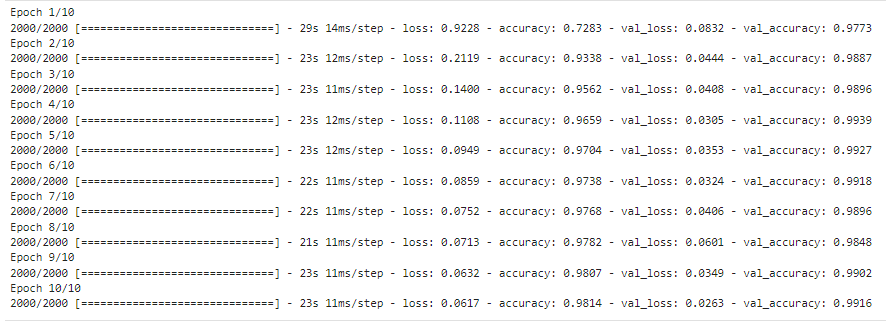
**Fig 18 : Images after processing**

There were different mixes of models taken a stab at, foreseeing the traffic signs and contribution for the vehicle to drive in independent mode. Neural Network layers were

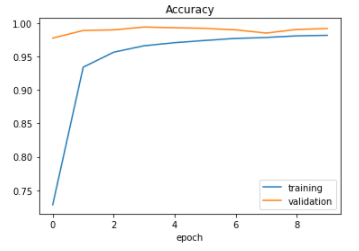
coordinated in series and different mixes of Time-Distributed Convolution layers,MaxPooling, Flatten, Dropout, Dense, Adam etc are utilized in models. The best

performing ones are displayed exhaustively. Allude to the model postings for the boundaries used to fabricate them. The significant level perspective on layers used to fabricate the models is displayed in the going with design figures.





**Fig 19 : Epoch**



**Fig 20 : Accuracy and loss of model**

1. Recurrent Neural Networks (RNN):

RNNs are a class of artificial neural networks in which the connections between units form a directed loop. Recurring networks, unlike feedback networks, have the feedback loop connected to their past decisions and take their own output as input (like a memory). This memory (feedback) helps to learn sequences and predict subsequent values, thus being able to resolve dependencies over time. To deal with non-linearity and keep gradients or weights within a specified range, neural networks use activation functions. Sigmoid, tanh, RELU, and other RNN building blocks are examples of these functions. Even though they are extremely strong, traditional RNNs have significant flaws, such as the well-known problem of disappearing or expanding gradients.

1. LSTM (Long Short-Term Memory):

LSTM is a kind of RNN that is better at dealing with disappearing and exploding gradient issues. A forget gate, input gate, and output gate make up an LSTM block. In the LSTM cell, a short demonstration of how these gates function may be seen. Figure 1:



**Fig 21. LSTM Cell (Long Short-Term Memory)**

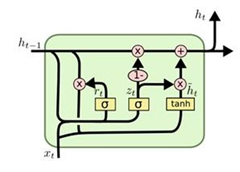
The following are the gates that are involved in the design of the LSTM cell:

Forget about the gate: The "forget gate layer," a sigmoid activation layer (illustrated in red), makes this decision.

Only the selected input is allowed to flow through the input gate. The "input gate layer" determines which values are allowed to flow through. Following that, a tanh layer of activation will aid in the creation of a state update. (as shown in orange)

Gate for output: Only some aspects of the cell state will be output. The cell state is then sent through that layer (to force the values to be between 1 and) and multiplied by the sigmoid layer gate's output. This is the layer that contains the output (shown in blue).

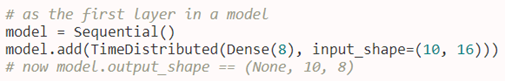
# GRU (Gated Recurrent Unit):

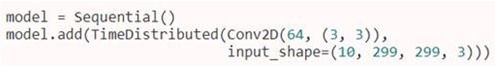
The Gated Recurrent Unit is comparable to the LSTM mentioned previously. Gated mechanisms are utilized, similar to LSTMs, and are meant to update memory content using the update gate, which is similar to the input gate. The GRU utilizes a reset gate to clear its memory, similar to the LSTM's forget gate.  
  


**Fig 22.GRU (Gated Recurrent Unit) Cell**

The LSTM and GRU outperform the standard RNN unit in the majority of studies. However, studies have not identified significant differences in performance between the LSTM and the GRU.

1. Time-Distributed Layers:

A time distributed layer is another sort of layer that is occasionally employed in deep learning networks. Keras includes time-distributed layers as wrapper layers. This wrapper layer is applied to each temporal slice of an input. The input must be at least three-dimensional, with the first index serving as the time dimension. These Time-Distributed may be applied to a dense layer to each of the time steps individually or in conjunction with convolutional layers.   
  
  
  
 **Fig 23. TimeDistributed Dense layer**

  
 **Fig 24. TimeDistributed Convolution layer**

There isn't a lot of material regarding TimeDistributed layers out there, but Jason Brownlee's "How to utilize TimeDistributed Layers for LSTM" is a good place to start.

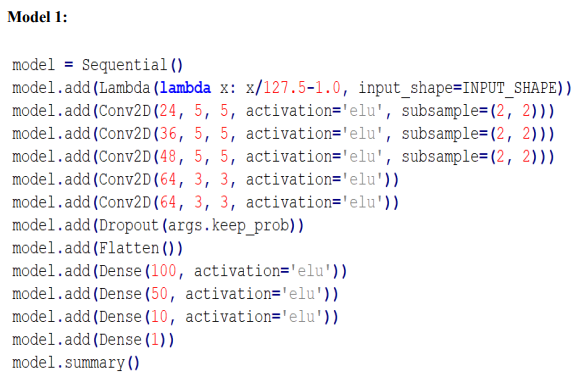
# RCNN (Combination of CNN and RNN): RCNN is the abbreviation for this combo (Recurrent Convolutional Neural Networks). There have been several RCNN implementations in recent years. This phrase is also known as region-based CNN (R-CNN), which is a prominent approach for object recognition in pictures. Every time this phrase is used in this project, it will relate to Recurrent CNNs. This combination can be produced using a variety of ways or procedures. In terms of picture categorization (more on spatial properties of data) and sequence prediction, both CNN and RNN are quite beneficial (temporal characteristics of data). The hybrid models can include a lot of convolution layers and another RNN branch (LSTM, GRU, or both) running in parallel or stacked in series.

**4.2 Network Architectures:**

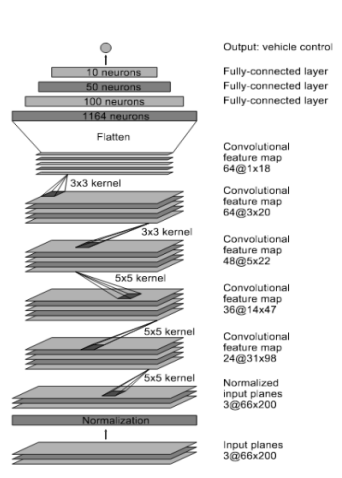
There were different mixes of models taken a stab at, foreseeing the controlling point and contribution for the vehicle to drive in independent mode. Neural Network layers were

coordinated in series and different mixes of Time-Distributed Convolution layers,MaxPooling, Flatten, Dropout, Dense, etc are utilized in models. The best

performing ones are displayed exhaustively. Allude to the model postings for the boundaries used to fabricate them. The significant level perspective on layers used to fabricate the models is displayed in the going with design figures.



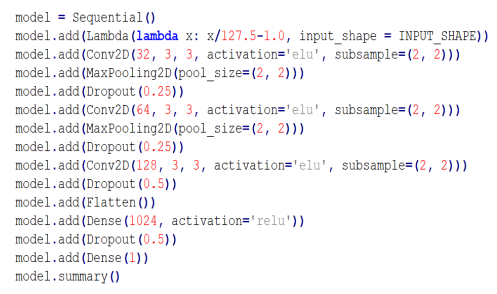
**Fig 25. Model 1**



**Fig 26. Architecture 1**

NVIDIA delivered a design for self-driving vehicles [4] and it is utilized in the project for reference to take care of the issue and for contrasting and the different other models attempted. Various designs have been normalized over the course of the years for building consecutive models of CNN like AlexNet, VGG-Net, GoogLeNet, ResNet, etc.

For model 2:



**Fig 27. Model 2**



**Fig 28. Architecture 2**

Model 3 is an architecture comparable to VGG-Net, with variants achieved by adjusting  
parameters to meet the project's needs. Figure 19 depicts an overview of the structures.



**Fig 29. Model 3**



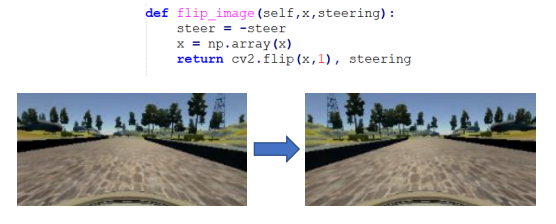
**Fig 30. Architecture 3**

**4.3 Augmentation and Image pre-processing:**

Image recognition prior to deep learning is not always optimal because image features are extracted and expressed using an algorithm designed based on the knowledge of researchers, which is called a handcrafted feature. Convolutional neural network (CNN), which is one type of deep learning, is an approach for learning classification and feature extraction from training samples. This chapter focuses on object detection and scene under-standing (semantic segmentation) and describes its application to image recognition and its trends,and also how to generalize the behavior of the car.

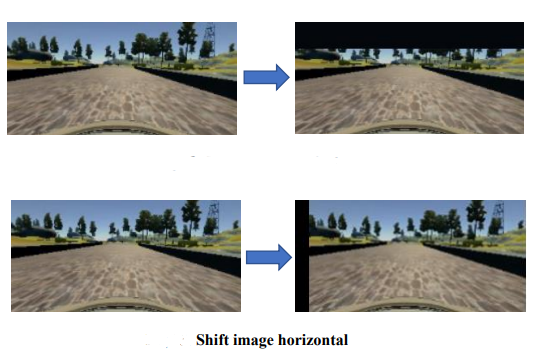
The greatest test was summing up the conduct of the vehicle on Track\_2 which was never prepared for. In a genuine circumstance, we can never prepare a self-driving vehicle model for all tracks imaginable, as the information will be too colossal to even consider handling. Likewise, it is unimaginable to expect to assemble the dataset for every one of the climate conditions and streets. Accordingly, there is a need to come up with a thought of summing up the conduct on various tracks. This issue is tackled utilizing picture pre-preparing and increase methods, which will be examined in the following segment:

* **Flip (horizontal):** The picture is flipped evenly (for example an identical representation of the first picture is passed to the dataset). The thought process behind this is that the model gets prepared for comparable sorts of turns on inverse sides as well. This is significant in light of the fact that Track\_1 incorporates just left turns. The scrap of code and change of a picture in the wake of flipping it tends to be found in the Figure:



**Fig 31. Flip image**

* **Shift (horizontal/vertical):** The image is moved just barely, here it is upward change in and level shift:

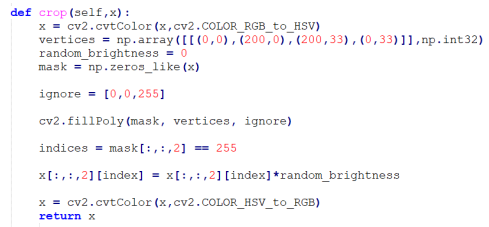


**Fig 32 & Fig 33. Shift image vertical & Shift image horizontal**

* **Crop:** The images in the dataset have important provisions in the lower part where the street is apparent. The outside climate over a specific picture piece won't ever be utilized to decide the yield and accordingly can be edited. Roughly, 30% of the top piece of the picture is cut and passed in the preparation set. The piece of code and change of a picture in the wake of trimming and resizing it to unique picture can be found in after Figure:

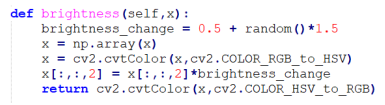


**Fig 34. Crop image**

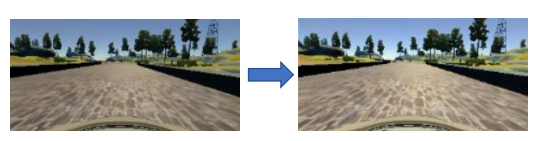


**Fig 35. Crop image (continued)**

* **Brightness:** To sum up the climate conditions with brilliant radiant days or shady, lowlight conditions, the splendor expansion can end up being exceptionally valuable. The code scrap and increment of splendor can be found in the accompanying figure. Additionally, we have arbitrarily likewise let down the degree of brilliance for different conditions

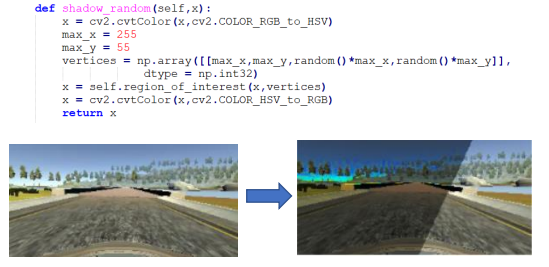
****

**Fig 36. Brightness increase**

****

**Fig 37. Brightness increased(continued)**

* **Shadows:** Indeed, even subsequent to taking into contemplation the light conditions, there are still possibilities that there are shadows out and about. This will give an occurrence of half lit and half low light scenes in the picture. To project irregular shadows and address this shadow fitting problem,this expansion is applied on the dataset. An example shadow expansion with its code bit is displayed in the accompanying figure:

****

**Fig 38. Random shadows**

* **Random Blur:** To take care of the distortion effect in the camera while capturing the images, this augmentation is used as an image captured is not clear every time. Sometimes, the 29 camera goes out of focus, but the car still needs to fit that condition and keep the car steady. This random blur augmentation can take such scenarios into consideration. The sample code snippet and the transformation can be seen in following figure:



**Fig 39. Random blur**

* **Noise:** This adds arbitrary commotion to the pictures by thinking about the messy conditions by mimicking residue or soil particles and twists while catching the picture. The example of the code piece and change can be found in this figure:



**Fig 40. Random noise**

**CHAPTER FIVE: Experimental Configurations**

The configurations in this section are used to build up the models for training the Python Client to deliver the Neural Network outputs that drive the automobile on the simulator. To find the ideal combination, parameters were tweaked and extensive tests were conducted. Despite the fact that each model had its own distinct characteristics and changed in performance with each modification, the following setup may be deemed the best:

* The data is trained using sequential models based on Keras with deep neural network layers.
* The data is trained using sequential models based on Keras with deep neural network layers.
* The dataset is used for training 80% of the time and testing 20% of the time.
* The number of iterations or sweeps across the entire dataset is 50 epochs.

We also attempted a higher number of epochs, but the model tried to "overfit." In other words, the model learns too much information from the training data, affecting performance on future datasets.

* Batch-size = 40, i.e. the number of picture samples passed through the network as a subset of data because the entire dataset is too large to be passed all at once.
* The learning rate is 0.0001, which refers to how the weights' coefficients or gradients change in the network.
* ModelCheckpoint() is a Keras function that allows you to save checkpoints and the best epoch based on the validation loss.

To create Neural Network models, many combinations of Convolution layers. Time-Distributed layer, MaxPooling layer, Flatten, Dropout, dense, and so on may be employed. Three of the better architectures we attempted out of a total of 10 are mentioned in the network architectures chapter.

**Chapter 6: Results:**

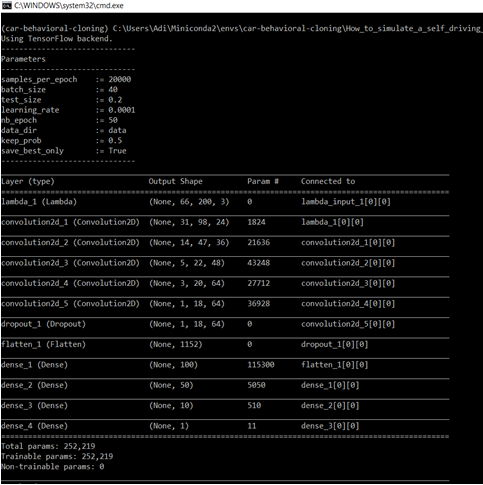
For each of the previously presented designs, the following outcomes were found. We needed to come up with two separate performance indicators to compare them.

1. Loss of value or accuracy (computed during training phase)

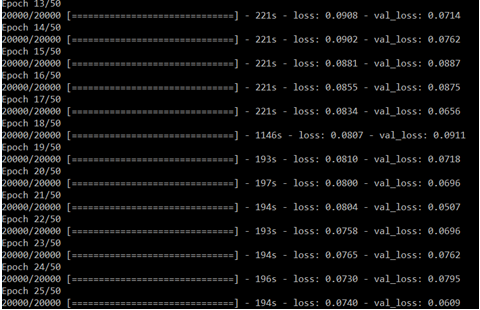
2. Track 2 generalization (drive performance)

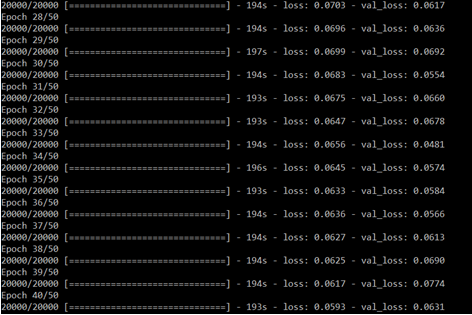
**Value loss or accuracy:**

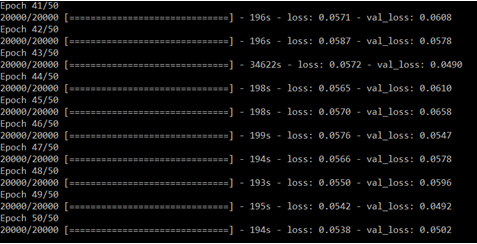
The "Loss" during each epoch of the training run is the first assessment parameter examined. Keras gives "val loss," which is the average loss after each epoch, to calculate value loss over time. The loss shown during the early epochs of the training phase is substantial, but it eventually decreases, as shown in the screenshots below.





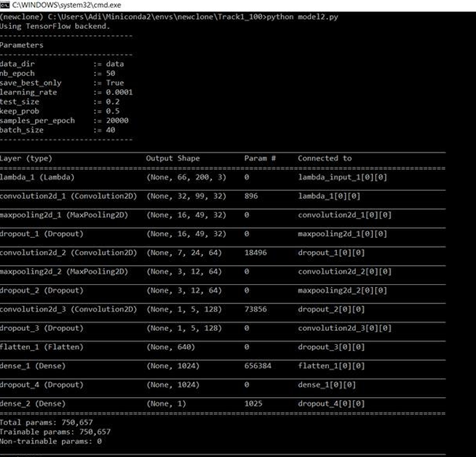


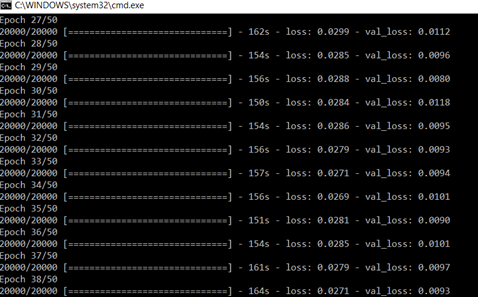


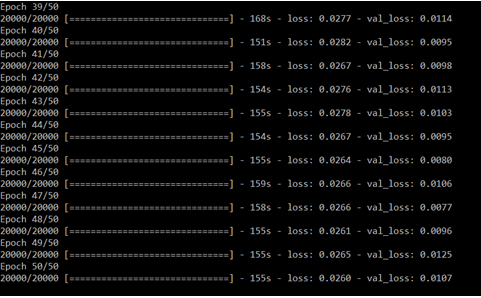


**Fig 41. Loss over epochs (Architecture\_1)**

The following screenshots demonstrates the the run of Architecture\_2 in the training phase:

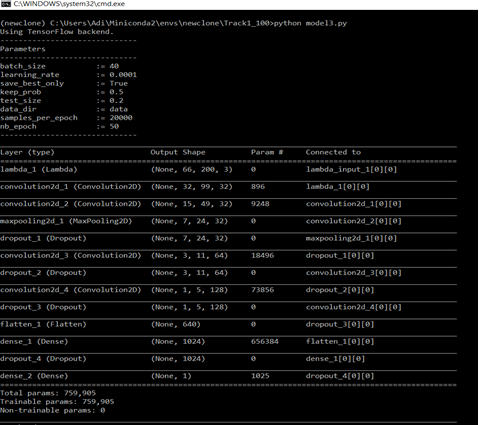
****

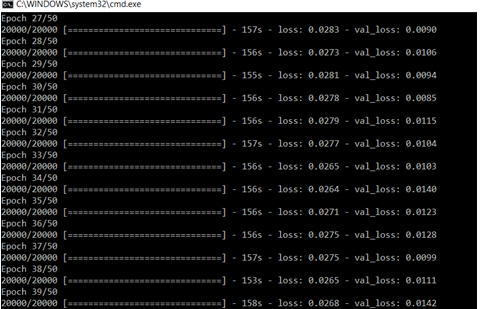
****

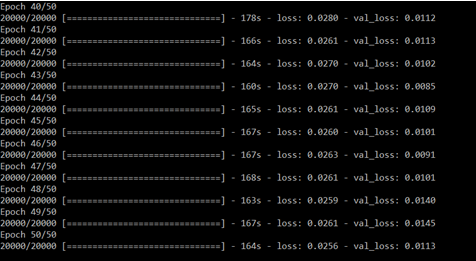
****

**Fig 42. Loss over epochs (Architecture\_2)**

The following screenshots demonstrates the run of Architecture\_3in the training phase:

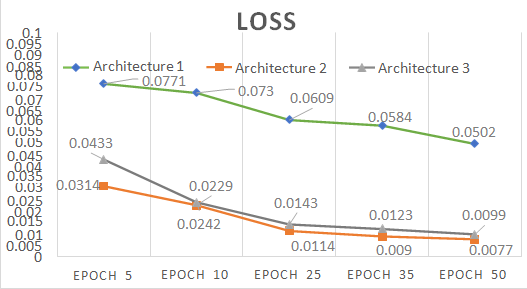






**Fig 43. Loss over epochs (Architecture\_3)**

For comparison, the loss must be plotted over epochs. I created a graph that shows the loss for each of the three designs. The graph drawn between 0 and 0.1 (loss values) allows for a more accurate comparison of the various design outcomes.



**Fig 44. Loss over epochs (Graph)**

As a result of the graph of Loss across epochs, it can be inferred that Architecture 2 did the best with the least loss of 0.0077 at the conclusion of 50 epochs. For Track 1 training and testing, this yielded the best results.

**Track 2 Generalization (Drive Performance)**

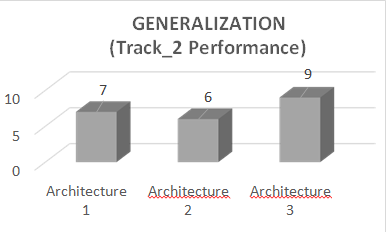
Generalization was the second criteria utilized to assess the outcomes**.** This may be characterized as how effectively the models forecast values when driving on a track for which they were not trained. The expected steering angle, brakes, and throttle are the values here. It's not something that can be charted, but it can be measured in terms of how far the car can travel on the second track without tipping over. The pace, twists, and track circumstances such as hills, shadows, and so on can all have an impact.   
The models are only trained on Track 1 data because it is easier, but they are tested on Track 2. As a result, working with Track 2 was quite difficult. During the experiment, a few new findings were made that support this claim.

* Despite performing well on Track 1 and providing the best accuracy during training (loss over epochs), most architectures were unable to complete even the first turn on Track 2.
* Overfitting for the Track 1 dataset might be the cause. Overfitting occurs when a software attempts to mimic the training data too accurately. The model trains for details and even noise for the data it has passed through in general. This, in turn, has a detrimental influence on the capacity to generalize.
* Another cause might be that Track 2 begins with a route that runs parallel to the one where the car begins, separated by a barrier. A snapshot of the mentioned circumstance is shown in the next image. This might be one of those sights that it will never see while preparing for Track 1.



**Track\_2 screenshot**

* Image preprocessing and augmentation techniques are utilized to get around this challenge.
* Despite the fact that Architecture 2 had the best accuracy (i.e. val loss was the smallest), it struggled on Track 2. In reality, while driving on Track 2, Architecture 3 produced superior results.



**Generalization rating (Graph)**

* As a result, in the above picture, a generalization rating graph has been created based on the relative distance the car traveled autonomously on Track 2. One was assigned to the worst performance, five to the middling, and ten to the greatest. Please keep in mind that these are qualitative rather than quantitative ratings. There were times when the car couldn't travel even a short distance and tried to turn onto the other parallel track right away, only to become stuck in the barrier immediately at the start. This scenario is a one-star scenario.
* Better results could be achieved by increasing the number of convolution and max-pooling layers. As a result, some topologies outperformed others, although bigger networks take longer to calculate (train). However, it is possible that increasing the number of convolution layers will not always enhance the results. When the number was increased further, relatively similar results were achieved, with occasionally poorer outcomes.

**Conclusion:**

Overall, autonomous cars are beneficial to society since they will give us more time, reduce accidents, and allow technology to grow. Self-driving cars have the potential to prevent 90% of all car accidents caused by human error. We can save money on car parts, there will be less waste in the environment, and injuries and deaths will be reduced if there are fewer accidents. However, automation is not without its drawbacks. Jobs will be lost, there will be feuds if a self-driving car and a standard car collide (liability issues), and a full transition to autonomous vehicles will be challenging. In general, automation is transforming jobs from labor-intensive to technologically-intensive. This will be problematic for some people, but it is beneficial in the long run because it stimulates education and innovation.

This study began with the models being trained and parameters being tweaked to achieve the greatest performance on the tracks, and then attempting to generalize that performance across multiple tracks. Because the models that performed well on one track performed badly on Track 2, picture augmentation and processing were required to accomplish real-time generalization. The usage of CNN for spatial characteristics and RNN for temporal features in the picture dataset makes this combination ideal for quickly constructing neural networks with little processing. Substituting recurrent layers for pooling layers might decrease information loss and is something worth investigating in future studies. It's fascinating to see how these models are trained using a combination of real-world data and simulator data. Then we will understand how a model may be taught in a simulator and then generalized to the actual world, or vice versa. In the realm of self-driving automobiles, there are several experimental implementations underway, and our project contributes to a substantial portion of them.

**Future Work:**

Deep neural network layers were employed in sequential models during the project's execution. The use of a parallel network of network layers to learn to follow specific behavior on distinct branches can increase the project's speed significantly. One branch can contain CNN layers, the other RNN layers, and the output combined with a dense layer at the end. RESNET is used to handle challenges of a similar nature (Deep Residual networks). a framework for modular learning, RESNET are more complex than their 'plain' counterparts (state-of-the-art deep neural networks), yet they need a similar amount of parameters (weights). Using Reinforcement Learning techniques to determine steering angles, throttle, and braking can also be a fantastic way to deal with such issues. Placing dummy automobiles and obstacles on the tracks would raise the level of difficulty in solving this problem, but it would bring it much closer to the real-time environment that self-driving cars would confront in the real world. The model's performance on real-world data might be a suitable test. The model was tested with a real-world dataset, but there was no method to evaluate it in a simulated setting. The major companies in the self-driving car sectors must already be experimenting with this on their autonomous cars. This would be an excellent experiment to test how this model performs in a real-world setting.

**REFERENCES**

1. Oliver Cameron, “Challenge #2: Using Deep Learning to Predict Steering Angles”,Published on 11 Aug 2016, [https://medium.co/udacity/challenge-2-](https://medium.com/udacity/challenge-2-using-deep-learning-to-predict-steering-angles-f42004a36ff3)

[using-deep-learning-to-predict-steering-angles-f42004a36ff3,](https://medium.com/udacity/challenge-2-using-deep-learning-to-predict-steering-angles-f42004a36ff3) accessed Jul 2017

1. Ujjwalkaran, “An intuitive Explanation to Convolutional Neural networks”, Published on 11 Aug 2016, [https://ujjwalkarn.me/2016/08/11/intuitive-](https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets)

[explanation-convnets](https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets), accessed Nov 2017

3. Andrej Karpathy, “The unreasonable effectiveness of Recurrent Neural Networks”, Published on 21 May, 2015, [http://karpathy.github.io/2015/05](http://karpathy.github.io/2015/05/21/rnn-effectiveness)

[/21/rnn-effectiveness,](http://karpathy.github.io/2015/05/21/rnn-effectiveness) accessed Nov 2017

4. [Mariusz Bojarski,](https://devblogs.nvidia.com/author/mbojarski/) “End-to-End Deep Learning for Self-Driving Cars”, Published on 17 Aug, 2016,<https://devblogs.nvidia.com/deep-learning-self->

[driving-cars/](https://devblogs.nvidia.com/deep-learning-self-driving-cars/),accessed Nov 2017

5. Jason Brownlee, “How to use TimeDistributed Layers for LSTM”,<https://machinelearningmastery.com/timedistributed-layer-for-long-short-term-memory-networks-in-python/>, accessed Nov 2017

6. Lucas Weist, “[Recurrent Neural Networks - Combination of RNN and CNN](https://wiki.tum.de/display/lfdv/Recurrent%2BNeural%2BNetworks%2B-%2BCombination%2Bof%2BRNN%2Band%2BCNN)”, Published on 7 Feb 2017,<https://wiki.tum.de/display/lfdv/Recurrent+>

[Neural+Networks+Combination+of+RNN+and+CNN,](https://wiki.tum.de/display/lfdv/Recurrent%2BNeural%2BNetworks%2BCombination%2Bof%2BRNN%2Band%2BCNN) accessed Nov 2017

7. [Dmytro Nasyrov](https://towardsdatascience.com/%40pharosprod?source=post_header_lockup) , “Behavioral Cloning. NVidia Neural Network in Action.” , Published on 21 Aug 2017,<https://towardsdatascience.com/behavioral->

[cloning-project-3-6b7163d2e8f9](https://towardsdatascience.com/behavioral-cloning-project-3-6b7163d2e8f9) , accessed Jan 2017

8. Sihan Li , “Demystifying ResNet”, Published on 20 May 2017,[https://arxiv.](https://arxiv.org/)

[org/](https://arxiv.org/)[abs/1611.01186](https://arxiv.org/abs/1611.01186), accessed Jan 2017

9. Franchois Chollet, “Building powerful image classification models using very little data”, Published on 5 June 2016,<https://blog.keras.io/building>

[-powerful-image-classification-models-using-very-little-data.html](https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html), accessed Jan 2017

10.  [Ivan Kazakov,](https://towardsdatascience.com/%40ivankazakov?source=post_header_lockup) “Vehicle Detection and Tracking”, Published on 14 May 2017,<https://towardsdatascience.com/vehicle-detection-and-tracking-44b851d70508>, accessed Feb 2017