Autonomous Car Using Deep Learning, Convolutional Neural Networks and Computer Vision

Submitted in partial fulfilment of the requirements for the degree of

Bachelor of Technology

in

Electronics and Communication Engineering

by

Meera L 18BEC0597

Srinidhi R 18BEC0402

Under the guidance of

Dr Vaegae Naveen Kumar

SENSE

VIT, Vellore.

Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

May, 2022

DECLARATION

I hereby declare that the thesis entitled "**Autonomous Car Using Deep Learning, Convolutional Neural Networks and Computer Vision**" submitted by Meera L and Srinidhi R, for the award of the degree of Bachelor of Technology in Electronics and Communication Engineering to VIT is a record of bonafide work carried out by me under the supervision of Dr Vaegae Naveen Kumar.

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Vellore

Date: 04.05.2022

Meera L

Srinidhi R

(Signature of the Candidate)

CERTIFICATE

This is to certify that the thesis entitled "**Autonomous Car Using Deep Learning, Convolutional Neural Networks and Computer Vision**" submitted by Meera L (18BEC0597), Srinidhi R (18BEC0402), SENSE, VIT, for the award of the degree of Bachelor of Technology in Electronics and Communication Engineering, is a record of bonafide work carried out by him / her under my supervision during the period, 03.01.2022 to 30.04.2022, as per the VIT code of academic and research ethics.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The thesis fulfils the requirements and regulations of the University and in my opinion meets the necessary standards for submission.

Place: Vellore

Date: 04.05.2022

Dr Vaegae Naveen Kumar

(Signature of the Guide)

Dr P Prakasam

(Head of the Department)

Bachelor of Technology in Electronics and Communication Engineering

ACKNOWLEDGEMENTS

First, we would like to thank Dr Vaegae Naveen Kumar for his valuable suggestions and guidance. We would also like to express our gratitude to VIT for giving us the opportunity to proceed with our project. Finally, we would like to acknowledge various blogs and research papers we referred for completing our project.

EXECUTIVE SUMMARY

Self-driving cars, an emerging technology, are fuelled by Deep Learning algorithms. They drive our society forward and create new opportunities in the mobility sector. This technology, when done right, can prove to be monumental in reducing the number of road accidents caused due to human error, thus saving thousands of lives every year. Using Convolutional Neural Networks results in achieving significant performance in various perceptions. It is promising when compared to other techniques in the latest years. The key factors behind these impressive results are their ability to learn millions of parameters using a large amount of labelled data. The proposed concept in this paper is to create an autonomous car, using CNN algorithms and Computer Vision, that can efficiently detect lanes, recognize and classify road signs, and predict the right steering angle and speed to manoeuvre the vehicle appropriately.

CONTENTS Page No.

List of Figures

List of Tables

List of Abbreviations

CNN	Convolutional Neural Network
RELU	Rectified Linear Activation Unit
ELU	Exponential Linear Unit
PSP-Net	Pyramid Scene Parsing Network
R-CNN	Region-based Convolutional Neural Network
ANN-MLP	Artificial Multilayer Perceptron
LSTM-CNN	Long Short-Term Memory CNN
SVM	Support Vector Machine

1. INTRODUCTION

1.1 Objective

Our main focus in this project is to simulate a Self-Driving Car for Indian atmosphere. Our project includes lane detection, traffic signal identification, and using Behavioural cloning model to train roads with potholes and other defects. The objective of this project is to achieve improved accuracy.

1.2 Motivation

Road travels are inevitable in a dynamic country like India. Humans are prone to delinquency and error. Increasing population indefinitely escalates the probability of error. According to NCRB, India has recorded 1.20 Lakh death cases due to negligence related to road accidents in 2020 alone! [1]. This whooping number is mainly due to distracted driving, drunk driving, speeding, and breaking traffic rules. This project is an attempt to reduce the accidents that occur in India due to careless driving and save thousands of lives.

1.3 Background

There have been various attempts to design a reliable self-driving car model. One of the prominent ideas is lane detection. A 2021 paper suggests using deep learning to detect lanes, identify traffic signals and dodge obstacles. Pyramid Scene Parsing Network (PSP-Net) is used to segment objects, dissect images and allocate a pixel category. [10]. There has been a lot of emphasis on traffic light detection. [4, 6, 8]. An autonomous downscaled model with a simple RC car as a base was used to detect lanes, identify traffic signals and avoid obstacles by using neural networks and image processing on Raspberry Pi. [4]. An autonomous car that can recognize the road, signals, stop signs, and obstacles and responds by making appropriate decisions like changing the vehicle's course, stopping on red signals, and moving on green signals, using Neural Networks was designed. [6]. A deep neural network-based model incorporating the "Faster R-CNN (region-based convolutional network) V2" model gives accurate and desired results. This deep learning method eradicates ample computational and time resources by using a pre-trained model. [8]. Practical Implementation helps in validating the results obtained by the simulation. Many models were presented with hardware implementation for a better understanding. [2, 6, 9]. Nvidia Jetson Nano, a developer kit, was used to model the autonomous car. The images collected by the camera installed on the vehicle were classified, using the ResNet network, by the action the car must perform to avoid crashing. [2]. A model consisting of three units was designed. First an input unit that contains a pi camera and an ultrasonic sensor. Second is the processing unit which consists of the laptop that acts as the server where the neural network will be running. And finally, the third unit is the RC control unit, the Arduino. This car was trained in various straight and curved track combinations, which were satisfactory.[6]. The self-driving car was implemented using a microcontroller, raspberry pi camera and open CV. The vehicle was trained in different tracks, recording thousands of images to train the model. A CNN with 'Relu' and 'Softmax' activation was used, making the system reliable and risk-free. [9]. A prototype based on monocular vision was proposed using a Deep Neural Network on Raspberry Pi. The primary focus was directly mapping input images to a predicted steering angle as the output. However, the camera latency can be significantly long based on the camera and the performance of Raspberry Pi, which is a lot higher than human perception, affecting the control performance negatively. [3]. A lidar sensor-based work presents a reinforcement-learning based approach with Deep Q Network implemented in autonomous driving that can be applied to large-scale cars. This expensive method helps in understanding the surroundings better. [5]. A study comparing three different machine learning algorithms: Artificial Multilayer Perceptron (ANN-MLP), Convolutional Neural Network - Long Short-Term Memory (CNN-LSTM), and Support Vector Machine (SVM) was used to determine the winner. Higher accuracy was obtained for the CNN-LSTM algorithm based on speed and obstacles. [7].

In this work, we aim to create a self-driving car model suitable for the Indian atmosphere. Numerous roads in India, especially in the countryside, don't have a well-defined lane. Even if the lanes are defined, maintenance is necessary to ensure their visibility. This paper proposes an extension of the Canny edge detection algorithm to manoeuvre the car with no lanes. Following the traffic signals are essential for a safe drive. We use CNN to classify various traffic signals. Behavioural cloning is used to portray the results. This is done with the aid of Udacity's self-driving car simulator.

2. PROJECT DESCRIPTION AND GOALS

Self-driving cars have to interpret the traffic signals and make a decision for a smooth drive. Convolutional neural networks and Keras are used to build a model to classify the signals. An image dataset with 43 different signals from BitBucket[15] is utilized for this project.

Since our model is aimed primarily toward Indian roads, detecting lanes becomes an essential step as the majority of the streets are either not well laid out or do not have defined lanes. The input data is from the camera that is to be installed in the vehicle.

Behavioural cloning captures and reproduces human sub cognitive skills in a computer program. We use an open-source simulator by Udacity to train and test our model. The car is driven through the training track multiple times to collect sufficient images to prepare the vehicle to operate autonomously. Achieving satisfying accuracy in deep neural networks requires significant training data. Image augmentation artificially creates multiple images through different techniques like rotation, shift, shear, zoom, flip, etc.

3. TECHNICAL SPECIFICATIONS

Open CV

TensorFlow

Keras

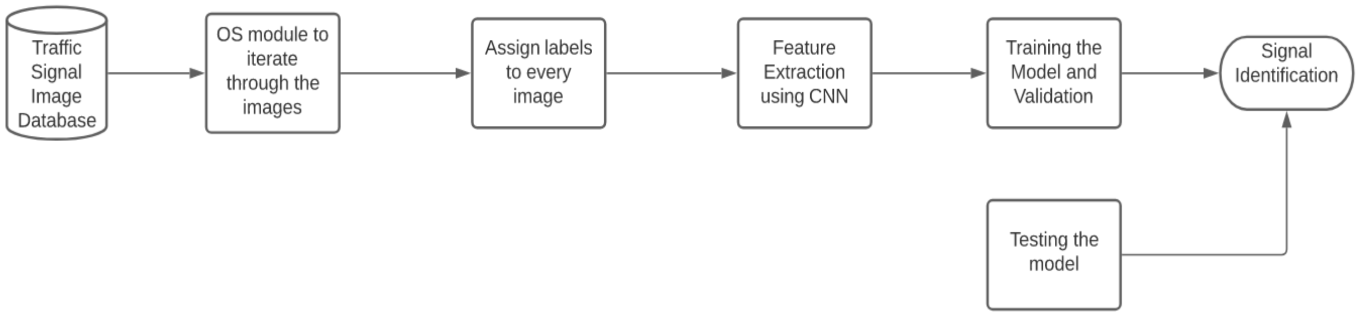
Atom

Google Colab

Udacity Self Driving Car Simulator

4. DESIGN APPROACH AND DETAILS

4.1 Design Approach



Traffic Signal Image Database
OS module to iterate through the images
Assign labels to every image
Feature Extraction using CNN
Training the Model and Validation
Signal Identification
Testing the model

Figure 1: Block Diagram of Traffic Signal Identification

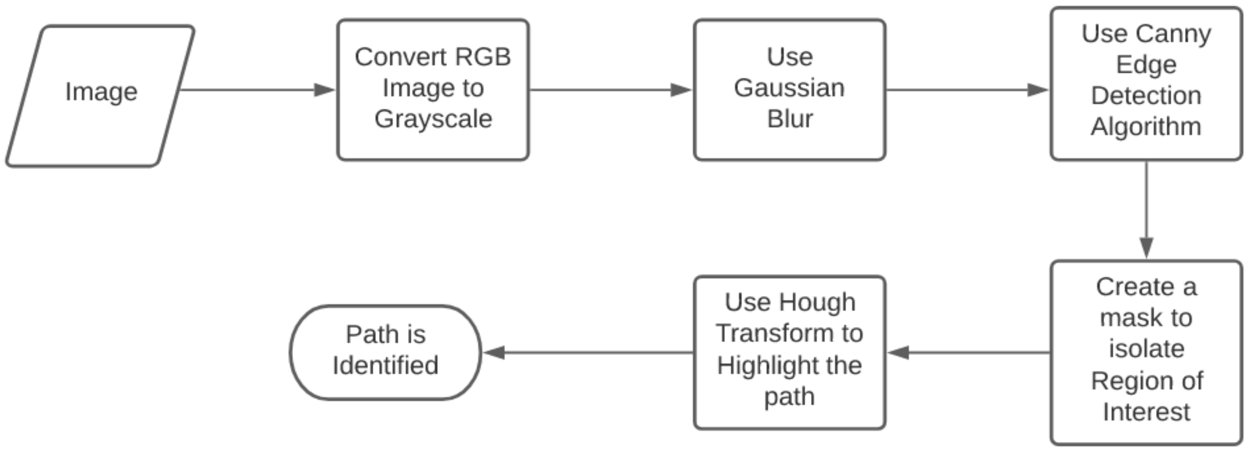


Image
Convert RGB Image to Grayscale
Use Gaussian Blur
Use Canny Edge Detection Algorithm
Create a mask to isolate Region of Interest
Use Hough Transform to Highlight the path
Path is Identified

Figure 2: Block Diagram of Lane Detection

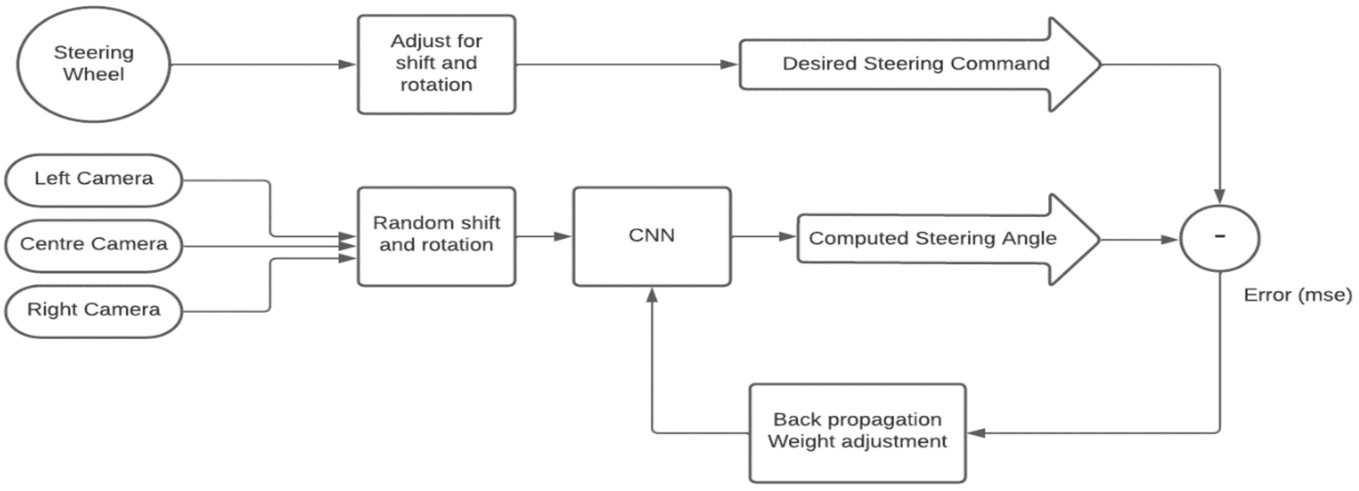


Figure 3: Block Diagram of Behavioral Cloning

4.2 Constraints, Alternatives and Trade offs

One constraint that we observed that our model, like any other model that employs the use of cameras to provide input to a real-time processing application, suffers from camera latency. Camera latency can be defined as the period from the time the camera sensor observes the scene to the time the computer actually reads the digitized image data. Compared to human perception, this is an extremely large figure and thus making it not suitable for safety-critical applications where time is the key factor.

In the lane detection part, some alternate edge detection algorithms that could be applied include Sobel, Roberts, Prewitt, Laplacian or Zero Crossing. But the Canny edge detection algorithm is known to be the most optimal algorithm and most commonly used in practical applications.

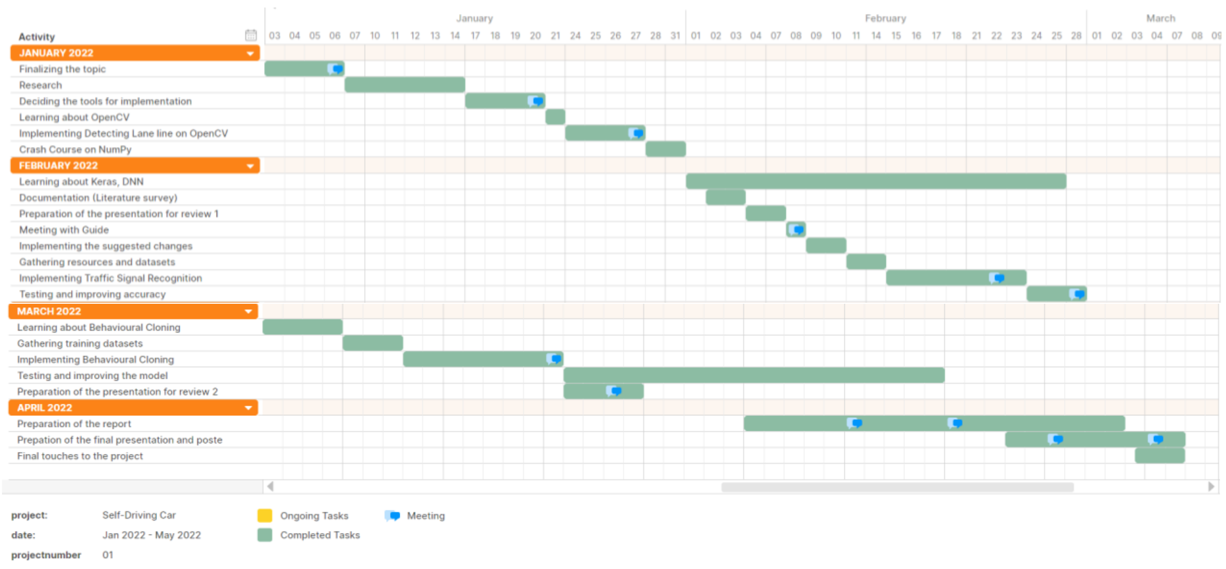
Other CNN architectures like ResNet or GooLeNet could be used as an alternative base model to the LeNet architecture that we used in our project. AlexNet is a popular CNN model that has proved to perform better than its counterparts in practise in terms of accuracy.

5. SCHEDULE, TASKS AND MILESTONES

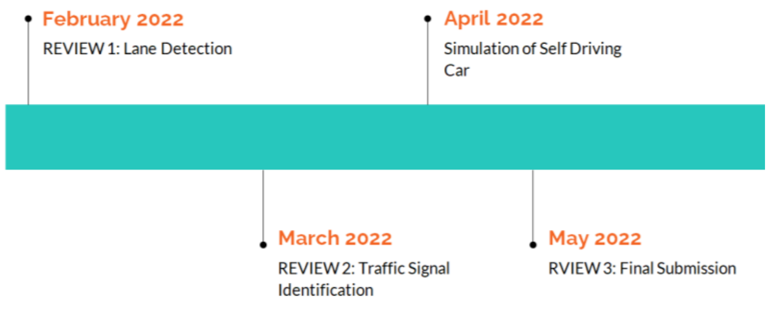
The following schedule was followed

- January – Discussion on selecting the project
- February – 1st review (Literature Survey, basic setup)
- March – 2nd review (Implementation of the classification model)
- April – Draft of paper and Poster
- May – Final touches and completion of project. Submission of report and poster

Tasks: The image below shows a detailed breakdown of our project timeline.



Milestones: The image below describes our major milestones of our project.


February 2022
REVIEW 1: Lane Detection
April 2022
Simulation of Self Driving Car
March 2022
REVIEW 2: Traffic Signal Identification
May 2022
RVIEW 3: Final Submission

6. PROJECT DEMONSTRATION

I. Traffic Signal Identification

The OS module iterates through the images and their classes and labels. Every image with its respective tags is stored as a list. The images captured by the installed cameras are given as the input, before which they have to be pre-processed. First, we are converting the RGB image as shown in Figure 4 into a greyscale image depicted in Figure 5 to reduce the number of channels and make the computation easier. Then we apply histogram equalization as illustrated in Figure 6 to standardize the lighting since the images that we encounter are not all going to be uniform; some might be too bright and some too dark, so it is essential to equalize them. Histogram equalization takes our histogram and spreads it at the ends to get a histogram that covers a higher range of brightness values, normalizes the lighting on all of our images, and results in uniform distribution of intensities. It also results in higher contrast which helps in feature extraction. And finally, we normalize the image by dividing every pixel value by 255. We then reshape the X arrays, and one hot encode the data labels before giving these images as inputs to the CNN model represented in Figure 7. We are using a modified LeNet model for identifying and classifying road signals.

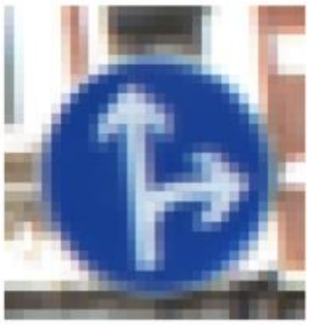


Figure 4: Sample Image

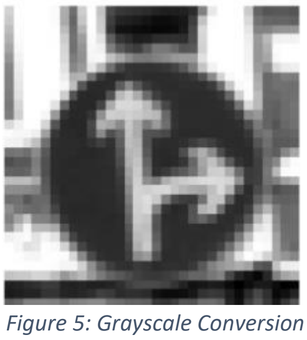


Figure 5: Grayscale Conversion

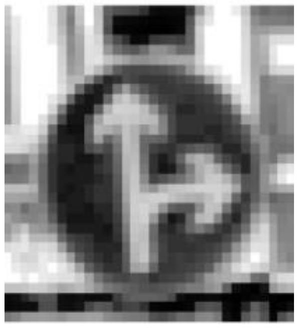
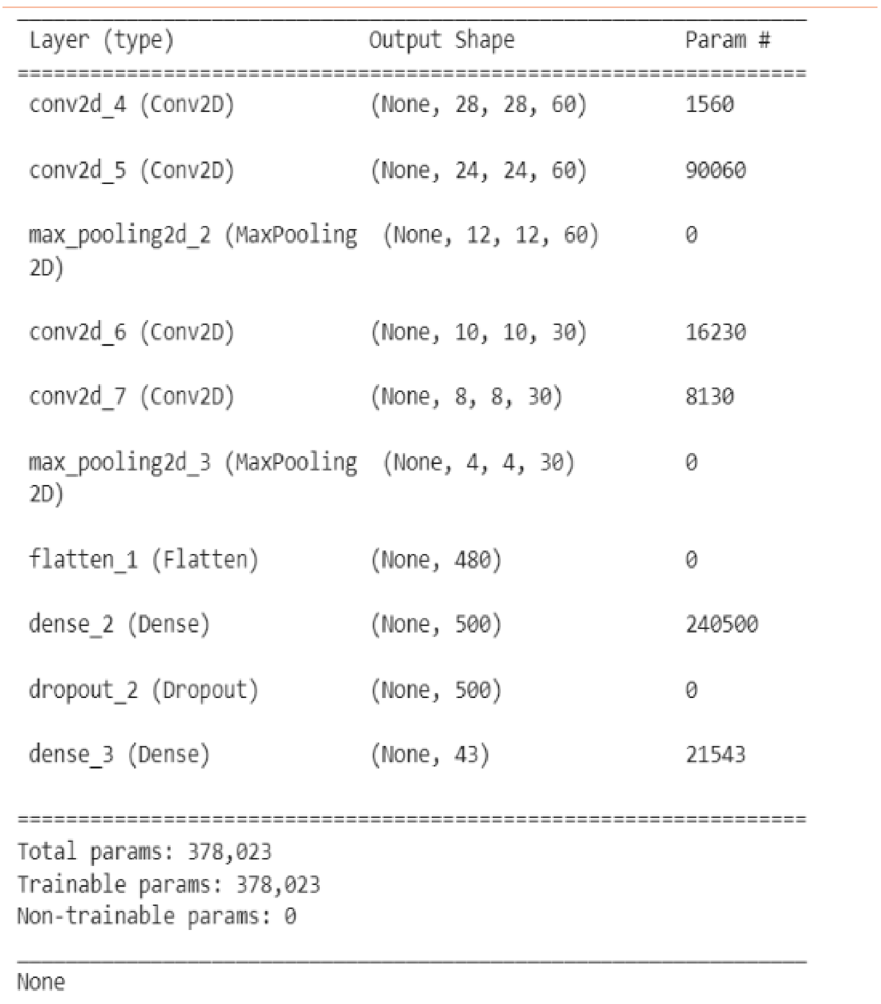


Figure 6: Histogram Equalization



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 28, 28, 60)        1560

 conv2d_5 (Conv2D)           (None, 24, 24, 60)        90060

 max_pooling2d_2 (MaxPooling  (None, 12, 12, 60)       0
 2D)

 conv2d_6 (Conv2D)           (None, 10, 10, 30)        16230

 conv2d_7 (Conv2D)           (None, 8, 8, 30)          8130

 max_pooling2d_3 (MaxPooling  (None, 4, 4, 30)         0
 2D)

 flatten_1 (Flatten)         (None, 480)               0

 dense_2 (Dense)             (None, 500)               240500

 dropout_2 (Dropout)         (None, 500)               0

 dense_3 (Dense)             (None, 43)                21543

=================================================================
Total params: 378,023
Trainable params: 378,023
Non-trainable params: 0
_________________________________________________________________
None
```

Figure 7: CNN Model

II. Lane Detection

The input data is from the camera that is to be installed in the vehicle. This image is first converted from an RGB scale to a greyscale image to reduce the number of channels from 3 to 1 as show in Figure 8, thus reducing the computational time and power required to process the image. Then we apply Gaussian blur to the image to remove noise which can otherwise create false edges and thus result in errors while detecting a lane. Figure 9 represents the image with Gaussian Blur. We then use the Canny edge detection algorithm, which uses the variation in intensity and traces out the edges by filling the pixels in white as show in Figure 10. We then create a region of interest to manoeuver the car on the desired path. Figure 11 represents our mask created around the desired region. Figure 12 shows our region of interest. Finally, we use Hough's transform as illustrated in Figure 13 to trace out the course for the vehicle and overlay the lanes onto the original image. We extend the idea of lane detection in an image to a video. We identify lanes in each frame of the video. We create a video capture object to capture the picture, which runs to the entire video length. Each of these frames undergoes the same process of a single image, thus identifying the desired lane as the vehicle moves forward.

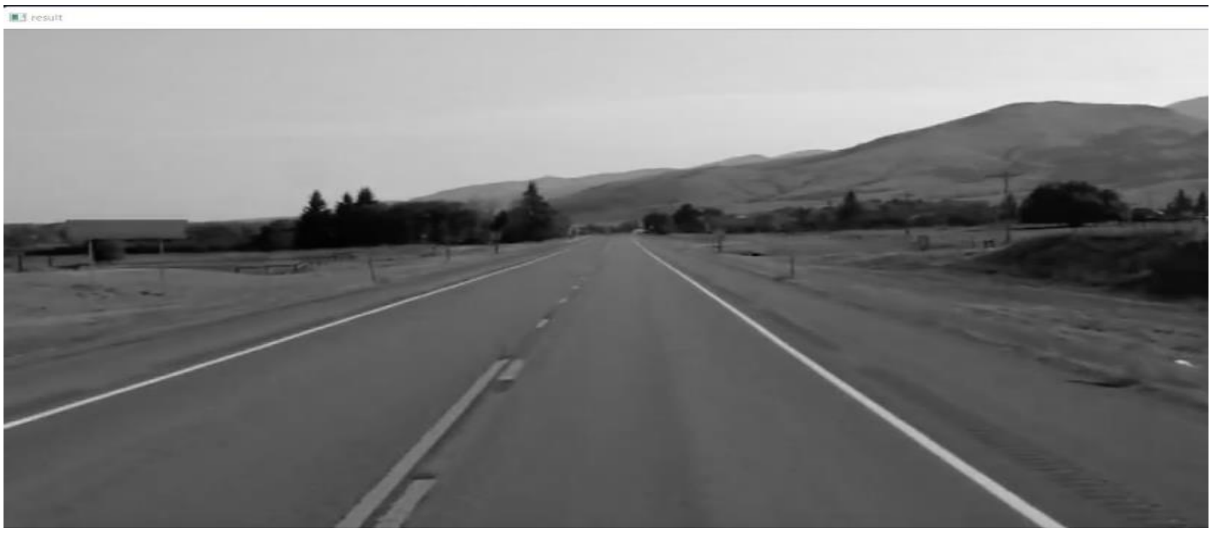


Figure 8: Grayscale Conversion

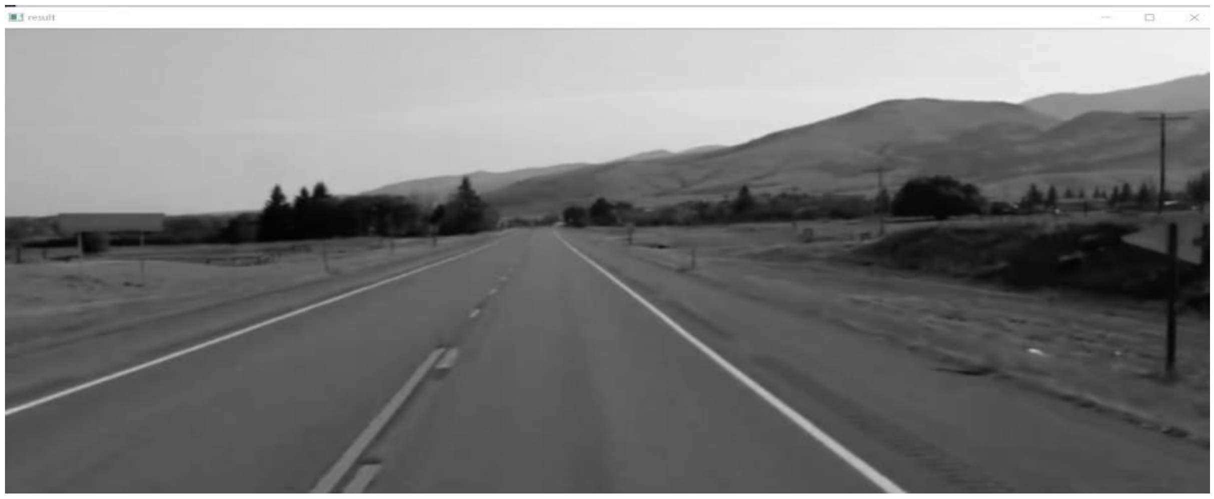


Figure 9: Gaussian Blur

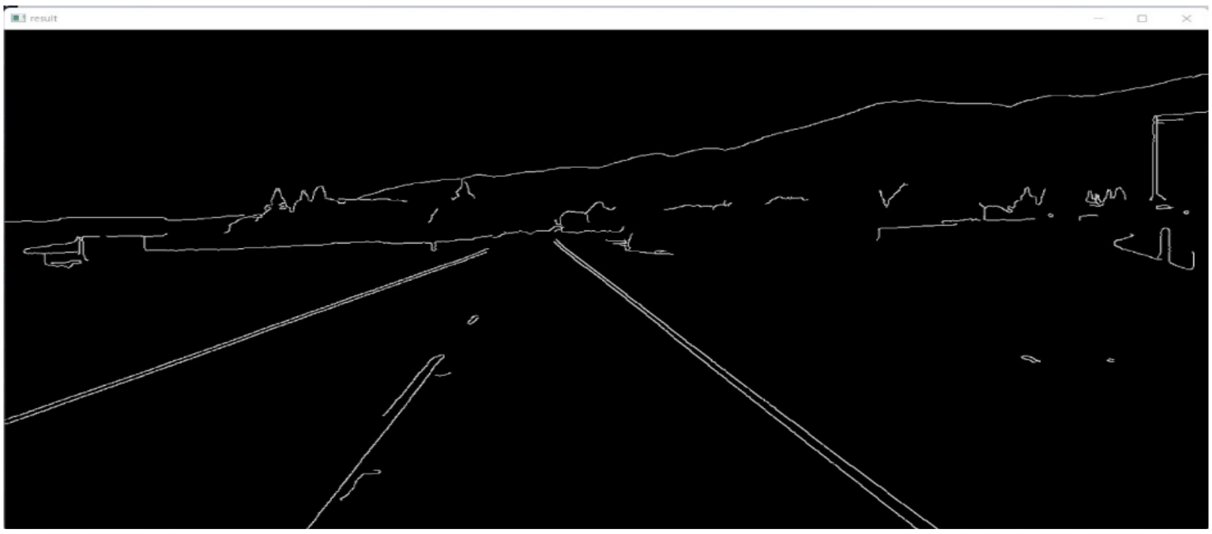


Figure 10: Canny's Edge Algorithm

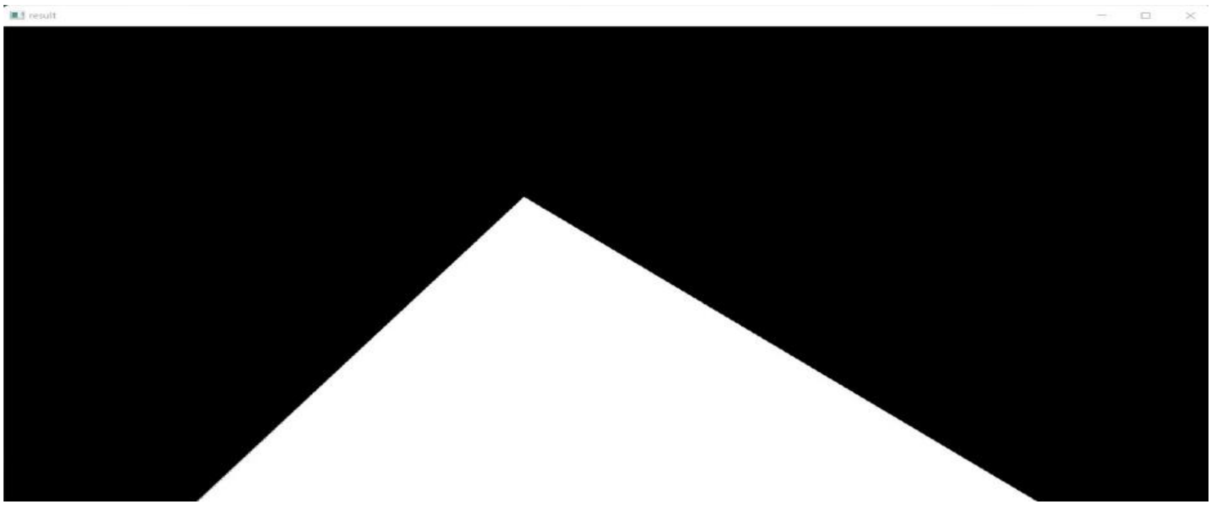


Figure 11: Mask

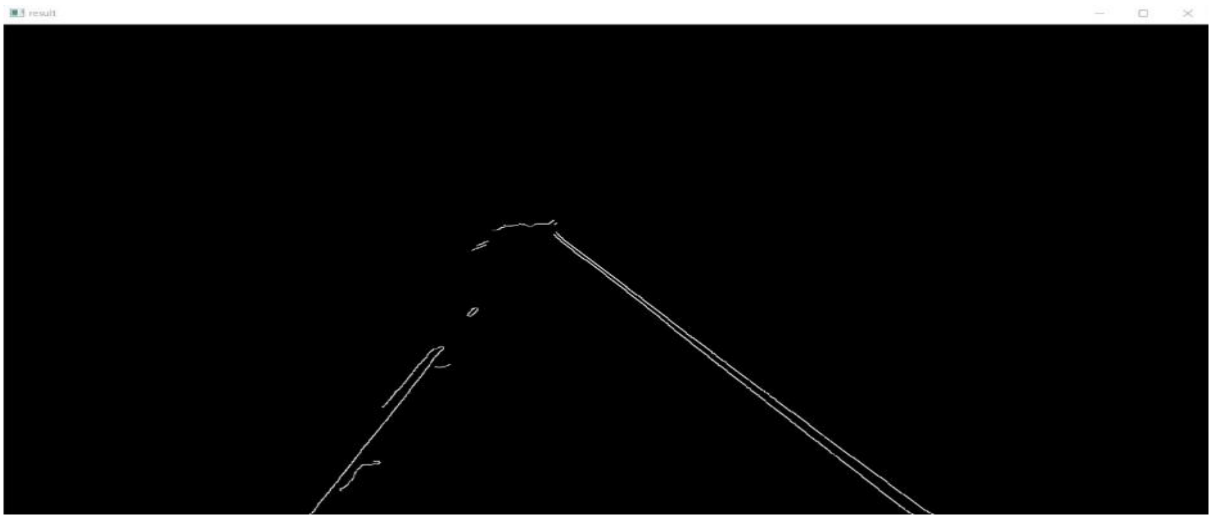


Figure 12: Region of Interest

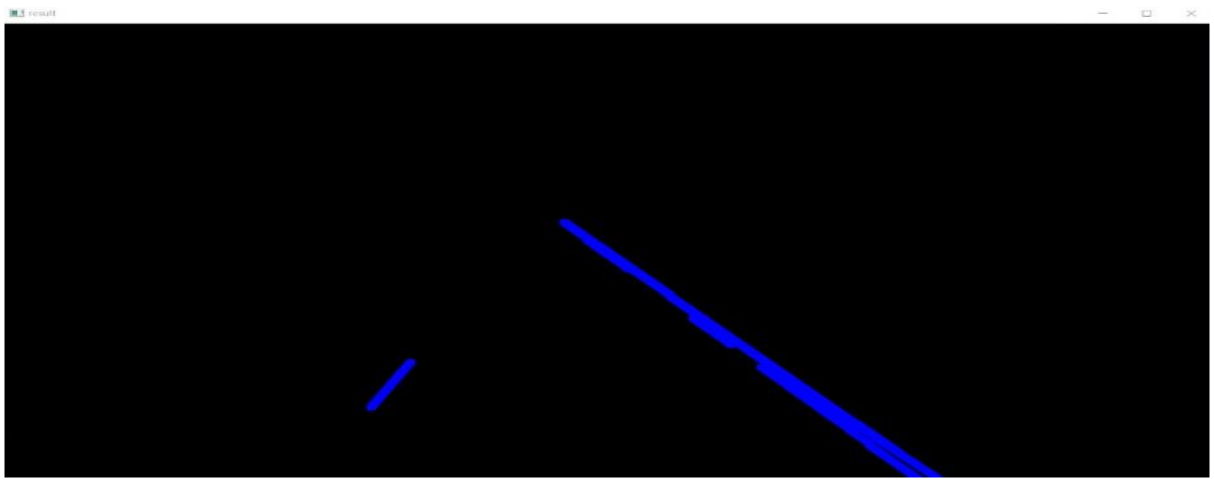


Figure 13: Hough's Transform

III. Behavioural Cloning:

We introduce four image augmentation techniques. Better feature extraction can be achieved by introducing up to 30% zoom as show in Figure 14. 10% image panning is initiated in x and y directions as represented in Figure 15. This technique is similar to swiveling a still camera from a fixed position. Images with varying brightness introduce variety in the training dataset. Darker images result in a higher variance. Figure 16 illustrates altering brightness. The final image augmentation technique used is horizontal flip. About 50% of the initial dataset undergoes image augmentation, significantly increasing the variety. Figure 17 depicts the horizontal flip. Figure 18 shows the original image and the pre-processed image side-by-side. To familiarize the vehicle, we use the popular Nvidia model. Normalized images pass through five convolutional layers with multiple filters. The convolutional layers aid in feature extraction. A flattened layer is introduced, followed by three fully connected or dense layers. Finally, a single dense layer is added as the output. Figure 19 represents the Nvidia model we have used in this project. Flask, a micro web framework and Socket.IO, a JavaScript library, connect the model with the simulator. Socket.IO enables real-time bidirectional communication with the simulator.

```
Text(0.5, 1.0, 'Zoomed Image')
```

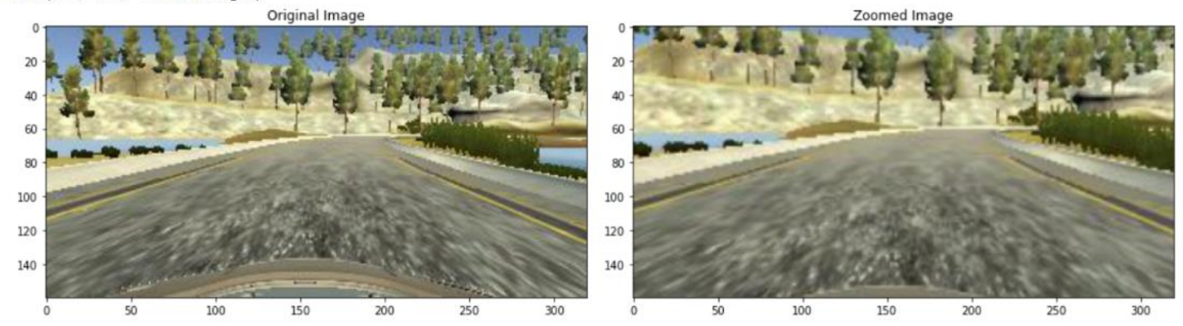


Figure 14: Image Augmentation – Zoom

```
Text(0.5, 1.0, 'Panned Image')
```

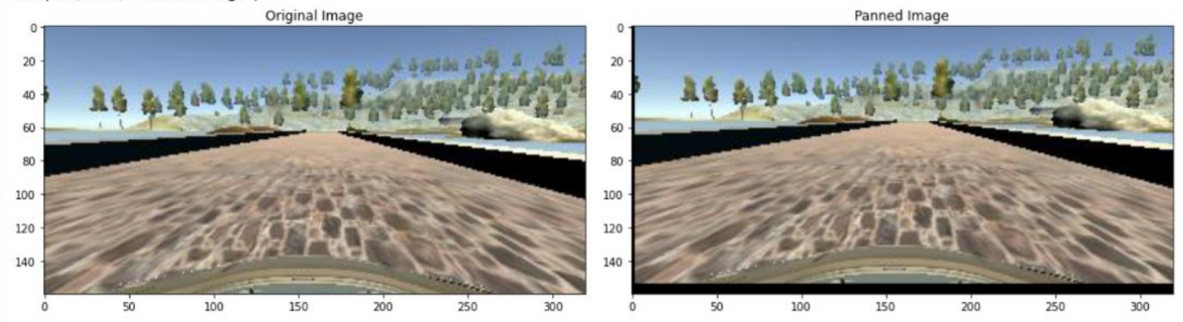


Figure 15: Image Augmentation –Panning

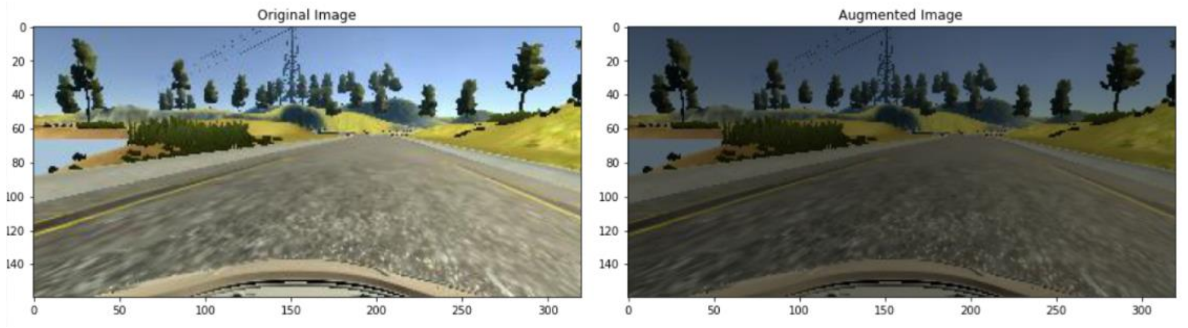


Figure 16: Image Augmentation – Altering Brightness

```
Text(0.5, 1.0, 'Flipped Image - Steering Angle:-0.3347006')
```

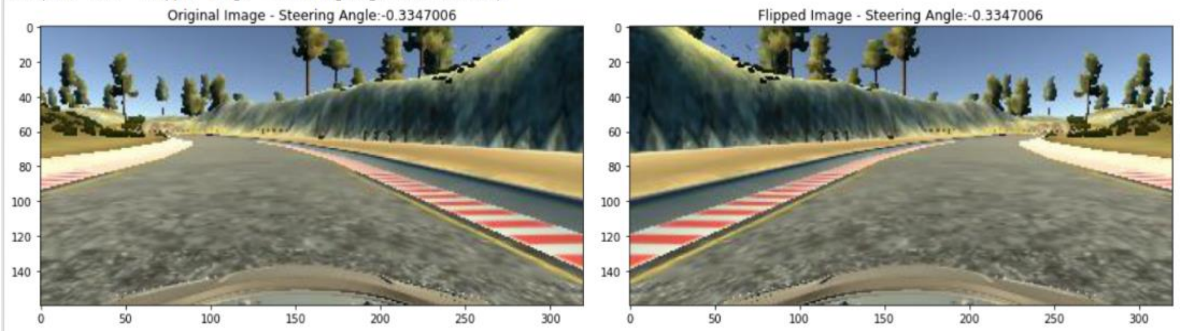

Original Image - Steering Angle:-0.3347006
Flipped Image - Steering Angle:-0.3347006

Figure 17: Image Augmentation – Flip

```
Text(0.5, 1.0, 'Preprocessed Image')
```

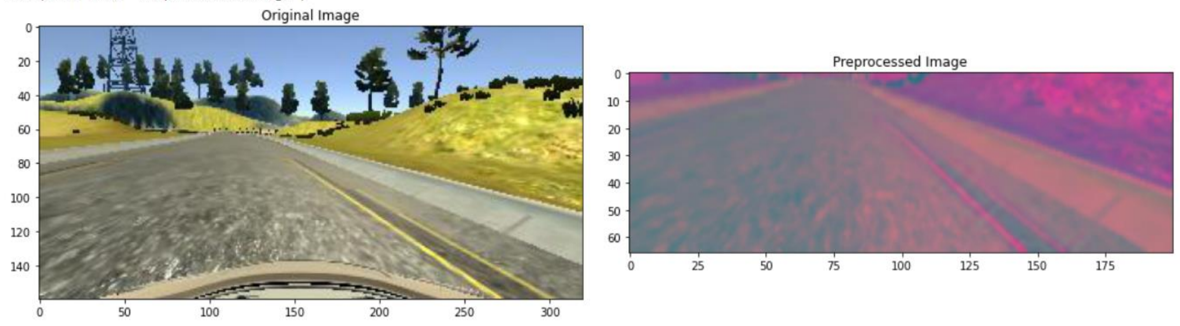
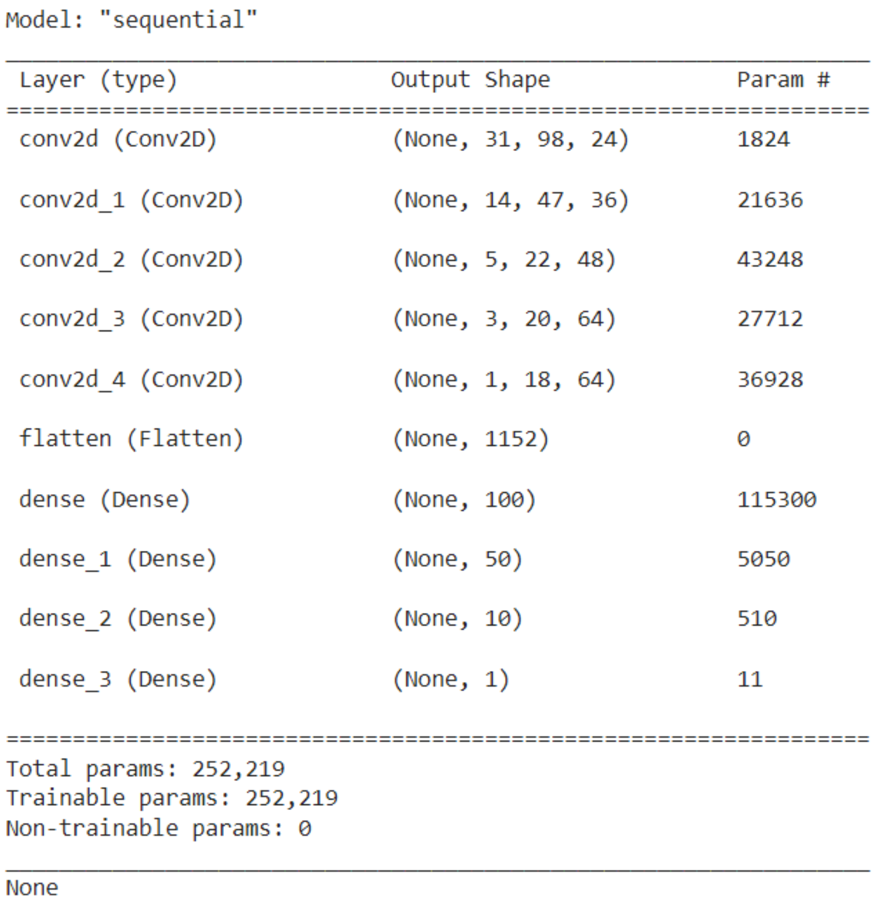


Figure 18: Image Preprocessing



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 31, 98, 24)        1824      
                                                                 
 conv2d_1 (Conv2D)           (None, 14, 47, 36)        21636     
                                                                 
 conv2d_2 (Conv2D)           (None, 5, 22, 48)         43248     
                                                                 
 conv2d_3 (Conv2D)           (None, 3, 20, 64)         27712     
                                                                 
 conv2d_4 (Conv2D)           (None, 1, 18, 64)         36928     
                                                                 
 flatten (Flatten)           (None, 1152)              0         
                                                                 
 dense (Dense)               (None, 100)               115300    
                                                                 
 dense_1 (Dense)             (None, 50)                5050      
                                                                 
 dense_2 (Dense)             (None, 10)                510       
                                                                 
 dense_3 (Dense)             (None, 1)                 11        
                                                                 
=================================================================
Total params: 252,219
Trainable params: 252,219
Non-trainable params: 0
_________________________________________________________________
None
```

Figure 19: Nvidia Model

6.1 Database and performance metrics

I. TRAFFIC SIGNAL IDENTIFICATION

The database we are using is called 'German traffic signs' from Bitbucket, a Git-based source code repository hosting service. This repository consists of 43 classes and is split into 39209 training images and 12630 test images. Figure 20 represents five random images from each of the 43 classes. It contains a broad set of images with several different backgrounds and varying lighting conditions.

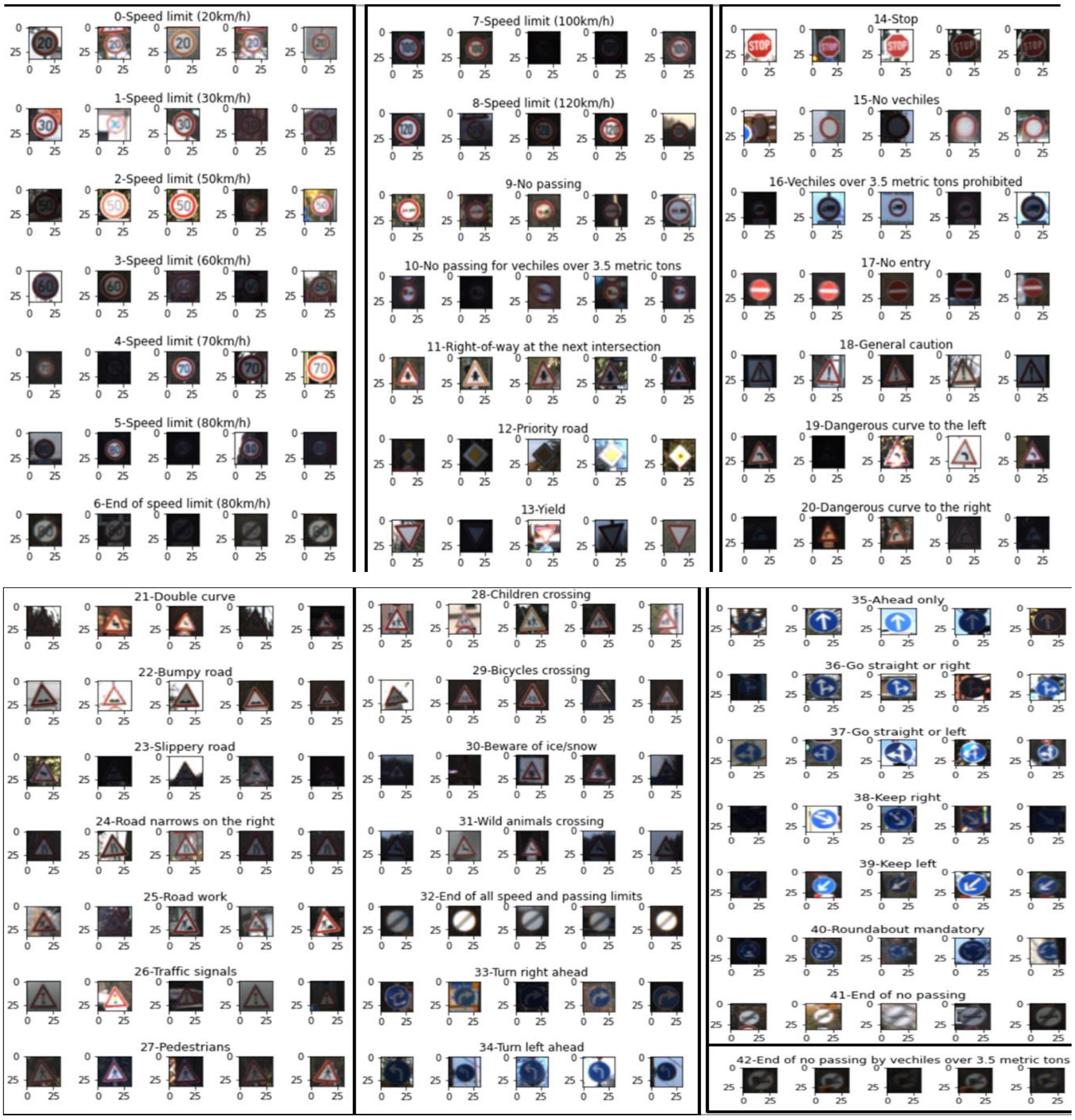


Figure 20: 43 Different Traffic Signals

To get a clear picture of the training dataset, the distribution of the images for each class is shown in the graph below in Figure 21.

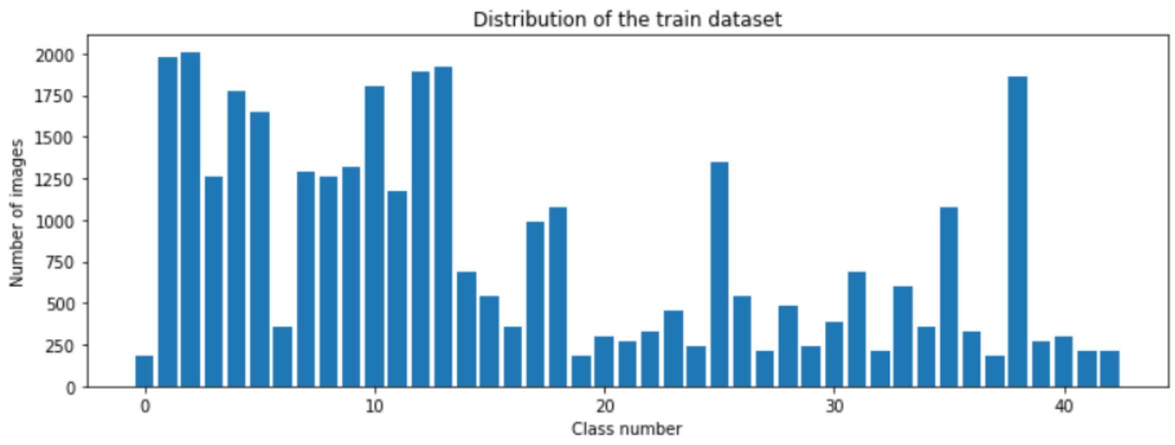


Figure 21: Distribution of images for each class

The model has a validation accuracy of 98.44% and a test accuracy of 96.68%.

II. LANE DETECTION

Canny's edge detector is the most effective method among the edge detection algorithms. By the end of Hough's transformation, multiple lines were identified instead of a single lane. We use the function 'polyfit', which returns a vector coefficient describing slope and y-intercept.

III. BEHAVIOURAL CLONING

The data used to train the CNN model to predict the steering angle is collected by manually controlling the car on Udacity's Self-driving car simulator. The images are captured from three different angles (left, right and centre) during this recording and fed into the CNN model as training parameters. The dataset consists of 19,654 images captured during the recording. A CSV file is also created recording the steering, throttle, reverse and speed values. This dataset is pushed into a git repository. Further, the dataset is refined based on steering angles removing excess images, with a steering angle of 0. 80% of the data is chosen as the training set, and 20% consists of the validation set. The validation loss of the model is 3.44%.

7. RESULT AND DISCUSSION

I. TRAFFIC SIGNAL IDENTIFICATION

Initially, we used the LeNet model to implement traffic signal classification, but we faced two challenges: (1) Low accuracy and (2) Overfitting data as seen in the below Figures 22 and 23.

<u>LeNet Model</u>

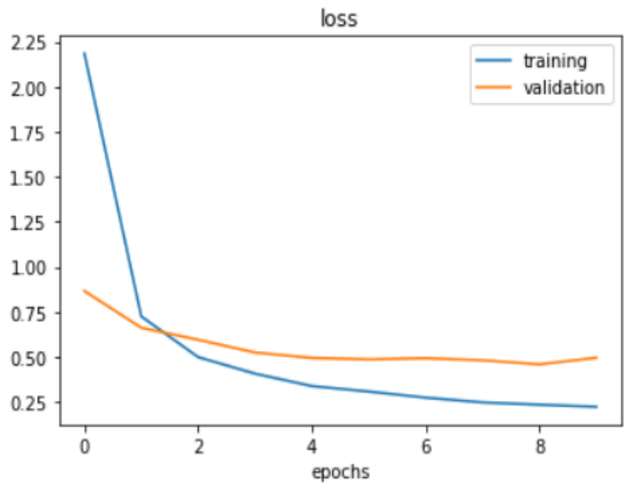


Figure 22: Training and Validation loss vs epochs for LeNet Model

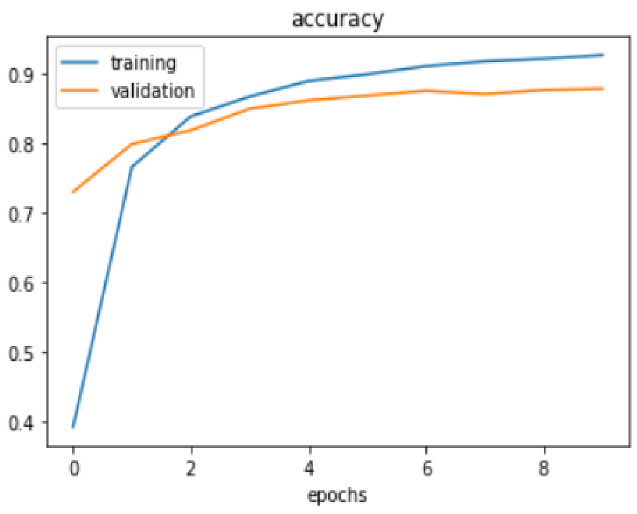


Figure 23: Training and validation accuracy vs epochs for LeNet Model

So, we had to make the following modifications to counter these issues and thus improve the performance of our model.

- First, we changed our learning rate from 0.01 to 0.001. This improved our training accuracy, but our validation accuracy suffered a little.

- Then we tried increasing the number of filters inside the convolutional layer in an attempt to extract more features. So, we doubled the number of filters. This improved our accuracy, but the model was still overfitting.
- To solve this, we increased the number of convolution layers. This reduced the number of our total parameters since the dimensions of our image also decreased with each convolutional layer. So, by the time our image reaches the fully connected layer, it has much smaller dimensions. This also reduces the computing power and increases accuracy.
- But despite this, overfitting was still present. So, we added another dropout layer after pooling, which successfully stopped the model from overfitting.
- But while testing our model, some of the images were classified incorrectly. So, we added a fit generator right before the definition of the model. A fit generator uses data augmentation to create a new dataset and add variety to our training dataset. This allows the model to extract relevant features more accurately and allows it to obtain more feature-related data. And this was an important step to do because we have a smaller dataset and a wider variety. In doing this, our model was able to classify all our test images correctly.

Thus, our modified and improved model has a validation accuracy of 98.44% and a test accuracy of 96.68%. The training and validation loss and accuracy are depicted in the Figures 24 and 25. Further, a random image from the internet (Figure 26) is used to test the model and the results are shown in Figure 27.

<u>Improved Model</u>

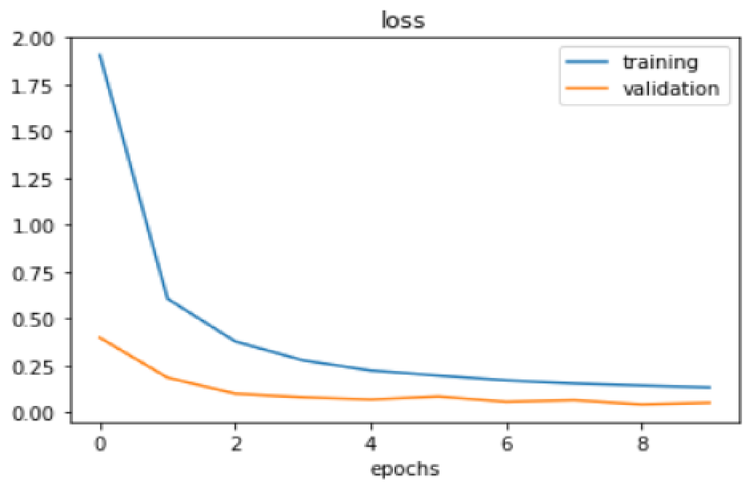


Figure 24: Training and Validation loss vs Epochs for Improved Model

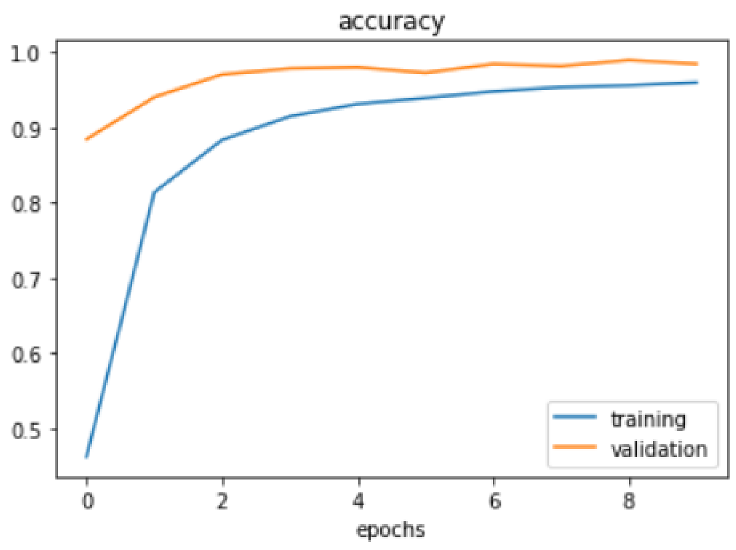


Figure 25: Training and Validation Accuracy vs Epochs for Improved Model

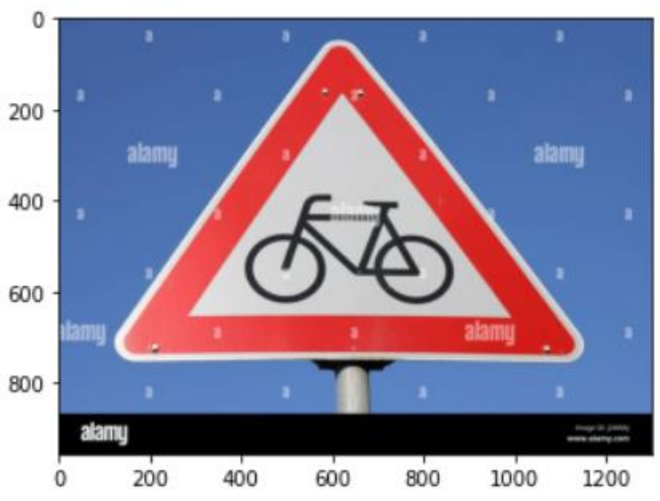
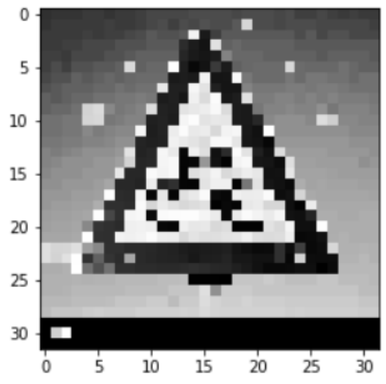


Figure 26: A random image from internet to test the model

```
#PREPROCESSING
img = np.asarray(img)
img = cv2.resize(img, (32, 32))
img = preprocessing(img)
plt.imshow(img, cmap = plt.get_cmap('gray'))
print(img.shape)
```

```
(32, 32)
```



```
#RESHAPING
img = img.reshape(1, 32, 32, 1)
```

```
#TESTING
print("predicted sign: "+ str( np.argmax(model.predict(img), axis=-1) ))
```

```
predicted sign: [29]
```

Figure 27: Traffic signal class identified successfully

In this section, we are comparing our model with 2 others that have used the same base model (LeNet) and dataset (German Traffic Signs) to evaluate our performance.

First, we are considering a paper from 2017 titled “An in-car camera system for traffic sign detection and recognition.”. Here they first filter out the images that do not have panels using the colour information parts. Then they select a region where the image blocks possibly are and extract the candidate region from this image block. Then, using deep learning, they verify the candidate areas and finally identify the type of road sign. This system primarily focuses on the HSI colour space and categorises the candidate are into smaller pieces based on colour, size, texture and similarity. Finally, they use CNN to identify the candidate region. This system was designed only for red road signals and the test accuracy depends on the selection of candidate area using the selective search method. They used gradient descent method to carry out the training. They achieved an accuracy of 80.5% in classifying the road signs.

The second paper from 2019 we have considered is titled "A robust system for road sign detection and classification using LeNet architecture based on convolutional neural network". It has two phases: detecting and classifying. They used the HOG algorithm to describe different orientations and distribution of image gradients and to extract aspect characteristics and shape to detect road signals. Then they used an SVM classifier to filter out the false positives. Then, to classify the road signs, they used Convolutional Neural Networks based on a modified LeCun's model where they classified the German traffic signs into their predefined categories to make a prediction. This model achieved an accuracy of 96.23% in classifying the traffic signs.

Table 1: Test accuracy for traffic signal identification

Paper	[11]	[12]	Our Model
Test accuracy	80.5%	96.23%	96.68%

Our model has a better test accuracy and performance compared to both the papers that were considered.

II. LANE DETECTION

The model was able to identify the lanes in a test video using Canny's Edge Detector and Hough's transform as seen in Figure 28.

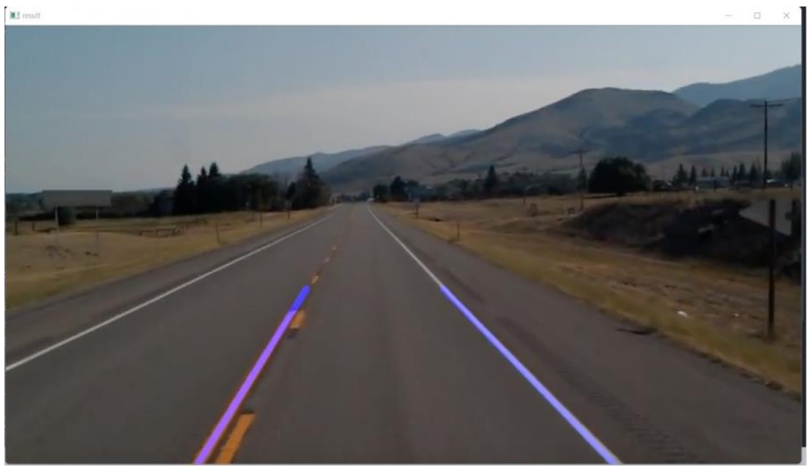


Figure 28: Lane Identified

III. BEHAVIOURAL CLONING

Initially, we start with the "relu" activation function. But, the relu function can cause "dead relu" when a node in the neural network dies and only feed a value of zero to the nodes that follow. If significant nodes fail, the loss of the model remains stagnant. To overcome this shortcoming, we introduce the "elu" activation function. Elu activation function has the same outcome as that of the relu activation function in the positive region. But, unlike the relu function, the elu function returns a negative value in the negative region. Dropout layers are removed from the Nvidia model to avoid overfitting. We run the batch generator function with ten epochs with 300 steps each to fit the model. At the end of 10 epochs, the validation loss is 3.44%. In Figure 29, the training loss is more than the validation loss throughout the loss vs epoch graph, indicating there is no overfitting.

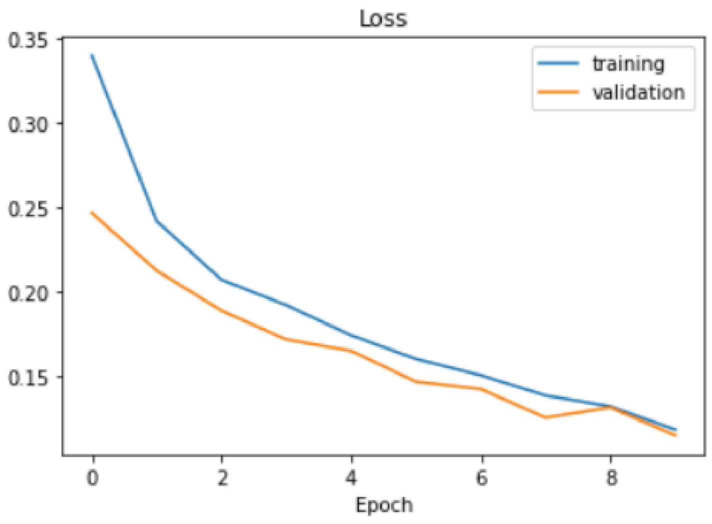
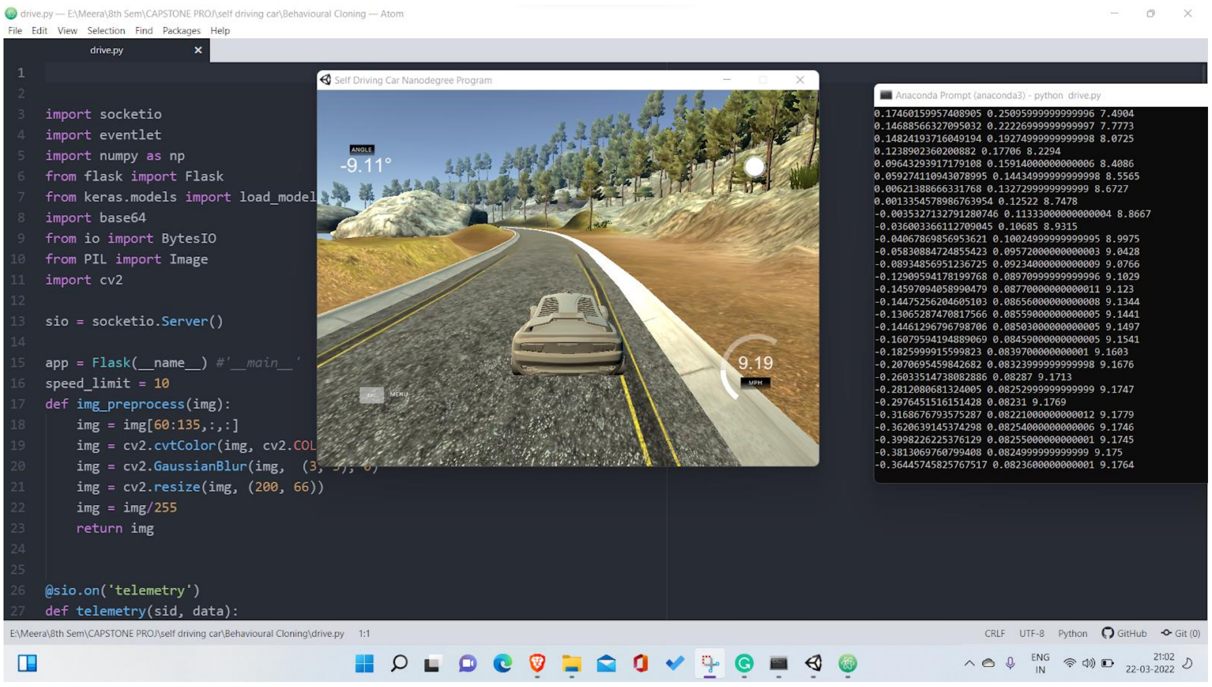


Figure 29: Training and Validation Loss vs Epoch for Behavioral Cloning model

Despite training the model in one of the two tracks, the car successfully runs on the other without crashing, verifying our model's validity as illustrated in Figures 30 and 31.

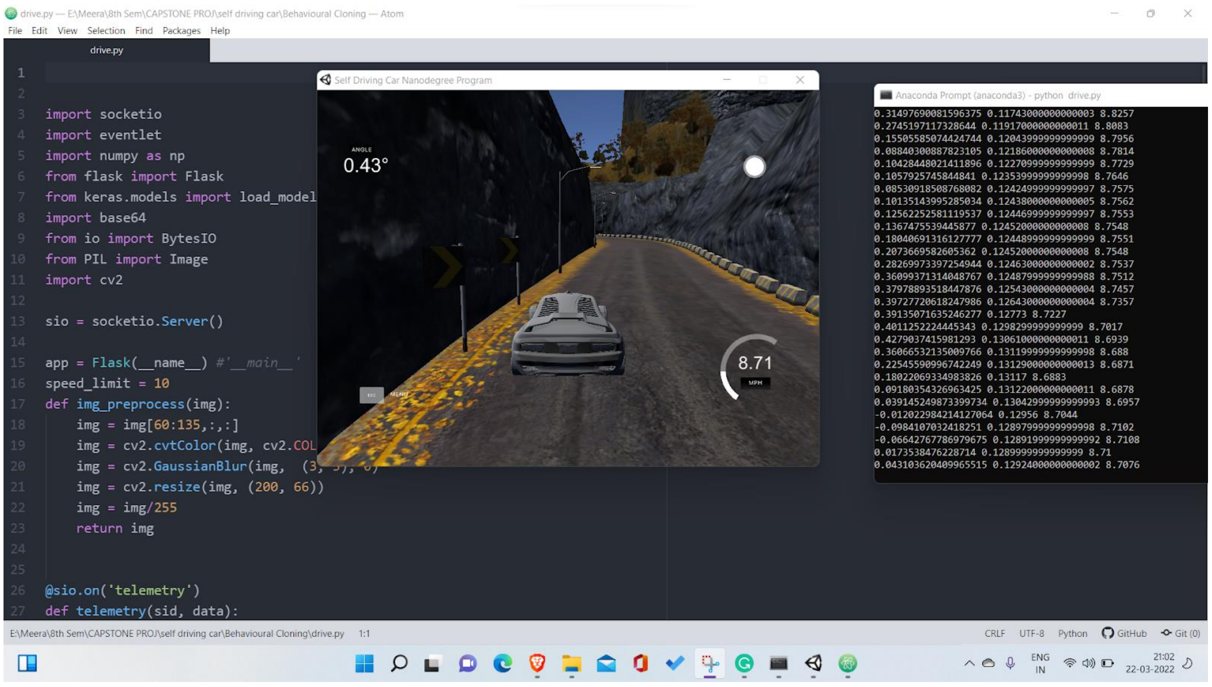

drive.py — E:\Meera\8th Sem\CAPSTONE PROJ\self driving car\Behavioural Cloning — Atom
File Edit View Selection Find Packages Help
drive.py
Self Driving Car Nanodegree Program
ANGLE
-9.11°
9.19
MPH
E:\Meera\8th Sem\CAPSTONE PROJ\self driving car\Behavioural Cloning\drive.py 1:1
CRLF UTF-8 Python GitHub Git (0)
ENG IN 21:02 22-03-2022

```python

import socketio
import eventlet
import numpy as np
from flask import Flask
from keras.models import load_model
import base64
from io import BytesIO
from PIL import Image
import cv2

sio = socketio.Server()

app = Flask(__name__) #'__main__'
speed_limit = 10
def img_preprocess(img):
    img = img[60:135,:,:]
    img = cv2.cvtColor(img, cv2.COL
    img = cv2.GaussianBlur(img,  (3, 3), 0)
    img = cv2.resize(img, (200, 66))
    img = img/255
    return img


@sio.on('telemetry')
def telemetry(sid, data):
```

Anaconda Prompt (anaconda3) - python drive.py

```
0.17460159957408905 0.25095999999999996 7.4904
0.14688566327095032 0.22226999999999997 7.7773
0.14824193716049194 0.19274999999999998 8.0725
0.1238902360200882 0.17706 8.2294
0.09643293917179108 0.15914000000000006 8.4086
0.059274110943078995 0.14434999999999998 8.5565
0.00621388666331768 0.13272999999999999 8.6727
0.0013354578986763954 0.12522 8.7478
-0.0035327132791280746 0.11333000000000004 8.8667
-0.036003366112709045 0.10685 8.9315
-0.04067869856953621 0.10024999999999995 8.9975
-0.05830884724855423 0.09572000000000003 9.0428
-0.08934856951236725 0.09234000000000009 9.0766
-0.12909594178199768 0.08970999999999996 9.1029
-0.14597094058990479 0.08770000000000011 9.123
-0.14475256204605103 0.08656000000000008 9.1344
-0.13065287470817566 0.08559000000000005 9.1441
-0.14461296796798706 0.08503000000000005 9.1497
-0.16079594194889069 0.08459000000000005 9.1541
-0.1825999915599823 0.08397000000000001 9.1603
-0.2070695459842682 0.08323999999999998 9.1676
-0.26033514738082886 0.08287 9.1713
-0.2812080681324005 0.08252999999999999 9.1747
-0.2976451516151428 0.08231 9.1769
-0.3168676793575287 0.08221000000000012 9.1779
-0.3620639145374298 0.08254000000000006 9.1746
-0.3998226225376129 0.08255000000000001 9.1745
-0.3813069760799408 0.08249999999999999 9.175
-0.36445745825767517 0.08236000000000001 9.1764
```

Figure 30: Testing the model in autonomous mode


drive.py — E:\Meera\8th Sem\CAPSTONE PROJ\self driving car\Behavioural Cloning — Atom
File Edit View Selection Find Packages Help
drive.py
Self Driving Car Nanodegree Program
ANGLE
0.43°
8.71
MPH
E:\Meera\8th Sem\CAPSTONE PROJ\self driving car\Behavioural Cloning\drive.py 1:1
CRLF UTF-8 Python GitHub Git (0)
ENG IN 21:02 22-03-2022

```python

import socketio
import eventlet
import numpy as np
from flask import Flask
from keras.models import load_model
import base64
from io import BytesIO
from PIL import Image
import cv2

sio = socketio.Server()

app = Flask(__name__) #'__main__'
speed_limit = 10
def img_preprocess(img):
    img = img[60:135,:,:]
    img = cv2.cvtColor(img, cv2.COL
    img = cv2.GaussianBlur(img,  (3, 3), 0)
    img = cv2.resize(img, (200, 66))
    img = img/255
    return img


@sio.on('telemetry')
def telemetry(sid, data):
```

Anaconda Prompt (anaconda3) - python drive.py

```
0.31497690081596375 0.11743000000000003 8.8257
0.2745197117328644 0.11917000000000011 8.8083
0.15505585074424744 0.12043999999999999 8.7956
0.08840300887823105 0.12186000000000008 8.7814
0.10428448021411896 0.12270999999999999 8.7729
0.1057925745844841 0.12353999999999998 8.7646
0.08530918508768082 0.12424999999999997 8.7575
0.10135143995285034 0.12438000000000005 8.7562
0.12562252581119537 0.12446999999999997 8.7553
0.1367475539445877 0.12452000000000008 8.7548
0.18040691316127777 0.12448999999999999 8.7551
0.2073669582605362 0.12452000000000008 8.7548
0.28269973397254944 0.12463000000000002 8.7537
0.36099371314048767 0.12487999999999988 8.7512
0.37978893518447876 0.12543000000000004 8.7457
0.39727720618247986 0.12643000000000004 8.7357
0.39135071635246277 0.12773 8.7227
0.4011252224445343 0.12982999999999999 8.7017
0.4279037415981293 0.13061000000000011 8.6939
0.36066532135009766 0.13119999999999998 8.688
0.22545590996742249 0.13129000000000013 8.6871
0.18022069334983826 0.13117 8.6883
0.09180354326963425 0.13122000000000011 8.6878
0.039145249873399734 0.13042999999999993 8.6957
-0.012022984214127064 0.12956 8.7044
-0.0984107032418251 0.12897999999999998 8.7102
-0.06642767786979675 0.12891999999999992 8.7108
0.0173538476228714 0.12899999999999999 8.71
0.043103620409965515 0.12924000000000002 8.7076
```

Figure 31: Testing the model in a new track in autonomous mode

First, we are considering a paper from 2018 titled “Self-Driving Car Steering Angle Prediction Based On Deep Neural Network An Example Of CarND Udacity Simulator” [13]. This paper has a dataset of 54000 images. They come up with two different neural network architectures. The first version consists of set of convolutional layers, followed by fully connected layer and an output layer. An initial input lambda layer is used to normalize the input pixel values. However, the second architecture, consists of increased regulators and batch-normalization. This approach was incorporated to speed up the learning and increase the probability of convergence to obtain 78.5% accuracy.

The second paper from 2021 we have considered is titled “Simulation of Self Driving Car Using Deep Learning” [14]. This model deals with two different CNN architectures. Model ‘A’ consists of 11 layers including input layer. There are five convolutional layers followed by one flatten layer and four dense layers. The model uses ‘elu’ activation function. The accuracy obtained for Model ‘A’ is 96.83%. Model ‘B’ consists of seven layers in total. The first four layers are Convolution2D layers followed by a flatten and two dense layers. Model ‘B’ obtained a higher accuracy of 76.67%.

Table 2: Validation accuracy for behavioral cloning model

Paper	[13]	[14]	Our Model
Validation accuracy	78.5%	96.83%	96.56%

8. SUMMARY

In this project we aimed to design a model of Self-driving car that is suitable for Indian roads that are often not well laid out or have several defects in them. For this purpose, we have proposed a model of a self-driving car that contains three different components. The first part includes lane detection using the Canny's Edge Detection Algorithm. For the second part, we have considered 43 diverse traffic signals. An improved LeNet model results in higher accuracy; hence it is used to identify and classify the traffic signs with an accuracy of 96.68%. Finally, we have the behavioral cloning part of capturing and reproducing human sub cognitive skills in a computer program. We use an open-source simulator by Udacity to train and test our CNN based Nvidia model with an accuracy of 96.56%.

9. REFERENCES

[1]https://indianexpress.com/article/india/1-20-lakh-deaths-due-to-negligence-road-accidents-2020-ncrb-data-7519336/#:~:text=The%20country%20also%20logged%201.35,lockdowns%2C%20according%20to%20government%20data

[2] Rocco Febbo, Brendan Flood, Julian Halloy, Patrick Lau, Kwai Wong, and Alan Ayala. 2020. “Autonomous Vehicle Control Using a Deep Neural Network and Jetson Nano.” In Practice and Experience in Advanced Research Computing (PEARC ’20), July 26–30, 2020, Portland, OR, USA. ACM, New York, NY, USA.

[3] Truong-Dong Do, Minh-Thien Duong, Quoc-Vu Dang, and My-Ha Le, “Real-Time Self-Driving Car Navigation Using Deep Neural Network” in International Conference on Green Technology and Sustainable Development (GTSD), 2018

[4] Abdur R. Fayjie, Sabir Hossain, Doukhi Oualid, and Deok-Jin Lee, “Driverless Car: Autonomous Driving Using Deep Reinforcement Learning. In Urban Environment” in 15th International Conference on Ubiquitous Robots (UR) Hawaii Convention Center, Hawai'i, USA, June 27-30, 2018

[5] U. Karni, S. S. Ramachandran, K. Sivaraman, and A. K. Veeraraghavan, "Development Of Autonomous Downscaled Model Car Using Neural Networks And Machine Learning," 2019 3rd International Conference on Computing Methodologies and Communication (ICCMC), Erode, India, 2019, pp. 1089-1094

[6] Tej Kurani, Nidhip Kathiriya, Uday Misty, Prof. Lukesh Kadu, Prof. Harish Motekar, “Self Driving Car Using Machine Learning” in International Research Journal of Engineering and Technology (IRJET), 2020.

[7] Wuttichai Vijikunsawat, Peerasask Chantngarm, “Comparison of Machine Learning Algorithms on Self-Driving Car Navigation using Nvidia Jetson Nano” in 17th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), 2020

[8] Vu Thanh Dat, Phan Quang Huy, Nguyen Dinh Tra, Tran Hai Anh, Bui Ngoc Anh, Ngo Tung Son, “A Deep Learning Based Implementation for Self-Driving Car” in Association for Computing Machinery, 2021.

[9] Aditya Kumar Jain, “Working model of Self-driving car using Convolutional Neural Network, Raspberry Pi and Arduino” in Proceedings of the 2nd International conference on Electronics, Communication and Aerospace Technology (ICECA 2018), IEEE, 2018.

[10] Ruturaj Kulkarni, Shruti Dhavalikar, Sonal Bangar, “Traffic Light Detection and Recognition for Self Driving Cars using Deep Learning”, IEEE, 2018.

[11] Huang S-C, Lin H-Y, Chang C-C (2017) An in-car camera system for traffic sign detection and recognition. In: Joint 17th world congress of international fuzzy systems association and 9th international conference on soft computing and intelligent systems (IFSA-SCIS)

[12] Amal Bouti, Med Adnane Mahraz, Jamal Riff, Hamid Tairi (2019) A robust system for road sign detection and classification using LeNet architecture based on convolutional neural network. In: Springer

[13] M.V. Smolyakov , A.I. Frolov , V.N. Volkov , I.V. Stelmashchuk , “Self-Driving Car Steering Angle Prediction Based On Deep Neural Network An Example Of CarND Udacity Simulator” in 12th International Conference on Application of Information and Communication Technologies (AICT), IEEE, 2018.

[14] Sangita Lade, Parth Shrivastav, Saurabh Waghmare, Sudarshan Hon, Sushil Waghmode, Shubham Teli, “Simulation of Self Driving Car Using Deep Learning” in International Conference on Emerging Smart Computing and Informatics (ESCI), 2021.

[15] https://bitbucket.org/jadslim/german-traffic-signs/src/master/

Autonomous Car using Deep Learning, Convolutional Neural Networks and Computer Vision

Srinidhi R and Meera L| Dr. Vaegae Naveen Kumar| SENSE

Motivation/ Introduction

Road travels are inevitable in a dynamic country like India. Humans are prone to delinquency and error. Increasing population indefinitely escalates the probability of error. According to NCRB, India has recorded 1.20 Lakh death cases due to negligence related to road accidents in 2020 alone! This whooping number is mainly due to distracted driving, drunk driving, speeding, and breaking traffic rules. This project is an attempt to reduce the accidents that occur in India due to careless driving and save thousands of lives

SCOPE of the Project

Our main focus in this project is to simulate a Self-Driving Car for Indian atmosphere. Our project includes lane detection, traffic signal identification, and using Behavioural cloning model to train the car for a smooth ride in autonomous mode. The objective of this project is to achieve improved accuracy.

Methodology

1. **Traffic Signal Identification**: Convolutional neural networks and Keras are used to build a model to classify the signals. An image dataset with 43 different signals from BitBucket is utilized for this project. First, we are converting the RGB image into a greyscale image to reduce the number of channels and make the computation easier. Then we apply histogram equalization to standardize the lighting since the images that we encounter are not all going to be uniform; some might be too bright and some too dark, so it is essential to equalize them. Histogram equalization also results in higher contrast which helps in feature extraction. And finally, we normalize the image by dividing every pixel value by 255. We then reshape the X arrays, and one hot encode the data labels before giving these images as inputs to the CNN model. We are using a modified LeNet model for identifying and classifying road signals.

2. **Lane Detection**: The input data is from the camera that is to be installed in the vehicle. This image is first converted from an RGB scale to a greyscale image to reduce the number of channels from 3 to 1, thus reducing the computational time and power required to process the image. Then we apply Gaussian blur to the image to remove noise which can otherwise create false edges and thus result in errors while detecting a lane. We then use the Canny edge detection algorithm, which uses the variation in intensity and traces out the edges by filling the pixels in white. We then create a region of interest to maneuver the car on the desired path. Finally, we use Hough's transform to trace out the course for the vehicle and overlay the lanes onto original image.

3. **Behavioral Cloning:** We use an open-source simulator by Udacity to train and test our model. The car is driven through the training track multiple times to collect sufficient images to prepare the vehicle to operate autonomously. We introduce four image augmentation techniques; zoom, panning, altering brightness and flip. About 50% of the initial dataset undergoes image augmentation, significantly increasing the variety. To familiarize the vehicle, we use the popular Nvidia model. Normalized images pass through five convolutional layers with multiple filters. The convolutional layers aid in feature extraction. A flattened layer is introduced, followed by three fully connected or dense layers. Finally, a single dense layer is added as the output. Flask, a micro web framework and Socket.IO, a JavaScript library, connect the model with the simulator. Socket.IO enables real-time bidirectional communication with the simulator.

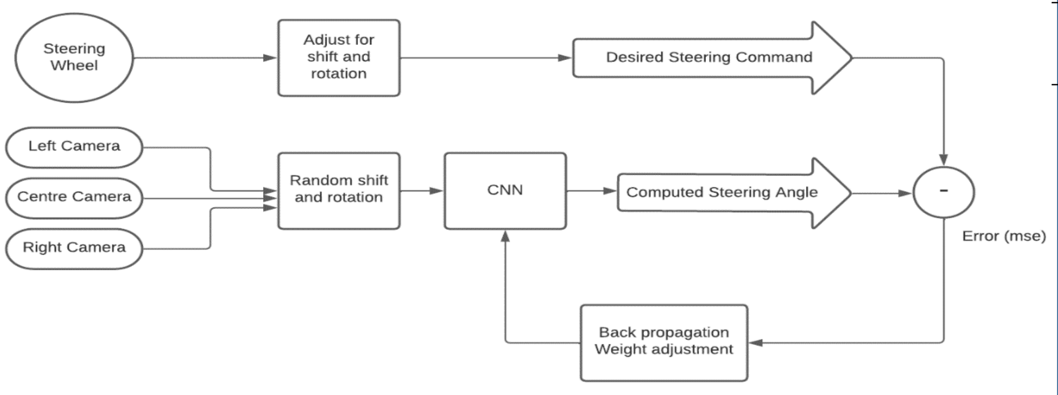


Figure 1: Block diagram for behavioural cloning

Results

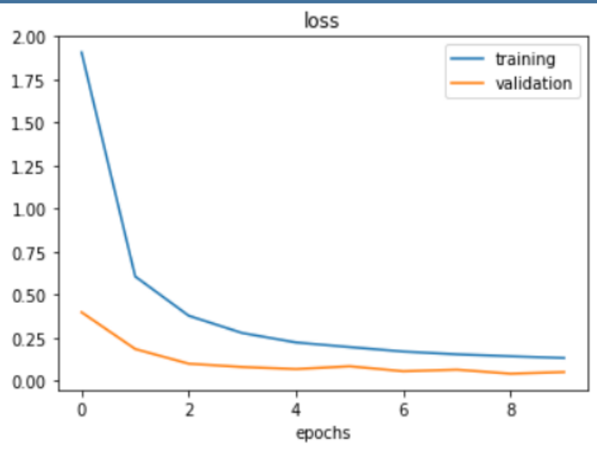


Figure 2

Figure 2: Training and Validation loss vs Epochs for Improved Model. In this figure, we can see the validation loss curve is below the training loss curve indicating there is no overfitting.

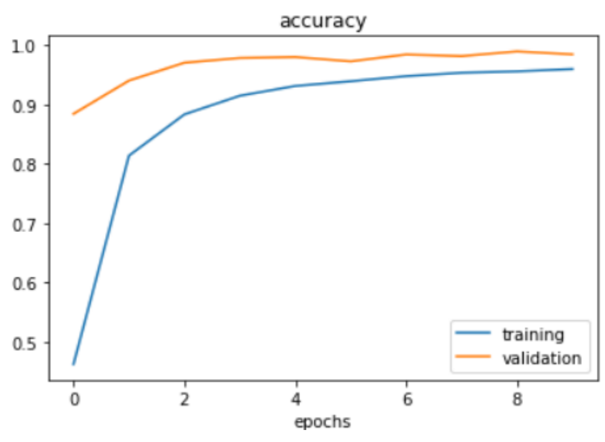


Figure 3

Figure 3: Training and Validation accuracy vs Epochs for Improved Model. In this figure, we can see the validation accuracy curve is above the training accuracy curve indicating there is no overfitting. Our model achieved a test accuracy of 96.68%.

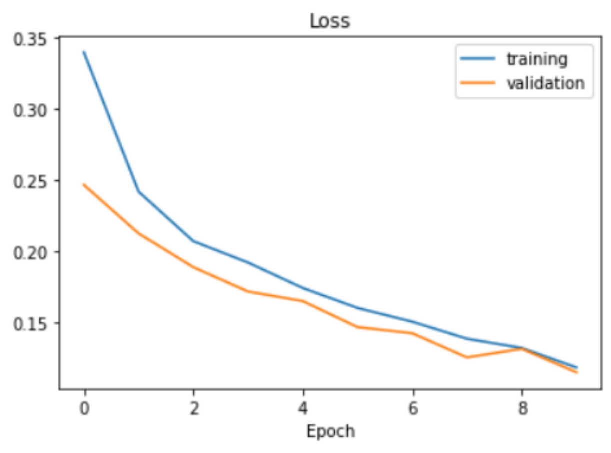


Figure 4

Figure 4: Training and Validation Loss vs Epoch for Behavioral Cloning model. In this figure, we can see the validation loss curve is below the training loss curve indicating there is no overfitting. Our model achieved a validation accuracy of 96.56%.

Conclusion/ Summary

In this project, we have proposed a model of a self-driving car. This project contains three different parts. The first part includes lane detection using the Canny's Edge Detection Algorithm. For the second part, we have considered 43 diverse traffic signals. An improved LeNet model results in higher accuracy; hence it is used to identify and classify the traffic signs. Finally, we have the behavioral cloning part of capturing and reproducing human sub cognitive skills in a computer program. We use an open-source simulator by Udacity to train and test our CNN based Nvidia model.

Contact Details

srinidhi.r2018@vitstudent.ac.in meera.l2018@vitstudent.ac.in

Acknowledgments/ References

[1] Huang S-C, Lin H-Y, Chang C-C (2017) An in-car camera system for traffic sign detection and recognition. In: Joint 17th world congress of international fuzzy systems association and 9th international conference on soft computing and intelligent systems (IFSA-SCIS)
[2]Amal Bouti, Med Adnane Mahraz, Jamal Riff, Hamid Tairi (2019) A robust system for road sign detection and classification using LeNet architecture based on convolutional neural network. In: Springer
[3] M.V. Smolyakov , A.I. Frolov , V.N. Volkov , I.V. Stelmashchuk , "Self-Driving Car Steering Angle Prediction Based On Deep Neural Network An Example Of CarND Udacity Simulator" in 12th international Conference on Application of Information and Communication Technologies (AICT), IEEE, 2018.

Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

School of Electronics Engineering

CERTIFICATE OF AUTHENTICATION OF CAPSTONE PROJECT REPORT

This is to certify that **Ms. Meera L (18BEC0597)** has done capstone project **“Autonomous Car Using Deep Learning, Convolutional Neural Networks and Computer Vision”** under my supervision. I authenticate the submission of project report and e-poster. The project report is satisfactory and adheres to the standards of **VELLORE INSTITUTE OF TECHNOLOGY.** She has communicated a research paper **“Autonomous Car Using Deep Learning, Convolutional Neural Networks and Computer Vision”** for the Scopus Journal titled **Automotive Innovation**.

[illegible]

Dr. Vaegae Naveen Kumar
Associate Professor
School of Electronics Engineering
VIT, Vellore, Tamil Nadu, India
E-mail: vegenaveen@vit.ac.in
+91-9566560355, +91-8072346003.

From: Automotive Innovation (AUIN)
Sent: 03 May 2022 11:28
To: L Meera
Subject: Automotive Innovation - Submission Confirmation Autonomous Car Using Deep Learning, Convolutional Neural Networks and Computer Vision for co-author - [EMID:2004e5eb8d8ecc7a]

Re: "Autonomous Car Using Deep Learning, Convolutional Neural Networks and Computer Vision"
Full author list: Srinidhi Ramesha; L Meera; NAVEEN KUMAR VAEGAE

Dear Ms. L Meera,

We have just received the submission entitled: "Autonomous Car Using Deep Learning, Convolutional Neural Networks and Computer Vision" for possible publication in Automotive Innovation, and you are listed as one of the co-authors.

The manuscript has been submitted to the journal by Dr. NAVEEN KUMAR VAEGAE who will be able to track the status of the paper through his/her login.

If you have any objections, please contact the editorial office as soon as possible. If we do not hear back from you, we will assume you agree with your co-authorship.

Thank you very much.

With kind regards,

Springer Journals Editorial Office
Automotive Innovation