Department of Electrical Engineering
RCC INSTITUTE OF INFORMATION TECHNOLOGY
CANAL SOUTH ROAD, BELIAGHATA, KOLKATA - 700015, WEST BENGAL

Deep Learning Based Implementation for Self -Driven Cars

A comprehensive project report has been submitted in partial fulfillment of the requirements for the degree of

by

Name	Roll No.	Registration No:
Swikriti Singh	**11701619023**	**037723 OF 2019-2020**
Abhimanyu Prasad	**11701619024**	**037719 OF 2019-2020**
Tanay Das	**11701620028**	**201170101620008 OF 2020-2021**

Under the supervision of

Dr.Alok Kole
Professor (Dept. Of EE)

Department of Electrical Engineering
RCC INSTITUTE OF INFORMATION TECHNOLOGY
CANAL SOUTH ROAD, BELIAGHATA, KOLKATA - 700015, WEST BENGAL
Maulana Abul Kalam Azad University of Technology (MAKAUT)

Deep Learning Based Implementation for Self -Driven Cars

A comprehensive project report has been submitted in partial fulfillment of the requirements for the degree of

by

Name	Roll No.	Registration No:
Swikriti Singh	**11701619023**	**037723 OF 2019-2020**
Abhimanyu Prasad	**11701619024**	**037719 OF 2019-2020**
Tanay Das	**11701620028**	**201170101620008 OF 2020-2021**

Under the supervision of

Dr.Alok Kole
Professor (Dept. Of EE)

Department of Electrical Engineering
RCC INSTITUTE OF INFORMATION TECHNOLOGY
CANAL SOUTH ROAD, BELIAGHATA, KOLKATA - 700015, WEST BENGAL
Maulana Abul Kalam Azad University of Technology (MAKAUT)

ACKNOWLEDGEMENT

It is my great fortune that I have got opportunity to carry out this project work under the supervision of **Dr. Alok Kole** in the Department of Electrical Engineering, RCC Institute of Information Technology (RCCIIT), Canal South Road, Beliaghata, Kolkata-700015, affiliated to *Maulana Abul Kalam Azad University of Technology (MAKAUT)*, West Bengal, India. I express my sincere thanks and deepest sense of gratitude to my guide for his constant support, unparalleled guidance and limitless encouragement.

I wish to convey my gratitude to Prof. (Dr.) Shilpi Bhattacharya, HOD, Department of Electrical Engineering, RCCIIT and to the authority of RCCIIT for providing all kinds of infrastructural facility towards the research work.

I would also like to convey my gratitude to all the faculty members and staffs of the Department of Electrical Engineering, RCCIIT for their whole hearted cooperation to make this work turn into reality.

Signature of Students		

Place:

Date:

CERTIFICATE
To whom it may concern

This is to certify that the project work entitled **Deep Learning Based Implementation For Self Driven Cars** is the bona fide work carried out by **Swikriti Singh(11701619023) Abhimanyu Prasad(11701619024) Tanay Das(11701620028)**, a student of B.Tech in the Dept. of Electrical Engineering, RCC Institute of Information Technology (RCCIIT), Canal South Road, Beliaghata, Kolkata-700015, affiliated to Maulana Abul Kalam Azad University of Technology (MAKAUT), West Bengal, India, during the academic year 2022-23, in partial fulfillment of the requirements for the degree of Bachelor of Technology in Electrical Engineering and this project has not submitted previously for the award of any other degree, diploma and fellowship.

Signature of the Guide
Name:Dr. Alok Kole
Designation: Professor
(Dept. of EE, RCCIIT)

Signature of the HOD, EE
Name:Dr. Shilpi Bhattacharya Deb
Designation:HOD
(Dept. of EE, RCCIIT)

Signature of the External Examiner
Name:
Designation:

CONTENTS

CHAPTER:1 INTRODUCTION

Going by means of vehicle is at present one of the most perilous kinds of transportation with over a million passing every year around the world. As nearly all car crashes (particularly fatal ones) are caused by driver error, driverless vehicles would viably dispose of about all dangers related to driving just as driver fatalities, road safety, mobility for everyone, and injuries. The self-driving vehicle is the dormant beast that can make a huge difference. The selfdriving vehicle is a vehicle outfitted with an autopilot framework and is equipped for driving without the help of human administrator. This innovation was fabricated initially using autonomy approach yet with the progress in the field of PC vision and ai we can utilize the deep learning approach. There are a couple of troubles that need to have been met before executing self-driving car. One of the most important functionalities of a self-driving vehicle is autonomous lateral control. Autonomous lateral motion traditionally depends on image processing. The process is divided into detecting byways, locating by way centers, track planning, track following, and a control logic.The precision of such frameworks depends essentially on how well-tuned the picture processing filters are. These strategies are amazingly sensitive to assortments in lighting and are slanted to false identification. An algorithm tuned to detect lane markings in a place where it is bright and sunny may not do well in a place where it is dark and gloomy. This framework utilizes a convolutional neural system to yield controlling points dependent on street images basically the model is set up to copy human driving behavior. Since an end-to-end model doesn't take a shot at truly described standards, it is less affected by changes in lighting conditions.

1.1 BACKFROUND OF STUDY AND MOTIVATION

There are many motivations for self-driving cars. Some of the most common motivations include:

Safety: Self-driving cars have the potential to significantly reduce traffic accidents. According to the National Highway Traffic Safety Administration (NHTSA), human error is a factor in 94% of all traffic accidents. Self-driving cars, which are not susceptible to human error, could potentially eliminate this major cause of accidents.

Increased productivity: Self-driving cars could free up time for people to do other things while they are traveling. This could lead to increased productivity in the workplace and in other areas of life.

Economical benefits: Self-driving cars could lead to economic benefits in a number of ways. For example, they could reduce the cost of transportation, increase fuel

efficiency, and create new jobs in the transportation industry.

Environmental benefits: Self-driving cars could have a positive impact on the environment. For example, they could reduce emissions of greenhouse gases and other pollutants.

In addition to these motivations, self-driving cars also have the potential to improve accessibility for people with disabilities and to make transportation more affordable for people in rural areas.

While there are many potential benefits to self-driving cars, there are also some challenges that need to be addressed before they can become a reality. These challenges include developing safe and reliable self-driving technology, ensuring that self-driving cars are accessible to everyone, and addressing the legal and regulatory issues surrounding self-driving cars.

Despite the challenges, self-driving cars have the potential to revolutionize transportation. They could make our roads safer, our lives more productive, and our environment cleaner.

1.2 OBJECTIVE

Self-driving cars promise a range of potential benefits such as:

- **Greater Road Safety**

1. According to the survey done by the government, 94% of the accidents are caused by driver's behaviour and carelessness. Self-Driving vehicles can reduce this problem.

2. One of the greatest achievements may be reduction of impaired driving, drugged driving, unbelted vehicle occupants, speeding and distraction.

- **Greater Freedom**

1. Specially abled people will also be able to drive without being dependent on someone else and live the life they want.

2. Senior citizens can also be able to travel anywhere they want.

3. It can reduce the cost of personal transportation by providing affordable mobility.

- **Saving Money**

1. Self-Driven vehicles can avoid the costs of accidents, including

medical bills, lost work time and vehicle repair.

2. Fewer accidents may reduce the cost of insurance.

- **Reduced Congestion**

1. Fewer crashes mean fewer roadway backups.

2. Self driven vehicles maintain a safe and consistent distance between vehicles, helping to reduce the number of stop-and-go waves that produce road congestion

- **Environmental Gains**

1. Fewer traffic jams save fuel and reduce greenhouse gases from needless idling.

TECHNOLOGIES USED

Technologies that are used in the implementation of this project are

- TensorFlow
- Keras
- Numpy
- Scikit-learn
- OpenCV
- MiniConda

CONCEPTS USED

- CNN (Convolutional Neural Network)
- RNN (Recurrent Neural Network)
- Time Distributed Layers
- RCNN

CHAPTER:2 LITERATURE REVIEW

The ideas of self-driving cars date back to the middle ages and has grown into a reality in the contemporary society. In the middle ages, Leonardo Da

Vinci first drew a basic impression ofa driverless a car but it was rather simple. In the 20th century, significant steps were made in the invention of self-driving cars when Houdina Radio Control company first demonstrated a driverless car in 1925 (Kang et al. 32). The car was navigated by radio on a traffic route along Broadway. Despite their impact, at the time, the cars were mostly for exhibition (Kang et al. 41). The growth of self-driving cars is clearly demonstrated by its history and evolution, self-car driving tests and future projections.The first bold step in self-driving car technology was made by Norman Bel Geddes who came up with the Idea of an automated highway system. According to Boeglin (171) the system allowed cars to move autonomously over a highway fitted with electronic circuits. The project however as Birdsall (47) failed due to insufficient funds. An akin project came up in the United Kingdom as revealed by Kang et al (42) where they used a more advanced driverless system whereby both the car and roads were automated, however Kang et al (43) further stipulates that the breakthrough suffered a fate similar to Norman Bel Geddes' idea. Boeglin (172) explains thatthe cost had become a recurring bottleneck in the construction of self-driven cars, suggesting thatthe focus had to shift from automated roads to smarter cars. In 1960 Fagnant & Kockelman (34) point out that a Stanford engineering student, James Adams became the first to implement the shift when he designed a remote-controlled Lunar Rover. Birdsall (51) cites a revolutionary idea in 1977 by Japan's Tsukuba Mechanical Engineering Lab that led to the existence of standalone self-driving systems which did not depend on remote control technology.

the concept of machine vision for navigation (Birdsall 52). In the 80's aerospace engineer Ericks Dickmanns applied artificial intelligence to self-transportation and derived a concept that could lead to electric cars moving at high speeds. Fagnant & Kockelman (53) posit that the innovation named the VAMORS prototype which used a computer program and cameras was tested successfully in 1986

CHAPTER:3 METHODOLOGIES

TensorFlow

TensorFlow is a Python library for fast numerical computing created and released by Google. It is a foundation library that can be used to create Deep Learning models directly or by using wrapper libraries that simplify the process built on top of TensorFlow.

This an open-source library for dataflow programming. It is widely used for machine learning applications. It is also used as both a math library and for large computation.

Keras

A high-level API that uses TensorFlow as the backend is used. Keras facilitate in building the models easily as it more user friendly.

It is written in Python and is used to make the implementation of neural networks easy. It also supports multiple backend neural network computation.

Numpy

It provides with high-level math function collection to support multi-dimensional metrices and arrays. This is used for faster computations over the weights (gradients) in neural networks.

In Python we have lists that serve the purpose of arrays, but they are slow to process.

NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.

Scikit-learn

It is a machine learning library for Python which features different algorithms and Machine Learning function packages.

Scikit-learn is an open source data analysis library, and the gold standard for Machine Learning (ML) in the Python ecosystem. Key concepts and features include Algorithmic decision-making methods, including:

Classification: identifying and categorizing data based on patterns.

Regression: predicting or projecting data values based on the average mean of existing and planned data.

Clustering: automatic grouping of similar data into datasets.

OpenCV (Open Source Computer Vision Library)

It is designed for computational efficiency with focus on real-time 4 applications. In this project, OpenCV is used for image preprocessing and augmentation techniques.

OpenCV is a Python library that allows you to perform image processing and computer vision tasks. It provides a wide range of features, including object detection, face recognition, and tracking.

MiniConda Environment

It is an open source distribution for Python which simplifies package management and deployment. It is best for large scale data processing.

Miniconda is a free minimal installer for conda. It is a small, bootstrap version of Anaconda that includes only conda, Python, the packages they depend on, and a small number of other useful packages, including pip, zlib and a few others.

CONCEPTS USED IN THE PROJECT

2.1 Convolutional Neural Networks (CNN)

It is a type of feed-forward neural network computing system that can be used to learn from input data. Learning is accomplished by determining a set of weights or filter values that allow the network to model the behavior according to the training data. The desired output and the output generated by CNN initialized with random weights will be different. This difference (generated error) is backpropagated through the layers of CNN to adjust the weights of the neurons, which in turn reduces the error and allows us produce output closer to the desired one .

CNN is good at capturing hierarchical and spatial data from images. It utilizes filters that look at regions of an input image with a defined window size and map it to some output. It then slides the window by some defined stride to other regions, covering the whole image. Each convolution filter layer thus captures the properties of this input image hierarchically in a series of subsequent layers, capturing the details like lines in image, then shapes, then whole objects in later layers. CNN can be a good fit to feed the images of a dataset and classify them into their respective classes.

2.2 Recurrent Neural Networks (RNN)

RNN are a class of artificial neural networks where connections between units form a directed cycle. Recurrent networks, unlike feedforward networks, have the feedback loop connected to their past decisions, ingesting their own outputs as input (like a memory). This memory (feedback) helps to learn sequences and predict subsequent values, thus being able to solve dependencies over time. For example, consider the case when the next

word in a sentence is dependent on a previously occurring word or context. RNN will be an excellent choice for such scenarios. They are designed to recognize patterns in sequences of data, such as text, handwriting and so on. They are also applicable to images that can be separated (decomposed) into a sequence of patches.

Neural networks have activation functions to take care of the non-linearity and to squash the gradients or weights in certain range. Some of these functions are sigmoid, tanh, RELU and so on that are the building blocks of RNN. Though these are very powerful, there are some shortcomings in conventional RNN, such as the well-known problem of vanishing or exploding gradients.Also, training might take a very long time.

2.3 Time-Distributed Layers

Another type of layers sometimes used in deep learning networks is a Timedistributed layer. Time-Distributed layers are provided in Keras as wrapper layers. Every temporal slice of an input is applied with this wrapper layer. The requirement for input is that to be at least three-dimensional, first index can be considered as temporal dimension. These Time-Distributed can be applied to a dense layer to each of the timesteps, independently or even used with Convolutional Layers.

```
# as the first layer in a model
model = Sequential()
model.add(TimeDistributed(Dense(8), input_shape=(10, 16)))
# now model.output_shape == (None, 10, 8)
```

Fig. 3: TimeDistributed Dense layer

```
model = Sequential()
model.add(TimeDistributed(Conv2D(64, (3, 3)),
                          input_shape=(10, 299, 299, 3)))
```

Fig. 4: TimeDistributed Convolution layer

2.4 RCNN (Combination of CNN and RNN)

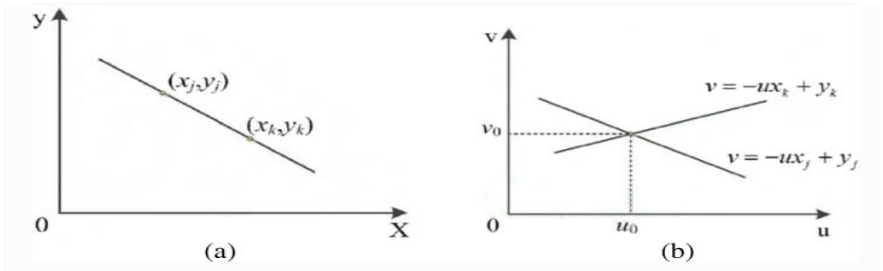
The acronym used to denote this combination as RCNN (Recurrent Convolutional Neural Networks). In recent times, there have been many implementations using RCNN. There is another abbreviation for this term (R-CNN) as region-based CNN which is a popular technique for object detection in images. In this project, every time this term is used, it will refer to Recurrent CNNs.

There are several techniques or methods for which this combination can be realized. Individually, both CNN and RNN are extremely useful in image classification (more about spatial characteristics of data) and sequence prediction (temporal characteristics of data). The hybrid models can have a bunch of convolution layers and another branch of RNN (may include LSTM or GRU or both) in parallel or they can be stacked in series. In this project, there are experiments for a variety of architectures. There are results plotted by different implementations that were tried on the driving dataset .

Edge Detection of Road Images (Hough Space Algorithm):

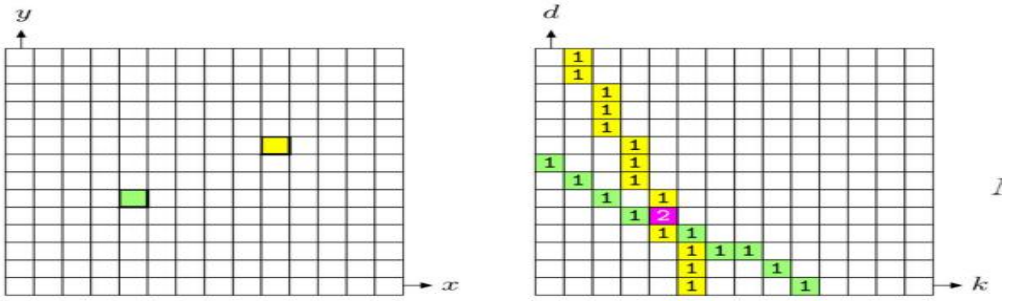
Lane detection is one of the basic concepts when trying to understand how self-driving cars work. We built a program that can identify lane lines in a picture or a video and learnt how the Hough transform plays a huge role in achieving this task. Hough transform comes almost as the last step in detecting straight lines in the region of interest.

The picture we have right now is just a series of pixels and we cannot find a geometrical representation directly to know the slope and intercepts.
Since images are never perfect, we cannot loop through the pixels to find the slope and intercept since it would be a very difficult task. This is where Hough transform can be used. It helps us to figure out the prominent lines and connect the disjoint edge points in an image.
Its principle is to convert the detection problem from the image space to the parameter space and find the peak of the accumulator in the parameter space so as to achieve the target detection.



Hough transform in Cartesian coordinates. **a** Image coordinate space. **b** Parameter space

With the aid of accumulator cells of pixels matrices, we joined the points of the image space to obtain the joining outline of the edge image. This was accomplished by voting the intersections of parameterized space to obtain the edge outline.



(a) Image Space (b) Accumulator Array

3.1 UDACITY SIMULATOR AND DATASET

Udacity has built a simulator for self-driving cars and made it open source for the enthusiasts, so they can work on something close to a real-time environment. It is built on Unity, the video game development platform. The simulator consists of a configurable resolution and controls setting and is very user friendly.The graphics and input configurations can be changed according to user preference and machine configuration.

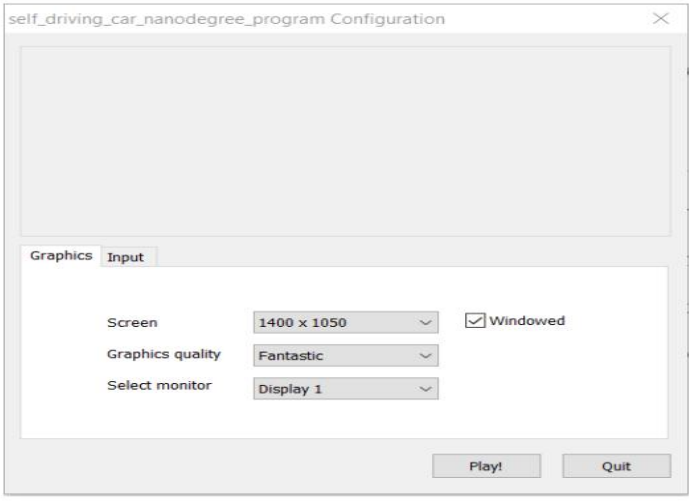


Fig 5. Configuration screen

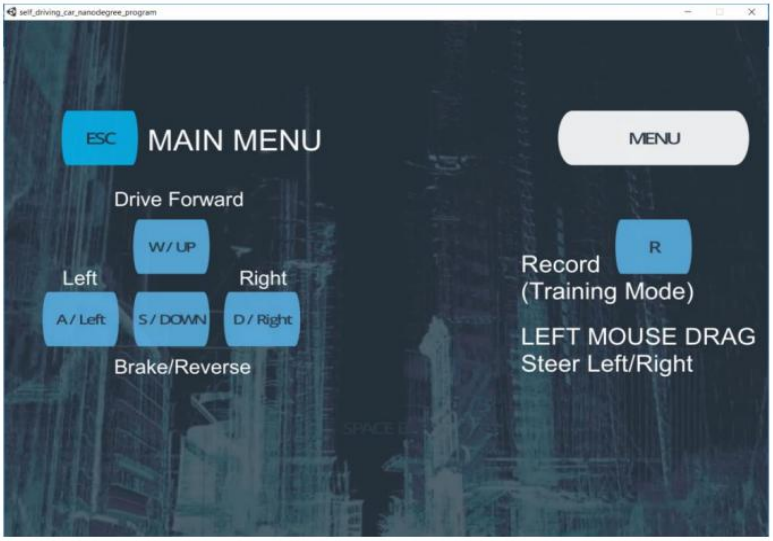


Fig. 6: Controls Configuration

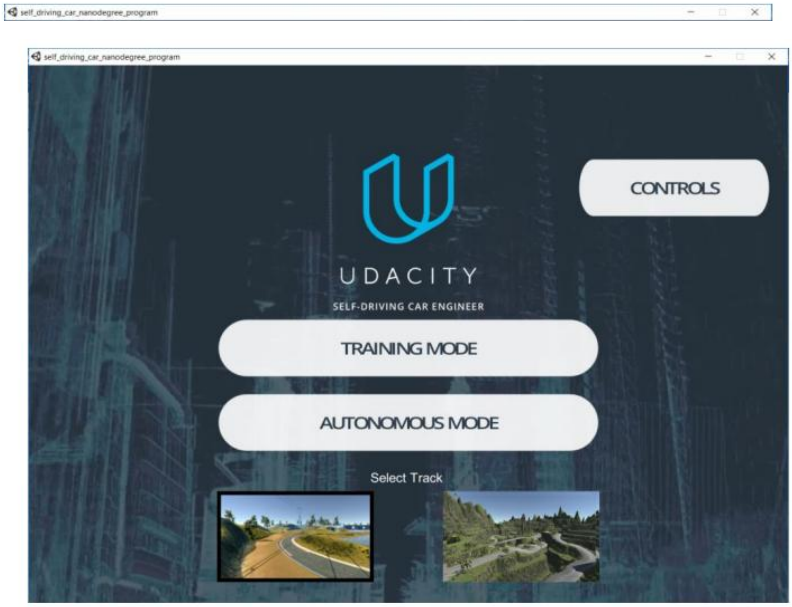


Fig. 7: First Screen

The simulator involves two tracks. One of them can be considered as simple and another one as complex that can be evident in the screenshots attached


self_driving_car_nanodegree_program
ANGLE
0.00°
RECORD
0.00
MPH

Fig 8. Track_1

There are two modes for driving the car in the simulator: (1) Training mode and (2) Autonomous mode. The training mode gives you the option of recording your run and capturing the training dataset. The small red sign at the top right of the screen in the Figure 9 depicts the car is being driven in training mode. The autonomous mode can be used to test the models to see if it can drive on the track without human intervention. Also, if you try to press the controls to get the car back on track, it will immediately notify you that it shifted to manual controls.

Fig 10. Autonomous mode

The simulator's feature to create your own dataset of images makes it easy to work on the problem. Some reasons why this feature is useful are as follows:

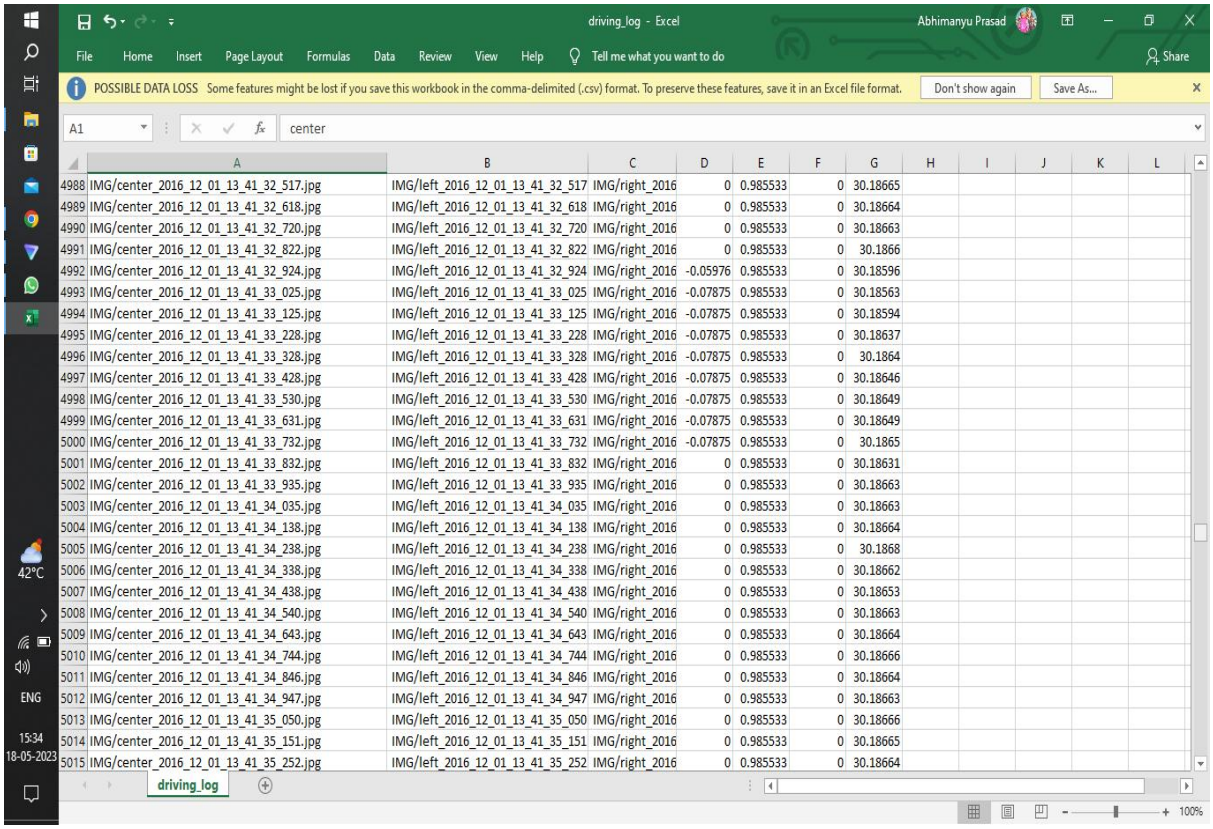
• The simulator has built the driving features in such a way that it simulates that there are three cameras on the car. The three cameras are in the center, right and left on the front of the car, which captures continuously when we record in the training mode.

• The stream of images is captured, and we can set the location on the disk for saving the data after pushing the record button. The image set are labelled in a sophisticated manner with a prefix of center, left, or right indicating from which camera the image has been captured.

• Along with the image dataset, it also generates a datalog.csv file. This file contains the image paths with corresponding steering angle, throttle, brakes, and speed of the car at that instance.

Fig 11. Dataset sample

Column 1, 2, 3: contains paths to the dataset images of center, right and left respectively
Column 4: contains the steering angle Column value as 0 depicts straight, positive value is right turn and negative value is left turn.
Column 5: contains the throttle or acceleration at that instance
Column 6: contains the brakes or deceleration at that instance
Column 7: contains the speed of the vehicle



	A	B	C	D	E	F	G
4988	IMG/center_2016_12_01_13_41_32_517.jpg	IMG/left_2016_12_01_13_41_32_517	IMG/right_2016	0	0.985533	0	30.18665
4989	IMG/center_2016_12_01_13_41_32_618.jpg	IMG/left_2016_12_01_13_41_32_618	IMG/right_2016	0	0.985533	0	30.18664
4990	IMG/center_2016_12_01_13_41_32_720.jpg	IMG/left_2016_12_01_13_41_32_720	IMG/right_2016	0	0.985533	0	30.18663
4991	IMG/center_2016_12_01_13_41_32_822.jpg	IMG/left_2016_12_01_13_41_32_822	IMG/right_2016	0	0.985533	0	30.1866
4992	IMG/center_2016_12_01_13_41_32_924.jpg	IMG/left_2016_12_01_13_41_32_924	IMG/right_2016	-0.05976	0.985533	0	30.18596
4993	IMG/center_2016_12_01_13_41_33_025.jpg	IMG/left_2016_12_01_13_41_33_025	IMG/right_2016	-0.07875	0.985533	0	30.18563
4994	IMG/center_2016_12_01_13_41_33_125.jpg	IMG/left_2016_12_01_13_41_33_125	IMG/right_2016	-0.07875	0.985533	0	30.18594
4995	IMG/center_2016_12_01_13_41_33_228.jpg	IMG/left_2016_12_01_13_41_33_228	IMG/right_2016	-0.07875	0.985533	0	30.18637
4996	IMG/center_2016_12_01_13_41_33_328.jpg	IMG/left_2016_12_01_13_41_33_328	IMG/right_2016	-0.07875	0.985533	0	30.1864
4997	IMG/center_2016_12_01_13_41_33_428.jpg	IMG/left_2016_12_01_13_41_33_428	IMG/right_2016	-0.07875	0.985533	0	30.18646
4998	IMG/center_2016_12_01_13_41_33_530.jpg	IMG/left_2016_12_01_13_41_33_530	IMG/right_2016	-0.07875	0.985533	0	30.18649
4999	IMG/center_2016_12_01_13_41_33_631.jpg	IMG/left_2016_12_01_13_41_33_631	IMG/right_2016	-0.07875	0.985533	0	30.18649
5000	IMG/center_2016_12_01_13_41_33_732.jpg	IMG/left_2016_12_01_13_41_33_732	IMG/right_2016	-0.07875	0.985533	0	30.1865
5001	IMG/center_2016_12_01_13_41_33_832.jpg	IMG/left_2016_12_01_13_41_33_832	IMG/right_2016	0	0.985533	0	30.18631
5002	IMG/center_2016_12_01_13_41_33_935.jpg	IMG/left_2016_12_01_13_41_33_935	IMG/right_2016	0	0.985533	0	30.18663
5003	IMG/center_2016_12_01_13_41_34_035.jpg	IMG/left_2016_12_01_13_41_34_035	IMG/right_2016	0	0.985533	0	30.18663
5004	IMG/center_2016_12_01_13_41_34_138.jpg	IMG/left_2016_12_01_13_41_34_138	IMG/right_2016	0	0.985533	0	30.18664
5005	IMG/center_2016_12_01_13_41_34_238.jpg	IMG/left_2016_12_01_13_41_34_238	IMG/right_2016	0	0.985533	0	30.1868
5006	IMG/center_2016_12_01_13_41_34_338.jpg	IMG/left_2016_12_01_13_41_34_338	IMG/right_2016	0	0.985533	0	30.18662
5007	IMG/center_2016_12_01_13_41_34_438.jpg	IMG/left_2016_12_01_13_41_34_438	IMG/right_2016	0	0.985533	0	30.18653
5008	IMG/center_2016_12_01_13_41_34_540.jpg	IMG/left_2016_12_01_13_41_34_540	IMG/right_2016	0	0.985533	0	30.18663
5009	IMG/center_2016_12_01_13_41_34_643.jpg	IMG/left_2016_12_01_13_41_34_643	IMG/right_2016	0	0.985533	0	30.18664
5010	IMG/center_2016_12_01_13_41_34_744.jpg	IMG/left_2016_12_01_13_41_34_744	IMG/right_2016	0	0.985533	0	30.18666
5011	IMG/center_2016_12_01_13_41_34_846.jpg	IMG/left_2016_12_01_13_41_34_846	IMG/right_2016	0	0.985533	0	30.18664
5012	IMG/center_2016_12_01_13_41_34_947.jpg	IMG/left_2016_12_01_13_41_34_947	IMG/right_2016	0	0.985533	0	30.18663
5013	IMG/center_2016_12_01_13_41_35_050.jpg	IMG/left_2016_12_01_13_41_35_050	IMG/right_2016	0	0.985533	0	30.18666
5014	IMG/center_2016_12_01_13_41_35_151.jpg	IMG/left_2016_12_01_13_41_35_151	IMG/right_2016	0	0.985533	0	30.18665
5015	IMG/center_2016_12_01_13_41_35_252.jpg	IMG/left_2016_12_01_13_41_35_252	IMG/right_2016	0	0.985533	0	30.18664

3.2 EXPERIMENTAL CONFIGURATIONS

This section consists of the configurations used to set up the models for training the Python Client to provide the Neural Network outputs that drive the car on the simulator. The tweaking of parameters and rigorous experiments were tried to reach the best combination. Though each of the models had their unique behaviors and differed in their performance with each tweak, the following combination of configuration can be considered as the optimal:

• The sequential models built on Keras with deep neural network layers are used to train the data.

• Models are only trained using the dataset from Track_1.

• 80% of the dataset is used for training, 20% is used for testing.

• Epochs = 50, i.e. number of iterations or passes through the complete dataset. Experimented with larger number of epochs also, but the model tried to “overfit”. In other words, the model learns the details in the training data too well, while impacting the performance on new dataset.

• Batch-size = 40, i.e. number of image samples propagated through the network, like a subset of data as complete dataset is too big to be passed all at once.

• Learning rate = 0.0001, i.e. how the coefficients of the weights or gradients change in the network. 19

• ModelCheckpoint() is the function provided in Keras to save checkpoints and to save the best epoch according to the validation loss. There are different combinations of Convolution layer, Time-Distributed layer, MaxPooling layer, Flatten, Dropout, dense and so on, that can be used to implement the Neural Network models. Out of around ten different architectures I tried, three of the best ones are discussed in the chapter on network architectures.

3.3 NETWORK ARCHITECTURES

There were various combinations of architectures tried, predicting the steering angle and input for the car to drive in autonomous mode. Neural Network layers were organized in series and various combinations of Time-Distributed Convolution layers, MaxPooling, Flatten, Dropout, Dense and so on are used in architectures. The best performing ones are shown in detail. Refer to the model listings for the parameters used to build them. The high-level view of layers used to build the models is shown in the accompanying architecture figures.

Model 1:

```
model = Sequential()
model.add(Lambda(lambda x: x/127.5-1.0, input_shape=INPUT_SHAPE))
model.add(Conv2D(24, 5, 5, activation='elu', subsample=(2, 2)))
model.add(Conv2D(36, 5, 5, activation='elu', subsample=(2, 2)))
model.add(Conv2D(48, 5, 5, activation='elu', subsample=(2, 2)))
model.add(Conv2D(64, 3, 3, activation='elu'))
model.add(Conv2D(64, 3, 3, activation='elu'))
model.add(Dropout(args.keep_prob))
model.add(Flatten())
model.add(Dense(100, activation='elu'))
model.add(Dense(50, activation='elu'))
model.add(Dense(10, activation='elu'))
model.add(Dense(1))
model.summary()
```

Fig 14. Model 1

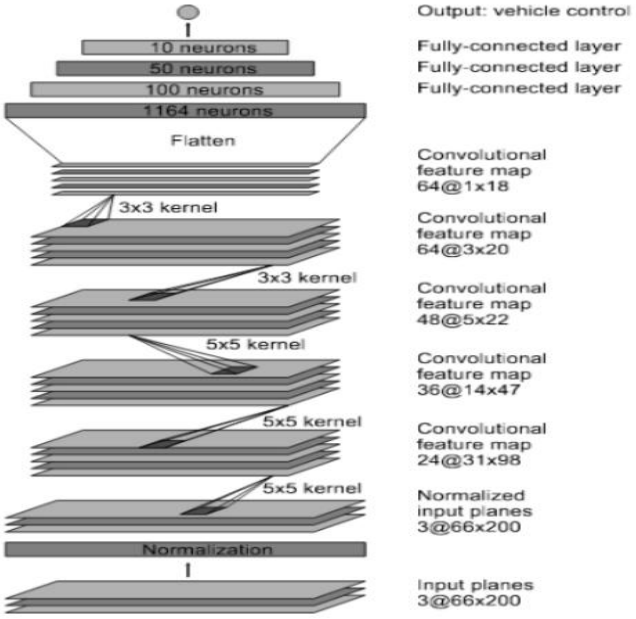


Fig 15. Architecture 1, [4]

NVIDIA released an architecture for self-driving cars [4] and it is used in the project for reference to solve the problem and for comparing with the various other architectures tried. Different architectures have been standardized over the years for building sequential models of CNN like AlexNet, VGG-Net, GoogLeNet, ResNet and so on. Model_2 is an architecture similar to AlexNet, with a slight variation by tweaking parameters to suit the problem in the project.

Model 2:

```
model = Sequential()
model.add(Lambda(lambda x: x/127.5-1.0, input_shape = INPUT_SHAPE))
model.add(Conv2D(32, 3, 3, activation='elu', subsample=(2, 2)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(64, 3, 3, activation='elu', subsample=(2, 2)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(128, 3, 3, activation='elu', subsample=(2, 2)))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1))
model.summary()
```

Fig 16. Model 2

Fig 17. Architecture 2

Model_3 is an architecture similar to VGG-Net, with variations by tweaking parameters to suit the problem in the project

Model 3:

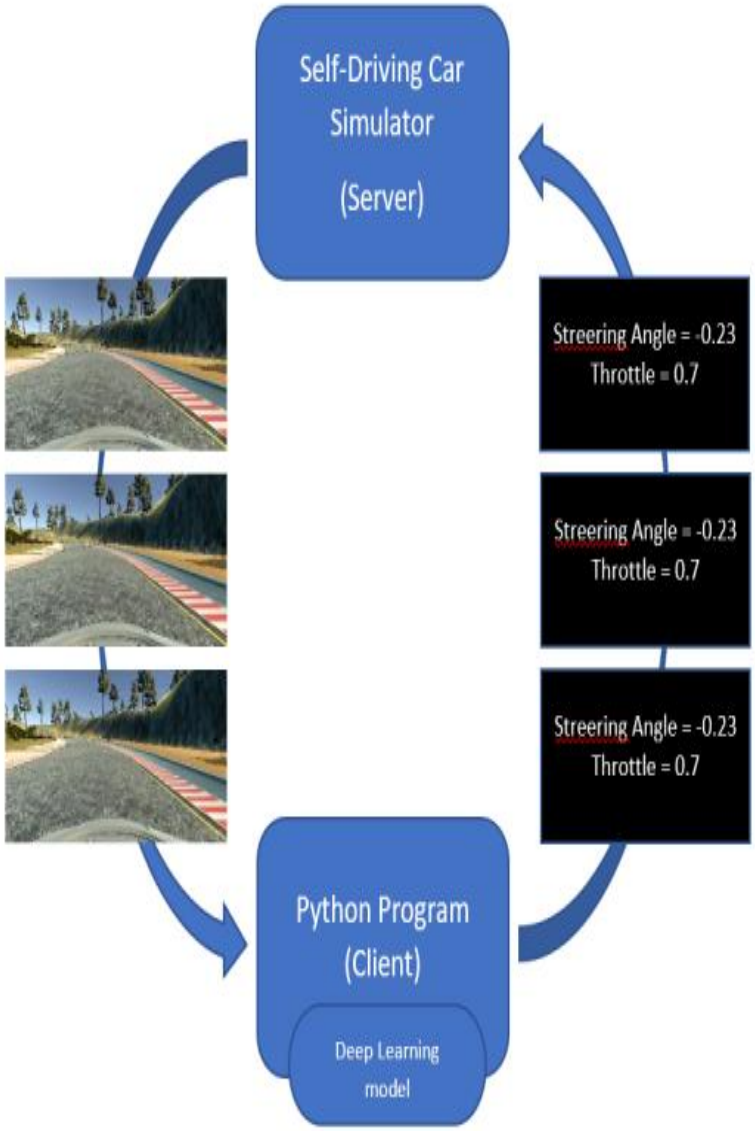
```
model = Sequential()
model.add(Lambda(lambda x: x/255.-0.5,input_shape=input_shape))
model.add(Convolution2D(3,1,1, name='conv0'))
model.add(Convolution2D(32,filter_size,filter_size, name='conv1'))
model.add(ELU())
model.add(Convolution2D(32,filter_size,filter_size, name='conv2'))
model.add(ELU())
model.add(MaxPooling2D(pool_size=pool_size))
model.add(Dropout(0.5))
model.add(Convolution2D(64,filter_size,filter_size, name='conv3'))
model.add(ELU())
model.add(Convolution2D(64,filter_size,filter_size, name='conv4'))
model.add(ELU())
model.add(MaxPooling2D(pool_size=pool_size))
model.add(Dropout(0.5))
model.add(Convolution2D(128,filter_size,filter_size,name='conv5'))
model.add(ELU())
model.add(Convolution2D(128,filter_size,filter_size,,name='conv6'))
model.add(ELU())
model.add(MaxPooling2D(pool_size=pool_size))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(512))
model.add(ELU())
model.add(Dropout(0.5))
model.add(Dense(64))
model.add(ELU())
model.add(Dropout(0.5))
model.add(Dense(16))
model.add(ELU())
model.add(Dropout(0.5))
model.add(Dense(1))
```

Fig 18. Model 3

Fig 19. Architecture 3

CHAPTER:4(IMPLEMENTATION AND TESTING)

4.1 FLOWCHART



4.2 AUGMENTATION AND IMAGE PRE-PROCESSING

The biggest challenge was generalizing the behavior of the car on Track_2 which it was never trained for. In a real-life situation, we can never train a self-driving car model for every track possible, as the data will be too huge to process. Also, it is not possible to gather the dataset for all the weather conditions and roads. Thus, there is a need to come up with an idea of generalizing the behavior on different tracks. This problem is solved using image preprocessing and augmentation techniques, which will be discussed in the following section:

• Crop

The images in the dataset have relevant features in the lower part where the road is visible. The external environment above a certain image portion will never be used to determine the output and thus can be cropped. Approximately, 30% of the top portion of the image is cut and passed in the training set.

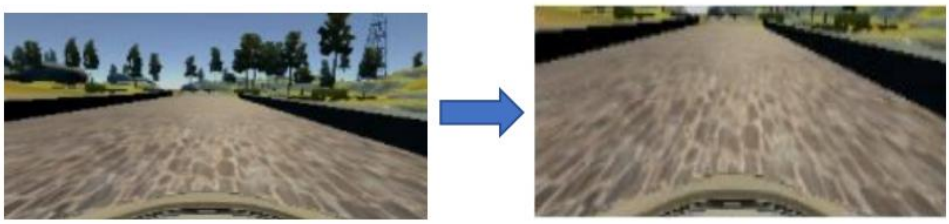


Fig 20. Crop image

```
def crop(self,x):
    x = cv2.cvtColor(x,cv2.COLOR_RGB_to_HSV)
    vertices = np.array([[(0,0),(200,0),(200,33),(0,33)]],np.int32)
    random_brightness = 0
    mask = np.zeros_like(x)

    ignore = [0,0,255]

    cv2.fillPoly(mask, vertices, ignore)

    indices = mask[:,:,2] == 255

    x[:,:,2][index] = x[:,:,2][index]*random_brightness

    x = cv2.cvtColor(x,cv2.COLOR_HSV_to_RGB)
    return x
```

Fig 20. Crop image (continued)

• Flip (horizontal)

The image is flipped horizontally (i.e. a mirror image of the original image is passed to the dataset). The motive behind this is that the model gets trained for similar kinds of turns on opposite sides too. This is important because Track_1 includes only left turns.

```
def flip_image(self,x,steering):
    steer = -steer
    x = np.array(x)
    return cv2.flip(x,1), steering
```

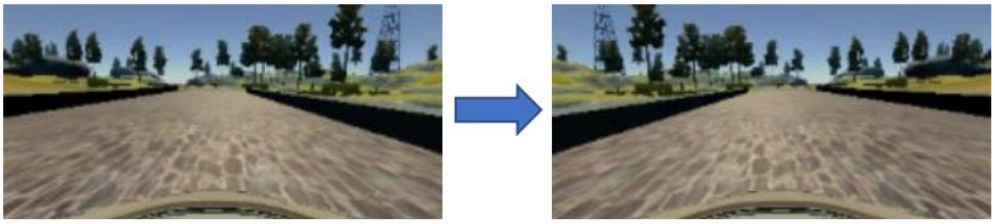


Fig 21. Flip image

• Shift (horizontal / vertical)

The image is shifted by a small amount

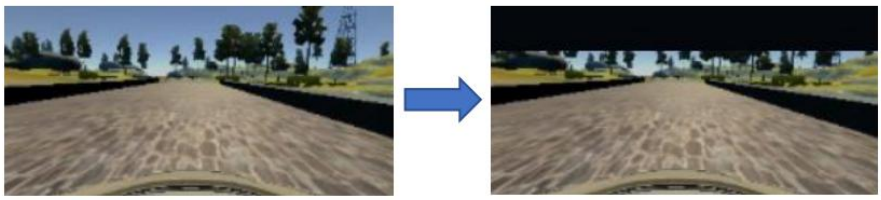


Fig 22. Shift image vertical

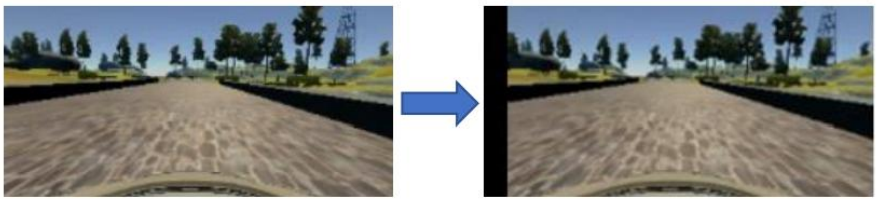


Fig 23. Shift image horizontal

• Brightness

To generalize to the weather conditions with bright sunny day or cloudy, lowlight conditions, the brightness augmentation can prove to be very useful.

```
def brightness(self,x):
    brightness_change = 0.5 + random()*1.5
    x = np.array(x)
    x = cv2.cvtColor(x,cv2.COLOR_RGB_to_HSV)
    x[:,:,2] = x[:,:,2]*brightness_change
    return cv2.cvtColor(x,cv2.COLOR_HSV_to_RGB)
```

Fig 24. Brightness increase

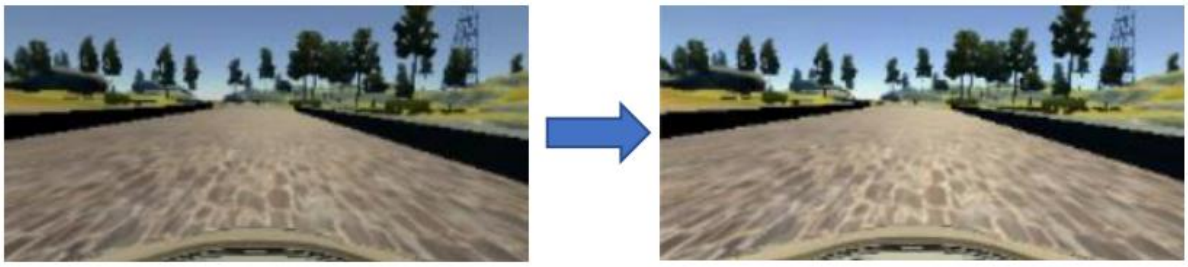


Fig 24. Brightness increase (continued)

• Shadows

Even after taking into considerations the light conditions, there are still chances that there are shadows on the road. This will give an instance of half lit and half lowlight scenes in the image. To cast random shadows and solve this shadow fitting problem, this augmentation is applied on the dataset.

```
def shadow_random(self,x):
    x = cv2.cvtColor(x,cv2.COLOR_RGB_to_HSV)
    max_x = 255
    max_y = 55
    vertices = np.array([[max_x,max_y,random()*max_x,random()*max_y]],
                        dtype = np.int32)
    x = self.region_of_interest(x,vertices)
    x = cv2.cvtColor(x,cv2.COLOR_HSV_to_RGB)
    return x
```

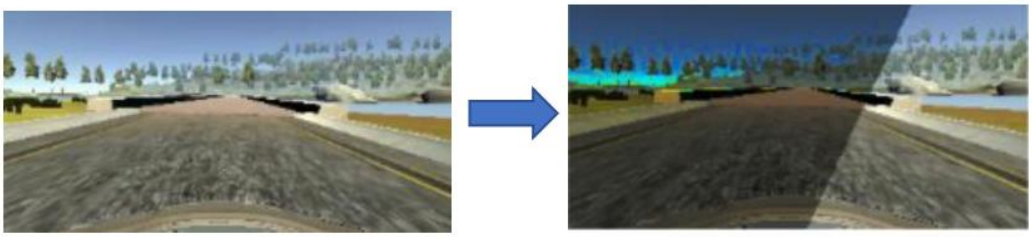


Fig 25. Random shadows

- 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.

```
def random_blur(self,x):
    size = 1+int(random()*7)
    if(size%2 == 0):
        size = size + 1
    x = cv2.GaussianBlur(x,(size,size),0)
    return x
```

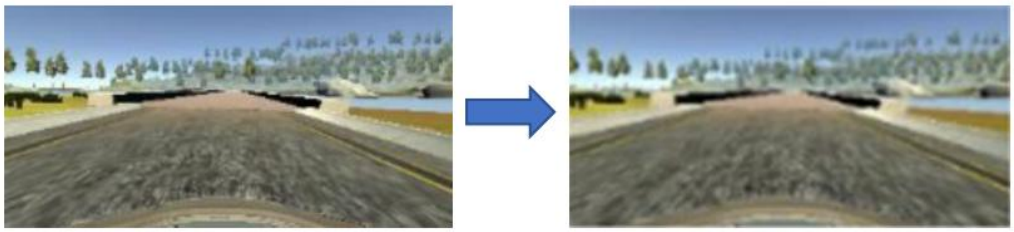


Fig 26. Random blur

• Noise

This adds random noise to the images by taking into consideration the unclean conditions by simulating dust or dirt particles and distortions while capturing the image.

```
def noise(self,x):
    return random_noise(x, mode='gaussian')
```

Fig 27. Random Noise

4.2 EXPERIMENTAL RESULTS

The following results were observed for each of the previously described architectures. For a comparison between them, we had to come up with two different performance metrics.

1. Value loss or Accuracy (computed during training phase)
2. Generalization on Track_2 (drive performance)

Value loss or Accuracy

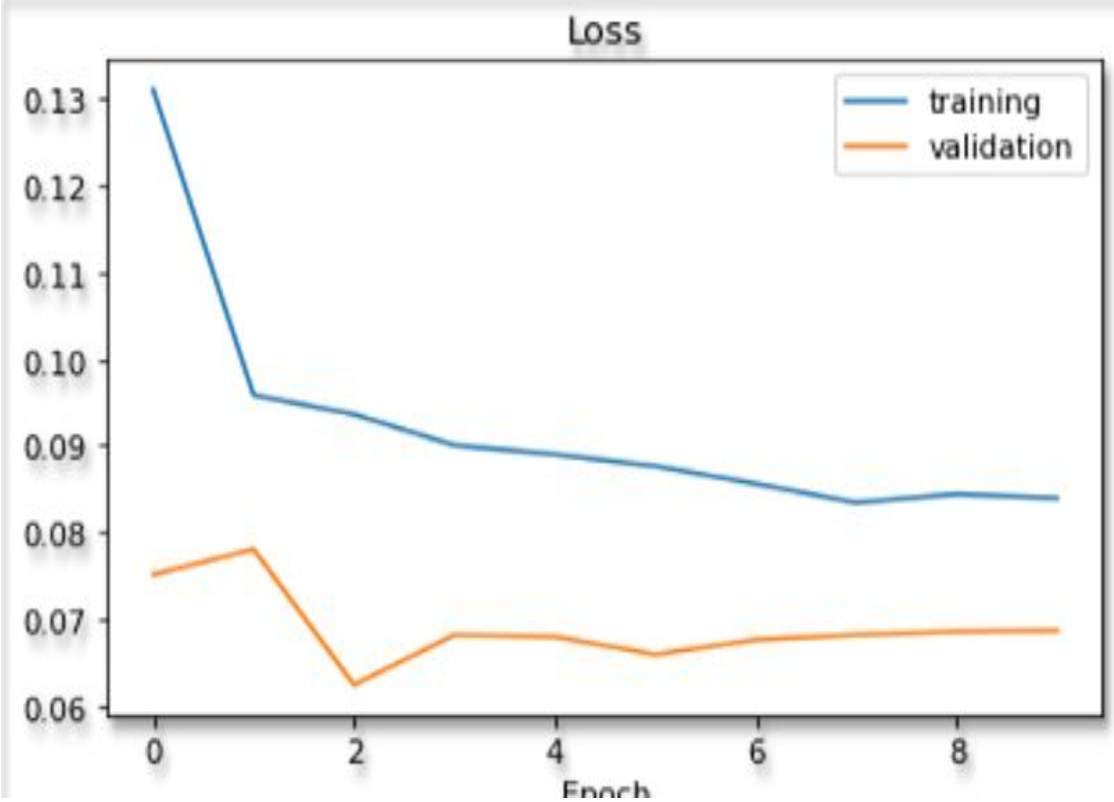
The first evaluation parameter considered here is “Loss” over each epoch of the training run. To calculate value loss over each epoch, Keras provides “val_loss”, which is the average loss after that epoch. The loss observed during the initial epochs at the beginning of training phase is high, but it falls gradually.

Layer (type)	Output Shape	Param #	Connected to
lambda_1 (Lambda)	(None, 66, 200, 3)	0	lambda_input_1[0][0]
convolution2d_1 (Convolution2D)	(None, 31, 98, 24)	1824	lambda_1[0][0]
convolution2d_2 (Convolution2D)	(None, 14, 47, 36)	21636	convolution2d_1[0][0]
convolution2d_3 (Convolution2D)	(None, 5, 22, 48)	43248	convolution2d_2[0][0]
convolution2d_4 (Convolution2D)	(None, 3, 20, 64)	27712	convolution2d_3[0][0]
convolution2d_5 (Convolution2D)	(None, 1, 18, 64)	36928	convolution2d_4[0][0]
dropout_1 (Dropout)	(None, 1, 18, 64)	0	convolution2d_5[0][0]
flatten_1 (Flatten)	(None, 1152)	0	dropout_1[0][0]
dense_1 (Dense)	(None, 100)	115300	flatten_1[0][0]
dense_2 (Dense)	(None, 50)	5050	dense_1[0][0]
dense_3 (Dense)	(None, 10)	510	dense_2[0][0]
dense_4 (Dense)	(None, 1)	11	dense_3[0][0]

```
these are available on your machine and could speed up CPU computations.
20000/20000 [==============================] - 7260s - loss: 0.1631 - val_loss: 0.1205
Epoch 2/50
20000/20000 [==============================] - 194s - loss: 0.1491 - val_loss: 0.1063
Epoch 3/50
20000/20000 [==============================] - 194s - loss: 0.1352 - val_loss: 0.0966
Epoch 4/50
20000/20000 [==============================] - 203s - loss: 0.1261 - val_loss: 0.0900
Epoch 5/50
20000/20000 [==============================] - 221s - loss: 0.1204 - val_loss: 0.0771
Epoch 6/50
20000/20000 [==============================] - 224s - loss: 0.1143 - val_loss: 0.0864
Epoch 7/50
20000/20000 [==============================] - 221s - loss: 0.1084 - val_loss: 0.0764
Epoch 8/50
20000/20000 [==============================] - 221s - loss: 0.1070 - val_loss: 0.0910
Epoch 9/50
20000/20000 [==============================] - 221s - loss: 0.1013 - val_loss: 0.0813
Epoch 10/50
20000/20000 [==============================] - 221s - loss: 0.1010 - val_loss: 0.0730
Epoch 11/50
20000/20000 [==============================] - 222s - loss: 0.0989 - val_loss: 0.0962
Epoch 12/50
20000/20000 [==============================] - 221s - loss: 0.0920 - val_loss: 0.0762
```

```
Epoch 13/50
20000/20000 [==============================] - 221s - loss: 0.0908 - val_loss: 0.0714
Epoch 14/50
20000/20000 [==============================] - 221s - loss: 0.0902 - val_loss: 0.0762
Epoch 15/50
20000/20000 [==============================] - 221s - loss: 0.0881 - val_loss: 0.0887
Epoch 16/50
20000/20000 [==============================] - 221s - loss: 0.0855 - val_loss: 0.0875
Epoch 17/50
20000/20000 [==============================] - 221s - loss: 0.0834 - val_loss: 0.0656
Epoch 18/50
20000/20000 [==============================] - 1146s - loss: 0.0807 - val_loss: 0.0911
Epoch 19/50
20000/20000 [==============================] - 193s - loss: 0.0810 - val_loss: 0.0718
Epoch 20/50
20000/20000 [==============================] - 197s - loss: 0.0800 - val_loss: 0.0696
Epoch 21/50
20000/20000 [==============================] - 194s - loss: 0.0804 - val_loss: 0.0507
Epoch 22/50
20000/20000 [==============================] - 193s - loss: 0.0758 - val_loss: 0.0696
Epoch 23/50
20000/20000 [==============================] - 194s - loss: 0.0765 - val_loss: 0.0762
Epoch 24/50
20000/20000 [==============================] - 196s - loss: 0.0730 - val_loss: 0.0795
Epoch 25/50
20000/20000 [==============================] - 194s - loss: 0.0740 - val_loss: 0.0609
```

```
20000/20000 [==============================] - 194s - loss: 0.0703 - val_loss: 0.0617
Epoch 28/50
20000/20000 [==============================] - 194s - loss: 0.0696 - val_loss: 0.0636
Epoch 29/50
20000/20000 [==============================] - 197s - loss: 0.0699 - val_loss: 0.0692
Epoch 30/50
20000/20000 [==============================] - 194s - loss: 0.0683 - val_loss: 0.0554
Epoch 31/50
20000/20000 [==============================] - 193s - loss: 0.0675 - val_loss: 0.0660
Epoch 32/50
20000/20000 [==============================] - 193s - loss: 0.0647 - val_loss: 0.0678
Epoch 33/50
20000/20000 [==============================] - 194s - loss: 0.0656 - val_loss: 0.0481
Epoch 34/50
20000/20000 [==============================] - 196s - loss: 0.0645 - val_loss: 0.0574
Epoch 35/50
20000/20000 [==============================] - 193s - loss: 0.0633 - val_loss: 0.0584
Epoch 36/50
20000/20000 [==============================] - 194s - loss: 0.0636 - val_loss: 0.0566
Epoch 37/50
20000/20000 [==============================] - 194s - loss: 0.0627 - val_loss: 0.0613
Epoch 38/50
20000/20000 [==============================] - 194s - loss: 0.0625 - val_loss: 0.0690
Epoch 39/50
20000/20000 [==============================] - 194s - loss: 0.0617 - val_loss: 0.0774
Epoch 40/50
20000/20000 [==============================] - 193s - loss: 0.0593 - val_loss: 0.0631
```



Generalization on Track_2 (Drive Performance)

The second metric that was used to evaluate the results is generalization. This can be defined as how well the values are predicted by the models to drive over a different track it was not trained for. Here, the values are the predicted steering angle, brakes, and throttle. It is not something that can be plotted, but the evaluation can be done in terms of how far the car drives on the second track, without toppling down. The several factors affecting this can be the speed, turns, conditions on track like elevations, shadows, and so on. As discussed, the models are only trained on Track_1 data which was simpler but tested on Track_2. Thus, it was significantly challenging when dealing with Track_2. Few upcoming observations were noted while experimenting, that supports this claim.

• Though it performed well on Track_1 and gave the best accuracy during training (loss over epochs), most architectures were not able to even take the first turn over the Track_2.

• The reason could be overfitting for the Track_1 dataset. Overfitting refers to a situation in which the program tries to model the training data too well. In general, the model trains for details and even the noise for the data it has passed through. This, in turn, negatively impacts the generalizing ability.

• Other reasons could be that, Track_2 starts with a road running parallel to the one the car starts at, with a barrier in between.

Fig 32. Track_2 screenshot

CHAPTER:5 (CONCLUSION REFERENCE)

5.1 CONCLUSION

This project started with training the models and tweaking parameters to get the best performance on the tracks and then trying to generalize the same performance on different tracks. The models that performed best on 1 track did poorly on Track_2, hence there was a need to use image augmentation and processing to achieve real time generalization. The use of CNN for getting the spatial features and RNN for the temporal features in the image dataset makes this combination a great fit for building fast and lesser computation required neural networks. Substituting recurrent layers for pooling layers might reduce the loss of information and would be worth exploring in the future projects. It is interesting to find the use of combinations of real world dataset and simulator data to train these models. Then we can get the true nature of how a model can be trained in the simulator and generalized to the real world or vice versa. There are many experimental implementations carried out in the field of self-driving cars and this project contributes towards a significant part of it.

5.2 FUTURE WORK

In the implementation of the project the deep neural network layers were used in sequential models. Use of parallel network of network layers to learn track specific behavior on separate branches can be a significant improvement towards the performance of the project. One of the branches can have CNN layers, the other with the RNN layers and combining the output with a dense layer at the end. There are similar problems that are

solved using RESNET (Deep Residual networks) a modular learning framework. RESNET are deeper than their 'plain' counterparts (state-of-art deep neural networks) yet require similar number of parameters (weights). Implementing Reinforcement Learning approaches for determining steering angles, throttle and brake can also be a great way of tackling such problems.

Placing fake cars and obstacles on the tracks, would increase the level of challenges faced to solve this problem, however, it will take it much closer to the real-time environment that the self-driving cars would be facing in the real world. How well the model performs on real world data could be a good challenge. The model was tried with the real-world dataset, but there was no way of testing it on an environment like a simulator.

The big players in the self-driving car industries must be already trying this on their autonomous vehicles. This would be a great experiment to see, how this model really works

in the real time environment.

5.3REFERENCE

1. Oliver Cameron, "Challenge #2: Using Deep Learning to Predict Steering Angles", Published on 11 Aug 2016, https://medium.com/udacity/challenge-2-using-deeplearning-to-predict-steering-angles-f42004a36ff3, accessed Jul 2017

2. Ujjwalkaran, "An intuitive Explanation to Convolutional Neural networks", Published on 11 Aug 2016, 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/2015/05/21/rnneffectiveness, accessed Nov 2017

4. Mariusz Bojarski, "End-to-End Deep Learning for Self-Driving Cars", Published on 17 Aug, 2016, 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-termmemory-networks-in-python/, accessed Nov 2017

6. Lucas Weist, "Recurrent Neural Networks - Combination of RNN and CNN", Published on 7 Feb 2017, https://wiki.tum.de/display/lfdv/Recurrent+Neural+Networks+Combination+of+RNN+and+CNN, accessed Nov 2017

7. Dmytro Nasyrov , "Behavioral Cloning.NVidiaNeuralNetwork in Action." , Published on 21 Aug 2017, https://towardsdatascience.com/behavioral-cloning-project-3-6b7163d2e8f9 , accessed Jan 2017

8. Sihan Li , "Demystifying ResNet", Published on 20 May 2017, 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-imageclassification-models-using-very-little-data.html, accessed Jan 2017

10. Ivan Kazakov, "Vehicle Detection and Tracking", Published on 14 May 2017, https://towardsdatascience.com/vehicle-detection-and-tracking-44b851d70508, accessed Feb 2017