# End-to-End Deep Learning for Self-Driving Cars

*Technical Report*

*submitted in fulfilment of the requirement for the Degree*

*of*

**Bachelor of Technology**

**(Electronics and Communication Engineering)**

***By***


**Abhay Kumar Poddar 1904005**
**Raushan Kumar 1904059**
**Suraj Kumar 1904145**


*Under the guidance*

*of*

**Dr. Rajarshi Bhattacharya**

**(Assistant Professor)**



**Department of Electronics and Communication Engineering**
**NATIONAL INSTITUTE OF TECHNOLOGY PATNA**

**Patna, Bihar, India-800005**

# CERTIFICATE

It is certified that the major project work titled **"End to End Deep Learning for Self-Driving Car"** has been carried out by Abhay Kumar Poddar (1904005) , Raushan Kumar (1904059) and Suraj Kumar(1904145) who are bonafide students of National Institute of Technology Patna, in partial fulfilment for the award of the degree of Bachelor of Technology in Electronics and Communication Engineering of the National Institute of Technology Patna during the session 2019-2023. It is certified that all corrections/suggestions indicated during the internal assessment have been incorporated in the report deposited in the departmental library. The minor project report has been approved as it satisfies the academic requirements in respect of project work prescribed by the institution for the said degree. During the project period, they showed keen interest and sincerity towards the accomplishment of this project. We wish them success in all their future endeavours.

---

**Dr. Rajarshi Bhattacharya**

**Project Supervisor**

**Assistant Professor**

## ACKNOWLEDGEMENT

We express my sincere gratitude to the Department of Electronics & Communication Engineering for extending all kinds of help while carrying out this work.

We are deeply thankful to my guide, Dr. Rajarshi Bhattacharya, who spared his valuable time to review my work and gave me constant encouragement to continue my work.

Suraj Kumar - 1904145
Abhay Kumar Poddar - 1904005
Raushan Kumar - 1904059

Department of Electronics & Communication Engineering
NIT PATNA

# TABLE OF CONTENTS

# Chapter 1

## Objective of the project

The purpose of this project is to simulate a self-driving car.

Self-Driving cars are among the trending technologies and come with various advantages which include:

- Making transportation more accessible for those who cannot drive
- Decreasing traffic congestion
- Reducing crashes and saving lives
- Providing cheaper and comparatively hassle-free transportation

# Chapter 2

## Introduction

In the previous ten years, autonomous driving systems have made great strides forward (ADS). A variety of ADS products are likely to become commercially accessible during the next decade, therefore the field may be expected to continue to develop at its present rate. [3] It is expected that autonomous driving technology will cause a sea change in the transportation industry, with far-reaching consequences for how people travel, how different modes of transportation are used, and how businesses operate.

**Problem Definition:**

To obtain our objective of a Self-driving car, we want to control the steering angle of the vehicle automatically. For the purpose of attaining the steering angle of the vehicle, we want to test and train our model in a real-time environment, for that purpose we are using an open-source simulator released by Udacity for self-driving cars. The idea is to mimic human driving behaviour using convolutional neural networks and then test that behaviour in the simulator.. We call this strategy, "Behavioural Cloning," because it's meant to replicate human behaviour, including driving.

When we launch the simulator, it will have two separate courses and two distinct modes: training and autonomous. User training mode driving generates the dataset in the simulator. Sometimes referred to as "excellent" driving data, this set is highly desirable. After the deep learning model has been trained using user data, it is put to the test on the racetrack. Generalizing the performance across different tracks is another obstacle. To put it another way, the model is trained using data collected from one track and then put to the test in the simulator's testing environment.

**Technologies Used:**

Technologies that are utilized in the execution of this project, as well as the reasons for utilizing these technologies, are discussed in this part:

**Python:** As a high-level language, Python can be used for a wide variety of projects. The theory behind its layout places an emphasis on making code easy to read by heavily indenting it. Since it is dynamically typed and garbage-collected, Python is a very versatile language. It's compatible with a wide variety of programming styles, including structured, object-oriented, and functional.

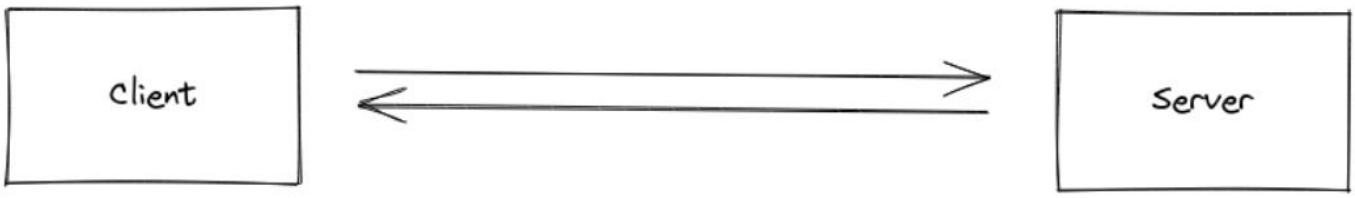
**TensorFlow:** Google's TensorFlow is a software library that anybody may access and use for free. It is used for machine learning and artificial intelligence. It can be applied to a wide variety of jobs, but the training and inference of deep neural networks is its primary area of concentration. We will use TensorFlow API called Keras to build our model, which will predict the steering angle of an autonomous vehicle.

**NumPy:** NumPy is a python package language that offers support for massive, multi-dimensional arrays and matrices, in addition to a large number of high-level mathematical functions that may be used to work on these numbers. NumPy was developed by the Python Software Foundation.

**Pandas:** Python's pandas package provides a set of tools for working with and analysing data. There are data structures and functions provided for working with numerical tables and time series in particular.

**OpenCV:** The real-time functionality of modern systems relies heavily on OpenCV, a huge open-source library for feature extraction, machine learning, and image processing. It can be used to analyse videos, and photos, or even recognize text written by a human hand.

**Socketio:** Low-latency, bidirectional, event-based communication between a client and server is made possible by the Socket.IO package.




**Flask:** Python flask is a tiny web framework. Since it does not rely on any external resources, we classify it as a microframework. Typical features for which there are already third-party open source libraries are missing, like a database abstraction layer, a form validation technique, and so on.

# Chapter 3

## Challenges and Issues

- Very Less Data

- High Computational Cost and Memory Consumption

  With regard to the DL structure, CNN needs significant computational costs and uses a lot of memory space. This is a problem for autonomous vehicle steering since it raises the cost of the vehicle by impacting both the hardware and the software. Researchers should look at various methods to lessen the computational cost and memory consumption of the DL structure used in AV steering control.

- Yet To Attain Optimum Steering Performance

  When it comes to steering control, DL has not yet reached its full potential. By enhancing the neural network design with batch normalisation and skip connections, we can improve the steering control performance. As an added bonus, it may be made capable of handling obstacle avoidance by including a time component into the steering angle prediction or by researching mediated sensory approaches employing implicit systems.

- Not able to detect coming cars

  For now, we are testing our model on one-way roads only, In two ways roads, it would be very difficult to control the steer angle and speed as we are working here only on images to calculate distances between two objects that need sensor antennas.

# Chapter 4

## Approaches to the problem

For solving this problem, we have used CNN model as we have to predict the steering angle after taking images as input from the Udacity simulator. CNN has worked very well because of its operation on images.

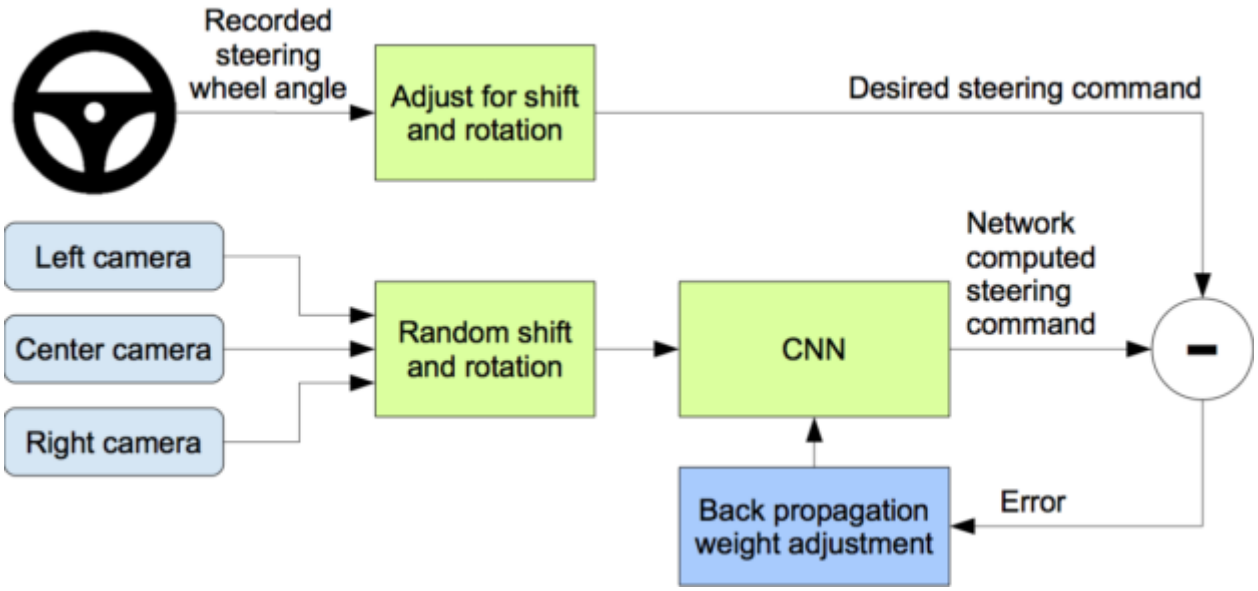



Fig 1: Complete process of project

The project will contain 4 steps for its execution:

- Data Collection
- Data Augmentation
- CNN Model
- Connect our model to Udacity Simulator to check How well It performs in real environments

## Data Collection

Udacity has created an open-source simulator for autonomous vehicles so that enthusiasts can practice in a setting that's as similar as possible to real life. It's based on the popular gaming engine Unity. The simulator's resolution and controls are both adjustable, and the interface is straightforward.

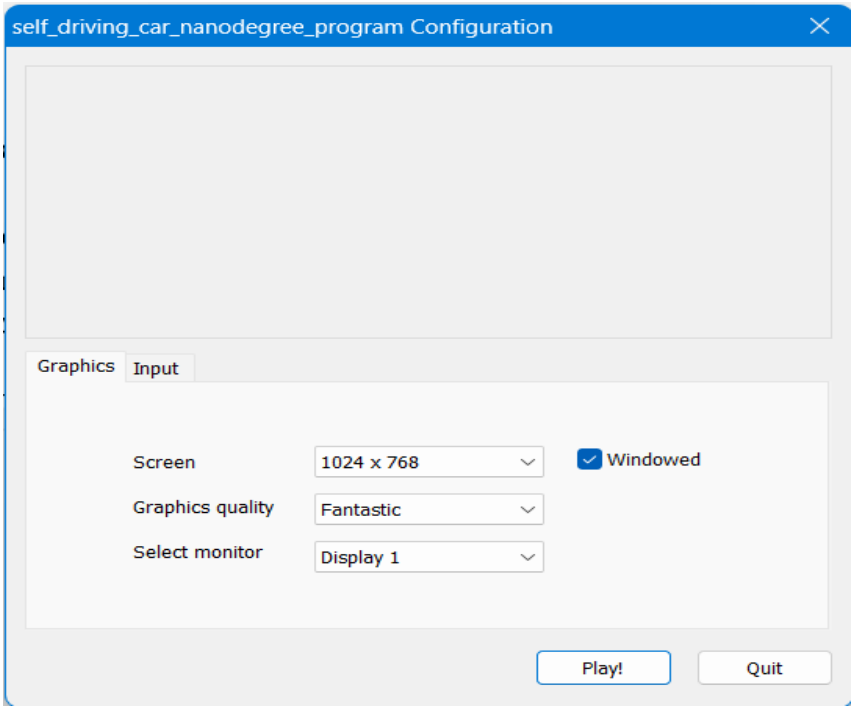



Fig 2: simulator configuration screen

It is possible to customize the visuals and input settings to suit individual tastes and hardware as shown in figure 2. To begin the simulation, the user simply clicks the "Play!" button.

Figure 3 shows the keyboard controls, which may be accessed by clicking the Controls tab.

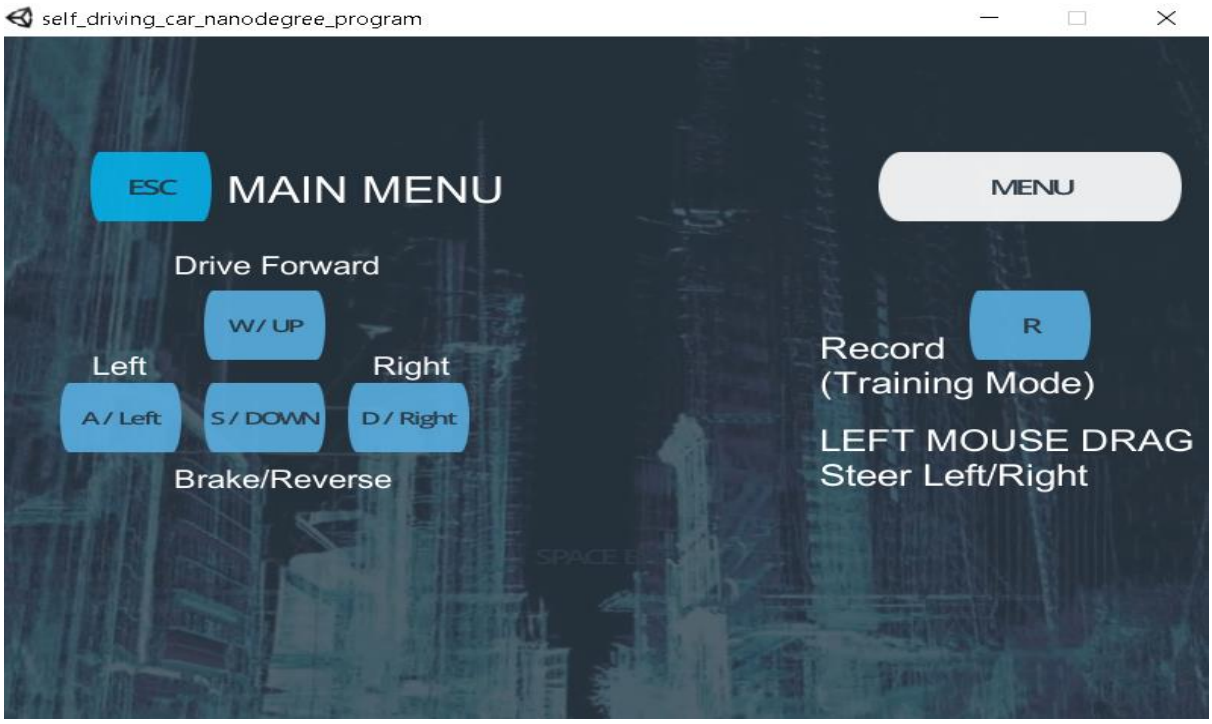



Fig 3: Vehicle control configuration in simulation

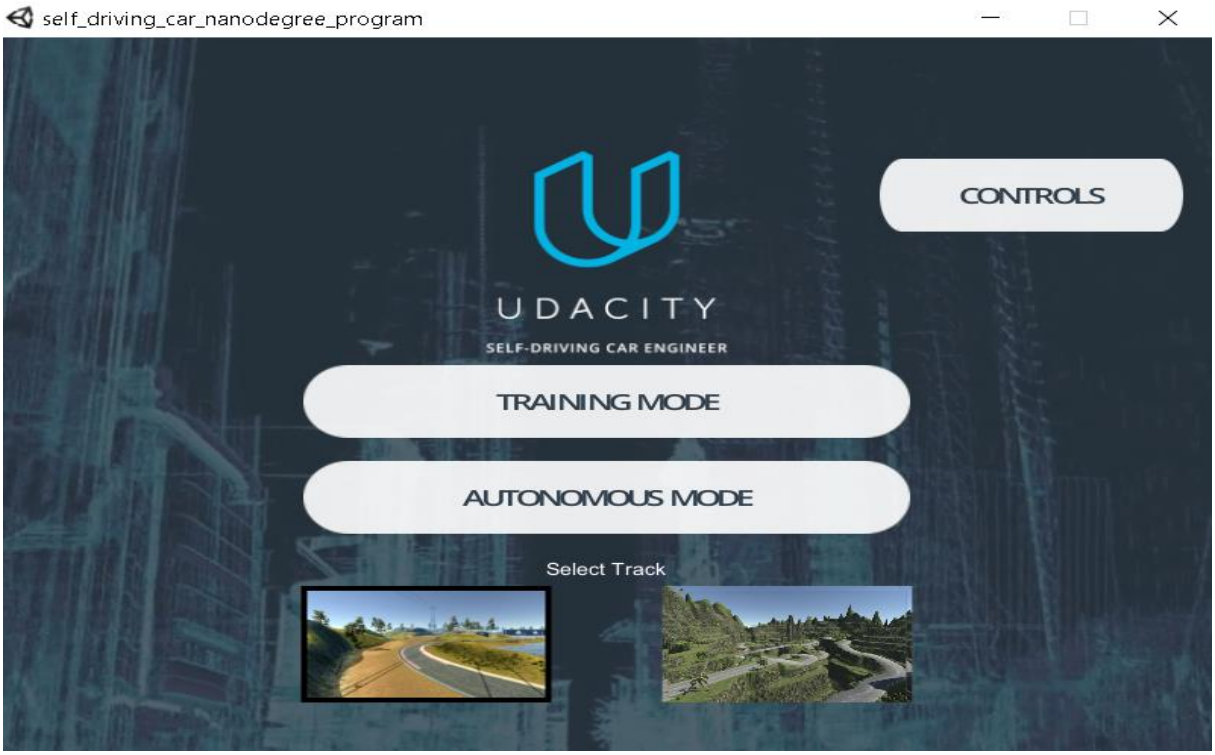



Fig 4: Simulator home page

Fig 4 shows the home page of the emulator where we can see that there are two types of tracks are provided. One tract is the very simplest one i.e. on this tract driving is easier and is represented in Fig 5 and the other one is a little complex and its representation is shown in Fig 6. The second track is difficult to drive due to sharp curves, gradients and shadowed environments even by humans manually.


Fig 5: Track 1 (easier to drive)


Fig 6: Track 2 (difficult to drive)

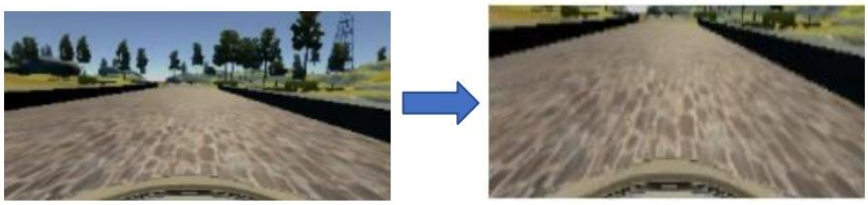
# Data Augmentation

The hardest part was getting the car to behave consistently on Track 2, which it had never been exposed to during training. Real-world applications of self-driving cars are limited by the impossibility of training a model on all feasible routes. It is also impossible to collect the dataset for all potential climates and roadways. Therefore, it's important to think of a way to generalize the behaviour over multiple tracks.

Techniques for picture pre-processing and augmentation, which address this issue, are detailed below.: -

1. **Crop**

   There are important details in the dataset, especially in the lowest third of the pictures where the road can be seen. Since the external environment above a cut-off point in the photograph will never be used to determine the final product, that area can safely be removed throughout the editing process. In the training set, just the upper 30% of the images are used. In Figure 7, you can see the code snippet and the resulting picture transformation after cropping and scaling.



```
def crop(image):
    img = image[65:160,0:320]
    return img
```

Fig 7: Crop image

### *2.* Flip (horizontal)

The image is horizontally inverted (the dataset is provided with a mirror copy of the original). The idea is to train the model to handle symmetrical twists from both directions. Due to the fact that all turns on Track 1 are left, this is crucial information.
Fig 8, displays the code fragment and the resultant altered image.

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

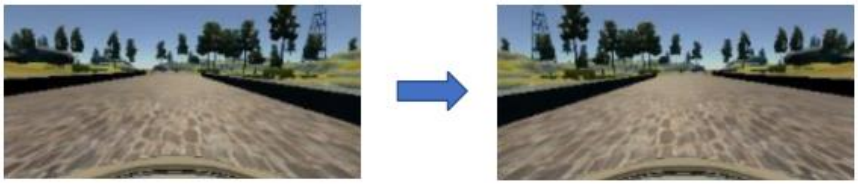


Fig 8: horizontally flipped image

### 3. Shift (*horizontal/vertical*)

A very slight translation has been applied to the image; in this case, the vertical shift can be seen in Fig 9, and the horizontal shift can be seen in Fig 10.

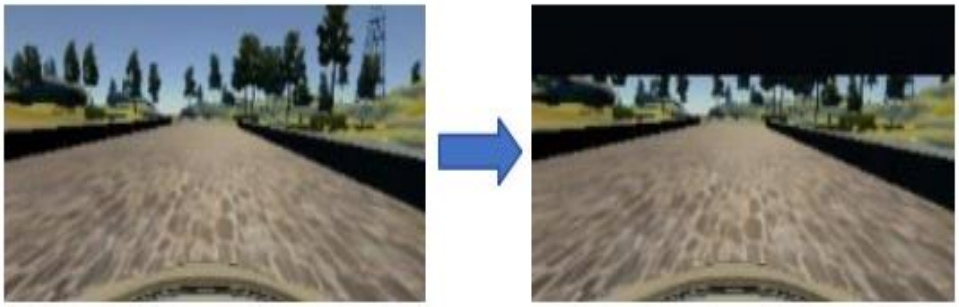


Fig 9: vertical shifting of image

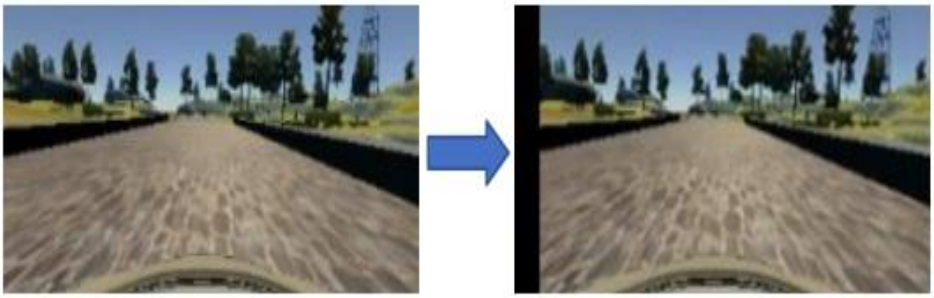


Fig 10: horizontal shifting of image

## 4. Brightness

In order to generalize to the weather circumstances, whether it be a brilliant sunny day or gloomy, lowlight situations, the enhancement of the brightness can prove to be of great benefit. Fig 11 demonstrates the code portion in question as well as the brightness increase. In a similar manner, I have arbitrarily decreased the amount of brightness for a variety of different environments.

```
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)
```

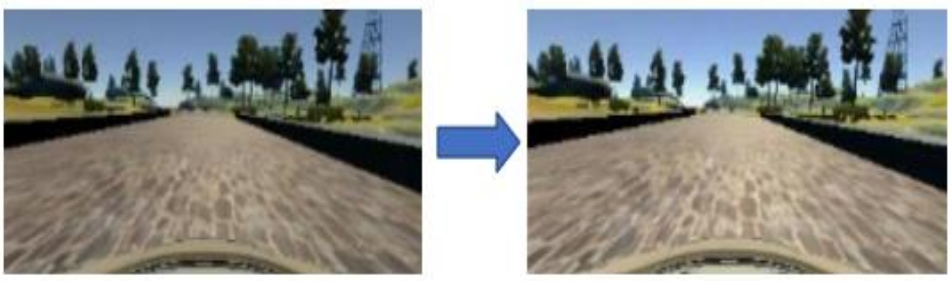


Fig 11: Code and Increased brightness

## 5. Shadowing

A potential of road shadows exists even when the present lighting conditions are considered. This will result in the image containing scenes that are both well-lit and poorly lit in equal measure. The dataset has had this augmentation done to it so that random shadows can be thrown, which will also alleviate the problem of shadow fitting. Figure 12 presents a sample shadow enhancement along with its associated code snippet for your perusal.

```
def shadow_random(self,x):
    x = cv2.cvtColor(x,cv2.COLOR_RGB_to_HSV)
    max_x = 255
    max_y = 55
    verticles = 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
```

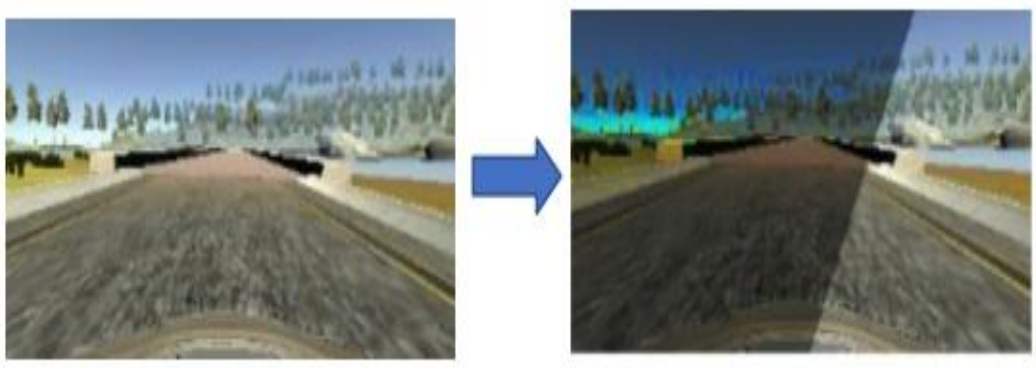


Fig 12: random shadow generated in view

## 6. Blur

This enhancement is used since a camera's acquired picture is not always crystal clear, and the distortion effect is a major contributor to this. Sometimes the camera isn't in focus, yet driving steadily is still essential. This random blur enhancement may account for such contingencies. Figure 13 displays the transformed example code for your perusal.

```
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
```

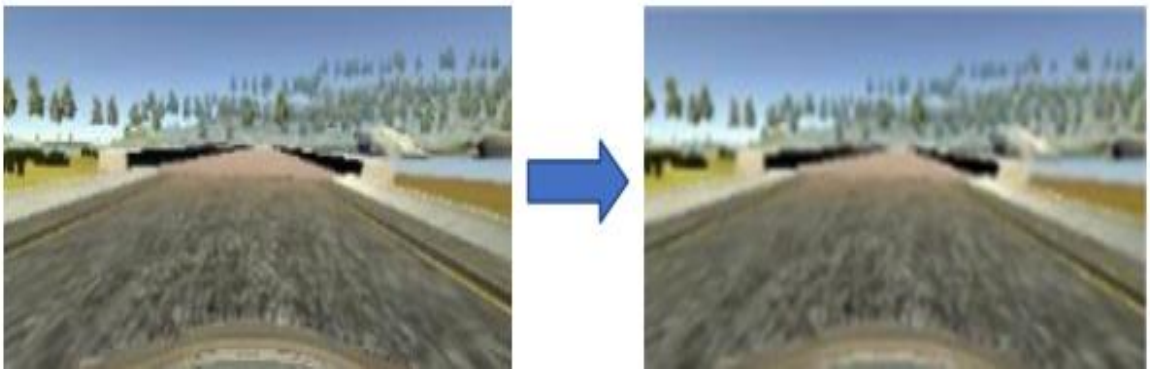


Fig 13: Random blurred view

# CNN Model

• A feed-forward neural network computing system known as CNN is a sort of system that may be used to learn from data that is fed into it. The process of learning involves establishing a collection of weights or filter values that, when applied to the network, enable it to model behaviour in accordance with the data used for training.

• CNN is effective in extracting hierarchical information and spatial details from pictures. It does this by making use of filters that examine particular areas of an input image inside a window of a certain size and then map those areas to some output. After that, it moves the window across the screen in specified predetermined increments, eventually covering the entire image. Each layer of the convolution filter thus captures the properties of the input image in a hierarchical fashion in a succession of successive layers, initially collecting the details such as lines in the image, then capturing shapes, and finally capturing full objects in later levels.

• CNN may be an effective tool for feeding the photos of a dataset into it so that the images can be classified into their appropriate categories.

CNN = Feature extraction + Classification

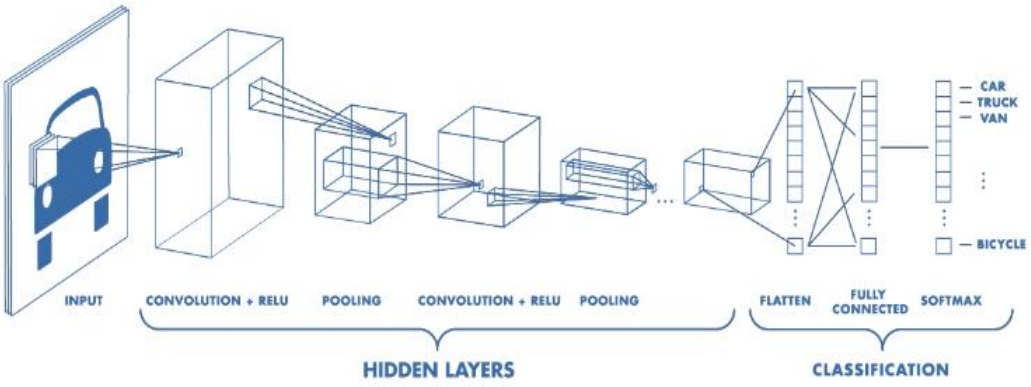


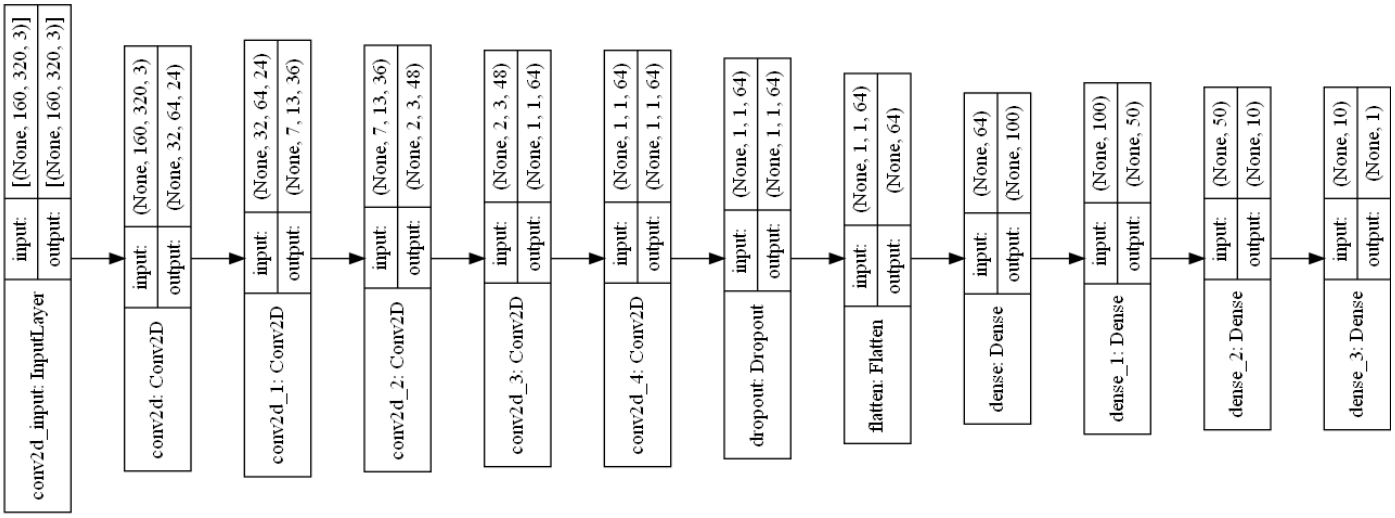

Fig 14: CNN Architecture

## CNN Model used in Project

## Code for Model 1:

```
def build_model():

    model = Sequential()
    model.add(Conv2D(24, 5, 5, activation='elu', padding = 'same',input_shape=INPUT_SHAPE))
    model.add(Conv2D(36, 5, 5, activation='elu', padding = 'same'))
    model.add(Conv2D(48, 5, 5, activation='elu', padding = 'same'))
    model.add(Conv2D(64, 3, 3, activation='elu', padding = 'same'))
    model.add(Conv2D(64, 3, 3, activation='elu', padding = 'same'))
    model.add(Dropout(0.5))
    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()

    return model

```

# Architecture of model 1:

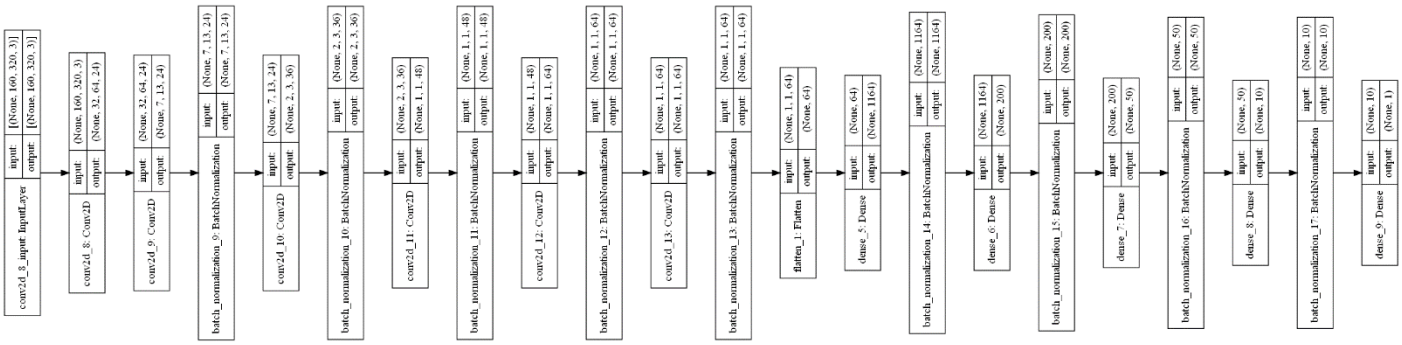



# Code for Model 2:

```
def build_model():

    model = Sequential()

    model.add(Conv2D(24, 5, 5, activation='elu', padding = 'same',input_shape=INPUT_SHAPE))

    # We have a series of 3 5x5 convolutional layers with a stride of 2x2
    model.add(Conv2D(24, 5, 5, activation='relu', padding = 'same'))
    model.add(BatchNormalization())

    model.add(Conv2D(36, 5, 5,  activation='relu', padding = 'same'))
    model.add(BatchNormalization())

    model.add(Conv2D(48, 5, 5,  activation='relu', padding = 'same'))
    model.add(BatchNormalization())


    # This is then followed by 2 3x3 convolutional layers with a 1x1 stride
    model.add(Conv2D(64, 3, 3, activation='relu', padding = 'same'))
    model.add(BatchNormalization())

    model.add(Conv2D(64, 3, 3, activation='relu', padding = 'same'))
    model.add(BatchNormalization())

    # Flattening the output of last convolutional layer before entering fully connected phase
    model.add(Flatten())

    # Fully connected layers
    model.add(Dense(1164, activation='relu'))
    model.add(BatchNormalization())

    model.add(Dense(200, activation='relu'))
    model.add(BatchNormalization())

    model.add(Dense(50, activation='relu'))
    model.add(BatchNormalization())

    model.add(Dense(10, activation='relu'))
    model.add(BatchNormalization())

    # Output layer
    model.add(Dense(1))

```

```
    model.summary()

    plot_model(model, show_shapes=True, to_file='model_v4.png')

    return model

```

## Architecture of model 2:




## Code for connecting python model and Udacity Simulator which will act as a client-server protocol:

```
import argparse
import numpy as np
import cv2
from keras.models import load_model     #load our saved model

import base64   #decoding camera images

import socketio     #real-time server
import eventlet     #concurrent networking
import eventlet.wsgi    #web server gateway interface
from flask import Flask     #web framework

from PIL import Image   #image manipulation
from io import BytesIO      #input output



#initialize our server
sio = socketio.Server()

#flask (web) app
app = Flask(__name__)

MAX_SPEED = 18
MIN_SPEED = 10

#and a speed limit
speed_limit = MAX_SPEED

def Preprocess(image):
    """
    image Processing
    """
    #print("we are in preprocess")
    image = cv2.cvtColor(image, cv2.COLOR_RGB2YUV) # Convert the image from RGB to YUV (This is what the NVIDIA model does)
    #image = cv2.resize(image, (320, 80))
```

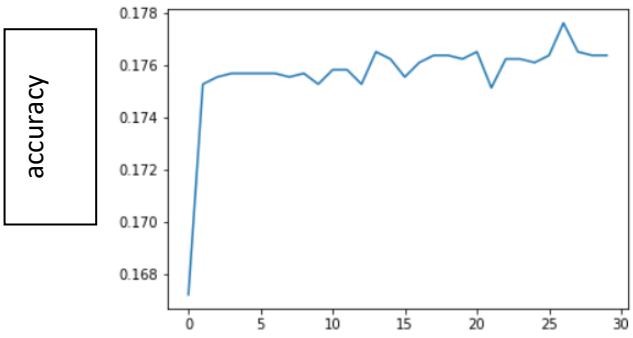
```
    image = image/255 # normalize pixels value between 0 & 1
    return image

@sio.event
def connect_error(data):
    print("The connection failed!")

#registering event handler for the server
@sio.on('telemetry')
def telemetry(sid, data):
    if data:
        # The current steering angle of the car
        steering_angle = float(data["steering_angle"])

        # The current throttle of the car, how hard to push peddle
        throttle = float(data["throttle"])

        # The current speed of the car
        speed = float(data["speed"])

        # The current image from the center camera of the car
        image = Image.open(BytesIO(base64.b64decode(data["image"])))

        try:

            image = np.asarray(image)       # from PIL image to numpy array
            image = Preprocess(image)       # apply the preprocessing
            image = np.array([image])       # the model expects 4D array
            # predict the steering angle for the image
            #print(image.shape)

            steering_angle = float(model.predict(image, batch_size=1))
            #steering_angle = 0.2

            # lower the throttle as the speed increases
            # if the speed is above the current speed limit, we are on a downhill.
            # make sure we slow down first and then go back to the original max speed.
            global speed_limit
            if speed > speed_limit:
                speed_limit = MIN_SPEED  # slow down
            else:
                speed_limit = MAX_SPEED

            throttle = 1.0 - steering_angle**2 - (speed/speed_limit)**2

            print('{} {} {}'.format(steering_angle, throttle, speed))
            send_control(steering_angle, throttle)

        except Exception as e:
            print(e)

    else:
        print("we are not in telemetry")
        sio.emit('manual', data={}, skip_sid=True)

@sio.on('connect')
def connect(sid, environ):
    print("connect ", sid)
    send_control(0, 0)

def send_control(steering_angle, throttle):
    sio.emit(
                "steer",
                data={
                        'steering_angle': steering_angle.__str__(),
                        'throttle': throttle.__str__()
                    },
                skip_sid=True
            )

if __name__ == '__main__':

    parser = argparse.ArgumentParser(description='car Driving')
    parser.add_argument(
```

```
109.         'model',
110.         type=str,
112.     )
113.
114.     args = parser.parse_args()
115.
116.     #load model
117.     model = load_model(args.model)
118.
119.     print("Server has started for autonomous car driving")
120.
121.     # wrap Flask application with engineio's middleware
122.     app = socketio.Middleware(sio, app)
123.
124.     # deploy as an eventlet WSGI server
125.     eventlet.wsgi.server(eventlet.listen(('', 4567)), app)
126.
```

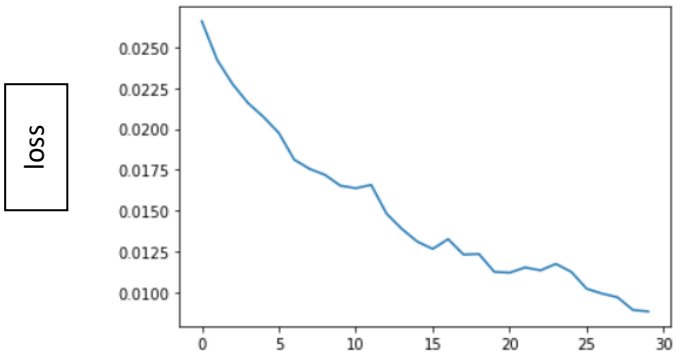
# Chapter 5

## Results:

Accuracy and Loss of model 1:

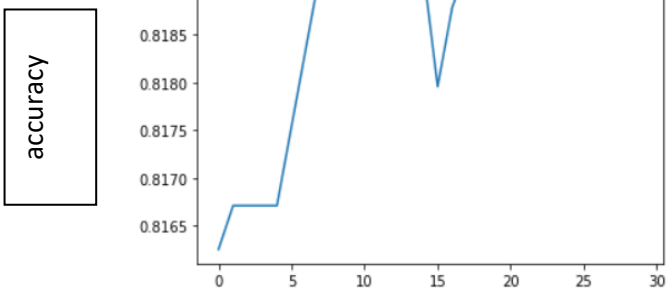



No.of epochs

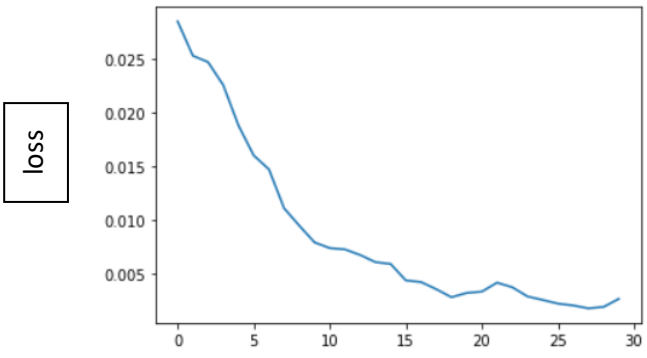



no. of epochs

Accuracy and Loss of model 2:




No of epochs




no of epochs

# Chapter 6

## Conclusion

- Before attempting to generalize the model's performance to new tracks, we trained it to get the best possible results on the tracks and optimized its parameters.
- We trained the CNN model using Adam as the optimizer. After much, tweaking of parameters, and experimentation of multiple models, we ended up with one model that is able to drive the car autonomously.
- We had showed that CNNs can learn the whole job of lane and road following without being manually decomposed into road or lane marker recognition.

# References


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

[2] Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L.D., Monfort, M., Muller, U., Zhang, J. and Zhang, X., 2016. End-to-end learning for self-driving cars. arXiv preprint arXiv:1604.07316.

[3] 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

[4] Net-Scale Technologies, Inc. Autonomous off-road vehicle control using end-to-end learning, July 2004. Final technical report. URL: http://net-scale.com/doc/net-scale-dave-report.pdf.

[5] L. D. Jackel, D. Sharman, Stenard C. E., Strom B. I., , and D Zuckert. Optical character recognition for self-service banking. AT&T Technical Journal, 74(1):16–24, 1995.