ikerlan
MEMBER OF BASQUE RESEARCH & TECHNOLOGY ALLIANCE

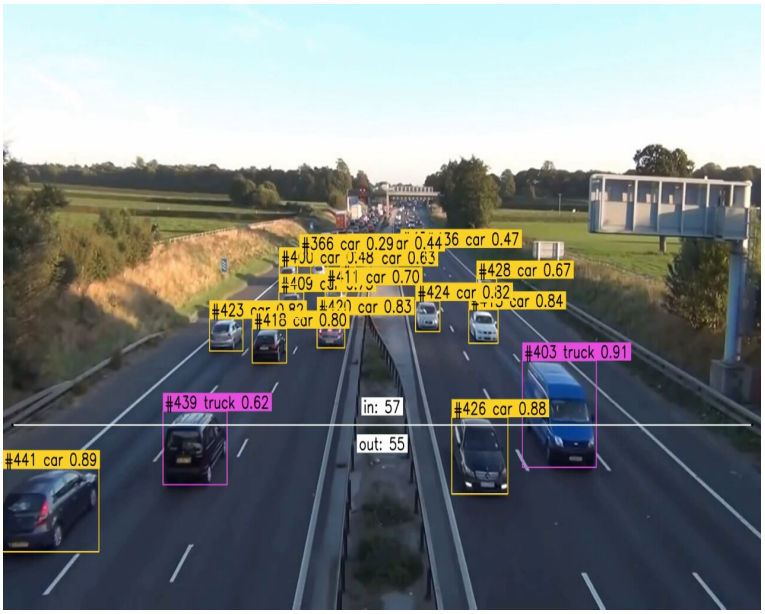
eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea

BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

GRADO EN INGENIERÍA INFORMÁTICA EN GESTIÓN Y SISTEMAS DE INFORMACIÓN

FINAL DEGREE PROJECT

RESEARCH AND EVALUATION OF OPENPILOT THROUGH ADVERSARIAL EXAMPLES



Student: Elorriaga Aramburu, Unai

Director: Mikel Egaña, Xabier E. Barandiaran

Course: 2023-2024

Date: November 18, 2024

Abstract

Deep Learning (DL) models have revolutionized modern algorithms, enabling them to learn patterns among large datasets and produce results over new unseen data. End-to-end Autonomous Driving (AD) systems are very convenient since they make real-time driving decisions through a Deep Neural Network (DNN) model, but they are also safety-critical since being confused by an attack could lead to accidents.

An Adversarial Example (AE) is a carefully crafted input designed to mislead neural networks, causing them to misclassify or make incorrect predictions. This is especially concerning in safety-critical applications, such as object identification in autonomous vehicles or facial recognition in security systems.

This Final Degree Project (FDP) integrates the CARLA simulator with the Openpilot AD system, enabling the engagement of a simulated vehicle. Then, research was conducted on the architecture of Openpilot, such as modules, DNN models, data formats, C++ source code, and vehicle sensors.

During the development, a successful Carlini & Wagner (CW) attack has been implemented against image classification models, and then, white-box and black-box scenarios have been designed and implemented against the Supercombo model, trying to fool this system to cause a rear-end collision.

Despite the challenges encountered implementing attacks against the Supercombo model due to various constraints, this FDP provides significant insights for users and the Ikerlan S.Coop. organization. It contributes valuable knowledge on Openpilot's architecture and source code, the design and implementation of adversarial attack scenarios, and an assessment of Openpilot's security. Additionally, a comprehensive GitHub repository has been developed, offering a step-by-step methodology for replicating the process, enabling further exploration in this area, and laying the foundation for future advancements.

In conclusion, although neural networks are a powerful tool for Machine Learning (ML), it is important to address the problem of neural network safety and seek solutions to ensure their effectiveness and reliability in critical applications, especially when human safety is at stake. Current research in this field is essential to improve the security of neural networks and promote their use in critical and high-security environments.

Resumen:

Los modelos de DL han revolucionado los algoritmos modernos, permitiéndoles aprender patrones en grandes conjuntos de datos y producir resultados sobre nuevos datos no vistos. Los sistemas de AD de extremo a extremo son muy convenientes, ya que toman decisiones de conducción en tiempo real mediante un modelo de DNN, pero también son críticos en términos de seguridad, ya que ser confundidos por un ataque podría llevar a accidentes.

Un AE es una entrada cuidadosamente elaborada diseñada para engañar a las redes neuronales, provocando que clasifiquen incorrectamente o hagan predicciones erróneas. Esto es especialmente preocupante en aplicaciones críticas para la seguridad, como la identificación de objetos en vehículos autónomos o el reconocimiento facial en sistemas de seguridad.

Este FDP integra el simulador CARLA con el sistema de AD Openpilot, habilitando el funcionamiento de un vehículo simulado. Posteriormente, se ha investigado sobre la arquitectura de Openpilot, incluyendo módulos, modelos de DNN, formatos de datos, código fuente en C++ y sensores del vehículo.

Durante el desarrollo, se ha implementado con éxito un ataque de CW contra modelos de clasificación de imágenes, y luego, se han diseñado e implementado escenarios de caja blanca y caja negra contra el modelo Supercombo, con el objetivo de engañar a este sistema para provocar una colisión por alcance.

A pesar de los desafíos encontrados al implementar ataques contra el modelo Supercombo debido a varias restricciones, este FDP proporciona importantes conocimientos para los usuarios y la organización Ikerlan S.Coop. Contribuye con conocimiento valioso sobre la arquitectura y el código fuente de Openpilot, el diseño e implementación de escenarios de ataque adversario y una evaluación de la seguridad de Openpilot. Además, se ha desarrollado un completo repositorio en GitHub, que ofrece una metodología paso a paso para replicar el proceso, lo cual permite una mayor exploración en esta área y sienta las bases para futuros avances.

En conclusión, aunque las redes neuronales son una herramienta poderosa para el ML, es importante abordar el problema de la seguridad de las redes neuronales y buscar soluciones para asegurar su efectividad y confiabilidad en aplicaciones críticas, especialmente cuando la seguridad humana está en juego. La investigación actual en este campo es esencial para mejorar la seguridad de las redes neuronales y promover su uso en entornos críticos y de alta seguridad.

Laburpena:

DL modeloek algoritmo modernoak iraultzailea izan dute, datu multzo handien artean ereduak ikasteko eta datu berri ikusi gabekoetan emaitzak lortzeko aukera emanez. Amaieratik amaierara doazen AD sistemak oso erosoak dira, DNN eredu baten bidez errealitatean gidatze-erabakiak hartzen baitituzte, baina, aldi berean, segurtasun kritikoak dira, erasoren batek nahasiko balitu istripuak eragin baititzake.

AE bat sarrera arretaz egindako bat da, sare neuronala engainatzeko diseinatua, okerreko sailkapenak edo aurreikuspenak eginez. Hau bereziki kezkagarria da segurtasun kritikoak diren aplikazioetan, hala nola ibilgailu autonomoen objektu identifikazioan edo segurtasun-sistemetan aurpegi ezagutzan.

FDP honek CARLA simuladorea Openpilot AD sistemarekin integratzen du, ibilgailu simulatu baten erabilera ahalbidetuz. Ondoren, Openpilot-en arkitekturari buruzko ikerketa egin da, hala nola moduluak, DNN modeloak, datu formatuak, C++ iturburu-kodea eta ibilgailuaren sentsoreak.

Garapenaren bitartean, CW eraso bat arrakastaz ezarri da irudi-sailkapen ereduen aurka, eta, ondoren, kaxa zuriko eta kaxa beltzeko agertokiak diseinatu eta ezarri dira Supercombo ereduan, sistema hau engainatzeko atzealdeko talkaren bat eragiteko.

Supercombo ereduaren aurkako erasoak ezartzean hainbat murrizketen ondorioz aurkitutako erronkei aurre egiteaz gain, FDP honek ezagutza garrantzitsua eskaintzen die erabiltzaileei eta Ikerlan S.Coop. erakundeari. Openpilot-en arkitekturari eta iturburu-kodeari buruzko ezagutza baliotsua, eraso adversarioko agertokien diseinua eta ezarpena eta Openpilot-en segurtasunaren ebaluazioa eskaintzen ditu. Gainera, GitHub biltegi oso bat garatu da, prozesua errepikatzeko urratsez urratseko metodologia eskainiz, arlo honetan esplorazio gehiago ahalbidetuz eta etorkizuneko aurrerapenentzako oinarriak jarriz.

Azkenik, sare neuronalak ML-rako tresna indartsuak diren arren, sare neuronalaren segurtasunaren arazoari aurre egitea eta aplikazio kritikoetan eta segurtasun handiko inguruneetan haien eraginkortasuna eta fidagarritasuna bermatzeko irtenbideak bilatzea garrantzitsua da. Arlo honetako egungo ikerketa funtsezkoa da sare neuronalaren segurtasuna hobetzeko eta ingurune kritiko eta segurtasun handikoetan haien erabilera sustatzeko.

Graphical Abstract / Resumen Gráfico / Laburpen Grafikoa

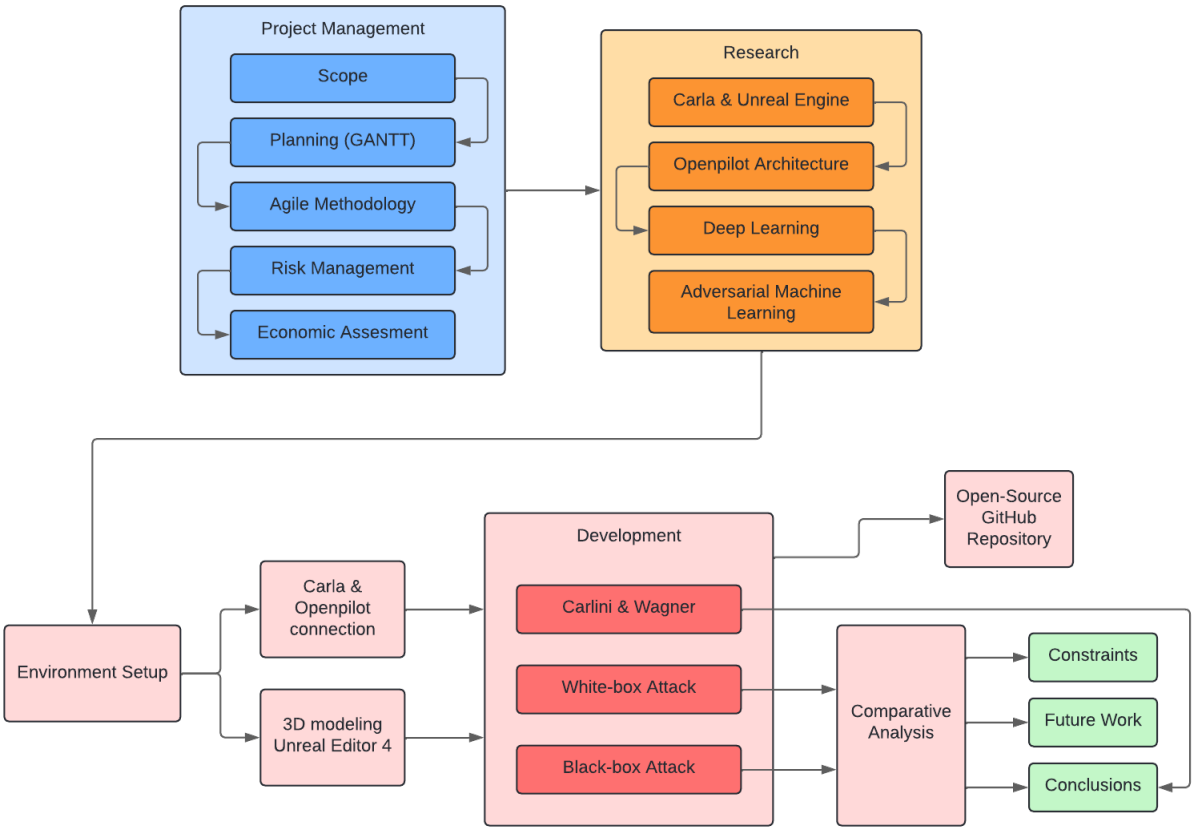


Figure 1: Visual diagram summarizing the overall process of the Final Degree Project

Contents

List of Figures

List of Tables

Abreviations

ADAS Advanced Driver Assistance System

AD Autonomous Driving

ACC Adaptative Cruise Control

DNN Deep Neural Network

CNN Convolutional Neural Network

ALC Automated Lane Centering

FCW Fordward Collision Warning

LDW Lane Departure Warning

USB Universal Serial Bus

ML Machine Learning

DL Deep Learning

AI Artificial Intelligence

FF Feed-Fordward

BP Back-Propagation

SL Supervised Learning

UL Unsupervised Learning

AML Adversarial Machine Learning

EOT Expectation Over Transform

WBS Work Breakdown Structure

AE Adversarial Example

FDP Final Degree Project

RDI Research, Development and Innovation

INE Instituto Nacional de Estadística

DDT Dynamic Driving Tasks

EC Evolutionary Computation

ES Evolution Strategy

RNN Recurrent Neural Network

GRU Gated Recurrent Unit

BPTT Backpropagation Through Time

FGSM Fast Gradient Sign Method

I-FGSM Iterative Fast Gradient Sign Method

CW Carlini & Wagner

ICR Indirect Cost Rate

CIFAR Canadian Institute for Advanced Research

SDG Sustainable Development Goal

1 Introduction

1.1 Context & Motivation

Artificial Intelligence (AI), particularly, ML algorithms have become widely popular, having the ability to rapidly process and learn patterns in large amounts of data, providing predictions over new, unseen data [1]. However, ML algorithms are also susceptible to malicious attacks, which can cause erratic behaviors and pose risks in many fields [2].

Consider the potential consequences of these emerging technologies. For instance, envision being in an autonomous vehicle that relies on ML algorithms to make decisions, and it malfunctions, leading to uncontrolled accidents [3, 4]. Alternatively, imagine serving as the Chief Financial Officer of a major corporation and falling victim to a deepfake fraud during a video call, resulting in a financial loss of $25 million [5]. Another example involves an AI-powered chatbot inadvertently disclosing confidential information [6]. These scenarios highlight the critical need for robust security measures, particularly where human safety is concerned.

How safe and trustworthy are ML algorithms nowadays? Are they a risk to society?

Considering the lack of investigation in secure AD systems [7], this FDP aims to conduct research on the security and robustness of ML algorithms in AD systems, particularly Openpilot.

To achieve this, ML algorithms, Adversarial Machine Learning (AML) attacks, and Openpilot architecture will be researched. Then, the CW state-of-the-art algorithm will be implemented against two models to learn to craft AEs: a fine-tuned RESNET-50 model and a CNN model built and trained from scratch, using the CIFAR-10 dataset. Then, three scenarios will be designed to craft AEs against Openpilot and tested within an AD simulator. These algorithms will be compared based on their effectiveness and efficiency in creating adversarial scenarios, specifically focusing on inducing rear-end collisions.

1.2 Objectives

The primary aim of this project is to learn how AEs are crafted and explore the architecture (modules, algorithms, etc.) of AD systems, specifically Openpilot, assessing the security of Openpilot ML models against AEs. To methodically address these challenges, the specific objectives of this project are outlined as follows:

- **Learn ML algorithms**: Investigate how DL models work. How can a DL algorithm learn patterns among data, make predictions, how they are trained, and different types of models (DNNs, CNNs, etc.).
- **Investigate AML algorithms**: Investigate existing types of AML attacks and different state-of-the-art algorithms for each.
- **Implement an AE attack**: Before deepening into Openpilot and try to mislead a complex AD system, learn to implement an state-of-the-art attack to craft AEs and mislead a DL model.
- **Examine Openpilot's architecture**: Learn how an AD system works: modules, services, algorithms, etc. Also, how they receive information about real-world status (e.g. pedestrians, other vehicles, traffic signs, etc.) and the process of lane decision-making.
- **Research AE state-of-the-art against AD systems**: Investigate the existing literature to learn which algorithms are used to create AE and analyze previous experiments conducted within the field of AD systems.
- **Running an AD vehicle in CARLA simulator**: Set up the CARLA simulator linked with Openpilot, followed by learning to control the simulation environment, including vehicle spawning and engagement through Openpilot.
- **Explore 3D modeling in Unreal Editor 4**: Customize a vehicle's appearance using Unreal Editor 4 and develop skills in 3D modeling tools.
- **Development of AE algorithms against Openpilot**: Design attack scenarios to craft AE against AD systems.
- **Comparative Analysis**: Perform a comparative analysis of the generated patches, focusing on their crafting efficiency and effectiveness in adversarial contexts.
- **GitHub methodology**: Provide a comprehensive guide for new users, hosted on GitHub, detailing the methodology of implementing white-box and

black-box adversarial attacks against DL models, specifically, image classification and Openpilot AD system. This repository will serve as a hands-on introduction to adversarial attacks, equipping users with the steps necessary to reproduce experiments and further explore the security of autonomous driving models.

1.3 Contributions

This subsection describes which contributions this project makes to the field of AML and AD systems:

- **Contribution to knowledge**: All the research in this work provides valuable insights for the company mentioned in subsection 1.5, which is keen on exploring these fields and understanding the current state-of-the-art to further its future works:
  - Openpilot AD system's input and output processing, system's architecture, modules, and the main DNN working explanations.
  - White-box approaches algorithms explanations: CW and Openpilot.
  - Black-box approaches algorithms explanations: (1+1)Evolution Strategy (ES), Gaussian mutation, Expectation Over Transform (EOT) and Disappearance attack.
  - Contribution to open-source knowledge in a GitHub repository with a step-by-step methodology for the company in subsection 1.5 (or any user) to reproduce the overall process and later deepen into this field considering conclusions in section 7 and future work subsection 7.3.
- **Integration of CARLA and Openpilot**: Successfully set up and integrated the CARLA simulator with the Openpilot autonomous driving system, enabling a realistic simulation environment for testing.
- **Customization with Unreal Editor**: Developed 3D modeling skills using Unreal Editor 4 to customize the appearance of vehicles in the CARLA simulator, enhancing the realism of the simulation environment and the strength of EOT technique.
- **Design of AE attacks**: Implementation of adversarial algorithms in python.
  - CW: Against custom CNN and fine-tuned RESNET-50.
  - Openpilot: Iterative white-box and black-box approaches.

- **Comparative Analysis**: Performed a detailed comparative analysis of the effectiveness and efficiency of the different developed algorithms, highlighting their respective strengths and weaknesses.
- **Evaluation of Openpilot's Robustness**: Conducted a ground-based assessment of Openpilot's robustness against adversarial attacks by running simulations and analyzing the system's responses.

1.4 Document Roadmap

This section serves as a guide to understand the structure of this FDP:

1. **Project management (section 4)**: FDP planning, risk evaluation and economic assesment.
2. **Background (section 2)**: Basic context information needed to understand the mechanisms of ML and AD, as well as state-of-the-art AE algorithms, to understand development section.
3. **Development (section 5)**: AD environment setup, change a vehicle appearance with 3D modeling, implementation and creation of AEs and evaluation.
4. **Conclusions, Constraints and Future work (section 7)**: Findings after analyzing the development section, limitations in the context of this specifical FDP and future work to enhance or follow the work.

1.5 Company

The FDP under discussion has been formulated during a tenure as an internship researcher, in partnership with Ikerlan S.Coop[1], a Research, Development and Innovation (RDI) technology center of the Basque Country, situated in Arrasate.

The research and development scope of the company encompasses a broad spectrum of sectors, detailed as follows:

- Transportation and mobility
- Energy
- Advanced manufacturing
- Electronics and ICT

[1] https://www.ikerlan.es

- Food and beverage
- Automotive
- Aeronautics and aerospace
- Healthcare

This project is categorized within the "Transportation and Mobility" sector and focuses on researching cybersecurity aspects relevant to AI within this specific domain.

1.6 Connection with the University's Democratic Principles

1.7 Connection with Sustainable Development Goals

This section outlines how the FDP aligns with some specific Sustainable Development Goal (SDG) to support a more sustainable environment.

1.7.1 SDG 3: Good Health and Well-being

SDG 3 focuses on ensuring healthy lives and promoting well-being for all ages. All the work in this FDP ensures the critical need to implement robust security measures in ML models, especially when human safety is concerned.

1.7.2 SDG 4: Quality Education

SDG 4 aims to provide inclusive and equitable quality education and to promote lifelong learning opportunities for everyone. The FDP contributes to this goal by enhancing educational outcomes in the field of AML and AD systems.

1.7.3 SDG 11: Sustainable Cities and Communities

This SDG aims to make cities and human settlements inclusive, safe, resilient, and sustainable. The FDP contributes to this goal by designing some adversarial scenarios against AD systems, specifically Openpilot. By ensuring the safety of AD systems, safer and more reliable systems are developed for smart and sustainable cities.

2 Background

This section provides an overview of all necessary concepts to contextualize the project and ensure the reader understands all the basic information before delving into the development.

2.1 Driving simulator: CARLA & Unreal Engine

The CARLA simulator represents a cornerstone in the development and testing of AD systems. It is an open-source simulator for autonomous driving research [8] built on Unreal Engine 4. Provides open digital assets (vehicles, buildings, town layouts, ...) and also sensor integration for vehicles, cameras, dynamic actors, and much more.

The simulator consists of a scalable client-server architecture [9] as shown in Figure 2. The server side manages all the simulation itself: updating the world state, and its actors, sensor managing, ... On the other hand, the client-side controls the actors' behavior, world conditions, traffic management, and is highly customizable for integrating any functionality with autonomous driving vehicles within the simulator.

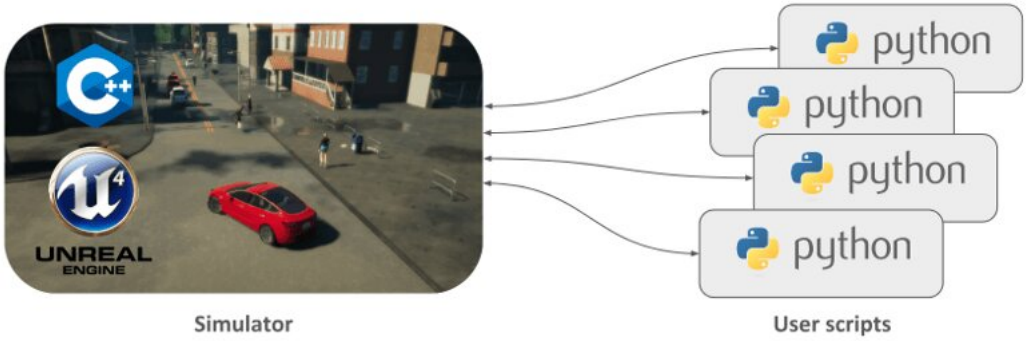


Figure 2: Carla simulator basic client/server structure

The vehicles are known as Actors [10], and they are configured by the blueprint classes, already-made models with animations and a series of attributes [11].
To customize the appearance of a blueprint (e.g. changing the back of a truck), Unreal Editor 4 is used as shown in Figure 3, a powerful tool for 3D videogame development [12].

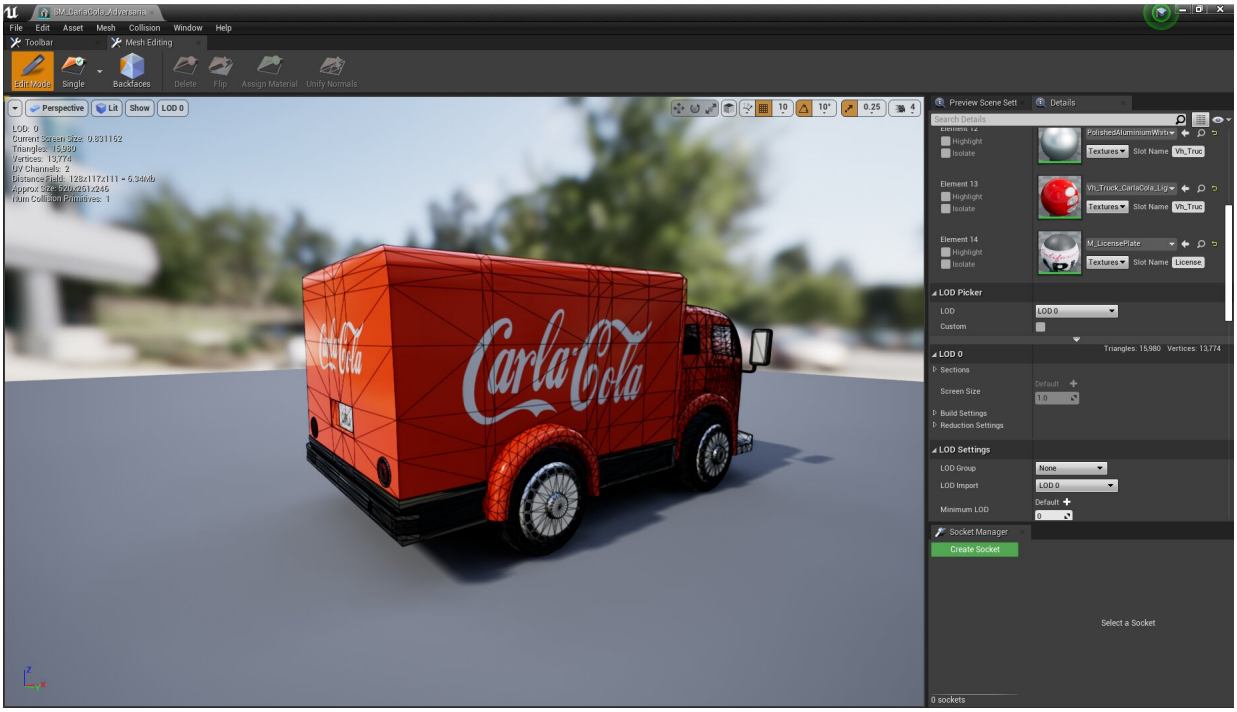


Figure 3: UnrealEditor Blueprint

2.2 Automated Driving: OpenPilot

An AD system is a complex architecture of modules, services, and algorithms that enable a vehicle to drive itself, either partially or fully. It processes a wide range of environmental data, including traffic signs, pedestrians, and other vehicles, to navigate safely and efficiently.

Openpilot is an open-source, level-2 Advanced Driver Assistance System (ADAS) [13]. It performs some self-driving functions like Adaptative Cruise Control (ACC), Automated Lane Centering (ALC), Fordward Collision Warning (FCW), and Lane Departure Warning (LDW) [14].

However, it is not capable of driving anonymously as only level 4-5 ADAS are fully responsible for Dynamic Driving Tasks (DDT) [13]. Although it can perform basic steering, acceleration, and braking when required, a human is still required to be in the driver's seat to take over at any time [15].

It is crucial to know the architecture of Openpilot, including its modules, sensors, services, libraries, algorithms, and neural networks. Understanding this will

clarify how an AD system processes environmental data to make decisions such as accelerating, braking, or changing direction.

2.2.1 Openpilot Architecture

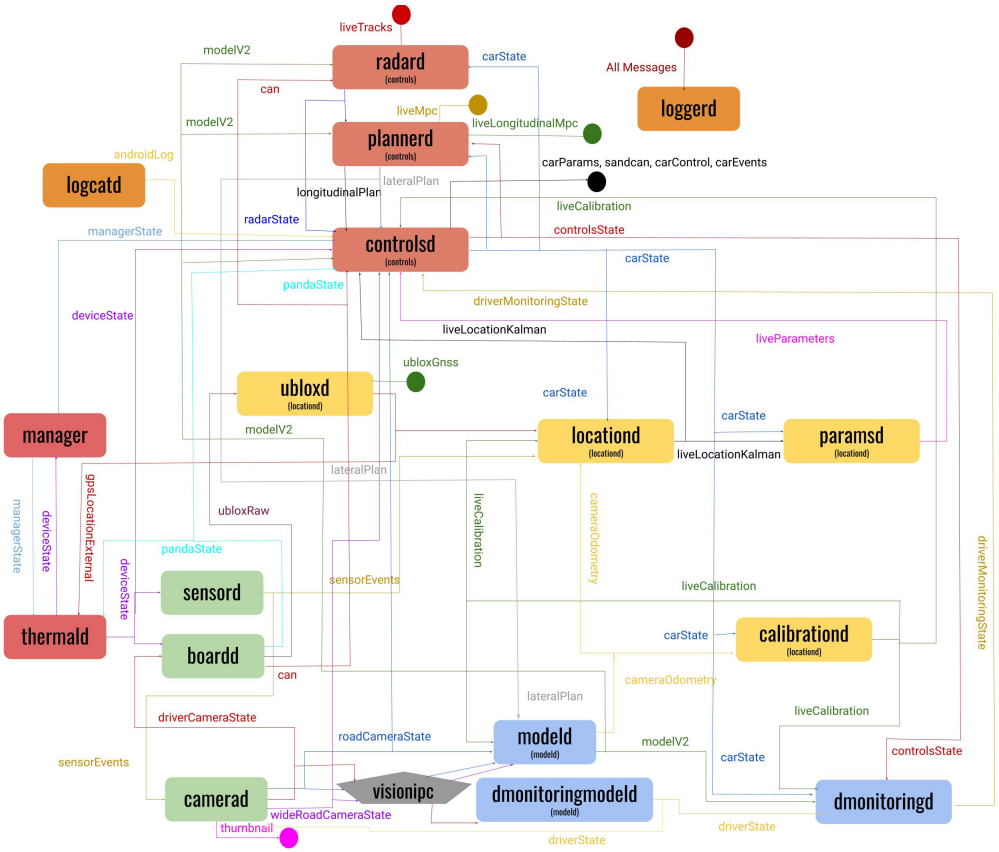


Figure 4: Complete Openpilot service architecture [16]. Green: Sensors and Actuators, Blue: Neural Network Runners, Yellow: Localization and Calibration, Red: Controls, Orange: System, Logging, and Miscellaneous services.

As shown in Figure 4, the Openpilot architecture works with modules, sensors, services, and libraries. The most important ones for this FDP purposes are described as follows [16]:

- **camerad**: It captures images from the road and driver camera. camerad writes the image data directly to visionipc which can efficiently pass images around with low overhead.

- **sensord**: configures and reads the rest of the sensors (gyro, accelerometer, magnetometer, and light sensors).
- **modeld**: reads in the image stream from visionipc into the main driving neural network, the Supercombo model, which outputs the desired driving path along with other metadata including lane lines, lead cars, road edges, and more.
- **plannerd**: based on the predicted paths by the Supercombo model, prepares lateral planning (steering) and longitudinal planning (gas/brake).
- **controlsd**: It is the service that actually controls the car. It receives the plan from plannerd, in the form of curvatures and velocities/accelerations, and converts it to control signals.

2.2.2 Supercombo: End-to-End Neural Network

The Openpilot's main driving Neural Network is Supercombo, also called *trajectory planning model* [17]. This Neural Network is responsible for predicting the vehicle's trajectory directly from the camera images and using plannerd and controlsd modules, to tell the actuators which actions they should take (brake, gas, steer, etc.) [17].

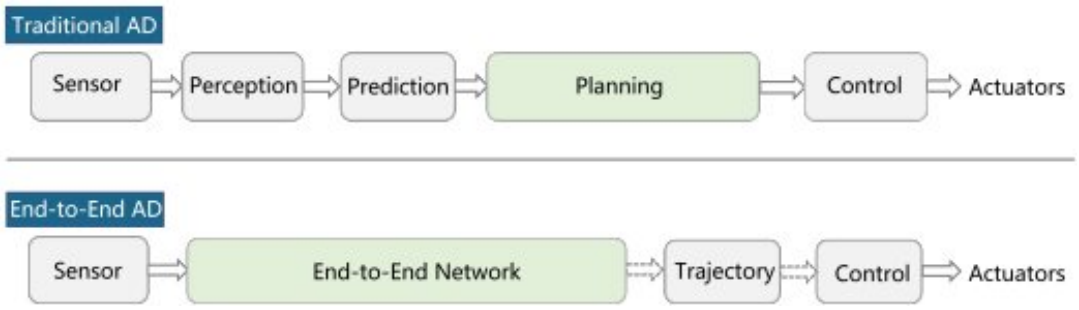


Figure 5: End-to-End AD model architecture [17]

Unlike traditional models where different modules perform separated tasks (perception, prediction, planning, etc.), the Supercombo model adopts an end-to-end network strategy [18], being responsible for extracting features from sensor data and planning the future trajectory of the vehicle [17] as shown in Figure 5.

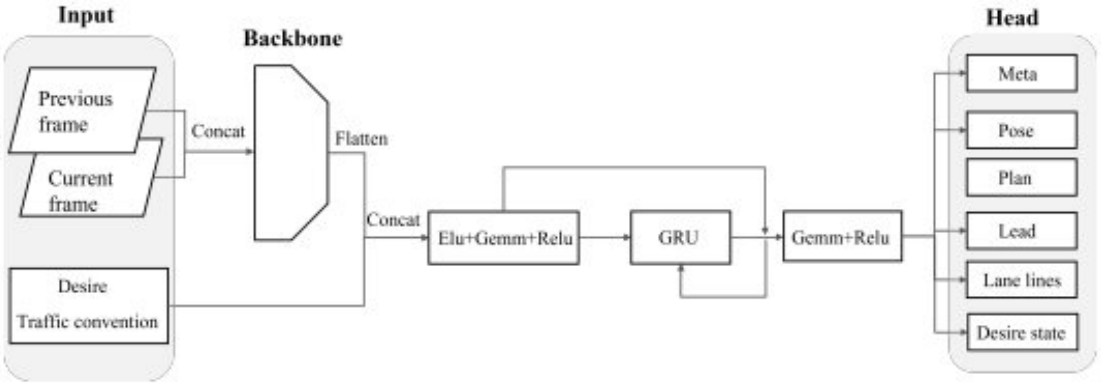


Figure 6: Supercombo model [17]

The model in Figure 6 is comprised by the following parts [17], described as follows:

- **Input**:
  - Preprocessed two consecutive frames, originally 3-channel RGB into 6-channel YUV format (YUV420) with size 12x128x256 (both frames)
  - Desire: Encoded vector that tells Openpilot models intended actions like lane changing, accelerating, etc... This is taken into account as a guide for model predictions, helping the backbone to choose the best future trajectory.
  - Traffic convention: one-hot encoded vector to tell the model whether traffic is right-hand or left-hand traffic
- **Backbone**: The backbone network is based on Google's EfficientNet-B2 [19]. The Gated Recurrent Unit (GRU), captures temporal information between frames and data.
- **Prediction Head**: The output includes 5 possible trajectories, among which the one with the biggest confidence is chosen. Also, Supercombo predicts lane lines, road edges, stop lines, position, speed of leads, and other relevant data.

The complete model architecture can be examined in Netron App by downloading the model from the official GitHub page.

Unfortunately, the training process and the massive collection of data of the Supercombo model are not available to the public [17]. For this reason, the training process is inferred, since GRU neural networks are based on a Recurrent Neural Network (RNN), which are trained with the backpropagation algorithm with specifically a variant called Backpropagation Through Time (BPTT) [20, 21], and

Efficient-B2 based on a CNN which is trained with the most popular algorithm, backpropagation [19], explained in subsubsection 2.4.5. The explanation of the training process will help to understand better how a DNN works for a better understanding of the research topic.

2.3 Bridge: Connecting CARLA and OpenPilot

The bridge is a python script that connects Openpilot's driving assistant with CARLA's virtual environment through the client API and Openpilot architecture's messaging modules.

While driving, Openpilot will receive signals from the virtual world through the signals from the actor's sensors that the bridge receives from CARLA as if they were from the real world and Openpilot will emit signals depending on the decision it makes at that moment regarding the situation of the world through the signals received. These signals emitted by Openpilot will be sent to the bridge and will interact with CARLA server to update the world status.

Figure 7: CARLA and Openpilot connected in a simulation

As seen in Figure 7, the CARLA and Openpilot windows are both interacting in the same virtual world.

2.4 Deep learning: Delving into Neural Networks

ML is a subset of AI that has revolutionized how algorithms work nowadays, being able to handle a large amount of data rapidly [1]. Unlike traditional programming, where a programmer explicitly writes the algorithm logic and provides inputs to get outputs, ML learns the relationship between provided inputs and outputs, crafting an algorithm to process new, unseen data [1] as shown in Figure 8.

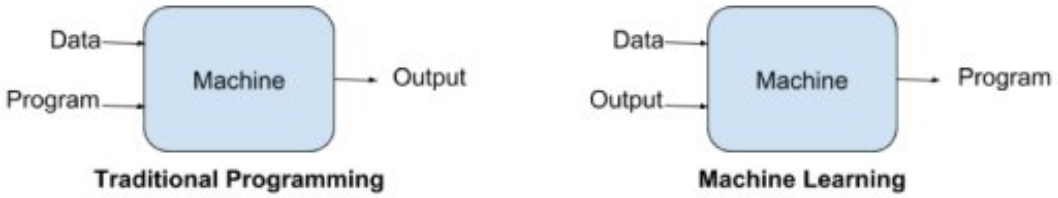


Figure 8: Traditional programming and Machine Learning paradigms [22].

The process of learning requires a large amount of data, it is called "training", and the performance of these algorithms can be evaluated to measure the confidence of their decisions, through a process known as "testing" as shown in Figure 9 using different evaluation metrics [23, 24]. Multiple research studies have demonstrated that their performance significantly surpass the traditional programming methods [25, 22].

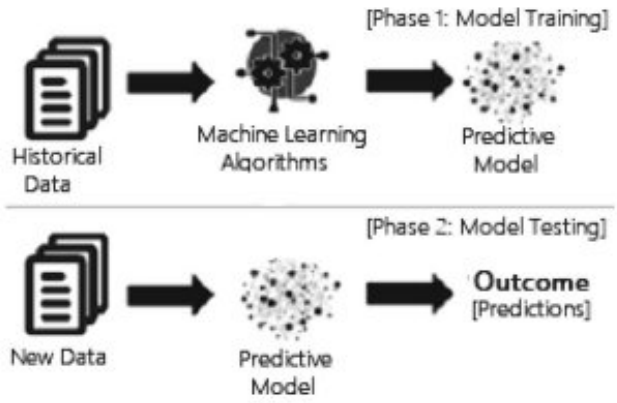


Figure 9: A general structure of a machine learning-based predictive model considering both the training and testing phase [1].

The collection of data used to train a ML model is called "dataset", and there are different types of ML algorithms according to the purpose and managed data type [1]. The most basic ones, shown in Figure 10 are described as follows:

- **Supervised Learning (SL)** [26]: This category of ML involves models that learn from labeled data. It includes:
  - **Binary classification**: This task involves categorizing the data into one of two classes.
  - **Multiclass classification**: Extends binary classification to multiple classes, where the model must choose one of several categories.
  - **Regression** [27]: Involves predicting a continuous value based on input data, through a linear or polynomial function.
- **Unsupervised Learning (UL)** [28, 29]: This type of ML aims to discover patterns in the input data and solve various problems without the guidance of labeled data. Clustering, which groups similar data points, is a common example.

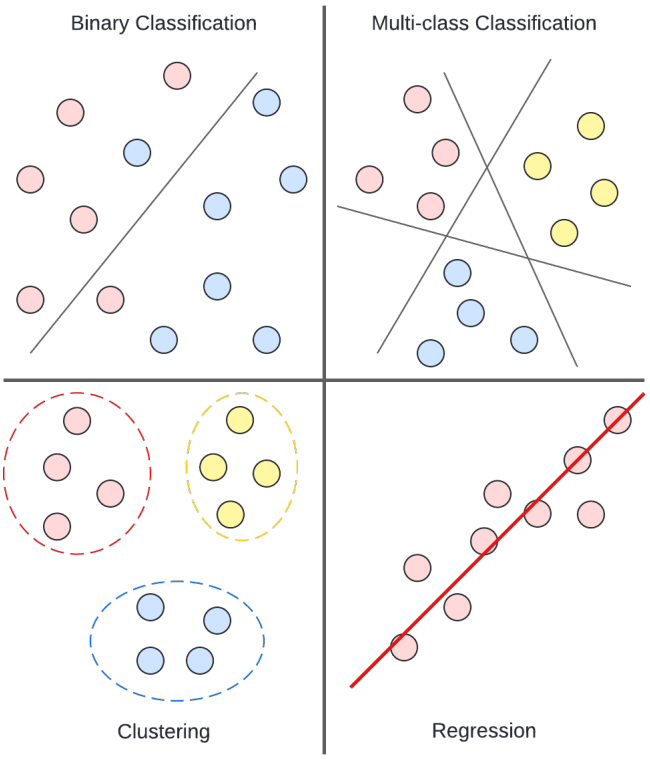


Figure 10: ML Model types: Binary classification, Multi-class Classification, Clustering, and Regression.

2.4.1 The Primitive Neuron: Perceptron

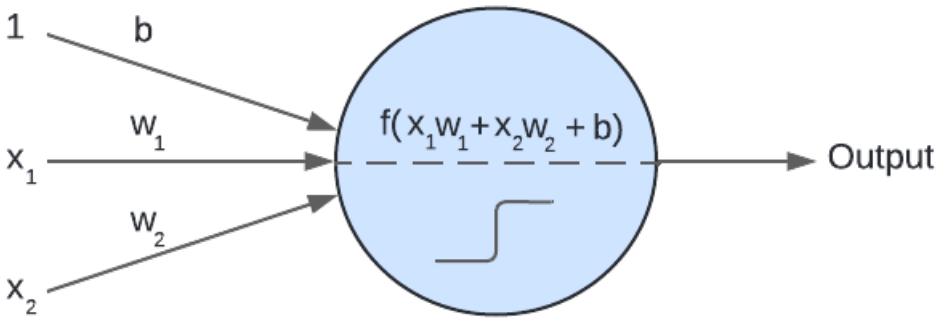


Figure 11: Single neuron: The perceptron

The perceptron in Figure 11, also known as neuron, is the most basic unit of computation in a neural network. It receives **inputs** (x) from other nodes or external sources and computes an **output** (y). To achieve this, each neuron is composed of some **parameters** [30]:

- **Weights** (w): Each input has an associated weight, which indicates how relatively important is respect other inputs.
- **Bias** (b): A constant value that shifts the polynomial function and guarantees successful learning of a pattern in the entire data space as shown in Figure 12.

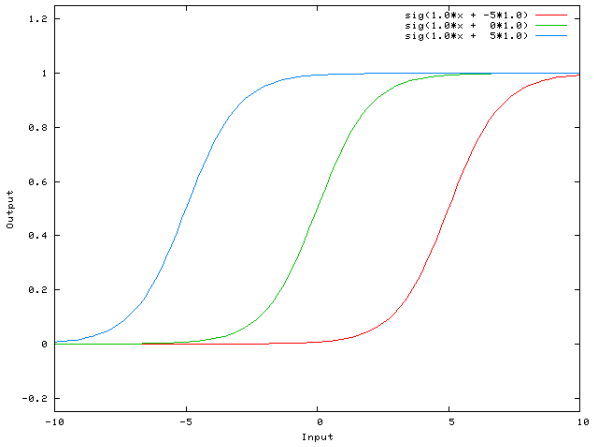


Figure 12: Example of polynomial function when bias is -0.5, 0 and 0.5 respectively [31]

To make a prediction, the neuron makes a weighted sum of the inputs and then applies an **activation function** (a) for non-linearity, thereby producing an output [30, 32]:

$$y = a(z) = a(\sum_i w_i \cdot x_i b)$$

where "a" is the activation function.

The activation function serves to introduce non-linearity, which is crucial since most real-world data is non-linear, enabling neurons to effectively learn these representations [33]. Each activation function has a different purpose:

- **Sigmoid**: The Sigmoid activation function is a smooth, S-shaped curve. It maps any real-valued number into the range (0, 1). The function is especially useful in binary classification problems where we need to output a probability value. The mathematical formula for the Sigmoid function is:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- **Softmax**: The Softmax activation function is used primarily in the output layer of neural networks for multi-class classification problems. It converts the raw output scores of a neural network into probabilities, which sum to 1. Each output value is exponentiated and then divided by the sum of the exponentiated values. The mathematical formula for the Softmax function for an input vector $\mathbf{x}$ is:

$$\text{Softmax}(\mathbf{x}_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

- **TanH**: The TanH (hyperbolic tangent) activation function is similar to the Sigmoid function but maps inputs to a range between -1 and 1. This can be beneficial as it centers the data around zero, which helps in making the optimization process more efficient. The mathematical formula for the TanH function is:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

- **ReLU**: The ReLU (Rectified Linear Unit) activation function is one of the most widely used activation functions in deep learning. It is simple and effective, introducing non-linearity by outputting zero for any negative input and the input itself for any positive input. This can significantly speed up the training process. The mathematical formula for the ReLU function is:

$$\text{ReLU}(x) = \max(0, x)$$

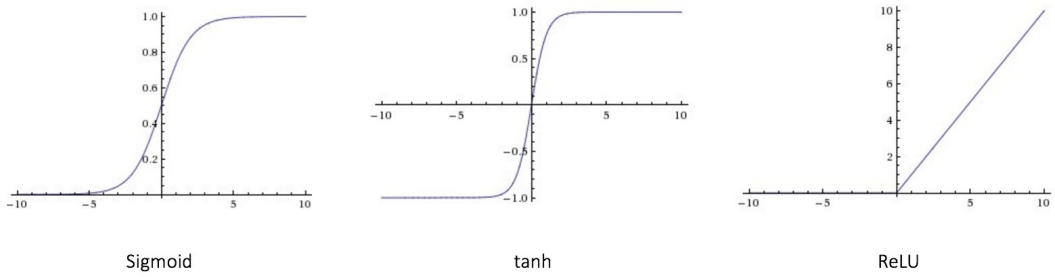


Figure 13: Usual activation functions in neurons

2.4.2 Deep Neural Networks

DL is a specialized area within ML, which is able to digest unprocessed data and learn patterns between given inputs and outputs. For this purpose, DL uses neural networks with many layers, also known as DNN to model complex patterns in data [34, 35, 36, 37].

A DNN is a data processing structure in which multiple nodes that process simple data, are interconnected to provide outputs. These networks were originally inspired by the workings of living brains [38], and although they are far from matching its capacity, they allow them to "learn" from large amounts of data [36].

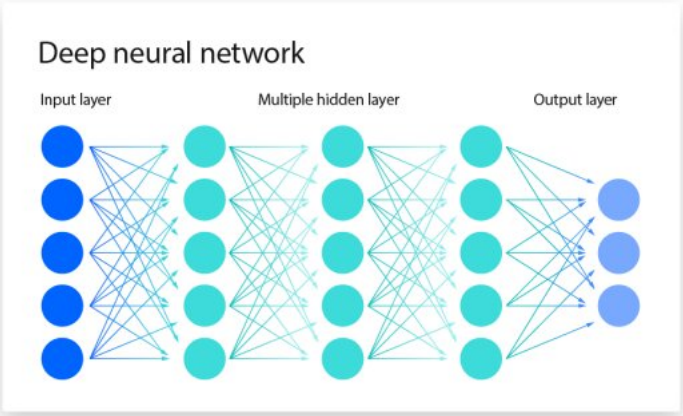


Figure 14: A Deep Neural Network's architecture.

Furthermore, DNN, neurons are organized into layers. Each layer receives information from the preceding layers, processes this data, and forwards it to the subsequent layer. Various types of layers exist to perform these functions [34, 39]:

- **Input Layer**: This is where the network receives its input data
- **Hidden Layers**: These are the intermediate layers between the input and output layers. There can be as many layers as the complexity of the neural network. Each layer has a specific goal that contributes to the overall function of the network.
- **Output Layer**: This layer produces the final output of the network, corresponding to the task it has been trained to perform.

2.4.3 Convolutional Neural Networks

As explained in subsection 2.2, Openpilot's Supercombo model is based on a CNN, which is a type of DNN specifically designed for processing structured grid data, such as images. CNNs are particularly effective due to their capability to recognize patterns in images such as, animals, stop signs, vehicles, persons, and so on... They usually have the following layers [40] as shown in Figure 15:

- **Input layer**: Input data, for example, an image.
- **Convolutional layer**: This layer applies a set of learnable filters to the input to create feature maps. Each filter scans across the input image and performs a dot product, capturing specific features such as edges, textures, or colors.

- **Pooling layer**: Used to reduce spatial dimensions of the feature maps, helping to avoid overfitting and reduce computational complexity.
- **Dropout layer**: Randomly disables a subset of neurons during training to prevent overfitting and encourage feature generalization.
- **Fully connected layer**: These layers are placed at the end of the CNN architecture, and serve crucial roles in pattern recognition derived from the features extracted by preceding convolutional and pooling layers. Each node in a fully connected layer receives input from all neurons of the previous layer, effectively integrating learned features across the entire input image. This comprehensive integration allows the network to make final predictions based on complex patterns and relationships in the data.
- **Output layer**: Like a traditional DNN output layer, each neuron, thanks to the activation function used, represents the probability that the given input belongs to that specific class.

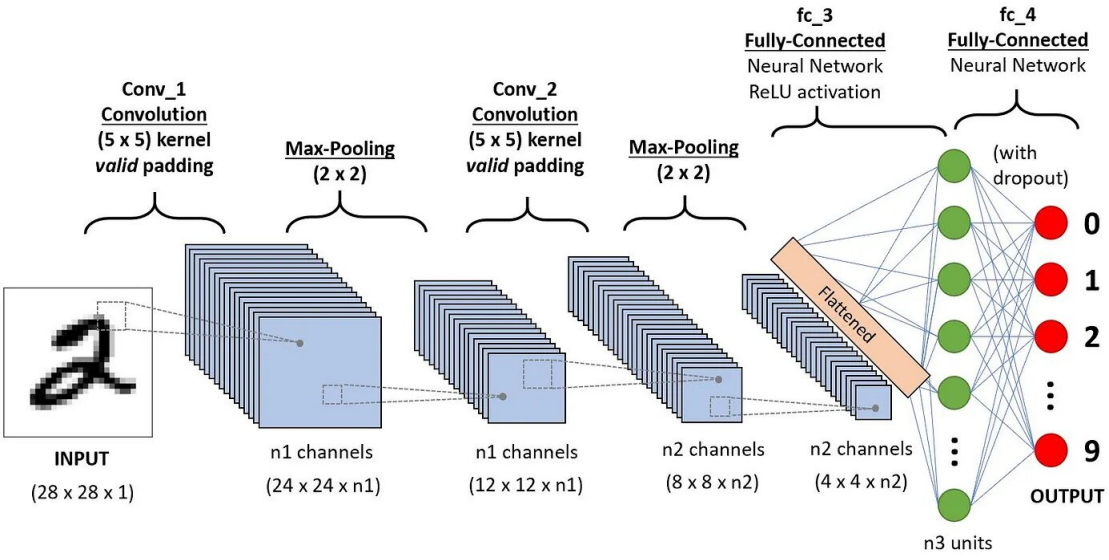


Figure 15: A Convolutional Neural Network's architecture.

2.4.4 Making a Prediction: Feed-Fordward

Feed-Fordward (FF) is a neural network process in which, given some inputs, the DNN passes all data to the following neurons through all the layers until obtaining a final output. [34, 41]:

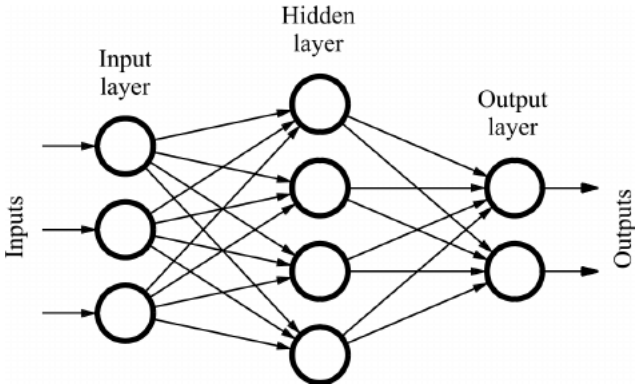


Figure 16: Feed-Fordward in DNNs

Normally, the activation functions of the output layers prepare a probability distribution to know which class is going to be selected [34]. The most common activation functions are sigmoid for binary classification and softmax for multi-class classification.

2.4.5 Model Training: Backpropagation

Back-Propagation (BP) is the process of **training** a classification DNN model based on a labeled dataset. Specifically, training a DNN involves optimizing numerous parameters (weights and biases) during supervised learning to improve the model's performance [42, 43].

The process involves making a FF pass for each dataset sample, calculating a **loss function**, which measures how far the DNN prediction is from the real label or "error committed", and then updating the model parameters based on committed error [44]. A widely used loss function is Cross-Entropy [45]:

$$\text{Binary Cross Entropy (BCE)}: \quad \text{BCE}(y, \hat{y}) = -\left[y \log(\hat{y}) + (1-y)\log(1-\hat{y})\right]$$

$$\text{Multi-Class Cross Entropy (MCE)}: \quad \text{MCE}(y, \hat{y}) = -\sum_{c=1}^{C} y_c \log(\hat{y}_c)$$

where y is the true label and $\hat{y}$ is the predicted label.

After computing the loss, the back-propagation algorithm updates the model parameters [44]. This update process involves several key concepts:

Gradients: Gradients represent the partial derivatives of the loss function with

respect to the model parameters. During back-propagation, these gradients are calculated using the chain rule and provide the direction and magnitude for adjusting the weights to minimize the loss. Mathematically, if L is the loss and w represents the weights, the gradient $\frac{\partial L}{\partial w}$ indicates how L changes with a small change in w [44].

Learning Rate: The learning rate η is a hyperparameter that controls the size of the steps taken to reach a minimum of the loss function. It determines how much the model's weights are adjusted with respect to the gradient. If the learning rate is too small, the model might converge very slowly. If it is too large, the model might overshoot the minimum and fail to converge [46].

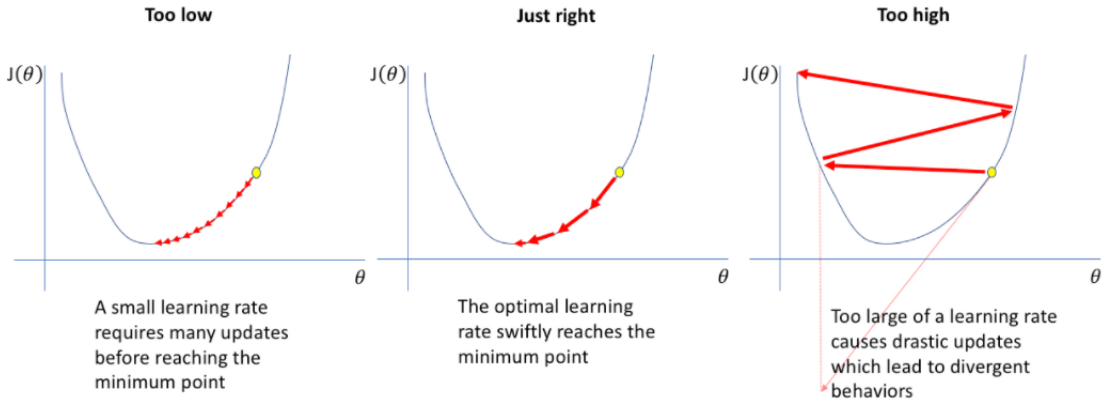


Figure 17: Choosing the right learning rate [47]

Update Rule: The typical update rule for updating the weights w in a neural network during back-propagation is [44]:

$$w_i^j \leftarrow w_i^j - \eta \frac{\partial L}{\partial w_i^j} \quad \forall\, w_i^j \in \text{DNN parameters}$$

where η is the learning rate, and $\frac{\partial L}{\partial w_i^j}$ is the gradient of the loss function with respect to each weight **i** in each layer **j**.

Epochs: An epoch is one complete pass through the entire training dataset. During training, multiple epochs are performed to ensure the model has adequately learned from the data. Choosing the right number of epochs is important to avoid underfitting or overfitting [34].

Batches: Instead of updating the model parameters after every single training example (stochastic gradient descent) or after processing the entire dataset (batch

gradient descent), a common practice is to use mini-batches. In mini-batch gradient descent, the dataset is divided into small batches, and the model parameters are updated after each batch. This approach balances the robustness of batch gradient descent and the computational efficiency of stochastic gradient descent. When all the batches are computed, we consider an epoch completed [34].

Batch Size: The batch size is the number of training examples in one forward/backward pass. A smaller batch size provides a more accurate estimate of the gradient but is computationally more expensive [34].

Momentum: Momentum is a technique used to accelerate the convergence of gradient descent by considering the past gradients to update the weights. The momentum term helps to navigate along the relevant direction and reduce oscillations [48].

Optimization Algorithms: Various optimization algorithms can be used to update the weights during back-propagation. Some of the popular ones include Stochastic Gradient Descent (SGD), Adam, RMSprop, and Adagrad. Each algorithm has its own advantages and is suitable for different types of problems [49, 50].

Fine-tune: Consists of a technique in which the last few layers of a pre-trained model are changed to be trained with new data, while the initial layers are frozen. The knowledge of the base model is transferred, known as "transfer learning" [51].

By iterating through these steps during each epoch and batch, the model gradually learns to minimize the loss function, improving its performance on the given task [52].

2.5 State of the Art: Adversarial Machine Learning

This section provides an overview of the literature on AEs targeting AD systems, including the state-of-the-art attacks utilized in this work.

DNN-based AD systems are very convenient, since they enable the autonomous driving of vehicles, from perceiving real-world status (other vehicles, persons, animals, traffic signs and lights, etc.) through sensors to taking appropriate actions such as braking, accelerating, or turning the vehicle as explained in subsection 2.2.

However, these systems are also safety-critical, as recent studies have demonstrated that adversarial attacks can be highly effective, exploiting the vulnerabilities in the neural network models used in AD systems [53, 54, 55, 56, 57]. Therefore, they

can result in erratic driving behaviors, potentially causing unsafe acceleration and rear-end collisions [58, 59].

Among all the existing adversarial attack techniques, this area of study is essential for ensuring the safety of AD systems. Researchers are working to develop mitigations, focusing on robust model training and the creation of new architectures designed to face such attacks [60, 61, 62, 63].

Despite these efforts, the rapid progression of attack methods frequently surpasses the development of defensive measures, posing a continuous challenge in safeguarding AD systems. Therefore, ongoing research and development are essential to reinforce the resilience of autonomous driving technologies against adversarial attacks [60, 61].

AML is a field of study focused on the robustness and vulnerabilities of machine learning DNN models against malicious attacks [64, 65]. An adversary (someone who intends to make an adversarial attack) tries to discover any weaknesses a model may have, manipulating and exploiting them [66].

2.5.1 Adversarial Examples

Adversarial examples are carefully crafted inputs designed to mislead DNNs predictions and fool them into making wrong classifications or object detections [67, 68]. Not only are being widely developed, but also being highly effective. Some examples where this kind of attack has been implemented:

- Traffic signs misclassification [69].
- AD system fooling [58, 59, 54, 53].
- Face recognition [70, 71, 72].
- Image classification [73].
- Object detection, such as persons, vehicles, etc. [74, 75, 76].

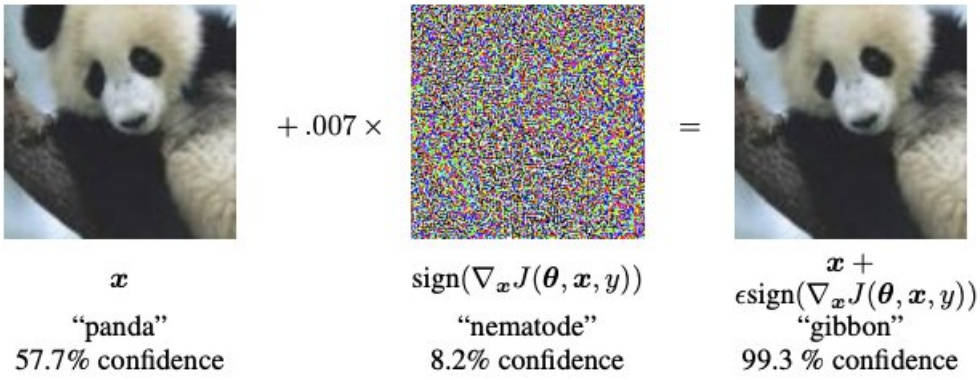


Figure 18: Adversarial Example [77]

As shown in Figure 18, an AE can be formally defined as follows:

$$x' = x + \lambda \cdot \delta$$

Being x the original image, δ a perturbation, λ the magnitude of the perturbation, and x' the adversarial example.

A perturbation is an intentionally crafted modification made to input data, designed to mislead a model and make incorrect predictions. The magnitude or strength of the perturbation determines how much the input data is altered and ensures that the changes remain imperceptible to the human eye.

Adversarial attacks can be classified into three main groups:

- **White-Box Attacks**: Full knowledge and access to the model parameters for otimizing the perturbation through the gradient descent [67, 66, 78].
- **Black-Box Attacks**: No access to the target model parameters, the perturbation has to be trained to minimize or maximize the loss function through an iterative method modifying the inputs and evaluating outputs [67, 79, 80].
- **Grey-Box Attacks**: Partial knowledge of the model but cannot use it during the attacki [81].

On the other hand, Adversarial Examples can be divided into two groups:

- **Targeted Adversarial Example**: The goal is generating a perturbation to make the DNN misclassify the input into a specific, incorrect label [67, 82].

- **Untargeted Adversarial Example**: The goal is generating a perturbation to make the DNN misclassify the input incorrectly, without a specific target [67].

2.5.2 FGSM: Fast Gradient Sign Method

The Fast Gradient Sign Method (FGSM) is one of the first methods presented for creating adversarial examples [66]. As shown in Figure 18, an adversary introduces a small perturbation to the input, which is scaled by a value λ to control its magnitude.

This method falls under the category of untargeted attacks, which means that the attacker does not choose a specific target for the desired result. Instead, the method chooses the closest class (the one that requires less perturbation for a misclassification) to deceive the model with.

To generate the perturbation, after choosing the model and the input example to perturb, first, the input is fed through the model, calculated the loss function L (See subsubsection 2.4.5) and calculates the gradients of L with respect to the input (See Equation 1):

$$\nabla_{\mathbf{x}} L(\theta, \mathbf{x}, y) \tag{1}$$

Then, the sign of the gradient is used to determine the direction in which the loss function increases the most (See Equation 2):

$$sign(\nabla_{\mathbf{x}} L(\theta, \mathbf{x}, y)) \tag{2}$$

Lastly, the result is scaled to control its magnitude and added to the original image to obtain the adversarial example (See Equation 3):

$$\mathbf{x\_adv} = \mathbf{x} + \lambda \cdot sign(\nabla_{\mathbf{x}} L(\theta, \mathbf{x}, y)) \tag{3}$$

FGSM consists of a single iteration, also called *single shot* [66]. A variant of this method was also proposed, applying an iterative perspective called Iterative Fast Gradient Sign Method (I-FGSM) [65]. Moreover, the authors suggest using this method for maximizing the likelihood of a chosen class, making it a targeted attack (where a specific class is chosen to be the result of the adversarial example instead of just deceiving the model with any class except the correct one). This method takes the base calculations of FGSM to calculate the perturbations; however, instead of directly adding the perturbation to the image and returning the adversarial example, it iterates through the result adding numerous perturbations to the image, trying to get the optimal one (See Equation 4):

$$\mathbf{x\_adv}_{t+1} = \mathbf{x} + \lambda \cdot sign(\nabla_{\mathbf{x\_adv}_t} L(\theta, \mathbf{x\_adv}_t, y)) \tag{4}$$

2.5.3 Carlini&Wagner

The CW attack [78] is a state-of-the-art adversarial attack based on optimization that aims to find a minimal perturbation that can fool a target model. The authors propose three attacks, each based on a different distance metric. Moreover, compared to previous adversarial example generation algorithms, these attacks are often much more effective (and never worse) [78]. Here the adversary optimizes the objective function iteratively, adjusting the perturbation to minimize the magnitude while maximizing the attack's success rate.

Considering the equation of the original optimization problem (See Equation 5):

$$\begin{aligned} &\text{minimize } D(\mathbf{x}, \mathbf{x}+\delta) \\ &\text{such that } C(\mathbf{x}+\delta) = t \\ &\mathbf{x}+\delta \in [0,1]^n \end{aligned} \tag{5}$$

Being D the function used to quantify the distance between the original input and the perturbed one, C the classification function used. And lastly, ensuring that the adversarial example produced is valid (in the case of images, all the values need to be constrained between 0 and 1 for all the color channels). As the classification function $C(x+\delta) = t$ is highly non-linear, making the optimization problem is really difficult to solve with previously existing algorithms.

That is why the authors proposed a different approach (See Equation 6):

$$\begin{aligned} &\text{minimize } D(\mathbf{x}, \mathbf{x}+\delta) \\ &\text{such that } f(\mathbf{x}+\delta) \leq 0 \\ &\mathbf{x}+\delta \in [0,1]^n \end{aligned} \tag{6}$$

By changing the classification function to an objective function, the non-lineality problem is solved. Then they proceed to reformulate the problem introducing the function on the minimization problem and multiplying it by a constant c to ensure that when performing the gradient descent, none of the terms takes precedence over the other (See Equation 7):

$$\begin{aligned} &\text{minimize } D(\mathbf{x}, \mathbf{x}+\delta) + \ c \cdot f(\mathbf{x}+\delta) \\ &\mathbf{x}+\delta \in [0,1]^n \end{aligned} \tag{7}$$

Lastly, they replace the distance metric D with the L_p norm of δ (See Equation 8):

$$\begin{aligned} &\text{minimize } \|\delta\|_p + \ c \cdot f(\mathbf{x}+\delta) \\ &\mathbf{x}+\delta \in [0,1]^n \end{aligned} \tag{8}$$

The authors proposed and tested numerous objective functions, specified in Equation 9:

$$
\begin{aligned}
f_1(\mathbf{x\_adv}) &= -\text{loss}_{F,t}(\mathbf{x\_adv}) + 1 \\
f_2(\mathbf{x\_adv}) &= (\max_{i \neq t}(F(\mathbf{x\_adv})_i) - F(\mathbf{x\_adv})_t)^+ \\
f_3(\mathbf{x\_adv}) &= \text{softplus}(\max_{i \neq t}(F(\mathbf{x\_adv})_i) - F(\mathbf{x\_adv})_t) - \log(2) \\
f_4(\mathbf{x\_adv}) &= (0.5 - F(\mathbf{x\_adv})_t)^+ \\
f_5(\mathbf{x\_adv}) &= -\log(2F(\mathbf{x\_adv})_t - 2) \\
f_6(\mathbf{x\_adv}) &= (\max_{i \neq t}(Z(\mathbf{x\_adv})_i) - Z(\mathbf{x\_adv})_t)^+ \\
f_7(\mathbf{x\_adv}) &= \text{softplus}(\max_{i \neq t}(Z(\mathbf{x\_adv})_i) - Z(\mathbf{x\_adv})_t) - \log(2)
\end{aligned}
\tag{9}
$$

As shown in Figure 19, they made a comparison among the proposed objective functions and finally realized that the 6th function was the best and they chose that for their attacks.

	Best Case						**Average Case**						**Worst Case**					
	Change of Variable		Clipped Descent		Projected Descent		Change of Variable		Clipped Descent		Projected Descent		Change of Variable		Clipped Descent		Projected Descent	
	mean	prob	mean	prob	mean	prob	mean	prob	mean	prob	mean	prob	mean	prob	mean	prob	mean	prob
f_1	2.46	100%	2.93	100%	2.31	100%	4.35	100%	5.21	100%	4.11	100%	7.76	100%	9.48	100%	7.37	100%
f_2	4.55	80%	3.97	83%	3.49	83%	3.22	44%	8.99	63%	15.06	74%	2.93	18%	10.22	40%	18.90	53%
f_3	4.54	77%	4.07	81%	3.76	82%	3.47	44%	9.55	63%	15.84	74%	3.09	17%	11.91	41%	24.01	59%
f_4	5.01	86%	6.52	100%	7.53	100%	4.03	55%	7.49	71%	7.60	71%	3.55	24%	4.25	35%	4.10	35%
f_5	1.97	100%	2.20	100%	1.94	100%	3.58	100%	4.20	100%	3.47	100%	6.42	100%	7.86	100%	6.12	100%
f_6	1.94	100%	2.18	100%	1.95	100%	3.47	100%	4.11	100%	3.41	100%	6.03	100%	7.50	100%	5.89	100%
f_7	1.96	100%	2.21	100%	1.94	100%	3.53	100%	4.14	100%	3.43	100%	6.20	100%	7.57	100%	5.94	100%

Figure 19: Evaluation of all combinations of one of the seven possible objective functions [78]

The perturbation is mapped to the hyperbolic tangent space to ensure that the generated images are within a valid input range (See Equation 10). They decided to use this space for two main reasons [78]:

- **Constraint satisfaction**: : The tanh function naturally satisfies the constraints applied to the perturbation on CW attacks, such as L_2-norm or L_∞-norm bounds, ensuring that the perturbation remains within the desired range. The tanh function restricts its output values between -1 and 1, which aligns with the constraints and prevents the perturbation from becoming excessively large.
- **Smooth mapping**: The tanh function provides a continuous and smooth mapping between the perturbation and input spaces. The function's differentiability facilitates the gradient computation and enables efficient optimization of the objective function.

$$\delta_i = \frac{1}{2}(\tanh(\mathbf{w}_i) + 1) - \mathbf{x\_adv}_i \tag{10}$$

Being w the variable to be optimized.

They presented three possible approaches for the attack using different L_p norms:

- **L_0-norm**: Measures the number of non-zero elements in a vector. In this context, it represents the number of pixels modified in the perturbation. By minimizing the L_0-norm, the attack seeks to change as few pixels as possible, focusing on a few critical pixels. It makes it so that a few pixels go under a significant modification, thus making it clear to the human eye.
- **L_2-norm**: Computes the Euclidean distance between two vectors, which in this case, represents the pixel-wise differences between the original input and the adversarial example. By minimizing the L_2-norm the attack aims to make small changes across the overall pixels of the initial image. It distributes the perturbation more evenly across the input, minimizing the visual impact on the human eye.
- **L_∞-norm**: Calculates the maximum absolute difference between corresponding elements of two vectors. This method constrains the perturbation such that the absolute maximum change in any individual pixel value does not exceed a certain threshold. By minimizing the L_∞-norm, the attack aims to keep the largest pixel change across all pixels within the allowed limit.

A comparison between generated AEs with the three norms proposed can be appreciated in Figure 20.

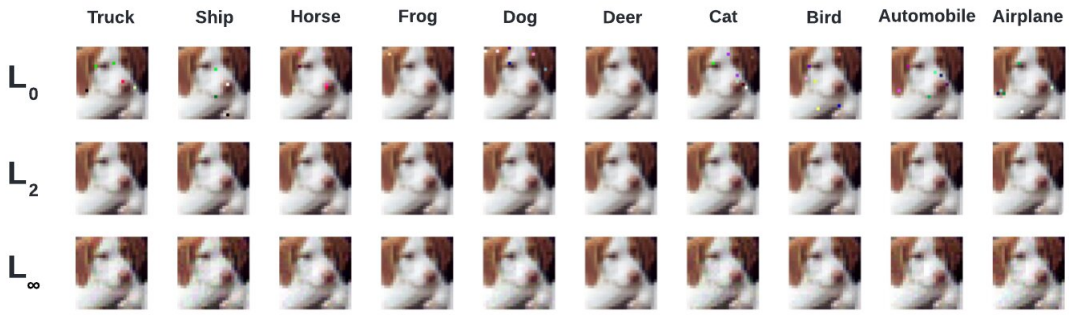


Figure 20: Examples of Adversarial images for each norm, starting from an image of a dog and converting it to each class of CIFAR-10 dataset [78]

2.5.4 Disappearance Attack

The disappearance attack is a type of AML attack that aims to fool object detection models to make objects "disappear" and not being recognized [58, 83]:

- **Targeted attack**: The goal is to specifically prevent the detection model from recognizing or detecting a particular class or type of object.
- **Untargeted attack**: The goal is to generally degrade the model's ability to detect any object correctly without specificity.

The general steps to implement this kind of attack are detailed as follows:

1. **Understand the model**: Gain knowledge of how the DNN processes inputs and outputs.
2. **Define Loss function**: The loss function aims to maximize the probability that the target object class is misclassified, effectively making it "disappear" from detection.
3. **Optimize the perturbation to minimize the Loss**: Use an iterative optimization method so that the model fails to detect the target object class, as shown in Algorithm 1.
4. **Evaluation**: After crafting the perturbed image through adversarial modifications, test it in the desired physical or simulated environment to assess the effectiveness of the attack.

Algorithm 1 Disappearance Attack

Input: model, image, target, epochs, lr
Step 1: Convert Image to Tensor with Gradient Tracking:
    image ← `torch.tensor(image, requires_grad=True)`
Step 2: Initialize Optimizer:
    optimizer ← `optim.Adam([image], lr=lr)`
Step 3: Start the Iterative Optimization Loop:
for $i = 1$ to epochs or loss threshold convergence **do**
    Step 4: Zero Out Gradients:
        optimizer.zero_grad()
    Step 5: Forward Pass (Compute Model Output):
        output ← model(image)
    Step 6: Compute Disappearance Loss:
        loss ← disappearance_loss(output, target)
    Step 7: Backpropagation (Compute Gradients):
        loss.backward()
    Step 8: Update the Image:
        optimizer.step()
end for
Step 9: Return the Final Modified Image:
    return image.detach()

2.5.5 Expectation Over Transform

EOT is a technique used in the context of adversarial examples to make them more robust and more effective in causing the ML models to make a misclassification [84, 85].

The main idea is applying some transformations and randomization (for example: weather status, lighting, ...) to the target area of the input image. Therefore, thanks to these scenarios, the adversarial example becomes more robust against defense methods [58].

The workflow of EOT is explained as follows [84]:

1. **Transformations definitions**: A set of transformations T are defined.
2. **Expected Adversarial Loss**: Is the average loss over all possible transformations. This expectation ensures that the adversarial example x' is optimized not just for a single input but for the distribution of inputs resulting

from the transformations. Mathematically, this can be expressed as:

$$\mathbb{E}_{t\sim T}\left[\text{Loss}(f(t(x')), y)\right]$$

Where t represents a transformation from T, x' is the perturbed adversarial example, f is the model and y is the target label.

3. **Optimization Process**: Typically involves iterative algorithms like gradient descent to minimize the expected adversarial loss. This can be expressed as:

$$x' \leftarrow x' - \eta \nabla_{x'} \mathbb{E}_{t\sim T}\left[\text{Loss}(f(t(x')), y)\right]$$

Where η is the learning rate, $\nabla_{x'}$ denotes the gradient descent with respect to x', and y is the target label.

2.5.6 (1+1) Evolution Strategy for Backbox Optimization

Evolutionary Computation (EC) is a subfield of AI that uses mechanisms inspired by biological evolution, such as reproduction, mutation, recombination, and selection, to solve complex optimization and search problems. This approach simulates the process of natural selection by generating a population of potential solutions and iteratively improving them based on their performance [86, 87, 80].

The (1+1) ES is a prominent method in the field of EC designed for optimizing problems following mimics of natural evolutionary processes such as mutation and selection [88].

Moreover, it operates with a simple concept where a single parent generates a single offspring per generation through mutation [89]. The algorithm can be summarized as follows:

Algorithm 2 (1+1) Evolution Strategy

```
Initialize x randomly
Loss ← Evaluate f(x)
while termination conditions not met do
    x_new ← Mutate x
    Loss_new ← Evaluate f(x_new)
    if Loss_new ≤ Loss then
        x ← x_new
    end if
end while
```

The selection process involves evaluating the fitness of both the parent and the offspring and selecting the better solution:

$$x_{\text{next}} = \begin{cases} x_{\text{new}} & \text{if } f(x_{\text{new}}) \leq f(x) \\ x & \text{otherwise} \end{cases}$$

Mutation in (1+1)-ES is typically implemented as a Gaussian perturbation, whilst Evaluation or Loss depends on the specific the goal of the perturbation generation. Gaussian mutation is a fundamental mechanism in ES, particularly effective in continuous optimization problems. This mutation process involves adding Gaussian noise to candidate solutions, thus exploring the solution space around the current point [90]. The mathematical expression for Gaussian mutation can be represented as follows:

$$x_{\text{new}} = x + \sigma \cdot N(0, I)$$

Where:

- x is the current solution vector.
- σ denotes the mutation strength or step size, which can be adaptive.
- $N(0, I)$ represents a vector of random values drawn from a standard normal distribution.

3 Characterization of models and datasets

3.1 Models

3.1.1 RESNET-50

Developed by Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, ResNet, or Residual Network, was introduced in their 2015 paper as a profound advancement in convolutional neural networks [91]. The specific variant, ResNet-50, is a image classification model with 50 layers deep and is part of the ResNet family which also includes models like ResNet-101 and ResNet-152, differing primarily in the depth of layers, represented by the digit attached to their name.

3.1.2 Supercombo

Li Chen described their efforts to replicate Openpilot's Supercombo model in their research [17]. As explained in subsection 2.2, it is the main DNN of the Openpilot AD system, which is responsible for predicting future trajectories of the controlled vehicle. Thus, enables Level 2 autonomous driving, capable of handling multiple critical driving functions such as steering, acceleration, and braking autonomously.

3.2 Datasets

3.2.1 CIFAR-10

Created in 2009 by Vinod Nair Alex Krizhevsky and Geoffrey Hinton [92], the Canadian Institute for Advanced Research (CIFAR) database has two official variants, CIFAR-10 and CIFAR-100. The difference between these two is the number of classes they contain, represented by the digit attached to their name. This FDP uses the CIFAR-10 version. The specific classes it contains are the following: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Each class includes 5,000 training images and 1,000 test samples. They are all 32×32 sized RGB 3-channel color images.

3.2.2 Comma2k19

Introduced by Harald Schafer, the comma2k19 dataset is a valuable resource for autonomous driving research [93]. This dataset includes approximately 33 hours of driving data collected from natural driving conditions across California's 280 highway. The comma2k19 dataset is especially useful for developing and testing

algorithms under real-world conditions as it captures a wide array of driving scenarios. It features comprehensive data from multiple sensors such as cameras and GPS, which are essential for analyzing vehicle behavior. Researchers and developers use this dataset to refine perception systems and advance autonomous driving technologies.

4 Project Management

Before starting to develop the project goals, it is very important to elaborate a planning, identifying which tasks are going to be developed, project cost estimation, possible risks to face, needed equipment, etc.

4.1 Project Scope

This project's scope gathers all the necessary tasks to gradually complete the project. A Work Breakdown Structure (WBS) shown in Figure 21 has been developed, in which primary tasks and subtasks are listed:

Figure 21: WBS diagram planning

4.2 Temporal Planning (GANTT)

In this section, the project planning is detailed through a GANTT chart in Figure 22, in which we can appreciate how long each task will take to complete along the project's duration.

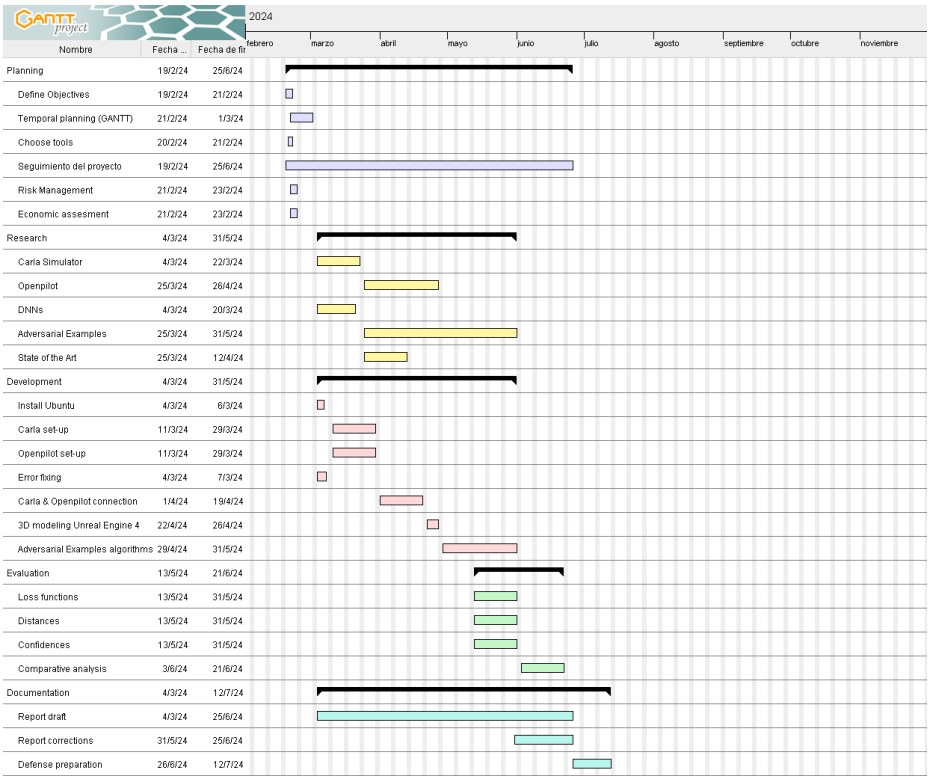


Figure 22: GANNT planning timeline

The following is a GANTT chart showing the actual time required for each task to be completed. As shown in Figure 23 the estimated project's duration has been extended by 1 month. Mainly, it can be seen how the setup of the programs took longer due to the difficulty of setting them up, as well as the research process of the Openpilot architecture, which was more complex than expected.

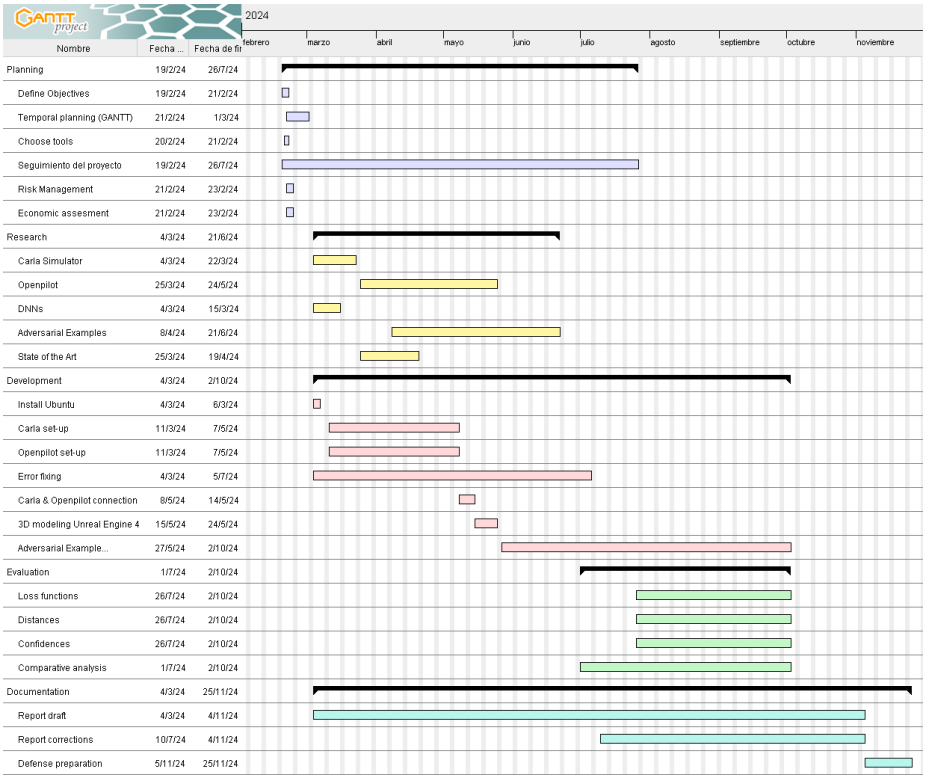


Figure 23: GANTT real timeline

4.3 Project Management Methodology

During the development of this project, an agile methodology, specifically a variant of **Scrum**, has been employed. Weekly meetings were conducted between the student and the tutor to ensure continuous progress and alignment. OneNote was utilized to document the tasks completed each week, as well as to outline the tasks to be undertaken in the upcoming week. In the **Appendix C** are shown all the weekly meetings contents.

4.4 Risk Management

During the project's development, some adversary events can occur by deferring its duration or making it impossible to carry out some tasks. That is why it is very important to make a risk management plan before starting to develop it. It is necessary to identify and manage all possible risks so that their impact is lower than if they appeared by surprise.

For that, the visual diagram in Figure 24 summarizes prevented risks.

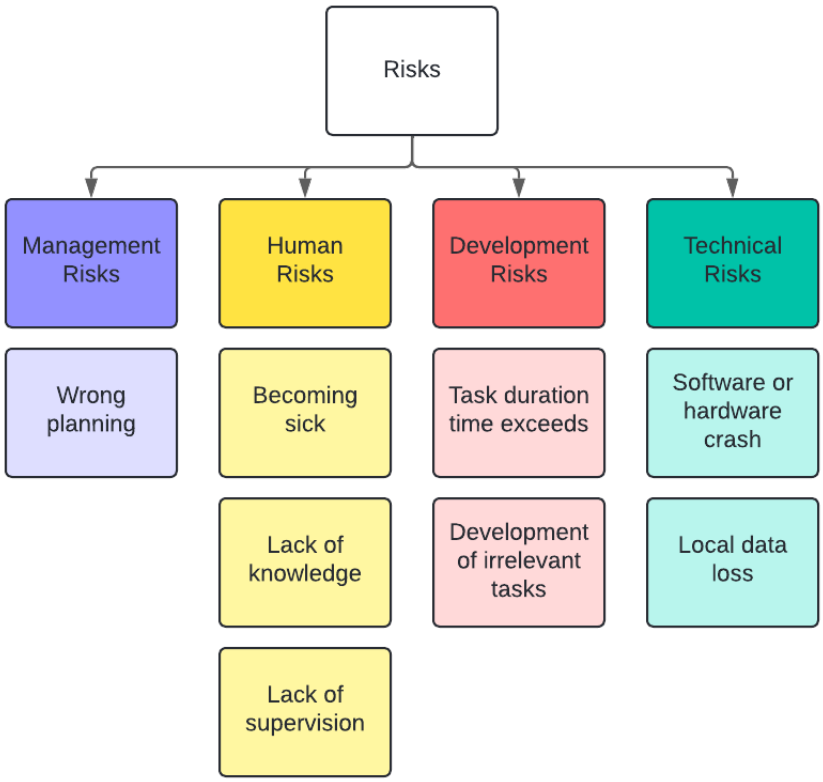


Figure 24: Possible risks during the project's development

The explanation of each risk is going to be done using the table model in Table 9.

Risk name	
Description	Description of the risk
Preventive plan	What to do for minimizing probabilities to happen
Contingency plan	What is going to be done if the risk happens
Probability	Very low, Low, Medium, High, Extreme
Impact	Very low, Low, Medium, High, Extreme

Table 1: Descriptive table model for risk management.

Wrong planning	
Description	The project's planning is wrongly focused
Preventive plan	Periodic supervision and analysis if the project is following a good course
Contingency plan	Stop doing unnecessary tasks, try to reuse developed software or use learned knowledge in the new focus
Probability	Medium
Impact	Extreme

Table 2: Chart for wrong planning risk.

Becomming sick	
Description	Contract a disease and become illness for a period of time
Preventive plan	Good habits, feeding, no stress, sport and have fun with friends
Contingency plan	Take care and try not to neglect the project as much as possible
Probability	Medium
Impact	Medium

Table 3: Chart for becoming sick risk.

Lack of knowledge	
Description	Start developing algorithms without necessary knowledge
Preventive plan	Be sure that the approach to be implemented is feasible and argue it with trustable information (e.g. books, papers, etc.)
Contingency plan	Evaluate situation, research more and focus again on the approach, trying to reuse as much work as possible
Probability	Medium
Impact	High

Table 4: Chart for lack of knowledge risk.

Lack of supervision	
Description	The project's trajectory is not being evaluated and going in the wrong direction
Preventive plan	Weekly meetings
Contingency plan	Talk with the tutor to make more meetings and make periodic reviews
Probability	Low
Impact	High

Table 5: Chart for lack of supervision risk.

Task duration time exceeds	
Description	Description of the risk
Preventive plan	What to do for minimizing probabilities to happen
Contingency plan	What is going to be done if the risk happens
Probability	Very low, low, medium, high, extreme
Impact	Very low, low, medium, high, extreme

Table 6: Descriptive table model for risk management.

Development of irrelevant tasks	
Description	Time is employed in unnecessary tasks
Preventive plan	Robust project's tasks planning
Contingency plan	Try to reuse as much work as possible and focus again on the tasks
Probability	Low
Impact	Medium

Table 7: Chart for development of unnecessary tasks risk.

Software or hardware crash	
Description	Facing errors during the project
Preventive plan	knowledge about official docs of used tools
Contingency plan	Fix the errors using official docs
Probability	Low
Impact	Medium

Table 8: Chart for software or hardware crashing risk.

Local data loss	
Description	Loss of vital information
Preventive plan	Backups
Contingency plan	Recover last backup
Probability	Very low
Impact	Extreme

Table 9: Descriptive table model for risk management.

4.5 Economic Assessment

After analyzing the scope, planning, and possible risks of the project, this section specifies which resources are going to be used during the development of the project, breaking down all the estimated costs and the total cost of carrying out the project forward.

4.5.1 Software

Teams: Collaboration platform developed by Microsoft that combines chat, video meetings, file storage, and integration with other Microsoft Office apps and services to facilitate workplace communication and teamwork
Price: 6.99€/month

OneNote: Digital note-taking application by Microsoft that allows users to create, organize, and share notes across multiple devices.
Price: 6.99€/month

PowerPoint: Presentation software by Microsoft used to create, edit, and display slide-based presentations.
Price: 6.99€/month

Overleaf (LaTeX): Overleaf is an online LaTeX editor that enables users to collaboratively write, edit, and publish scientific documents.
Price: 9.99€/month

Ubuntu: Popular, user-friendly open-source Linux operating system
Price: Free

Rufus: Tool for creating bootable USB drives from ISO files.
Price: Free

Draw.io: Online diagramming tool used to create flowcharts, network diagrams, and other visual representations.

Price: Free

Lucidchart: Online platform for creating and sharing diagrams, flowcharts, and other visual representations collaboratively.
Price: Free

Visual Studio Code: Open-source code editor by Microsoft with support for debugging, version control, and extensions.
Price: Free

Python: Versatile, high-level programming language known for its readability and broad applicability in web development, data science, automation, and more.
Price: Free

PyTorch: Open-source machine learning framework developed by Facebook's AI Research lab, known for its dynamic computational graph and support for deep learning models.
Price: Free

Openpilot: Open-source software platform for advanced driver assistance systems (ADAS) and automated driving features in vehicles.
Price: Free

Carla simulator: Open-source simulator for autonomous driving research and development, providing realistic environments and scenarios for testing algorithms and vehicles.
Price: Free

Google scholar: Web search engine specifically designed to locate academic papers and scientific documents across a wide range of disciplines.
Price: Free

Total software cost: $6.99€ \cdot 5 + 9.99€ = 44.94€$

4.5.2 Hardware

Laptop: Provided laptop by the company for everyday work.
Price: 470€

Desktop Computer: Used desktop computer for development. Specifications are detailed as follows:
Price: 1473€

- **Intel(R) Xeon(R) E-2124 CPU @ 3.30GHz**: CPU component of the desktop computer.

Price: 285€

Quadro RTX 4000 (8GB): GPU component of the desktop computer.
Price: 1078€

RAM (32GB): RAM component of the desktop computer.
Price: 70€

HHD (1TB): HHD memory component of the desktop computer.
Price: 40€

Screen: Two screens of 27" for the desktop computer.
Price: 263,78€ x 2 = 527.56€

Total hardware cost: 470€ + 1473€ + 527.56€ = 2470.56€

4.5.3 Salary

Salary: Based on the *Instituto Nacional de Estadística (INE)* statistics, an estimation as a professional scientist researcher in a part-time job for 5 months.
Salary: 1.349,46€ · 5 = 6747.3€

4.5.4 Indirect Costs

Indirect Costs: Generally, the evaluation of the indirect cost is made based on previous similar works. Since no history of similar works is available, the Indirect Cost Rate (ICR) is going to be inferred. It is commonly accepted that indirect costs represent between 2% and 5% of direct project costs. The mean is going to be considered as the ICR in this FDP: 3.5%.

Considering the following costs:

- **Software**: 44.94€
- **Hardware**: 2470.56€
- **Salary**: 6747.3€

Indirect cost: 9262.8€ · 0.035 = 324.198€

4.5.5 Total Costs

Altogether, the costs of this project are detailed in Table 10:

Economical breakdrown	
Software	44.94€
Hardware	2470.56€
Salary	6747.3€
Indirect costs	324.20€
Total	**9587€**

Table 10: Breakdown of total economical costs of the FDP.

5 Development

This section provides detailed information about the followed steps during the development of the FDP. This section aims to implement some AE attacks against Openpilot based on research in section 2 and develop a methodology as a user guide for someone who wants to learn and reproduce these attacks.

5.1 Workflow

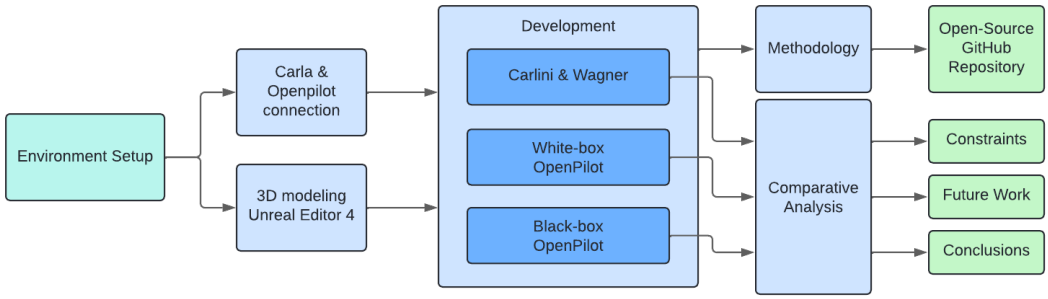


Figure 25: Development's process workflow

To facilitate the understanding of the overall development process, the Figure 25 shows a visual workflow diagram. The followed steps are detailed as follows:

- **Enviromental setup**: Install Carla simulator and Openpilot AD system software successfully, fixing lots of installation problems.
- **Bridge connection**: Successfully establish a connection between software and enable Openpilot to control a vehicle during a simulation. For that, vehicles have been spawned and some sensors configured.
- **3D modeling with Unreal Editor 4**: Learn how to use Unreal Editor 4, and develop 3D modeling skills to customize the rear appearance of a vehicle.
- **Implementation of AE attacks against Openpilot**: Based on the research made in subsection 2.5, some white-box and black-box scenarios have been formulated, and then implemented considering limitations in subsection 7.2 to test the robustness of Openpilot.
- **Comparative Analysis**: Assessment of the different developed approaches, highlighting the strengths and weaknesses of each.

- **Methodology**: Generical step-by-step guide to implement AE black-box and white-box scenarios against Openpilot.

5.2 Project Setup

To develop algorithms to craft AEs against OpenPilot, we must first set-up the autonomous driving environment. The right procedure is explained step by step in **Appendix A**.

First of all, when this project started, a basic setup was done by another previous internship student, but this setup included a precompiled version of Carla, which was not useful, since source code, not a compiled version, is needed for Unreal Editor 4.26.2 be able to open Carla project in editor mode and access to world's assets. During a period of time, research about how to customize the appearance of an Actor with the Python client has been performed, but it was impossible and Unreal Editor is needed.

During the set-up process, Ubuntu OS has been installed twice: versions 18.04 and 20.04, since different versions of Carla and Openpilot use different versions of Ubuntu. Finally, Ubuntu 20.04 was chosen, because Carla 0.9.14 and Openpilot 0.9.4 versions both are integrated with that Ubuntu version.

5.3 Carla and Openpilot Connection

To establish a communication between Carla and Openpilot, a Python script called bridge.py (explained in subsection 2.3) is provided by the source code of Openpilot.

This script has been improved to append new functionalities:

- Spawn a vehicle in the simulation world that is engaged by Openpilot, to be driven anonymously in Listing 6:
- Spawn a vehicle to be used as an adversary, with the AE in the rear in Listing 7:
- Function to spawn new random vehicles, and be able to apply Traffic Manager settings in Listing 8:
- Function to apply Traffic Manager rules, only are applied if the Openpilot vehicle is engaged, to make the adversarial vehicle start driving at the same time Listing 9:

5.4 3D Modeling: Unreal Editor 4.26.2

To customize the appearance of any object in Unreal Editor, materials are used. The rear side of the vehicle has been customized in two different ways:

- **Changing the appearance of the Static Mesh**: This approach involves adapting the material size to the back dimensions as shown in Figure 26, and then the material is applied into the back as shown in Figure 27. The physics of the Static Mesh needs to be disabled.

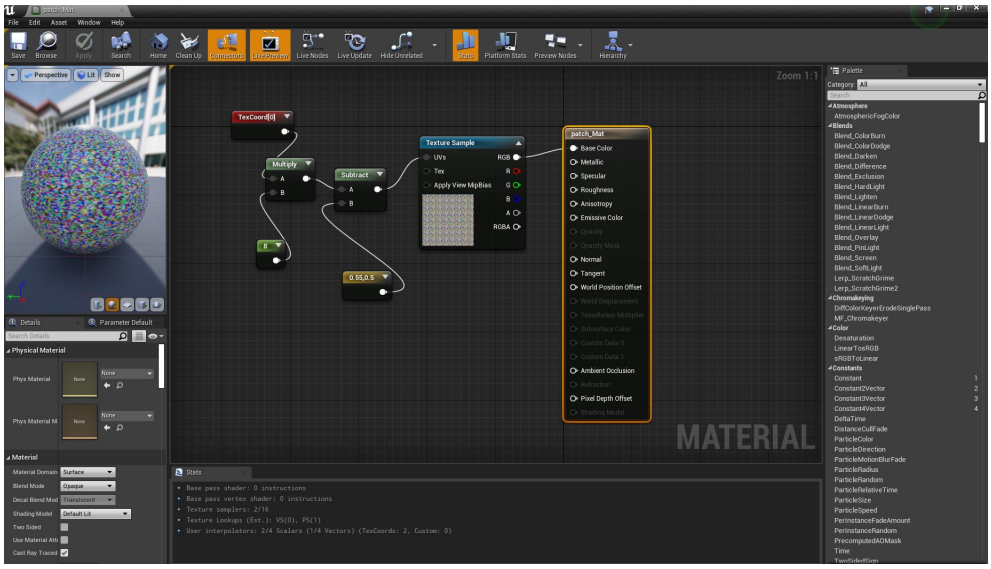


Figure 26: Material hierarchy which adapts the size of an image to the rear of a vehicle.

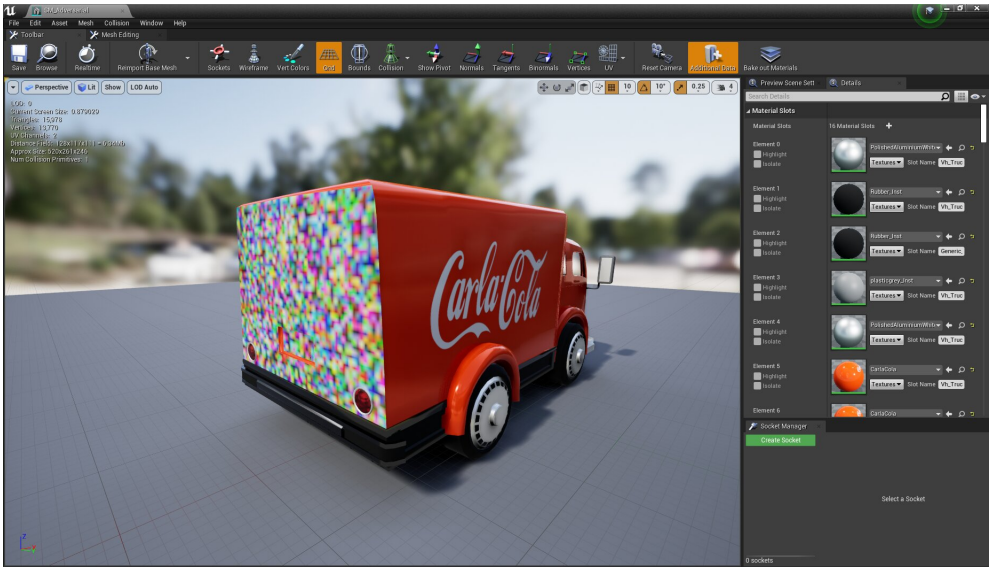


Figure 27: Customized vehicle applying material in Figure 26.

- **Adding a Plane to the rear of the vehicle with an image**: This approach involves appending a plane to the back of the vehicle, and then applying created material in Figure 28 to the plane as shown in Figure 29. The physics of the Plane needs to be disabled.

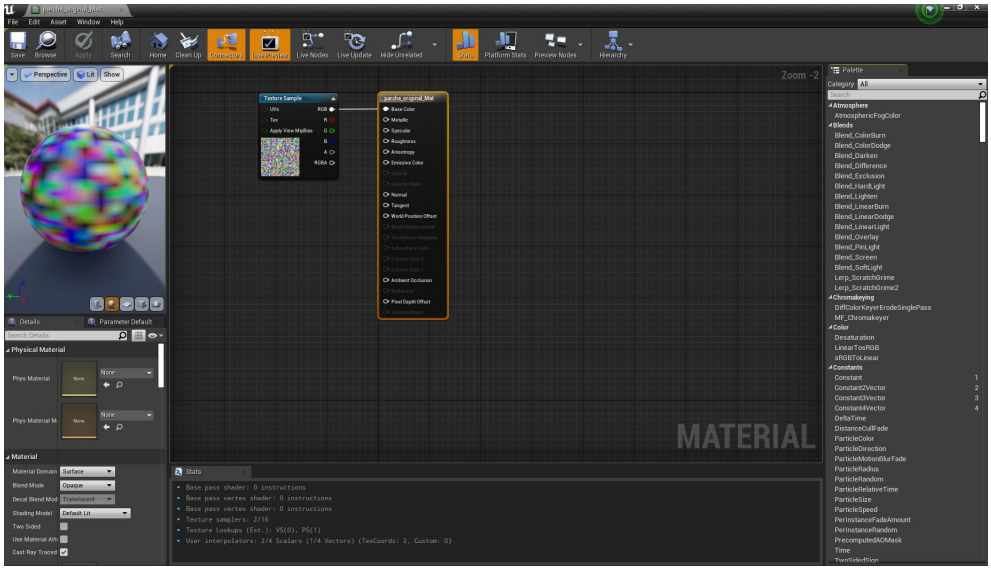


Figure 28: Material hierarchy which adapts the size of an image to a plane.

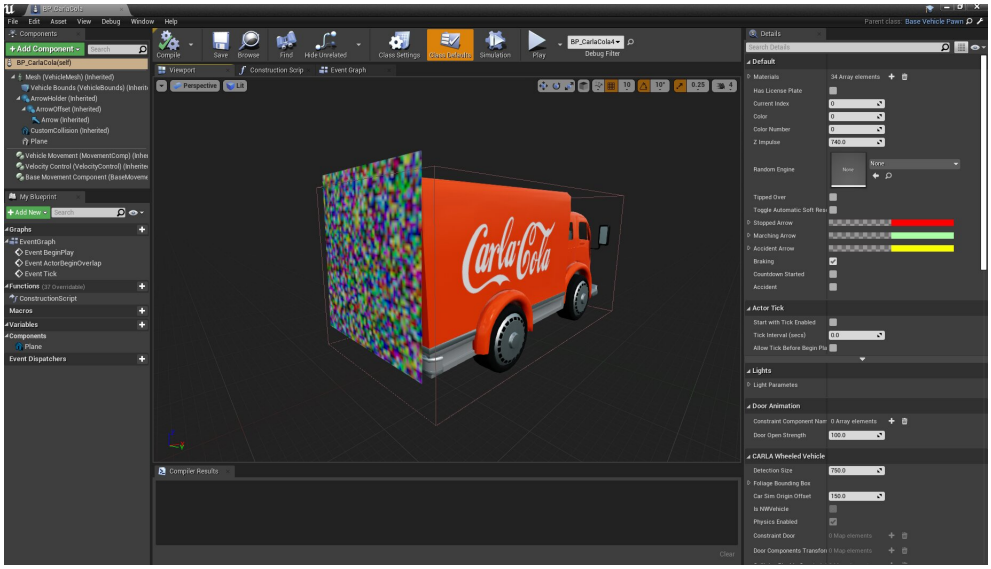


Figure 29: Customized vehicle applying material in Figure 28.

5.5 Attack 1: Carlini & Wagner White-box attack

After making the FGSM tutorial and getting used to AEs [66], this attack aims to learn how to implement and generate AEs against image classification models, using the CW algorithm [78] and the CIFAR-10 dataset [92] before deepening into Openpilot and trying to mislead a complex AD system.

The whole implemented code in a Jupyter Notebook can be found in the GitHub repository [94] and deeper explanations in [95].

Since attacking an object detection model is more difficult than attacking an image classifier, Shang-Tse Chen et al. [96] have demonstrated that a state-of-the-art object detection model like Faster R-CNN can be fooled using the CW algorithm.

This attack is focused on the L_2-norm for generating imperceptible AEs for the human eye. The followed steps are detailed as follows:

- **Dataset preparation**: The CIFAR-10 model is downloaded and prepared to train CNN models with 10 output classes: train, validation and test slipt.
- **Model training**: Two image recognition models have been used to attack:
  - Fine-tuned RESNET-50 image classification model.
  - Built and trained from scratch CNN model.

- **Optimization problem**: Since L_2-norm is used, the used optimization function is detailed in Equation 11:

$$\begin{aligned} &\text{minimize} \|\frac{1}{2}(tanh(\mathbf{w}_i)+1-\mathbf{x})\|_2^2 + c \cdot f(\frac{1}{2}(tanh(\mathbf{w})+1)) \\ &\text{with f defined as} \\ &f(x') = max(max\{Z(\mathbf{x'})_i; i \neq t\} - Z(\mathbf{x'})_t, -\kappa) \end{aligned} \tag{11}$$

  The variable κ is used to control how confident the adversarial example needs to be in order to be considered successful. Specifically, it sets a threshold for how much more confident the prediction for the adversarial target class must be compared to the actual class of the input. As all kappa values in original CW experiments were considered $\kappa = 0$ [78], the same is taken.

- **Iterative algorithm**: An iterative algorithm has been implemented, which gradually updates pixels of the perturbed image using the gradient descent algorithm with the Adam optimization, and loss function in Equation 11. The Figure 30 shows the CW attack against custom-built and trained CNN, and Figure 31 shows the CW attack against fine-tuned RESNET-50 model.

Figure 30: Truck being misled by custom-built and trained CNN. The perturbation is calculated with CW L_2-norm.

Figure 31: Truck being misled by fine-tuned RESNET-50 model. The perturbation is calculated with CW L_2-norm.

5.6 Attack 2: White-box attack against Supercombo model

The first approach against the Openpilot AD system consists of a white-box attack against the Openpilot's Supercombo model. The visual diagram in Figure 32 facilitates the workflow understanding formulated in this methodology.

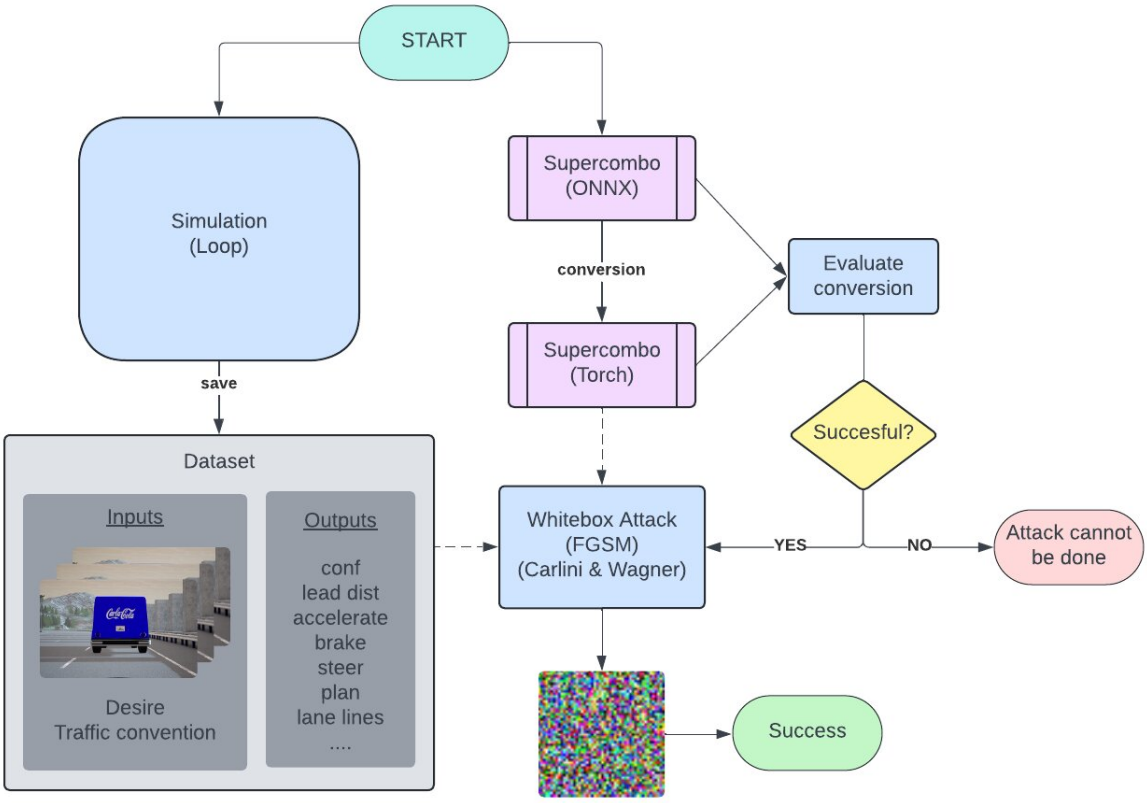


Figure 32: Visual diagram about the formulated workflow for the attack 1.

Accessing the Supercombo model's parameters makes it easier to calculate an AE input that misleads the model to obtain the desired output.

The whole implemented code in a Jupyter Notebook can be found in the GitHub repository: https://github.com/un4rch/openpilot_AdvExamples/blob/main/attacks/attack_whitebox.ipynb and deeper explanations in https://github.com/un4rch/openpilot_AdvExamples/blob/main/docs/openpilot_whitebox_attack.md.

The followed steps for this attack are detailed as follows:

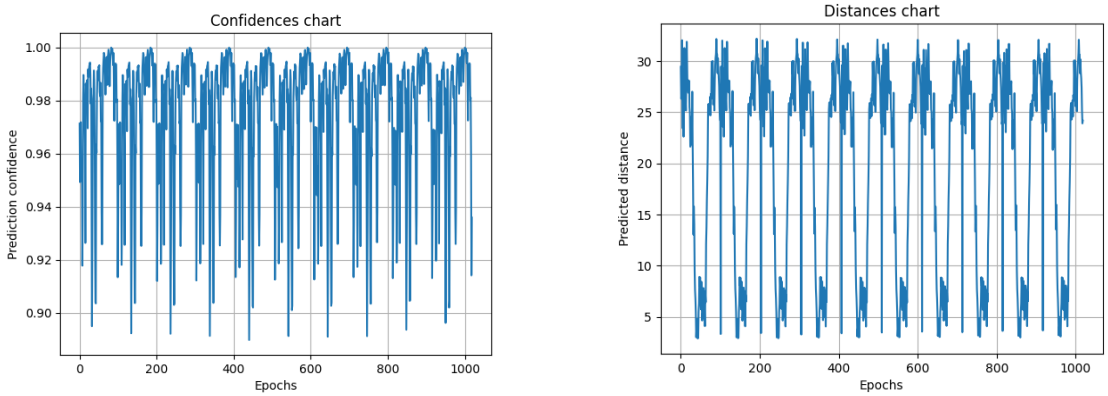
- **Setup the environment:** Install the required libraries such as ONNX, PyTorch, and ONNX Runtime. Convert the Openpilot Supercombo model from ONNX format to PyTorch for easier manipulation.
- **Data preparation:** Gather the driving frames from the simulation and prepare them as inputs to the model. Frames must be processed into the YUV4:2:0 format since this is the expected input by the Supercombo model.
- **Model query:** Perform model inference on the prepared frames with the adversarial patch applied. The model processes the frames and outputs predictions.
- **Disappearance loss**: The loss function in Equation 14 used to make the lead vehicle "disappear" for Openpilot's Supercombo model can be found in Listing 14, understanding explanations in subsubsection 2.5.4. This loss takes into account the following items [58]:
  - $s(z_i)$: Adversarial simulation prediction confidence.
  - $d(f_i)$: Real simulation (no AE placed in the lead vehicle) distance predictions.
  - $d(z_i)$: Adversarial simulation (AE placed in the lead vehicle) distance predictions.
  - AE smooth pattern.

$$\begin{aligned} L &= L_{conf} + \lambda_1 L_{dis} + \lambda_2 L_{TV} \\ L_{conf} &= -log(1 - s(z_i)) \\ L_{dis} &= \left| \frac{d(z_i)}{d(f_i)} \right| \\ L_{TV} &= \sum_{i,j} \left(|\delta_{i+1,j} - \delta_{i,j}| + |\delta_{i,j+1} - \delta_{i,j}| \right) \end{aligned} \tag{12}$$

  where $\mathbf{z}_i$ represents a frame with an AE placed, $\mathbf{f}_i$ represents the real frame with no AE, $\mathbf{d}$ represents detected distance, $\mathbf{s}$ the detection score and being $\lambda_1 = 0.01$ and $\lambda_2 = 0.001$.
- **Expectation over Transform**: The considered loss function in Equation 15 is the mean of all the disappearance losses calculated during the simulation, as explained in subsubsection 2.5.5. Each disappearance loss is calculated using different frames of the simulation, so the AE is evaluated under different "transformations".

$$Loss \leftarrow \mathbb{E}_{t \sim T} \left[\text{Loss}(f(t(x')), y) \right] \tag{13}$$

- **Adversarial patch optimization:** Using an iterative algorithm and the Adam optimizer, the adversarial patch is updated after each query to minimize the confidence score for detecting vehicles and maximizing the predicted distance to the lead vehicle.
- **Evaluate attack success:** The success of the attack is evaluated by checking how the model's perception is affected by the adversarial patch. Metrics such as model confidence in lead vehicle detection and predicted distance to lead vehicle are monitored.



(a) Confidence values indicating how sure is Openpilot about its distance detection.

(b) Distances detected concerning the vehicle ahead.

Figure 33: Evaluation charts of the attack 3 performance.

The performance of this attack can be appreciated in Figure 33. These charts shows a total of 10 epochs and 102 images, with the following metrics:

- Prediction confidence
- Predicted distance

As we can see in the charts, despite the iterative optimization of the adversarial patch, both the predicted distances and the confidence values have shown minimal or no improvement across the epochs. The lack of significant change in these metrics may indicate that the model's robustness to this specific adversarial approach is higher than initially anticipated. The attack is more complex than expected and it may be using some sensors or data transformations that have not been successfully replicated.

5.7 Attack 3: (1+1) Evolution Strategy in Black-box Optimization

The second approach against the Openpilot AD system consists of a ES in a Black-box optimization scenario. The visual diagram in Figure 34 facilitates the workflow understanding formulated in this methodology.

The bridge.py file that runs the Openpilot vehicle and optimizes the adversarial patch can be found in the GitHub repository [97] and deeper explanations in [98]

In black-box scenarios, ES algorithms are used by introducing mutations to a population, iteratively finding an AE that produces the desired output [88]. Considering the constraints in the context of this FDP explained in subsection 7.2, (1+1)-ES algorithm has been chosen.

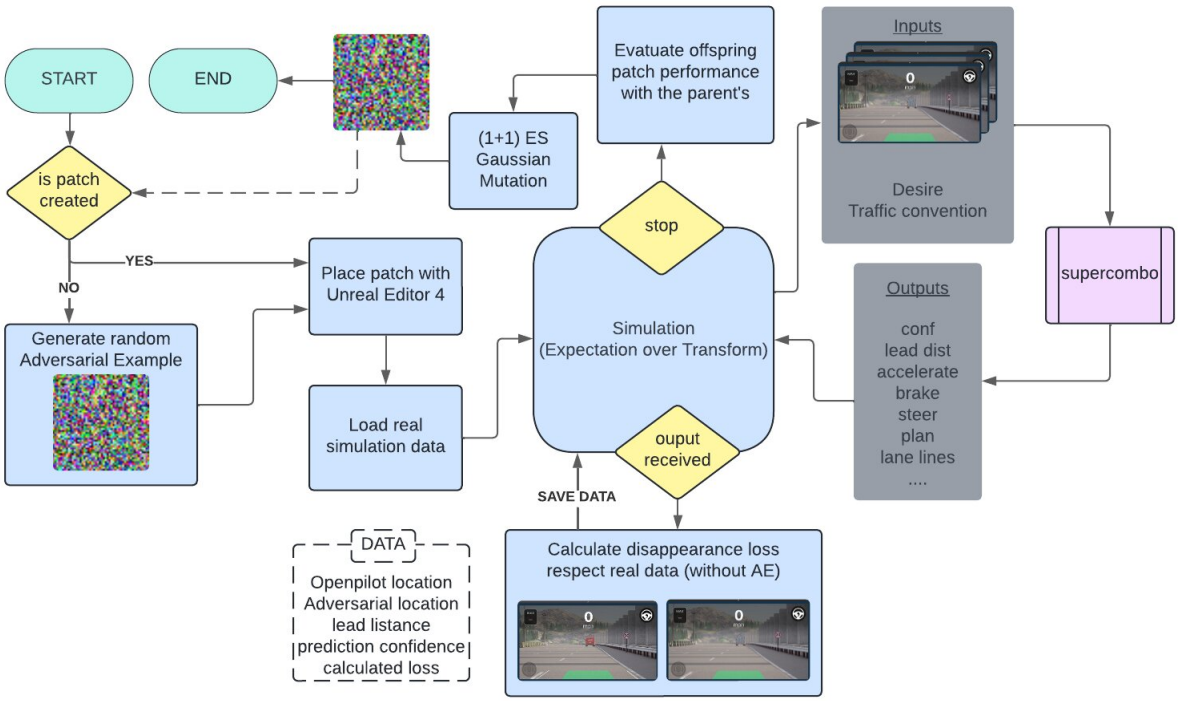


Figure 34: Visual diagram about the formulated workflow for the attack 3.

The formulated methodology is detailed as follows:

- **Blackbox scenario**: The attack assumes that there is no access to the DNN

model's parameters.

- **(1+1)-ES**: As explained in subsubsection 2.5.6, considering an iterative algorithm, starting from a random AE, in each generation a unique offspring is generated applying mutation using the code in Listing 12
- **Gaussian mutation**: To mutate the offspring, the Gaussian mutation is used, explained in 2.5.6 and using the code in Listing 13.
- **Choosing offspring**: The current offspring's performance is evaluated compared to its parent. Then, the better one is chosen and a mutation is applied to generate a new offspring
- **Disappearance loss**: The loss function in Equation 14 used to make the lead vehicle "disappear" for Openpilot's Supercombo model can be found in Listing 14, understanding explanations in subsubsection 2.5.4. This loss takes into account the following items [58]:
  - $s(z_i)$: Adversarial simulation prediction confidence.
  - $d(f_i)$: Real simulation (no AE placed in the lead vehicle) distance predictions.
  - $d(z_i)$: Adversarial simulation (AE placed in the lead vehicle) distance predictions.
  - AE smooth pattern.

$$\begin{aligned} L &= L_{conf} + \lambda_1 L_{dis} + \lambda_2 L_{TV} \\ L_{conf} &= -log(1 - s(z_i)) \\ L_{dis} &= \left| \frac{d(z_i)}{d(f_i)} \right| \\ L_{TV} &= \sum_{i,j} \left(|\delta_{i+1,j} - \delta_{i,j}| + |\delta_{i,j+1} - \delta_{i,j}| \right) \end{aligned} \tag{14}$$

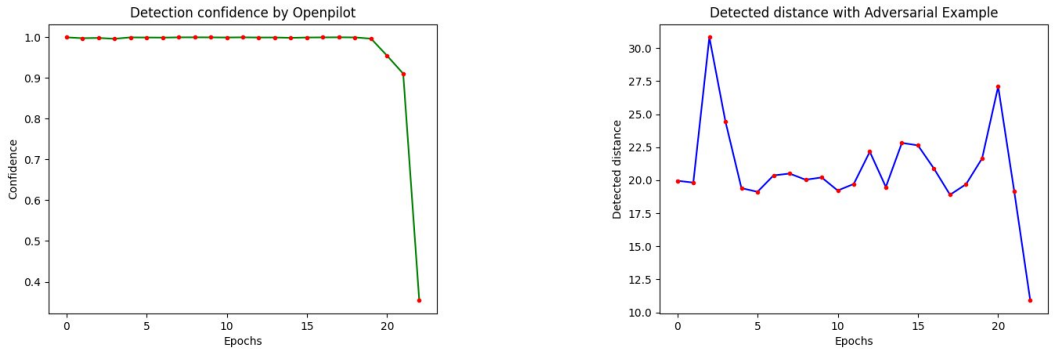
  where $\mathbf{z}_i$ represents a frame with an AE placed, $\mathbf{f}_i$ represents the real frame with no AE, $\mathbf{d}$ represents detected distance, $\mathbf{s}$ the detection score and being $\lambda_1 = 0.01$ and $\lambda_2 = 0.001$.

- **Expectation over Transform**: The considered loss function in Equation 15 is the mean of all the disappearance losses calculated during the simulation, as explained in subsubsection 2.5.5. Each disappearance loss is calculated using different frames of the simulation, so the AE is evaluated under different "transformations".

$$Loss \leftarrow \mathbb{E}_{t \sim T} \left[\text{Loss}(f(t(x')), y) \right] \tag{15}$$

- **Targeted attack**: The objective is to make the model detect that the lead vehicle is farther away than it actually is thanks to the disappearance loss approach.
- **Unreal Editor 4**: The AE is placed in the rear of the adversarial vehicle customizing the blueprint appearance as explained in subsection 5.4. It is not placed with code, unlike previous approaches.

As shown in Figure 34, when the simulation starts, the offspring and parent AEs are loaded using the code in Listing 15, and during the simulation, the offspring's performance is evaluated using the EOT technique in combination with the implemented disappearance loss. In fact, the best between the parent and the offspring is chosen and evoluted with the Gaussian mutation for the next iterations.



(a) Confidence values indicating how sure is Openpilot about its distance detection.

(b) Distances detected concerning the vehicle ahead.

Figure 35: Evaluation charts of the attack 3 performance.

This attack's performance can be shown in Figure 35. This chart shows a total of 20 iterations, with the following metrics:

- Prediction confidence
- Predicted distance

Therefore, what can be concluded from these metrics is that considering the lack of iterations (20 are too low for training an AE), and considering the constraints in subsection 7.2, the AE still not making considerable effect against Openpilot's Supercombo model. But thanks to the nature of the implemented (1+1)-ES, an improving trend can be observed, especially in the detected leads.

So, if more iterations had been performed, the AE would have been better trained and the trend would be much clearer.

5.8 Comparative Analysis

Having formulated, implemented, and assessed some state-of-the-art algorithms to craft AE, this subsection highlights the strengths and weaknesses of the formulated attacks.

5.8.1 Attack 1: White-box CW attack

The strengths of this approach are described as follows:

- **High Precision in Perturbation**: This attack method is known for its ability to generate robust perturbations. The use of the L_2-norm ensures that the perturbations are minimal yet effective, maintaining the visual similarity between the original and the adversarial examples.
- **Versatile Algorithm**: Since CW attack is widely used against image classification models, it can also be used against object detection models [96].
- **Control over the Confidence of Misclassifications**: The CW attack allows for adjustable confidence settings in generating adversarial examples, enabling the creation of samples that are misclassified with high confidence by the target model. This confidence is measured by the kappa (κ) parameter.

On the other hand, the weaknesses are the following ones:

- **Requirement for the Model Knowledge**: Being a white-box attack requires complete knowledge of the model's architecture and parameters, which may not always be available in practical scenarios.
- **Computational Intensity**: This attack involves solving an optimization problem that can be computationally expensive, especially for larger models or higher-resolution images.

5.8.2 Attack 2: White-box attack against the Supercombo model

The strengths of this approach are described as follows:

- **High Effectiveness**: By accessing the model parameters, these attacks can generate highly effective AEs. The perturbations are crafted in a way that maximizes their impact on the model's predictions through the gradient

descent algorithm, making them significantly more likely to induce misclassification or other erroneous behavior.

- **Targeted Attacks**: Thanks to the model's parameters, is much easier to find an input pattern that misleads the model classification into a specific target.
- **Iteration speed**: Accessing directly the model, the epochs are performed in a more efficient way than running them in simulations and having to compile the Carla simulator in each simulation.

On the other hand, the weaknesses are the following ones:

- **Dependency on Model Access**: This attack can only be performed if having access to the model directly. In this case, the model could be used but managing all the architecture to access it keeps out of the project's scope.
- **Data Handling**: Learning to work with unmanageable data like frames in YUV4:2:0 and raw outputs, makes it very difficult to set up the white-box scenario.
- **Model Interaction Difficulty**: Openpilot's architecture uses the cereal messaging library for asynchronous communication between modules, making it complex to manage data without connecting to essential modules.

5.8.3 Attack 3: (1+1) Evolution Strategy in Black-box Optimization

The strengths of this approach are described as follows:

- **No Model Access Required**: Unlike white-box attacks, this method does not require direct access to the model parameters, making it applicable in more restrictive environments.
- **Robust EOT**: Thanks to simulating with the AE placed in the vehicle's rear side, makes the AE generation more robust since the expected Loss is taken into account.
- **Adaptation**: Not being the optimal algorithm but the one that fits better with the constraints in subsection 7.2.

On the other hand, the weaknesses are the following ones:

- **Low Efficiency**: Planing the AE at the back of an Actor customizing the blueprint appearance and having to compile the simulator for each iteration, makes the process too slow.

- **Slow convergence**: Since the natural process of the algorithm is to select always the best mutation, more optimal ES algorithms could be considered.

6 Methodology: GitHub repository

This section aims to explain general methodologies to attack the Openpilot AD system crafting AEs using the white-box and black-box strategies.

The whole Methodology, from CARLA and Openpilot setup until Python adversarial attacks source code is located in a GitHub repository [99], as a contribution to knowledge for the company in subsection 1.5.

6.1 Introduction

Adversarial example attacks involve introducing carefully crafted perturbations into input data to fool machine learning models, especially deep learning models. These perturbations are often imperceptible to human eyes but can significantly alter the output of the model. In the context of autonomous driving systems like Openpilot, adversarial attacks could trick the system into making erroneous driving decisions by manipulating its perception model.

This methodology outlines the steps for conducting adversarial attacks, focusing on both **whitebox** and **blackbox** approaches, with an emphasis on Openpilot's object detection systems and longitudinal planning.

6.2 General Overview of Adversarial Attacks

Adversarial attacks aim to deceive machine learning models by introducing perturbations to their inputs. In the context of an autonomous driving system, these inputs could be camera images, lidar data, or radar signals. The goal is to manipulate the system into making incorrect predictions, such as not detecting existent obstacles.

There are two main categories of adversarial attacks:

1. **Whitebox Attacks**: The attacker has full knowledge of the model architecture, parameters, and gradients.
2. **Blackbox Attacks**: The attacker has no direct access to the model's internals and must rely only on input-output observations.

6.3 Whitebox Attacks

A white-box attack consists of accessing the model's parameters, architecture, and internal workings. This allows the attacker to analyze the entire neural network, including its weights, biases, and the data flow through the network layers. To

craft adversarial examples (AEs) against the Openpilot autonomous driving (AD) system, the white-box approach involves the following steps:

6.3.1 Steps for Whitebox Attacks

1. **Model Analysis**: Understanding the architecture and components of the Openpilot system, including the neural network models it uses. The key resources for this analysis are:
   - Openpilot components and architecture
   - Supercombo model
   - Inputs for Openpilot
   - Outputs of the driving model
   - Qcom cameras
   - Interaction with the Supercombo model
   - Model execution
   - Cereal messaging package
   - Visionipc system
   - Simulation connection script to gather environmental data
   - YUV image formats and formats from another source
   - Existing adversarial attack efforts on Openpilot and another project
2. **Model Format Conversion**: Since the Openpilot Supercombo model uses the ONNX format, it's necessary to convert the model into a more manageable framework, such as PyTorch, to enable gradient-based attacks. Conversion can be done using libraries like onnx2torch.
3. **Data Preparation**: Collect camera frame images from the car's sensors. These images will be saved and used later for feeding into the Supercombo model to craft adversarial examples.
4. **Data Parsing**: Convert the data formats to ensure compatibility with the neural network's input specifications.
5. **Model Queries**: Access the model's parameters, such as weights and biases, to calculate gradients. These gradients are crucial for updating the adversarial example and steering the output toward the desired misclassification.

6. **Loss Function**: Define a loss function that measures the impact of the adversarial example. The loss will be used to update the pixels of the adversarial example iteratively.
7. **Iterative Algorithm**: Use an iterative process to query the Supercombo model, evaluate the output with the loss function, and update the adversarial example using gradients. Continue until the perturbation reaches a predetermined threshold or number of iterations.
8. **Testing and Validation**: Once the adversarial example is crafted, run simulations or real-world tests to validate its effectiveness. This step is crucial to ensure that the adversarial example consistently fools the model across various scenarios.

This method requires significant knowledge of the system's internals but can be highly effective in scenarios where the attacker has access to the model's parameters. The white-box approach is one of the most precise ways to craft adversarial examples as it allows for fine-tuning of perturbations based on complete access to the model.

6.4 Blackbox Attacks

A black-box attack consists of crafting adversarial examples (AEs) without access to the model's architecture, weights, and biases. In this scenario, the attacker interacts with the system only through its inputs and outputs, treating the model as a "black box." The goal is to manipulate the input data, such as camera images, to mislead the system into making incorrect decisions. To attack the Openpilot autonomous driving (AD) system using a black-box strategy, the following steps can be followed (during simulation executions, customizing bridge.py file):

6.4.1 Steps for Blackbox Attacks

1. **Initialize Random Patch**: If no adversarial patch has been created yet, generate a random RGB pixel patch. This patch will serve as the basis for subsequent iterations and modifications.
2. **Place Patch**: The adversarial patch should be strategically placed on the rear part of the lead vehicle in the driving scenario. There are two approaches to do this:
   - **Approximate Positioning**: Use real-time simulation to approximate the patch's position in each iteration. This method is faster but may lack precision.

  - **UnrealEditor4 Positioning**: Use the UnrealEditor4 simulation environment to place the patch more accurately during each iteration, although this method is slower due to the overhead of editing.

3. **Data Preparation**: Load the simulation data from previous iterations, enabling comparison of the current simulation's output with previous runs. This comparison helps in assessing whether the adversarial patch is having the intended effect or needs further updates.
4. **Model Interaction**: Since you don't have access to the internal workings of the model, interact with Openpilot's end-to-end model via messaging packages, replicating how the real Openpilot interacts with the model. Use messaging frameworks such as VisionIPC to send inputs and retrieve outputs.
5. **Loss Function**: Evaluate the model's output by comparing it with the results from previous executions. Define a loss function that measures how well the adversarial patch is achieving its goal, such as causing lane deviations or improper braking decisions.
6. **Patch Update**: After evaluating the loss, update the adversarial patch's parameters. These updates can be applied randomly, for example using a Gaussian distribution, or any other update rule that seeks to maximize the model's misclassification.
7. **Iterative Method**: Continuously update the patch through an iterative process. After each iteration, evaluate the patch's performance using the loss function and continue updating until a predefined threshold is reached, or a set number of iterations have been completed.

This black-box approach is less direct than the white-box method but is more practical in real-world settings where attackers typically do not have access to the internal model parameters. By interacting with the system only through its inputs and outputs, the attacker can iteratively refine adversarial examples to exploit vulnerabilities in Openpilot's perception model.

6.5 Defenses Against Adversarial Attacks

Adversarial attacks pose significant challenges to the security and reliability of autonomous driving systems like Openpilot. Various defense mechanisms have been proposed to mitigate the risks posed by adversarial examples (AEs). The most effective defenses often involve a combination of techniques designed to make machine learning models more robust. Below are some commonly used defenses against adversarial attacks, along with relevant references to academic research.

1. **Adversarial Training**: Adversarial training is one of the most widely studied defense mechanisms. In this approach, the model is trained not only on clean data but also on adversarially perturbed data. By continuously exposing the model to adversarial examples during the training process, the model learns to become more robust against such attacks.
   - **Key Paper**: Goodfellow et al. (2015) - Explaining and Harnessing Adversarial Examples
   - **Summary**: The authors introduce adversarial training as a defense by training neural networks on both clean and adversarial examples, demonstrating improved robustness against perturbations.
   - **Another Reference**: Madry et al. (2017) - Towards Deep Learning Models Resistant to Adversarial Attacks
   - **Summary**: This paper presents adversarial training using Projected Gradient Descent (PGD) as a method for creating robust models, which remains one of the most effective defenses against strong attacks.
2. **Input Preprocessing**: Input preprocessing involves modifying the inputs before feeding them into the model. The goal is to remove or reduce the adversarial noise that may have been introduced. Common preprocessing methods include image transformations like resizing, JPEG compression, or applying denoising filters.
   - **Paper**: Guo et al. (2018) - Countering Adversarial Images Using Input Transformations
   - **Summary**: The authors propose using a series of transformations such as JPEG compression, total variance minimization, and bit-depth reduction to defend against adversarial attacks. These transformations can significantly reduce the effectiveness of adversarial perturbations.
   - **Paper**: Xie et al. (2018) - Mitigating Adversarial Effects Through Randomization
   - **Summary**: This work shows that applying random transformations (like random resizing and padding) to the inputs can make it harder for adversarial perturbations to generalize across different transformations, thus improving robustness.
3. **Ensemble Methods**: Ensemble-based defenses involve using multiple models to make a decision collectively. By combining the predictions of several models, it becomes harder for adversarial perturbations to fool all models

simultaneously.

- **Paper**: Tramèr et al. (2018) - Ensemble Adversarial Training: Attacks and Defenses
- **Summary**: This paper introduces ensemble adversarial training, where the training process involves adversarial examples generated from multiple models. The authors show that this approach leads to more robust models, capable of withstanding transfer-based attacks.

4. **Defensive Quantization**: This defense reduces the precision of the input data or the model's weights, thus reducing the impact of small, imperceptible perturbations caused by adversarial examples. Quantization makes the network less sensitive to small input perturbations.
   - **Paper**: Lin et al. (2019) - Defensive Quantization: When Efficiency Meets Robustness
   - **Summary**: The authors propose quantizing neural networks to improve their robustness against adversarial attacks. By using lower-precision representations for weights and activations, the model becomes more resistant to small, malicious perturbations in input data.

6.6 Conclusion

Adversarial attacks represent a significant challenge to the safety and reliability of autonomous driving systems like Openpilot. While whitebox attacks allow for precise and targeted perturbations, blackbox attacks can still be highly effective even without full access to the model. Developing robust defenses and continuously testing against adversarial attacks is crucial to ensuring the long-term safety of autonomous vehicles.

7 Conclusions and Future Work

7.1 Conclusions

The field of end-to-end AD vehicles using DNNs is rapidly advancing. These networks are fundamental for processing large amounts of data and making real-time driving decisions [18]. However, the security of these systems is also safety-critical, since a successful attack could lead to accidents [58, 53].

This FDP has achieved its intended objectives. First, the successful integration of the CARLA simulator with Openpilot was established, enabling the creation of a virtual environment for testing adversarial attacks. The CW attack was successfully implemented against image classification models, and both white-box and black-box attacks were conducted against the Openpilot system. Although none of the attacks succeeded in misleading Openpilot, the project contributed valuable knowledge to understanding its architecture, including the Supercombo model, C++ source code, module connections, data preparation, model runners, and machine learning algorithms.

A GitHub repository was developed as a comprehensive methodology for implementing adversarial attacks, serving as a guide for new users interested in exploring AML in autonomous driving systems. This knowledge primarily benefits Ikerlan S.Coop. and provides a foundation for further exploration in this field. The results of this project offer insights into the strengths and weaknesses of various adversarial attack strategies and contribute to a better understanding of Openpilot's resilience.

Finally, it is important to emphasize the critical need of implementing mitigations for AEs. The potential risks posed by AEs to the safety and reliability of AD systems cannot be overstated. Developing and integrating effective measures is crucial for ensuring the robustness of AD systems against AML attacks, thereby enhancing their overall safety and reliability [100, 101, 102].

7.2 Constraints

This section describes in detail the limitations faced during the project, both for the research analysis and for the development:

- **Openpilot architecture**: Openpilot's architecture is much more complex than expected. This makes it difficult to manage the communication between its modules in a Python implementation and retrieve correctly relevant data, taking sensors and data transformations into account.

- **Openpilot data formats**: As well as the complex architecture, the data types used by Openpilot are not easy to handle, since inputs are in YUV4:2:0 format and AEs are in RGB format, and the output has a shape (1, 6120) of floats. Retrieving relevant data from the output array requires deep knowledge and understanding of the source code so that the data follows the same flow as in real driving time.
- **Black-box approach**: Due to the complex management of Openpilot's architecture, the black-box optimization approach has been considered, but this approach has a lot of constraints. The CARLA simulator is required in real-time to collect data and communicate with Openpilot's Supercombo DNN without access to the model's parameters, consequently, gradients cannot be used, which makes the process so much slower.
- **Lack of time for iterations**: Since the black-box scenarios are not optimal, more time is needed to optimize AEs. Although the computer has 8GB of RAM, most of the time it is a limitation, since the simulator lags, resulting in erratic behaviors in the vehicle and it is necessary to be running the CARLA simulator until a good execution, approximately 30 minutes. In addition, for each iteration, the simulator has to be compiled to correctly apply the EOT technique, which takes an average of 50 minutes.

7.3 Future Work

The next steps to deepen this FDP involve the following future lines:

- **More iterations**: More time is needed to make more iterations and see the evolution and the performance of the implemented black-box scenarios.
- **Optimal ES algorithm in black-box scenarios**: When managing correctly the architecture of Openpilot, ES algorithms with a bigger population than "1" could be implemented, to make the AE learning more optimal in black-box scenarios.
- **Whitebox attack**: Fix any problem in the developed algorithm gathering more information. Also, CW algorithm is widely used against image classification tasks, but Shang-Tse Chen et al. [96] have demonstrated that an object detection model like Faster R-CNN can be fooled using this algorithm. Then, the algorithm could be abstracted to integrate it with the Supercombo model and generate more robust AEs.
- **Transfer learning**: After fooling Openpilot AD system, develop approaches that creates AEs to attack general AD systems. This consists of creating universal patches for one DNN and "transferring" the knowledge of the AE [73,

82] to another AD system's DNNs.

- **Mitigations**: Once the AE attacks are created, the next step would be implementing defenses for making the AD models more robust. This could be achieved using AEs as training samples [100, 101, 102].

References

[1] Iqbal H Sarker. "Machine learning: Algorithms, real-world applications and research directions". In: *SN computer science* 2.3 (2021), p. 160 (See pages 13, 24).

[2] Dan Hendrycks et al. "Unsolved problems in ml safety". In: *arXiv preprint arXiv:2109.13916* (2021) (See page 13).

[3] Lauren Richards. "Tesla Autopilot Crashes: With at Least a Dozen Dead, 'Who's at Fault, Man or Machine?'" In: *Impakter* (2022). URL: https://impakter.com/tesla-autopilot-crashes-with-at-least-a-dozen-dead-whos-fault-man-or-machine/ (See page 13).

[4] Tesla accident. URL: https://www.skynettoday.com/briefs/tesla-investigations (See page 13).

[5] Deepfake during videocall. URL: https://www.trendmicro.com/en_us/research/24/b/deepfake-video-calls.html (See page 13).

[6] Milad Nasr et al. "Scalable extraction of training data from (production) language models". In: *arXiv preprint arXiv:2311.17035* (2023) (See page 13).

[7] Mansi Girdhar, Junho Hong, and John Moore. "Cybersecurity of Autonomous Vehicles: A Systematic Literature Review of Adversarial Attacks and Defense Models". In: *IEEE Open Journal of Vehicular Technology* 4 (2023), pp. 417–437. DOI: 10.1109/OJVT.2023.3265363 (See page 13).

[8] Carla Simulator. URL: https://carla.org/ (See page 18).

[9] Carla Simulator 0.9.14 Documentation. URL: https://carla.readthedocs.io/en/0.9.14/start_introduction/ (See page 18).

[10] *CARLA Actors*. URL: https://carla.readthedocs.io/en/latest/core_actors/ (See page 18).

[11] *CARLA Blueprints*. URL: https://carla.readthedocs.io/en/latest/core_actors/#blueprints (See page 18).

[12] *Unreal Editor 4 in CARLA*. URL: https://carla.readthedocs.io/en/0.9.14/build_linux/#unreal-engine (See page 18).

[13] Shuncheng Tang et al. "Issue categorization and analysis of an open-source driving assistant system". In: *2021 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE. 2021, pp. 148–153 (See page 19).

[14] OpenPilot. URL: https://www.comma.ai/openpilot (See page 19).

[15] *Openpilot Limitations*. URL: https://github.com/commaai/openpilot/blob/fa310d9e2542cf497d92f007baec8fd751ffa99c/docs/LIMITATIONS.md (See page 19).

[16] OpenPilot architecture. URL: https://blog.comma.ai/openpilot-in-2021/ (See page 20).

[17] Li Chen et al. “Level 2 autonomous driving on a single device: Diving into the devils of openpilot”. In: *arXiv preprint arXiv:2206.08176* (2022) (See pages 21, 22, 44).

[18] Pranav Singh Chib and Pravendra Singh. “Recent advancements in end-to-end autonomous driving using deep learning: A survey”. In: *IEEE Transactions on Intelligent Vehicles* (2023) (See pages 21, 77).

[19] Mingxing Tan and Quoc Le. “Efficientnet: Rethinking model scaling for convolutional neural networks”. In: *International conference on machine learning*. PMLR. 2019, pp. 6105–6114 (See pages 22, 23).

[20] Rahul Dey and Fathi M Salem. “Gate-variants of gated recurrent unit (GRU) neural networks”. In: *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*. IEEE. 2017, pp. 1597–1600 (See page 22).

[21] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. “On the difficulty of training recurrent neural networks”. In: *International conference on machine learning*. Pmlr. 2013, pp. 1310–1318 (See page 22).

[22] Abdulrahman Alassadi and Tadas Ivanauskas. *Classification performance between machine learning and traditional programming in Java*. 2019 (See page 24).

[23] Phuong T Nguyen et al. “Evaluation of a machine learning classifier for metamodels”. In: *Software and Systems Modeling* 20.6 (2021), pp. 1797–1821 (See page 24).

[24] Meshal Shutaywi and Nezamoddin N Kachouie. “Silhouette analysis for performance evaluation in machine learning with applications to clustering”. In: *Entropy* 23.6 (2021), p. 759 (See page 24).

[25] Iqbal H. Sarker. “Deep learning: A comprehensive overview on techniques, taxonomy, applications and research directions”. In: *Springer Link* 2.420 (2021), p. 20 (See page 24).

[26] Trevor Hastie et al. “Overview of supervised learning”. In: *The elements of statistical learning: Data mining, inference, and prediction* (2009), pp. 9–41 (See page 25).

[27] Mohan S Acharya, Asfia Armaan, and Aneeta S Antony. “A comparison of regression models for prediction of graduate admissions”. In: *2019 international conference on computational intelligence in data science (ICCIDS)*. IEEE. 2019, pp. 1–5 (See page 25).

[28] Rui Xu and Donald Wunsch. “Survey of clustering algorithms”. In: *IEEE Transactions on neural networks* 16.3 (2005), pp. 645–678 (See page 25).

[29] Happiness Ugochi Dike et al. “Unsupervised learning based on artificial neural network: A review”. In: *2018 IEEE International Conference on Cyborg and Bionic Systems (CBS)*. IEEE. 2018, pp. 322–327 (See page 25).

[30] Marius-Constantin Popescu et al. "Multilayer perceptron and neural networks". In: *WSEAS Transactions on Circuits and Systems* 8.7 (2009), pp. 579–588 (See pages 26, 27).

[31] Bias Explanation in Neurons. URL: https://stackoverflow.com/questions/2480650/what-is-the-role-of-the-bias-in-neural-networks (See page 27).

[32] Ke-Lin Du et al. "Perceptrons". In: *Neural Networks and Statistical Learning* (2019), pp. 81–95 (See page 27).

[33] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. "Activation functions in neural networks". In: *Towards Data Sci* 6.12 (2017), pp. 310–316 (See page 27).

[34] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. 2016 (See pages 28–33).

[35] Pramila P Shinde and Seema Shah. "A review of machine learning and deep learning applications". In: *2018 Fourth international conference on computing communication control and automation (ICCUBEA)*. IEEE. 2018, pp. 1–6 (See page 28).

[36] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), pp. 436–444 (See page 28).

[37] Xuedan Du et al. "Overview of deep learning". In: *2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC)*. IEEE. 2016, pp. 159–164 (See page 28).

[38] Samuel Schmidgall et al. "Brain-inspired learning in artificial neural networks: a review". In: *APL Machine Learning* 2.2 (2024) (See page 28).

[39] *Neural Network Architecture*. URL: https://cs231n.github.io/neural-networks-1/#nn (See page 29).

[40] Neena Aloysius and M. Geetha. "A review on deep convolutional neural networks". In: *international conference on communication and signal processing (ICCSP)* (2017) (See page 29).

[41] *Feed-Fordward pass on a fully connected neural network*. URL: https://cs231n.github.io/neural-networks-1/#feedforward (See page 30).

[42] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2020 (See page 31).

[43] Randall J Erb. "Introduction to backpropagation neural network computation". In: *Pharmaceutical research* 10 (1993), pp. 165–170 (See page 31).

[44] Raul Rojas and Raúl Rojas. "The backpropagation algorithm". In: *Neural networks: a systematic introduction* (1996), pp. 149–182 (See pages 31, 32).

[45] Anqi Mao, Mehryar Mohri, and Yutao Zhong. "Cross-entropy loss functions: Theoretical analysis and applications". In: *International conference on Machine learning*. PMLR. 2023, pp. 23803–23828 (See page 31).

[46] Leslie N Smith. "Cyclical learning rates for training neural networks". In: *2017 IEEE winter conference on applications of computer vision (WACV)*. IEEE. 2017, pp. 464–472 (See page 32).

[47] *Learning Rate in DNNs*. URL: https://neptune.ai/blog/how-to-choose-a-learning-rate-scheduler (See page 32).

[48] Ning Qian. "On the momentum term in gradient descent learning algorithms". In: *Neural networks* 12.1 (1999), pp. 145–151 (See page 33).

[49] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747* (2016) (See page 33).

[50] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014) (See page 33).

[51] Grega Vrbančič and Vili Podgorelec. "Transfer learning with adaptive fine-tuning". In: *IEEE Access* 8 (2020), pp. 196197–196211 (See page 33).

[52] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), pp. 533–536 (See page 33).

[53] Tong Wu et al. "Physical adversarial attack on vehicle detector in the carla simulator". In: *arXiv preprint arXiv:2007.16118* (2020) (See pages 33, 34, 77).

[54] Zelun Kong et al. "Physgan: Generating physical-world-resilient adversarial examples for autonomous driving". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 14254–14263 (See pages 33, 34).

[55] Pengfei Jing et al. "Too good to be safe: Tricking lane detection in autonomous driving with crafted perturbations". In: *30th USENIX Security Symposium (USENIX Security 21)*. 2021, pp. 3237–3254 (See page 33).

[56] Wei Jia et al. "Fooling the eyes of autonomous vehicles: Robust physical adversarial examples against traffic sign recognition systems". In: *arXiv preprint arXiv:2201.06192* (2022) (See page 33).

[57] Amirhosein Chahe et al. "Dynamic adversarial attacks on autonomous driving systems". In: *arXiv preprint arXiv:2312.06701* (2023) (See page 33).

[58] Yanan Guo et al. "An Adversarial Attack on DNN-based Adaptive Cruise Control Systems". In: (2021) (See pages 34, 40, 41, 63, 66, 77).

[59] *DEMO Adversarial Example against AD system*. URL: https://sites.google.com/view/acc-adv/ (See page 34).

[60] Yao Deng et al. "An analysis of adversarial attacks and defenses on autonomous driving models". In: *2020 IEEE international conference on pervasive computing and communications (PerCom)*. IEEE. 2020, pp. 1–10 (See page 34).

[61] Ali Shafahi et al. "Are adversarial examples inevitable?" In: *arXiv preprint arXiv:1809.02104* (2018) (See page 34).

[62] Ishai Rosenberg et al. "Defense methods against adversarial examples for recurrent neural networks". In: *arXiv preprint arXiv:1901.09963* (2019) (See page 34).

[63] Ruochen Jiao et al. "End-to-end uncertainty-based mitigation of adversarial attacks to automated lane centering". In: *2021 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2021, pp. 266–273 (See page 34).

[64] Hui Wei et al. "Physical adversarial attack meets computer vision: A decade survey". In: *arXiv preprint arXiv:2209.15179* (2022) (See page 34).

[65] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. "Adversarial machine learning at scale". In: *arXiv preprint arXiv:1611.01236* (2016) (See pages 34, 36).

[66] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and harnessing adversarial examples". In: *arXiv preprint arXiv:1412.6572* (2014) (See pages 34–36, 60).

[67] Xiaoyong Yuan et al. "Adversarial examples: Attacks and defenses for deep learning". In: *IEEE transactions on neural networks and learning systems* 30.9 (2019), pp. 2805–2824 (See pages 34–36).

[68] Jiajun Lu, Hussein Sibai, and Evan Fabry. "Adversarial examples that fool detectors". In: *arXiv preprint arXiv:1712.02494* (2017) (See page 34).

[69] Chawin Sitawarin et al. "Rogue signs: Deceiving traffic sign recognition with malicious ads and logos". In: *arXiv preprint arXiv:1801.02780* (2018) (See page 34).

[70] Mahmood Sharif et al. "Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition". In: *Proceedings of the 2016 acm sigsac conference on computer and communications security*. 2016, pp. 1528–1540 (See page 34).

[71] XiaoZhen Han and Ran Jin. "A small sample image recognition method based on ResNet and transfer learning". In: *2020 5th International Conference on Computational Intelligence and Applications (ICCIA)*. IEEE. 2020, pp. 76–81 (See page 34).

[72] Bangjie Yin et al. "Adv-makeup: A new imperceptible and transferable attack on face recognition". In: *arXiv preprint arXiv:2105.03162* (2021) (See page 34).

[73] Tom B Brown et al. "Adversarial patch". In: *arXiv preprint arXiv:1712.09665* (2017) (See pages 34, 78).

[74] Simen Thys, Wiebe Van Ranst, and Toon Goedemé. "Fooling automated surveillance cameras: adversarial patches to attack person detection". In:

Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops. 2019, pp. 0–0 (See page 34).

[75] Kaidi Xu et al. "Adversarial t-shirt! evading person detectors in a physical world". In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part V 16*. Springer. 2020, pp. 665–681 (See page 34).

[76] Xiaopei Zhu et al. "Fooling thermal infrared pedestrian detectors in real world using small bulbs". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 35. 4. 2021, pp. 3616–3624 (See page 34).

[77] Adversarial panda. URL: https://www.tensorflow.org/tutorials/generative/adversarial_fgsm?hl=en (See page 35).

[78] Nicholas Carlini and David Wagner. "Towards evaluating the robustness of neural networks". In: *2017 ieee symposium on security and privacy (sp)*. Ieee. 2017, pp. 39–57 (See pages 35, 37–39, 60, 61).

[79] Chuan Guo et al. "Simple black-box adversarial attacks". In: *International conference on machine learning*. PMLR. 2019, pp. 2484–2493 (See page 35).

[80] Jiaxin Zhang et al. "A novel evolution strategy with directional gaussian smoothing for blackbox optimization". In: *arXiv preprint arXiv:2002.03001* (2020) (See pages 35, 42).

[81] Battista Biggio and Fabio Roli. "Wild patterns: Ten years after the rise of adversarial machine learning". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2018, pp. 2154–2156 (See page 35).

[82] Zhibo Wang et al. "Towards transferable targeted adversarial examples". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2023, pp. 20534–20543 (See pages 35, 79).

[83] Dawn Song et al. "Physical adversarial examples for object detectors". In: *12th USENIX workshop on offensive technologies (WOOT 18)*. 2018 (See page 40).

[84] Anish Athalye et al. "Synthesizing robust adversarial examples". In: *arXiv preprint arXiv:1707.07397* (2018) (See page 41).

[85] Paul Andrei Sava et al. "Assessing the impact of transformations on physical adversarial attacks". In: *Proceedings of the 15th ACM Workshop on Artificial Intelligence and Security*. 2022, pp. 79–90 (See page 41).

[86] Agoston E Eiben and James E Smith. *Introduction to evolutionary computing*. Springer, 2015 (See page 42).

[87] Tim Salimans et al. "Evolution strategies as a scalable alternative to reinforcement learning". In: *arXiv preprint arXiv:1703.03864* (2017) (See page 42).

[88] Tobias Glasmachers. "Global convergence of the (1+ 1) evolution strategy to a critical point". In: *Evolutionary computation* 28.1 (2020), pp. 27–53 (See pages 42, 65).

[89] Michael Emmerich, Ofer M Shir, and Hao Wang. *Evolution Strategies*. 2018 (See page 42).

[90] Jiaxin Zhang et al. "A novel evolution strategy with directional gaussian smoothing for blackbox optimization". In: *arXiv preprint arXiv:2002.03001* (2020) (See page 43).

[91] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778 (See page 44).

[92] Alex Krizhevsky. "Learning Multiple Layers of Features from Tiny Images". In: 2009. URL: https://api.semanticscholar.org/CorpusID:18268744 (See pages 44, 60).

[93] Harald Schafer et al. "A commute in data: The comma2k19 dataset". In: *arXiv preprint arXiv:1812.05752* (2018) (See page 44).

[94] Carlini and Wagner GitHub implementation. URL: https://github.com/un4rch/openpilot_AdvExamples/blob/main/attacks/carlini_wagner_cifar_10.ipynb (See page 60).

[95] Carlini and Wagner GitHub explanations. URL: https://github.com/un4rch/openpilot_AdvExamples/blob/main/docs/carlini_wagner_attack.md (See page 60).

[96] Shang-Tse Chen et al. "Physical Adversarial Attack on Object Detectors". In: *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2018 (See pages 60, 68, 78).

[97] Openpilot Evolution Strategy GitHub implementation. URL: https://github.com/un4rch/openpilot_AdvExamples/blob/main/attacks/bridge.py (See page 65).

[98] Openpilot Evolution Strategy GitHub explanations. URL: https://github.com/un4rch/openpilot_AdvExamples/blob/main/docs/openpilot_blackbox_attack.md (See page 65).

[99] GitHub methodology repository. URL: https://github.com/un4rch/openpilot_AdvExamples (See page 71).

[100] Cihang Xie et al. "Adversarial examples improve image recognition". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 819–828 (See pages 77, 79).

[101] Ziang Yan, Yiwen Guo, and Changshui Zhang. "Deep defense: Training dnns with improved adversarial robustness". In: *Advances in Neural Information Processing Systems* 31 (2018) (See pages 77, 79).

[102] Yinpeng Dong et al. "Adversarial distributional training for robust deep learning". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 8270–8283 (See pages 77, 79).
[103] Carla Simulator Linux Build. URL: https://carla.readthedocs.io/en/0.9.14/build_linux/ (See pages 88, 89).
[104] Ubuntu 20.04.06 ISO file. URL: https://releases.ubuntu.com/focal/ (See page 88).
[105] Rufus Booteable USB Toolkit. URL: https://rufus.ie/en/ (See page 88).
[106] CUDA 14.4.1 Toolkit. URL: https://developer.nvidia.com/cuda-12-4-1-download-archive?target_os=Linux&target_arch=x86_64&Distribution=Ubuntu&target_version=20.04&target_type=deb_local (See page 88).
[107] OpenPilot 0.9.4 GitHub repository. URL: https://github.com/commaai/openpilot/tree/fa310d9e2542cf497d92f007baec8fd751ffa99c (See page 88).
[108] OpenPilot 0.9.4 Installation Guide. URL: https://github.com/commaai/openpilot/blob/fa310d9e2542cf497d92f007baec8fd751ffa99c/tools/README.md (See page 89).
[109] Carla 0.9.14 compressed release download. URL: https://github.com/carla-simulator/carla/releases/tag/0.9.14 (See page 89).
[110] Carla 0.9.14 source code. URL: https://github.com/carla-simulator/carla/tree/580ae28c9a33f6c0307fd1737d37d2856360fda0 (See page 89).
[111] Gain access to UnrealEditor GitHub. URL: https://www.unrealengine.com/en-US/ue-on-github (See page 90).
[112] Patchelf not found error solution. URL: https://github.com/carla-simulator/carla/issues/7196#issuecomment-1976496048 (See page 91).
[113] Patchelf not found error solution. URL: https://github.com/carla-simulator/carla/issues/4906#issuecomment-1575221992 (See page 91).

8 Appendix A: Environment setup

8.1 System requirements

Hardware requirements for OpenPilot are not so demanding, instead, CARLA is a high performance driving simulator that needs a powerful computer to run fluently. The overall requirements are the following ones[103]:

- **Ubuntu 20.04.06**: Native, not a virtual machine
- **RAM**: 32GB
- **GPU**: 8GB
- **HDD/SSD**: More than 130GB
- **Ports**: Two TCP ports available (by default, 2000 and 2001)
- **Wifi**: Good internet connection

8.2 Ubuntu (20.04.06)

All our project is going to run over the operating system Ubuntu 20.04.06, since it is the most stable and compatible with all the different software we are going to use. For the installation, we need to download the ISO file[104] and mount the iso on a bootable Universal Serial Bus (USB) using a software like rufus[105].

Now that we have ubuntu installed in native operating system, we are going to install some base packages that are going to be necessary along the rest of the setup.

- CUDA 12.4.1 Toolkit: follow the official guide[106], this is going to allow CARLA simulator render all assets and run UnrealEngine to customize our simulation.
- python 3.8

8.3 OpenPilot (0.9.4)

We are going to start cloning the github repository of openpilot[107], using the follwing command:

```
rm -rf ~/.pyenv*
pip3 install pre-commit
git clone https://github.com/commaai/openpilot.git
```

This will clone the master repository, but we want the 0.9.4 version, so we have to switch to the last commit of that version:

```
cd openpilot
git reset --hard fa310d9e2542cf497d92f007baec8fd751ffa99c
```

Now we are going to update the local repository submodules, we need exactly referenced dependencies for that commit version:

```
git submodule update --init
```

Now, following the installation guide[108] in the README.md, we are going to run the following commands:

```
tools/ubuntu\_setup.sh
poetry shell
scons -u -j\$(nproc)
```

8.4 CARLA (0.9.14) with UnrealEditor (4.26.2)

There are two types of CARLA installations:

1. Compiled GitHub package[109]: Only compressed file needs to be downloaded and decompressed. This way can launch carla server directly but we cannot customize blueprints in real-time simulation.
2. Source code from GitHub[110]: More complex and heavy setup, with UnrealEditor and all the source code, we can access the blueprint files, and then compile to launch the server.

For this project purpose, we are going to install the source code and configure UnrealEditor with CARLA. We are going to follow the official carla 0.9.14 installation guide[103], but it is going to be explained step by step, due to the fact that it has been two months of research to make the environment work successfully and error resolution:

- First of all, we are going to install some general software requirements:

```
sudo apt-get update &&
sudo apt-get install wget software-properties-common &&
sudo add-apt-repository ppa:ubuntu-toolchain-r/test &&
wget -O - https://apt.llvm.org/llvm-snapshot.gpg.key|sudo apt-
    key add - &&
sudo apt-add-repository "deb http://apt.llvm.org/xenial/ llvm-
    toolchain-xenial-8 main" &&
sudo apt-get update
```

Listing 1: General software requirements

- Now, specific software requirements for Ubuntu 20.04.06:

```
sudo apt-add-repository "deb http://apt.llvm.org/focal/ llvm-
    toolchain-focal main"
sudo apt-get install build-essential clang-10 lld-10 g++-7 cmake
     ninja-build libvulkan1 python python-dev python3-dev python3
    -pip libpng-dev libtiff5-dev libjpeg-dev tzdata sed curl
    unzip autoconf libtool rsync libxml2-dev git
sudo update-alternatives --install /usr/bin/clang++ clang++ /usr
    /lib/llvm-10/bin/clang++ 180 &&
sudo update-alternatives --install /usr/bin/clang clang //usr/
    lib/llvm-10/bin/clang 180
```

Listing 2: Ubuntu 20.04.06 requirements

- Python dependencies:

```
pip3 install --upgrade pip
pip install --upgrade pip
pip install --user setuptools &&
pip3 install --user -Iv setuptools==47.3.1 &&
pip install --user distro &&
pip3 install --user distro &&
pip install --user wheel &&
pip3 install --user wheel auditwheel
```

Listing 3: Python requirements

- Now we are going to install UnrealEditor 4.26.2, but it is very **important** to have your GitHub account linked to UnrealEngine's account, and have access to it's repository. This step is very well explained in this guide[111]. Once you have access to the repository, run the following commands:

```
git clone --depth 1 -b carla https://github.com/CarlaUnreal/
    UnrealEngine.git ~/UnrealEngine_4.26
cd ~/UnrealEngine_4.26
./Setup.sh && ./GenerateProjectFiles.sh && make
cd ~/UnrealEngine_4.26/Engine/Binaries/Linux && ./UE4Editor
```

Listing 4: UnrealEditor

- Now we are going to clone the carla repository, but specifically we are going to clone the 0.9.14 version:

```
git clone https://github.com/carla-simulator/carla.git
cd carla
git reset --hard 580ae28c9a33f6c0307fd1737d37d2856360fda0
./Update.sh
```

Listing 5: Carla 0.9.14

- At this point, we have CARLA and UnrealEditor installed but, for making them connect, we need to run a aditional commad:

```
echo "export UE4_ROOT=~/UnrealEngine_4.26" >> ~/.bashrc
```

- Now, restart the terminal or open a new one, go to the root folder of CARLA where has been cloned and some special commands can be run some for managing CARLA project:

Command	Description
make help	Prints all available commands
make pythonAPI	Builds the CARLA client
make launch	Launches CARLA server in Editor window
make package	Builds CARLA and creates a packaged version

Table 11: Important CARLA make commands

8.5 Faced errors resolution

This section is dedicated to show the solution to faced problems during the installation and environment setup:

- Update.sh gives a "404 Error: Not Found" becouse CARLA's assets repository has changed along the time since 0.9.13 version [112]. So we have to switch the following line:

```
CONTENT_LINK=http://carla-assets.s3.amazonaws.com/${CONTENT_ID}.
    tar.gz
```

  with the follwing one:

```
CONTENT_LINK=https://carla-assets.s3.us-east-005.backblazeb2.com
    /${CONTENT_ID}.tar.gz
```

- "carla/Build/patchelf-install/bin/patchelf Error, Not Found", the solution can be found in the following github issue [113]

9 Appendix B: Developed Scripts

```
vehicle_bp = blueprint_library.filter('vehicle.tesla.*')[1]
vehicle_bp.set_attribute('role_name', 'hero')
spawn_points = world_map.get_spawn_points()
assert len(spawn_points) > self._args.num_selected_spawn_point, f'''
    No spawn point {self._args.num_selected_spawn_point}, try a value
    between 0 and {len(spawn_points)} for this town.'''
spawn_point = spawn_points[self._args.num_selected_spawn_point]
print(f"OPENPILOT: {spawn_point}")
global OPENPILOT_VEHICLE
OPENPILOT_VEHICLE = world.spawn_actor(vehicle_bp, spawn_point)
self._carla_objects.append(OPENPILOT_VEHICLE)
max_steer_angle = OPENPILOT_VEHICLE.get_physics_control().wheels[0].
    max_steer_angle
```

Listing 6: Spawn Openpilot vehicle

```
bp = blueprint_library.find('vehicle.carlamotors.carlacola')
for idx,spawn in enumerate(world.get_map().get_spawn_points()): print
    (f"{idx}: {spawn}")
transform = world.get_map().get_spawn_points()[0]
global ADVERSARIAL_VEHICLE
ADVERSARIAL_VEHICLE = world.spawn_actor(bp, transform)
self._carla_objects.append(ADVERSARIAL_VEHICLE)
print(f"CARLACOLA: {transform}")
```

Listing 7: Spawn adversarial vehicle

```
def spawnRandomActors(world, actor, numActors):
    bpl = world.get_blueprint_library()
    actor_blueprints = bpl.filter(actor)
    spawn_points = world.get_map().get_spawn_points()
    maxActors = min(len(spawn_points), numActors)
    print("Number of actors to spawn: " + str(maxActors))
    spawnedActors = []
    for i, spawn_point in enumerate(random.sample(spawn_points,
        maxActors)):
        try:
            actor = world.try_spawn_actor(random.choice(
                actor_blueprints), spawn_point)
            world.tick()
            if actor:
                spawnedActors.append(actor)
        except Exception as e:
            print(e)
    return spawnedActors
```

```
spawnedActors = spawnRandomActors(world, '*vehicle*', NUM_ACTORS)
```

Listing 8: Spawn other vehicles

```
def applyTrafficManagerSettings(world, client, auto_drive=True):
  traffic_manager = client.get_trafficmanager()
  traffic_manager.set_synchronous_mode(True)
  ego_vehicle = getEgoVehicle(world, client)
  for vehicle in world.get_actors().filter('*vehicle*'):
    if vehicle.id != ego_vehicle.id:
      vehicle.set_autopilot(auto_drive)

def getEgoVehicle(world, client):
    for actor in world.get_actors().filter('vehicle.*'):
        if hasattr(actor, 'attributes') and 'role_name' in actor.
            attributes:
            role_name = actor.attributes['role_name']
            if role_name == 'hero':
                print(f"Found hero vehicle (ID {actor.id})")
                return actor
    return None

if is_openpilot_engaged:
        global TRAFFIC_APPLIED
        if not TRAFFIC_APPLIED:
          vehiculos empiecen a moverse a la vez (igualdad de
              situaciones en todas las simulaciones)
          if OPENPILOT_VEHICLE.get_velocity().x > 1 or
              OPENPILOT_VEHICLE.get_velocity().y > 1 or
              OPENPILOT_VEHICLE.get_velocity().z > 1:
            TRAFFIC_APPLIED = True
            applyTrafficManagerSettings(world, client, AUTO_PILOT)
```

Listing 9: Apply Traffic Manager config

```
import numpy as np
import onnx
import onnxruntime as ort
import torch
from onnx2torch import convert
import sys

def retrieve_weights_biases(model):
    # Initialize a dictionary to hold the weights and biases
    weights_biases = {}

    # Iterate through the initializers in the model graph
    for initializer in model.graph.initializer:
        # Convert the initializer tensor to numpy array
        weight_array = onnx.numpy_helper.to_array(initializer)
```

```
            # Store the weights and biases using their names as keys
            weights_biases[initializer.name] = weight_array

    # Display the extracted weights and biases
    #for name, array in weights_biases.items():
    #    print(f"{name} shape: {array.shape}")

    return weights_biases # {name: array}

def compare_weights(wb_onnx, wb_torch, mapping):
    for onnx_name, torch_name in mapping.items():
        onnx_weights = wb_onnx[onnx_name]
        torch_weights = wb_torch[torch_name].detach().cpu().numpy()

        # Check shapes
        if onnx_weights.shape != torch_weights.shape:
            print(f"Mismatch in shape for {onnx_name} (ONNX) and {
                torch_name} (Torch)")
            continue

        # TODO comparar ambos arrays
        # Calculate the difference
        #print(onnx_weights)
        #print(torch_weights)
        difference = np.abs(onnx_weights - torch_weights).max()
        # Optionally, you can also use relative error if that's
            useful
        relative_error = np.abs((onnx_weights - torch_weights) / np.
            where(onnx_weights != 0, onnx_weights, 1)).max()
        print(f"[*] {onnx_name} vs {torch_name}")
        print(f"\tDifference: {difference}")
        print(f"\tRelative Error: {relative_error}")

if __name__ == '__main__':
    # Load model and runtime
    model_path = "./supercombo.onnx"
    model = onnx.load(model_path)
    session = ort.InferenceSession(model_path)
    torch_model = convert(model)
    torch.save(torch_model.state_dict(), 'supercombo_torch.pth')
    print(f"[*] Suppercombo torch model saved: supercombo_torch.pth")

    wb_onnx = retrieve_weights_biases(model)
    wb_torch = {name: param.data for name, param in torch_model.
        named_parameters()}
    mapping = {
        "supercombo.vision._en._conv_stem.weight": "supercombo/vision
            /_en/_conv_stem/Conv.weight",
        "supercombo.vision._en._conv_stem.bias": "supercombo/vision/
```

```
        _en/_conv_stem/Conv.bias",
    "temporal_policy.temporal_hydra.resblock.block_a.0.weight": "
        temporal_hydra/resblock/block_a/block_a/0/Gemm.weight",
    "temporal_policy.temporal_hydra.resblock.block_a.0.bias": "
        temporal_hydra/resblock/block_a/block_a/0/Gemm.bias",
    "temporal_policy.temporal_hydra.in_layer.plan.weight": "
        temporal_hydra/plan/Gemm.weight",
    "temporal_policy.temporal_hydra.in_layer.plan.bias": "
        temporal_hydra/plan/Gemm.bias",
    "supercombo.extra._en._blocks.10._depthwise_conv.weight": "
        supercombo/extra/_en/_blocks/10/_depthwise_conv/Conv.
        weight",
    "supercombo.extra._en._blocks.10._depthwise_conv.bias": "
        supercombo/extra/_en/_blocks/10/_depthwise_conv/Conv.bias"
        ,
    "temporal_policy.nav_encoder.0.block_a.0.weight": "
        nav_encoder/nav_encoder/0/block_a/block_a/0/Gemm.weight",
    "temporal_policy.nav_encoder.0.block_a.0.bias": "nav_encoder/
        nav_encoder/0/block_a/block_a/0/Gemm.bias"
}
compare_weights(wb_onnx, wb_torch, mapping)

onnx_keys = wb_onnx.keys()
torch_keys = wb_torch.keys()

for onnx,torch in zip(onnx_keys, torch_keys):
    #if not (onnx in mapping.keys() and torch in mapping.values()
        ):
    #print(f"{torch}")
    None

"""counter_onnx = 0
for ik,iv in wb_onnx.items():
    if ik not in mapping.keys():
        print(f"Onnx Layer: {ik}")
    counter_onnx += 1
#print(type(torch_model.named_parameters()))
counter_torch = 0
for name, param in torch_model.named_parameters():
    if name not in mapping.values():
        print(f"Torch Layer: {name}")
    counter_torch += 1
    #print(f"Shape: {param.shape}")
    #print(f"Values: \n{param.data}\n")"""
print()
print(f"ONNX  params: {len(wb_onnx.keys())}")
print(f"TORCH params: {len(wb_torch.keys())}")
```

Listing 10: Evaluate onnx2torch conversion

```
def adversarial_attack(lead_conf, lead_dist, prev_dist, epochs=1):
  shared_data['updating'] = True
  global DISTANCES_LIST
  global MAX_PATCH
  global FINAL_PREV_IMG
  global FINAL_LEAD_IMG
  list_distances = load_distances()
  if not os.path.exists(ADVERSARIAL_DIR + "/patches/frames"):
    os.makedirs(ADVERSARIAL_DIR + "/patches/frames")
  save_image(FINAL_LEAD_IMG, ADVERSARIAL_DIR + "/patches/frames/lead_
      "+str(MAX_PATCH+1)+"_"+str(RANDOM_DIST)+".png")
  save_image(FINAL_PREV_IMG, ADVERSARIAL_DIR + "/patches/frames/prev_
      "+str(MAX_PATCH+1)+"_"+str(RANDOM_DIST)+".png")
  try:
    tensor = torch.tensor(shared_data['patch'], dtype=torch.float32,
        requires_grad=True, device='cuda')
    print(tensor.shape)
    print(tensor)
    optimizer = optim.Adam([tensor], lr=100)
    for i in range(epochs):
      print("[*] Epoch "+str(i))
      optimizer.zero_grad()  # Resetear gradientes
      loss = disappearance_loss(tensor, lead_conf, lead_dist,
          prev_dist)  # Calculate loss
      loss.backward()  # Calculate gradientd
      print(tensor.grad)
      if torch.any(torch.isnan(tensor.grad)) or torch.any(torch.isinf
          (tensor.grad)):
        print("Detected NaN or Inf in gradients. Skipping this update
            .")
      optimizer.step()  # Update patch
    shared_data['patch'] = tensor.detach().cpu().numpy() # convierte
        el tensor a numpy.ndarray de nuevo
    thread = threading.Thread(target=save_image, args=(shared_data['
        patch'], ADVERSARIAL_DIR+'/patches/png/'+str(SAVE_PATCH)+'_'+
        str(MAX_PATCH+1)+".png"))
    thread.start()
    THREADS.append(thread)
    np.save(ADVERSARIAL_DIR+'/patches/npy/'+str(SAVE_PATCH)+'_'+str(
        MAX_PATCH+1)+'.npy', shared_data['patch'])
  except Exception as e:
    print(e)
  finally:
    shared_data['updating'] = False
  list_distances.append((prev_dist, lead_dist, lead_conf, loss.item()
      ))
  save_distances(list_distances)

def save_distances(distances):
```

```
  filepath = ADVERSARIAL_DIR + "/patches/distances.pkl"
  with open(filepath, 'wb') as file:
    pickle.dump(distances, file)

def disappearance_loss(image, conf, dist, prev_dist, l1=0.01, l2
    =0.001):
    conf_tensor = torch.tensor(conf, dtype=torch.float32).
        requires_grad_(True).cuda()
    lconf = -torch.log(1-conf_tensor)
    ldis = -torch.abs(torch.tensor(dist, dtype=torch.float32).
        requires_grad_(True).cuda() / torch.tensor(prev_dist, dtype=
        torch.float32).requires_grad_(True).cuda())
    ind = torch.arange(0, 50-1, dtype=torch.int32)
    ltv = torch.sum(torch.abs(image[ind+1, :, :] - image[ind, :, :]))
         + torch.sum(torch.abs(image[:, ind+1, :] - image[:, ind, :]))
    loss = lconf + l1 * ldis + l2 * ltv
    return loss
```

Listing 11: Disappearance attack using gradients

```
def one_plus_one_evolution_strategy_algorithm(data_list, lr=100, sgth
    =25):
  # INPUT: [(prev_dist, lead_dist, conf, loss)]
  # media de cada distances_list
  d_mean = np.mean([elem[2] for elem in data_list])
  c_mean = np.mean([elem[3] for elem in data_list])
  l_mean = np.mean([elem[4] for elem in data_list])
  d_mean_prev = np.mean([elem[2] for elem in patch_prev_info[1]])
  c_mean_prev = np.mean([elem[3] for elem in patch_prev_info[1]])
  l_mean_prev = np.mean([elem[4] for elem in patch_prev_info[1]])
  print(f"{d_mean}␣;␣{c_mean}␣;␣{l_mean}") # medidas de patch_act
  print(f"{d_mean_prev}␣;␣{c_mean_prev}␣;␣{l_mean_prev}") # medidas
      de patch_prev
  # comparar con patch_prev_info[1:]
  if ((d_mean > d_mean_prev and c_mean < c_mean_prev)) or patch_prev
      is None:
    patch_next = gaussian_mutation(patch_act, lr)
    with open(ADVERSARIAL_DIR+'/patches/npy/'+str(SAVE_PATCH)+'_'+str
        (MAX_PATCH+1)+'.pkl', 'wb') as file:
      pickle.dump([patch_act, data_list], file)
    print("[*]␣Patch␣successfully␣updated!")
  else:
    patch_next = gaussian_mutation(patch_prev, lr)
    print("[!]␣Patch␣not␣updated!")
  with open(ADVERSARIAL_DIR+'/patches/npy/patch_act.pkl', 'wb') as
      file:
    pickle.dump(patch_next, file)
```

Listing 12: (1+1) Evolution Strategy)

```
def gaussian_mutation(image, sigma=0.05, blur_sigma=1):
    """
    Apply Gaussian mutation to an image.
    """
    noise = np.random.normal(0, sigma, image.shape)
    smooth_noise = gaussian_filter(noise, sigma=blur_sigma)
    mutated_image = image + smooth_noise
    mutated_image = np.clip(mutated_image, 0, 255).astype(np.uint8)
    return mutated_image
```

Listing 13: Gaussian Mutation

```
def disappearance_loss(image, conf, dist, real_dist, l1=0.01, l2
    =0.001):
    # Confidence loss component
    lconf = -math.log(1-conf)
    # Distance loss component
    ldis = -abs(dist / real_dist)
    # Total Variation loss component
    ind = np.arange(0, 50-1)
    ltv = np.sum(abs(image[ind+1, :, :] - image[ind, :, :])) + np.sum
        (abs(image[:, ind+1, :] - image[:, ind, :]))
    # Combine all components
    loss = lconf + l1 * ldis + l2 * ltv
    return loss
```

Listing 14: Disappearance loss

```
def load_patch(filename=None):
  global MAX_PATCH
  MAX_PATCH = 0
  if filename is None:
    numbers = []
    for filename in os.listdir(ADVERSARIAL_DIR+"/patches/npy"):
      if filename.endswith('.pkl'):
        match = re.search(r'\d+', filename)
        if match:
            number = int(match.group())
            numbers.append(number)
    if len(numbers) > 0:
      MAX_PATCH = max(numbers)
    filename = ADVERSARIAL_DIR+"/patches/npy/patch_"+str(MAX_PATCH)+"
        .pkl"
  else:
    filename = ADVERSARIAL_DIR+"/patches/npy/"+filename
  if os.path.exists(filename):
    with open(filename, 'rb') as file:
      patch = pickle.load(file)
  else:
```

```
    patch = [np.random.randint(0, 256, (PATCH_SIZE[0], PATCH_SIZE[1],
        3), dtype=np.uint8), 999999, 999999, 999999]
  return patch

def initialize_dirs():
  if not os.path.exists(ADVERSARIAL_DIR+'/patches'):
    os.makedirs(ADVERSARIAL_DIR+"/patches")
  if not os.path.exists(ADVERSARIAL_DIR+'/patches/npy'):
    os.makedirs(ADVERSARIAL_DIR+"/patches/npy")
  if not os.path.exists(ADVERSARIAL_DIR+'/patches/png'):
    os.makedirs(ADVERSARIAL_DIR+"/patches/png")
  if not os.path.exists(ADVERSARIAL_DIR+'/frames/'):
    os.makedirs(ADVERSARIAL_DIR+'/frames/')

# Initialize used directories
initialize_dirs()
# Cargar datos de la simulacion sin parche
if os.path.exists(ADVERSARIAL_DIR+"/patches/npy/real_sim.pkl"):
  with open(ADVERSARIAL_DIR+"/patches/npy/real_sim.pkl", 'rb') as
      file:
    REAL_DATA = pickle.load(file)[1]
else:
  REAL_SIM = True
# Load actual simulation patch in use
if os.path.exists(ADVERSARIAL_DIR+"/patches/npy/patch_act.pkl"):
  with open(ADVERSARIAL_DIR+"/patches/npy/patch_act.pkl", 'rb') as
      file:
    patch_act = pickle.load(file)
else:
  patch_act = np.random.randint(0, 256, (PATCH_SIZE[0], PATCH_SIZE
      [1], 3), dtype=np.uint8)
# Load previous patch
if os.path.exists(ADVERSARIAL_DIR+"/patches/npy/patch_"+str(MAX_PATCH
    )+".pkl"):
  with open(ADVERSARIAL_DIR+"/patches/npy/patch_"+str(MAX_PATCH)+".
      pkl", 'rb') as file:
    patch_prev_info = pickle.load(file)
    patch_prev = patch_prev_info[0]
```

Listing 15: Prepare data for simulation

```
from torch.utils.data import random_split

# Training dataset (80% train / 20% validation)
dataset_train = CIFAR10('./data', train=True, download=True,
    transform = transforms.Compose([transforms.ToTensor()]))
random_seed = 42
torch.manual_seed(random_seed)
val_size = int(0.2*len(dataset_train)) # Validation size
```

```
train_size = int(len(dataset_train)-val_size) # Training size
train_ds, val_ds = random_split(dataset_train, [train_size, val_size
    ]) # Dataset for training phase
train_dl = torch.utils.data.DataLoader(train_ds, batch_size=128*2,
    shuffle=True, num_workers=2) # Training data loader
val_dl = torch.utils.data.DataLoader(val_ds, batch_size=128*2,
    shuffle=True, num_workers=2) # Validation data loader

# Testing dataset
test_ds = CIFAR10('./data', train=False, download=True,transform=
    transforms.Compose([transforms.ToTensor()])) # Testing dataset
test_dl = torch.utils.data.DataLoader(test_ds,batch_size=128*2,
    shuffle=True, num_workers=2) # Testing data loader
```

Listing 16: CIFAR-10 dataset preparation code

```
import torch
import torchvision.models as models

# Load a pre-trained RESNET-50 model for training
model = models.resnet50(pretrained=True)

# Freeze all layers in the model
for param in model.parameters():
    param.requires_grad = False

# Replace the final fully connected layer for CIFAR-10 number of
    classes
# CIFAR-10 has 10 classes
num_classes = 10
model.fc = nn.Linear(model.fc.in_features, num_classes)

# Define loss function
criterion = nn.CrossEntropyLoss()

# Optimize only the parameters of the final layer
optimizer = optim.Adam(model.fc.parameters(), lr=0.001)

def train_model(model, train_dl, val_dl, criterion, optimizer,
    num_epochs=10):
    train_losses = []
    val_losses = []
    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0
        for images, labels in train_dl:
            # Move data to GPU if available
            images, labels = images, labels
```

```
            # Zero the parameter gradients
            optimizer.zero_grad()

            # Forward pass
            outputs = model(images)
            loss = criterion(outputs, labels)

            # Backward pass and optimize
            loss.backward()
            optimizer.step()

            running_loss += loss.item() * images.size(0)

        epoch_loss = running_loss / len(train_dl.dataset)
        train_losses.append(epoch_loss)

        # Validation loss
        model.eval()
        with torch.no_grad():
            val_running_loss = 0.0
            for images, labels in val_dl:
                images, labels = images, labels
                outputs = model(images)
                loss = criterion(outputs, labels)
                val_running_loss += loss.item() * images.size(0)

            val_loss = val_running_loss / len(val_dl.dataset)
            val_losses.append(val_loss)

        print(f'Epoch {epoch+1}, Train Loss: {epoch_loss:.4f}, 
            Validation Loss: {val_loss:.4f}')

    print('Finished Training')
    return train_losses, val_losses

train_losses, val_losses = train_model(model, train_dl, val_dl,
    criterion, optimizer, num_epochs=20)
```

Listing 17: RESNET-50 fine-tuning with CIFAR-10 dataset

```
class Cifar10CNN(nn.Module):
    def __init__(self, pNumClasses, pNumInputSize, pNumChannels):

        super(Cifar10CNN, self).__init__()

        self.conv_1 = nn.Sequential(
            nn.Conv2d(pNumChannels, pNumInputSize, kernel_size=3,
                padding=1),
            nn.ReLU(),
```

```
            nn.Dropout(p=0.2)
        )

        self.conv_2 = nn.Sequential(
            nn.Conv2d(pNumInputSize, pNumInputSize*2, kernel_size=3,
                padding=1),
            nn.ReLU(),
            nn.Dropout(p=0.2)
        )

        self.conv_3 = nn.Sequential(
            nn.Conv2d(pNumInputSize*2, pNumInputSize*4, kernel_size
                =3, padding=1),
            nn.ReLU(),
            nn.Dropout(p=0.2)
        )

        self.conv_4 = nn.Sequential(
            nn.Conv2d(pNumInputSize*4, pNumInputSize*4, kernel_size
                =3, padding=1),
            nn.ReLU(),
            nn.Dropout(p=0.2)
        )

        self.network = nn.Sequential(
            self.conv_1, # Convolution Layer 1: output dimension
                32-3+1 = 30
            self.conv_2, # Convolution Layer 2: output dimension
                30-3+1 = 28
            nn.MaxPool2d(kernel_size=2, stride=2), # Pooling Layer 1:
                 output dimension 28/2 = 14
            self.conv_3, # Convolution Layer 3: output dimension
                14-3+1 = 12
            self.conv_4, # Convolution Layer 4: output dimension
                12-3+1 = 10
            nn.MaxPool2d(kernel_size=2, stride=2), # Pooling Layer 2:
                 10/2 = 5
            nn.Flatten(), # Flatten Layer 1
            nn.Linear(pNumInputSize*4*8*8, 512),
            nn.ReLU(),
            nn.Dropout(p=0.2),
            nn.Linear(512, pNumClasses) # Fully Connected Layer (
                CIFAR-10 has output 10 classes)
        )

    def forward(self, x):
        #return F.softmax(self.network(x),dim=1)
```

```
        return self.network(x)
```

Listing 18: Custom CNN for CIFAR-10 model

```
cifarModel = Cifar10CNN(10, 32, 3)

optimizer = torch.optim.Adam(cifarModel.parameters(), lr=0.001)
loss_function = nn.CrossEntropyLoss()

def train_model(model, train_dl, val_dl, criterion, optimizer,
    num_epochs=10):
    train_losses = []
    val_losses = []
    for epoch in range(num_epochs):
        # Training phase
        model.train()  # Set the model to training mode
        running_loss = 0.0
        for images, labels in train_dl:
            # Move data to the device the model is on, typically GPU
                if available
            if torch.cuda.is_available():
                images, labels = images.cuda(), labels.cuda()

            optimizer.zero_grad()  # Zero the parameter gradients to
                ensure clean updates

            outputs = model(images)  # Forward pass: compute
                predicted outputs
            loss = criterion(outputs, labels)  # Calculate loss based
                 on the criterion given

            loss.backward()  # Backpropagation, compute gradients
            optimizer.step()  # Apply gradients and update model
                parameters

            running_loss += loss.item() * images.size(0)  # Multiply
                loss by the batch size for total loss

        epoch_loss = running_loss / len(train_dl.dataset)  #
            Calculate average loss over the entire dataset
        train_losses.append(epoch_loss)

        # Validation phase
        model.eval()  # Set the model to evaluation mode
        val_running_loss = 0.0
        with torch.no_grad():  # Disable gradient calculation for
            efficiency
            for images, labels in val_dl:
                if torch.cuda.is_available():
```

```
                images, labels = images.cuda(), labels.cuda()

            outputs = model(images)  # Compute predictions
            loss = criterion(outputs, labels)  # Calculate loss
            val_running_loss += loss.item() * images.size(0)  #
                Aggregate loss

        val_loss = val_running_loss / len(val_dl.dataset)  #
            Average loss over validation set
        val_losses.append(val_loss)

    print(f'Epoch {epoch+1}, Train Loss: {epoch_loss:.4f}, 
        Validation Loss: {val_loss:.4f}')

print('Finished Training')
return train_losses, val_losses

train_losses, val_losses = train_model(cifarModel, train_dl, val_dl,
    loss_function, optimizer, num_epochs=10)
```

Listing 19: CNN trained with CIFAR-10 dataset

10 Appendix C: Weekly Meetings

Planteamiento de trabajo

viernes, 16 de febrero de 2024 8:40

- Introducción a los ejemplos adversarios y a las redes neuronales,, con tutoriales.
- Poner en marcha CARLA, ejecutando lo que hizo Jon
- Conocer CARLA, intentar ejecutar algún otro camino
- Subir algún otro vehículo
- Analizar las diferentes mapas que existen, ejecutar openpilot en diferentes mapas
- Analizar la implicación que tiene en crear los ejemplos adversarios en la nueva versión de OpenPilot.
- Implementar los ejemplos adversarios

29/02/2024

Friday, February 16, 2024 7:38 AM

Tareas desarrolladas esta semana:

- Introduccion a openpilot y redes neuronales CNN
- Desplegar carla y openpilot
- Interacción básica con el entorno de simulación de carla
- Comprender como funciona internamente carla con openpilot y documentarlo

Tareas a desarrollar la proxima semana:

- Seguir con la interacción básica
- Documentar el cliente carla
- Arreglar errores de simulacion

14/03/2024

jueves, 14 de marzo de 2024 9:05

Haciendo:

- Tratar de editar modelos de vehiculos con editor de unreal engine
- Documentacion sobre cliente python

Problemas:

- Versiones que usa carla de UnrealEngine y UnrealEditor no cuadran 4.26 vs 4.26.2

Por hacer:

- Recibir permiso para instalar UnrealEditor 4.26 con fin de modificar vehiculos de CARLA
- Documentar todos los pasos previos

03/22/2024

viernes, 22 de marzo de 2024 9:46

Hecho:

- Instalar UnrealEditor y que funcione con CARLA
  - Guia Build Linux de CARLA (carla y UnrealEditor a traves de github)
  - Versión compilada de carla que no traía los ficheros necesarios para abrirse en UnrealEditor
- Editar un camión en maya2024 y añadir una foto en la parte trasera del camión

Por Hacer:

- Documentar los pasos de instalación y problemas
- Meter el camión en carla a través de UnrealEditor y generar una versión compilada de carla que se conecte con el bridge de openpilot

09/04/2024

martes, 9 de abril de 2024 15:31

Hecho:

- Conseguir instalar CARLA junto con UnrealEditor4.26 (version 0.9.15)
- Poner una foto en un camión en la parte trasera y que los cambios permanezcan en la ejecución

Problemas:

- Version mismatch entre cliente 0.9.13 y servidor 0.9.15, por lo que para instalar carla con unreal en la version 0.9.13, haría falta Ubuntu 18.04 en una nueva partición. Necesario que tanto el cliente y el servidor sean 0.9.13, ya que la version de openpilot que queremos usar (0.9.2) usa dicha version de CARLA.
- ¿Por qué hace falta Ubuntu 18.04?: https://carla.readthedocs.io/en/0.9.13/build_linux/

Por Hacer:

- Reinstalar carla con openpilot (0.9.13) --> con la idea de poder cambiar la parte trasera de un camion
- Arreglar que el vehiculo openpilot conduce bugueado
- Generar el ejemplo adversario y reemplazarlo por la imagen temporal de Unreal

25/04/2024

Wednesday, April 24, 2024 8:11 AM

Hecho:

- Conseguir poner en marcha CARLA 0.9.14 (servidor) con Openpilot 0.9.2 (cliente)
- Configurar plugins para que se puedan editar las texturas de un static mesh (modelo 3d)
- Meter una imagen e intentar redimensionarla para que cuadre con el tamaño de la parte trasera

Por Hacer:

- Conseguir que cuadre el tamaño de la imagen
- Completar documentacion
- Hacer el tutorial de los ejemplos adversarios con tensorflow
- Leer el paper del ataque a openpilot

02/05/2024

jueves, 2 de mayo de 2024 10:06

Hecho:

- Conseguir meter una imagen en la parte trasera de un "Blueprint class" de un camion en UnrealEditor.
- Compilar la versión de UnrealEditor y que aparezca la imagen en la simulación junto con OpenPilot.
- Tutorial de tensorflow sobre ejemplos adversarios
- Leer paper de ejemplos adversarios contra openpilot en carla

Por hacer:

- Arreglar que openpilot tome el control del coche
- Generar la imagen adversaria mediante un script de python. ¿De que imagen? ¿Que gradiente nos interesa?

Problemas:

- w. Es posible mover el coche con las teclas WASD pero las teclas 1 y 2 no hacen que openpilot tome el control:
  - Radar Error: Please restart the car
  - Openpilot unavailable: Press set to engage
    - pip3 install torch
    - Import torch (en ./openpilot/selfdrive/modeld/runners/onnx_runner.py)

09/05/2024

jueves, 9 de mayo de 2024 10:48

Hecho:

- Conseguir que Carla 0.9.14 y OpenPilot 0.9.4 se integren y funcionen correctamente (conducción normal y openpilot frena correctamente al encontrarse con el vehículo de en frente)

Openpilot conduce correctamente durante un tiempo, no hay conducción errática, al menos durante un tiempo, suficiente para la demo.	

-

Por Hacer:

- ¿Cómo generar el ejemplo adversarial?
- Documentación
- Intentar entender bien como desarrollan el ataque adversario contra Openpilot

16/05/2024

jueves, 16 de mayo de 2024 11:39

Hecho:

- Preparar la presentación
- Investigar sobre distintos ataques adversarios blackbox
- Meter un parche random y actualizarlo en funcion de la velocidad

Por Hacer:

- Recortar del video original el parche y probar en carla
- Empezar con la memoria

23/05/2024

jueves, 23 de mayo de 2024 10:30

Hecho:

- Como funciona openpilot (supercombo model)

Por Hacer:

- Entender bien EfficientNet-B2 si es clasificación de imágenes o detección de objetos
- Entender la arquitectura del modelo EN-B2
- Intentar buscar algún ejemplo adversario contra la arquitectura EfficientNet-B2
- ¿¿como tratar con YUV420??

28/05/2024

martes, 28 de mayo de 2024 14:13

Hecho:

- Avanzar en la documentación del TFG (sobretodo background)
- Convertir modelo onnx a pytorch
- Comparar sus capas y que se ha convertido correctamente
- Diagrama de flujo de la arquitectura de openpilot con carla y ataque adversarial

Por Hacer:

- Hacer las predicciones con ambos modelos (onnx y torch convertido) con el fin de asegurar que los outputs tengan los mismos valores
- Que el formato de los inputs y outputs este bien como para hacer llamadas al modelo y recibir respuestas (comprobar que el output es el mismo que el original)
- Convertir el modelo supercombo.onnx a formato pytorch
- Invesigar "Carlini & Wagner"
- Los inputs del dataset local estan en YUV, no se puede guardar RGB ya que en algun momento hay que hacer la conversion ¿cómo gestionar el formato yuv?

06/06/2024

jueves, 6 de junio de 2024 11:33

Hecho:

- Investigar Carlini & Wagner y otras alternativas
- Elegir Disappearance attack + Expectation over Transform
- Entender la formula para aplicar disappearance loss
- Empezar a implementar ejemplo adversario

Por Hacer:

- Implementar algoritmo para el adversarial example
- Investigar como conseguir los parámetros de entrada para el loss: confianza, distancias e imágenes
- Desarrollar el ejemplo adversarial en simulación como blackbox. Hacerlo con el modelo onnx en local no es viable, ya que los outputs están codificados y hace falta usar servicios de openpilot para decodificarlos y los inputs en YUV420, para nada compatible para modificar píxeles RGB.

13/06/2024

jueves, 13 de junio de 2024 10:04

Hecho:

- Implementar un prototipo del algoritmo para generar el ejemplo adversarial
- Integrar la simulación, la recolección de frames, detección de velocidad (acelerar/frenar)... para poder ejecutar el ataque adversarial

Por Hacer:

- Diferencia entre thread y process en python. Usar process para hilos en segundo plano (paralelismo)

19/06/2024

miércoles, 19 de junio de 2024 15:39

Hecho:

- Terminar metodología para terminar de implemetar el ejemplo adversarial con parche

Por Hacer:

- Evaluar el primer método (colocar el parche en la parte trasera del camión en la simulación)
- Si no funciona el primer método, investigar un segundo método, implementarlo y comparar resultados para análisis horizontal (por ejemplo otra loss function)
- Documentación del TFG

28/06/2024

viernes, 28 de junio de 2024 10:10

Hecho:

- Terminar de documentar background en el TFG
- Debugguear el disappearance attack implementado
- Guardar datos de la simulación con npy y pickle y crear scripts para generar graficas para visualizar confianza, diferencia de distancias y loss a lo largo de las iteraciones

Por Hacer:

- Documentar planificación y desarrollo
- Intentar arreglar el disappearance attack: lo que ocurre es que la imagen con el parche que se le pasa a openpilot no tiene en cuenta gradientes a la hora de recibir la respuesta de openpilot (confianza, distancia, ….) por lo que, a la hora de hacer el loss, el parche no se actualiza respecto a la distancia, sino solamente busca hacer un patrón homogeneo. Probar a cambiar el requires_grad=True en el tensor a la hora de crearlos.
- Desarrollo horizontal:

04/07/2024

lunes, 15 de julio de 2024 16:07

Hecho:

- Documentación de la memoria (Planificacion, reescribir background)
- Terminar las implementaciones
- Hacer iteraciones para generar gráficas

Por Hacer:

- Documentar desarrollo, conclusiones, terminar planificacion y abstract
- Realizar más iteraciones para generar graficas de evaluación

11/07/2024

lunes, 15 de julio de 2024 16:08

Hecho:

- Documentación: Terminar planificación, desarrollo
- Terminar de realizar iteraciones

Por Hacer:

- Documentar Abstract y conclusiones
- Correcciones de documentación

During the months of July, August and September, the project has continued to be developed without weekly meetings, since this work has been carried out outside the internship.