Yarmouk Private University

Faculty of Informatics and Communication Engineering

Department of Software Engineering

Self-Driving Car Using Deep Learning

By:

MHD ADEL MOUMOU

Under the guidance and supervision of:

Dr. Nouar AlDahoul

A thesis submitted in fulfillment of the requirements for the degree of

BACHELOR OF ENGINEERING

at

YARMOUK PRIVATE UNIVERSITY

Semester of 2019-1

ABSTRACT

Self-driving car topic has been raised in the era of big data. Many approaches have been proposed for self-driving car topic, some of these solutions are based only on sensors LIDAR, Radar, Camera…etc. but the solutions that are based on only sensors are expensive in terms of the cost of the sensors. Deep learning has been proposed for the perception module of the self-driving car system object detection, ء…etc. This project proposes a self-driving car approach implemented based on only camera sensors, end-to-end deep learning, multi-task deep learning, a high-level planner such as A* and imitation learning. Our system is able to drive most of the time (85% in the simulator) without human intervention.

TABLE OF CONTENT

Table OF FIGURES

LIST OF TABLES:

Abbreviations

Abbreviation	Definition
AISTATS	Artificial Intelligence and Statistics
ANN	Artificial Neural Networks
CNN	Convolutional Neural Network
DDNs	Deep Neural Networks
FC	Fully Connected
FOV	Field-of-view
GPS	Global Positioning System
GPU	Graphics Processing Unit
ICLR	International Conference on Learning Representations
IL	Imitation Learning
IMU	Inertial Measurement Unit
IOT	Internet of Things
JAUS	Joint Architecture for Unmanned Systems
LSTM	Long Short-Term Memory
MDF	Mission Data Files

MLP	Multi-layer perceptron
NIPS	Neural Information Processing Systems
PWM	Pulse-width modulation
RC	Radio Control
RGB	Red – Green – Blue
RL	Reinforcement Learning
RNDF	Route Network Definition File
RNN	Recurrent Neural Network
ROS	Robot Operating System

1. Chapter One (Introduction)

1.1 Overview

Automated driving has been an active research area for the last couple of decades. Road statistics show that 95% of all car crashes are related to human error. In 2011, 30.000 people were killed on the roads in the European Union, and many more were injured. [1]

Autonomous driving has the potential of saving thousands of lives by removing human error from driving. It can also help to make driving more efficient, which results in lower fuel costs and a decreased impact on the environment.

The majority of car manufacturers are currently developing vehicles that can achieve different levels of automated driving. These levels are defined as:

Level 0. All major systems are controlled by humans

Level 1. Certain systems, such as cruise control or automatic braking, may be controlled by the car, one at a time

Level 2. The car offers at least two simultaneous automated functions, like acceleration and steering, but requires humans for safe operation

Level 3. The car can manage all safety-critical functions under certain conditions, but the driver is expected to take over when alerted

Level 4. The car is fully-autonomous in some driving scenarios, though not all

Level 5. The car is completely capable of self-driving in every situation.

1.2 Problem Formulation

An automated vehicle has to be aware of its surroundings to a certain degree to be able to drive safely. It has to detect vehicles, pedestrians, cyclists and in some cases (wild) animals to assess whether or not they pose a collision risk.

To achieve proper detection, the vehicle has to be equipped with an adequate sensor set so that driving, in general, can be separated into three different driving

environments: highway, rural and urban, a division commonly seen in the literature.

All these environments have different characteristics, i.e. pedestrians and cyclists can be encountered frequently in the urban environment, but much less frequently in rural environments, and are not allowed on a highway at all. Furthermore, different construction rules and guidelines apply to each environment.[8]

This report will focus on urban areas depending on the **4th Level** of Self driving cars. These environments have well-defined rules and guidelines which allow effective analysis of the different driving scenarios.

1.3 Self-Driving Cars Benefits

1. Road Safety

Road accidents result in 1.25 million deaths and 20-50 million[1] injuries worldwide. If nothing changes, road traffic injuries may become the fifth leading cause of death by 2030.

Self-driving cars may be the solution to road accidents. Since human error is the cause of around 90% of traffic accidents, delegating most or all of the vehicle operating responsibilities to the ADS can increase road safety. While fully automated vehicles haven't been commercially adopted yet, many cars today are equipped with sensors and advanced systems that alert drivers to dangers.

2. Less Traffic

Research by INRIX [2]ranks the U.S as the fifth most congested country, with Thailand in the lead for the title of "World's Most Congested Country". Drivers in the U.S spend an average of 41 hours a year in traffic jams.

[1] https://www.asirt.org/safe-travel/road-safety-facts/
[2] Global Traffic Scorecard - https://inrix.com/scorecard/

Self-driving car technology, such as connected cars, may offer a solution to clogged roads. Connected cars can communicate with each other can help optimize routes for each individual vehicle, creating a network of information that helps distribute traffic flow.

3. Reduced Emissions

The US transportation sector produces **30%** of all U.S global warming emissions[3]. Despite the dire environmental consequences of emissions, more vehicles are deployed daily, partly due to the demand for more delivery trucks for last-mile delivery. Traffic jams, excessive speed, and braking and re-accelerating also contribute to pollution.

Self-driving cars help reduce the pollution emitted by vehicles. Autonomous capabilities such as consistent driving speeds and keeping a measured distance between vehicles can reduce unnecessary braking and re-acceleration. Electronic models of self-driving cars with an electric or hybrid engine further reduce pollution by eliminating or lessening the use of fuel.

1.4 Project Objective

The project aims to control the car to move from point A to point B within a specific environment according to a navigator and provide the car with a system to avoid obstacles and take into account light signals using deep learning.

[3] https://www.ucsusa.org/resources/car-emissions-global-warming

2. Chapter Two
(Literature Review)

To achieve the result of our project, lots of research papers have been read which are related to ours and These are some of them.

2.1 Agile Autonomous Driving Using End-To-End Deep Imitation Learning [2]

This research paper presents an end-to-end imitation learning system for agile, off-road autonomous driving using only low-cost on-board sensors. By imitating a model predictive controller equipped with advanced sensors, they train a deep neural network control policy to map raw, high-dimensional observations to continuous steering and throttle commands.

They aim to design a reflexive driving policy that uses only low-cost, on-board sensors (e.g. monocular camera, wheel speed sensors).

They adopt deep neural networks (DNNs) to parametrize the control policy and learn the desired parameters from the robot's interaction with its environment. While the use of DNNs as policy representations for RL is not uncommon, in contrast to most previous work that showcases RL in simulated environments [3], the agent is a high-speed physical system that incurs real-world cost: collecting data is a cumbersome process, and a single poor decision can physically impair the robot and result in weeks lost while replacing parts and repairing the platform. Therefore, the direct application of model-free RL techniques is not only sampled inefficient but costly and dangerous in our experiments.

Table 2-1 Comparison of Agile Autonomous method to prior work on LL form autonomous driving

METHODS	TASKS	OBSERVATIONS	ACTION	ALGORITHM	EXPERT	EXPERIMENT
[4]	On-road low-speed	Single image	Steering	Batch	Human	Real & simulated
[5]	On-road low-speed	Single image &laser	Steering	Batch	Human	Real & simulated
[6]	On-road low-speed	Single image	Steering	Batch	Human	Simulated
[7]	Off-road low-speed	Left & right images	Steering	Batch	Human	Real
[8]	On-road unknown speed	Single image	Steering &break	Online	Pre-specified policy	Simulated
THIS METHOD	On-road high-speed	Single image &wheel speeds	Steering & throttle	Batch & online	Model predictive controller	Real & simulated

2.1.1 The Autonomous Driving System for This Paper

Building on the previous analyses, they design a system that can learn to perform fast off-road autonomous driving with only on-board measurements. The overall system architecture for learning end-to-end DNN driving policies is illustrated in (Figure 2-1) It consists of three high-level controllers (an expert, a learner, and a safety control module) and a low-level controller, which receives steering and throttle commands from the high-level controllers and translates them to pulse-width modulation (PWM) signals to drive the steering and throttle actuators of a vehicle. the paper assumes the expert is algorithmic and has access to expensive sensors (GPS and IMU) for accurate global state estimates and resourceful computational power.

The expert is built on multiple hand-engineered components, including a state estimator, a dynamics model of the vehicle, a cost function of the task, and a trajectory optimization algorithm for planning. By contrast, the learner is a DNN policy that has access to only a monocular camera and wheel speed sensors and is required to output steering and throttle command directly. In this setting, the sensors that the learner uses can be significantly cheaper than those of the expert; specifically on our experimental platform, the Auto Rally car, the IMU and the GPS sensors required by the expert together cost more than $6,000, while the sensors used by the learner's DNN policy cost less than $500.

The safety control module has the highest priority among all three controllers and is used to prevent the vehicle from high-speed crashing. The software system was developed based on the Robot Operating System (ROS) in Ubuntu. Also, a Gazebo based simulation environment was built using the same ROS interface but without the safety control module; the simulator was used to evaluate the performance of the software before real track tests.

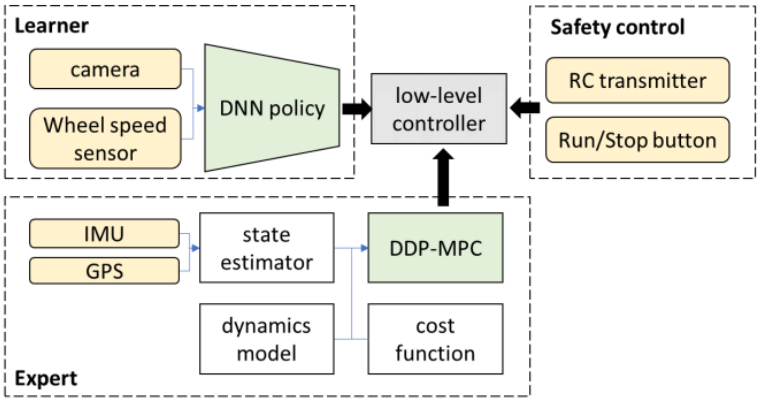

Figure 2-1 Autonomous driving system architecture

2.1.2 Learning a DNN Control Policy

The learner's control policy π is parametrized by a DNN containing ~10 million parameters. As illustrated in (Figure 2-2), the DNN policy, consists of two sub-networks: a convolutional neural network (CNN) with 6 convolutional layers, 3 max-pooling layers, and 2 fully connected layers, that takes 160×80 RGB monocular images as inputs,5 and a feedforward network with a fully-connected hidden layer that takes wheel speeds as inputs. The convolutional and max-pooling layers are used to extract lower-dimensional features from images. The DNN policy uses 3 × 3 filters for all convolutional layers and rectified linear unit (ReLU[4]) activation for all layers except the last one.

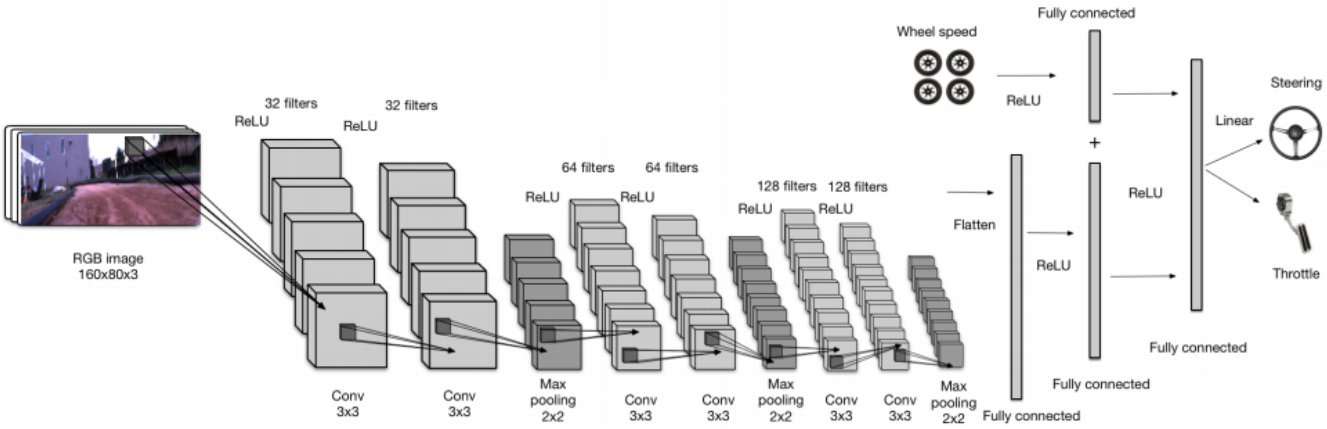

Figure 2-2 The DNN Control Policy

[4] The most commonly used activation function in neural networks.

Max-pooling layers with 2×2 filters are integrated to reduce the spatial size of the representation (and therefore reduce the number of parameters and

computation loads). The two sub-networks are concatenated and then followed by another fully-connected hidden layer. The structure of this DNN was selected empirically based on experimental studies of several different architectures. The optimization problem is solved using ADAM[5], which is a stochastic gradient descent algorithm with an adaptive learning rate. The neural network policy does not use the state, but rather the synchronized raw observation as input. the paper does not perform any data selection or augmentation techniques in any of the experiments. The only pre-processing was scaling and cropping of raw images.

In (Figure 2-3) there are some examples of vehicle trajectories, where online IL avoids the crashing case encountered by batch IL. (b) and (c) depict the test runs after training on 9,000 samples. iterations increased. Experimental data was collected on an outdoor track and consisted of changing lighting conditions and environmental dynamics. In the experiments, the rollouts about to crash were terminated remotely by overwriting the autonomous control commands with the run-stop button or the RC transmitter in the safety control module; these rollouts were excluded from the data collection.

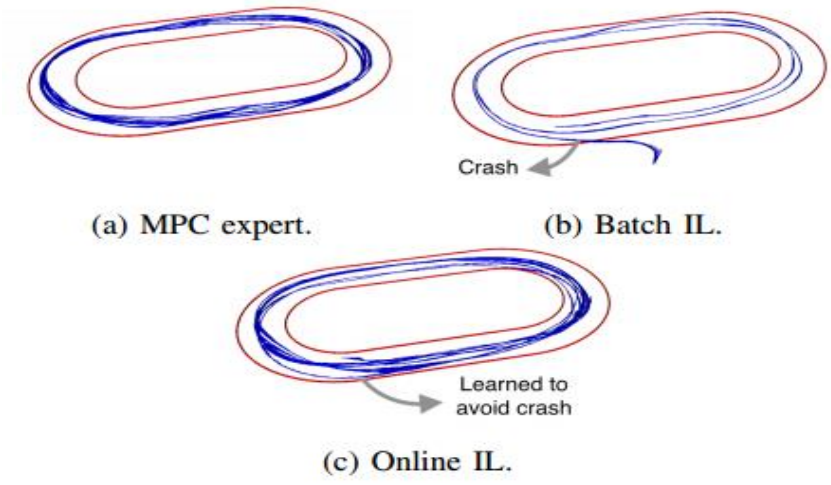

Figure 2-3 Examples of vehicle trajectories

[5] Optimization algorithm in Deep Learning.

Total loss denotes the imitation loss in (6), which is the average of the steering and the throttle losses. Completion is defined as the ratio of the traveled time steps to the targeted time steps (3,000). All result in (Table 2-2) represents the average performance over three independent evaluation trials

Table 2-2 Average performance over three independent evaluation trials.

POLICY	AVG. SPEED	TOP SPEED	TRAINING DATA	COMPLETION RATION	TOTAL LOSS	STEERING/THROTTLE LOSS
EXPERT	6.05 m/s	8.14 m/s	N/A	100%	0	0
BATCH	4.97 m/s	5.51 m/s	3000	100%	0.108	0.092/0.124
BATCH	6.02m/s	8.18 m/s	6000	51%	0108	0.162/0.055
BATCH	5.79 m/s	7.78 m/s	9000	53%	0.123	0.193/0.071
BATCH	5.95 m/s	8.01 m/s	12000	69%	0.105	0.125/0.083
ONLINE (1 ITER)	6.02 m/s	7.88 m/s	6000	100%	0.090	0.112/0.067
ONLINE (2 ITER)	5.89 m/s	8.02 m/s	9000	100%	0.075	0.095/0.055
ONLINE (3 ITER)	6.07 m/s	8.06 m/s	12000	100%	0.064	0.073/0.055

(Figure 2-4) shows the performance of online and batch IL in the distance (meters) traveled without crashing. The policy trained with a batch of 3,000 samples was used to initialize online IL.

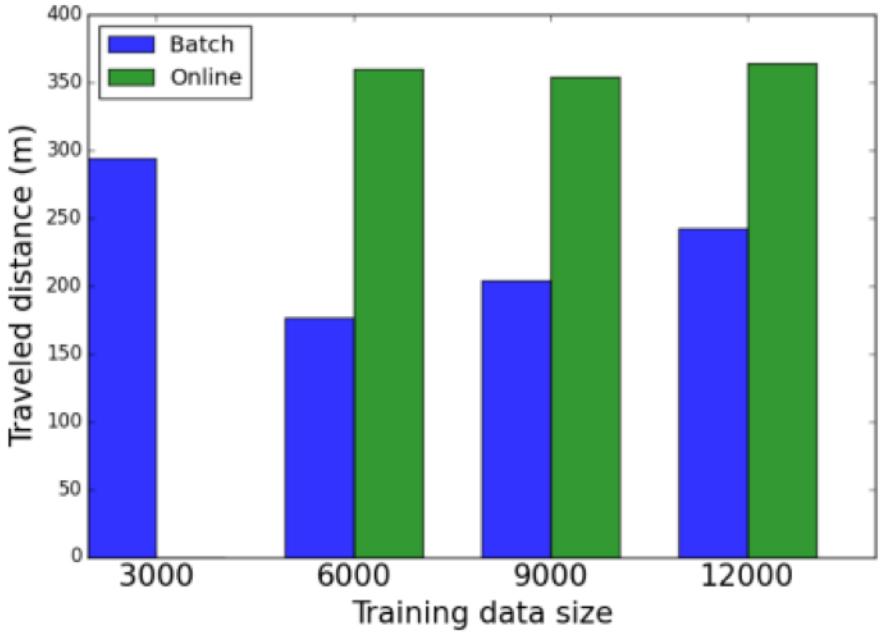

Figure 2-4 Performance of online and batch learning

(Figure 2-5) shows the performance comparison between our DNN policy and its CNN sub-network in terms of batch IL loss, where the horizontal axis is the size of data used to train the neural network policies

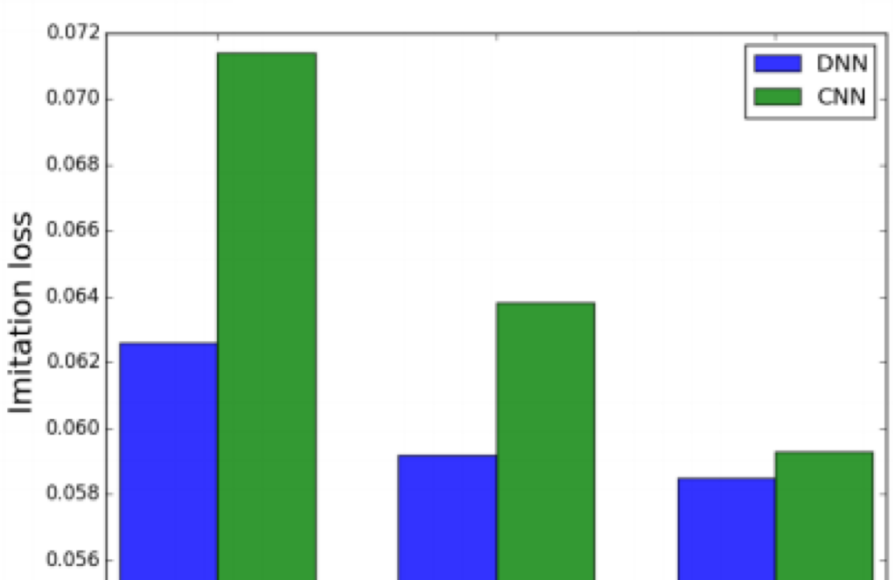

Figure 2-5 a of neural network policies.

2.2 An End-To-End Deep Neural Network for Autonomous Driving Designed for Embedded Automotive Platforms [9]

In this paper, the proposed solution is an end-to-end deep neural network for autonomous driving. The main objective of their work was to achieve autonomous driving with a light deep neural network suitable for deployment on embedded automotive platforms such as Raspberry Pi[6]. 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 of this paper. The network architecture, computational complexity, and performance evaluation during autonomous driving using our network are compared with two other convolutional neural networks that the paper implemented 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 the AlexNet model. While the 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.

2.2.1 System Architecture

The autonomous driving system can be generally divided into four blocks: sensors, perception subsystem, planning subsystem, and control of the vehicle, (Figure 2-6) shows a vehicle sensing the world using many different sensors mounted on the vehicle. The information from

[6] A series of small single-board computers developed to promote the teaching of basic computer science in schools and developing countries.

the sensors is processed in a perception block, whose components combine sensor data into meaningful information. The planning subsystem uses the output from the perception block for behavior planning and both short- and long-range path planning. The control module ensures that the vehicle follows the path provided by the planning subsystem and sends control commands to the vehicle.

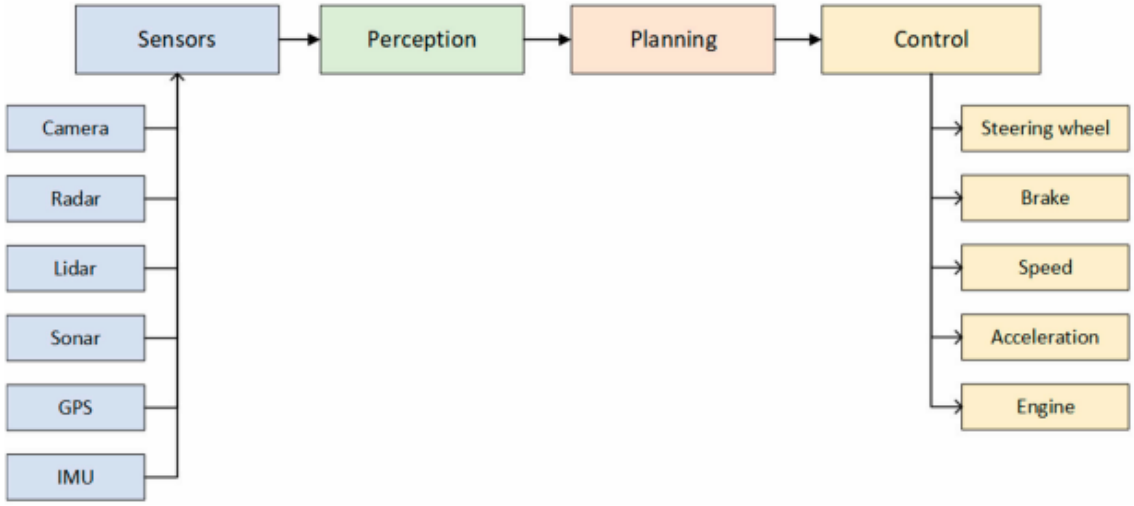

Figure 2-6 Autonomous vehicle system block diagram

An end-to-end deep neural network designed for autonomous driving uses camera images as an input, which is a raw signal (i.e., pixel), and steering angle predictions as an output to control the vehicle.

End-to-end learning presents the training of neural networks from the beginning to the end without human interaction or involvement in the training process. The purpose of end-to-end learning is that the system automatically learns internal representations of the necessary processing steps, such as detection of useful road characteristics, based only on the input signal.

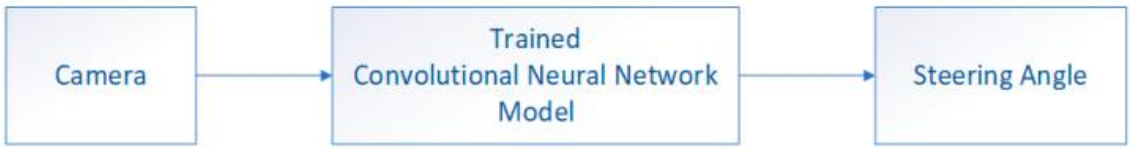

Figure 2-7 Block diagram of an end-to-end autonomous driving system

This paper presents a novel deep neural network developed for end-to-end learning for autonomous driving, called J Net, which is designed for embedded automotive platforms. In addition to this, for objective evaluation of J-Net, the discussions of the results of the re-implementations of PilotNet and AlexNet. AlexNet is originally developed for object classification, but after our modifications, the new AlexNet-like model is suitable for autonomous driving. Firstly, the novel deep neural network architecture J-Net is developed, the model is trained and used for inference during autonomous driving. Secondly, this paper compares architectures of three network architectures, J-Net, PitotNet, and AlexNet, and discusses computational complexity.

Next, the implemented models have been trained with the same dataset that is collected. The data collection and inference are done in the self-driving car simulator designed in Unity environment by Udacity. Finally, the trained models have been used for real-time autonomous driving in a simulator environment. The results of autonomous driving using each of these three deep neural network models have been presented and compared. Results of the autonomous driving are given as video recordings of the autonomous driving in a representative track in a simulator environment, in addition to the qualitative and quantitative performance evaluation of autonomous driving parameters during inference.

This paper presents a novel network for a simplified solution of end-to-end learning for autonomous driving. The input in this autonomous driving system is only the camera image, the raw pixel. Output is steering angle prediction. The aim is to achieve the computationally light model that can be deployed to an embedded automotive platform for real-time inference.

Therefore, the goal is to develop the right deep neural network architecture that can achieve acceptable accuracy, the successful autonomous driving in a representative track, but which operates in real-time within the power and energy limitations of its target embedded automotive platform. To achieve the approach that they took was not to reduce the parameters of some of the well-known neural networks that can be used for autonomous driving, but to start from scratch,

developing a network architecture layer by layer until they found the satisfying solution. The general workflow to find an appropriate model size is to start with relatively few layers and parameters and to increase the size of the layers or add new layers until returns diminish concerning validation loss. In summary, the contributions of the approach proposed in this paper are:

1. modified the convolutional neural network for image classification, AlexNet, and used the new AlexNet-like model for end-to-end learning for autonomous driving.

2. proposed a novel deep neural network J-Net that, despite having a light architecture in comparison with AlexNet and PilotNet, can successfully perform autonomous driving. This recommends the J-Net model for deployment on low-performing embedded automotive platforms.

3. demonstrated the suitability of a new proposed network J Net for autonomous driving for embedded automotive platforms by doing the performance evaluation of autonomous driving in simulated environment, where the J-Net shows the best performance in terms of latency and frame rate, among three implemented solutions.

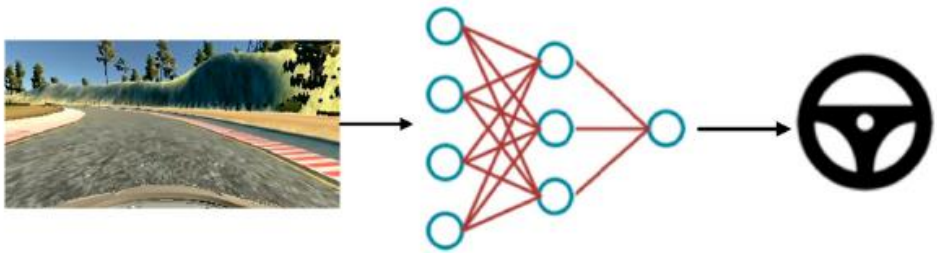

Figure 2-8 Real-time autonomous driving architecture

2.2.2 Dataset Collection

Firstly, to collect the data that would be used for training the end-to-end DNN model, a human driver was driving the vehicle and simultaneously recording the images and steering measurements. If the simulator environment for autonomous driving was used, the vehicle

was driven in manual (training) mode by a human driver using a keyboard, mouse, or joystick, and the dataset was automatically collected. Data acquired during manual driving mode were camera images and the steering angle values per each frame. The images were used as the feature set, and the steering measurements as the label set. The speed of the vehicle was fixed due to simplicity.

Data collected using this approach were used for training the neural network that will learn to drive based only on the input data, without any further human interaction. This technique is also known as behavior cloning. Secondly, the deep neural network for autonomous driving was trained on this dataset to predict the steering angle. Finally, the trained model was used for inference, a real-time autonomous driving in the same simulator environment. The metrics of successful driving on the representative track was that the vehicle remains on the track at all times during autonomous driving. A block diagram of the autonomous driving framework that used is presented in (Figure 2-9).

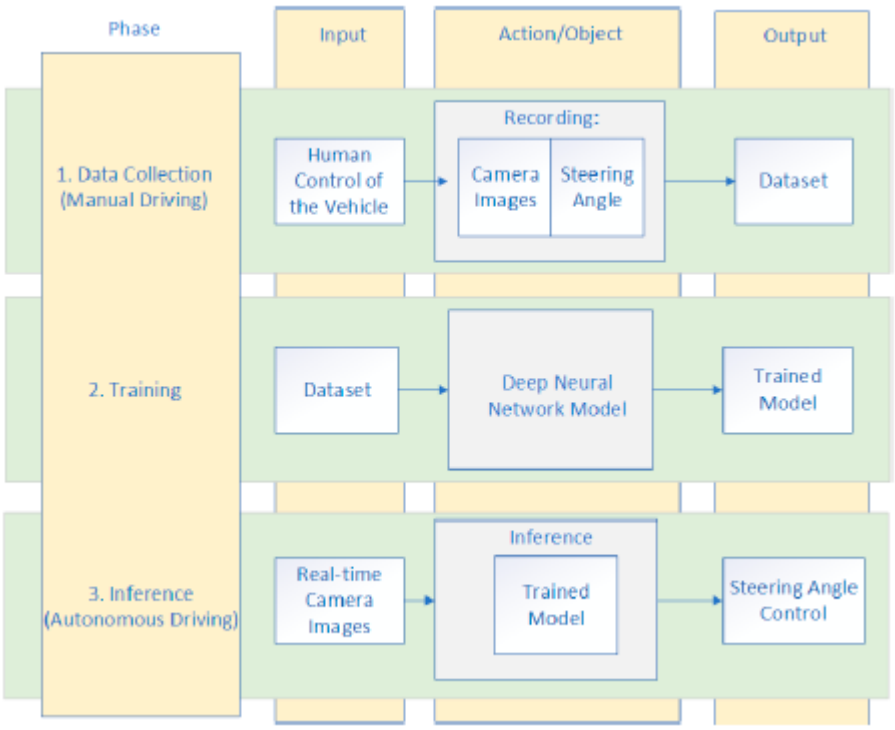

Figure 2-9 Block diagram of the autonomous driving framework

Data collection was done while the vehicle was driving in manual mode on the representative track. Image data were acquired from the three cameras mounted on the vehicle in the simulator environment. At the end of the manual ride, the images were stored together with the table containing information about image titles and steering angle values per each recorded frame. An example of images recorded by all three cameras in one frame is presented in (Figure 2-10). Three cameras were used for training purposes. During the data collection, in each frame, the images from three cameras were captured with the same steering measurement value. The slight difference in the field of view per each central left and right camera leads to a better generalization of the model.

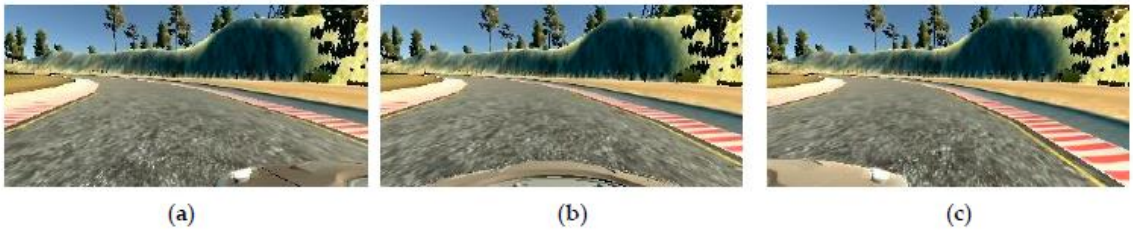

Figure 2-10 Example of data collection taken simultaneously using the three cameras mounted on the vehicle: (a) left, (b) center and (c) right cameras

The total number of acquired samples was 34,288 images with a resolution of (320, 160, 3). Each recorded image was 320 pixels high, 160 pixels wide, and three channels deep—a color RGB image. The average memory size of one recorded image was about 13.5 kB. Each image was paired with the corresponding steering angle value that was normalized in range between -1 and 1. After applying data augmentation, the total number of samples in our dataset was 68,576. The data was split into training and validation categories, where 80% of data was chosen for training, 54,861 samples, and 20% of the data for the validation, 13,715 samples.

Table 2-3 Dataset classifying

	Training	Validation	Total
Number of Samples	54,861	13,715	68,576
Percentage of Total Dataset	80%	20%	100%

Testing was done in real-time during inference. Real-time acquired images from the central camera were continuously sent to the trained machine learning model that was used for the control of the vehicle steering angle. Evaluation of the successful model was done by observing if the vehicle was able to drive autonomously during the entire predefined track. If the vehicle was off the road, it was considered as unsuccessful autonomous driving

2.2.3 Proposed Approach

The leading idea during the design process was to achieve end-to-end autonomous driving using the light model (computationally least expensive model), while simultaneously achieving the best possible performance, in this case, the autonomous driving on the representative path. The computationally least expensive model is usually the model with the least number of parameters that effects their memory footprint and computations. Therefore, the type and size of layers, kernel sizes, and a number of feature maps have an influence on computational performance. The performance of autonomous driving was examined in a self-driving car simulator. The final architecture of the J-Net model was the result of experimenting with building blocks of deep neural networks different number of layers, kernel sizes for convolutional layers, number of feature maps, placement of pooling layers, and, in the end, experimenting with the size and number of the fully-connected layers. The block diagram of this experimental shallow CNN is presented in (Figure 2-11).

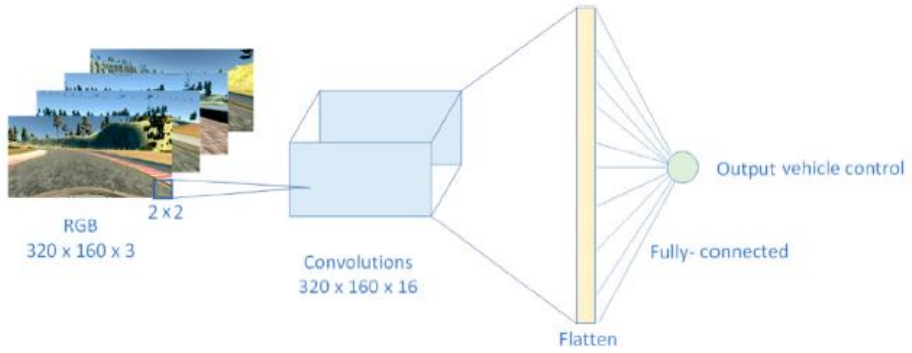

Figure 2-11 Architecture of experimental shallow deep neural network with just one convolutional layer with 32 feature maps, one flattened layer and, at the end, the fully connected layer

The first step in the design of the novel solution was to use an extremely shallow CNN; the performed 2D convolution operation on the raw data of the input image, where the channel input image had the dimensions 320 * 160, the used two-dimensional kernel size was 2 * 2. The kernel was used to extract the patches of the image in convolutional operation. The output of the convolution operation was S(i,j), the two-dimensional feature map tensor. The weights (w) were shared across patches for a given layer in a CNN to detect the Sensors representation regardless of where in the image it was located

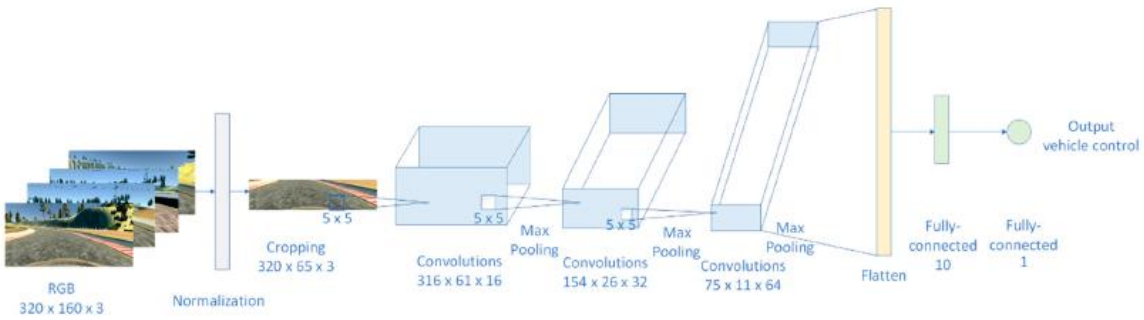

Figure 2-12 Architecture of proposed end-to-end deep neural network, j-Net, used for autonomous driving. The network has three convolutional layers with 16,32, and 64 feature maps, one flattened layer, and two fully-connected (dense) layer. Max pooling is place

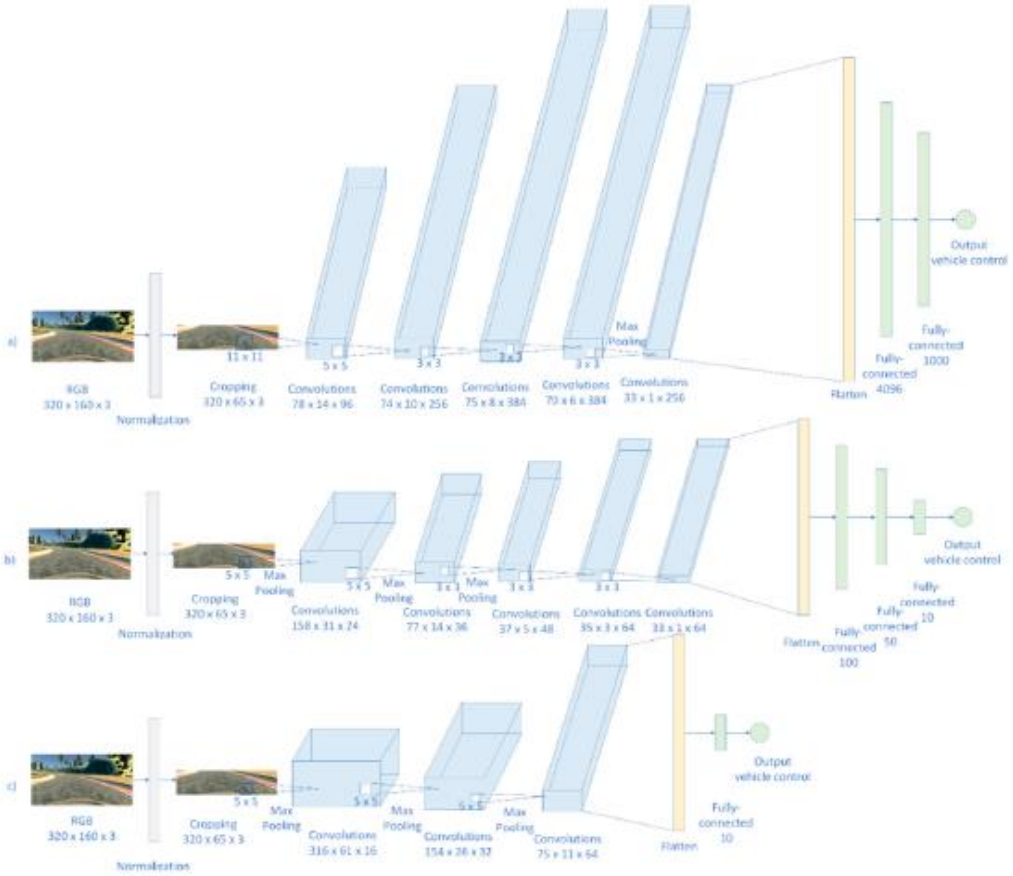

Figure 2-13 Comparison of deep neural network architectures that we implemented and used for end-to-end autonomous driving. (a) AlexNet, (b) PilotNet, (c) J-Net.

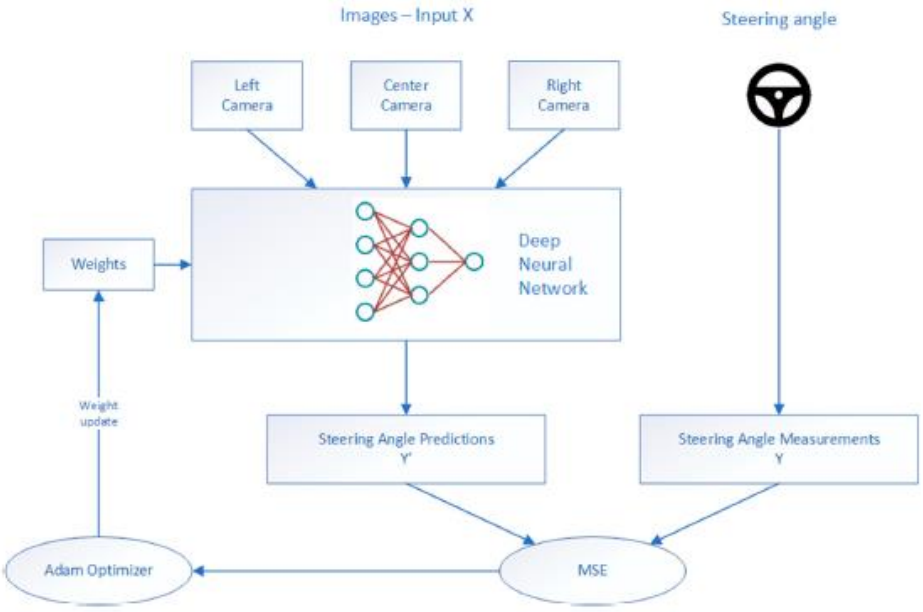

Figure 2-14 Training flowchart. Data collection is done in manual driving mode by acquiring images from three cameras mounted on the vehicle. Central, left and right in parallel with recording the steering angle paired with each image. During the training process, the correction factor has been added to the left and right image

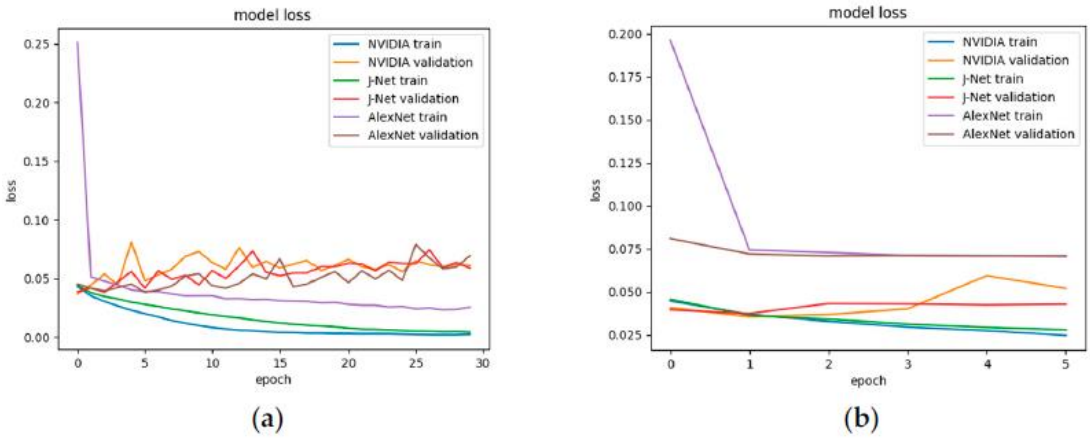

Figure 2-15 The model loss for training and validation for AlexNet, PilotNet, and J-Net. (a) for training in 30 epochs, (b) for training in 6 epochs

Table 2-4 layers and number of trainable parameters for J-Net, PilotNet and AlexNet

Layers and Parameters		AlexNet	PilotNet	J-Net
Convolutional		5	5	3
Flatten		1	1	1
Dense (Fully-Connected)		3	4	2
Total number of trainable parameters		42,452,305	348,219	150,965
Operations	Multiplication	42,45m	347,82k	150,84k
	Addition	42,45m	347,82k	150,84k

Total number of trainable parameters is calculated based on input image size. After lambda normalization and cropping, the size of input to the first convolutional layer is $65 \times 320 \times 3$.

Table 2-5 Size of the trained model

	AlexNet	PilotNet	J-Net
Number of epochs used for training	30	4	6
Size of the trained model	509.5MB	4.2MB	1.8MB

Table 2-6 Qualitative performance evaluation of autonomous driving using AlexNet, PilotNet and J-Net

Autonomous Driving	AlexNet	PilotNet	J-Net
Autonomous Driving on representative track	Successful	Successful	Successful
Handling the curves	Good	Medium	Medium/Good
Keeping to the trajectory canter	Good	Medium	Medium
Driving on different surface textures	Good	Good	Good

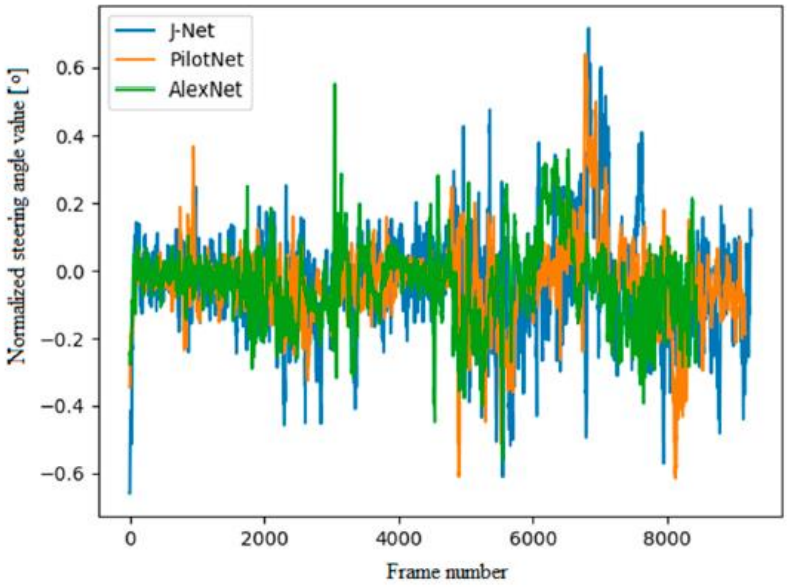

Figure 2-16 Steering angle prediction used for autonomous driving, presented as normalized absolute values of. Steering angle in degrees.

2.3 Deep Learning Based Motion Planning for Autonomous Vehicle Using Spatiotemporal LSTM Network [10]

This paper presents a spatiotemporal LSTM network for autonomous vehicle motion planning which is able to generate the steering motion reaction by exploiting the spatiotemporal information.

The convolutional LSTM has the ability to learn the time-serial features about the traffic environment, Meanwhile, the 3D-CNN [7]can extract the spatial information.

Experimental results verified the validity and stability of the proposed deep learning method. The paper proposes a motion planning model based on deep learning (named as spatiotemporal LSTM network), which is able to generate a real-time reflection based on spatiotemporal information extraction. To be specific, the model based on the spatiotemporal LSTM network has three main structures. Firstly, the Convolutional Long-short Term Memory (Conv-LSTM) is used to extract hidden features through sequential image data. Then, the 3D Convolutional Neural Network(3D-CNN) is applied to extract the spatiotemporal information from the multi-frame feature information.

Finally, the fully connected neural networks are used to construct a control model for the autonomous vehicle steering angle. The experiments demonstrated that the proposed method can generate a robust and accurate visual motion planning results for the autonomous vehicle.

[7] In a 3D CNN, the kernels move through three dimensions of data (height, length, and depth) and produce 3D activation maps.

2.3.1 Proposed Approach

This section will explain the core ideology of Conv-LSTM and describe the architecture of the proposed Spatiotemporal LSTM Network which combines Conv-LSTM and 3D Convolutional Neural Network for motion planning.

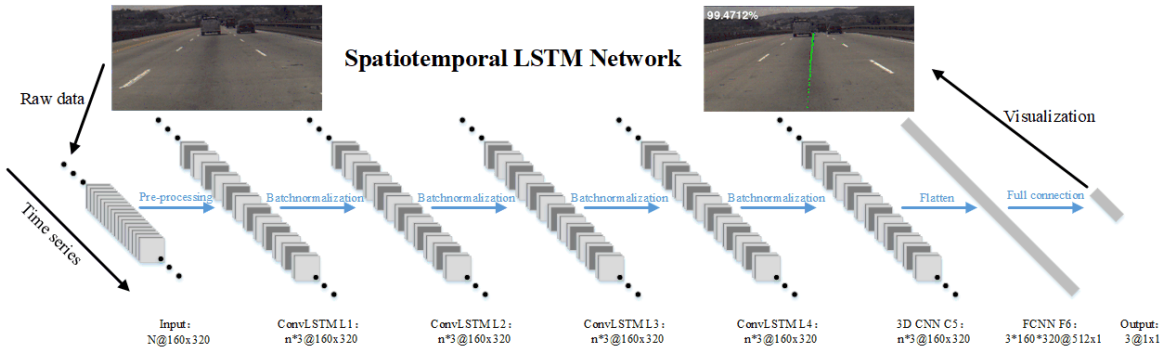

Figure 2-17 The architecture of the Spatiotemporal LSTM Network

2.3.2 Proposed Network Architecture

The overall network architecture of the proposed approach for autonomous vehicle motion planning using deep learning is shown in (Figure 2-17). For the whole network processing, the input is multi-frame picture segment and the output layer is the steering, after passing the spatiotemporal LSTM network. Since the original collected information is single-frame picture information collected by the camera, it needs preprocessing, they perform preprocessing before entering the system.

In the pre-processing process, single-frame picture information is combined in time series to form a multi-frame picture segment based on time series. After that, the multi-frame image segment based on time series is input into the system, which is followed by 4 layers of Conv-LSTM layers and a 3D-CNN layer. Finally, the output of the motion planning result works out through 2 layers of FCNN.

2.3.3 Data Preprocessing

In the process of constructing a deep learning network, it is very important to analyze and preprocess the original data. The raw data used in this paper is the picture frame output by the camera in the car. To make these time-series based picture frames better adapted to the proposed network to achieve better training performance, it needs to be preprocessed, the preprocessing is shown as (Figure 2-18). Also, grouping single frame images based on time can also allow input data to contain richer temporal and spatial data.

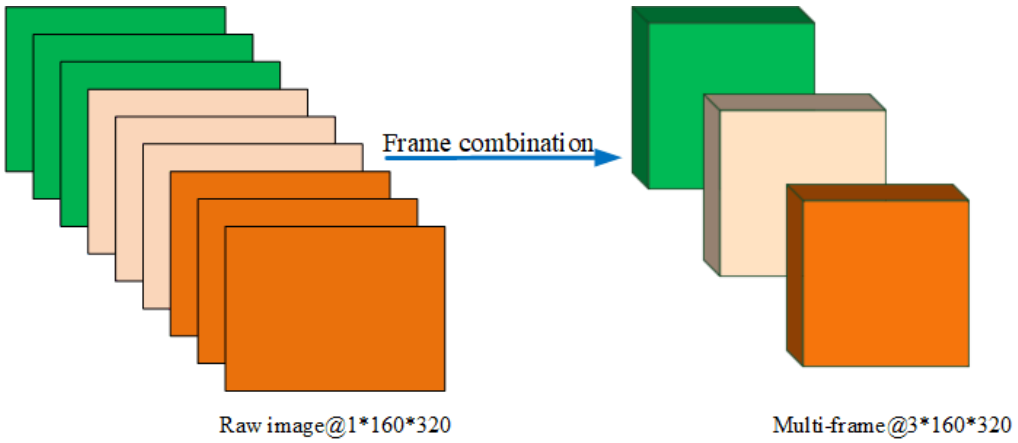

Figure 2-18 The data pre-processing approach which is used to transform the single frame image to multi-frame data

First, through the frame combination processing of the input image, the discrete image frame is input to the convolutional LSTM as time serial image segment for processing. Compared to traditional, they use shorter time series to generate multi-frame picture fragments. This is because taking into account the time-related efficiency is a very important factor in the movement of vehicles, so here a three-frame combination is used to generate timing clips and feed it into the proposed spatiotemporal network.

2.3.4 Spatiotemporal Feature Extraction

The proposed spatiotemporal LSTM network is based on convolutional LSTM (Conv-LSTM), 3D convolutional neural network(3D-CNN) and fully connected neural network (FCNN) to extract the spatiotemporal features of time serial image segment. In this proposed network, 4 Conv-LSTM layers are used and each of them is combined with a batch normalization layer (BN) to prevent the problem of low training efficiency caused by data distribution offset in deep networks. For traditional LSTM, it is often used in sequence-based processing systems because it has sequence-based data analysis and processing capabilities. However, traditional LSTM does not have a good ability to learn continuous pictures. Fortunately, the proposal of Conv-LSTM makes the temporal memory training based on multi-frame picture segments greatly enhanced. The output of the previous layer is the input of the next layer. The difference is that after the convolution operation is added, not only the timing relationship can be obtained, but also features can be extracted like a convolutional layer to extract spatial features. In this way, the spatiotemporal characteristics can be obtained. the core process of Conv-LSTM is shown as (Figure 2-19).

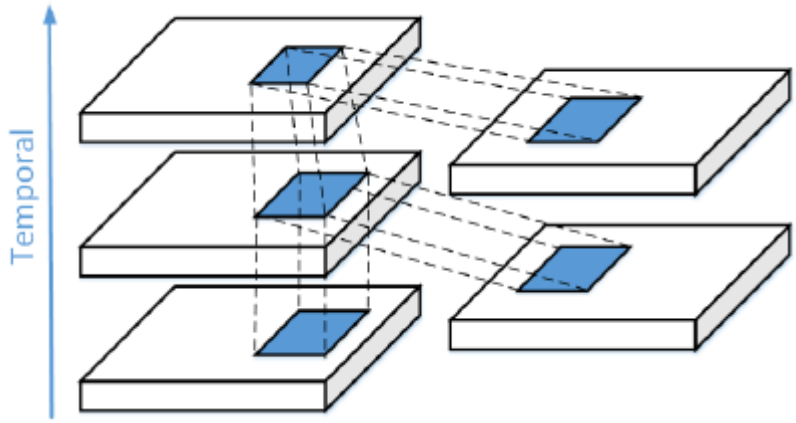

Figure 2-19 The core architecture of Conv-LSTM

As can be seen from the figure, the connection between the input and each gate at this time is replaced by convolution by the feedforward type, and the convolution operation is also replaced between the state and the state. In terms of the autonomous driving process, it is not a discrete behavior like object recognition but a continuous response through the recognition of the environment. So, in this proposed method, they use a four layers Conv-LSTM structure to extract the temporal hidden feature information and help the model learn better from the data. The working principle of Conv-LSTM used in this paper can be shown as follows:

$$it = \sigma\,(Wxi * Xt + Whi * Ht-1 + Wci \circ Ct-1 + bi) \quad (1)$$
$$ft = \sigma\,(Wxf * Xt + Whf * Ht-1 + Wcf \circ Ct-1 + bf) \quad (2)$$
$$Ct = ft + it \circ \tanh\,(Wxc * Xt + Whc * Ht-1 + bc) \quad (3)$$
$$ot = \sigma\,(Wxo * Xt + Who * Ht-1 + Wco \circ Ct + bo) \quad (4)$$
$$Ht = ot \circ \tanh\,(Ct) \quad (5)$$

Where it, ft, ct represent the value from the input gate, forget gate and output gate at moment t respectively, at the same time, the o means the Hadamard product between two matrices. The entire system of equations represents the state generating mechanism from moment t−1 to moment t. It should be noted that the X, C, H, i, f and o are all three-dimensional tensors. The latter two dimensions represent the spatial information of the rows and columns. (The first dimension should be the time dimension). So, it can predict the characteristics of the center grid based on the characteristics of the surrounding points in the grid. Considering that in the deep network training process, the low-level network updates the parameters during training and causes the distribution of upper-level input data to change. So, at the input of each layer of the network, a batch normalization layer is inserted, that means, a normalization process is performed first, and then the next layer of the network is entered. The main process of batch normalization is shown as follows:

$$\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \quad (6)$$

$$\sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2 \quad (7)$$

$$\widehat{x_i} \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \quad (8)$$

$$y_i \leftarrow \gamma\widehat{x_i} + \beta \equiv BN_{\gamma,\beta}(x_i) \quad (9)$$

Where xi represents the element of the input batch, m represents the size of the batch, so μ represents the overall mean of the batch. B is the variance of the batch, which represents the stability of this batch of data. b xi is the result of batch standardization of batch data. Finally, the parameters γ and β allow xi to be transformed and reconstructed to obtain the results of normalized data without changing the feature distribution. For the image process methods, the traditional way of using 2D-CNN to operate on time-serial images is generally to use CNN to identify each frame of the video. This method does not take into account the inter-frame motion information in the time dimension. Using 3D-CNN can better capture the temporal and spatial characteristics of the video. Traditional 2DCNN and the 3D-CNN convolution operation on the images are shown as (Figure 2-20) & (Figure 2-21):

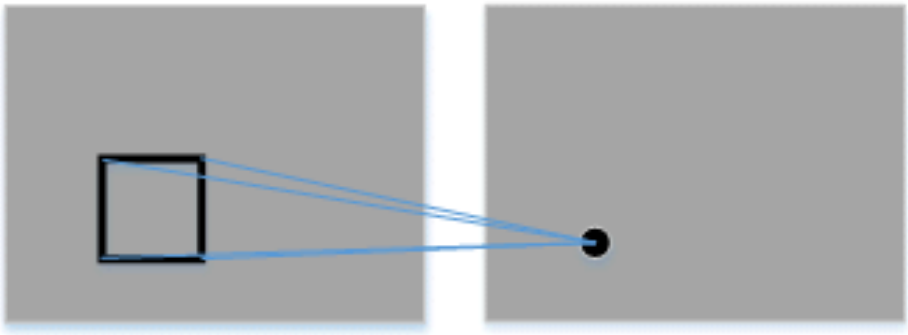

Figure 2-20 2DCNN convolution operation on the image using 2D convolution kernel.

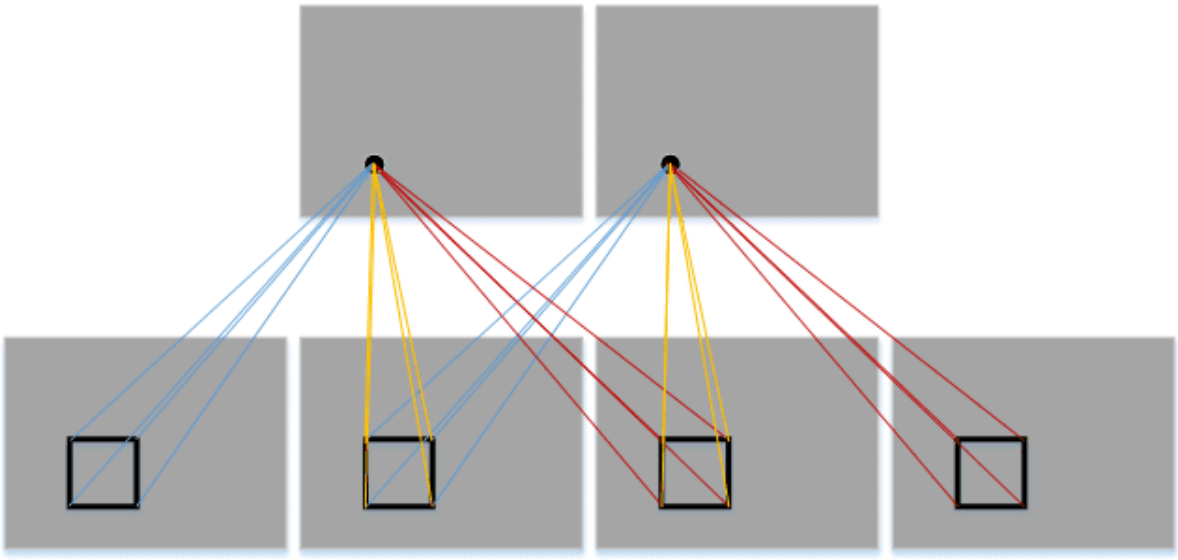

Figure 2-21 3D-CNN convolution operation on the image using 3D convolution kernel.

For 3D-CNN, the convolution operation has a time dimension of 3, which means a convolution operation on three consecutive frames of images. The above 3D convolution is to build a cube by stacking multiple consecutive frames, and then use the 3D convolution kernel in the cube. In this structure, each feature map in the convolutional layer is connected to multiple adjacent consecutive frames in the previous layer, thus capturing motion information. The value of a position of a convolutional map is obtained by convolving local receptive fields at the same position of three consecutive frames one level above.

2.3.5 Results & Analysis

The visualization experimental result proposed in this paper is shown in (Figure 2-22). The blue curves represent the real driving motion. At the same time, the green curves represent the planning motion generated by this paper's method. Obviously, the green and blue curves are very well-fitted which means that the proposed method works well for autonomous vehicles. During the training process, found that in each epoch of the training (generally from the 5th start), the loss value first declines rapidly then rebounds a bit and then steadily decreases until the epoch ends. After analysis, they reach the deep spatiotemporal neural network will always start training from the state of lower loss of the previous epoch during training. However, this proposed training data is time-based sequence data, which means maybe the model generate overfitting data based on the antecedent data from the previous moment first and then, through the optimizer, the model will keep finding the real general minimize state again. In order to further evaluate this method, they evaluate both this paper's and Hotz's methods. they have

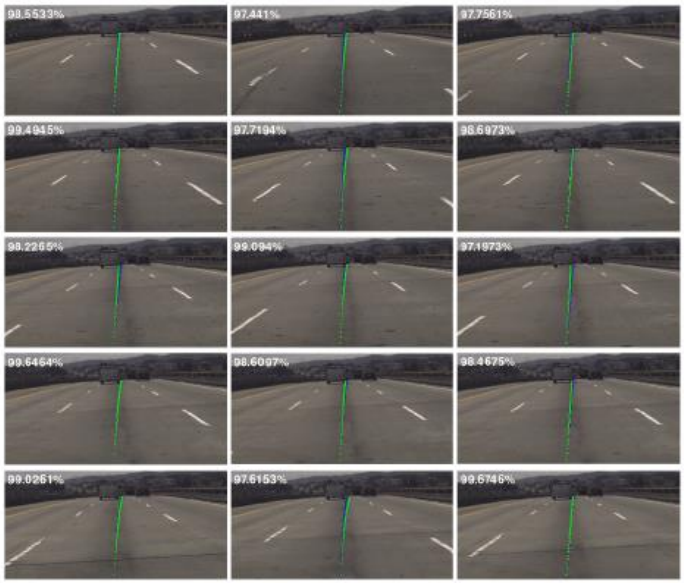

Figure 2-22 The visualization result of propos emotion planning model experiments

constructed a neural network based on 2D-CNN and FCNN according to Hotz's scheme. Then randomly selected a section of data on the highway and verified this paper's and Hotz's models respectively. Finally, the evaluation results are shown in (Figure 2-23). Through the figure can see that both this paper's and Hotz's method can make great motion planning for the real-time traffic environment. But it is clear that Hotz's approach has greater stability fluctuations. In comparison, this paper's method not only has a better average evaluation performance but also has a lower average variation performance. So, this paper's method can be more reliable and stable while generating autonomous motion planning results.

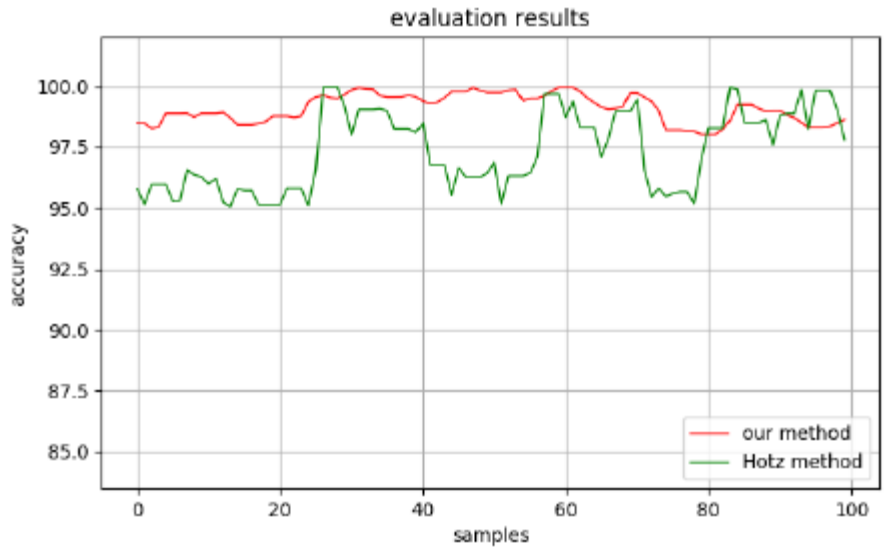

Figure 2-23 Experiments curve of this paper's and Hotz's methods

2.4 End-To-end Driving via Conditional Imitation Learning [11]

Figure 2-24 Conditional imitation learning allows an autonomous vehicle trained end-to-end to be directed by high-level commands. (a) We train and evaluate robotic vehicles in the physical world (top) and in simulated urban environments (bottom). (b) The vehicles

Imitation learning is receiving renewed interest as a promising approach to training autonomous driving systems. Demonstrations of human driving are easy to collect at scale. Given such demonstrations, imitation learning can be used to train a model that maps perceptual inputs to control commands; for example, mapping camera images to steering and acceleration. This approach has been applied to lane following [12], [13] and off-road obstacle avoidance [14]. However, these systems have characteristic limitations. For example, the network trained by Bojarski et al. [15] was given control over lane and road following only. When a lane change or a turn from one road to another were required, the human driver had to take control [13].

Why has imitation learning not scaled up to fully autonomous urban driving? One limitation is in the assumption that the optimal action can be inferred from the perceptual input alone. This assumption often does not hold in practice: for instance, when a car approaches an intersection, the camera input is not sufficient to predict whether the car should turn

left, right, or go straight. Mathematically, the mapping from the image to the control command is no longer a function. Fitting a function approximator is thus bound to run into difficulties. This had already been observed in the work of Pomerleau: “Currently upon reaching a fork, the network may output two widely discrepant travel directions, one for each choice. The result is often an oscillation in the dictated travel direction” [1]. Even if the network can resolve the ambiguity in favor of some course of action, it may not be the one desired by the passenger, who lacks a communication channel for controlling the network itself.

This paper addresses this challenge with conditional imitation learning. At training time, the model is given not only the perceptual input and the control signal but also a representation of the expert’s intention. At test time, the network can be given corresponding commands, which resolve the ambiguity in the perceptuomotor mapping and allow the trained model to be controlled by a passenger or a topological planner, just as mapping applications and passengers provide turn-by-turn directions to human drivers. The trained network is thus freed from the task of planning and can devote its representational capacity to driving. This enables scaling imitation learning to vision-based driving in complex urban environments.

This paper evaluates the presented approach in realistic simulations of urban driving and on a 1/5 scale robotic truck. Both systems are shown in (Figure 2-24). Simulation allows us to thoroughly analyze the importance of different modeling decisions, carefully compare the approach to relevant baselines, and conduct detailed ablation studies. Experiments with the physical system demonstrate that the approach can be successfully deployed in the physical world. Recordings of both systems are provided in the supplementary video.

2.4.1 Network Architecture

Assume that each observation **o** = {i, m} comprises an image **i** and a low-dimensional vector **m** that the paper refers to as measurements, following Dosovitskiy and Koltun [15]. The controller F is represented by a deep network. The network takes the image **i**, the measurements **m**, and the command **c** as inputs, and produces an action **a** as its output. The action space can be discrete, continuous, or a hybrid of these. In paper's driving experiments, the action space is continuous and two-dimensional: steering angle and acceleration. The acceleration can be negative, which corresponds to braking or driving backwards. The command **c** is a categorical variable represented by a one-hot vector.

they study two approaches to incorporating the command **c** into the network. The first architecture is illustrated in (Figure 2-25) (a). The network takes the command as an input, alongside the image and the measurements. These three inputs are processed independently by three modules: an image module I(i), a measurement module M(m), and a command module C(c). The image module is implemented as a convolutional network, the other two modules as fully-connected networks. The outputs of these modules are concatenated into a joint representation:

$$j = J(i, m, c) = \{I(i), M(m), C(c)\}$$

The control module, implemented as a fully-connected network, takes this joint representation and outputs an action A(j). The paper refers to this architecture as command input. It is applicable to both continuous and discrete commands of arbitrary dimensionality. However, the network is not forced to take the commands into account, which can lead to suboptimal performance in practice. they, therefore, designed an alternative architecture, shown in (Figure 2-25) (b).

The image and measurement modules are as described above, but the command module is removed. Instead, the paper assumes a discrete set of commands

c ={$\mathbf{c}^0$,…,$\mathbf{c}^k$} (including a default command $\mathbf{c}^0$ corresponding to no specific command given) and introduce a specialist branch $\mathbf{A}^i$ for each of the commands $\mathbf{c}^i$. The command **c** acts as a switch that selects which branch is used at any given time. The output of the network is this:

$$F(i, m, c^i) = A^i(J(i, m))$$

The paper refers to this architecture as branched. The branches $\mathbf{A}^i$ are forced to learn sub-policies that correspond to different commands. In a driving scenario, one module might specialize in lane following, another in right turns, and a third in left turns. All modules share the perception stream.

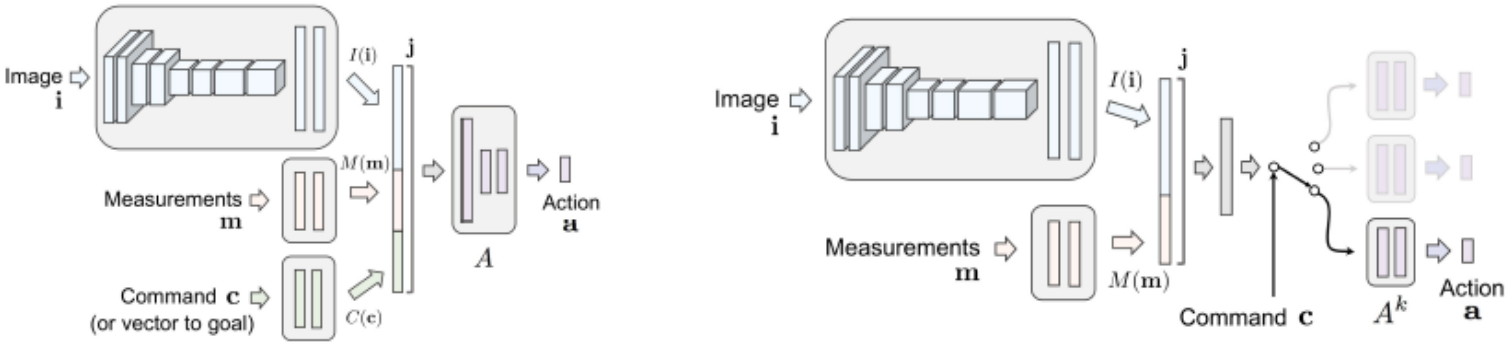

Figure 2-25 *Two network architectures for command-conditional imitation learning. (a) command input: the command is processed as input by the network, together with the image and the measurements. The same architecture can be used for goal-conditional learning (one of the baselines in our experiments), by replacing the command by a vector pointing to the goal. (b) branched: the command acts as a switch that selects between specialized sub-modules.*

2.4.2 Training Data Distribution

When performing imitation learning, a key decision is how to collect the training data. The simplest solution is to collect trajectories from natural demonstrations of an expert performing the task. This typically leads to unstable policies, since a model that is only trained on expert trajectories may not learn to recover from disturbance or drift [15], [17]. To overcome this problem, training data should include observations of recoveries from perturbations.

In DAgger[8] [16], the expert remains in the loop during the training of the controller: the controller is iteratively tested and samples from the obtained trajectories are re-labeled by the expert. In the system of Bojarski et al. [13], the vehicle is instrumented to record from three cameras simultaneously: one facing forward and the other two shifted to the left and to the right. Recordings from the shifted cameras, as well as intermediate synthetically re-projected views, are added to the training set with appropriately adjusted control signals – to simulate recovery from drift. In this paper, they adopt a three-camera setup inspired by Bojarski et al. [13].

However, they have found that the policies learned with this setup are not sufficiently robust. Therefore, to further augment the training dataset, they record some of the data while injecting noise into the expert's control signal and letting the expert recover from these perturbations. This is akin to the recent approach of Laskey et al. [18], but instead of i.i.d. the noise they inject temporally correlated noise designed to simulate gradual drift away from the desired trajectory. An example is shown in (Figure 2-26). For training, they use the driver's corrective response to the injected noise (not the noise itself). This provides the controller with demonstrations of recovery from drift and unexpected disturbances but does not contaminate the training set with demonstrations of veering away from desired behavior.

In Figure (2-26) The plot on the left shows steering control [rad] versus time [s]. In the plot, the red curve is an injected triangular noise signal, the green curve is the driver's steering signal, and the blue curve is the steering signal provided to the car, which is the sum of the driver's control and the noise. Images on the right show the driver's view at three points in time (trajectories overlaid post-hoc for visualization). Between times 0 and roughly 1.0, the noise produces a drift to the right, as illustrated in the image (a). This triggers a human reaction, from 1.0 to 2.5 seconds, illustrated in (b). Finally, the car recovers from the disturbance, as shown in (c). Only the driver-provided signal (green curve on the left) is used for training.

[8] An iterative algorithm that trains a deterministic policy that achieves good performance guarantees under its induced distribution of states

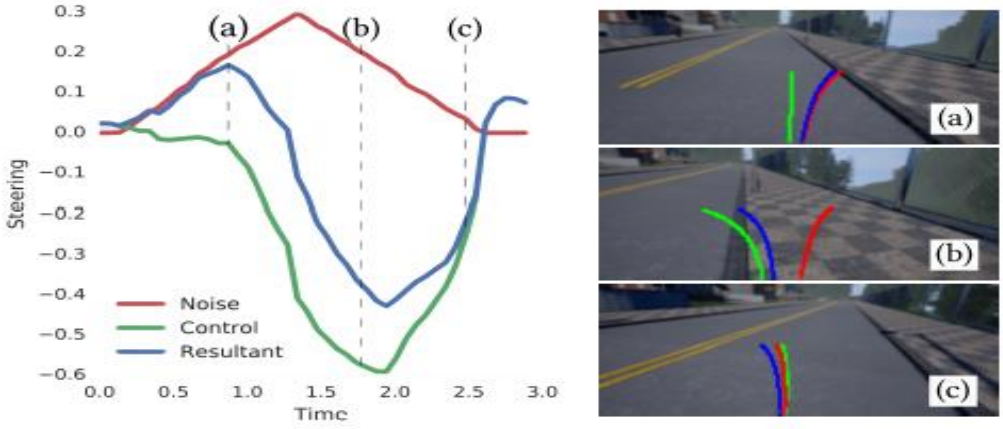

Figure 2-26 Noise injection during data collection. and a fragment show from an actual driving sequence from the training set.

2.4.3 Data Augmentation

They found data augmentation to be crucial for good generalization. So, they perform augmentation online during network training. For each image to be presented to the network, and they apply a random subset of a set of transformations with randomly sampled magnitudes. Transformations include a change, in contrast, brightness, and tone, as well as the addition of Gaussian blur, Gaussian noise, salt-and-pepper noise, and region dropout (masking out a random set of rectangles in the image, each rectangle taking roughly 1% of the image area).

2.4.4 System Setup

This paper evaluated the presented approach in a simulated urban environment and on a physical system – a 1/5 scale truck. In both cases, the observations (images) are recorded by one central camera and two lateral cameras rotated by 30 degrees with respect to the center. The recorded control signal is two-dimensional: steering angle and acceleration. The steering angle is scaled between -1 and 1, with extreme values corresponding to full left and full right, respectively.

The acceleration is also scaled between -1 and 1, where 1 corresponds to full forward acceleration and -1 to full reverse acceleration.

In addition to the observations (images) and actions (control signals), the commands provided by the driver has recorded, and they use a set of four commands: continue (follow the road), left (turn left at the next intersection), straight (go straight at the next intersection), and right (turn right at the next intersection). In practice, they represent these as one-hot vectors.

During training data collection, when approaching an intersection, the driver uses buttons on a physical steering wheel (when driving in simulation) or on the remote control (when operating the physical truck) to indicate the command corresponding to the intended course of action. The driver indicates the command when the intended action becomes clear, akin to turn indicators in cars or navigation instructions provided by mapping applications. This way they collect realistic data that reflects how a higher-level planner or a human could direct the system.

2.4.5 Simulated Environment

In this paper they use CARLA [18], an urban driving simulator, to corroborate design decisions and evaluate the proposed approach in a dynamic urban environment with traffic. CARLA is an open-source simulator implemented using Unreal Engine 4[9]. It contains two professionally designed towns with buildings, vegetation, and traffic signs, as well as vehicular and pedestrian traffic. (Figure 2-27) provides maps and sample views of Town 1, used for training, and Town 2, used exclusively for testing.

[9] Unreal Engine is the world's most open and advanced real-time 3D creation platform for photoreal visuals and immersive experiences.

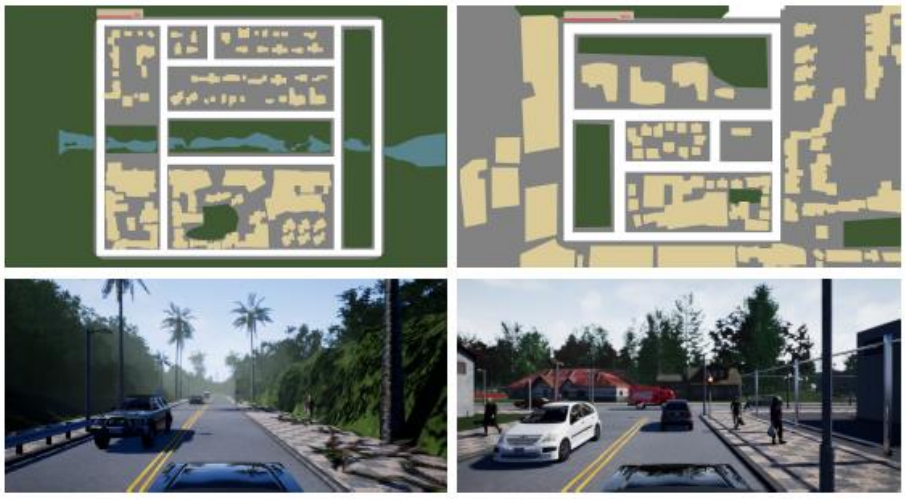

Figure 2-27 Simulated urban environments.

To collect training data, a human driver is presented with a first-person view of the environment (center camera) at a resolution of 800 600 pixels. The driver controls the simulated vehicle using a physical steering wheel and pedals and provides command input using buttons on the steering wheel.

The driver keeps the car at a speed below 60 km/h and strives to avoid collisions with cars and pedestrians, but ignores traffic lights and stop signs. the recorded images are from the three simulated cameras, along with other measurements such as speed and the position of the car.

The images are cropped to remove part of the sky. CARLA also provides extra information such as distance traveled, collisions, and the occurrence of infractions such as drift onto the opposite lane or the sidewalk. This information is used in evaluating different controllers.

3. Chapter Three (Big Data)

3.1 Overview

90% of the available data has been created in the last two years and the term Big Data has been around 2005, when it was launched by O'Reilly[10] Media. However, the usage of Big Data and the need to understand all available data has been around much longer.

Big Data refers to complex and large data sets that none of the traditional data management tools are able to store and process it efficiently to uncover valuable information that can aid in decision making. (add result sentence)

3.2 Big Data 5 Vs

As known Big Data comprises from 3Vs to the extended model of 5Vs:

1. **Volume**: refers to an amount of data or size of data that can be in quintillion when comes to big data. Handling terabytes of data instead of hundreds of gigabytes.

2. **Velocity**: refers to how fast data is growing; data is exponentially growing and at a very fast rate. Big data works in a distributed system, that would increase the speed of handling and managing the data than the traditional one.

3. **Variety:** refers to different types of data like social media, web server logs etc. Data can come from different sources.

4. **Veracity:** refers to an uncertainty of data like social media means if the data can be trusted or not.

[10] A learning company that publishes books, produces tech conferences, and provides an online learning platform.

5. **Value:** refers to the data which we are storing and processing is worth and how we are getting benefit from this huge amount of data.

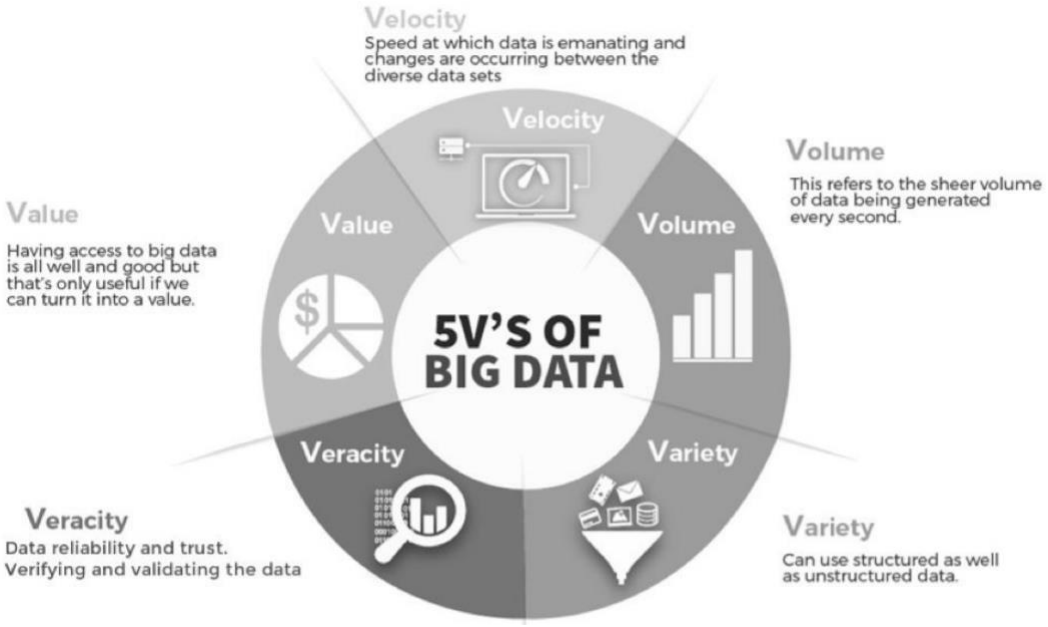

Figure 3-1 Big Data Illustration with 5V's

3.3 Big Data Types

Forms of Big Data refers to a huge volume of data that can be structured, semi- structured and unstructured.

1. **Structured:** By structured data, we mean data that can be processed, stored, and retrieved in a fixed format. It refers to highly organized information that can be readily and seamlessly stored and accessed from a database by simple search engine algorithms. For instance, the employee table in a company database will be structured as the employee details, their job positions, their salaries, etc., will be present in an organized manner.

2. **Unstructured:** Unstructured data refers to the data that lacks any specific form or structure whatsoever. This makes it very difficult and time consuming to process and analyze unstructured data. Email is an example of unstructured data.

3. **Semi-structured**: Semi-structured data pertains to the data containing both the formats mentioned above, that is, structured and unstructured data. To be precise, it refers to the data that although has not been classified under a particular repository (database), yet contains vital information or tags that segregate individual elements within the data. [19]

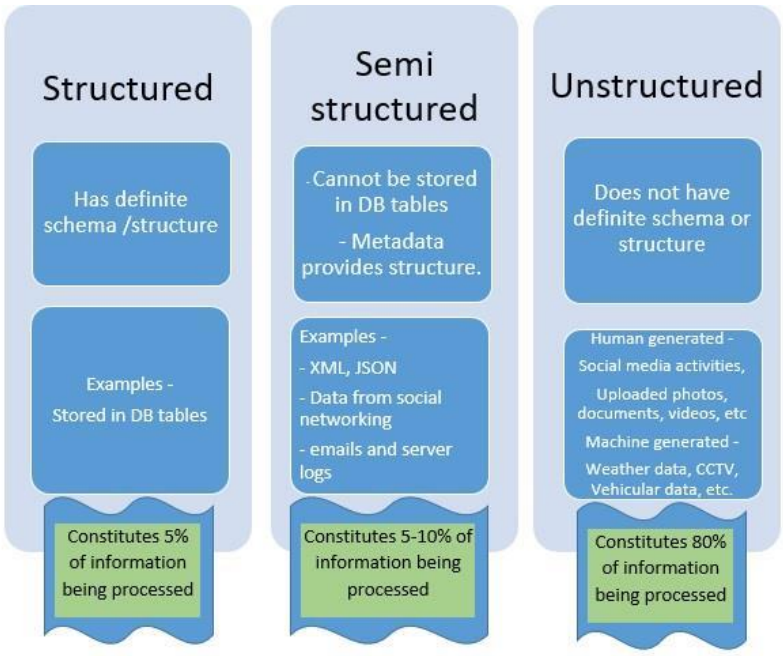

Figure 3-2 Big Data Forms

3.4 Data Sources

Big data is used by organizations for many purposes. However, before companies can set out to extract insights and valuable information from big data, they must know several big data sources available. Data, as we know, is massive and exists in various forms. If it is not classified or sourced well, it can end up wasting precious time and resources. To achieve success with big data, companies must have the know-how to sift between the various data sources available and accordingly classify its usability and relevance.

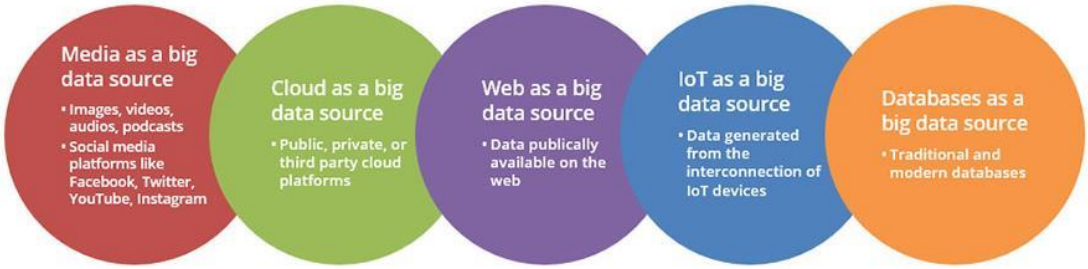

Figure 3-3 Big Data Different Sources

- **MEDIA AS A BIG DATA SOURCE**

  Media is the most popular source of big data, as it provides valuable insights on consumer preferences and changing trends. Since it is self-broadcasted and crosses all physical and demographical barriers, it is the fastest way for businesses to get an in-depth overview of their target audience, draw patterns and conclusions, and enhance their decision-making. Media includes social media and interactive platforms, like Google, Facebook, Twitter, YouTube, Instagram, as well as generic media like images, videos, audios, and podcasts that provide quantitative and qualitative insights on every aspect of user interaction.

- **CLOUD AS A BIG DATA SOURCE**

Today, companies have moved ahead of traditional data sources by shifting their data on the cloud. Cloud storage accommodates structured and unstructured data and provides business with real-time information and on-demand insights. The main attribute of cloud computing is its flexibility and scalability. As big data can be stored and sourced on public or private clouds, via networks and servers, cloud makes for an efficient and economical data source.

- **THE WEB AS A BIG DATA SOURCE**

The public web constitutes big data that is widespread and easily accessible. Data on the Web or 'Internet' is commonly available to individuals and companies alike. Moreover, web services such as Wikipedia provide free and quick informational insights to everyone. The enormity of the Web ensures for its diverse usability and is especially beneficial to start-ups and SME's, as they don't have to wait to develop their own big data infrastructure and repositories before they can leverage big data.

- **IOT AS A BIG DATA SOURCE**

Machine-generated content or data created from IoT constitute a valuable source of big data. This data is usually generated from the sensors that are connected to electronic devices. The sourcing capacity depends on the ability of the sensors to provide real-time accurate information. IoT is now gaining momentum and includes big data generated, not only from computers and smartphones but also possibly from every device that can emit data. With IoT, data can now be sourced from medical devices, vehicular processes, video games, meters, cameras, household appliances, and the like.

- **DATABASES AS A BIG DATA SOURCE**

  Businesses today prefer to use an amalgamation of traditional and modern databases to acquire relevant big data. This integration paves the way for a hybrid data model and requires low investment and IT infrastructural costs. Furthermore, these databases are deployed for several business intelligence purposes as well. These databases can then provide for the extraction of insights that are used to drive business profits. Popular databases include a variety of data sources, such as MS Access, DB2, Oracle, SQL, and Amazon Simple, among others.

3.5 Big Data & Self-Driving Cars

3.5.1 Sensors

This section is going to describe the main types of sensors are used to guide an autonomous car, then will discuss sensor metrics and how they relate to some of the important aspects of automated driving.

3.5.2 Sensor Metrics

The sensor metrics will ultimately determine the performance of the sensor in a specific application, and therefore it is important to know what these metrics mean. The following metrics are the most important when selecting a sensor for automotive purposes:

- **Range:** This determines what the sensor can cover in terms of distance.

- **Field-of-view (FOV):** This is the opening angle of the sensor, and together with the range, it provides a pie-slice shaped coverage pattern. An example is shown in (Figure 3-4).

- **Angular**: How much the measure dangle actually deviates from the real angle.

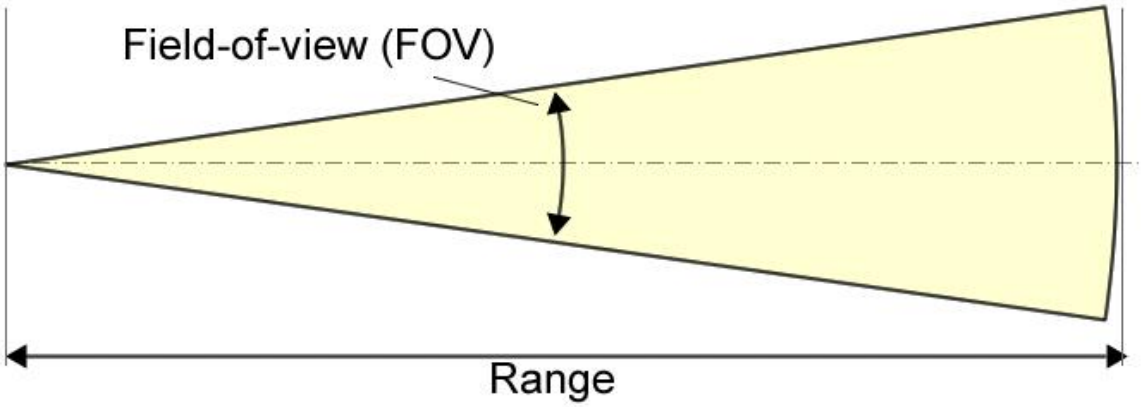

Figure 3-4 The range and FOV of a sensor determine the detection coverage

The range and FOV together determine what the sensor can cover. It should be noted that the FOV is composed of a horizontal and vertical component. (Figure 3-4). only shows a 2D illustration.

3.5.3 Sensor Types

The three primary autonomous vehicle sensors are camera, radar, and lidar. Working together, they provide the car visuals of its surroundings and help it detect the speed and distance of nearby objects, as well as their three-dimensional shape.[20]

- **The Camera**

Autonomous vehicles rely on cameras placed on every side front, rear, left and right, to stitch together a 360-degree view of their environment. Some have a wide field of view as much as 120 degrees and a shorter range. Others focus on a narrower view to provide long-range visuals.

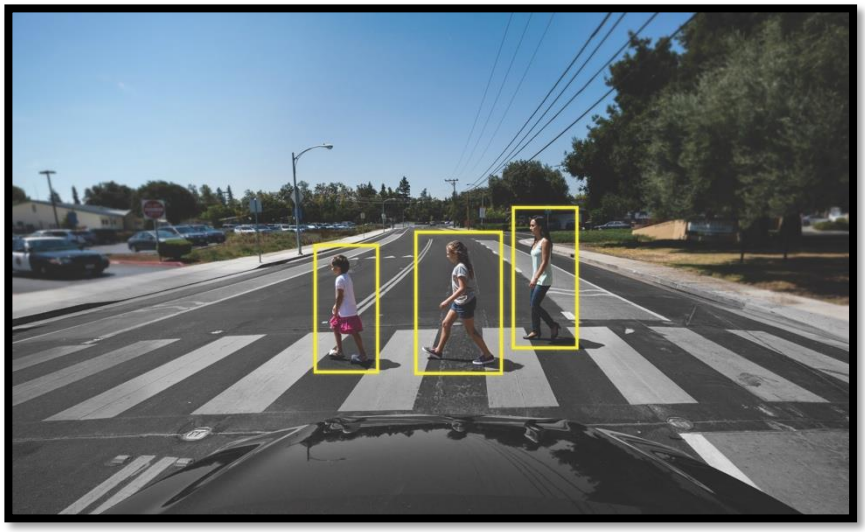

Figure 3-5An autonomous vehicle uses camera data to perceive objects in its environment

Some cars even integrate fish-eye cameras, which contain super-wide lenses that provide a panoramic view, to give a full picture of what's behind the vehicle for it to park itself.

Though they provide accurate visuals, cameras have their limitations. They can distinguish details of the surrounding environment; however, the distances of those objects need to be calculated to know exactly where they are. It's also more difficult for camera-based sensors to detect objects in low visibility conditions, like fog, rain or nighttime.[20]

- **The Radar**

Radar sensors can supplement camera vision in times of low visibility, like night driving, and improve detection for self-driving cars.
Radars transmit radio waves in pulses. Once these waves hit an object they bounce back to the sensor, providing data on the speed and location of the object.

Like the vehicle's cameras, radar sensors typically surround the car to detect objects at every angle. They're able to determine speed and distance, however, they can't distinguish between different types of vehicles.

While the data provided by surround radar and camera are sufficient for lower levels of autonomy, they don't cover all situations without a human driver. That's where lidar comes in. [20]

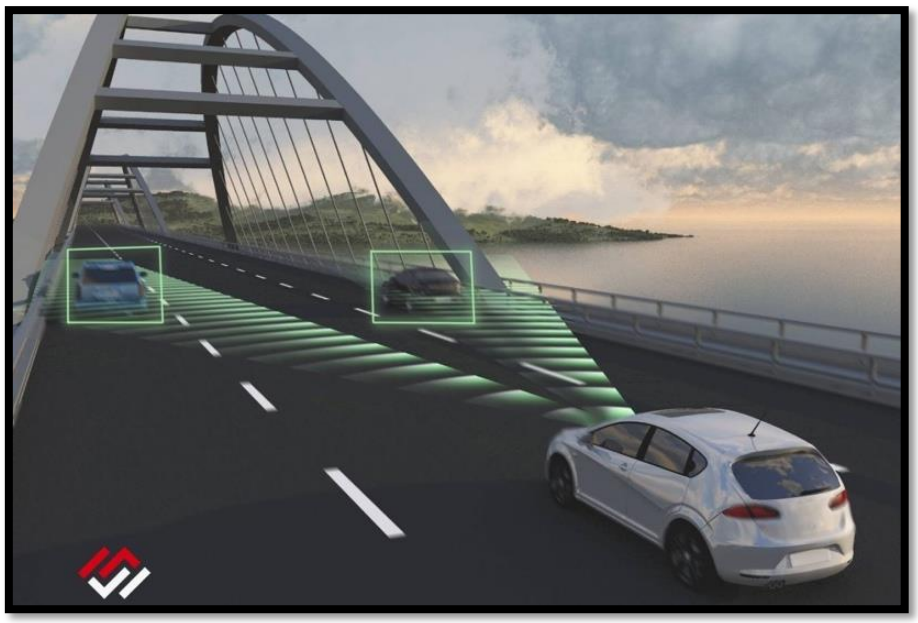

Figure 3-6 Radar sensing technology

- **The Lidar**

Lidar is an acronym of Light Detection and Ranging. If we ever want to achieve a fully driverless car (level 5 automation), we must get used to using lidar sensors. These unique sensors make it possible to have a 3D vision around the car. It provides shape and depth to surrounding cars and pedestrians as well as the road geography. Similarly, to radars, it works just as well in low-light conditions or when the cameras are covered.[20]

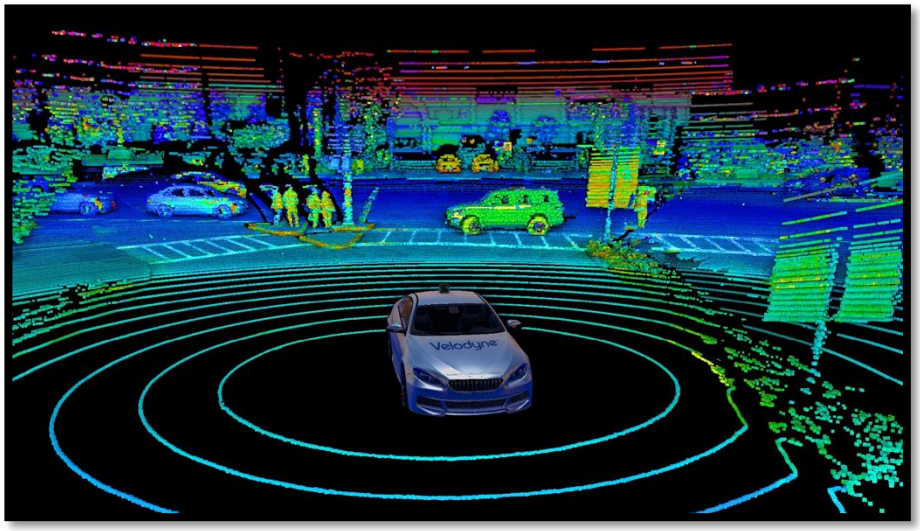

Figure 3-7 Lidar sensing technology

However, the Lidar sensors are more expensive to implement as much as ten times the cost of camera and radar, and have a more limited range. So, in our project we had focused on Camera and Radar sensors.

4. Chapter Four (Methodology)

The proposed self-driving car system that is used in this project uses the following methodologies.

4.1 Deep Learning

Deep learning is the process of learning the patterns in the data using artificial neural networks (ANN), an artificial neural network with 2 or less than 2 layers is called a shallow neural network (Figure 4-1), whereas an artificial neural network with 3 or more than 3 layers is called a deep neural network (Figure 4-2).

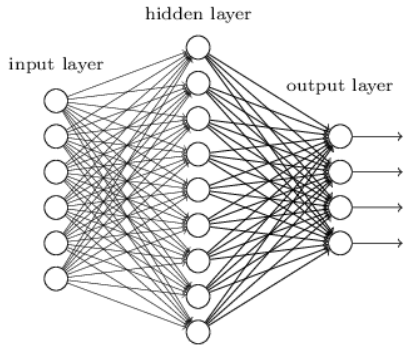

Figure 4-1 Shallow Neural Network

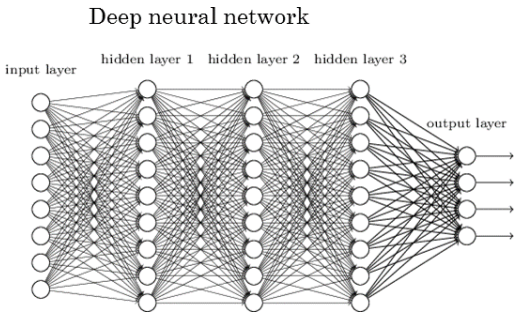

Figure 4-2 Deep Neural Network

Deep learning takes over the following domains:

- **Computer Vision:** Face recognition, self-driving car, object detection…etc.
- **Natural Language Processing:** Machine translation (English to Arabic) …etc.
- **Time Series Forecasting:** Temperature prediction, stock price prediction...etc.
- **Speech Recognition**: Recognizing a spoken language**.**

4.2 Representation Learning

In a deep neural network, each hidden layer learns to represents the input with its own representation. Earlier layer learns to have primary representation such as edges of an image, whereas deeper layers learn to have abstract representations such as faces, cars in an image…etc (Figure 4-3).

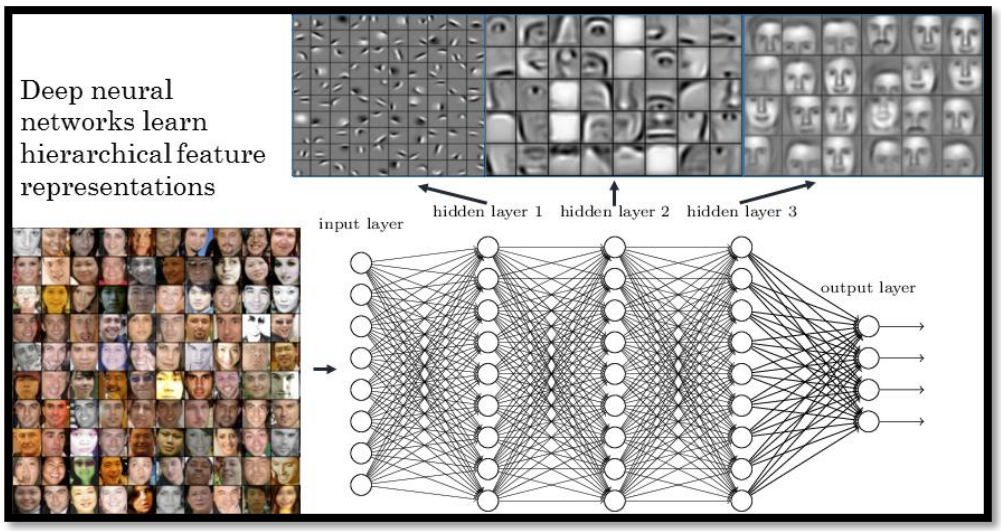

Figure 4-3 Representation Learning in Neural Network

4.3 Deep Learning & Big Data

Deep learning has been raised in the era of big data, because the performance of deep neural networks increases with the increasing of data volume. Traditional machine learning algorithms such as (Logistic regression, Support vector machines...etc.) have this limitation (Figure 4-4).

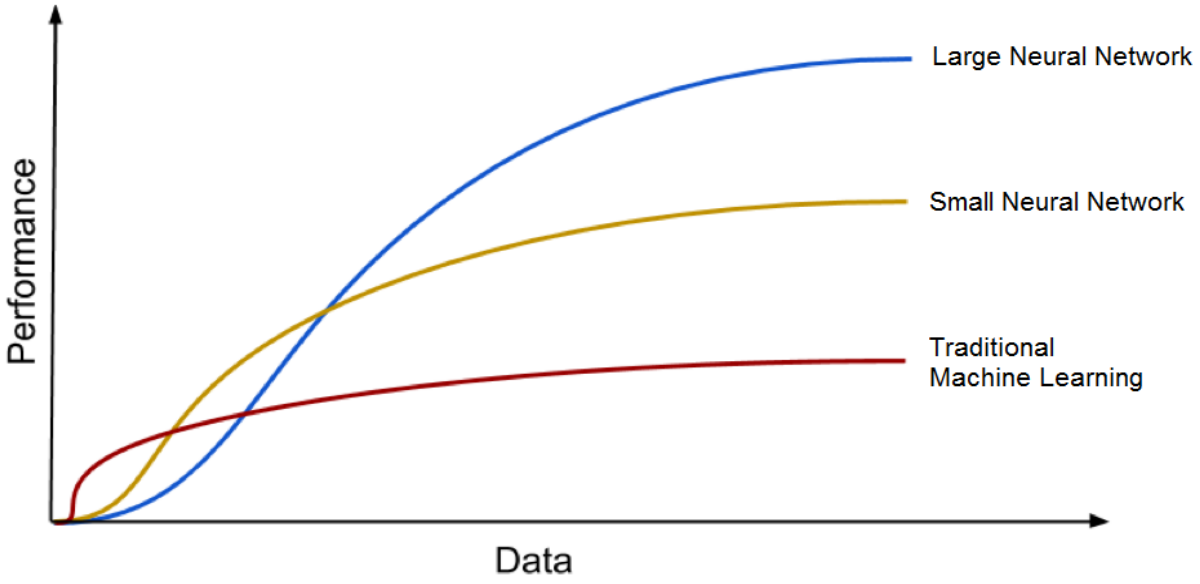

Figure 4-4 Deep Learning in The Big Data Era

Deep neural networks have several architectures which are:

4.4 Convolutional Neural Network

Convolutional neural network (CNN) has 3 major components:

- **Convolution Layer:** which takes an image multi-dimensional matrix as an input and convolves this matrix with a kernel, the output is a multi-dimensional matrix.

- **Max Pool Layer:** This layer is responsible to reduce the number of features reduce the dimensions of the multi-dimensional matrix by applying a kernel with a certain size to the output of the convolution layer which is a multi-dimensional matrix, the output is the reduced multi-dimensional matrix.

- **Fully Connected Layer:** This layer is the output layer.

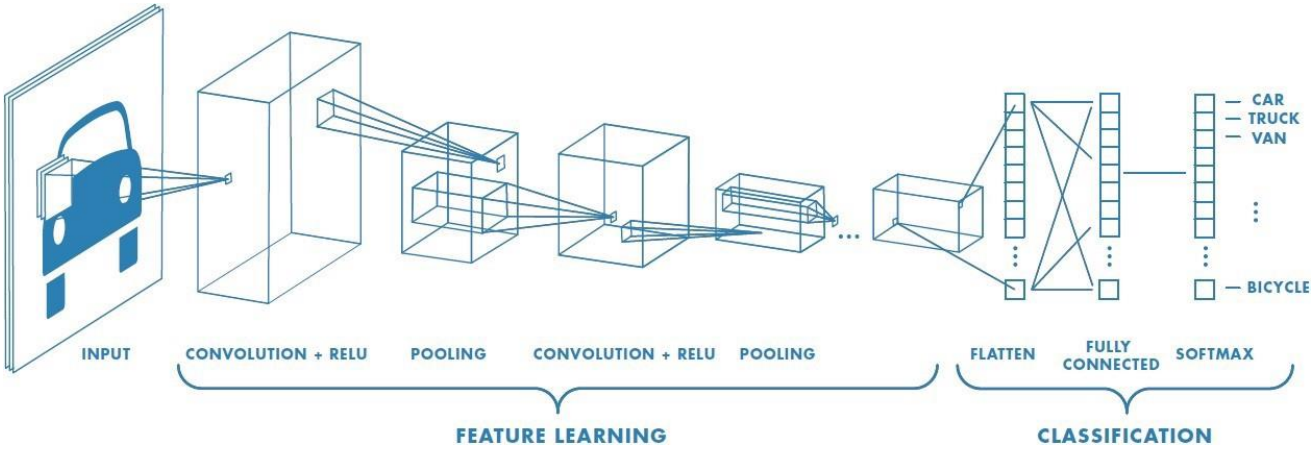

Figure 4-5 CNN Architecture

4.5 Recurrent Neural Network

Recurrent neural network (RNN) architectures focus on the order of an input of sequence such as text where the order of words in the text have has a meaning.

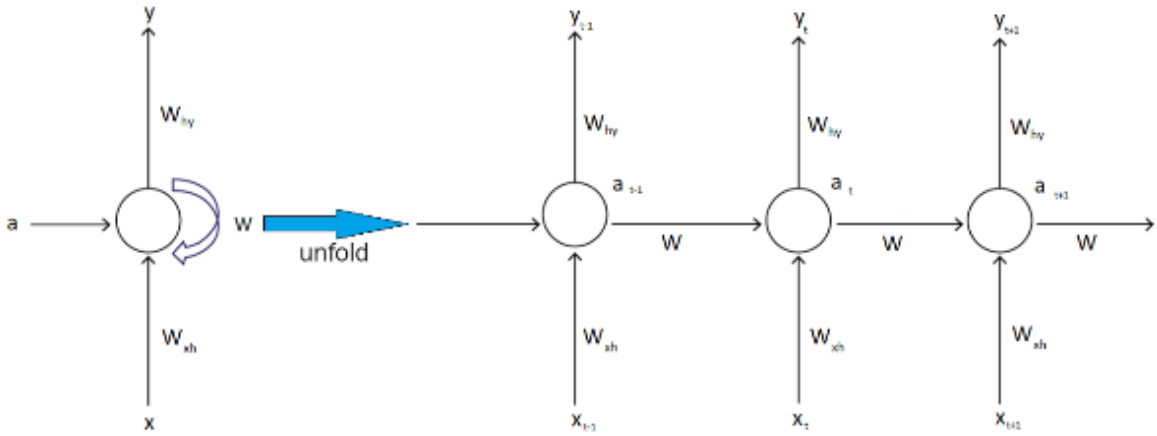

Figure 4-6 RNN Architecture

RNN architecture has several types:

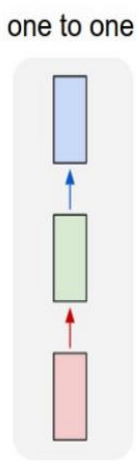

Figure 4-7 One to One Architecture

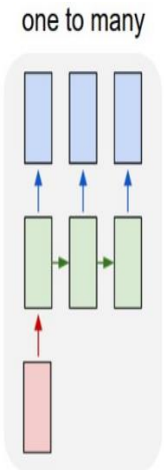

Figure 4-8 One to Many Architecture

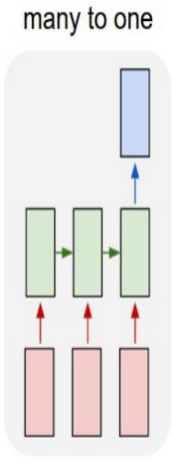

Figure 4-9 Many to One Architecture

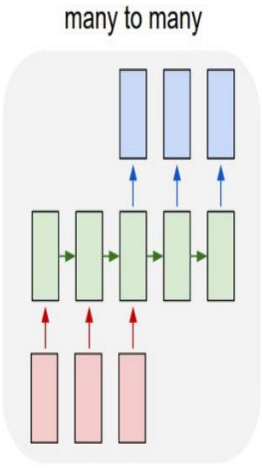

Figure 4-10 Many to Many Architecture

- **One to one architecture:** It is a regular neural network which takes for example an image and outputs a classification prediction.

- **One to many architectures:** It takes an input image for example and outputs a sequence prediction for example an image description as a text.

- **Many to One architecture:** It takes a sequence text, speech…etc. as input and outputs for example a classification prediction.

- **Many to Many architectures**: It takes a sequence as input text, speech…etc. and outputs a sequence.

4.6 Multi-Layer Perceptron

Multi-layer perceptron (MLP) or fully connected neural network (FC) is regular neural network which takes a vector of certain dimension as an input and outputs a classification prediction, regression prediction.

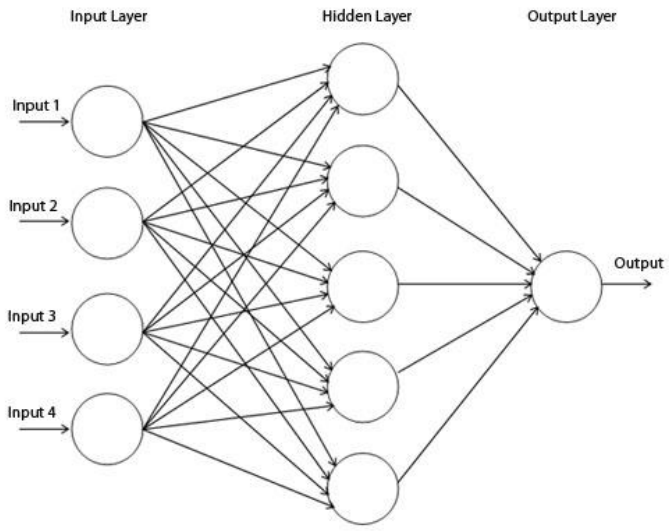

Figure 4-11Fully Connected Neural Network

4.6 Deep Neural Network as Computational Graph

Computational Graph is a set of nodes and a set of edges. Each edge can be considered as a multi-dimensional matrix/tensor that we called, each node is a mathematical operation such as a convolutional layer or max-pooling layer or activation function.

A neural network is a computational graph that is each layer is a node and each edge is a tensor / multi-dimensional matrix. When we build a deep model, we build our computational graph. After building the graph, we should think about how can we optimize train our computational graph, we should use the optimization algorithms to optimize train our graph through minimizing the error of our graph such gradient descent.

In order to train our graph, we must specify a learning rate 0.001 traditionally and pass it to the optimization algorithm, then when the graph computes the final node which means the output layer, then the backpropagation algorithm computes the gradients of the nodes and edges in the graph with respect to the loss then the graph updates its edges through the gradient descent optimization procedure in order to minimize the error through this basic formula:

$$w := w - \alpha \frac{\partial error}{\partial w}$$

A computational graph reduces the loss through the formula that is given above in an iterative approach which means the graph is fed with a batch of samples in the training set such as 64 batch size then apply the backpropagation to that batch and updates its parameters using the gradient descent with respect to that error's batch and keep feeding batches until the last batch and start from the first batch and keep training and so on.

4.7 Deep Learning for Self-Driving Car

In order to make a car drive from a starting point to a target point in an environment, we should have a navigator that directs the car in an intersection for example LEFT, RIGHT or STRAIGHT.

In this project we have used the following methodologies in order to implement a self-driving car system:

- **A* algorithm**: A* algorithm finds the shortest path from a starting point (A) to an ending point (B), the A* in our self-driving car system directs the car agent through a list of commands LEFT, RIGHT or STRAIGHT in an intersection.

Figure 4-12 Left Command from A*

Figure 4-13 Right Command from A*

Figure 4-14 Straight Command from A*

- **End-To-End Deep Neural Network:** The term End-to-End means learning to drive from point A to point B directly in one step without going through intermediate steps object detection, semantic segmentation…etc. End-to-End learning requires a huge amount of data in order to learn the driving task.

- **Multi-Task Learning:** The term multi-task learning means training a deep neural network on multiple tasks at the same time steering, press brake in order to stop the car or press gas in order to move the car…etc. When we have more than one loss in a deep neural network, then we are applying multi-task learning. Multi-task learning works well on complex tasks such as driving instead of building multiple models in order to make the car drive, building only one model to learn to control the car is better.

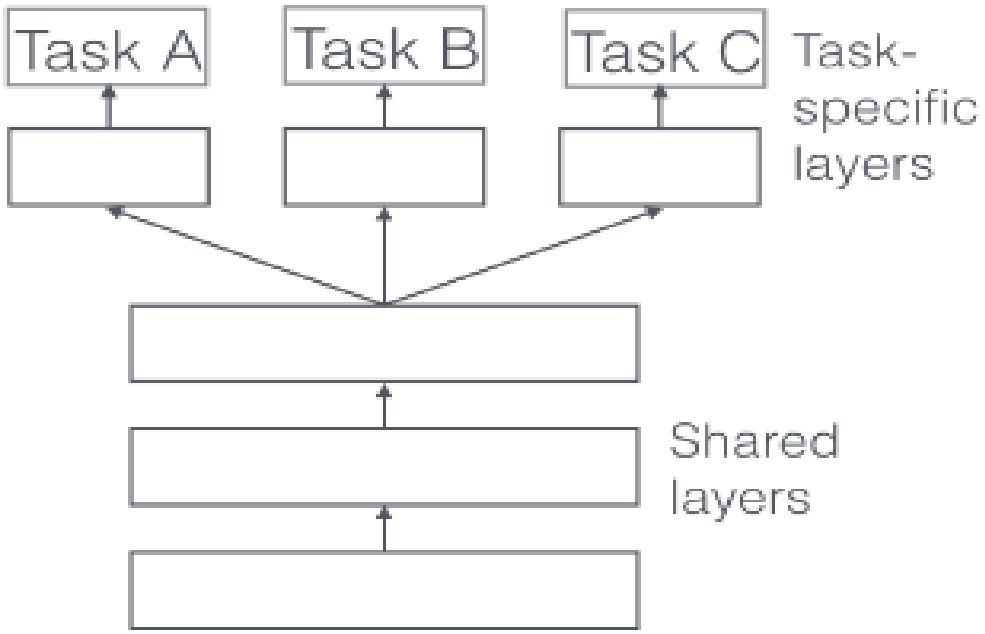

Figure 4-15 Multi-Task Learning

- **Imitation Learning:** The most important part in this project is how to train a deep neural network or a computation graph to perform such task. Imitation supervised learning plays an important role in the training of the computation graph, imitation learning is the behavioral cloning of an expert that is performing a task driving a car, cooking…etc.

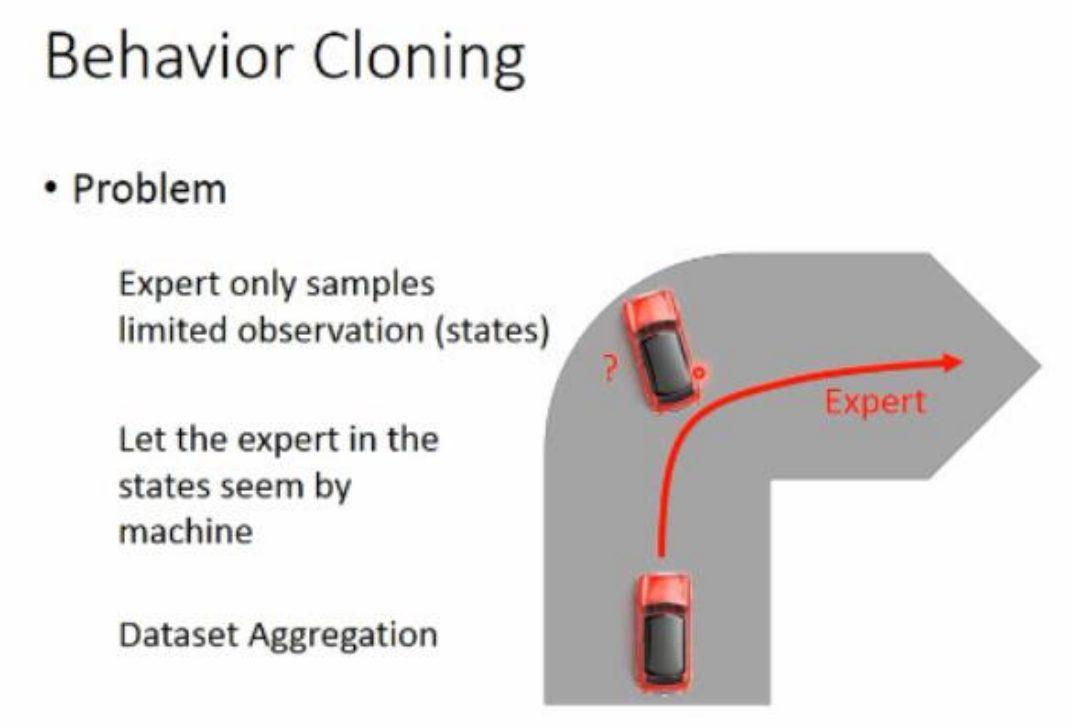

Figure 4-16 Behavioral Cloning

- **Dataset Aggregation:** Collecting a driving dataset in one pass is not enough to learning driving task, dataset aggregation training algorithm plays an important role in the learning process, dataset aggregation training algorithm is an iterative training process, in the initial time an expert collects its behaviors, then the expert trains the deep neural network on the initial dataset, then the expert tests the deep network in the driving environment, then the network makes mistakes, for example, steering the car left instead of steering the car right…etc., then the expert tries to correct the network and collect the corrections of the mistakes then, the expert trains the deep network again on the initial dataset and the corrections of mistakes and tests again to collect more mistakes and train…and so on.

Algorithm 43 SUPERVISEDIMITATIONTRAIN($\mathcal{A}, \tau_1, \tau_2, \ldots, \tau_N$)

$D \leftarrow \langle (x, a) \ : \ \forall n \ , \ \forall (x, a, \ell) \in \tau_n \rangle$ // collect all observation/action pairs

return $\mathcal{A}(D)$ // train multiclass classifier on D

Algorithm 44 SUPERVISEDIMITATIONTEST(f)

for $t = 1 \ldots T$ **do**

  $x_t \leftarrow$ current observation

  $a_t \leftarrow f(x_t)$ // ask policy to choose an action

  take action a_t

  $\ell_t \leftarrow$ observe instantaneous loss

end for

return $\sum_{t=1}^{T} \ell_t$ // return total loss

Figure 4-17 Imitation Learning Training Algorithm

Algorithm 45 DAGGERTRAIN($\mathcal{A}$, MaxIter, N, expert)

$\langle \tau_n^{(0)} \rangle_{n=1}^{N} \leftarrow$ run the expert N many times

$D_0 \leftarrow \langle (x, a) \ : \ \forall n \ , \ \forall (x, a, \ell) \in \tau_n^{(0)} \rangle$ // collect all pairs (same as supervised)

$f_0 \leftarrow \mathcal{A}(D_0)$ // train initial policy (multiclass classifier) on D_0

for $i = 1 \ldots MaxIter$ **do**

  $\langle \tau_n^{(i)} \rangle_{n=1}^{N} \leftarrow$ run policy f_{i-1} N-many times // trajectories by f_{i-1}

  $D_i \leftarrow \langle (x, \text{expert}(x)) \ : \ \forall n \ , \ \forall (x, a, \ell) \in \tau_n^{(i)} \rangle$ // collect data set

// observations x visited by f_{i-1}

// but actions according to the expert

  $f_i \leftarrow \mathcal{A}\left(\bigcup_{j=0}^{i} D_j \right)$ // train policy f_i on union of all data so far

end for

return $\langle f_0, f_1, \ldots, f_{\text{MaxIter}} \rangle$ // return collection of all learned policies

Figure 4-18 Dataset Aggregation Training Algorithm

The inputs of the deep neural network of the self-driving car are:

- **Camera Sensors:** Front, left and right sensor images of size $(200 \times 200 \times 3)$.

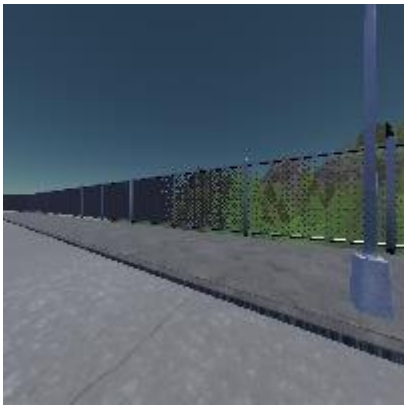

Figure 4-19 Right Camera Sensor

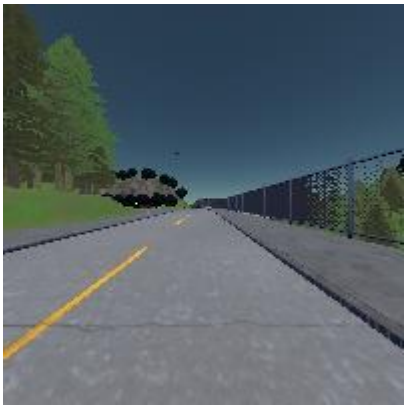

Figure 4-20 Front Camera Sensor

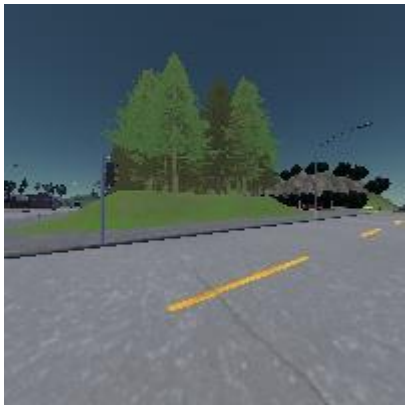

Figure 4-21 Left Camera Sensor

- **Speed Measurement**: The speed measurement has a range [0, 100] and its unit is km/h.

- **A* Command**: The deep neural network of the self-driving car system is conditioned on A* algorithm where the deep network takes from the A* algorithm a command STRAIGHT, LEFT or RIGHT at each time and follows the A* command.

The deep neural network of this system is conditioned on A* high level planner, it is end-to-end network and multi-task deep network which learns multiple tasks at a time. Multi-task deep model has multiple outputs. The outputs of the deep neural network of the self-driving car are:

- **Steer Angle:** The Steer angle is a regression output layer has range between [-1, +1].
- **Acceleration:** Acceleration is a classification output layer and has 3 types of values:

    1. Accelerate: Accelerate means Gas.
    2. Brake: Which is stop
    3. Normal: Do nothing

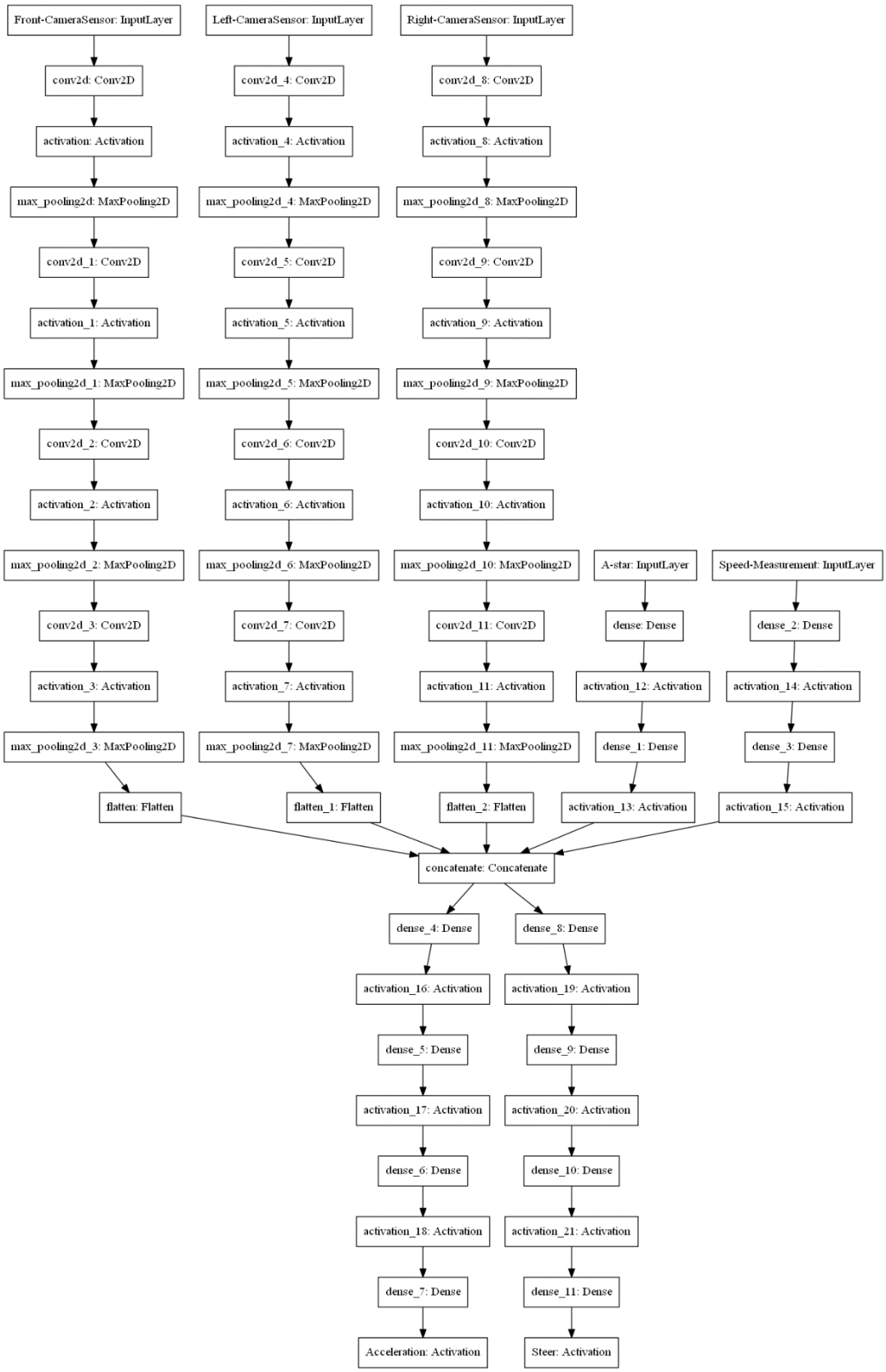

Figure 4-22 The Computational Graph of the Self-Driving-Car System

(Figure 4-22) illustrates the term End-to-End learning, the camera sensors, A* command and the speed measurement inputs are one end and the steer and acceleration outputs are the other end which becomes End-to-End.

5. Chapter Five (Experimental Results & Analysis)

5.1 Configuring the Deep Neural Network & The Urban Environment

In order to build a self-driving car system first, we have to choose and study the chosen urban environment of the self-driving car, for example, the road lane of the urban environment, the traffic signs and, the vehicles…etc. then, we have to build the appropriate deep neural network of the environment in the term of the FPS performance metric which means we have to build the computational graph that achieves at least 12 FPS for example and minimizes the error of the computation graph against the expert's dataset. In our self-driving car system, we chose the value of [10,12] FPS to be the minimum value the deep neural network should achieve then, we have applied the A* algorithm to the chosen urban environment in order to receive the command to drive from point A to point B in the shortest path.

To summarize the main point in order to build a self-driving car system:

- Choosing and study the urban environment the road lane, traffic signs and, vehicle number…etc.
- Building the deep neural network model that satisfies a certain FPS metric [10,12] FPS which is the range that is used in this project.
- Building a high-level planner such as A* which is used in this project in order to receive the commands that direct us to drive from point A to point B in the shortest path.

The A* algorithm is used in this project to direct the deep neural network in order to drive from point A to point B. When the deep neural network reaches a road intersection, the A* algorithm should direct the deep neural network in which road the deep network should drive STRAIGHT, RIGHT or LEFT. The graph of the chosen urban environment consists of waypoints that represent the nodes of the graph, the A* heuristic is the distance between two waypoints.

After configuring the urban environment, the deep neural network of the self-driving car system and A* high-level planner, we built the controller that controls the self-driving car system's vehicle in order to train it by an expert using imitation learning and data aggregation algorithm.

5.2 Dataset Preparation & Training the End-to-End Deep Neural Network

First, We have created the AI vehicles of the simulator in order to create the world in the chosen urban environment then, we have created a vehicle that represents our self-driving car agent after that, we have created only the RGB camera sensors in our self-driving car agent, then we have configured the camera and speed sensors settings such as field of view (FOV) to be the value of 110, the dimensions of the sensor images front left and right to be (200×200 ×3), the angle of each camera sensor, the angle of the front camera has been rotated 0°, the angle of the left camera has been rotated 300° and the angle of the right camera has been rotated 60°. After configuring the camera sensors settings. we start collecting the driving dataset by applying the imitation learning and dataset aggregation algorithm, we have collected it in the initial time a few samples of expert's driving, then by running the deep neural network model in the chosen urban environment, we have collected the corrections of the mistakes done by the deep network and training the deep network again on the initial dataset and the corrections of the mistakes done by the deep network, then running the deep network again in the chosen urban environment and collecting corrections of the mistakes done by the deep network such as driving out of the lane…etc. Then, training the deep network again on the initial dataset, old corrections of the mistakes and the new corrections of the mistakes…so on this is the imitation learning and data aggregation algorithm. We have nearly 300000 driving and corrections of the mistakes done by the deep network samples. We have trained the deep neural network of this self-driving car system about a month training and correcting mistakes…so on (data aggregation).

The deep neural network is trained using the stochastic gradient descent ADAM optimization algorithm which consists of the formulas:

$$m_0 := 0(\text{Initialize initial 1st moment vector})$$
$$v_0 := 0(\text{Initialize initial 2nd moment vector})$$
$$v_h at_0 := 0(\text{Initialize initial 2nd moment vector})$$
$$t := 0(\text{Initialize timestep})$$

Figure 5-1 ADAM Initialization

$$t := t + 1$$
$$lr_t := \text{learning\_rate} * \sqrt{1 - beta_2^t}/(1 - beta_1^t)$$
$$m_t := beta_1 * m_{t-1} + (1 - beta_1) * g$$
$$v_t := beta_2 * v_{t-1} + (1 - beta_2) * g * g$$
$$variable := variable - lr_t * m_t/(\sqrt{v_t} + \epsilon)$$

Figure 5-2 ADAM Optimization Algorithm

We have normalized the expert's dataset that is collected using imitation learning and data aggregation algorithm by $X = \frac{X}{255}$ where X is the 3 RGB image inputs front, left and right, we haven't normalized the speed measurement input as well as the A* command input.

Showing some of the samples below to illustrate the dataset

Outputs = {Steer Angle = 0.0, Acceleration = Normal}
Inputs = {Speed = 50 km/h, A* Command = STRAIGHT, Camera Images}

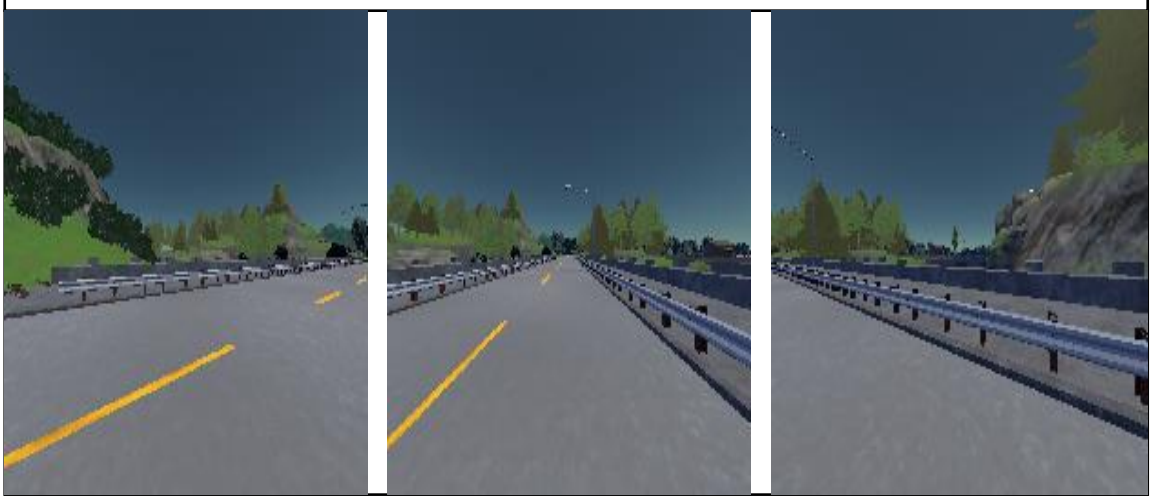

Figure 5-3 Sample 1

Outputs = {Steer Angle = 0.2145, Acceleration = Normal}
Inputs = {Speed = 24 km/h, A* Command = RIGHT, Camera Images}

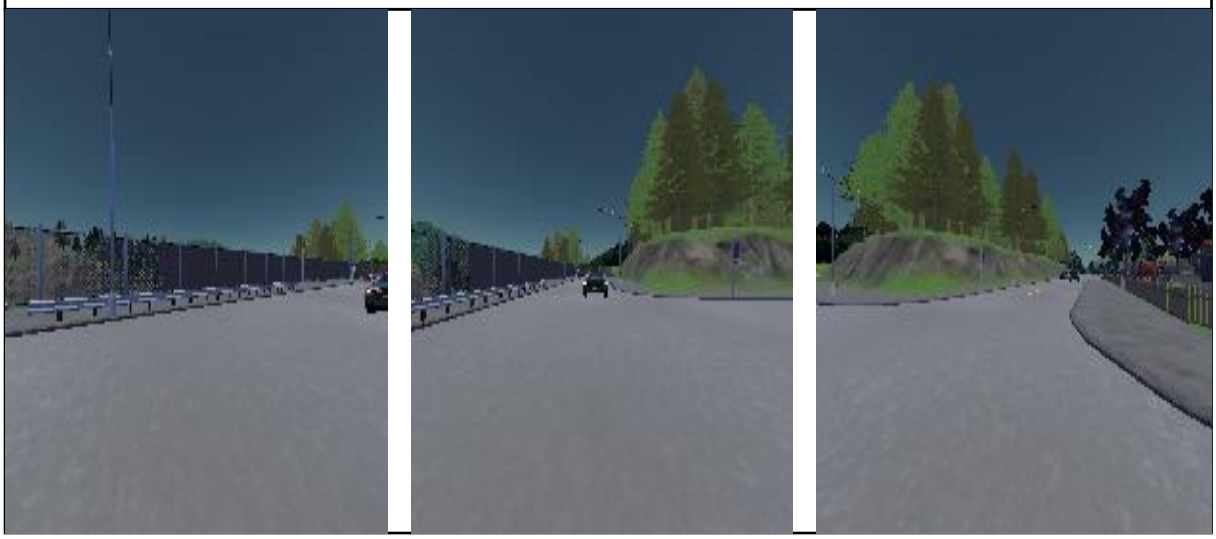

Figure 5-4 Sample 2

Outputs = {Steer Angle = 0.0, Acceleration = Brake}
Inputs = {Speed = 18 km/h, A* Command = STRAIGHT, Camera Images}

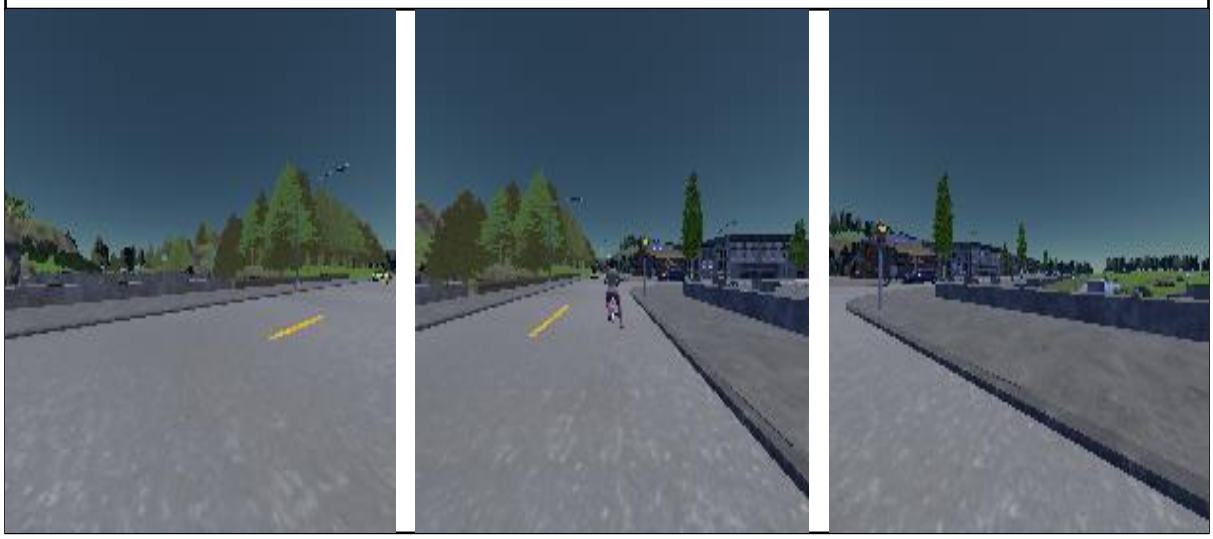

Figure 5-5 Sample 3

Outputs = {Steer Angle = 0.0, Acceleration = Accelerate}
Inputs = {Speed = 0 km/h, A* Command = RIGHT, Camera Images}

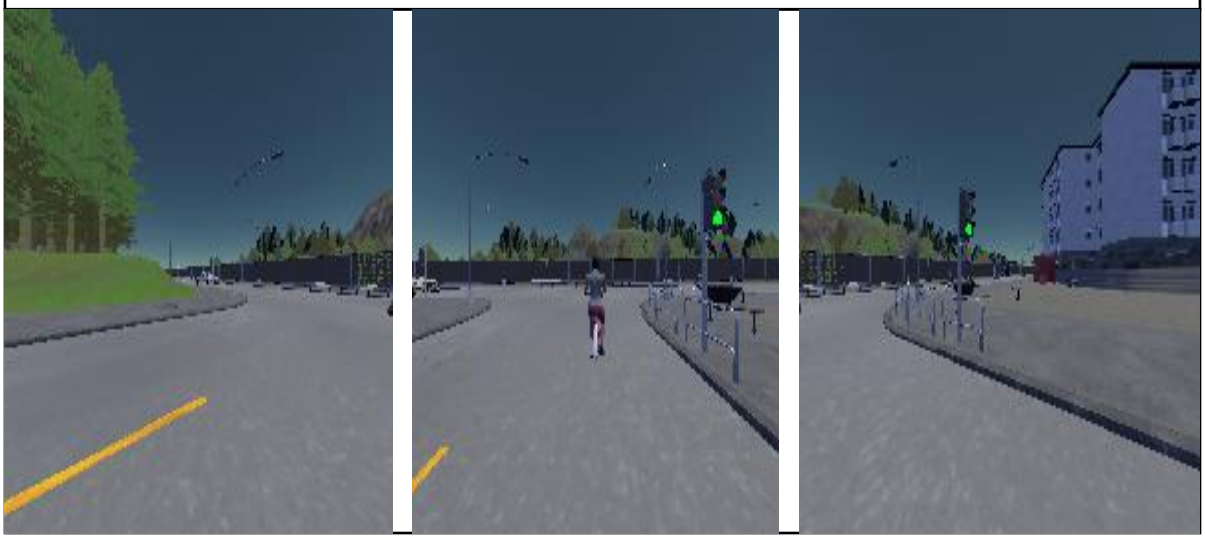

Figure 5-6 Sample 4

Outputs = {Steer Angle = - 0.376, Acceleration = Normal}
Inputs = {Speed = 21 km/h, A* Command = LEFT, Camera Images}

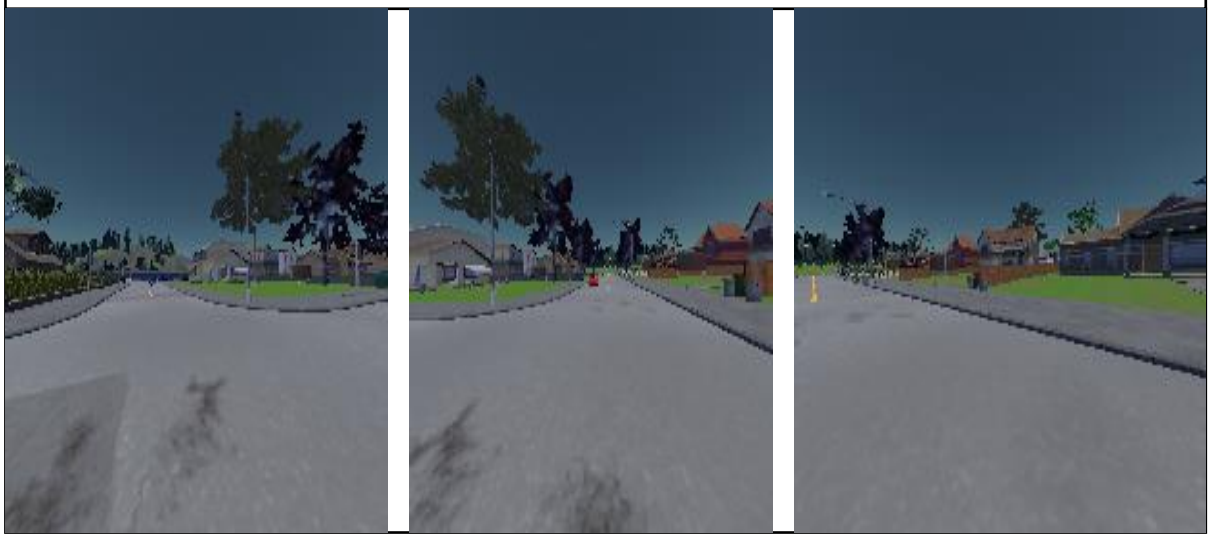

Figure 5-7 Sample 5

Outputs = {Steer Angle = - 0.3135, Acceleration = Normal}
Inputs = {Speed = 26 km/h, A* Command = STRAIGHT, Camera Images}

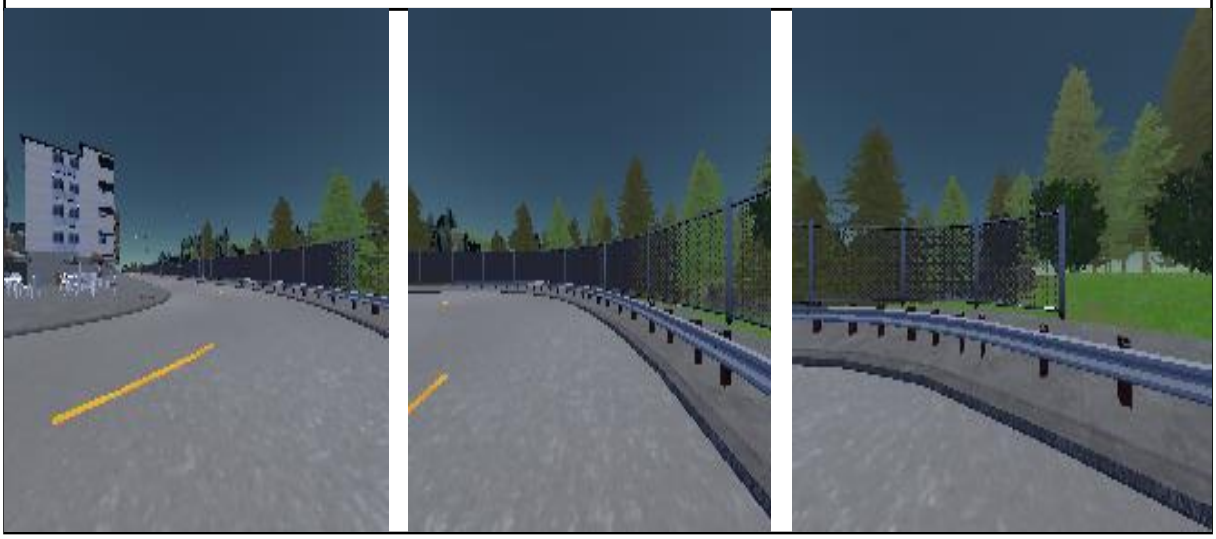

Figure 5-8 Sample 6

5.3 Performance Metrics

In the classical machine learning or deep learning problems we focus on the accuracy of a classifier model, in the self-driving task, no accuracy or loss specifies the real performance of the deep model of the self-driving system because the environment is dynamic. We have specified the real performance of the deep model of this self-driving-car system by specifying a 20 random pair of (start, target) points, then we have tested the self-driving-car system in Carla environment by running the self-driving-car system between each of these 20 pair (start, target) points, then we have counted how many times the system collisions with other cars out of all seen other cars, how many times the car agent did not stop at red traffic signs or did not move at green traffic signs at all out of all green and red traffic conditions, how many times the car agent did wrong steering at A* commands LEFT, RIGHT and STRAIGHT…etc. out of all seen commands. (Table 6-1) below illustrates these statistics that are the basic performance metrics of the self-driving-system.

When we compute the statistics, we calculate the accuracy of each one steering left at A* left command, right at A* right command…etc. then, we calculate the average accuracy

Table 6-1 Performance metric

#Shown Vehicles	Collisions	
	Yes	No
14	4	10
#Red Traffic Signs	Stop	
	Yes	No
20	20	0
#Green Traffic Signs	Move	
	Yes	No
21	18	3
Follow A* Commands		
#Command Left	Steering Left	
	Yes	No
11	10	1
#Command Right	Steering Right	
	Yes	No
3	3	0
#Command Straight	Steering Straight	
	Yes	No
3	3	0

6. Chapter Six
(Software Libraries)

This section provides some details about libraries and the simulator which used in this project.

6.1 Carla Simulator

Carla is an open-source urban environment built in Unreal engine, Carla is built for self-driving car research purpose, Carla simulates the physics of the real world in the simulation in order to build a powerful self-driving car agents that work well in the real world, Carla simulates the traffic movement of the real world as well as the traffic signs…etc.

Figure 6-1 Carla Urban Environment

Carla provides the commonly used sensors such as camera sensors (RGB, Depth Map…etc.), LIDAR, radar…etc.

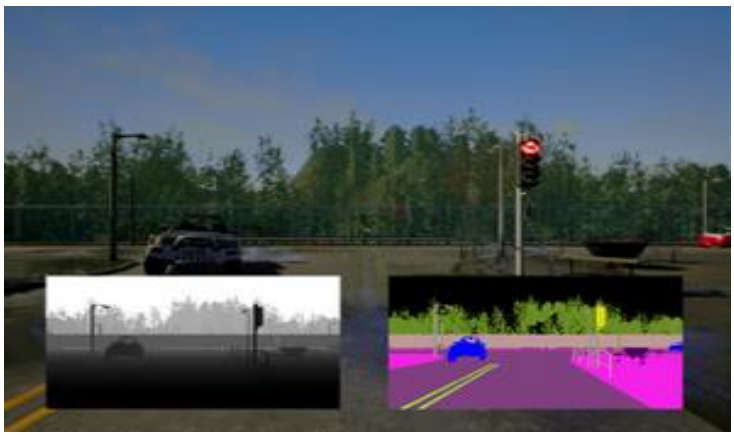

Figure 6-2 Camera Sensors (RGB, Depth Map, Segmentation Map).

Carla provides the most common parameters of the used camera sensor in the simulator such as field of view (FOV), camera resolution, frame per second (FPS), shutter speed…etc. The configurations of the used sensors in the self-driving system is crucial.

Carla provides the speed of a moving object in the simulator, acceleration of a moving object…etc. Carla provides the control over the number of AI vehicles and pedestrians in the simulator.

Figure 6-3 Carla World

6.1.1 Highlighted Features

- **Scalability Via a Server Multi-Client Architecture:** multiple clients in the same or in different nodes can control different actors.

- **Flexible API:** CARLA exposes a powerful API that allows users to control all aspects related to the simulation, including traffic generation, pedestrian behaviors, weathers, sensors, and much more.
- **Autonomous Driving Sensor Suite:** users can configure diverse sensor suites including LIDARs, multiple cameras, depth sensors and GPS among others.

- **Fast Simulation for Planning and Control:** this mode disables rendering to offer a fast execution of traffic simulation and road behaviors for which graphics are not required**.**

- **Maps Generation:** users can easily create their own maps following the Open Drive standard via tools like Roadrunner.

- **Traffic Scenarios Simulation:** our engine Scenario Runner allows users to define and execute different traffic situations based on modular behaviors.

- **ROS Integration:** CARLA is provided with integration with ROS via our ROS-bridge.

- **Autonomous Driving Baselines:** we provide Autonomous Driving baselines as runnable agents in CARLA, including an Auto Ware agent and a Conditional Imitation Learning agent.

6.2 TensorFlow

TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks.

It is a symbolic math library, and is also used for machine learning applications such as neural networks. It is used for both research and production at Google.

TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache License 2.0 on November 9, 2015.

6.3 OpenCv

OpenCV (Open source computer vision) is a library of programming functions mainly aimed at real-time computer vision.

Originally developed by Intel, it was later supported by Willow Garage then Itseez (which was later acquired by Intel). The library is cross-platform and free for use under the open-source BSD license.

OpenCV supports the deep learning frameworks TensorFlow, Torch/PyTorch and Caffe.

7. Chapter Seven
(Conclusion)

7.1 Conclusion

We built our self-driving car system that is based on end-to-end and multi-task deep neural network where the deep neural network is trained on a chosen environment in Carla simulator. The deep neural network of our self-driving car system is trained to drive correctly avoiding collisions, follow the traffic signs, follow the lane of a road and to be conditioned on A* algorithm where the A* gives a set of commands LEFT, RIGHT and STRAIGHT to the deep neural network in order to drive from point A to point B in the shortest path. The deep neural network is trained using imitation learning (behavioral cloning), data aggregation algorithm. The proposed deep neural network model is able to drive most of the time (85%) without human intervention.

7.2 Future Work

Self-driving car is not a problem, self-driving car is a topic. It is an open problem and has no final solution, so many solutions can be considered to try. A software company called Wayve[11] has proposed a solution that is different from the proposed solution of Tesla, the solution of Wayve is based on end-to-end deep learning and imitation learning where the solution of Tesla consists of 48 deep neural networks, each neural network has a specific task. Our proposed self-driving car system is an autonomous car of level 4 which means our proposed self-driving car system is able to drive only in a very similar environment to the environment that is trained on. In the future we plan to:

- Developing the data aggregation algorithm in order to enhance the expert's driving dataset, because the data aggregation is responsible of the quality of the data.
- Developing the proposed deep neural network in real world environments.

[11] https://wayve.ai

7.3 Difficulties

The limitations of this project were:

- The expert's driving dataset is collected by us.
- Dealing with the huge amount of dataset (Big Data), the dataset.
- Each epoch of training takes about 3 hours.
- No powerful resources such as GPU of type (Nvidia 1080 Titan) and solid-state drives (SSD), there were only a regular GPU of type (Nvidia 930 MX), CPU (core i7 7th generation HQ) and hard disk drives (HDD).

References

[1] K.J. Bussemaker – “Sensing requirements for an automated vehicle for high- way and rural environments” – MSc thesis – December 2014.

[2] Yunpeng Pan, Ching-A Cheng, Kamil Saigol, Keuntaek Lee, Xinyan Yan, Evangelos A. Theodorou, and Byron Boots – “Agile Autonomous Driving using End-to-End Deep Imitation Learning “– Institute for Robotics and Intelligent Machines, School of Electrical and Computer Engineering Georgia Institute of Technology, Atlanta, Georgia 30332–0250.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, – “Human-level control through deep reinforcement learning “–Nature, 518(7540):529–533 –2015.

[4] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang – “End to end learning for self-driving cars “– arXiv preprint arXiv:1604.07316, 2016.

[5] Dean A Pomerleau. Alvinn– “An autonomous land vehicle in a neural network. In Advances in Neural Information Processing Systems “– pages 305–313 –1989.

[6] Viktor Rausch, Andreas Hansen, Eugen Solowjow, Chang Liu, Edwin Kreuzer, and J. Karl Hedrick– “Learning a deep neural net policy for end-to-end control of autonomous vehicles “–In IEEE American Control Conference – 2017.

[7] Urs Muller, Jan Ben, Eric Cosatto, Beat Flepp, and Yann L Cun– “Off-road obstacle avoidance through endto- end learning. In Advances in Neural Information Processing Systems “– pages 739–746 –2006.

[8] C.K.I Williams and C.E. Rasmussen– “Gaussian processes for machine learning “–MIT Press –2006.

[9] Jelena Kocic´, Nenad Jovicˇic´ and Vujo Drndarevic´ – “An End-to-End Deep Neural Network for Autonomous Driving Designed for Embedded Automotive Platforms “–School of Electrical Engineering, University of Belgrade – May 2019.

[10] Zhengwei Bai, Baigen Cai, Wei ShangGuan, Linguo Chai – "Deep Learning Based Motion Planning for Autonomous Vehicle Using Spatiotemporal LSTM Network "–School of Electronic and Information Engineering Beijing Jiaotong University Beijing, School of Electronic and Information Engineering State Key Laboratory – March 2019.

[11] Felipe Codevilla, Matthias Mu¨ller, Antonio Lo´pez, Vladlen Koltun, Alexey Dosovitskiy – "End-to-end Driving via Conditional Imitation Learning "–TIN2017.

[12] D. Pomerleau. ALVINN – "An autonomous land vehicle in a neural network" – In NIPS[12] – December 1988.

[13] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba. – "End to end learning for self-driving cars "– arXiv:1604.07316, 2016.

[14] Y. LeCun, U. Muller, J. Ben, E. Cosatto, and B. Flepp – "Off-road obstacle avoidance through end-to-end learning "– In NIPS– December 2005.

[15] A. Dosovitskiy and V. Koltun– "Learning to act by predicting the future "– In ICLR[13], 2017.

[16] S. Levine and V. Koltun– "Guided policy search "– In ICML, 2013.

[17] S. Ross, G. J. Gordon, and J. A. Bagnell– "A reduction of imitation learning and structured prediction to no-regret online learning "– In AISTATS[14], 2011.

[18] M. Laskey, A. Dragan, J. Lee, K. Goldberg, and R. Fox. Dart– "Optimizing noise injection in imitation learning "–In CoRL[15], 2017.

[19] Joud Khattab – "Using Social Network Analysis to Understand and Analyze Sustainable Development Goals" – MSc thesis – December 2019.

[12] Conference and Workshop on Neural Information Processing Systems.
[13] The Conference on Learning Representations (ICLR).
[14] An interdisciplinary gathering of researchers at the intersection of computer science, artificial intelligence, machine learning, statistics, and related areas.
[15] Conference on Robot Learning

[20] Katie Burke – “How does a self-driving car see? “– Nvidia blogs – April 2019.

Notes

Notes