

Virtual Self Driving Car using Convolution Neural Network in Deep Learning

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Technology degree in Information Technology

by

R.Vijay Vinayak (37120084)

R.N.Akash (37120004)



**DEPARTMENT OF INFORMATION TECHNOLOGY
SCHOOL OF COMPUTING**

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

Accredited with Grade "A" by NAAC

**JEPPIAAR NAGAR, RAJIV GANDHI SALAI,
CHENNAI - 600 119**

MARCH – 2021



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited with Grade "A" by NAAC

Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai – 600 119

www.sathyabama.ac.in



DEPARTMENT OF INFORMATION TECHNOLOGY

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of R.Vijay Vinayak (37120084), R.N.Akash (37120004) who carried out the project entitled "Virtual Self Driving Car using Convolution Neural Network in Deep Learning" under my supervision from November 2020 to April 2021.

Internal Guide

Dr.Y.Bevis Jinila M.E., Ph.D.

Head of the Department

Dr.R.Subhashini M.E., Ph.D.

Submitted for Viva voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

I R.N.Akash (37120004), R.Vijay Vinayak (37120084) hereby declare that the Project Report entitled "Virtual Self Driving Car Using Convolution Neural Network in Deep Learning" done by me under the guidance of Dr.Y.Bevis Jinila is submitted in partial fulfillment of the requirements for the award of Bachelor of Technology degree in Information Technology.

DATE:

PLACE:

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management** of **SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T.Sasikala M.E., Ph.D, Dean**, School of Computing, **Dr.R.Subhashini M.E., Ph.D. , Head** of the Department of Information Technology for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr.Y.Bevis Jinila M.E., Ph.D.** for his valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Information Technology** who were helpful in many ways for the completion of the project.

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	LIST OF FIGURES	i
	ABSTRACT	ii
1	INTRODUCTION	
	1.1 GENERAL OVERVIEW	1
	1.2 OUTLINE OF THE PROJECT	2
2	AIM AND SCOPE OF PRESENT INVESTIGATION	
	2.1 AIM	4
	2.2 LITERATURE SURVEY	4
	2.3 PROBLEM STATEMENT	5
	2.4 OBJECTIVE	6
	2.5 CNN TO PROCESS VISUAL DATA	7
3	METHODOLOGY	
	3.1 OVERVIEW OF THE PLATFORM	9
	3.2 LIBRARIES USED	11
	3.3 WORKING MODULE	15

	3.4 CONVOLUTION NEURAL NETWORKS	17
	3.5 CNN APPLICATIONS	18
	3.6 THE DAVE-2 SYSTEM	22
4	PERFORMANCE AND ANALYSIS	
	4.1 TECHNOLOGY USED	26
	4.2 FEASIBILITY	29
	4.3 RESULT AND DISCUSSION	30
5	CONCLUSION AND FUTURE WORK	
	5.1 CONCLUSION	32
	5.5 SUMMARY	32
6	REFERENCE	34
7	APPENDICES	
	7.1 SAMPLE CODE	36
	7.2 SCREENSHOTS	42

LIST OF FIGURES

FIGURE NO	FIGURE NAME	PAGE NO
1	System Architecture	8
2	Tensorflow Architecture	14
3	Convolution Neural Network Achitecture	15
4	Block Diagram of vehicle Simulator	16
5	High level view of data collection system	23
6	Training neural network	24
7	Training network used to generate steering commands from camera	25
8	Beginning of training the model	21
9	End of training the model	21
10	Tensorboard	22

ABSTRACT

Virtual Self Driving Car using Convolution Neural Network in Deep Learning

Recently, the technological advancements have led to the adoption of technology in the automobile sector. There has been a constant improvement in the design and development of self driving cars. The primary goal of self driving car is to make the vehicle move in the correct lane. So, lane detection is one of the preliminary steps to make this possible. Existing literatures have suggested various schemes and methods to implement and improve the efficiency of the self driving cars. In this paper, a deep neural network namely the CNN is used as a model.

Chapter – 1

INTRODUCTION

1.1 General Overview

The advancements in technology have led to the rapid development of different types of innovations in the automobile sector. Frequently, new inventions and changes can be addressed in the automobile industry. In a gist, there are a variety of vehicles coming on road in a short span. Due to the increase in the number of vehicles on road, there is an exponential increase in the number of accidents reported every year. So safe driving has become the focus of attention. To make this possible, numerous researches have been carried out in this field to make the transportation autonomous and to make the transportation as Intelligent Transportation System (ITS). One solution for this is self driving car.

A self driving car is a vehicle that is capable of navigating and sensing its environment without human involvements. Self driving detects its environment using variety of techniques like GPS, radar and computer vision. Here our problem is to implement the utilization of autonomous vehicles in everyday use to reduce accidents caused by manual transportation.

The primary step in self driving cars is lane detection. In case of lane detection, several parameters have to be considered namely the curvature of the road, edge of the lane and deviation from the centre of the lane to detect a lane. Advanced driver assistance systems include lane departure warning, lane keeping assist and adaptive cruise control can help people to analyze the driving environment and provides required feedback for safe driving.

From the existing literatures it is evident that, many schemes like color coding, neural network and deep neural networks schemes have been employed for lane detection. It is evident from the analysis that, there is a short fall in the improper selection of features to do the process. Hence, feature selection has to be appropriate to improve the lane detection process. In this paper, a CNN model is proposed with efficient feature selection scheme. Self-driving vehicles are going to be of enormous economic impact

over the coming decade. Creating models that meet or exceed the ability of a human driver could save thousands of lives a year. Udacity has an ongoing challenge to create an open source self-driving car [19]. In their second challenge Udacity released a dataset of images taken while driving along with the corresponding steering angle and ancillary sensor data for a training set (left, right, and center cameras with interpolated angles based on camera angle). The goal of the challenge was to find a model that, given an image taken while driving, will minimize the RMSE (root mean square error) between what the model predicts and the actual steering angle produced by a human driver. In this project, we explore a variety of techniques including 3D convolutional neural networks, recurrent neural networks using LSTM, ResNets, etc. to output a predicted steering angle in numerical values. The motivation of the project is to eliminate the need for hand-coding rules and instead create a system that learns how to drive by observing. Predicting steering angle is one important part of the end-to-end approach to self-driving car and would allow us to explore the full power of neural networks. For example, using only steering angle as the training signal, deep neural networks can automatically extract features to help position the road to make the prediction. The two models that will be discussed are a model that used 3D convolutional layers followed by recurrent layers using LSTM (long short term memory). This model will explore how temporal information is used to predict steering angle. Both 3D convolutional layers and recurrent layers make use of temporal information. The second model to be discussed uses transfer learning (using lower layers of a pre-trained model) by using a high quality model trained on another dataset. Transfer learning helps to mitigate the amount of training data needed to create a high quality model.

1.2 Outline of the Project

Every year, there are around *1.25 million* deaths caused by road accidents. That's equivalent to *3,287 deaths* on a daily basis! As a teenager just learning how to drive, this is a scary fact that lingers at the back of my mind. On top of that, there's also a ridiculous amount of traffic that we have to suffer through, which just creates

unnecessary frustration for most people.

The primary step in self driving cars is lane detection. In case of lane detection, several parameters have to be considered namely the curvature of the road, edge of the lane and deviation from the centre of the lane to detect a lane. Advanced driver assistance systems include lane departure warning, lane keeping assist and adaptive cruise control can help people to analyze the driving environment and provides required feedback for safe driving.

From the existing literatures it is evident that, many schemes like color coding, neural network and deep neural networks schemes have been employed for lane detection. It is evident from the analysis that, there is a short fall in the improper selection of features to do the process. Hence, feature selection has to be appropriate to improve the lane detection process. In this paper, a CNN model is proposed with efficient feature selection scheme.

CHAPTER 2

AIM AND SCOPE OF PRESENT INVESTIGATION

2.1 Aim

A self-driving cars is a vehicle that is capable of sensing its environment and navigating without human input .Self-driving cars can detect environments using a variety of techniques such as radar ,GPS and computer vision .Our problem is to look at the implications of utilizing autonomous vehicles in everyday use to reduce negative impacts on society caused by manual transportation .By using the virtual self driving user can easily identify the working flow and implement into real world. Recently, the technological advancements have led to the adoption of technology in the automobile sector. There has been a constant improvement in the design and development of self driving cars. The primary goal of self driving car is to make the vehicle move in the correct lane. So, lane detection is one of the preliminary steps to make this possible. Existing literatures have suggested various schemes and methods to implement and improve the efficiency of the self driving cars. In this paper, a deep neural network namely the CNN is used as a model.

2.2 Literature Survey

This section briefly presents the related works in the literature on the implementation of self driving cars.

LeCun et al.[1] has proposed the ability to learn how to do more can be greatly enhanced by providing issues from the workplace. This paper shows how such issues can be integrated into a back-end network through network design.

Alex Krizhivsky [2] has proposed a deep convolution neural network of 1.2 million image splits in the 10000 different categories. In experimental data, it was found that 37.5% of the top one and top five error ratings and 15% which is best when compared to previous case. The network, consisting of 70 million parameters and 65000 nodes, consists of 5 specific layer,here few follows integration layer, and 3 FC layers with a

final 10000 softmax. To complete training the model fast, we have created non-complementary neurons and GPU effectively use of cnn function.

INet Scale Technologies Inc.[3] proposed an Independent international vehicle controls have significant commercial and military benefits but the effectiveness of current systems is not strong enough to allow for deployment. With some difficulty coping with unfamiliar areas, sound sensor data and self-reading improve performance while driving. Advances in pattern recognition machine learning, control problems, and, most importantly, end-to-end learning promises improvement in these areas. The aim of this project was to make a short-term effort and focus on exploring the benefits of learning with the final machine of this app. The test car was built, consisting of a commercial radio truck equipped with low-speed video cameras and other sensors. The task of the vehicle was to navigate an unknown open space while avoiding any obstacles such as rocks, trees, ditches, and lakes. All work was learned by machine from examples. No manually designed algorithms were used and the paid network was not manually installed in any way. A well-trained vehicle has been able to navigate successfully on complex and difficult paths through obstacles. A skill that sometimes a human driver could not repeat even after many attempts. The entire project was completed in seven months by a team of three to seven people with a total of almost \$ 30,000 computer hardware including a car, sensors, remote and training and data storage computers.

Danwii Wung and Fing [4] proposed a all wheel drive. This paper develops a four-wheel drive algorithm for aligning a all wheel-drive vehicle based on car kinematics. The alignment given by the steering is completely used in trajectory planning. The two different algorithm consists of steering angle planning and speed of vehicle planning. Vehicle route and driving torque limits are considered. Simulation results presented to demonstrate the use of the proposed algorithm.

2.3 Problem Statement

In order to increase roadway safety autonomous vehicles are under development and are the focus of many research projects. The Rochester Institute of Technology wishes to have a reentry point to the field of autonomous vehicle research and currently does

not have a vehicle to use as a starting point. A self-driving car is a vehicle that is capable of sensing its environment and navigating without human input. Self-driving cars can detect environments using a variety of techniques such as radar, GPS and computer vision. Our problem is to look at the implications of utilizing autonomous vehicles in everyday use to reduce negative impacts on society caused by manual transportation. By using the virtual self-driving user can easily identify the working flow and implement into real world. To facilitate RIT's goal a remote control golf cart was determined to be the first stepping stone. The scope of this project includes converting a golf cart to a remote control vehicle. The safety of the vehicle's passengers and bystanders is of the utmost concern. Therefore the vehicle is required to be low speed and contain the ability for passengers to take control.

2.4 Objective

The main objective of our work was to achieve autonomous driving with a light deep neural network suitable for deployment on embedded automotive platforms. The main idea of our work was to develop a solution for autonomous driving for a light automotive platform that has limited hardware resources, processor power, and memory size. Having those hardware restrictions in mind, we are aiming to design a light deep neural network (DNN). Recently, the technological advancements have led to the adoption of technology in the automobile sector. There has been a constant improvement in the design and development of self-driving cars. The primary goal of self-driving car is to make the vehicle move in the correct lane. So, lane detection is one of the preliminary steps to make this possible. Existing literatures have suggested various schemes and methods to implement and improve the efficiency of the self-driving cars. In this paper, a deep neural network namely the CNN is used as a model. Automation can help reduce the number of crashes on our roads. Over the last years the automotive industry has advanced significantly towards a future without human drivers. Researchers are currently trying to overcome the technological, political and social challenges involved in making autonomous vehicles mainstream. These vehicles need to be safe, reliable and cost-efficient. Connecting them and creating coordination mechanisms could help

achieving these goals. his project proposes a vehicle-to-everything (V2X) coordination protocol for autonomous vehicles (AVCP: Autonomous Vehicle Coordination Protocol) and a testing environment where it will be evaluated. The AVCP aims to significantly reduce travel time and increase security for autonomous vehicles by enabling them to exchange sensing and routing information with each other and with roadside units (RSUs). Self-driving vehicles have expanded dramatically over the last few years. Udacity has release a dataset containing, among other data, a set of images with the steering angle captured during driving. The Udacity challenge aimed to predict steering angle based on only the provided images. We explore two different models to perform high quality prediction of steering angles based on images using different deep learning techniques including Transfer Learning, 3D CNN, LSTM and ResNet. If the Udacity challenge was still ongoing, both of our models would have placed in the top ten of all entries.

2.5 CNN to Process Visual Data

CNNs have revolutionized the computational pattern recognition process. Prior to the widespread adoption of CNNs, most pattern recognition tasks were performed using an initial stage of hand-crafted feature extraction followed by a classifier. The important breakthrough of CNNs is that features are now learned automatically from training examples. The CNN approach is especially powerful when applied to image recognition tasks because the convolution operation captures the 2D nature of images. By using the convolution kernels to scan an entire image, relatively few parameters need to be learned compared to the total number of operations.

While CNNs with learned features have been used commercially for over twenty years ,their adoption has exploded in recent years because of two important developments. First, large, labeled data sets such as the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) are now widely available for training and validation. Second, CNN learning algorithms are now implemented on massively parallel graphics processing units (GPUs), tremendously accelerating learning and inference ability.

The CNNs that we describe here go beyond basic pattern recognition. We developed a system that learns the entire processing pipeline needed to steer an automobile. The

groundwork for this project was actually done over 10 years ago in a Defense Advanced Research Projects Agency (DARPA) seedling project known as DARPA Autonomous Vehicle (DAVE), in which a sub-scale radio control (RC) car drove through a junk-filled alley way. DAVE was trained on hours of human driving in similar, but not identical, environments. The training data included video from two cameras and the steering commands sent by a human operator.

In many ways, DAVE was inspired by the pioneering work of Pomerleau, who in 1989 built the Autonomous Land Vehicle in a Neural Network (ALVINN) system. ALVINN is a precursor to DAVE, and it provided the initial proof of concept that an end-to-end trained neural network might one day be capable of steering a car on public roads. DAVE demonstrated the potential of end-to-end learning, and indeed was used to justify starting the DARPA Learning Applied to Ground Robots (LAGR) program, but DAVE's performance was not sufficiently reliable to provide a full alternative to the more modular approaches to off-road driving. (DAVE's mean distance between crashes was about 20 meters in complex environments.)

About a year ago we started a new effort to improve on the original DAVE, and create a robust system for driving on public roads. The primary motivation for this work is to avoid the need to recognize specific human-designated features, such as lane markings, guard rails, or other cars, and to avoid having to create a collection of "if, then, else" rules, based on observation of these features. We are excited to share the preliminary results of this new effort, which is aptly named: DAVE-2.

CHAPTER 3

METHODOLOGY

3.1 Overview of the Platform

In this section the proposed approach for implementing self driving car is briefly given. To use a self-driving car, the first Convolution Neural Network (CNN) is trained using a database, the Convolution Neural Network (CNN) places a green pixel map from the front-facing camera to the driving instructions. The system is trained to automatically learn the internal representation step by step, such as finding useful road features, with a personal directional angle only as a training signal. It is not needed to explicitly train it to find, for example, a road map.

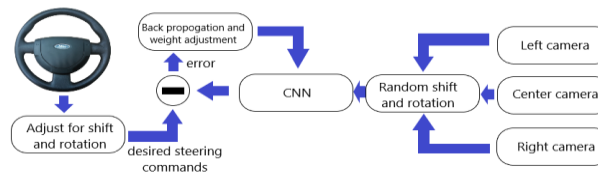


Fig. 1 System Architecture

Automobile manufacturers such as Ford, General Motors, Tesla, and other companies such as NVIDIA are investing billions of dollars in autonomous vehicle driving research. According to Intel, by 2050, this fast-growing industry will be worth \$ 7 trillion [25]. Governments of countries in Europe and of the USA are creating regulations for self-driving cars, as a result of the latest advances of the industry. The benefits of fully autonomous vehicles can go way beyond removing the need of a human driver. Transportation services such as Uber will start using selfdriving cars instead of human drivers, and might become a cheaper and better alternative for end consumers than owning a car. This will represent a shift on the way cities are planned, as fewer parking places will be needed, and most importantly, in a smart city with most of its vehicles being connected and autonomous, traffic optimization will be able to be heavily applied by coordinating movement. This will result in a major decrease on travel time, and it will also save lives as emergency services will be able to reach their destinations faster. In

contrast, there are many problems that need to be addressed regarding autonomous vehicles. Tesla's car autopilot first fatal crash in 2016 [9], for example, brought up discussions about reliability, safety and legal liability regarding autonomous vehicles among researchers. Other important topics, such as security and transparency will need to be heavily discussed as these vehicles become mainstream. Another problem of this relatively new field is the lack of data sharing between vehicles. Autonomous vehicles are complex systems that rely heavily on technologies such as LIDAR, GPS, high-definition maps, and artificial intelligence for navigation and collision avoidance. This means that each autonomous vehicle is constantly collecting and analyzing a very high volume of data, which according to Intel is about 2.6 terabytes per hour [27]. If this data was shared between the vehicles it could be used for coordinating movement, which could increase safety and also optimize traffic on cities and highways. Produces better result because internal function do it themselves to increase the performance of the entire system, instead of improving the intermediate options chosen by man, e. g., route discovery. Such methods are logically preferred to translate people easily which does not automatically verify the operation of the system. Self-driving vehicles are expected to revolutionise the way people and goods are moved around, with major benefits expected in terms of safety, accessibility and traffic flow. When it comes to safety, human error is generally regarded as the biggest single contributor to road traffic accidents – with some official reports estimating that it is a factor in more than 90% of serious accidents. Self-driving cars still have to prove that they can achieve the same high mileage that human drivers currently attain on average without incident, but once they have passed that milestone it is expected that their more regulated, less erratic style of driving and interacting with other road users will lead to far fewer accidents. As well as making our roads safer, self-driving vehicles should make them more accessible for those who are unable or unwilling to drive – due to age, frailty or disability – opening up new mobility options for people who currently need to rely on others for their transport needs. It is hard to fully calculate the effect that self-driving vehicles will have on traffic congestion, particularly since they are expected to radically alter the nature of driving. The aforementioned 'new journeys' that could be made by people who are

currently unable to drive may lead to an increase in total journeys made, as could the possibility of cars driving without any people on board at all – for example, when driving to pick up passengers. However, many experts believe this will be offset by the increased efficiency of self-driving vehicles, the potential ability of autonomous vehicles to drive closer to each other and the time savings brought about by the anticipated reduction in the number of accidents. Research carried out as part of UK Autodrive concluded that traffic delays could be reduced by as much as 30% if all of the vehicles on our roads were replaced by fully self-driving vehicles – although this research did not consider the net effect of the potential new journey

3.2 Libraries Used

OS : This module provides a portable way of using operating system dependent functionality. If you just want to read or write a file see `open()`, if you want to manipulate paths, see the `os.path` module, and if you want to read all the lines in all the files on the command line see the `fileinput` module. For creating temporary files and directories see the `tempfile` module, and for high-level file and directory handling see the `shutil` module.

Tensorflow : Machine learning is a complex discipline. But implementing machine learning models is far less daunting and difficult than it used to be, thanks to machine learning frameworks—such as Google’s TensorFlow—that ease the process of acquiring data, training models, serving predictions, and refining future results.

Created by the Google Brain team, TensorFlow is an open source library for numerical computation and large-scale machine learning. TensorFlow bundles together a slew of machine learning and deep learning (aka neural networking) models and algorithms and makes them useful by way of a common metaphor. It uses Python to provide a convenient front-end API for building applications with the framework, while executing those applications in high-performance C++.

TensorFlow can train and run deep neural networks for handwritten digit classification, image recognition, word embeddings, recurrent neural networks, sequence-to-sequence models for machine translation, natural language processing, and PDE (partial differential equation) based simulations. Best of all, TensorFlow supports production prediction at scale, with the same models used for training.

TensorFlow allows developers to create dataflow graphs—structures that describe how data moves through a graph, or a series of processing nodes. Each node in the graph represents a mathematical operation, and each connection or edge between nodes is a multidimensional data array, or tensor.

TensorFlow provides all of this for the programmer by way of the Python language. Python is easy to learn and work with, and provides convenient ways to express how high-level abstractions can be coupled together. Nodes and tensors in TensorFlow are Python objects, and TensorFlow applications are themselves Python applications.

The actual math operations, however, are not performed in Python. The libraries of transformations that are available through TensorFlow are written as high-performance C++ binaries. Python just directs traffic between the pieces, and provides high-level programming abstractions to hook them together.

TensorFlow applications can be run on most any target that's convenient: a local machine, a cluster in the cloud, iOS and Android devices, CPUs or GPUs. If you use Google's own cloud, you can run TensorFlow on Google's custom TensorFlow Processing Unit (TPU) silicon for further acceleration. The resulting models created by TensorFlow, though, can be deployed on most any device where they will be used to serve predictions.

TensorFlow 2.0, released in October 2019, revamped the framework in many ways based on user feedback, to make it easier to work with (e.g., by using the relatively simple Keras API for model training) and more performant. Distributed training is easier to run thanks to a new API, and support for TensorFlow Lite makes it possible to deploy

models on a greater variety of platforms. However, code written for earlier versions of TensorFlow must be rewritten—sometimes only slightly, sometimes significantly—to take maximum advantage of new TensorFlow 2.0 features.

CV2 : OpenCV is a huge open-source library for computer vision, machine learning, and image processing. OpenCV supports a wide variety of programming languages like Python, C++, Java, etc. It can process images and videos to identify objects, faces, or even the handwriting of a human. When it is integrated with various libraries, such as Numpy which is a highly optimized library for numerical operations, then the number of weapons increases in your Arsenal i.e whatever operations one can do in Numpy can be combined with OpenCV.

This OpenCV tutorial will help you learn the Image-processing from Basics to Advance, like operations on Images, Videos using a huge set of Opencv-programs and projects.

OpenCV was started at Intel in 1999 by Gary Bradsky, and the first release came out in 2000. Vadim Pisarevsky joined Gary Bradsky to manage Intel's Russian software OpenCV team. In 2005, OpenCV was used on Stanley, the vehicle that won the 2005 DARPA Grand Challenge. Later, its active development continued under the support of Willow Garage with Gary Bradsky and Vadim Pisarevsky leading the project. OpenCV now supports a multitude of algorithms related to Computer Vision and Machine Learning and is expanding day by day.

OpenCV supports a wide variety of programming languages such as C++, Python, Java, etc., and is available on different platforms including Windows, Linux, OS X, Android, and iOS. Interfaces for high-speed GPU operations based on CUDA and OpenCL are also under active development. OpenCV-Python is the Python API for OpenCV, combining the best qualities of the OpenCV C++ API and the Python language.

Python is a general purpose programming language started by Guido van Rossum that became very popular very quickly, mainly because of its simplicity and code readability.

It enables the programmer to express ideas in fewer lines of code without reducing readability.

Compared to languages like C/C++, Python is slower. That said, Python can be easily extended with C/C++, which allows us to write computationally intensive code in C/C++ and create Python wrappers that can be used as Python modules. This gives us two advantages: first, the code is as fast as the original C/C++ code (since it is the actual C++ code working in background) and second, it is easier to code in Python than C/C++. OpenCV-Python is a Python wrapper for the original OpenCV C++ implementation.

OpenCV-Python makes use of Numpy, which is a highly optimized library for numerical operations with a MATLAB-style syntax. All the OpenCV array structures are converted to and from Numpy arrays. This also makes it easier to integrate with other libraries that use Numpy such as SciPy and Matplotlib.

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."

Math : Some of the most popular mathematical functions are defined in the math module. These include trigonometric functions, representation functions, logarithmic functions, angle conversion functions, etc. In addition, two mathematical constants are also defined in this module. Pi is a well-known mathematical constant, which is defined as the ratio of the circumference to the diameter of a circle and its value is 3.141592653589793.

3.3 Working Module

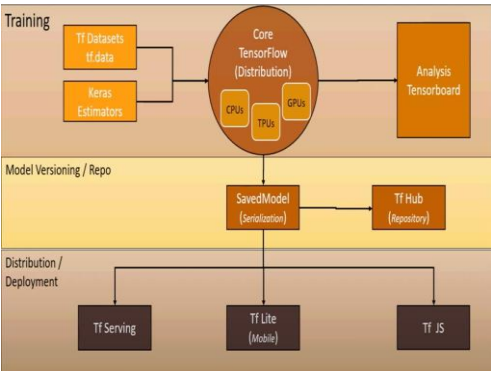


Fig. 2. Tensorflow Architecture

Tensorflow can be used across a range of task but has a particular focus on training and inference of deep neural networks.Steps involved in implementing tensorflow use case: First it process the given dataset, that defines features and labels.Then it divides the given dataset into two parts namely training and testing.We train our model on the training data and try it tries to reduce error.Finally we test our trained model in test data as shown in fig.2 .Here aim is to achieve the highest possible accuracy by repeating the process to decrease the loss.

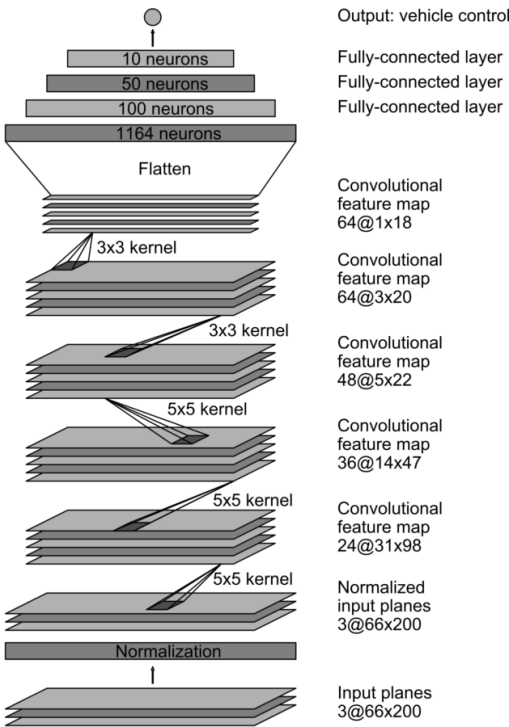


Fig. 3. Convolution Neural Network Achitecture (CNN)

Using a neural network for autonomous vehicle navigation was pioneered by Pomerleau (1989) who built the Autonomous Land Vehicle in a Neural Network (ALVINN) system. The model structure was relatively simple, comprising a fully-connected network, which is tiny by today's standard. The network predicted actions from pixel inputs applied to simple driving scenarios with few obstacles. However, it demonstrated the potential of neural networks for end-to-end autonomous navigation. Last year, NVIDIA released a paper regarding a similar idea that benefited from ALVINN. In the paper, the authors used a relatively basic CNN architecture to extract features from the driving frames. The layout of the architecture can be seen. Augmentation of the data collected was found to be important. The authors used artificial shifts and rotations of the training set. Left and right cameras with interpolated steering angles were also incorporated. This framework was successful in relatively simple real-world scenarios, such as highway lane-following and driving in flat, obstacle-free courses. Recently, more attempts on using deep CNNs and RNNs to tackle the challenges of video classification, scene parsing, and object detection have stimulated the applications of more complicated CNN architectures in autonomous driving. "Learning Spatiotemporal Features with 3D Convolutional Networks" introduces how to construct 3D Convolutional Networks to capture spatiotemporal features in a sequence of images or videos. "Beyond Short Snippets: Deep Networks for Video Classification" describes two ways including using LSTM to deal with videos

We train network equipment to reduce bugs between the issuance of a network control command, or a human driver command or a fixed direction command for central and different angle images. Figure 2 shows the layer structure, consisting of different layers, including a standard performance layer, five layers of solution, three layers that are full integrated.

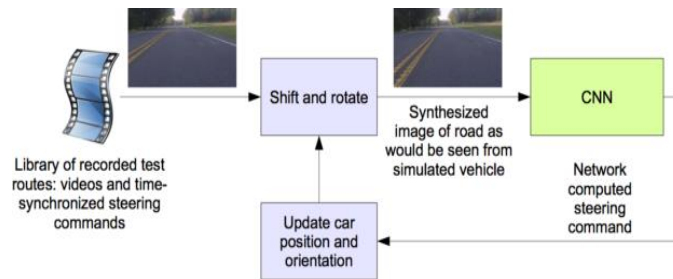


Fig. 4. Block Diagram of vehicle Simulator

Here the video file is converted to image frames. Those frames are stored in a file. While training, the network assigns specific angle to each frames and finally the model is created and stored in the following directory. Manual drivers cannot maintain the vehicle at the center of the lane. So the network while assigning the steering angle it also produces the lane center angle that is stored in each frame. Now the model is created with steering angle for all frames. Now it can implement into real world. It can be tested by placing camera in car front bumper. Here we can see the steering angle operating by itself accurately by maintain the vehicle in the center of the road. The simulator uses the image frames to get the steering angle while the camera in the car front bumper is running live. The model helps to coordinate with the recorded frames and live camera. Finally we get the steering angle accurately. CNN manual instructions and recorded driver's instructions are included in the powerful car model to revive the position and position of the car being made. According to the video CNN checks for the available frames in the database and aligns the steering wheel accordingly. Everytime the new image is added to CNN and this method is repeated.

3.4 Deep Convolution Neural Network

In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery. They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on the shared-weight architecture of the convolution kernels that shift over input features and provide translation equivariant responses. Counter-intuitively, most convolutional

neural networks are only equivariant, as opposed to invariant, to translation. They have applications in image and video recognition, recommender systems, image classification, image segmentation, medical image analysis, natural language processing, brain-computer interfaces, and financial time series.

CNNs are regularized versions of multilayer perceptrons. Multilayer perceptrons usually mean fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The "full connectivity" of these networks make them prone to overfitting data. Typical ways of regularization, or preventing overfitting, include: penalizing parameters during training (such as weight decay) or trimming connectivity (skipped connections, dropout, etc.) CNNs take a different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble patterns of increasing complexity using smaller and simpler patterns embossed in their filters. Therefore, on a scale of connectivity and complexity, CNNs are on the lower extreme.

Convolutional networks were inspired by biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field.

CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns to optimize the filters (or kernels) through automated learning, whereas in traditional algorithms these filters are hand-engineered. This independence from prior knowledge and human intervention in feature extraction is a major advantage.

3.5 Convolution Neural Network Applications

Image recognition

CNNs are often used in image recognition systems. In 2012 an error rate of 0.23% on the MNIST database was reported. Another paper on using CNN for image classification reported that the learning process was "surprisingly fast"; in the same paper, the best published results as of 2011 were achieved in the MNIST database and the NORB database.

When applied to facial recognition, CNNs achieved a large decrease in error rate. Another paper reported a 97.6% recognition rate on "5,600 still images of more than 10 subjects". CNNs were used to assess video quality in an objective way after manual training; the resulting system had a very low root mean square error.

The ImageNet Large Scale Visual Recognition Challenge is a benchmark in object classification and detection, with millions of images and hundreds of object classes. In the ILSVRC 2014, a large-scale visual recognition challenge, almost every highly ranked team used CNN as their basic framework. The winner GoogLeNet (the foundation of DeepDream) increased the mean average precision of object detection to 0.439329, and reduced classification error to 0.06656, the best result to date. Its network applied more than 30 layers. That performance of convolutional neural networks on the ImageNet tests was close to that of humans. The best algorithms still struggle with objects that are small or thin, such as a small ant on a stem of a flower or a person holding a quill in their hand. They also have trouble with images that have been distorted with filters, an increasingly common phenomenon with modern digital cameras. By contrast, those kinds of images rarely trouble humans. Humans, however, tend to have trouble with other issues. For example, they are not good at classifying objects into fine-grained categories such as the particular breed of dog or species of bird, whereas convolutional neural networks handle this.[citation needed]

In 2015 a many-layered CNN demonstrated the ability to spot faces from a wide range of angles, including upside down, even when partially occluded, with competitive performance. The network was trained on a database of 200,000 images that included faces at various angles and orientations and a further 20 million images without faces. They used batches of 128 images over 50,000 iterations.

Video analysis

Compared to image data domains, there is relatively little work on applying CNNs to video classification. Video is more complex than images since it has another (temporal) dimension. However, some extensions of CNNs into the video domain have been explored. One approach is to treat space and time as equivalent dimensions of the input and perform convolutions in both time and space. Another way is to fuse the features of two convolutional neural networks, one for the spatial and one for the temporal stream. Long short-term memory (LSTM) recurrent units are typically incorporated after the CNN to account for inter-frame or inter-clip dependencies. Unsupervised learning schemes for training spatio-temporal features have been introduced, based on Convolutional Gated Restricted Boltzmann Machines and Independent Subspace Analysis.

Natural language processing

CNNs have also been explored for natural language processing. CNN models are effective for various NLP problems and achieved excellent results in semantic parsing, search query retrieval, sentence modeling, classification, prediction and other traditional NLP tasks.

Anomaly Detection

A CNN with 1-D convolutions was used on time series in the frequency domain (spectral residual) by an unsupervised model to detect anomalies in the time domain.

Drug discovery

CNNs have been used in drug discovery. Predicting the interaction between molecules and biological proteins can identify potential treatments. In 2015, Atomwise introduced AtomNet, the first deep learning neural network for structure-based rational drug design. The system trains directly on 3-dimensional representations of chemical interactions. Similar to how image recognition networks learn to compose smaller, spatially proximate features into larger, complex structures, AtomNet discovers chemical features, such as aromaticity, sp³ carbons and hydrogen bonding.

Subsequently, AtomNet was used to predict novel candidate biomolecules for multiple disease targets, most notably treatments for the Ebola virus and multiple sclerosis.

Health risk assessment and biomarkers of aging discovery

CNNs can be naturally tailored to analyze a sufficiently large collection of time series data representing one-week-long human physical activity streams augmented by the rich clinical data (including the death register, as provided by, e.g., the NHANES study). A simple CNN was combined with Cox-Gompertz proportional hazards model and used to produce a proof-of-concept example of digital biomarkers of aging in the form of all-causes-mortality predictor.

Checkers game

CNNs have been used in the game of checkers. From 1999 to 2001, Fogel and Chellapilla published papers showing how a convolutional neural network could learn to play checker using co-evolution. The learning process did not use prior human professional games, but rather focused on a minimal set of information contained in the checkerboard: the location and type of pieces, and the difference in number of pieces between the two sides. Ultimately, the program (Blondie24) was tested on 165 games against players and ranked in the highest 0.4%. It also earned a win against the program Chinook at its "expert" level of play.

Go

CNNs have been used in computer Go. In December 2014, Clark and Storkey published a paper showing that a CNN trained by supervised learning from a database of human professional games could outperform GNU Go and win some games against Monte Carlo tree search Fuego 1.1 in a fraction of the time it took Fuego to play. Later it was announced that a large 12-layer convolutional neural network had correctly predicted the professional move in 55% of positions, equalling the accuracy of a 6 dan human player. When the trained convolutional network was used directly to play games of Go, without any search, it beat the traditional search program GNU Go in 97% of games, and matched the performance of the Monte Carlo tree search program Fuego simulating ten thousand playouts (about a million positions) per move.

A couple of CNNs for choosing moves to try ("policy network") and evaluating positions ("value network") driving MCTS were used by AlphaGo, the first to beat the best human player at the time.

Time series forecasting

Recurrent neural networks are generally considered the best neural network architectures for time series forecasting (and sequence modeling in general), but recent studies show that convolutional networks can perform comparably or even better. Dilated convolutions might enable one-dimensional convolutional neural networks to effectively learn time series dependences. Convolutions can be implemented more efficiently than RNN-based solutions, and they do not suffer from vanishing (or exploding) gradients. Convolutional networks can provide an improved forecasting performance when there are multiple similar time series to learn from. CNNs can also be applied to further tasks in time series analysis (e.g., time series classification or quantile forecasting).

Cultural Heritage and 3D-datasets

As archaeological findings like clay tablets with cuneiform writing are increasingly acquired using 3D scanners first benchmark datasets are becoming available like HeiCuBeDa providing almost 2.000 normalized 2D- and 3D-datasets prepared with the GigaMesh Software Framework. So curvature based measures are used in conjunction with Geometric Neural Networks (GNNs) e.g. for period classification of those clay tablets being among the oldest documents of human history.

3.6 The Dave-2 System

Three cameras are mounted behind the windshield of the data-acquisition car, and time stamped video from the cameras is captured simultaneously with the steering angle applied by the human driver. The 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).

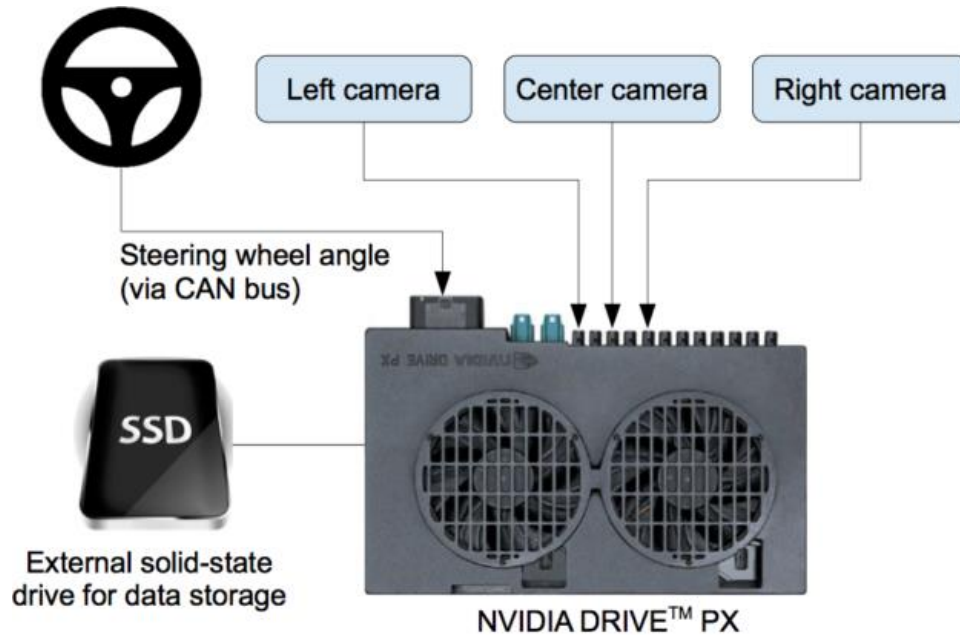


Fig. 5: High-level view of the data collection system.

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 also learn how to recover from any mistakes, or 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 center of the lane and rotations from the direction of the road.

The images for two specific off-center shifts can be obtained from the left and the right cameras. Additional shifts between the cameras and all rotations are simulated through viewpoint transformation of the image from the nearest camera. Precise viewpoint transformation requires 3D scene knowledge which we don't have, so we 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 for a

more complete rendering it introduces distortions for objects that stick above the ground, such as cars, poles, trees, and buildings. Fortunately these distortions don't pose a significant problem for network training. The steering label for the transformed images is quickly adjusted to one that correctly steers the vehicle back to the desired location and orientation in two seconds.

Figure 3 shows a block diagram of our training system. Images are fed into a CNN that then computes a proposed steering command. The proposed command is compared to the desired command for that image, and the weights of the CNN are adjusted to bring the CNN output closer to the desired output. The weight adjustment is accomplished using back propagation as implemented in the Torch 7 machine learning package.

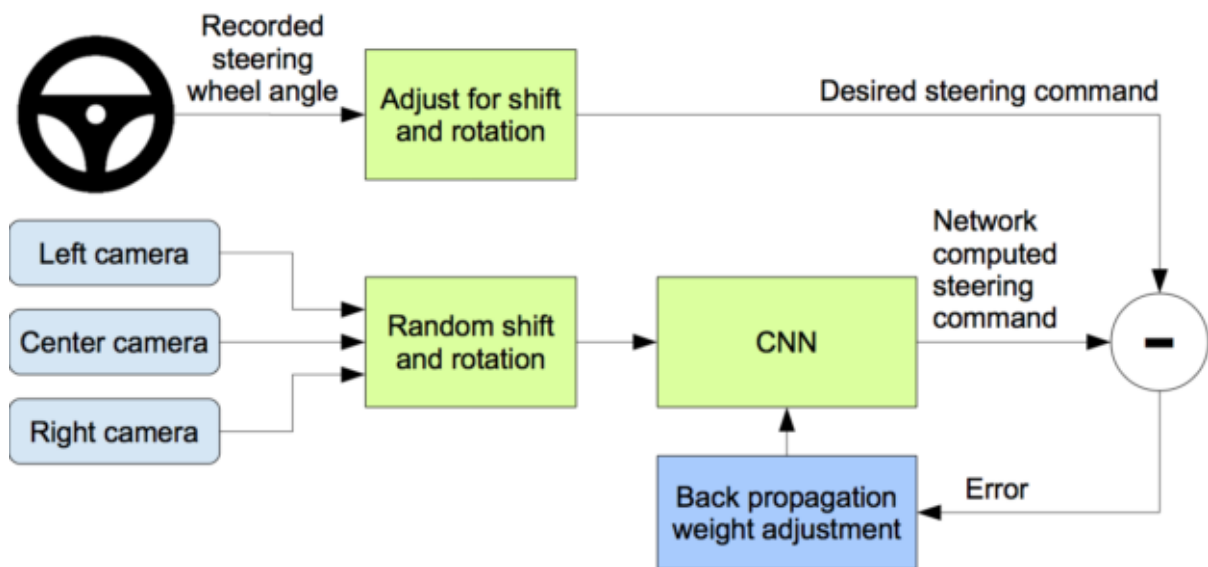


Fig. 6: Training the neural network.

Once trained, the network is able to generate steering commands from the video images of a single center camera. Figure 4 shows this configuration.

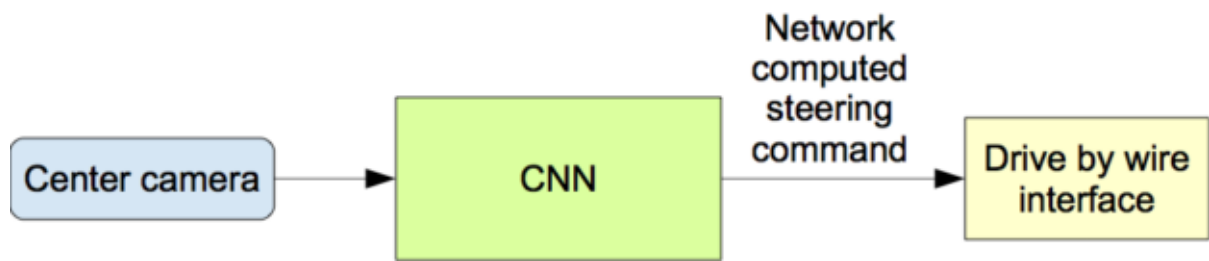


Fig. 7: The trained network is used to generate steering commands from a single front-facing center camera.

CHAPTER 4

PERFORMANCE ANALYSIS

4.1 Technology Used

OS : This module provides a portable way of using operating system dependent functionality. If you just want to read or write a file see `open()`, if you want to manipulate paths, see the `os.path` module, and if you want to read all the lines in all the files on the command line see the `fileinput` module. For creating temporary files and directories see the `tempfile` module, and for high-level file and directory handling see the `shutil` module.

Tensorflow : Machine learning is a complex discipline. But implementing machine learning models is far less daunting and difficult than it used to be, thanks to machine learning frameworks—such as Google’s TensorFlow—that ease the process of acquiring data, training models, serving predictions, and refining future results.

Created by the Google Brain team, TensorFlow is an open source library for numerical computation and large-scale machine learning. TensorFlow bundles together a slew of machine learning and deep learning (aka neural networking) models and algorithms and makes them useful by way of a common metaphor. It uses Python to provide a convenient front-end API for building applications with the framework, while executing those applications in high-performance C++.

TensorFlow can train and run deep neural networks for handwritten digit classification, image recognition, word embeddings, recurrent neural networks, sequence-to-sequence models for machine translation, natural language processing, and PDE (partial differential equation) based simulations. Best of all, TensorFlow supports production prediction at scale, with the same models used for training.

TensorFlow allows developers to create dataflow graphs—structures that describe how data moves through a graph, or a series of processing nodes. Each node in the graph

represents a mathematical operation, and each connection or edge between nodes is a multidimensional data array, or tensor.

TensorFlow provides all of this for the programmer by way of the Python language. Python is easy to learn and work with, and provides convenient ways to express how high-level abstractions can be coupled together. Nodes and tensors in TensorFlow are Python objects, and TensorFlow applications are themselves Python applications.

The actual math operations, however, are not performed in Python. The libraries of transformations that are available through TensorFlow are written as high-performance C++ binaries. Python just directs traffic between the pieces, and provides high-level programming abstractions to hook them together.

TensorFlow applications can be run on most any target that's convenient: a local machine, a cluster in the cloud, iOS and Android devices, CPUs or GPUs. If you use Google's own cloud, you can run TensorFlow on Google's custom TensorFlow Processing Unit (TPU) silicon for further acceleration. The resulting models created by TensorFlow, though, can be deployed on most any device where they will be used to serve predictions.

TensorFlow 2.0, released in October 2019, revamped the framework in many ways based on user feedback, to make it easier to work with (e.g., by using the relatively simple Keras API for model training) and more performant. Distributed training is easier to run thanks to a new API, and support for TensorFlow Lite makes it possible to deploy models on a greater variety of platforms. However, code written for earlier versions of TensorFlow must be rewritten—sometimes only slightly, sometimes significantly—to take maximum advantage of new TensorFlow 2.0 features.

CV2 : OpenCV is a huge open-source library for computer vision, machine learning, and image processing. OpenCV supports a wide variety of programming languages like Python, C++, Java, etc. It can process images and videos to identify objects, faces, or even the handwriting of a human. When it is integrated with various libraries, such

as Numpy which is a highly optimized library for numerical operations, then the number of weapons increases in your Arsenal i.e whatever operations one can do in Numpy can be combined with OpenCV.

This OpenCV tutorial will help you learn the Image-processing from Basics to Advance, like operations on Images, Videos using a huge set of Opencv-programs and projects.

OpenCV was started at Intel in 1999 by Gary Bradsky, and the first release came out in 2000. Vadim Pisarevsky joined Gary Bradsky to manage Intel's Russian software OpenCV team. In 2005, OpenCV was used on Stanley, the vehicle that won the 2005 DARPA Grand Challenge. Later, its active development continued under the support of Willow Garage with Gary Bradsky and Vadim Pisarevsky leading the project. OpenCV now supports a multitude of algorithms related to Computer Vision and Machine Learning and is expanding day by day.

OpenCV supports a wide variety of programming languages such as C++, Python, Java, etc., and is available on different platforms including Windows, Linux, OS X, Android, and iOS. Interfaces for high-speed GPU operations based on CUDA and OpenCL are also under active development. OpenCV-Python is the Python API for OpenCV, combining the best qualities of the OpenCV C++ API and the Python language.

Python is a general purpose programming language started by Guido van Rossum that became very popular very quickly, mainly because of its simplicity and code readability. It enables the programmer to express ideas in fewer lines of code without reducing readability.

Compared to languages like C/C++, Python is slower. That said, Python can be easily extended with C/C++, which allows us to write computationally intensive code in C/C++ and create Python wrappers that can be used as Python modules. This gives us two advantages: first, the code is as fast as the original C/C++ code (since it is the actual C++ code working in background) and second, it is easier to code in Python than C/C++. OpenCV-Python is a Python wrapper for the original OpenCV C++ implementation.

OpenCV-Python makes use of Numpy, which is a highly optimized library for numerical operations with a MATLAB-style syntax. All the OpenCV array structures are converted to and from Numpy arrays. This also makes it easier to integrate with other libraries that use Numpy such as SciPy and Matplotlib.

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."

Math : Some of the most popular mathematical functions are defined in the math module. These include trigonometric functions, representation functions, logarithmic functions, angle conversion functions, etc. In addition, two mathematical constants are also defined in this module.

Pi is a well-known mathematical constant, which is defined as the ratio of the circumference to the diameter of a circle and its value is 3.141592653589793.

4.2 Feasibility

An explorative in-lab user study was conducted to gain deeper insights into the take-over behavior of drivers. With the help of this study, the proposed and implemented immediate handover assistant was evaluated. Tensorflow can be used across a range of task but has a particular focus on training and inference of deep neural networks. Steps involved in implementing tensorflow use case: First it process the given dataset, that defines features and labels. Then it divides the given dataset into two parts namely training and testing. We train our model on the training data and try it tries to intended to reveal whether drivers that are distracted by a realistic distractor task are able to take over the control from one moment to the other, and if they feel comfortable with such handovers. The study was conducted with the implemented prototype and the approaching screens reduce error. Finally we test our trained model in test data as

shown in fig.1 .Here aim is to achieve the highest possible accuracy by repeating the process to decrease the loss.

This study of fog scenario.

4.3 Result and Discussion

1. Training the Model

```
Last login: Tue Feb 23 07:44:47 on ttys000
akashnandagopal@Akashs-MacBook-Pro ~ % cd Desktop
akashnandagopal@Akashs-MacBook-Pro Desktop % cd pro
akashnandagopal@Akashs-MacBook-Pro pro % python3 train.py
WARNING:tensorflow:From /Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/tensorflow/python/compat/v1_
orflow/python/ops/variable_scope.py:163: tf.nn.conv2d is deprecated and will be removed in a future version.
Instructions for updating:
non-resource variables are not supported in the long term
WARNING:tensorflow:From /Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/tensorflow/python/util/dispatch.py:100:
tf.nn.conv2d is deprecated and will be removed in a future version.
Instructions for updating:
Please use 'rate' instead of 'keep_prob'. Rate should be set to 'rate = 1 - keep_prob'.
2021-02-23 07:46:16.283857: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary is optimized with oneAPI
dynamic CPU instructions in performance-critical operations: AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2021-02-23 07:46:16.991121: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:258] None of the MLIR optimization passes
Epoch: 0, Step: 0, Loss: 18.2302
Epoch: 0, Step: 10, Loss: 6.56868
Epoch: 0, Step: 20, Loss: 6.57767
Epoch: 0, Step: 30, Loss: 6.58664
Epoch: 0, Step: 40, Loss: 6.21181
Epoch: 0, Step: 50, Loss: 6.36895
Epoch: 0, Step: 60, Loss: 6.7569
Epoch: 0, Step: 70, Loss: 6.18133
Epoch: 0, Step: 80, Loss: 6.07573
Epoch: 0, Step: 90, Loss: 5.98887
Epoch: 0, Step: 100, Loss: 6.07628
Epoch: 0, Step: 110, Loss: 5.92587
Epoch: 0, Step: 120, Loss: 6.85154
Epoch: 0, Step: 130, Loss: 5.98298
Epoch: 0, Step: 140, Loss: 5.74228
Epoch: 0, Step: 150, Loss: 5.66672
Epoch: 0, Step: 160, Loss: 5.56228
Epoch: 0, Step: 170, Loss: 5.66396
Epoch: 0, Step: 180, Loss: 5.64383
```

Fig.8. Beginning of training the model

1. Training the Model

```
Epoch: 29, Step: 3120, Loss: 0.143451
Epoch: 29, Step: 3130, Loss: 0.224523
Epoch: 29, Step: 3140, Loss: 0.141183
Epoch: 29, Step: 3150, Loss: 0.138307
Epoch: 29, Step: 3160, Loss: 0.141191
Epoch: 29, Step: 3170, Loss: 0.145083
Epoch: 29, Step: 3180, Loss: 0.151927
Epoch: 29, Step: 3190, Loss: 0.14836
Epoch: 29, Step: 3200, Loss: 0.143089
Epoch: 29, Step: 3210, Loss: 0.155034
Epoch: 29, Step: 3220, Loss: 0.14869
Epoch: 29, Step: 3230, Loss: 0.144165
Epoch: 29, Step: 3240, Loss: 0.472837
Epoch: 29, Step: 3250, Loss: 0.142371
Epoch: 29, Step: 3260, Loss: 0.178216
Epoch: 29, Step: 3270, Loss: 0.138012
Epoch: 29, Step: 3280, Loss: 0.143052
Epoch: 29, Step: 3290, Loss: 0.135788
Epoch: 29, Step: 3300, Loss: 0.21534
Epoch: 29, Step: 3310, Loss: 0.141223
Epoch: 29, Step: 3320, Loss: 0.141223
Epoch: 29, Step: 3330, Loss: 0.137741
Epoch: 29, Step: 3340, Loss: 0.134547
Epoch: 29, Step: 3350, Loss: 0.142568
Model saved in file: ./save/model.ckpt
Run the command line:
--> tensorboard --logdir=./logs
Then open http://0.0.0.0:6006/ into your web browser
akashnandagopal@Akashs-MacBook-Pro pro %
```

Fig.9. End of training the model

In fig 5, It starts training the model and we can see the training accuracy value. The loss values in higher in the beginning of training the model, which means the accuracy is lesser. In fig. 6, the loss value is reduced which means the accuracy in

increased after repeating the process. The process is repeated till 30 epoch and finally the model gets trained and the model is stored at the following directory. After training we can visualize the graph using tensorboard Fig.7 .

2. Graph Visualization

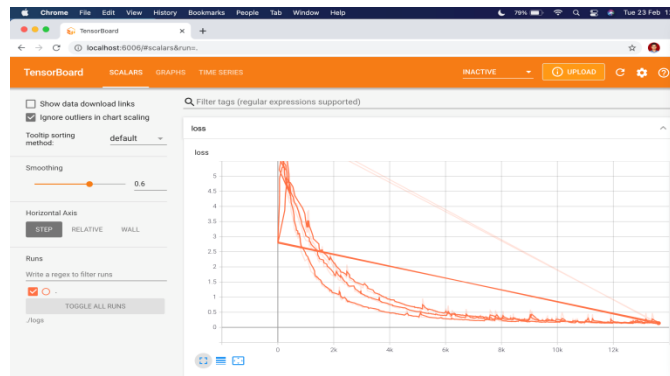


Fig. 10. Tensorboard

Using TensorFlow we can use the feature of tensorboard where we can see the network and the number of nodes created for the model and also the number of epoch and loss value. TensorBoard is the interface used to visualize the graph and other tools to understand, debug, and optimize the model. It is a tool that provides measurements and visualizations for machine learning workflow. It helps to track metrics like loss and accuracy, model graph visualization, project embedding at lower-dimensional spaces, etc.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

Convolution neural network align the steering wheel according to the value given by model. This approach only needs less amount of training data from humans, the system learns from that minimum training data and finally produces the accurate steering angle. Used convolutional neural networks (CNNs) to map the raw pixels from a front-facing camera to the steering commands for a self-driving car. This powerful end-to-end approach means that with minimum training data from humans, the system learns to steer, with or without lane markings, on both local roads and highways. The system can also operate in areas with unclear visual guidance such as parking lots or unpaved roads.

5.2 Future Work

In examining the final leader board from Udacity our models would have placed fourth (transfer learning model) and tenth (3D convolutional model with LSTM layers). These results were produced solely from the models without any external smoothing function. We have shown that both transfer learning and a more advanced architecture have promise in the field of autonomous vehicles. The 3D model was limited by computational resources, but overall it still provided a good result from a novel architecture. In future work the 3D model's architecture could be expanded by having a larger and deeper layers, which may produce better results. These models are far from perfect and there is substantial research that still needs to be done on the subject before models like these can be deployed widely to transport the public. These models may benefit from a wider range of training data. For a production system, a model would have to be able to handle the environment in snowy conditions. Generate adversarial models, GANs, could be used to transform a summer training set into a winter one. Additionally, GANs could be used to generate more scenes with sharp angles. Additionally, a high quality simulator could be used with deep reinforcement learning. A

potential reward function could be getting from one point to another while minimizing time, maximizing smoothness of the ride, staying in the correct lane/following the rules of the road, and not hitting objects. Recently, the technological advancements have led to the adoption of technology in the automobile sector. There has been a constant improvement in the design and development of self driving cars. The primary goal of self driving car is to make the vehicle move in the correct lane. So, lane detection is one of the preliminary steps to make this possible. Existing literatures have suggested various schemes and methods to implement and improve the efficiency of the self driving cars. In this paper, a deep neural network namely the CNN is used as a model.

CHAPTER 6

REFERENCE

- [1] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316, 2016.
- [2] M. Bojarski, P. Yeres, A. Choromanska, K. Choromanski, B. Firner, L. Jackel, and U. Muller. Explaining how a deep neural network trained with end-to-end learning steers a car. arXiv preprint arXiv:1704.07911, 2017.
- [3] A. El Sallab, M. Abdou, E. Perot, and S. Yogamani. Deep reinforcement learning framework for autonomous driving. *Autonomous Vehicles and Machines, Electronic Imaging*, 2017.
- [4] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Learning hierarchical features for scene labeling. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1915– 1929, 2013.
- [5] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.
- [6] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.
- [7] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten. Densely connected convolutional networks. arXiv preprint arXiv:1608.06993, 2016.
- [8] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167, 2015.

- [9] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In Proceedings of the IEEE conference on Computer Vision and Pattern Recognition, pages 1725–1732, 2014.
- [10] D. Kingma and J. Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.
- [11] D. P. Kingma and M. Welling. Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114, 2013.
- [12] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602, 2013.
- [13] D. A. Pomerleau. Alvin, an autonomous land vehicle in a neural network. Technical report, Carnegie Mellon University, Computer Science Department, 1989.
- [14] E. Santana and G. Hotz. Learning a driving simulator. arXiv preprint arXiv:1608.01230, 2016.
- [15] S. Shalev-Shwartz, S. Shammah, and A. Shashua. Safe, multi-agent, reinforcement learning for autonomous driving. arXiv preprint arXiv:1610.03295, 2016.
- [16] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.
- [17] C. Szegedy, A. Toshev, and D. Erhan. Deep neural networks for object detection. In Advances in Neural Information Processing Systems, pages 2553–2561, 2013.
- [18] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Learning spatiotemporal features with 3d convolutional networks. In The IEEE International Conference on Computer Vision (ICCV), December 2015.
- [19] Udacity. An open source self-driving car, 2017.

CHAPTER 7

APPENDICES

7.1 Source Code

//Training the Model

```
def weight_variable(shape):
```

```
    initial = tf.truncated_normal(shape, stddev=0.1)
```

```
    return tf.Variable(initial)
```

```
def bias_variable(shape):
```

```
    initial = tf.constant(0.1, shape=shape)
```

```
    return tf.Variable(initial)
```

```
def conv2d(x, W, stride):
```

```
    return tf.nn.conv2d(x, W, strides=[1, stride, stride, 1], padding='VALID')
```

```
x = tf.placeholder(tf.float32, shape=[None, 66, 200, 3])
```

```
y_ = tf.placeholder(tf.float32, shape=[None, 1])
```

```
x_image = x
```

#first convolutional layer

W_conv1 = weight_variable([5, 5, 3, 24])

b_conv1 = bias_variable([24])

h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1, 2) + b_conv1)

#second convolutional layer

W_conv2 = weight_variable([5, 5, 24, 36])

b_conv2 = bias_variable([36])

h_conv2 = tf.nn.relu(conv2d(h_conv1, W_conv2, 2) + b_conv2)

#third convolutional layer

W_conv3 = weight_variable([5, 5, 36, 48])

b_conv3 = bias_variable([48])

h_conv3 = tf.nn.relu(conv2d(h_conv2, W_conv3, 2) + b_conv3)

#fourth convolutional layer

W_conv4 = weight_variable([3, 3, 48, 64])

b_conv4 = bias_variable([64])

```
h_conv4 = tf.nn.relu(conv2d(h_conv3, W_conv4, 1) + b_conv4)
```

```
#fifth convolutional layer
```

```
W_conv5 = weight_variable([3, 3, 64, 64])
```

```
b_conv5 = bias_variable([64])
```

```
h_conv5 = tf.nn.relu(conv2d(h_conv4, W_conv5, 1) + b_conv5)
```

```
#FCL 1
```

```
W_fc1 = weight_variable([1152, 1164])
```

```
b_fc1 = bias_variable([1164])
```

```
h_conv5_flat = tf.reshape(h_conv5, [-1, 1152])
```

```
h_fc1 = tf.nn.relu(tf.matmul(h_conv5_flat, W_fc1) + b_fc1)
```

```
keep_prob = tf.placeholder(tf.float32)
```

```
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
```

```
#FCL 2
```

```
W_fc2 = weight_variable([1164, 100])
```

```
b_fc2 = bias_variable([100])
```

```
h_fc2 = tf.nn.relu(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)
```

```
h_fc2_drop = tf.nn.dropout(h_fc2, keep_prob)
```

```
#FCL 3
```

```
W_fc3 = weight_variable([100, 50])
```

```
b_fc3 = bias_variable([50])
```

```
h_fc3 = tf.nn.relu(tf.matmul(h_fc2_drop, W_fc3) + b_fc3)
```

```
h_fc3_drop = tf.nn.dropout(h_fc3, keep_prob)
```

```
#FCL 3
```

```
W_fc4 = weight_variable([50, 10])
```

```
b_fc4 = bias_variable([10])
```

```
h_fc4 = tf.nn.relu(tf.matmul(h_fc3_drop, W_fc4) + b_fc4)
```

```
h_fc4_drop = tf.nn.dropout(h_fc4, keep_prob)
```

```

#Output

W_fc5 = weight_variable([10, 1])

b_fc5 = bias_variable([1])


#linear

#y = tf.multiply((tf.matmul(h_fc4_drop, W_fc5) + b_fc5), 2)

## atan

y = tf.multiply(tf.atan(tf.matmul(h_fc4_drop, W_fc5) + b_fc5), 2)

//Running the Model

sess = tf.InteractiveSession()

saver = tf.train.Saver()

saver.restore(sess, "save/model.ckpt")


img = cv2.imread('steering_wheel_image.jpg',0)

rows,cols = img.shape


smoothed_angle = 0


cap = cv2.VideoCapture(0)

while(cv2.waitKey(10) != ord('q')):

```

```

ret, frame = cap.read()

image = scipy.misc.imresize(frame, [66, 200]) / 255.0

degrees = model.y.eval(feed_dict={model.x: [image], model.keep_prob: 1.0})[0][0] *
180 / scipy.pi

call("clear")

print("Predicted steering angle: " + str(degrees) + " degrees")

cv2.imshow('frame', frame)

#make smooth angle transitions by turning the steering wheel based on the
difference of the current angle

#and the predicted angle

smoothed_angle += 0.2 * pow(abs((degrees - smoothed_angle)), 2.0 / 3.0) * (degrees
- smoothed_angle) / abs(degrees - smoothed_angle)

M = cv2.getRotationMatrix2D((cols/2,rows/2),-smoothed_angle,1)

dst = cv2.warpAffine(img,M,(cols,rows))

cv2.imshow("steering wheel", dst)

cap.release()

cv2.destroyAllWindows()

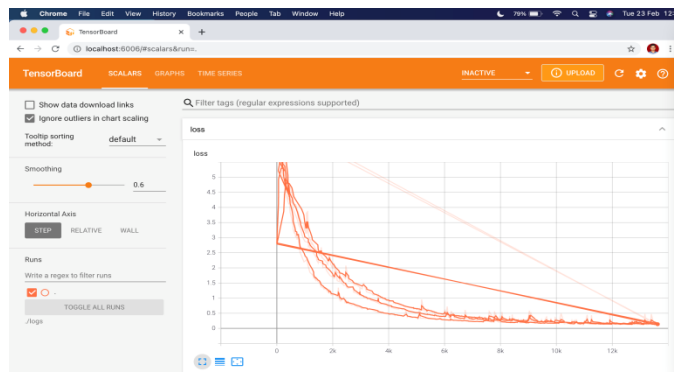
```

7.2 SCREENSHOT

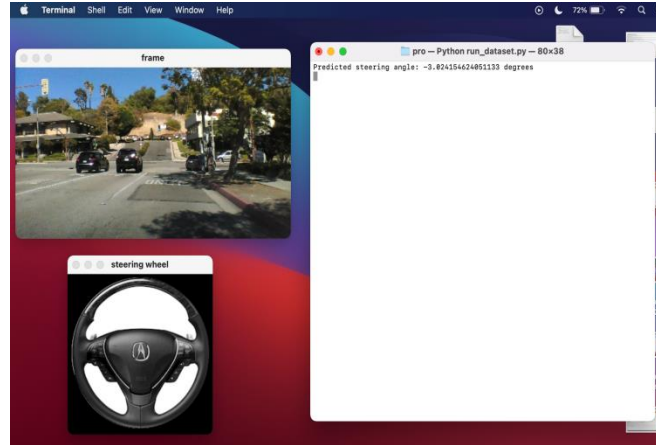
1. Training the Model

```
Epoch: 29, Step: 3120, Loss: 0.143451
Epoch: 29, Step: 3130, Loss: 0.224523
Epoch: 29, Step: 3140, Loss: 0.141103
Epoch: 29, Step: 3150, Loss: 0.138307
Epoch: 29, Step: 3160, Loss: 0.141191
Epoch: 29, Step: 3170, Loss: 0.145083
Epoch: 29, Step: 3180, Loss: 0.151027
Epoch: 29, Step: 3190, Loss: 0.14036
Epoch: 29, Step: 3200, Loss: 0.143089
Epoch: 29, Step: 3210, Loss: 0.155034
Epoch: 29, Step: 3220, Loss: 0.14069
Epoch: 29, Step: 3230, Loss: 0.144165
Epoch: 29, Step: 3240, Loss: 0.472037
Epoch: 29, Step: 3250, Loss: 0.142371
Epoch: 29, Step: 3260, Loss: 0.178216
Epoch: 29, Step: 3270, Loss: 0.138012
Epoch: 29, Step: 3280, Loss: 0.143052
Epoch: 29, Step: 3290, Loss: 0.135788
Epoch: 29, Step: 3300, Loss: 0.21534
Epoch: 29, Step: 3310, Loss: 0.141223
Epoch: 29, Step: 3320, Loss: 0.141223
Epoch: 29, Step: 3330, Loss: 0.137741
Epoch: 29, Step: 3340, Loss: 0.134547
Epoch: 29, Step: 3350, Loss: 0.142568
Model saved in file: ./save/model.ckpt
Run the command line:
--> tensorboard --logdir=./logs
Then open http://0.0.0.0:6006/ into your web browser
akashnandagopal@Akashs-MacBook-Pro pro %
```

2. Tensorboard



3. Running model through data set



4. Running the model through live web cam feed

