Scenario generation using a Generative Adversarial Network (GAN)

Sankar, Varun

Award date:
2020

Department of Mathematics and Computer Science
Electronic Systems Research Group

Scenario generation using a Generative Adversarial Network (GAN)

Master Thesis

Varun Sankar

Supervisors:
Dr. Dip Goswami (Electronic Systems, TU/e)
ir. Anne van der Heide (Siemens Digital Industries Software)

Mentor:
Ph.D. Candidate Sajid Mohamed (Electronic Systems, TU/e)

Eindhoven, October 2020

Abstract

In the last decade, cars are progressively being equipped with Advanced Driver Assistance Systems (ADAS). They have the capability to enhance road safety and driver comfort. With rapid increase in the usage of automated driver functionalities in vehicles, it is important that these systems are sufficiently validated. It is not practical and safe to deploy the ADAS into vehicles without proper validation as it can endanger human lives. Physically testing these systems in a car and driving millions of kilometers are highly expensive and time consuming task. Hence testing them in virtual environment can speed up the development of these systems. Automated vehicles can be tested in a virtual environment on a large set of testing scenarios. Testing on the virtual environment provides a safe and efficient means to design and evaluate ADAS systems. A scenario consists of many elements, which can be divided into three main groups: the environment model (all the static elements like road elements, buildings and traffic signs), the vehicle under test (the ego vehicle), and the other road users. The presence of different components in a scenario leads to huge parameter space which can be challenging to handle. A potential solution is to resort to machine learning techniques.

In this work, we analyse the feasibility of generating scenarios using Generative Adversarial Network (GAN). We study different types of GAN models available, analyse them both quantitatively and qualitatively and then select an appropriate type of GAN model to generate scenarios. The scenario chosen for this project is parking lot occupation where we try to generate realistic parking scenarios and demonstrate the generated scenarios on Simcenter PreScan software. With this approach, we reduce the time taken to generate different test scenarios and hence this can lead to faster testing and reduce time to market of the ADAS.

Abbreviations

ADAS Advanced Driver Assistance Systems. 1, 3, 10, 11, 22, 57

AI Artificial Intelligence. 12

API Application Programming Interface. 23

CAN Controller Area Network. 6

CGAN Conditional Generative Adversarial Network. 18, 26, 29–31, 34, 67, 68

CoG Center of Gravity. 24, 26, 45, 48–52

DNN Deep Neural Network. 13, 16

EM Earth Mover's. 19

FID Frechet Inception Distance. 9, 44

GAN Generative Adversarial Network. 3–5, 7–10, 14, 16–19, 21, 24, 26–29, 31, 34, 44, 45, 58, 65, 66

GPS Global Positioning System. 6

GUI Graphical User Interface. 22

HIL Hardware-in-the-loop. 11

JS Jenson-Shannon. 18, 19, 31

KDE Kernel Density Estimation. 3

KL Kullback-Leibler. 15, 16

SAE Society of Automotive Engineers. 1

SIL Software-in-the-loop. 11

TGAN Tabular Generative Adversarial Network. 9

UAV Unmanned Aerial Vehicle. 9

VAE Variational Autoencoder. 14, 16

WGAN Wasserstein Generative Adversarial Network. 19

WGAN-GP Wasserstein Generative Adversarial Network with gradient penalty. 5, 20, 26, 31–35, 39, 43–46, 69, 70

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Within the last decade, many innovations in autonomous driving are based on ADAS. Highly automated driving functions aim to support drivers in various situations. Most of the road accidents occur due to human errors. These ADAS functions are developed to automate, adapt and enhance vehicle systems to achieve a high level of road safety and better driving. Some of the driver assist functions are shown in Figure 1.1. These functions are anticipated to be key in achieving a high level of road safety, further reduction in harmful emissions, improving traffic flow and increasing comfort and ensuring mobility for all [23]. The Society of Automotive Engineers (SAE) defines 6 levels of automation [9] ranging from 0 (fully manual) to 5 (fully autonomous), as shown in Figure 1.2.

1. Level 0 - No automation.
2. Level 1 - Driver assistance like adaptive cruise control, lane keep assist etc.
3. Level 2 - Partial automation, providing assist in controlling speed and steering.
4. Level 3 - Conditional automation where vehicles are capable of driving themselves, but only under ideal conditions and with limitations. A human intervention is required to take over if needed.
5. Level 4 - High automation where all driving functions are automated but constrained to known use cases
6. Level 5 - Full automation

In the levels 0-2, human driver monitors the driving environment. The driver is in control of driving whenever these support functions are engaged. The driver must steer, brake or accelerate when needed to maintain safety. In the levels 3-5, the automated driving systems monitors the driving environment. The driver is not driving the vehicle when the support functions are engaged. These automated driving features will not require the driver to take over driving except in case of level 3 where the driver must interfere to maintain safety.

With the rapid increase in usage of automated driver functionalities in vehicles to support the driver, it is crucial that these systems are validated sufficiently to guarantee the safety of the vehicle and other road users. According to Wachenfeld et al. [65], around 6 billion test kilometers are needed for validation of automated driver functionalities. Therefore physically testing, driving millions of kilometers, these systems in a car are expensive and very time-consuming task. To speed up the development of automated driver functions and to increase the testing coverage, a scenario-based approach would be an alternate effective solution for testing in a virtual environment.

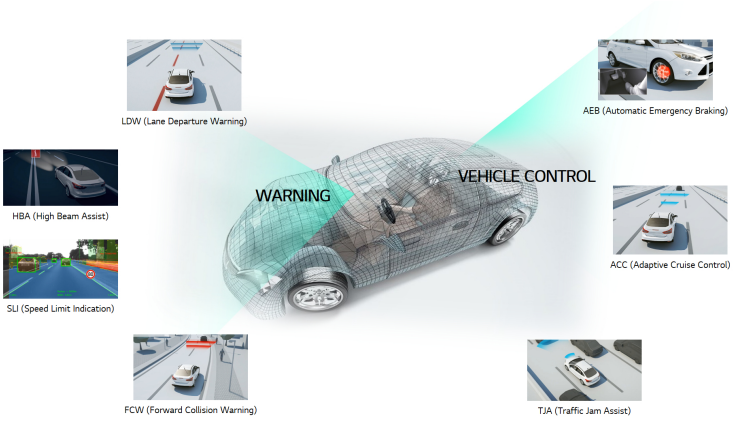


Figure 1.1: Examples of autonomous driving functionalities [1]

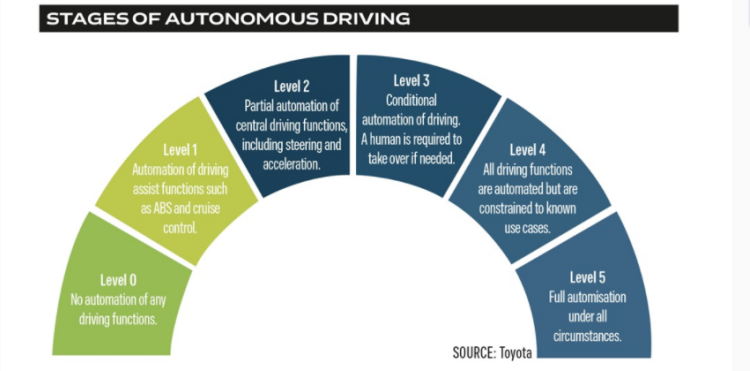


Figure 1.2: SAE Levels of automation [8]

1.1 What is a scenario?

A scenario consists of many elements, which can be divided into three main groups: the environment model (all the static elements like road elements, buildings and traffic signs), the vehicle under test (the ego vehicle), and the other road users. In other words, a scenario is a quantitative description of the ego vehicle, its activities and/or goals, its dynamic environment (consisting of traffic environment and conditions) and its static environment [23]. From the perspective of the ego vehicle, a scenario contains all relevant events.

There are three levels of abstraction for scenario representation [42]:

1. Functional Scenario: Functional scenario depicts the most abstract level of scenario representations. This includes operating scenarios on a semantic level. The representation includes a linguistic and consistent description of entities and relationship between the entities.

2. Logical Scenario: Logical scenario depicts the detailed representation of functional scenarios with the help of state-space variables. It represents the entities and relations of those entities with the parameter ranges in state-space. Parameter ranges can be specified using probability distributions.

3. Concrete Scenario: Concrete scenario describes the entities and relationship between the entities using distinct values for each parameter in the state space. The concrete scenario is the basis for test case generation in the testing phase.

While the functional scenario depicts the most abstract form of scenario representation understandable to humans, the logical and the concrete scenarios demand an efficient machine-readable representation. The logical and the concrete scenario demand different levels of details for representation. On one hand, logical scenario asks for a scenario representation via parameter ranges in state-space providing multiple degrees of freedom for the determination of concrete parameters. On the other hand, the concrete scenario demands for a representation that includes concrete parameter values of all entities involved. This form of representation is required for reproducible test case execution. An example of a scenario is shown in Figure 1.3. The scenario depicts a situation where the ego vehicle (blue car) tries to overtake vehicle ahead (black car) on a highway. The first action performed by the ego vehicle is turn on the left indicator before making the lane change. Then lane change action is performed by the ego vehicle where it moves to the left lane on the highway. Then it follows the lane to overtake the black vehicle.

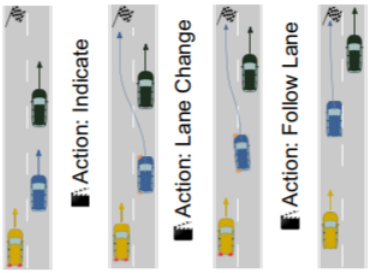


Figure 1.3: An example of a scenario [62]

1.2 Motivation

Testing on the virtual environment with simulation software provides a safe and efficient means to design and evaluate ADAS systems. The simulated scenarios are completely quantifiable, reproducible and controllable. Automated vehicles can be tested in a virtual environment on a large set of testing scenarios. Scenarios are generated to test the decision-making capabilities of automated vehicles.

In literature, there are two ways to generate scenarios. First is the data-driven approach, and the second is the knowledge-based approach. In the data-driven approach, the main idea is to collect measurement data and determine and categorize occurring scenarios. In general mathematical functions (like Kernel Density Estimation (KDE) or polynomial functions) are created to parameterize the scenarios. In the knowledge-based approach, scenarios are generated from existing knowledge like road traffic regulations, functional description, expert knowledge etc. First, the knowledge is structured according to 5-layer model by Bagschik et al. [12], and then it is parameterized and converted to data formats required by simulation softwares like Simcenter Prescan which uses OpenDRIVE [5] and OpenSCENARIO [6] that describe the road network and the traffic participants and the environment. For a purely data-driven approach, the measurement data do not describe all aspects of the scenario. Hence diversity of scenario generation is difficult. The drawback of the knowledge-driven approach is that the process of transforming a functional scenario to state-space representation takes considerable manual effort. Also, the presence of different components in a scenario leads to a huge parameter space which can be challenging to handle.

A potential solution is to leverage to machine learning techniques. One of the most recent and promising machine learning frameworks is the GAN, as shown in Figure 1.4. GANs have emerged as a powerful framework for learning complex data distributions and produce samples which are as close to real data as possible. A GAN is a class of machine learning models that has two neural network contest against each other in a mini-max game. So given a training dataset, this model

learns to reproduce new data that are statistically similar to the training dataset. Thus by using GAN, we can generate variations of scenario. Using GANs allows us to generate relevant scenarios. Also, complex scenarios can be modelled with considerable ease. The main advantage of using GAN is that it increases the automation to generate scenarios.

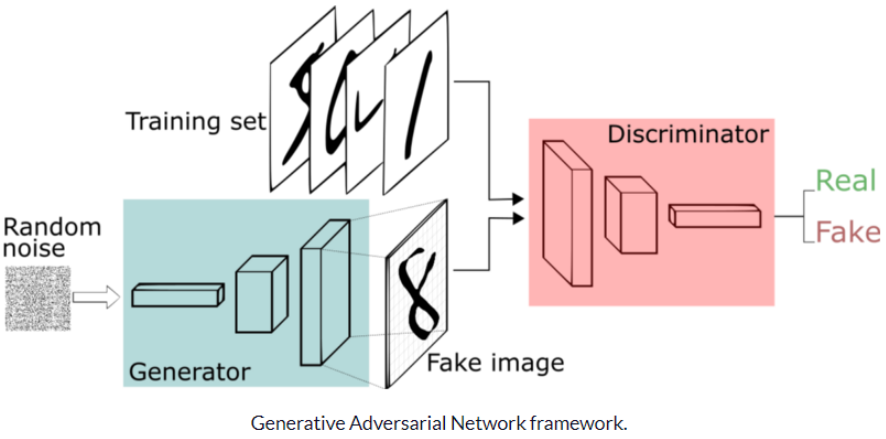


Figure 1.4: Generative Adversarial Network framework [59]

1.3 Problem statement

This thesis aims to generate realistic parking scenarios using Generative Adversarial Network. The generated parked cars location are done for various multiplicities using GAN. The number of cars to be parked is given as input to the generator network. A variant of parking scenario, cars overlapping two parking slots, is also generated. The generation of these scenarios is a non-trivial task for various reasons. To begin with, the vast majority of Generative Adversarial Networks focuses on images and are successful. They are used for tasks like texture synthesis [32] , image super-resolution [71], image inpainting [20], image editing [72] and Image-to-Image translations [73, 31]. Due to the strong spatial relationship of the inputs along with the quickly verifiable generation quality, either with well-known scores or with a simple visual inspection, the GANs have proven to work well with images. However, the same is not extensively used for tabular data generation. For parking scenario generation in the virtual environment, the tabular based inputs approach is taken instead of using image inputs to the generative model. As aforementioned generative model work well with image-based datasets, this work approaches the task of generating data in tabular form. The detailed approach is explained in sections 4, 5 and 6.

1.4 Research questions

Following research questions are defined, that guides towards achieving the goal of the project:

- Can Generative Adversarial Network be used to generate parking scenarios while preserving the underlying distribution and patterns of the training data?
  - ⋆ Which Generative Adversarial Network model architecture can be used for generation of scenario?
  - ⋆ How should the dataset look-like in order to achieve the goal of the project?
  - ⋆ How to evaluate the performance of the model generating new scenarios?

1.5 Thesis outline

This thesis consists of eight chapters. Chapter 1 provides the introduction to the topic and provides a problem statement and research questions. The related work on the topic is discussed in Chapter 2. Chapter 3 provides background information about the Generative Adversarial Network and types of GAN. Chapter 4 elaborates about methodology chosen to approach the problem. Chapter 5 explains the selection of a suitable GAN model to implement the parking scenario. Chapter 6 explains the optimization of WGAN-GP model chosen to implement the parking scenario. Chapter 7 provides the evaluation metrics and analysis of the results. Finally, chapter 8 provides the conclusion to this thesis and future directions to this work.

Chapter 2

Related work

This chapter presents some of the research work done related to Generative Adversarial Networks and scenario generation.

2.1 Scenario generation

In literature, there are currently two approaches for scenario generation. One approach is that scenarios are generated through the data-driven approach, while the other one is through the knowledge-driven approach.

In the data-driven approach, the main idea is to collect measurement data and determine and categorize occurring scenarios. Data-driven approach is followed in [23], [19], [53]. In [47], [48], [46], [18], [17], scenarios are identified based on the camera data. Streetwise method [23] uses a continuous process of mining scenarios out of real-world data. These data are generated from the sensor's output of the ego vehicle like accelerometer, camera, radar and Global Positioning System (GPS). Also, sometimes the contents of the scenario are extended by combining further on-board information from the Controller Area Network (CAN) bus. From these data driving events and activities are detected using hybrid techniques that combine physical/deterministic models with data analytics. The detection method not only provides an overview of the type and frequency of an event but also parameters describing it. Thus scenarios are parameterised. Pütz et al. [53] present a similar approach that describes a concept for a toolchain to collect data (for example measurement data and traffic simulation data), calculate metrics to characterize scenarios and cluster them as logical scenarios. These logical scenarios are provided as a basis for generating test cases.

In knowledge-based scenario generation, the main idea is to generate scenarios using existing knowledge, like regulatory recommendations, and augment these scenarios. The first step, to generate functional scenarios, the available knowledge has to be structured, varied on a semantic level and described linguistically. Bagschik et al., [12] present a knowledge-based approach to creating scenes using ontology. With functional scenarios as base scenarios are then represented in terms of state-space variables. These are termed as logical scenarios. These logical scenarios are later concretized by choosing a concrete value within the defined value range of the parameters. The comparison between the two approaches is shown in Figure 2.1.

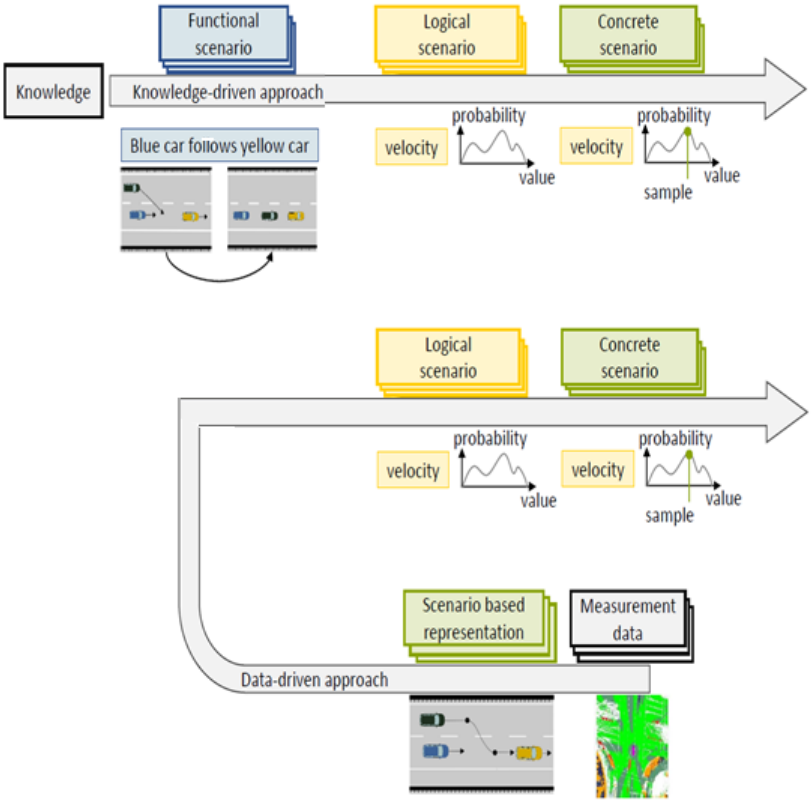


Figure 2.1: Comparison of a knowledge-driven and a data-driven approach for scenario generation [41]

2.2 Generative Adversarial Networks (GANs)

Generative Adversarial Networks [25] were introduced by Ian J. Goodfellow and his colleagues in 2014. Since its inception, the GANs have been described as "the most interesting idea in the last ten years in Machine Learning" by Yann LeCun, a chief scientist at Facebook. Over the past few years, GANs have become very prominent in the field of generative models.

2.2.1 GANs for images and music generation

Since the emergence of Generative Adversarial Networks [25], there have been constant improvements in the field of data generation. GANs have gained more and more attention from the scientific community especially, and a vast majority of GANs focuses on images. GANs are used for many tasks like anime images generation [33], image inpainting [20], image-to-image translations [73, 31] and text to image synthesis [55]. Figure 2.2 shows the generated anime images using GAN from the paper titled "Towards the Automatic Anime Characters Creation with Generative Adversarial Networks". Image inpainting is a technique used for repairing the images or refilling missing parts. The aim of inpainting is to recontruct the damaged image without introducing noticable changes. The author [20] proposes to use generative adversarial networks for image inpainting. The Figure 2.3 shows the results of image inpainting using GAN. The effectiveness of the new approaches to generate new related data is driving an increasing number of researchers in this direction.

Research has also been done on various other fields, such as music [44] thanks to the spatial

correlation of nearby values in time series, approaches employing Recurrent Neural Networks for the generation of this type of data have been developed.

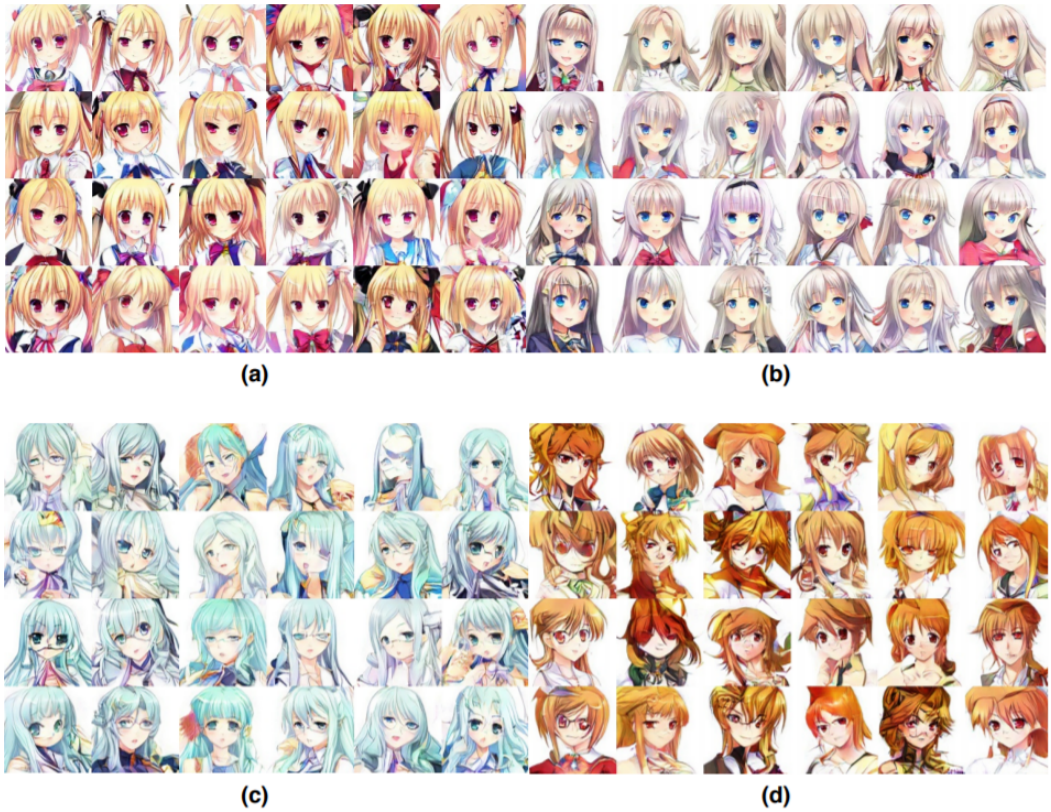


Figure 2.2: Generated anime images using GAN [33]

Figure 2.3: Visual comparison between PGGAN-RES (a GAN model) and Neural Patch Synthesis (NPS) on 512x512 paris street view dataset [20]

2.2.2 GANs for tabular data generation

With success on tasks related to images, GANs have also been used for generating tabular data. Xu et al. [69] proposed Tabular Generative Adversarial Network (TGAN) which could generate tabular data like medical or educational records. The goal was to provide a model that is capable of generating continuous and categorical data. MedGAN [16] generates high-dimensional discrete variables(e.g., binary and count features) which uses a combination of an autoencoder and generative adversarial networks. Park et al. TGAN [52] uses generative adversarial networks to synthesize fake tables that are statistically similar to the original table. By generating fake tables, the privacy concerns of sharing the data to public is overcome.

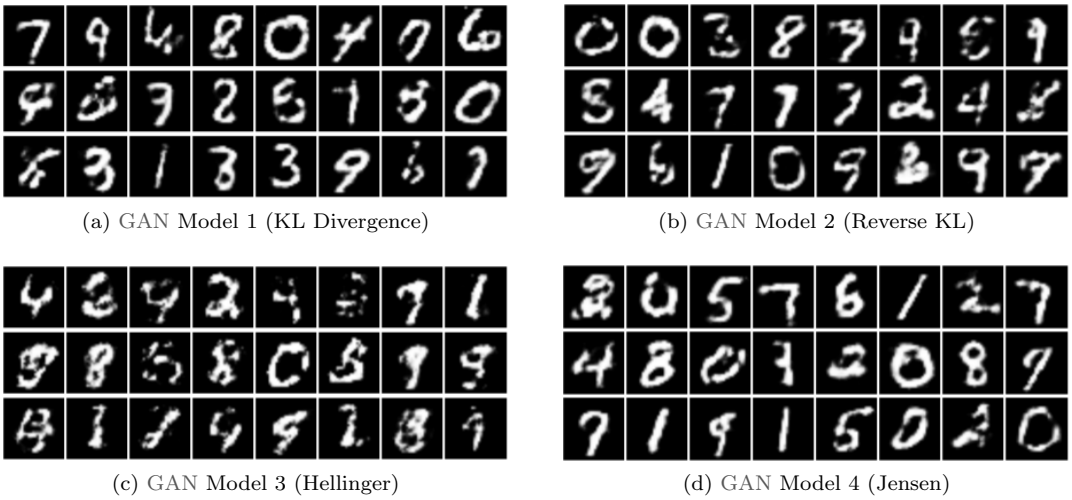
2.2.3 Usage of GANs in autonomous driving fields

In the autonomous driving fields, the main application where GANs are used are image-to-image-translation[63] for style transfer across different conditions of lighting, weather etc. Since autonomous driving systems have to be extremely robust and this requires training the model to all possible scenarios which can happen in real life. Collecting such a dataset is quite infeasible in practice, but provides a promising path to ameliorate this issue by generating realistic dataset. In VeGAN [35] the author proposes to train a GAN to generate images of vehicles that look like images taken from a top-down view of an Unmanned Aerial Vehicle (UAV). In data-driven maneuver modelling [36] the author proposes to utilise unsupervised machine learning to train neural networks to solve the modeling problem. In this paper, generative model architectures are adapted from image domain to time series domain which focuses on lane-changing maneuvers on highways. The models learn a small set of intuitive parameters without the need for labeled data and use them to generate new realistic trajectories. The neural networks are based on the InfoGAN [15] and beta-VAE [29] architectures

2.2.4 Evaluation of performance of GANs

GANs are generative models that are able to generate new samples, and researchers often evaluate GAN generators via manual and visual inspection. This involves using a generator to create a batch of synthetic images and then evaluating the quality and diversity of images generated compared to its target data. This method is deployed in many research works [25][54][58][43][50]. The Figure 2.4 shows visual inspection method used in one such work [50]. In this work, the authors visualized 24 samples from four different GAN models and compared their performance via visual inspection. Although manual inspection is the simplest method to evaluate the model it has few limitations like subjective bias, requires knowledge of what is realistic and what is not part of target domain and time consuming. A paper titled "Pros and Cons of GAN Evaluation" by Ali Borji [14] reviews the list of quantitative and qualitative measures for evaluating generative models.

Most widely used evaluation metrics to capture the quality and diversity of generated images are inception score [13] and Frechet Inception Distance (FID) score [28]. The inception score metric gives a score to the generated output image that measures how realistic the GAN's output is. Higher the inception score better the generated output. FID measures the Wasserstein-2 distance [66] between feature vectors calculated for real and generated images. For FID score, the lower scores correlate to a high quality of images generated.



(a) GAN Model 1 (KL Divergence)

(b) GAN Model 2 (Reverse KL)

(c) GAN Model 3 (Hellinger)

(d) GAN Model 4 (Jensen)

Figure 2.4: An example of manual inspection method used by researchers [50]. The authors compared four different GANs by visualizing selected samples and manually inspecting them.

2.3 Simulation frameworks

Exhaustive testing of ADAS is critical before introducing these systems in road cars. There are several simulation softwares available to perform ADAS testing [22] [2] [45] [21]. Simcenter Prescan software, developed by Siemens, provides a simulation platform to prototype, test and validate ADAS. Several design iterations can be performed in a quick and cost-effective way by simply modifying the system's parameters and running the simulation again. Therefore, by using Simcenter Prescan the amount of work needed to bring an ADAS to the market can be significantly reduced. delivers robust initial designs in the concept phase, rapid optimization in the development phase and a fast launch in the confirmation phase. Simcenter also supports industry standards like OpenDRIVE [5] and OpenSCENARIO [6]

Son et al. [22] show a testing framework based on co-simulation of two softwares namely: Siemens Simcenter Amesim and Simcenter Prescan for verification and validation of ADAS. The simulation takes into account the account high fidelity vehicle dynamics, traffic environment, sensor models, planning and control algorithms and are demonstrated on adaptive cruise control application. Simcenter amesim focuses on the vehicle dynamics part such as powertrain, suspension steering etc. and prescan focuses on the environment outside of the vehicle. Thus a combination of both softwares provides a basis for verification and validation. The cosimulation stucture is shown is Figure 2.5. The sensing layer gathers the input from sensors and applies algorithms to detect scenes, objects etc. The planning layer is responsible for deciding the desired trajectories of the vehicles and steer them along that trajectory. The vehicle dynamics offers plant modelling and capabilities to connect to controls design and validate control strategies. There are two main requirements in control design: stability and tracking. Stability is necessary to prevent the vehicle from going off road and stabilization is obtained through feedback control design.

Roggero et al. [56] show different ways of generating virtual driving scenarios using MATLAB. They include designing driving scenarios using both scripts and graphical user interfaces, importing scenarios from existing libraries and generating scenarios using data recorded by in-vehicle sensors. Also the paper describes how to create variations in existing scenario through programming. After the scenario are generated they can be used in closed-loop simulations to model and test controllers. MATLAB automated driving toolbox [2], also supports industry standards like OpenDRIVE [5] and OpenSCENARIO [6].

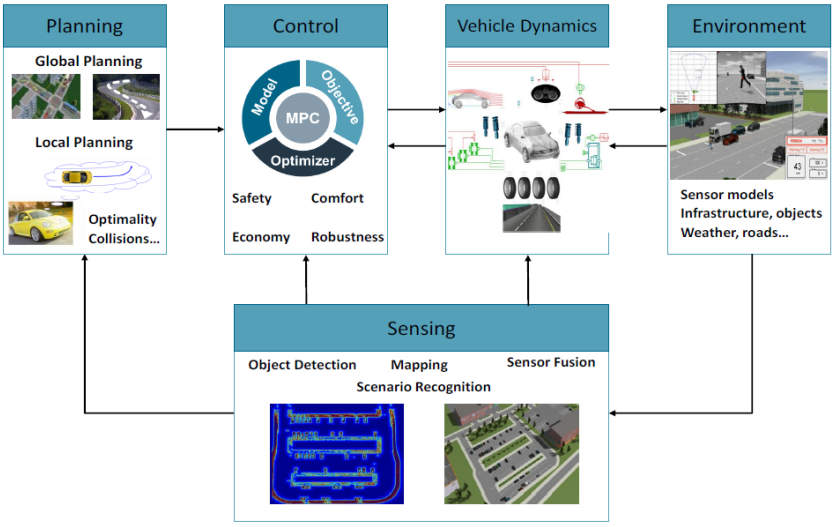


Figure 2.5: Co-simulation framework for testing ADAS [22]

The IMACS [45] simulation framework enables closed-loop simulation and validation of image-based control systems, e.g. autonomous lane-keeping assist system. IMACS is a framework for performance evaluation of image in the closed-loop system. IMACS allows both Software-in-the-loop (SIL) simulation and Hardware-in-the-loop (HIL) validation. The scenarios can be created, generated or imported from the associated physics simulation engine. IMACS currently supports generating scenarios for different weather conditions, traffic situations, and road conditions [18]. The generated scenarios are then used in closed-loop simulations to validate the image-based controllers.

dSPACE tool chain provides efficient possibilities for realistic scenario simulations of autonomous driving. Deng et al. [21] developed a HIL simulation system for testing and validating autonomous driving functionalities. The HIL system is a closed-loop system consisting of many model, hardware components and other subsystems connected through electrical interfaces where the host vehicle is modeled and executed under real time. This closed loop system uses dSPACE-based HIL simulator for communication and executing the real-time vehicle models. This simulator (which can be controlled via host PC) communicates with a hardware (dSPACE MicroAutobox) which contains the control algorithm. MATLAB and simulink are used as modeling environment for vehicles, sensors, actuators, and control algorithms. VehSimRT is used to represent a full 3D and nonlinear vehicle dynamics under real-time, to simulate vehicle dynamics. Gelbal et al. [24] proposes to solve the freezing robot problem in autonomous vehicles by interaction with the pedestrians. These autonomous vehicles are used to transport elderly or disabled people inside buildings like airports. To achieve and validate the design the author proposes to simulink simulations and HIL testing. The HIL simulations were achieved by running simulation in real time on main control unit called dSPACE MicroAutobox2.

Chapter 3

Background

In this chapter, a general understanding of the concepts used in this work is explained. At first, an overview of machine learning and deep learning are provided. This is followed by an introduction to generative models and types of generative models. Further, we elaborate on the concepts of Generative Adversarial Networks and different types of Generative Adversarial Networks.

3.1 Notations

To avoid ambiguity and maintain clarity we will use the below notations throughout this thesis work:

- $\mathbf{p_{data}}$: Distribution of the real data
- $\mathbf{p_g}$: Distribution of the generated data
- $\mathbf{p_z}$: Distribution of noise variable z
- $\mathbf{x}$: Sample from $\mathbf{p_{data}}$
- $\mathbf{z}$: Sample from $\mathbf{p_z}$
- **G** : Generator
- **D** : Discriminator

3.2 Machine learning

Machine learning is an application of Artificial Intelligence (AI) that gives the machines/systems the ability to learn and improve from experience without being explicitly instructed. Machine learning focuses on the development of computer programs that can access data and use it to learn for themselves and carry out specific tasks. Machine learning algorithms build models that take training data as input and usually return a classification, a label or a prediction related to the training data.

The process of learning begins with detection of patterns in training dataset and applying them to drive the decision making in the future. The primary aim is to allow the computers to learn automatically without human intervention or assistance and adjust actions accordingly.

The machine learning algorithms are often categorised as supervised or unsupervised.

1. Supervised machine learning algorithms are typically used for regression or classification problems. The models are trained on the labeled training dataset, meaning the dataset contains both the inputs and desired outputs, to predict about the output values. The model, when sufficiently trained, can make predictions for any new input dataset. The learning algorithm compares its predicted output with the ground truth and finds errors in order to modify the model accordingly.

2. Unsupervised machine learning algorithms are used when the information available is not labeled. Unsupervised learning studies how systems can infer a function to describe a hidden structure from unlabeled data. The system does not figure out the right output, but it explores the data and can draw inferences from datasets to describe hidden structures from unlabeled data.

3. Reinforcement learning is a class of machine learning technique in which the system interacts with its environment by producing actions in order to maximise its reward/performance. Richard S. Sutton and Andrew G. Barto in his book, "Reinforcement learning: An introduction" [61] defines reinforcement learning as a machine learning technique where the agent must learn the behaviour at run-time by interacting with the dynamic environment using trial-and-error methods. Learning involves an interaction between the learner and the environment.

3.3 Deep learning

Deep Learning is a sub-field of machine learning, which uses multiple layers of artificial neurons to learn representation [38]. It has gained notable success in speech recognition, natural language processing and many other fields including computer vision. The basic building blocks for most deep learning approaches are artificial neurons with trainable parameters, and these parameters are trained by the backpropagation procedure [37].

The most commonly used network architecture in deep learning is the Deep Neural Network (DNN). A neural network consists of several layers of neurons. Figure 3.1 illustrates one such neuron.

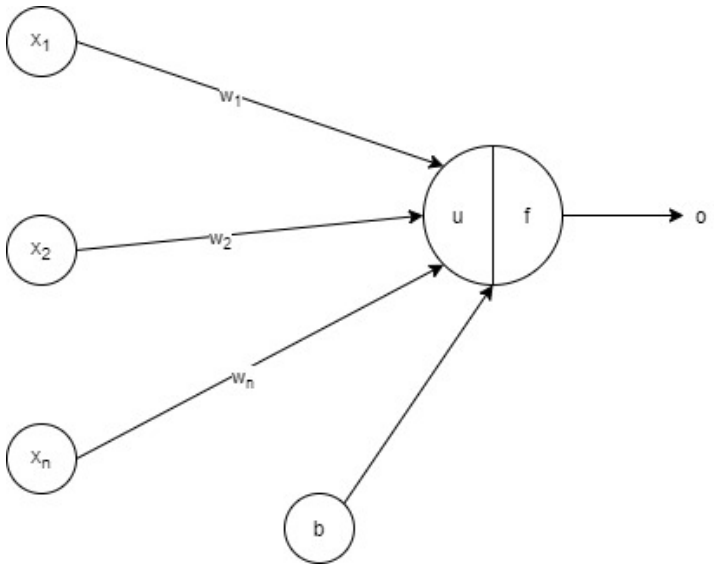


Figure 3.1: An illustration of a single neuron with n inputs, n+1 learnable weights and bias parameters. After the affine transformation, activation function f is applied.

Each neuron applies an affine transformation of the input $\mathbf{x} = \left[x_1, x_2, ..., x_\mathrm{n}\right]^T$:

$$u = \sum_{i=1}^{n} w_\mathrm{i} x_\mathrm{i} + b \tag{3.1}$$

where $\mathrm{w_i}$ is the weight corresponding to $\mathrm{x_i}$ and b is the bias. A non-linear activation function f is applied after the affine transformation. Activation functions could be sigmoid, hyperbolic tangent (tanh), ReLU [10] etc:

$$o = f(u) \tag{3.2}$$

The output o is obtained after the affine transformation and the non-linear activation. By connecting multiple neuron across different layers a deep neural network is formed as shown in Figure 3.2

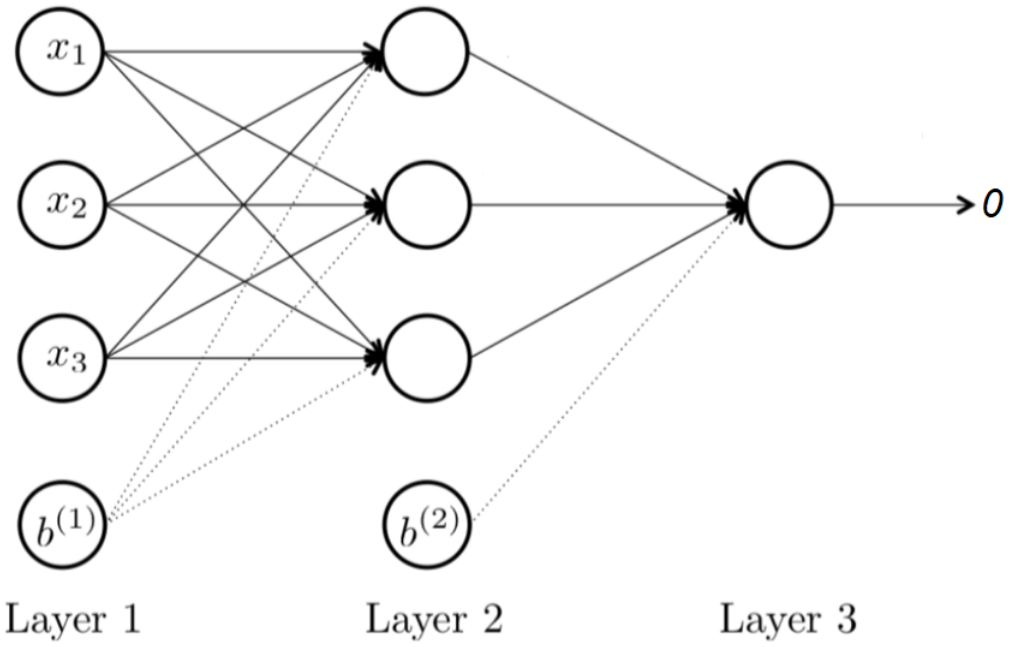


Figure 3.2: A three layer neural network

3.4 Generative models

In the recent past, the popularity of generative models has gained lots of interest due to staggering improvement in the deep learning field. The generative models aim at learning the real distribution of the training set so as to generate new data points with some variation. Although these models rely on a huge amount of data, well-designed models are capable of producing stunning realistic content of various kinds like images, texts, sounds etc. There are two prominent families of generative models, namely: Variational Autoencoder (VAE)s and Generative Adversarial Networks (GANs). While the VAE aims at maximising the lower bound of the data log-likelihood, the GAN aims at achieving equilibrium between the generator and the discriminator networks. These are discussed in detail below.

3.4.1 Autoencoders

Before discussing Variational Autoencoders [67], let us discuss what an autoencoder does. Autoencoder is an unsupervised artificial neural network that learns to efficiently compress and encode data and then learns how to reconstruct the data back from the reduced encoded representation to a representation that is as close as possible to the original data. The architecture of the Autoencoder is shown in Figure 3.3. Autoencoder network consists of two connected neural networks: the encoder and the decoder. The encoder model takes in an input and converts it into a smaller and reduced encoded representation. The decoder model is responsible to reconstruct the data from the reduced encoded representation to data as close to the original input as possible. These networks are trained to reconstruction loss which is either mean-squared error or cross-entropy loss [3] between the output and the input. Intuitively this loss measures how well the decoder is performing and how close the output is compared to the original input.

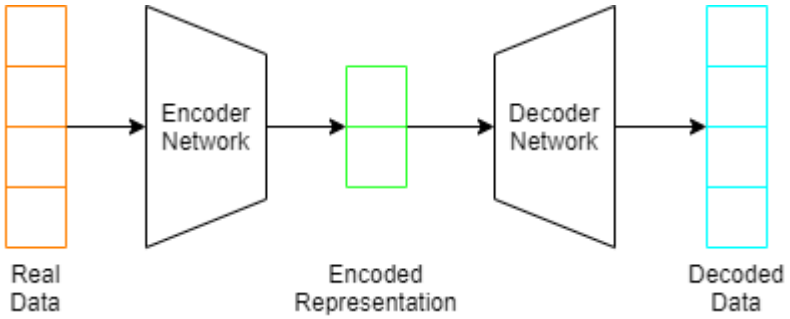


Figure 3.3: Architecture of an Autoencoder

The fundamental problem of using the autoencoders for generation is that the conversion of inputs to latent space may have gaps and discontinuities. This can be seen from Figure 3.4, where clusters are visible. The figure shows the visualisation of encodings from 2D latent space of an autoencoder trained on MNIST dataset. Clearly there are formation of various clusters. This lack of regularity in the latent space is normal because there is nothing in the training of autoencoder that enforces any regularity or organisation in latent space. Generation of a variation from a latent space that has discontinuities will result in unrealistic output, because decoder has no idea how to deal with the discontinuous region of the latent space.

3.4.2 Variational Autoencoders (VAEs)

The Variational Autoencoders differs from the autoencoders in the fact that their latent spaces are continuous, allowing random sampling and interpolation. This property makes them effective for generative modelling. So a variational autoencoder can be defined as an autoencoder whose training is regularised in order to prevent overfitting, thereby ensuring the latent space has good properties that enable the generative process. Continuous latent space generation is achieved by the encoder network outputting two vectors mean μ and standard deviation σ instead of one in case of the traditional autoencoder. While the mean vector governs where the encoding of an input should be centered, the standard deviation controls how much the mean encoding can vary. Thus the model can produce varying samples due to variation in the encoding. Ideally, we want encodings that are as close as possible to each other while still being distinct allowing smooth interpolation and enabling the construction of the new samples. This is achieved by introducing Kullback-Leibler (KL) divergence into the loss function. The KL divergence quantifies between two probability distribution i.e. how much they diverge from each other. By minimizing the KL divergence, the probability distribution parameters (μ and σ) are optimized to resemble that of the target distribution closely. The KL divergence between two probability distribution p and q

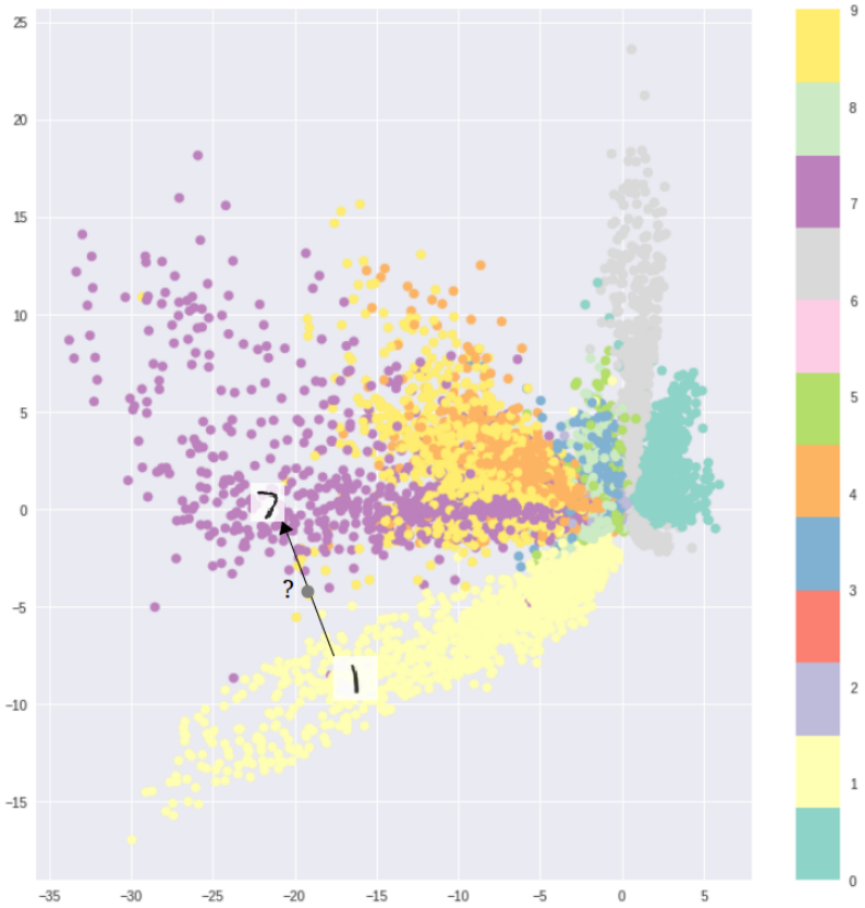


Figure 3.4: Latent space of Autoencoder trained on MNIST dataset [60]

is defined as:

$$KL(p||q) = \sum_{i=1}^{n} p_{\text{i}}. \log \frac{p_{\text{i}}}{q_{\text{i}}} \tag{3.3}$$

Thus the loss function (F) for the VAE is given by:

$$F_{\text{VAE}} = L(x, \tilde{x}) + KL(d(x|z)||p(z)) \tag{3.4}$$

where L is either mean-squared or cross-entropy error between x(input) and $\tilde{x}$(generated output), $d(x|z)$ is probability of x given z and p(z) is a prior Gaussian distribution usually standard normal distribution.

In this loss function, the first term is the reconstruction error between the input and the generated data, and the second term is the KL-divergence.

3.4.3 Generative Adversarial Network (GAN)

Generative Adversarial Network [25] (also known as vanilla GANs) was introduced by Ian Goodfellow et al. in 2014. Since its introduction, GANs have become a hot field in machine learning. GANs have had huge success as a generative model that is able to generate new meaningful content. To begin with, GANs are DNN/algorithmic architecture that uses two neural networks competing against each other in a minimax game in order to generate new synthetic data that mimic the distribution of the real data. The architecture of GAN is shown in Figure 3.5. The

two neural networks are generator network (G) and discriminator network (D). G is trained to generate realistic samples, while D is trained to distinguish the samples produced from G and those belonging to real dataset. The generator network G takes z as input with density p_z and returns an output G(z) that should follow the distribution of data p_{data}. While the discriminator tries to distinguish between real data x and generated data x_g of density p_g by outputting a class probability $D(x) \in [0,1]$ to indicate if samples are fake or real.

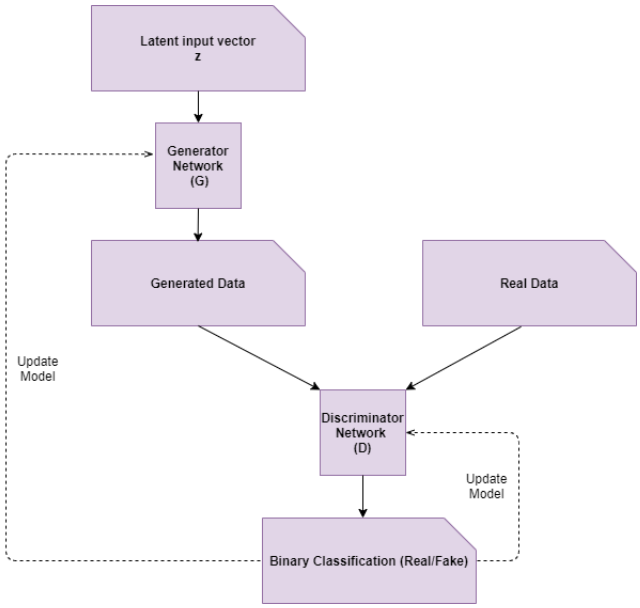


Figure 3.5: Architecture of Generative Adversarial Network

At first the GAN is initialised with random weights. The aim is to train these initial random weights in order to make the generated data look similar to the training data. The training of GAN is done simultaneously on both G and D using an adversarial setting, where weights of the G and D are updated alternatively. The D is trained to maximize the probability of assigning the correct label to both real data and generated data from G, while G is trained to maximize D's uncertainty by minimizing log(1 - D(G(z))). This results in the minmax game with loss function L(G,D) [25]:

$$\begin{aligned} \min_G \max_D L(G,D) &= \mathbb{E}_{x\text{-}p_{data}}[logD(x)] + \mathbb{E}_{z\text{-}p_z}[log(1 - D(G(z)))] \\ &= \mathbb{E}_{x\text{-}p_{data}}[logD(x)] + \mathbb{E}_{x\text{-}p_g}[log(1 - D(x))] \end{aligned} \tag{3.5}$$

The goal of the generator is to fool the discriminator, whereas the discriminator is responsible for distinguishing between real and generated data. The GAN model converges when the D model and G model reach the Nash equilibrium [30]. Nash equilibrium term is commonly used in game theory. Nash equilibrium is a state in which no player can raise its individual gain by choosing a different strategy. At this optimal point of the training, the ideal D is unable to distinguish between real and generated data, meaning the ideal G generates data successfully approximated to the real data distribution p_{data}.

For a fixed generator G, the optimum discriminator D* given a sample x is [25]:

$$D^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)} \tag{3.6}$$

The loss for a optimum discriminator would be [11]

$$L(G, D^*) = 2D_{JS}(p_{data}||p_g) - 2log2 \tag{3.7}$$

Essentially the loss function of generative adversarial network quantifies the similarity between the generative data distribution p_g and the real data distribution p_{data} by Jenson-Shannon (JS) divergence [4] when the discriminator is optimal. The best G* that replicates the real data distribution leads to the minimum L(G*, D*) = -2log2, which is aligned with equations above. Thus when the model is trained to the optimal, the D outputs 0.5 for any sample of data.

A vanilla GAN model typically suffers from a phenomenon called mode collapse. Mode collapse refers to a situation when the generator produces only a limited variety of samples. In other words, the generator learns to output only a subset of possible realistic modes. Complete mode collapse is very uncommon, but partial mode collapse occurs frequently. There are several ways to overcome the mode collapse. Few of the techniques used are feature matching [58], minibatch discrimination [58], One-sided label smoothing [58] etc. Some other frequently used techniques to overcome mode collapse are discussed in 3.4.5 and 3.4.6. These models use different cost function, which help prevent these mode collapse. Apart from these, there are other models which allow conditional training and sampling like CGAN [51] and InfoGAN [15]. In CGAN, the labels act as an extension to the latent vector z to generate and discriminate images better. To an extent, these models could also be used to learn multi-modal generation.

3.4.4 Conditional GAN (CGAN)

CGAN [43] was introduced by Mehdi Mirza et al. in 2014. The main motivation of introducing CGAN is that in unconditional GAN or vanilla GAN, there is no control over modes of data being generated. By conditioning the model on additional information (like label, class, etc.) it directs the generation process to a particular target. Using this additional information, such as class label, improves the performance of GAN in terms of more stable training, faster training and result in better quality data generation. The architecture of CGAN is similar to that of GAN with the addition of label to both generator and discriminator network. The architecture of CGAN is shown in Figure 3.6. Similar to vanilla GAN, a CGAN has the same generator G and discriminator D network. The only difference compared to vanilla GAN is that additional labels are passed as input to both G and D.

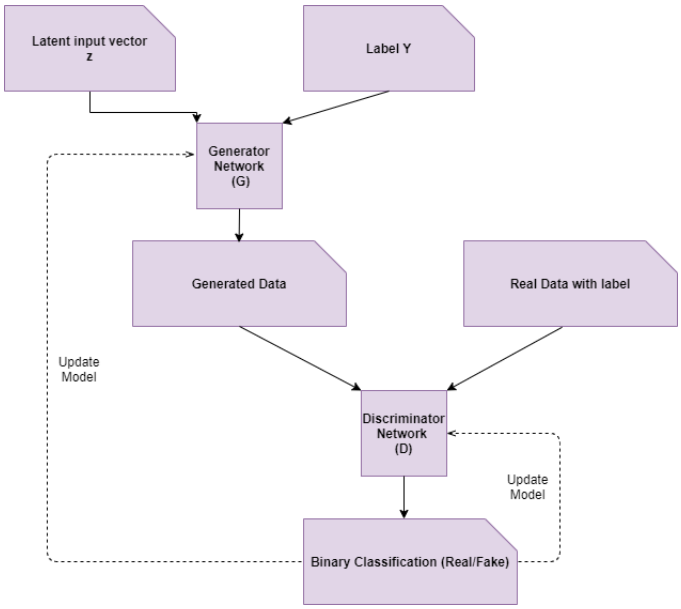


Figure 3.6: Architecture of Conditional Generative Adversarial Network

The loss function of the CGAN [43]:

$$\min_G \max_D L(G,D) = \mathbb{E}_{\mathrm{x\sim p_{data}}}[logD(x|y)] + \mathbb{E}_{\mathrm{z\sim p_z}}[log(1-D(G(z|y)))] \tag{3.8}$$

3.4.5 Wasserstein GAN (WGAN)

Wasserstein Generative Adversarial Network (WGAN) [39] was introduced by Arjovsky et al. in 2017 and it is an adaptation from the vanilla GAN. This model uses an alternative way of training. The model seeks to minimize the distance between the distribution of training data and generated data. This distance measure is called 1-Wasserstein distance. The architecture of the WGAN model is similar to the vanilla GAN model. It has two neural networks called critic (C) and generator (G). The critic network is equivalent to the discriminator network in vanilla GAN, with the difference being that the critic, instead of classifying data as real or fake, outputs a score of how real or fake the data is. This small change results in a more stable training of the model. A correlation between loss metric and generated sample quality (not possible in vanilla GAN) exists as WGAN loss reflects on the generated sample quality. Lower the loss, better the quality of the generated samples. The model also suffers no mode collapse and the results are often better than vanilla GAN.

Wasserstein Distance

The Wasserstein distance or Earth Mover's (EM) distance is the minimum cost of transporting mass in converting from one data distribution to another. The Wasserstein distance for real data distribution $\mathrm{P_{data}}$ and the generated data distribution $\mathrm{P_g}$ is mathematically defined as follows [39]:

$$W(P_{\mathrm{data}}, P_{\mathrm{g}}) = \inf_{\gamma\in\Pi(P_{\mathrm{data}},P_{\mathrm{g}})} \mathbb{E}_{\mathrm{(x,y)\sim\gamma}}[\| x-y \|] \tag{3.9}$$

$\Pi(\mathrm{P_r},\mathrm{P_g})$ denotes the set of all joint distributions γ(x,y) whose marginals are $\mathrm{P_{data}}$ and $\mathrm{P_g}$ respectively. In otherwords γ(x,y) represents amount of mass that needs be transported from x to y to transform the distribution from $\mathrm{P_{data}}$ to $\mathrm{P_g}$.

The Wasserstein distance has advantages over JS-divergence. Even when two distributions are located in lower-dimensional manifolds without overlaps, Wasserstein distance can still provide a meaningful and smooth representation of the distance in-between. This is illustrated in the WGAN paper [39].

The equation of Wasserstein distance is intractable due to the computation of the infimum. The formula is re-formulated using the Kantorovich-Rubinstein duality [64]

$$W(P_{\mathrm{data}}, P_\theta) = \sup_{\|f\|_{\mathrm{L}}\leq 1} \mathbb{E}_{\mathrm{x\sim p_{data}}}[f(x)] - \mathbb{E}_{\mathrm{x\sim p_g}}[f(x)] \tag{3.10}$$

where sup is the least upper bound and f is a 1-Lipschitz function that satisfy $|f(x_1)-f(x_2)| \leq |x_1-x_2|$. So to calculate the Wasserstein distance, we need to find the 1-Lipschitz function. A deep network can be built to learn it but the ensure the Lipschitz constraint weight clipping is needed for the Critic network.

$$\omega \leftarrow clip(\omega, -c, c)$$

Limitations of WGAN

The instability in the training of WGAN comes from the fact that the model performance is very sensitive to weight clipping (ω) hyperparameter [39]. If clipping parameter is small, it can lead to vanishing gradients, and if the clipping parameter is large, then it can take a long time to converge. The clipping measure was chosen for its simplicity in implementation and good performance.

3.4.6 Wasserstein GAN with gradient penalty (WGAN-GP)

Gulrajani et al. introduced WGAN-GP [26]. The weight clipping in the critic was a poor way to ensure 1-Lipschitz continuity. Not only that the model performance is very sensitive to weight clipping hyperparameter, it also reduces the capacity of the model and limits the capability to model complex functions. Thus WGAN-GP uses gradient penalty to ensure the Lipschitz constraint instead of weight clipping. The WGAN-GP penalises the model if the gradient norm moves away from its target norm value of 1. The loss function of the WGAN-GP [26]:

$$L = \underbrace{\mathop{\mathbb{E}}_{\tilde{x}\text{-}\mathrm{P_g}}[D(\tilde{x})] - \mathop{\mathbb{E}}_{\mathrm{x}\text{-}\mathrm{P_r}}[D(x)]}_{OriginalCriticLoss} + \underbrace{\lambda \mathop{\mathbb{E}}_{\hat{x}\text{-}\mathrm{P}_{\hat{x}}}[(\parallel \nabla_{\hat{x}} D(\hat{x}) \parallel_2 - 1)^2]}_{Gradientpenalty} \tag{3.11}$$

where $\hat{x}$ sampled from $\tilde{x}$ and x with k uniformly sampled between 0 and 1. $\hat{x}$=k$\tilde{x}$+(1-k)x with 0$\leq$ k$\leq$ 1. λ is set to 10.

The addition of gradient penalty adds to the computational complexity that may not be desirable, but it still produces some higher quality output and convergence compared to other models.

Chapter 4

Methodology

This chapter presents the chosen methods and approaches to execute the research. To begin with, parking scenario is use case is explained followed by an overview of the workflow of this thesis. Then, the simcenter prescan simulation environment is explained, which is used for the demonstration of the scenario. This is followed by data collection from the simulation environment for dataset creation. The artificial intelligence part of the workflow is explained in chapters 5 and 6

4.1 Parking scenario use case

For this work the scenario chosen for demonstration using GANs is parking lot occupation. The main objective is to train a GAN model to create variations in realistic parking scenario. There are two variants of parking scenario are selected: Non-overlapping parking and overlapping parking scenarios. Non-overlapping parking refers to the case where cars are properly parked within a particular parking slot. For non-overlapping parking scenario eight parking slots are considered out of which four parking slots are on one row and other four parking slots are on the another parallel row. Number of parked cars to be generated is given as label to the generator model. An example non-overlapping parking scenario can be seen in Figure 4.1. Overlapping parking scenario refers to a case when the cars are parked overlapping two adjacent parking slot. To implement overlapping parking scenario four parking slots are considered. Similar to non-overlapping parking scenario, number of parked cars to be generated is given as input to the generator model. An example of overlapping parking scenario is shown in Figure 4.2.

Figure 4.1: Example of non-overlapping parking scenario

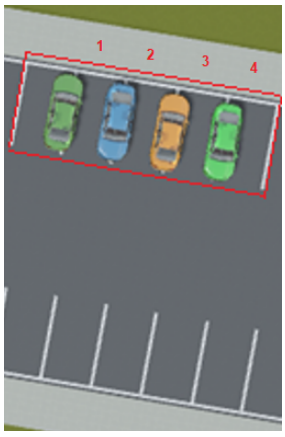


Figure 4.2: Example of overlapping parking scenario

4.2 Overview

Figure 4.3 explains the workflow of the thesis.

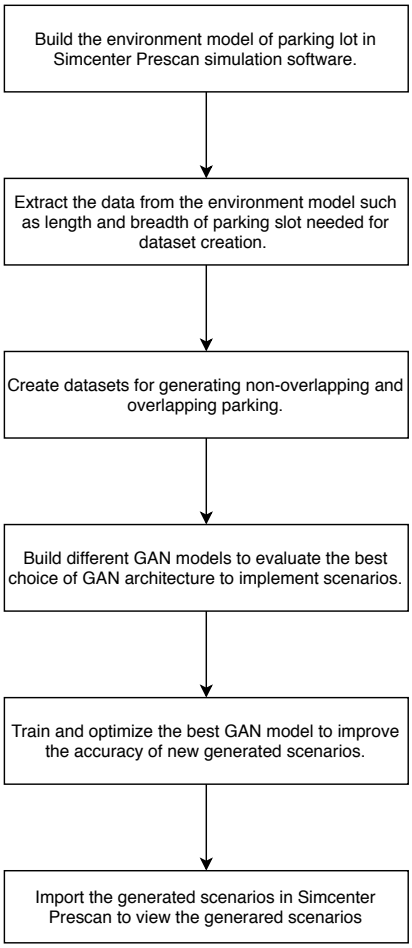


Figure 4.3: Workflow

4.3 Simcenter Prescan simulation environment

The simulation environment on which the demonstration of the scenario performed is Simcenter Prescan. Prescan is a simulation environment for developing ADAS systems. We can use it to validate the ADAS and automated driving functionalities virtually. The Prescan simulation environment works as follows:

1. Build scenario
2. Model sensors
3. Add control systems
4. Run experiment

The first step is replicating the traffic scenario by using the elements of the prescan database such as roads, infrastructure components, actors etc. using either Graphical User Interface (GUI)

or an Application Programming Interface (API). The second step is adding sensor models to ego-vehicle to capture its interaction with the surroundings. Then control systems can be added to design and verify algorithms for data processing, sensor fusion decision making and control. With all these data, we can run experiments. The four stages of prescan are shown in Figure 4.4.

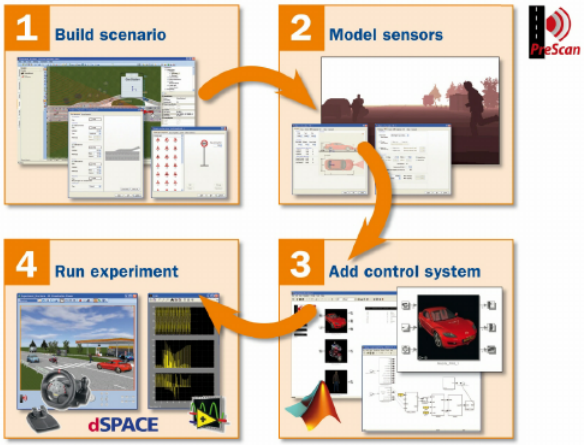


Figure 4.4: The four stages of Prescan [27]

For this project, we are interested in building parking scenario, and for this, we need an environment model. In the Simcenter Prescan software, different library elements are available which include actors (cars, animated humans, trucks, etc.), infrastructural elements (buildings, roads, trees etc.) sensors and others. The necessary elements are chosen and dragged onto the build area to build the environment. The snapshot of the environment model is shown in Figure 4.5. On this built environment, the parking scenario is demonstrated for a particular number of parking slots.

Figure 4.5: Parking lot model in Simcenter Prescan

4.4 Data collection and dataset creation

Since no predefined datasets are available to generate parking scenarios, a relevant dataset has to be generated in order to achieve the goal of the project. Two variants of parked scenarios are intended to be generated. The first variant is parked cars scenario without overlap between two slots and the second variant is cars overlapping the two slots.

When placing the object in the simulation environment, it is the centroid of the object that determines its location. For an actor like the car, it is CoG. Apart from x and y-CoG coordinate, there is also the orientation of the car, which is defined in terms of heading or yaw angle. The parameters of the car are shown in Figure 4.6. The heading angle considered for this project is $90° \pm 5°$ and $270° \pm 5°$. In the parking scenario, the cars must be placed within a slot, and each parking slot has a defined area. The data, such as the length and breadth of a parking slot, are determined from the created simulation environment. x-CoG, y-CoG and heading values of a parking slot are chosen to have a normal distribution. The probability density function of the normal distribution is given by:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} \tag{4.1}$$

where the parameter μ is the mean of the distribution and the parameter σ is the standard distribution.

(a) Center of gravity of a car (b) Heading of a car

Figure 4.6: Parameters of the car actor

For the first variant of the parking scenario without overlap, eight parking slots are considered. To train the GAN a dataset is created. The dataset created is in the tabular form. Each parking slot has four parameters, namely: x-CoG, y-CoG, heading and status. The first three parameters are the same as described previously. The status contains discrete data 0 (to indicate not parked) and 1 (to indicate parked). Each of these four parameters is stacked column-wise. Since there are eight parking slots, there would be 32 columns in total. Additionally, there is one more column added to the dataset, namely class which acts as the label to indicate the number of cars parked. So, in this case, there are three classes. Class '0' indicates $\leq$ two cars parked, Class '1' indicates three to five cars parked, and Class '2' indicates $\geq$ six cars parked. In total, there are 33 columns in the dataset. A snapshot of parameters of parking slots in shown in Figure 4.7. Based on this structure, three different datasets are created labeled as 'Dataset1', 'Dataset2', 'Dataset3' with 1000, 3000 and 5000 rows of data respectively.

For the second variant of the scenario considering overlapping between two slots, four parking slots are considered. Again in this case each parking slot has four parameters as in the case of non-overlapping parking. Like the other case each parameters are stacked columnwise. Since there are four parking slots there would be 16 columns in total. One additional label column is added at the end. For this case only one label is considered indicating $\geq$ three cars parked. For this dataset, labeled as 'Dataset4', 3000 rows of data are taken. In all datasets for each slot, x-CoG, y-CoG and heading are treated as continuous values and status is treated as discrete/categorical value. The summary of datasets are tabulated in Table 4.1:

Figure 4.7: Parameters of parking slots

Dataset	**#Features**	**#D**	**#C**	**#Rows**	**#Classes/labels**
Dataset1	32	8	24	5000	3
Dataset2	32	8	24	3000	3
Dataset3	32	8	24	1000	3
Dataset4	16	4	12	3000	1

Table 4.1: Properties of each dataset. #D indicates the number of discrete/categorical columns, #C the number of continuous columns.

The status parameter of each slot are discrete columns since it is binary with two values 0 or 1. The x-COG, y-COG and heading of each slot are continuous columns since they have a range of values. The units of x-COG and y-COG parameters are in meters (m) and unit of heading is in degree (°).

Chapter 5

Selection of suitable GAN model

In this chapter, a detailed analysis is done on which type of GAN model to be chosen for the final implementation. Three different types of GAN models are considered namely vanilla GAN, CGAN and WGAN-GP. Each model architecture is described and the reasons why each model could or could not be used.

5.1 Dataset for GAN model selection

To determine the best suitable GAN, we create two datasets. The first dataset is used to train vanilla GAN and CGAN. This dataset has two parameters x-CoG and y-CoG (and additional label added to data for CGAN). This dataset is created to achieve the generation of a good distribution of parking location for four slots.

The second dataset is used to train CGAN and WGAN-GP. The dataset is created for four slots. For each slot, three parameters are used namely x-CoG, y-CoG and status. Each of these values is stacked column-wise resulting is a tabular dataset. A class label is added to the dataset to indicate three or more cars parked. After the dataset is created, normalisation of the data is done in the range of [-1,1] as a data preprocessing step. The goal of the normalisation is to change the values of the features in the dataset to a common scale without distorting differences in the range of values. The main reason for normalisation is that gradient descent [70] converges much faster, resulting in faster training. Min-max normalisation, one of the most common ways to normalise data, is used. For each feature, x-COG, y-COG and status, the minimum value of that feature gets transformed into -1, and maximum value of that feature gets transformed into +1, and every other value lies in the range of -1 and 1.

5.2 Model using vanilla GAN

The first step before training a Generative Adversarial Network is to define the generator and discriminator networks. The main idea is to define the simplest neural network architectures for both generator and discriminator networks. Both the generator and discriminator networks are simple feedforward networks with an input layer, 3-4 hidden layers and an output layer. Since the dataset lacks any spatial structure amongst the variables, densely connected layers are used instead of convolutional layers. The neurons in densely connected layers are connected to every input and output of the layer, allowing the network to learn its own relationships among the features. For the discriminator network, LeakyReLU activation function [40], [68], [7] is used on hidden layers. For the generator network, ReLU activation function [49], [7] is used on hidden layers. These activation functions are chosen as suggested in the GAN literature [54]. The output of the generator layer uses tanh activation function [10] and the output of the discriminator network uses sigmoid activation function [10]. The sigmoid function is used to classify the image as real or

fake. The architecture of the generator and discriminator networks are shown in Tables 5.1 and 5.2, respectively.

Layer	Layer Type	Output dimensions	Description
Input Layer	Input	(None,2)	Latent vector (None,2)
Hidden Layer 1	Dense	(None,30)	Dense layer with 30 neurons followed by ReLU activation function
Hidden Layer 2	Dense	(None,15)	Dense layer with 15 neurons followed by ReLU activation function
Hidden Layer 3	Dense	(None,5)	Dense layer with 5 neurons followed by ReLU activation function
Output Layer	Dense	(None,2)	Dense layer with 2 neurons followed by tanh activation function

Table 5.1: Architecture of the generator network of vanilla GAN. Total number of trainable parameters in generator network are 647. Architecture diagram is shown in Appendix A.1

Layer	Layer Type	Output dimensions	Description
Input Layer	Input	(None,2)	Input data (None,2)
Hidden Layer 1	Dense	(None,50)	Dense layer with 50 neurons followed by LeakyReLU activation function
Hidden Layer 2	Dense	(None,25)	Dense layer with 25 neurons followed by LeakyReLU activation function
Hidden Layer 3	Dense	(None,10)	Dense layer with 10 neurons followed by LeakyReLU activation function
Hidden Layer 4	Dense	(None,5)	Dense layer with 5 neurons followed by LeakyReLU activation function
Output Layer	Dense	(None,1)	Dense layer with 1 neuron followed by sigmoid activation function

Table 5.2: Architecture of the discriminator network of vanilla GAN. Total number of trainable parameters in discriminator network are 1746. Architecture diagram is shown in Appendix A.2

Training

The code implemented for building the GAN framework and training the model is in python language with the Keras and TensorFlow libraries. Basically, once the generator and discriminator networksgra are defined, the training has to be done until the generator learns the distribution similar to that of the real dataset.

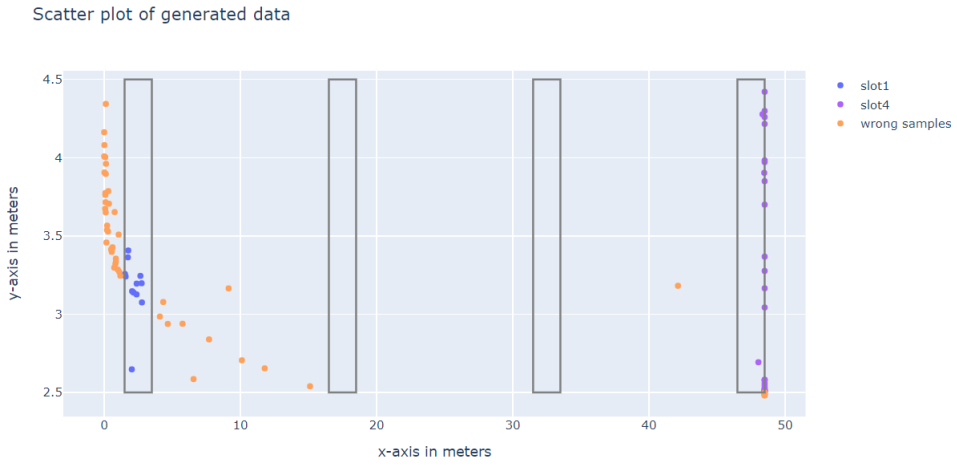
The Algorithm 1 describes the training procedure of vanilla GAN [25]. For this research, the models are trained for 1000 epochs, and the algorithm is designed to save the model of the weight after every 200 epochs and losses after every epoch.

Algorithm 1: Training of GAN with stochastic gradient descent [57]. k is the number of times the discriminator network is updated for each generator update. n is the minibatch size [57], and w and θ are parameters of discriminator D and generator G respectively. In this algorithm, Adam optimizer [34] is used with hyperparameters: learning rate α = 0.0015, exponential decay rate for first moment β_1=0, exponential decay rate for second moment β_2=0.9

for *number of training iterations* **do**
  for *k steps* **do**
    Sample a minibatch of n noise samples z $\sim p_z$
    Sample a minibatch of n real data samples x $\sim p_{data}$
    Update the discriminator:
    w$\leftarrow Adam(\nabla_w \frac{1}{n} \sum_{i=1}^{n} -logD(x) - log(1 - D(G(z))))$
  end
  Sample a minibatch of n noise samples z $\sim p_z$
  Update the generator:
  $\theta \leftarrow Adam(\nabla_g \frac{1}{n} \sum_{i=1}^{n} log(1 - D(G(z))))$
end

Results

After the training is complete, the saved weights of the generator model are loaded to generate new parking locations. Initial results show that the model suffered from a phenomenon called mode collapse. Mode collapse is a phenomenon where the generator produces only a subset of all possible modes of data as described in 3.4.3. This can be seen in Figure 5.1. From the 100 generated points, 35 points generated are in slot 4, and 12 are in slot 1 and the remaining 53 points are not in any of the four slots. Also, there are no points in slot 2 and slot 3. It shows that the generator model has not completely learnt the distribution of real data. Since there are no points generated in slot2 and slot3 and most points are in slot4 we say that mode collapse phenomenon has occurred.




```
Number in first slot: 12
Number in second slot: 0
Number in third slot: 0
Number in fourth slot: 35
Wrong samples: 53
47
```

Figure 5.1: Scatter plot showing the results of generation of coordinates from trained vanilla GAN model for four parking slots

5.3 Model using CGAN

CGAN is a variant of vanilla GAN where new data generation can be conditioned on a class label. In vanilla GAN, there is no control over the modes of data to be generated. Since strong mode collapse was observed in the results using vanilla GAN model, an additional parameter (label/class) is fed to the generator to generate the samples corresponding to the label. The labels are also added to the discriminator input to distinguish the real data better. The only difference between the vanilla GAN and CGAN architecture is that additional label inputs to both generator and the discriminator model. Rest of the model architecture remains the same. The model is trained to the loss function as described in 3.5. The architecture of the generator and discriminator networks are shown in Tables 5.3 and 5.4, respectively.

Layer	Layer Type	Output dimensions	Description
Input Layer	Concatenate	(None,3)	Concatenation of latent vector (None,2) + label (None,1)
Hidden Layer 1	Dense	(None,30)	Dense layer with 30 neurons followed by ReLU activation function
Hidden Layer 2	Dense	(None,15)	Dense layer with 15 neurons followed by ReLU activation function
Hidden Layer 3	Dense	(None,5)	Dense layer with 5 neurons followed by ReLU activation function
Output Layer	Dense	(None,2)	Dense layer with 2 neurons followed by tanh activation function

Table 5.3: Architecture of the generator network of CGAN. Total number of trainable parameters in generator network are 677. Architecture diagram is shown in Appendix A.4

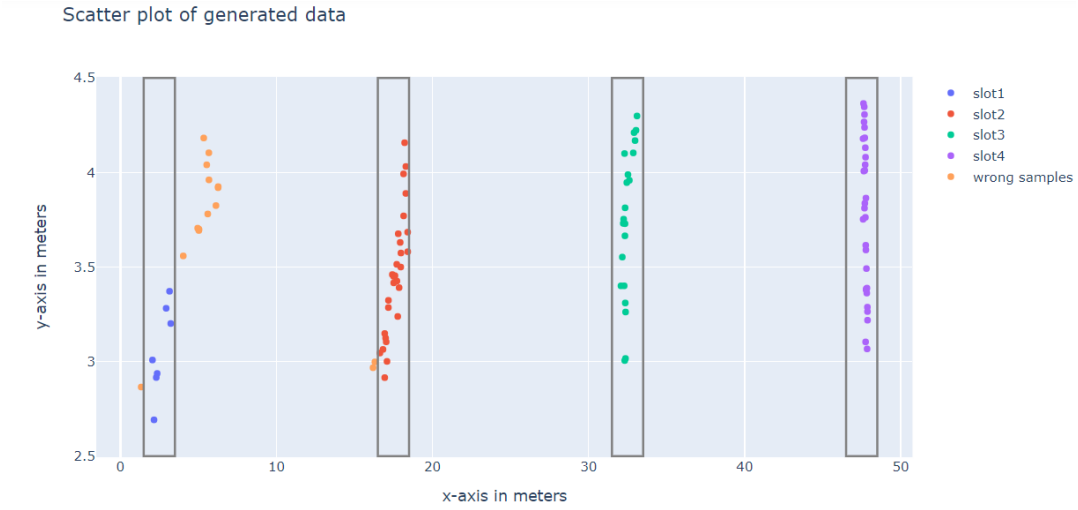
Layer	Layer Type	Output dimensions	Description
Input Layer	Concatenate	(None,3)	Concatenation of input data (None,2) + label (None,1)
Hidden Layer 1	Dense	(None,50)	Dense layer with 50 neurons followed by LeakyReLU activation function
Hidden Layer 2	Dense	(None,25)	Dense layer with 25 neurons followed by LeakyReLU activation function
Hidden Layer 3	Dense	(None,10)	Dense layer with 10 neurons followed by LeakyReLU activation function
Hidden Layer 4	Dense	(None,5)	Dense layer with 5 neurons followed by LeakyReLU activation function
Output Layer	Dense	(None,1)	Dense layer with 1 neuron followed by sigmoid activation function

Table 5.4: Architecture of the discriminator network of CGAN. Total number of trainable parameters in discriminator network are 1796. Architecture diagram is shown in Appendix A.3

Results

After training is complete, the generator model is used to generate new parking locations. Additional information in the form of class labels resulted in improvement in GAN in the form of more

stable and faster training and eliminated mode collapse to a large extent. This can be seen in Figure 5.2. From the plot, we can see that now the CGAN model is able to generate coordinates in all the four slots with most of the points falling within the appropriate designated slots with few points (represented by orange points in figure) falling outside the range of a slot indicating wrong samples.




```
Number in first slot: 7
Number in second slot: 28
Number in third slot: 21
Number in fourth slot: 29
Wrong samples: 15
85
```

Figure 5.2: Scatter plot showing the results of generation of coordinates from trained CGAN model for four parking slots

Next, the model is trained with the second dataset, which includes the status of the parking for each slot. On implementing this with CGAN, it was observed that some of the intra modes within a class were not captured by the model. In other words, the mode collapse issue was not entirely solved by using CGAN. This can be seen in figure 5.3. This plot shows the generation of 10 parking scenarios for three or more cars parked. With four slots, there could be four possible combinations which can indicate three or more cars parked amongst those four slots. The matrix below the plot indicates the status of the slot where 1 indicates parked, and 0 indicates free slot. There are ten rows of data in the matrix indicating the status for ten scenarios. From the matrix, we see that only one combination of 3 or more cars parked is being generated where cars would be parked in slots 1, 2 and 4 and no car on slot 3. However, the other three possible combinations are missing indicating mode collapse.

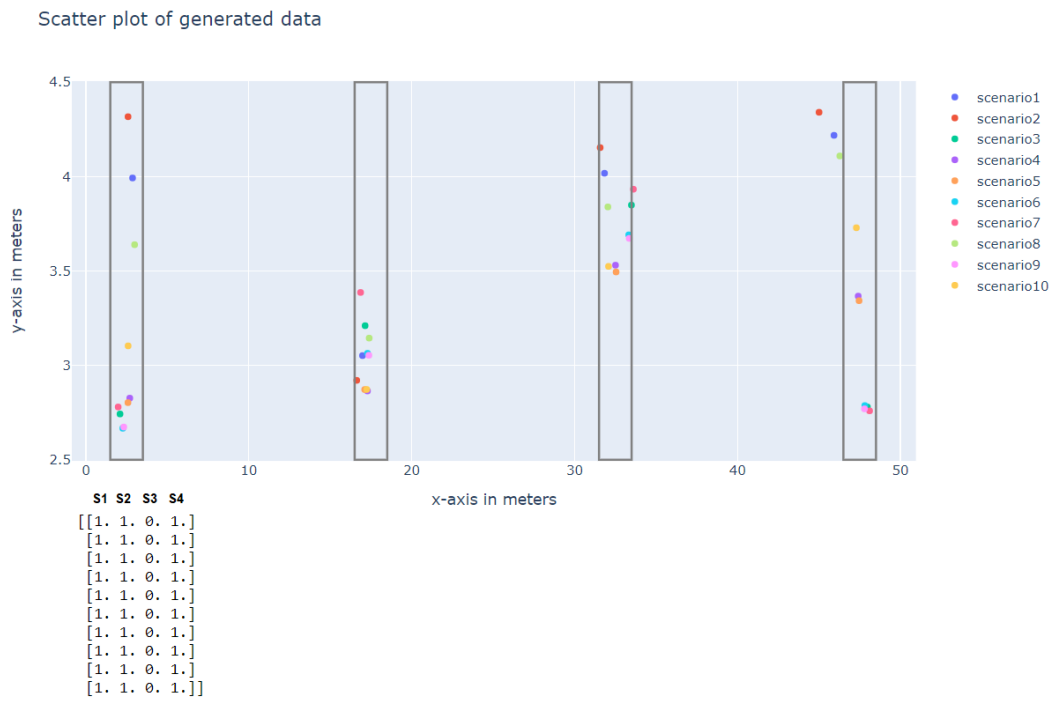


Figure 5.3: Scatter plot of 10 generated scenarios indicating greater than three or more cars parked using CGAN model for four parking slots

5.4 Model using WGAN-GP

WGAN-GP is a variant of GAN that uses Wasserstein distance with gradient penalty as described in 3.4.6, rather than JS-divergence, to measure the difference between the model and the target distribution. Using this metric, the drawbacks of vanilla GAN such as slow training and mode collapse problems are overcome. Architectures are defined for the critic/discriminator and the generator networks. Both the critic and the generator are a simple feedforward network with an input layer, three hidden layers and an output layer. All the hidden layers are dense/fully connected layers. The activation functions of hidden layers of the critic network are ReLU activation, and the activation function of the hidden layers of the generator network is the LeakyReLU activation function. On the output layer, the activation function on the critic network is linear activation, and the activation function on the generator function is tanh activation. The architecture of the generator and discriminator networks are shown in 5.5 and 5.6, respectively.

5.4.1 Training

The code implemented for building the WGAN-GP framework and training the model is in python language with the Keras and TensorFlow libraries. After the generator and discriminator models are defined, the training has to be done until the generator learns the distribution similar to that of the real dataset.

The Algorithm 2 describes the training procedure of WGAN-GP [26]. For this research, the models are trained for 10000 epochs, and the algorithm is designed to save the model weights after every 2000 epochs and losses after every epoch to check the intermediate results as well.

Layer	Layer Type	Output dimensions	Description
Input Layer	Concatenate	(None,21)	Concatenation of latent vector (None,20) + label (None,1)
Hidden Layer 1	Dense	(None,128)	Dense layer with 128 neurons followed by ReLU activation function
Hidden Layer 2	Dense	(None,64)	Dense layer with 64 neurons followed by ReLU activation function
Hidden Layer 3	Dense	(None,32)	Dense layer with 32 neurons followed by ReLU activation function
Output Layer	Dense	(None,12)	Dense layer with 12 neurons followed by tanh activation function

Table 5.5: Architecture of the generator network of WGAN-GP. Total number of trainable parameters in the generator network are 13548. Architecture diagram is shown in Appendix A.6

Layer	Layer Type	Output dimensions	Description
Input Layer	Concatenate	(None,13)	Concatenation of input data (None,12) + label (None,1)
Hidden Layer 1	Dense	(None,128)	Dense layer with 128 neurons followed by LeakyReLU activation function
Hidden Layer 2	Dense	(None,64)	Dense layer with 64 neurons followed by LeakyReLU activation function
Hidden Layer 3	Dense	(None,32)	Dense layer with 32 neurons followed by LeakyReLU activation function
Output Layer	Dense	(None,1)	Dense layer with 1 neuron followed by linear activation function

Table 5.6: Architecture of the discriminator network of WGAN-GP. Total number of trainable parameters in the discriminator network are 12161. Architecture diagram is shown in Appendix A.5

Algorithm 2: WGAN with gradient penalty. Gradient penalty $\lambda = 10$, number of critic iterations per generator iteration $n_{critic} = 5$. In this algorithm, Adam optimizer [34] is used with hyperparameters: learning rate $\alpha = 0.0015$, exponential decay rate for first moment β_1=0, exponential decay rate for second moment β_2=0.9

while θ *has not converged* **do**
  for j=*1 to* n_{critic} **do**
    for i=*1 to* m **do**
      Sample real data sample x $\sim p_{\text{data}}$
      Sample noise samples z $\sim p_{\text{z}}$
      Sample a random number $\epsilon \sim U[0, 1]$
      $\tilde{\text{x}} \leftarrow G(z)$
      $\hat{\text{x}} \leftarrow \epsilon \text{x} + (1\text{-}\epsilon)\tilde{\text{x}}$
      $\text{L}_D^{(i)} \leftarrow D_{\text{w}}(\tilde{x}) - D_{\text{w}}(x) + \lambda(\| \nabla_{\hat{x}} D_{\text{w}}(\hat{x}) \|_2 - 1)^2$
    end
    $\text{w} \leftarrow Adam(\nabla_{\text{w}} \frac{1}{m} \sum_{i=1}^{m} L^{(i)})$
  end
  Sample a batch of m noise samples of z $\sim p_{\text{z}}$
  $\theta \leftarrow Adam(\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} -D_{\text{w}}(G_\theta(z)))$
end

5.4.2 Results

The plot in the Figure 5.4 shows the generation of 10 parking scenarios for three or more cars parked. From the Figure 5.4, it can be seen that for class label 3, the model can generate all possible combinations of 3 cars parked across four parking slots.

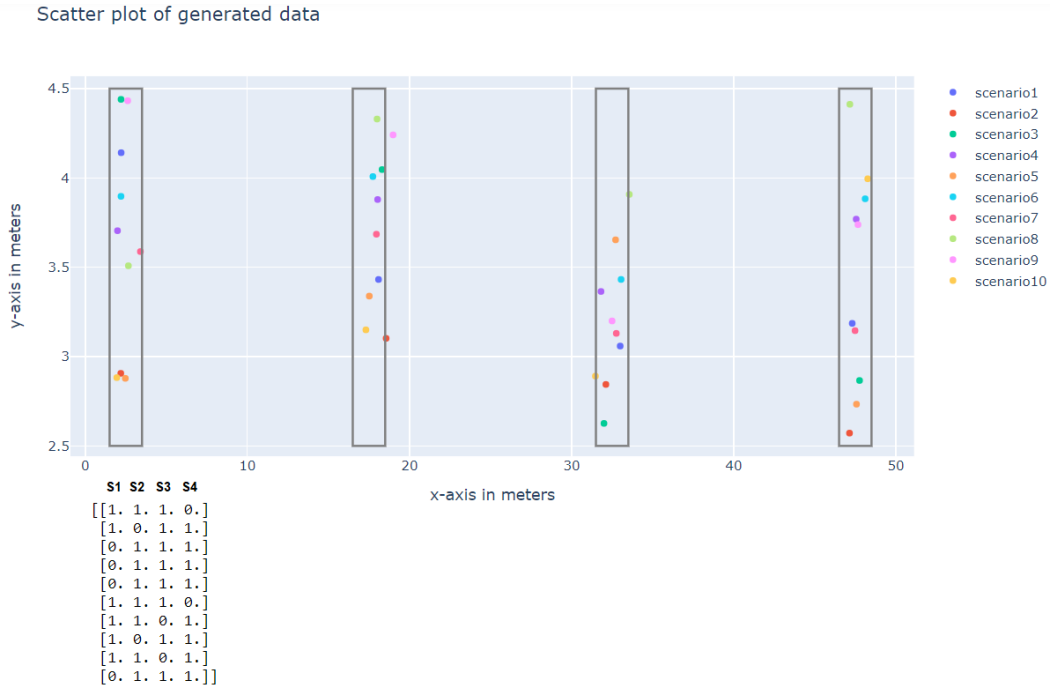


Figure 5.4: Scatter plot showing the results of generation of coordinates from trained WGAN-GP model for four parking slots with status information

5.5 Conclusion

Based on experiment results we can conclude that vanilla GAN and CGAN suffers from mode collapse problem wherein only a subset of all modes of scenarios are being generated. However, when we use WGAN-GP with gradient penalty, the mode collapse issue is no longer observed, and thus it is the best choice of the model. Further implementations are carried out using WGAN-GP model.

Chapter 6

Optimization of WGAN-GP model

This chapter describes the architectural details of final model designed along with the hyperparameters chosen to implement both the variants of parking scenario (Non-overlapping and overlapping parking). The final model is based on WGAN-GP 3.4.6 as concluded in section 5.

6.1 Model architecture for non-overlapping scenario for eight parking slots

6.1.1 Critic network

The architecture of the critic network is shown in Table 6.1. The critic network is a simple feedforward network. The input to the critic network is concatenation of data (real or generated) and a label parameter. The network has four hidden layers. All are followed by LeakyReLU activation function and dropout layer. Firstly dense/fully connected layers are in hidden layers. This is because since the dataset lacks spatial structure amongst its variables. The neurons in densely connected layers are connected to every input and output of the layer, allowing the network to learn its own relationships amongst the features. The leakyReLU activation function is used for hidden layers of critic network because it helps the gradient to flow easier through the network architecture. The term dropout refers to dropping of neurons in a neural network layer at random during the training phase. The dropout layer is used to prevent the model from overfitting and improve generalisation error. This is very computationally cheap and effective regularisation technique used. The output of the critic network has one neuron with linear activation function. Total trainable parameters in critic model are 51969.

6.1.2 Generator network

The architecture of the generator network is shown in Table 6.2. Similar to the critic network, the generator network is a simple feedforward network. The input to generator network is a random vector z (called latent/noise vector) and a label. The network has four hidden layers. All hidden layers are followed by ReLU activation and dropout layer. Similar to critic network dense layers are used for the same reasons as in critic network. ReLU activation is used because they seem to work better on generator network [54]. The dropout layer is used to prevent the model from overfitting and improve generalisation error. The output of the generator network has 32 neurons with tanh activation function. Total trainable parameters in generator model are 53344.

Layer	Layer Type	Output dimensions	Description
Input Layer	Concatenate	(None,33)	Concatenation of input data (None,32) + label (None,1)
Hidden Layer 1	Dense	(None,256)	Dense layer with 256 neurons followed by LeakyReLU activation function and dropout layer
Hidden Layer 2	Dense	(None,128)	Dense layer with 128 neurons followed by LeakyReLU activation function and dropout layer
Hidden Layer 3	Dense	(None,64)	Dense layer with 64 neurons followed by LeakyReLU activation function and dropout layer
Hidden Layer 4	Dense	(None,32)	Dense layer with 32 neurons followed by LeakyReLU activation function and dropout layer
Output Layer	Dense	(None,1)	Dense layer with 1 neuron followed by linear activation function

Table 6.1: Architecture of the critic network for non-overlapping parking scenario. Total number of trainable parameters in the critic model are 51969. The Figure is shown in 6.1

Layer	Layer Type	Output dimensions	Description
Input Layer	Concatenate	(None,51)	Concatenation of latent vector (None,50) + label (None,1)
Hidden Layer 1	Dense	(None,32)	Dense layer with 32 neurons followed by ReLU activation function and dropout layer
Hidden Layer 2	Dense	(None,64)	Dense layer with 64 neurons followed by ReLU activation function and dropout layer
Hidden Layer 3	Dense	(None,128)	Dense layer with 128 neurons followed by ReLU activation function and dropout layer
Hidden Layer 4	Dense	(None,256)	Dense layer with 256 neurons followed by ReLU activation function and dropout layer
Output Layer	Dense	(None,32)	Dense layer with 32 neurons followed by tanh activation function

Table 6.2: Architecture of the generator network for non-overlapping parking scenario. Total trainable parameters in generator model are 53344. The Figure is shown in 6.2.

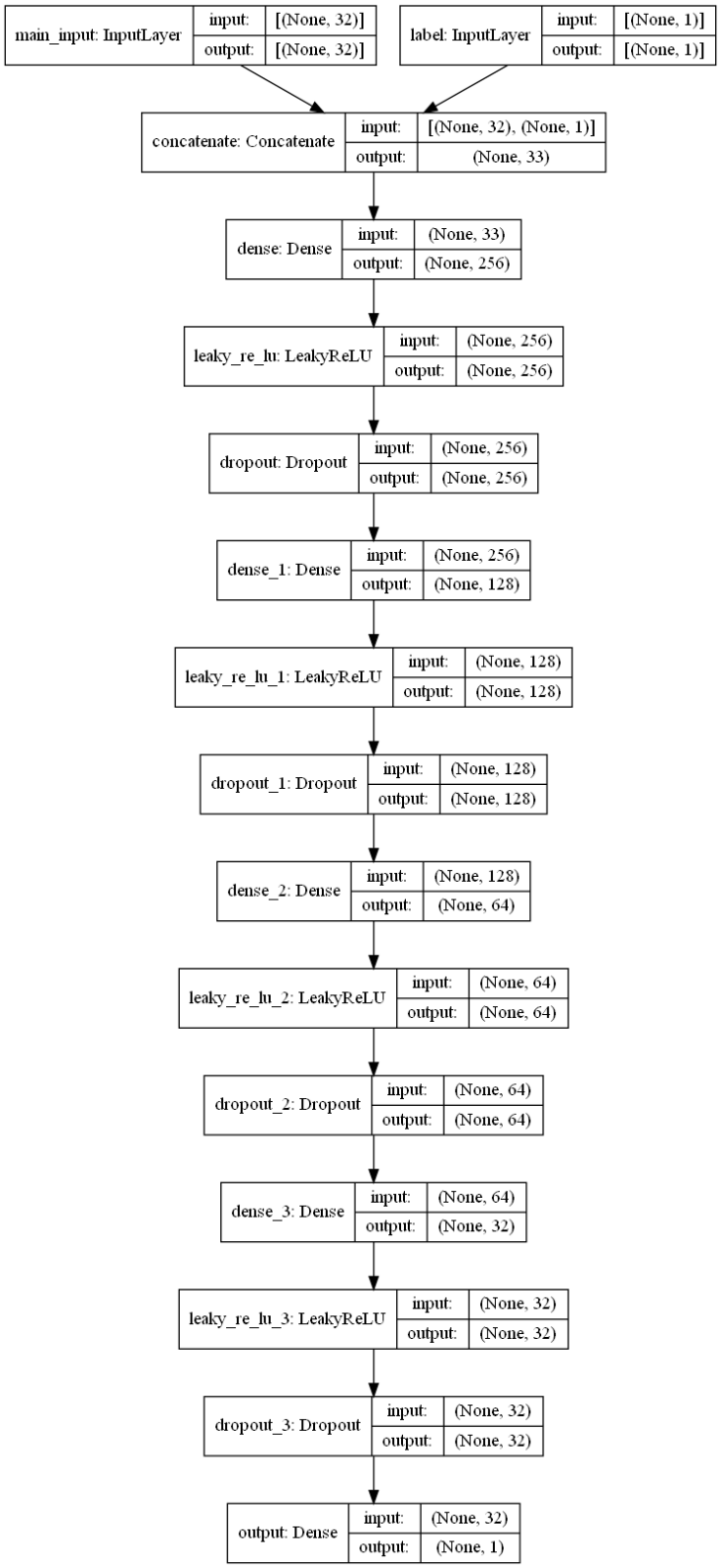


Figure 6.1: Architecture of the critic network for non-overlapping parking scenario.

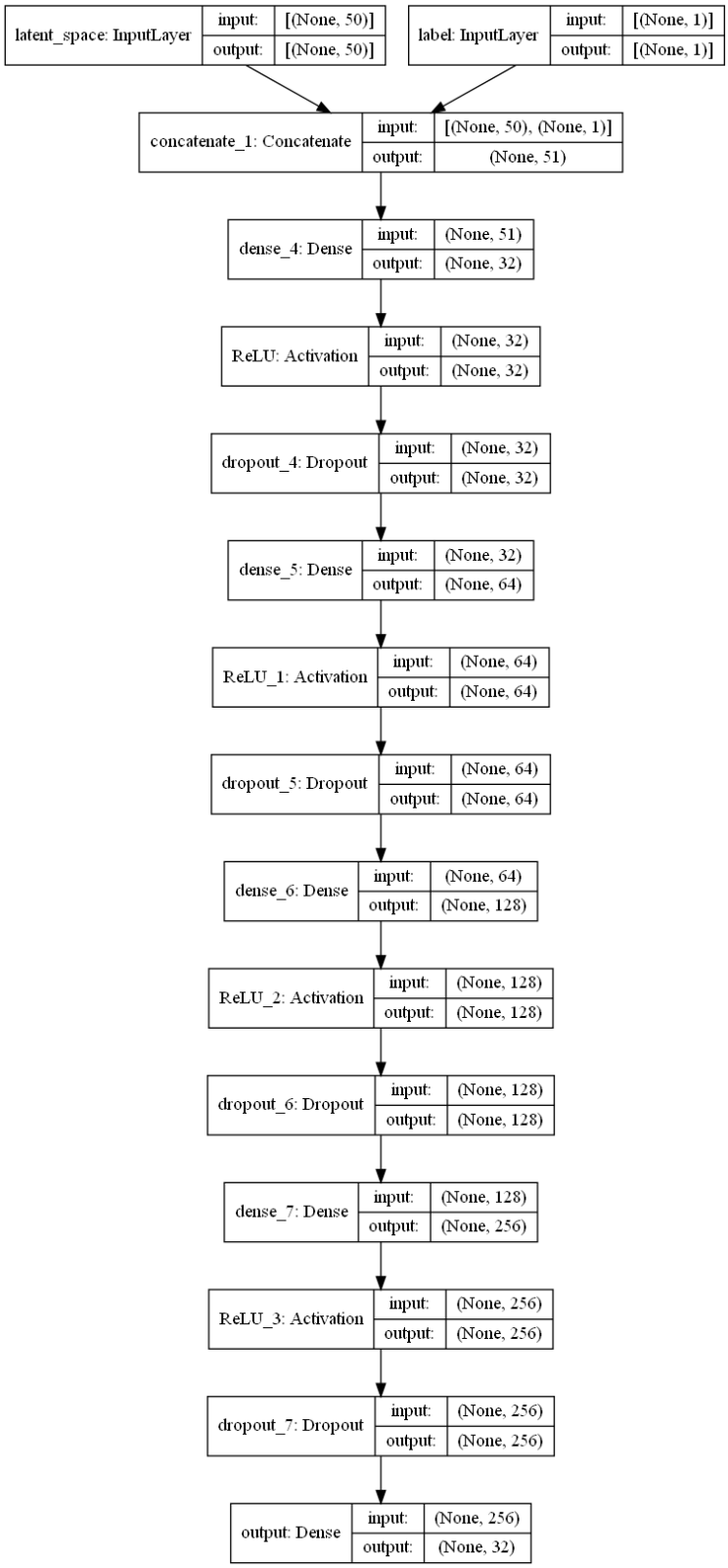


Figure 6.2: Architecture of the generator network for non-overlapping parking scenario.

6.1.3 Model hyperparameters

After defining architectures of the models used, the hyperparamters chosen for training process are defined. A hyperparameter is a parameter whose values is set before the training of the model begins. All the hyperparamters of the models are tabulated in Table 6.3. Some of the hyperparameters are chosen by performing several iterations of training experiments and some of the hyperparameters are same as in literature [26]. n_{critic} and λ were chosen from WGAN-GP literature and the values were chosen to be same as it was used in original paper. Rest of the hyperparameters were experimentally decided. Experiments were conducted to decide the values of hyperparameters and the ones which were giving high accuracy was chosen to be the final value.

Hyperparameter	Value	Description
Batch Size	512	Number of samples processed before the model parameters (weights and biases) are updated
Learning rate for generator Network	0.0015	Tuning parameter that determines the step size at each iteration while moving towards a minimum of a loss function
Learning rate for critic Network	0.0015	Tuning parameter that determines the step size at each iteration while moving towards a minimum of a loss function
Number of epochs	50000	Number of complete passes through the entire training dataset.
Latent dimensions	50	Size of the latent space
Dropout rate	0.2	Fraction of the input units to drop
n_{critic}	5	Number of critic iterations per generator iteration
λ	10	Gradient penalty lambda hyperparameter
Optimizer	Adam optimizer	Adam optimizer [34] with default parameters

Table 6.3: Hyperparameters chosen for the models for non-overlapping parking scenario

6.1.4 Training and new scenario generation

The WGAN-GP model is trained using the Algorithm 2 for 50000 epochs. The datasets used for training of this model are Dataset1, Dataset2 and Dataset3 as described in section 4.1. The model is trained separately on each of the three datasets. The batch size is set to 512 samples. The weights of the critic and generator network are updated using Adam optimizer [34]. For every 5 updates of the critic network, the generator network is updated once. Once the training is complete the trained generator model is used to generate new scenarios.

After training, firstly the weights trained generator network is loaded. Then we feed labels and number of new scenarios to be generated by the model. In this case we generate 1000 new scenarios. After the new scenarios are generated, they are firstly denormalised of the generated data is performed. This is done such that generated data are available in expected original form. Once the generated data is denormalised they can be used for further analysis. Each of the 1000 scenarios are then visualised on the Simcenter Prescan simulation environment and evaluation is performed on those 1000 scenarios. The results of the generated scenarios are shown in chapter 7.

6.2 Model architecture for overlapping parking scenario for four parking slots

The overlapping parking scenario is generated considering four parking slots. The architectures of the critic and generator networks are shown in Tables 6.4 and 6.5. The main difference in architecture of critic and generator compared to non-overlapping scenario is that it has 1 less hidden layer since only four slots are considered instead of eight slots in case of non-overlapping scenario. And since only four slots are considered the output neurons in generator model is 16 neurons instead of 32 neurons in non-overlapping scenario. Rest of the architecture is similar to non-overlapping scenario.

Layer	Layer Type	Output dimensions	Description
Input Layer	Concatenate	(None,17)	Concatenation of input data (None,16) + label (None,1)
Hidden Layer 1	Dense	(None,128)	Dense layer with 128 neurons followed by LeakyReLU activation function and dropout layer
Hidden Layer 2	Dense	(None,64)	Dense layer with 64 neurons followed by LeakyReLU activation function and dropout layer
Hidden Layer 3	Dense	(None,32)	Dense layer with 32 neurons followed by LeakyReLU activation function and dropout layer
Output Layer	Dense	(None,1)	Dense layer with 1 neuron followed by linear activation function

Table 6.4: Architecture of the critic network for overlapping parking scenario. Total number of trainable parameters in the critic network are 12673. The Figure is shown in 6.3

Layer	Layer Type	Output dimensions	Description
Input Layer	Concatenate	(None,51)	Concatenation of latent vector (None,50) + label (None,1)
Hidden Layer 1	Dense	(None,32)	Dense layer with 32 neurons followed by ReLU activation function and dropout layer
Hidden Layer 2	Dense	(None,64)	Dense layer with 64 neurons followed by ReLU activation function and dropout layer
Hidden Layer 3	Dense	(None,128)	Dense layer with 128 neurons followed by ReLU activation function and dropout layer
Output Layer	Dense	(None,16)	Dense layer with 16 neurons followed by tanh activation function

Table 6.5: Architecture of the generator network for overlapping parking scenario. Total number of trainable parameters in the generator network are 14160. The Figure is shown in 6.4

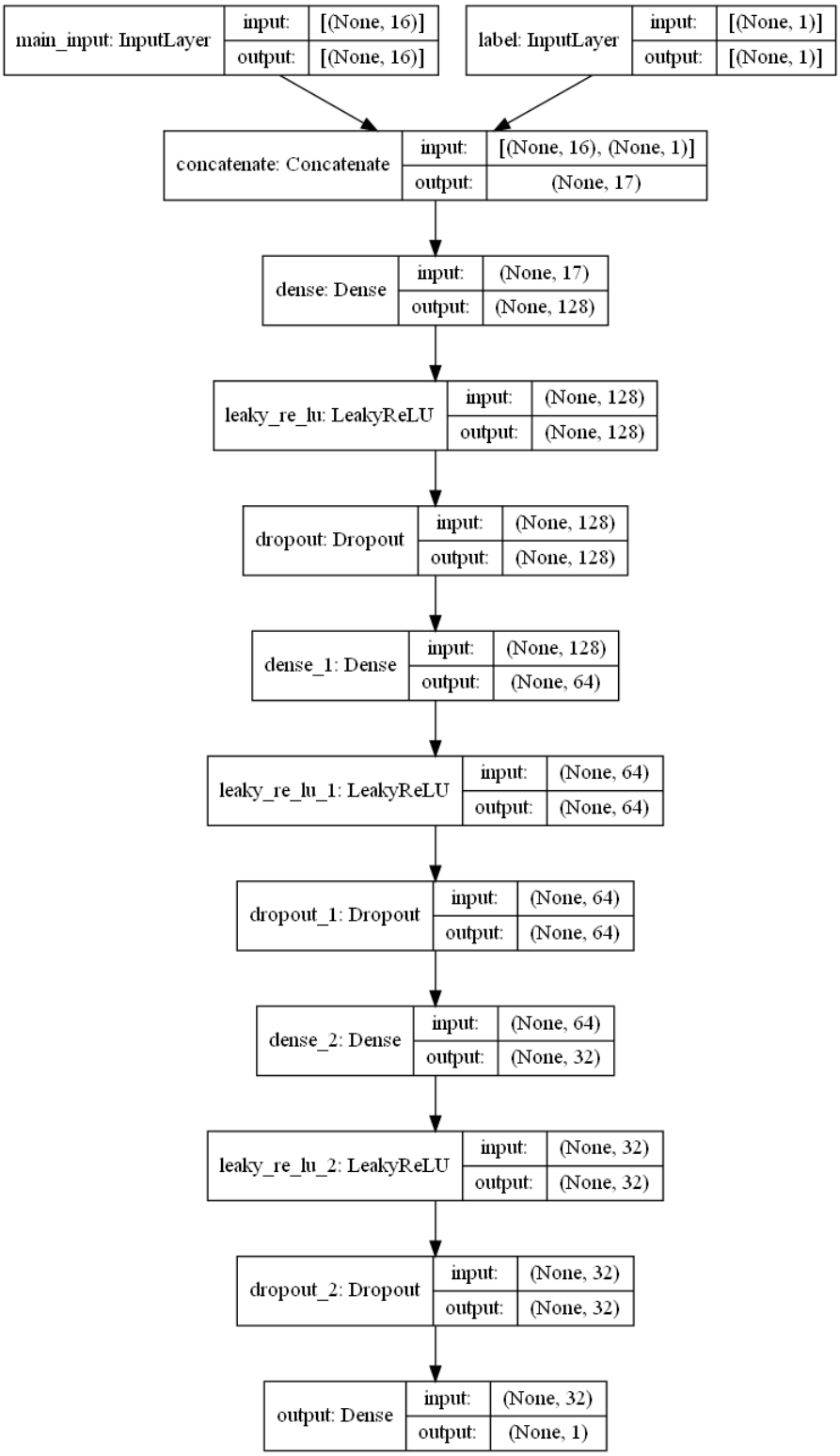


Figure 6.3: Architecture of the critic network for overlapping parking scenario.

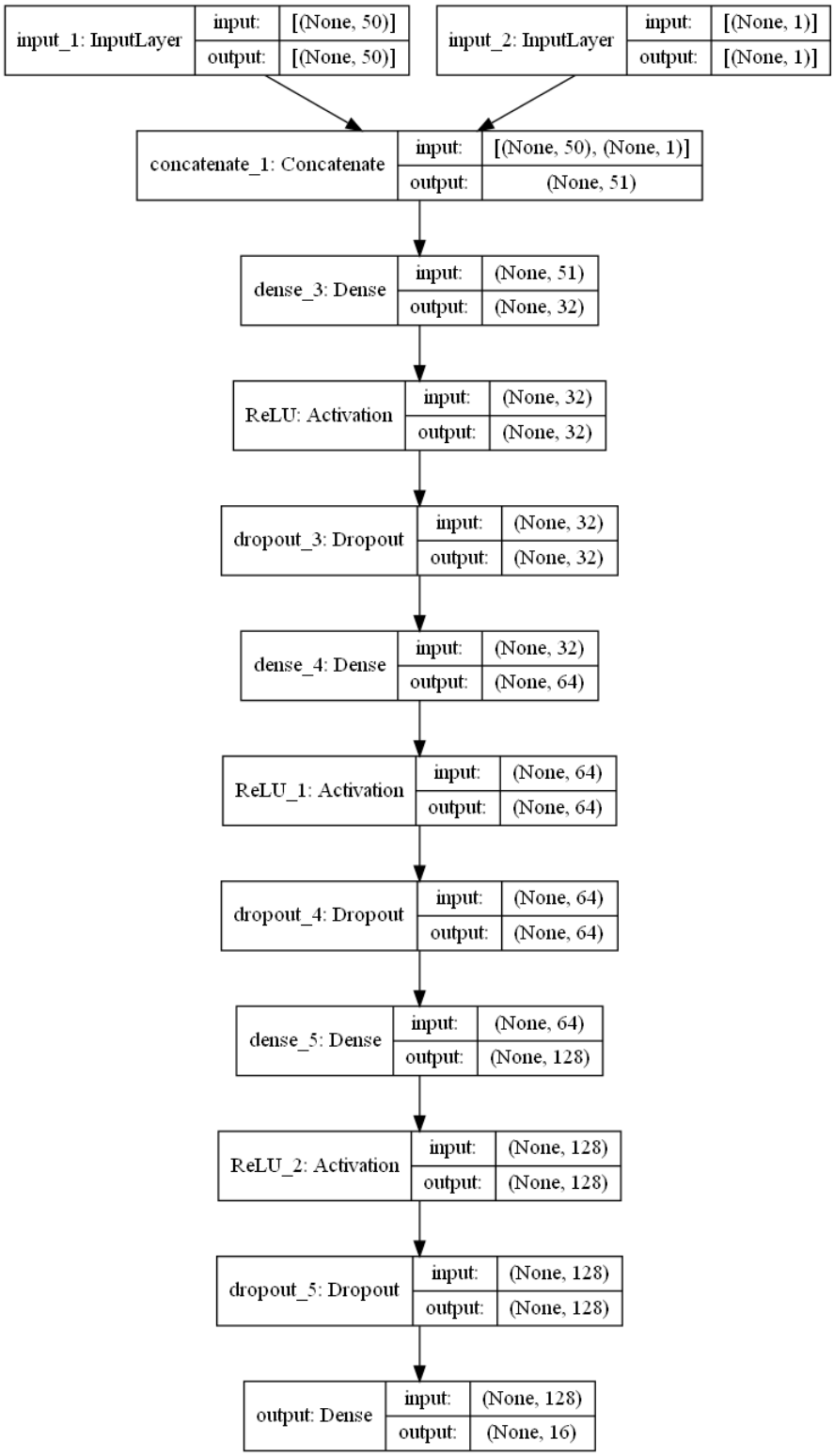


Figure 6.4: Architecture of the generator network for overlapping parking scenario.

6.2.1 Training and new scenario generation

The hyperparameters for the models defined for overlapping scenarios are shown in Table 6.6. The WGAN-GP model is trained using the algorithm 2 for 50000 epochs. The dataset used for training of this model is Dataset4 as described in section 4.1. The batch size is set to 512 samples. The weights of the critic and generator network are updated using Adam optimizer [34]. For every 5 updates of the critic network, the generator network is updated once. Once the training is complete the trained generator model is used to generate new scenarios.

After training, firstly the weights trained generator network is loaded. Then we feed labels and number of new scenarios to be generated by the model. In this case we generate 1000 new scenarios. After the new scenarios are generated, they are firstly denormalised of the generated data is performed. This is done such that generated data are available in original expected form. Once the generated data is denormalised they can be used for further analysis. Each of the 1000 scenarios are then visualised on the Simcenter Prescan simulation environment and evaluation is performed on those 1000 scenarios. The results of the new generated scenarios are shown in chapter 7.

Hyperparameter	**Value**	**Description**
Batch Size	512	Number of samples processed before the model parameters (weights and biases) are updated
Learning rate for generator Network	0.0015	Tuning parameter that determines the step size at each iteration while moving towards a minimum of a loss function
Learning rate for critic Network	0.0015	Tuning parameter that determines the step size at each iteration while moving towards a minimum of a loss function
Number of epochs	50000	Number of complete passes through the entire training dataset.
Latent dimensions	50	Size of the latent space
Dropout rate	0.2	Fraction of the input units to drop
n_{critic}	5	Number of critic iterations per generator iteration
λ	10	Gradient penalty lambda hyperparameter
Optimizer	Adam optimizer	Adam optimizer [34] with default parameters

Table 6.6: Hyperparameters chosen for the models for overlapping parking scenario

Chapter 7

Evaluation and results

This chapter outlines the different evaluation methods that have been used to evaluate the performance of generated scenarios by the WGAN-GP model as discussed in chapter 6. Also, the results of the WGAN-GP models to implement overlapping and non-overlapping parking scenarios are presented.

7.1 Evaluation metrics

When evaluating the generated output, we check for two aspects, the diversity and the quality. From the diversity perspective, we check the ability of the generator to produce diverse samples capturing different modes in the real data and from the quality viewpoint, we evaluate the generator's ability to generate realistic samples. Most widely used evaluation metrics are inception score [13] and FID score [28]. The inception score metric gives a score to the generated output image that measures how realistic the GAN's output is. Higher the inception score better the generated output. Frechet Inception Distance measures the Wasserstein-2 distance [66] between feature vectors calculated for real and generated images. For FID score, the lower score correlate to a high quality of images generated. These scores can be used only on image datasets and since tabular dataset is used for this work, we cannot use these standard evaluation procedures.

7.1.1 Visual inspection

For this work, we visually inspect the similarity between the distributions of each column (attribute) between the generated and real data. We plot the histogram and cumulative percentage of each column of real and generated data on top of each other in order to compare them. By plotting them, we can gain some insight into a column distribution, and this works for both categorical and continuous columns.

Secondly, after the new scenarios are generated, they are imported on the prescan environment using MATLAB where they can be observed visually on the environment. From this, we can quite evidently evaluate the generated scenarios. We evaluate generated scenarios both in terms of the diversity and the quality aspect. The variations in generated scenarios are easily detectable, and thus we can examine the ability of the GAN to generate diverse samples. Also, the content of images is relatively simple and easy to compare. Therefore, we can evaluate the GAN's ability to generate high-quality sample by comparing them to ground-truth data.

7.1.2 Accuracy of the generated scenarios

Another evaluation metric used to assess the generated scenarios is in terms of accuracy. This metric is used only for non-overlapping scenario generation. In order to determine the accuracy of the model, we first generate 'n' scenarios from trained GAN model. After the scenarios are

generated a scatter plot between the x-CoG and y-CoG is plotted, which indicate where they are lying within each parking slot. For example, Figure 7.1 shows an example of 3 different parking scenarios across four slots using a scatter plot. Each scenario has 4 coordinates points representing x-CoG and y-CoG of the car in each of the 4 slots. From this figure, we can see that there are 12 points in total for 3 scenarios out of which 2 points are lying outside the valid bounded area of each slot. Valid bounded area refers to the area of a each parking slot. There are two different accuracy metric calculated. First accuracy metric is determined by the ratio of the number of generated data points lying within the valid bounded area of all slots to the total number of points generated.

$$Accuracy_{\text{points}} = \frac{\text{Number of generated data points lying within the valid bounded area of slots}}{\text{Total number of generated points}} \tag{7.1}$$

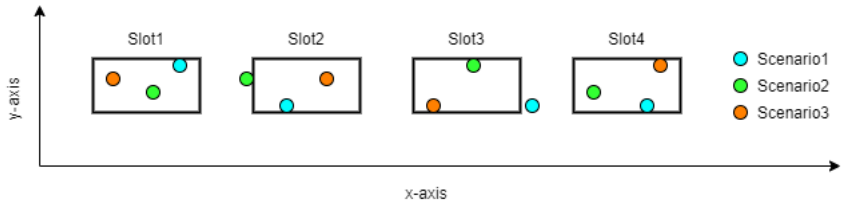


Figure 7.1: An example of scenario generation across 4 slots.

Secondly, we determine the accuracy of the complete valid scenario. In Figure 7.1, out of the three generated scenarios, only scenario3 is valid scenario because scenario3 have no data point lying outside bounding area and each of the scenario1 and scenario2 has a data point lying outside the valid bounding area of a slot. When a data point in a scenario lies outside the bounding area, it results in a case where a car is not parked within a slot. Thus, the scenario is invalidated.

$$Accuracy_{\text{scenario}} = \frac{\text{Number of valid scenarios}}{\text{Total number of generated scenarios}} \tag{7.2}$$

Second accuracy metric is a very stringent metric because even if the one coordinate in a scenario lies outside the bounding box we consider the complete scenario to be invalid.

7.1.3 Number of duplicated and repeated scenarios generated

Another metric considered to evaluate the generated output is based on the number of duplicated and repeated scenarios generated. In order to determine duplicated and repeated scenarios, we first generate some 'n' number of scenarios from trained GAN model. Each scenario is a row of generated data. So firstly, we compare whether a row of generated data is identical to any row of data in training/real data. If so then the scenario is duplicated. Ideally, it is not desired to have duplicates of real data being generated by the model. Similarly, we check if there are any row of generated data being repeated. Ideally, it is desired to have n distinct scenarios being generated from the model.

7.2 Results

This section presents the detailed analysis and results of the experiments as described in chapter 6.

7.2.1 Loss plot

The Figure 7.2 shows the loss plot of the WGAN-GP model for eight slots non-overlapping scenario and the Figure 7.3 shows the loss plot of the WGAN-GP model for four slots overlapping-scenario

as described in section 6. The Wasserstein loss function seeks to increase the gap between the scores of the real and the fake data by the critic. From the plots, this is quite evident. The red line indicates the critic score for the real data and the green line indicates the critic score for the fake data and from the plot we can see a clear distinction in the gap between the scores of the real/training and the fake data. The purple line in the plots indicates the gradient penalty value and the blue line is the total critic loss as discussed in chapter 3.4.6. The orange line indicates the generator loss. We expect generator loss to have similar value to that of critic score for real/training data because the generator tries to generate samples of similar distribution to real/training data. The Wasserstein loss function is not an absolute and comparable loss. This means there is no fixed value that defines how much this loss should be for it to be classified as a good loss. Rather this depends on the model configuration and dataset. The only thing what is important is that convergence behaviour must be observed. This can be seen the plots. Although the loss plot is different for both the models and we see convergence after 25000 epochs in 7.2 and after 35000 epochs in 7.3 clearly evident from the plot.

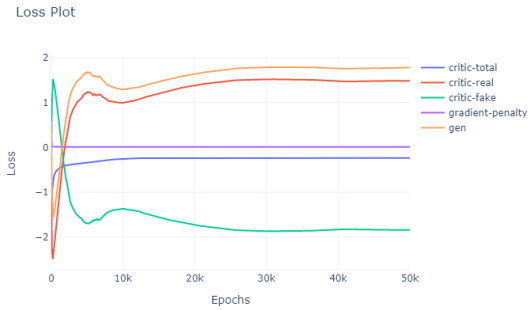


Figure 7.2: Loss plot of WGAN-GP model non-overlapping scenario using Dataset1 for eight parking slots

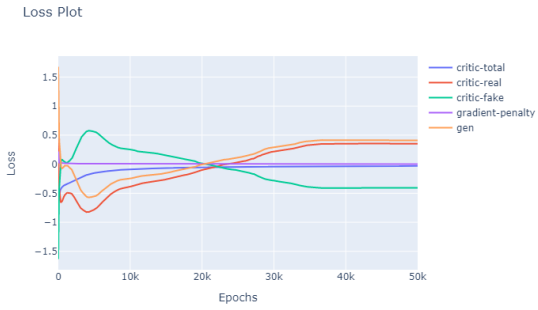


Figure 7.3: Loss plot of WGAN-GP model for overlapping scenario using Dataset4 for four parking slots

7.2.2 Accuracy

We use the accuracy metric to evaluate the non-overlapping parking scenario. Table 7.1 shows the accuracy of the generated non-overlapping parking scenarios for eight parking slots for different datasets. After training of the model 1000 scenarios are generated for each dataset. Each dataset differ in terms of number of samples of training data with 5000, 3000 and 1000 samples respectively. The accuracy of the generated scenarios are calculated as discussed in 7.1.2.

Dataset	Scenario	$\text{Accuracy}_{\text{points}}$	$\text{Accuracy}_{\text{scenario}}$
Dataset1 (5000 samples)	Non-overlapping (8 slots)	95-97%	~75%
Dataset2 (3000 samples)	Non-overlapping (8 slots)	93-95%	~70%
Dataset3 (1000 samples)	Non-overlapping (8 slots)	85-88%	~30%

Table 7.1: Accuracy of generated non-overlapping scenarios for different dataset

We can see that for dataset1 (5000 samples) the $\text{Accuracy}_{\text{points}}$ is about 95-97% and $\text{Accuracy}_{\text{scenario}}$ is about 75%. For dataset2 (3000 samples), $\text{Accuracy}_{\text{points}}$ is about 93-95% and $\text{Accuracy}_{\text{scenario}}$ is about 70%. Similarly for dataset3 (1000 samples), $\text{Accuracy}_{\text{points}}$ is about 85-88% and $\text{Accuracy}_{\text{scenario}}$ is only 30%. We can see that as the number of training samples in the dataset decreases the total accuracy of the generated scenarios also decreases. By having more number of samples in the dataset, the model was able to learn the distribution better and generalise, thus resulting in better accuracy.

Dataset	Scenario	Mean distance of data points outside bounding box (in meters)	Variance of data points outside bounding box (in meters)
Dataset1 (5000 samples)	Non-overlapping (8 slots)	0.136	0.0133
Dataset2 (3000 samples)	Non-overlapping (8 slots)	0.152	0.026
Dataset3 (1000 samples)	Non-overlapping (8 slots)	0.114	0.0096

Table 7.2: Error in generated scenarios for different dataset

For each of the point lying outside the valid bounded area of a slot, the distance measure was done to check how far it is lying outside. The mean and variance were calculated for those points lying outside the valid bounded area of a slot. The table 7.2 shows the mean distance and variance of all the generated points lying outside the bounded area of slots. We see that the mean distance of generated points outside is 0.136 meters for a model trained with Dataset1. Also, the mean distance of generated points lying outside for the models trained with Dataset2 and Dataset3 is 0.152 meters and 0.114 meters respectively. Although there were some points lying outside the bounding area of the slot they are not very far outside the slot and rather lie just outside the edge of the slots. Thus we can say that model has learnt the distribution well.

7.2.3 Duplicated and repeated scenarios generated

Table 7.3 shows the number of duplicated scenario generated compared to training dataset as well as the number of repeated scenario generated by the model. The duplicated and repeated scenarios are checked on the generated scenarios by truncating x-COG, y-COG and heading angle values to one, two and three decimal places for every slot. It was observed that scenarios with features/columns truncated to even one decimal place there were no duplication or repetition. This suggests that the model did a good job in generating new scenarios and also ensuring that there are no repeated scenarios generated.

Dataset	**Scenario**	**Number of duplicated scenarios**	**Number of repeated scenarios**
Dataset1 (5000 samples)	Non-overlapping (8 slots)	0	0
Dataset2 (3000 samples)	Non-overlapping (8 slots)	0	0
Dataset3 (1000 samples)	Non-overlapping (8 slots)	0	0
Dataset4 (3000 samples)	Overlapping (4 slots)	0	0

Table 7.3: Number of duplicated and repeated scenarios generated for different datasets with values of the features/columns truncated to one decimal place.

7.2.4 Visual inspection

The visual inspection of the results provides some insight into the generated scenario. Figures 7.4, 7.5 and 7.6 shows the histogram and cumulative percentage of few of the individual columns of the generated samples and training samples of non-overlapping scenario. And Figures 7.7 and 7.8 shows the plot for overlapping scenario .The generated samples are represented in yellow and training samples are represented in blue. From the inspection, we can see that the distribution of the generated data follows the distribution of the real/training data. Thus we can say that there are no signs of mode collapse occurring.

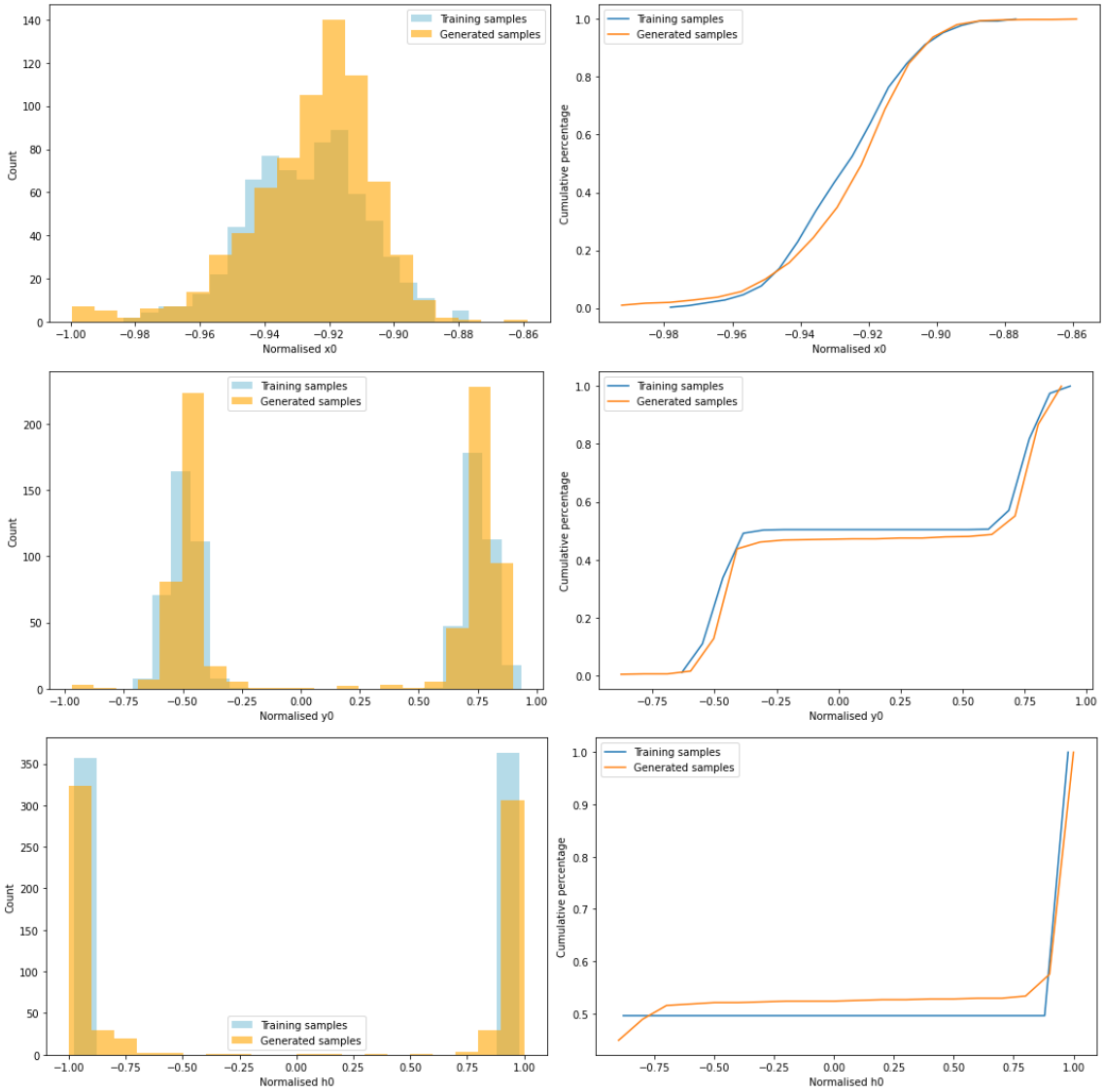


Figure 7.4: Histogram and cumulative percentage of normalised x-CoG, y-CoG and heading of slot1 of non-overlapping scenario

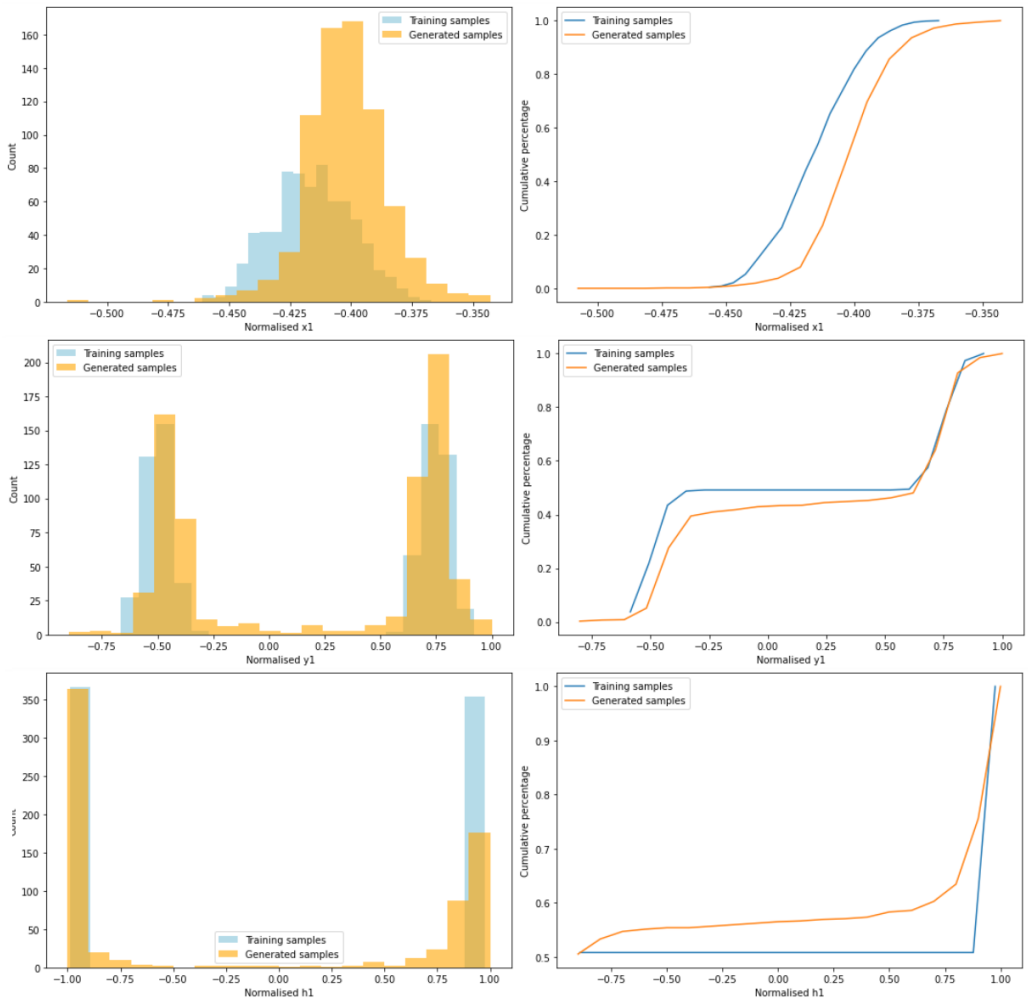


Figure 7.5: Histogram and cumulative percentage of normalised x-CoG, y-CoG and heading of slot2 of non-overlapping scenario

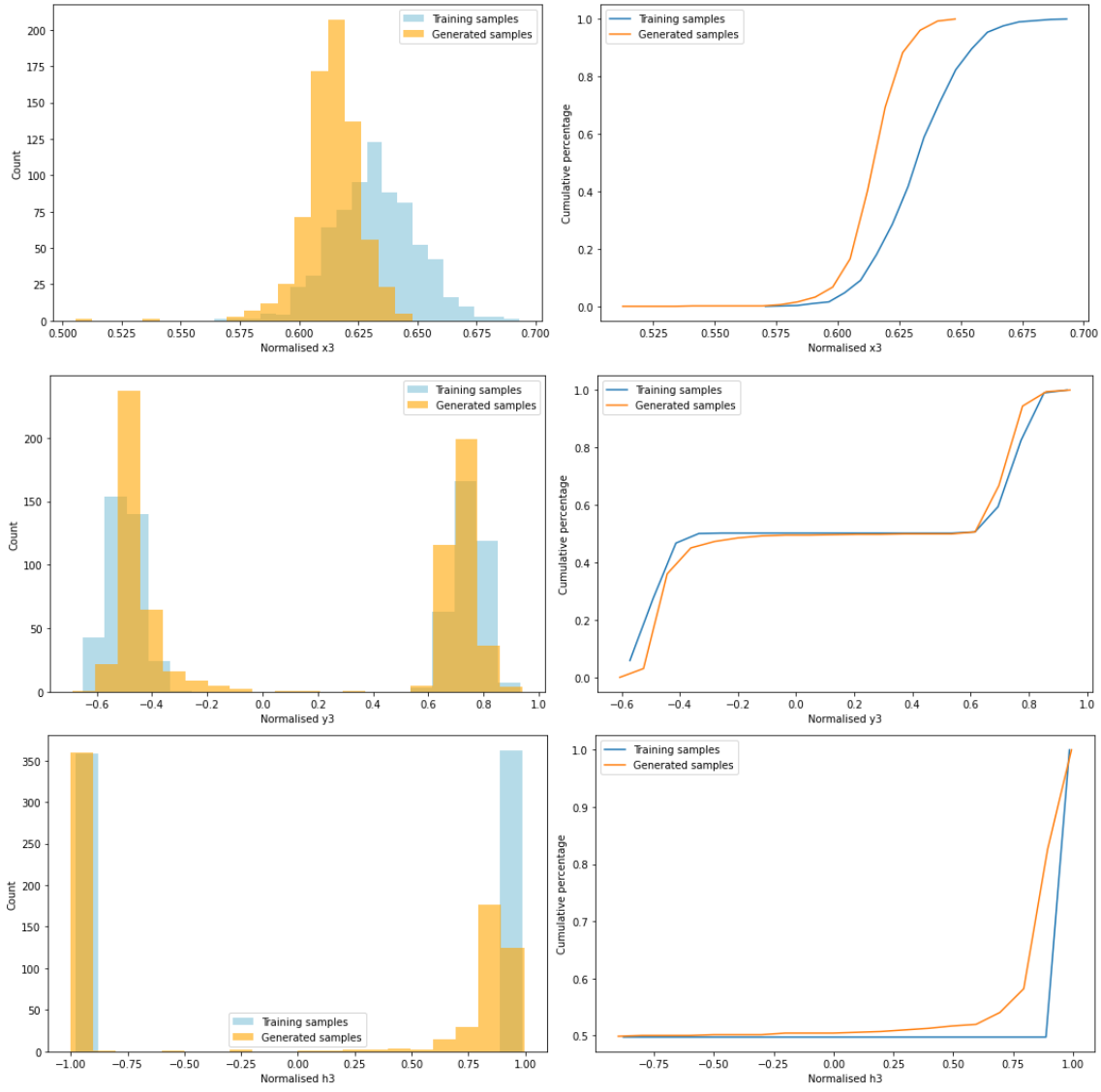


Figure 7.6: Histogram and cumulative percentage of normalised x-CoG, y-CoG and heading of slot3 of non-overlapping scenario

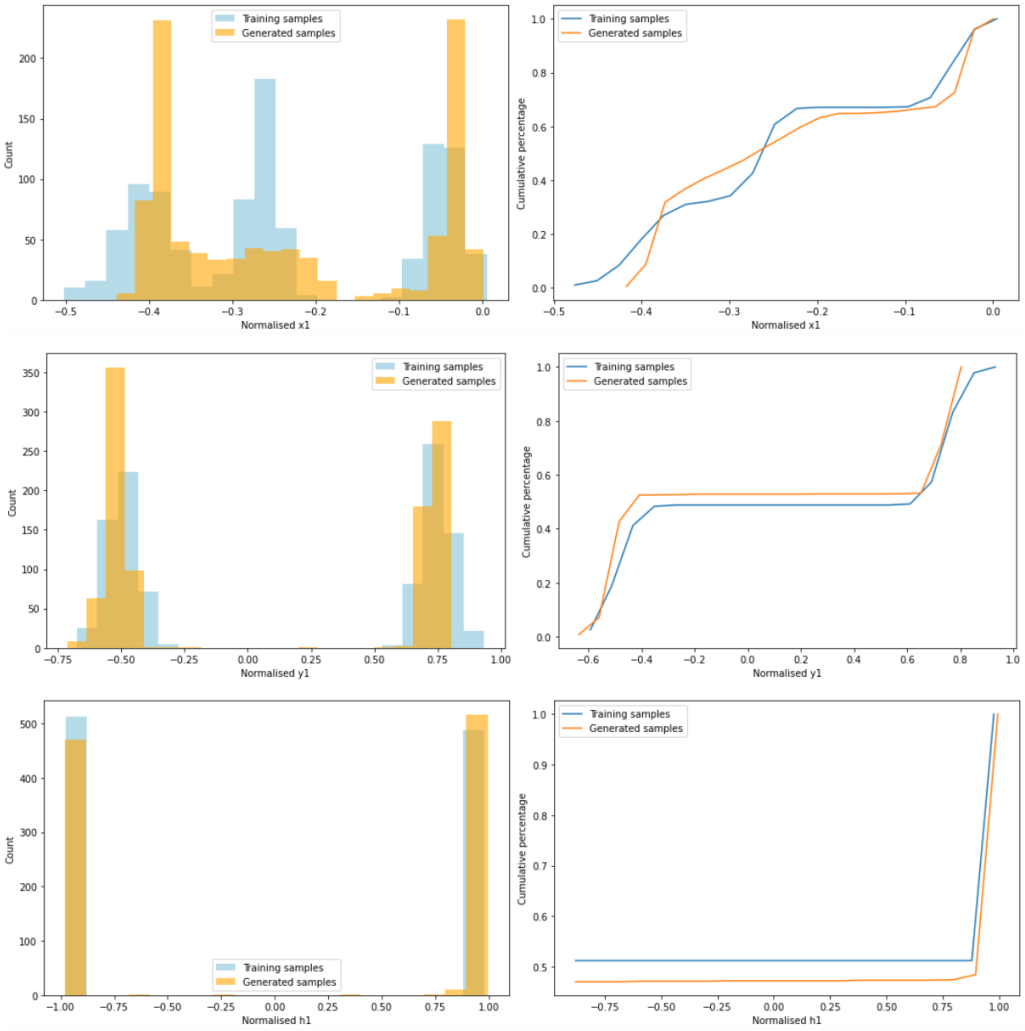


Figure 7.7: Histogram and cumulative percentage of normalised x-CoG, y-CoG and heading of slot2 of overlapping scenario

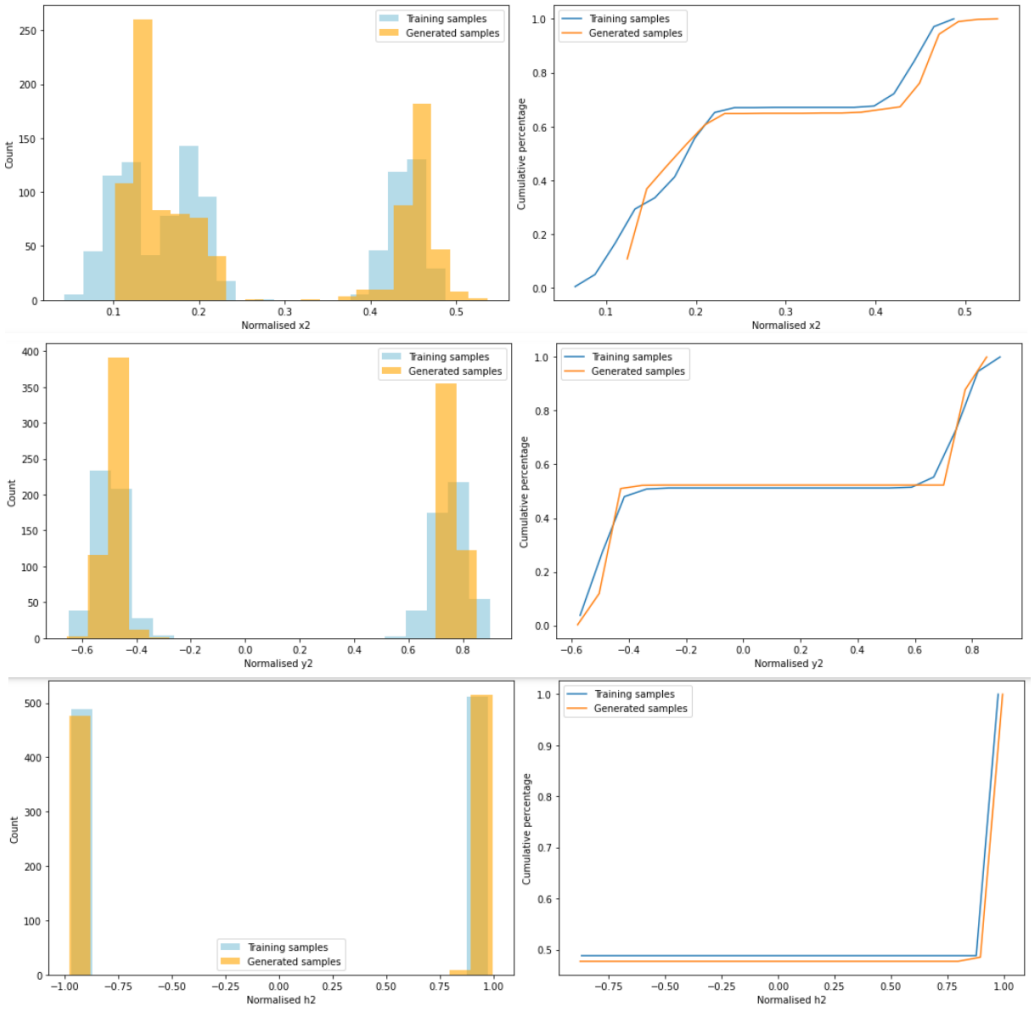


Figure 7.8: Histogram and cumulative percentage of normalised x-CoG, y-CoG and heading of slot3 of overlapping scenario

After the scenarios are generated, they are imported to the Simcenter Prescan software to visualise the output on the simulation environment. Figures 7.9, 7.10 and 7.11 show some non-overlapping parking scenarios across eight parking slots. Figures 7.12, 7.13, 7.14, 7.15 show some overlapping parking scenarios across four parking slots.

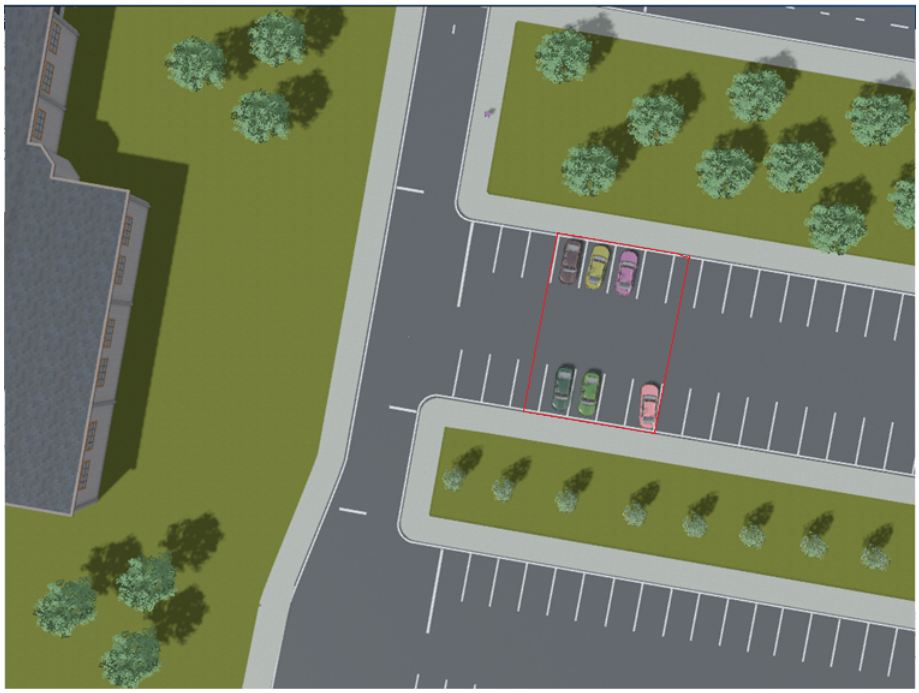


Figure 7.9: First non-overlapping scenarios of parked cars across eight slots

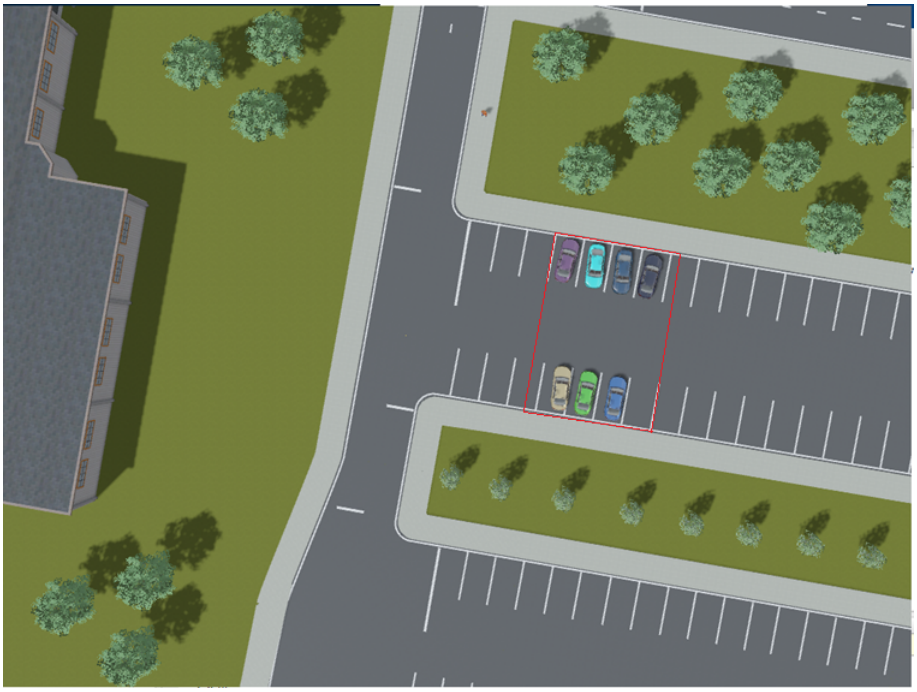


Figure 7.10: Second non-overlapping scenarios of parked cars across eight slots

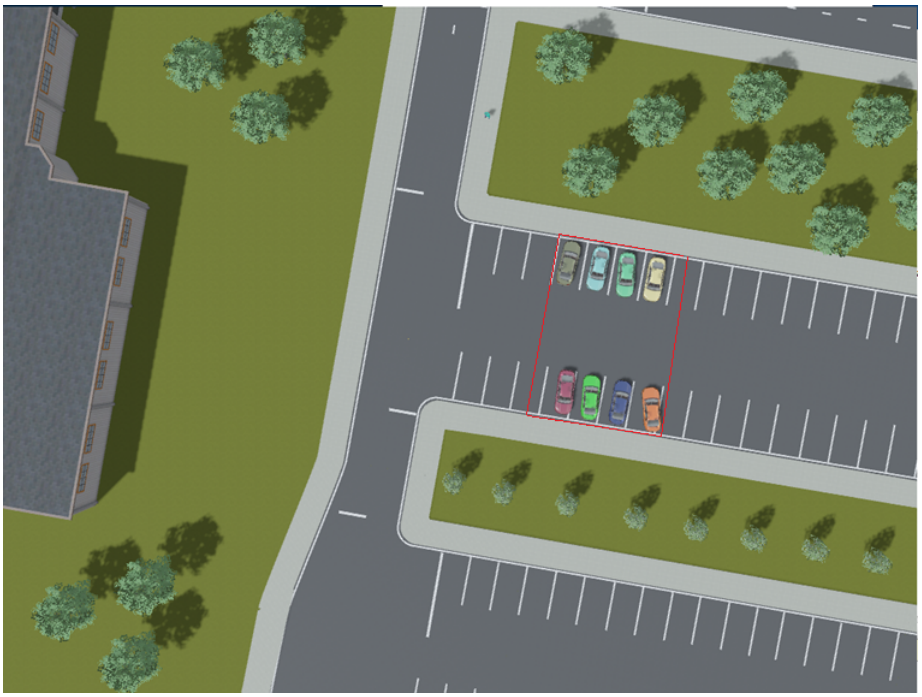


Figure 7.11: Third non-overlapping scenarios of parked cars across eight slots

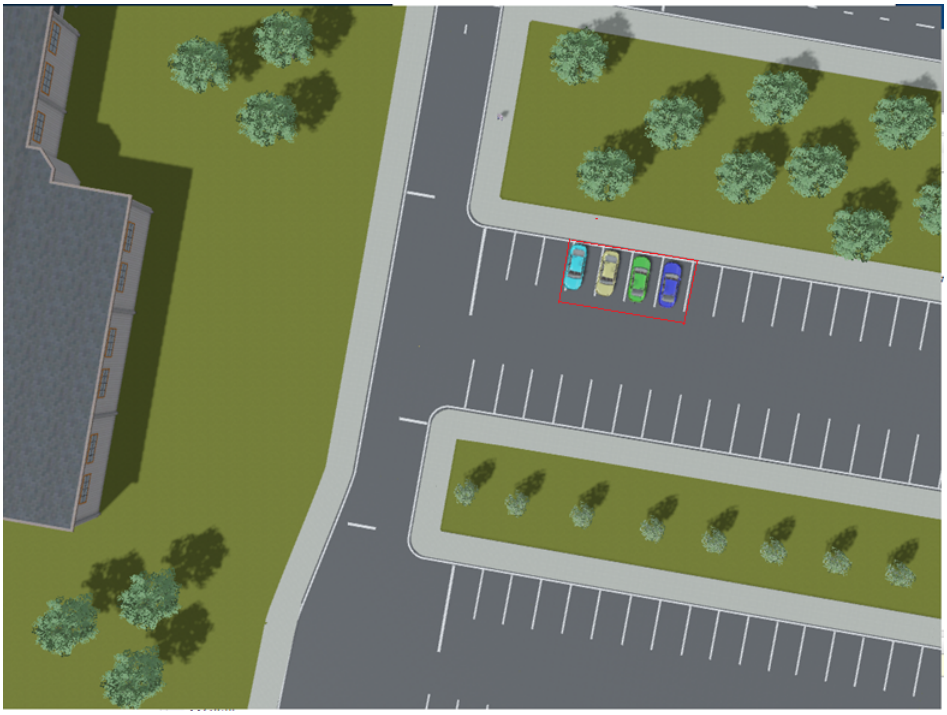


Figure 7.12: First overlapping scenarios of parked cars across four slots

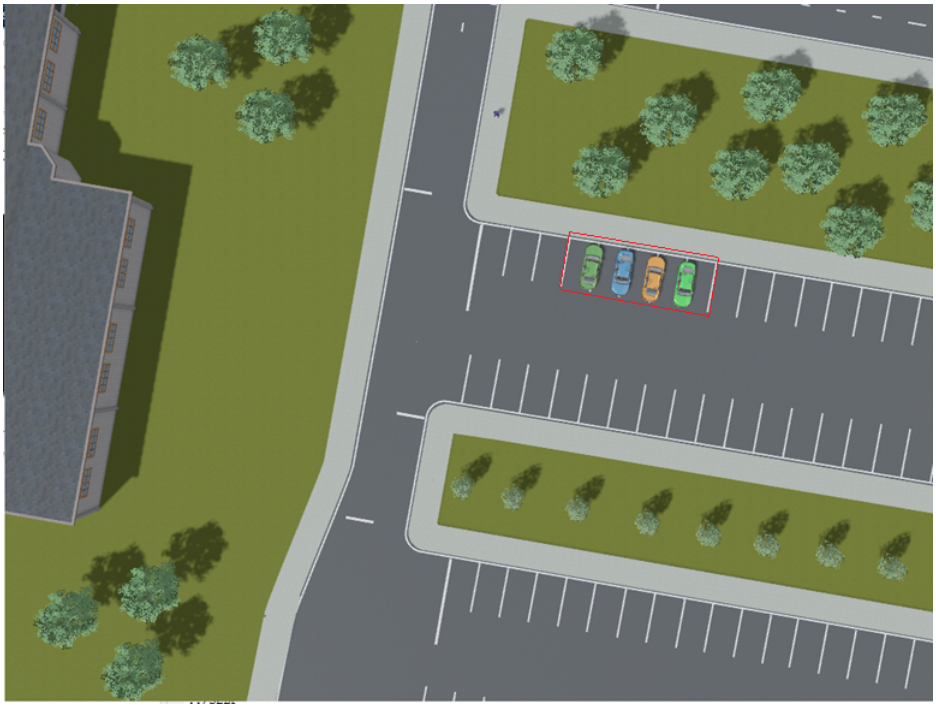


Figure 7.13: Second overlapping scenarios of parked cars across four slots

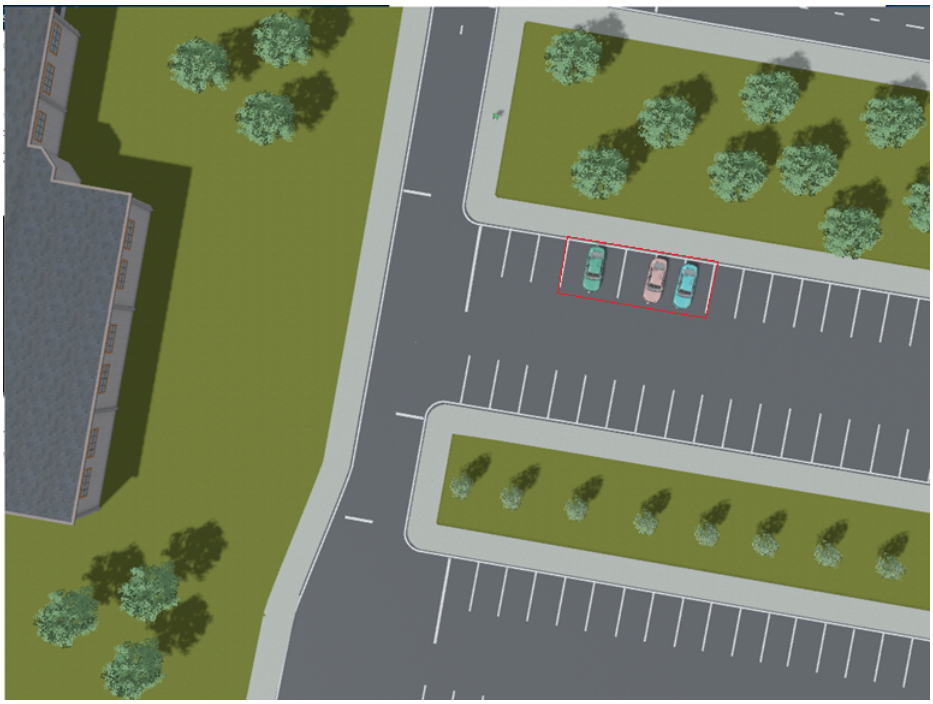


Figure 7.14: Third overlapping scenarios of parked cars across four slots

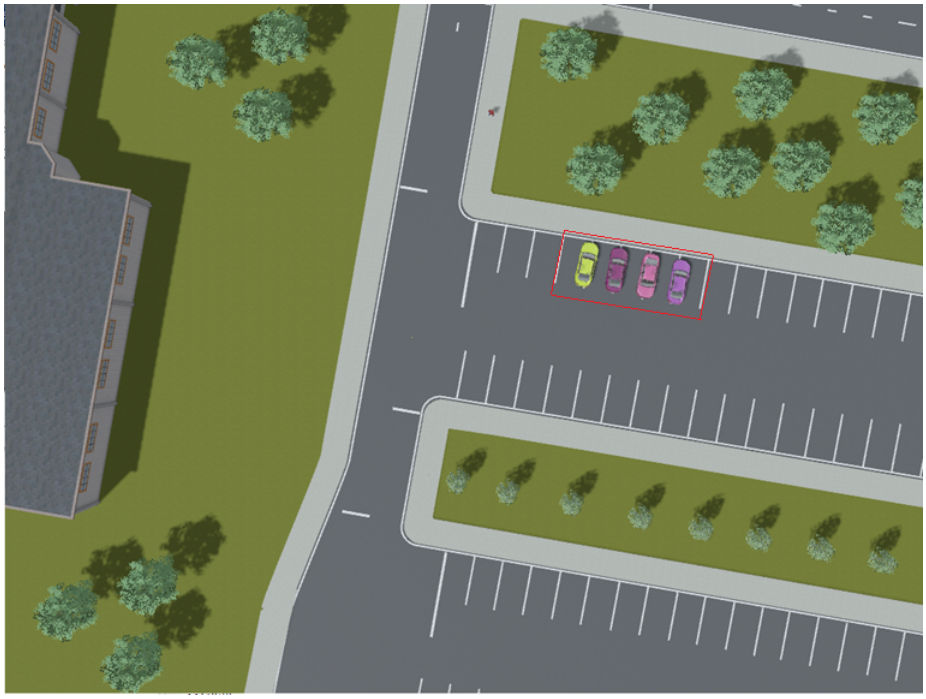


Figure 7.15: Fourth overlapping scenarios of parked cars across four slots

Chapter 8

Conclusions

8.1 Summary

In this thesis, we investigated the feasibility of scenario generation for testing autonomous vehicles using Generative Adversarial Networks (GANs). The goal of the project was to generate realistic parking scenarios. Two variants of parking scenario were generated, namely:- overlapping and non-overlapping parking scenario. The approach to this work was not trivial since GANs are typically used on image datasets, but for this work, it was not the case. For this work, the dataset is in tabular form. Since dataset was not readily available to approach the work, suitable datasets had to be generated to achieve the goal of the project.

Different GAN models were studied and analysed, and several experiments were performed in order to determine the suitable model for the final implementation. Vanilla GAN and conditional GAN models suffered from mode collapse problem. The model with Wasserstein GAN with gradient penalty overcomes the mode collapse problems. Thus for final scenario generation, Wasserstein GAN with gradient penalty models were designed with simple architectures for generator and discriminator networks.

After training the models, the generator network was used to generate new scenarios. The new scenarios generated were evaluated on several metrics such as accuracy, visual inspection and number of duplicated and repeated scenarios to evaluate the performance of the generator network. We observed that we were able to achieve good accuracy for non-overlapping scenarios, and there were no duplicate or repeated scenarios being generated by the model. Finally, newly generated scenarios were imported and demonstrated on the Simcenter Prescan environment.

To conclude, a framework was successfully developed to demonstrate the parking scenario using Generative Adversarial Network. Using this approach we could reduce the time taken to generate new scenarios and hence this can lead to faster testing and reduce time to market for ADAS.

8.2 Future work

1. Currently this parking scenario has been developed for one particular car model (Audi Sedan A8) used in Simcenter Prescan environment. This can be extended for different car models by adding additional label parameter in the training dataset to indicate the model of the car.

2. Current model demonstrates the parking scenario for eight slots in case of non-overlapping and four slots in case of overlapping scenarios. This can be extended to a higher number of parking slots.

3. We could investigate the parking scenario generation using GAN in an alternate way by using image datasets. In order approach this, the images of different parking scenarios needs to be captured from the environment model by placing camera to create the dataset. This dataset can be used to train the GAN models. After training, the parameters (x-COG, y-COG and heading angle) of the car is needed to be extracted from the new generated scenario image which would require lot of postprocessing and could be cumbersome. However, it would be interesting to see the results by this approach if it could achieve better results compared to the work done in this thesis.

Bibliography

[1] Advanced Driver Assistance System. URL: http://tadviser.com/index.php/Product:LG_ADAS_%28Advanced_Driver_Assistance_System%29. vii, 2

[2] Autonomous Driving System Toolbox. URL: https://nl.mathworks.com/products/automated-driving.html#scenario-generation. 10

[3] Cross entropy loss. URL: https://machinelearningmastery.com/cross-entropy-for-machine-learning/. 15

[4] Jensen-Shannon and Kullback-Leibler divergence. URL: https://medium.com/datalab-log/measuring-the-statistical-similarity-between-two-samples-using-jensen-shannon-and-kullback-leibler-8d05af514b15. 18

[5] OpenSCENARIO. URL: http://www.opendrive.org/. 3, 10

[6] OpenSCENARIO. URL: http://www.openscenario.org/. 3, 10

[7] ReLU, LeakyReLU and Sigmoid activation functions. URL: https://medium.com/@himanshuxd/activation-functions-sigmoid-relu-leaky-relu-and-softmax-basics-for-neural-networks-and-deep-8d9c70eed91e. 26

[8] SAE Levels of Automation. URL: https://www.news24.com/fin24/Finweek/Featured/autonomous-vehicles-its-time-to-get-out-of-the-drivers-seat-20180228. vii, 2

[9] SAE Levels of Driving Automation. URL: https://www.sae.org/news/press-room/2018/12/sae-international-releases-updated-visual-chart-for-its-%E2%80%9Clevels-of-driving-automation%E2%80%9D-standard-for-self-driving-vehicles. 1

[10] Understanding activation functions. URL: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0. 14, 26

[11] Martin Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks, 2017. 17

[12] Gerrit Bagschik, Till Menzel, and Markus Maurer. Ontology based scene creation for the development of automated vehicles. 06 2018. 3, 6

[13] Shane Barratt and Rishi Sharma. A note on the inception score, 2018. 9, 44

[14] Ali Borji. Pros and cons of gan evaluation measures, 2018. 9

[15] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. InfoGAN: Interpretable representation learning by information maximizing generative adversarial nets. *Advances in Neural Information Processing Systems*, pages 2180–2188, 2016. 9, 18

[16] Edward Choi, Bradley Malin, and Jon Duke. Generating Multi-label Discrete Patient Records using Generative Adversarial Networks. 68:1–20, 2017. 9

[17] S. De, S. Mohamed, K. Bimpisidis, D. Goswami, T. Basten, and H. Corporaal. Approximation trade offs in an image-based control system. In *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1680–1685, 2020. 6

[18] Sayandip De, Sajid Mohamed, Dip Goswami, and Henk Corporaal. Approximation-aware design of an image-based control system. *IEEE Access*, 2020. 6, 11

[19] E. de Gelder and J. Paardekooper. Assessment of automated driving systems using real-life scenarios. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 589–594, 2017. 6

[20] Ugur Demir and Gozde Unal. Patch-Based Image Inpainting with Generative Adversarial Networks. 2018. vii, 4, 7, 8

[21] W. Deng, Y. H. Lee, and A. Zhao. Hardware-in-the-loop simulation for autonomous driving. In *2008 34th Annual Conference of IEEE Industrial Electronics*, pages 1742–1747, 2008. 10, 11

[22] Tong Duy Son, Ajinkya Bhave, and Herman Van der Auweraer. Simulation-based testing framework for autonomous driving development. 03 2019. vii, 10, 11

[23] Hala Elrofai, Jan-Pieter Paardekooper, and Erwin de Gelder. Streetwise: Scenario-based safety validation of connected and automated driving. page 28, 2018. 1, 2, 6

[24] Ş. Y. Gelbal, E. Altuğ, and E. F. Keçeci. Design and hil setup of an autonomous vehicle for crowded environments. In *2016 IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*, pages 1370–1375, 2016. 11

[25] Ian J Goodfellow, Jean Pouget-abadie, Mehdi Mirza, Bing Xu, and David Warde-farley. Generative Adversarial Nets. pages 1–9. 7, 9, 16, 17, 27

[26] Ishaan Gulrajani. Improved Training of Wasserstein GANs. 20, 31, 39

[27] F. Hendriks, Martijn Tideman, R. Pelders, R. Bours, and X. Liu. Development tools for active safety systems: Prescan and vehil. pages 54 – 58, 08 2010. vii, 23

[28] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium, 2018. 9, 44

[29] Irina Higgins, Loic Matthey, Xavier Glorot, Arka Pal, Benigno Uria, Charles Blundell, Shakir Mohamed, and Alexander Lerchner. Early visual concept learning with unsupervised deep learning, 2016. 9

[30] Charles A. Holt and Alvin E. Roth. The nash equilibrium: A perspective. *Proceedings of the National Academy of Sciences*, 101(12):3999–4002, 2004. 17

[31] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 4, 7

[32] Nikolay Jetchev, Urs Bergmann, and Roland Vollgraf. Texture Synthesis with Spatial Generative Adversarial Networks. (ii), 2016. 4

[33] Yanghua Jin and Yingtao Tian. Towards the Automatic Anime Characters Creation with Generative Adversarial Networks. 92(Summer 2017):1–16. vii, 7, 8

[34] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. 28, 33, 39, 43

[35] Robert Krajewski and Tobias Moers. VeGAN : Using GANs for Augmentation in Latent Space to Improve the Semantic Segmentation of Vehicles in Images from an Aerial Perspective. *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1440–1448, 2019. 9

[36] Robert Krajewski, Tobias Moers, Dominik Nerger, and Lutz Eckstein. Data-Driven Maneuver Modeling using Generative Adversarial Networks and Variational Autoencoders for Safety Validation of Highly Automated Vehicles. *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, 2018-November:2383–2390, 2018. 9

[37] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989. 13

[38] Yann LeCun, Y. Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–44, 05 2015. 13

[39] S. Chintala M. Arjovsky and L. Bottou. Wasserstein GAN. 2017. 19

[40] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013. 26

[41] Till Menzel, Gerrit Bagschik, Leon Isensee, Andre Schomburg, and Markus Maurer. From functional to logical scenarios: Detailing a keyword-based scenario description for execution in a simulation environment, 2019. vii, 7

[42] Till Menzel, Gerrit Bagschik, and Markus Maurer. Scenarios for development, test and validation of automated vehicles. *2018 IEEE Intelligent Vehicles Symposium (IV)*, Jun 2018. 2

[43] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets, 2014. 9, 18

[44] Olof Mogren. Continuous recurrent neural networks with adversarial training. (Nips), 2016. 7

[45] S. Mohamed, S. De, K. Bimpisidis, V. Nathan, D. Goswami, H. Corporaal, and T. Basten. IMACS: A Framework for Performance Evaluation of Image Approximation in a Closed-loop System. In *MECO*, pages 1–4, 2019. 10, 11

[46] Sajid Mohamed, Asad Ullah Awan, Dip Goswami, and Twan Basten. Designing image-based control systems considering workload variations. In *58th IEEE Conference on Decision and Control (CDC)*, 2019. 6

[47] Sajid Mohamed, Dip Goswami, Vishak Nathan, Raghu Rajappa, and Twan Basten. A scenario-and platform-aware design flow for image-based control systems. *Microprocessors and Microsystems*, 75:103037, 2020. 6

[48] Sajid Mohamed, Diqing Zhu, Dip Goswami, and Twan Basten. Optimising quality-of-control for data-intensive multiprocessor image-based control systems considering workload variations. In *21st Euromicro Conference on Digital System Design (DSD)*, pages 320–327, 2018. 6

[49] Vinod Nair and Geoffrey Hinton. Rectified linear units improve restricted boltzmann machines vinod nair. volume 27, pages 807–814, 06 2010. 26

[50] Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. f-gan: Training generative neural samplers using variational divergence minimization, 2016. vii, 9, 10

[51] Simon Osindero. Conditional Generative Adversarial Nets. pages 1–7. 18

[52] Noseong Park, Mahmoud Mohammadi, Hongkyu Park, and Youngmin Kim. Data Synthesis based on Generative Adversarial Networks. 11(10), 2018. 9

[53] Andreas Pütz, Adrian Zlocki, Julian Bock, and Lutz Eckstein. System validation of highly automated vehicles with a database of relevant traffic scenarios. 2017. 6

[54] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2016. 9, 26, 35

[55] Scott Reed, Zeynep Akata, Xinchen Yan, and Lajanugen Logeswaran. Generative Adversarial Text to Image Synthesis. 2016. 7

[56] Tripp W. Corless M. Roggero M, Sharma S. and Chaganti S.K. Creation and variation of traffic scenarios for virtual validation of automated driving systems. 2020. 10

[57] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017. 28

[58] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans, 2016. 9, 18

[59] Thalles Silva. An intuitive introduction to generative adversarial networks. URL: `https://www.freecodecamp.org/news/an-intuitive-introduction-to-generative-adversarial-networks-gans-7a2264a81394/`. vii, 4

[60] Irhum Skafkat. *Intuitively Understanding Variational Autoencoders*, 2018. vii, 16

[61] R. Sutton and A. Barto. Reinforcement learning: An introduction. *IEEE Transactions on Neural Networks*, 16:285–286, 2005. 13

[62] S. Ulbrich, T. Menzel, A. Reschka, F. Schuldt, and M. Maurer. Defining and substantiating the terms scene, situation, and scenario for automated driving. In *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, pages 982–988, 2015. vii, 3

[63] Michal Uricar, Pavel Krizek, David Hurych, Ibrahim Sobh, Senthil Yogamani, and Patrick Denny. Yes, we gan: Applying adversarial techniques for autonomous driving, 2019. 9

[64] Cédric Villani. *Optimal transport – Old and new*, volume 338, pages xxii+973. 01 2008. 19

[65] Walther Wachenfeld and Hermann Winner. *The Release of Autonomous Vehicles*, pages 425–449. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016. 1

[66] L. Wasserstein. Markov processes over denumerable products of spaces describing large systems of automata. 1969. 9, 44

[67] Max Welling. An Introduction to Variational Autoencoders. 2019. 15

[68] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network, 2015. 26

[69] Lei Xu and Kalyan Veeramachaneni. Synthesizing Tabular Data using Generative Adversarial Networks. 9

[70] Jiawei Zhang. Gradient descent based optimization algorithms for deep learning models training, 2019. 26

[71] Lijun Zhao, Huihui Bai, Jie Liang, Bing Zeng, Anhong Wang, and Yao Zhao. Simultaneous color-depth super-resolution with conditional generative adversarial networks. *Pattern Recognition*, 88:356–369, 2019. 4

[72] Jun-Yan Zhu, Philipp Krähenbühl, Eli Shechtman, and Alexei A. Efros. Generative visual manipulation on the natural image manifold. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 597–613, Cham, 2016. Springer International Publishing. 4

[73] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. 4, 7

Appendix A

Architectures of GAN models

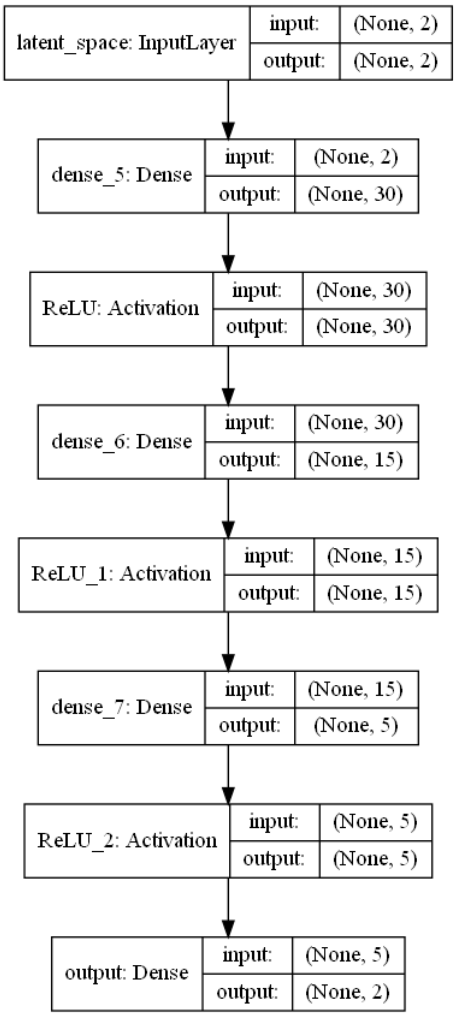


Figure A.1: Architecture of the generator network of vanilla GAN

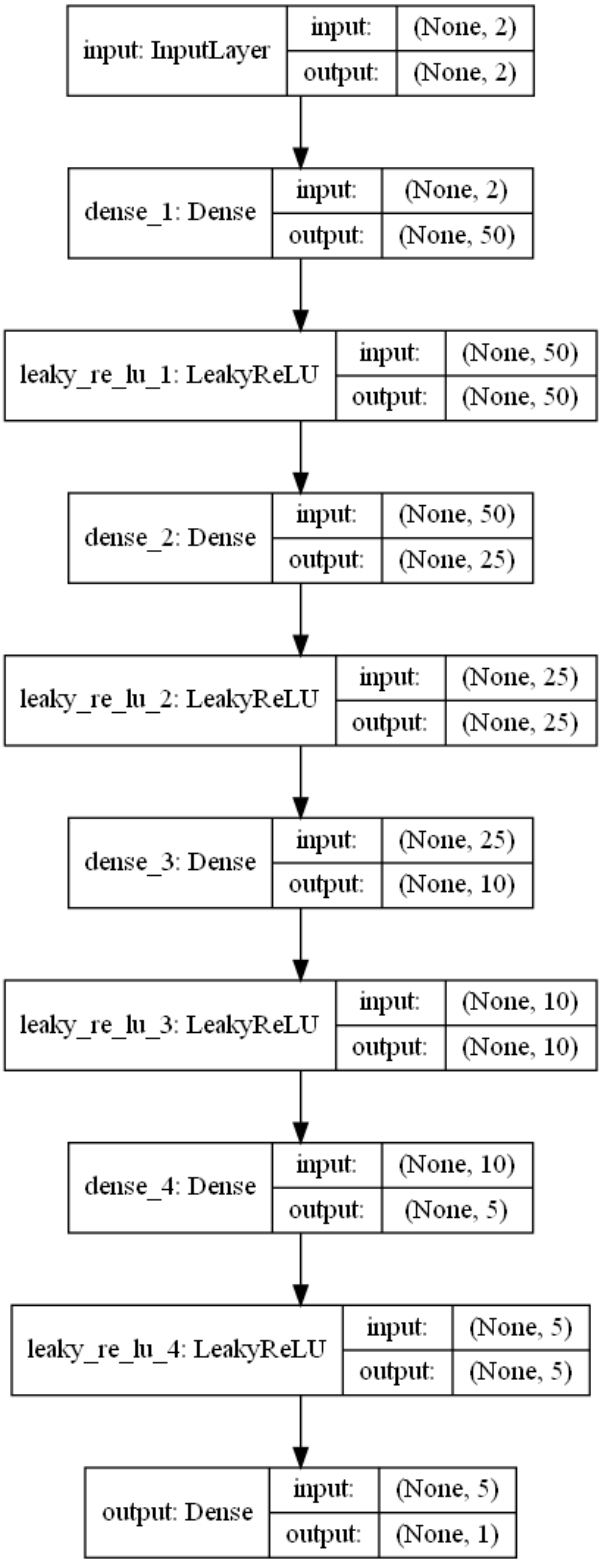


Figure A.2: Architecture of the discriminator network of vanilla GAN

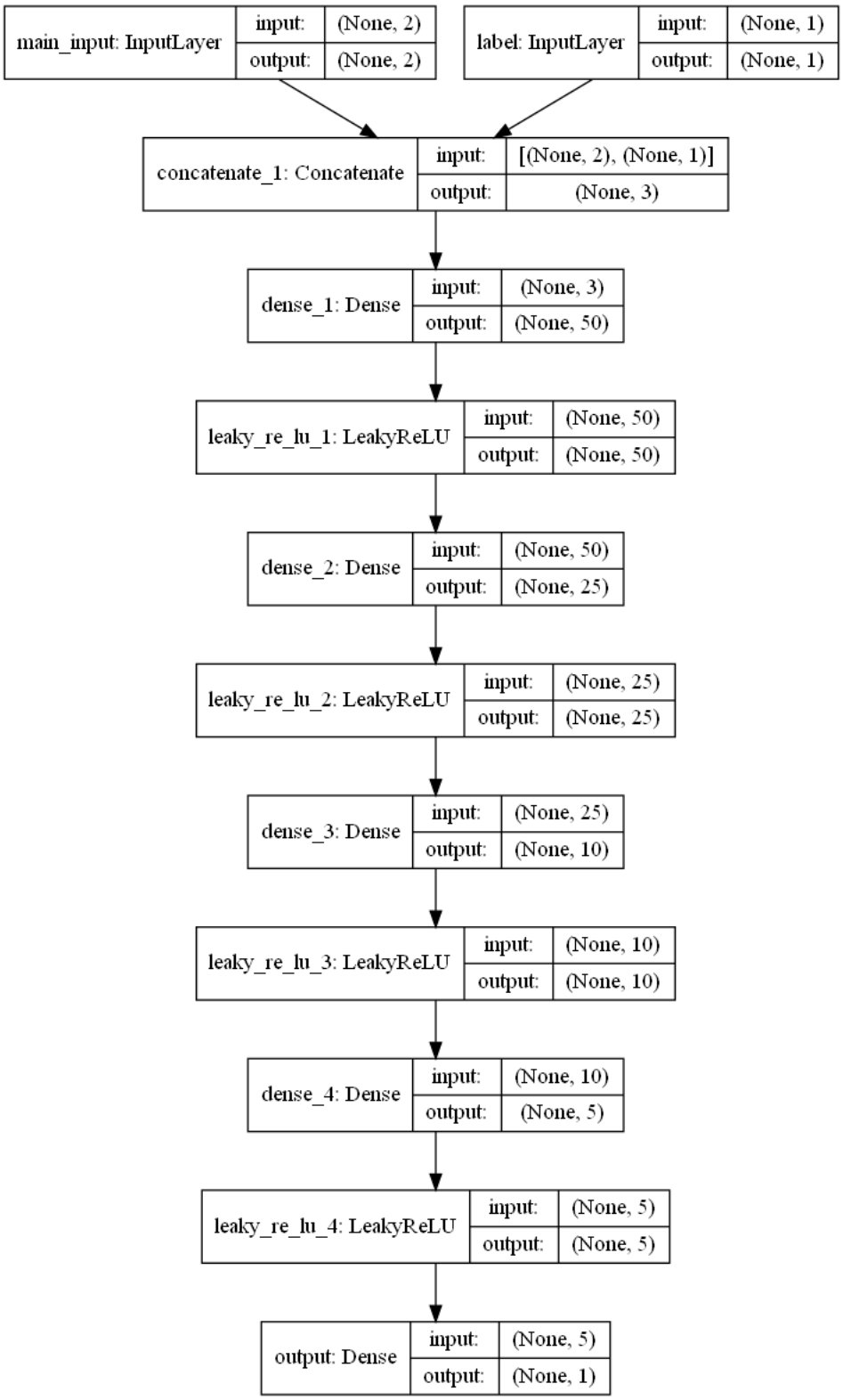


Figure A.3: Architecture of the discriminator network of CGAN

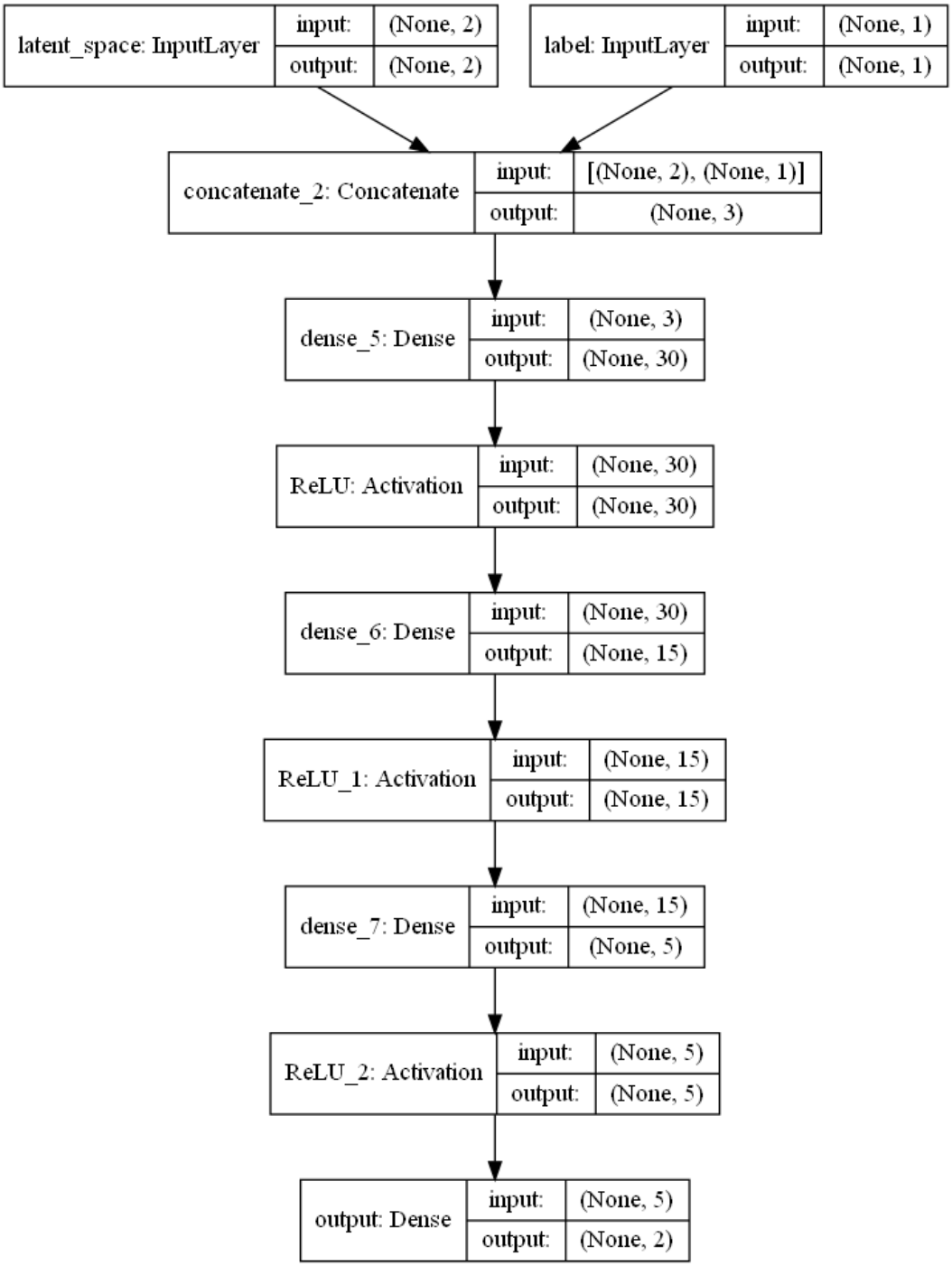


Figure A.4: Architecture of the generator network of CGAN

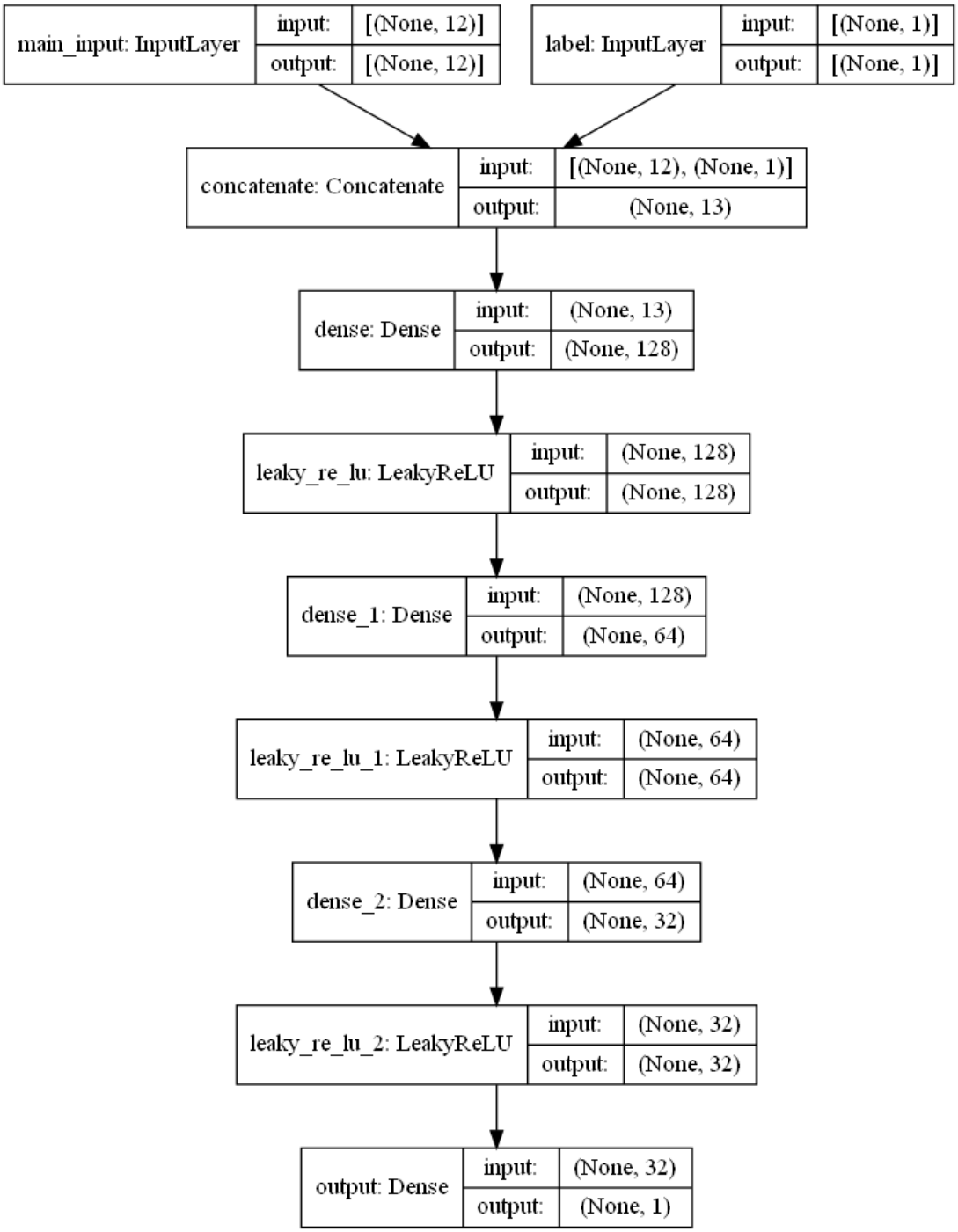


Figure A.5: Architecture of the discriminator network of WGAN-GP

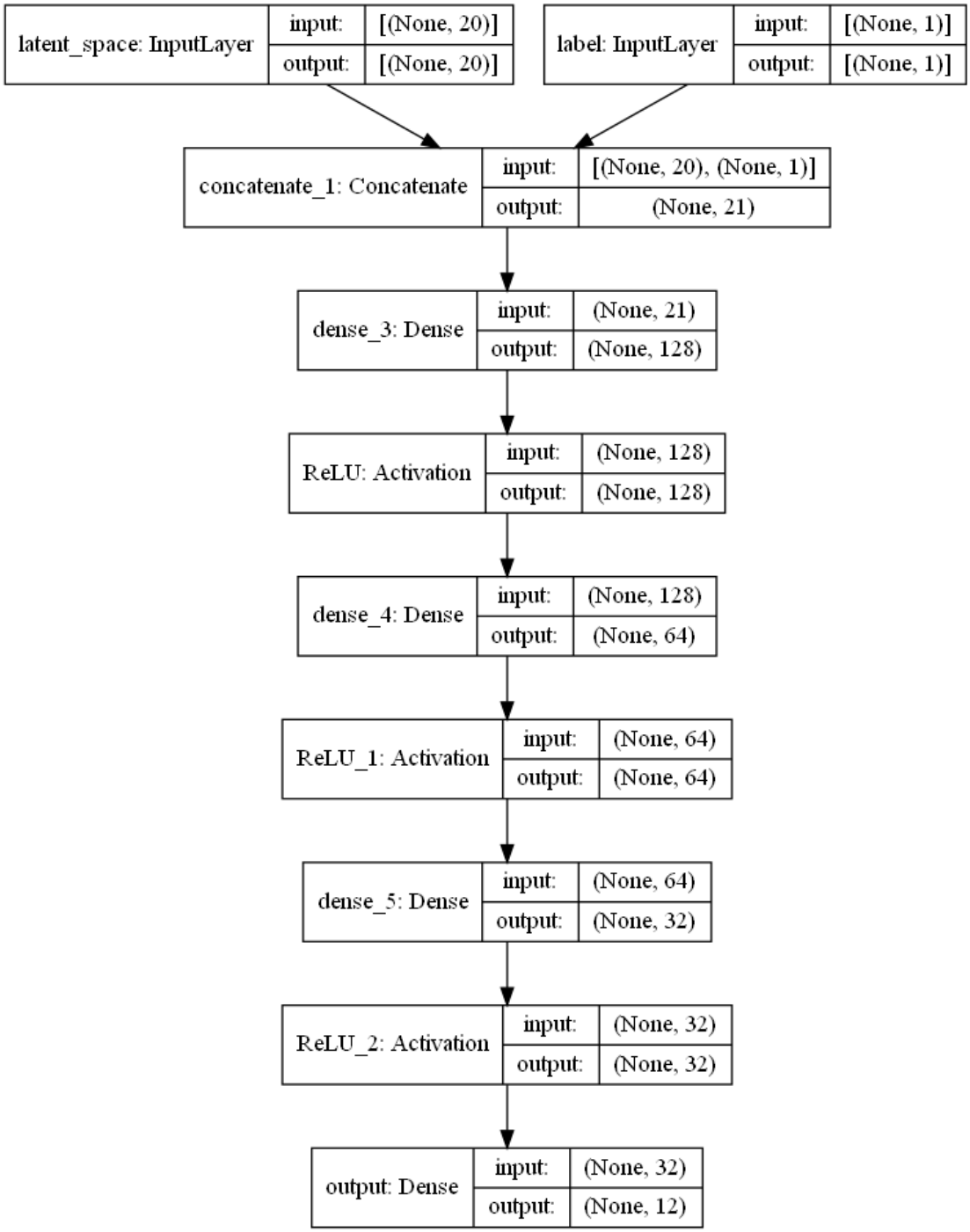


Figure A.6: Architecture of the generator network of WGAN-GP