Contents

1 Introduction

The last decade has seen massive growth in the field of Autonomous Driving, primarily due to proliferation of graphical processing unit(GPU), and several projects like Google(Waymo) [1], Berkeley-DeepDrive [2], Apollo [3], making their datasets open-source which have made it easier for people to work on these data and achieve better performance gains.

Training a deep neural network(DNN) forms the core of making a car autonomous. By using supervised learning, one can achieve reliable results as it gives greater control at each stage of training. The data-driven approach collects data in advance and labels it appropriately. It can then be fed to the DNN using supervised learning algorithms to train the best model possible.

Ever since the discovery of Alexnet in 2012 [4], the convolutional neural network(CNN) and deep learning(DL) are preferred choices to analyse images. However, it is well known that the camera sensors are susceptible even to a slight change in weather conditions. Sensors like radar [5], LIDAR [6], ultrasonic[7], depth camera give additional depth information for obstacle detection. These values then are fused with the camera images to make data fusion possible.

Even though there are some public data available, it is still not enough to reliably train a DNN. Then there is the cost of building an autonomous car. Fortunately, the last years have seen growth in reliable simulators which helped massively to collect data to help explore this field of research. To name a few simulators that are being actively used – LGSVL [8], Nvidia Drive [9], Carla [10], CarMaker [11]. In this thesis, the LGSVL simulator is used.

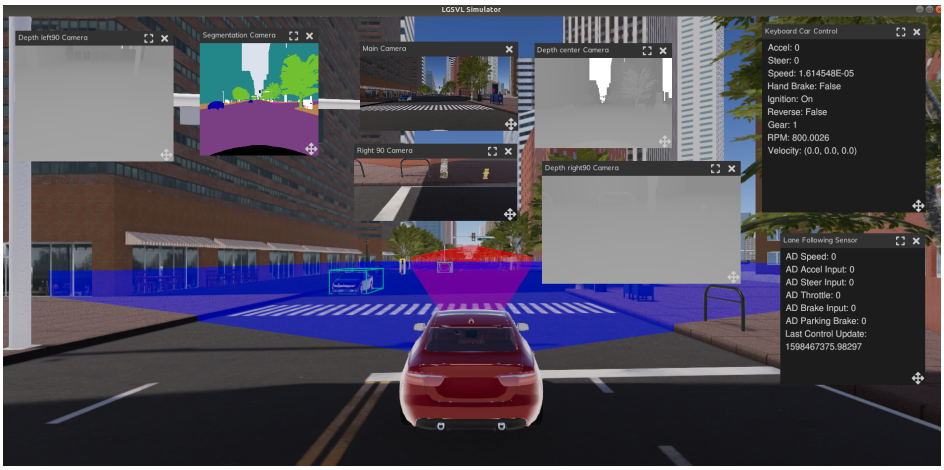

Figure 1.1: LGSVL[8] simulator active with all sensors

The LGSVL simulator allows the use of different sensors with minimal effort. The data from different sensors are published through websocket. So to capture these data, we need an interface/protocol which can understand the sent data's message type and enable the receiving node to store them. However, the data from each sensor arrives at different rates. Hence it is necessary to collect and synchronise them in the order of their arrival before storing, so to not lose their integrity and thereby

prevent corrupting the dataset. Robotic operating system(ROS) [12] and its functionalities fulfil this purpose. It allows seamless transfer of simulator's data by subscribing to sensor nodes in the form of topics. Then the subscribing node with the help of ROS synchronises it as necessary for storage.

So, now the data that resembles real-world is stored locally for later analysis and research.

1.1 Motivation

Though autonomous driving is one of the favourite research areas in mobility, a significant challenge is still the cost associated with integrating all the necessary sensors. Representing the environment around the vehicle(ego vehicle) requires information from all in-car sensors. The resources demanded to make an optimal decision are also a challenge. The motivation for this thesis is to use a simulator, do the required tests and determine whether using a simulator does indeed help in perceiving the environment and accomplish the goal of driving in the real-world.

As briefly mentioned, the high cost of associated sensors such as LIDAR [13], has put off many smaller research groups from implementing them in their work. By using a simulator, again at a low cost, we can conduct adequate tests on how different constellation of sensors work, how different modalities interact with each other and what impact these factors have on the overall performance of the DNN.

Finally, implement an end-to-end system which simulates real-world behaviour which can then be applied to future research and make it more robust.

1.2 Goal

The desired goals of this thesis are listed below:

- Building an autonomous driving framework -
  - ROS - use ROS2 to synchronise the data received from the simulator through a rosbrige, use functionalities such as slop and cache, to sort the data according to their received time in order not to scramble the information. During the evaluation, use the same functionalities to send command controls to the simulator.
  - Rosbridge - use a bridge transport protocol that connects the ros to the simulator.
  - Docker - set up a work environment that is independent of hardware or operating system which allows easy running of the commands for data collection and evaluation.
- Implement an end-to-end neural network architecture which learns to drive by predicting the steering commands from image pixels. Also, apply state of the art DL techniques to it.
- Implement a system that can efficiently collect and label data.
- Implement and analyse different constellation of sensors with different data fusion techniques.

1.3 Related Work

In 2012, Alexnet [4] used CNNs to do object classification which, then in Computer Vision became the dominated approach for classification. Both Chen *et al.* [14] and Bojarski *et al.* [15] extended [4]'s approach of using CNN and showed that in addition to classification, CNN can extract features

from images. Then they went on to demonstrate through an end-to-end network(which self-optimises itself based on its inputs), that steering angles can be predicted to keep the car in the lane of a road.

In a different field, but using CNN, Sergey Levine *et al.* [16] in 2016 corroborated that it was indeed possible to extract features with CNN and predict motor control actions in *object picking robots*.

Then, Xu *et al.* [17] in the same year with CNN-LSTM architecture showed that using the previous ego-motion events helped predict future ego-motion events. Using CNNs in an end-to-end architecture raised some questions on how it reached its decisions. So in 2017, both [18], [19] did visual analysis after the CNN layers to better understand the module's functionality. Vehicle control is more than just steering control. For smoother control, acceleration and braking are necessary besides steering. Both acceleration and deacceleration are dependent on the user's driving style, lane speed limit and traffic etc. Yand *et al.* [20] used CNN-LSTM architecture and provided the LSTM with feedback speed to determine the velocity of the ego vehicle.

Besides vehicle control, perceiving the environment is necessary for collision avoidance. The RGB colour camera sensors don't provide the depth information which is critical for collision avoidance. Hence, it is essential to fuse other sensors with diverse modalities with RGB to predict an optimal output. Liu *et al.* [21] provided rules in fusing data. They said that it was essential to pick out only vital information and discard other noisy data. They also described the techniques involved in data fusion – early/late fusion, hybrid fusion, model ensemble and joint training. Park *et al.* [22] gave us methods to enhance the features by using feature amplification or multiplicative fusion. Zhou *et al.* [23] detailed how fusing data into CNN affects the overall performance.

Even though the fused dataset gives a performance boost, it performs worse compared to individual modality. The combined fused model overfits more than its counterparts. The fundamental drawback of *gradient descent* in backpropagation causes the networks to overfit. This paper [24] introduced a technique called *gradient blending* to counteract this problem.

Xiao *et al.*[25] applied all the fusion techniques mentioned above with an imitation based end-to-end network[26]. They concluded that RGB images with depth information(obtained through a different modality) could indeed result in better performing end-to-end network model.

1.4 Contribution

2 Fundamentals

2.1 Machine learning: What and why?

Machine learning is all about learning from data; gaining knowledge from it. More the data, more the chances to learn. Machine learning was initially thought of as automating redundant human tasks and later developed into something that allowed solving complex mathematical problems. It was seen just an addition to humans than extension of them. Machine learning these days are required to perform tasks that are quite obvious and natural to humans such as recognising faces in images or perceive the road environment around the vehicle and make decisions instinctively.

All these attributes require to extend the field of machine learning.The figure 2.1 shows how artificial intelligence(AI) is divided into specific areas – Machine learning(ML) and deep learning(DL).

So, in this chapter, a brief overview is given on the concepts that are implemented in the later chapters.

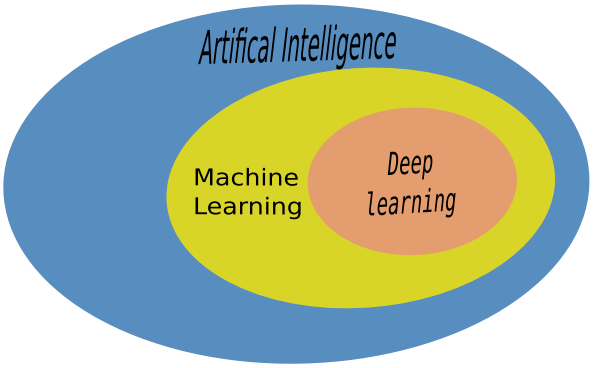

Figure 2.1: Schema of AI, ML and DL

2.1.1 Learning algorithms

Machine learning provides a means to tackle tasks that are complex to solve through fixed programmes and designed by human beings [27]. A learning algorithm is an algorithm which gains the ability to learn from data. A ML algorithm is one that gains the ability to learn from an experience E with respect to some class of tasks T and performance measure P [28]. With experience, the algorithm can improve its performance.

Tasks, T

The two major tasks in ML are *classification* and *regression*.

In classification related tasks, the system identifies which of k categories an input belongs to. A function $f : \mathbb{R}^n \rightarrow \{1, \ldots, k\}$ is used by the learning algorithm to solve this task. When $y = f(x)$, the model assigns an input described by vector $\mathbf{x}$ to a category identified by numeric code y. There

are other variants of the classification task, for example, where f outputs a probability distribution over classes [29]. Alexnet [4] is one of the examples of classification task that used it to do object recognition.

Regression is similar to classification except that the output is a continuous value.

A function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ predicts a numerical value for some input. Predicting the steering control value is a prime example for a regression task.

There are of course other tasks but only classification and regression are used in this thesis. Hence the narrow focus.

Performance measure, P

To evaluate the performance of a ML algorithm, it is a must to design quantitative measure of its performance. Usually this performance measure P is specific to a task T. There are two distinct types of measurements – accuracy and error rate.

If the goal is to learn a mapping from inputs x to outputs y, where $y \in \{1, \ldots, C\}$, with C being the number of classes. If $C = 2$, this is called binary classification (in which case we often assume $y \in \{0, 1\}$); if $C > 2$, this is called multi class classification. If the class labels are not mutually exclusive (e.g., somebody may be classified as tall and strong), we call it multi-label classification [30].

Accuracy is a proportion of how much the model produces the correct outputs. So in the case of binary classification, if the function f predicts a probability densities $\hat{y} \in \{0.3, 0.7\}$, for a ground truth y of value 1, then P is 70% accurate or the error rate is 30%.

It is essential that the model is evaluated with a data that it has not seen before. This data *testing set*, gives a good judgement on the performance of the trained model.

Experience, E

The ML algorithms can be classified into *supervised*, *unsupervised* and *reinforcement* learning based on the kind of experience they are allowed to have. A learning algorithm is allowed to gain experience by going through the *dataset*. A dataset is collection of all the examples for a given task. For example, to classify which category a shown image belongs to has collection of images as dataset [31]. Sometimes datasets are also called as *data points*.

The focus will be on supervised learning in our case. A random vector $\mathbf{x}$ explicitly attempts to learn the probability distribution $p(\mathbf{x})$ and predicts $\mathbf{y}$ from $\mathbf{x}$, usually estimating $p(\mathbf{y} \mid \mathbf{x})$. The CIFAR dataset [31], for example, contain images as features which inturn have *targets* or *labels* associated with it. Here supervised learning(SL), the target functionality(labels) is known. So it uses the images and predicts the probability distribution to classify the images in the corresponding label.

2.2 Deep Learning

Deep learning is a subset of machine learning. It takes all the algorithms, concepts from the machine learning, and narrows the focus to enable a model learn from data such that tasks involve less human involvement, huge amount of data, and parameters.

2.2.1 Simple neural network

Linear regression is one of the common SL algorithms. It solves the regression problem. For example, if there is vector $\mathbf{x} \in \mathbb{R}^n$ as input and predict a scalar value $y \in \mathbb{R}$ as its output, then in linear regression, output is a linear function of the input. We can define it as

$$\hat{y} = \mathbf{w}^T \mathbf{x} \tag{2.1}$$

where $\mathbf{w} \in \mathbb{R}^n$ is a vector of parameters.

$\mathbf{w}$ is usually referred to as a set of weights that determine how each feature affects the prediction. A $\mathbf{w}_i$ is simply multiplied with a feature x_i to predict $\hat{y}$. By manipulating the $\mathbf{w}_i$ value, the corresponding feature x_i has an effect on the prediction $\hat{y}$.

A learning algorithm, in this case linear regression, is implemented as a perceptron. It is a single-layer neural network as first suggested by Rosenblatt in 1958. They generally consists of four main parts – input nodes x_i, weights w_i, bias b_0(if necessary), net sum Σ and an activation function σ. This is shown in the figure 2.2.

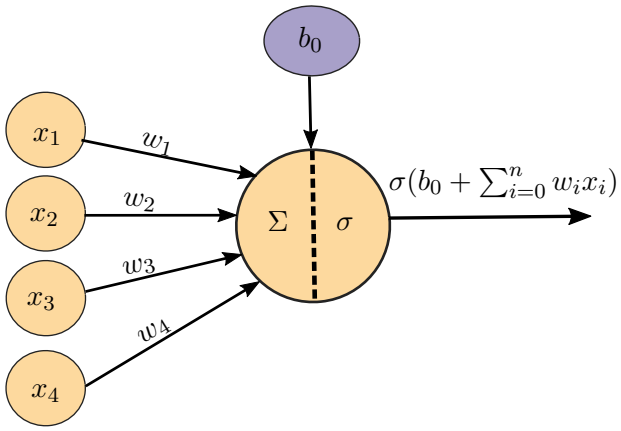

Figure 2.2: A simple neutral network

2.2.2 Activation function

The common activation functions used are Rectified Linear unit(ReLu), Sigmoid, tanh and softmax function. For each type of activation, σ then decides if the input received is relevant or not relevant. To convert linear inputs to non-linear, all that has to be done is to use a non-linear activation function. The figure 2.3, shows the characteristics of some of the activation functions.

For classification tasks, usually the last layer of the networks is equipped with softmax activation layer. This function normalises the output to a probability distribution over predicted output classes.

2.2.3 Multilayer feedforward networks

Deep feedforward networks or multilayer perceptrons are the quintessential deep learning models. Its goal is to approximate function f^*. In the below figure 2.4, information flows from inputs $\mathbf{x}$ to output y using a mapping function $\mathbf{y} = f(\mathbf{x}; \theta)$ where θ are the parameters values which the MLP learns for optimal approximation.

They are called feedforward as there are no feedback connections in which outputs of the model are fed back into itself. Feedforwards networks with feedbacks are called *recurrent neutral networks*.

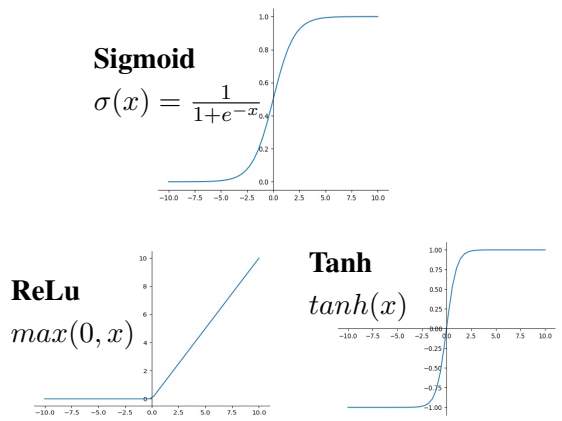

Figure 2.3: Activation functions

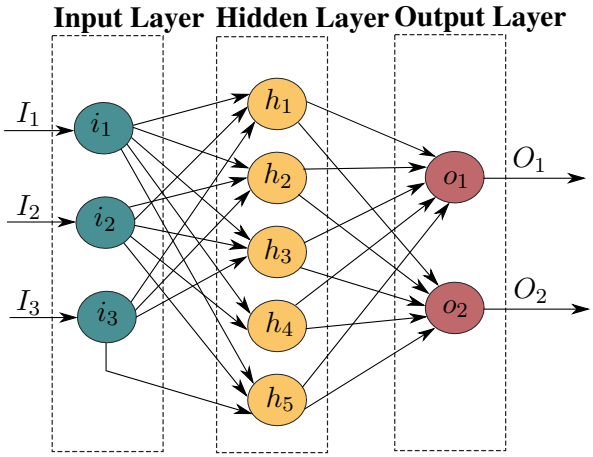

Figure 2.4: Multi layer perceptrons

Feedforwards networks form the core for many commercial applications. For example, the convolutional neural networks used for object detection are a special kind of feedforward networks.

More the hidden layers, more the depth of the feedforward networks. The width is given by the dimensionality of the hidden layer.

2.2.4 Loss function

As mentioned before, a mapping function f noisily approximates the input x to output y. So, the noise or the deviation from the true value(ground truth) must be kept at minimum. The function that calculates the deviation is called *cost* or *loss* function. It is important to choose the right loss function for a model.

For multi-label classification tasks, *categorical cross-entropy* function is used. For each category, cross-entropy is calculated. The difference between the cross-entropy of training data and the model's predictions is the cost function.

$$CCE = -\frac{1}{N}\sum_{i=1}^{N}[\hat{y_i}\log(y_i) + (1-\hat{y_i})\log(1-y_i) \tag{2.2}$$

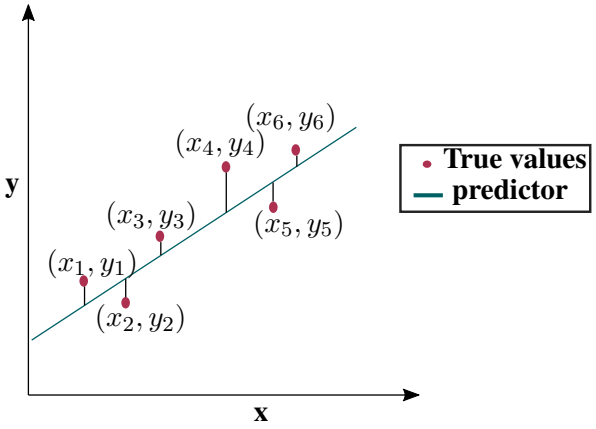

Figure 2.5: Mapping from x to y. The predictor is shown as linear line. The distance between the true values and predictor gives the loss. The sum of all the distances gives the loss function.

For regression tasks, the models are subjected to loss functions such as *mean absolute error*(MAE), *mean squared error*(MSE) and *mean squared logarithmic error*(MSLE). In MAE, the mean of absolute differences among predictions and expected results are calculated.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2.3}$$

In MSE, the mean of squared differences among predictions and true outputs are calculated.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.4}$$

In MSLE, the mean of relative distances between predictions and true outputs are calculated.

$$MSLE = \frac{1}{n}\sum_{i=1}^{n}(log(y_i + 1) - log(\hat{y}_i + 1))^2 \tag{2.5}$$

2.2.5 Gradient descent

Gradient descent is another technique to minimise the cost function parameterised by a model parameter $\mathbf{w}$. The first derivative(or gradient) gives the slope of the cost function. Hence, to minimise it, direction opposite to the gradient is chosen.

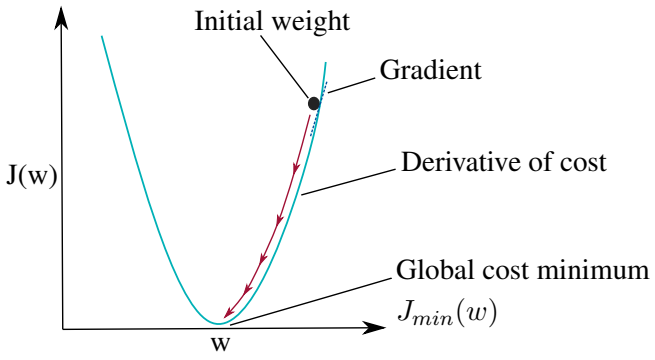

Figure 2.6: Finding the stochastic gradient descent

The rate at the which the gradient step reduces is given by the *learning rate*. It is one of the important parameters in training a model. It is also easily controlled by the user. If the learning rate is high, greater the step size of each gradient; possibly causing the step to miss the global minima. Lower the learning rate, more steps or training cycles needed to reach the global minima. Greater care must be taken in choosing the learning rate when training a model.

2.2.6 Backpropagation

Backpropagation are a class of algorithms which help in training feedforward neural networks for supervised learning. A model is said to fit when the gradient computation of the loss function is efficient w.r.t the weights of single input-output in the network. Backpropagation performs the effective gradient computation using the loss functions explained above.

2.2.7 Optimizer

The loss function explains how far the predictions are compared to the true outputs in a mathematical way. During training process, certain parameters can be tweaked to help the loss function predict correct and optimised results. However, there are question such as how to change them, by how much and when?

This is exactly optmizer's function. As explained in 2.2.5, gradient descent and learning rate form the core of optimizer's functionality. *Stocastic gradient descent*(SGD) is one of the oldest techniques in which gradients for all of training examples are calculated on every pass. Hence, they are slow and require much computation power. Some of the other popular optimizers are Adam [32], Adagrad [33], RMSprop [1]. In this work, Adam is used. Adam stands for adaptive moment estimation. It is a combination of all the advantages of two other extensions of SGD – Adagrad and RMSprop. Adam is computationally efficient, straight forward to implement, invariant to diagonal rescale of the gradients, less effort need to hyperparameters tuning.

2.2.8 Challenges in Machine learning algorithms

1. insufficient labelled data
2. poor quality data and irrelevant features
3. overfitting/underfitting a model

The first two issues can be solved if the user is careful during data collection and does preprocessing before feeding the data into the training model. However, if the training or the test data is too small, the model is subject to underfitting or overfitting. Though our aim is to reduce the error in the training set, we also need to reduce the error in the test set. The gap between training and testing error is also important parameter. Underfitting occurs when the model is not able to obtain sufficiently low error value for the training set. And if the gap between training and testing error is too large, overfitting happens. The sweet spot is to stop training the model when the gap between the two sets is at a minimum value. Left of the optimal point, the model underfits. Right of it, the model overfits. The below figure 2.7 shows it very well. Validation error is the error calculated for the test set.

[1] RMSprop is an unpublished, adaptive learning rate method proposed by Geoff Hinton in Lecture 6e of his Coursera Class [34]

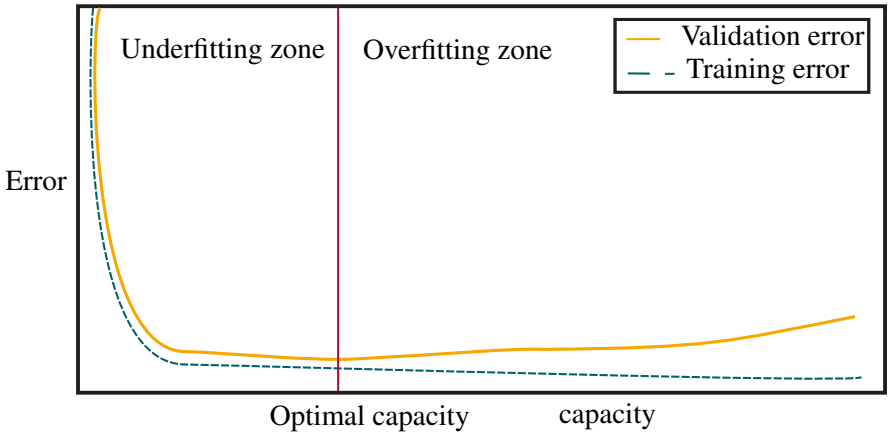

Figure 2.7: Relationship between capacity and error. Inspired from [27]

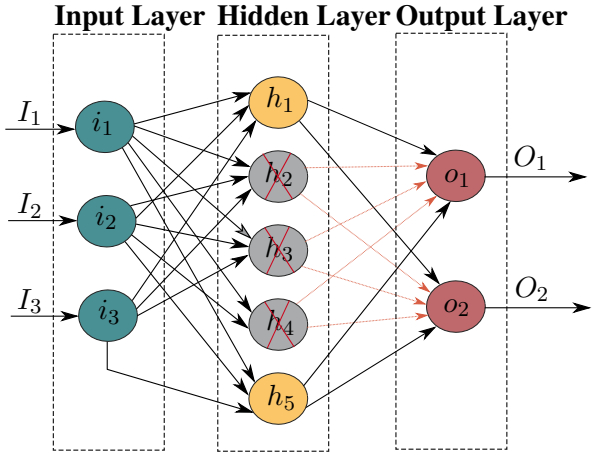

Figure 2.8: Illustrating dropout functionality

2.2.9 Dropout

DNNs contain multiple non-linear hidden layers and which makes them easily learn complex relationships between their inputs and outputs. With a small training set, this relationship adds sampling noise that won't exist in the real-world data even if drawn from the same distribution. This leads to overfitting and several methods have been developed to reduce its effect.

1. early stopping as soon as the validation error gets worse than the training error.

2. L1 and L2 regularisation which penalises the weights [35].

3. Randomly drop units(along with their connection) from the neutral network during training [36]. Figure 2.8 illustrates how to do the random dropping of units.

2.2.10 Convolution Neutral Network - CNN

Convolutional neural network(CNN) or in short Convnets is a deep learning algorithm for object recognition tasks. Alexnet [4] was one of the first models to demonstrate the power of CNNs in object classification. This was then used in various areas of research, where images were primary inputs, to achieve remarkable extensions. What makes CNN stand out for image analysis? The network takes images as inputs, reduces them into a form easier to process, without losing features which are critical

for a good prediction. So not only this is important to consider while designing an architecture but also while scaling massive dataset.

Convolution Layer

The images which are taken as inputs are just n-dimensional matrix with pixel values. So a convolution operation can be easily carried on it using a filter or *kernel*. A kernel matrix is pre-defined according to the task. Usually the size of the kernel is tiny compared to images' which facilitates easy convolution. A *stride* is the value of a step taken by the kernel after each convolution. If stride = 1, then it is called *non-strided*. Convolution remarkably extracts the high-level features such as edges. Normally there are many Convnets in an architecture. Each layer extracts a different feature or expands on last layer's task.

If the dimensionality of the convolved feature stays the same or increased compared to the input, then it is called *same padding*. If the dimensionality is reduced, *valid padding*. Padding is extremely useful for solving boundary conditions.

Pooling Layer

This layer is similar to convolutional layer. Its task is to decrease the computational power required to process data, usually done through reducing the dimensionality. It is, furthermore, useful to extract dominant features that are rotational and positional invariant, thus maintaining the goal of training the model.

There are two types of pooling – *max pooling* and *average pooling*. Max Pooling returns the maximum value from the portion of the image covered by the Kernel. On the other hand, Average Pooling returns the average of all the values from the portion of the image covered by the Kernel. Pooling also helps in reducing the noisy pixels which sometimes skew feature extraction.

Flatten and Fully Connected Layer

The main goal for extracting features from the images is to do some task; for example - classification. So the extracted features must be converted into a form understandable for the MLP (2.2.3), which happens to be 1-dimensional vector. This is exactly the task of *flatten layer*.

MLP gets a vector as input and feeds it to a feedforward network in *fully connected layer*. Fully connected layer then outputs the necessary values depending on the task.

It is important to remember that each layer employs an activation function(2.2.2) to introduce linearity or non-linearity to the inputs.

2.2.11 Recurrent neural networks - RNN

One of the drawbacks of neural networks is that they always start from scratch;with no memory of the previous state. If a neural network has to be used for word prediction, knowledge of previous letter and word is necessary. Recurrent neural networks addresses this issue.

RNN provide the temporal dynamic behaviour. A typical RNN looks like in the figure 2.9. The left hand side shows it folded and right hand side unfolded in time. RNN, however, suffers from *long term dependencies*. This is well explored in [37] and [38].

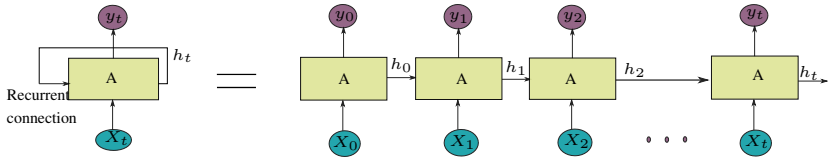

Figure 2.9: A Simple RNN

2.2.12 LSTM

The shortcomings of RNN are overcome by LSTM - *Long Short Term Memory*. They are a special kind of RNN which was first introduced by [39]. They remember information for long periods as their default behaviour with ease. The figure 2.10 shows how the structure of a LSTM differs from simple RNN. The LSTM employs gates and activation functions to add or delete information from the previous state.

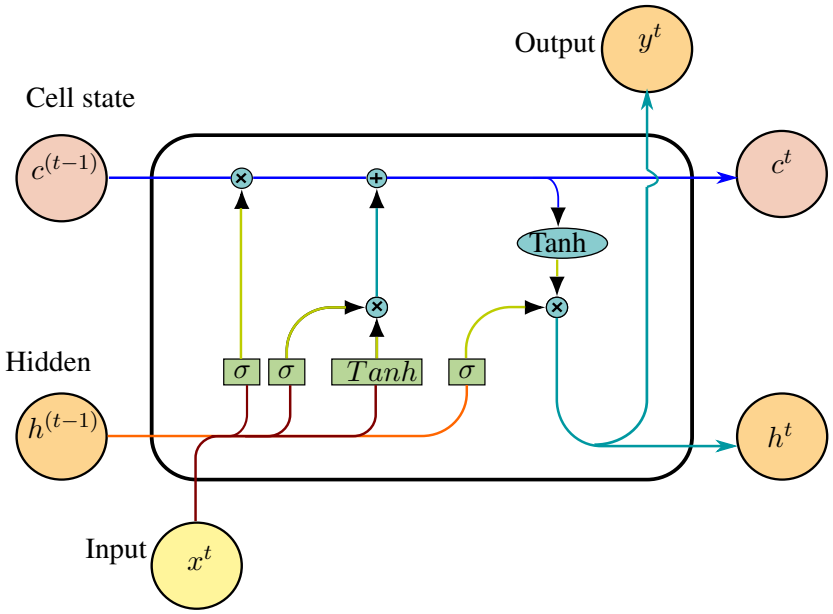

Figure 2.10: LSTM Architecture

2.3 Sensors

Deep neural networks need data – images or measurements to perform necessary tasks. These information/data are captured using sensors.

2.3.1 Visual Sensors

Visual sensors are one of the commonly used sensors for image capture of the environment. Usually cameras are used.

RGB Colour Camera

A camera in Red-Green-Blue colour spectrum is used to capture images. These images are then processed into different spectrum range to do feature analysis.

Depth Camera

A Depth camera returns an image where the shades on the grey-scale correspond to the depth of objects.

Segmentation Camera

This sensor returns an image where objects are coloured corresponding to user-defined tag. For example, a car is blue, pedestrian red, road boundaries white etc.

2.3.2 Measurement Sensors

These sensors are required for provided information other than visual such as telemetry data.

Radar Sensor

A common sensor in weather forecast which is known for its long range ability and resistant to adverse weather conditions. With radar, relative information of the environment are collected such as distance between vehicles, relative positions of them etc.

Control Sensor

With this sensor, telemetry information can be collected.

2.4 Sensor/Data Fusion

To allow DNNs make the best perception of the environment, it is necessary to fuse data from several sensors and feed that combined data into the DNN. This technique of fusing information exists for decades [40]. Often used data fusion technique is *Kalman filtering* and its variants. Some of the notable papers using that technique are [41], [42].

For autonomous driving, RGB and depth information(RGB-D) is vital for obstacle avoidance. [25] uses data fusion to get better results for their experiment.

2.4.1 Types of Data Fusion

There are two traditional approaches to data fusion – *early fusion* and *late fusion*. In early fusion, all the sensor inputs are concatenated before being fed to the CNN. Whereas in late fusion, each sensor inputs are fed to separate convolutional layer and down the line, they are concatenated together.

These techniques can be seen in action in this [24] recently published paper from Facebook research team.

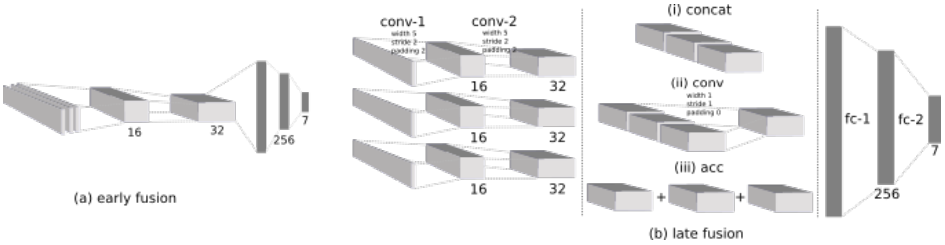

Figure 2.11: This figure is taken from this [43] paper where they describe early and late fusion architectures and also present three types of late fusion.

2.5 Machine Learning Library

To train the neural networks, we need a ML library framework to programme it. *Tensorflow* [44], *Keras* [45], *Pytorch* [46] are some of the popular ML frameworks used today. For this thesis, we use Keras, a python high-level wrapper for tensorflow. With keras, one can easily design DNN architectures with minimal effort. All the DL techniques we discussed above are reduced to a bunch of human understandable commands. Keras has several application programming interfaces(APIs) – Models, Layers and Callbacks.

2.5.1 Models API

They are of two types – Sequential and Functional. Sequential is just stack of layers with one input and one output. Functional can handle models with non-linear topology, shared layers, and multiple inputs or outputs. Since functional API offers flexibility, it is used here. In addition to offering the overall functionality, this API has the power to implement optimizers (2.2.7), loss functions (2.2.4).

2.5.2 Layers API

The layers needed for CNN – convolution (2.2.10), pooling (2.2.10), normalization, regularization, activation (2.2.2) and time series operations with LSTMs (2.2.12) are easily implemented with this API.

2.5.3 Callbacks API

With this API, some of the overfitting challenges can be automatically avoided. Some functionalities available are early stopping (1) and ModelCheckpoint (2.2.8).

Early stopping sets an epoch parameter *n*. If the gap between training and validation loss don't improve/reduce for the next *n* defined epochs, the training is automatically stopped.

With ModelCheckpoint, the gap is monitored w.r.t a monitoring parameter;usually minimum validation loss or maximum validation accuracy. Then automatically the best model gets saved.

In order to visualise the performance of the training, *TensorBoard* class is used.

2.6 Robotic Operating System - ROS

The Robotic Operating System(ROS) [47] is a set of software libraries and tools created to help developers build robot applications. From drivers to state-of-the-art algorithms, and with powerful

developer tools, ROS is a necessary set of tools for any robotics project. And its all open source.

ROS environment was first developed by Willow Garage for the PR2 robot [48]. PR2 is a humanoid robot that can navigate autonomously in a known environment. Since then, ROS is now used in all kinds of robots in various fields. With its popularity, many companies manufacture ROS compatible robots. This massively helps integrating multiple components to communicate with each other.

ROS(ROS1), since its launch, was considered as a middleware/interface between components. There was a parent to which all the children were connected. Every child node had to go through the parent every time to discover another node. In today's, expanding robotics market, this approach is outdated. This led to the development of ROS2 [49].

2.6.1 ROS2

ROS2 uses a data distribution service(DDS) for publishing and subscribing instead of custom message handler. With DDS, the transmission performance is also improved. Each node is *peer-to-peer* and can contact other nodes efficiently. ROS2 is simply not an extension of ROS1; although some of the functionalities have been ported.

2.6.2 ROS2 concepts

In this section we will study different concepts used in the thesis.

Nodes

A node is an entity that uses ROS protocol to communicate with each other. In a ROS graph, there are networks of nodes and connections between them.

Messages

Messages are ROS data type that are used when subscribing or publishing to a topic.

Topics

A topic is named information *bus* over which nodes exchange messages. A topic usually begins with *"/"* followed by the topic name. For example, "/radar" is topic associated with radar bus. Each topic carries information of a particular message type. This message type can either be a standard or custom type.

Subscriber and Publisher

If a node subscribes to a topic, then the node is called a *subscriber*. If it publishes to a topic, then it is a *publisher*.

Both publisher and subscriber when they are initialised over a topic, a *queue size* is defined. Depending on the queue size, a topic's messages can be queued and processed as needed. The figure 2.12 shows a subscriber and publisher node exchange data with each other.

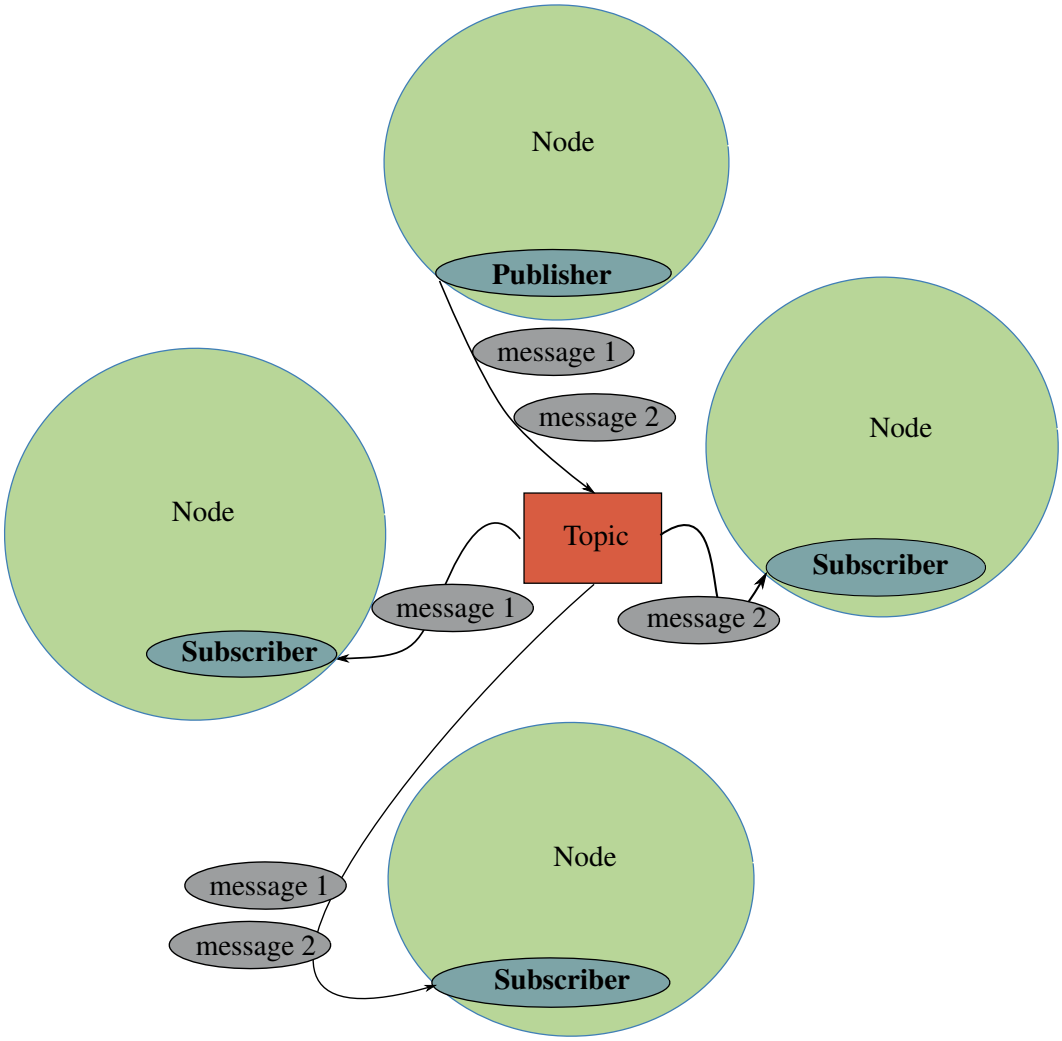

Figure 2.12: A graph showing how a publisher or subscriber node interact and exchange messages with each other through a topic.

Spins and Callbacks

In computer programming, spinning is a technique in which a process repeatedly checks to see if the condition is true. In ROS, a node is set to spin with no or a condition. This enables it to do its tasks as programmed.

Also from computer programming, callbacks is a function that executes at a given time. There are two types of callbacks – *blocking* and *deferred*. In ROS, deferred callback is used. It means that the callback function is invoked after a node returns something. It can be a subscriber receiving a message from its subscribed topic.

Rosbridge

We are aware that there are some non-ROS robots which would need to communicate with ROS ones. So a rosbridge [50] acts as communication API between these two. The rosbridge specification is programming language and transport agnostic.

Message Filters

Since the main goal of this thesis is to do data fusion, we need to use ROS to communicate with different sensor nodes. These sensor nodes receive and transmit data at different rates. So we need a filter that can trap the received or transmitted messages, serialize them(so as to not lose data's integrity) and make it possible for storage.

With the help of a filter, all the nodes can be made to wait till every node receives the message and then invoke the callback function only once or multiple times as per design. Inside the callback, further operations can be carried out before saving.

Message_filter [51] is one such filter. It has the functionalities we are looking for, such as TimeSynchronizer, cache(a buffer to store messages while waiting for others), and slop(an extra delay parameter to TimeSynchronizer modules which defines the delay(seconds) with which the incoming messages can be synchronized.) Caution must be kept when choosing the slop value. Otherwise, the data will lose its integrity.

In the next chapter, we will see how the LGSVL [8] simulator is used.

2.7 Docker

Docker [52] is an open-source tool designed to make it easier to create, deploy, and run applications by using containers. These software containers allow developers to package an application with all the parts it needs, such as libraries and other dependencies, and deploy it as image based packages. By doing so, developers can be sure that their application will run smoothly irrespective of the client's environment. This also allows for easy debugging and development.

Docker can also loosely considered as virtual machine(VM). But unlike a VM, rather than using a whole operating system, a docker shares the kernel of the system and shipping only the applications that are not in the host machine. Also a docker container is independent of host machine's applications. This greatly improves performance.

However, docker has its drawbacks. For example, building a docker image takes sometimes long to compile and consume a lot of resources. So not everyone can build an image and regularly as they wish.

In this thesis, docker containers are used for data collection from the simulator and later for evaluation of the model with the simulator.

2.8 Notes:

1. draw cnn
2. if possible draw for fusion techniques.

3 Simulation and Simulator

3.1 Why is simulation needed?

One of the important questions to ask before explaining about simulation is to understand why should one need a simulator to do simulation. As explained in the previous chapter (2), deep neural network(DNN) using supervised learning algorithm needs huge amount of labelled data. Since the cost of collecting that amount of data in real road vehicle is too expensive, researchers have sought the help of simulators. A simulator is an application which simulates a real-world environment, virtually. With the help of a simulator, one can collect any amount of data they wish for their project.

3.2 What is in a simulator?

Data collection is one of the most important phase in supervised learning. So caution must be taken in choosing a simulator. A simulator must fulfil certain conditions to be qualified as a good one.

- It must have an vehicle that can move around in a virtual map.
- The vehicle must be equipped with appropriate sensors for perceiving the environment properly.
- The virtual map must try to mirror the real-world to an extent. That mean it should have proper terrain to drive around, lane marking for lane detection, other cars to mirror the real world traffic, pedestrians, and real world weather conditions.
- It must provide a medium to collect data and allow interfaces to transfer the data. It should also be able to receive data in case the user needs to validate the data collected.
- Finally and most importantly, support end-to-end, full stack simulation.

3.3 LG SVL simulator

A simulator chosen for this thesis is from LG research centre in Silicon Valley, California called LGSVL simulator. It is an open source project where the code is regularly published at Github [53]. This simulator satisfies all the conditions listed above. They provide an out-of-the-box solution which can meet the needs of developers wishing to focus on testing their autonomous vehicle algorithms. It also supports Apollo [54] and Autoware [55].

3.3.1 How is LGSVL simulator developed?

LGSVL simulator's core simulation engine is developed using the Unity game engine [56]. Unity game engine is written in C# programming language. Since a game engine inherently supports animation, the simulator is able to extend that functionality easily. In addition to Unity, also supports

several libraries necessary to compute complex mathematical operations. With Unity's latest High Definition Render Pipeline(HDRP, LGSVL is able to simulate photo-realistic virtual environments that match the real world.

3.3.2 Overview of LG SVL simulator

User AD Stack

It supports User autonomous driving(AD) stack. That means a user can develop, test and verify through simulation. The user AD stack connects to LGSVL Simulator through a communication bridge interface; a bridge is selected based on the user AD stacks runtime framework. This bridge interface can use a standard protocol such ROS, ROS2 or custom one like CyberRT [54].

In addition LGSVL supports plug-in component which a user can develop and attach it to the simulator. The simulator during runtime picks up this plug-in.

Simulation Engine

As mentioned above, LGSVL uses Unity's latest HDRP game engine.

Sensor and vehicle models

It supports sensor arrangement and importantly they are customisable. The sensors are added and removed through JSON formatted text along with its parameters. These parameters include sensor type, position of the sensor, topic name, publishing rate, and in some sensors reference frame of measurement. Some of the popular sensors like camera sensors, radar and LIDAR are supported. In addition, users can add their own custom sensors as plug-in. Fig.3.1 gives a good overview of some of the sensors in action.

Vehicles provide a medium to travel the environment. Hence, vehicle dynamics is also important.

Environment and maps

An environment, in this case, virtual, is a primary component in autonomous driving simulation to provide many input to AD system. An environment affects almost all the functionalities in a AD system such as perception, prediction and tracking modules. It also affects the vehicle dynamics which is the key factor in vehicle control mechanism. Through changes in the HD map, the environment affects localization and planning modules. Finally, weather conditions such as rain, fog, night driving naturally affect the environment. So caution must be taken while design the environment.

LGSVL supports creating, editing and exporting HD maps of existing 3d virtual environment. 3D environment also defines the rules how agents must behave such as stopping at traffic lights, giving way to priority traffic, respect lane boundaries etc.

As of writing, LGSVL supports virtual Sanfranciso city HD map. They also support smaller maps like Shalun and Cubetown.

Test scenarios

Test scenarios enable users to test their AD stack by simulating in an environment and comparing and contrasting correct and expect behaviours. A lot of variables like HD maps, traffic movement behaviour and their density, time of the day, weather condtions etc. also play a role while testing. It is also possible to write scripts with the help of Python API, scenarios can be created and tested.

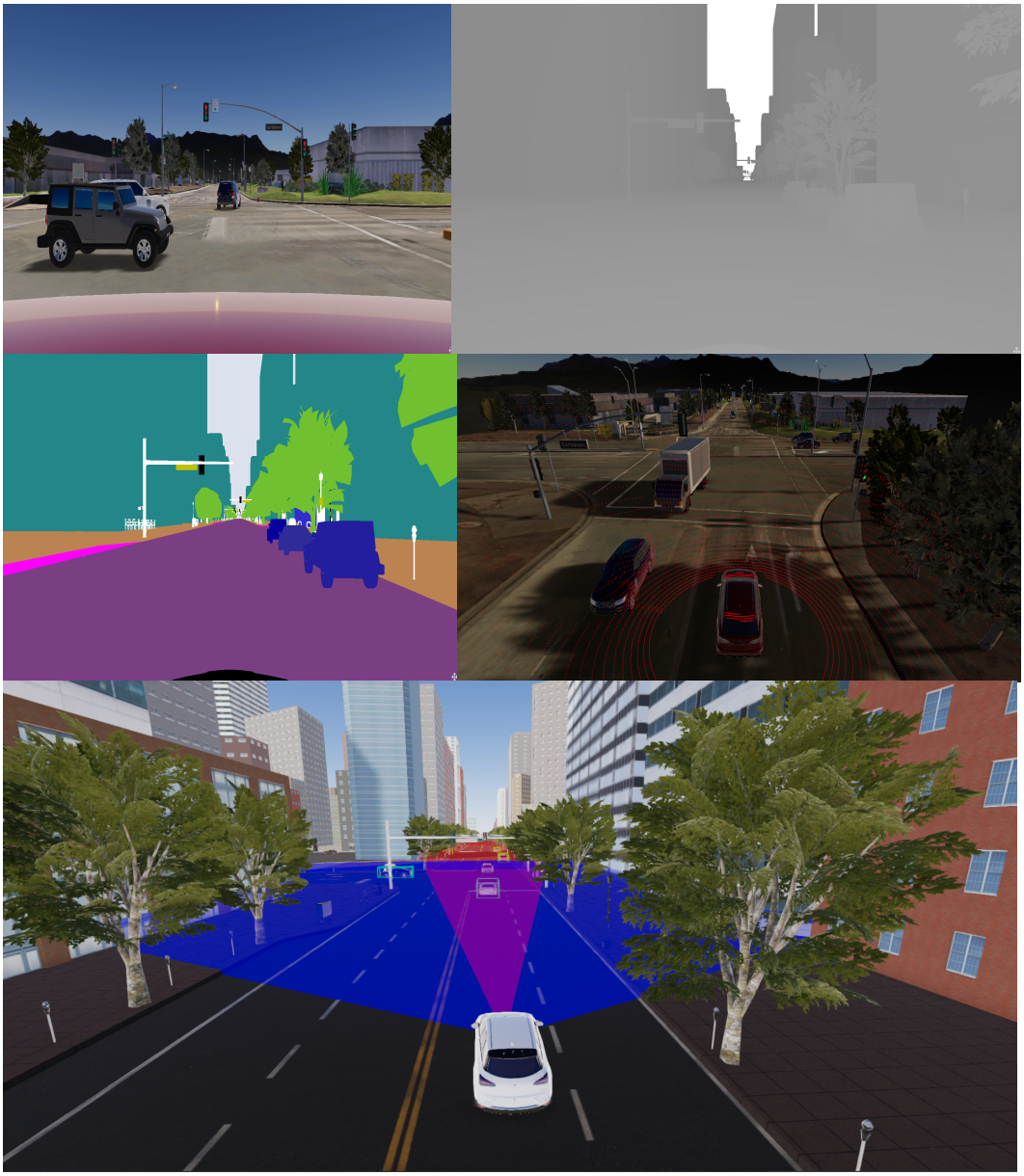

Figure 3.1: Different types of sensors in LGSVL simulator. Anticlockwise(from top): RGB colour camera, Segmentation camera, Radar(also 3D bounding boxes), LiDAR, Depth camera

Thus LGSVL simulator [8] provides the best virtual environment to conduct our experiments for autonomous driving.

4 Implementation

4.1 What to include here?

1. Docker - Dockerfile contents - docker scripts - how they are invoked.
2. LGSVL - C# language, Unity engine, code organisation, sensor plugin and its structure, data type conversion, WebUI - JSON- sensors used. Making Radar work, Increasing traffic density and time at the signal.
3. Data collection - ROS2- ros2 message types - topics - msgfilters - approx-sync - callback- cv2 libraries - saving. Also ROS2 web bridge.
4. Preprocessing - CV2 image manipulation - stacking, concatenate - Time series LSTM stuff - saving in HDF5.
5. Training - Loading hdf5 - splitting data - Functional API - normalisation - datafusion - early, late - concat - batch normalisation - stride - kernel - flatten - FC - prediction - Activation function - loss function - learning rate - optimizer - epochs - shuffling - batch size - callbacks - ES - MC - TB - saving models.
6. Evaluation - similar stuff to data collect - image manipulation - prediction - publishing. How to validate evaluation?

5 Evaluation

5.1 Testbed setup

1. dataset constant - Only dry day data? or everything?
2. CNN parameters variable.
3. epochs, learning rate, optimizer, loss function constant
4. activation function constant or variable?
5. for LSTM - timesteps value?
6. graph - training, val loss vs epochs?
7. graph - loss vs learning rate?
8. graph - timesteps vs loss?
9. graph - evaluation performance comparison?
10. graph - Accel, brake, noaction predicted vs ?
11. graph - brake, steering predicted vs ?
12. graph - accel, brake, steering predicted vs ?
13. graph - accel,brake, steering, distance(radar) predicted vs ?
14. graph - with seg camera vs without?
15. graph - with radar vs without?
16. graph - imbalanced vs balanced Cross entropy?

List of Figures

References

[1] Google, "Waymo." https://waymo.com/

[2] B. University, "Berkeley-deepdrive. [online]." https://deepdrive.berkeley.edu/

[3] Apollo, "Apolloscape dataset." http://apolloscape.auto/

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[5] H. Rohling and E. Lissel, "77 ghz radar sensor for car application," in *Proceedings International Radar Conference*, 1995, pp. 373–379.

[6] D. Lv, X. Ying, Y. Cui, J. Song, K. Qian, and M. Li, "Research on the technology of lidar data processing," in *2017 First International Conference on Electronics Instrumentation Information Systems (EIIS)*, 2017, pp. 1–5.

[7] A. Carullo and M. Parvis, "An ultrasonic sensor for distance measurement in automotive applications," *IEEE Sensors Journal*, vol. 1, no. 2, pp. 143–, 2001.

[8] G. Rong, B. H. Shin, H. Tabatabaee, Q. Lu, S. Lemke, M. Možeiko, E. Boise, G. Uhm, M. Gerow, S. Mehta *et al.*, "Lgsvl simulator: A high fidelity simulator for autonomous driving," *arXiv preprint arXiv:2005.03778*, 2020. https://www.lgsvlsimulator.com/

[9] Nvidia, "Nvidia simulator." https://www.nvidia.com/en-us/self-driving-cars/drive-constellation/

[10] Carla, "Carla simulator." https://carla.org/

[11] IPG-Automotive, "Carmaker- simulator." https://ipg-automotive.com/products-services/simulation-software/carmaker/

[12] ROS, "Robotic operating system." https://index.ros.org/doc/ros2/

[13] "Verge report on lidar." https://www.theverge.com/2020/1/7/21055011/lidar-sensor-self-driving-mainstream-mass-market-velodyne-ces-2020

[14] Z. Chen and X. Huang, "End-to-end learning for lane keeping of self-driving cars," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, 2017, pp. 1856–1860.

[15] 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," 2016.

[16] S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *The International Journal of Robotics Research*, 03 2016.

[17] H. Xu, Y. Gao, F. Yu, and T. Darrell, "End-to-end learning of driving models from large-scale video datasets," *CoRR*, vol. abs/1612.01079, 2016. http://arxiv.org/abs/1612.01079

[18] J. Kim and J. F. Canny, "Interpretable learning for self-driving cars by visualizing causal attention," *CoRR*, vol. abs/1703.10631, 2017. http://arxiv.org/abs/1703.10631

[19] M. Bojarski, P. Yeres, A. Choromanska, K. Choromanski, B. Firner, L. D. Jackel, and U. Muller, "Explaining how a deep neural network trained with end-to-end learning steers a car," *CoRR*, vol. abs/1704.07911, 2017. http://arxiv.org/abs/1704.07911

[20] Z. Yang, Y. Zhang, J. Yu, J. Cai, and J. Luo, "End-to-end multi-modal multi-task vehicle control for self-driving cars with visual perceptions," pp. 2289–2294, 08 2018.

[21] K. Liu, Y. Li, N. Xu, and P. Natarajan, "Learn to combine modalities in multimodal deep learning," 2018.

[22] E. Park, X. Han, T. L. Berg, and A. C. Berg, "Combining multiple sources of knowledge in deep cnns for action recognition." in *WACV*. IEEE Computer Society, 2016, pp. 1–8. http://dblp.uni-trier.de/db/conf/wacv/wacv2016.html#ParkHBB16

[23] Y. Zhou and K. Hauser, "Incorporating side-channel information into convolutional neural networks for robotic tasks," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 2177–2183.

[24] W. Wang, D. Tran, and M. Feiszli, "What makes training multi-modal classification networks hard?" in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 12 695–12 705.

[25] Y. Xiao, F. Codevilla, A. Gurram, O. Urfalioglu, and A. M. López, "Multimodal end-to-end autonomous driving," *CoRR*, vol. abs/1906.03199, 2019. http://arxiv.org/abs/1906.03199

[26] F. Codevilla, M. Mller, A. Lopez, V. Koltun, and A. Dosovitskiy, "End-to-end driving via conditional imitation learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 4693–4700.

[27] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[28] T. Mitchell, "Machine learning," *McCraw Hill*, 1996.

[29] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, p. 100, http://www.deeplearningbook.org.

[30] K. P. Murphy, *Machine learning: a probabilistic perspective*, Cambridge, MA, 2012, p. 3.

[31] C. I. for Advanced Research, "Cifar 10 image dataset." https://www.cs.toronto.edu/~kriz/cifar.html

[32] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[33] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, no. null, p. 21212159, jul 2011.

[34] G. Hilton, "RMSProp - Unpublished method from coursera lecture," 2012. http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

[35] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, p. 85117, Jan 2015. http://dx.doi.org/10.1016/j.neunet.2014.09.003

[36] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 06 2014.

[37] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[38] S. Hochheiter and J. Schmidhuber, "Untersuchungen zu dynamischen neuronalen netzen. masters thesis, technische universitaet muenchen," *Diplomarbeit*, June 1991. http://people.idsia.ch/~juergen/SeppHochreiter1991ThesisAdvisorSchmidhuber.pdf

[39] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. https://doi.org/10.1162/neco.1997.9.8.1735

[40] B. Khaleghi, A. Khamis, F. O. Karray, and S. N. Razavi, "Multisensor data fusion: A review of the state-of-the-art," *Inf. Fusion*, vol. 14, no. 1, p. 2844, Jan. 2013. https://doi.org/10.1016/j.inffus.2011.08.001

[41] V. Malyavej, W. Kumkeaw, and M. Aorpimai, "Indoor robot localization by rssi/imu sensor fusion," in *2013 10th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology*, 2013, pp. 1–6.

[42] Y. Dobrev, S. Flores, and M. Vossiek, "Multi-modal sensor fusion for indoor mobile robot pose estimation," in *2016 IEEE/ION Position, Location and Navigation Symposium (PLANS)*, 2016, pp. 553–556.

[43] S. Bohez, T. Verbelen, E. De Coninck, B. Vankeirsbilck, P. Simoens, and B. Dhoedt, "Sensor fusion for robot control through deep reinforcement learning," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 2365–2370.

[44] Google, "Tensorflow, a open source ml framwork." https://www.tensorflow.org/

[45] Google, "Keras, a high level python wrapper for tensorflow." https://keras.io/

[46] Facebook, "Pytorch, a open source ml framework." https://pytorch.org/

[47] ROS, "About ros." https://www.ros.org/about-ros/

[48] E. Ackerman and E. Guizzo, "Wizards of ros: Willow garage and the making of the robot operating system." https://spectrum.ieee.org/automaton/robotics/robotics-software/wizards-of-ros-willow-garage-and-the-making-of-the-robot-operating-system

[49] B. Gerkey, "Why ros2?" http://design.ros2.org/articles/why_ros2.html

[50] ROS, "Rosbridge suite." http://wiki.ros.org/rosbridge_suite

[51] ROS, "Message filters." http://wiki.ros.org/message_filters

[52] Docker, "Docker - how to get started." https://docs.docker.com/get-started/

[53] L. S. team, "Lgsvl simulator github page." https://github.com/lgsvl/simulator

[54] Baidu, "Baidu-apollo." https://github.com/ApolloAuto/apollo

[55] S. Kato, E. Takeuchi, Y. Ishiguro, Y. Ninomiya, K. Takeda, and T. Hamada, "An open approach to autonomous vehicles," *IEEE Micro*, vol. 35, no. 6, pp. 60–68, 2015.

[56] U. Technologies, "Unity." https://unity.com/