STEERING IN SELF DRIVING CARS USING DEEP LEARNING

A PROJECT REPORT

Submitted by

MARIA ANSON A (1715016)

In partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

ALAGAPPA CHETTIAR GOVERNMENT COLLEGE OF ENGINEERING AND TECHNOLOGY, KARAIKUDI - 630 003

(An Autonomous Institution Affiliated to Anna University, Chennai)

MARCH 2021

ALAGAPPA CHETTIAR GOVERNMENT COLLEGE OF ENGINEERING AND TECHNOLOGY, KARAIKUDI - 630 003

(An Autonomous Institution Affiliated to Anna University, Chennai)

BONAFIDE CERTIFICATE

Certified that this project report "**STEERING IN SELF DRIVING CARS USING DEEP LEARNING**" is the bonafide work of **(17CS019)** who carried out the project work under my supervision.

SIGNATURE	**SIGNATURE**
Dr.L.RASIKANNAN,	**Mrs K.CHANDRAPRABHA M.E.,**
ASSISTANT PROFESSOR	HEAD OF THE DEPARTMENT
DEPT OF COMPUTER SCIENCE AND ENGINEERING,	DEPT OF COMPUTER SCIENCE AND ENGINEERING,
Alagappa Chettiar Government College of	Alagappa Chettiar Government College
of Engineering and Technology,	of Engineering and Technology,
Karaikudi - 630 003.	Karaikudi - 630 003.

Submitted for (15CSZ03) Project Viva Voce examination held on _______

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

I thank the Almighty for keeping me healthy and providing me good circumstances to finish the project.

I would like to express my deep sense of gratitude and respectful regards to our Principal **Dr.K.MANONMANI, M.E.,Ph.D.,**AlagappaChettiar Government College of Engineering and Technology, Karaikudi-630 003 for giving an opportunity to do this project.

I owe my sincere thanks to **Prof.K.CHANDRAPRABHA M.E., Head of Department, Computer Science and Engineering,** AlagappaChettiar Government College of Engineering and Technology, Karaikudi-630 003 for providing various facilities and also giving valuable guidance during the project work.

I thank my Mentor **Dr.L.RASIKANNAN Assistant Professor** Department of Computer Science and Engineering, AlagappaChettiar Government College of Engineering and Technology, Karaikudi-630 003 for his full support in doing my project.

I would like to thank my Faculty Advisors **Dr.L.RASIKANNAN Assistant Professor** and **Dr.N.HEMARAJINI Associate Professor** Department of Computer Science and Engineering, AlagappaChettiar Government College of Engineering and Technology, Karaikudi - 630 003 for their support and encouragement.

I like to express my hearty thanks to all our beloved faculty members, non-teaching staff and my friends for their valuable suggestions and co-operation for finishing my project.

ABSTRACT

Self Driving cars are the hot spot area in the automobile industry. Currently we are in the level 4 of autonomous cars which was achieved by the brand Tesla. Conventional methods use separate machine learning models for object detection, path detection, segmentation, etc and combine their outputs to achieve the automation.

Our approach is an end to end system which is done by a convolutional neural network (CNN) to map raw pixels from a single front-facing camera directly to steering commands. With minimum training data from humans the system learns to drive in traffic on local roads with or without lane markings and on highways i.e we never train models to detect lane or road, etc.

The training signal will be the steering angle with the corresponding images. Thus we will be able to create a smart system which is very efficient in terms of memory consumption. The dataset contains approximately 63,000 images, 3.1GB of Data which was recorded around Rancho Palos Verdes and San Pedro California. The images in the dataset are converted into HSV color space to account for the differences in lighting conditions and are stored in a pickle format for faster processing and accessing the data. Then the model is trained on CNN to obtain the steering angle as output. The output is given as the input to the mechanical steering which is illustrated using a simulation.

End-to-end learning leads to better performance and smaller systems. Better performance results because the internal components self-optimize to maximize overall system performance, instead of optimizing human-selected intermediate criteria, e. g., lane detection. Such criteria understandably are selected for ease of human interpretation which doesn't automatically guarantee maximum system performance. Smaller networks are possible because the system learns to solve the problem with the minimal number of processing steps.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

CNN	CONVOLUTIONAL NEURAL NETWORK
SAE	SOCIETY OF AUTOMOTIVE ENGINEERS
AI	ARTIFICIAL INTELLIGENCE
SVM	SUPPORT VECTOR MACHINE
ANN	ARTIFICIAL NEURAL NETWORK
GPU	GRAPHICS PROCESSING UNIT
API	APPLICATION PROGRAMMING INTERFACE
R&D	RESEARCH AND DEVELOPMENT
CONVNET	CONVOLUTIONAL NEURAL NETWORK
DL	DEEP LEARNING
RELU	RECTIFIED LINEAR UNIT
NN	NEURAL NETWORK
FC	FULLY CONNECTED

CHAPTER 1

1. INTRODUCTION

1.1. OBJECTIVE OF THE PROJECT

The objective of the project is to steer the car using images from the camera installed at the dashboard so that autonomy can be achieved in cars. The working model to achieve the task is Convolutional Neural Network - a state of the art deep learning model for feature extraction on image is trained on 63,000 images.

1.2. SCOPE OF THE PROJECT

- End to End system for modelling the steering task

- Efficient usage of computational resources
- Lesser training data for the task

1.3. OUTLINE OF THE PROJECT

General approach to this problem will be creating separate deep learning models for object detection, path detection, image segmentation, etc. and combining the output of those models to achieve the task. But this leads to a lot of computational power when deployed and a heavy amount of training time for the model. The model uses convolutional neural networks (CNNs) to map the raw pixels from a front-facing camera to the steering commands for a self-driving car. This powerful end-to-end approach means that with minimum training data from humans, the system learns to steer, with or without lane markings, on both local roads and highways. The system can also operate in areas with unclear visual guidance such as parking lots or unpaved roads. The system is trained to automatically learn the internal representations of necessary processing steps, such as detecting useful road features, with only the human steering angle as the training signal. We do not need to explicitly train it to detect, for example, the outline of roads.

1.4. DOMAIN EXPLANATION

1.4.1. SELF DRIVING CAR

An autonomous car is a vehicle capable of sensing its environment and operating without human involvement. A human passenger is not required to take control of the vehicle at any time, nor is a human passenger required to be present in the vehicle at all. An autonomous car can go anywhere a traditional car goes and do everything that an experienced human driver does. The SAE uses the term automated instead of autonomous. One reason is that the word autonomy has implications beyond the electromechanical. A fully autonomous car would be self-aware and capable of making its own choices. The term self-driving is often used interchangeably with autonomous.

However, it's a slightly different thing. A self-driving car can drive itself in some or even all situations, but a human passenger must always be present and ready to take control. Self-driving cars would fall under Level 3 (conditional driving automation) or Level 4 (high driving automation). They are subject to geofencing, unlike a fully autonomous Level 5 car that could go anywhere. Autonomous cars rely on sensors, actuators, complex algorithms, machine learning systems, and powerful processors to execute software. Autonomous cars create and maintain a map of their surroundings based on a variety of sensors situated in different parts of the vehicle. Radar sensors monitor the position of nearby vehicles.

Video cameras detect traffic lights, read road signs, track other vehicles, and look for pedestrians. Lidar (light detection and ranging) sensors bounce pulses of light off the car's surroundings to measure distances, detect road edges, and identify lane markings. Ultrasonic sensors in the wheels detect curbs and other vehicles when parking.Sophisticated software then processes all this sensory input, plots a path, and sends instructions to the car's actuators, which control acceleration, braking, and steering. Hard-coded rules, obstacle avoidance algorithms, predictive modeling, and object recognition help the software follow traffic rules and navigate obstacles.

1.4.2. WHY DEEP LEARNING

Deep learning is a subset of machine learning in artificial intelligence that has networks capable of learning unsupervised from data that is unstructured or unlabeled. Also known as deep neural learning or deep neural network. Deep learning is an artificial intelligence (AI) function that imitates the workings of the human brain in processing data and creating patterns for use in decision making.

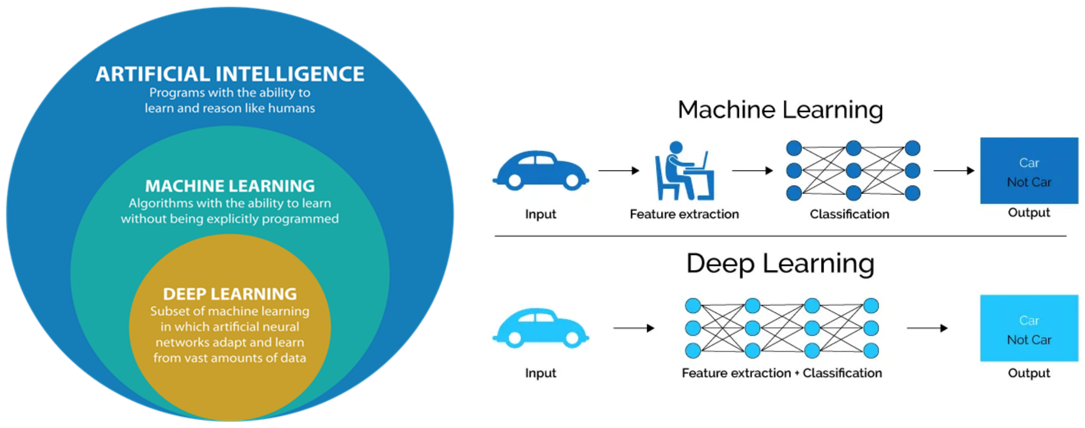

ARTIFICIAL INTELLIGENCE
Programs with the ability to learn and reason like humans
MACHINE LEARNING
Algorithms with the ability to learn without being explicitly programmed
DEEP LEARNING
Subset of machine learning in which artificial neural networks adapt and learn from vast amounts of data
Machine Learning
Input
Feature extraction
Classification
Car
Not Car
Output
Deep Learning
Input
Feature extraction + Classification
Car
Not Car
Output

Fig.1.1 Domain Chart

It is just a type of Machine Learning, inspired by the structure of a human brain. Deep learning algorithms attempt to draw similar conclusions as humans would by continually analyzing data with a given logical structure. To achieve this, deep learning uses a multi-layered structure of algorithms called neural networks. The design of the neural network is based on the structure of the human brain. Just as we use our brains to identify patterns and classify different types of information, neural networks can be taught to perform the same tasks on data.The individual layers of neural networks can also be thought of as a sort of filter that works from gross to subtle, increasing the likelihood of detecting and outputting a correct result.The human brain works similarly. Whenever we receive new information, the brain tries to compare it with known objects. The same concept is also used by deep neural networks.

The first advantage of deep learning over machine learning is the needlessness of the so-called feature extraction. Long before deep learning was used, traditional machine learning methods were mainly used. Such as Decision Trees, SVM, Naïve Bayes Classifier and Logistic Regression.These algorithms are also called flat algorithms. Flat here means that these algorithms can not normally be applied directly to the raw data (such as .csv, images, text, etc.). We need a preprocessing step called Feature Extraction. The result of Feature Extraction is a representation of the given raw data that can now be used by these classic machine learning algorithms to perform a task.

For example, the classification of the data into several categories or classes. Feature Extraction is usually quite complex and requires detailed knowledge of the problem domain. This preprocessing layer must be adapted, tested and refined over several iterations for optimal results. On the other side are the artificial neural networks of Deep Learning. These do not need the Feature Extraction step. In other words, we can also say that the feature extraction step is already part of the process that takes place in an artificial neural network. The layers are able to learn an implicit representation of the raw data directly and on their own. Here, a more and more abstract and compressed representation of the raw data is produced over several layers of artificial neural-nets.

This compressed representation of the input data is then used to produce the result. The result can be, for example, the classification of the input data into different classes. During the training process, this step is also optimized by the neural network to obtain the best possible abstract representation of the input data. This means that the models of deep learning thus require little to no manual effort to perform and optimize the feature extraction process. Let us look at a concrete example. For example, if you want to use a machine learning model to determine if a particular image is showing a car or not, we humans first need to identify the unique features or features of a car (shape, size, windows, wheels, etc.) extract the feature and give them to the algorithm as input data.In this way, the algorithm would perform a classification of the images. That is, in machine learning, a programmer must intervene directly in the action for the model to come to a conclusion. In the case of a deep learning model, the feature extraction step is completely unnecessary. The model would recognize these unique characteristics of a car and make correct predictions. In fact, refraining from extracting the characteristics of data applies to every other task you'll ever do with neural networks. Just give the raw data to the neural network, the rest is done by the model.

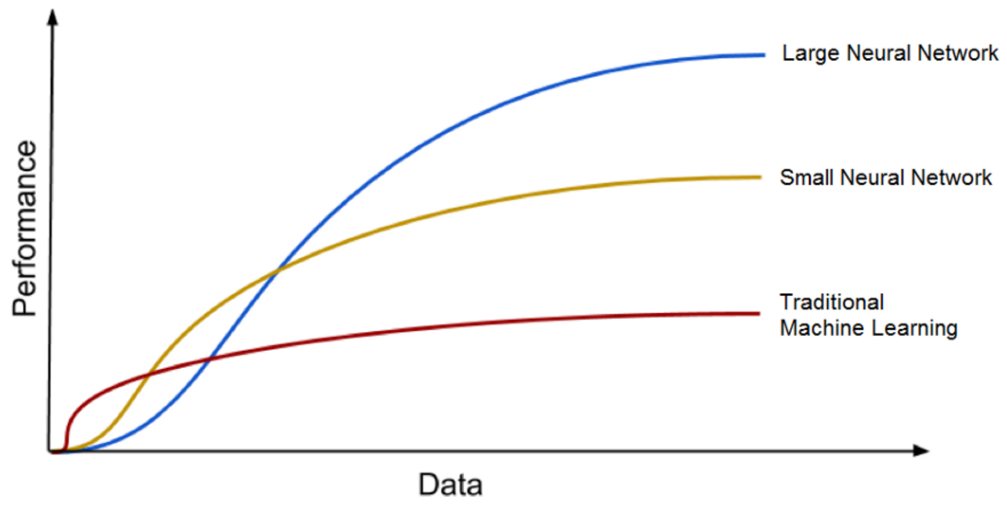


Fig.1.2 Data vs Performance Graph

Deep Learning models tend to increase their accuracy with the increasing amount of training data, where's traditional machine learning models such as SVM and Naive Bayes classifier stop improving after a saturation point.

"The analogy to deep learning is that the rocket engine is the deep learning models and the fuel is the huge amounts of data we can feed to these algorithms."

Biological Neural Networks

The concept behind biological neural networks, so when we will later discuss the artificial neural network in more detail we can see parallels with the biological model. Artificial neural networks are inspired by the biological neurons that are found in our brains. In fact, the artificial neural networks simulate some basic functionalities of the neural networks in our brain, but in a very simplified way. Let's first look at the biological neural networks to derive parallels to artificial neural networks. In short, a biological neural network consists of numerous neurons.

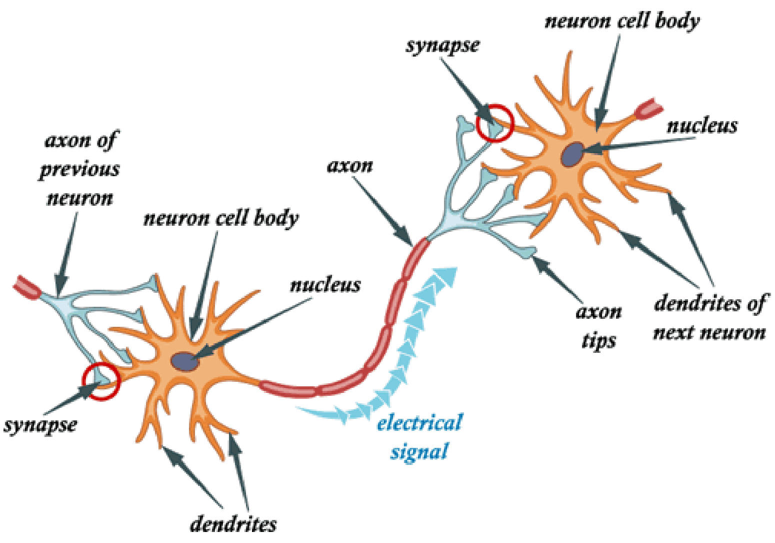


Fig.1.3 Biological Neural Network

A typical neuron consists of a cell body, dendrites, and an axon. Dendrites are thin structures that emerge from the cell body. An axon is a cellular extension that emerges from this cell body. Most neurons receive signals through the dendrites and send out signals along the axon. At the majority of synapses, signals cross from the axon of one neuron to the dendrite of another. All neurons are electrically excitable due to the maintenance of voltage gradients in their membranes. If the voltage changes by a large enough amount over a short interval, the neuron generates an electrochemical pulse called an action potential. This potential travels rapidly along the axon and activates synaptic connections as it reaches them.

1.4.3. ARTIFICIAL NEURAL NETWORK

Neural networks enable us to perform many tasks, such as clustering, classification or regression. With neural networks, we can group or sort unlabeled data according to similarities among the samples in this data. Or in the case of classification, we can train the network on a labeled dataset in order to classify the samples in this dataset into different categories. Artificial neural networks have unique capabilities that enable deep learning models to solve tasks that machine learning models can never solve.All recent advances in artificial intelligence in recent years are due to deep learning. Without deep learning, we would not have self-driving cars, chatbots or personal assistants like Alexa and Siri. The Google Translate app would continue to be as primitive as 10 years ago (before Google switched to neural networks for this App), and Netflix or Youtube would have no idea which movies or TV series we like or dislike. Behind all these technologies are neural networks. We can even go so far as to say that today a new industrial revolution is taking place, driven by artificial neural networks and deep learning.

At the end of the day, deep learning is the best and most obvious approach to real machine intelligence we've had so far. Artificial neural networks are inspired by the biological neurons that are found in our brains. In fact, the artificial neural networks simulate some basic functionalities of the neural networks in our brain, but in a very simplified way. In short, a biological neural network consists of numerous neurons. If the voltage changes by a large enough amount over a short interval, the neuron generates an electrochemical pulse called an action potential. This potential travels rapidly along the axon and activates synaptic connections as it reaches them. In general, neural networks can perform the same tasks as classical algorithms of machine learning. However, it is not the other way around.

NODE

A neural network generally consists of a collection of connected units or nodes. We call these nodes neurons. These artificial neurons loosely model the biological neurons of our brain. A neuron is simply a graphical representation of a numeric value (e.g. 1.2, 5.0, 42.0, 0.25, etc.).

WEIGHTS

Any connection between two artificial neurons can be considered as an axon in a real biological brain.The connections between the neurons are realized by so-called weights, which are also nothing more than numerical values.

ABSTRACTED WORKING

When an artificial neural network learns, the weights between neurons are changing and so does the strength of the connection Meaning: Given training data and a particular task such as classification of numbers, we are looking for certain set weights that allow the neural network to perform the classification. The set of weights is different for every task and every dataset. We can not predict the values of these weights in advance, but the neural network has to learn them. The process of learning we also call as training.

1.4.4. WORKING OF NEURAL NETWORK WITH ARCHITECTURE

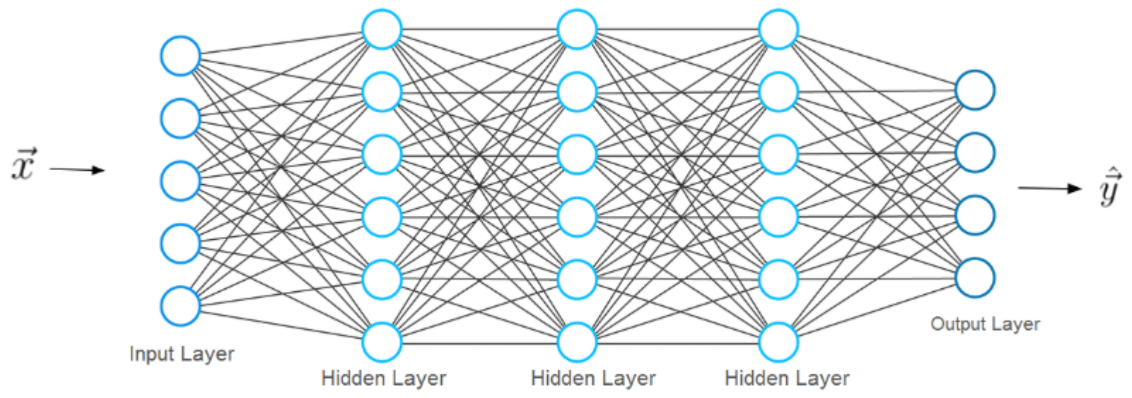
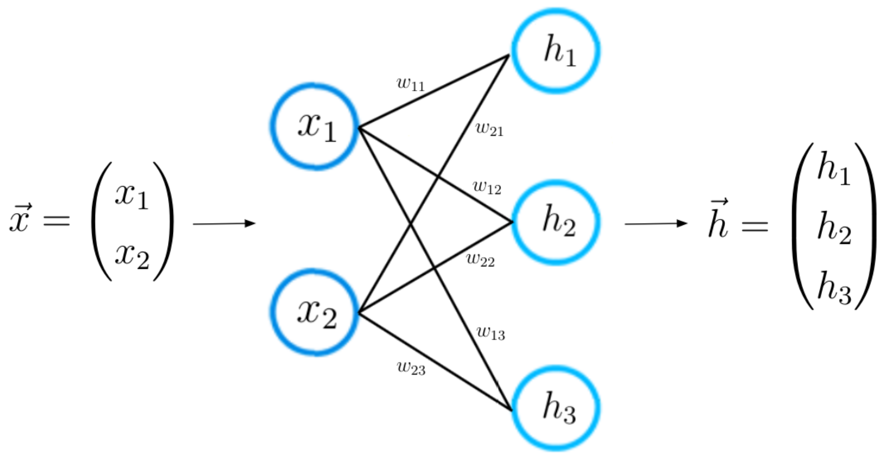


Fig.1.4 Artificial Neural Network Layers

The typical neural network architecture consists of several layers. We call the first layer as the input layer. The input layer receives the input x, data from which the neural network learns. For example, for classification of handwritten numbers, these input x would represent the images of these numbers (x is basically an entire vector where each entry is a pixel).The input layer has the same number of neurons as there are entries in the vector x which means each input neuron represents one element in the vector x.

The last layer is called the output layer, which outputs a vector y representing the result that the neural network came up with. The entries in this vector represent the values of the neurons in the output layer. For example classification, each neuron in the last layer would represent a different class. If the classification is a handwritten digit recognition, the input x belongs to one of the possible classes (one of the digits 0–9). As you can imagine the number of output neurons must be the same as there are classes. In order to obtain a prediction vector y, the network must perform certain mathematical operations. These operations are performed in the layers between the input and output layers. We call these layers the hidden layers. For a given input feature vector x, the neural network calculates a prediction vector, which we call here as h.



$$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \longrightarrow \quad \longrightarrow \vec{h} = \begin{pmatrix} h_1 \\ h_2 \\ h_3 \end{pmatrix}$$

Fig.1.5 Forward Propagation

This step is also referred to as the forward propagation. With the input vector x and the weight matrix W connecting the two neuron layers, we compute the dot product between the vector x and the matrix W.

$$\begin{aligned} \vec{x}^T \cdot W &= \begin{pmatrix} x_1, & x_2 \end{pmatrix} \cdot \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{pmatrix} \\ &= \begin{pmatrix} x_1 w_{11} + x_2 w_{21}, & x_1 w_{12} + x_2 w_{22}, & x_1 w_{13} + x_2 w_{23} \end{pmatrix} \\ &= \begin{pmatrix} z_1, z_2, z_3 \end{pmatrix} = \vec{z}, \ \vec{h} = \sigma(\vec{z}) \end{aligned}$$

z. The final prediction vector h is obtained by applying a so-called activation function to the vector z. In this case, the activation function is represented by the letter Sigma. An activation function is only a nonlinear function that performs a nonlinear mapping from z to h. There are 3 most common activation functions that are used in Deep Learning, which are tanh, sigmoid, and ReLu.

During training, these weights are adjusted, some neurons become more connected, some neurons become less connected. As in a biological neural network, learning means the alteration of weights. Accordingly, the values of z, h and the final output vector y are changing with the weights. Some weights make the predictions of a neural network closer to the actual ground truth vector y_hat, some weights increase the distance to the ground truth vector.

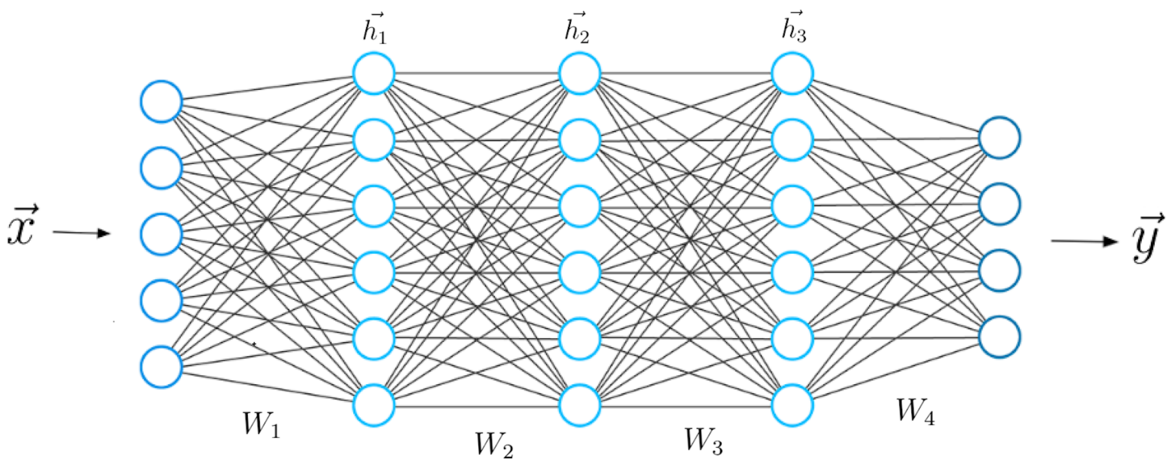


Fig.1.6 Neural Network Working

$\mathscr{X}$ - The input to the neural network. → indicates it is a vector

$\mathscr{Y}$ - The output of the neural network. → indicates it is a vector

W - The weights of the network

h - The layers of the network

We calculate the dot product between the input x and the first weight matrix W1 and apply an activation function to the resulting vector to obtain the first hidden vector h1. h1 is now considered as the input for the upcoming third layer. The whole procedure from before is repeated until we obtain the final output y. After we get the prediction of the neural network, in the second step we must compare this prediction vector to the actual ground truth label. We call the ground truth label as vector y_hat. While the vector y contains the predictions that the neural network has computed during the forward propagation (and which may, in fact, be very different from the actual values), the vector y_hat contains the actual values. Mathematically, we can measure the difference between y and y_hat by defining a loss function in which the value depends on this difference between them.

$$\mathcal{L}(\theta) = \frac{1}{2}(\vec{y} - \hat{\vec{y}})^2$$

Minimizing the loss function directly leads to more accurate predictions of the neural network, as the difference between the prediction and the label decreases. Minimizing the loss function automatically causes the neural network model to make better predictions regardless of the exact characteristics of the task at hand. You only have to select the right loss function for the task. Fortunately, there are only two loss functions that you should know about to solve almost any problem that you encounter in practice. These loss-functions are the Cross-Entropy Loss and the Mean Squared Error Loss.

During gradient descent, we use the gradient of a loss function (or in other words the derivative of the loss function) to improve the weights of a neural network. In the first step, we must choose a loss function for the task. The goal now is to repeatedly update the weight parameter until we reach the optimal value for that particular weight. This is the time when we need to use the gradient of the loss function. Each time we are performing the update of the weights, we move down the negative gradient towards the optimal weights.

$$w_{11_{new}} = w_{11_{old}} - \epsilon \cdot \nabla_{w_{11}} \mathcal{L}(w_{11})$$

The factor epsilon in this equation is a hyperparameter called the learning rate. The learning rate determines how quickly or how slowly you want to update the parameters. Please keep in mind that the learning rate is the factor with which we have to multiply the negative gradient and that the learning rate is usually quite small. After each gradient descent step or weight update, the current weights of the network get closer and closer to the optimal weights until we eventually reach them and the neural network will be capable of doing the predictions we want to make.

1.4.5. APPLICATION

Some of the most dramatic improvements brought about by deep learning have been in the field of computer vision. For decades, computer vision relied heavily on image processing methods, which means a whole lot of manual tuning and specialization. Deep learning, on the other hand, ignores nearly all traditional image processing, and it has resulted in dramatic improvements to every computer vision task.

The main application of the model is designed for autonomous cars.

Level 1 automation some small steering or acceleration tasks are performed by the car without human intervention, but everything else is fully under human control.

Level 2 automation is like advance cruise control or original autopilot system on some Tesla vehicles, the car can automatically take safety actions but the driver needs to stay alert at the wheel.

Level 3 automation still requires a human driver, but the human is able to put some "safety-critical functions" to the vehicle, under certain traffic or environmental conditions. This poses some potential dangers as humans pass the major tasks of driving to or from the car itself, which is why some car companies (Ford included) are interested in jumping directly to level 4.

Level 4 automation is a car that can drive itself almost all the time without any human input, but might be programmed not to drive in unmapped areas or during severe weather. This is a car you could sleep in.

Level 5 automation means full automation in all conditions.

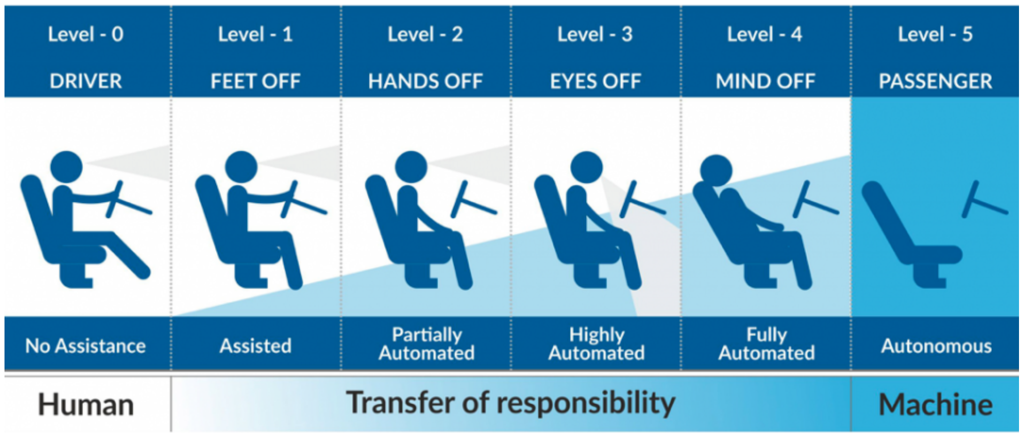


Fig.1.7 Automation levels in car

Since these levels don't mean much to people outside the industry, car makers often don't talk about their technology in these specific SAE terms. The big potential promise for people is either cars that drive themselves for a large part of a person's highway commute (level 3) or cars that can drive themselves almost as long as you live in a covered metropolitan area (level 4).

Most executives referring to "self-driving" are referring to levels 3 and 4. Whenever possible, we distinguish between what kind of self-driving a specific executive is speaking about (i.e. Highway autonomy, or full metropolitan autonomy). Full autonomy is level 5 and is still an area in which a lot of MNC has invested a huge amount on R&D for cars without human intervention in driving. It takes about 16 years for them to be ready for the road because of the certain kilometers it has to achieve without causing any problems.

CHAPTER 2
2. LITERATURE REVIEW

2.1 Title : An Introduction To Convolutional Neural Networks

Author : Keiron Teilo O'shea

The field of machine learning has taken a dramatic twist in recent times, with the rise of the Artificial Neural Network (ANN). These biologically inspired computational models are able to far exceed the performance of previous forms of artificial intelligence in common machine learning tasks. One of the most impressive forms of ANN architecture is that of the Convolutional Neural Network (CNN). CNNs are primarily used to solve difficult image-driven pattern recognition tasks and with their precise yet simple architecture, offers a simplified method of getting started with ANNs. This document provides a brief introduction to CNNs, discussing recently published papers and newly formed techniques in developing these brilliantly fantastic image recognition models. This introduction assumes you are familiar with the fundamentals of ANNs and machine learning.

2.2 Title : Understanding Of A Convolutional Neural Network

Author : Saad Albawi; Tareq Abed Mohammed; Saad Al-zawi

The term Deep Learning or Deep Neural Network refers to Artificial Neural Networks (ANN) with multi layers. Over the last few decades, it has been considered to be one of the most powerful tools, and has become very popular in the literature as it is able to handle a huge amount of data. The interest in having deeper hidden layers has recently begun to surpass classical methods performance in different fields; especially in pattern recognition.

One of the most popular deep neural networks is the Convolutional Neural Network (CNN). It takes this name from mathematical linear operation between matrices called convolution. CNN has multiple layers; including convolutional layer, non-linearity layer, pooling layer and fully-connected layer. The convolutional and fully-connected layers have parameters but pooling and non-linearity layers don't have parameters.

CNN has an excellent performance in machine learning problems. Specially the applications that deal with image data, such as largest image classification data set (Image Net), computer vision, and in natural language processing (NLP) and the results achieved were very amazing. In this paper we will explain and define all the elements and important issues related to CNN, and how these elements work. In addition, we will also state the parameters that affect CNN efficiency. This paper assumes that the readers have adequate knowledge about both machine learning and artificial neural networks.

2.3 Title : A Comprehensive Analysis Of Deep Regression

Author : St´ephane Lathuili`Ere, Pablo Mesejo, Xavier Alameda-pineda

Deep learning revolutionized data science, and recently its popularity has grown exponentially, as did the amount of papersemploying deep networks. Vision tasks, such as human pose estimation, did not escape from this trend.

There is a large number of deep models, where small changes in the network architecture, or in the data pre-processing, together with the stochastic nature of the optimization procedures, produce notably different results, making extremely difficult to sift methods that significantly outperform others. This situation motivates the current study, in which we perform a systematic evaluation and statistical analysis of vanilla deep regression, i.e. convolutional neural networks with a linear regression top layer. This is the first comprehensive analysis of deep regression techniques.

We perform experiments on four vision problems, and report confidence intervals for the median performance as well as the statistical significance of the results, if any. Surprisingly, the variability due to different data preprocessing procedures generally eclipses the variability due to modifications in the network architecture. Our results reinforce the hypothesis according to which, in general, a general-purpose network (e.g. VGG-16 or ResNet-50) adequately tuned can yield results close to the state-of-the-art without having to resort to more complex and ad-hoc regression mode

2.4. Title : An End-to-end Deep Neural Network For Autonomous Driving Designed For Embedded Automotive Platforms

Author : Jelena Kocić,* Nenad Jovičić, And Vujo Drndarević

In this paper, one solution for an end-to-end deep neural network for autonomous driving is presented. The main objective of our work was to achieve autonomous driving with a light deep neural network suitable for deployment on embedded automotive platforms. There are several end-to-end deep neural networks used for autonomous driving, where the input to the machine learning algorithm are camera images and the output is the steering angle prediction, but those convolutional neural networks are significantly more complex than the network architecture we are proposing. The network architecture, computational complexity, and performance evaluation during autonomous driving using our network are compared with two other convolutional neural networks that we re-implemented with the aim to have an objective evaluation of the proposed network.
The trained model of the proposed network is four times smaller than the PilotNet model and about 250 times smaller than AlexNet model. While complexity and size of the novel network are reduced in comparison to other models, which leads to lower latency and higher frame rate during inference, our network maintained the performance, achieving successful autonomous driving with similar efficiency compared to autonomous driving using two other models. Moreover, the proposed deep neural network downsized the needs for real-time inference hardware in terms of computational power, cost, and size.

2.5. Title : The Yan: [Self Driving Car Using Deep Learning]

Author : Ms. Sujeetha, Chitrak Bari, Gareja Prdip, Siddhant Purohit

When an automobile is on the road, there is a driver at the steering wheel and he is incharge. Driverless technology is being developed and tested by universities, institutions, and companies for quite some time. The idea of a car without a driver gives us a feeling of skepticism and we all will try to avoid it. In this paper, we are discussing an autonomous robotically handled driving vehicle. In our project, we are using many features such as mapping, tracking and local planning. We can successfully create a car that can demonstrate proper lane changes, parking, and

U-turns on its own. The different innovations we are using are obstacles and curb detection methods, road vehicle trackers, and checking different traffic situations. This will make a robust autonomous self driven car. It will successfully demonstrate proper parking allotment, lane changes, and automatic U-turns. We can do this using the obstacle and various curb detection methods, the vehicle tracker.

2.6 Title : End To End Learning For Self-driving Cars

Author : Nvidia Corporation

We trained a convolutional neural network (CNN) to map raw pixels from a sin-gle front-facing camera directly to steering commands. This end-to-end approach proved surprisingly powerful. With minimum training data from humans the sys-tem learns to drive in traffic on local roads with or without lane markings and on highways. It also operates in areas with unclear visual guidance such as in parking lots and on unpaved roads. The system automatically learns internal representations of the necessary process-ing steps such as detecting useful road features with only the human steering angle as the training signal. We never explicitly trained it to detect, for example, the out-line of roads. Compared to explicit decomposition of the problem, such as lane marking detec-tion, path planning, and control, our end-to-end system optimizes all processing steps simultaneously.

We argue that this will eventually lead to better perfor-mance and smaller systems. Better performance will result because the internal components self-optimize to maximize overall system performance, instead of op-timizing human-selected intermediate criteria, e. g., lane detection. Such criteria understandably are selected for ease of human interpretation which doesn't auto-matically guarantee maximum system performance. Smaller networks are possi-ble because the system learns to solve the problem with the minimal number of processing steps.

We used an NVIDIA DevBox and Torch 7 for training and an NVIDIA DRIVER MPX self-driving car computer also running Torch 7 for determining where to drive. The system operates at 30 frames per second (FPS).

The CNNs are able to learn the entire task of lane and road following without manual decomposition into road or lane marking detection, semantic abstraction,path planning, and control. A small amount of training data from less than a hundred hours of driving was sufficient to train the car to operate in diverse conditions, on highways, local and residential roads in sunny, cloudy, and rainy conditions. The CNN is able to learn meaningful road features from a very sparse training signal.

2.7. Title : Deeppicar: A Low-cost Deep Neural Network-basedautonomous Car

Author : Michael G. Bechtel, Elise Mcellhiney, Minje Kim, Heechul Yun

We present DeepPicar, a low-cost deep neural net-work based autonomous car platform. DeepPicar is a small scale replication of a real self-driving car called DAVE-2 by NVIDIA.DAVE-2 uses a deep convolutional neural network (CNN), which takes images from a front-facing camera as input and produces car steering angles as output. DeepPicar uses the same net-work architecture—9 layers, 27 million connections and 250Kparameters—and can drive itself in real-time using a web camera and a Raspberry Pi 3 quad-core platform. Using DeepPicar, we analyze the Pi 3's computing capabilities to support end-to-end deep learning based real-time control of autonomous vehicles.
We also systematically compare other contemporary embedded computing platforms using the DeepPicar's CNN-based real-time control workload.We find that all tested platforms, including the Pi 3, are capable of supporting the CNN-based real-time control, from20 Hz up to 100 Hz, depending on hardware platform. However,we find that shared resource contention remains an important issue that must be considered in applying CNN models on shared memory based embedded computing platforms; we observe upto 11.6X execution time increase in the CNN based control loop due to shared resource contention.

To protect the CNN workload,we also evaluate state-of-the-art cache partitioning and memory bandwidth throttling techniques on the Pi 3. We find that cache partitioning is ineffective, while memory bandwidth throttling is an effective solution

CHAPTER 3

3. SYSTEM ANALYSIS

3.1. PROBLEM DEFINITION

The problem definition is to train a convolutional neural network (CNN) to map raw pixels from a sin-gle front-facing camera directly to steering commands. This end-to-end approach should be powerful such that with minimum training data from humans the sys-tem learns to drive in traffic on local roads with or without lane markings and on highways. CNNs have revolutionized pattern recognition. Prior to the widespread adoption of CNNs,most pattern recognition tasks were performed using an initial stage of hand-crafted feature extrac-tion followed by a classifier. The breakthrough of CNNs is that features are learned automatically from training examples. The CNN approach is especially powerful in image recognition tasks because the convolution operation captures the 2D nature of images.

Also, by using the convolution kernels to scan an entire image, relatively few parameters need to be learned compared to the total number of operations. While CNNs with learned features have been in commercial use for over twenty years, their adoption has exploded in the last few years because of two recent developments. First, large, labeled data sets such as the Large Scale Visual Recognition Challenge (ILSVRC) have become available for training and validation. Second, CNN learning algorithms have been implemented on the massively parallel graphics processing units (GPUs) which tremendously accelerate learning and inference.

3.2. EXISTING SYSTEM

The existing system requires separate lane marking detection, path planning, object detection and other control models whose outputs are combined to achieve autonomy which requires a strenuous amount of computational power. The training amount required to build those models will be huge as well as the inference time that is while deploying the models at an actual car and making predictions for achieving autonomy. Compared to explicit decomposition of the problem, such as lane marking detection, path planning, and control, our end-to-end system optimizes all processing steps simultaneously.

3.3. PROPOSED SYSTEM

We propose a CNN that goes beyond pattern recognition. It learns the entire processing pipeline needed to steer an automobile. Images are fed into a CNN which then computes a proposed steering command. The proposed command is compared to the desired command for that image and the weights of the CNN are adjusted to bring the CNN output closer to the desired output. The weight adjustment is accomplished using back propagation. Precise viewpoint transformation requires 3D scene knowledge which we don’t have. We therefore approximate the transformation by assuming all points below the horizon are on flat ground and all points above the horizon are infinitely far away. This works fine for flat terrain but it introduces distortions for objects that stick above the ground, such as cars,poles, trees, and buildings. Fortunately these distortions don’t pose a big problem for network train-ing. The steering label for transformed images is adjusted to one that would steer the vehicle back to the desired location and orientation in two seconds.

3.4. ADVANTAGES OF THE PROPOSED SYSTEM

1. Less time for training and inference at deploy
2. Computationally very efficient
3. Real time implementation compatibility
4. End to end approach

CHAPTER 4

4. SYSTEM REQUIREMENTS

4.1. HARDWARE REQUIREMENTS

● CPU Type	:	Intel i7
● Clock Speed	:	2.80 GHz
● RAM Size	:	8 GB
● GPU Size	:	4 GB
● Hard Disk	:	300 GB
● Monitor Type	:	15 inch color monitor
● Keyboard Type	:	Internet keyboard
● Accelerator	:	GPU - CUDA

4.2 SOFTWARE REQUIREMENTS

● Operating System	:	Windows OS 10
● Language	:	Python
● Tool	:	Jupyter notebook, Google colab
● Libraries	:	pandas, NumPy, pickle, keras, matplotlib, tensorflow

4.3 SOFTWARE SPECIFICATION

JUPYTER NOTEBOOK

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modelling, data visualization, machine learning, and much more. The Jupyter Notebook is based on a set of open standards for interactive computing. Think HTML and CSS for interactive computing on the web. These open standards can be leveraged by third party developers to build customized applications with embedded interactive computing. Programs in Jupyter notebooks are different from typical programs as they are constructed by a collection of code snippets interleaved with text and visualisation. This allows interactive exploration and snippets may be executed in different order which may give rise to different results due to side-effects between snippets

GOOGLE COLAB

Google Colab is a free Jupyter notebook environment that runs entirely in the cloud. Most importantly, it does not require a setup and the notebooks that you create can be simultaneously edited by your team members - just the way you edit documents in Google Docs. Colab supports many popular machine learning libraries which can be easily loaded in your notebook. allows you to write and execute Python in your browser, with

- Zero configuration required
- Free access to GPUs
- Easy sharing

Colab notebooks allow you to combine executable code and rich text in a single document, along with images, HTML, LaTeX and more. With Colab you can harness the full power of popular Python libraries to analyse and visualise data.

PYTHON

Python is an interpreted, high-level and general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant indentation. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects. Python is dynamically-typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library. Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently whereas other languages use punctuation, and it has fewer syntactic constructions than other languages. Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. Python supports Object-Oriented style or technique of programming that encapsulates code within objects.

CHARACTERISTICS OF PYTHON

Some of the important characteristics of Python Programming are

- It supports functional and structured programming methods as well as OOP.

- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

LIBRARIES

PANDAS

Pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis / manipulation tool available in any language. It is already well on its way toward this goal.

pandas is well suited for many different kinds of data:

- Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet
- Ordered and unordered (not necessarily fixed-frequency) time series data.
- Arbitrary matrix data (homogeneously typed or heterogeneous) with row and column labels
- Any other form of observational / statistical data sets. The data actually need not be labelled at all to be placed into a pandas data structure

Robust IO tools for loading data from flat files (CSV and delimited), Excel files, databases, and saving / loading data from the ultrafast HDF5 format. Time series-specific functionality: date range generation and frequency conversion, moving window statistics, date shifting and lagging.

NUMPY

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. It brings the computational power of languages like C and Fortran to Python, a language much easier to learn and use. It provides a simple yet powerful data structure: the n-dimensional array. This is the foundation on which almost all the power of Python's data science toolkit is built. It take advantage of two important concepts at once:

- Vectorization
- Broadcasting

Vectorization is the process of performing the same operation in the same way for each element in an array. Broadcasting is the process of extending two arrays of different shapes and figuring out how to perform a vectorized calculation between them.

TENSORFLOW

TensorFlow is a computational framework for building machine learning models. It is the second generation system from Google Brain headed by Jeff Dean. Launched in early 2017, it has disrupted the ML world by bringing in numerous capabilities from scalability to building production ready models.

TensorFlow provides a variety of different tool kits that allow you to write code at your preferred level of abstraction. For instance, you can write code in the Core TensorFlow (C++) and call that method from Python code. You can also define the architecture on which your code should run (CPU, GPU etc.). In the above hierarchy, the lowest level in which you can write your code is C++ or Python.

These two levels allow you to write numerical programs to solve mathematical operations and equations. Although this is not highly recommended for building Machine Learning models, it offers a wide range of math libraries that ease your tasks. The next level in which you can write your code is using the TF specific abstract methods which are highly optimised for model components. For example, using the tf.layers method abstract you can play with the layers of a neural net. You can build a model and evaluate the model performance using the tf.metrics method. The most widely used level is the tf.estimator API, which allows you to build (train and predict) production ready models with ease.

The estimator API is insanely easy to use and well optimised. Although it offers less flexibility, it has all that is needed to train and test your model. The basic data type in this framework is a Tensor. A Tensor is an N-dimensional array of data. For instance, you can call a Scalar (a constant value such as integer 2) as a 0-dimension tensor. A vector is a

1-dimensional tensor and a matrix is a 2-dimensional tensor. Mostly TensorFlow is used as a backend framework whose modules are called through Keras API. Typically, TensorFlow is used to solve complex problems like Image Classification, Object Recognition, Sound Recognition, etc.

KERAS

Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library. Keras is a high-level neural networks library that is running on the top of TensorFlow, CNTK, and Theano. Using Keras in deep learning allows for easy and fast prototyping as well as running seamlessly on CPU and GPU. This framework is written in Python code which is easy to debug and allows ease for extensibility. Keras is an API designed for human beings, not machines. Keras follows best practices for reducing cognitive load: it offers consistent & simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear & actionable error messages. It also has extensive documentation and developer guides.

The main advantages of Keras are described below:

- **User-Friendly:** Keras has a simple, consistent interface optimized for common use cases which provides clear and actionable feedback for user errors.
- **Modular and Composable:** Keras models are made by connecting configurable building blocks together, with few restrictions.
- **Easy To Extend:** With the help of Keras, you can easily write custom building blocks for new ideas and research.
- **Easy To Use:** Keras offers consistent & simple APIs which helps in minimizing the number of user actions required for common use cases, also it provides clear and actionable feedback upon user error.

PICKLE

Python pickle module is used for serializing and de-serializing python object structures. Python pickle module is used for serializing and de-serializing a Python object structure. Any object in Python can be pickled so that it can be saved on disk. What pickle does is that it "serializes" the object first before writing it to file. Pickling is a way to convert a python object (list, dict, etc.) into a character stream. The idea is that this character stream contains all the information necessary to reconstruct the object in another python script. The process to convert any kind of python objects (list, dict, etc.) into byte streams (0s and 1s) is called pickling or serialization or flattening or marshalling. We can convert the byte stream (generated through pickling) back into python objects by a process called as unpickling.

- Comes handy to save complicated data.
- Easy to use, lighter and doesn’t require several lines of code.
- The pickled file generated is not easily readable and thus provide some security.

MATPLOTLIB

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

Matplotlib produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shell, web application servers, and various graphical user interface toolkits. Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python

CHAPTER 5

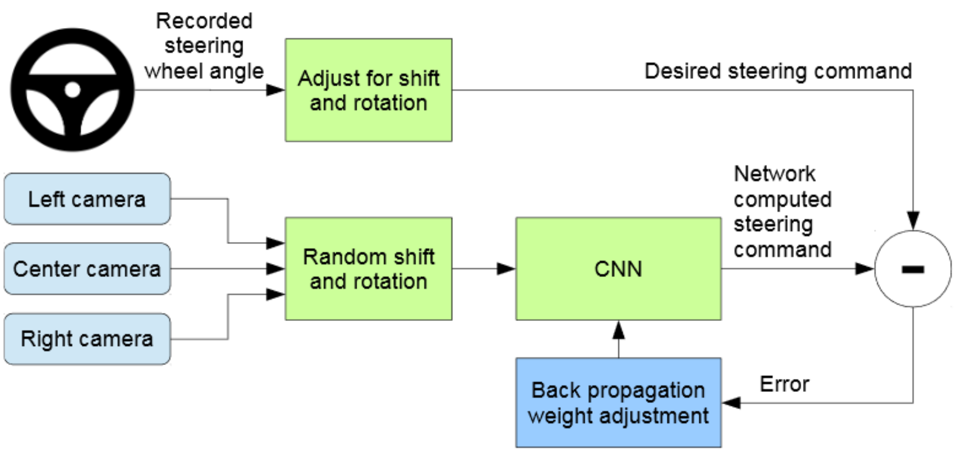
5. SYSTEM DESIGN

5.1. SYSTEM ARCHITECTURE

A system architecture or systems architecture is the conceptual model that defines the structure, behavior and more views of a system. An architecture description is a formal description and representation of a system, organized in a way that supports reasoning about the structures and behaviours of the system. System architecture can comprise system components, the externally visible properties of those components, the relationships (e.g. the behavior) between them. It can provide a plan from which products can be procured, and systems developed, that will work together to implement the overall system. There have been efforts to formalize languages to describe system architecture, collectively these are called architecture description languages (ADLs).

Various organizations define systems architecture in different ways including

- An allocated arrangement of physical elements which provides the design solution for a consumer product or life-cycle process intended to satisfy the requirements of the functional architecture and the requirements baseline.
- Architecture comprises the most important, pervasive, top-level, strategic inventions, decisions and their associated rationales about the overall structure (i.e. essential elements and their relationships) and associated characteristics and behaviour.
- If documented, it may include information such as a detailed inventory of current hardware, software and networking capabilities.


Recorded steering wheel angle
Adjust for shift and rotation
Desired steering command
Left camera
Center camera
Right camera
Random shift and rotation
CNN
Network computed steering command
-
Back propagation weight adjustment
Error

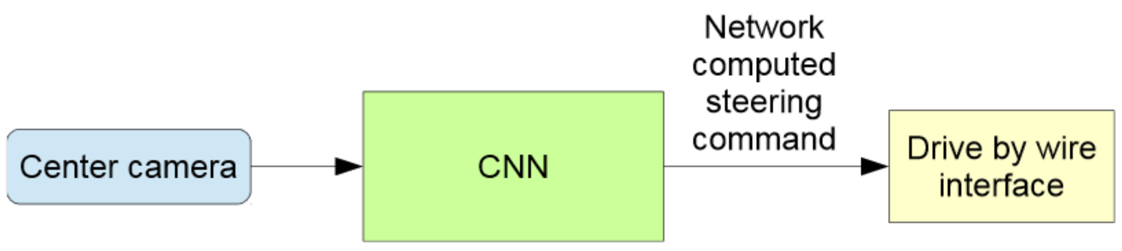


Fig 5.1 Architecture Design

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

5.2 CLASS DIAGRAM

A class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects. The class diagram is the main building block of object-oriented modeling. It is used for general conceptual modeling of the structure of the application, and for detailed modeling translating the models into programming code. Class diagrams can also be used for data modeling. The classes in a class diagram represent both the main elements, interactions in the application, and the classes to be programmed.

In the diagram, classes are represented with boxes that contain three compartments:

- The top compartment contains the name of the class. It is printed in bold and centered, and the first letter is capitalized.
- The middle compartment contains the attributes of the class. They are left-aligned and the first letter is lowercase.
- The bottom compartment contains the operations the class can execute. They are also left-aligned and the first letter is lowercase.

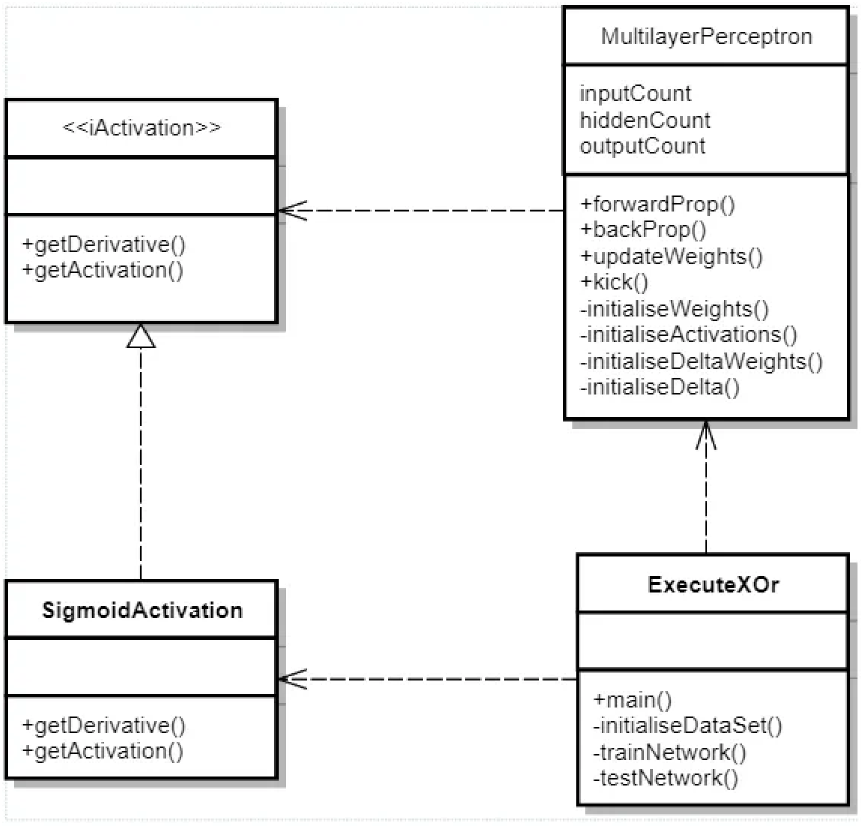
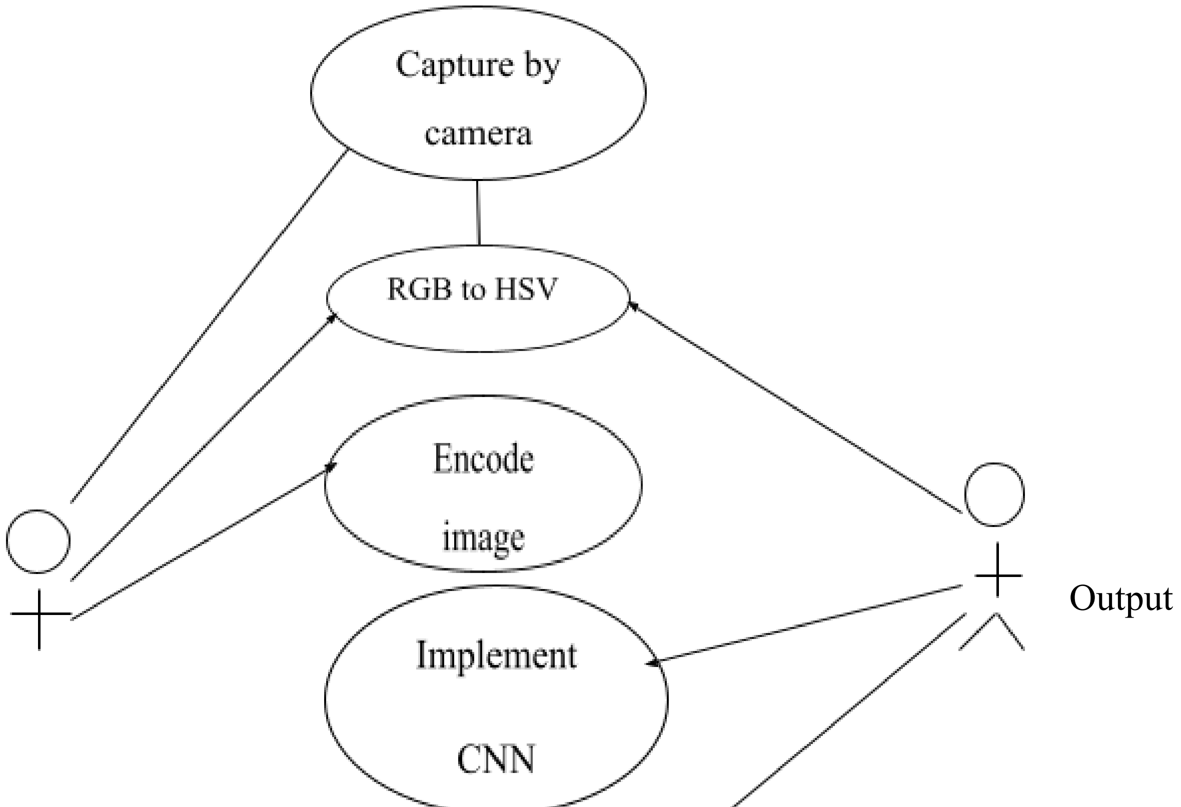


Fig 5.2 Class Diagram

5.3 USE CASE DIAGRAM

A use case diagram is a dynamic or behavior diagram in UML. Use case diagrams model the functionality of a system using actors and use cases. Use cases are a set of actions, services, and functions that the system needs to perform. Use case diagrams are usually referred to as behavior diagrams used to describe a set of actions (use cases).



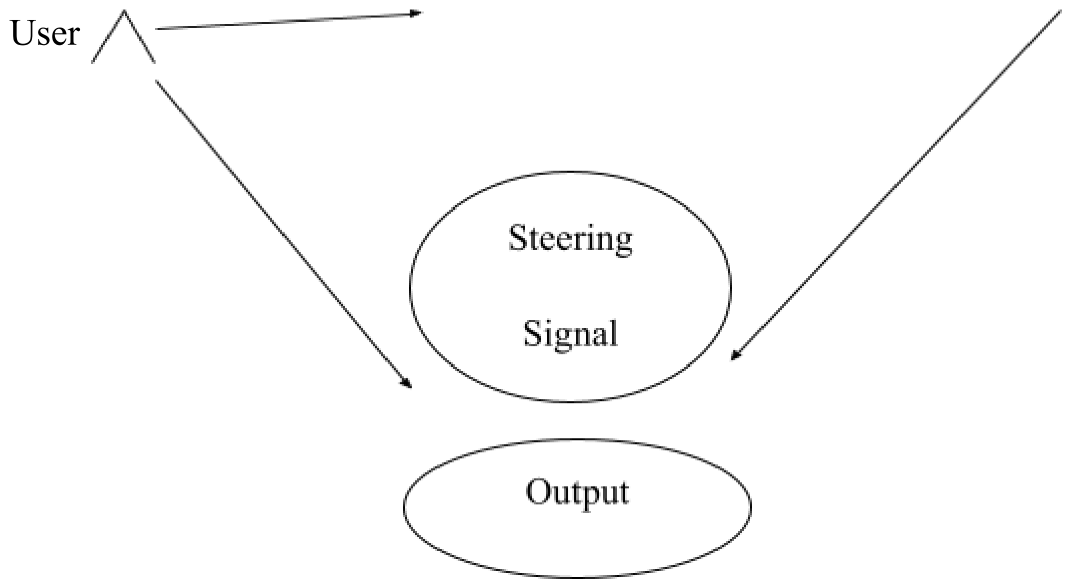


Fig.5.3 Usecase Diagram

5.4 FLOW DIAGRAM

The DFD takes an input-process-output view of a system in which data objects flow into the software are transformed by processing elements and resultant data objects flow out of the software. Data objects represented by labelled arrows and transformations are represented by circles also called as bubbles. DFD is presented in a hierarchical fashion that is the first data flow model that represents the system as a whole. Subsequent DFD refined the context diagram (level 0 DFD), providing increasing details with each subsequent level. The DFD enables the software engineer to develop models of the information domain and functional domain at the same time. As the DFD is refined into greater levels of details the analyst performs an implicit functional decomposition of the system. At the same time the DFD refinement results in a corresponding refinement of the data as it moves through the process that embodies the applications.

A context level DFD for the system the primary external entities produce information for use by the system and consume information generated by the system. The labeled arrow represents data objects or object hierarchy. They can be used to analyze an existing system or model a new one. Like all the best diagrams and charts, a DFD can often visually “say” things that would be hard to explain in words, and they work for both technical and nontechnical audiences. Data-flow diagrams can be regarded as inverted Petri nets, because places in such networks correspond to the semantics of data memories. Analogously, the semantics of transitions from Petri nets and data flows and functions from data-flow diagrams should be considered equivalent.

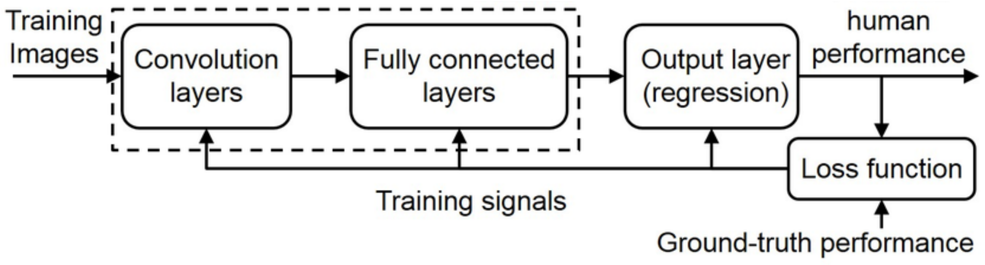
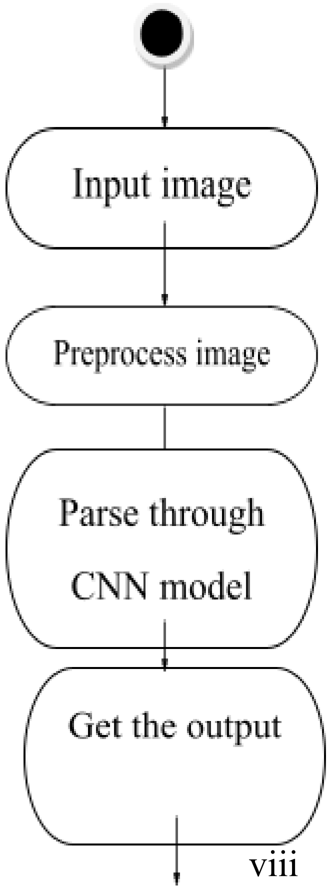


Fig. 5.4 Flow Diagram

5.5 ACTIVITY DIAGRAM

An activity diagram visually presents a series of actions or flow of control in a system similar to a flowchart or a data flow diagram. Activity diagrams are often used in business process modelling. They can also describe the steps in a use case diagram. Activities modeled can be sequential and concurrent. In both cases an activity diagram will have a beginning (an initial state) and an end (a final state). Activity is a particular operation of the system. Activity diagrams are not only used for visualizing the dynamic nature of a system.



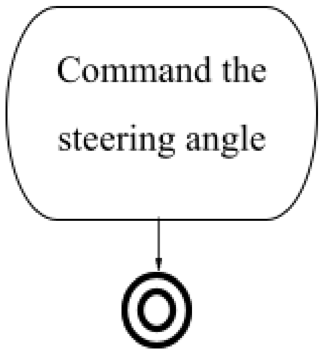


Fig .5.5 Activity Diagram

CHAPTER 6
6. MODULES DESCRIPTION

6.1 MODULE

A module is a separate unit of software or hardware. Typical characteristics of modular components include portability, which allows them to be used in a variety of systems and interoperability which allows them to function with the components of other systems. The modules used in this project are

- Data collection
- Data pre-processing
- Data modelling
- Data validation & testing
- Deployment

6.2. DATA COLLECTION

Data was recorded around Rancho Palos Verdes and San Pedro California. It contains approximately 63,000 images consuming 3.1GB of storage. The images are recorded from a car dashcam with labeled steering angles of each image in a .txt file. It is acquired from a public repository at github.

6.3. DATA PRE-PROCESSING

HSV is allegedly better than RGB for some computer-vision applications because it separates brightness from color. So the image is converted from RGB to HSV to account for the lighting differences in images and the data is stored in a pickle format which allows processing and accessing the data much faster and it can be easily converted into a numpy data type which is converted into tensor data type at the time of model running.

6.4 DATA MODELING

The models are mathematical algorithms that are trained using data. The training processes a large amount of data through an algorithm to maximize likelihood or minimize cost, yielding a trained model. Analysing data from many wells in different conditions, the model learns to detect all the types of patterns and distinguish these from normal operation. A model attempts to replicate a specific decision process that a team of experts would make if they could review all available data. The process of modeling means training a machine learning algorithm to predict the labels from the features, tuning it for the business need, and validating it on holdout data. The output from modeling is a trained model that can be used for inference, making predictions on new data points. Modeling is independent of the previous steps in the machine learning process and has standardized inputs which means we can alter the prediction problem without needing to rewrite all our code. In this project, I used the convolutional neural network for model training.

WHY NOT ARTIFICIAL NEURAL NETWORK

An image is nothing but a matrix of pixel values. So why not just flatten the image (e.g. 3x3 image matrix into a 9x1 vector) and feed it to a Multi-Level Perceptron for classification purposes; In cases of extremely basic binary images, the method might show an average precision score while performing prediction of classes but would have little to no accuracy when it comes to complex images having pixel dependencies throughout.

A ConvNet is able to successfully capture the Spatial and Temporal dependencies in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights. In other words, the network can be trained to understand the sophistication of the image better.

WHY CONVOLUTIONAL NEURAL NETWORK

The convolutional neural network (CNN) gives a great impact over the past few years in various domains. It is mostly used because it has more benefits over the other types of neural networks like Artificial Neural Network (ANN) as it has reduced the number of training parameters. In mathematics and image processing the term convolution means generating a function based on the operation of two different functions. Unlike image processing there is no need of pixel to pixel calculations in CNN as it has the aspect of extracting contextual information from the input data.

There is no need of paying attention where the features are located in the data which is the advantage of Deep Learning. It makes it so popular to handle huge amounts of data to process without any hardships.Instead of vector-like features in tabular data, and encoded text data, we deal here with an image. Images do not necessarily have labels, sub-labels for regions and therefore features should be extracted, or smartly reduced. Let’s say we needed to capture a line in an image. In order to capture such lines from the image, an intuitive solution is to use matrix filtering.

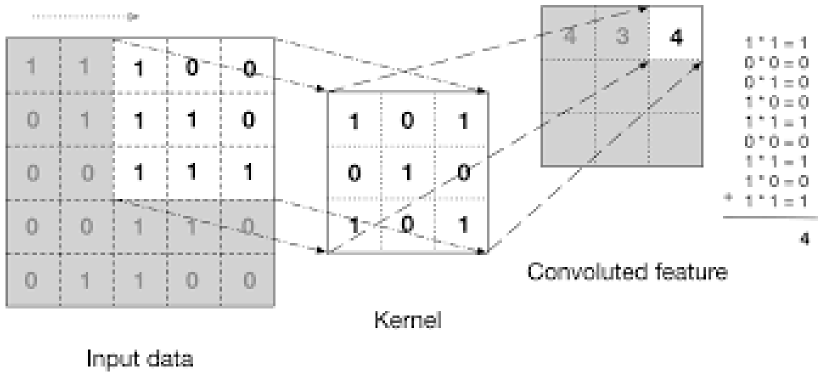


Fig.6.1 Convolution

Here the filter is a 3x3 matrix that has value 1 on its diagonal and 0 elsewhere. This filter will, therefore, be used as a mask for cross-shaped regions, or totally dark regions. Outputting a high value after entry-wise matrix multiplication means that the filter shape and the region analyzed matches. Note that matrices for filtering do not always have to be valued at 0 and 1, it can also have -1 in order to impose a strict shape search. Unlike what we did previously, for general task feature extraction, filters can have various sizes and when reaching too great of sizes, enumerating them or guessing these different filters become too difficult. Plus, most of the time we ourselves ignore what feature we should be extracting for the classification. A CNN does not require us to guess what filter we should use. Indeed the training of a CNN is the training of these filters which are weights for the network.

Thus a CNN can totally uncover details that humans would never notice, to determine for instance the age of someone on a picture. That is the strength of a CNN. In simpler words we give an input image and we humans know what will be the output so we adjust the values in the network so that we obtain our output. This principle is called Learning a problem which is the backbone of Machine Learning. Here the system learns the values of convolutional filters and the vectors associated with the calculation in the neural network.

CONVOLUTIONAL NEURAL NETWORK COMPONENTS:

The commonly used architecture of convolutional neural network consists of four important components

- Convolutional Layer
- Pooling layer
- Activation function
- Dropout
- Fully Connected layer

CONVOLUTIONAL LAYER

Convolution layer by default has the potential of image sampling and dimensionality reduction of the given input image. This makes the algorithm much more efficient. This layer is for extracting the features maps from the input image. Generally, convolution layer creates various feature maps of an image by applying filters on it.

In simple expression we can say that,

Z= X*f

Where

X is the input image

f is the filter

Always * refers to the convolution operation.

The feature maps are created by summing up the product values of each element of input and the filter value on the basis of element-wise. In this way an arbitrary number of feature maps are created for an image. There will be N number of distinct features created from a single layer of neural network and that will be passed on to the next layer of network. If the dimensionality of an image is (n,n) and filter is (f,f) then the dimension of the output vector will be ((n-f+1),(n-f+1)). It is made to pass through the number of convolution layers for distinct feature maps extraction.

An illustration of feature map generation is

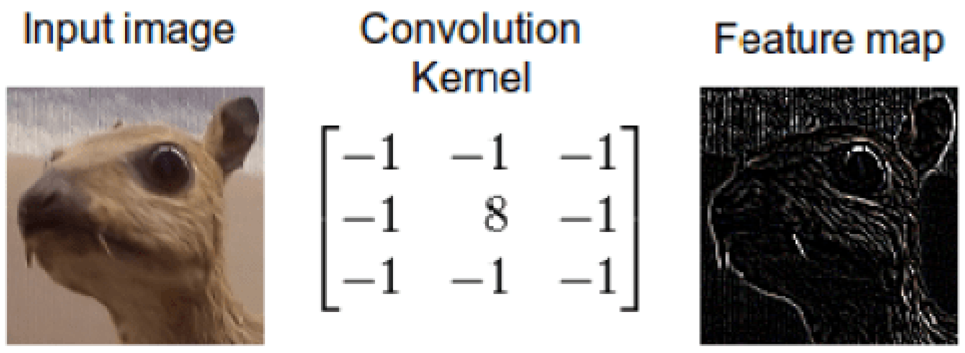

Input image
Convolution Kernel
$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$
Feature map

Fig.6.2 Feature Map

POOLING LAYER

In most cases, a Convolutional Layer is followed by a Pooling Layer. The primary aim of this layer is to decrease the size of the convolved feature map to reduce the computational costs. This is performed by decreasing the connections between layers and independently operates on each feature map**.** The pooling layer down samples the image which will result in lesser complexity of processing in the upcoming layers. Moreover, it creates a window in which the feature is found which depends on the application. In the pooling layer, a window of input vector is passed through a pooling function. The pooling layer generates the output vector. There are few pooling techniques there. One of the most commonly used pooling technique is max pooling. Max pooling creates various subdivisions of the image according to the pooling function specified and returns the maximum value of each subregion and forms an output vector. The advantage of pooling technique is it will reduce the number of trainable parameters which will increase the efficiency of the algorithm.

FULLY CONNECTED LAYER

The features extracted from the convolution and the pooling layer is made to parse into the fully connected layer. The fully connected layer will only work on the one-dimensional data. So, in order to feed the features maps created into this layer, the multi-dimensional features maps will be converted into the data of one-dimensional format which is called as flattening. Then the linear and non-linear transformation of the data will be done in the layer using the formula,

$$\mathbf{aij=\sigma((W*X)\ ij+b)}$$

where

X is the input vector

W is the weight vector

b is the bias(constant)

σ is the non-linearity function

The weight matrix will be automatically created by CNN using random initialization.

DROPOUT

Usually, when all the features are connected to the FC layer, it can cause overfitting in the training dataset. Overfitting occurs when a particular model works so well on the training data causing a negative impact in the model’s performance when used on new data. To overcome this problem, a dropout layer is utilised wherein a few neurons are dropped from the neural network during the training process resulting in reduced size of the model. On passing a dropout of 0.3, 30% of the nodes are dropped out randomly from the neural network.

ACTIVATION FUNCTION

If we design our neural network without any nonlinear function, it will give the linear data. Then it will not learn any complex pattern from the data. So we have to make some non-linear transformation of the outcome from each layer and also to normalize the values. In order to apply non-linearity to the data, we are applying some non-linear functions called activation functions. It is added to each layer of the fully connected layers and the convolutional layers of the neural network. There are many activation functions like sigmoid, ReLU, tanh, softmax, etc., Sigmoid is mostly used for binary classification which returns the output in the range of 0 to 1. But ReLU is better than the sigmoid activation function. The activation function will be chosen based on the type of the problem. This figure represents how the CNN model uses the low level feature like edges to detect is there a car or not as the network goes deeper.

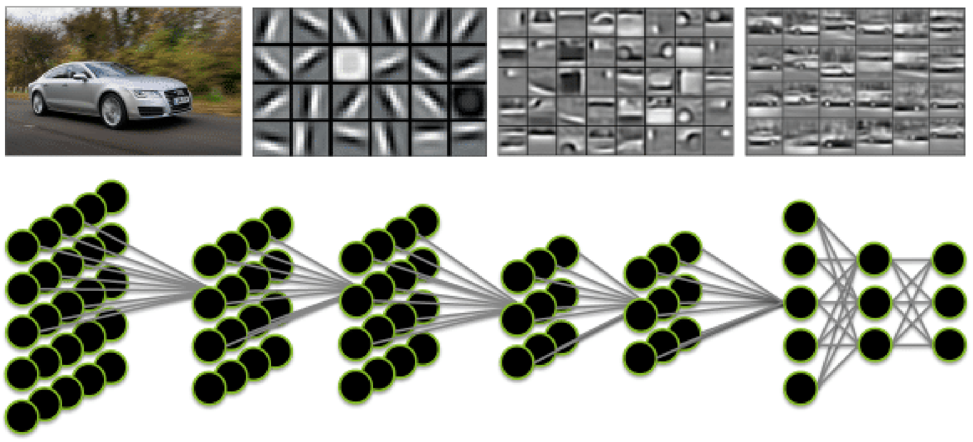


Fig.6.3 Feature Map Generation

ARCHITECTURE

Convolutional Neural Networks (CNNs) leverage spatial information, and they are therefore well suited for classifying images. These networks use an ad hoc architecture inspired by biological data taken from physiological experiments performed on the visual cortex. Our vision is based on multiple cortex levels, each one recognizing more and more structured information. First, we see single pixels, then from that we recognize simple geometric forms, and more sophisticated elements such as objects, faces, human bodies, animals, and so on.

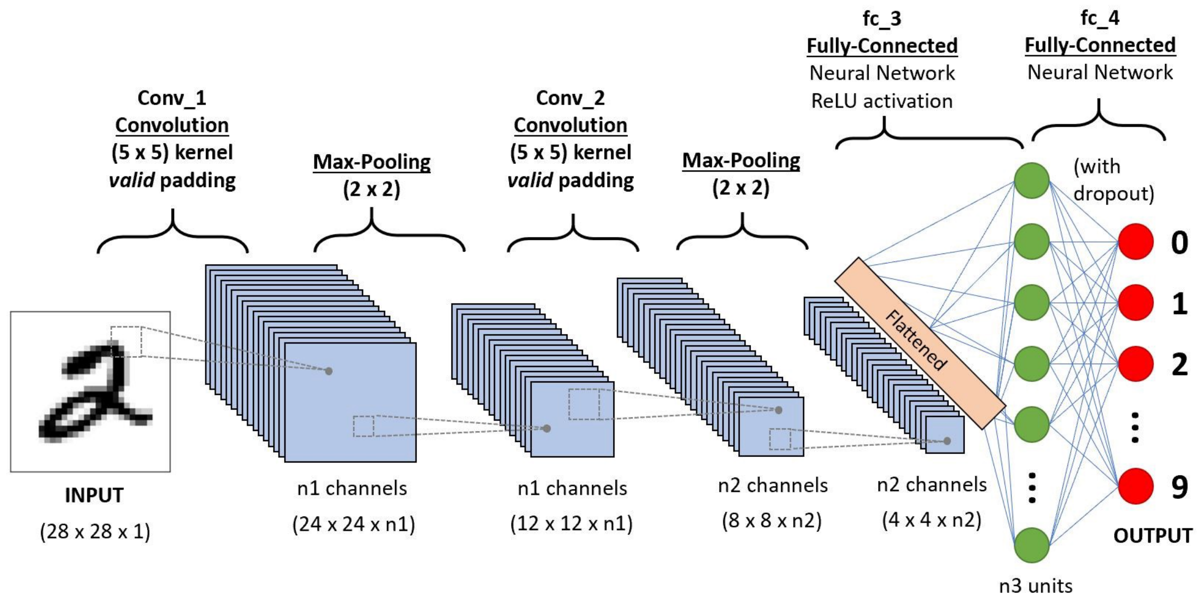


Fig.6.4 Architecture

The architecture is the collection of the entire components. The above figure is a simple CNN used for digit recognition. A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics. The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area. Convolutional neural networks (CNN) are among the more popular neural network frameworks that are used in complex applications like deep learning models for computer vision and image recognition.

Over the years, CNNs have undergone a considerable amount of rework and advancement. This has left us with a plethora of CNN models.Thus a CNN can totally uncover details that humans would never notice, to determine for instance the age of someone on a picture. That is the strength of a CNN. In simpler words we give an input image and we humans know what will be the output so we adjust the values in the network so that we obtain our output. This principle is called Learning a problem which is the backbone of Machine Learning. Here the system learns the values of convolutional filters and the vectors associated with the calculation in the neural network. A CNN does not require us to guess what filter we should use. Indeed the training of a CNN is the training of these filters which are weights for the network.

FINE TUNING

A neural network learns or approximates a function to best map inputs to outputs from examples in the training dataset. The learning rate hyperparameter controls the rate or speed at which the model learns. Specifically, it controls the amount of apportioned error that the weights of the model are updated with each time they are updated, such as at the end of each batch of training examples. Given a perfectly configured learning rate, the model will learn to best approximate the function given available resources (the number of layers and the number of nodes per layer) in a given number of training epochs (passes through the training data). Generally, a large learning rate allows the model to learn faster, at the cost of arriving on a sub-optimal final set of weights. A smaller learning rate may allow the model to learn a more optimal or even globally optimal set of weights but may take significantly longer to train. At extremes, a learning rate that is too large will result in weight updates that will be too large and the performance of the model (such as its loss on the training dataset) will oscillate over training epochs. Oscillating performance is said to be caused by weights that diverge (are divergent). A learning rate that is too small may never converge or may get stuck on a suboptimal solution.

It is important to find a good value for the learning rate for your model on your training dataset.The learning rate may, in fact, be the most important hyperparameter to configure for your model. The optimal learning rate can be found out by training different models with different learning rate and we can evaluate with a validation dataset to find the model which has generalised better.

A validation dataset is a sample of data held back from training your model that is used to give an estimate of model skill while tuning model’s hyperparameters.The validation dataset is different from the test dataset that is also held back from the training of the model, but is instead used to give an unbiased estimate of the skill of the final tuned model when comparing or selecting between final models.

CHAPTER 7

7. SYSTEM TESTING

7.1 TESTING

Testing is a series of different tests whose primary purpose is to fully exercise the computer based system. Although each test has a different purpose, all works should verify that all system elements have been properly integrated and performed an allocated function. Testing is the process of checking whether the developed system works according to the actual requirement and objectives of the system. The philosophy behind testing is to find the errors. A good test is one that has a high probability of finding an undiscovered error. A successful test is one that uncovers the undiscovered error. Test cases are devised with this purpose in mind. A test case is a set of data that the system will process as an input.

7.2 UNIT TESTING

All modules are tested individually as soon as they are completed and are checked for their correct functionality. Unit Testing of software applications is done during the development (coding) of an application. The objective of Unit Testing is to isolate a section of code and verify its correctness. In procedural programming, a unit may be an individual function or procedure. Unit Testing is usually performed by the developer.

7.3 INTEGRATION TESTING

The entire project is split into small programs. Each of these single programs gives a frame as an output. These programs are tested individually; at last all these programs are combined together by creating another program where all these constructors are used. It gives a lot of programs by not functioning in an integrated manner. Integration Testing focuses on checking data communication amongst these modules.

7.4 VALIDATION TESTING

It is the process of evaluating software during the development process or at the end of the development process to determine whether it satisfies the specified business requirements. Validation testing ensures that the product actually meets the clients' needs. It can also be defined as to demonstrate that the product fulfils its intended use when deployed in an appropriate environment. It answers the question 'are we building the right product?'.

7.5 BLACKBOX TESTING

Black box testing is defined as a testing technique in which functionality of the Application Under Test (AUT) is tested without looking at the internal code structure, implementation details and knowledge of internal paths of the software. This type of testing is based entirely on software requirements and specifications. In BlackBox Testing we just focus on inputs and output of the software system without bothering about internal knowledge of the software program.

7.6 WHITEBOX TESTING

White Box Testing is defined as the testing of a software solution's internal structure, design, and coding. In this type of testing, the code is visible to the tester. It focuses primarily on verifying the flow of inputs and outputs through the application, improving design and usability, strengthening security.

White box testing is also known as Clear Box testing, Open Box testing, Structural testing, Transparent Box testing, Code-Based testing, and Glass Box testing. It is usually performed by developers. The term "WhiteBox" was used because of the see-through box concept. The clear box or WhiteBox name symbolizes the ability to see through the software's outer shell (or "box") into its inner workings. Likewise, the "black box" in "Black Box Testing" symbolizes not being able to see the inner workings of the software so that only the end-user experience can be tested.

CHAPTER 8
8. CONCLUSION AND FUTURE ENHANCEMENT

8.1. CONCLUSION

The CNNs are able to learn the entire task of lane and road following without manual decomposition into road or lane marking detection, semantic abstraction, path planning, and control. A small amount of training data from less than a hundred hours of driving was sufficient to train the car to operate in diverse conditions, on highways, local and residential roads in sunny, cloudy, and rainy conditions. The CNN is able to learn meaningful road features from a very sparse training signal (steering alone).

The system learns for example to detect the outline of a road without the need of explicit labels during training. More work is needed to improve the robustness of the network, to find methods to verify the robustness, and to improve visualization of the network-internal processing steps. A lot of simulation should be done if the approach is selected for real time since the domain includes human health.

8.2. FUTURE ENHANCEMENT

Apart from the CNN which I have proposed, a separate CNN should be trained for traffic sign classification which will be able to give signals to the autonomous car when to stop, what speed to follow, etc. Also some warning signals like steep road ahead, curvy road ahead can be used to alert the driver as well as modify the steering angle which can give a heads up to our end to end approach. The speed signal detected by our classifier can be used to actuate the acceleration by considering the percentage of speed to be followed given by the user. Also an autonomy score can be created which can be calculated by the number of interventions during the entire duration of simulation which will be very useful to let us know the percentage of autonomy we have achieved.

CHAPTER 9
APPENDICES

9.1. APPENDIX - 1 : SOURCE CODE

DATA LOADING

```
from IPython.display import Image
import os
folder = '\\Dataset\\driving_dataset'
mapfile = '\\data.txt'
train = folder+mapfile
trial = os.getcwd() + folder + '\\0.jpg'
## Image read check
print("Raw image :", trial)
Image(trial)
```

DATA PREPROCESSING

```
def preprocess(img):
    greyscale = cv2.cvtColor(np.float32(img), cv2.COLOR_RGB2HSV)
    processed = cv2.resize(greyscale[:,:,1], (100,100))
    return processed
import numpy as np
import cv2
## Preprocess trial
dis = cv2.imread(trial)
out = preprocess(dis)
plt.imshow(out)
print("HSV image of raw image", trial)
## Dataset creation
```

```
train_path = os.path.join(os.getcwd(),'Dataset','data')
train_file = os.path.join(train_path, 'data.txt')
import scipy
## Creating a list X with image array values
## Creating a list y with respective angles
X = []
y = []
with open(train_file) as fp:
    for line in fp:
        path, angle = line.strip().split(' ')[:2]
        angle = angle.split(',')[0]
        image_path = os.path.join(train_path, path)
        X.append(image_path)
        y.append(float(angle) * scipy.pi/180)

## Loading HSV converted value of X in features list
features = []
for i in range(len(X)):
    if i%10000 == 0:
        print("Loaded {} images".format(i))
    img = plt.imread(X[i])
    features.append(preprocess(img))
## Storing the data in pickle format
import pickle
features = np.array(features).astype('float32')
labels = np.array(y).astype('float32')
with open("features_file", "wb") as f:
    pickle.dump(features, f, protocol=4)
with open("labels_file", "wb") as f:
    pickle.dump(labels, f, protocol=4)
```

DATA MODELLING

```
## Data loading from pickle format
import pickle
with open('features_file', 'rb') as f:
    features = np.array(pickle.load(f))
with open('labels_file', 'rb') as f:
    labels = np.array(pickle.load(f))
print("A total of {} images is used".format(features.shape[0]))
print("Features dimension :", features.shape)
print("Labels dimension :", labels.shape)
## Train/Test split
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
## For not following a pattern of outputs
features , labels = shuffle(features, labels)
train_x, test_x, train_y, test_y = train_test_split(features, labels, test_size=0.3)
## Transforming the image size/shape to our model
train_x = train_x.reshape(train_x.shape[0], 100, 100, 1)
test_x = test_x.reshape(test_x.shape[0], 100, 100, 1)
print("The shape of train_x", train_x.shape)
print("The shape of train_y", train_y.shape)
print("The shape of test_x", test_x.shape)
print("The shape of test_y", test_y.shape)
## Model architecture
from keras.layers import Dense, Activation, Flatten, Conv2D, Lambda
from keras.layers import MaxPooling2D, Dropout
from keras.models import Sequential
from keras.callbacks import ModelCheckpoint
import keras.backend as K
```

```
model = Sequential()
model.add(Lambda(lambda x: x / 127.5 - 1., input_shape=(100, 100, 1)))
model.add(Conv2D(32, (5, 5), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D((2, 2), padding='valid'))
model.add(Conv2D(32, (5, 5), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D((2, 2), padding='valid'))
model.add(Conv2D(64, (5, 5), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D((2, 2), padding='valid'))
model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D((2, 2), padding='valid'))
model.add(Conv2D(128, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(128, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D((2, 2), padding='valid'))
model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(1024))
model.add(Dense(256))
model.add(Dense(64))
model.add(Dense(1))
print(model.summary())
```

FINE TUNING THE MODEL

```
## Trying out the model for convergence for different learning rate
rates = [1e-8,1e-7,1e-6,1e-5,1e-4, 1e-3, 1e-2, 1e-1]
import keras
```

```
models = {}
for lr in rates:
    opt = keras.optimizers.Adam(learning_rate = lr)
    model.compile(optimizer= opt, loss="mse", metrics =['accuracy' ])
    models[lr] = model
print(models)
## Early stopping the model for convergence in the mid range of epochs
from keras import callbacks
earlystopping = callbacks.EarlyStopping(monitor ="val_loss",
                                        mode ="min", patience = 8,
                                        restore_best_weights = True)
## Training individual models and saving them in a dictionary
trained = {}
for model in models.keys():
    print (model)
    temp = models[model].fit(train_x, train_y, validation_data=(test_x, test_y), epochs= 25,
batch_size=64, callbacks=[earlystopping])
    trained[model] = temp
## Evaluating the model on test dataset
scores = []
for model in models.keys():
    print("{}:".format(model))
    scores.append(min(trained[model].history['val_loss']))
    print("Validation loss :",min(trained[model].history['val_loss']))
print("The most minimum loss is with 0.1 learning rate:", min(scores))
```

SAVING THE BEST MODEL

```
## Training the model with 0.1 learning rate and saving it
opt = keras.optimizers.Adam(learning_rate = 0.1)
model.compile(optimizer= opt, loss="mse", metrics =['accuracy' ])
```

```
progress = model.fit(train_x, train_y, validation_data=(test_x, test_y), epochs= 25,
batch_size=32, callbacks=[earlystopping])
model.save('Saved_steering_model.h5')
## Evaluating the model on validation dataset
print("Validation loss :",model.evaluate(test_x,test_y)[0])
```

PLOTTING THE TRAINING CURVE

```
# Plotting loss curve using matplotlib
plt.plot(progress.history['loss'])
plt.plot(progress.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='lower right')
plt.ylim(0,0.5)
plt.show()
```

SIMULATION USING VIDEO

```
import numpy as np
import cv2
from keras.models import load_model
model = load_model('Saved_steering_model.h5')
## Preprocessing the image for the model in format of the images trained
def preprocess(img):
    image_x = 100
    image_y = 100
    img = cv2.resize(img, (image_x, image_y))
    img = np.array(img, dtype=np.float32)
    img = np.reshape(img, (-1, image_x, image_y, 1))
    return img
## Predict function using trained model
def predict(model, image):
    processed = preprocess(image)
    steering_angle = float(model.predict(processed, batch_size=1))
    steering_angle = steering_angle * 60
    return steering_angle
## Displaying the steering output in a video
steer = cv2.imread('steering-wheels.jpg', 0)
rows, cols = steer.shape
smoothed_angle = 0
## Creating a simulation using the video
cap = cv2.VideoCapture('./run.mp4')
while (cap.isOpened()):
    ret, frame = cap.read()
    gray = cv2.resize((cv2.cvtColor(frame, cv2.COLOR_RGB2HSV))[:, :, 1], (100, 100))
    steering_angle = predict(model, gray)
    print(steering_angle)
    cv2.imshow('frame', cv2.resize(frame, (600, 400), interpolation=cv2.INTER_AREA))
    smoothed_angle += 0.2 * pow(abs((steering_angle - smoothed_angle)), 2.0 / 3.0) * (
        steering_angle - smoothed_angle) / abs(
        steering_angle - smoothed_angle)
    M = cv2.getRotationMatrix2D((cols / 2, rows / 2), -smoothed_angle, 1)
    dst = cv2.warpAffine(steer, M, (cols, rows))
    cv2.imshow("steering wheel", dst)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

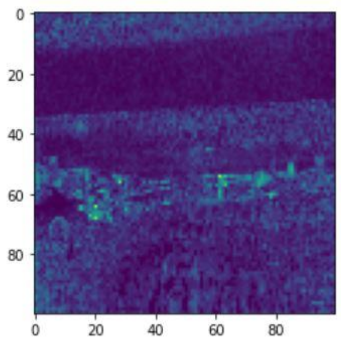
9.2. APPENDIX - 2 : SCREENSHOTS

DATA LOADING

```
Raw image : D:\Main Directory\My works\Project\Autonomous car\Steering\Dataset\driving_dataset\0.jpg
```

DATA PREPROCESSING

```
HSV image of raw image D:\Main Directory\My works\Project\Autonomous car\Steering\Dataset\driving_dataset\0.jpg
```

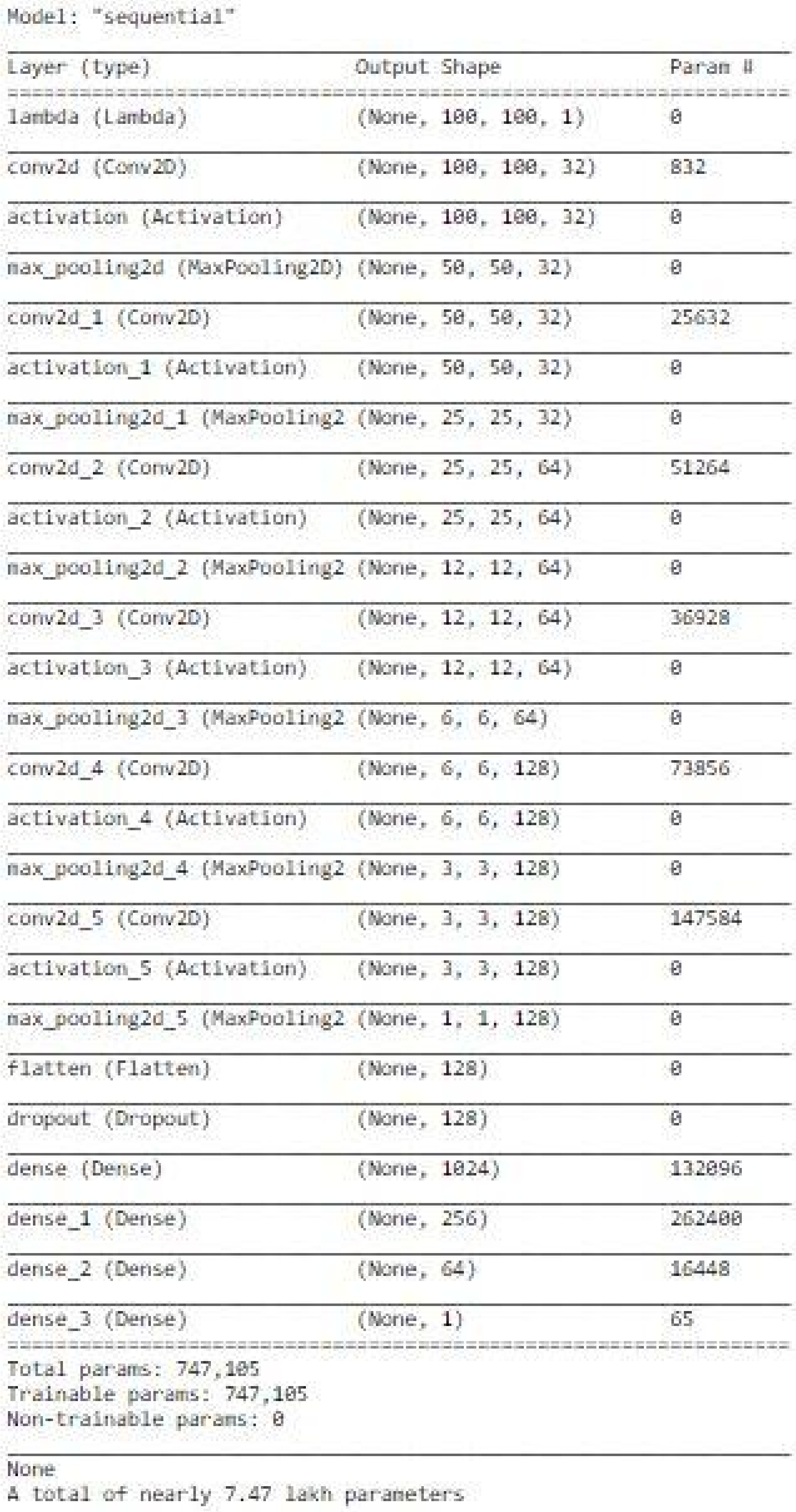


```
Loaded 10000 images
Loaded 20000 images
Loaded 30000 images
Loaded 40000 images
Loaded 50000 images
Loaded 60000 images
```

DATA MODELLING

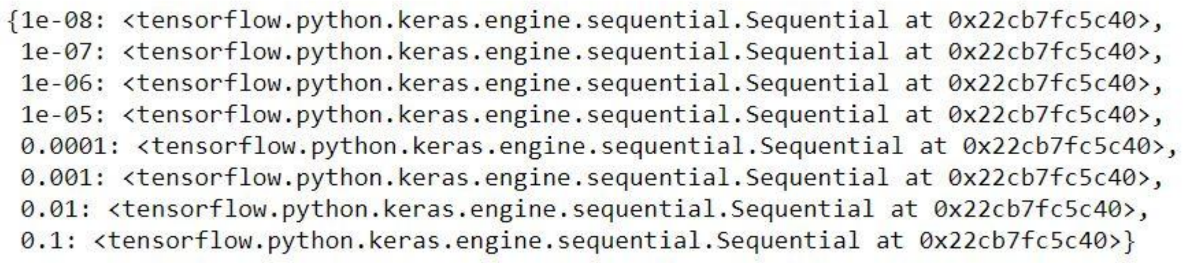
```
A total of 63825 images is used
Features dimension : (63825, 100, 100)
Labels dimension : (63825,)
```

```
The shape of train_x (44677, 100, 100, 1)
The shape of train_y (44677,)
The shape of test_x (19148, 100, 100, 1)
The shape of test_y (19148,)
```



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lambda (Lambda)              (None, 100, 100, 1)       0
_________________________________________________________________
conv2d (Conv2D)              (None, 100, 100, 32)      832
_________________________________________________________________
activation (Activation)      (None, 100, 100, 32)      0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 50, 50, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 50, 50, 32)        25632
_________________________________________________________________
activation_1 (Activation)    (None, 50, 50, 32)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 25, 25, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 25, 25, 64)        51264
_________________________________________________________________
activation_2 (Activation)    (None, 25, 25, 64)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 12, 12, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 12, 12, 64)        36928
_________________________________________________________________
activation_3 (Activation)    (None, 12, 12, 64)        0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 6, 6, 128)         73856
_________________________________________________________________
activation_4 (Activation)    (None, 6, 6, 128)         0
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 3, 3, 128)         0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 3, 3, 128)         147584
_________________________________________________________________
activation_5 (Activation)    (None, 3, 3, 128)         0
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 1, 1, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 128)               0
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 1024)              132096
_________________________________________________________________
dense_1 (Dense)              (None, 256)               262400
_________________________________________________________________
dense_2 (Dense)              (None, 64)                16448
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 65
=================================================================
Total params: 747,105
Trainable params: 747,105
Non-trainable params: 0
_________________________________________________________________
None
A total of nearly 7.47 lakh parameters
```

FINE TUNING THE MODEL



```
{1e-08: <tensorflow.python.keras.engine.sequential.Sequential at 0x22cb7fc5c40>,
 1e-07: <tensorflow.python.keras.engine.sequential.Sequential at 0x22cb7fc5c40>,
 1e-06: <tensorflow.python.keras.engine.sequential.Sequential at 0x22cb7fc5c40>,
 1e-05: <tensorflow.python.keras.engine.sequential.Sequential at 0x22cb7fc5c40>,
 0.0001: <tensorflow.python.keras.engine.sequential.Sequential at 0x22cb7fc5c40>,
 0.001: <tensorflow.python.keras.engine.sequential.Sequential at 0x22cb7fc5c40>,
 0.01: <tensorflow.python.keras.engine.sequential.Sequential at 0x22cb7fc5c40>,
 0.1: <tensorflow.python.keras.engine.sequential.Sequential at 0x22cb7fc5c40>}
```

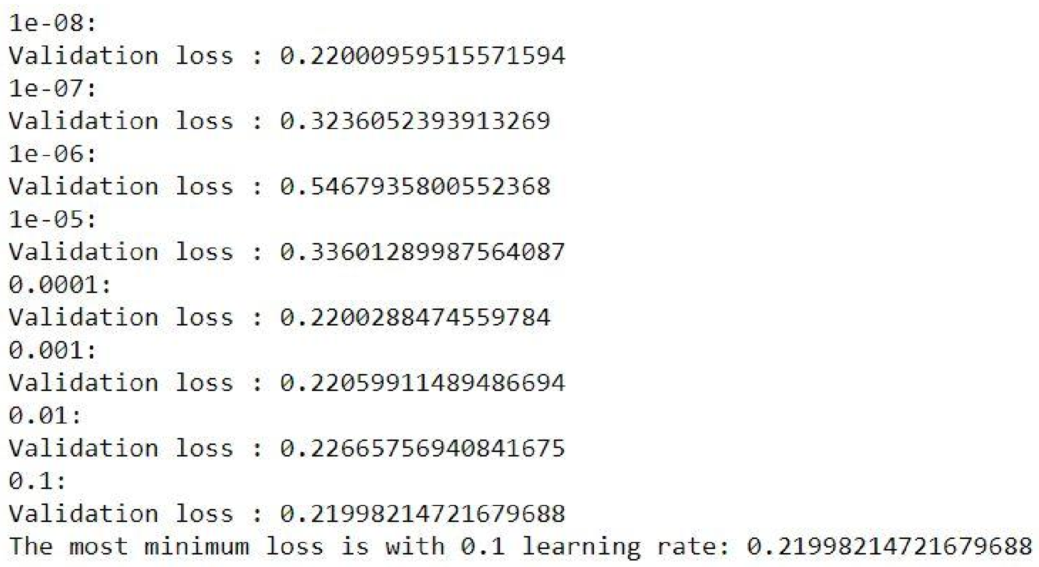
```
1e-08
Epoch 1/25
699/699 [==============================] - 413s 590ms/step - loss: 2122486043584383.2500 - accuracy: 0.0209 - val_loss: 0.2200 -
val_accuracy: 0.0397
Epoch 2/25
699/699 [==============================] - 403s 576ms/step - loss: 0.2142 - accuracy: 0.0387 - val_loss: 0.2234 - val_accuracy: 0.0397
Epoch 3/25
699/699 [==============================] - 403s 576ms/step - loss: 0.2229 - accuracy: 0.0390 - val_loss: 0.2369 - val_accuracy: 0.0397
Epoch 4/25
699/699 [==============================] - 402s 576ms/step - loss: 0.2082 - accuracy: 0.0397 - val_loss: 0.2203 - val_accuracy: 0.0397
Epoch 5/25
699/699 [==============================] - 408s 584ms/step - loss: 0.2153 - accuracy: 0.0393 - val_loss: 0.2201 - val_accuracy: 0.0397
Epoch 6/25
699/699 [==============================] - 408s 584ms/step - loss: 0.2086 - accuracy: 0.0382 - val_loss: 0.2390 - val_accuracy: 0.0397
Epoch 7/25
699/699 [==============================] - 408s 584ms/step - loss: 0.2270 - accuracy: 0.0396 - val_loss: 0.2496 - val_accuracy: 0.0397
Epoch 8/25
699/699 [==============================] - 420s 601ms/step - loss: 0.2383 - accuracy: 0.0391 - val_loss: 2.2511 - val_accuracy: 0.0397
Epoch 9/25
699/699 [==============================] - 422s 604ms/step - loss: 0.3507 - accuracy: 0.0364 - val_loss: 0.2564 - val_accuracy: 0.0397
----------------------------------------------------------------------------------------------------------------------
1e-07
Epoch 1/25
699/699 [==============================] - 403s 577ms/step - loss: 0.3575 - accuracy: 0.0347 - val_loss: 0.6788 - val_accuracy:
0.0000e+00
Epoch 2/25
699/699 [==============================] - 403s 577ms/step - loss: 129003.7812 - accuracy: 0.0290 - val_loss: 408419.2188 -
val_accuracy: 0.0000e+00
Epoch 3/25
699/699 [==============================] - 410s 587ms/step - loss: 527032.9375 - accuracy: 0.0192 - val_loss: 212.2400 - val_accuracy:
0.0397
Epoch 4/25
699/699 [==============================] - 410s 586ms/step - loss: 680998.8125 - accuracy: 0.0195 - val_loss: 0.3236 - val_accuracy:
0.0397
Epoch 5/25
699/699 [==============================] - 410s 587ms/step - loss: 973754.4375 - accuracy: 0.0200 - val_loss: 36.1337 - val_accuracy:
0.0000e+00
Epoch 6/25
699/699 [==============================] - 402s 576ms/step - loss: 1326989.8750 - accuracy: 0.0204 - val_loss: 1.4870 - val_accuracy:
0.0000e+00
Epoch 7/25
699/699 [==============================] - 407s 582ms/step - loss: 1898115.0000 - accuracy: 0.0200 - val_loss: 861.6982 - val_accuracy:
0.0397
Epoch 8/25
```

```
   699/699 [==============================] - 409s 586ms/step - loss: 2281385.5000 - accuracy: 0.0191 - val_loss: 16.7983 - val_accuracy:
0.0397
   Epoch 9/25
   699/699 [==============================] - 410s 586ms/step - loss: 2305743.7500 - accuracy: 0.0203 - val_loss: 15.7114 - val_accuracy:
0.0397
   Epoch 10/25
   699/699 [==============================] - 408s 584ms/step - loss: 3063886.0000 - accuracy: 0.0201 - val_loss: 4668.7715 - val_accuracy:
0.0000e+00
   Epoch 11/25
   699/699 [==============================] - 409s 585ms/step - loss: 2836647.5000 - accuracy: 0.0196 - val_loss: 3320718.7500 -
val_accuracy: 0.0000e+00
   Epoch 12/25
   699/699 [==============================] - 406s 581ms/step - loss: 2172742.7500 - accuracy: 0.0205 - val_loss: 2.1392 - val_accuracy:
0.0000e+00
   ----------------------------------------------------------------------------------------------------------------
   1e-06
   Epoch 1/25
   699/699 [==============================] - 403s 577ms/step - loss: 2667947.7500 - accuracy: 0.0195 - val_loss: 4.4885 - val_accuracy:
0.0000e+00
   Epoch 2/25
   699/699 [==============================] - 410s 587ms/step - loss: 2820200.0000 - accuracy: 0.0199 - val_loss: 0.5468 - val_accuracy:
0.0397
   Epoch 3/25
   699/699 [==============================] - 405s 579ms/step - loss: 2866011.0000 - accuracy: 0.0196 - val_loss: 26.8732 - val_accuracy:
0.0000e+00
   Epoch 4/25
   699/699 [==============================] - 410s 587ms/step - loss: 2639450.5000 - accuracy: 0.0197 - val_loss: 545.6756 - val_accuracy:
0.0397
   Epoch 5/25
   699/699 [==============================] - 410s 586ms/step - loss: 2971923.7500 - accuracy: 0.0208 - val_loss: 312.4680 - val_accuracy:
0.0397
   Epoch 6/25
   699/699 [==============================] - 410s 586ms/step - loss: 2617916.0000 - accuracy: 0.0187 - val_loss: 376564.4688 -
val_accuracy: 0.0397
   Epoch 7/25
   699/699 [==============================] - 408s 584ms/step - loss: 2749072.2500 - accuracy: 0.0210 - val_loss: 3862.7913 - val_accuracy:
0.0397
   Epoch 8/25
   699/699 [==============================] - 410s 586ms/step - loss: 2040028.0000 - accuracy: 0.0195 - val_loss: 48387840.0000 -
val_accuracy: 0.0397
   Epoch 9/25
   699/699 [==============================] - 409s 585ms/step - loss: 2486937.7500 - accuracy: 0.0202 - val_loss: 1281235.7500 -
val_accuracy: 0.0000e+00
   Epoch 10/25
   699/699 [==============================] - 409s 585ms/step - loss: 2633970.2500 - accuracy: 0.0201 - val_loss: 4.1456 - val_accuracy:
0.0397
   ----------------------------------------------------------------------------------------------------------------
   1e-05
   Epoch 1/25
   699/699 [==============================] - 409s 584ms/step - loss: 3248562.5000 - accuracy: 0.0225 - val_loss: 0.3360 - val_accuracy:
0.0397
   Epoch 2/25
```

```
  699/699 [==============================] - 409s 586ms/step - loss: 2973484.5000 - accuracy: 0.0212 - val_loss: 14067.3174 -
val_accuracy: 0.0000e+00
  Epoch 3/25
  699/699 [==============================] - 410s 586ms/step - loss: 2325099.5000 - accuracy: 0.0208 - val_loss: 2504887.7500 -
val_accuracy: 0.0397
  Epoch 4/25
  699/699 [==============================] - 409s 586ms/step - loss: 2279789.5000 - accuracy: 0.0198 - val_loss: 103599.1172 -
val_accuracy: 0.0397
  Epoch 5/25
  699/699 [==============================] - 409s 585ms/step - loss: 1666137.0000 - accuracy: 0.0204 - val_loss: 2.5614 - val_accuracy:
0.0397
  Epoch 6/25
  699/699 [==============================] - 410s 586ms/step - loss: 2428257.2500 - accuracy: 0.0202 - val_loss: 339.4226 - val_accuracy:
0.0397
  Epoch 7/25
  699/699 [==============================] - 408s 583ms/step - loss: 1361650.7500 - accuracy: 0.0207 - val_loss: 0.3491 - val_accuracy:
0.0397
  Epoch 8/25
  699/699 [==============================] - 409s 585ms/step - loss: 1747373.0000 - accuracy: 0.0214 - val_loss: 1007128.6875 -
val_accuracy: 0.0000e+00
  Epoch 9/25
  699/699 [==============================] - 410s 586ms/step - loss: 1189839.1250 - accuracy: 0.0210 - val_loss: 96577.9375 -
val_accuracy: 0.0397
  ------------------------------------------------------------------------------------------------------------------
  0.0001
  Epoch 1/25
  699/699 [==============================] - 409s 585ms/step - loss: 25239916.0000 - accuracy: 0.0325 - val_loss: 0.2322 - val_accuracy:
0.0397
  Epoch 2/25
  699/699 [==============================] - 409s 585ms/step - loss: 0.2192 - accuracy: 0.0387 - val_loss: 0.2344 - val_accuracy: 0.0397
  Epoch 3/25
  699/699 [==============================] - 408s 584ms/step - loss: 0.2261 - accuracy: 0.0387 - val_loss: 0.2200 - val_accuracy: 0.0397
  Epoch 4/25
  699/699 [==============================] - 408s 584ms/step - loss: 0.2377 - accuracy: 0.0387 - val_loss: 0.6279 - val_accuracy:
0.0000e+00
  Epoch 5/25
  699/699 [==============================] - 409s 586ms/step - loss: 0.2438 - accuracy: 0.0386 - val_loss: 0.2395 - val_accuracy: 0.0397
  Epoch 6/25
  699/699 [==============================] - 408s 584ms/step - loss: 0.2513 - accuracy: 0.0384 - val_loss: 0.2248 - val_accuracy: 0.0397
  Epoch 7/25
  699/699 [==============================] - 410s 586ms/step - loss: 0.3235 - accuracy: 0.0355 - val_loss: 5.6813 - val_accuracy: 0.0397
  Epoch 8/25
  699/699 [==============================] - 408s 584ms/step - loss: 72028.9531 - accuracy: 0.0254 - val_loss: 0.6076 - val_accuracy:
0.0000e+00
  Epoch 9/25
  699/699 [==============================] - 409s 585ms/step - loss: 50893.4844 - accuracy: 0.0256 - val_loss: 2.7313 - val_accuracy:
0.0000e+00
  Epoch 10/25
  699/699 [==============================] - 409s 585ms/step - loss: 39896.2188 - accuracy: 0.0262 - val_loss: 0.3854 - val_accuracy:
0.0397
  Epoch 11/25
```

```
   699/699 [==============================] - 408s 584ms/step - loss: 27151.4707 - accuracy: 0.0254 - val_loss: 1.1658 - val_accuracy:
0.0000e+00
   ----------------------------------------------------------------------------------------------------------------
   0.001
   Epoch 1/25
   699/699 [==============================] - 409s 585ms/step - loss: 549290688.0000 - accuracy: 0.0326 - val_loss: 0.2206 - val_accuracy:
0.0397
   Epoch 2/25
   699/699 [==============================] - 409s 585ms/step - loss: 0.2195 - accuracy: 0.0387 - val_loss: 0.2306 - val_accuracy: 0.0397
   Epoch 3/25
   699/699 [==============================] - 407s 582ms/step - loss: 0.2201 - accuracy: 0.0387 - val_loss: 0.2537 - val_accuracy: 0.0397
   Epoch 4/25
   699/699 [==============================] - 409s 585ms/step - loss: 0.2244 - accuracy: 0.0387 - val_loss: 0.3339 - val_accuracy: 0.0397
   Epoch 5/25
   699/699 [==============================] - 408s 584ms/step - loss: 0.2267 - accuracy: 0.0387 - val_loss: 0.2357 - val_accuracy: 0.0397
   Epoch 6/25
   699/699 [==============================] - 408s 584ms/step - loss: 0.2391 - accuracy: 0.0387 - val_loss: 0.2440 - val_accuracy: 0.0397
   Epoch 7/25
   699/699 [==============================] - 407s 583ms/step - loss: 0.2559 - accuracy: 0.0383 - val_loss: 0.4652 - val_accuracy:
0.0000e+00
   Epoch 8/25
   699/699 [==============================] - 407s 583ms/step - loss: 0.3306 - accuracy: 0.0357 - val_loss: 1.2836 - val_accuracy:
0.0000e+00
   Epoch 9/25
   699/699 [==============================] - 408s 583ms/step - loss: 1.0052 - accuracy: 0.0299 - val_loss: 1.0797 - val_accuracy: 0.0397
   ----------------------------------------------------------------------------------------------------------------
   0.01
   Epoch 1/25
   699/699 [==============================] - 408s 583ms/step - loss: 89226.6719 - accuracy: 0.0267 - val_loss: 0.9912 - val_accuracy:
0.0397
   Epoch 2/25
   699/699 [==============================] - 407s 583ms/step - loss: 52415.7305 - accuracy: 0.0268 - val_loss: 2.7752 - val_accuracy:
0.0000e+00
   Epoch 3/25
   699/699 [==============================] - 407s 583ms/step - loss: 27924.4297 - accuracy: 0.0273 - val_loss: 0.2789 - val_accuracy:
0.0397
   Epoch 4/25
   699/699 [==============================] - 407s 582ms/step - loss: 15096.8604 - accuracy: 0.0266 - val_loss: 2.2632 - val_accuracy:
0.0397
   Epoch 5/25
   699/699 [==============================] - 407s 582ms/step - loss: 15259.9580 - accuracy: 0.0308 - val_loss: 1.5291 - val_accuracy:
0.0397
   Epoch 6/25
   699/699 [==============================] - 407s 582ms/step - loss: 4217.3984 - accuracy: 0.0254 - val_loss: 0.2267 - val_accuracy: 0.0397
   Epoch 7/25
   699/699 [==============================] - 407s 583ms/step - loss: 1702.6603 - accuracy: 0.0273 - val_loss: 0.2878 - val_accuracy: 0.0397
   Epoch 8/25
   699/699 [==============================] - 406s 581ms/step - loss: 898.6379 - accuracy: 0.0264 - val_loss: 9.1311 - val_accuracy: 0.0397
   Epoch 9/25
   699/699 [==============================] - 408s 583ms/step - loss: 1089.2991 - accuracy: 0.0285 - val_loss: 0.6065 - val_accuracy: 0.0397
   Epoch 10/25
   699/699 [==============================] - 407s 582ms/step - loss: 551.9700 - accuracy: 0.0265 - val_loss: 8.7700 - val_accuracy: 0.0397
```

```
Epoch 11/25
699/699 [==============================] - 407s 582ms/step - loss: 382.4340 - accuracy: 0.0251 - val_loss: 0.2309 - val_accuracy: 0.0397
Epoch 12/25
699/699 [==============================] - 405s 580ms/step - loss: 161.9237 - accuracy: 0.0229 - val_loss: 3.1807 - val_accuracy: 0.0397
Epoch 13/25
699/699 [==============================] - 407s 582ms/step - loss: 97.3965 - accuracy: 0.0237 - val_loss: 19.1928 - val_accuracy: 0.0397
Epoch 14/25
699/699 [==============================] - 407s 583ms/step - loss: 71.3044 - accuracy: 0.0268 - val_loss: 0.6074 - val_accuracy: 0.0000e+00
----------------------------------------------------------------------------------------------------------------------
0.1
Epoch 1/25
699/699 [==============================] - 408s 584ms/step - loss: 10117654528.0000 - accuracy: 0.0312 - val_loss: 0.2200 - val_accuracy: 0.0397
Epoch 2/25
699/699 [==============================] - 408s 583ms/step - loss: 0.2140 - accuracy: 0.0387 - val_loss: 0.2205 - val_accuracy: 0.0397
Epoch 3/25
699/699 [==============================] - 407s 582ms/step - loss: 0.2143 - accuracy: 0.0387 - val_loss: 0.2200 - val_accuracy: 0.0397
Epoch 4/25
699/699 [==============================] - 408s 584ms/step - loss: 0.2150 - accuracy: 0.0387 - val_loss: 0.2210 - val_accuracy: 0.0397
Epoch 5/25
699/699 [==============================] - 408s 583ms/step - loss: 0.2153 - accuracy: 0.0387 - val_loss: 0.2207 - val_accuracy: 0.0397
Epoch 6/25
699/699 [==============================] - 409s 585ms/step - loss: 0.2169 - accuracy: 0.0387 - val_loss: 0.2209 - val_accuracy: 0.0397
Epoch 7/25
699/699 [==============================] - 408s 584ms/step - loss: 0.2175 - accuracy: 0.0387 - val_loss: 0.2530 - val_accuracy: 0.0397
Epoch 8/25
699/699 [==============================] - 409s 585ms/step - loss: 0.2248 - accuracy: 0.0387 - val_loss: 0.2305 - val_accuracy: 0.0397
Epoch 9/25
699/699 [==============================] - 409s 585ms/step - loss: 0.2229 - accuracy: 0.0387 - val_loss: 0.2533 - val_accuracy: 0.0397
```



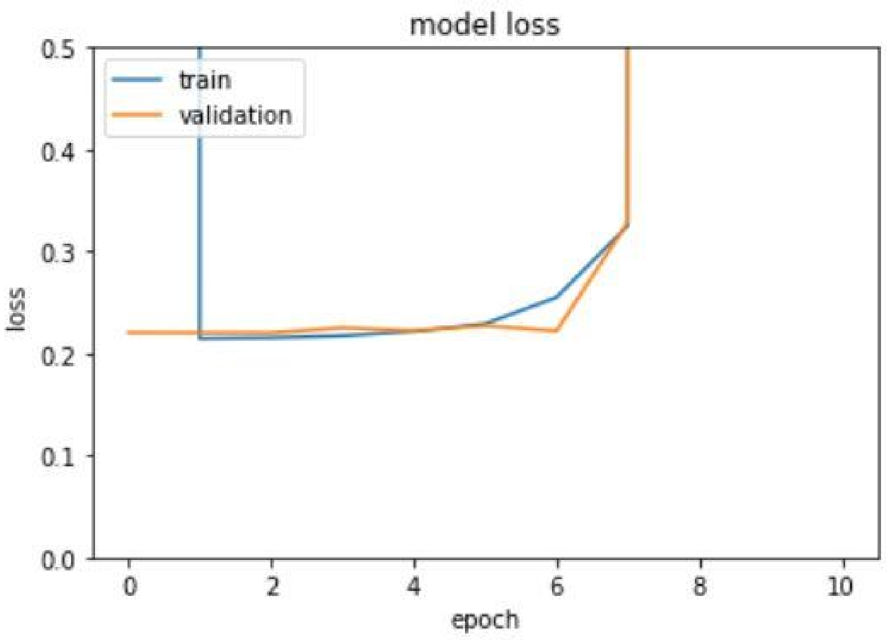
```
1e-08:
Validation loss : 0.22000959515571594
1e-07:
Validation loss : 0.3236052393913269
1e-06:
Validation loss : 0.5467935800552368
1e-05:
Validation loss : 0.33601289987564087
0.0001:
Validation loss : 0.2200288474559784
0.001:
Validation loss : 0.22059911489486694
0.01:
Validation loss : 0.22665756940841675
0.1:
Validation loss : 0.21998214721679688
The most minimum loss is with 0.1 learning rate: 0.21998214721679688
```

SAVING THE BEST MODEL

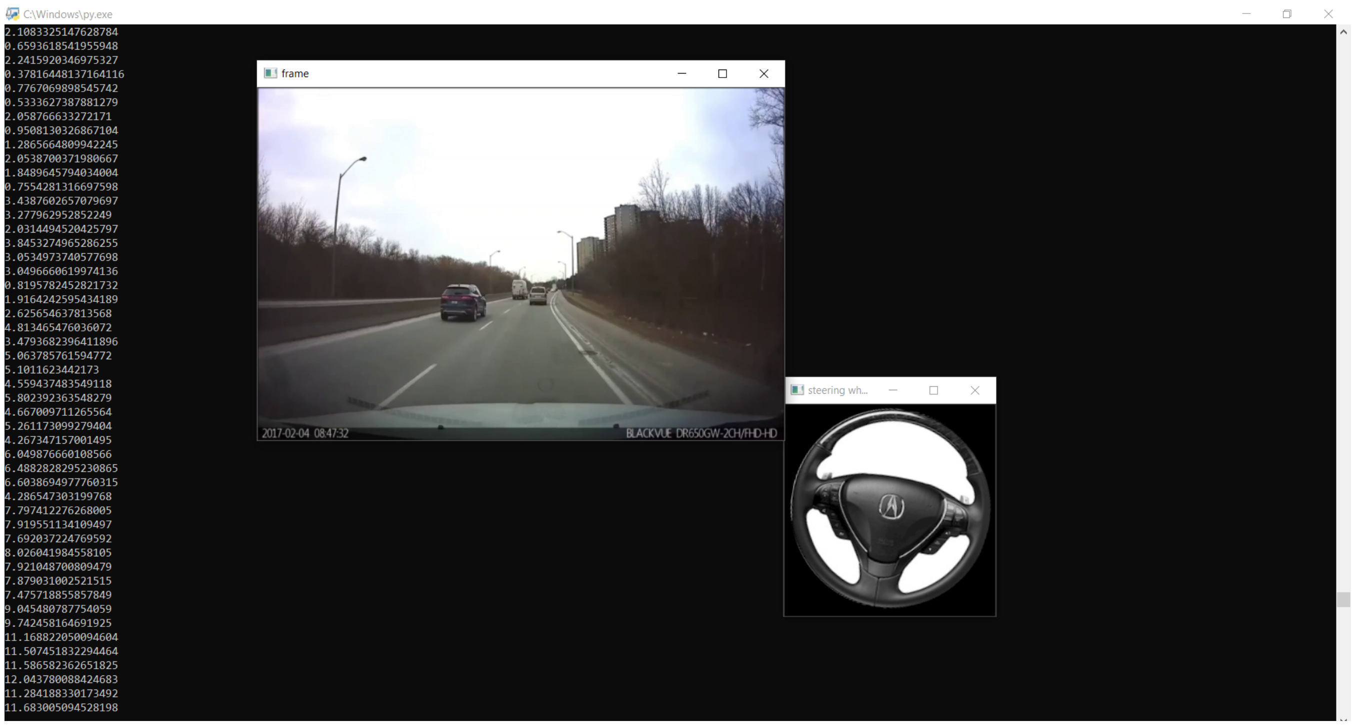
```
Epoch 1/25
```

```
  1397/1397 [==============================] - 443s 313ms/step - loss: 495554997073553.7500 - accuracy:
0.0279 - val_loss: 0.2203 - val_accuracy: 0.0397
  Epoch 2/25
  1397/1397 [==============================] - 436s 312ms/step - loss: 0.2118 - accuracy: 0.0383 - val_loss:
0.2202 - val_accuracy: 0.0397
  Epoch 3/25
  1397/1397 [==============================] - 425s 304ms/step - loss: 0.2186 - accuracy: 0.0374 - val_loss:
0.2202 - val_accuracy: 0.0397
  Epoch 4/25
  1397/1397 [==============================] - 420s 300ms/step - loss: 0.2270 - accuracy: 0.0381 - val_loss:
0.2251 - val_accuracy: 0.0397
  Epoch 5/25
  1397/1397 [==============================] - 419s 300ms/step - loss: 0.2235 - accuracy: 0.0378 - val_loss:
0.2221 - val_accuracy: 0.0397
  Epoch 6/25
  1397/1397 [==============================] - 418s 299ms/step - loss: 0.2275 - accuracy: 0.0396 - val_loss:
0.2272 - val_accuracy: 0.0397
  Epoch 7/25
  1397/1397 [==============================] - 420s 301ms/step - loss: 0.2395 - accuracy: 0.0378 - val_loss:
0.2221 - val_accuracy: 0.0397
  Epoch 8/25
  1397/1397 [==============================] - 423s 303ms/step - loss: 0.2987 - accuracy: 0.0364 - val_loss:
0.3288 - val_accuracy: 0.0397
  Epoch 9/25
  1397/1397 [==============================] - 423s 303ms/step - loss: 29219.5725 - accuracy: 0.0292 -
val_loss: 540.3596 - val_accuracy: 0.0397
  Epoch 10/25
  1397/1397 [==============================] - 424s 303ms/step - loss: 639426.4395 - accuracy: 0.0211 -
val_loss: 239806.0312 - val_accuracy: 0.0000e+00
  Epoch 11/25
  1397/1397 [==============================] - 429s 307ms/step - loss: 574850.2629 - accuracy: 0.0182 -
val_loss: 12.0982 - val_accuracy: 0.0000e+00
  599/599 [==============================] - 33s 55ms/step - loss: 0.2202 - accuracy: 0.0397
  Validation loss : 0.22015583515167236
```

PLOTTING THE TRAINING CURVE



SIMULATION USING VIDEO



```
2.1083325147628784
0.6593618541955948
2.2415920346975327
0.37816448137164116
0.7767069898545742
0.5333627387881279
2.058766633272171
0.9508130326867104
1.2865664809942245
2.0538700371980667
1.8489645794034004
0.7554281316697598
3.4387602657079697
3.277962952852249
2.0314494520425797
3.8453274965286255
3.0534973740577698
3.0496660619974136
0.8195782452821732
1.9164242595434189
2.625654637813568
4.813465476036072
3.4793682396411896
5.063785761594772
5.1011623442173
4.559437483549118
5.802392363548279
4.667009711265564
5.261173099279404
4.267347157001495
6.049876660108566
6.4882828295230865
6.6038694977760315
4.286547303199768
7.797412276268005
7.919551134109497
7.692037224769592
8.026041984558105
7.921048700809479
7.879031002521515
7.475718855857849
9.045480787754059
9.742458164691925
11.168822050094604
11.507451832294464
11.586582362651825
12.043780088424683
11.284188330173492
11.683005094528198
```

CHAPTER 10
REFERENCES

- A Comprehensive Analysis Of Deep Regression : St ́ephane Lathuili`Ere, Pablo Mesejo, Xavier Alameda-pineda.
- An End-to-end Deep Neural Network For Autonomous Driving Designed For Embedded Automotive Platforms : Jelena Kocić,* Nenad Jovičić, And Vujo Drndarević
- The Yan: [Self Driving Car Using Deep Learning] : Ms. Sujeetha, Chitrak Bari, Gareja Prdip, Siddhant Purohit
- End To End Learning For Self-driving Cars : Nvidia Corporation
- DeepPicar: A Low-cost Deep Neural Network-based Autonomous Car : Michael G. Bechtel, Elise McEllhiney, Minje Kim, Heechul Yun

WEB REFERENCES

- https://emerj.com/ai-adoption-timelines/self-driving-car-timeline-themselves-top-11-automakers/
- https://towardsdatascience.com/what-is-deep-learning-and-how-does-it-work-2ce44bb692ac
- https://emerj.com/ai-adoption-timelines/self-driving-car-timeline-themselves-top-11-automakers/
- https://ieeexplore.ieee.org/document/8308186/authors#authors
- https://arxiv.org/pdf/1604.07316.pdf
- https://www.tutorialspoint.com/python-pickling
- https://keras.io/
- https://towardsdatascience.com/convolutional-neural-networks-why-are-they-so-good-for-image-related-learning-2b202a25d757
- https://www.upgrad.com/blog/basic-cnn-architecture/
- https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148