A DRIVING DECISION STRATEGY BASED ON ML FOR AN AUTONOMOUS VEHICLE

A Major Project Report Submitted to

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY, HYDERABAD

In Partial Fulfillment of the requirement For the Award of the Degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

Submitted by

BINGI VINAY KUMAR

(HT.N0: 18N05A0503)

Under the supervision of

DR. MANISH TIWARI

Associate Professor

Department of Computer Science and Engineering

SREE CHAITANYA COLLEGE OF ENGINEERING

(Affiliated to JNTUH, HYDERABAD)

THIMMAPUR, KARIMNAGAR, TELANGANA-505527

JUNE-2021

SREE CHAITANYA COLLEGE OF ENGINEERING

(Affiliated to JNTUH, HYDERABAD)

THIMMAPUR, KARIMNAGAR, TELANGANA-505527

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Major project report entitled **"A DRIVING DECISION STRATEGY BASED ON ML FOR AN AUTONOMOUS VEHICLE"** is being submitted by Mr. **BINGI VINAY KUMAR,** bearing hall ticket number: **18N05A0503** for partial fulfillment of the requirement for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering** discipline to the **Jawaharlal Nehru Technological University, Hyderabad** during the academic year 2020 - 2021 is a bona fide work carried out by him under my guidance and supervision.

The result embodied in this report has not been submitted to any other University or institution for the award of any degree of diploma.

Project Guide

Dr. MANISH TIWARI
Associate Professor
Department of CSE

Head of the Department

Mr. KHAJA ZIAUDDIN
Associate Professor
Department of CSE

EXTERNAL EXAMINER

SREE CHAITANYA COLLEGE OF ENGINEERING

(Affiliated to JNTUH, HYDERABAD)

THIMMAPUR, KARIMNAGAR, TELANGANA-505527

Department of Computer Science and Engineering

DECLARATION

I**, BINGI VINAY KUMAR**, student of **Bachelor of Technology in Computer Science and Engineering,** during the academic year: 2020-2021**,** hereby declare that the work presented in this Project work entitled **"A DRIVING DECISION STRATEGY BASED ON ML FOR AN AUTONOMOUS VEHICLE"** is the outcome of my own bona fide work and is correct to the best of my knowledge and this work has been undertaken taking care of Engineering Ethics and carried out under the supervision of **Dr. MANISH TIWARI, Associate Professor**.

It contains no material previously published or written by another person nor material which has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgment has been made in the text.

BINGI VINAY KUMAR (HT.N0:18N05A0503)

DATE:

PLACE: KARIMNAGAR

SREE CHAITANYA COLLEGE OF ENGINEERING

(Affiliated to JNTUH, HYDERABAD)

THIMMAPUR, KARIMNAGAR, TELANGANA-505527

Department of Computer Science and Engineering

ACKNOWLEDGEMENTS

The Satisfaction that accomplishes the successful completion of any task would be incomplete without the mention of the people who make it possible and whose constant guidance and encouragement crown all the efforts with success.

I would like to express my sincere gratitude and indebtedness to my project supervisor **Dr. MANISH TIWARI, Associate Professor**, Department of Computer Science and Engineering, Sree Chaitanya College of Engineering, LMD Colony, Karimnagar. for his valuable suggestions and interest throughout the course of this project

I am also thankful to Head of the department **Mr. KHAJA ZIAUDDIN, Associate Professor & HOD**, Department of Computer Science and Engineering, Sree Chaitanya College of Engineering, LMD Colony, Karimnagar for providing excellent infrastructure and a nice atmosphere for completing this project successfully

I Sincerely extend my thanks to **Dr. G. VENKATESHWARLU, Principal,** Sree Chaitanya College of Engineering, LMD Colony, Karimnagar. for providing all the facilities required for completion of this project.

I convey my heartfelt thanks to the lab staff for allowing me to use the required equipment whenever needed.

Finally, I would like to take this opportunity to thank my family for their support through the work.

I sincerely acknowledge and thank all those who gave directly or indirectly their support in completion of this work.

BINGI VINAY KUMAR

ABSTRACT

In this paper, a Driving Decision Strategy(DDS) for navigation of Autonomous vehicle(self-driving vehicle) is proposed. Although this problem has been under research for several years, I noticed that the elevated accuracy results have not been achieved yet. Therefore, using machine learning to learn vehicle driving patterns and deep learning to obtain accurate road lane detection and vehicle trajectory a method using convolutional neural network (CNN) for training and simulation of unmanned vehicle model on the UDACITY platform has been developed. Here, I have mounted three cameras in front of a vehicle to follow three directions precisely left, right and center position to collect data. Data is collected in the form of images and videos that are captured from three cameras. In this research the label with two parameters, steering angle and speed from each image would also be created. After collecting the data, these parameters will be achieved by training CNN that is used to navigate the vehicle. With the combination of three cameras, the accuracy of this navigation task is improved significantly. When vehicle deviates to the left, I will compute the error of the steering angle value between the middle and left position. Then, the steering angle value will be adjusted to control the vehicle so that it could run in the center of the lane. Similarly, in the case when vehicles deviate to the right. Based on the simulation platform of UDACITY, I simulated and obtained the results with an accuracy beyond 96%.

TABLE OF CONTENTS

CHAPTER	TITLE	PAGE NO'S
	CERTIFICATE	**ii**
	DECLARATION	**iii**
	ACKKNOWLEDGEMENT	**iv**
	ABSTRACT	**v**
	TABLE OF CONTENTS	**vi-vii**
	LIST OF FIGURES	**viii-ix**
	LIST OF TABLES	**x**
CHAPTER 1	**INTRODUCTION**	**1-2**
	1.1 Scope of the project	1
	1.2 Existing System	1-2
	1.3 Proposed System	2
CHAPTER 2	**SYSTEM ANALYSIS**	**3-6**
	2.1 Literature Survey	3-5
	2.2 Feasibility Study	5-6
	2.3 System Specification	6
CHAPTER 3	**SYSTEM DESIGN AND DEVELOPMENT**	**7-13**
	3.1 Input Design	7
	3.2 Output Design	8
	3.3 UML Diagrams	8-13
CHAPTER 4	**IMPLEMENTATION**	**14-20**
	4.1 Programming Languages and Libraries	14-17
	4.2 Domains	17-20
CHAPTER 5	**MODULES**	**21-33**
	5.1 Data Collection	21-22
	5.2 CNN Algorithms	22-29
	5.3 Steering Adjustment	29-30
	5.4 Prediction	30-33
CHAPTER 6	**NETWORK ARCHITECTURE**	**34-38**

CHAPTER 7	**EXPERIMENTS**	**39-41**
	7.1 Training Process, Data and Feature	39-41
	7.2 Autonomous Vehicle Navigation	41
CHAPTER 8	**TESTING**	**42-46**
CHAPTER 9	**SOURCE CODE**	**47-50**
	9.1 DriveApp	47
	9.2 LoadData	48
	9.3 TrainModel	49-50
CHAPTER 10	**OUTPUT SCREEN**	**51-57**
	10.1 Left Side Steer	51-53
	10.2 Center Steer	54-55
	10.3 Right Side Steer	56-57
	CONCLUSION	**58-59**
	REFERENCES	**60**

LIST OF FIGURES

Fig No	Name of the Figure	Page No's
Fig 1.1	The Proposed Scheme for Autonomous Vehicle Navigation	2
Fig 3.1	Use Case Diagram	10
Fig 3.2	Class Diagram	11
Fig 3.3	Sequence Diagram	12
Fig 3.4	Activity Diagram	13
Fig 5.1	High-Level View of the Data Collection System	22
Fig 5.2	Neural Network Identification	22
Fig 5.3	Real to Pixel Image	23
Fig 5.4	CNN Recognizing Images	24
Fig 5.5	ReLU Function Graph	25
Fig 5.6	Input to Feature Mapping	25
Fig 5.7	Input Feature Mapping to Rectified Feature Mapping	26
Fig 5.8	Applied Pooling Filters	26
Fig 5.9	Input Image to Pooling Layer	27
Fig 5.10	Input Image to Final Layer	27
Fig 5.11	Input Layer to Fully Connected Layer	28
Fig 5.12	CNN Architecture.	29
Fig. 6.1	The Network Architecture	34
Fig. 6.2	Dropout Neural Net Model	35
Fig. 6.3	The Input Data	36
Fig. 6.4	The Augment Data Original Image	37
Fig. 6.5	The Augment Data Flip Image	37
Fig. 6.6.	The Diagram for Accuracy Evaluation of Proposed Network	38
Fig. 7.1	The Accuracy of the Proposed Deep Neural Network Architecture	39
Fig. 7.2	The Visualization Output of Three Convolutional Layers	40
Fig. 7.3	The Loss of the Proposed Deep Neural Network Architecture	41

Fig 10.1	Left Steer (1)	51
Fig 10.2	Left Steer (2)	51
Fig 10.3	Left Steer (3)	52
Fig 10.4	Left Steer (4)	52
Fig 10.5	Left Steer (5)	53
Fig 10.6	Left Steer (6)	53
Fig 10.7	Center Steer (1)	54
Fig 10.8	Center Steer (2)	54
Fig 10.9	Center Steer (3)	55
Fig 10.10	Center Steer (4)	55
Fig 10.11	Right Steer (1)	56
Fig 10.12	Right Steer (2)	56
Fig 10.13	Right Steer (3)	57
Fig 10.14	Right Steer (4)	57
Fig. 11.1	The Result Predicted Steering Angle and Speed of the Car After Training	58

LIST OF TABLES

Table No	Name of the Table	Page No's
Table 11.1	The Effect of the Number of Samples to the Accuracy of the Model	58
Table 11.2	The Effect of the Epoch Sample to the Accuracy of the Model	58

CHAPTER 1
INTRODUCTION

1.1. SCOPE OF THE PROJECT

Today, with the considerable development of technology and transportation, many vehicles are equipped with the self driving mode to support the driver to maintain health while driving long distances as well as to reduce traffic accident risk.

Navigating trajectory for vehicles was one of the most important aspects of the development of the unmanned vehicle model. There were many methods to do this, but the way to get the best results and match with Industrial Revolution 4.0 is using an algorithm regarding Machine Learning field. Specifically, we made the convolutional neural network (CNN) algorithm to navigate for autonomous vehicles.

Deep learning is a subfield of machine learning that is inspired by an artificial neural network. A specific kind of such a deep network is the convolutional neural network, which is commonly referred to as CNN or ConvNet. The difference of CNN, in comparison to the traditional neural network, is the number of neural in a class may be reduced but the number of hidden layers is greater and called a deep network. They are trained by backpropagation strategy. So, it can build the intelligent systems with high accuracy.

1.2. EXISTING SYSTEM

Formerly, there were many studies on navigation for vehicles, including methods lane detection such as Real-time Lane detection for autonomous navigation, a lane tracking system for intelligent vehicle application, lane detection with moving vehicles in the traffic scenes or the other papers regarding this method are mentioned in. Although this method gives convincing accuracy about lane detection there are several reasons that make the unsuccessful detection.

The first reason is subjective. After detecting two lines we need to calculate and draw a virtual line at the center then estimate the offset angle between the body of the vehicles and the virtual line to adjust steering of the car so that the vehicles are always in the middle of two lines under any circumstance. The calculations of steering angle that are mentioned above are

complicated and can cause many errors. The second reason is objective. Several of the roads are lack of lane or the lane marking is blurred. Also, when the vehicles are running in the sloping street, the camera was mounted at previous will head to the sky and do not keep up with lane at the ahead. This can also lead to detection will be incorrect.

In addition, the GPS method is also applied for navigating self-driving vehicles. If GPS is used standalone, it will cause an error is quite high so apply this method on the model of unmanned vehicles will be exceedingly difficult and requires an adjustment of the error .

1.3. PROPOSED SYTEM

Noticed with the restrictions mentioned above, we propose a method using a convolutional neural network to navigate the self-driving vehicles.

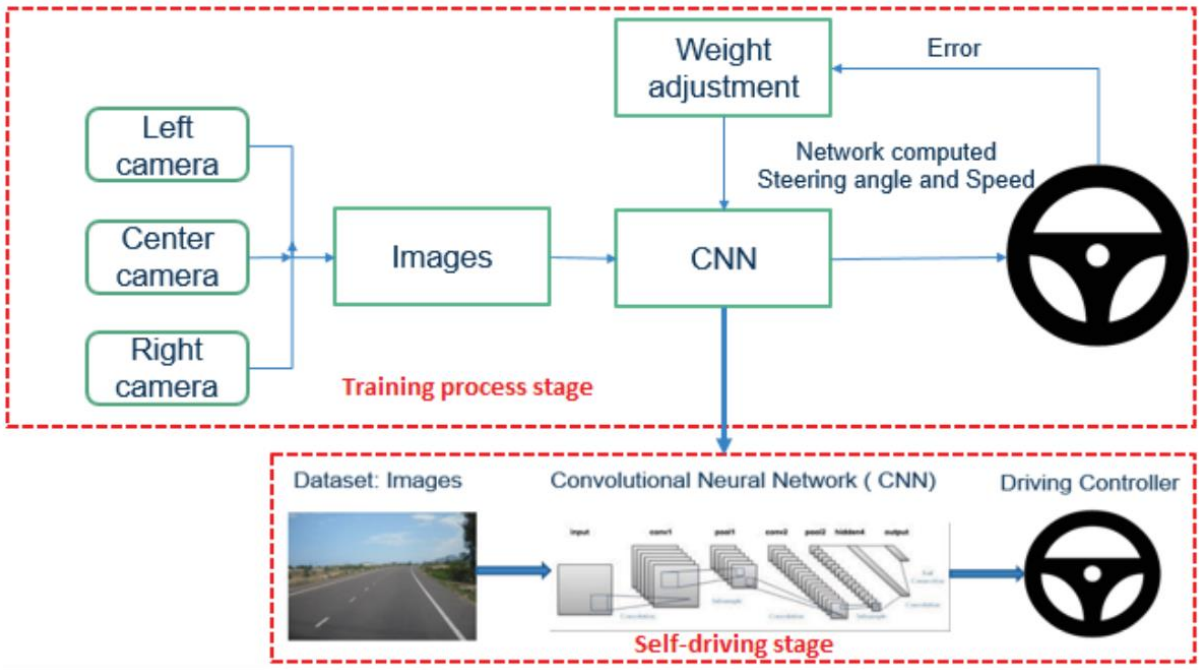


Fig. 1.1 The Proposed Scheme for Autonomous Vehicle Navigation

In this work, we demonstrate autonomous driving in a simulation environment by predicting steering wheel angles and speed value from raw images which are trained through CNN. Data was collected from three cameras later are preprocessed and fed into a CNN that then calculates the value of steering angle and speed. The proposed command is compared to the desired command for that image and the weight of the CNN are adjusted to obtain the better result (Above figure).

CHAPTER 2

SYSTEM ANALYSIS

2.1. LITERATURE SURVEY

1. Real-time lane detection for autonomous navigation

A lane detection method based on a road model or feature needs correct acquisition of information on the lane in an image. It is inefficient to implement a lane detection algorithm through the full range of an image when it is applied to a real road in real time because of the calculating time. The paper defines two searching ranges for detecting a lane in a road. First is a search mode that searches the lane without any prior information of the road. Second is a recognition mode, which is able to reduce the size and change the position of a search range by predicting the position of a lane through the acquired information in a previous frame. It allows us to extract accurately and efficiently the edge candidate points of a lane without any unnecessary searching. By means of an inverse perspective transform that removes the perspective effect on the edge candidate points, we transform the edge candidate information in the image coordinate system into the plane-view image in the world coordinate system. We define a linear approximation filter and remove faulty edge candidate points by using it. The paper aims to approximate more correctly the lane of an actual road by applying the least-mean square method with the fault-removed edge information for curve fitting.

2. A lane tracking system for intelligent vehicle applications

An image-based lane tracking system for use in intelligent vehicles is developed. For each frame, the algorithm develops estimates of the geometry and width of the current lane ahead of the vehicle and also the position and orientation of the vehicle with respect to the center-line of the lane. Basic image processing techniques are used to extract a candidate set of lane marker locations from the image. These are used to generate a pool of center-line candidates with properties dependent on the lane markers. A method of elimination based on dynamic programming is used to isolate a final set of center-line candidates that constitute the actual geometry of the road. The road geometry is modeled using a clothoid curve, which stipulates that the curvature of the road varies as a linear function of arc length. The clothoid center-line representation also aids in determining the offset of the vehicle from the center-line and the heading angle of the vehicle with respect to the road. Finally, a Kalman filter is applied to the

estimated parameters to preserve smoothness and to predict lane parameters for the next image frame. A set of confidence measures for the estimated data is calculated for use by a higher-level data fusion algorithm The system gives an estimate of parameters under normal traffic and driving conditions and runs in real-time on off-the-shelf hardware.

3. Lane detection with moving vehicles in the traffic scenes

A lane-detection method aimed at handling moving vehicles in the traffic scenes is proposed in this brief. First, lane marks are extracted based on color information. The extraction of lane-mark colors is designed in a way that is not affected by illumination changes and the proportion of space that vehicles on the road occupy. Next, for vehicles that have the same colors as the lane marks, we utilize size, shape, and motion information to distinguish them from the real lane marks. The mechanism effectively eliminates the influence of passing vehicles when performing lane detection. Finally, pixels in the extracted lane-mark mask are accumulated to find the boundary lines of the lane. The proposed algorithm is able to robustly find the left and right boundary lines of the lane and is not affected by the passing traffic. Experimental results show that the proposed method works well on marked roads in various lighting conditions.

4. Road lane detection using H-maxima and improved hough transform

A fast and improved algorithm with the ability to detect unexpected lane changes is aimed in this paper. A short segment of a long curve has relative low curvature which is approximated as a straight line. Based on the characteristics of physical road lane, this paper presents a lane detection technique based on H-MAXIMA transformation and improved Hough Transform algorithm which first defines the region of interest from input image for reducing searching space, divided the image into near field of view and far field of view. In near field of view, Hough transform has been applied to detect lane markers after image noise filtering. The proposed method has been developed using image processing programming language platform and was tested on collected video data. Promising result was obtained with high efficiency of detection.

5. Autonomous vehicle based in cooperative GPS and inertial systems

A system including Global Positioning Systems (GPS) and digital cartography is a good solution to carry out vehicle's guidance. However, it has inconveniences like high sensibility to multipath and interference when the GPS signal is blocked by external agents. Another system is mandatory to avoid this error. This paper presents a cooperative system based on GPS and

Inertial Navigation Systems (INS) for automated vehicle position. The control system includes a decision unit to choose which value is the correct. In case GPS is working at top precision, it takes the control. On the other part, GPS signal can be lost, and inertial control system guides the car in this occasion. A third possibility is contemplated: we receive the signal from GPS, but the accuracy is over one meter. Now, position value is obtained by means of both systems. Experimental results analyze two situations: guidance in an urban area where GPS signal can be occluded by buildings or trees during short time intervals and the possibility of loss of the signal in long time to simulate the circulation in tunnels. Good results have been observed in tests and it demonstrates how a cooperative system improves the automated vehicle guidance.

2.2. FEASIBILITY STUDY

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are

- ECONOMICAL FEASIBILITY
- TECHNICAL FEASIBILITY
- SOCIAL FEASIBILITY

ECONOMICAL FEASIBILITY

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus, the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

TECHNICAL FEASIBILITY

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources.

This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

SOCIAL FEASIBILITY

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

2.3. SYSTEM SPECIFICATION:

HARDWARE REQUIREMENTS:

- Processor : Minimum Intel i3
- Hard Disk : 500 GB.
- Ram : Minimum 8 GB.

SOFTWARE REQUIREMENTS:

- Operating system : Windows 10
- Coding Language : Python
- Front-End : Python
- IDE : PyCharm 2020 Community.

CHAPTER 3

SYSTEM DESIGN AND DEVELOPMENT

3.1. INPUT DESIGN

The input design is the link between the information system and the user. It comprises the developing specification and procedures for data preparation and those steps are necessary to put transaction data into a usable form for processing can be achieved by inspecting the computer to read data from a written or printed document or it can occur by having people keying the data directly into the system. The design of input focuses on controlling the amount of input required, controlling the errors, avoiding delay, avoiding extra steps and keeping the process simple. The input is designed in such a way so that it provides security and ease of use with retaining the privacy. Input Design considered the following things:

- What data should be given as input?
- How the data should be arranged or coded?
- The dialog to guide the operating personnel in providing input.
- Methods for preparing input validations and steps to follow when error occur.

OBJECTIVES

1. Input Design is the process of converting a user-oriented description of the input into a computer-based system. This design is important to avoid errors in the data input process and show the correct direction to the management for getting correct information from the computerized system.

2. It is achieved by creating user-friendly screens for the data entry to handle large volume of data. The goal of designing input is to make data entry easier and to be free from errors. The data entry screen is designed in such a way that all the data manipulates can be performed. It also provides record viewing facilities,

3. When the data is entered it will check for its validity. Data can be entered with the help of screens. Appropriate messages are provided as when needed so that the user will not be in maize of instant. Thus, the objective of input design is to create an input layout that is easy to follow

3.2. OUTPUT DESIGN

A quality output is one, which meets the requirements of the end user and presents the information clearly. In any system results of processing are communicated to the users and to other system through outputs. In output design it is determined how the information is to be displaced for immediate need and also the hard copy output. It is the most important and direct source information to the user. Efficient and intelligent output design improves the system's relationship to help user decision-making.

1. Designing computer output should proceed in an organized, well thought out manner; the right output must be developed while ensuring that each output element is designed so that people will find the system can use easily and effectively. When analysis design computer output, they should Identify the specific output that is needed to meet the requirements.
2. Select methods for presenting information.
3. Create document, report, or other formats that contain information produced by the system.

The output form of an information system should accomplish one or more of the following objectives.

- Convey information about past activities, current status or projections of the
- Future.
- Signal important events, opportunities, problems, or warnings.
- Trigger an action.
- Confirm an action.

3.3. UML DIAGRAMS

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.

The goal is for UML to become a common language for creating models of object-oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems.

The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.

The UML is a very important part of developing objects-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

UML Specifying:

Specifying means building models that are precise, unambiguous and complete. In particular, the UML address the specification of all the important analysis, design and implementation decisions that must be made in developing and displaying a software intensive system.

UML Visualization:

The UML includes both graphical and textual representation. It makes easy to visualize the system and for better understanding **UML Constructing.**

UML models can be directly connected to a variety of programming languages and it is sufficiently expressive and free from any ambiguity to permit the direct execution of models.

GOALS

The Primary goals in the design of the UML are as follows:

1. Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
2. Provide extendibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development process.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of OO tools market.
6. Support higher level development concepts such as collaborations, frameworks, patterns and components.
7. Integrate best practices.

USE CASE DIAGRAM:

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

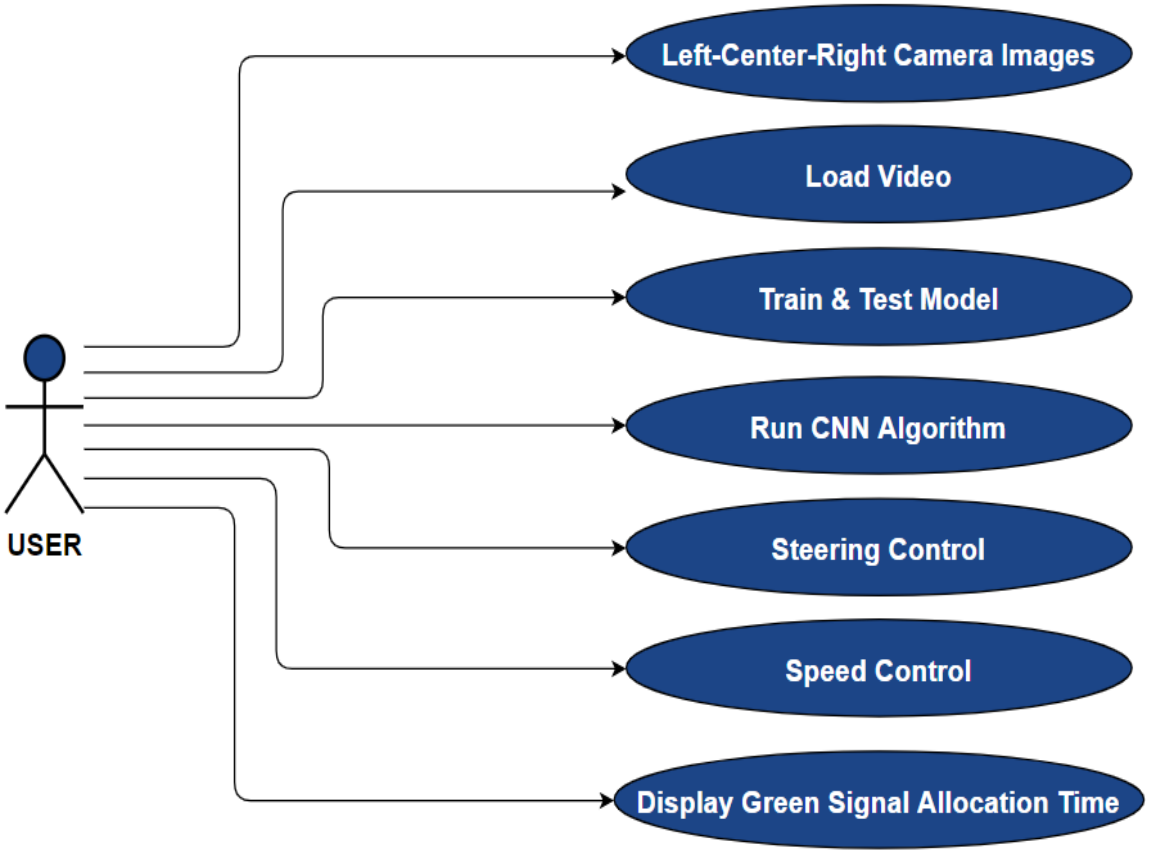


Fig. 3.1 Use Case Diagram

CLASS DIAGRAM:

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which class contains information.

Class diagrams are widely used to describe the types of objects in a system and their relationships. Class diagrams model class structure and contents using design elements such as classes, packages and objects. Class diagrams describe three different perspectives when designing a system, conceptual, specification, and implementation. These perspectives become evident as the diagram is created and help solidify the design. Class diagrams are arguably the most used UML diagram type. It is the main building block of any object-oriented solution. It shows the classes in a system, attributes and operations of each class and the relationship between each class. In most modeling tools a class has three parts, name at the top, attributes in the middle and operations or methods at the bottom. In large systems with many classes related classes are grouped together to create class diagrams. Different relationships between diagrams are show by different types of Arrows. Below is a image of a class diagram. Follow the scenario

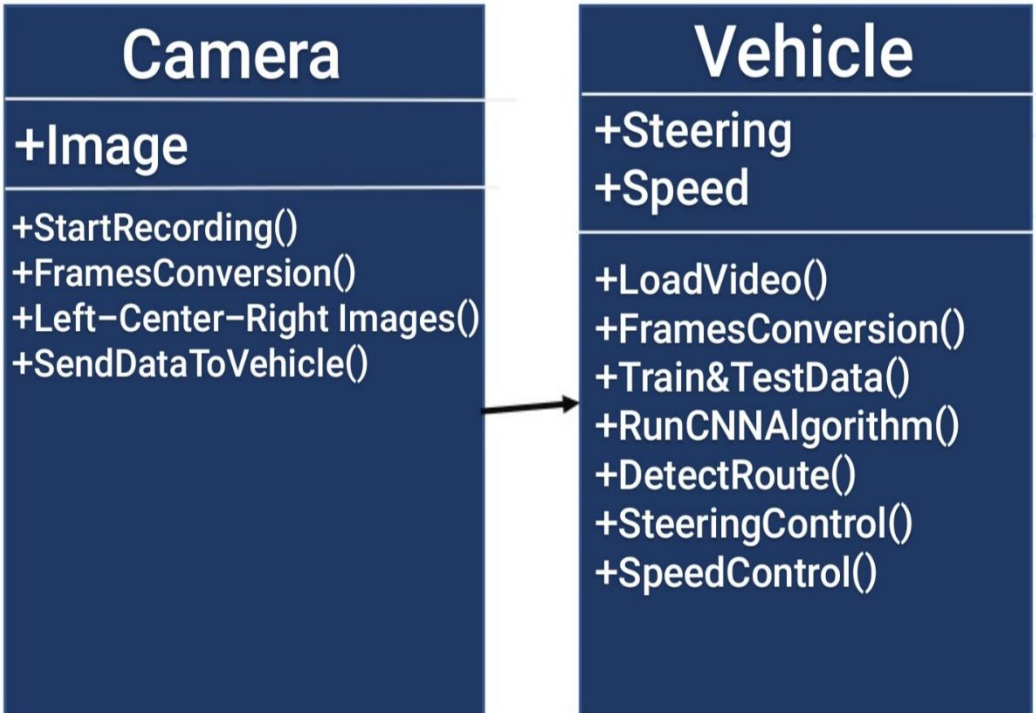


Fig: 3.2 Class Diagram

SEQUENCE DIAGRAM:

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.

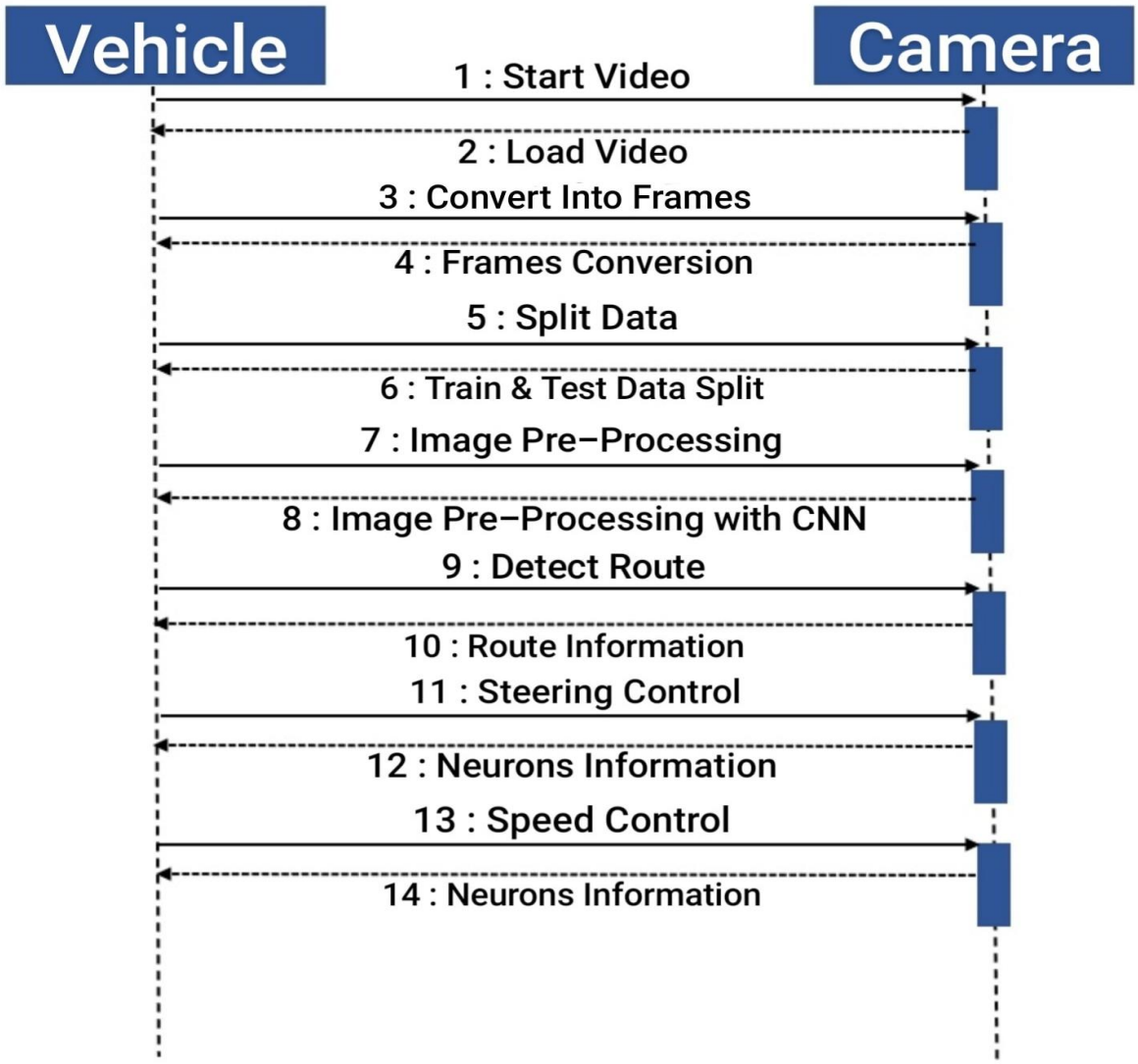


Fig 3.3 Sequence Diagram

ACTIVITY DIAGRAM:

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration, and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

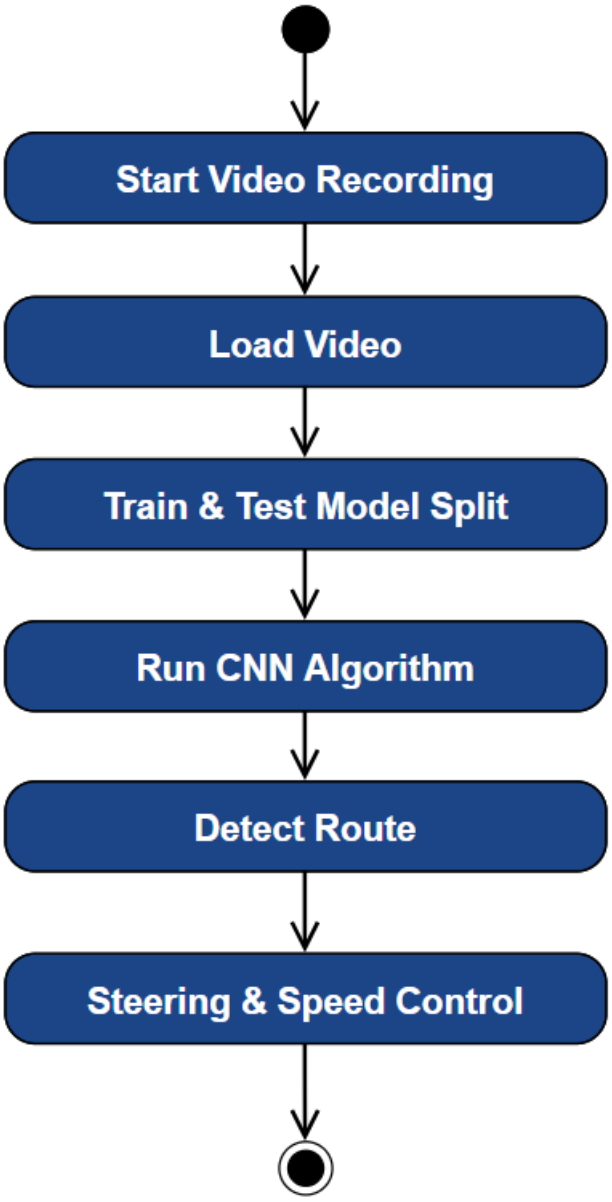


Fig 3.4 Activity Diagram

CHAPTER 4

IMPLEMENTATION

4.1. PROGRAMMING LANGUAGE & LIBRARIES

PYTHON

- Python is a high level, structured, open-source programming language that can be used for a wide variety of programming tasks.
- Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built-in data structures, combined with dynamic typing and dynamic binding, make it extremely attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed
- Python within itself is an interpreted programming language that is automatically compiled into bytecode before execution.
- It is also a dynamically typed language that includes (but does not require one to use) object-oriented features.

- NASA has used Python for its software systems and has adopted it as the standard scripting language for its Integrated Planning System.
- Python is also extensively used by Google to implement many components of its Web Crawler and Search Engines for managing its discussion groups.

History of Python

- Python was created by Guido Van Rossum.
- The design began in the late 1980s and was first released in February 1991.

NumPy

NumPy is a library for the python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays

We use **pip install numpy==1.20.3** in command prompt for numpy installation

Matplotlib

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK.

We use **pip install matplotlib** in command prompt for matplotlib installation

CV2

OpenCV (Open-Source Computer Vision Library) is a collection of algorithms for computer vision. its basics focus is on real time image processing, it is free for commercial and research use under a BSD license.

We use **pip install opencv-python** in command prompt for cv2 installation

Keras

Keras is a neural network API. It is a library written, specifically in python. Also, it works with other libraries and packages such as tensorflow which makes deep learning easier. Keras was developed to allow for quick experimentation and for fast prototyping.

We use **pip install keras==2.2.4** in command prompt for keras installation

h5py

The h5py package is a Pythonic interface to the HDF5 binary data format.

We use **pip install h5py==2.10.0** in command prompt for h5py installation

Scipy

SciPy in Python is an open-source library used for solving mathematical, scientific, engineering, and technical problems. It allows users to manipulate the data and visualize the

data using a wide range of high-level Python commands. SciPy is built on the Python NumPy extension. SciPy is also pronounced as "Sigh Pi."

TENSORFLOW

TensorFlow is a mathematical computation library for training and building your machine learning and deep learning model with a simple to use high level APIs.

TensorFlow is a free and open-source software library for machine learning. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks. Tensorflow is a symbolic math library based on dataflow and differentiable programming.

TensorFlow is also an end-to-end open-source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.

TensorFlow offers multiple levels of abstraction so you can choose the right one for your needs. Build and train models by using the high-level Keras API, which makes getting started with TensorFlow and machine learning easy.

If you need more flexibility, eager execution allows for immediate iteration and intuitive debugging. For large ML training tasks, use the Distribution Strategy API for distributed training on different hardware configurations without changing the model definition.

TensorFlow has always provided a direct path to production. Whether it’s on servers, edge devices, or the web, TensorFlow lets you train and deploy your model easily, no matter what language or platform you use.

Use TensorFlow Extended (TFX) if you need a full production ML pipeline. For running inference on mobile and edge devices, use TensorFlow Lite. Train and deploy models in JavaScript environments using TensorFlow.js.

Build and train state-of-the-art models without sacrificing speed or performance. TensorFlow gives you the flexibility and control with features like the Keras Functional API and Model Subclassing API for creation of complex topologies. For easy prototyping and fast debugging, use eager execution.

TensorFlow also supports an ecosystem of powerful add-on libraries and models to experiment with, including Ragged Tensors, TensorFlow Probability, Tensor2Tensor and BERT.

We use **pip install tensorflow==1.14.0** in command prompt for tensorflow installation

4.2. DOMAINS

MACHINE LEARNING

Machine learning is a method of teaching prediction based on some data. it is a branch of artificial intelligence, which numerically improves on data. Over as more data as add in algorithm the performance of the system is improved.

Machine learning is the practice of helping software perform a task without explicit programming or rules. With traditional computer programming, a programmer specifies rules that the computer should use. ML requires a different mindset, though. Real-world ML focuses far more on data analysis than coding. Programmers provide a set of examples, and the computer learns patterns from the data. You can think of machine learning as “programming with data”.

Machine learning algorithms make it possible for self-driving cars to exist. They allow a car to collect data on its surroundings from cameras and other sensors, interpret it, and decide what actions to take. Machine learning even allows cars to learn how to perform these tasks as good as (or even better than) humans.

Steps to solving an ML problem

There are multiple steps in the process of getting answers from data using ML. For a step-by-step overview, check out this guide that shows the complete workflow for text classification, and describes important steps like collecting a dataset, and training and evaluating a model with TensorFlow.

Anatomy of a neural network

A neural network is a type of model that can be trained to recognize patterns. It is composed of layers, including input and output layers, and at least one hidden layer. Neurons in each layer learn increasingly abstract representations of the data. For example, in this visual diagram we

see neurons detecting lines, shapes, and textures. These representations (or learned features) make it possible to classify the data.

Training a neural network

Neural networks are trained by gradient descent. The weights in each layer begin with random values, and these are iteratively improved over time to make the network more accurate. A loss function is used to quantify how inaccurate the network is, and a procedure called backpropagation is used to determine whether each weight should be increased, or decreased, to reduce the loss.

These are the three types of machine learning:

Supervised learning

In supervised learning we have several data points or samples described using predictive variables or features and the target variable our data represented in table structure. Game supervised learning is build a model its able to predict the target variable.

Unsupervised learning

It is a machine learning task the uncovering hidden patterns from unlabeled data.

Reinforcement learning (RL)

In in which machine or software agents interact with an environment reinforcement learning agents are able to automatically figure out how to optimize their behavior given a system of reward and punishments reinforcement learning draws inspiration from behavioral psychology.

COMPUTER VISION

It is a field that includes processing analyzing and understanding image in general high dimensional data from the real world in order to produce numerical and symbolic information or it is a technology of science and machine that see it obtain information from images.

Computer Vision in autonomous cars can lead to the designing and development of advanced and next-gen vehicles which can overcome driving obstacles while keeping passengers safe. Such cars can transport passengers to their destination eliminating human intervention.

For autonomous vehicles to efficiently navigate, they require a frame of reference and observe the surrounding environment using computer vision algorithms to outline a map of its surroundings and traverse the track. 3D reconstruction includes the use of computer vision to observe the outside surroundings using a depth-based 3D point cloud.

DEEP LEARNING

Deep learning is a powerful set of techniques for learning using neural network. Neural network are beautiful biologically inspired programing paradigm which enables a computer to learn from data.

Computer Vision requires a lot of Deep Learning for the task of detection of lanes.

In Lane Line Detection and Segmentation, we use Deep Learning over traditional techniques because they're faster and more efficient. Algorithms such as LaneNet are quite popular in the field of research to extract lane lines.

CONVOLUTIONAL NEURAL NETWORK

Convolutional Neural Network are designed to process data through multiple layers of arrays. This type of neural networks is used in application like image recognition of recognition of road. The primary difference between CNN and other ordinary neural network is that CNN takes input as a two-dimensional array and operates directly on the images rather than focusing on feature extraction which other neural network focus on. The dominant approach of CNN includes solutions for problems of recognition. Top companies like google and facebook have invested in research and developments towards recognition projects to get activites done with greater speed.

To perform image classification, a convolutional neural network is trained to recognize various objects, like traffic lights, lanes and pedestrians. A convolutional neural network performs convolution operations on images in order to classify them

A convolutional neural network (CNN) approach is used to implement autonomous vehicle by mapping pixels from the camera input to the steering commands. The network automatically learns the maximum variable features from the camera input, hence requires minimal human

intervention. Given realistic frames as input, the driving policy trained on the dataset by Udacity can adapt to real-world driving in a controlled environment.

A convolutional neural network uses three basic ideas:

- Local respective fields.
- Convolution
- Pooling

UDACITY

The basic goal was to use data collected from Udacity's Self Driving Car to build a model that would predict the steering angles for the vehicle. This problem was well suited for Convolutional Neural Networks (CNNs) that take in the forward-facing images from the vehicle and output a steering angle. This Challenge can be treated as a classification or regression problem. After some initial trials with classification (binning the steering angles into classes), regression proved more successful and easier to implement. A model's success was measured using the root mean square (RMS) error of the predicted steering versus the actual human steering.

The data collected from the Udacity car was images (mostly .jpg) and video from 3 cameras (left, center, right spaced 20 inches apart) and steering angles.

The goal in this project is to build a path planner that is able to create smooth, safe trajectories for the car to follow. advanced lane-finding algorithm using distortion correction, image rectification, color transforms, and gradient thresholding. Identified lane curvature and vehicle displacement. Overcomes environmental challenges such as shadows and pavement changes.

CHAPTER 5
MODULES

5.1. DATA COLLECTION

Training data is generally collected by driving on a wide variety of roads and in a diverse set of lighting and weather conditions. Most road data is collected on highways. Other road types include two-lane roads (with and without lane markings), residential roads with parked cars, tunnels, and unpaved roads. Data is collected in clear, cloudy, foggy, snowy, and rainy weather, both day and night. In some instances, we may face the sun being low in the sky, resulting in glare reflecting from the road surface and scattering from the windshield. Data is acquired by using our drive-by-wire test vehicle with cameras placed precisely at left, right and centre positions at the front of the car. The system has no dependencies on any particular vehicle make or model. Drivers were encouraged to maintain full attentiveness, but otherwise drive as they usually do. long hours of driving data should be collected.

Three cameras are mounted behind the windshield of the data-acquisition car. Time-stamped video from the cameras is captured simultaneously with the steering angle applied by the human driver. This steering command is obtained by tapping into the vehicle's Controller Area Network (CAN) bus. In order to make our system independent of the car geometry, we represent the steering command as 1/r where r is the turning radius in meters. We use 1/r instead of r to prevent a singularity when driving straight (the turning radius for driving straight is infinity). 1/r smoothly transitions through zero from left turns (negative values) to right turns (positive values). Training data contains single images sampled from the video, paired with the corresponding steering command (1/r). Training with data from only the human driver is not sufficient.

The network must learn how to recover from mistakes. Otherwise, the car will slowly drift off the road. The training data is therefore augmented with additional images that show the car in different shifts from the centre of the lane and rotations from the direction of the road. Images for two specific off-centre shifts can be obtained from the left and the right camera. Additional shifts between the cameras and all rotations are simulated by viewpoint transformation of the image from the nearest camera.

Precise viewpoint transformation requires 3D scene knowledge which we do not have. We therefore approximate the transformation by assuming all points below the horizon are on flat ground and all points above the horizon are infinitely far away. This works fine for flat terrain but it introduces distortions for objects that stick above the ground, such as cars, poles, trees, and buildings. Fortunately, these distortions do not pose a big problem for network training.

The steering label for transformed images is adjusted to one that would steer the vehicle back to the desired location and orientation in two seconds.

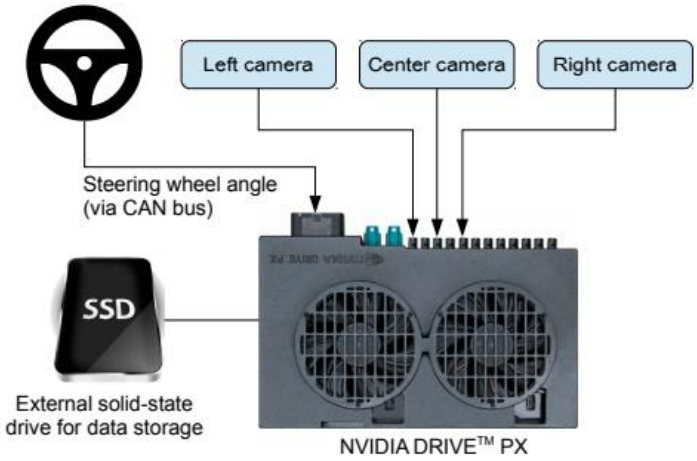


Fig 5.1 High-Level View of the Data Collection System

5.2. CNN ALGORITHM

A convolutional neural network is a feed-forward neural network that is generally used to analyze visual images by processing data with grid-like topology. It's also known as a ConvNet. A convolutional neural network is used to detect and classify objects in an image. Below is a neural network that identifies two types of flowers: Orchid and Rose.

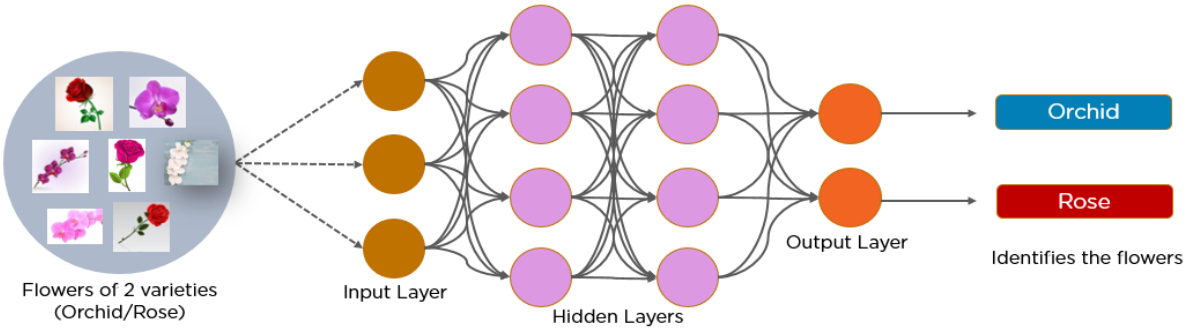


Fig 5.2 Neural Network Identification

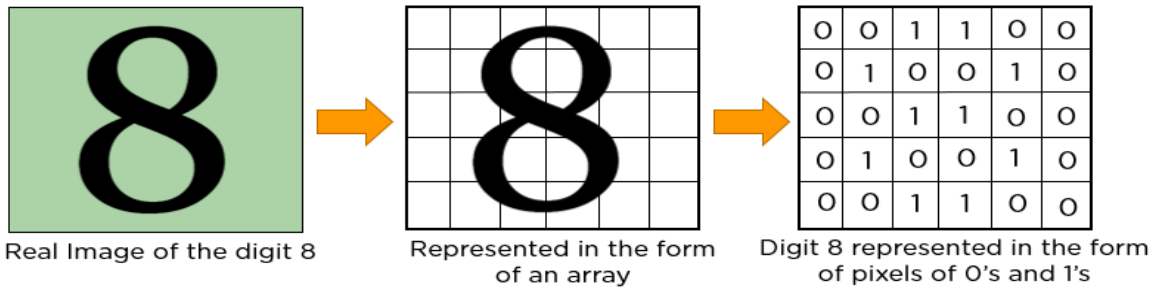


Fig 5.3 Real to Pixel Image

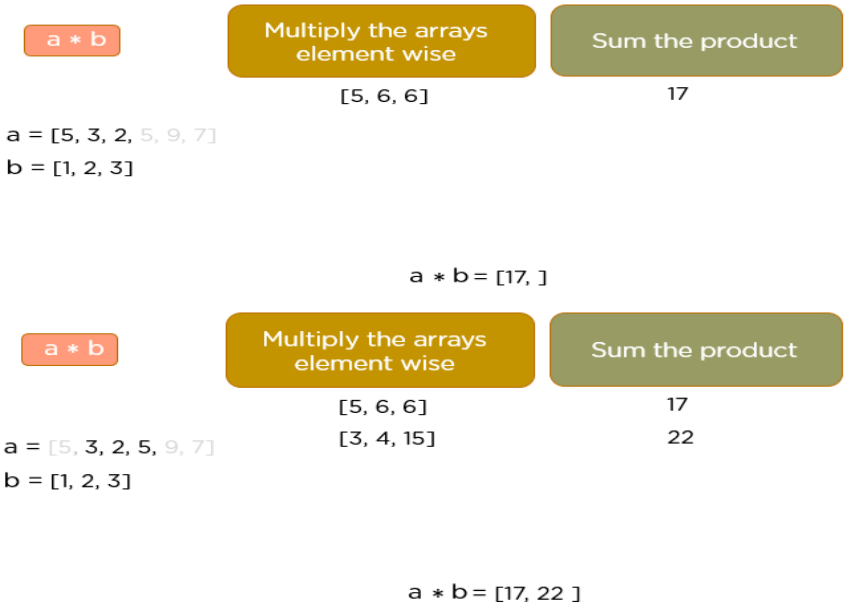
In CNN, every image is represented in the form of an array of pixel values . The convolution operation forms the basis of any convolutional neural network. Let us understand the convolution operation using two matrices, *a* and *b*, of 1 dimension.

a = [5,3,2,5,9,7]

b = [1,2,3]

In convolution operation, the arrays are multiplied element-wise, and the product is summed to create a new array, which represents a*b.

The first three elements of the matrix a are multiplied with the elements of matrix b. The product is summed to get the result.



The next three elements from the matrix a are multiplied by the elements in matrix b, and the product is summed up. This process continues until the convolution operation is complete.

CNN Recognizing Images, Consider the following images:

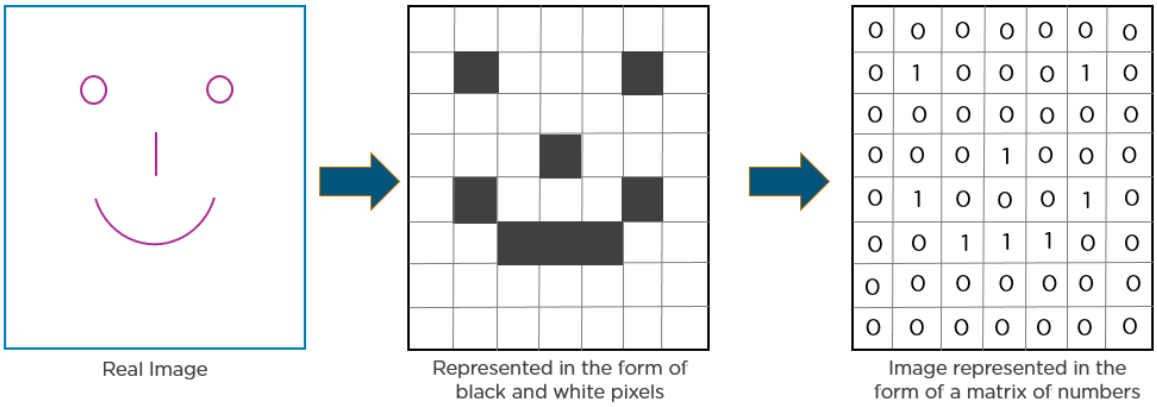


Fig 5.4 CNN Recognizing Images

As you can see from the above diagram, only those values are lit that have a value of 1

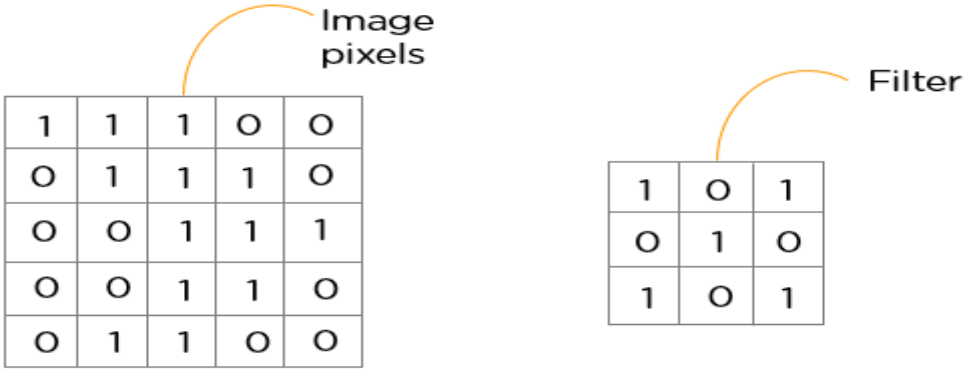
Layers in Convolutional Neural Network

A convolution neural network has multiple hidden layers that help in extracting information from an image. The four important layers in CNN are:

- Convolution layer
- ReLU layer
- Pooling layer
- Fully connected layer

Convolution Layer :

This is the first step in the process of extracting valuable features from an image. A convolution layer has several filters that perform the convolution operation. Every image is considered as a matrix of pixel values. Consider the below 5x5 image whose pixel values are either 0 or 1. There is also a filter matrix with a dimension of 3x3. Slide the filter matrix over the image and compute the dot product to get the convolved feature matrix.



ReLU Layer :

ReLU stands for the rectified linear unit. Once the feature maps are extracted, the next step is to move them to a ReLU layer.

ReLU performs an element-wise operation and sets all the negative pixels to 0. It introduces non-linearity to the network, and the generated output is a rectified feature map. Below is the graph of a ReLU function:

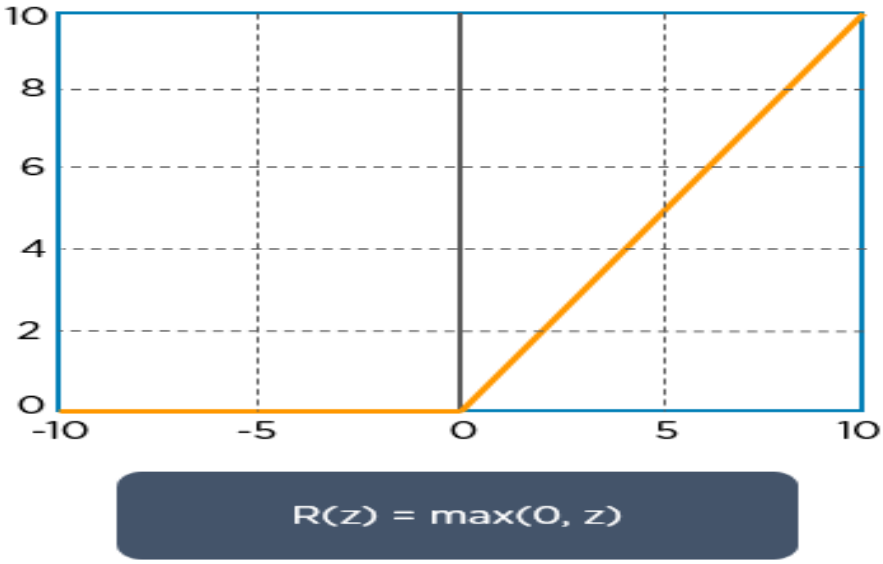


Fig 5.5 ReLU Function Graph

The original image is scanned with multiple convolutions and ReLU layers for locating the features.

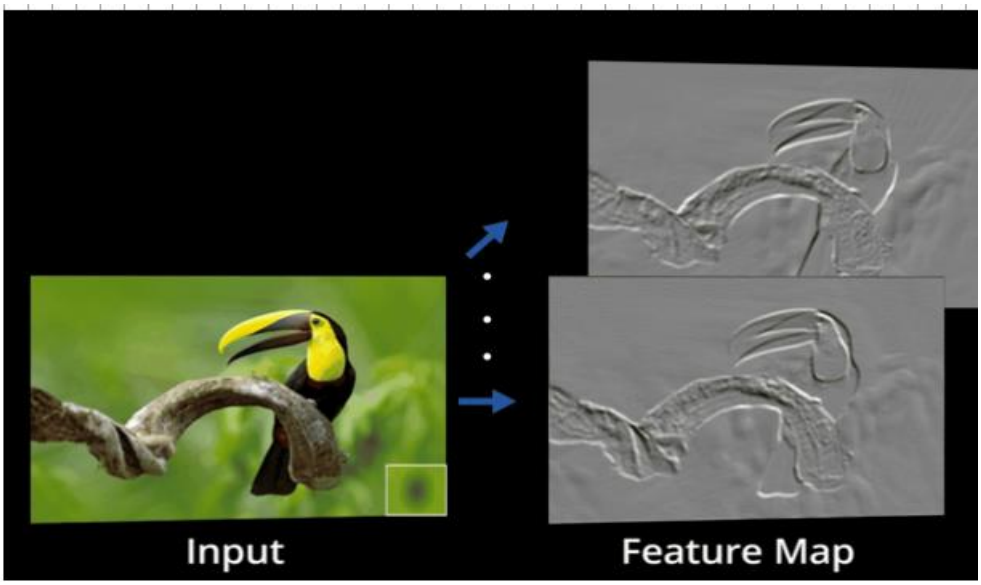


Fig 5.6 Input to Feature Mapping

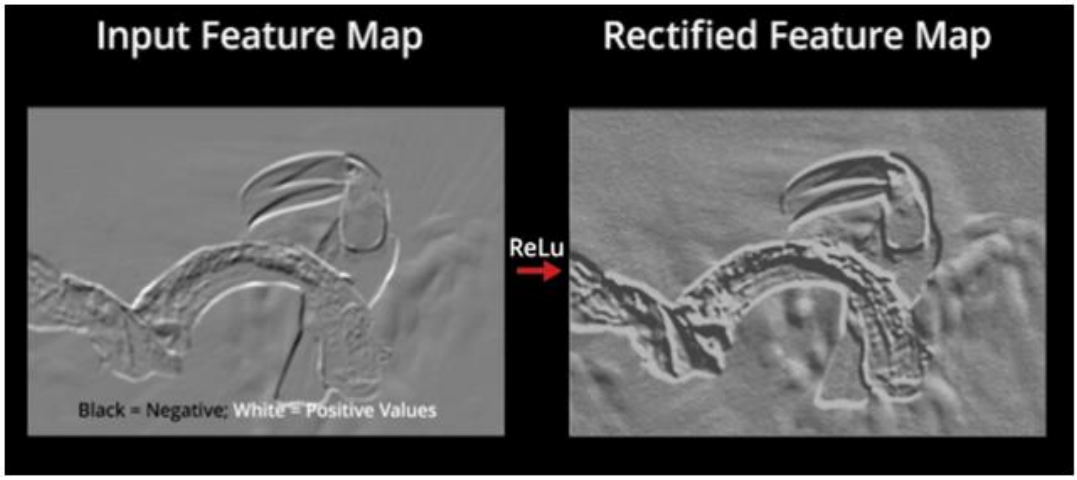
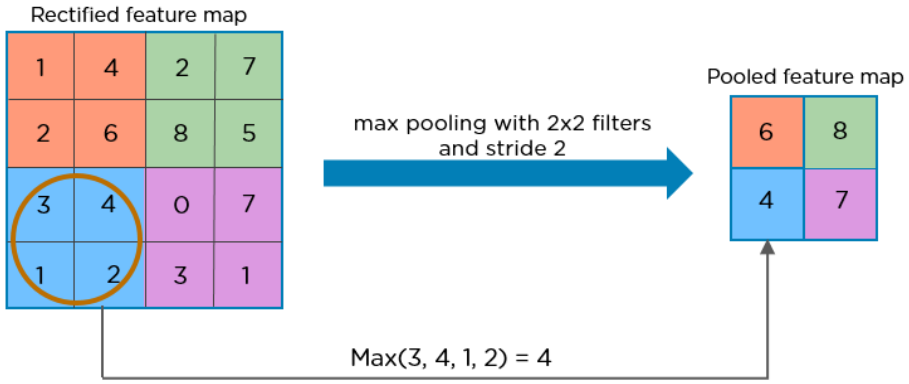


Fig 5.7 Input Feature Mapping to Rectified Feature Mapping

Pooling Layer :

Pooling is a down-sampling operation that reduces the dimensionality of the feature map. The rectified feature map now goes through a pooling layer to generate a pooled feature map.



The pooling layer uses various filters to identify different parts of the image like edges, corners, body, feathers, eyes, and beak.

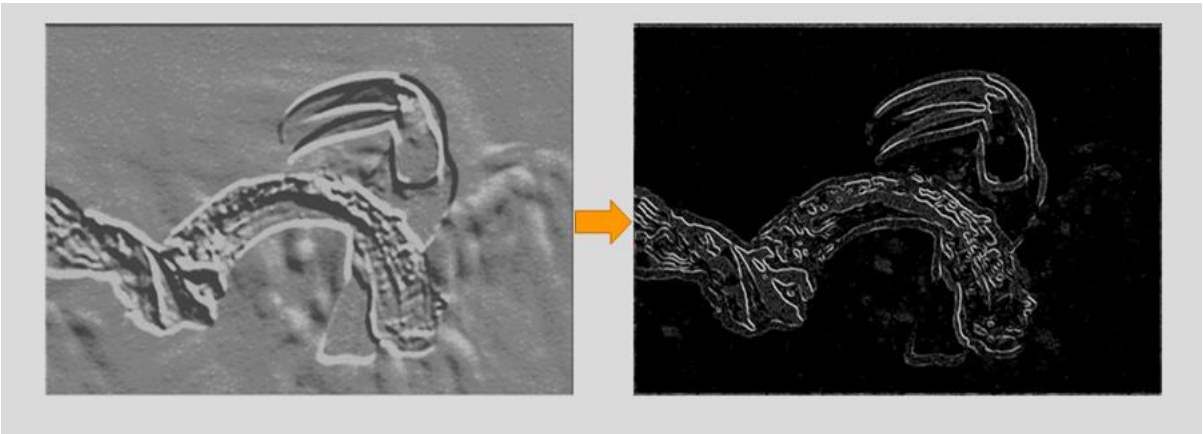


Fig 5.8 Applied Pooling Filters

Here's how the structure of the convolution neural network looks so far:

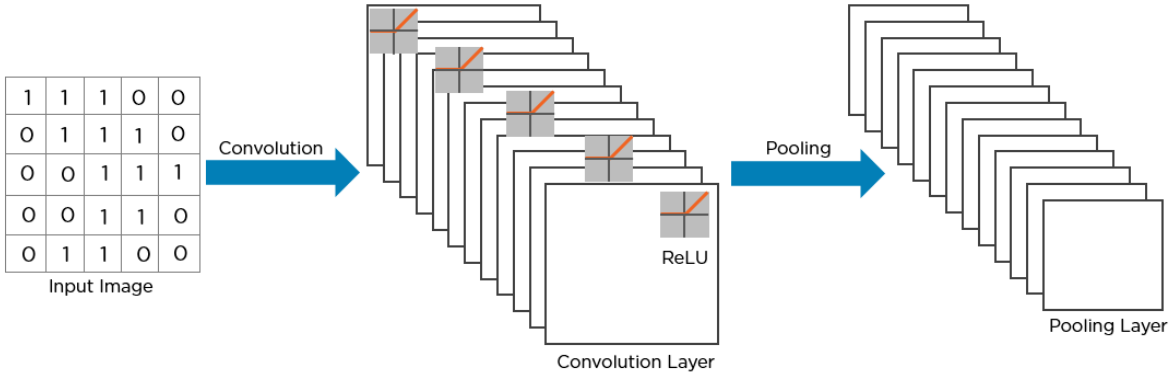
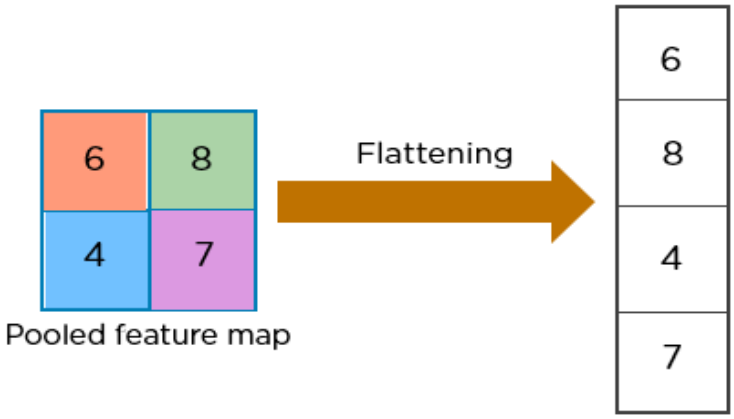


Fig 5.9 Input Image to Pooling Layer

The next step in the process is called flattening. Flattening is used to convert all the resultant 2-Dimensional arrays from pooled feature maps into a single long continuous linear vector.



The flattened matrix is fed as input to the fully connected layer to classify the image.

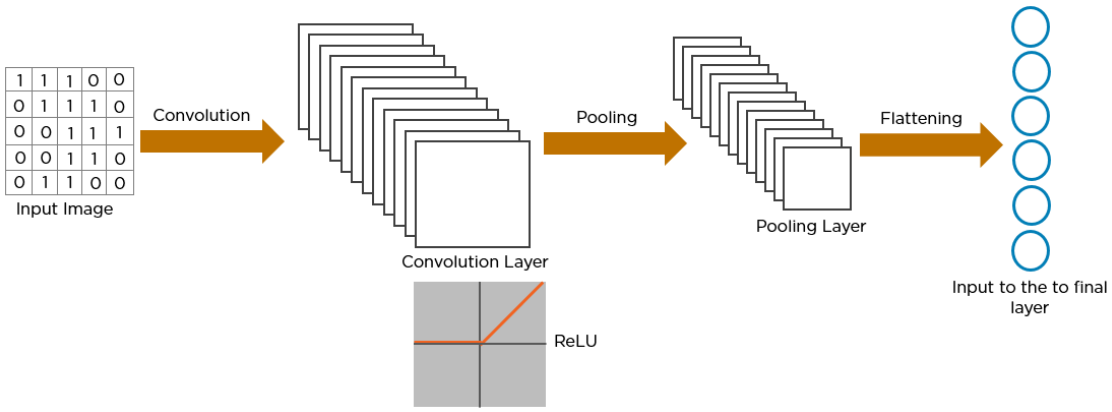


Fig 5.10 Input Image to Final Layer

Fully Connected layer :

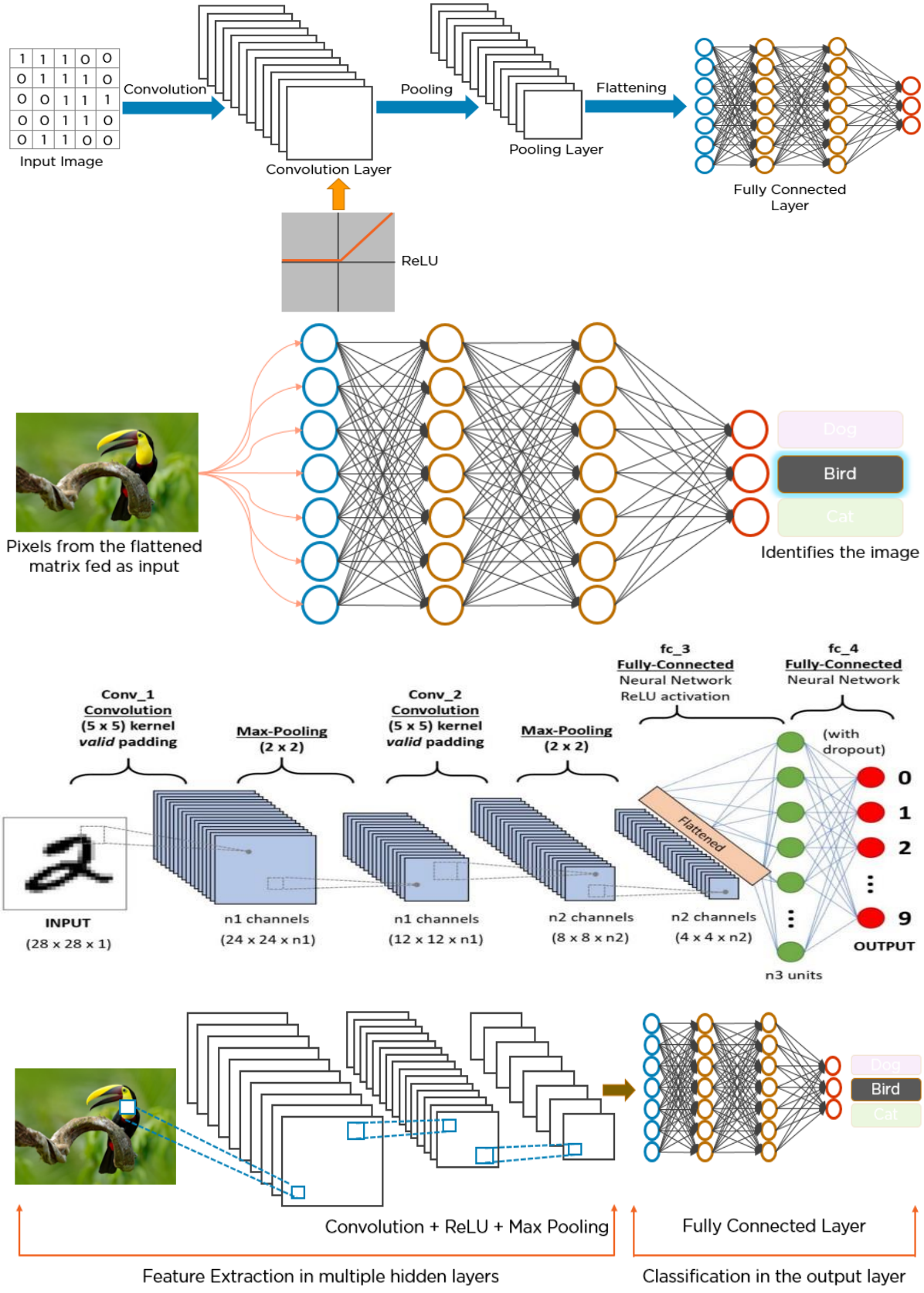


Fig 5.11 Input Layer to Fully Connected Layer

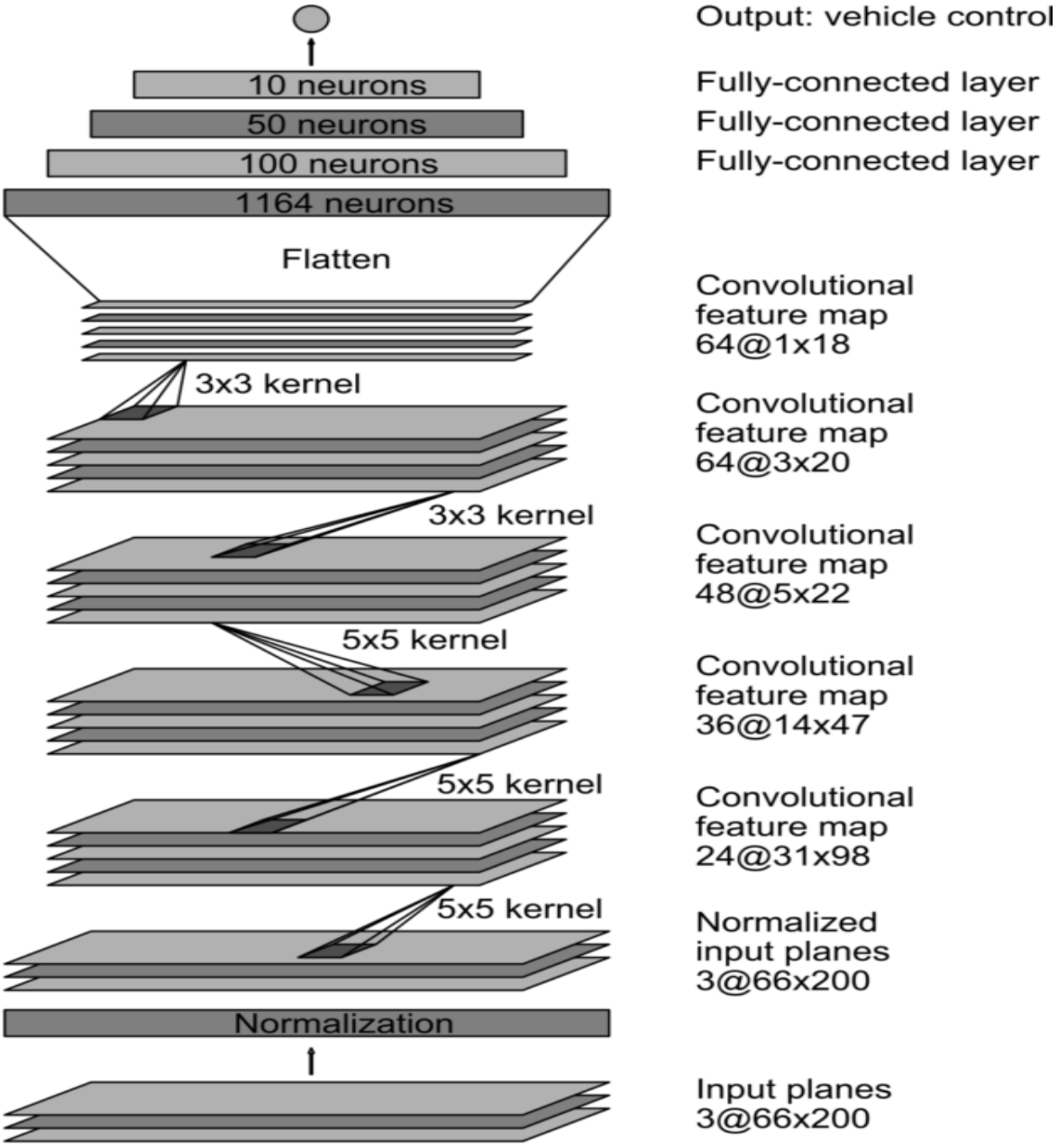


Fig 5.12 : CNN Architecture.

The network has about 27 million connections and 250 thousand parameters

5.3. STEERING ADJUSTMENT

Applying different types of smoothing to the steering angles output from model improved the RMS error. In the end exponential smoothing (Brown's Method) with a smoothing factor (alpha=0.4) is used. This method only looks at past steering angles rather than future steering angles. Using future steering angles is realistic for actual driving as you create a lag in the steering that is relative to how far in the future you look. Even the exponential

smoothing creates a small time lag to the steering. Looking at the steering angles with and without smoothing, we can estimate the time lag due to smoothing to be 1 timestamp or 1/20 of a second. While the lag is an undesired outcome, the positive outcome is a much smoother driving experience.

Steering Angle model

Follow the instructions to train a deep neural network for self-steering cars.

Download dataset

./get_data.sh

Start training data server in the first terminal session

./server.py --batch 200 --port 5557

Start validation data server in a second terminal session

./server.py --batch 200 --validation --port 5556

Train steering model in a third terminal

./train_steering_model.py --port 5557 --val_port 5556

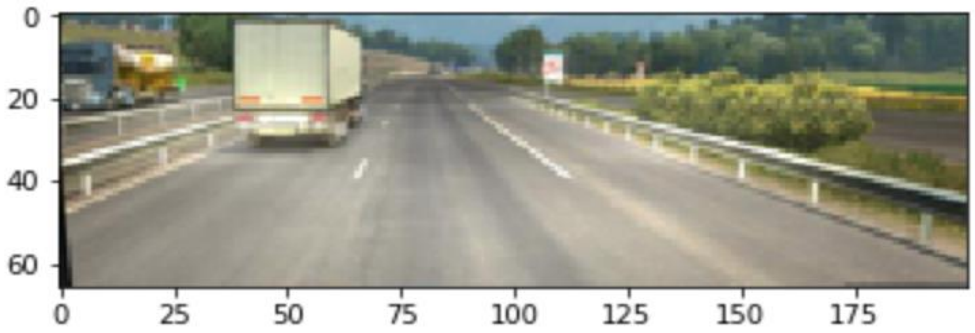
Visualize results

./view_steering_model.py ./outputs/steering_model/steering_angle.json

5.4. PREDICTION

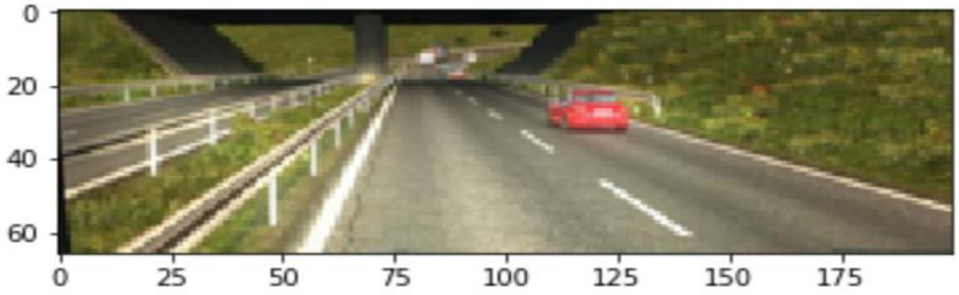
Here are some results (to remind — negative implies left turn and vice-versa)

```
ground_truth= 128 , prediction= 10.327421759348363
```

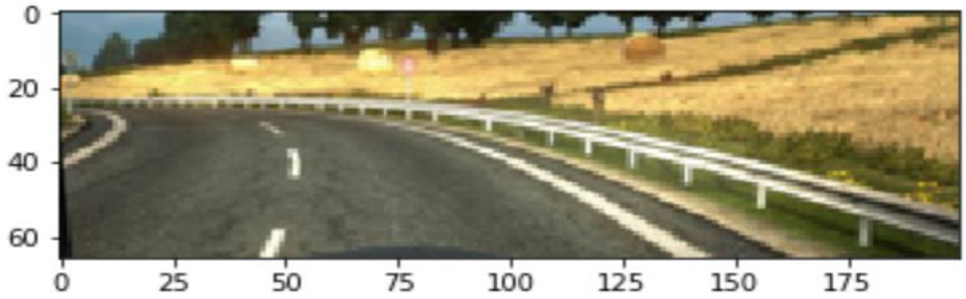


ground_truth= 512 , prediction= 961.4895473599435

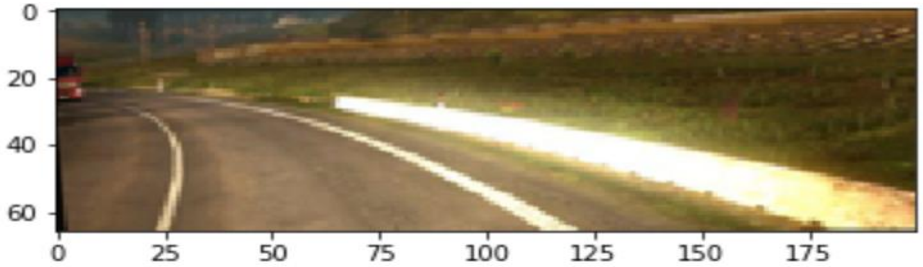
ground_truth= -81 , prediction= -52.85606846809387



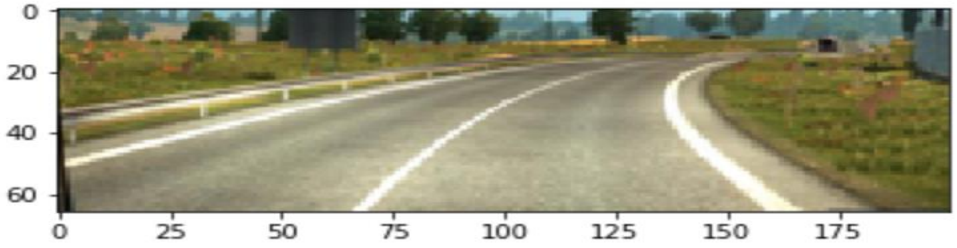
ground_truth= -849 , prediction= -801.5645657837391



ground_truth= -2097 , prediction= -2043.342570590973

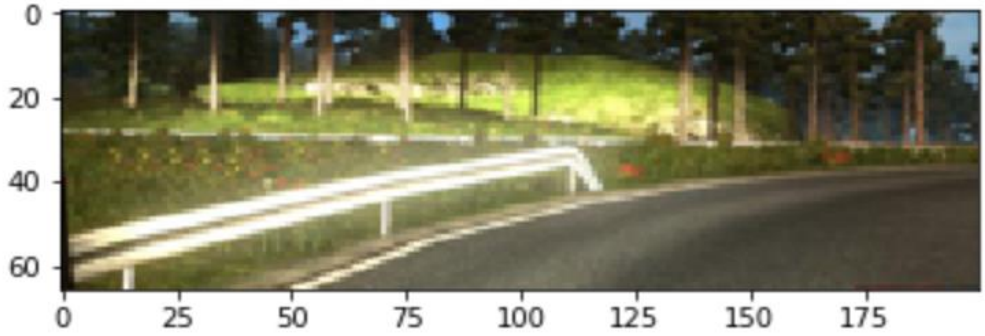


ground_truth= 2700 , prediction= 2746.8402734041215

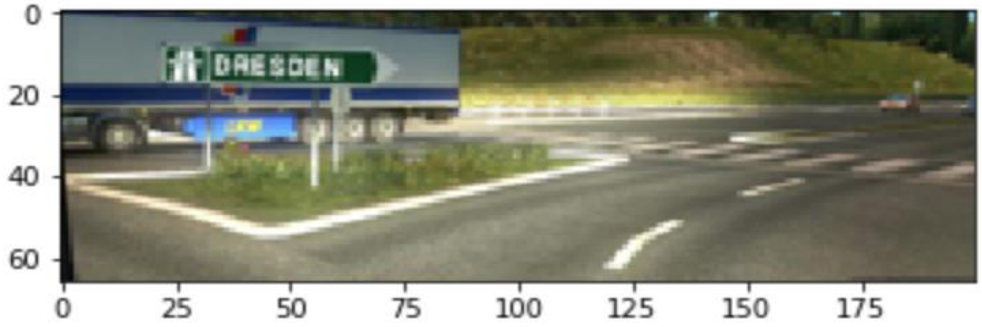


Although, there were very few examples of sharp turns, model has still learnt those patterns.

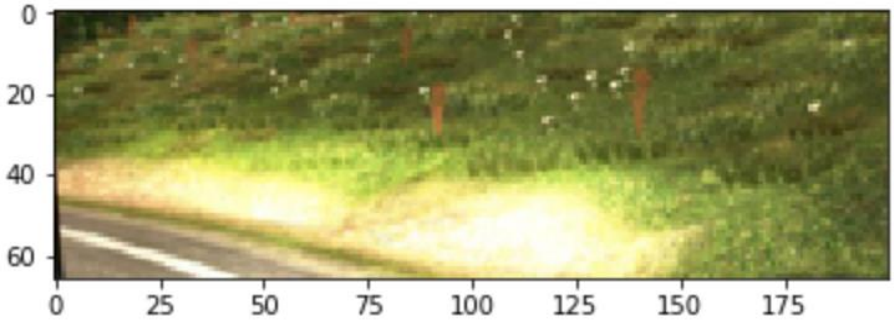
```
ground_truth= 8353 , prediction= 8723.821258020402
```



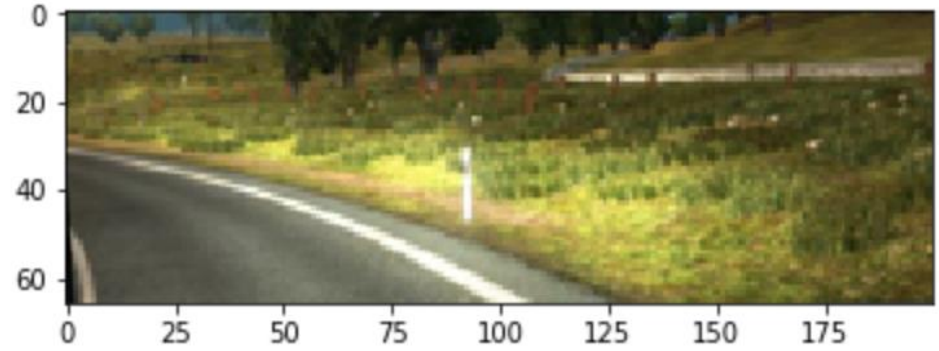
```
ground_truth= 8517 , prediction= 9235.380327129364
```




```
ground_truth= -10818 , prediction= -11630.210034370422
```

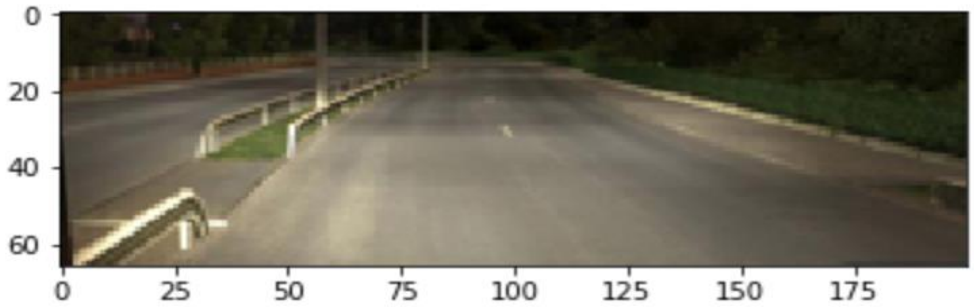


```
ground_truth= -5757 , prediction= -6225.846249675751
```

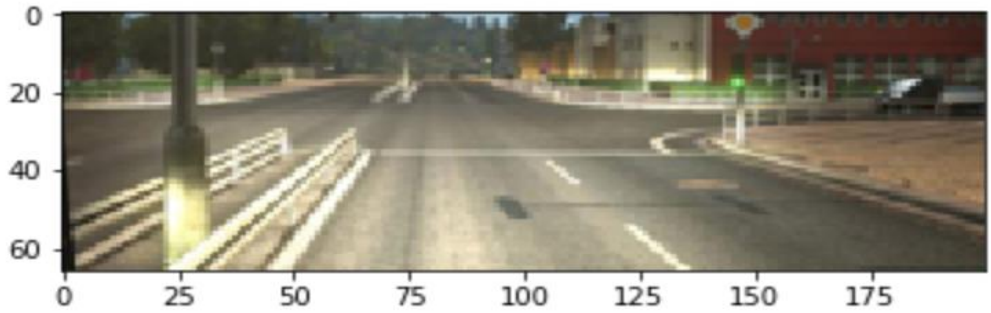


There were cases where model performed better than ground truth labels.

```
ground_truth= -257 , prediction= 49.845069491118196
```




```
ground_truth= 480 , prediction= 45.034957341849804
```



Model predictions are better than ground truth labels

And in some cases model didn't perform very well.

```
ground_truth= 340 , prediction= -34.68989791218191
```


```
ground_truth= 0 , prediction= -127.03235848546029
```

CHAPTER 6
NETWORK ARCHITECTURE

A convolutional neural network is used to training data. Authors train the weights of our network to minimize the mean squared error between the command of output vehicle control and the command of the human driver. The network architecture consists of 9 layers, including 1 normalized layer, 3 convolutional layers, 3 max-pooling layers and 2 fully connected layers. Our network architecture was described in Figure below.

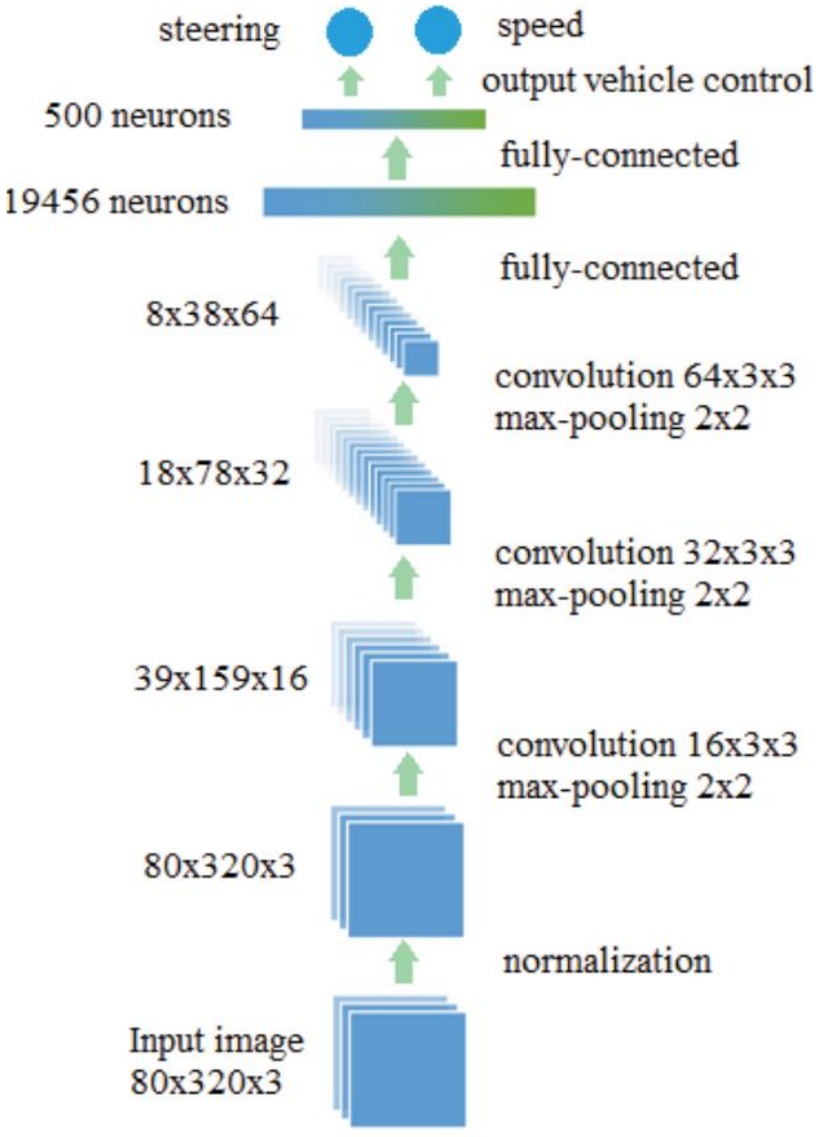


Fig. 6.1. The Network Architecture

The first layer implements normalized all images have value pixels from -1 to 1. The convolutional layers were designed by experiment, we use kernels have size 5x5 with non-stride for the first layer, the others are 3x3 also non-stride. The respective depth of each layer is 16, 32 and 64.

The max-pooling layers were interleaved with the convolutional layers. Max-pooling layer is one of the powerful tools that usually used by CNN. This is a method that resizes large images but keeps the most important information about them. It involves sliding a small windowpane along one image and getting the maximum value from the window at each step. After pooling, an image will have about a quarter of pixels compared to the beginning. This reduces the number of hyperparameters needing to calculate, thence decreasing the computational time and avoiding the overfitting. All max-pooling layers were chosen with kernel was 2x2 and non-stride.

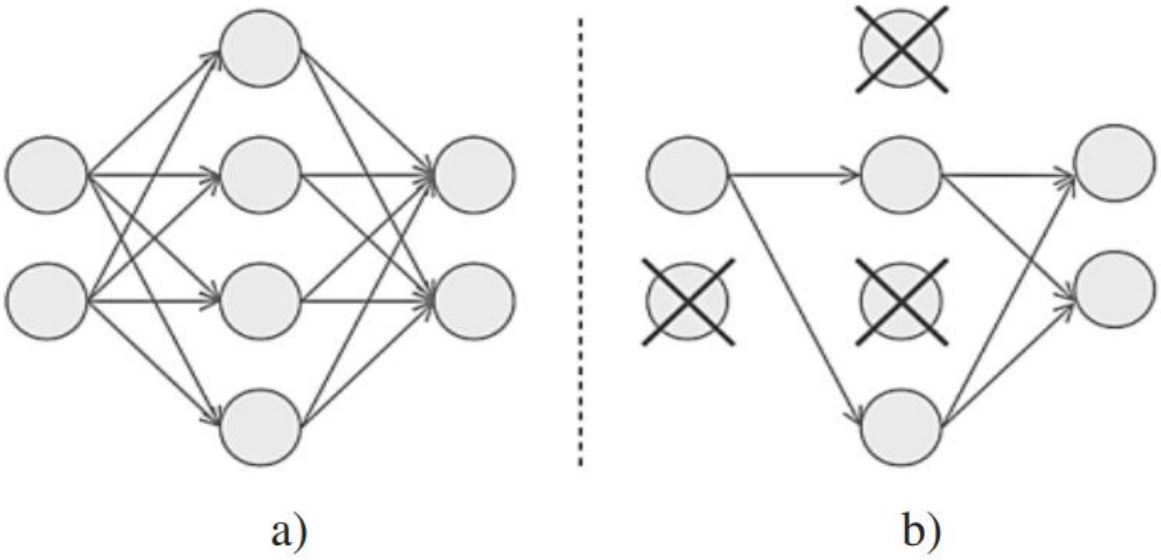


Fig. 6.2. Dropout Neural Net Model. a) The Standard Neural Network.
b) The Neural Network After Applying Dropout Technique

The fully connected layers were designed with gradually reducing sizes: 19456 and 500. The output layer is 2 because our model predicts two values, one is the steering angle and another is speed.

Besides, the layers activation were ELUs (exponential linear units) following each convolutional layers to improve convergence are also used. Function ELUs try to make mean

activations closer to zero which speeds up learning. It has been shown that ELUs can obtain higher classification accuracy than ReLUs. The output of a ReLU class is the same size as the input, the difference is that all negative values of the image will be removed shortly afterward.

$$\overline{f(x)} = \begin{cases} x; x \geq 0 \\ a(e^{x} - 1); x < 0 \end{cases}$$

With a was a hyper-parameter to b e and a >= 0 was a constraint.

For this project, we used the mean-square-loss function. This function is common for regression problem that is simply the mean of the sum of the square difference between the actual and predicted results.

$$MSE = \frac{1}{n}\sum\left(y_i - \hat{y}_i \right)^2$$

To optimize this loss, the Adam optimizer was used. This optimizer is usually chosen for deep learning application. We used the default parameters of Adam provide in Keras.

(Learning rate of 0.001, β1=0.9 , β2=0.999 , ε=le-8 and decay = 0).

Fig. 6.3: The Input Data

Fig. 6.4. The Augment Data Original Image

Fig. 6.5. The Augment Data Flip Image

Convolutional neural networks contain multiple non-linear hidden layers, this makes them become complication models and lead to complex relationships between their inputs and outputs. To make the architecture more robust and to prevent overfitting, dropout layers added to the network are implemented. Dropout disables neurons in the network by a given probability and prevents co-adaption of features. For this work, we have applied a dropout rate of 20% (Figure 5.2).

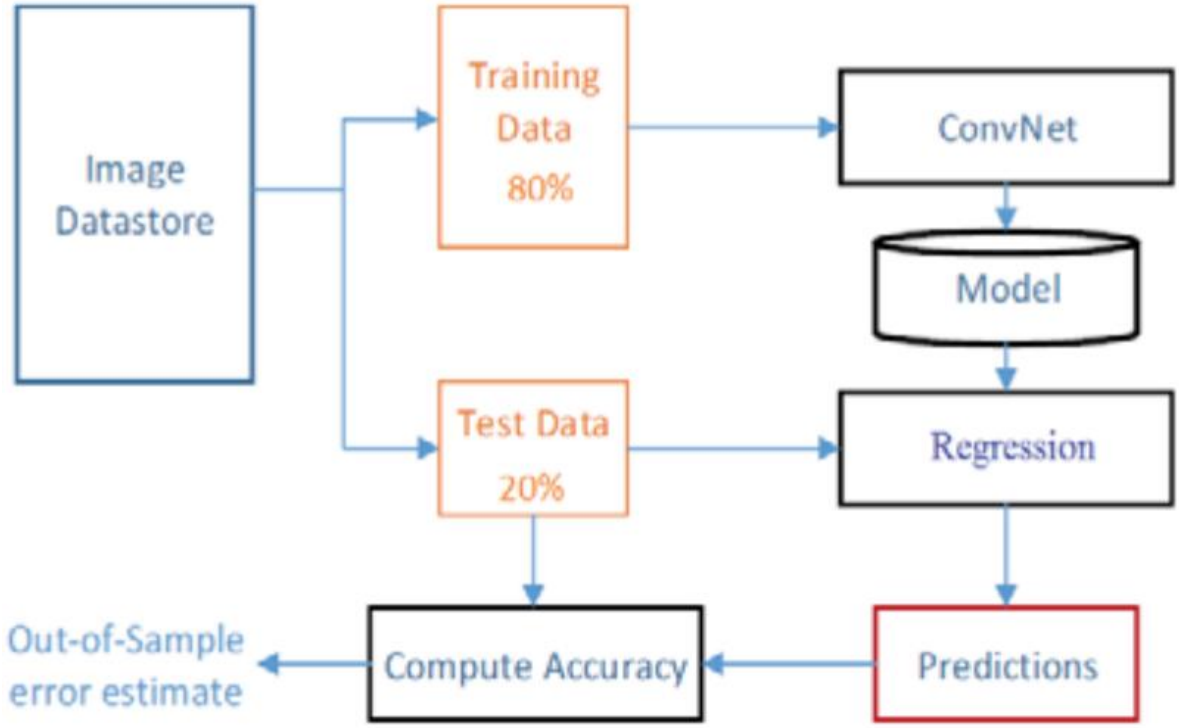


Fig. 6.6. The Block Diagram for Accuracy Evaluation of Proposed Network

CHAPTER 7
EXPERIMENTS

7.1. TRAINING PROCESS, DATA AND FEATURE

Udacity provided the basic simulation framework for navigating the autonomous vehicle. The training data were collected by driving multiple turns on the road. In the training mode, the simulator records steering, speed and images with size is 160x320 from three cameras (Figure 5.3). In addition to augmenting more data to improve accuracy by flipping for all the images previously captured is carried out (Figures 5.4&5.5). Therefore, the image size will be cropped to 80x320. The total samples are approximately 15500 images.

From the stored dataset, we divide them into two parts separately. One is the training data and another is test data with proportion 80:20. A training diagram was shown in Figure 5.6 We take advantage of GPU NVIDIA GeForce 2060 with memory was 6GB available to train the network with three times, each turn was 30, 50, 100 epochs. Total time that we implement training in conjunction with 100 epochs for 15500 samples was approximately 280 minutes. Therefore, the average one epoch takes 168 seconds to complete. Some images represent the output feature convolutional layers were shown in figure 6.2.

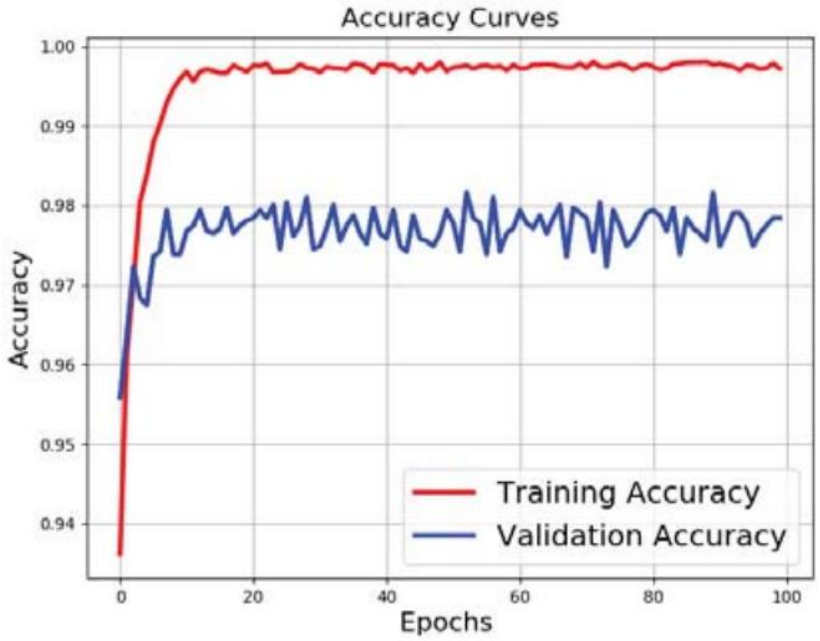


Fig. 7.1. The Accuracy of the Proposed Deep Neural Network Architecture

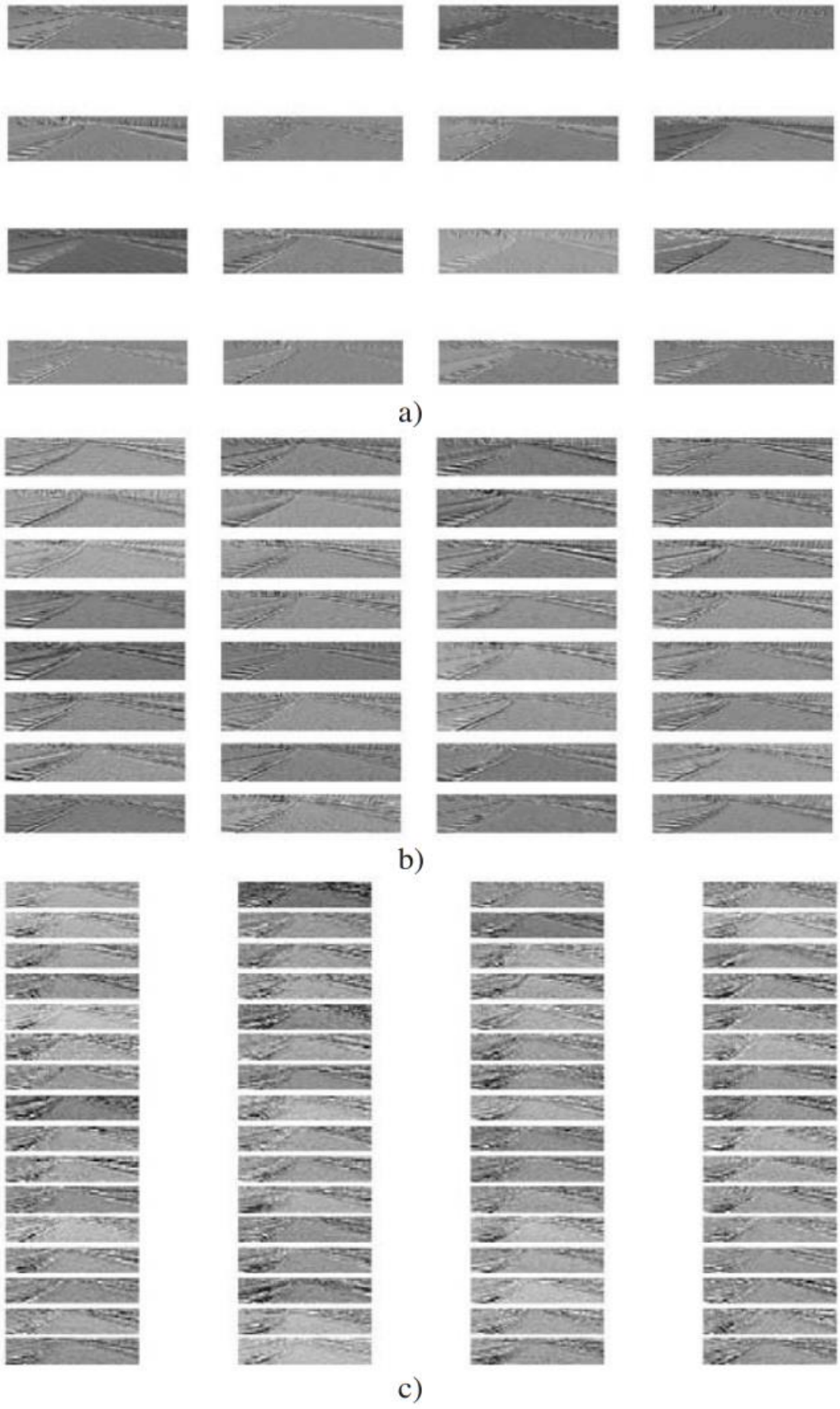


Fig. 7.2. The Visualization Output of Three Convolutional Layers.
a) Convolutional Layer 1, b) Convolutional Layer 2, c) Convolutional Layer 3

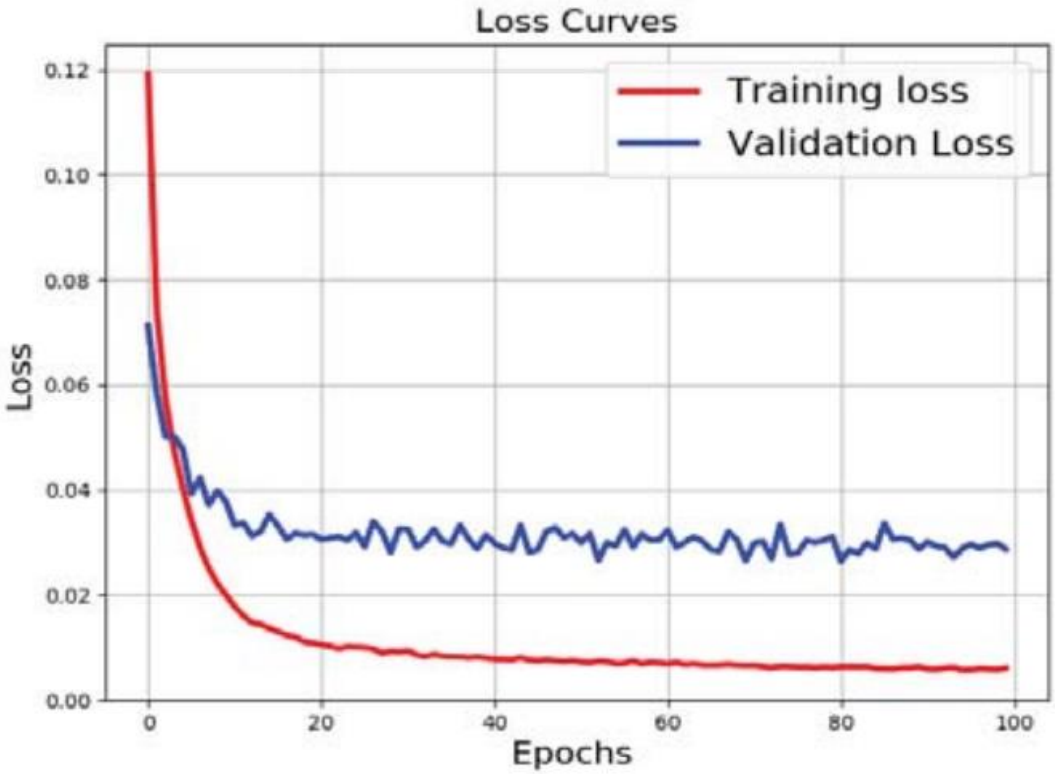


Fig. 7.3. The Loss of the Proposed Deep Neural Network Architecture

7.2. Autonomous vehicle navigation

We ended up with one virtual car that was able to drive autonomously on the road at simulation of UDACITY through a network had been trained in advance. The model predicted steering angle and speed accomplish convincing accuracy above 97% (Figure 6.1). The result of this experiment on the simulation software is vehicle can drive automatically and follow the lane relatively stable and do not get out of the lane. Value steering and speed are predicted relatively accurately (Figure 6.4).

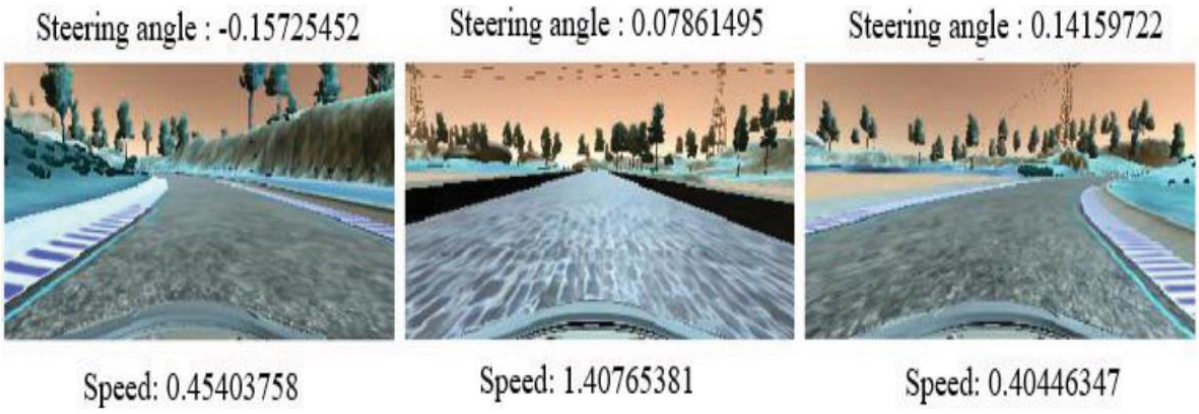


Fig. 7.4. The Result Predicted Steering Angle and Speed of the Car After Training

CHAPTER 8

TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of tests. Each test type addresses a specific testing requirement.

Autonomous vehicles are safety-critical systems that require a high level of dependability so software and hardware testing for autonomous vehicles is the only sane way of evaluating them on the safety and practicality parameters. Testing autonomous vehicle systems bestows a degree of trust and certainty regarding the capabilities of the self-driving car. It analyzes and assures that all the systems involved in the decision-making process are working in complete tandem with one another without any anomalies.

Testing is centered on the following:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

VARIOUS TESTING METHODOLOGIES OF AN AUTONOMOUS VEHICLE

SYSTEM TESTING

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

PRE-EMPTIVE SOFTWARE TESTING

Pre-emptive software testing for self-driving cars identifies any bugs present in the software before the car gets on the road. As all the sensors, radars, and cameras depend on super-fast connectivity, IoT testing of these units also become essential for assessing if all the necessary information is communicated smoothly or not.

TESTING THE AI SYSTEMS

Testing the AI systems in the autonomous vehicles ensures that the data fed to the system is understood properly and the predictions offered are viable and accurate. Therefore not the software, but software testing is the real driver of the driverless cars.

SIMULATION TESTING

High-fidelity simulation is required in autonomous vehicle testing. The dedicated software containing mathematical representation of the subsystems should be used in order to achieve realistic system dynamic, which can be validated with hardware-in-the-loop techniques. High level algorithms for trajectory planning, vision based processing, and multi-vehicles interactions are examples of suitable fields based on game engines.

AUTONOMOUS VEHICLE DESIGN AND SYSTEM TESTING

The autonomous vehicle development starts with a definition of the functional requirements in terms of the desired functions, from the basic autonomous driving functional requirements to handle short term and long term planning, avoid dangerous collision and driving safety obey the traffic rules, and with further constraints or requirements on safety, mobility, passenger comfort, and intelligent operational.

Autonomous vehicles are safety-critical systems that require a high level of dependability, a term covering reliability, fail-safety, and fault-tolerance. In addition to the system safety, it requires driving safety and vehicle safety, so regularly testing the vehicle and testing all software and hardware components regularly is the key.

PERCEPTION LAYER FUNCIONAL TESTING

The Perception Layer is responsible for the acquisition of all data, from vision, Lidar, or radar based sensors. Then they are merged into a unique fusion map. By physical tests, software test or HIL simulation test, both the various sensors and environment perception layer are tested. The assessment criteria are obtained, including the state and errors of the posture and localization, the detected pedestrians, lanes, traffic signs and lights, other vehicle and other related elements.

DECISION LAYER FUNCTIONAL TESTING

The Decision Layer is fed by the Perception Layer providing feedback data to further optimize the data acquisition and interprets all incoming data from it to generate a reasonable output to the Action Layer. The Situational Assessment provides the input evaluation for short and long term planners; they should influence each other to avoid short term decisions which do not accomplish the overall goal. Artificial Intelligence algorithms are commonly used in the Decision Layer mainly due to the highly non-linear behavior of real environment such as Neural Networks, Machine Learning, etc.

This comprehends the middle level supervision from simple tasks such as follow a line or a path and speed control to more complex tasks such as adjusting speed anticipating a curve or collision avoidance. Evaluation of autonomous vehicle decision making modules is done by way of test drive or simulation test. The driving system reaction characteristic are used for indicators, including reaction time and operating correctness etc.

NAVIGATION LAYER FUNCTIONAL TESTING

The Navigation Layer functions testing are done by test drive or simulation. The navigation level performs higher level tasks related to driving such as controlling the global objectives, trajectory planning, efficiency and commodity, taking into account the driving conditions. The Path planning error is used for assessment criteria; evaluate the capability of the algorithms to avoid collisions with other objects, at any time.

Avoidance of any type of collisions in the simulator is important during the initial stages of the development of an autonomous vehicle.

UNIT TESTING

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

Test strategy and approach

Field testing will be performed manually and functional tests will be written in detail.

Test objectives

- All field entries must work properly.
- Pages must be activated from the identified link.
- The entry screen, messages and responses must not be delayed.

Features to be tested

- Verify that the entries are of the correct format
- No duplicate entries should be allowed
- All links should take the user to the correct page.

INTEGRATION TESTING

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

WHITE BOX TESTING

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

BLACK BOC TESTING

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

ACCEPTANCE TESTING

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

TEST RESULTS:

All the test cases mentioned above passed successfully. No defects encountered.

CHAPTER 9

SOURCE CODE

9.1. DriveApp

```
import numpy as np
import cv2
from keras.models import load_model

model = load_model('Autopilot.h5')

def keras_predict(model, image):
    processed = keras_process_image(image)
    steering_angle = float(model.predict(processed, batch_size=1))
    steering_angle = steering_angle * 100
    return steering_angle

def keras_process_image(img):
    image_x = 40
    image_y = 40
    img = cv2.resize(img, (image_x, image_y))
    img = np.array(img, dtype=np.float32)
    img = np.reshape(img, (-1, image_x, image_y, 1))
    return img

steer = cv2.imread('steering_wheel_image.jpg', 0)
rows, cols = steer.shape
smoothed_angle = 0

cap = cv2.VideoCapture('run.mp4')
while (cap.isOpened()):
    ret, frame = cap.read()
    gray = cv2.resize((cv2.cvtColor(frame, cv2.COLOR_RGB2HSV))[:, :, 1], (40, 40))
    steering_angle = keras_predict(model, gray)
    print(steering_angle)
    cv2.imshow('frame', cv2.resize(frame, (500, 300), interpolation=cv2.INTER_AREA))
    smoothed_angle += 0.2 * pow(abs((steering_angle - smoothed_angle)), 2.0 / 3.0) * (
        steering_angle - smoothed_angle) / abs(
        steering_angle - smoothed_angle)
    M = cv2.getRotationMatrix2D((cols / 2, rows / 2), -smoothed_angle, 1)
    dst = cv2.warpAffine(steer, M, (cols, rows))
    cv2.imshow("steering wheel", dst)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

9.2. LoadData

```
import cv2
import numpy as np
import csv
import pickle
import matplotlib.pyplot as plt

features_directory = './data/'
labels_file = './data/driving_log.csv'

def preprocess(img):
    resized = cv2.resize((cv2.cvtColor(img, cv2.COLOR_RGB2HSV))[:, :, 1], (40, 40))
    return resized

def data_loading(delta):
    logs = []
    features = []
    labels = []
    with open(labels_file, 'rt') as f:
        reader = csv.reader(f)
        for line in reader:
            logs.append(line)
        log_labels = logs.pop(0)

    for i in range(len(logs)):
        for j in range(3):
            img_path = logs[i][j]
            img_path = features_directory + 'IMG' + (img_path.split('IMG')[1]).strip()
            img = plt.imread(img_path)
            features.append(preprocess(img))
            if j == 0:
                labels.append(float(logs[i][3]))
            elif j == 1:
                labels.append(float(logs[i][3]) + delta)
            else:
                labels.append(float(logs[i][3]) - delta)
    return features, labels

delta = 0.2
features, labels = data_loading(delta)

features = np.array(features).astype('float32')
labels = np.array(labels).astype('float32')

with open("features_40", "wb") as f:
    pickle.dump(features, f, protocol=4)
with open("labels", "wb") as f:
    pickle.dump(labels, f, protocol=4)
```

9.3. TrainModel

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
from keras.layers import Input, Dense, Activation, Flatten, Conv2D, Lambda
from keras.layers import MaxPooling2D, Dropout
from keras.utils import print_summary
import tensorflow as tf
from keras.models import Sequential
from keras.callbacks import ModelCheckpoint
import pickle
from keras.optimizers import Adam




def keras_model():
    model = Sequential()
    model.add(Lambda(lambda x: x / 127.5 - 1., input_shape=(40, 40, 1)))

    model.add(Conv2D(32, (3, 3), padding='same'))
    model.add(Activation('relu'))
    model.add(MaxPooling2D((2, 2), padding='valid'))

    model.add(Conv2D(64, (3, 3), padding='same'))
    model.add(Activation('relu'))
    model.add(MaxPooling2D((2, 2), padding='valid'))

    model.add(Conv2D(128, (3, 3), padding='same'))
    model.add(Activation('relu'))
    model.add(MaxPooling2D((2, 2), padding='valid'))

    model.add(Flatten())
    model.add(Dropout(0.5))

    model.add(Dense(128))

    model.add(Dense(64))
    model.add(Dense(1))

    model.compile(optimizer=Adam(lr=0.0001), loss="mse")
    filepath = "Autopilot.h5"
    checkpoint1 = ModelCheckpoint(filepath, verbose=1, save_best_only=True)
    callbacks_list = [checkpoint1]


    return model, callbacks_list
```

```
def loadFromPickle():
    with open("features_40", "rb") as f:
        features = np.array(pickle.load(f))
    with open("labels", "rb") as f:
        labels = np.array(pickle.load(f))

    return features, labels



def augmentData(features, labels):
    features = np.append(features, features[:, :, ::-1], axis=0)
    labels = np.append(labels, -labels, axis=0)
    return features, labels


def main():
    features, labels = loadFromPickle()
    features, labels = augmentData(features, labels)
    features, labels = shuffle(features, labels)
    train_x, test_x, train_y, test_y = train_test_split(features, labels, random_state=0,
                                                        test_size=0.1)
    train_x = train_x.reshape(train_x.shape[0], 40, 40, 1)
    test_x = test_x.reshape(test_x.shape[0], 40, 40, 1)
    model, callbacks_list = keras_model()
    model.fit(train_x, train_y, validation_data=(test_x, test_y), epochs=5, batch_size=64,
              callbacks=callbacks_list)
    print_summary(model)
    model.save('Autopilot.h5')


main()
```

CHAPTER 10

OUTPUT SCREEN

10.1. LEFT SIDE STEER

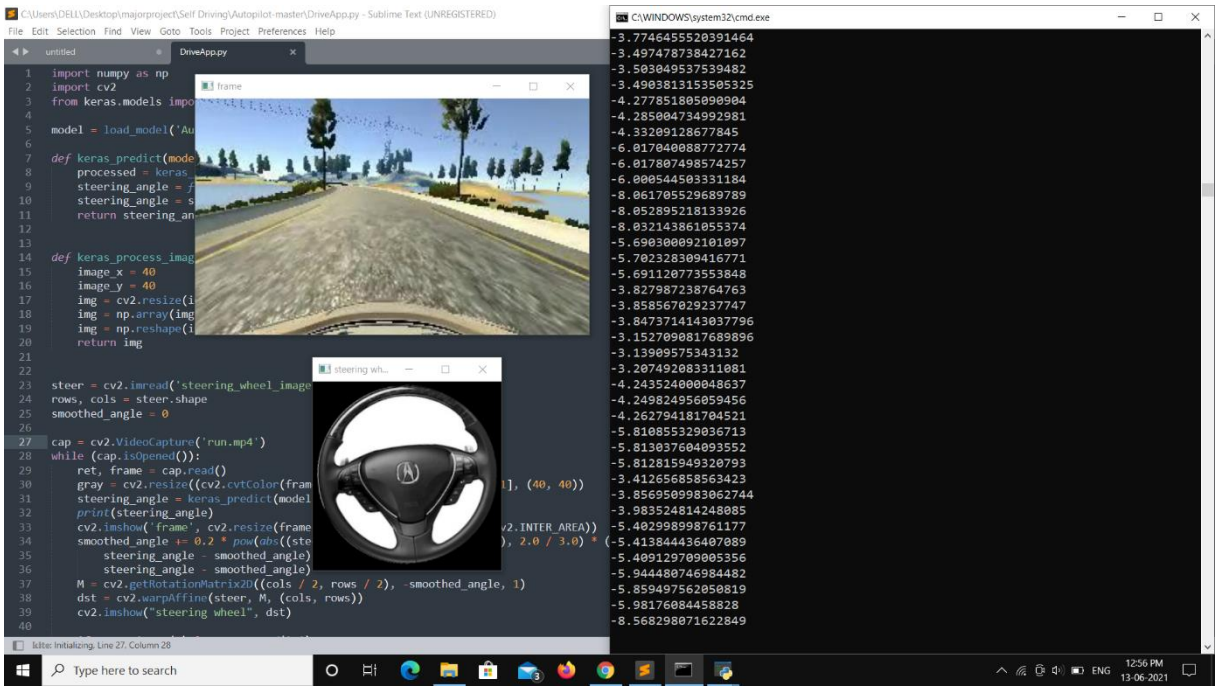


Fig 10.1 Left Steer (1)

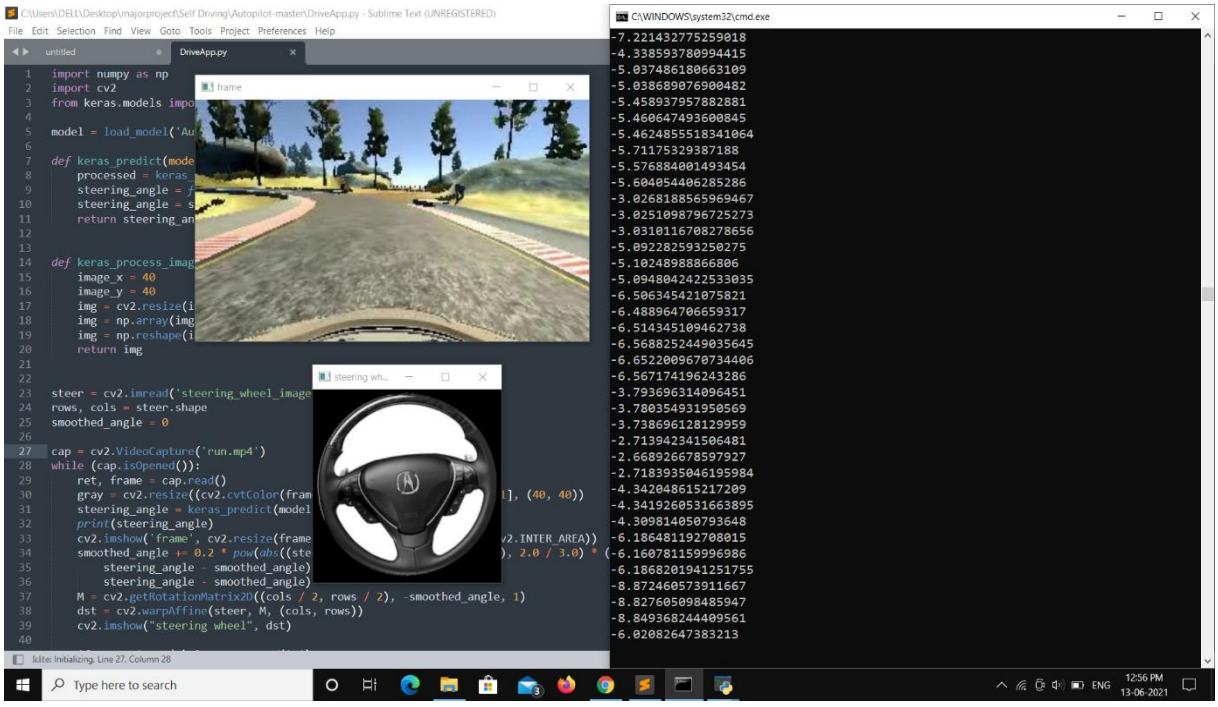


Fig 10.2 Left Steer (2)

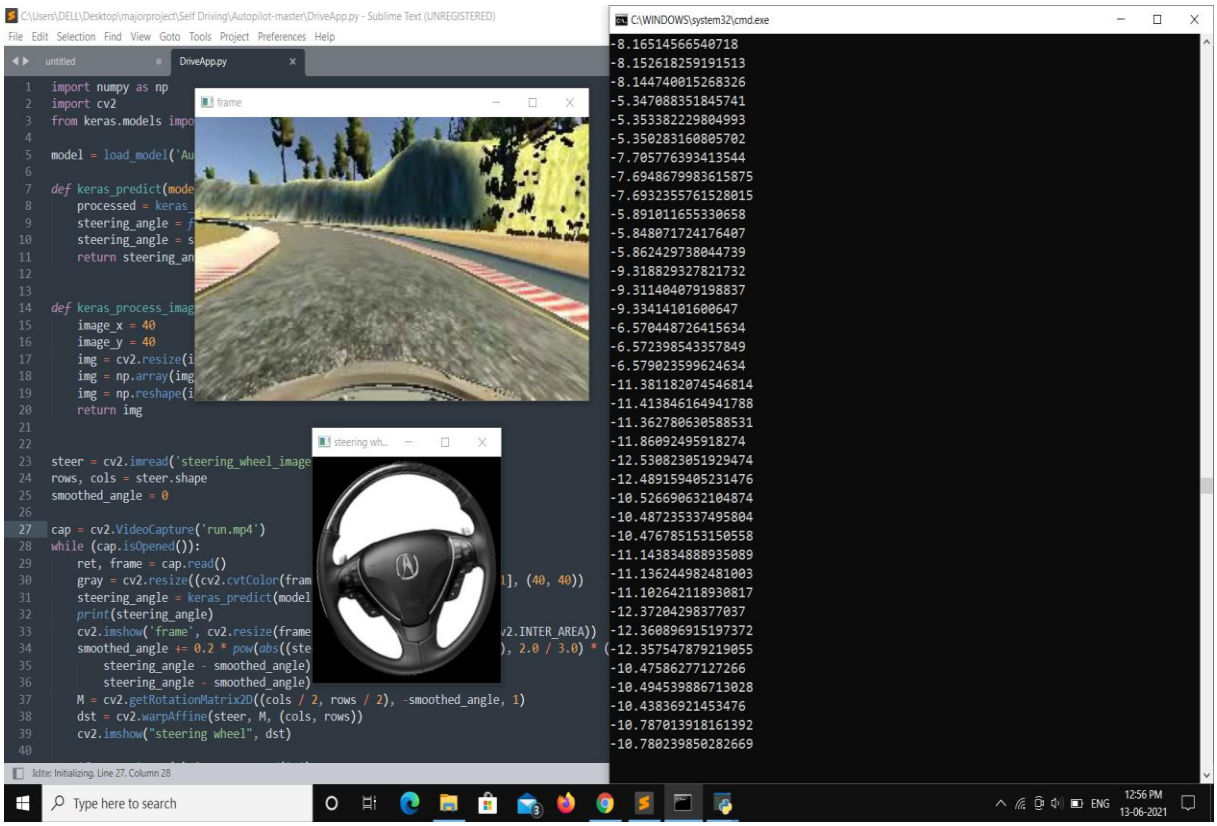


Fig 10.3 Left Steer (3)

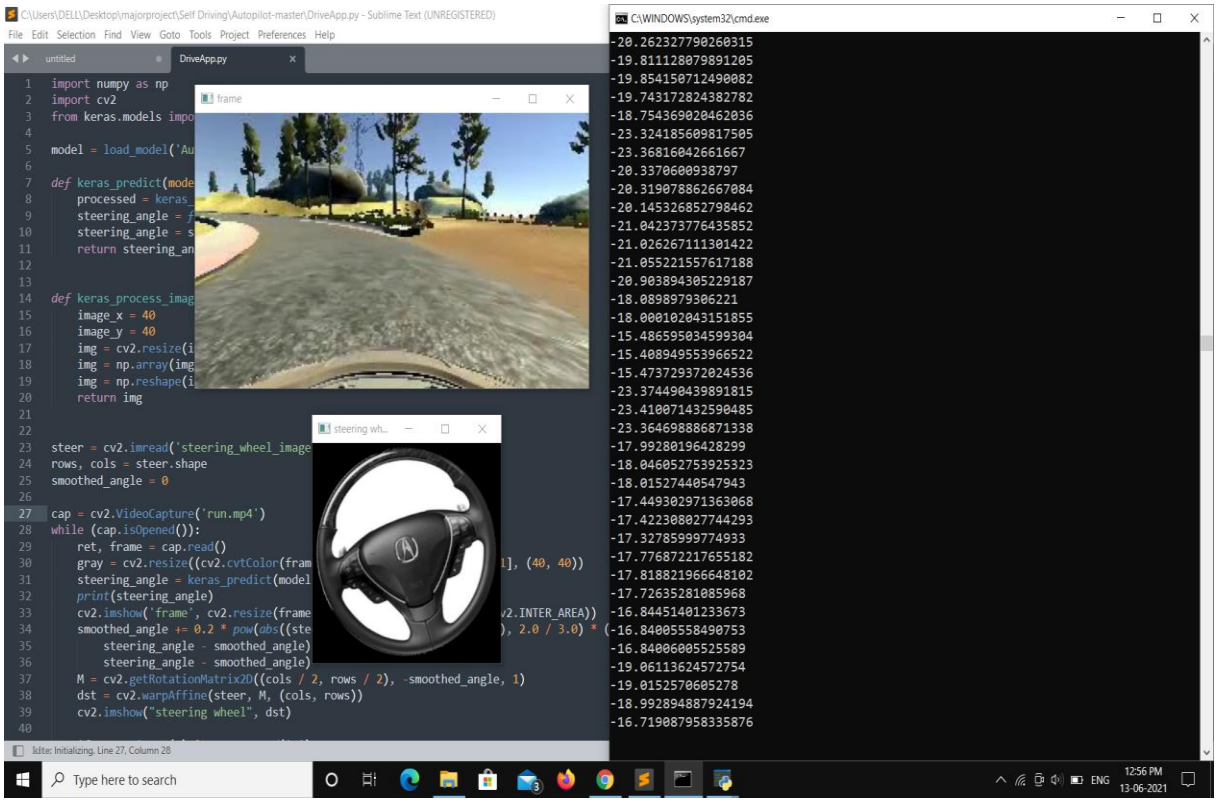


Fig 10.4 Left Steer (4)

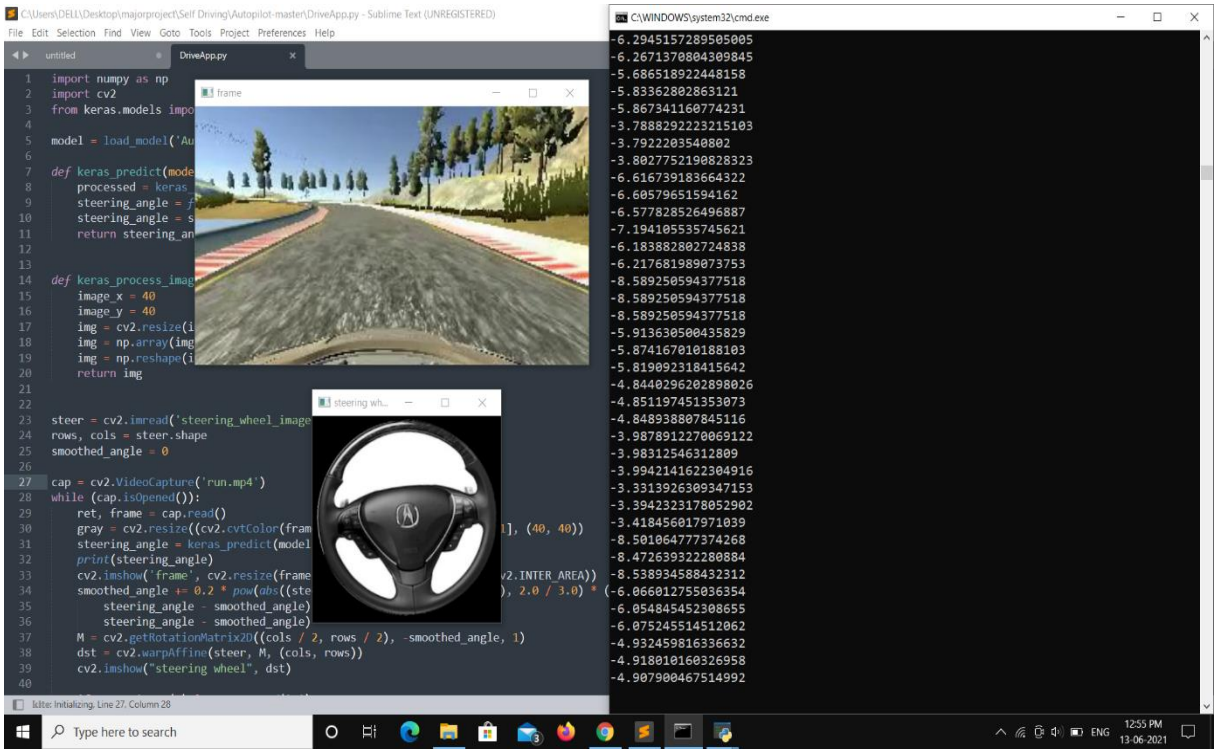


Fig 10.5 Left Steer (5)

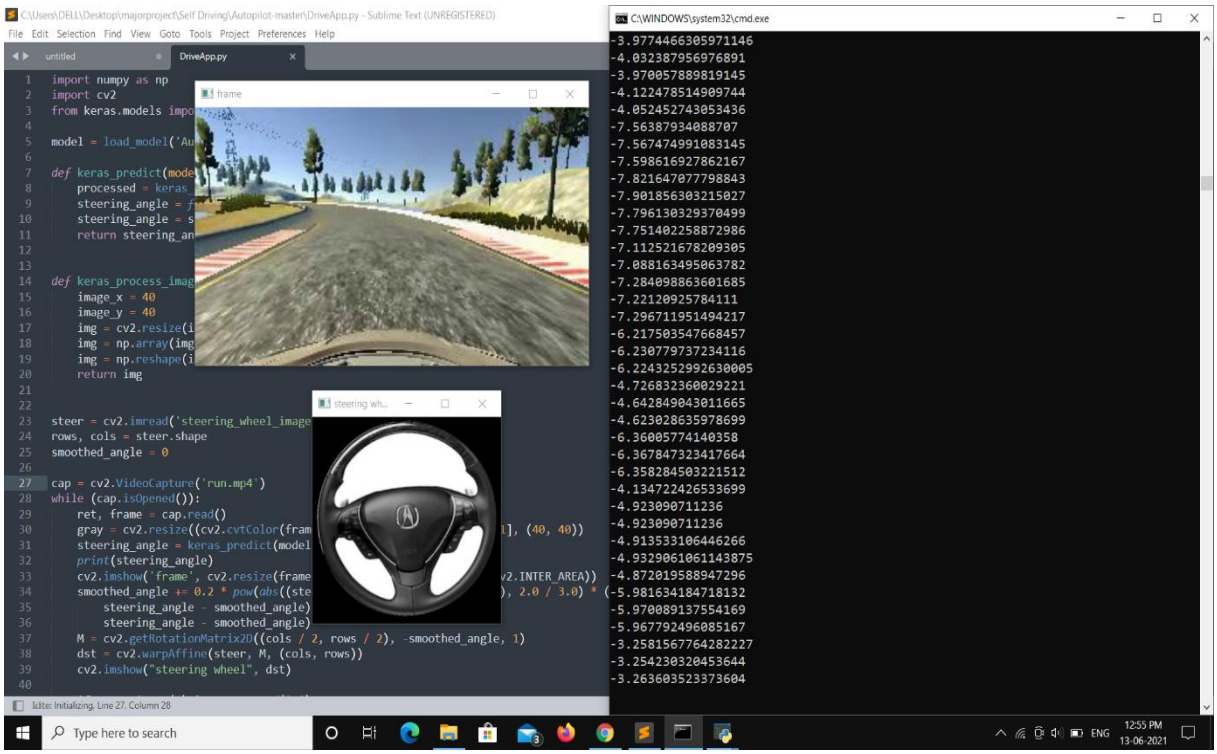


Fig 10.6 Left Steer (6)

10.2. CENTER STEER

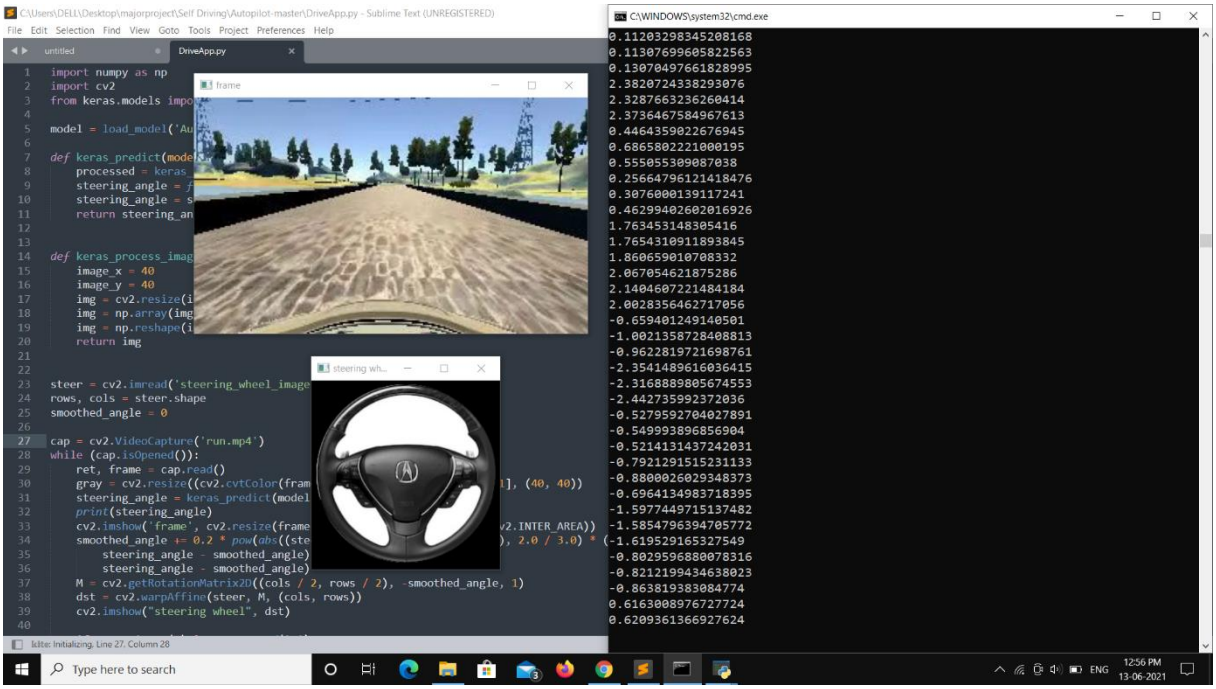


Fig 10.7 Center Steer (1)

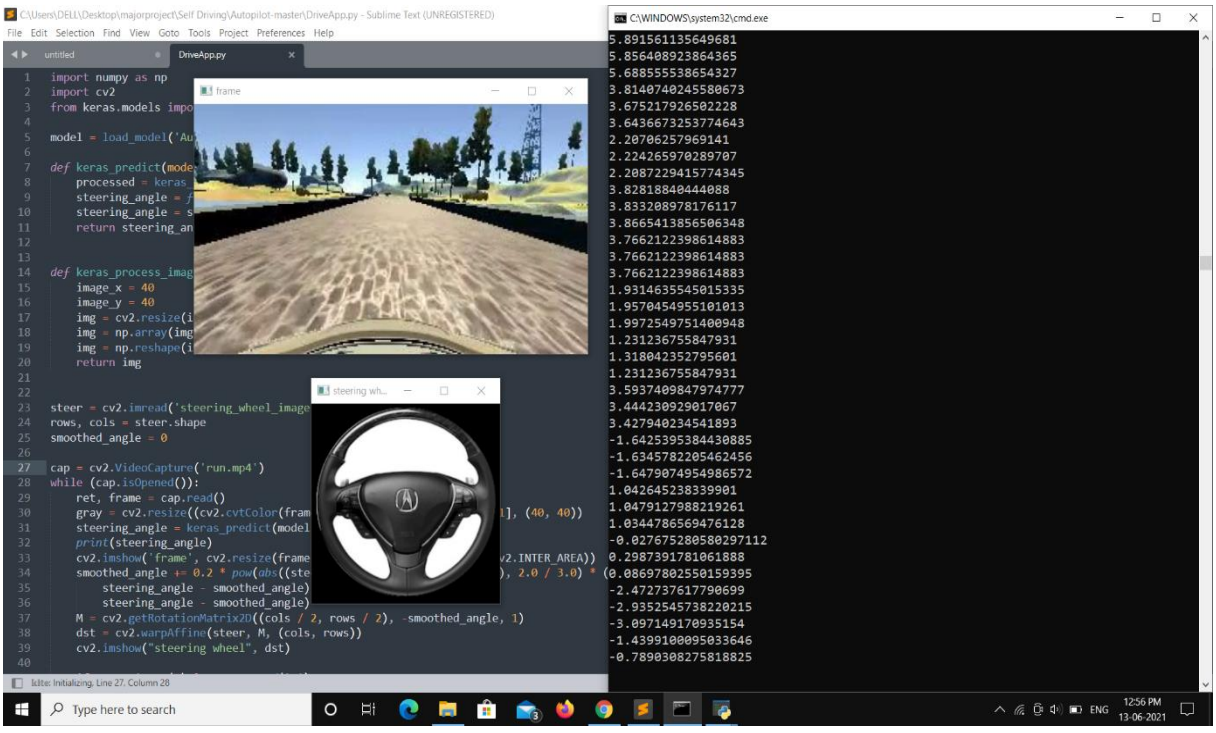


Fig 10.8 Center Steer (2)

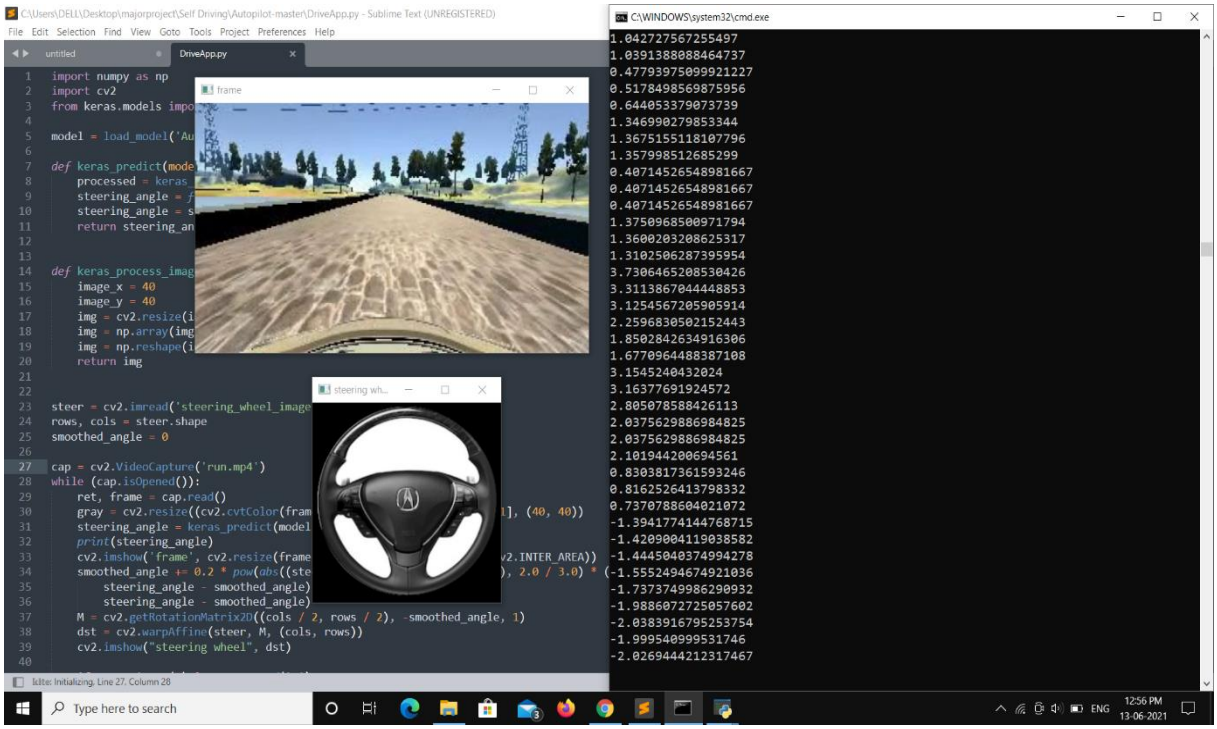


Fig 10.9 Center Steer (3)

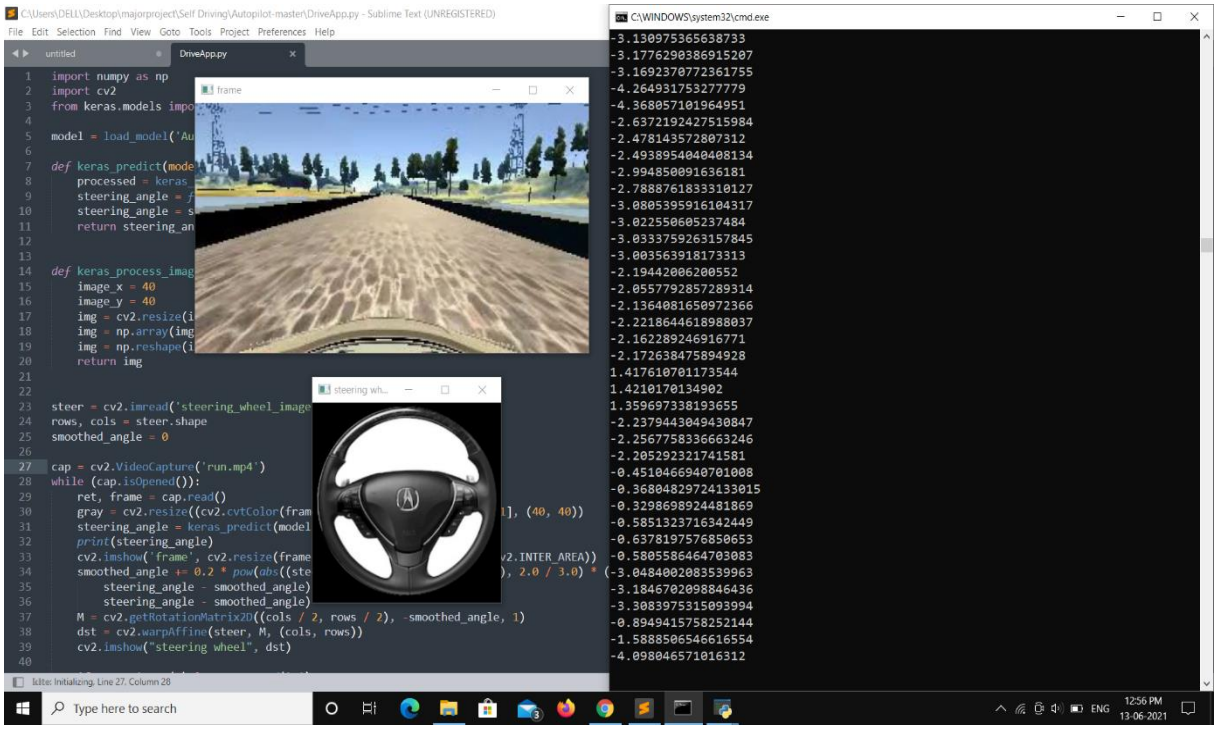


Fig 10.10 Center Steer (4)

10.3.RIGHT SIDE STEER

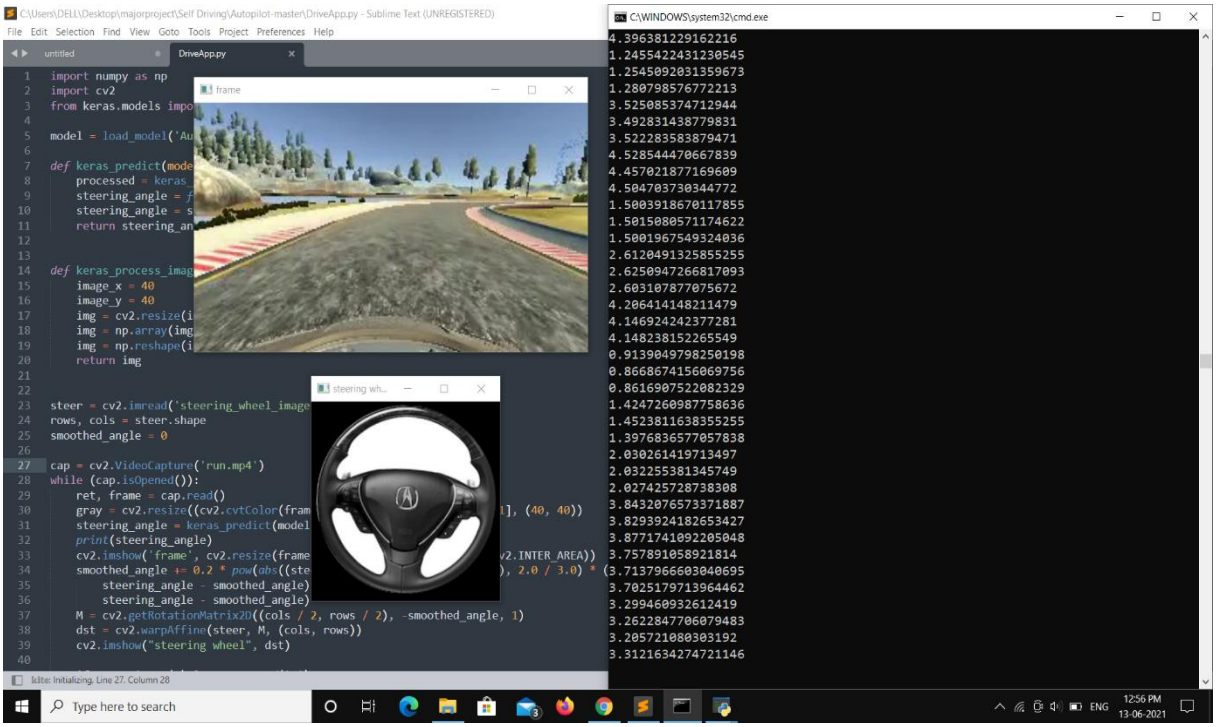


Fig 10.11 Right Steer (1)

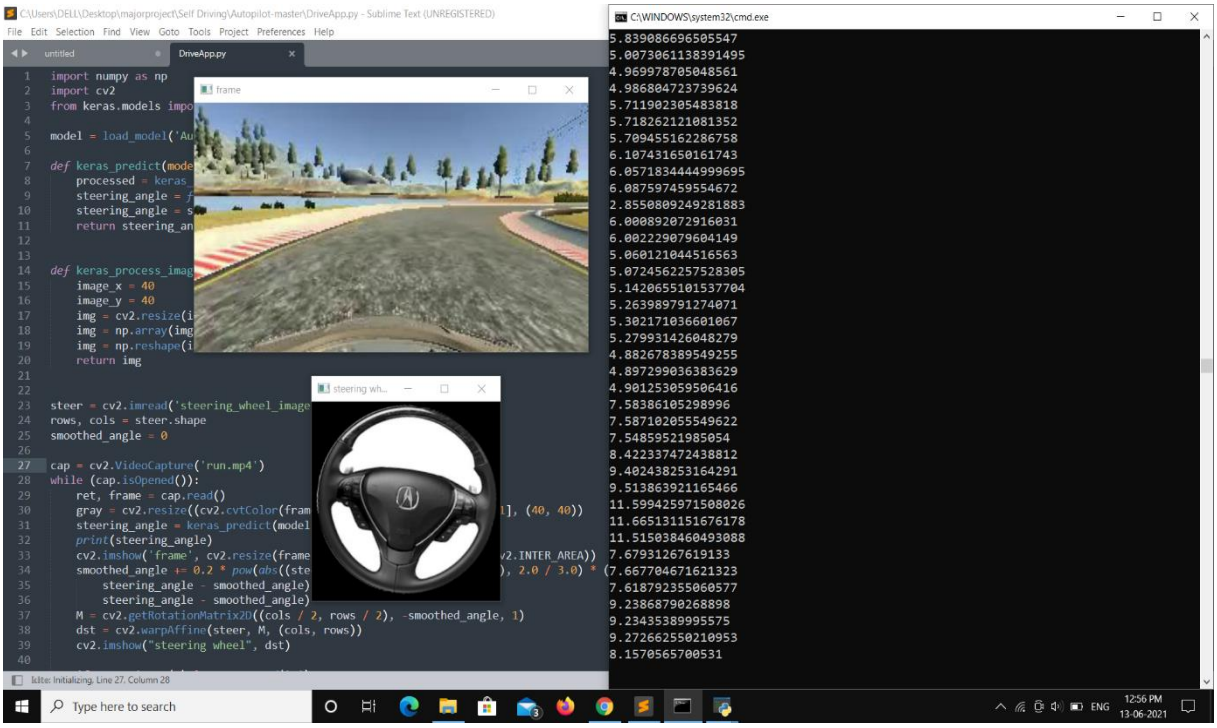


Fig 10.12 Right Steer (2)

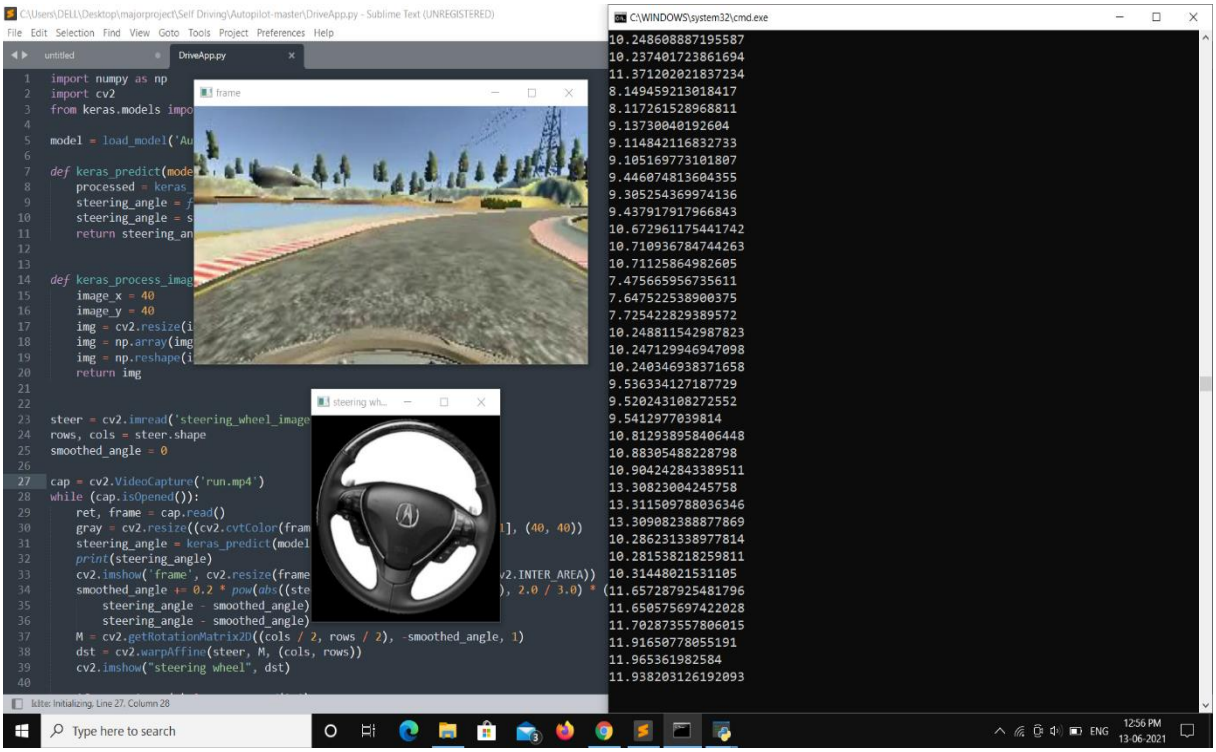


Fig 10.13 Right Steer (3)

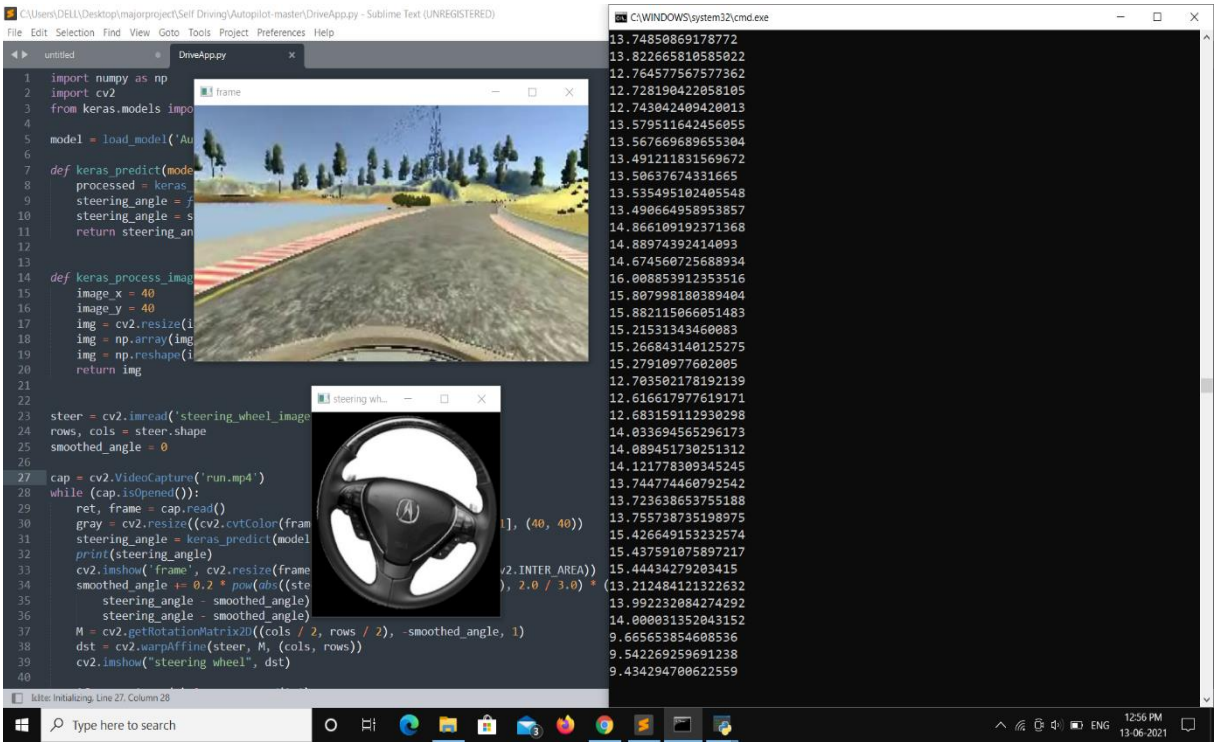


Fig 10.14 Right Steer (4)

CONCLUSION

Data is one of the most important matter led to the accuracy of our model (Table 11.1). Collecting data through three cameras led to the calibration of the steering angle and ensured the vehicle to always run in the center of the lane, which is key to enhance the accuracy.

Moreover, increasing the epoch so that the model approaches the convergence position is a way to have a model with the persuading result (Table 11.2).

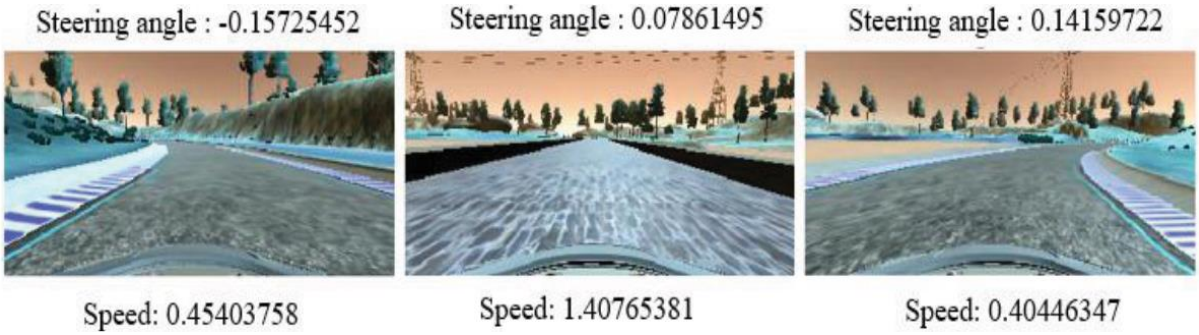


Fig. 11.1. The Result Predicted Steering Angle and Speed of the Car After Training

NUMBER	NUMBER OF SAMPLES	EPOCH	TIME OF TRAINING	ACCURACY
1	10000	100	173 Minutes	96.78%
2	12500	100	212 Minutes	97.15%
3	15000	100	280 Minutes	98.23%

TABLE 11.1. THE EFFECT OF THE NUMBER OF SAMPLES TO THE ACCURACY OF THE MODEL

NUMBER	NUMBER OF SAMPLES	EPOCH	TIME OF TRAINING	ACCURACY
1	15000	30	92 Minutes	96.08%
2	15500	50	155 Minutes	97.77%
3	15000	100	280 Minutes	98.23%

TABLE 11.2. THE EFFECT OF THE EPOCH OF SAMPLES TO THE ACCURACY OF THE MODEL

On the contrary, in this work has several drawbacks. Firstly, because the simulated environment is so ideal hence the noise from the outside environment is almost nonexistent.

The second is the matter about the mechanical error of the vehicle is also ignored. With the restrictions mentioned above, authors will soon experiment real-time autonomous vehicle to prove robust of network architecture. In short, authors have created a model which predicts the steering wheel angles and speed for a vehicle using a convolutional neural network. In spite of having obtained a satisfactory outcome but in the near future we are going to approach the several researches following:

1. Real-time Self-Driving Car navigation using the deep neural network will be the matter that we interested.
2. Experiment with a more complicated landscape.
3. Take into consideration with the model rely on ResNets/VGG through transfer learning method.
4. Consider reinforcement learning as an alternate method to improve the driving ability.

REFERENCES

1) S. G. Jeong, C. S. Kim, K. S. Yoon, J. N. Lee, J. I. Bae, and M. H. Lee, "Real-time Lane detection for autonomous navigation," in Intelligent Transportation Systems, 2001 IEEE, 2001, pp. 508-513.
2) K. A. Redmill, S. Upadhya, A. Krishnamurthy, and U. Ozguner, "A lane tracking system for intelligent vehicle applications," in Intelligent Transportation Systems, 2001 IEEE, 2001, pp. 273-279.
3) H. Y. Cheng, B. S. Jeng, P. T. Tseng, and K.-C. Fan, "Lane detection with moving vehicles in the traffic scenes," IEEE Transactions on intelligent transportation systems, vol. 7, no. 4, pp. 571-582, 2006.
4) K. Ghazali, R. Xiao, and J. Ma, "Road Lane detection using H-maxima and improved hough transform," in Computational Intelligence, Modelling and Simulation (CIMSiM), 2012 Fourth International Conference on, 2012, pp. 205-208.
5) M. Aly, "Real time detection of lane markers in urban streets," in Intelligent Vehicles Symposium, 2008 IEEE, 2008, pp. 7-12.
6) S. Hong, M. H. Lee, S. H. Kwon, and H. H. Chun, "A car test for the estimation of GPS/INS alignment errors," IEEE Transactions on Intelligent Transportation Systems, vol. 5, no. 3, pp. 208-218, 2004.
7) V. Milanés, J. E. Naranjo, C. González, J. Alonso, and T. de Pedro, "Autonomous vehicle based in cooperative GPS and inertial systems," Robotica, vol. 26, no. 5, pp. 627-633, 2008.
8) D. A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)", 2015.
9) D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization", 2014
10) N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," The Journal of Machine Learning Research, vol. 15, no. 1, pp. 1929-1958, 2014.