

Deep Learning-Assisted Autonomous Vehicle

Submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology in **Electronics & Instrumentation Engineering**

by

PRATYUSH BHATTACHARYA

16BEI0088

Under the guidance of

Dr. Balaji S

School of Electrical Engineering,

VIT, Vellore.



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

May 2020

DECLARATION

I hereby declare that the thesis entitled “**Deep Learning Assisted Autonomous Vehicle**” submitted by me, for the award of the degree of **Bachelor of Technology in Electronics and Instrumentation Engineering** to VIT is a record of bonafide work carried out by me under the supervision of **Dr. Balaji S.**

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

Place : Vellore

Date : 13/05/2020



Pratyush Bhattacharya

Signature of the Candidate

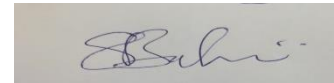
CERTIFICATE

This is to certify that the thesis entitled “**Deep Learning Assisted Autonomous Vehicle**” submitted by **Pratyush Bhattacharya 16BEI0088**, School of Electrical Engineering, VIT, Vellore, for the award of the degree of **Bachelor of Technology in Electronics and Instrumentation Engineering**, is a record of bonafide work carried out by him under my supervision, as per the VIT code of academic and research ethics.

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

Place : Vellore

Date : 13/05/2020



Signature of the Guide

The thesis is satisfactory / unsatisfactory

Internal Examiner

External Examiner

Approved by
Vivekanandan

Head of the Department

Department of Instrumentation Engineering

ACKNOWLEDGEMENTS

The capstone project opportunity I had with Vellore Institute of Technology, Vellore was a great opportunity for personal growth and career building. I consider myself blessed for having got such a wonderful opportunity and would like to express my gratitude first and foremost to VIT Vellore, for having given me the tools required to understand the work during my project. It is incumbent on me to be forever grateful to the esteemed teaching faculties of VIT for imparting such unparalleled knowledge.

I express my deepest thanks to Dr. Balaji S, my Project Guide, for taking time out to ensure that I had a seamless understanding of the outcomes of my project by giving me insightful advice to make my project more industry apt and realistic.

I would like to place on record my best regards and deepest sense of gratitude to Mr. Vivekananda S., Head Of Department EIE, for his priceless guidance, which turned out to be highly valuable for my capstone project.

I am confident that this opportunity is a big milestone and a stepping-stone for my future career as it gave me an in-depth glimpse into the knowledge and skills required in the real world. I wish to collaborate and continue my co-operation with all of you in the future.

Sincerely

Pratyush Bhattacharya

Executive Summary

The project focuses on the design and development of a level-2/level-2+ autonomous driving model that can successfully navigate, drive itself and perform feature recognition. To facilitate testing, the project involves building a simulator, where real-world conditions can be emulated. Convolutional neural networks (CNN) are used here to associate raw pixels (collected from the training images) from three front-facing cameras directly to steering commands. This approach is particularly utilitarian, as in most real-world scenarios the images will be collected in real time and it would be expected to get an immediate driving signal response. One of the foremost reasons for using CNNs are that, they have revolutionized pattern recognition. Before the advent of CNN architectures, each use-case had to build their own feature extraction modules followed by a classifier. The novelty of CNNs are that they learn a required set of features directly from the training images. As most images captured are 2-D in nature, the convolutional operation helps to capture that nature.

In this project, there will be a self-built architecture of a CNN along with the environment which will emulate a car and it's immediate surroundings. For the training set of images, the car will be driven by a human driver on a certain road, whereas for the testing of the model, the car will be driven on a completely different track. Due to the high learning capability of the CNN models, many further features which may not be explicitly mentioned by the developer, will also be learned. Once the network is trained, we can observe the simulator car drive autonomously, without any user assistance, due to the generated steering commands from the trained model's output. After addressing the technical details of the project, the design approaches and constraints will also be addressed. Apart from that, there will also be a review of the existing codes and standards present for self-driving cars. Finally, the cost analysis between our proposed model and existing model will be compared and analyzed.

	<u>CONTENTS</u>	<u>PAGE NO.</u>
	ACKNOWLEDGEMENTS	I
	EXECUTIVE SUMMARY	II
	TABLE OF CONTENTS	III
	LIST OF FIGURES	IV
	LIST OF TABLES	VI
	ABBREVIATIONS	VI
1.	INTRODUCTION	1
	1.1 Objective	1
	1.2 Motivation	3
	1.3 Background	4
2.	PROJECT DESCRIPTION AND GOALS	9
3.	TECHNICAL SPECIFICATION	11
4.	DESIGN APPROACH AND DETAILS	30
	4.1 Design Approach	30
	4.2 Codes and Standards	38
	4.3 Constraints, Alternatives and Trade-offs	39
5.	SCHEDULE, TASKS AND MILESTONES	41
6.	PROJECT DEMONSTRATION	43
7.	COST ANALYSIS	48
8.	SUMMARY	49
9.	REFERENCES	50

List of Figures

Figure No.	Title	Page No.
1.	Traditional pattern recognition was performed with two modules: a fixed feature extractor, and a trainable classifier	11
2.	Pixel representation of B/W image and Colored image	12
3.	Basic representation of a B/W image in 0's and 1's	12
4.	Steps followed by a CNN	13
5.	Here w^i indicates the direction in which the loss function moves. The goal of the SGD is to reach the minima in as few steps as possible	15
6.	A three layer network showing the back-propagation method	16
7.	Architecture of one of the very first CNNs used for digit recognition	18
8.	A 2x2 convolutional kernel	18
9.	The convolution input and output	18
10.	Demonstration of a feature map from an input image	19
11.	Representation of different feature maps populated to detect certain features	20
12.	ReLU layer applied to the feature maps after convolution	22
13.	Leaky ReLU and Parameterized ReLU	23
14.	Pictures of cheetahs with their characteristic features in different locations of the image	23
15.	Max pooling applied to the feature map obtained from the convolutional layer	24
16.	Reached the pooling step of the entire CNN	25
17.	Flattening the pooled matrix of features into a column vector	25
18.	Full connection after the flattening step	26
19.	Example of the training of a network to classify dogs and cats	27
20.	Trained network with values	27
21.	Summary of how CNNs work	29

22.	Unity Development environment	33
23.	Designing the 'jungle' track	33
24.	Aerial view of the jungle track	34
25.	Lake track development	34
26.	Lake track aerial view	35
27.	Frontal view of the car	35
28.	Side-view of the car	36
29.	Back-view of the car	36
30.	Client-server model for the self-driving car	37
31.	Project schedule timeline	42
32.	Training data folder. We can see in the lower left corner, there are 15,555 images in this training folder	43
33.	Code snippet for the model	44
34.	Running the drive.py file with the model.h5 file to start the client-server connection	45
35.	Final build folder containing the exe for the self-driving car	45
36.	First dialogue box after double clicking on the exe file	46
37.	Menu screen for selecting the mode and the track	46
38.	As visible, the car is driving itself autonomously on the 'jungle' track!	47

List of Tables

Table No.	Title	Page No.
1.	Convolutional research model developed	31

List of Abbreviations

CNN	Convolutional Neural Network
ANN	Artificial Neural Network
BPN	Back Propagation Network
SAE	Society of Automotive Engineers
DARPA	Defense Advanced Research Projects Agency
AI	Artificial Intelligence
HCAI	Human-Centered Artificial Intelligence
CES	Consumer Electronics Show
DL	Deep Learning
B/W	Black and White
RGB	Red Green Blue
MSE	Mean Squared Error
SGD	Stochastic Gradient Descent
ReLU	Rectified Linear Unit
PReLU	Parameterized Rectified Linear Unit
HAV	Highly Autonomous Vehicles
ISO	International Standards Organization
ASIL	Automotive Safety Integrity Level
IEC	International Electrotechnical Commission
PAS	Publicly Available Specification
SOTIF	Safety of the Intended Functionality
ADAS	Advanced Driver Assistance Systems
CENELEC	European Committee for Electrotechnical Standardization
ARP	Address Resolution Protocol
MIL-STD	United States Military Standard
COLAB	Colaboratory
HDF	Hierarchical Data Format

EXE	Executable
LIDAR	Light Detection and Ranging
RADAR	Radio Detection and Ranging

1. INTRODUCTION

In this project, there will be a self-built architecture of a CNN along with the environment which will emulate a car and its immediate surroundings. For the training set of images, the car will be driven by a human driver on a certain road, whereas for the testing of the model, the car will be driven on a completely different track. Due to the high learning capability of the CNN models, many further features which may not be explicitly mentioned by the developer, will also be learned. Once the network is trained, we can observe the simulator car drive autonomously, without any user assistance, due to the generated steering commands from the trained model's output. After addressing the technical details of the project, the design approaches and constraints will also be addressed. Apart from that, there will also be a review of the existing codes and standards present for self-driving cars. Finally, the cost analysis between our proposed model and existing model will be compared and analyzed.

1.1 Objective

A self-driving car, also known as an autonomous vehicle, is a vehicle that is capable of sensing its environment and moving safely with little or no human input. SAE International, an automotive agency has released a classification system for the segregation of automation levels (ranging from fully manual to fully automated systems). This classification system is unique as it focuses on the amount of driver intervention and attentiveness required, rather than the vehicle capabilities. In SAE's automation level definitions, "driving mode" means "a type of driving scenario with characteristic dynamic driving task requirements (e.g., expressway merging, high speed cruising, low speed traffic jam, closed-campus operations, etc.)". The automation levels are:

- Level 0: Automated system sends warnings to the user and may intervene for a moment, but cannot sustain vehicle control.
- Level 1 ("hands on"): The driver and the automation framework share control of the vehicle. The driver must be prepared to retake full control whenever.
- Level 2 ("hands off"): The automation framework assumes complete responsibility of the vehicle's actions (accelerating, braking, and steering); though the driver must

monitor the driving and be prepared to mitigate any failures immediately if the automation system fails.

- Level 3 ("eyes off"): The driver can safely dismiss attention away from the driving tasks, e.g. the driver can take a call or watch an entertainment feature. The vehicle is equipped to handle automation failures by triggering an immediate response, like emergency braking.
- Level 4 ("mind off"): Same as level 3, but driver attention can be completely foregone and the automation system is expected to completely handle the safety of the passengers. E.g. the driver may all together leave the driver's seat or even go to sleep. Self-driving is supported only in limited spatial areas (geo-fenced).
- Level 5 ("steering wheel optional"): No human intervention is required at all. A Robotic taxi is a perfect example for such a system.

The project focuses on the design and development of a level-2/level-2+ autonomous driving model that can successfully navigate, drive itself and perform feature recognition. To facilitate testing, the project involves building a simulator, where real-world conditions can be emulated. We use convolutional neural networks (CNN) to map raw pixels (collected from the training images) from three front-facing cameras directly to steering commands. This approach is particularly utilitarian, as in most real-world scenarios the images will be collected in real time and it would be expected to get an immediate driving signal response. One of the foremost reasons for using CNNs are that, they have revolutionized pattern recognition. Before the advent of CNN architectures, each use-case had to build their own feature extraction modules followed by a classifier. The novelty of CNNs are that they learn a required set of features directly from the training images. As most images captured are 2-D in nature, the convolutional operation helps to capture that nature.

The objectives of the project can be succinctly summarized as follows:-

- Develop a reliable simulator to emulate real-world conditions in which a car will be driving.
- Use the Unity simulator to collect training data of good driving behaviour.
- Design and build, a convolution neural network using a Python framework that will predict steering angles from images.

- Train and validate the model with a training and validation set.
- Test that the model successfully drives around the track autonomously without leaving the road.

1.2 Motivation

Autonomous vehicles are the technology of the future. The needs for innovation and improvement in driving technology have been in development since 1939, when General Motors created their first autonomous vehicle prototype guided by radio-controlled electromagnetic fields generated with magnetized metal spikes embedded in the roadway. Once that was showcased to the world, there was no looking back. The world's top automation teams got together to convert 'self-driving cars' from a science fiction concept to a reality in a realizable time frame.

Currently, until 2020 the most advanced car in the market has reached an SAE automation level 3. While Google's Waymo claims to have reached SAE automation level 4/4+, it is only limited to a certain district in a city and it is not capable of driving itself anywhere outside of that region.

The main motivation for this project is to get involved and start development in this technology, which is going to 'drive' the future. Self-driving technology is the next biggest thing of the future. All the companies and investments will be for this technology. Self-driving cars will become the new norm as they will ultimately help fuel productivity, reliability, alleviate stress and improve mobility.

Productivity of a person travelling in an autonomous vehicle will increase manifold. No longer will people have to complain about quotidian tasks such as driving to-and-fro their work place, being stuck in traffic jams, losing out on work hours due to traveling. Most corporate companies will endorse the use of autonomous cars as they will indirectly help improve their work output, thereby improving profits.

Reliability of self driving cars is another aspect which will have significant gains on humans as a whole. Once the entire fleet of cars on the streets are of self-driving cars, we can be assured that accidents will reduce exponentially. Since all the vehicles on the road will be driving autonomously, they can have remote, secure communication with all the

other vehicles to decide on routes and traffic conditions. Reliability is one of the main aspects of autonomous vehicles and until that is improved, we need to work on improving the technology.

Reduction in stress of a human while driving is one of the main motivations of building a self-driving car. We have all experienced varying levels of fatigue and stress when going on long drives. Especially when driving cross-country, we have to be even more vigilant as we have to keep a look-out for other cars and also because we are on unfamiliar roads. With the advent of fully autonomous vehicles, we can hope to relieve this stress component completely. The cars will take care of the terrain, speed, other vehicles and will take the required decisions; providing the passengers in the car with comfort and thus the passengers can reach their destination comfortable and refreshed.

Finally, mobility in a self-driving car will be improved. The human will no longer have to worry about parking in tight spaces, or whether the car can travel along a certain stretch of the road. The car will take care of all these parameters and will take the decisions, thereby improving mobility of the vehicle. It is also likely that self-driving cars will have better fuel mileages as they can calculate the optimum speed to drive at to have the lowest fuel consumption while not compromising on travel times and comfort.

1.3 Background

The possibility that individuals are poor drivers is very much archived in mainstream society. While this thought is regularly over-performed, there is some fact to it in that we are on occasion diverted, tired, alcoholic, medicated, and nonsensical chiefs. Be that as it may, this doesn't mean it is anything but difficult to plan and manufacture a discernment control framework that drives superior to the normal human driver. The 2007 DARPA Urban Challenge was a landmark accomplishment in apply autonomy, when 6 of the 11 self-sufficient vehicles in the finals effectively explored a urban domain to come to the finish line, with the first place finisher going at a normal speed of 15 mph. The accomplishment of this opposition drove numerous to proclaim the completely self-governing driving undertaking a "tackled issue", one with just a couple of staying muddled subtleties to be settled via automakers as a component of conveying a business item. Today,

more than ten years after the fact, the issues of confinement, mapping, scene observation, vehicle control, direction advancement, and more significant level arranging choices related with self-governing vehicle improvement stay brimming with open difficulties that presently can't seem to be completely understood by frameworks fused into a creation stages (for example offered available to be purchased) for even a confined operational space. The testing of model vehicles with a human director liable for taking control during periods where the AI framework is "uncertain" or unfit to securely continue remains the standard. The DARPA Urban Challenge was just a first step down a lengthy, difficult experience toward creating self-ruling vehicle frameworks. The Urban Challenge had no individuals taking an interest in the situation with the exception of the expert drivers controlling the other 30 vehicles out and about that day. The creators accept that the present genuine test is one that has the person as a necessary piece of each part of the framework.

This test is made particularly difficult because of the massive fluctuation inborn to the driving assignment because of the accompanying elements:

- The basic vulnerability of human conduct as spoke to by each kind of social association and conflict goals between vehicles, walkers, and cyclists.
- The inconstancy between driver styles, understanding, and different qualities that add to their understanding, trust, and utilization of automation.
- The complexities and edge-instances of the scene recognition and understanding issue.
- The under impelled nature of the control issue for each human-tuned in mechanical framework in the vehicle: from the driver communication with the steering wheel to the tires reaching the street surface.
- The normal and unexpected confinement of and flaws in the sensors.
- The reliance on software with all the challenges inherent to software-based systems: bugs, vulnerabilities, and the constantly changing feature set from minor and major version updates.

As people, we normally underestimate how much knowledge, in the mechanical autonomy feeling of the word, is required to effectively achieve enough circumstance mindfulness and understanding to explore through a world loaded with typically unreasonable

individuals moving about in vehicles, on bicycles, and by walking. It might be a long time before most of vehicles out and about are completely self-governing. During this time, the human is probably going to remain the basic leader either as the driver or as the manager of the AI framework doing the driving. In this unique situation, Human-Centered Artificial Intelligence (HCAI) is a territory of software engineering, mechanical technology, and experience plan that intends to accomplish a more profound mix among human and artificial knowledge. Almost certainly, HCAI will assume a basic job in the arrangement of advances (algorithms, sensors, interfaces, and interaction paradigms) that help the driver's job in checking the AI framework as it performs anyplace from simply over 0% to just shy of 100% of the fundamental driving and higher request item and occasion discovery errands

Following are some of the milestones in the history of the development of autonomous vehicles:

- It didn't take long after the introduction of the motorcar for designers to begin considering autonomous vehicles. In 1925, the designer Francis Houdina shows a radio-controlled vehicle, which he passes through the boulevards of Manhattan without anybody at the steering wheel. As per the New York Times, the radio-controlled vehicle can turn over its motor, change gears, and sound its horn, “as if a phantom hand were at the wheel.”
- In 1969, John McCarthy — a.k.a. one of the establishing fathers of computerized reasoning — depicts something like the cutting edge self-ruling vehicle in a paper titled "Computer-Controlled Cars." McCarthy alludes to a "automatic chauffeur," equipped for exploring an open street by means of a "television camera input that uses the same visual input available to the human driver." He composes that clients ought to have the option to enter a goal utilizing a console, which would provoke the vehicle to quickly drive them there. Extra commands permit clients to change driving destination, stop at a rest room or café, slow down, or accelerate on account of a crisis. No such vehicle is fabricated, yet McCarthy's paper spreads out the crucial differences for analysts to move in the direction of.
- In the mid 1990s, Carnegie Mellon scientist Dean Pomerleau composes a PhD thesis, portraying how neural systems could permit a self-driving vehicle to take in raw pictures from the road and yield steering controls continuously. Pomerleau isn't the only analyst working on self-driving vehicles, yet his utilization of neural nets

demonstrates far more effective than elective endeavors to physically isolate pictures into "road" and "non-road" classifications. In 1995, Pomerleau and individual scientist Todd Jochem take their Navlab self-driving vehicle framework on the road. Their basic autonomous minivan (they need to control speed and braking) makes a trip 2,797 miles across the nation from Pittsburgh, Pennsylvania to San Diego, California in an excursion the pair names "No Hands Across America."

- In 2002, DARPA announces its Grand Challenge, offering specialists from top research establishments a \$1 million prize in the event that they can construct an autonomous vehicle ready to explore 142 miles through the Mojave Desert. At the point when the test commences in 2004, none of the 15 contenders can finish the course. The "winning" section makes it less than eight miles in a few hours, before bursting into flames. It's a damaging blow to the objective of building genuine self-driving vehicles.
- While autonomous vehicles despite everything appear to be path later on in the time of the 2000s, self-leaving frameworks start to develop — exhibiting that sensors and autonomous road technologies are preparing near for certifiable situations. Toyota's Japanese Prius hybrid vehicle offers parallel parking assistance from 2003, while Lexus before long includes a automatic parallel parking for its Lexus LS car, Ford incorporates Active Park Assist in 2009, and BMW tails one year later with its own parallel parking assistant.
- Beginning in 2009, Google starts building up its self-driving vehicle venture, now called Waymo, stealthily. The task is at initially driven by Sebastian Thrun, the previous executive of the Stanford Artificial Intelligence Laboratory and co-creator of Google Street View. Within a couple of years, Google announces that its autonomous vehicles have aggregately traveled 300,000 miles under autonomous control without one single mishap happening. In 2014, it uncovers a model of a driverless vehicle with no steering wheel, gas pedal or brake pedal, subsequently being 100 percent autonomous. By the end of last year, in excess of 2 million miles had been driven by Google's autonomous vehicle.
- By 2013, significant car organizations including General Motors, Ford, Mercedes Benz, BMW, and others are overall taking a shot at their own self-driving vehicle

technologies. Nissan focuses on a launch date by announcing that it will discharge several driverless vehicles by the year 2020. Different vehicles, for example, the 2014 Mercedes S-Class, include semi-autonomous features, for example, self-steering, the capacity to remain inside paths, mishap avoidance, and more.

- Audi claims its cutting edge A8 luxury sedan car will be it's primary production vehicle with SAE Level 3 autonomy. The A8's Traffic Jam Pilot permits the vehicle to drive itself with no human intervention, yet just under specific conditions. The framework just works in traffic at speeds up to 37 mph, in separated highways with clearly stamped entrance and leave paths.
- At CES 2018, NVidia uncovered another self-driving vehicle chip called Xavier that will join artificial-intelligence abilities. The organization at that point announced that it was collaborating with Volkswagen to create AI for future self-driving vehicles. While not the main exertion to saturate autonomous vehicles with AI (Toyota was at that point already looking into the idea with MIT and Stanford), the VW-Nvidia joint effort is the first to associate AI to production ready hardware. It opens up the opportunities for self-driving vehicles to perform better, just as for new convenience highlights like digital assistants.

2. PROJECT DESCRIPTION AND GOALS

The project is titled 'Deep Learning-Assisted Autonomous Vehicle'. The main objective of this project is to develop a reliable self-driving car model, which can be extrapolated to a real system. This project aims to do this by building a simulator, which can emulate real-world conditions, and have the car drive on it. For the car to 'learn' how to drive, it would have to collect training data. This would be facilitated by the user driving the car as 'humanly' as possible. This would teach the car many intricacies of human driving, such as maintaining a straight line while driving, speeding up on a straight stretch of the road, slowing down on bends, braking before obstacles etc. The car would see and collect this data from 3 cameras attached to its bonnet. The three cameras would face left, center and right respectively. This would give it a 180-degree field of view of the car's vision, which is enough to get reliable driving data and 'see' the surroundings. The rest of the parameters such as speed of the vehicle, angle of the wheels, braking etc., are collected from the car's data which is fed into an excel sheet which records all the data.

Once the simulator is placed in training mode, the user drives the car around the track. These tracks consist of a variety of real-world bumpy roads, smooth roads, extreme sharp bends etc. The cameras attached to the front-bumper of the car collect images at a pre-defined frame rate. In addition, the auxiliary car data is written to an excel sheet, so it can be read by a program to process the data.

After the training data is collected, the data is fed to a neural network model to train. This is the 'deep-learning' part of the project. Here, a very deep neural network model is used to process out the data from the images. Here is where the beauty of deep learning is observed. It utilizes the complex and almost impeccable feature-recognition capability of the human mind. When we humans look at someone and then meet the person after a considerable amount of time, we are able to recognize them almost immediately. While we may be able to recognize them almost immediately, we don't realize the intricate and complex methodology of feature recognition employed by our human brain to recognize the person. For example, the brain may recognize features such as jawbones, cheekbone depth, eye color, hair color, face contour etc. But we only remember the 'face' and not the intricate features which our brain learns by recognizing these features. Similarly, when the

artificial neural network is given the images of the road to train on, it employs the brain's feature recognition capabilities to extract many such features that are not understandable to us as humans. For example, it may employ grain count to determine the roughness of the road. Many other important features, which we never explicitly trained it to learn, are also learnt by the system. Things like lane-detection, outline of roads etc. are learnt by the system through its feature recognition. These are employed along with the other learnt features to make the car drive itself just as a human would.

After the training of the artificial neural network, the tuned parameters of the network are saved to a file. These parameters are fed to the car when the car is put in the autonomous mode. The car passes the images it collects live to the neural network. Since the network is now trained, it accurately calculates the outputs and immediately 'directs' the cars wheels to move in accordance with how human's drive.

The goals of the project can be mentioned as follows:

- Develop the mathematics of the neural network and the depth of the layers.
- Build a reliable simulator to emulate a real car and its immediate surroundings.
- Collect 'good driving behavior' as training data.
- Feed the training data to the network to tune its parameters.
- Test out the model by subjecting the car to a completely new track, on which it has not been trained.
- Have the car drive itself seamlessly with a good autonomy.

3. TECHNICAL DETAILS AND SPECIFICATIONS

We use Convolutional Neural Networks (CNNs) to develop a good training model. The ideology is to map raw pixels from the images collected as training data, directly to steering commands. This is known as an end-to-end approach. CNNs have become a revolutionary tool in pattern recognition. Prior to the advent of CNNs, most feature recognition tasks were done by using case-crafted feature extractors followed by classifiers. CNNs have streamlined the process by directly learning the features from the images themselves. The convolution operation employed by CNNs is especially powerful as it can easily recognise the 2-D nature of the image.

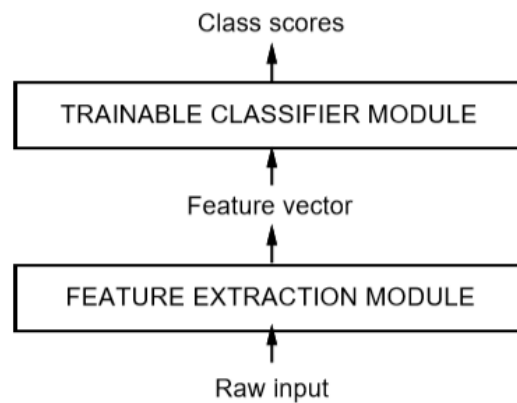


Figure 1. Traditional pattern recognition was performed with two modules: a fixed feature extractor, and a trainable classifier

CNNs work by taking in an input image, and give out an output label. Basically, it classifies that image. In most cases, CNNs work by giving a probability. They give a probability of the output label. For example, if we feed in an image of an animal such as a cheetah, it will give the output as a 98% surety of it being a cheetah, with a 1% chance of it being a leopard and a 1% chance of it being a cat. The entire system of operation of CNNs boil down to features. If the system has not seen enough features, then it will not be able to classify the image with higher accuracy. The neural network recognizes the features at a very basic level. They use the pixels for classifying the images. Pixels are the fundamental blocks of any image. Each pixel represents intensities of that pixel which makes up the image. This gives the computer a way to work with images. It helps to represent the image in a digital form, which is how computers process images.

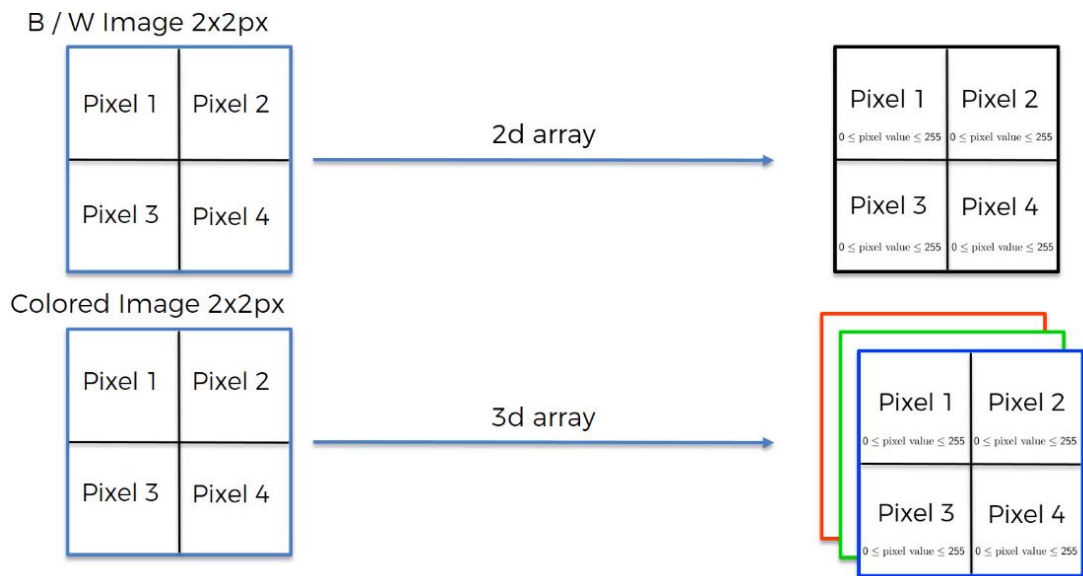


Figure 2. Pixel representation of B/W image and Colored image

A very rudimentary example of classification of pixels would be if white is 0 and black is 1. Then the image would look as follows:

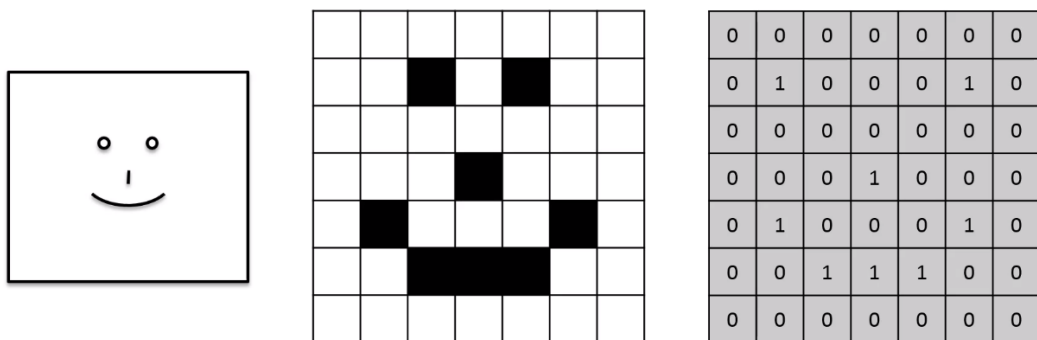


Figure 3. Basic representation of a B/W image in 0's and 1's

All images are fundamentally an array (or matrix) of pixels. All black-and-white pixels consist of a matrix of 0's and 1's. All coloured images consist of 3 channels of RGB (Red Green Blue) with each pixel in each channel varying from 0 to 255. With 0 being pitch black and 255 being completely white.

CNNs work on this array of pixels by extracting data and features from these matrices. All convolutional neural networks go through the steps of convolution, max pooling, flattening and full connection.

A CNN usually inputs a tensor of order 3, e.g., an image with H rows, W columns and 3 channels (R,G,B colour channels). The input is then sequentially processed in a serial manner. Each processing step is called a layer. These layers could be convolution layers, max pooling layers, normalization layers, flattening layers etc.

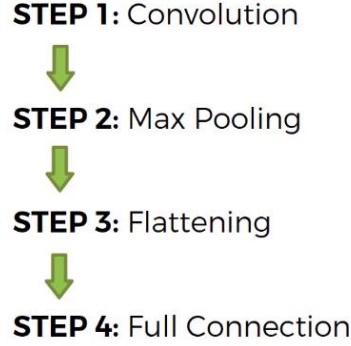


Figure 4. Steps followed by a CNN

Before defining the functions of the above layers, we can give an abstract representation of a CNN.

$$x^1 \longrightarrow \boxed{w^1} \longrightarrow x^2 \longrightarrow \dots \longrightarrow x^{L-1} \longrightarrow \boxed{w^{L-1}} \longrightarrow x^L \longrightarrow \boxed{w^L} \longrightarrow z \quad (1)$$

The above equation demonstrates how a CNN runs layer by layer in a forward pass. The input is denoted by x^1 , usually an image. It goes through processing in the first layer in the first box. The parameters involved in the first layer's processing are collectively stored in a tensor w^1 . The output from the first layer is x^2 , which subsequently acts as an input to the second processing layer. The processing proceeds in this fashion till all layers in the CNN have been traversed. This gives an output x^L . However, one final layer for processing is added which consists of backward-error propagation.

Most classification problems may have a number of classes. If we assume the problem to have C classes, generally we output x^L as a C dimensional vector, whose i^{th} entry contains the prediction values for each class. The last layer is generally a loss calculating layer. Let us suppose ' \mathbf{t} ' is the corresponding target output for the input \mathbf{x}^1 , then a cost function is used to measure the discrepancy between the CNN prediction \mathbf{x}^L , and the target \mathbf{t} . For example, a simple loss function would be the Mean-squared-error function.

$$z = \frac{1}{2} \|\mathbf{t} - \mathbf{x}^L\|^2 \quad (2)$$

However, generally more complex loss functions are used. In case of a regression problem, the squared loss is used, in classification problems cross entropy loss is used.

Suppose all the parameters of a CNN model, w^1, \dots, w^{L-1} have been learned. Now this model is ready for prediction. During prediction, the CNN only makes a forward run. That is, the model only goes forward. Starting from the input x^1 , we make it pass the processing of the first layer, and get x^2 . Subsequently, x^2 is passed to the second layer and so on. The final result is achieved as $x^L \in R^C$, which estimates the probabilities of x^1 belonging to the C categories. The output of the CNN prediction is given as:

$$\arg \max_i x_i^L \quad (3)$$

The loss layer is not needed during prediction and is only used during training of the data. The parameterized values (w^1, \dots, w^{L-1}) of any CNN model need to be optimally found so as to minimize the loss. If we proceed randomly and hope for the loss to be minimized automatically after the training is done, it is highly likely the training will take a computationally infinite amount of time and may not be able to minimize the loss ultimately. To make models more efficient and find the optimum minimal loss, we use mathematical algorithms to reach the minima. Let us assume one such training example, x^1 is taken for training its parameters. The training process involves running the CNN bi-directionally. The forward pass results in an output x^L , which is a prediction of the current CNN parameters. Now we need to compare the prediction with the target value, 't' corresponding to x^1 . Finally after these steps, we achieve the loss 'z'. This value is more like a supervision signal which guides how the parameters should be updated. One of the most popular methods for updating the parameters is using Stochastic Gradient Descent. It is given by:

$$x_{i+1} = x_i + \alpha \nabla f(x_i) \quad (4)$$

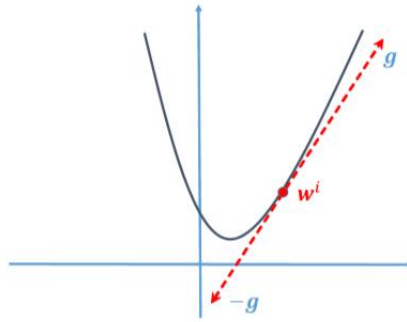


Figure 5. Here w^i indicates the direction in which the loss function moves. The goal of the SGD is to reach the minima in as few steps as possible

The nabla symbol indicates the partial derivative and measures the rate of increase of loss, \mathbf{z} with respect to the changes in different dimensions of \mathbf{w}^i . This partial derivative is called the gradient. The loss is updated in the opposite direction of the gradient so as to minimize it. We have to take care to not move too far from the minima. If upon a subsequent update, we move too far from the minima, then the loss value will keep oscillating. This oscillation may not allow us to reach the minima in finite time. To prevent this from happening, we update the parameters only by a small amount in every iteration. This is done by the learning rate α . Generally $\alpha > 0$ and set to a small number (e.g. $\alpha = 0.001$). Now it is possible that the loss is reduced for one particular training example, but for another example it increases. To get a definite lookout of whether the loss is increasing or decreasing, we need to update the parameters using all the training examples. When all the training examples have been used to process the parameters in one pass, then it is said that one *epoch* is over. Then we repeat these epochs for a large number of times so as to minimize the loss yet not start over-fitting with the data. If we update the parameters using only one training example, we may get a very unstable-oscillating loss function which may not converge to the minimum. To prevent this from happening, we update the parameters using the gradient on a subset of training examples. This is where the name ‘batch gradient descent’ or ‘stochastic gradient descent’ comes from. It is a common standard to set 32 or 64 examples as a mini-batch. In this method, the parameters are updated after getting the gradient from those mini sets.

The method used most accurately for sending the error back to the input is known as ‘back-propagation’. In a back-propagation neural network, the learning algorithm has two phases. In the first phase, a training input pattern is presented to the input layer. The network

propagates the input pattern layer by layer until we reach the final layer. Here the output is compared with the input. If there is a significant difference, the calculated error is propagated backwards through the network from the output to the input layer. The weights are modified as the error is propagated.

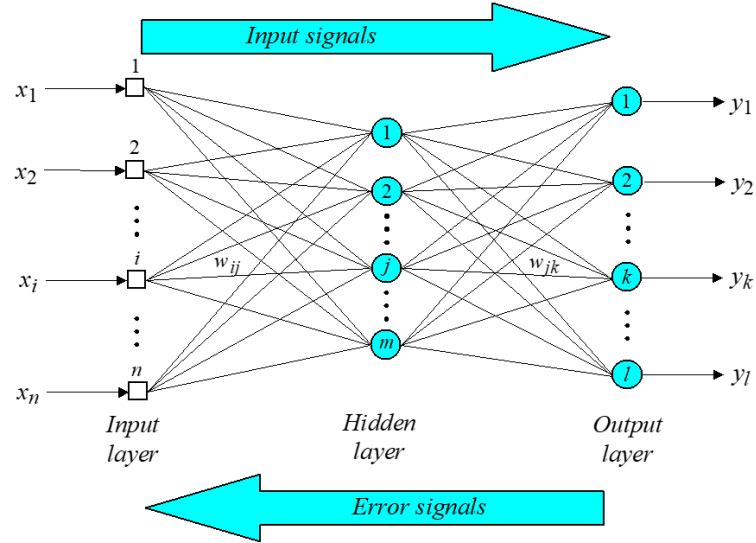


Figure 6. A three layer network showing the back-propagation method

The first step in any network is the initialization of the weights and thresholds to random values distributed uniformly within a certain range. An example of the range of initialization may be $(-2.4/F_i, +2.4/F_i)$. Here F_i is the total number of inputs of neuron 'i' in the network. The weight initialization is done on a neuron-for-neuron basis.

The second step in the algorithm is the activation of the network. In this step, we apply the input $x_1(p), x_2(p), x_3(p), \dots, x_n(p)$ and desired outputs $y_{d,1}(p), y_{d,2}(p), \dots, y_{d,n}(p)$. Then we calculate the actual outputs of the neurons in the hidden layers. In Figure 6, we have only one hidden layer, so we calculate the outputs of that layer. It is given by:

$$y_j(p) = \text{sigmoid} \left[\sum_{i=1}^n x_i(p) \cdot w_{ij}(p) - \theta_j \right] \quad (5)$$

Here, sigmoid is the sigmoid activation function, given by:

$$f(x) = \frac{1}{1 + e^{-\lambda x}} \quad (6)$$

Then we calculate the output of the neurons in the output layer. We use the following formula:

$$y_k(p) = \text{sigmoid} \left[\sum_{j=1}^m x_{jk}(p) \cdot w_{jk}(p) - \theta_k \right] \quad (7)$$

In the third step, we do the weight training. Here we update the weights, which back propagate the errors associated with the output neurons. We calculate the error gradient for the neurons in the output layer as follows:

$$\delta_k(p) = y_k(p) \cdot [1 - y_k(p)] \cdot e_k(p) \quad (8)$$

Where,

$$e_k(p) = y_{d,k}(p) - y_k(p) \quad (9)$$

Next, we calculate the weight corrections given by:

$$\Delta w_{jk}(p) = \alpha \cdot y_j(p) \cdot \delta_k(p) \quad (10)$$

And, then update the weights at the output neurons by:

$$w_{jk}(p+1) = w_{jk}(p) + \Delta w_{jk}(p) \quad (11)$$

Then we calculate the error gradient for the neurons in the hidden layer:

$$\delta_j(p) = y_j(p) \cdot [1 - y_j(p)] \cdot \sum_{k=1}^l \delta_k(p) w_{jk}(p) \quad (12)$$

Get the weight corrections:

$$\Delta w_{ij}(p) = \alpha \cdot x_i(p) \cdot \delta_j(p) \quad (13)$$

And update the weights at the hidden neurons:

$$w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p) \quad (14)$$

The final step is to increase the value of ‘i’ by one and repeat the above steps. This completes the cycle of back-propagation and it is carried out as many times as the number of epochs.

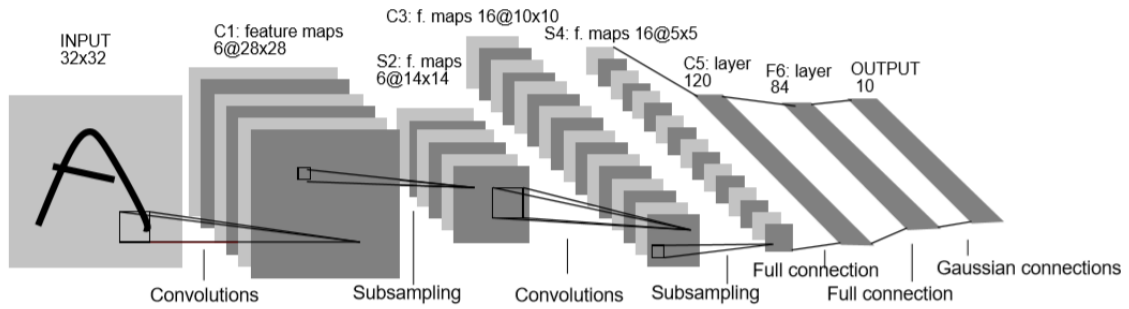


Figure 7. Architecture of one of the very first CNNs used for digit recognition

One of the foremost steps in any CNN is the **convolution step**. Convolution is a mathematical operation, which takes two functions and outputs a third function thereby going to show how the shape of one function is modified by the other.

$$(f * g)(t) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau) g(t - \tau) d\tau \quad (15)$$

As we can see in the above formula, the convolution operation is a combined integration of two functions with limits over infinity, which gives us a function defined by the effects of the functions on each other. In terms of an image, the convolution operation involves taking a convolution kernel (filter, mask etc.) and overlapping it on top of the input image, so as to compute the product between the numbers at the same location in the kernel and the input and get a single output number by summing all the numbers above.

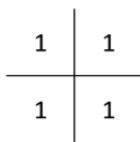


Figure 8. A 2x2 convolutional kernel

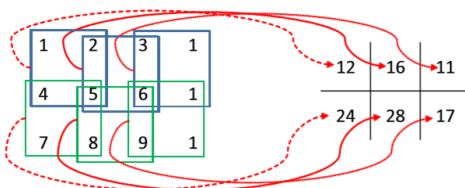


Figure 9. The convolution input and output

In the Figures 8 and 9, we can see the convolution result at the top left portion of the image is given by $1 \times 1 + 1 \times 4 + 1 \times 2 + 1 \times 5 = 12$. Similarly, if the kernel is moved down by one

pixel, we get $1 \times 4 + 1 \times 7 + 1 \times 5 + 1 \times 8 = 24$. We continue this operation till we reach the bottom of the image. The output of the convolutional operation produces a '**Feature Map**'. Different filters give different feature maps which can be very useful in extracting features from an image. There are many different filters such as Sobel filter, Robert's filter, edge detection filter, sharpening filter, smoothing filter etc.

Generally, most use case scenarios employ the 3x3 feature detectors. But some networks such as 'Alex Net' use 7x7 feature detectors. The 'step' at which the filter is moved from pixel to pixel is called the stride. It is the number of pixels shifted by the kernel over the input image. Generally a stride of 2 is used for images. A stride of 2 means the filter will move 2 pixels after each operation. Sometimes the filter does not fit the image completely, and extends out of the image. In this case, we use 'padding' to deal with the issue. We may use zero-padding, which consists of appending zeros to the edges of the image so as to provide meaningful input to the filter.

One of the main uses of the convolution operation is to reduce the size of the image. As the stride of the filter increases, the size of the image decreases.

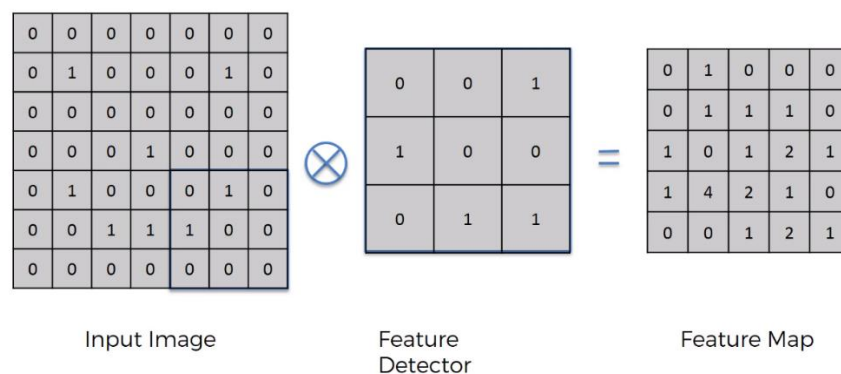


Figure 10. Demonstration of a feature map from an input image

As we can see in Figure 10, the size of the image has reduced. In this case, a stride of 1 was used. If a stride of 2 was used, the image would have been further reduced. The strides for a kernel are usually decided based on the size of the input image. If the size of the image is quite large, then the strides are increased, whereas if the image is in a manageable size range, the stride is kept at 1. One of the most important roles of the feature detectors is to 'match the features'. It does so by having a set pattern on the map. When this set pattern

matches with the image the most, it gives a high output. As we can see in Figure 10, there is a strong pattern matching when the feature detector is moved over the lower left portion of the image, thereby giving a strong output of 4. So, the feature map basically helps us to preserve the most important features of a certain image. It does so by ‘extracting’ the most prominent and important features of the image and preserving them in feature maps. In essence, it gets rid of the non-essential portions of the image. Now, one of the main concerns a human may have is that how to decide which features are important and which are unnecessary. If we would have to brute force our way to find the number of important features in an image, it would take a many trials and errors to get to the desired features. This is where the computer steps in. Through its multiple rounds of training, it fine-tunes the feature detectors to extract the required features for the desired output.

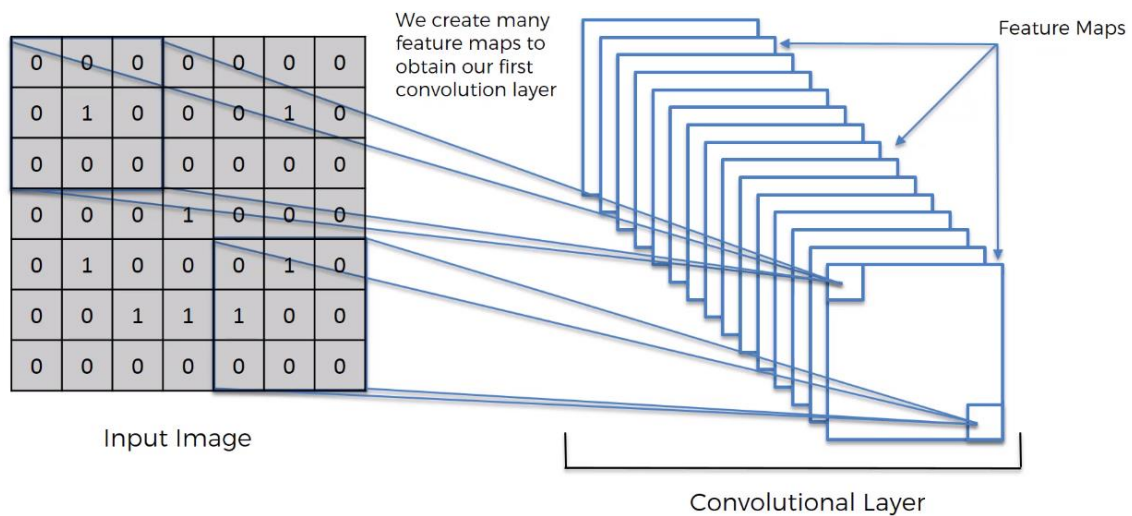


Figure 11. Representation of different feature maps populated to detect certain features

In Figure 11, we can see the different feature maps created from an input image. One of the feature maps may be detecting the edges in the image, the other may be detecting the contours, yet another may be sharpening the image etc.

Some of the filters used for feature detection are as follows:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & 5 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

This filter is used for '**sharpening**' the image

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

This filter is used for '**blurring**' the image

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

This filter is used for '**edge-enhance**'

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & -4 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

This filter is used for '**edge-detect**'

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & -2 & -1 & 0 & 0 \\ 0 & -1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

This filter is used for '**emboss**'

After the convolutional layer, we apply the Rectified Linear Unit (ReLU). This is a rectifier function. The reason we apply the rectifier function is to increase non-linearity in the image. One of the main reasons why we want to increase non-linearity in our network is because

images themselves are highly non-linear. Generally an image is non-linear in nature, the transition between pixels, the background and the border etc. , all indicate the non-linear nature of an image. When we run the convolution operation (or other mathematical operations), we risk converting the image into something linear, which is highly undesirable. To preserve the non-linear aspect of the image, we apply the ReLU layer to break up any linearity introduced by the convolution operation. It has been observed via experiments, that if the non-linear operation is removed, the system performance drop significantly. The formula for the ReLU function is:

$$y = \max(0, x) \quad (16)$$

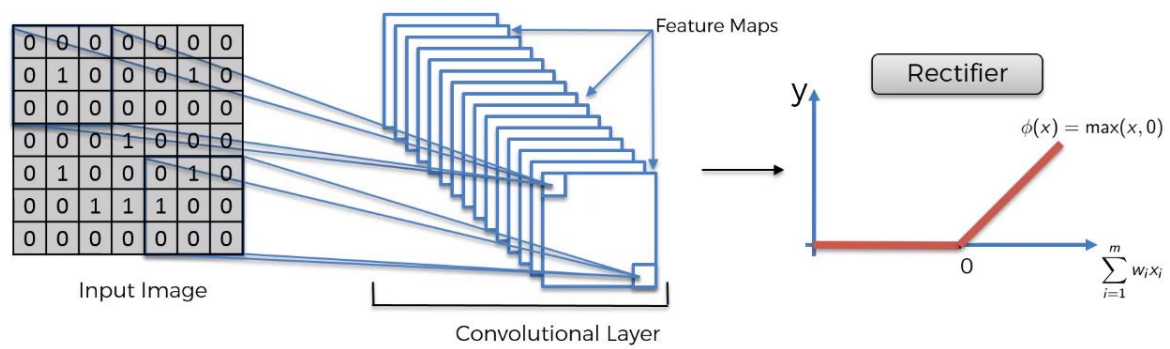


Figure 12. ReLU layer applied to the feature maps after convolution

There are other variations of ReLU, which also exist, like Leaky ReLU and Parameterized ReLU (PReLU). Leaky ReLU has a small slope for negative values instead of completely zero, e.g.

$$y = 0.2x \text{ for } x < 0 \quad (17)$$

Whereas, Parameterized ReLU instead of having a user-defined slope, has the computer decide the parameter for itself, e.g.

$$y = ax \text{ for } x < 0 \quad (18)$$

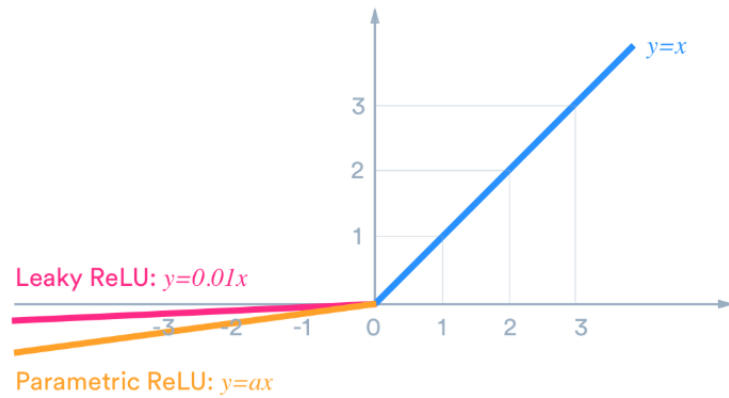


Figure 13. Leaky ReLU and Parameterized ReLU

The second step in a CNN, after convolution (and ReLU) is ‘**Pooling**’. One of the prime reasons for introducing pooling, is so that the neural network has spatial invariance by reducing the resolution of the feature maps. It means that if a certain feature is being searched by the network and that feature is in different positions in different images, it should still be able to detect it. Each pooled feature map corresponds to one feature map from the previous layer. For e.g., if a number of images of a cheetah is given to the network and the cheetah is looking in different directions in every image, then the network should not get confused w.r.t to finding a particular feature of cheetahs.



Figure 14. Pictures of cheetahs with their characteristic features in different locations of the image

For example, in figure 14 we can see pictures of cheetahs. In the above picture, we observe that the cheetahs are staring in different directions, some are not in the centre of the image, some have a different texture etc. Now if the network has learnt from the convolution step that a characteristic feature of a cheetah is the ‘tear-drop shadow’ which runs down from their eyes, then in some of the above images, it may not be able to recognize those features at all and may mis-classify the image as something other than a cheetah.

To prevent the above phenomenon from happening, we use pooling. Basically, if the features are a bit distorted, then the network should have the flexibility to find those features.

In max-pooling, we take the maximum out of a pixel area from the image and then adjust the stride and move ahead.

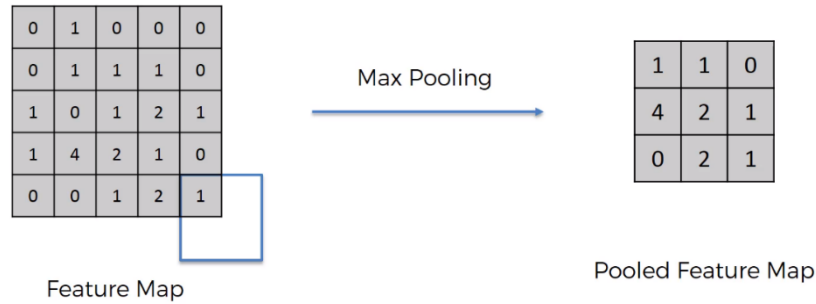


Figure 15. Max pooling applied to the feature map obtained from the convolutional layer

In Figure 15, we observe that by applying max-pooling to the feature map we are not losing any important features of the image. Technically, we are getting rid of 75% of the unnecessary features and only preserving the most important features. This helps to improve computational speed as the size of the matrix is reduced. Max-pooling is defined as follows:

$$a_j = \max_{N \times N} (a_i^{n \times n} u(n, n)) \quad (19)$$

We can observe spatial invariance in the Figure 15 too. For example, if the number 4 in the feature map represents the cheetah’s tear-drop feature, then if the number 4 was present anywhere else in the image, it would still be in the pooled feature map. By reducing the size of the feature map, we are automatically reducing the number of parameters which would go into the final layer. This means lower processing times and higher computational speeds and more importantly it helps to prevent over-fitting.

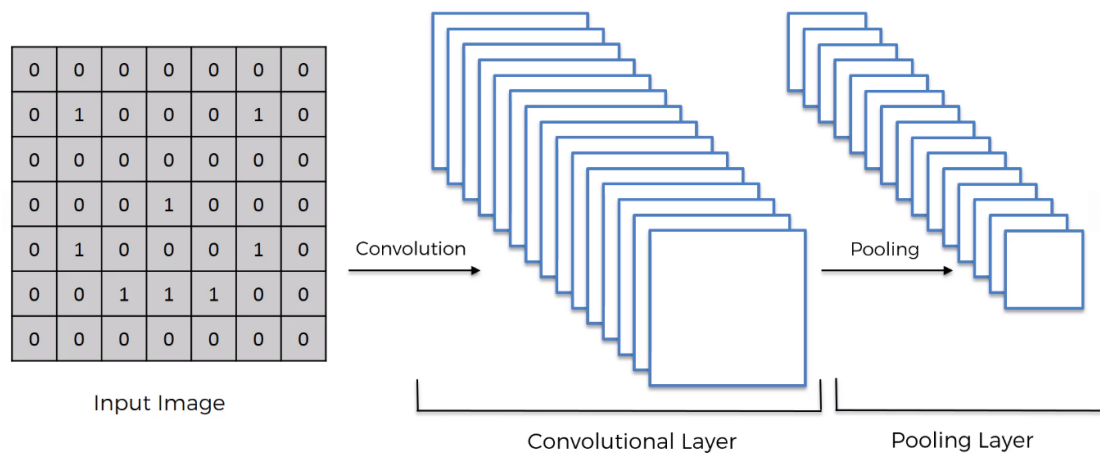


Figure 16. Reached the pooling step of the entire CNN

The third step after pooling is ‘**flattening**’. Flattening is a very simple process by which the data from the pooled feature maps is converted into a column vector. The reason for doing this is to feed the pooled feature map, values into the input neurons of the neural network. Since the input neurons cannot accept a matrix of values, we convert the matrix values into a column vector to feed into the network.

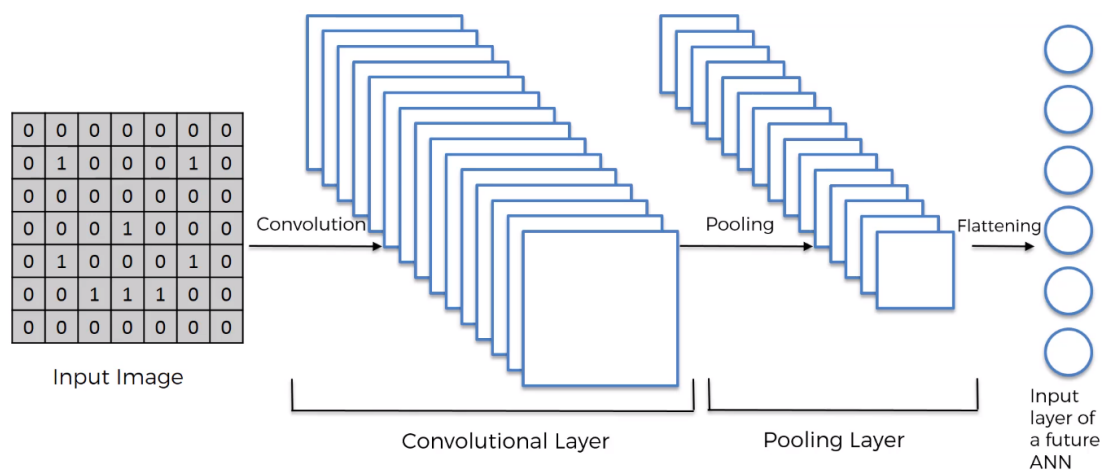


Figure 17. Flattening the pooled matrix of features into a column vector

The fourth and final step in a CNN is the ‘**fully-connected**’ layer. In this step, we add an artificial neural network to the end of the pooling step.

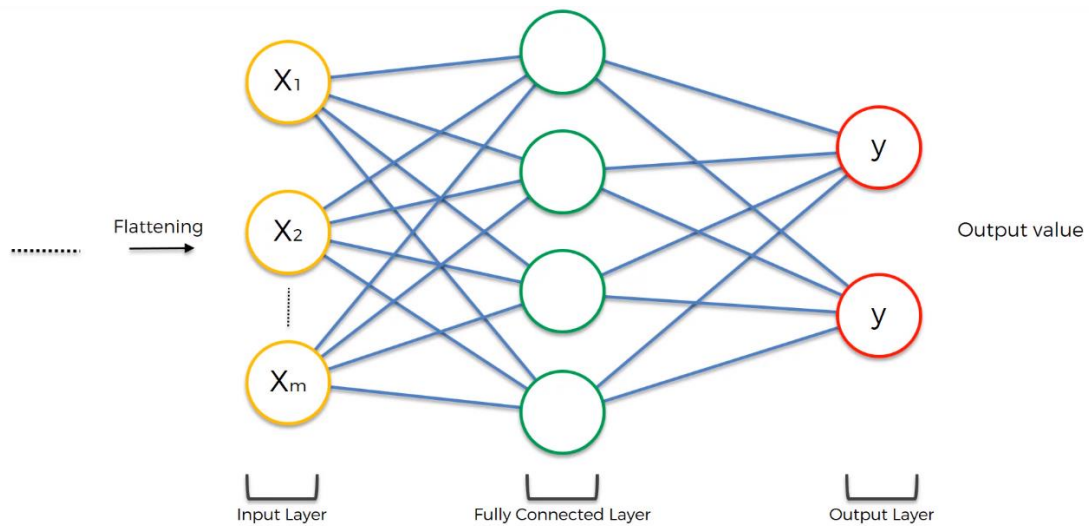


Figure 18. Full connection after the flattening step

In CNN's the hidden layers are called full connected layers because here each hidden neuron is connected to every preceding input. The task of the fully connected layer is to combine the features from the previous pooled maps into more attributes that predict the classes even better.

The number of output neurons depend on the number of classifications required. For example, if there are 3 or more classes of images which need to be classified, then we would have 3 output neurons.

One of the main things to understand is how the network trains its specific neurons to classify the images into their classes. We can take an example for that. Let us assume we have pictures of dogs and cats, which need to be classified. In the first pass, a certain image is fed into forward into the network. After it reaches the output neurons, the network classifies the image as a cat whereas it was actually a dog. So, in this case the cross-entropy loss function calculates the error in the predictions by back-propagating the values to the start of the network. This process is carried on from the very start to the finish till the entire network is trained. By ‘trained’, we mean that that weights and biases of the network are optimally tuned so as to correctly classify the image.

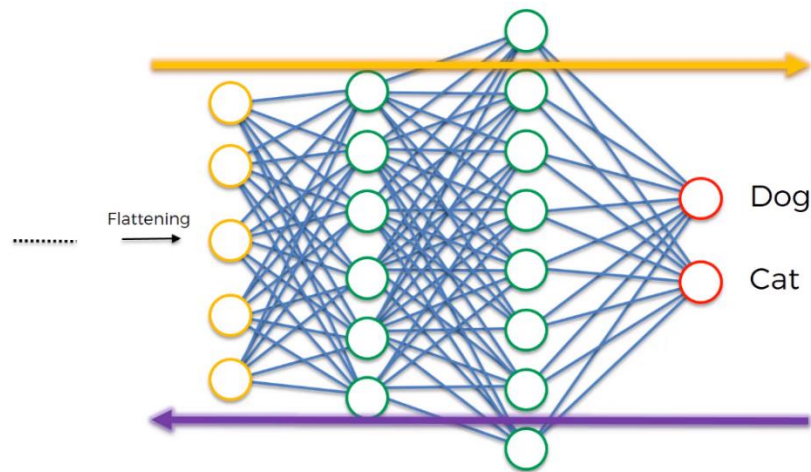


Figure 19. Example of the training of a network to classify dogs and cats

The feature detectors after the convolution step are adjusted to improve the collection of features and prediction values.

Let us assume that the network is now fully trained. The important thing to understand now is how the values in the network change as and when the input to the network changes. For the sake of clarity, let us assume the neurons in the network have the values as shown below in Figure 20.

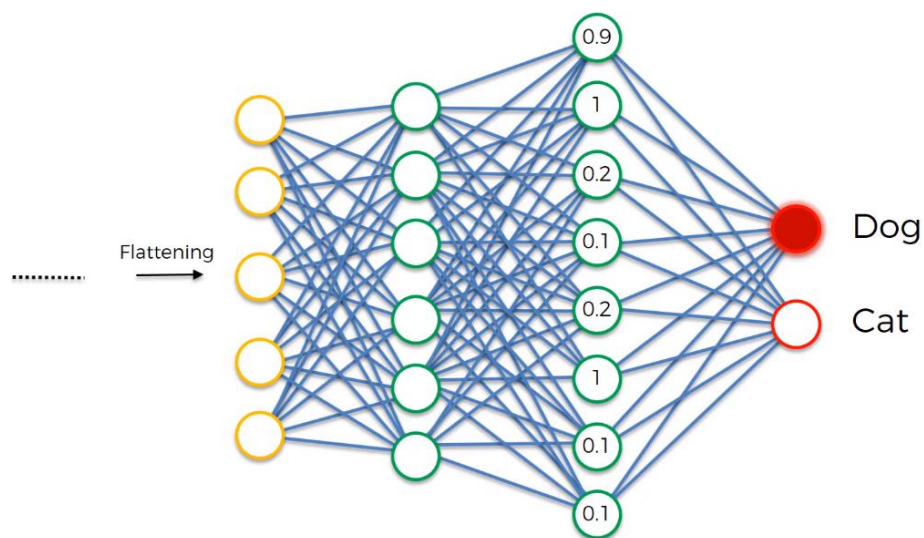


Figure 20. Trained network with values

The numbers shown in the above image are arbitrary and can be any number, but here we constraint them between '0' and '1'. '1' means that the neuron was very confident that it found a certain feature and '0' would mean that the neuron did not find a particular feature.

In Figure 20 we see that the second neuron from the top is firing with a value of ‘1’. This means that it is very confident about a certain feature. That particular neuron is firing the value of ‘1’ to both the dog and the cat output neuron. Along with that neuron, the first neuron is firing the value of 0.9 and the sixth neuron is firing the value of 1. Let us assume the second neuron has detected an eyebrow feature, the first neuron has detected a large nose and the sixth neuron has detected floppy ears. When this happens, the network has been trained to realise that whenever these features appear, those neurons fire up and classify as dogs and hence give the output of a dog. Whereas, the cat output neuron does not fire because it has been trained to recognize the features of a cat and it knows that the above features are not features of a cat. While, even though the cat neuron receives the same input from the neurons, it has realized that whenever those particular neurons fire up, the output is not a cat. Hence, the cat output neuron has learnt to ‘ignore’ those particular neurons.

That is how the neurons in any network understand the importance of the features for detecting a certain class of an image. In our use-case of a self-driving car, there is only one output neuron, which is basically the steering angle of the car as it drives.

One final step is applied to the outputs of the neurons, to make sure that the values lie between zero and one. The output from the neurons may be any value (and not necessarily add up to one). To make sure that the output values add up to one, we introduce the Softmax layer in the output. Softmax is the multi-classification counter-part of the Sigmoid function. The Softmax formula is:

$$f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}} \quad (20)$$

Using the Softmax function, the highest probability of a certain class will determine the target class. In simple terms, it calculates the probabilities of one class over the probabilities of all the other classes.

As a summary to how the CNN will work:

To an input image, we apply multiple feature detectors (also called filters) to get the feature maps. This comprises of the ‘convolutional’ layer. Then on top of that crucial layer, we apply the ReLU function to remove any linearity from the image. After that we apply the ‘pooling’ layer to the convolutional layer. This helps to create pooled feature maps which have a lot of advantages. It helps to discard unnecessary features and mainly preserve spatial invariance in the images. More-over pooling significantly reduces the size of the image and prevents over-fitting to the data. After the pooling step, we ‘flatten’ the pooled images into one long column vector to input them to the input layer of the artificial neural network. Then we have the final ‘fully-connected’ layer which helps perform the voting towards a class prediction.

The entire network is trained through forward and back-propagation process and through many epochs, we achieve the trained network.

Along with the weight and the biases, the feature detectors from the convolutional layer are also adjusted and the minimal loss function is achieved by using stochastic gradient descent.

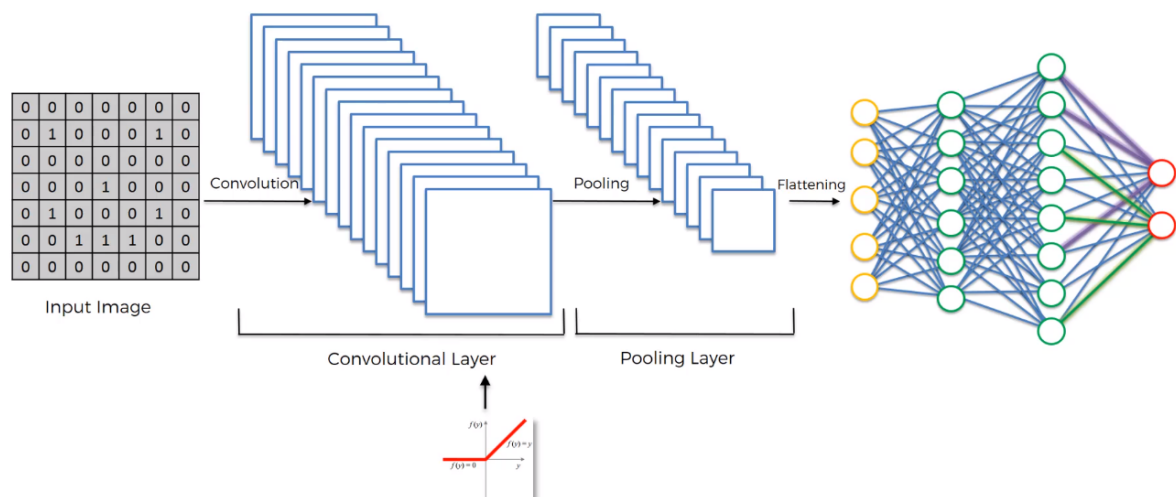


Figure 21. Summary of how CNNs work

4. DESIGN APPROACH AND DETAILS

4.1 Design Approach

The design approach involved creating a robust CNN architecture from scratch so that it can have a high level of autonomy, developing a car to emulate the real world conditions of a car and building a reliable simulator for the car to drive and train on.

The first design task to be handled was that of the architecture of the CNN. I went through numerous research papers and articles. A few worthy mentions, which helped me develop an idea on how to develop a CNN are ‘Gradient Based Learning applied to Document Recognition’ by Yann LeCun, ‘ImageNet Classification with Deep Convolutional Networks’ by Geoffrey Hinton et. al, ‘Visualizing and Understanding Convolutional Neural Networks’ by Zeiler and Fergus, ‘Very Deep Convolutional Networks for Large-scale Image Recognition’ by Karen Simonyan and Andrew Zisserman, ‘Going Deeper with Convolutions’ which came to be the basis for ‘GoogLe net’ , ‘Deep Residual Learning for Image Recognition’ by Microsoft Research team.

The common aspect of all these papers was the depth of the CNN. The papers focused on building a deep layer for the hierarchical nature of images to work. The main outcome of these papers was to have the network work robustly with image classification and localization.

In accordance with the design approach of the above papers, I designed the network to be very deep and have robust training of the network by designing drop-out neurons. These drop-out neurons, while not part of a layer specifically, emulate the human brain. They do so in the following way: In every few iterations, a few of the neurons are dropped from the training process, so that they learn the parameters on fewer images and training data. This makes them more robust, as there may come a scenario where the data provided to the network may not be enough to classify. In that case, if there are some neurons which have trained specifically on lesser data, can fire up and help the network maintain a reasonable amount of accuracy instead of the network completely failing.

Following is the architecture I came up with:

Table 1. Convolutional research model developed

<u>Conv. Net Configuration</u>
9 weight layers
Input Image (RGB image)
Normalization layer
Convolutional layer – 64
Convolutional layer – 64
Max Pooling layer – 64
Convolutional layer – 128
Convolutional layer – 128
Max Pooling layer – 128
Convolutional layer – 256
Convolutional layer – 256
Max Pooling layer – 256
Fully Connected Layer – 4096
Fully Connected Layer – 4096
Fully Connected Layer – 1000
Mean Square Error (MSE)
Output

In the above model, the input image is of size (160,320) with three channels (RGB). In the normalization layer, we convert all the pixel values in the range of (0, 0.5) and crop the images by (70, 25) i.e., 70 units from the height and 25 from the width. The reason for doing this is that the captured images from the car's bonnet consist of a major portion of the sky in the height part of the image, and other auxiliary images (like trees, rocks etc.) in the width portion of the image. These auxiliary parts of the image are not required for feature recognition, and hence to lessen the load on the computational system, those parts are cropped out.

In the convolutional layer, we choose a 3x3 feature detector with a stride of 2 and this outputs 64 feature maps. The activation function after this layer is the ReLU function. From these convolutional maps, we pull another set of 64 feature maps using convolution on the previous maps. Again, we use the ReLU activation function on the feature maps.

In the next layer, we apply the Max Pooling step where we use a 2x2-feature detector with a stride of 2, with 64 pooled features. In the next layer, we apply two Convolutional layers with 3x3 feature detectors with a stride of 1 and which outputs 128 feature maps each. Here, we apply the ReLU layer as an activation function. Next, a Max-pooling layer, with 2x2 feature detectors with a stride of 1, to output 128 pooled features. In the next two layers, we continue the above process by applying the convolutional layer of 256 feature output maps. Here we continue to use the 2x2 feature maps along with the strides of 1. The activation function used in the end is ReLU again. The final weight layer to be applied is the Max Pool layer with 256 pooled feature sets. Here 2x2 detectors with a stride of 1 are used. After this step, we use the optional step of 'dropout'. Here, we choose 25% of the neurons to drop out in random iterations, to make the model more robust.

Then the outputs from the pooled layer are connected to the fully connected layers of 4096 neurons with activation function of ReLU. This is in turn connected again to 4096 neurons with activation function of ReLU. This is connected to 1000 neurons with activation function of ReLU and those are subsequently connected to the output of the model which calculates the loss function by Mean-Square-Error and back-propagates it.

The above model has been trained and tested on both the training set collected from the car and even on image recognition sets of dogs and cats. And, the model has performed robustly with little to no human intervention required.

The simulator for the environment and the car has been developed in Unity. The interactions between the car and the prefabs have been programmed in C#. There have been two tracks designed in the simulator, one is a 'jungle' track and the other is a 'lake' track.

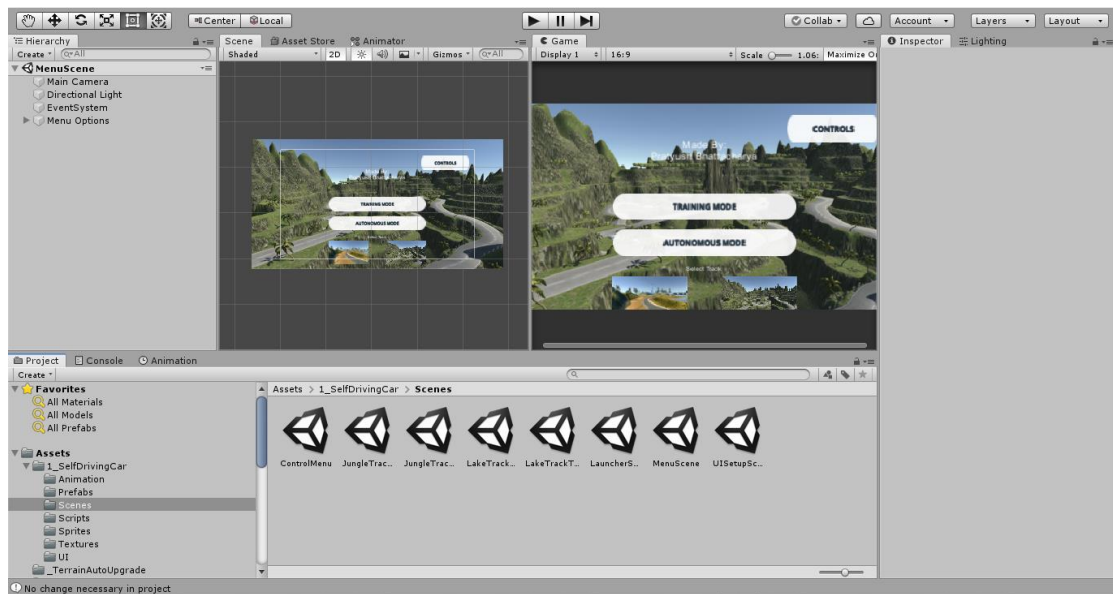


Figure 22. Unity Development environment

Following are a few design images of the jungle track:

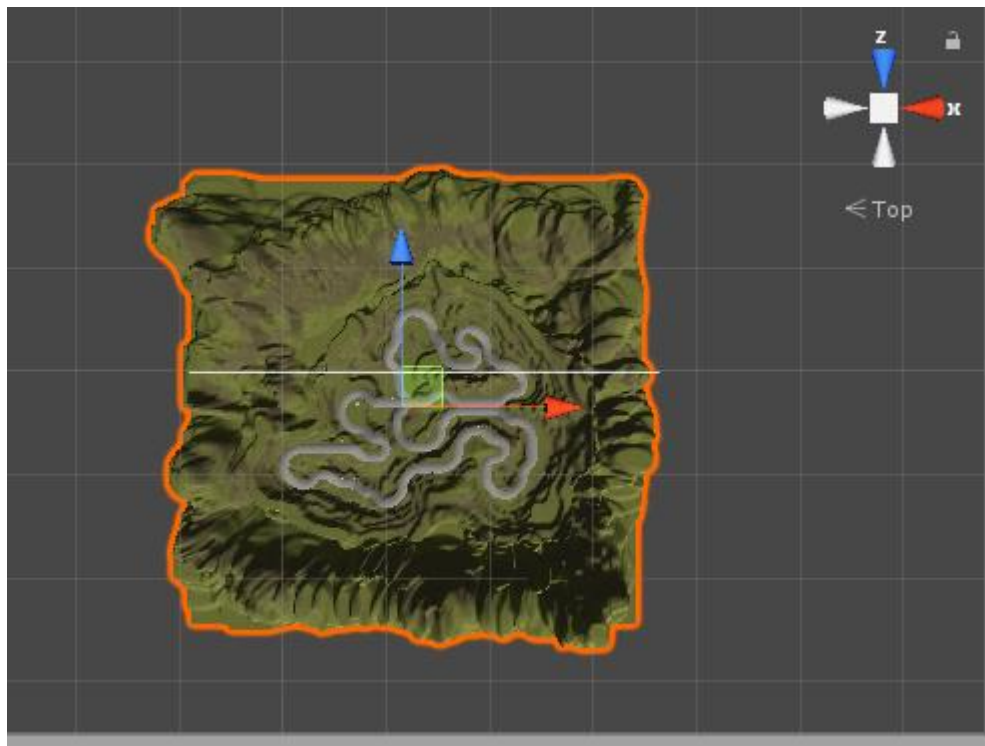


Figure 23. Designing the 'jungle' track

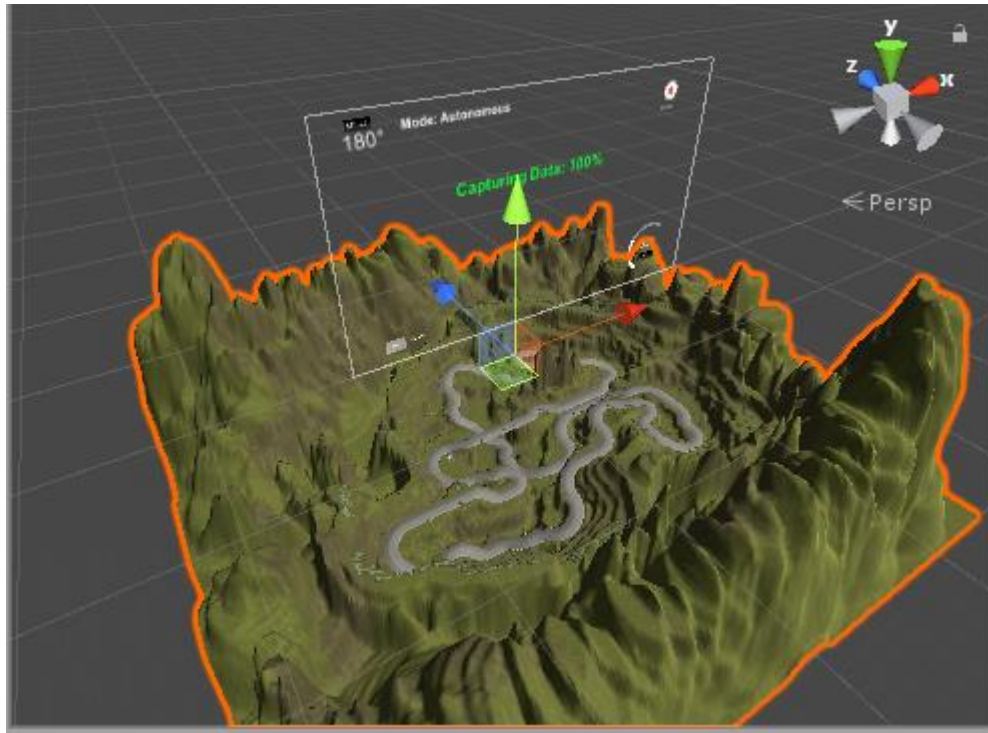


Figure 24. Aerial view of the jungle track

Following are the developmental designs of the 'lake' track:

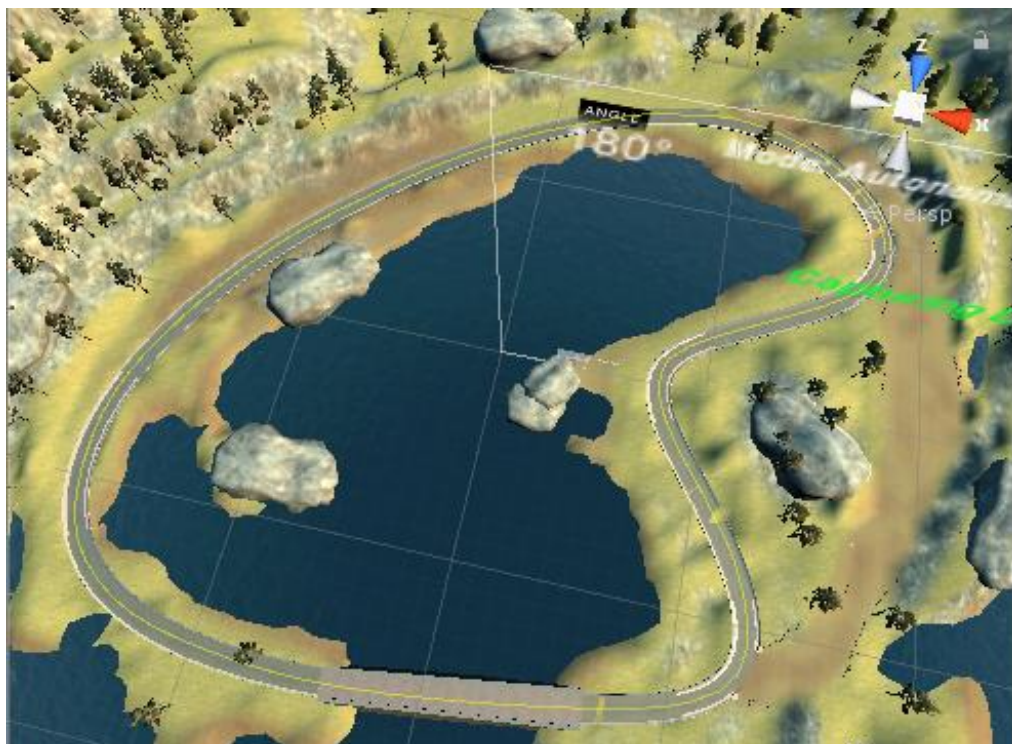


Figure 25. Lake track development



Figure 26. Lake track aerial view

Following are the images of the development of the car from different angles:



Figure 27. Frontal view of the car

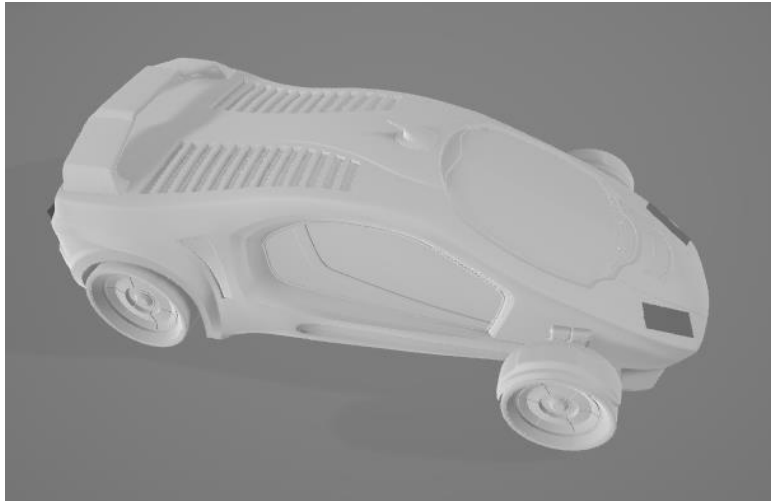


Figure 28. Side-view of the car



Figure 29. Back view of the car

The car's front wheels have been made to protrude out of the car so as to provide higher flexibility and mobility while turning.

Once the model has completed training, it stores the learned parameters in a 'model.h5' file. This format of the file is used, because it stores the parameters in a hierarchical format thereby, highly improving speed of reading the data from the file. We run the program by typing in "python drive.py model.h5". Here, the python program is receiving two parameters, 'drive.py' and 'model.h5'. The drive.py file contains a basic PI controller for the car to use while driving and socket connections to set up a client-server model with the simulator.

After typing in the above command, we open the simulator, which we made in Unity. Then we go to autonomous mode. Here the simulator acts as a server to the program to connect to and serve images to. The program acts as a client by processing the images provided from the simulator and feeding back the steering angle and throttle to the car.

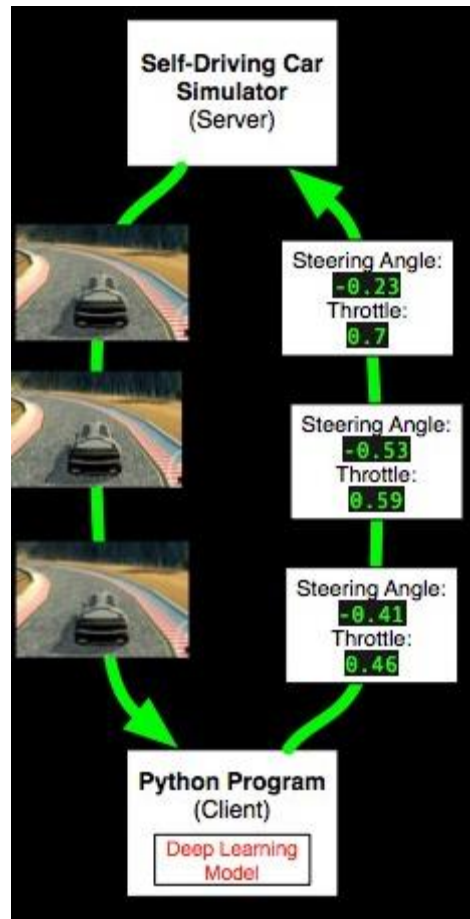


Figure 30. Client-server model for the self-driving car

4.2 Codes and Standards

Currently there exist standards for self driving cars and the impact of their safety on the world. These standard address computer-based system safety for conventional models. While these standards are essential, they do not completely address the issue of how “Highly Autonomous Vehicles” (HAV) will be deployed. The strategy used by the standards are based on the draft “UL 4600 Standard for Safety for the Evaluation of Autonomous Products” which intends to cover HAVs etc.

Some of the current standards that exist are:

ISO 26262: This standard focuses on the aspect that traditionally codes and standards have based their overall safety and use case on the principle that the human driver is himself responsible for the ultimate safety of himself and others on the roads. This has resulted in a focus on functional safety. Functional safety means, the ability of a system to mitigate failure of it's systems sufficiently, for identified potential hazards. The amount of intervention and mitigation required depends on the severity of the potential hazard scenario, operational conditions on being exposed to the hazards and most importantly, human driver maneuverability of the autonomous system in the case of a failure. These factors combine into an “Automotive Safety Integrity Level” (ASIL). The assigned ASIL for a function decides what type of mitigation and avoidance principles to be applied based on a pre-determined risk-table. It also includes specified design and analysis tasks that must be performed. The ISO 26262 is consistent with other safety standards such as IEC 61508. In summary, the emphasis of this standard is to avoid design faults and prevent the effect of equipment failure during operations.

ISO/PAS 21448: All the more as of late, the automotive industry has made a safety standard for driver help works that could neglect to work appropriately regardless of whether no gear flaw is available.

The ISO/PAS 21448 "Safety of the Intended Functionality" (SOTIF) standard tends to those issues. It principally considers relieving dangers because of surprising working conditions (the intended capacity may not generally work in these because of confinements of sensors and calculations) and holes in prerequisites (absence of complete depiction about what the intended capacity really is).

Features of this standard incorporate covering:

- Insufficient situational mindfulness
- Foreseeable abuse and human-machine association issues
- Issues emerging from operational condition (weather, foundation, and so on.)
- An accentuation on distinguishing and filling necessity holes (evacuating "questions")
- By and by, an accentuation on identifying operational situations.

ISO 21448 expands the extent of ISO 26262 to cover ADAS functionality. Both unequivocally license broadening the extension further.

There are various other safety norms from different spaces including: **IEC 61508** for chemical process control; **CENELEC EN 50128** for rail systems; **MIL-STD882E** for military systems; and **SAE ARP 4754A** just as **SAE ARP 4761** for aviation. While these give extra safety viewpoint, none covers the full scope of HAV issues. Principally this is because of suspicions of human administrator accessibility (e.g., airplane pilots), total prerequisites distinguishing proof, or potentially essentially rearranged operational condition contrasted with HAVs (e.g., secured rail option to proceed). While these standards give significant knowledge and standards, more is needed to give intensive direction to HAVs.

4.3 Constraints, Alternatives and Trade-offs

There have been some constraints that have hindered performance of the self-driving car system. These involved lack of a high-power processing system. While the model's accuracy rate could have been further improved with more training data, there was a limitation due to the processing capabilities of the systems in hand. The more the data was increased, the more the processing time was increasing exponentially. More-over, to overcome this training was tried on 'Google Colaboratory' which is an online service provider where it provides it's high performance systems for access to the general public for training. While, this did help to mitigate some of the issues, it comes with an added disadvantage that if we connect to any of their system, we only have a connection time of 12 hours and after that time, the system is disconnected and is transferred to another system thereby making us lose the progress of our trained system. Another constraint which came

with using 'Google COLAB' was the issue of bandwidth. Precisely, when we tried to upload the training set on to the online computers for processing, the process took quite a long time which ate away into the 12 hours thereby reducing the time available for training. The training data folder had to be uploaded to 'Google Drive' and from there imported onto 'Google COLAB'. This is a very time-consuming and slow process as the 'COLAB' servers have very low bandwidth for this kind of data transfer.

As an alternative to the above constraint of transfer-time, I had to write a separate python script to convert the training image folder into a HDF5 format. This script writing took up quite a measureable chunk of time from the project, as nobody had attempted conversion of a full image set with its labels into a HDF5 format. This format stores the data within it in a hierarchical format thereby contributing to high speed transfers and reading. The entire training set was converted into a training.h5 file and then uploaded to 'Google Drive' from where it got imported into 'Google COLAB' within minutes, thereby saving tremendous time for the actual training process.

The trade-off of using very deep neural networks are that the system may become unstable and may not be able to output any meaningful data for the car to use. This may lead to the car rotating round and round in circles and not be able to drive autonomously. While constructing a CNN architecture, it is important to keep in mind the dimensions of the image, otherwise by the time, the model reaches the last pooled layer, it may be reading only a few pixels which may lead to over-fitting to the data.

5. SCHEDULE, TASKS AND MILESTONES

The schedule for this project was created in December 2019. The project started in the first week of December (on 02-December-2019). The zeroth review consisted of having a topic ready and having an idea as to how the project will proceed. By 15th of December, I had made a plan for the entire project. This was especially important as a project of such a huge scale, requires a plan and a deadline without which the project cannot be completed and the developer is left focussing on a few things more than giving a holistic focus on all the topics for the project. By 31st December, I had researched on the mathematics behind the neural networks and the implementation details of them. I had reviewed numerous research papers for this and came to the mathematical conclusion for activation functions and back-propagation methodology. During this time, I had started working on the simulator model on Unity as it was quite a big chunk of work which would not have been completed had I not started by the December 2019. The basic model and the training track for the simulator was ready by 19th January 2020. So, I decided to collect the training data for the model by driving the car in the simulator and recording the images collected from the car into a training folder. The next step was to clean the data by deciding how much of the image to keep for training and also to segregate the images into training and testing data (as the entire simulator model was not built by then, I decided to test it's performance by using some of the collected data itself). The next and most important part of the project, was to develop and implement a CNN model from scratch. This took quite a lot of trial and error. The model was more or less ready by mid-February 2020. After this step, the training data was fed to the CNN model and training was commenced. Here is where one of the constraints discussed in the sections above came up where the time taken to upload the images was so humongous, that training was becoming impossible. After a lot of research, I realized that I had to write a particular script to improve the speed of the training. This step ate away a lot of time, but fortunately the model worked and was ready by 1st March 2020. The second review was conducted by the esteemed panel, and I could show them that 80% of my work was complete. By the end of March the model had been trained many times without any foreseeable errors and thus was deemed as complete. After that step, the documentation work was started, which led to the work on the project report.

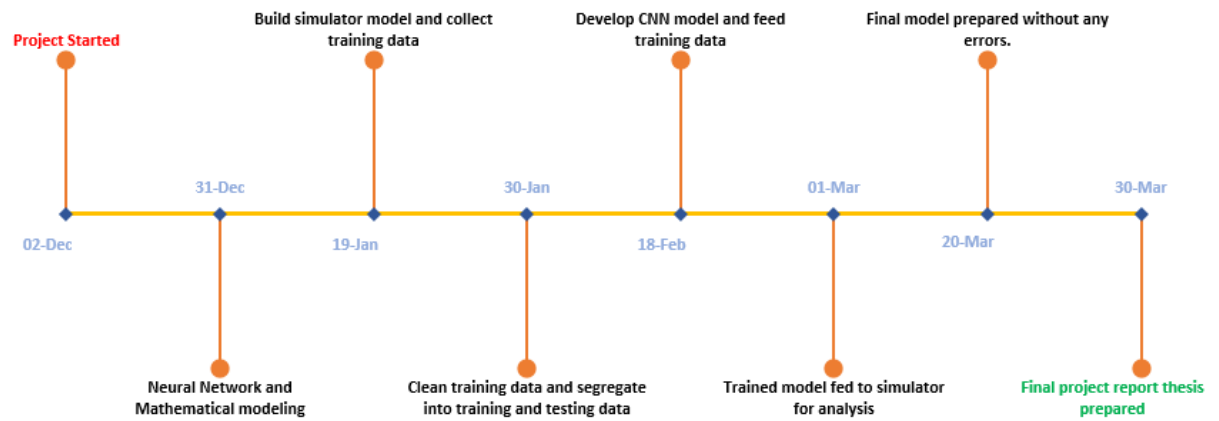


Figure 31. Project schedule timeline

A few notable milestones for the project were:

- Built Unity simulator completely.
- Development of the CNN architecture.
- Solving the constraint of overhead on ‘Google-COLAB’.
- Seeing the car drive autonomously.

6. PROJECT DEMONSTRATION

The project demonstration involves firing up the simulator with the trained model and have the car drive itself.

First, we observe the training data saved in a folder.

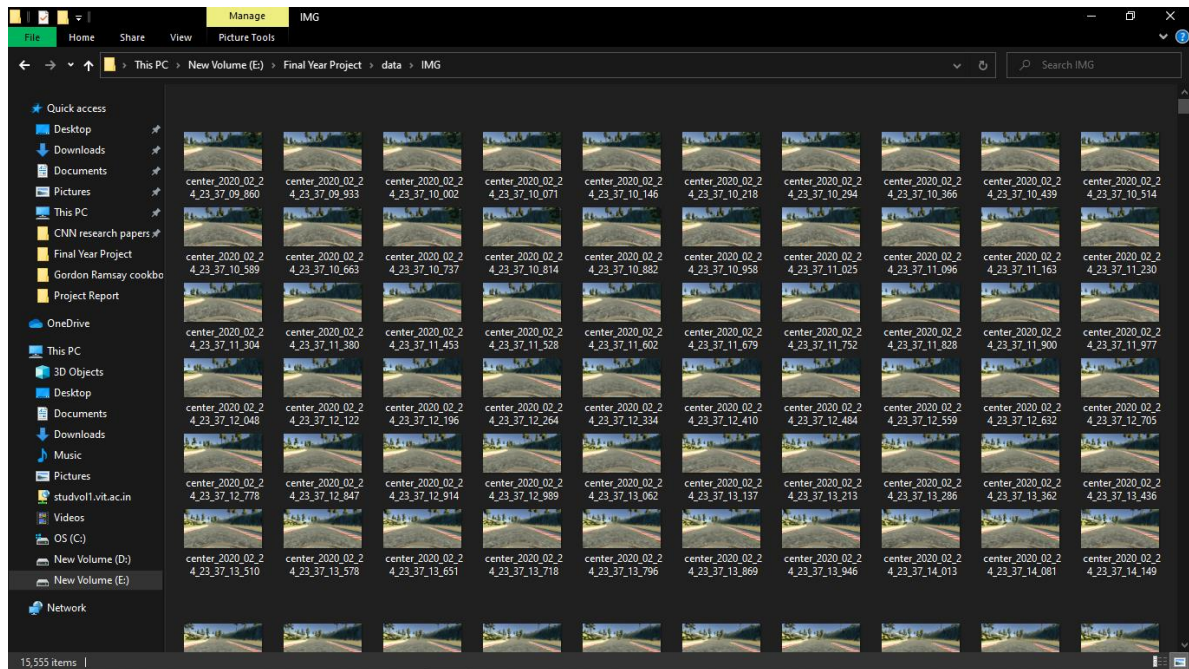


Figure 32. Training data folder. We can see in the lower left corner, there are 15,555 images in this training folder

This training data has been trained on the research model. The research model has been programmed on 'Google COLAB'. We observe a code snippet:

```

model = Sequential()
model.add(Lambda(lambda x: (x / 255.0) - 0.5, input_shape=(160,320,3)))
model.add(Cropping2D(cropping=((70,25), (0,0))))

#max pooling layers, we will use with a stride of 2 and with a pool size of 2,2
model.add(Conv2D(64,(3,3), padding="same", activation='relu'))
model.add(Conv2D(64,(3,3), padding="same", activation='relu'))

model.add(MaxPooling2D(pool_size = (2, 2), strides=(2,2))

#Since after maxpooling the size of the image has decreased,
#reducing the filter size to 3x3
model.add(Conv2D(128,(3,3), padding="same", activation='relu'))
model.add(Conv2D(128,(3,3), padding="same", activation='relu'))

model.add(MaxPooling2D(pool_size = (2, 2), strides=(2,2))

model.add(Conv2D(256,(3,3), padding="same", activation='relu'))
model.add(Conv2D(256,(3,3), padding="same", activation='relu'))
model.add(Conv2D(256,(3,3), padding="same", activation='relu'))

model.add(MaxPooling2D(pool_size = (2, 2), strides=(2,2))

model.add(Conv2D(512,(3,3), padding="same", activation='relu'))
model.add(Conv2D(512,(3,3), padding="same", activation='relu'))
model.add(Conv2D(512,(3,3), padding="same", activation='relu'))

model.add(MaxPooling2D(pool_size = (2, 2), strides=(2,2))

model.add(Conv2D(512,(3,3), padding="same", activation='relu'))
model.add(Conv2D(512,(3,3), padding="same", activation='relu'))
model.add(Conv2D(512,(3,3), padding="same", activation='relu'))

model.add(MaxPooling2D(pool_size = (2, 2), strides=(2,2))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(4096, activation='relu'))

```

Figure 33. Code snippet for the model

Next, we navigate to the folder where the model.h5 file is saved and open the terminal to get the client-server connection between the program and the simulator started.

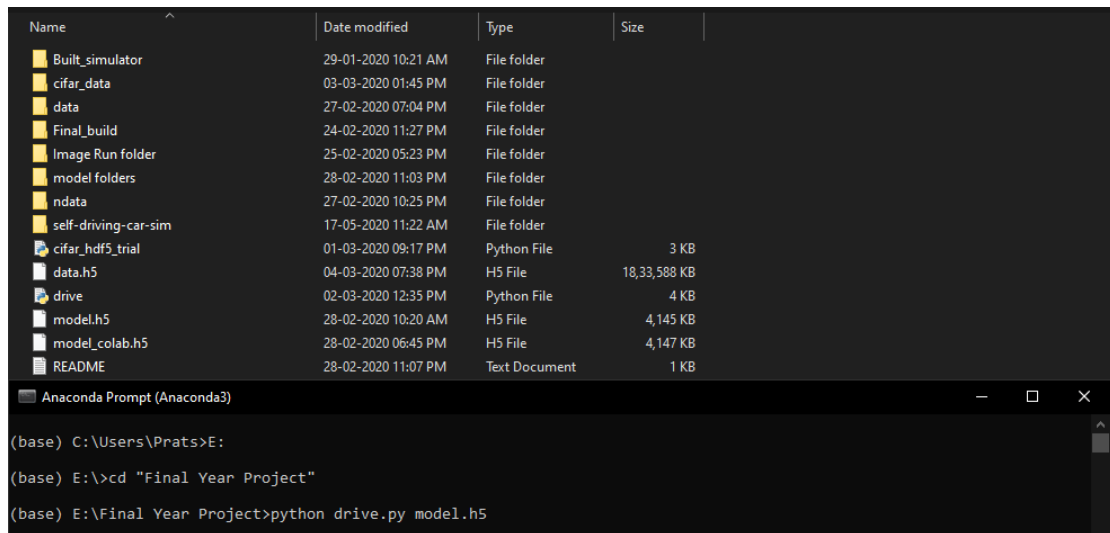


Figure 34. Running the drive.py file with the model.h5 file to start the client-server connection

Next, we navigate to the 'Final_build' folder which contains the executable for the simulator:

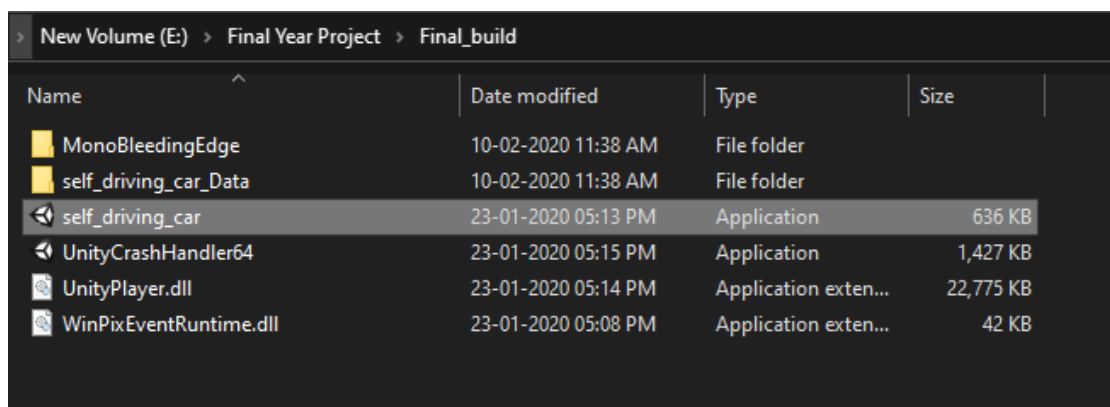


Figure 35. Final build folder containing the exe for the self-driving car

Now we run the file titled 'self_driving_car'. Following are the snippets with the application open:

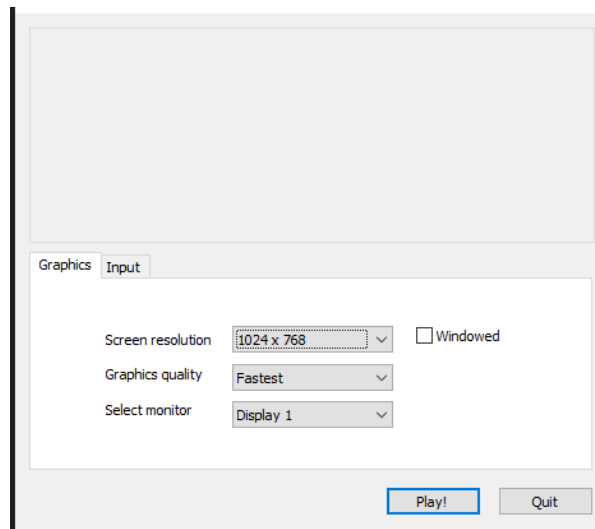


Figure 36. First dialogue box after double clicking on the exe file



Figure 37. Menu screen for selecting the mode and the track

After this screen, we choose the ‘Autonomous Mode’ for driving the car and we can choose from either of the two tracks below. In this case, we select the second track, as shown highlighted in the picture below.

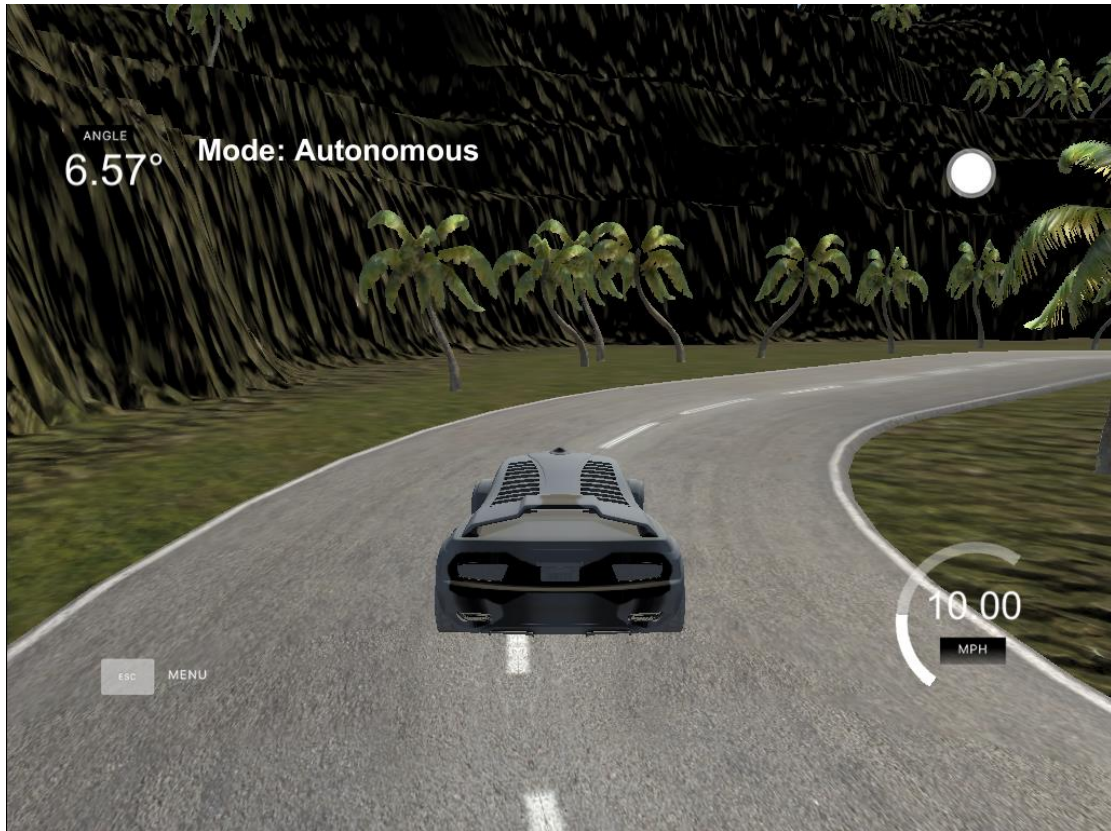


Figure 38. As visible, the car is driving itself autonomously on the 'jungle' track!

7. COST ANALYSIS

The current model built by us requires the presence of 3 cameras and a good on-board processor for quick calculations and responses to the car's wheel. Following are the costs for the specifics required:

- Cost of 3 good quality video cameras for viewing the front of the car for driving:
 $3 \times 100 = \$300 = \text{Rupees } 22,500$.
- Cost of on-board processor, here we consider the NVIDIA PX2 processor for on-board processing of signals (if the processor is bought in bulk, it costs \$2000 whereas if it is bought as a stand-alone unit it costs \$15000): $(\$2000 - \$15000) = \text{Rupees } (1, 50,000 - 11, 25,000)$.

The total cost for outfitting a pre-existing car with the **proposed** hardware:

Rupees 11, 47,500.

Now considering the costs of the sensors outfitted to the traditional self-driving cars in the market:

- Cost of LIDAR: $\$8000 = \text{Rupees } 6, 00,000$.
- Cost of on-board processor: $\$15000 = \text{Rupees } 11, 25,000$.

The total cost for outfitting a pre-existing car with the **existing hardware** in the market:

Rupees 17, 25,000.

As we can see, there is a cost saving of **Rupees 5, 77,500.**

SUMMARY

Based on the entire discussion above, we have observed that a self-driving car is achievable with the current available technology. We have used the revolutionary technology of Convolutional Neural Networks to achieve the learning for such a network. While it is true, that there will have to many further tests before such a model can be put to test on an actual car, it is good to realize that there is a path through which we can reach our final goal of having a Level-4/5 autonomous vehicle. There are new innovations in the field of CNN architecture every year. Microsoft released an architecture for their CNN which achieved levels of accuracy which surpasses human capabilities too. Such networks will become the standard modus-operandi as the prevalence of self-driving cars increases. The above model has a good accuracy rate and may even do well on actual road conditions. Though, it is likely that the model will have to re-collect the training data by driving on the actual road and obtaining new samples to train on.

The model proposed by us, is a cost-effective venture as it used 3 video cameras to map the road ahead of itself. Current autonomous vehicles, get inputs by using expensive technology such as LIDAR, RADAR etc. While it is true, that performance is enhanced by using these sensors, ultimately they are cannot be extrapolated to real life cars, otherwise the cost of production of those cars would go into millions of dollars. The hope is to improve on the current technology proposed by us and make further fine-tuning to the CNN network, so that human intervention is reduced. Moreover, it is important that these models are in sync with the codes and standards being laid out for autonomous vehicles, as that will ensure that mitigation of failures are present and the system is overall highly reliable and robust for the human passengers.

REFERENCES

1. Mariusz Bojarski, “End to End Learning for Self-Driving Cars”, NVIDIA Corporation, April 2016
2. Ross Girshick, “Fast R-CNN”, Microsoft Research, September 2015
3. Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”, January 2016
4. Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation”, UC Berkeley, October 2014.
5. Karen Simonyan, Andrew Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition”, University of Oxford, April 2015.
6. Matthew D. Zeiler, Rob Fergus, “Visualizing and Understanding Convolutional Networks”, New York University, November 2013.
7. Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, “Deep Residual Learning for Image Recognition”, Microsoft Research, December 2015.
8. Geoffrey E. Hinton, “Image Net Classification with Deep Convolutional Neural Networks”, University of Toronto, May 2017
9. Adam Jones, Scott Devitt, Ravi Shanker, “Self-Driving the New Auto Industry Program”, Morgan Stanley, November 2013
10. Todd Litman, “Autonomous Vehicle Implementation Predictions”, Victoria Transport Policy Institute, March 2020.
11. Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, “Going Deeper with Convolutions”, Google Inc., University of North Carolina, Chapel Hill, University of Michigan Ann Arbor, Magic Leap Inc, October 2015.
12. <https://blog.valohai.com/self-driving-with-valohai>
13. <https://stats.stackexchange.com/questions/360899/difference-between-strided-and-non-strided-convolution>
14. <https://towardsdatascience.com/types-of-convolutions-in-deep-learning-717013397f4d>
15. <https://zhengludwig.wordpress.com/projects/self-driving-rc-car/>
16. https://www.researchgate.net/publication/311843099_Self-

[driving and driver relaxing vehicle](#)

17. <https://events.windriver.com/wrcd01/wrcm/2017/05/Open-Standards-for-Autonomous-Cars-White-Paper.pdf>
18. <https://www.perforce.com/blog/qac/how-autonomous-vehicle-technology-driving-coding-standards>
19. https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html
20. <https://www.extremetech.com/computing/274795-tesla-dumps-nvidia-goes-it-alone-on-ai-hardware>
21. <https://blogs.nvidia.com/blog/2019/04/15/how-does-a-self-driving-car-see/>
22. <https://www.nvidia.com/en-us/self-driving-cars/drive-platform/hardware/>
23. <https://www.wired.com/2015/04/cost-of-sensors-autonomous-cars/>
24. <https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148>

Capstone Project

Project Title	Deep-Learning Assisted Autonomous Vehicle
Team Members (Names and Reg. Nos.)	Pratyush Bhattacharya, 16BEI0088
Faculty Guide	Dr. Balaji S
Semester / Year	VIII / IV year
Project Abstract (not more than 200 words)	<p>The project focuses on the design and development of a level-2/level-2+ autonomous driving model that can successfully navigate, drive itself and perform feature recognition. To facilitate testing, the project involves building a simulator, where real-world conditions can be emulated. Convolutional neural networks (CNN) are used here to associate raw pixels (collected from the training images) from three front-facing cameras directly to steering commands. This approach is particularly utilitarian, as in most real-world scenarios the images will be collected in real time and it would be expected to get an immediate driving signal response. One of the foremost reasons for using CNNs are that, they have revolutionized pattern recognition. Before the advent of CNN architectures, each use-case had to build their own feature extraction modules followed by a classifier. The novelty of CNNs are that they learn a required set of features directly from the training images. As most images captured are 2-D in nature, the convolutional operation helps to capture that nature.</p> <p>In this project, there will be a self-built architecture of a CNN along with the environment which will emulate a car and it's immediate surroundings. For the training set of images, the car will be driven by a human driver on a certain road, whereas for the testing of the model, the car will be driven on a completely different track. Due to the high learning capability of the CNN models, many further features which may not be explicitly mentioned by the developer, will also be learned. Once the network is trained, we can observe the simulator car drive autonomously, without any user assistance, due to the generated steering commands from the trained model's output. After addressing the technical details of the project, the design approaches and constraints will also be addressed. Apart from that, there will also be a review of the existing codes and standards present for self-driving cars. Finally, the cost analysis between our proposed model and existing model will be compared and analyzed.</p>
Project Title	Deep-Learning Assisted Autonomous Vehicle
List codes and standards that significantly affect your project. (Must)	ISO26262, IEC 61508, ISO/PAS 21448, SAE ARP 4754A, SAE ARP 4761, CENELEC EN 50128
List at least two significant realistic design constraints	<ol style="list-style-type: none"> i. While the model's accuracy rate could have been further improved with more training data, there was a limitation due to the processing capabilities of the systems in hand. The more the data was increased, the more the processing time was increasing exponentially.

that are applied to your project. (Must)	<p>ii. While, Google COLAB helped to mitigate some of the constraints, it comes with an added disadvantage that if we connect to any of their system, we only have a connection time of 12 hours and after that time, the system is disconnected and is transferred to another system thereby making us lose the progress of our trained system.</p>
Briefly explain two significant trade-offs considered in your design, including options considered and the solution chosen (Must)	<p>The trade-off of using very deep neural networks are that the system may become unstable and may not be able to output any meaningful data for the car to use. This may lead to the car rotating round and round in circles and not be able to drive autonomously. While constructing a CNN architecture, it is important to keep in mind the dimensions of the image, otherwise by the time, the model reaches the last pooled layer, it may be reading only a few pixels which may lead to over-fitting to the data.</p>
Describe the computing aspects, if any , of your project. Specifically identifying hardware-software trade-offs, interfaces, and/or interactions	<p>The CNN network architecture was modeled in Python. The simulator model for the car and their environment has been modeled in Unity. While Python is currently the best framework for neural networks related tasks, the processing complexity of extremely deep systems causes a limitation on the depth of the network to be designed.</p> <p>Provided sufficient funds are available, there will virtually be no hardware-software trade-offs as current on-board processing hardware is more than capable to handle the computing capabilities required. Despite that, with a robust enough model and with enough hidden layers, a model can take two to three weeks to train on even a high powered GPU-enabled system.</p>