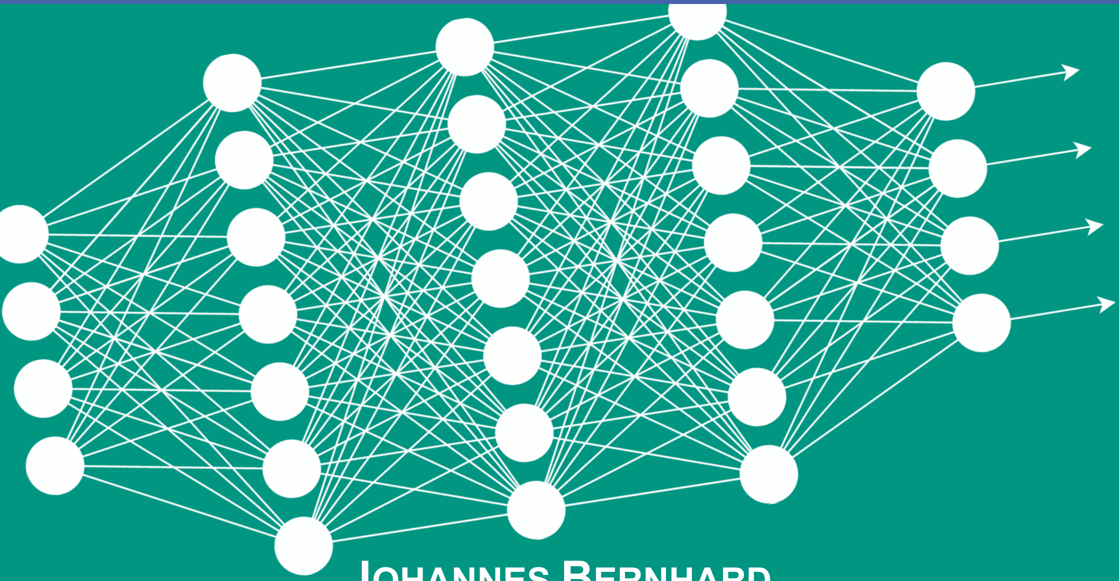
Adaptive scenario selection for simulative testing of perception functions in automated driving

Karlsruher Institut für Technologie

Doctoral Thesis



Johannes Bernhard

Adaptive scenario selection for simulative testing of perception functions in automated driving

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

von der KIT-Fakultät für
Elektrotechnik und Informationstechnik
des Karlsruher Instituts für Technologie

eingereichte

DISSERTATION

von

M. Sc. Johannes Bernhard
geb. in Friedrichshafen

Tag der mündlichen Prüfung:	12.06.2024
Hauptreferent:	Prof. Dr.-Ing. Eric Sax
Korreferent:	Prof. Dr.-Ing. Hans-Christian Reuss

Danksagung

Diese Dissertation entstand im Rahmen einer Industriepromotion mit dem Karlsruhe Institut für Technologie und der ZF Friedrichshafen AG. Die letzten fünf Jahre waren spannend und zugleich herausfordernd, jedoch voller wertvoller Erfahrungen und Erfolge. Ohne die Unterstützung meines privaten und beruflichen Umfelds wäre diese Arbeit nicht möglich gewesen.

Zu Beginn möchte ich mich bei meinem Doktorvater, Prof. Dr.-Ing. Eric Sax, für die erstklassige Betreuung bedanken. Seine wertvolle Unterstützung, sowohl bei meinen Forschungsarbeiten als auch bei persönlichen Anliegen, hat maßgeblich zum Erfolg meiner Dissertation beigetragen.

Mein herzlicher Dank gilt auch Prof. Dr.-Ing. Hans-Christian Reuss für die Übernahme des Korreferats. Seine inhaltlichen Rückmeldungen und wertvollen Anmerkungen haben maßgeblich zur Qualität dieser Dissertation beigetragen und mich hervorragend auf die Prüfung vorbereitet. Ebenso möchte ich mich bei Prof. Dr.-Ing. Thomas Zwick für den Vorsitz der Prüfungskommission sowie bei Prof. Dr. Ivan Peric und Prof. Dr.-Ing. Sander Wahls für ihre Unterstützung als Beisitzer bedanken.

Darüber hinaus möchte ich allen Kolleginnen und Kollegen bei der ZF Friedrichshafen AG meinen herzlichen Dank aussprechen. Besonders danke ich meinen Betreuern Martin Sedlacek und Thomas Schulik, die mich während der Promotionszeit mit wertvollen Ratschlägen begleitet haben. Ebenso danke ich Lucas Fonseca, Lars Schories und Joachim Naas für die angenehme Zusammenarbeit und ihre Unterstützung im Laufe der Arbeit.

Ich möchte mich auch bei Mark Schutera und Jonas Schmid für die interessanten Projekte bedanken, an denen wir gemeinsam gearbeitet haben.

Ein großer Dank geht an meine Doktorandengruppe für die gemeinsamen Seminare und Klausuren. Die interessanten Diskussionen haben meine Arbeit bereichert. Zudem war der Austausch stets eine wertvolle Unterstützung während meiner Promotionszeit.

Mein größter Dank gilt meiner Frau Rebecca, die mich bei jedem Schritt dieser Dissertation unterstützt hat. Ohne ihre unermüdliche Hilfe und ihr Verständnis wäre diese Arbeit nicht möglich gewesen. Ebenso danke ich meinen Eltern und Geschwistern, die mir stets den Rücken gestärkt und mich in jeder Phase ermutigt haben.

Zusammenfassung

Die Integration von Low-Level-Fahrsystemen zur Fahrzeugsteuerung, wie z. B. Notbremsassistenten oder Adaptive Cruise Control, haben das Potenzial, die Sicherheit im Straßenverkehr signifikant zu verbessern. Allerding wird die Einführung von vollautomatisierten Fahrsystemen aufgrund derzeit unbeantwortbarer Fragen im Bereich der automatisierten Umfeldperzeption gehemmt. Innerhalb der letzten Dekade konnten jedoch Fortschritte in den Forschungsbereichen Machine Learning, High-Performance Computing, Robotik und Automobiltechnik erzielt werden. Durch die Nutzung künstlicher neuronaler Netze konnten Ergebnisse generiert werden, die mit der Leistung des menschlichen Auges vergleichbar sind.

Obwohl die nicht-lineare Entscheidungsfindung ein zentraler Vorteil von KNNs während des automatisierten Erlernens von Datenstrukturen ist, birgt sie auch Schwierigkeiten. Diese betreffen sowohl die Interpretation des Outputs als auch die Behebung eines eventuellen Fehlverhaltens. Dabei spielen die Qualität und Diversität eine zentrale Rolle innerhalb des Lernprozesses eines Neuronalen Netzes. Daher ist die Zusammenstellung eines umfassenden Datensatzes eine zentrale Aufgabe, insbesondere für das Testen eines Models. Die Aussagekraft von Testprozessen sowie berechneter Key Performance Indicators hängen stark vom Umfang und der Diversität des zusammengestellten Testdatensatzes ab. Das Sammeln von Entwicklungsdaten im realen Verkehr hat jedoch mehrere Nachteile. Zu diesen zählen die hohen Kosten für die Durchführung der Datenaufzeichnung, die gegebenenfalls mangelhafte Qualität von Daten Labels und eine eingeschränkte Skalierbarkeit. Des Weiteren können die aufgezeichneten Daten bei zunehmender Datenmenge repetitiv werden und gefährliche Verkehrsszenarien oder extreme Wetterereignisse unterrepräsentieren.

Diese Dissertation befasst sich mit der Nutzung von synthetischen Bilddaten, mit dem Ziel die Schwächen eines KNNs systematisch aufzudecken. Hierfür wird eine Pipeline zur Testung von KNNs eingeführt. Die Pipeline nutzt die Aspekte des Szenario-basierten Testens und kombiniert diese mit Simulationssoftware zu Generierung von Bilddaten. Durch den Einsatz von adaptiven Samplingstrategien werden iterativ Testfälle generiert, die bei einem zu testenden KNN ein Systemversagen auslösen. Dabei können Bestandteile von Verkehrsszenarien identifiziert werden, die für das KNN besonders herausfordernd sind.

Im Rahmen dieser Arbeit wurde eine umfangreiche Implementierung der vorgestellten Pipeline durchgeführt und die Validität des Ansatzes nachgewiesen. Mit Hilfe der Pipeline können kritische Szenarien für das KNN effizient generiert werden. Durch die Verwendung der Bilddaten kritischer Szenarien im Trainingsprozess kann die Leistung des Modells signifikant verbessert werden. Dadurch wurde die Robustheit des Modells gegenüber kritischen Szenarien erhöht.

Diese Dissertation trägt dazu bei, den Test- und Entwicklungsprozess von neuronalen Netzen in der Umfeldperzeption von automatisierten Fahrsystemen zu verbessern. Dennoch können im Bereich des simulativen Testens aufgrund noch fehlender Forschung einige Fragestellungen noch nicht abschließend beantwortet werden.

Abstract

Integrating low-level driving systems for vehicle control, such as emergency brake assist or cruise control, has shown the potential to significant safety improvements in traffic. The prospect of fully automated driving systems has been questioned, among other things, due to limits in automated environmental perception and scene understanding. Recently, advances have been made in machine learning, high-performance computing, robotics, and automotive engineering research areas. The accuracy of machine learning systems has reached performances comparable to human capabilities. While machine learning-driven perception can significantly outperform traditional perception approaches, it is no silver bullet for solving all challenges in computer vision simultaneously.

A key feature of neural networks, wildly dominating this area of machine learning systems, is their non-linearity for decision-making, being boon and bane alike. While being able to encode information from development data into the network structure autonomously, this restricts the ability to debug and interpret behavior. The quality and diversity of development data plays a main role when training neural networks. Hence, assembling a comprehensive data set is a key task, especially for model testing. Generally, the validity of traditional tests and performance metrics in perception depends on the volume and variety of the assembled test data set. However, collecting image data from real traffic has many drawbacks, including high labeling and data recording costs, deficiencies in labeling quality, and poor scalability. Furthermore, recorded data may get repetitive over time, with dangerous traffic scenarios or extreme weather events being rare.

This thesis discusses the simulative generation of synthetic image data to reveal the weaknesses of a (perception) system-under-test. This thesis introduces a testing pipeline for perception function by utilizing and expanding on existing test frameworks. The proposed pipeline leverages scenario-based testing and synthetic data generation combined with adaptive testing strategies to iteratively test the perception system for scenarios where the performance does not comply with the required behavior. In this context, adaptive strategies assume that a perception function performance depends highly on the scenario's semantic features, such as early failure detection.

Extensive testing of the proposed pipeline demonstrates the validity of the approach and sets it in the context of alternative approaches for testing neural networks. Using the pipeline, neural network failures can be detected efficiently, and neural network deficiencies can be identified. When using the generated insights and failure data, the performance of the trained model can be improved to be more robust and less prone to challenging driving scenarios.

While there are remaining questions that have to be the subject of further research, this thesis contributes to improving the test and development process of neural networks in the perception of automated driving systems.

Table of Contents

Glossary and Math Symbols . . . . **xvii**

1 Introduction **1**

1.1 Automated driving and its impact on public traffic 1

1.2 Deep learning as enabler of automated driving systems . . . 6

1.3 Scientific contribution 12

2 Fundamental techniques of environmental perception in automated driving **17**

2.1 Traditional programming vs. machine learning 17

2.2 Types of machine learning 18

2.3 Requirements for machine learning algorithms 22

2.4 Deep learning - origin and foundations 30

2.5 Object detection frameworks 35

3 Design and testing of deep neural network-based perception functions **43**

3.1 Basic terminologies 43

3.2 Development process for object detection 45

3.3 Safety concerns regarding the use of DNN in automated driving tasks 48

3.4 Techniques for testing of perception functions 50

3.4.1 Benchmark testing 51

3.4.2 Coverage-guided testing 52

3.4.3 Adversarial testing 55

4 Adaptive test case sampling for DNN-based perception functions **59**

4.1 Simulative testing - Advantages and weaknesses 59
4.2 Introducing pipeline for adaptive scenario selection for simulative testing 64
4.3 Defining quality metrics for an adaptive test case sampling . 75
4.4 Presentation of sampling strategies 83
4.4.1 Covering arrays and t-wise testing 83
4.4.2 Evolutionary sampling 84
4.4.3 Coupling the use of meta-model with fixed-sized-candidate-set method 86

5 Implementing proposed pipeline for fault detection . . . 97
5.1 Overview 97
5.2 Designing a scenario space for adaptive sampling 99
5.3 Autonomous emergency braking controller 107
5.4 System-under-Test: YOLOv5 107
5.5 Key performance indicator: adjusted Time-to-Collision . . . 111
5.6 Visualization of experimental process 112

6 Results of pipeline for fault detection 115
6.1 Process set-up and Scenario variable selection 115
6.2 Random sampling and pair-wise testing 119
6.3 Evaluation of adaptive testing 123

7 Active Learning using adaptive Testing 133
7.1 Process overview for active learning 133
7.2 Data selection and preparation 137
7.3 Active learning testing results 138

8 Conclusion and outlook 143
8.1 Discussion 143
8.2 Future Work 147

Publications Johannes Bernhard 151

Bibliography 153

Appendices **173**
A Convolutional Neural Network Architecture 173
B Gaussian Process Regression 179

List of Figures

1.1 Different applications of deep learning technologies for perception in automated vehicles. 7
1.2 Data distribution of the BDD100k data set. 9
1.3 Criteria "safe" and "known" for scenarios in SOTIF context. . . 10
1.4 Visualization of virtualization capabilities of simulative frameworks. 12

2.1 Differences between the development processes of traditional software and machine learning models. 18
2.2 The basic concept of reinforcement learning. 21
2.3 Classic confusion matrix for binary classifiers. 23
2.4 Example for adversarial attacks on a classifier for the MNIST. . . 25
2.5 Concept of heatmap approaches for visualization of computer vision algorithms. 27
2.6 Importance of interpretability in machine learning. 28
2.7 a) Binary *Linear Threshold Unit*. b) Single perceptron with three inputs and one binary output. 31
2.8 a) Application of a single perceptron on the OR problem. b) Application of two perceptrons on the XOR problem. 32
2.9 Network with one hidden layer to solve the XOR problem. . . . 33
2.10 Comparison between step function and sigmoid function. 33
2.11 Sample digits of the MNIST database of handwritten digits. . . . 34
2.12 Schematic representation of the YOLO framework. 36
2.13 Schematic representation of the SSD framework. 37
2.14 Multi-stage framework for R-CNN. 38
2.15 Schematic representation of the intersection over union (IoU). . 39
2.16 Receiver operating characteristic (ROC) curve for a given class. . 40

3.1 Best practice for collecting development data for perception functions. 46

3.2 Visualization of the importance of domain coverage. 48
3.3 Example of full-image attack on GoogLeNet on ImageNet data. . 57
3.4 Example of the patch-level DPATCH attack on YOLO object detector. 57

4.1 Relation between increasing driven distance and unique scenarios recorded. 60
4.2 Overview on synthetic image data and synthetic labels. 61
4.3 Simulative process for synthetic image generation. 63
4.4 Schematic representation of functional testing. 64
4.5 The six layers used to describe a traffic scenario as defined by the PEGASUS project. 65
4.6 Overview of scenario-based testing process [5]. 67
4.7 Scenario description scheme from PEGASUS project [6]. 68
4.8 Schematic representation of the simulative testing process. . . . 74
4.9 Visualization of the sequences τ_i and Θ_i as defined in Def. 4.15. 77
4.10 Visualization of failure region. 80
4.11 Visualization of ω-measure. 82
4.12 Example of pair-wise testing with three scenario variables and two characteristics per variable. 84
4.13 Schematic representation of evolutionary testing process. 86
4.14 Schematic representation of meta-model based testing process. . 87
4.15 Schematic representation of the relationship between performance evaluation process and the meta-model. 88
4.16 Relationship between true underlying function and fitted model for linear regression tasks. 92
4.17 Standard feed-forward neural network that can be used for regression tasks. 93
4.18 Posterior distribution of Gaussian process. 96

5.1 Experimental pipeline to evaluate the proposed testing strategies. 98
5.2 Defined functional scenarios, derived from the pedestrian-vehicle interactions from the Euro-NCAP scenarios. 101
5.3 Functional scenarios as implemented in the experiments. 103
5.4 Different weather and daytime effects. 106

5.5 Simulation framework for SuT evaluation. 108
5.6 Visualization of YOLO output. 110
5.7 YOLOv5 custom training process. 111
5.8 Visualization of execution run of concrete scenario. 114

6.1 Test results using random testing for $\mathcal{F}_1 - \mathcal{F}_4$. 120
6.2 Test results using random testing for $\mathcal{F}_5 - \mathcal{F}_8$. 121
6.3 Parallel coordinates plot for Random Testing and functional scenario $\mathcal{F}_2$. 122
6.4 Visualization of two failures from random testing on $\mathcal{F}_2$. 123
6.5 a) Parallel coordinates plot for evolutionary testing on functional scenario $\mathcal{F}_6$. b) Parallel coordinates plot for the failures detected by meta-model based testing using GPR on functional scenario $\mathcal{F}_6$. 125
6.6 Visualization of extractions from two test executions during testing on $\mathcal{F}_6$. 126
6.7 Visualization of non-failure test execution during evolutionary testing. 128
6.8 Detected Failure from evolutionary testing on $\mathcal{F}_5$. 131

7.1 Process of re-training and testing the new YOLOv5 model weights. 134
7.2 Visualization of additional functional scenarios. 135
7.3 Extract from the process of ensuring data quality during re-training. 137
7.4 Histogram for functional scenarios $\mathcal{F}_1^*$ and $\mathcal{F}_2^*$ containing the test results for each model. 139
7.5 Histogram for functional scenarios $\mathcal{F}_3^*$ and $\mathcal{F}_4^*$ containing the test results for each model. 140
7.6 Visualization of testing improved YOLOv5 versions. 141

8.1 Placing the proposed testing process in context with SOTIF. . . 145
8.2 Validation of geometric precision of camera simulation using a checkerboard pattern [7]. 147
8.3 Probability of failure calculation for scenario $\mathcal{S}$. 148
8.4 Scenario clustering to generate functional scenarios from real-world trajectory data [1]. 149
.1 Standard building blocks for convolutional neural networks. . . . 175
.2 AlexNet-architecture. 176

.3 Standard activation functions for convolutional neural networks. 177
.4 a) Covariance matrix using the radial basis function. b) Pair-wise covariance $k(x, x')$. 180
.5 Five different function realizations sampled from a Gaussian process. 181
.6 Posterior distribution of Gaussian process. 182
.7 Posterior distribution of Gaussian process. 184

List of Tables

5.1 Defined functional scenarios, derived from the pedestrian-vehicle interactions from the Euro-NCAP scenarios for AEB systems [8]. 102

6.1 Implemented testing strategies. 116
6.2 Discretization of continuous variables. Including the categorical variables Animation and Pedestrian Blueprint, there are 8102 2-wise combinations. 118
6.3 Resulting number of detected failures $\Theta_{1,000}$ for 1,000 test executions using random sampling and pair-wise sampling. 119
6.4 Resulting number of detected failures $\Theta_{1,000}$ for 1,000 test executions using different sampling strategies. 124
6.5 *C-Measure* for 1,000 test executions using different sampling strategies. 129
6.6 Combinatorial coverage from different test strategies for total tested concrete scenarios and the concrete scenarios that result in failures. 132

7.1 Defined functional scenarios, derived from the pedestrian-vehicle interactions from the Euro-NCAP scenarios for AEB systems [8]. 136
7.2 Resulting number of failures during pair-wise testing of new model weights for all additional functional test scenarios. 138

Glossary and Math Symbols

Glossary

ACC	Adaptive cruise control
AEB	Automated emergency brake system
AI	Artificial intelligence
ADAS	Automated driver assistance systems
CV	Computer vision
DL	Deep learning
FN	False negative
FP	False postive
GPR	Gaussian process regression
LR	Linear regression
mAP	Mean average precision
ML	Machine learning
NN/DNN	Neural network/ Deep-neural network
NNR	Neural network regression
ODD	Operational domain design
SOTIF	Safety of the intended functionality
SuT	System under Test
TN	True negative
TP	True positive

TTC	Time to collision

Math Symbols

$\mathfrak{P}(\cdot)$	Power set operator
$\mathcal{F}$	Functional scenario (Def. 4.1)
S	Scenario variable (Def. 4.2)
$\mathcal{S}$	Logical scenario (Def. 4.2)
$\mathfrak{S}$	Space of logical scenarios (Def. 4.3)
$\mathfrak{F}$	Space of functional scenarios (Def. 4.3)
$I(\cdot)$	Interpreter (Def. 4.3)
s	Concrete scenario (Def. 4.4)
$\mathcal{B}$	Output space of perception function (Def. 4.5)
$\mathcal{I}$	Image space (Def. 4.5)
$T(\cdot)$	Ground truth function (Def. 4.5)
$D(\cdot)$	Perception system (Def. 4.6)
$G(\cdot)$	Simulation process (Def. 4.7)
$K(\cdot, \cdot)$	Key-performance-indicator (Def. 4.8)
$\psi(\cdot)$	Simulative testing process (Def. 4.9)
$\mathcal{T}$	Space of observed concrete scenarios (Def. 4.13)
$\tilde{\mathcal{S}}$	Observed test set (Def. 4.13)
$\phi(\cdot)$	Sampling strategy (Def. 4.14)
$(\tau_i)_{i\in\mathbb{N}}$	Failure sequence (Def. 4.15)
$(x_i)_{i\in\mathcal{N}}$	Sequence defined as sampling order (Def. 4.15)
$\Theta(\cdot)$	Accumulated failure sequence (Def. 4.15)

1 Introduction

1.1 Automated driving and its impact on public traffic

The primary purpose of partly and fully automated transport systems is to improve safety for all road users. Another purpose is to increase mobility opportunities and to make further benefits possible. Technological development obeys the principle of personal autonomy, which means that individuals enjoy freedom of action for which they themselves are responsible.

(Ethical rules for automated and connected vehicular traffic)

This quote was used by the *Ethics Commission Automated and Connected Driving - German Federal Ministry of Transport and Digital Infrastructure* as their first ethical rule regarding the use of automated driving technology in public traffic [9]. The release and development of automated cars are directly tied to their impact on the safety of all involved road users [10]. Although the number of traffic fatalities in Germany has decreased significantly from 11,300 in 1991 to 2,788 in 2022, accidents on public roads remain the second non-natural cause of death [11, 12]. Due to the vulnerability of road users, the unpredictability of environmental influences, and the behavior of people and technology, automated vehicles have little to no margin of error.
While traffic incident statistics show the potential risks of malfunctions in the vehicle's technology, they also reveal the potential to increase public traffic safety, often measured by casualties per driven kilometers [13]. According to the German Federal Statistical Office (*Statistisches Bundesamt*), driver misconduct is responsible for 88% of all 342,852 accidents with personal injuries on German Roads in 2018. The remaining accidents can be attributed to technical failure of vehicles (0.8%), misconduct of pedestrians (3%), and environmental influences (7.2%).

Studies on accident statistics highlight the ability of advanced driver assistance systems (ADAS) to increase road safety, proving the decrease of collisions for vehicles equipped with ADAS components such as emergency brake assistance (EBA) or emergency lane keeping (ELK) [14, 15]. For example, the Insurance Institute for Highway Safety compared police-reported crashes and insurance claims for vehicles equipped with and without different ADAS technologies, such as emergency brake assistance with an automatic brake mechanism, showing a reduction of front-to-rear crashes with injuries by up to 56%. This positive impact on traffic safety was not lost on lawmakers, with the European Union mandating every new vehicle from 2022 and later to be equipped with the following ADAS systems with regulation 2019/2144 [16]: Intelligent speed assistance, alcohol interlock installation facilitation, driver drowsiness and attention warning, advanced driver distraction warning, emergency stop signal, reversing detection, event data recorder.

While industry, legislation, and research emphasize the possible improvements for road safety, the general public rather mistrusts the vision of fully automated vehicles [17]. Surveys from the American Automobile Association for 2023 state that 68% of their 1,140 interviewed U.S. adults see safety issues as a significant problem associated with automated driving and have a negative attitude toward self-driving vehicles [18]. Notably, the negative attitude has increased over the past three years.

Apart from the safety dimension, automated vehicles bear significant economic opportunities with a focus on commercial traffic. A 2023 report for long-haul trucking in the US has estimated the average labor cost per driven mile to be \$0.72, resulting in 31% of the overall \$2.31/driven mile [19]. Apart from profitability-driven reduction of costs, a general shortage of workforce and drivers is plaguing the trucking industry that will only increase over time [20]. In Europe a mere 5% of truck drivers are aged bellow 25 years with 12% overall worldwide, leading to a doubling of truck driver shortage within the next five years [21]. The introduction of automated driving systems might be able to mitigate these developments and keep vital supply chains operable.

Automated vs. autonomous driving

Regarding self-driving cars, the two terminologies *autonomous* and *automated* are often used synonymously. Despite the high dependence, the difference between these two terms is significant when introducing the five level scheme of automated driving in Sec. 1.1. The Cambridge Dictionary states the definitions as follows:

Automated
Carried out by machines or computers without needing human control.

Autonomous
Independent and having the power to make its own decisions.

While automation is conveniently defined for engineering, the definition for autonomy is more philosophic and used primarily to describe dependencies between countries, organizations, and individuals. Wood et al. [22] stated that despite the general public adopting the term *autonomous driving* to describe self-driving cars, the term *automated driving* actually is more accurate. The idea of automated driving can be derived quite easily from the automation definition as a vehicle that moves from one place to another without human control. Since most concepts for self-driving cars depend on prescribed destinations and utilize communication with other road users, the term autonomous becomes vague. Hence, the actual SAE J3016 norm [23] describes levels of automation instead of autonomy, which often is misconceived.

The five levels of automated driving

To create a standard for the automotive industry and research, the Society of Automotive Engineers (SAE International) designed a standard for the different stages of automated driving. The SAE standard J3016 was published in 2014 and became widely regarded as the industry standard for classifying driving automation levels. The automation level of a driving function can be determined by assessing some of its key properties. The main questions that have to be answered (Driver or system?) are:

- Who controls the vehicle (steering, acceleration and braking)?

- Who monitors the driving environment?
- Who will be the fallback level in dynamic driving tasks?

The official definitions in the SAE J3016 [23] for the different levels of automation are as follows:

0. Level: No Automation
The full-time performance by the human driver of all aspects of the dynamic driving task, even when enhanced by warning or intervention systems.

1. Level: Driver Assistance
The driving mode-specific execution by a driver assistance system of either steering or acceleration/deceleration using information about the driving environment and with the expectation that the human driver performs all remaining aspects of the dynamic driving task.

2. Level: Partial Automation
The driving mode-specific execution by one or more driver assistance systems of both steering and acceleration/deceleration using information about the driving environment and with the expectation that the human driver performs all remaining aspects of the dynamic driving task.

3. Level: Conditional Automation
The driving mode-specific performance by an automated driving system of all aspects of the dynamic driving task with the expectation that the human driver will respond appropriately to a request to intervene.

4. Level: High Automation
The driving mode-specific performance by an automated driving system of all aspects of the dynamic driving task, even if a human driver does not respond appropriately to a request to intervene.

5. Level: Full Automation
The full-time performance by an automated driving system of all aspects of the dynamic driving task under all roadway and environmental conditions that can be managed by a human driver.

A critical threshold in this schema is the transition from level 3 to level 4, due to the absence of required human supervision.

Assessing the safety of automated vehicles

Wachenfeld et.al. [24] pointed out that for an approval of fully-automated driving on public roads it should be shown that the ratio

$$V_{acc} = \frac{R_{add}}{R_{avo}} \tag{1.1}$$

between caused risk R_{add} and avoided risk R_{avo} should be less than 1 to ensure an overall positive impact on road safety. This idea is related to the so-called GAMAB principle that states in French *Globalement au moins aussi bon*, which translates to *performing globally at least as good as* [6]. The GAMAB principle is also embedded in the ethics report of the German Federal Ministry of Transport and Digital Infrastructure [9]:

> *The licensing of automated systems is not justifiable unless it promises to produce at least a diminution in harm compared with human driving, in other words, a positive balance of risks.*
>
> (German Federal Ministry of Transport and Digital Infrastructure [9])

This concept of a positive risk balance is also highlighted in the norm *ISO/TR 4804:2020* [25] on safety and cybersecurity for automated driving systems as a central requirement for the release of automated vehicles.

However, to calculate V_{acc} precisely, concrete measurements for R_{add} and R_{avo} need to be provided. One approach could measure risk by the number of accidents caused and avoided per kilometers. Although the total number of accidents in conventional traffic is available, it is not readily apparent to dissect which accidents would have been prevented by automated vehicles. Furthermore, the number of new accidents is unknown, so this approach has to be modified. This could be done by assuming that the number of "unavoidable" accidents is equal for the average human driver and the tested automated vehicle. Then V_{acc} could be approximated by dividing

$$\frac{\#Accidents_{\text{average human driver}}}{\text{driven kilometers}} \text{ by } \frac{\#Accidents_{\text{automated vehicle}}}{\text{driven kilometers}}. \tag{1.2}$$

The data for the first term can be retrieved from public traffic statistics. However, the second term cannot be observed straightforwardly. Since testing of an automated vehicle is required before release, accident statistics per kilometer cannot be obtained from customers. That leaves only statistics from supervised testing, with safety personnel who can intervene in hazardous situations. However, apart from the considerable danger of testing vehicles on public roads, reliable statistics require many test kilometers to be driven [26]. Based on data from the Federal Statistical Office of Germany, Wachenfeld [24] estimates the driving distance between two fatal accidents to be about 210 million. However, testing this amount of kilometers via real-traffic tests in real traffic is unrealistic [27, 28]. These and other concerns force the industry to find further approaches to testing automated cars, where the importance of simulated testing comes into play. Using virtual test environments that can predict the behavior of automated vehicles and other road users, test engineers can examine whether their product can handle hazardous situations safely [29, 30].

1.2 Deep learning as enabler of automated driving systems

Computer vision for automated driving

The developments in automated driving during the last decade are tied to the advances made in machine learning technologies [31], enabled by a rapid increase in computing power [32], and their ability to achieve human-like performance in tasks that are crucial for the safe behavior of the vehicle, such as object detection [33] or image classification [34, 35]. Machine learning algorithms are applied to various tasks in the architecture of automated vehicles [36, 37], such as perception, behavior prediction, and motion planning. Over the past years, models from the sub-discipline deep learning have shown to be the best approaches in machine learning for this purpose. Deep learning refers to machine learning models that are driven by deep neural networks. See Ch. 2 for a detailed definition of deep learning and the embedding in the field of machine learning.

Although perception functions are developed to a wide range of sensors, such as thermal imaging cameras [39] or acoustic sensors [40], the general consensus in industry and research focusses on three sensor types [41]: *Camera*, *Lidar*

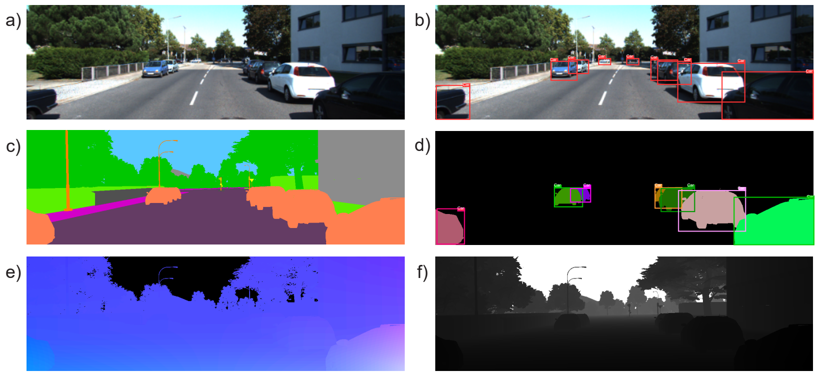


Figure 1.1: Different applications of deep learning technologies for perception in automated vehicles: a) Original Image, b) Object Detection and Tracking: 2D/3D bounding boxes. c) Semantic Segmentation: Pixel-wise classification. d) Instance Segmentation: Combination of bounding box detection and semantic segmentation. e) Optical Flow Estimation: Pixel-wise 3D motion in the scene relative to the vehicle. f) Depth Estimation: Pixel-wise distance estimation. Images and ground truths are provided by the KITTI benchmark data set [38].

and *Radar*. The sensors have advantages and disadvantages complementing each other.

Camera: The majority of work that has been done in computer vision concerns digital image data collected by cameras. A primary advantage of camera data is detecting texture features benefiting object detection and classification. Texture features allow distinction between objects with the same physical dimensions. On the flip side, camera data lacks depth information. The distance, dimension, and speed of an object is estimated based on color contrasts, which can be misleading. Furthermore, camera sensors are vulnerable to weather and lighting conditions such as snow, fog, or dazzling light sources.

Light Detection and Ranging (Lidar): Object detection algorithms for lidar sensors have seen a surge during the last years as lidar eradicates some of the shortcomings of camera data. Lidar sensors actively emit laser signals, usually in the 900nm wavelength range [41], to receive the signal's reflection on objects. Based on the duration between emission and reception, distance information about the object can be retrieved.

Hence, lidar sensors create high-resolution point clouds used for object detection. Compared to camera data, the availability of depth information allows precise localization of objects. However, lidar sensors are vulnerable to external light sources, and like camera sensors, they create a high amount of data. Since automated vehicles need to perform real-time perception, this represents a significant challenge.

Radio Detection and Ranging (Radar): Radar sensors are based on the same principle as lidar sensors. The sensor can estimate an object's distance by sending out and receiving electromagnetic signals in the microwave range. Furthermore, utilizing the Doppler effect, radar sensors can retrieve objects' speed. Depending on the signal wavelength, radar sensors operate long range up to 200 meters or in the vehicle's immediate surroundings. Compared to lidar, point clouds from radar sensors have a much lower resolution, making them more efficient to process [42].

To build up a representation of the vehicle's surroundings, the perception components gather information on the environment. Fig. 1.1 displays common tasks that have to be performed by the perception components using on image data [38].

Data as driving and limiting factor for perception

As deep learning algorithms are data-driven methods, they require vast amounts of labeled data to provide an algorithm with example data. A more formal definition of labels, as well as how deep learning algorithms use them for training is shown in Sec. 2.2. For this purpose, new data sets are published regularly, with some of the most critical benchmarks being the KITTI data set [38] and BDD100k data set [43]. These data sets are generated by traffic recordings being labeled manually. While this approach seems straightforward, it bears significant drawbacks. The data set distribution of recorded objects, weather artifacts, and road conditions will inevitably follow the real traffic distribution. While this does not seem problematical, it implies an inherited bias towards more common data fragments.

To highlight this point, Fig. 1.2 shows distribution information on the BDD100k data set [43]. With 100,000 labeled images, the BDD100k represents one of the most diverse development and benchmark data sets in camera-based percep-

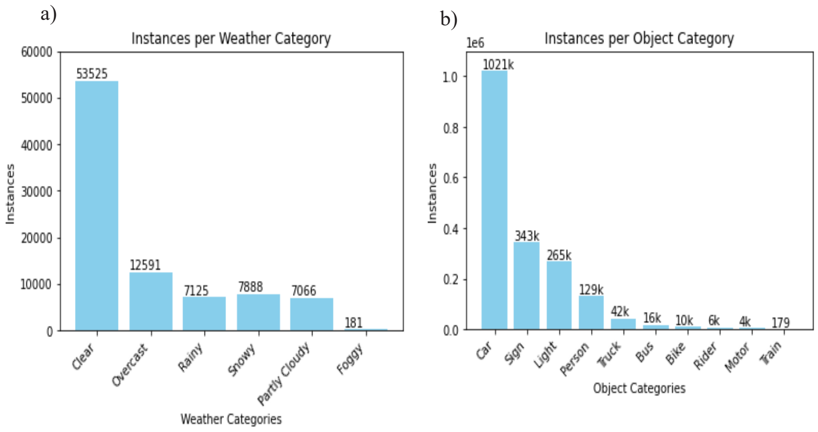


Figure 1.2: Data distribution inside the BDD100k data set [43] for a) Weather data (~13% of the 100,000 images do not have a weather label), b) Object bounding boxes. Note the imbalance at the expense of challenging situations such as *Foggy* weather or vulnerable road users such as *Person* or *Rider*.

tion. However, there are only 181 images in foggy weather, which is severely underrepresented. While foggy weather indeed occurs with a low probability in real traffic, it also negatively affects the performance of perception functions, with [44] showing a dramatic drop off of object detection networks in hazy conditions. Rare environmental conditions often pose additional risk to a perception function [45, 46]. Hence, relying solely on recordings for test data generation could require recording an immense amount of data that has to be labeled. At the same time, the labeling process is mostly the financial bottleneck of data generation due to high recording, labeling, and storage costs [47, 48].

The ISO 21448 norm *Road vehicles — Safety of the intended functionality* (SOTIF) [49] highlights the issue of limited scenario knowledge during model validation. ISO 21448 focuses on the critical aspect of scenario knowledge in the context of model validation for autonomous vehicles. The norm introduces the concepts of "safe" and "known" criteria, dividing the vast space of traffic scenarios into distinct categories. Fig. 1.3 visually represents this division, underscoring the importance of managing scenario knowledge for ensuring

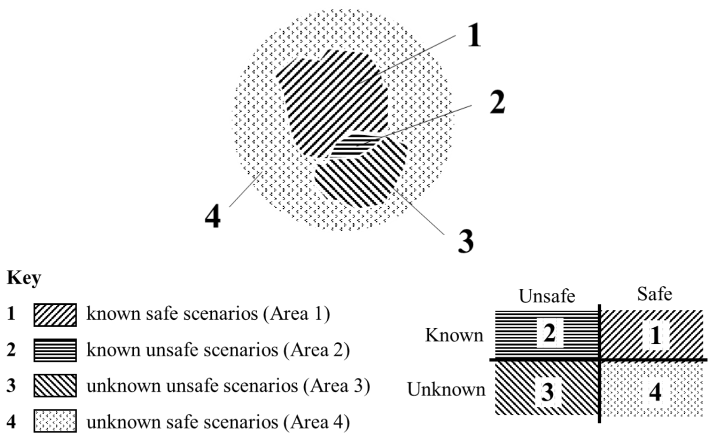

1
2
3
4
Key
1 ▨ known safe scenarios (Area 1)
2 ▤ known unsafe scenarios (Area 2)
3 ▧ unknown unsafe scenarios (Area 3)
4 ░ unknown safe scenarios (Area 4)
Unsafe
Safe
Known
Unknown
2
1
3
4

Figure 1.3: Criteria "safe" and "known" for scenarios in SOTIF context.

the safety of autonomous vehicles. In Sec 8.1, this theme gets addressed in context of the proposed testing techniques. There are two main tasks to manage during the development process:

- One key task involves expanding the size of known scenarios to observe a system's behavior comprehensively. This expansion serves a dual purpose: providing evidence of reliable model performance and defining the operational domain within which the system must operate. By increasing the repertoire of known scenarios, developers can enhance their understanding of how the autonomous vehicle responds to diverse situations, thus bolstering the reliability of the model.
- The second crucial task focuses on improving the model's performance to enlarge the safe scenario space in which the vehicle can operate securely. This entails refining the vehicle's ability to navigate and respond appropriately to a broader spectrum of scenarios, ultimately increasing the overall safety of autonomous operations. Through advancements in deep learning algorithms, sensor technologies, and decision-making processes, developers can optimize the model to operate safely in a wider array of scenarios.

Applying this logic to the BDD100k example above, the space of traffic scenarios in foggy conditions has to be considered as unknown and unsafe, as there is a significant drop-off in object detection performance.

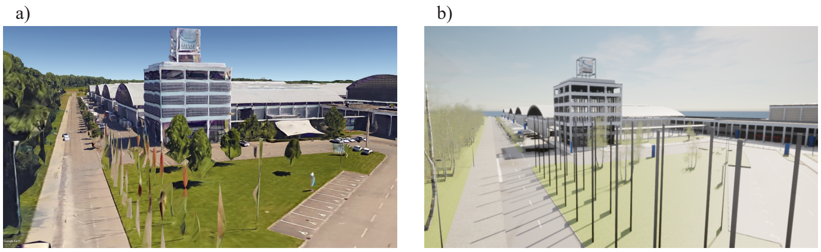


Figure 1.4: Visualization of virtualization capabilities of simulative frameworks. a) Image of Messe Friedrichshafen (exhibition center) in Google earth. b) Virtual representation of Messe Friedrichshafen inside CARLA simulator [4].

1.3 Scientific contribution

In light of the presented challenges for industry and research when assessing the safety of ADAS technology concerning environmental perception in Sec. 1.1 and Sec. 1.2, this dissertation focuses on the simulative testing of perception functions. Due to the notorious struggle of obtaining labeled image data when testing perception functions, simulation software generates desired synthetic image data to complement the recorded image data set and stress the perception function.

In contrast to expensive labeling, simulation software automatically generates ground truth information, allowing the observation of whether the behavior of the perception function complies with the functional requirements. The virtual KITTI data set [50] is a prime example for the generation of unsafe and unknown scenarios. By modeling scenarios from the original KITTI data set in a virtual environment, the scenarios can be re-simulated in changing weather conditions to address the issue of unbalanced data sets. The ability to generate a proxy of the real world is highlighted in Fig. 1.4 with a complete virtualization of the exhibition center in Friedrichshafen in the CARLA simulation. The most significant advantages of synthetic image data can be summarized with:

Generate data "on-demand" Synthetic data generation allows the creation of a wide range of scenarios, including rare or extreme cases that may be challenging to encounter in the real world, hence helping in testing the robustness of the perception function under diverse conditions. Furthermore, synthetic environments offer precise control over various simulation parameters, such as lighting, weather conditions, and traffic density. This control is valuable for assessing specific variables and understanding their impact on system performance.

Closed-loop testing and reduced risk for participants Closed-loop testing of perception functions requires the functions' output to influence the next testing step. Especially for autonomous vehicles or other safety-critical systems, simulation data eliminates the risk to real participants or objects. This is particularly important when testing in potentially dangerous or unpredictable scenarios, as perception failure would cause dangerous vehicle behavior.

High label quality Many resources are poured into data labeling. Faulty or imprecise labels compromise the perception function's performance. Synthetic data is generated with correct label quality as the simulation software assigns correct label classes to each pixel.

"Cheap" data generation and scalability Unlike expensive data collection campaigns in real-world traffic with subsequent labeling, simulation tools generate low-cost sensor data with corresponding ground truth. Google Cloud Services provides a watermark for the cost of data labeling of real-world data with up to 1$ per frame depending on the labeling task [51], not including the cost for recording in real traffic. Meanwhile, simulation software such as CARLA runs near real-time on mid-tier GPUs like the NVIDIA Quadro P2000 used for this dissertation's experiments, allowing parallelization and up-scaling of the data generation process.

Data generation early in the development process and rapid prototyping Data collection and testing in real traffic or on dedicated test tracks requires not only having the final camera sensor available but also the final sensor outline, which may be subject to change during the development process of the ADAS system. Such a change would require a new data collection campaign. By changing the virtual sensor setup or the sensor model, synthetic data generation can be triggered immediately without having all final system components in place, increasing the development speed and enabling rapid prototyping.

However, while simulation tools resolve some key weaknesses of real-world data collection, problems arise with the high variability of the space of traffic scenarios. The simulation needs to be structured and guided systematically, optimizing the use of all available resources. Randomly running a traffic simulation would decrease cost and increase scalability and speed but not use the "on-demand" advantage above. In practical terms, despite the powerful simulation tools, the testing process must select a subset of traffic scenarios for simulation and testing due to these constraints, which can be summarized by:

- Although simulation software can theoretically generate **every** possible traffic scenario, the testing process cannot generate **all** scenarios together due to simulation constraints.

Given a standardized description of traffic scenarios, a traffic scenario space is spanned from which scenarios can be sampled. Hence, there is a need for a test case generator as defined by:

Definition 1.1 (Test case generator [52])**.** *A test case generator is a software tool that accepts as input source code, test criteria, specifications, or data structure definitions; it uses these inputs to generate test input data and, sometimes, determines expected results.*

For the sake of this dissertation, a **sampling strategy** is defined as the internal logic of the test case generator that guides the test case generation (see Def. 4.14). Sampling strategies are vital to use the available simulation resources most efficiently to address the problems arising from the traffic scenarios' vast combinatorial space. Assuming that the simulation tool can generate every traffic scenario, the test process capacity is insufficient to test and save all traffic scenarios. Hence, a scenario sampling strategy must be conceived to prioritize traffic scenarios based on coverage and performance metrics.

Provided the issues regarding the safety assurance of DL-driven perception components in automated driving, the scientific contribution and research questions of this dissertation can be summarized as follows:

- With the introduction of deep learning, established development processes are subject to change and new concerns arise regarding the use in safety critical applications. **What are the key challenges and requirements for DL-driven perception functions in automated driving?**
- The issue of testing deep learning models has been addressed by multiple academic and industrial works. **What are existing approaches to test the performance and robustness of deep learning models?**
- To ideally use simulation resources most efficiently, a parametrized and machine-readable scenario space is required. **How can simulation software be used to build up a pipeline for systematic testing of perception functions in a parametrized traffic scenario space?**
- **Using simulative software for data generation, how can parameters in synthetic data of environmental images be identified that pose risks to the perception?**
- Using the generated image data from the testing process, DL-driven perception functions can be re-trained to address insufficiencies. Testing on additional scenarios, how significant is the improvement in performance to a perception function compared to the original perception function. **Can adaptively generated synthetic image data be utilized to improve the performance and maturity level of a perception function?**

2 Fundamental techniques of environmental perception in automated driving

2.1 Traditional programming vs. machine learning

The distinction to traditional programming should be examined to understand the challenges that the safety assessment of machine learning algorithms poses. While traditional programming usually utilizes hand-crafted rules for decision-making, machine learning algorithms derive their decision-making from the data with which they were trained.

Their real strength lies in this data-based concept of machine learning algorithms: *Some machine learning applications are intended to learn properties of data sets where the correct answers are not already known to human users [53]. There would be no need for machine learning if these questions would already be answered [54, 55].* Based on training examples, the model automatically generates the rules to solve the task-at-hand. This rule generation is guided by a fitness function that measures the models' quality [56]. Hence, the training of machine learning models almost always comes down to a numerical optimization problem.

However, this also adds another dimension of complexity, as three primary sources can cause failures of machine learning models: The data it was trained with, the framework that has been chosen for a specific task, and the code that executes the chosen and trained model [55]. A representation of the development processes of traditional software and machine learning models is displayed in Fig. 2.1. While in traditional software engineering, the underlying rationale is hard coded by the developer, the machine learning developer specifies a model architecture. Finally, specific decision-making is derived from the concrete model trained with the collected training data.

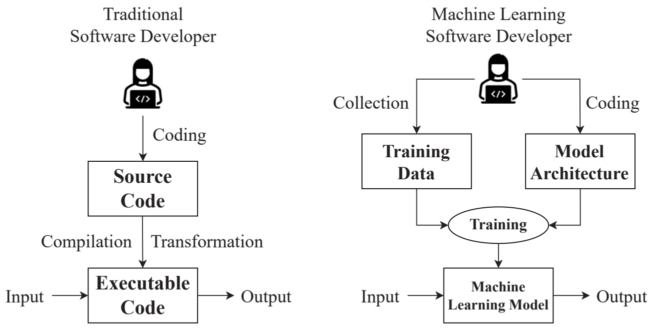


Figure 2.1: Differences between the development processes of traditional software and machine learning models. Figure by Li et al. [57]

.

2.2 Types of machine learning

Machine learning models can be categorized into three main classes, depending on the available training data and the scheme the model extracts information with: Supervised learning, reinforcement learning and unsupervised learning [55].

Supervised learning

The main objective of supervised learning is to make predictions of a target value of future data based on available training data [58]. Assume there is a true underlying function

$$f : \mathcal{X} \rightarrow \mathcal{Y}, \tag{2.1}$$

for which no mathematical definition is available. Supervised algorithms aim to find a way to imitate the behavior of the function, using a training set $\hat{\mathcal{X}} \subset \mathcal{X}$ for which the results $\hat{\mathcal{Y}} = f(\hat{\mathcal{X}})$ of the real underlying function are available. In this context, the data set $\hat{\mathcal{X}}$ and $\hat{\mathcal{Y}}$ are referred to as *ground truth*. A schematic representation for this task can be seen in Fig. 3.2, used to demonstrate the importance of input space coverage. Referring to the introduction in Sec. 2.1, if the value of the underlying function $f(x)$ is available for all $x \in \mathcal{X}$, there would be no need for fitting a machine learning model, as a simple look-up table would be sufficient.

A key characteristic of supervised learning, especially when compared to unsupervised learning, is that the nature of the target space $\mathcal{Y}$ is known, meaning that we know of which elements $\mathcal{Y}$ consists. Depending on this manner, there are two tasks of supervised learning [58]:

Classification: In classification tasks, the true underlying function $f(x)$ maps every element of $\mathcal{X}$ to some class-id $\mathcal{Y} = \{c_1, .., c_n\}$, making the target space $\mathcal{Y}$ discrete. The machine learning model is trained to make predictions on this class assignment for data point instances with the true class being unknown. An intuitive example for this task is the calculation of class probabilities $\{P(\mathcal{Y} = c_1|x), .., P(\mathcal{Y} = c_n|x)\}$, where each element of the vector denotes the calculated probability of x belonging to the corresponding class. The predicted class would be the class with the highest probability. Such concepts are also applied if, rather than probabilities, class scores are calculated. This example demonstrates the importance of knowing the nature of $\mathcal{Y}$: If there is a class $\hat{c}$ of which the model does not know, no elements $x \in \mathcal{X}$ could be mapped to $\hat{c}$.

Regression: Regression tasks can be interpreted as the continuous version of classification tasks. Usually, the target space $\mathcal{Y}$ is some sub-set of $\mathbb{R}^n$, $n \in \mathbb{N}$. A typical regression model predicts the actual value of $f(x)$ for a given $x \in \mathcal{X}$ and aims for a minimal prediction error. The continuous manner of regression tasks often is ideal for typical numerical optimization, such as gradient descent optimization. Hence, discrete classification tasks are usually transformed to regression problems to apply these optimization techniques. Afterward, the regression results are discretized back to the target space of the classification task. This concept will reappear in Sec. 2.4 when the training of neural networks is presented.

Unsupervised learning

In contrast to supervised learning, unsupervised learning models are applied if there is no information on an underlying function. Instead, the information has to be extracted from the data structure by analyzing distances between data points in the available data set. Hence, unsupervised learning methods are highly dependent on distance metrics that measure the dissimilarity between data points. There are two main categories of tasks in unsupervised learning [58]:

Clustering: Clustering tasks basically are a form of classification with nothing known about target space $\mathcal{Y}$. The model assumes that there is an underlying grouping of the elements. However, the training data does not provide explicit information on a possible target space $\mathcal{Y}$, starting with the number of groups. Instead of imitating the behavior of a true underlying function, clustering models group the available training data points so that the variance inside each group is minimized.

Dimension reduction: Dimension reduction is a method that is used to compress data, based on an example data set. An example notation of a dimension reduction function is

$$f : \mathbb{R}^n \rightarrow \mathbb{R}^m, \qquad n > m, \tag{2.2}$$

where data from a high-dimensional space $\mathbb{R}^n$ is embedded in a lower-dimensional space $\mathbb{R}^m$. The function should be chosen so that the loss of information is minimized. This means that the structure of distances between data points is preserved. A prominent example of dimension reduction is the down-scaling of images if the original resolution is unnecessarily high for a computer vision task. Dimension reduction is a valuable tool in data visualization, as data from four-(or higher) dimensional spaces are challenging to display in charts. Embedding the data in a three-(or less) dimensional space allows for a more intuitive visualization if there is no significant loss of information.

Reinforcement learning

Reinforcement learning models aim to find an optimal decision-making process to perform a task. In reinforcement learning, the decision-making process is called an *agent*. This agent performs so-called *actions* in an *environment*. The past /current /future positions of the actor inside the environment are called *states*. The main difference to supervised learning is that the agent does not

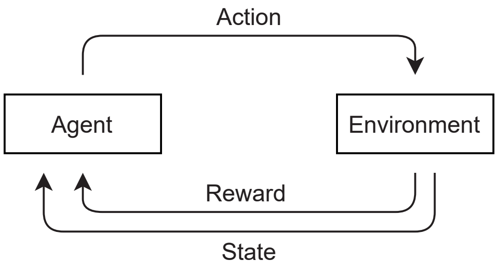


Figure 2.2: The basic concept of reinforcement learning. Based on the current state and previously observed rewards, an actor tries to choose the optimal action in its environment and receives the following reward feedback and the future state of the actor.

make plain predictions for a given data point but rather executes a strategy of decisions that aims to achieve a long-term objective. Decisions are *good* or *bad*, instead of *right* or *wrong*. The actor evaluates each available decision for expected *reward* or *punishment* and chooses its action accordingly. Based on previously observed consequences for its actions, the actor learns to optimize its actions. This concept is displayed in Fig. 2.2.

The most prominent example of a reinforcement task is the chess game. The decision-making process (actor) performs actions on the chessboard (environment) in order to maximize the reward (winning the game). Here, the strategic queen's sacrifice is an excellent example of the differentiation between supervised learning and reinforcement learning. While a supervised algorithm would categorically reject the surrender of the most powerful piece as a *wrong* decision, a reinforcement agent could swallow a short-time *bad* decision for achieving long-term success.

2.3 Requirements for machine learning algorithms

There is a range of quality requirements that a machine learning model must comply with when applied for safety-critical tasks. Defining requirements has been done in several academic contributions, noteworthy for this dissertation Zhang et al. [55] and Cheng et al. [59], as well as governing and regulating bodies, such as the European Commission with their *Ethics guidelines for trustworthy AI* [60] and the International Organization for Standardization with the norms ISO 25010:2011 [61] and ISO/TR 4804:2020 [25]. The existing work on requirements overlaps and complements each other. These requirements address various aspects of the model's functionality and properties. The following enumeration summarizes and details vital requirements that a machine learning model must comply with. Although the requirements are universal for machine learning applications, the focus is on applying machine learning algorithms in perception tasks.

Correctness

Correctness refers to the degree to which the model can be relied on. The generation of accurate results is one of the central requirements for a machine learning model. Correctness can be interpreted as the probability of *getting things right* [55]. Since correctness is the essential requirement for this dissertation, some key concepts are presented in the following.

Evaluating the correctness of machine learning models is usually done using performance metrics. A well-known and intuitive system of evaluating classifiers is the calculation of model accuracy derived from confusion matrices, as shown in Fig. 2.3. Confusion matrices map the true class of an element against the class predicted by a classification or clustering model. While Fig. 2.3 displays the standard confusion matrix for binary classifiers (true/false), the underlying rational can be transferred to multi-class problems.

The accuracy of a model can then be calculated with the formula

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN}, \tag{2.3}$$

		Decision	
		Positive	Negative
Ground Truth	Positive	True Positive (TP)	False Negative (FN)
	Negative	False Positive (FP)	True Negative (TN)

Figure 2.3: Classic confusion matrix for binary classifiers.

where TP, TN, FP, FN is defined according to Fig. 2.3. For a binary classification task, a true positive (TP) is a data point labeled $positive$ that was correctly predicted. Conversely, a false positive (FP) is a data point labeled $positive$ that was predicted as $negative$. True negatives (TN) and false negatives (FN) are defined inversely. Hence, accuracy measures the fraction of data points that fall on the diagonal of the confusion matrix.

However, the number of TN is often not available or hard to calculate. For example, imagine the task of object detection: While TPs are all correctly detected objects, FNs are all objects that are not detected, and FPs are all false detections, the set of TNs are all non-objects that has not been detected. Hence, the TNs are all possible false bounding boxes that the object detector has not detected. The number of TNs would be almost impossible to calculate and hardly bears valuable information anyway. In this case, accuracy may be misleading, as accuracy naturally converges to 1 if the number of TN increases - as it is likely in unbalanced data sets.

In such cases, a machine learning model is evaluated by the two measurements *Precision* and *Recall* that do not consider the amount of TNs.

Precision evaluates how many of the positive decisions are truly positive:

$$\text{Precision} = \frac{TP}{TP + FP}. \tag{2.4}$$

Optimizing precision can be done using a more restrictive selection strategy. For example, if the model assigns only elements as positive if there is high confidence, the number of FPs is minimized.

Recall evaluates how many of the truly positive elements have been selected by the model:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2.5}$$

Optimizing recall is somewhat trivial using an aggressive selection strategy. By assigning every element to positive, all truly positive elements are selected. Hence to ensure reliable behavior on the available data, a good balance between precision and recall has to be found.

If the model generates continuous results, correctness is measured by so called *loss*-functions. A loss evaluates how far the models' predictions are from their true values and - with other words - how bad the model is [62]. Loss functions are usually derived from distance functions to measure dissimilarity. The most common example for a loss function is the mean squared error

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2, \tag{2.6}$$

with $\hat{Y}_i$ being the prediction for value Y_i.

Robustness

The IEEE Standard Glossary of Software Engineering Terminology defines robustness as following.

Definition 2.1 (Robustness of a system)**.** *The degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions" [52].*

While correctness addresses general performance requirements of the model, robustness addresses the models' resilience towards adversarial perturbations [55]. In computer vision tasks there are two main sources that are related to perturbations:

Natural Distortions: A well-known source of challenges of neural networks in computer vision is natural effects [63]. There is a significant drop

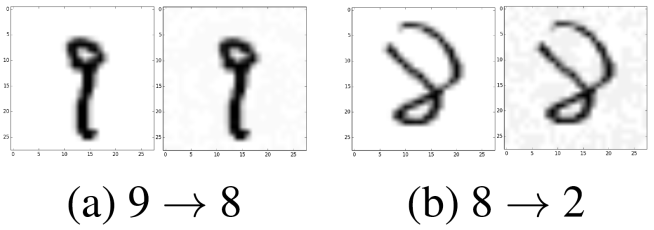


Figure 2.4: Example for adversarial attacks on a classifier for the MNIST (Modified National Institute of Standards and Technology database) database of handwritten digits. Small changes in form of random noise to the input image lead to a misclassification. Figure by [66].

in performance if the perception function is confronted with adversarial weather effects, such as rain or fog [46,64]. Complete resilience towards environmental effects is unfeasible - if an object is fully covered by fog, there is no visual information for reliable detection. However, certain robustness in adversarial weather has to be assured. The use of adaptive data generation using simulative data to increase robustness is a research goal of this dissertation and will be presented later.

Adversarial Attacks: In contrast to natural distortions, adversarial attacks are perturbations especially targeting the network's performance. The idea behind adversarial attacks is the assumption that small, for the human eye not striking, artificial changes to an image may be able to fool a network to make a misclassification [65]. An example can be seen in Fig. 2.4. These attacks are mainly generated by adversarial models trained to trick the perception function. For example, placing adversarial patches in the real world could cause the miss-detection of pedestrians through an ADAS vehicles' perception function with fatal consequences.

Exploring a network's vulnerability towards adversarial effects and using these insights during the models' development is vital to estimate and improve the robustness.

Coverage

Coverage or completeness is a requirement that addresses various aspects of the development process. Coverage requirements ensures that the test set of the model covers a wide range of the states and conditions [59]. In machine learning testing two main research areas deal with coverage issues:

Neuron Coverage: Neuron coverage arises from the well-known practice of code coverage in traditional software development applied to the connecting structure of neural networks. In a nutshell, code coverage is based on the assumption that a test suite for software should be able to trigger the execution of each line and branch of the software's code [66]. Simultaneously, neural networks should be tested by data sets that trigger areas of the networks. Finally, the test data set's information value is measured by the share of nodes and branches of the network that are executed at least once while running the data set [67]. If not all areas of the network were triggered during testing, there is a risk of unexpected behavior if these regions are triggered during deployment

Scenario-based Coverage: Scenario-based coverage is a black-box approach to measure completeness of data. Intuitively, a machine learning method should have been tested by a test set covering the different situations the network may be confronted with during deployment [68]. Scenario-based coverage is vital to prevent unexpected behavior for some environmental corner cases. As presented in this dissertation, simulative data generation techniques are a viable tool to increase coverage metrics of existing test sets to meet the coverage requirements [69].

A more detailed review of techniques for optimizing coverage metrics is provided later in the dissertation.

Interpretability

Google's machine learning glossary defines interpretability of machine learning models as "The ability to explain or to present an ML model's reasoning in understandable terms to a human" [62]. For social acceptance and public trust in machine learning-driven systems, it is essential for humans to understand the cause of decisions [55] and what the underlying machine learning model

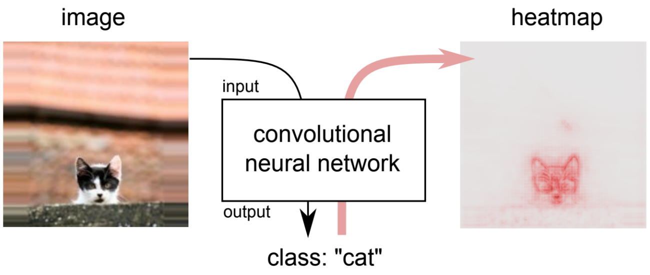


Figure 2.5: Concept of heatmap approaches for visualization of computer vision algorithms. The decision of the neural network is traced back to each pixel of the original image, allowing to analyze which image components caused the decision. Figure by [71].

has learnt [59]. These insights can be used to avoid discrimination, identify weaknesses and transfer knowledge to other situations [55, 70].

There are approaches to retrieve the reasoning of perception functions, such as heatmap algorithms for object detection [71,72]. In general, heatmaps visualize the parts of the image that triggered a decision, allowing for interpretation, as displayed in Fig. 2.5. Ideally, the relevant image parts for a detection overlays with the object's position, assuring no overlooked influences to the neural network [59].

Ribeiro et al. [73] visualized this aspect. While training a classifier to distinguish between the class *husky* and *wolf*, the training data for class wolf showed the wolves in a snow environment and the huskies in a non-snow environment. As a result, the classifier learned to distinguish based on the environment. Hence, a husky in a snow environment may be misclassified as a wolf, as shown in Fig. 2.6.

Efficiency

Efficiency relates to the amount of time that the system requires to perform its designated function [52]. Especially in applications with high-dimensional data, training and deployment of machine learning models may be too slow. Especially in automated driving, this requirement is of particular interest due to the catastrophic impact of delays in model performance [74].

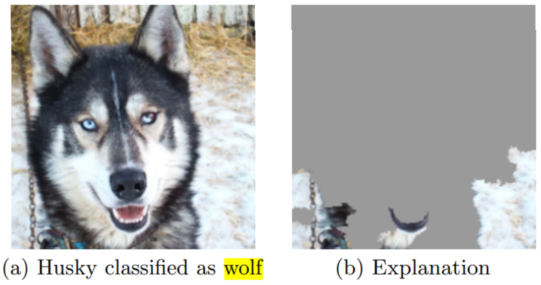


Figure 2.6: Importance of interpretability in machine learning. Ribeiro et al. [73] trained a simple classifier to distinguish between the classes *husky* and *wolf*, with the wolves in a snow environment and the huskies in a non-snow environment. The model learns to make its decision based on the background, which causes misclassifications if there is a husky in a snow environment.

Huang et al. [75] examined the effect of increasing model complexity to different object detection frameworks, e.g., by varying the number of bounding box proposals or using different feature extraction backbones, on model speed and efficiency. They found a significant trade-off between model speed and model accuracy that has to be managed. However, in some applications, the relevance of efficiency requirements may even lead to preferring a model with lower correctness measurements due to better efficiency [55, 76].

Fairness

The ISO norm ISO 25010:2011 *Systems and Software Quality Requirements and Evaluation* defines fairness as freedom from discrimination and bias [61]. Discrimination may arise from an individual's characteristics that should be considered as *protected* [55, 77]. Protected characteristics include race, color, sex, religion, national origin, citizenship, age, disabilities, or familial status [55].

A perception function can only discriminate by characteristics that it can perceive. An intuitive example of fairness in computer vision is the detection rate of wheelchair users. Although there is a rare appearance rate for wheelchair users in traffic, ignoring low detection rates on this traffic participant class is unacceptable. Using standard methods for reliability analysis, there would be a risk of this deficiency being "swept under the carpet" due to low appearance rates. Transferring the requirement of fairness towards an object detection algorithm, each road user should have "the same right to be detected".

In contrast to the other ethical requirements that are presented in the next paragraph, fairness is a quantitatively measurable characteristic of a machine learning model given a knowledge of the classes.

Requirements on ethical Framework

Additionally to the presented technical and measurable requirements, there are ethical requirements on the development and application process framework. The European Commission defines seven requirements of artificial intelligence in their *Ethics guidelines for trustworthy AI* [60]: *1) Human agency and oversight*, *2) Technical robustness and safety*, *3) Privacy and data governance*, *4) Transparency*, *5) Diversity, non-discrimination and fairness*, *6) Societal and environmental well-being*, and *7) Accountability*. While the report uses the terminology of artificial intelligence rather than machine learning, the reports' definition of artificial resembles mostly the description of machine learning in Sec. 2.1 and hence will be substituted with the term machine learning. Requirements 2), 4), and 5) have been already discussed above as correctness, robustness, interpretability, and fairness. The other requirements are summarized as follows:

Human agency and oversight: A machine learning model should not interfere with fundamental rights, such as the autonomy and decision-making of humans [60]. The machine learning model should assist the users' decision-making instead of patronizing the user in user-machine interactions. This includes the users' right to overrule the machine learning models' decisions.

Privacy and data governance: Apart from requirements towards data quality related to scenario-based coverage, this requirement also addresses the correctness of data and the accessibility of data. During all stages of data collection, model development, and model deployment, private and individual information must be protected from unauthorized access from third parties. This includes the user's initial data and data generated on the user during deployment. Furthermore, the data should be free from socially constructed biases, inaccuracies, errors, and mistakes.

Societal and environmental well-being: The European Commission sets the principle that a machine learning model should benefit "all human beings including future generations" and highlights the ecological responsibility of machine learning models. This also applies to machine learning algorithms in situations relating to the democratic process and the manipulation of political decision-making and the electoral process.

Accountability: Before applying machine learning in critical applications, a framework needs to be created that handles the responsibility and accountability for the machine learning model. This includes the independent auditing of models in safety-critical applications. Furthermore, reporting negative impacts and weaknesses should be transparent, including protection for whistle-blowers, NGOs, trade unions, or other entities.

2.4 Deep learning - origin and foundations

The early development of artificial neural networks was inspired by biological neurons' functionality in the human brain. In simple terms, the human brain consists of neurons that communicate by sending and receiving electrical impulses. As soon as received signals' totality exceeds a neuron's threshold, the neuron activates and sends a signal into the network.

In 1943, [78] formulated a logical model of a neuron. If the neuron cell's binary inputs exceed a threshold, the neuron's output is set from zero to one. Fig. 2.7 a shows a representation of this so-called *Linear Threshold Unit* or LTU. However, the LTU did not include weights for the input, and the threshold had to be set manually.

The first approach to introduce weighted connections between the nodes that can be learned was inspired by a neurological theory of psychologist Donald Hebb [79] in 1949. Hebb assumed that the connections between two neurons strengthen when the receiving neuron activates after receiving a signal from the transmitting neuron. Based on these findings, in 1953, Frank Rosenblatt [80] introduced the perceptron. In contrast to the LTU, the binary inputs are

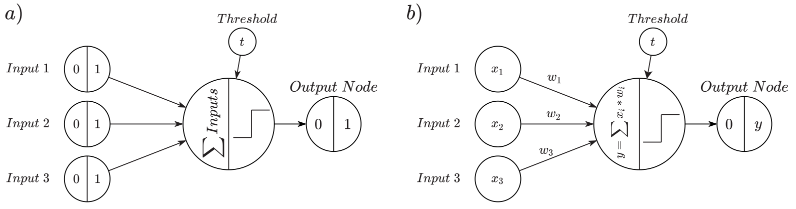


Figure 2.7: a) Binary *Linear Threshold Unit*. If the totality of inputs reaches a threshold, the LTU activates. b) Single perceptron with three inputs and one binary output. The output is set to y if the weighted sum of inputs exceeds a certain threshold and set to 0 otherwise.

weighted to represent the nodes' connection strength. If the weighted sum of inputs exceeds a threshold, the neuron is activated. Fig. 2.7 b shows a single perceptron with three input and one output value.

The Hebbian learning algorithm, which is considered the first algorithm to train weights, is based on the Hebbian theory. The rule is expressed by the equation

$$w_i^{new} = w_i^{old} + \mu * x_i * \hat{o}, \tag{2.7}$$

where w_i is the weight between input i and the output neuron, x_i is the value of the input, $\hat{o}$ is the actual output of the perceptron, and μ is the learning rate that indicates how much the weight has to be updated. Hence, after activating the output neuron, the weight is updated. If either input or the output is zero, the weight is not updated. However, this learning rule is not supervised.

A single perceptron with a linear activation function can only be applied to linear separable sets, like the OR-function. An application of a linear perceptron is displayed in Fig. 2.8 a. The perceptron activates for data points that lie above the hyperplane, i.e., if $1 * x_1 + 1 * x_2 > 0.5$. On the other hand, non-linear separable sets like the XOR function cannot be solved by a single linear perceptron.

A multi-layer neural network can be created using multiple perceptrons' outputs as inputs to a new perceptron. An application of two linear perceptrons to solve the XOR problem is shown in Fig. 2.8 b. The neuron in the second layer activates if both first layer neurons activate: The point $\{x_1, x_2\}$ has to be above hyperplane 1 ($1 * x_1 + 1 * x_2 > 0.5$) and bellow hyperplane 2 ($-1 * x_1 + -1 * x_2 > -1.5$). Fig. 2.9 displays a graph for this multi-layer network. The output layer has the weights $\{1, 1\}$ and the threshold 1.5. Hence, only if both hidden neurons activate the output layer activates. The hidden neurons are often referred to as *features*. The activation of a neuron indicates the presence of this feature.

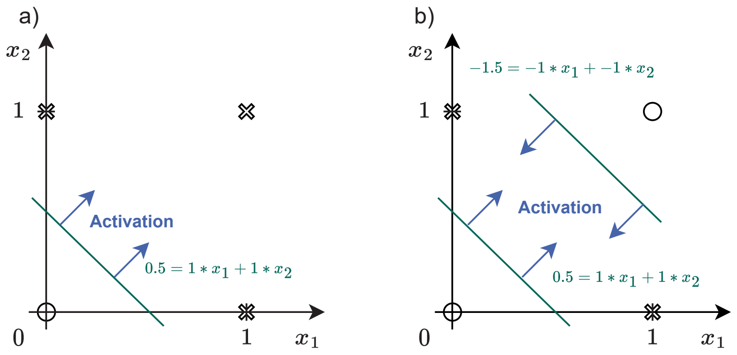

a)
x_2
1
0
1
x_1
Activation
$0.5 = 1 * x_1 + 1 * x_2$
b)
x_2
$-1.5 = -1 * x_1 + -1 * x_2$
1
0
1
x_1
Activation
$0.5 = 1 * x_1 + 1 * x_2$

Figure 2.8: a) Application of a single perceptron with weights $w = \{1, 1\}$ and threshold $t = 0.5$ on the OR problem. The decision is *cross* for all data points with $1 * x_1 + 1 * x_2 > 0.5$. The hyperplane separates the two classes *cross* and *circle*. b) Application of two perceptrons with weights $w_1 = \{1, 1\}, w_2 = \{-1, -1\}$ and thresholds $t_1 = 0.5$, $t_2 = -1.5$ on the XOR problem. Data points that satisfy both conditions are assigned to *cross*.

In 1986, Rumelhart et al. [81] introduced the backpropagation algorithm. Backpropagation is a supervised method to adjust weights by calculating the difference between a network's actual output and the desired output, also called ground truth. The error is propagated backward through the layers. Hence, the weights between the next-to-last and last layers are updated first, while the weights between the input and the first hidden layers are updated last. Despite some changes during the last decades, this principle remains a standard learning method for neural networks. Backpropagation depends on differentiable activation functions to perform gradient descent methods. Hence, using the traditional step function for optimization was unfeasible. Today, there is a wide range of activation functions. Fig. 2.10 shows the sigmoid function that is a continuously differentiable approximation of the step function. However, the absence of sufficient computing capabilities and high volume data sets has curbed the research in this area. Since the mid-2000s, the

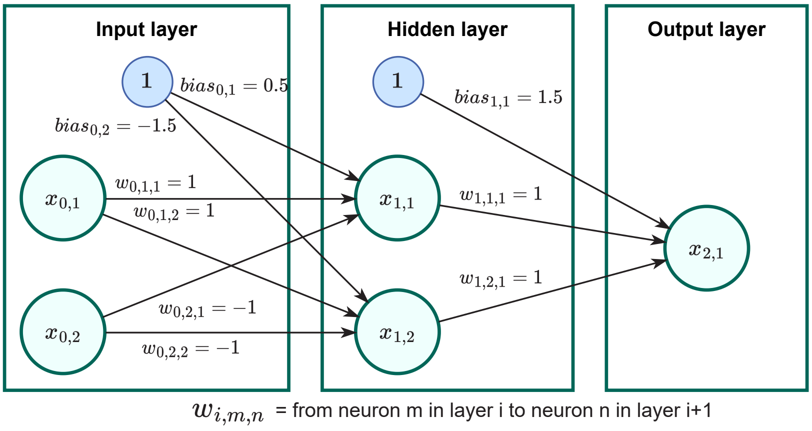
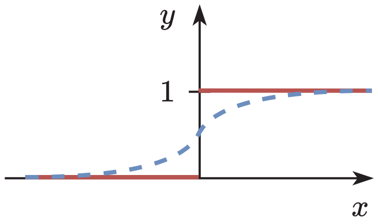


Figure 2.9: Network with one hidden layer to solve the XOR problem (Weights are chosen according to Fig. 2.8). Output of neurons are binary.

availability of resources and theoretical improvements has led to deep learning in many disciplines, such as time-series analysis, computer vision, or natural language processing [82].



$$step-function = \begin{cases} 1, & if\ x > 0 \\ 0 & otherwise \end{cases}$$

$$sigmoid-function = \frac{1}{1+e^{-t}}$$

Figure 2.10: Comparison between step function and sigmoid function. In contrast to the traditional step function, the sigmoid function is continuously differentiable.

Fully connected networks, such as the network of Fig. 2.9, where each neuron of layer i is connected to each $i+1$, have shown difficulties in computer vision tasks. Connecting all nodes of two layers results in a nearly unfeasible amount of connections when working with high-dimension images. Furthermore, every feature in hidden layers is globally connected to the input layer. Hence, the network struggles to extract local features, such as edges or contrasts. Convolutional structures have been introduced to counter these shortcomings. Convolutional layers consist of multiple filter matrices that slide over the image. Activation of a filter represents a local feature at this position. Fig. .1 a and Fig. .1 b display the difference between a fully connected layer and a convolutional layer. Applying concatenated convolutional layers returns more global features and structures. For face recognition, local or low-level features can be horizontal or vertical edges or specific contrasts. Global or high-level features can be facial characteristics, such as the nose or the eyes.

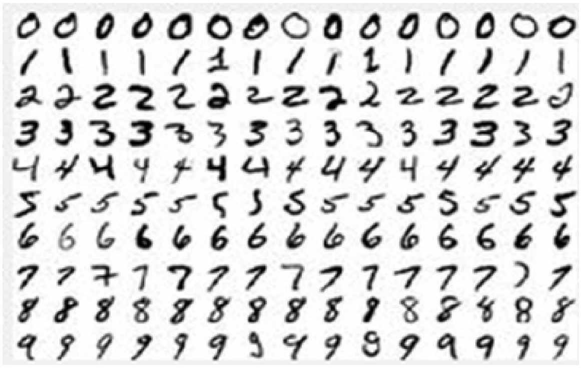


Figure 2.11: Sample digits of the MNIST (Modified National Institute of Standards and Technology database) database of handwritten digits.

Convolutional neural networks for computer vision tasks had a significant breakthrough in 1998 by LeCun et al. [83] with the creation of LeNet-5. The network significantly outperformed existing methods for recognition of 32× 32-pixel images of handwritten zip code digits. Fig. 2.11 shows a sample out of the MNIST data set, which is famous for being an accessible database for image classification algorithms.

In 2012 Krizhevsky et al. [84] published AlexNet, which achieved remarkable results by winning the ImageNet challenge with an error rate of 16%, outperforming the runner up by a considerable lead of 9%. Since then, deep learning technologies have made big leaps, being competitive to human perception [34].

2.5 Object detection frameworks

Appx. A contains a detailed description of the main building blocks in deep learning and how convolutional neural networks operate. Object detection is based on a multilevel pipeline that has to perform object localization as well as classification. Often, classification networks build the core of these pipelines. They can be defined as a function that assigns images out of the image domain to classes out of the class domain, formally defined by

$$C : [0, ..., 255]^{h\times w\times d} \rightarrow \{c_1, ..., c_n\}, \tag{2.8}$$

where $h,\ w$ are the dimensions of the image, d is the number of color channels (usually three channels for RGB) and $c = \{c_1, ..., c_n\}$ is a vector containing all available classes. The values for each position of the image and each color channel are discretized and can take on a value between 0 and 255. Such a classification network is displayed in Fig. .2. With an object classification network as a backbone, there are different processes to localize objects in images. Object detector pipelines extract a list of object bounding box detections. There is a range of frameworks, of which three basic concepts are detailed in the following.

You only look once (YOLO)

Its straight forward architecture and ability to run real-time has made the YOLO framework [85] a popular tool. As the name suggests, a single network processes an input image to get a fixed number of bounding box proposals. For YOLO, images are divided in a $S \times S$ grid. For each field, the network estimates B proposals for bounding boxes for objects centered in this field along with a confidence value representing the chance of the bounding box containing an object. A bounding box proposal consists of a five-dimensional vector (x, y, h, w, p). x and y represent the bounding box's center, h and w represent the bounding box's height and width, and p represents the confidence. Suppose no object in the image has its center in a given field. In that case, the confidence values for the B proposals are expected to be close to zero. Furthermore, for each field, the network estimates a class probability for each object class. Overall, with n classes, the network estimates a $B * 5 + n$ values for each field. Fig. 2.12 displays a schematic representation of the original YOLO network with a 7×7 grid, $B = 2$ bounding box proposals per field, and $n = 20$ classes.

However, the YOLO network returns $S \times S \times B$ different bounding box proposals. Many are likely to overlap while referring to the same object. This problem is solved by the non-max suppression method, which is structured as follows. The first step is to discard all bounding boxes with $p < t_p$, where t_p is a threshold for minimum confidence. The bounding box with the highest confidence is extracted and added to the output list. Afterward, the Intersection over Union (IoU) between this bounding box and the remaining bounding boxes are calculated. If $IoU > t_{IoU}$, with t_{IoU} being a second threshold, the candidate gets removed. This process is repeated until there are no candidates left.

While the efficient and straightforward trainable structure is advantageous for YOLO, there are significant drawbacks compared to alternative approaches. The fixed size of the output restricts the number of objects detected. Furthermore, each field can be the center of only one object, which is problematic on crowded images or small objects in groups.

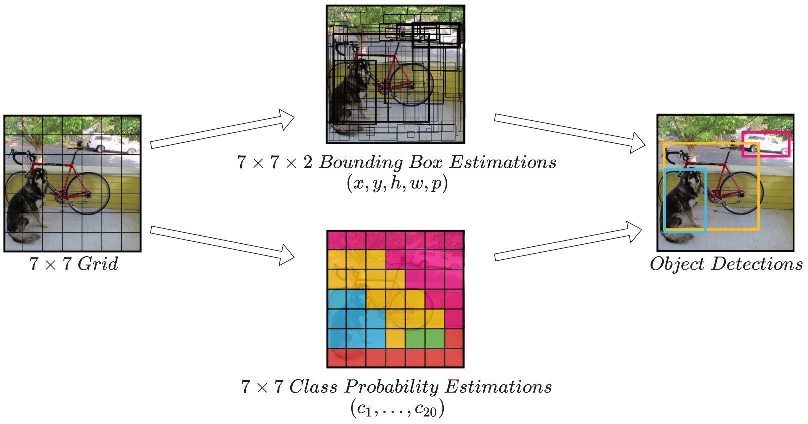

7×7 *Grid*
$7 \times 7 \times 2$ *Bounding Box Estimations*
(x, y, h, w, p)
7×7 *Class Probability Estimations*
$(c_1, \ldots, c_{20})$
Object Detections

Figure 2.12: Schematic representation of the YOLO framework [85] using a 7×7 grid, $B = 2$ bounding box proposals per field, and $n = 20$ classes. Hence, the network's output for this grid has the dimension $7 \times 7 \times (B * 5 + n) = 7 \times 7 \times 30$.

Single shot detector (SSD)

The SSD framework [86] is based on similar principles as the YOLO framework. An SSD object detector consists of a single network that produces a fixed number of bounding box proposals. In contrast to YOLO, the network contains more layers, making it more precise while increasing computing time. Using a convolutional structure, similar to the one classification networks have, the SSD network generates feature maps of different sizes. These feature maps encode information on objects centered in the input image's respective area. For each field of these feature maps, the SSD network predicts the presence of an object that matches into a default bounding box. These default boxes have different aspect ratios to test different object shapes. For each default box and each field, the network predicts relative shape and position offsets, as well as class confidences that indicate the chance of an object being present. Hence, bounding box estimation and classification are done by the same network on the same feature map. Fig. 2.13 shows a schematic representation of the SSD framework with two feature maps of size 8×8 and 4×4. For each field, the presence of four bounding boxes with different sizes is estimated. This allows the detection of objects of different sizes and shapes. Like the YOLO framework, the SSD framework utilizes non-max suppression to extract the final output from the whole set of bounding boxes.

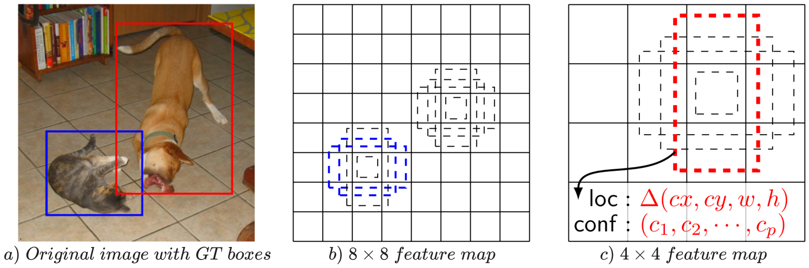

a) Original image with GT boxes
b) 8×8 feature map
c) 4×4 feature map
loc : $\Delta(cx, cy, w, h)$
conf : $(c_1, c_2, \cdots, c_p)$

Figure 2.13: Schematic representation of the SSD framework [86]. The network transforms the image to feature maps of different sizes (Here 8×8 *and* 4×4). For each field of each feature map, the network performs a prediction for n default bounding boxes. For each bounding box, there are predicted offsets and confidences for p different object classes $(c_1, ..., c_p)$.

Region-based convolutional neural networks (R-CNN)

Region-based object detection frameworks have proven to achieve high accuracy, essential for safety-critical applications, making them the dominant methods for automated vehicles. The basic concept of R-CNN was introduced in 2014 by [87]. In contrast to SSD and YOLO, R-CNN methods do not estimate a fixed number of bounding boxes for default regions but are based on a three-staged approach. The first stage applies a region proposal algorithm to the image to get rectangular object proposals. The proposed regions are cut out on the second stage, warped to a standard size, and processed by a neural network to be transformed into a feature representation. The original framework uses a *Support Vector Machine*-classifier to assign the feature representations to a class. A schematic representation of this process is shown in Fig. 2.14. Like the other frameworks, the extraction of bounding boxes of all proposals is done by non-max suppression. While SSD and YOLO are based on single networks that can be trained end-to-end, the original R-CNN framework is modular to substitute the components. Hence, the third stage is often performed by classification networks directly incorporated into the second stage. However, in the original R-CNN concept, each region is individually processed by the second and third stages, making object detection time-consuming and computationally expensive. To improve speed and reduce computational effort, *fast R-CNN* [88] and *faster R-CNN* [74] have been introduced. These approaches incorporate YOLO and SSD's strengths into the R-CNN framework, such as a unified end-to-end learnable structure and a shared feature map that enables a fast computation while still being based on multi-stage processes that increase accuracy and reliability.

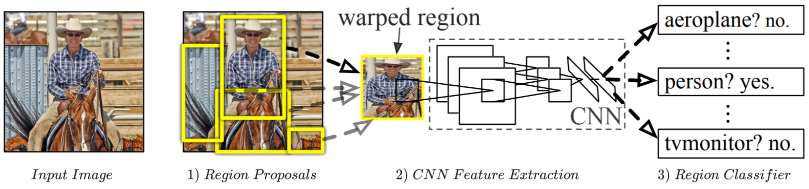


Figure 2.14: Multi-stage framework for R-CNN [87]. The first stage extracts proposals for regions of interest (RoI). These regions are then transformed to a standardized feature representation and assigned to an object class.

Performance evaluation using mean average precision (mAP)

To conclude the principles of deep learning methods for object detection, a performance measure must be established. As mentioned above, an object detector returns a set of bounding boxes proposals, called detections with attached confidences and predicted classes, since multi-class object detection is a hybrid between object classification and object localization. The number of proposals depends on the minimum confidence value threshold. The lower this threshold is chosen, the more proposals are returned, increasing the chance of an object being detected while also increasing the risk of false detections. An appropriate evaluation technique has to consider both localization and classification. One of the most common measures for this matter is the mean average precision (mAP) measure that has been established by the PASCAL Visual Objects Classes (VOC) challenge [89]. Since the average precision is a measure for binary classifiers, the n-class object detector results are split in n different binary results - one for each object class. The mAP measure is then defined by

$$mAP = \frac{1}{n}\sum_{i=1}^{n} average\ precision_{class_i}. \tag{2.9}$$

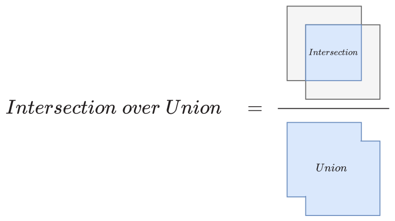


Figure 2.15: Schematic representation of the intersection over union (IoU).

Calculating the average precision requires a scheme to determine whether the detections are true or false. The geometric intersection over union (IoU) measure is used to evaluate a bounding box prediction's correctness. The calculation of an IoU is straightforward, given a bounding box prediction and a corresponding ground truth bounding box. The intersection is defined by the overlap between the boxes, while the bounding boxes' total area defines the

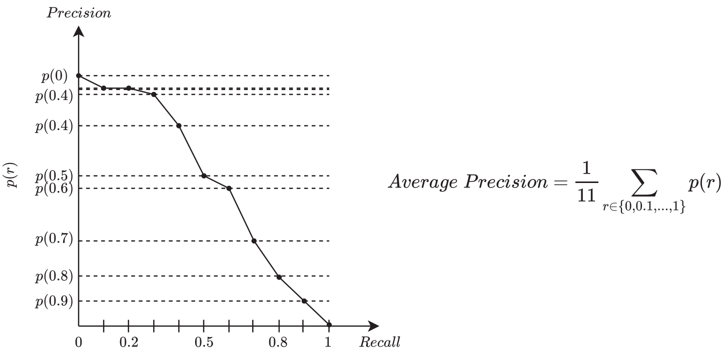


Figure 2.16: Receiver operating characteristic (ROC) curve for a given class. The curve is build by calculating recall and precision for different confidence levels for object detections. For the average precision, the confidence values are chosen such that the recall takes on the values $\{0, 0.1, \ldots, 1\}$. The corresponding precision values are averaged to get the AP value.

union. Fig. 2.15 displays the relationship between these two areas. For object detection, the IoU determines whether a predicted bounding box is correct or incorrect. A detection is called a *true positive (TP)* if there is a ground truth bounding box of the same class with $IoU > t$, where t is a threshold for minimum IoU (Usually 0.5). Otherwise, the detection is called a *false positive (FP)*. Furthermore, ground truth bounding boxes that are missed detections, i.g., there is no detection of the same class with $IoU > t$, are called *false negatives (FN)*.

Precision and recall are then defined by

$$Precision = \frac{TP}{TP + FP}, \qquad Recall = \frac{TP}{TP + FN}. \tag{2.10}$$

The average precision can be derived from the fact that an increase of the confidence threshold is likely to decrease the number of detections changing the relationship between precision and recall. Fig. 2.16 shows the so-called receiver operating characteristic (ROC) curve that can be created by calculating precision and recall for a range of different confidence thresholds. The average precision is determined by extracting a set of different confidence thresholds, such that the recalls with those thresholds are equal or near a set of desired recall values. [89] proposes the recall values $\{0, 0.1, ..., 1\}$. The AP is calculated by averaging the respective precisions. Finally, to retrieve the mAP, the APs for all classes are averaged.

3 Design and testing of deep neural network-based perception functions

3.1 Basic terminologies

Before release, software has to undergo an intensive testing process to ensure its quality. Depending on their objective and the requirements, tests can be differentiated in two classes. *Functional testing* focusses on the functional requirements set for the software and describes *what the software should do*. These requirements include correctness, accuracy, and robustness. *Non-functional testing* focuses on requirements that are not directly tied to the functional behavior of the software and describe *how the software should do it*. These requirements include efficiency, fairness, and interpretability [55].

Hence, functional testing aims to detect states where the tested software's behavior does not comply with the functional requirements. The IEEE Standard Glossary of Software Engineering Terminology defines these states as failures [52]. Uncovering a significant amount of failures requires the generation of an extensive data set to identify data patterns that increase the probability of a failure to occur.

Black-/White-/Gray- box testing

Testing techniques for neuronal networks can be categorized in black-, white-, and gray-box testing, similar to testing of traditional software [90].

White-Box Testing: In contrast to black-box testing, white-box testing techniques focus on the network's inner structure. Usually, this includes the network's architecture, the weights between neurons, and neuron activation patterns if data flows through the network. The advantages and disadvantages of white-box testing are inverse to black-box testing.

For example, the model-under-test cannot be exchanged offhand for an existing testing pipeline since some parts of the testing process may be set up for an explicit architecture. On the other hand, the analysis of the internal processes during training can identify and interpret possible failure causes on the feature level of a DNN.

Black-Box Testing: Black-box testing is applied if there are no assumptions made on the inner workings of the DNN. Instead, the DNN's response to a given input signal is evaluated. The main advantage of black-box testing techniques is the independence of the model-under-test. As long as the input and output data formats are equal, an existing testing process can be used for all perception functions, even if their functionality differs. On the other hand, ignoring information on the network's architecture and neuron patterns limits debugging options and explaining false predictions. Hence, black-box testing techniques may be too superficial in some cases. Another application of black-box testing techniques is the use of pre-defined meta-models trained to imitate the behavior of the black-box model. Applying white-box methods to these meta-models allows utilizing the advantages of white-box without the drawbacks. However, the meta-model only approximates the real model-under-test, and accordingly, the drawn conclusions should be treated with caution.

Gray-Box Testing: The term gray-box testing is used to test strategies that combine white-box and black-box testing characteristics. This includes strategies in which the knowledge of the internal processes is limited. Limited knowledge could be that only the architecture of the network may be known without the values of the individual weights. Sometimes gray-box testing is used for testing strategies in which the distribution of the training data is available.

Definition of image domains and the domain gap

The term *domain* is another essential and recurring theme in deep learning research, though sometimes utilized partly contradicting. The term is used in different contexts to portray the relationship between data points. A domain describes a group that contains all data points systematically related by a certain data characteristic. There is no explicit requirement or specification for the data characteristic that defines the domain. The most intuitive example of data domains is the algebraic sign in the space of real numbers. Consider $x \in \mathbb{R}$: We consider x in the negative domain if $x < 0$ and in the positive domain if $x \geq 0$. This can be transferred to the use of any arbitrary interval in the space $\mathbb{R}^n,\ n \in \mathbb{N}$.

In computer vision, a domain describes semantic characteristics of an image's appearance rather than plain numerical intervals. The most common uses are detailed below.

Scenario Level: Domains on scenario level refer to the content of an image: position and appearance of objects, weather, and daytime, or street geometry. The focus is on *what scenario is displayed*? A straight forward example is the categorization of images into the daytime domains *day, dawn, dusk,* and *night* [91].

Signal Level: For domains on signal level, the focus is on *how is the scenario displayed*? An engaging (and highly relevant for this dissertation) example of signal-level domains is the distinction between real and virtual images. The virtual KITTI data set is a set of virtual images with the scenarios configured in the same way as the original KITTI images. Although a virtual image would belong to the same scenario domain as its real counterpart, there is a discrepancy between virtual and real images. Other examples are sensor position and perspective on the vehicle or different sensor technology.

3.2 Development process for object detection

The development and evaluation of perception functions usually follow a simple principle: Via vehicles, sensor data is recorded in real traffic. After retrieving the ground truth information by labeling, a complete data set for model development is available. Since the recording was done in real traffic, the distribution of the data set should be equal to the data distribution during the final deployment. Hence, if the model performs well on the development data set, the model is assumed to perform well in deployment. This development data usually is split into three separate data sets: The train set and validation set for model design and the test set for model evaluation [92]. This process is shown in Fig. 3.1. The function of each split is detailed as follows:

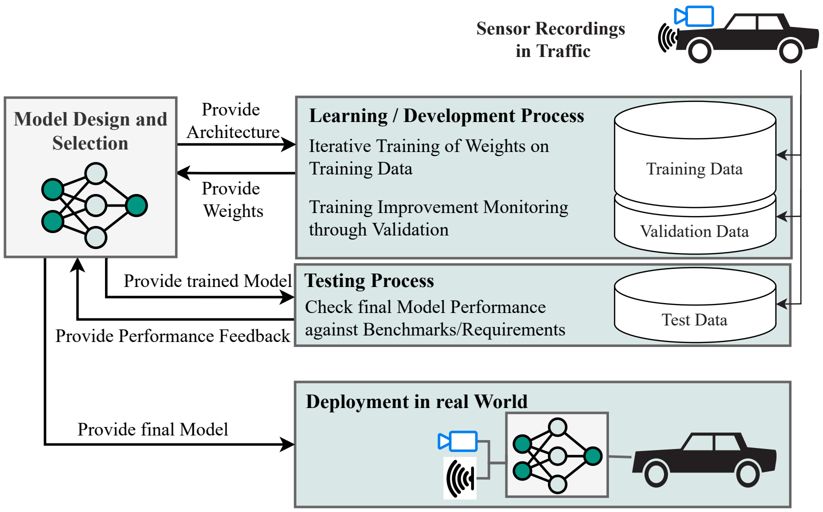

Sensor Recordings in Traffic
Model Design and Selection
Provide Architecture
Provide Weights
Learning / Development Process
Iterative Training of Weights on Training Data
Training Improvement Monitoring through Validation
Training Data
Validation Data
Provide trained Model
Provide Performance Feedback
Testing Process
Check final Model Performance against Benchmarks/Requirements
Test Data
Provide final Model
Deployment in real World

Figure 3.1: Best practice for collecting development data for perception functions: In real traffic, vehicles equipped with perception sensors record data that gets labeled to obtain ground truth information on the actual locations of objects in the sensor data. This data set gets partitioned into three separate data sets: Train set for model training, validation set for model selection, and a test set for model evaluation.

Train Set: The development process of machine learning algorithms starts with a training data set. In contrast to classical software solutions, machine learning methods are so-called data-based methods. In contrast to rule-based methods, the magic of machine learning algorithms lies in the training data. The models are trained to learn the patterns in the data that is provided in the training set. The learned model is adjusted to behave in the same way the underlying training data behaves.

Validation Set: Most machine learning methods inherit a variable architecture, such as a variable number of parameters that can be used. This causes a high number of model candidates for final deployment. Choosing the model that performs best on the training data could lead to overfitting, meaning that the model loses its ability to generalize to fit the training data well. On the other hand, choosing the model with the best performance on the validation set ensures a high accuracy on *unseen* data that has not been used during training, thus preventing overfitting.

Test Set: Holdout data set for model evaluation. While training set and validation set are used for model design, the test set is used for evaluating the final performance of the chosen machine learning model. This data set is completely excluded from the design process and often serves as benchmark.

The practice of using training-validation-test splits has been made common practice and is implemented on most major data sets for perception functions such as MS-Coco [93], BDD100k [43], or Cityscapes [94].

However, there are two assumptions on the data set properties that have to be made for the data split [82]:

identically distributed: The three data sets are drawn from the same data distribution, relating to sensor type, weather conditions, and objects. If the validation set follows a different distribution than the train set, the model development process could select a non-ideal model. If the test set follows a different distribution than the train and validation set, the model could perform poorly in the deployment domain.

independent: The three data sets are independent, and there should be adequate separation between the data sets. However, poor separation could arise from dividing individual recordings into the three data sets or recording the data sets at the same locations. In addition, dependence between the train and validation set and the test set could bias the performance evaluation of the final model.

A reliable behavior within the complete deployment domain must be guaranteed to apply machine learning algorithms in safety-critical applications. For example, if parts of the input space are not covered by the development data set, this could cause catastrophic consequences. Fig. 3.2 shows the importance of input space coverage when developing machine learning algorithms. Based on the available development data, a machine learning model is developed to imitate the behavior of the underlying function. For domains with a dense availability of data points, the model will perform well. However, suppose there are domains in the input space with a low density of available data points. In that case, the estimations of the fitted models for these domains become unreliable, as the extrapolation becomes uncertain, and the underlying function may behave significantly different than the extrapolation. For perception

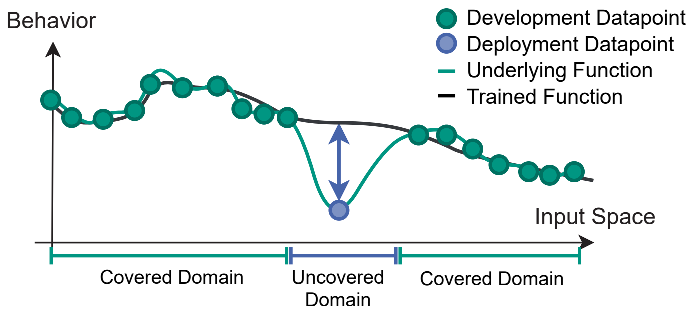


Figure 3.2: Visualization of the importance of domain coverage. The model is trained to imitate the underlying behavior of the development data. If there are domains in the input space that are not covered by available development data points, accurate predictions can not be guaranteed.

functions, situations that are challenging are relatively rare in real traffic, such as adverse weather conditions [45, 64].

3.3 Safety concerns regarding the use of DNN in automated driving tasks

Although DNNs achieve human-like performances in computer vision tasks, there are serious concerns regarding their reliability when applied in safety-critical automated driving tasks. A comprehensive overview on these concerns was presented by Willers et al. [95] and is summed up and commented with regard to requirements of machine learning models in Sec. 2.3:

a) Data distribution is not a good approximation of real world: Assessing the reliability of a perception function in real traffic can only be done if the distribution of scenarios in the development data approximates the distribution of scenarios in the real world. This point is directly tied to the assumptions of identically and independent distribution of data sets (see Sec. 3.2). While accurate knowledge of the real world data distribution is crucial for model development, especially regarding the data coverage of this distribution, strictly assessing the reliability of a perception function by using the appearance probabilities of traffic scenarios could cause unfair models. Low detection rates on rare road users could be neglected (following the description of the requirement *fairness*).

b) Distributional shift over time: Changes in the distribution of scenarios in the real world can distort the reliability estimates of the perception function. Such shifts include the introduction of novel objects to traffic. Furthermore, there has to be a steady monitoring of the traffic environment to identify distributional changes and the introduction of novel traffic artifacts.

c) Dependence on labeling quality: The labeling quality of the development data is crucial for supervised machine learning algorithms. Training a DNN using faulty labels can lead to an inherited defect. This weakness also refers to the problem of missing or inadequate label classes.

d) Unknown behavior in rare critical situations: Critical traffic scenarios may occur in real traffic with a low probability. Detecting such scenarios in real traffic may require a vast data recording effort, and there is no insurance that these scenarios will be captured.

e) Inadequate separation of test and training data: While the training and test data sets should follow the same distribution, they should be uncorrelated. For example, using training data to test the network would lead to overestimating the networks' performance.

f) Incomprehensible behavior: A DNN's ability to automatically learn to detect the relevant image features for correct predictions also implies an inherent lack of interpretability. In contrast to rule-based functions, DNNs are hard to debug since the correlations between image features and the networks' decisions are difficult to analyze. Incomprehensive behavior is a challenging characteristic of DNNs when examining whether a model complies with *interpretability* requirements.

g) Brittleness of DNNs: A lack of robustness has been a downside of DNNs subject to a range of studies. Even minor permutations in the image, such as noise, image translations, or weather effects, can cause misclassification. Adversarial attacks utilize such permutations to provoke system failures, as shown in Fig. 2.4. This concern is directly tied to the requirement *robustness* towards unapparent permutations of an image.

h) Unreliable confidence information: A DNN should estimate confidence for each prediction to indicate when the automated vehicle should instead rely on other sensors. However, using the DNNs output for class probabilities as confidence value has been shown to cause overconfidence in the network [96].

i) Insufficient consideration of safety in metrics: Most common metrics to measure the quality of DNNs for object detection, such as the mAP (Eq. 2.9), are not directly based on safety factors. For example, a prediction is more safety-relevant for the driving function the nearer an object is and the faster the object moves.

Safety concerns a)-e) are directly related to the data quality and composition of development data. These issues are often expressed with the quote *"Garbage in - Garbage out"* to emphasize the central role data has in the development process of machine learning functions. Safety concerns f)-g) arise with the non-linear nature of DNNs. Because of the connected architecture, insignificant changes to the input can propagate through the network and change the prediction. In this regard the non-linear structure of DNNs, usually their greatest strength, ironically is one of their major weaknesses. Safety concerns h)-i) are tied to difficulties in estimating the reliability of DNNs in automated driving tasks and evaluating their impact on traffic safety.

3.4 Techniques for testing of perception functions

There are different testing techniques to enhance the quality and scope of the available test cases. In this section, the most relevant for current research are presented. Note that the test generation via simulative tools is presented in Sec. 3.4.3.

3.4.1 Benchmark testing

Benchmark testing refers to evaluating and comparing the performance of different perception function standardized datasets and metrics. Benchmark testing helps researchers and practitioners assess the effectiveness and efficiency of various perception functions in a comparable setting. Some of the most significant data sets are:

BDD100k [43] The BDD100k, or Berkeley Deep Drive, is a large collection of 100,000 crowd-sourced videos recorded using dash cams. Labeled frames with annotations for various perception tasks, such as object detection, drivable area estimation, and instance segmentation, are available for each video.

KITTI [38] The KITTI (Karlsruhe Institute of Technology and Toyota Technological Institute) is an established benchmark for perception functions in an automated driving setting. The data was recorded in Karlsruhe in 2012 with about 7,500 images for training and testing each. In contrast to the BDD100k, the KITTI data set offers a complete standardized and calibrated sensor stack for automated driving.

Cityscapes [97] The Cityscapes data set is another data set for the development and benchmarking of perception functions for automated driving. The recording was performed in 50 different cities with 5,000 high-quality annotated images. Furthermore, there are additional 20,000 images with coarse annotations.

Additionally to the data sets, for each CV task there exists a range of standard metrics. However, using perception functions in safety critical applications requires novel metrics that take safety in consideration instead of ignoring the relevance of objects to the traffic event [98, 99]. For example, while the mAP metric is one of the most used metrics in object detection tasks, it ignores the traffic context. Missing the detection of a pedestrian on the other side of the road is not as critical as missing the detection the jaywalker right in front of the vehicle.

3.4.2 Coverage-guided testing

During testing, the requirement of completeness has to be proven (see Sec. 2.3). In classical programming, this is done by checking that the assembled set of test cases triggers all branches of the software code to find branches that cause failures [90]. Although the functionality of neural networks differs from traditional software, the concept of code coverage can be applied to the net-like structure of neural networks. A neural network consists of connected neurons that process incoming information to generate an output signal (see Sec. 2.4 and Fig. 2.9). An activation function determines the output signal of each neuron. In the neurons' most basic form, this activation function is a step function that takes on the value 0 for negative input and 1 for positive input (see Fig. 2.10). Hence, if the input value is positive, the neuron is considered triggered for this function. With this concept, coverage criteria for neural networks have been created that measure the coverage a data set provides on the internal structure. Thus, coverage-guided testing techniques are prime examples of white-box testing. There are two levels of coverage metrics [66, 100–102]: *Neuron-level* and *layer-level*. Coverage metrics on neuron-level observe the activation behavior of neurons independently and individually. In contrast, layer-level coverage metrics are based on combinations or sequences of neuron activations in the DNN.

Neuron-level coverage

In contrast to the example above, which uses the step function as the activation function for neurons, modern DNNs utilize continuous activation functions such as the ReLU function or the sigmoid function (see Sec. A and Fig. .3). The following metrics are defined for the ReLU activation function with

$$\mathrm{ReLU}(x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{else,} \end{cases} \tag{3.1}$$

but can be also applied to most alternative activation functions [102].

For simplicity, suppose $\phi(\mathbf{x})$ returns the output value of an individual neuron $\phi(\cdot)$ if the **whole** network is fed with input $\mathbf{x}$. Hence, this definition deviates from Eq. 3.1, which regards $\mathbf{x}$ as the immediate input to the neuron inside the network.

Neuron Coverage Basic neuron coverage is measured similarly to the example above. A neuron $\phi(\cdot)$ is considered to be covered by the test set $\mathcal{S}$ if there is a test case $\mathbf{x} \in \mathcal{S}$ for which the neuron's activation function $\phi(\mathbf{x})$ outputs a value greater than zero with $\phi(\mathbf{x}) > 0$. The neuron coverage of a test set is then equal to $\frac{\text{\#activated neurons by test set}}{\text{\#total neurons}}$.

***k*-multisection Neuron Coverage** Achieving full neuron coverage has been shown to be easily achievable. Thus, a more sophisticated metric for neuron activation has been developed. For k-multisection neuron coverage, the activation value is not interpreted binary with *activated* and *not activated*, but is categorized in pre-defined categories. The total output range of each neuron is partitioned in k-sections, hence the name. A section $S = [a, b]$ of neuron $\phi(\cdot)$ is considered to be covered by the data set if there is a test case for which the neuron's activation function $\phi(x)$ outputs a value with $\phi(x) \in S$. The metric is then equal to $\frac{\text{\#activated sections of all neurons by test set}}{\text{\#sections of all neurons}}$.

Strong Neuron Activation Coverage In contrast to the other metrics, strong neuron activation coverage takes information about activation behavior during network training into account. The strong neuron activation coverage observes the highest output value for the whole training data set for each neuron. A test data set should then reproduce these values at least once. Suppose $\hat{\phi}$ being the maximum activation value of neuron $\phi(\cdot)$ during training. The neuron is then considered as covered by the test set if there is a test case for which the activation value matches $\hat{\phi}$ with $\phi(x) - \hat{\phi} \approx 0$. The neuron coverage of a test set is then equal to $\frac{\text{\#strongly activated neurons by test set}}{\text{\#total neurons}}$.

Layer-level coverage

For layer-level coverage, the activation behavior of all neurons is observed to evaluate combinations and sequences of neuron activations.

Top-k Neuron Coverage For top-k neuron coverage, a neuron $\phi(\cdot)$ in the network layer l is considered as covered by test set $\mathcal{S}$ if there is a test case $\mathbf{x} \in \mathcal{S}$ for which the activation value $\phi(\mathbf{x})$ is in the k highest activation values of all neurons inside layer l. Then, the coverage metric measures which fraction of neurons have been activated by the test set $\mathcal{S}$.

***t*-Way Combination sparse Coverage** The t-way combination sparse coverage considers the behavior of neurons binary: non-activated and activated. However, the metric measures how well the test set $\mathcal{S}$ covers the t-wise combinations of activations across all neurons. Assume the pair-wise case $t = 2$, then a pair of neurons $\phi_1(\cdot), \phi_2(\cdot)$ is covered by test set $\mathcal{S}$ if there is a test case $\mathbf{x} \in \mathcal{S}$ with $\phi_1(\mathbf{x}) > 0$ and $\phi_2(\mathbf{x}) > 0$. The concept is equivalent to the concept of coverage arrays in Sec. 4.4.1.

Test case generation and limits

The presented coverage metrics are mainly used to quantify the quality of an existing data set and estimate how much data is still needed to achieve necessary coverage. However, testing techniques have been introduced to expand an existing data set to optimize coverage metrics. *Coverage-guided fuzzing* is an existing technique of software testing that has been adapted for DNNs [67, 103]. In general, fuzzing techniques generate a high quantity of input data to provoke failures. Random changes are made to the inputs to create mutates based on available test cases. The initial data set is called *corpus*. If mutants increase the coverage criteria, they are added to the corpus and serve as a base for the next mutants. Multiple approaches center around this concept, such as *TensorFuzz* [67] or *DeepHunter* [104]. However, while coverage metrics contribute to a safety argumentation for using DNNs in automated driving, they also bear limitations. For example, Sun et. al [66] showed that full neuron coverage of an MNIST classifier could be already achieved by only choosing a few samples from the test set randomly. Furthermore, full coverage does not guarantee the detection of all possible failure causes of a DNN [105, 106]. The most intuitive weakness of coverage-guided testing is that it does not imply completeness of the input space from a semantic perspective. Complete coverage of a perception function does not guarantee that all possible scenarios have been tested and that the whole input domain is covered.

3.4.3 Adversarial testing

Basic concept of adversarial testing

Adversarial testing or *adversarial attacks* of DNNs has seen significant research over the past years. The traditional definition of adversarial testing centers around the safety concern *brittleness* of DNNs [65, 107]. Through the connective structure, minor changes to the input values can propagate through the network to cause significant changes in the output value. Adversarial attacks aim to utilize this weakness to cause failures and test for local robustness.

There is a multitude of adversarial testing [101]. In short, the adversarial attack is performed by a function

$$\phi : \mathcal{X} \to \mathcal{X} \quad x \mapsto \hat{x} \tag{3.2}$$

that transforms the inputs to a DNN

$$f : \mathcal{X} \to \mathcal{Y}. \tag{3.3}$$

$\phi(\cdot)$ shall not change the semantic characteristic of the input, ideally being unidentifiable by a human observer. The objective of $\phi(\cdot)$ is to minimize performance of $f(\cdot)$, causing a falsification of a previously correct prediction. Thus, the data generation process is transformed into an optimization problem that permutes existing testing data. A data point $\phi(x) \in \mathcal{X}$ is called *adversarial example* if and only if the DNNs prediction for $\phi(x)$ is incorrect while the prediction for $x \in \mathcal{X}$ is correct.

Szegedy et. al [65] observed that adversarial examples for a network architecture $f_1(\cdot)$ are likely to be also adversarial examples for a network architecture $f_2(\cdot)$ if $f_1(\cdot)$ and $f_2(\cdot)$ are trained on subsets of the same data set. This indicates a certain transferability of $\phi(\cdot)$, which is development for $f_1(\cdot)$ to create adversarial examples for $f_2(\cdot)$.

Adversarial attacks have been shown to create severe difficulties for perception functions, reducing the mAP score of state-of-the-art object detectors such as Faster R-CNN and YOlO bellow 1% [108].

There are two different objectives for adversarial attacks [109]: The goal of an untargeted attack is to misguide a DNN to force a false prediction on the image. The attack does not provoke a particular misdetection but tries to fool the DNN. In contrast to untargeted attacks, a targeted attack tries to fool the DNN to make a particular error. For example, the attack may be explicitly designed to classify an image as *horse*.

Test case generation

There are two main branches of attacks, depending on how an image is altered to provoke a DNN failure [110]:

Full-Image Attack Full-image attacks, or global attacks, are enabled to alter the total image. That does not imply that every pixel value of the image has to be altered but enables the transformation function ϕ to change all pixel values. The addition of noise is a typical example of a global attack. Usually, the attack should not be visually recognizable by a human observer. An example of a full-image attack is displayed in Fig. 3.3.

Patch-Level Attack Patch-level attacks only change a limited image area by adding a local pattern [111]. In contrast to full-image attacks, the changing area is small but recognizable by a human observer. This kind of attack poses an explicit risk to the DNNs in automated driving, as patch-level attacks have shown to be applicable in the real world [112]. Here, the patch is not added by post-processing an image but by placing a print of the patch in front of the camera sensor.

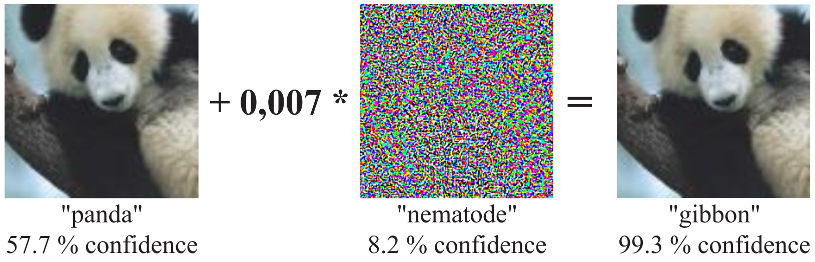
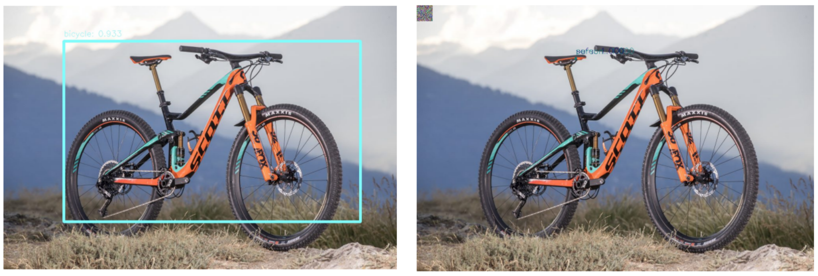


Figure 3.3: Example of full-image attack on GoogLeNet [113] on ImageNet [84] data. A mask in the size of the original image with small values is added to the image to force a misclassification. Note that the change is almost unrecognizable to a human observer. Image and experiment by Goodfellow et. al [107]



(a) No DPATCH (b) Witch DPATCH

Figure 3.4: Example of the patch-level DPATCH [108] attack on YOLO object detector [85] in the upper left corner. While the area of the patch-level attack is clearly visible for a human observer, the attack is restricted on a small area on the upper left corner.

4 Adaptive test case sampling for DNN-based perception functions

4.1 Simulative testing - Advantages and weaknesses

Synthetic image data to complement real world testing

The established process of testing perception function that solely rely on real data may have significant flaws. The development data set itself has to follow the distribution of the real world traffic and cover the whole input space [95, 115]. However, to appropriately represent real traffic, an immense amount of data has to be collected, which is a point that has already been raised in Sec. 1.3 [26]. Although there is a de facto endless number of traffic scenarios, data recordings in real traffic will inevitably get more and more repetitive [1, 114] with an increasing number of driven kilometers, as displayed in Fig. 4.1.

In the recent past, the use of simulative tools to complement real world data has been proposed, as studies showed evidence regarding the applicability of synthetic image data in the development process of perception functions [116–118]. The virtual KITTI data set [119] demonstrates how synthetic data simulation tools can substitute and complement the existing real world data. The data set was created as virtual reproduction of the real KITTI data set [38]. Fig. 4.2 displays some examples of this data set. Ground truth information is then generated cheaply by the simulative tool. Hence, underrepresented effects in real world data sets can be stuffed using synthetic data. For example, cyclists only account for 4% of annotations in the KITTI data set with 1,627 annotations in 7,400 images. The *synthetic data set for improving Cyclist Detection* from *Parallel Domain* [120] enhances the available real KITTI data set with synthetic images that contain annotated cyclists to improve the detection performance of a YOLOv3 model, especially on the class cyclist [121].

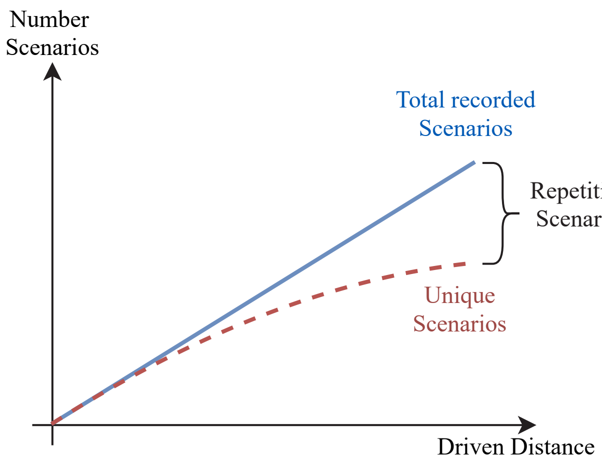


Figure 4.1: With an increasing driven distance in real traffic, the recording of real world data starts to collect more and already recorded traffic scenarios as the data recordings get more and more repetitive [1, 114].

Thus, the variety of virtual data sets, simulation tools, and scientific research has increased over the past years. For example, in the previous work on adaptive test case selection, the commercial tool CarMaker from IPG Automotive [122] was used for image generation [2]. Other approaches for image sensor simulation come from unexpected but consequential sources. Richter et. al [123] used the computer game Grand Theft Auto 5 to generate traffic image data.

Limits and weaknesses of simulative testing

While simulative testing of perception functions may seem like the perfect and flawless substitute for real world testing, some weaknesses must be addressed. Currently, the most significant and most urgent limitation of simulative testing of perception functions is the *domain gap*. In Sec. 3.1 image domains are defined, and the domain of real images is set against the domain of synthetic images. Experiments show that the performance of a model trained with real data may differ when using test data from the virtual domain and vice versa [118]. The transferability of results on synthetic data in the real domain

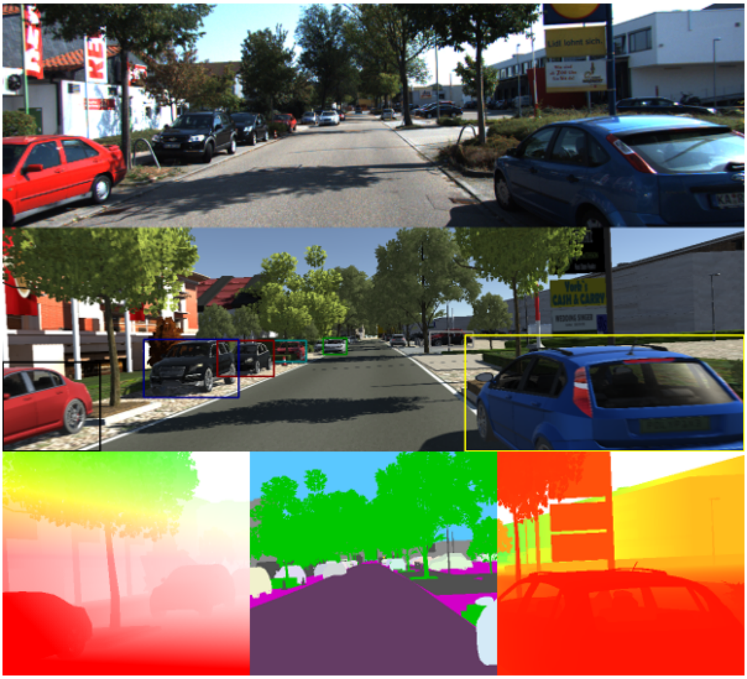


Figure 4.2: Synthetic image data from simulation tools can create imitations of real world traffic situations to substitute and complement real world data. This particular image is from the virtual KITTI data set [119] that was created as a virtual reproduction of parts of the real KITTI data set [38].

remains a question mark. However, this transferability is critical when creating a safety argument for perception functions: In the end, a perception function has to work in real traffic. Although this sounds like a deal-breaker for simulative testing, several research areas address this issue:

Physical Sensor Simulation: The most straightforward approach is increasing sensor simulation quality. Physical sensor simulation refers to sensor simulation in which the generated sensor signals correspond to the signals of a real sensor in a way that allows it to be used interchangeably with real sensor signals. The area includes tools to generate geometrically correct image data [124] or the addition of camera artifacts, such as lens flares [125] and light reflection [126].

Transfer Learning: In transfer learning, the objective is to transfer knowledge obtained from training one task to be used for another task. For simulative data, transfer learning implies using insights into a model's performance in the virtual domain on applying the function in the real domain [127].

I2I Domain Translation: Another approach is image-to-image (I2I) translation between different domains. I2I translation transforms a given image from a source domain to a target domain by changing its appearance. An example of this could be a translation from the *night* domain to *day* domain [91], or from *no-fog* domain to *fog* domain [128]. In the presented context, the virtual domain is considered the source domain, and the real domain is the target domain. Previous research for Sim-to-Real image translation includes transformation approaches such as fully convolutional networks [129] or generative approaches such as generative adversarial networks (GAN) [130].

Another weakness of simulative testing is the problem of *unknown-unknowns*. A simulation framework can only generate traffic elements that are known and coded into the framework. If, for instance, there are weather effects not accounted for in the simulation, such as raindrops at the lens, this may lead to unreliable test results of the perception function. The same applies to objects that may not be available in the simulation, such as wild animals.

These weaknesses highlight the fact that simulative testing, although being a convenient complement, cannot substitute testing with real world data.

Addressing the course of dimensionality

In contrast to the random nature of real world recordings, simulative tools can generate the data "on-demand" by feeding the specification of the desired image data to the tool [2,69]. The specifications include weather information, object properties, and positional arguments. Fig. 4.3 shows a schematic representation of the process of generating synthetic image data using the simulation tool CARLA [131]. In this example, the image space is parameterized by a

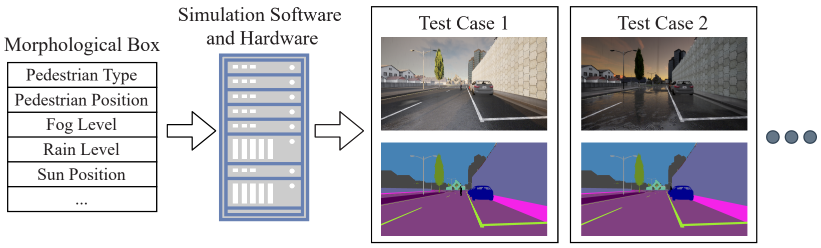


Figure 4.3: Simulative process for synthetic image generation. The simulation space is defined by a combinatorial space containing the available simulation variables. Feeding the simulation software and hardware with elements of the simulation space, image data, and corresponding ground truth labels is generated.

morphological box, also called Zwicky box. The morphological box contains all options for each image parameter. The space of possible images then arises from the Cartesian product of parameters.

The simulation process will inevitably run into the *curse of dimensionality* [132]. In machine learning, this expression describes specific issues when analyzing high-dimensional spaces. For example, there is a rapid volume increase when adding dimensions to a discrete or continuous space. See Fig. 4.3 as an example of a simulative process for a jaywalker scenario with five discrete simulation variables. Assuming each variable has ten characteristics, the total simulation space contains 10^5 elements. If one more variable with ten characteristics is added to this simulation space, the amount of elements increases to 10^6. In general, the amount of elements in a simulation space with n variables, with variable i having λ_i characteristics, is equal to

$$\text{no. elements} = \sum_{i=1}^{n} \lambda_i . \tag{4.1}$$

This formula applies only to discrete variables. In the context of software testing, this phenomenon is referred to as *combinatorial explosion*. Due to the high number of variables required to accurately describe image data, testing a significant fraction of the simulation space becomes unfeasible [68]. Relying on a random sampling of elements is unlikely to uncover failures, as critical concrete scenarios are rare.

4.2 Introducing pipeline for adaptive scenario selection for simulative testing

Adaptive test strategies

Adaptive test strategies have been proposed [133] to identify failures and critical scenarios [134–136] more efficiently. Unlike pre-configured experimental designs, adaptive strategies adjust the selection of test cases depending on already observed test cases. The relationship between the specifications of a test case and the perception function's performance is following patterns that can be exploited is assumed. The test result is transformed into a numerical optimization problem. Hence, this optimization problem can be incorporated into a sampling process that iteratively selects and observes the perception functions behavior. This process is displayed in Fig. 4.4. Finally, the testing gets converted into a minimization problem in which the performance of the system-under-test gives the objective function.

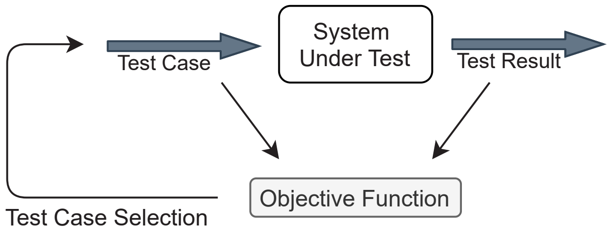


Figure 4.4: Schematic representation of functional testing to detect system states where the system-under-test fails to comply with the functional requirements. The behavior in concrete test cases is observed. New test cases are selected depending on an objective function derived from the system's performance [135].

Scenario-based testing

While coverage-guided testing of neural networks for perception tasks usually refers exclusively to neuron- and layer-level coverage, the application in automated driving enables for an alternative view on the term coverage (s. Sec. 2.3). Scenario-based testing focusses on covering the input space rather than covering all branches and nodes of the network. The set of test cases

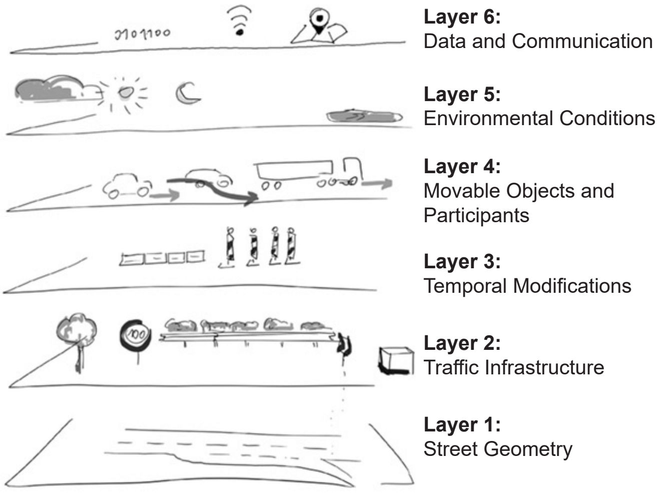


Figure 4.5: The six layers used to describe a traffic scenario as defined by the PEGASUS project [6].

should be able to sufficiently cover the traffic scenarios that can occur in real traffic [29, 137, 138].

There has been research to standardize and structure the representation of a driving scenario. The public-funded PEGASUS project [6] lays the foundation of scenario-based testing and defines five different layers for a standardized description of traffic scenarios for simulative testing. The five layers are displayed in Fig. 4.5 and include

1. **Road-Level:** Geometry and topology of traffic scenery.
2. **Traffic Infrastructure:** Building development and traffic infrastructure of traffic scenery.
3. **Temporary Manipulation of 1. and 2.:** Construction sites and temporary changes in traffic regulations.
4. **Objects:** Appearance and behavior of traffic participants, such as trajectories, size, and object textures.

5. Environment: Environmental effects on visibility and road conditions, such as weather and daytime.

The original 5-layer model has been extended to a 6-layer model [139]. The sixth layer includes digital information, such as digital map data or V2X communication information.

Furthermore, the PEGASUS project builds on a scenario logic, dividing scenarios in three categories: functional scenarios, logical scenarios, and concrete scenarios with the definitions being displayed in Fig. 4.7.

A process visualization is shown in Fig. 4.6 [5]. The process is simultaneously data-driven and expert-driven, with a central database containing the relevant logical scenarios and their corresponding parameter space. Using a testing platform and performance metrics, the SuT is evaluated on this parameter space via a test case variation method to generate an evaluation of the system's performance.

Formal definition of pipeline

The testing process of a deep neural network for object detection using simulated images relies on a simulation process that generates image data from formal specifications test cases. Hence, the simulation process requires a standardized scenario description format that specifies the image to be generated. The description format is a test space derived from pre-defined scenario variables. These variables relate to the environment (weather or sun position), the scene (road geometry and background), or objects (attributes of pedestrians or cyclists).

The following definitions and their relationship are visualized in Fig. 4.8.

Definition 4.1 (Functional scenario). *A functional scenario $\mathcal{F}$ is a linguistic description of a driving situation from an ego perspective, including the relevant information on the traffic participants, their intentions, the road geometry, and environmental conditions. $\mathfrak{F} = \{\mathcal{F}_i\}_{i \in \mathbb{N}}$ then describes the space of functional scenarios.*

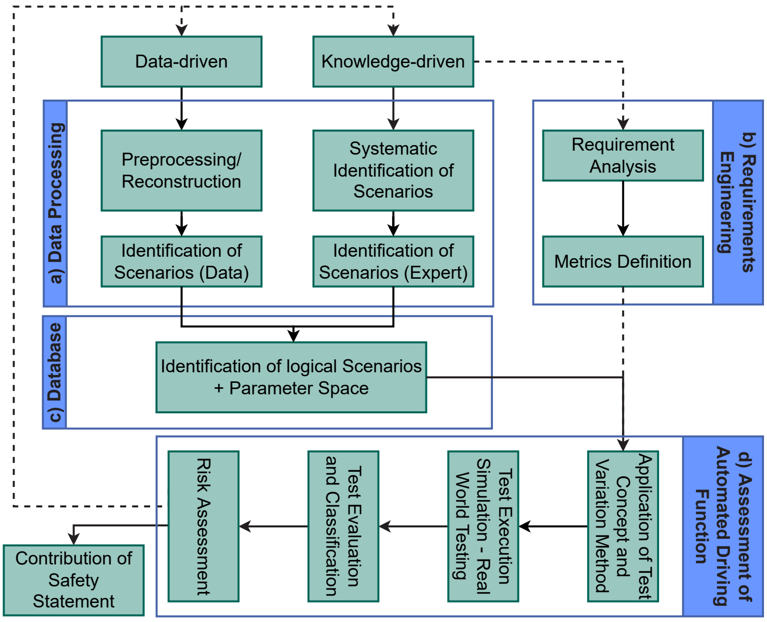
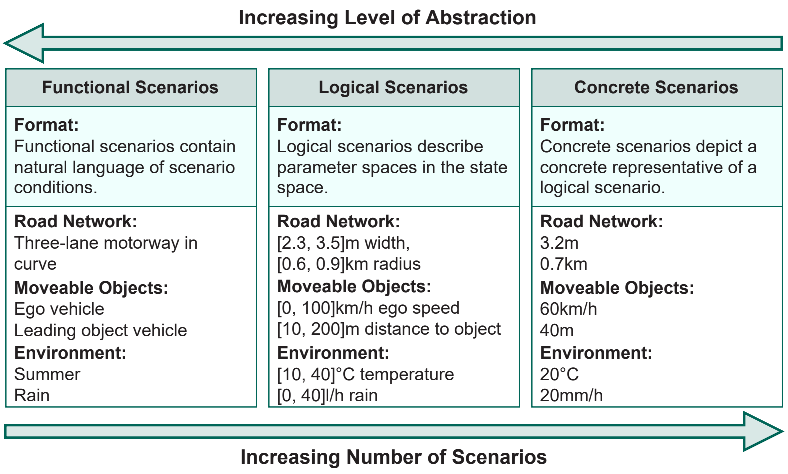

Data-driven
Knowledge-driven
a) Data Processing
Preprocessing/ Reconstruction
Systematic Identification of Scenarios
Identification of Scenarios (Data)
Identification of Scenarios (Expert)
b) Requirements Engineering
Requirement Analysis
Metrics Definition
c) Database
Identification of logical Scenarios + Parameter Space
d) Assessment of Automated Driving Function
Application of Test Concept and Variation Method
Test Execution Simulation - Real World Testing
Test Evaluation and Classification
Risk Assessment
Contribution of Safety Statement

Figure 4.6: Overview of scenario-based testing process [5].

Since functional scenarios are defined by linguistic descriptions and categorical numerical values (speed limit $\in$ 10, 30, 50, 70, 80, 100, 130 with each having finite options, $\mathfrak{F}$ is assumed to be discrete and finite itself.

Example 4.1 (Functional scenario)**.**

$$\mathcal{F} = \begin{array}{l} \textit{Straight road with two lanes} \\ \textit{Bus stop in urban environment, speed limit at 50km/h} \\ \textit{No temporary influences} \\ \textit{Ego passing a parking bus, pedestrian crossing the street} \\ \textit{Dawn, slight rain} \end{array}$$


Increasing Level of Abstraction
Increasing Number of Scenarios

Functional Scenarios	Logical Scenarios	Concrete Scenarios
Format: Functional scenarios contain natural language of scenario conditions.	**Format:** Logical scenarios describe parameter spaces in the state space.	**Format:** Concrete scenarios depict a concrete representative of a logical scenario.
Road Network: Three-lane motorway in curve	**Road Network:** [2.3, 3.5]m width, [0.6, 0.9]km radius	**Road Network:** 3.2m 0.7km
Moveable Objects: Ego vehicle Leading object vehicle	**Moveable Objects:** [0, 100]km/h ego speed [10, 200]m distance to object	**Moveable Objects:** 60km/h 40m
Environment: Summer Rain	**Environment:** [10, 40]°C temperature [0, 40]l/h rain	**Environment:** 20°C 20mm/h

Figure 4.7: Scenario description scheme from PEGASUS project [6]. Scenarios can be described on three levels of abstraction: Functional scenarios, logical scenarios and concrete scenarios.

Definition 4.2 (Scenario variable and logical scenario)**.** *A scenario variable* $S_i^j,\ j = \{1, \ldots, n_i\}$ *is a set of semantic values that specifies a semantic feature of the functional scenario* $\mathcal{F}_i$*, with* n_i *being the number of scenario variables for* $\mathcal{F}_i$*. The scenario variable can be continuous or discrete.*

Given a functional scenario $\mathcal{F}_i$*, the corresponding logical scenario space, or logical scenario, is defined as the Cartesian product over all relevant scenario variables* $\mathcal{S}_i = S_i^1 \times \ldots \times S_i^{n_i}$.

Definition 4.3 (Interpreter)**.** *With* $\mathfrak{F}$ *being the space of functional scenarios and* $\mathfrak{S}$ *being the space of logical scenarios, an interpreter function*

$$I : \mathfrak{F} \rightarrow \mathfrak{S}, \tag{4.2}$$

transforms the information of a functional scenario into the corresponding logical scenario.

Note that the amount and nature of scenario variables S_i^j is specific to the functional scenario $\mathcal{F}_i$, as a different constellation of traffic participants may result in different variables. The crucial core of Def. 4.3 is that there is a logical scenario space for each functional scenario. In this regard, the proposed approach differs from the traditional PEGASUS schema shown in Fig. 4.7. The original approach assumes that functional scenarios are different levels of abstraction with multiple logical scenarios for a given functional scenario. The approach of this work considers a logical scenario and the space of scenario variables of a functional scenario. While two functional scenarios may share the same logical scenario space, each functional scenario has one and only one logical scenario space.

Example 4.2 (Scenario Variable and Logical Scenario Space)**.** *An example of a discrete variable that relates to an image feature is the weather variable* $S^{weather} = \{\text{dry}, \text{rain}, \text{cloudy}\}$*, while an example for a continuous variable is the speed range of an object* $S^{speed} = [0kmh, 10kmh]$.

The logical scenario space would then consist of the Cartesian product $\mathcal{S} = S^{weather} \times S^{speed}$.

Definition 4.4 (Concrete scenario (Test case))**.** *Given a functional scenario* $\mathcal{F}$ *with logical scenario space* $\mathcal{S}$*, an element* $s \in \mathcal{S}$ *is called a concrete scenario or a test case in the context with the simulation framework. Hence a concrete scenario is a vector containing a concrete value from each scenario variable* $S^j,\ j \in \{1, ..., n\}$ *with n being the number of scenario variables of* $\mathcal{F}$.

Example 4.3 (Concrete scenario (Test case))**.** *Given* $\mathcal{S} = S^{weather} \times S^{speed}$*,* $s = (\text{dry}, 2kmh)$ *is a concrete scenario.*

Definition 4.5 (Ground truth function, image space, and ground truth space)**.** *A ground truth function for a perception system is defined as a function*

$$T : \mathcal{I} \rightarrow \mathcal{B}, \tag{4.3}$$

with $\mathcal{I}$ being the image space and $\mathcal{B}$ being the perception functions' output space.

The rather abstract nature of $\mathcal{B}$ as a formal space depends on the perception task. For segmentation tasks, $\mathcal{B}$ consists of all possible segmentation maps, while $\mathcal{B}$ consists of all possible combinations of object bounding boxes for an object detector.

Definition 4.6 (Perception system (System-under-Test))**.** *A perception system is defined as a function*

$$D : \mathcal{I} \rightarrow \mathcal{B}, \tag{4.4}$$

with $\mathcal{I}$ being the image space and $\mathcal{B}$ being the output space as defined in Def. 4.5.

A perception function should approximate the ground truth, and an optimal object detector would fulfill $D(x) = T(x),\ \forall x \in \mathcal{I}$.

The emphasis of this thesis is laid upon image-based perception systems. Interchangeably, the image space $\mathcal{I}$ can be replaced by radar/lidar signals or a combination of perception signals.

Definition 4.7 (Simulation process)**.** *For a given functional scenario $\mathcal{F}_i$ and an perception function $D(\cdot)$, a simulation process is defined as*

$$G_i : \mathcal{S}_i \rightarrow \mathfrak{P}(\mathcal{B}) \times \mathfrak{P}(\mathcal{I}), \tag{4.5}$$

with $\mathcal{I}$ being the image space, $\mathcal{B}$ being the output space, $\mathcal{S}_i$ being the logical scenario space for $\mathcal{F}_i$, and $\mathfrak{P}(\cdot)$ being the power set operation.

The simulation process iteratively generates $I_t^i \in \mathcal{I}$ at time step t with corresponding ground truth $I_t^i \in \mathcal{I}$ taking into account the results of $D(I_{t-1}^i)$ for a given concrete scenario $s \in S_i$.

Definition 4.8 (Key-performance-indicator)**.** *With $\mathcal{B}$ being the space of ground truth elements, the perception systems' performance of a simulation run $\{D(I_t^i)\} \in \mathfrak{P}(\mathcal{B})$ compared to the corresponding ground truth $\{T(I_t^i)\} \in \mathfrak{P}(\mathcal{B})$ is measured by a key-performance-indicator (KPI) function*

$$K : \mathfrak{P}(\mathcal{B}) \times \mathfrak{P}(\mathcal{B}) \to \mathbb{R} \qquad (B_1, B_2) \mapsto K(B_1, B_2). \tag{4.6}$$

Usually the higher the KPI value $K(B_1, B_2)$, the better the prediction B_1.

Example 4.4 (Key-performance-indicator)**.** *The mean Average Precision (mAP) is introduced with Eq. 2.9. The mAP is a standard measure for quantifying an object detectors' performance on a given data set for which the ground truth is given.*

Combining these definitions, the complete simulative testing process of the object detector on the scenario space $\mathcal{S}$ can be defined.

Definition 4.9 (Simulative testing process)**.** *Evaluating the behavior of an object detector $D(\cdot)$ on the scenario space $\mathcal{S}_i$ using a data generator $G_i(\cdot)$ and a KPI function $K(\cdot,\cdot)$ is done by the simulative testing process*

$$\psi_i : \mathcal{S}_i \to \mathbb{R}, \qquad s \mapsto \psi_i(s) = K(G_i^1(s), D(G_i^2(s))), \tag{4.7}$$

where $G_i^1(s)$ is the set of ground truth outputs of $G_i(s)$ and $D(G_i^2(s))$ is the set of predictions.

The testing process aims to identify system failures, defined as the concrete scenarios in which the performance of the object detector does not comply with a pre-defined requirement.

Definition 4.10 (System failure)**.** *Let $D(\cdot)$ be an object detector with $\psi_i(\cdot)$ being its simulative testing process on functional scenario $\mathcal{F}_i$ with logical scenario space $\mathcal{S}_i = I(\mathcal{F}_i)$. A system failure is then defined as a concrete scenario in which the performance of $D(\cdot)$ is equal or falls below a pre-defined threshold $t \in \mathbb{R}$:*

$$x \in \mathcal{S}_i \textit{ is failure} \Leftrightarrow \psi_i(x) \leq t. \tag{4.8}$$

Utilizing this definition to derive a probability of failure requires a probability density function $f_{\mathcal{S}_i}$ for logical scenario $\mathcal{S}_i$ containing information on the real world occurrence probability of logical scenario variables. Then, the probability of failure for this $p_{\text{fail}\mathcal{F}_i}$ is defined as the integral of $f_{\mathcal{S}_i}$ over the subspace of $\mathcal{S}_i$ where the perception systems' performance is below or equal the threshold t:

Definition 4.11 (Scenario failure probability)**.** *Let $\mathcal{S}_i$ be the logical scenario space of $\mathcal{F}_i$ with probability density function $f_{\mathcal{S}_i}$. The scenario failure probability is then defined as the probability of the system failing in functional scenario $\mathcal{F}_i$:*

$$p_{fail\,\mathcal{F}_i} = \mathbb{P}(\{\psi(\mathcal{S}_i) \leq t\}) = \int_{\{x \in \mathcal{S}_i | \psi(x) \leq t\}} f_{\mathcal{S}_i}(x)\, dx. \tag{4.9}$$

Estimation of the scenario failure probability can then be used to estimate an overall failure probability for the system across the total space of functional scenarios $\mathfrak{F}$. As Def. 4.2 mentions, functional scenarios are made from the linguistic description and are assumed to be finite. Then the space $\mathfrak{F}$ has a discrete probability distribution $q_{\mathfrak{F}} = (q_1, ..., q_n)$, stating the occurrence probability for each functional scenario.

Definition 4.12 (System failure probability)**.** *With $\mathfrak{F} = \{\mathcal{F}_1, ..., \mathcal{F}_n\}$ being the space of functional scenarios, let $p_{\mathfrak{F}} = (p_{fail\,\mathcal{F}_1}, ..., p_{fail\,\mathcal{F}_n})$ be the vector of* *scenario failure probabilities and* $q_{\mathfrak{F}} = (q_1, ..., q_n)$ *be the vector of occurrence probabilities, then the system failure probability is calculated by*

$$p_{failure} = p_{\mathfrak{F}} * q_{\mathfrak{F}}^T. \tag{4.10}$$

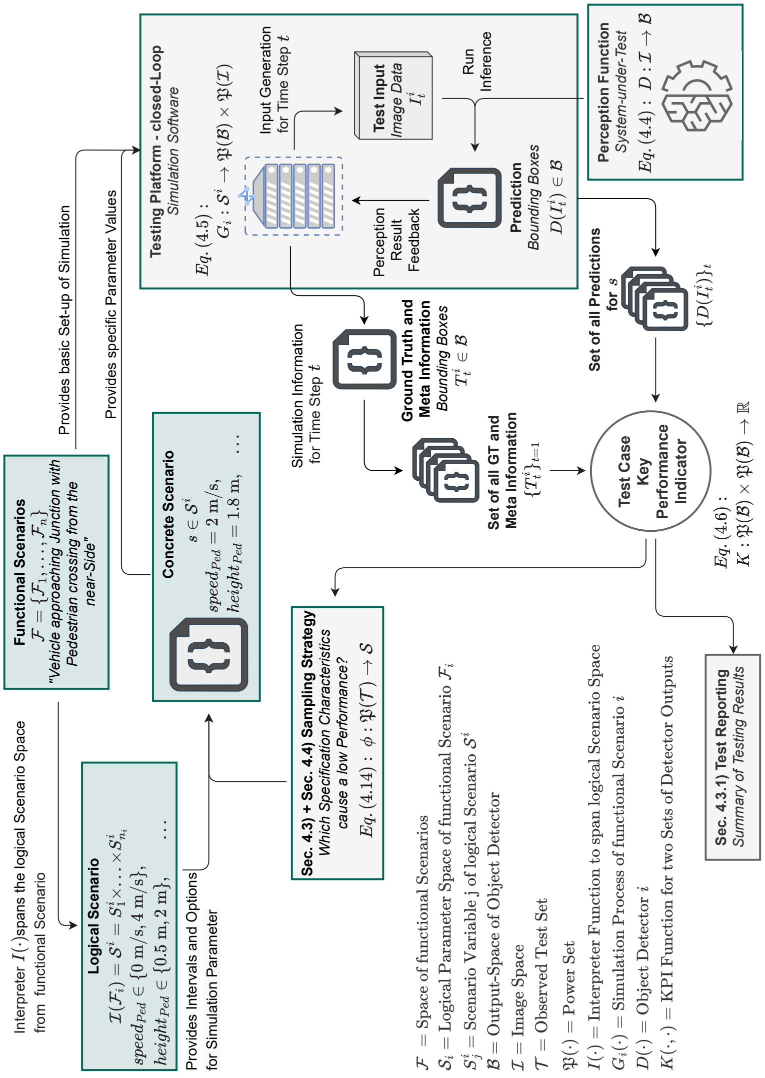

Interpreter $I(\cdot)$spans the logical Scenario Space from functional Scenario
Functional Scenarios
$\mathcal{F} = \{\mathcal{F}_1, \ldots, \mathcal{F}_n\}$
"Vehicle approaching Junction with Pedestrian crossing from the near-Side"
Provides basic Set-up of Simulation
Provides specific Parameter Values
Logical Scenario
$\mathcal{I}(\mathcal{F}_i) = \mathcal{S}^i = S_1^i \times \ldots \times S_{n_i}^i$
$speed_{Ped} \in \{0\ \mathrm{m/s}, 4\ \mathrm{m/s}\}$,
$height_{Ped} \in \{0.5\ \mathrm{m}, 2\ \mathrm{m}\}$, ...
Provides Intervals and Options for Simulation Parameter
Concrete Scenario
$s \in \mathcal{S}^i$
$speed_{Ped} = 2\ \mathrm{m/s}$,
$height_{Ped} = 1.8\ \mathrm{m}$, ...
Sec. 4.3) + Sec. 4.4) Sampling Strategy
Which Specification Characteristics cause a low Performance?
$Eq.\ (4.14): \phi : \mathfrak{P}(\mathcal{T}) \rightarrow \mathcal{S}$
Testing Platform - closed-Loop
Simulation Software
$Eq.\ (4.5):$
$G_i : \mathcal{S}^i \rightarrow \mathfrak{P}(\mathcal{B}) \times \mathfrak{P}(\mathcal{I})$
Input Generation for Time Step t
Test Input
Image Data
I_t^i
Run Inference
Perception Result Feedback
Prediction
Bounding Boxes
$D(I_t^i) \in \mathcal{B}$
Perception Function
System-under-Test
$Eq.\ (4.4): D : \mathcal{I} \rightarrow \mathcal{B}$
Simulation Information for Time Step t
Ground Truth and Meta Information
Bounding Boxes
$T_t^i \in \mathcal{B}$
Set of all GT and Meta Information
$\{T_t^i\}_{t=1}$
Set of all Predictions for s
$\{D(I_t^i)\}_t$
Test Case Key Performance Indicator
$Eq.\ (4.6):$
$K : \mathfrak{P}(\mathcal{B}) \times \mathfrak{P}(\mathcal{B}) \rightarrow \mathbb{R}$
Sec. 4.3.1) Test Reporting
Summary of Testing Results
$\mathcal{F}$ = Space of functional Scenarios
$\mathcal{S}_i$ = Logical Parameter Space of functional Scenario $\mathcal{F}_i$
S_j^i = Scenario Variable j of logical Scenario $\mathcal{S}^i$
$\mathcal{B}$ = Output-Space of Object Detector
$\mathcal{I}$ = Image Space
$\mathcal{T}$ = Observed Test Set
$\mathfrak{P}(\cdot)$ = Power Set
$I(\cdot)$ = Interpreter Function to span logical Scenario Space
$G_i(\cdot)$ = Simulation Process of functional Scenario i
$D(\cdot)$ = Object Detector i
$K(\cdot,\cdot)$ = KPI Function for two Sets of Detector Outputs

Figure 4.8: Schematic representation of the simulative testing process. Test inputs and the corresponding ground truths are generated based on a combinatorial scenario space. Applying an object detector allows a systematic performance assessment in the scenario space.

4.3 Defining quality metrics for an adaptive test case sampling

In theory, the testing process assumes that every concrete scenario in the scenario space can be generated by the data generator process $G_i(\cdot,\cdot)$. This would allow observing the perception systems' performance for every scenario. However, in practice, a test execution is computationally expensive and time-consuming. A high dimensionality of the logical scenario space $\mathcal{S}_i$ and constraints of the computational resources prevent the full factorial testing of the scenario space. Hence, the testing process executions have to be planned so that the scenario space is explored most efficiently.

Definition 4.13 (Observed concrete scenario and observed test set)**.** *With $\mathcal{S}$ being the scenario space and $\psi(\cdot)$ being the simulative testing process of object detector $D(\cdot)$ then observed concrete scenario is defined as follows:*

$$x \in \mathcal{S} \textit{ is observed concrete scenario} \Leftrightarrow \psi(x) \textit{ is known}. \tag{4.11}$$

Simultaneously an observed test set $\tilde{\mathcal{S}}$ is defined as follows:

$$\tilde{\mathcal{S}} \subset \mathcal{S} \textit{ is observed test set} \Leftrightarrow \psi(x) \textit{ is known } \forall x \in \tilde{\mathcal{S}}. \tag{4.12}$$

To make observed test sets more practical for the following definitions, a notation for the space of observed concrete scenarios is introduced as follows:

$$\mathcal{T} = \{(x,\psi(x))|x \in \mathcal{S}\} \subset \mathcal{S} \times [0,1]. \tag{4.13}$$

Hence, any element $(x,\psi(x)) \in \mathcal{T}$ consists of a concrete scenario specification and its test result. Likewise, any subset $\tilde{\mathcal{T}} \subset \mathcal{T}$ is an observed test set.

Building on an observed test set, a sampling strategy can be defined as the selection process of concrete scenarios out of the scenario space.

Definition 4.14 (Sampling strategy)**.** *A sampling strategy is defined as a function that selects an untested scenario out of the scenario space $\mathcal{S}$, based on an already observed test set $\tilde{\mathcal{T}}$:*

$$\phi : \mathfrak{P}(\mathcal{T}) \to \mathcal{S}, \quad \tilde{\mathcal{T}} \mapsto \phi(\tilde{\mathcal{T}}). \tag{4.14}$$

The sampling strategy takes an observed test set as input and returns an untested scenario. Then, the simulative testing process is triggered to observe the detectors' performance. Finally, the result is added to the observed test set, and the sampling strategy is applied again. Hence, the scenario space is explored according to the underlying objectives of the sampling strategy.

Definition 4.15 (Sampling order and failure sequence)**.** *A sampling order is defined as the recursive sequence $(x_i)_{i\in\mathbb{N}}$, $x_i \in \mathcal{S}$ that results from applying a sampling strategy $\phi(\cdot)$ iteratively to the scenario space $\mathcal{S}$:*

$$x_{n+1} = \phi\left(\bigcup_{i=1}^{n}(x_i, \psi(x_i))\right), \tag{4.15}$$

where $\bigcup_{i=1}^{n}(x_i, \psi(x_i))$ is the aggregation of all previous selected concrete scenarios.

Based on the sampling order, the strategies' ability to detect failures can be evaluated from the sequence of test results $(\psi_i)_{i\in\mathbb{N}}$, $\psi_i = \psi(x_i)$.

Then, with threshold $t > 0$, a failure sequence $(\tau_i)_{i\in\mathbb{N}}$ can be generated palely with

$$\tau_i = \begin{cases} 1 & \text{if } \psi_i \le t \\ 0 & \text{otherwise}. \end{cases} \tag{4.16}$$

Hence, τ_i equals 1 if x_i is a failure and is equal to 0 otherwise.

Finally, for evaluation, the function Θ is defined as the accumulation of the sequence τ_i:

$$\Theta : \mathbb{N} \to \mathbb{N}, \quad n \mapsto \sum_{i=1}^{n} \tau_i . \tag{4.17}$$

See Fig. 4.9 for a visualization.

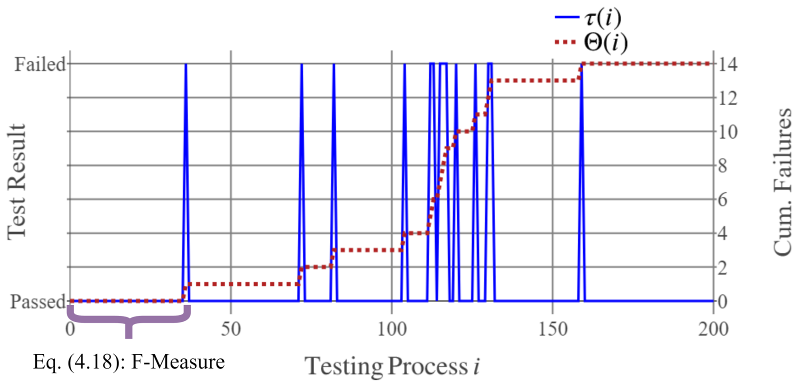


Figure 4.9: Visualization of the sequences τ_i and Θ_i as defined in Def. 4.15.

Defining detection metrics

To evaluate the effectiveness of the test strategies, metrics have to be introduced. Consider τ_i and Θ being defined as the failure sequence of a sampling order (see Def. 4.15). Then the failure detection metrics are defined as follows:

Definition 4.16 (*F-Measure*)**.** *The standard F-measure is the number of tests required to detect the first failure. Formally, that equates to the following:*

$$F\text{-}Measure = \min\ \Theta^{-1}(1). \tag{4.18}$$

The standard score of the F-measure is the first position in the testing process with $\Theta = 1$.

The number of tests before detecting the first failure may be influenced by some randomness and can be misleading. To design the original F-measure more robustly, the F_n-measure is introduced to measure the number of concrete scenarios to detect n failures:

$$F_n\text{-}Measure = \min\ \Theta^{-1}(n). \tag{4.19}$$

Another approach is to consider alternative sampling strategies for direct efficiency comparison. A strategy's ability to detect failures held against random concrete scenario selection generates an intuitive measure to evaluate the cost saving through adaptive testing. For this, the C-measure is introduced.

Definition 4.17 (*C-Measure*)**.** *The C-measure is defined by this work as the saved costs for test executions to detect n failures relative to random testing.*

Formally, the C-measure is expressed by

$$C_n\text{-}Measure = 1 - \frac{F_n^s}{F_n^r}, \tag{4.20}$$

where F_n^s is the F-measure of the test strategy for n failures and F_n^r is the F-measure for random concrete scenario selection for n failures.

Example 4.5 (*C-Measure*)**.** *Assume the testing process with random testing detecting the first failure after 100 test executions. Thus $F_1^r = 100$. Additionally, assume an adaptive strategy detecting the first failure in the first test execution. Thus $F_1^s = 1$. Then*

$$C_n\text{-}Measure = 1 - \frac{1}{100} = 0.99, \tag{4.21}$$

which states that the fraction of saved test executions is 99%.

Introducing failure regions to avoid oversampling

Even for low failure probabilities of the system-under-test in the scenario space, the set of failures in the scenario space may be uncountable infinite. For example, assume a system-under-test failing all concrete scenarios for sun altitude between 40° and 50° and sun azimuth between −5° and 5° due to blinding sunlight. As soon as the testing process establishes this failure root, sampling from this region in the scenario space does not provide additional information. However, a greedy sampling strategy has shown to focus on this region due to the high density of failures (compare with results in Sec. 6.3). The plain amount of detected failures would not represent the strategy's real performance since the objective is to find diverse failure causes instead of a high number of small variations of one failure. Optimally, the sampling strategy should ignore regions established as failure regions to sample for yet undetected failure regions. For this purpose, the concept of failure patterns is used and customized for the applications of this thesis.

Chan et al. [140] describe different patterns to characterize the distribution and occurrence of failures. Based on this concept, the following definitions introduce failure regions. Using these, the relevant region for the sampling process can be determined and potential concrete scenarios can be checked before test execution.

Def. 4.18 and Def. 4.19 motivate the idea formally for a discrete scenario space, while Def. 4.20 introduces a measure to examine whether testing a yet-untested concrete scenario provides more inside on the failure distribution.

Definition 4.18 (Neighbour concrete scenarios)**.** *Let $\mathcal{S} = S_1 \times ... \times S_n$ be a scenario space with ordered discrete numerical scenario variables $S_i = \{s_{i,1}, ..., s_{i,n_i}\}$. Two concrete scenarios $x_1, x_2 \in \mathcal{S}$ are defined as neighbours if the concrete scenarios are equal for all but one dimension $i \in \{1, ..., n\}$ for which the two values are next to each other. For neighbours a notation is introduced:*

$$x_1, x_2 \in \mathcal{S} \textit{ are neighbours } \Leftrightarrow \ x_1 \diamond x_2 \tag{4.22}$$

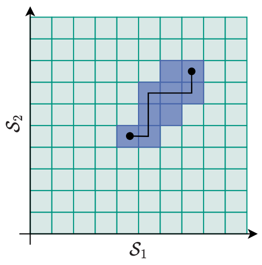


Figure 4.10: Visualization of failure region with non-failures in green and failures in dark blue. Every two concrete scenarios in the red area $\hat{\mathcal{S}} \subset \mathcal{S}_1 \times \mathcal{S}_2$ are connected failures.

Definition 4.19 (Connected failures and failure region)**.** *Let $x_1, x_2 \in \mathcal{S}$ be two concrete scenarios in the same same setup as Def. 4.18. Both concrete scenarios are failures as defined in Def. 4.8. x_1 and x_2 are now defined as connected failures if there exists a path from x_1 to x_2 via other failures that are neighbours:*

$$\begin{aligned} x_1, x_2 \in \mathcal{S} \textit{ are connected failures } \Leftrightarrow \exists\{\hat{x}_1, ..., \hat{x}_l\} \subset \mathcal{S} \textit{ with} \\ x_1 \diamond \hat{x}_1 \diamond ... \diamond \hat{x}_l \diamond x_2 \textbf{ and } \forall\, i \in \{1, ..., l\} : \hat{x}_i \textit{ is failure.} \end{aligned} \tag{4.23}$$

Using this definition, a set $\hat{\mathcal{S}} \subset \mathcal{S}$ is defined as failure region if all concrete scenarios are pair-wise connected failures:

$$\hat{\mathcal{S}} \subset \mathcal{S} \textit{ is failure region } \Leftrightarrow \forall\, x_1, x_2 \in \hat{\mathcal{S}} : x_1 \textit{ and } x_2 \textit{ are connected failures.} \tag{4.24}$$

Fig. 4.10 visualizes the concept of a failure region for a two-dimensional discrete space. Every failure is connected to every other failure in the failure region.

However, Def. 4.18 and Def. 4.19 are only applicable if the scenario space is discrete. For continuous scenario spaces, a more applicable measure has to be introduced to evaluate whether a candidate concrete scenario is inside a failure region:

Definition 4.20 (ω-measure and sampling-relevant concrete scenarios)**.** *Let $x \in \mathcal{S}$ be an untested concrete scenario and $\tilde{\mathcal{S}} \subset \mathcal{S}$ be a set of observed concrete scenarios. Define $\mathcal{B}(x) = \{\tilde{x} \in \tilde{\mathcal{S}} | d(x, \tilde{x}) < \epsilon\}$ as a metric ball around x that contains all observed concrete scenarios with distance $d(\cdot, \cdot)$ smaller than some threshold $\epsilon > 0$. The relevance score $\omega(x)$ is then defined as the difference between non-failures and failures inside $\mathcal{B}(x)$:*

$$\omega(x) = \#\{\tilde{x} \in \mathcal{B}(x) | \psi(\tilde{x}) > t\} - \#\{\tilde{x} \in \mathcal{B}(x) | \psi(\tilde{x}) \leq t\} \tag{4.25}$$

Let $t_\omega \in \mathbb{N}^-$ be a threshold for $\omega(x)$.

$$x \textit{ is sampling-relevant } \Leftrightarrow \omega(x) > t_\omega. \tag{4.26}$$

See Fig. 4.11 for a visual demonstration. Assume $t_\omega = -5$. Then both candidate concrete scenarios get evaluated on whether it is assumed that they belong to a failure region. The first candidate concrete scenario with $\omega_1 = -6 < t_\omega$ would be ignored since its immediate surrounding is already considered a failure region, and spending computation resources is expected to generate no additional insight into the distribution of failures in the scenario space. The second candidate concrete scenario with $\omega_2 = 2 > t_\omega$ is located in an area that still requires exploration. While the fact that the concrete scenario was selected by a sampling strategy indicates a high failure probability, $\omega_2 > t_\omega$ indicates that the area is not considered a failure region yet.

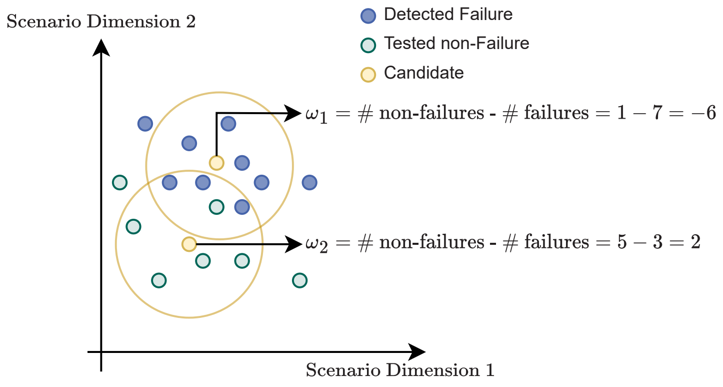

Scenario Dimension 2
Detected Failure
Tested non-Failure
Candidate
ω_1 = # non-failures - # failures = $1 - 7 = -6$
ω_2 = # non-failures - # failures = $5 - 3 = 2$
Scenario Dimension 1

Figure 4.11: Visualization of ω-measure: The difference between non-failures and failures inside the metric ball around candidate concrete scenarios indicates whether the candidate is inside a failure region.

4.4 Presentation of sampling strategies

4.4.1 Covering arrays and t-wise testing

Dealing with the combinatorial explosion has been a problem for traditional software, exhausting available testing resources with an unmanageable number of input variations, as illustrated in Eq. 4.1. To reduce the number of test executions of the input space, the use of covering arrays has been proposed [141]. Using covering arrays for concrete scenario generation, also called $t -$ wise testing [69], is a combinatorial technique to uniformly explore the input space for patterns that cause system failures. Covering arrays can be applied to finite input space with a high number of discrete variables. Instead of testing all possible combinations of the input variables, the set of concrete scenarios aims at covering all $t -$ wise combinations of characteristics across all variables. This set is then called a covering array. If $t = 2$, the covering array should contain concrete scenarios, such that there is at least one concrete scenario containing characteristic s_1 of scenario variable $\mathcal{S}_i$ and characteristic s_2 of scenario variable $\mathcal{S}_i$ $(i \neq j)$. This concept is closely related to the neuron coverage measure *t-Way Combination sparse Coverage* in Sec. 2.3.

Fig. 4.12 displays how this $2 -$ wise testing method, also called pair-wise testing, can be used to reduce the full-factorial test space from eight concrete scenarios to four concrete scenarios while upholding full coverage of pair-wise combinations. This approach examines combinatorial patterns in the input for their influence on the output. Such covering arrays can be generated by a range of different algorithms. The standard brute force algorithm for covering array generation is shown in Alg. 1.

Empirical studies in traditional software engineering have shown that the testing of all $3 -$ wise combinations can detect 90% of software bugs [142]. While most of the work in this research area was done for traditional software and hardware testing, utilizing simulative tools to extend existing image data sets for the development of perception functions to achieve full coverage of combinations has been proposed by Gladisch et al. [69]. The common use of covering arrays does not account for the performance of the system-under-test during the sampling process. A metric to evaluate the combinatorial coverage if specific characteristics are weighted with higher importance and an algorithm for concrete scenario selection has been introduced by Cheng et al. [68].

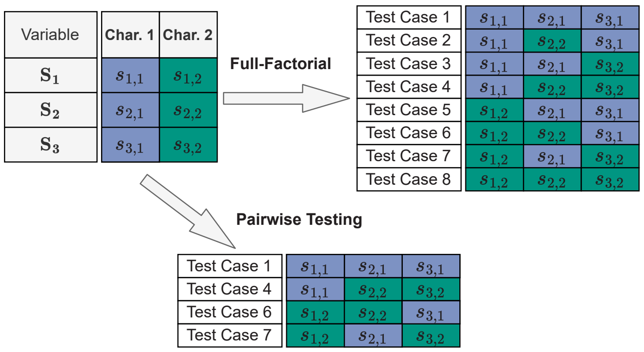

Variable
Char. 1
Char. 2
S_1
$s_{1,1}$
$s_{1,2}$
S_2
$s_{2,1}$
$s_{2,2}$
S_3
$s_{3,1}$
$s_{3,2}$
Full-Factorial
Test Case 1 $s_{1,1}$ $s_{2,1}$ $s_{3,1}$
Test Case 2 $s_{1,1}$ $s_{2,2}$ $s_{3,1}$
Test Case 3 $s_{1,1}$ $s_{2,1}$ $s_{3,2}$
Test Case 4 $s_{1,1}$ $s_{2,2}$ $s_{3,2}$
Test Case 5 $s_{1,2}$ $s_{2,1}$ $s_{3,1}$
Test Case 6 $s_{1,2}$ $s_{2,2}$ $s_{3,1}$
Test Case 7 $s_{1,2}$ $s_{2,1}$ $s_{3,2}$
Test Case 8 $s_{1,2}$ $s_{2,2}$ $s_{3,2}$
Pairwise Testing
Test Case 1 $s_{1,1}$ $s_{2,1}$ $s_{3,1}$
Test Case 4 $s_{1,1}$ $s_{2,2}$ $s_{3,2}$
Test Case 6 $s_{1,2}$ $s_{2,2}$ $s_{3,1}$
Test Case 7 $s_{1,2}$ $s_{2,1}$ $s_{3,2}$

Figure 4.12: Example of pair-wise testing with three scenario variables and two characteristics per variable. Overall there are eight possible combinations. Utilizing combinatorial testing, the number of concrete scenarios to cover each two-wise combination of characteristics can be reduced to four.

In general, the advantage of covering arrays is a uniform sampling of the test space. Thus, the testing of all $t-$ wise combinations in the scenario space can be guaranteed. However, covering arrays only can be applied to finite scenario spaces and require an eventual discretization of continuous variables. Furthermore, solely combinatorial sampling strategies do not incorporate adaptive elements that specifically provoke failures. Covering arrays can be used as an initial sampling to guarantee a certain level of coverage of the scenario space.

4.4.2 Evolutionary sampling

Evolutionary learning algorithms are often used for optimization problems where little is known about the relationship between a process's input and output. Evolutionary sampling for software testing is derived from the natural process of evolution [143, 144]. Evolutionary algorithms iteratively mutate already observed concrete scenarios $s \in \tilde{\mathcal{S}} \subset \mathcal{S}$ with a low-performance value $\psi(s)$ to increase the failure detection rate. The general algorithm starts with an initial set of n_e observed concrete scenarios $\tilde{\mathcal{S}}$. After evaluating $\psi(s)$ for each concrete scenario in $\tilde{\mathcal{S}}$ the l concrete scenarios with the lowest KPI values are extracted as the first parent generation, with l being a pre-defined parameter. Afterward, each element of the parent generation is mutated k

Algorithm 1 Brute force algorithm for covering array generation [142]

```
Require: concrete scenarios of finite scenario space $\mathcal{S}$ in arbitrary order
 1: Calculate total number of $t-$wise combinations $\lambda^t_{\mathcal{S}}$ in space $\mathcal{S}$
 2: Initialize number of covered $t-$wise combinations $\lambda^t_{cov} = 0$ and empty
    test set $\tilde{\mathcal{S}}$
 3: while $\lambda^t_{cov} \neq \lambda^t_{\mathcal{S}}$ do
 4:     Initialize $\lambda^t_{max} = 0$ and $x_{max} = []$
 5:     for every scenario $x \in \mathcal{S} \wedge x \notin \tilde{\mathcal{S}}$ do
 6:         Calculate amount of $t-$wise combinations of $x$ that are yet uncov-
    ered in $\tilde{\mathcal{S}}$ as $\lambda^t_x$
 7:         if $\lambda^t_x > \lambda^t_{max}$ then
 8:             $x_{max} = x$
 9:             $\lambda^t_{max} = \lambda^t_x$
10:         end if
11:     end for
12:     $\lambda^t_{cov} = \lambda^t_{cov} + \lambda^t_{max}$
13:     Append $x_{max}$ to $\tilde{\mathcal{S}}$
14: end while
15: return Test set $\tilde{\mathcal{S}}$
```

times, marginally changing one scenario parameter. A mutation $\dot{s}$ has to be a valid untested scenario ($\dot{s} \in \mathcal{S} \setminus \tilde{\mathcal{S}}$). After observing $\psi(\dot{s})$ for each mutation $\dot{s}$, the next parent generation is selected. This process emulates natural principle of *survival of the fittest* and is depicted in Fig. 4.13.

This brute force version of evolutionary optimization can be extended by changing the parent generation selection process. For example, joining random observed concrete scenarios to the parent generation adds an exploratory element. Furthermore, the mutation step can be extended by selecting certain scenario variables more likely to change. These variables are identified to have more impact on the object detectors' performance.

Applying evolutionary sampling for simulative test generation for automated driving functions has proven to achieve effective identification of challenging concrete scenarios [135,145]. The advantage of evolutionary concrete scenario sampling is the autonomy of a mathematical function and the low required pre-knowledge of the underlying relationship $\psi(\cdot)$ between the scenario space and

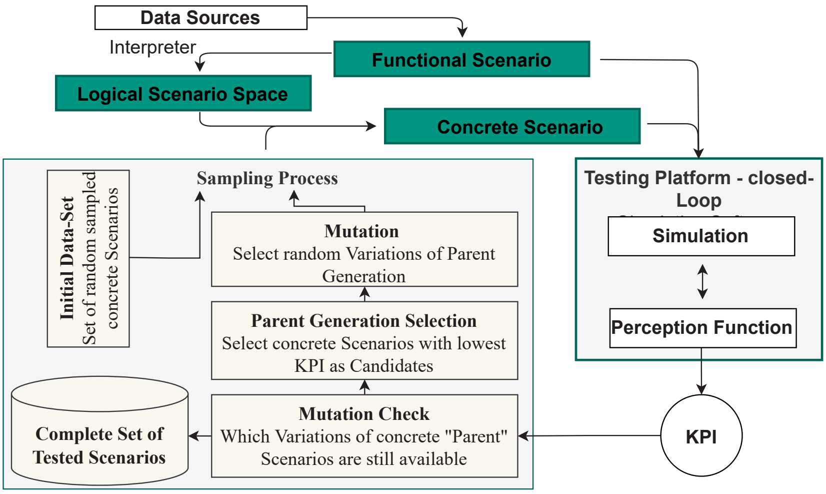

Data Sources
Interpreter
Functional Scenario
Logical Scenario Space
Concrete Scenario
Sampling Process
Initial Data-Set
Set of random sampled concrete Scenarios
Mutation
Select random Variations of Parent Generation
Parent Generation Selection
Select concrete Scenarios with lowest KPI as Candidates
Mutation Check
Which Variations of concrete "Parent" Scenarios are still available
Complete Set of Tested Scenarios
Testing Platform - closed-Loop
Simulation
Perception Function
KPI

Figure 4.13: Schematic representation of evolutionary testing process. Iteratively, concrete scenarios are selected for test execution, and the concrete scenarios with the lowest results are used to select new concrete scenarios by mutation.

the object detectors' performance. However, the significant random element of evolutionary algorithms may be more inefficient than alternative approaches like meta-model based sampling.

4.4.3 Coupling the use of meta-model with fixed-sized-candidate-set method

Meta-model based sampling strategies root in the assumption that the performance $\psi(\cdot)$ of the object detector $D(\cdot)$ on the scenario space S can be approximated by a mathematical function, called meta-model. While test executions are computationally expensive, a meta-model can be evaluated cheaply. Furthermore, derivations of meta-model can be used to apply numerical optimization algorithms.

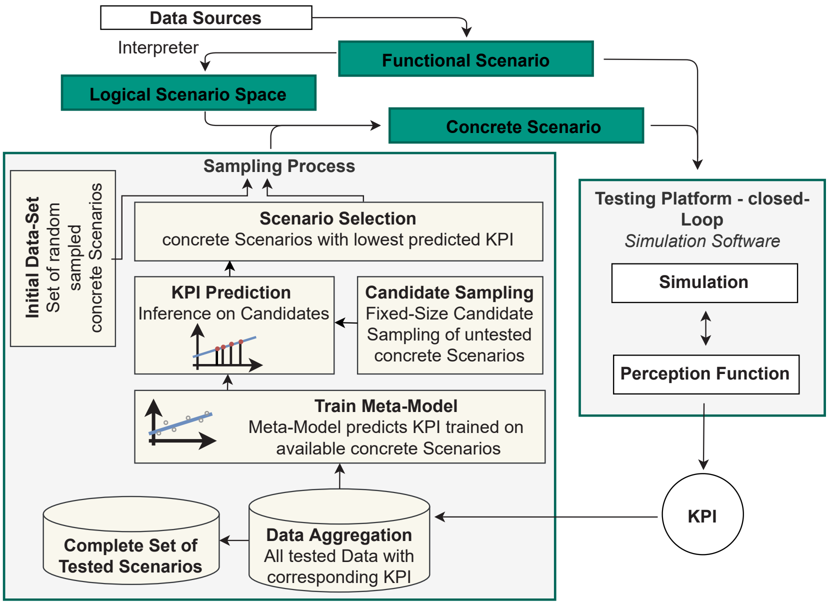

Data Sources
Interpreter
Functional Scenario
Logical Scenario Space
Concrete Scenario
Sampling Process
Initial Data-Set
Set of random sampled concrete Scenarios
Scenario Selection
concrete Scenarios with lowest predicted KPI
KPI Prediction
Inference on Candidates
Candidate Sampling
Fixed-Size Candidate Sampling of untested concrete Scenarios
Train Meta-Model
Meta-Model predicts KPI trained on available concrete Scenarios
Complete Set of Tested Scenarios
Data Aggregation
All tested Data with corresponding KPI
Testing Platform - closed-Loop
Simulation Software
Simulation
Perception Function
KPI

Figure 4.14: Schematic representation of meta-model based testing process.

Definition 4.21 (Meta-model). *Let $\mathcal{S}$ be a scenario space and $\tilde{\mathcal{S}} \subset \mathcal{S}$ be an observed subset as defined in Eq. 4.12. Based on $\tilde{\mathcal{S}}$, a so-called meta-model*

$$M_{\tilde{\mathcal{S}}} : \mathcal{S} \to [0, 1] \tag{4.27}$$

can be trained that emulates the behavior of the simulative testing process $\psi(\cdot)$ from Eq. 4.7.

Meta-model based testing aims to create a bridge between the scenario space and the object detectors' performance that allows estimating the performance reliability for concrete scenarios without triggering the test execution [146]. Setting up a meta-model requires an already observed test-set that serves for model training and model selection. Iteratively, the meta-model is used to select untested concrete scenarios that indicate a low performance of the object detector utilizing inter- and extrapolation. Triggering the simulative testing process for these concrete scenarios provides insight into the effect of scenario variables on the object detectors' performance. Variables that have significant impact on the relationship between scenario space and the system-under-tests' performance can be identified, and the detection rate of failures can be optimized. Furthermore, during the sampling process, the training data for the meta-model gets more comprehensive, theoretically increasing its accuracy. The basic algorithm for this adaptive test strategy is shown in Alg. 2.

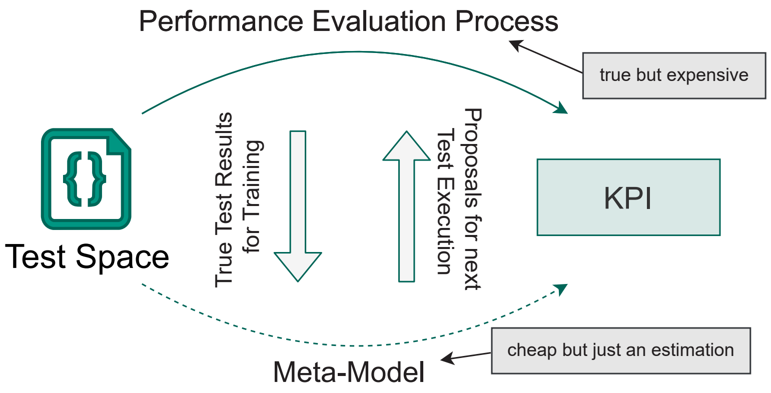

Performance Evaluation Process
true but expensive
Test Space
True Test Results for Training
Proposals for next Test Execution
KPI
cheap but just an estimation
Meta-Model

Figure 4.15: Schematic representation of the relationship between performance evaluation process and the meta-model. Test results from the performance evaluation process are used to train the meta-model. The meta-model is then used to generate proposals for the performance evaluation process.

The algorithm strictly chooses the subsequent concrete scenarios from the learned meta-model, making the meta-models' accuracy central for the performance of the sampling process. Hence, the choice of the meta-model is crucial. A multitude of regression functions are available for this task, such as Gaussian process regression [146], neural networks [147–149], and support vector machines [150]. As for all machine learning tasks, the chosen function represents the relationship between input and output space. However, if the meta-model is chosen wrongly or $\psi(\cdot)$ contains too much random behavior that cannot be learned, meta-model based sampling strategies cannot work efficiently.

Algorithm 2 Meta-model based sampling

```
Require: Scenario space $\mathcal{S}$, meta-model $M(s)$, batch size $b \in \mathbb{N}$, simulative
    testing process $\psi(\cdot)$
 1: Select initial concrete scenarios $\tilde{\mathcal{S}} \subset \mathcal{S}, \#\tilde{\mathcal{S}} = b$
 2: Observe real performances of concrete scenarios $\psi(\tilde{\mathcal{S}})$
 3: while Testing resources not exhausted and testing objectives not achieved
    do
 4:     if $M(s)$ is dependent on hyperparameters then
 5:         Optimize hyperparameters of $M(s)$ acc. to Alg.
 6:     end if
 7:     Fit $M_{\tilde{\mathcal{S}}}(s)$ using $\tilde{\mathcal{S}}$
 8:     Predict $M_{\tilde{\mathcal{S}}}(\mathcal{S})$ for a set of untested concrete scenarios selected using
    FSCS
 9:     Select $b_1$ sampling-relevant concrete scenarios with the lowest predic-
    tion and further $b_2$ sampling-relevant concrete scenarios by random for
    test executions
10:     Trigger simulative testing process $\psi(\cdot)$
11:     Add observed concrete scenarios to $\tilde{\mathcal{S}}$ and observe real performances
    $\psi(\tilde{\mathcal{S}})$
12: end while
13: return Order in which concrete scenarios were added and corresponding
    test results for evaluation
```

Fixed-sized-candidate-set method A problem with meta-model based sampling is that the scenario space $\mathcal{S}$ may be continuous or contain too many elements for predicting $M(x)$ for all possible concrete scenarios. A popular workaround for this problem is using a fixed-sized-candidate-set (FSCS) [151, 152]. Each iteration generates a random set of candidates for which the meta-model predicts the performance. The advantage is the reduced computational effort during the prediction phase. The disadvantage is that failures must be sampled into the FSCS first to be selected by the meta-model.

Meta-model selection Theoretically, fitting a meta-model can be done with every regression model. However, fitting a regression model is tied to making assumptions about the underlying relationship, and each model has advantages and disadvantages. This dissertation uses three models: Linear regression, neural network regression, and Gaussian process regression.

Hyperparameter optimization Some meta-models require pre-defined hyperparameters. An example is the architecture of a neural network, which includes the number of layers/neurons or types of activation functions. The performance of the sampling strategy depends highly on how well the meta-model is adjusted to the task. Techniques like grid-search [153], random-search [154], or evolutionary optimization [155] are established approaches to optimize hyperparameters.

The two meta-models presented here that use hyperparameters, neural network, and Gaussian process regression only use a limited set of hyperparameters. For optimization, a pre-defined grid of parameter specifications was set up. The optimization was done utilizing a grid of pre-defined hyperparameter combinations. The available data is partitioned into a development and test set for each iteration in the test process. After training the model for each parameter combination with the development set, the mean-squared error for the test set is calculated. After finishing the process, the best model is selected for the sampling process.

Linear regression

Linear regression is one of the most basic regression concepts. The relationship between an independent variable $\mathbf{x} \in \mathbb{R}^n$, also called explanatory variable, and a dependent variable $y \in \mathbb{R}$, also called response variable, is assumed to follow a linear model (LM). For meta-model based sampling, the explanatory variable is a numerical expression of the concrete scenario specification, while the response variable is the KPI value. The dependent variable is assumed to be generated by a linear combination of the dependent variable with the formal expression

$$\begin{aligned} f(\mathbf{x}) &= \beta_0 + \beta_1 * x_1 + \ldots + \beta_n * x_n + \epsilon \\ &= (\beta_0, \beta_1, \ldots, \beta_n) * \begin{pmatrix} 1 \\ x_1 \\ \ldots \\ x_n \end{pmatrix} + \epsilon = \boldsymbol{\beta} * \mathbf{x}^{-T} + \epsilon, \end{aligned} \tag{4.28}$$

with $\epsilon \sim (0, \sigma^2)$ being an error term that usually follows a normal distribution and $\boldsymbol{\beta} \in \mathbb{R}^{n+1}$ being the LM-parameters. For vector notation $\overline{\mathbf{x}} \in \mathbb{R}^{n+1}$ is the transformed explanatory variable that includes the 1 value that is multiplied with the intercept value β_0 (hence $f(0) = \beta_0$). However, similar to all regression problems, the real parameter set $\beta \in \mathbb{R}^{n+1}$ is unknown and thus has to be estimated based on a sample set $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_l\}$ with observed response values $\mathcal{Y} = \{f(\mathbf{x}_1), \ldots, f(\mathbf{x}_l)\} = \{\mathbf{y}_1, \ldots, \mathbf{y}_l\}$. For the LM-estimation the following matrix notation is introduced:

$$\mathbf{y} = \mathbf{X} * \boldsymbol{\beta} + \boldsymbol{\epsilon}, \qquad \text{with}$$
$$\mathbf{X} = \begin{pmatrix} 1 & x_{1,1} & \ldots & x_{n,1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{1,l} & \ldots & x_{n,l} \end{pmatrix}, \quad \mathbf{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_l \end{pmatrix}, \quad \boldsymbol{\beta} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_l \end{pmatrix}, \quad \boldsymbol{\epsilon} = \begin{pmatrix} \epsilon_1 \\ \vdots \\ \epsilon_l \end{pmatrix}. \tag{4.29}$$

Getting an usable estimator for $\boldsymbol{\beta}$ is done via the ordinary least squares estimation (OLS). The vector of estimation errors, the so-called residuals, when using an LM parameter vector $\hat{\boldsymbol{\beta}}$, is calculated with

$$\mathbf{e} = \mathbf{y} - \mathbf{X} * \hat{\boldsymbol{\beta}}. \tag{4.30}$$

The OLS estimation is done by finding the $\hat{\boldsymbol{\beta}}$ that minimizes the mean-squares error (MSE):

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \mathbf{MSE} = \arg\min_{\boldsymbol{\beta}} \frac{1}{l} \left((\mathbf{y} - \mathbf{X} * \boldsymbol{\beta}) * (\mathbf{y} - \mathbf{X} * \boldsymbol{\beta})^T \right). \tag{4.31}$$

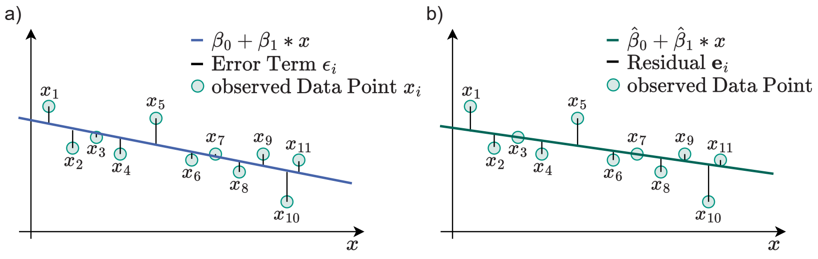


Figure 4.16: Relationship between true underlying function and fitted model for linear regression tasks. a) Sampled data points from the true underlying function with true parameters β_0, β_1. b) Fitted regression model using the available data points x_i with estimated parameters $\hat{\beta}_0, \hat{\beta}_1$.

For optimization, the function **MSE** is derived by $\hat{\boldsymbol{\beta}}$ and equated to 0 to deduce the estimator

$$\hat{\boldsymbol{\beta}} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}. \tag{4.32}$$

The relationship between the underlying true function and an estimated model is displayed in Fig. 4.16.

The advantages and disadvantages of LR models lie in their simplicity. Since the estimation is calculated as a linear combination of the input variables, each input variable x_i has its slope parameter β_i. The model does not assume any interaction between the variables. Thus, each slope parameter β_i directly indicates the influence x_i has on the estimation. This characteristic makes LR-models one of the most intuitive machine learning models and easy to interpret. If slope β_i is positive, an increase in the variable x_i will increase the model's output and vice versa. For example, if β_{fog} is negative, the scenario variable *Fog* is expected to have a negative influence on the KPI of the system-under-test. Furthermore, the calculation of the estimator is efficient, even when large amounts of data are observed.

However, LM models can only describe linear relationships between input and output and ignore any possible interaction between variables. The models' ability to capture complex relationships is limited and results can be inaccurate.

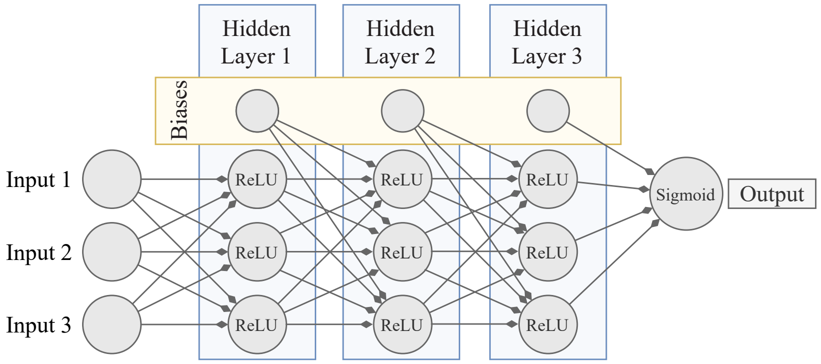


Figure 4.17: Standard feed-forward neural network that can be used for regression tasks.

Neural network

The second option of a meta-model focuses on a standard feed-forward neural network for regression. The functionality of neural networks is detailed in Sec. 2.4. Fig. 4.17 visualizes a standard architecture for such tasks, although the number of layers and nodes are variable. Using a sigmoid activation function for final prediction is convenient for KPI estimation for metrics between 0 and 1.

The advantages and disadvantages of using a neural network-based meta-model are contrary to a linear regression model. Neural networks can describe non-linear relationships between input and output variables. Due to the connected structure of neural networks, the model can capture interactions between input variables (an intuitive example is an interaction between sun azimuth and sun altitude) and are generally more capable.

On the other hand, one could argue that using a neural network as a meta-model shifts the interpretability problem from the perception function towards the meta-model. Furthermore, fitting a neural network requires several hyper-parameters, such as architecture, learning rate, or training iterations.

Gaussian Process Regression

Gaussian process regression (GPR), sometimes referred to as *Kriging*, differs significantly from the other two meta-models. In contrast to deterministic meta-models such as linear regression or neural networks, a Gaussian process is a probabilistic model [92, 156]. Additionally to a prediction itself, GPR generates an uncertainty estimation, indicating confidence in the prediction. A detailed description of GPR is provided in Appx. B.

A Gaussian process is a special form of a stochastic process

$$\{X_t | t \in \mathcal{X}\}, \tag{4.33}$$

with X_t being random variables indexed over the index set $\mathcal{X}$. The index set can be continuous or discrete.

A realization of a stochastic process is defined as the sequence of values that results when drawing from the distribution of the random variable of the stochastic process.

Then a realization can be described by a function

$$y = f(t), \tag{4.34}$$

with $f(t)$ being the outcome of random variable X_t.

A stochastic process $\{X_t | t \in \mathcal{X}\}$ is called a Gaussian process if and only if every finite selection of n random variables from $\{X_t | t \in \mathcal{X}\}$ follows a n-dimensional normal distribution. Hence, for $\{t_1, ..., t_n\} \subset \mathcal{X}$, the distribution $X = (X_{t_1}, ..., X_{t_n})$ follows a n-dimensional normal distribution:

$$X \sim \mathcal{N}(\mu, \Sigma), \tag{4.35}$$

with $\mu \in \mathbb{R}$ being the distributions mean vector and $\Sigma \in \mathbb{R}^n$ being the distributions covariance matrix.

Similar to the other regression models, using Gaussian processes for regression tasks requires assumptions on the data distribution. A stochastic process can be interpreted as a distribution of functions. For Gaussian processes, this distribution is some n-dimensional normal distribution. The distribution of realizations is then defined by

$$f(X) \sim \mathcal{N}(m(X), k(X, X)), \tag{4.36}$$

with $X = \{x_1, ..., x_n\}$ being the functions input domain, $m(X)$ being the distributions mean vector function, and $k(X, X)$ being the distributions kernel function that generates the covariance matrix. The mean function is defined as the expected value of a realization at the position x:

$$m(x) = \mathbb{E}[f(x)]. \tag{4.37}$$

This expected value denotes the prediction that the Gaussian process returns for a given point in the input space and can be interpreted analogously to the output of a linear regression model or a neural network.

The kernel function returns the covariance between the realization at two positions x and x'. The covariance matrix is then calculated element-wise with

$$k(x, x') = \mathbb{E}[(f(x) - m(x))(f(x') - m(x'))]. \tag{4.38}$$

The purpose of the kernel function is to describe the connection between the different positions of a realization function along the input axis. Furthermore, the kernel $k(x, x)$ denotes the point-wise variance of the Gaussian process and the uncertainty related to the corresponding prediction $m(x)$.

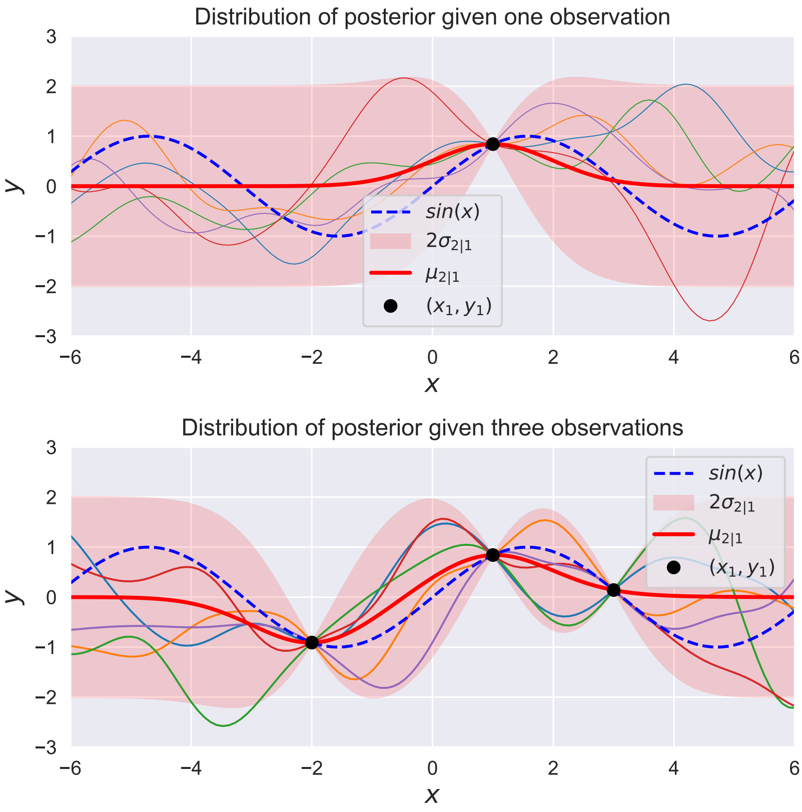


Figure 4.18: Posterior distribution of Gaussian process with $m(x) = 0$ and RBF kernel given one and three real observations from the underlying function $f(x) = sin(x)$. Additionally, five samples from each posterior are drawn and plotted.

5 Implementing proposed pipeline for fault detection

5.1 Overview

This chapter presents the implementation of a testing pipeline for an object detector using synthetic image data. The modules in the overview Fig. 4.8 must be filled with concrete software tools to evaluate the pipeline. The experiments are performed on a pedestrian detector network. First, image data is generated that stresses the pedestrian detector and causes failures in the prediction either by provoking false positives or false negatives. CARLA, a state-of-the-art simulation software for automated driving tasks, is used for image generation. The data generation process is structured using different functional scenarios based on the European New Car Assessment Program (Euro NCAP) framework for pedestrian test scenarios for Autonomous Emergency Braking (AEB) systems [8]. Each functional scenario displays a driving situation where an automated vehicle is confronted with a pedestrian intersecting the ego vehicle's path. Scenario variables of the logical scenario space span a parameterized scenario space for each base context that can be varied. Finally, the data generation is triggered for each concrete scenario out of the logical scenario space to generate a closed-loop test run for a basic AEB system. The synthetic image is fed to a custom-trained YOLOv5 [157] object detector trained to detect pedestrians exclusively. The vehicle activates a brake when the object detector correctly detects the pedestrian. The AEB controller aims to evaluate whether the object detector can detect the pedestrian early enough to avoid a collision in the concrete scenario. A custom KPI measure was introduced, based on the original Time-to-Collision (TTC) measure, called *adjusted Time-to-Collision* (aTTC), is introduced to quantify the performance. Then the sampling strategies are implemented to control the data generation process and iteratively select the concrete scenarios for the next round of data generation. This process is displayed in Fig. 5.1.

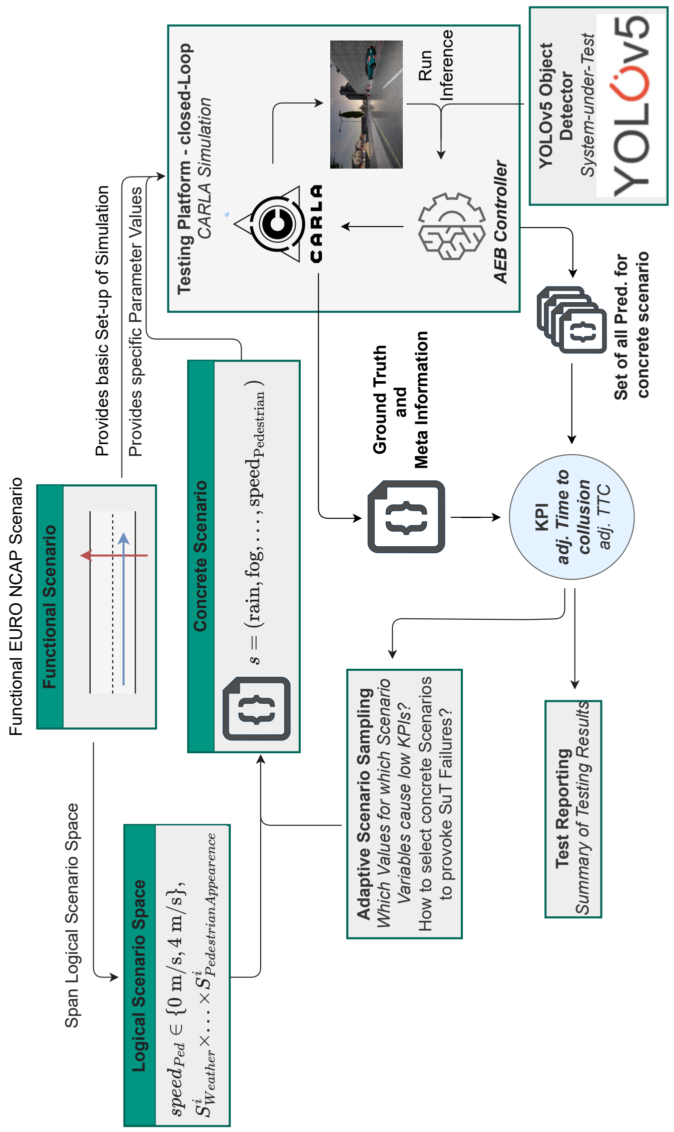

Functional EURO NCAP Scenario
Functional Scenario
Provides basic Set-up of Simulation
Provides specific Parameter Values
Span Logical Scenario Space
Logical Scenario Space
$speed_{Ped} \in \{0\ \mathrm{m/s}, 4\ \mathrm{m/s}\}$,
$S^i_{Weather} \times \ldots \times S^i_{PedestrianAppearence}$
Concrete Scenario
$s = (\mathrm{rain}, \mathrm{fog}, \ldots, \mathrm{speed}_{\mathrm{Pedestrian}})$
Testing Platform - closed-Loop
CARLA Simulation
CARLA
Run Inference
AEB Controller
YOLOv5 Object Detector
System-under-Test
YOLOv5
Ground Truth and Meta Information
Set of all Pred. for concrete scenario
KPI
adj. Time to collusion
adj. TTC
Adaptive Scenario Sampling
Which Values for which Scenario Variables cause low KPIs?
How to select concrete Scenarios to provoke SuT Failures?
Test Reporting
Summary of Testing Results

Figure 5.1: Experimental pipeline to evaluate the proposed testing strategies. Definition and structure follows Fig. 4.8 and equations in Sec. 4.2.

5.2 Designing a scenario space for adaptive sampling

Simulation platform

The experiments of this dissertation make use of CARLA(Car learning to Act) [131] - Release 0.9.14 [158]. CARLA is an open-source driving simulator developed to virtualize and support all development stages of automated driving functions from prototyping to validation. CARLA is an established simulation tool, used in a high number of recent work [159].

CARLA provides various layouts and blueprints for traffic assets, including buildings, vehicles, pedestrians, vegetation, and street signs. These assets can be assembled and spawned to generate on-demand traffic scenarios. CARLA utilizes the free-to-use Unreal Engine 4 (UE4) [160] for high-quality rendering and realistic physics simulation. In addition, an easy-to-use Python API can control CARLA. The usage is designed as a server-client system in which the client sends commands to the simulation server and receives the sensor output. The relevant sensor types for testing an object detector are the RGB camera and the instance segmentation sensor.

There are also pre-built maps that can be used for simulation. These maps depict various driving environments in great detail and can be navigated by traffic participants freely.

Functional scenario set-up for pedestrian detection

As shown in Fig. 5.1, each simulation is set up on a functional scenario. The functional scenario defines the scene and the objects of the scenario space while being able to provide further context information, such as weather conditions. For the experiments of a pedestrian detection network, eight functional scenarios are assembled, all derived from expert knowledge rather than data-driven approaches. The basic interactions between an ego vehicle and a jaywalking/crossing pedestrian are derived from the Euro NCAP scenarios for AEB systems for adult pedestrians [8]. The eight fundamental interactions are displayed in Fig. 5.2. A functional scenario is defined for each interaction, describing the vehicles' and pedestrians' actions and defining additional vehicles and their actions. Note that the functional scenario description for this implementation mainly focuses on layer 3 and layer 4 of the scenario model [139]. In the Euro NCAP framework, nearside refers to the side of the road the ego vehicle is driving on (right side for right-hand traffic), while farside refers to the opposite side of the road (left side for left-hand traffic). See the description of layers in Fig. 4.5.

Layer 1+2+3: Each functional scenario is defined for a separate fixed road segment with neither road geometry nor infrastructure being changed during sampling.

Layer 4: The traffic participants and their paths are predefined for each functional scenario. The core of each functional scenario is the interaction between a pedestrian and the ego vehicle, which is different for each scenario. These interactions are shown and described in Fig. 5.2 and Tab. 5.1.

Layer 5: Weather and environmental effects have significant impact on perception systems' performance. The available factors of rain, fog, and sun position are defined in the logical scenario space and not restricted by the functional scenario description.

Layer 6: As the AEB controller does not rely on digital information, such as digital map data and V2X communication, the testing process ignores this layer.

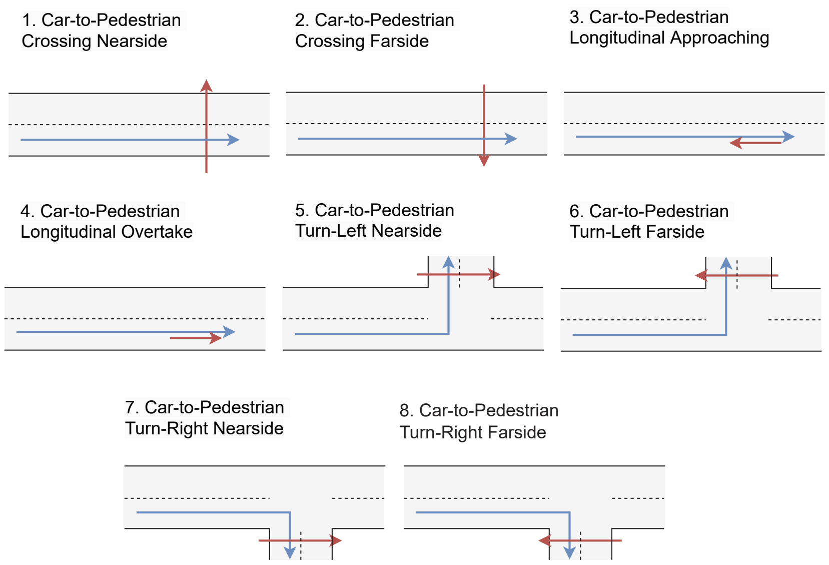
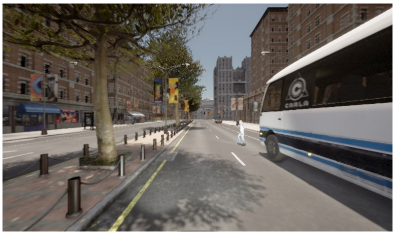

1. Car-to-Pedestrian Crossing Nearside
2. Car-to-Pedestrian Crossing Farside
3. Car-to-Pedestrian Longitudinal Approaching
4. Car-to-Pedestrian Longitudinal Overtake
5. Car-to-Pedestrian Turn-Left Nearside
6. Car-to-Pedestrian Turn-Left Farside
7. Car-to-Pedestrian Turn-Right Nearside
8. Car-to-Pedestrian Turn-Right Farside

Figure 5.2: Defined functional scenarios, derived from the pedestrian-vehicle interactions from the Euro-NCAP scenarios for AEB systems for adult pedestrians [8].

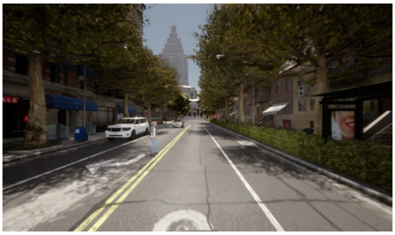
$\mathcal{F}_1$	Urban environment with bus station. Ego vehicle is passing a parking bus. A pedestrian is located behind the bus. The pedestrian disregards the traffic rules and crosses the street from the nearside in front of the vehicle.
$\mathcal{F}_2$	Urban environment with straight road with intensive vegetation on both sides and bus station. Ego vehicle is passing the station with an approaching passenger car on the opposite side of the road. A pedestrian disregards the traffic rules and crosses the street from the farside in front of the vehicle.
$\mathcal{F}_3$	Urban environment with bus station. The ego vehicle is passing the station with a vehicle leading on the left lane of the track. A pedestrian is approaching the station on the ego vehicle's lane.
$\mathcal{F}_4$	Urban environment with vegetation on the right side of the road. The ego vehicle is following the road with a vehicle leading on the left lane of the track. The vehicle is overtaking a pedestrian in the ego vehicle's lane.
$\mathcal{F}_5$	Urban environment with T-junction. The ego vehicle turns left with a leading vehicle on the new lane. A pedestrian crosses the new track from the nearside on a regular crosswalk.
$\mathcal{F}_6$	Urban environment with T-junction. The ego vehicle turns left with an approaching vehicle turning right from the opposite lane. A pedestrian crosses the new track from the farside on a regular crosswalk.
$\mathcal{F}_7$	Urban environment with T-junction. The ego vehicle turns right with an approaching passenger car crossing the junction and a stationary passenger car in the left lane of the ego vehicle's new driving track. A pedestrian disregards the traffic rules and crosses the street from the nearside in front of the vehicle to board the stationary passenger car.
$\mathcal{F}_8$	Urban environment with T-junction. The ego vehicle turns right with an approaching passenger car waiting on the new track and an impeding vehicle on the new track. A pedestrian crosses the new track from the farside on a regular crosswalk.

Table 5.1: Defined functional scenarios, derived from the pedestrian-vehicle interactions from the Euro-NCAP scenarios for AEB systems [8].

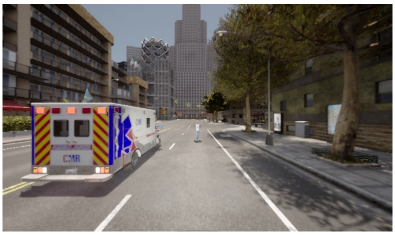
a) S1_c2p_crossing_nearside



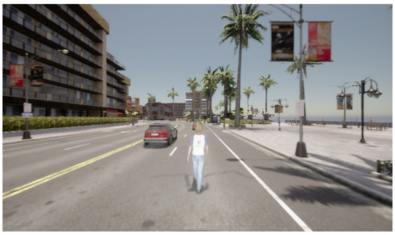
b) S2_c2p_crossing_farside



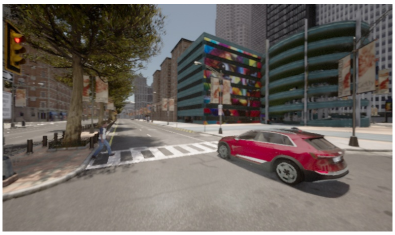
c) S3_c2p_longitudinal_approaching



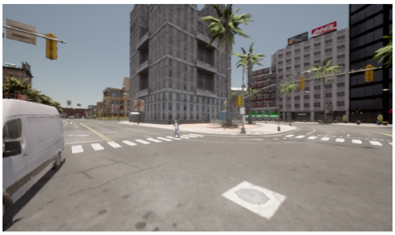
d) S4_c2p_longitudinal_overtake



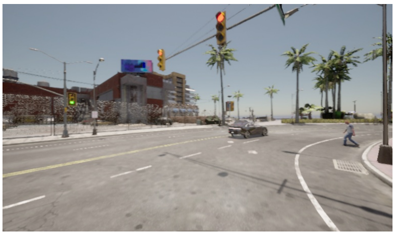
e) S5_c2p_turn_left_nearside



f) S6_c2p_turn_left_farside



g) S7_c2p_turn_right_nearside



h) S8_c2p_turn_right_farside

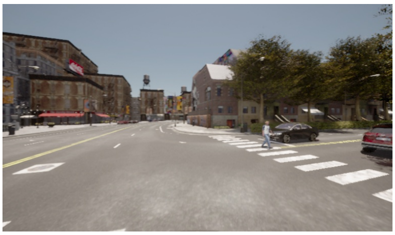


Figure 5.3: Functional scenarios as implemented in the experiments.

Logical scenario space

The following sub-sections detail the sampling process that can control the different logical scenario variables that are parameterized and the exact scenario space results from the options that the functional scenario offers.

Position and Velocity of Pedestrian

Positional information for the traffic participants. Partial occlusion of objects affects the detection performance and may cause system failures through failed or late detection. For each functional scenario, there is a $S_{PedShift}$ variable, which is an offset for the pedestrian's initial position and a $S_{PedSpeed}$ variable that sets the movement speed of the pedestrian with a range of up to ten kilometers per hour.

Pedestrian Blueprint

The different pedestrian blueprints change the appearance of a pedestrian in the image. Various pedestrian blueprints are available for traffic simulation. These objects differ in many aspects, including size, body shape, clothing, and skin color, allowing to inspect the influence of these characteristics on pedestrian detection performance.

Pedestrian Animation Behavior

As the behavior and movement of pedestrians significantly influence automated driving systems [161], they must be included in the simulation. Therefore, there is a set of six animations that are assigned to the pedestrians' movement. The skeleton alignments during the animations are generated by virtualizing real world recordings via the open-source website Deepmotion [162] and then overlaid with the skeleton inside the Carla simulation [4]. The pedestrian's speed is then used to adapt the animation speed.

The set of pedestrian animations is $S_{\text{Ped. Animation}}$ with the appearance being self-explanatory.

$$S_{\text{Ped. Animation}} = \{\text{walking}, \text{texting}, \text{crouching}, \text{limping}, \text{drunk}, \text{stumble}\} \tag{5.1}$$

Weather Variables

$\mathcal{S}_{Sun-Position}$ **(Numerical)** The sun parameters azimuth and elevation set the sun's position in the simulation and the daytime. With decreasing elevation, the sun's intensity decreases, creating a sunset. If decreased further, there is a seamless transition into the night. Different daytimes can be seen in Fig. 5.4.

$\mathcal{S}_{Fog}$ **(Numerical)** The overall thickness of the fog and the visual range can be changed.

$\mathcal{S}_{Precipitation}$ **(Numerical)** The precipitation parameter refers to the severity of rain in the form of small streaks in the images and to how puddles cover the road. Ponding affects how light is reflected on the street.

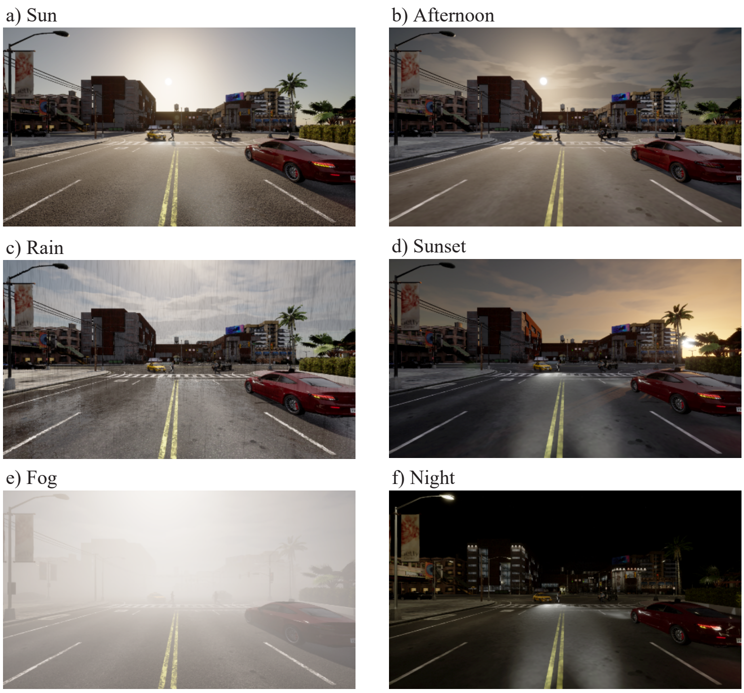


Figure 5.4: Different weather and daytime effects by the simulation. These effects can have major implications on the detection performance of perception functions.

5.3 Autonomous emergency braking controller

With a custom-trained pedestrian detection model at its core, an AEB controller is implemented to test the model's ability to avoid a pedestrian-related collision. An ego vehicle is simulated in the selected functional scenario and equipped with an RGB-camera sensor to generate image data with corresponding ground truth. With a pre-defined velocity, the vehicle follows a path with a pedestrian crossing during the simulation. A visualization is displayed in Fig. 5.5.

Iteratively, the pedestrian detector is applied to the image data. If no pedestrian is detected, the controller will follow the original path. If there is a true positive detection, the brake is activated with the braking intensity equaling the detection's confidence. Otherwise, the controller aims to hold the car's speed constant. The braking signal does not influence the steering command.

While an operational AEB controller does not have access to the ground truth information of real traffic, this avoids correct braking caused by a false positive while the pedestrian is not detected. Thus, false negatives during the simulation run cause delayed braking, while false positives do not impact the simulation directly. However, the false positives are indirectly accounted for during the KPI calculation, as the number of false positives lowers the test result, detailed in Sec. 5.5. Thus, during the simulation run, the AEB controller evaluates whether the pedestrian detector can detect the object early enough to avoid collision with full braking.

5.4 System-under-Test: YOLOv5

Architecture

As mentioned in Sec. 2.5, the *you only look once* (YOLO) framework [85] is a popular concept for object detection due to a good balance between detection accuracy and speed. Thus, a YOLO network is used as a system-under-test for the experiments and forms the core of the AEB controller. Running with low latencies is a key requirement for perception functions used in automated vehicles. Since its introduction in 2016, the original YOLO (sometimes called YOLOv1) framework has been optimized multiple times. The YOLO versions with highest academic impact are

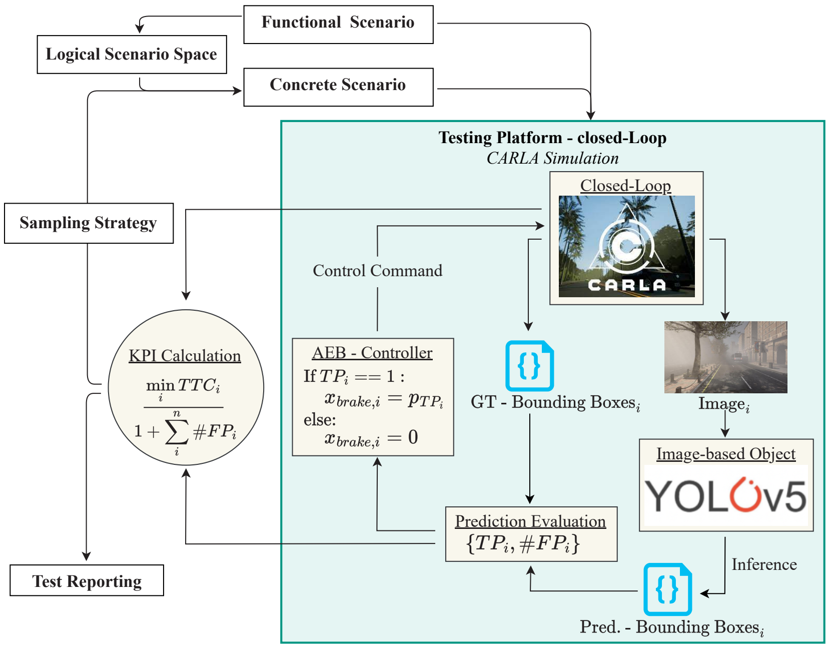


Figure 5.5: Simulation framework for SuT evaluation. Carla, the simulation software generates image data with corresponding ground truth. Depending on whether pedestrians are detected, the simulated vehicle activates the brake to avoid a collision.

YOLO (2016) [85], YOLOv2&YOLO9000 (2017) [163], YOLOv3 (2018) [164], and YOLOv4 (2020) [157].

The general concept behind YOLO is already described in Sec. 2.5. The YOLO method generally refers to an end-to-end trainable neural network that takes an input and generates a grid-like output of the image. Then, for each grid cell, the network predicts various values that indicate whether there is an object with its center in that grid cell. These values include the probability of an object being present, the position and size of the potential object's bounding box, and the class probabilities. A visualization of this grid is displayed in Fig. 5.6.

A more in-depth description of YOLOv5 [165] is provided, although black-box testing is unaffected by an object detector's architecture.

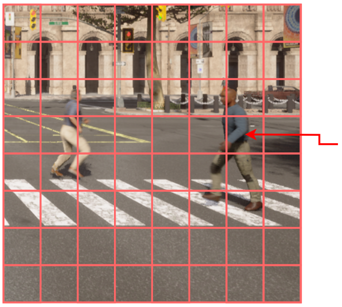
Backbone Most of today's neural networks resort to so-called backbone (or feature extractor) networks for initial feature extraction. The backbone transforms the input image into a standardized feature space that serves as input for the following network layers and can be interpreted as an encoder [166]. The advantage of using established backbones is that the user can apply an initial encoding of the input with a network that has been proven useful for the task. Suppose the backbone is already pre-trained to identify important image features for a certain application, such as pedestrian detection or cancer cell segmentation. The following neural network stages can be learned more efficiently in that case. YOLOv5 is built on a backbone called CSPDarknet53 [167].

Neck A neck is used if the backbone generates feature maps for different sizes (low- and high-dimensional features). The neck is an aggregation logic that collects the data from different sizes and passes it on to the prediction network. YOLOv5 utilizes Spatial Pyramid Pooling (SPP) and Path Aggregation Network (PANet). SPP pools from different output layers of the backbone network to generate a fixed-length feature vector that is then passed on. PANet regulates how the different data from the SPP and feature maps from the backbone are combined.

Head The head of an object detector generates the actual prediction of the whole model. YOLOv5 takes over the established head of the previous version, YOLOv3, which follows the original YOLO logic above, although there are extensions. One significant addition was introduced with YOLOv2: Anchor boxes - while the original YOLO was only capable of predicting one object for any given grid cell, the newer model hold for each grid cell several predefined bounding boxes. The probability of containing an object with the potential offset is predicted for each candidate bounding box.).

Dealing with the domain gap

There are limitations to simulative testing. The most significant is the domain gap (see Sec. 4.1). In addition, while synthetic and real images appear similar to a human observer, the behavior of neural networks may differ. Therefore, a



$$\begin{pmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ \dots \\ c_n \end{pmatrix} \begin{array}{l} = \text{Is there an object with center in the cell?} \\ = \text{x-position of object center within cell} \\ = \text{y-position of object center within cell} \\ = \text{Height of potential bounding box} \\ = \text{Width of potential bounding box} \\ = \text{Prob of potential object being from class 1} \\ = \dots \\ = \text{Prob of potential object being from class n} \end{array}$$

Figure 5.6: Visualization of YOLO output. The meta information of a potential object with its center in that grid cell is predicted for each grid cell. This process is done for different grid sizes.

model that achieves good results on virtual images may not be reliable for real images [118].

This can be addressed using a post-processing measure called domain adaptation image data [168]. There are methods to change an image's appearance from a source domain to a target domain. For example, using a domain adaptation model, images from the source domain "virtual data" can be translated into the target domain "real data". Another approach is to improve image rendering software to make the simulation more photo-realistic. For instance, the virtual KITTI 2 data set improves the virtual KITTI data set and offers improved graphic quality [50].

The experiments are performed solely in the virtual domain ignoring the domain gap. The object detector is trained to detect pedestrians in the virtual domain and tested in the virtual domain. Hence, training data from the simulation framework has to be generated to adjust the object detector for this domain. This training data is generated randomly to not manually bias the object detector, as it would be with real world recorded data.

By randomly spawning vehicles and pedestrians on the map and steering them to random positions, object tracks for the training data can be generated. First, a data set for development was generated using these functionalities and randomization of the weather parameters. Next, 3,000 images were assembled for each available map using this random data generation. Afterward, the data

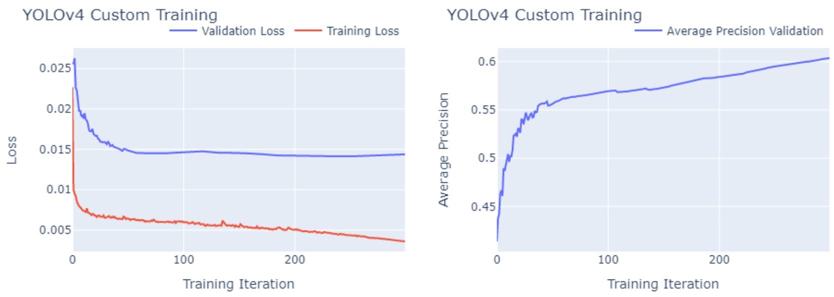


Figure 5.7: YOLOv5 custom training process. Training on randomly generated image data was performed to adapt the object detector for pedestrian detection on synthetic image data.

was split into traditional training/validation sets with a ratio of 90% to 10% and used to train the object detector.

5.5 Key performance indicator: adjusted Time-to-Collision

To measure the performance of the object detector, a KPI metric must be defined for the AEB controller. Usually, the standard benchmark measure for object detectors is the average precision defined in Sec. 2.5. However, the average precision is usually used to evaluate an object detector's performance for a large set of images since the interpolation of the ROC curve requires many predictions and ground truth objects. Hence, the average precision may not be helpful for such an application. Furthermore, the average precision is mainly safety-relevant information, such as whether a detected object is on a collision path with a vehicle or a non-critical part of the road geometry.

An established way to measure the criticality of driving situations is the *Time-to-Collision (TTC)*-measure. In short, the TTC measures the time left until two traffic objects on a collision path collide. For TTC

$$TTC = \begin{cases} 0 & \text{if collision happened} \\ \frac{||x_{CP} - x_{ego}|| - 2.5}{v_{ego}} & \text{else,} \end{cases} \tag{5.2}$$

with x_{CP} being the coordinates of collision point, x_{ego} being the coordinates of the ego, and v_{ego} being the speed of the ego vehicle. The term -2.5 adjusts for the size of ego-vehicle since x_{ego} refers to the vehicle's center. There is a contact if the distance between a pedestrian's and vehicle's coordinate is 2.5.

To quantify the SuT performance during the entire simulation run, the minimal TTC is selected and adjusted for false positives. This is done due to the omission of false positives during the simulation. The adjusted TTC measure for a simulation run with n time steps is defined by

$$\text{adj. TTC} = \frac{\min_i TTC_i}{1 + \sum_i^n \#FP_i}, \tag{5.3}$$

with TTC_i being the TTC value at time step i and $\#FP_i$ being the amount of false positives at time step i.

5.6 Visualization of experimental process

Displaying a dynamic simulation process in the form of a single graphic is hardly possible. A format has been introduced for visualization of testing results, as shown in Fig. 5.8.

a) Single Frame from Simulation A single image is selected from the simulation and displayed with the ground truth bounding box of a pedestrian in blue and the predicted bounding box with prediction confidence in red.

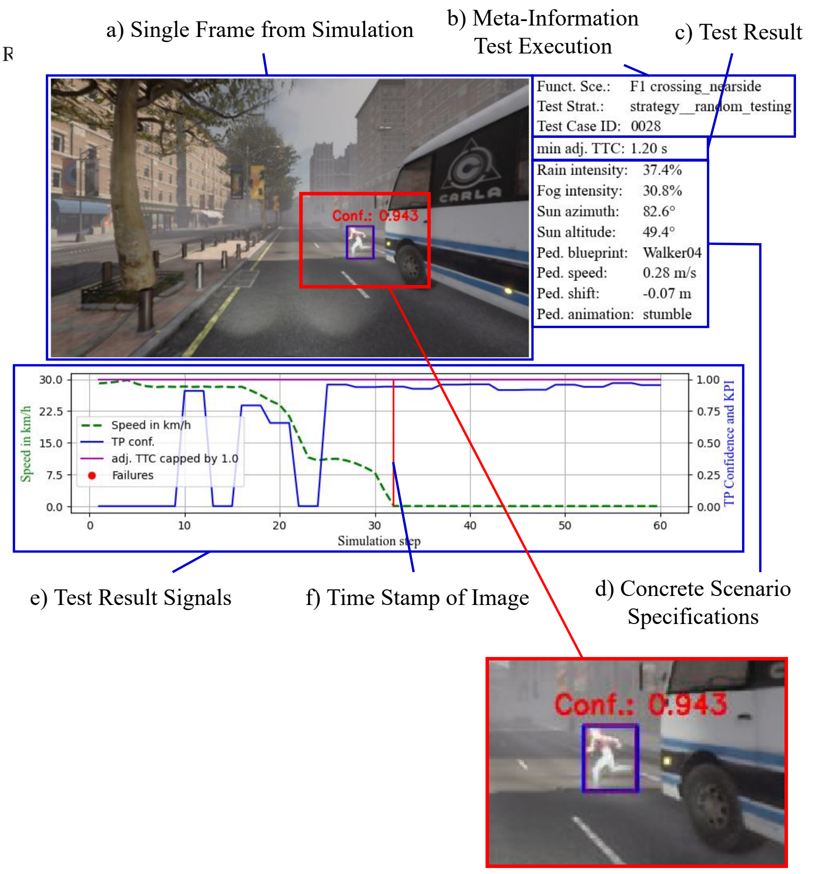
b) Meta-Information of Test Execution Contains functional scenario ID, test strategy used during test execution, and the execution ID during the test execution. The execution ID refers to the position during the simulation process: 28th tested concrete scenario during random testing on functional scenario $\mathcal{F}_1$.

c) Test Result Displays the resulting minimal adj. TTC during the test execution.

d) Concrete Scenario Specifications Contains the concrete scenario values of $s \in \mathcal{S}_i$ for each scenario variable during the test execution.

e) Test Result Signals Displays the course of key test execution signals during the full simulation. The blue line displays the prediction confidence of the true positive during each step with a zero value if there is no detection. The green line displays the speed of the ego vehicle. The purple line shows the adj. TTC during each time step of the test execution with a red marker if the value goes below the threshold $t = 0.1$.

f) Time Stamp of Image The red line shows the time step during the simulation from which the frame in a) is drawn.



Funct. Sce.:	F1 crossing_nearside
Test Strat.:	strategy__random_testing
Test Case ID:	0028
min adj. TTC:	1.20 s
Rain intensity:	37.4%
Fog intensity:	30.8%
Sun azimuth:	82.6°
Sun altitude:	49.4°
Ped. blueprint:	Walker04
Ped. speed:	0.28 m/s
Ped. shift:	-0.07 m
Ped. animation:	stumble

Figure 5.8: Visualization of execution run of concrete scenario. The parts of the image are explained in Sec. 5.6.

6 Results of pipeline for fault detection

6.1 Process set-up and Scenario variable selection

Utilizing the presented methods from Sec. 4.4.3 and tools from Sec. 5.1, an experimental implementation of the concepts in Sec. 3.4.3 is performed. Tab. 6.1 shows the implemented testing experiments. On each of the eight functional scenarios $\mathcal{F}_1, ..., \mathcal{F}_8$, all six test strategies are run with a given "test budget" of 1000 test executions, which is chosen because, during the experiments, the improvement in failure detection for adaptive testing has shown to bottom out after around 800-1,000.

The adaptive test strategies are executed batch-wise with a batch size of $n_b = 20$. While the batch size does not impact random testing, t-wise testing, or evolutionary testing, for meta-model-based testing, the batch size determines how many data points are added to the meta-model training data before re-training.

The failure threshold t for defining the execution of a concrete scenario as *failure* is chosen as $t = 0.1$, hence

$$adj.TTC \leq 0.1 \;\Leftrightarrow \text{failure detected.} \tag{6.1}$$

Overall, that implies that the number of test executions is calculated with

$$\text{test budget} \times \#\ \text{strategies} \times \#\ \text{functional scenarios} = 1000 \times 6 \times 8 = 48000.$$

Each concrete scenario took about 25-35 seconds for test execution. For 1000 test executions this adds up to 6.9-9.7 hours per test strategy and functional scenario and 333.3-466.6 hours for all experiments, excluding the test strategy overhead time for model training.

Other parameters for the simulation process are defined in Sec. 4.3 with ϵ (= 10% * number of logical scenario dimensions) being the size of the metric ball and $t_\omega = 5$ being the threshold for the relevance score. These parameters influence the size of the failure regions that prevent oversampling. Analysis of several simulation runs showed that this selection yields a good balance between avoiding oversampling and not restricting the selection process too much.

1	Random Testing	Random sampling of test cases using independent discrete or continuous uniform distributions for each scenario variable.
2	Pair-wise Testing	Test case sampling based on t-wise testing with $t = 2$ (see Sec. 4.4.1, Fig. 4.13). The test plan is initialized at the beginning of the testing process and then executed all at once. Pair-wise testing plan contains 986 test cases which were filled up to 1,000 by adding random test cases.
3	Evolutionary Testing	Iterative test case selection using evolutionary testing (see Sec. 4.4.2, Fig. 4.13). The test selection is executed with a batch size of $n_b = 20$. After each executed batch, the evolutionary sampling is performed.
4	LR Testing	Iterative test case selection using meta-model based testing with linear regression (see Sec. 4.4.3, Sec. 4.16, and Fig. 4.14). The test selection is executed with a batch size of $n_b = 20$.
5	NNR Testing	Analogous to LR Testing, while using neural network regression.
6	GPR Testing	Analogous to LR Testing, while using Gaussian process regression.

Table 6.1: Implemented testing strategies.

Strategy evaluation

For evaluating of effectiveness and efficiency of the test strategies, a range of measures has been defined in Sec. 4.3. In the following, the practical implementation of these measures is outlined.

Failures detected

The most straightforward measure is the count of detected failures, which is given by the number of test cases in which the adj. TTC falls below threshold t. Since all experiments are carried out with the same amount of test executions, this measure is comparable. However, it does not consider the efficiency of the test strategies during different phases of the testing process.

Saved cost of testing

Recall from Def. 4.17 the definition of the C-measure which measures saved costs for test executions to detect n failures through an intelligent test strategy relative to random testing. Formally, the C-measure is expressed by Eq. 4.20:

$$C_n\text{-}Measure = 1 - \frac{F_n^s}{F_n^r},$$

where F_n^s is the F-measure (Eq. 4.19) of the test strategy for n failures and F_n^r is the F-measure for random concrete scenario selection for n failures. For practical implementation, n is chosen as the number of failures detected within the test budget.

To calculate the $C_n - Measure$ (saved costs of test executions of adaptive strategy compared to random testing) for an intelligent test strategy, at least one failure has to be detected by random testing, with the test strategy detecting at least as many failures, as random testing. Since the first condition is not fulfilled for $\mathcal{F}_1$ and $\mathcal{F}_8$, a work-around has to be found without continuing random testing as this does not guarantee detecting any failure. The work-

Table 6.2: Discretization of continuous variables. Including the categorical variables Animation and Pedestrian Blueprint, there are 8102 2-wise combinations.

Variable Name	min. Value	Step Size	max. Value
Rain Intensity	0	10	100
Fog Intensity	0	10	100
Sun Azimuth	0	30	360
Sun Altitude	-20	10	90
Pedestrian Shift	-0.5	0.1	0.5
Pedestrian Speed	0	0.25	3

around was done by assuming that for *the next test execution* a failure is found, hence the transformed formula for Eq. 4.20 is given by

$$C_1\textit{-Measure} \quad = \quad 1 - \frac{F_1^s}{F^r} \quad = \quad 1 - \frac{F_1^s}{1001}. \qquad (6.2)$$

Combinatorial coverage

Combinatorial coverage is derived from the idea of combinatorial testing and t-wise coverage (compare to Sec. 4.4.1). First, continuous variables are discretized by defining intervals in which the values can fall. Then, for the discrete testing space, all possible $2 -$ wise combinations are generated (compare to Fig. 4.12). Overall, including the categorical variables, there are 8102 2-wise combinations. Finally, the test strategy is evaluated by the fraction of $2 -$ wise combinations covered by the tested concrete scenarios relative to the total $2 -$ wise combinations. Furthermore, the diversification of detected failures can be measured similarly to evaluate whether the test strategy detected a wide variety of failure causes or just a high number of slight variations of a few failures.

6.2 Random sampling and pair-wise testing

Applying random sampling establishes a baseline that the result of sampling strategies can be held against. The next batch of concrete scenarios is randomly selected at each iteration of random sampling without any particular strategy. This should resemble the stochastic data collection that occurs in real traffic and a missing test strategy.

The results for all functional scenarios using random testing are displayed in the scatter plots Fig. 6.1 and Fig. 6.2. Each point represents one test execution, with the y-axis representing the resulting adj. TTC of the test execution. The position of the x-axis displays the position in the total testing process in %, with the first concrete scenario execution plotted on the left and the last concrete scenario execution on the right. Additionally, the plot shows the average adj. TTC for each testing batch.

The plots visualize one of the initial hypotheses of the dissertation (see Sec. 1.3). Failures are sparsely distributed, and to rely on random data collection to generate failures requires significant computation resources. Especially the results of functional scenarios $\mathcal{F}_4$, $\mathcal{F}_5$, $\mathcal{F}_6$, and $\mathcal{F}_8$ display this predicament. In these functional scenarios, the random testing strategy did not detect a concrete scenario in which the object detector could not avoid a collision, formally defined as $adj.\ TTC < 0.1$. Each functional scenario has its own ceiling for adj. TTC, depending on the scenario set-up, such as the point when the pedestrian comes into the vehicles' view. Furthermore, for each functional scenario, the fluctuation of adj. TTC varies. This can be attributed to the different difficulties that the functional scenarios have.

Table 6.3: Resulting number of detected failures $\Theta_{1,000}$ for 1,000 test executions using random sampling and pair-wise sampling.

Strategy	$\mathcal{F}_1$	$\mathcal{F}_2$	$\mathcal{F}_3$	$\mathcal{F}_4$	$\mathcal{F}_5$	$\mathcal{F}_6$	$\mathcal{F}_7$	$\mathcal{F}_8$
Random Testing	**1**	**4**	**2**	0	0	0	**6**	0
Pair-wise	0	1	0	**2**	0	**2**	4	0

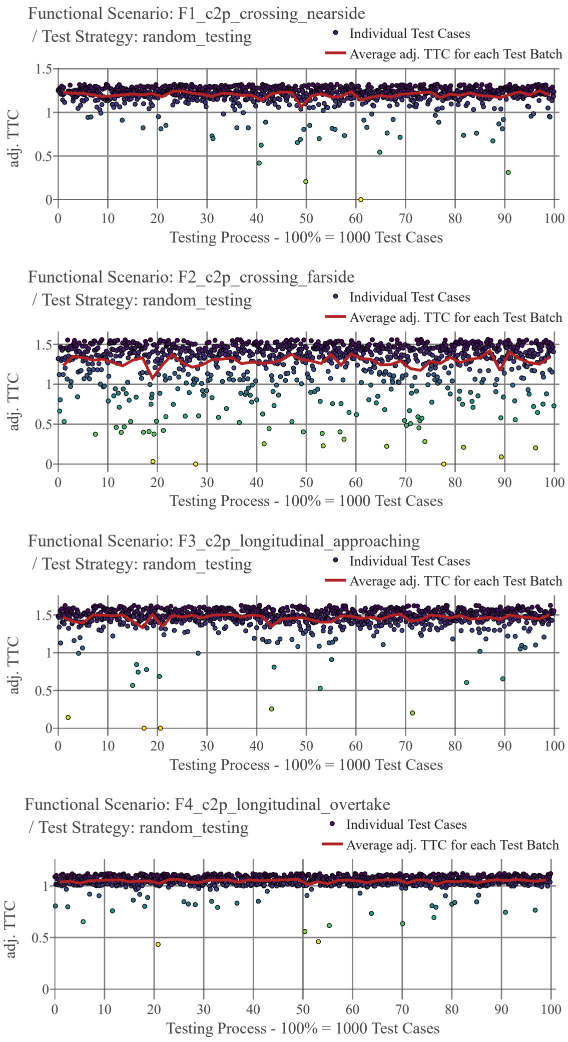


Figure 6.1: Test results using random testing for $\mathcal{F}_1 - \mathcal{F}_4$. Each point represents the result of the simulation execution for a concrete scenario, with the KPI result being on the y-axis and the position within the testing process on the x-axis.

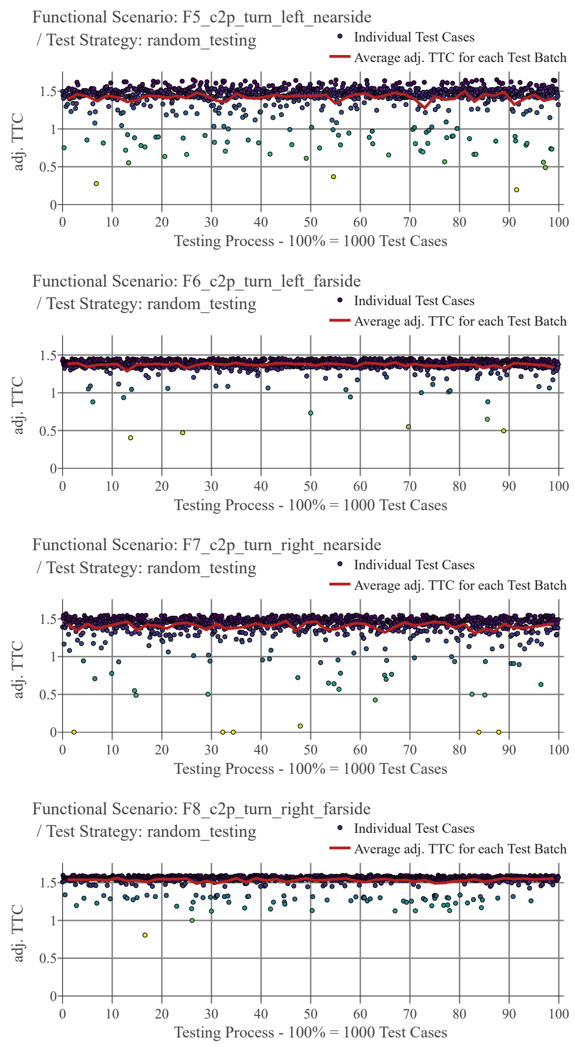


Figure 6.2: Test results using random testing for $\mathcal{F}_5 - \mathcal{F}_8$. Each point represents the result of the simulation execution for a concrete scenario, with the KPI result being on the y-axis and the position within the testing process on the x-axis.

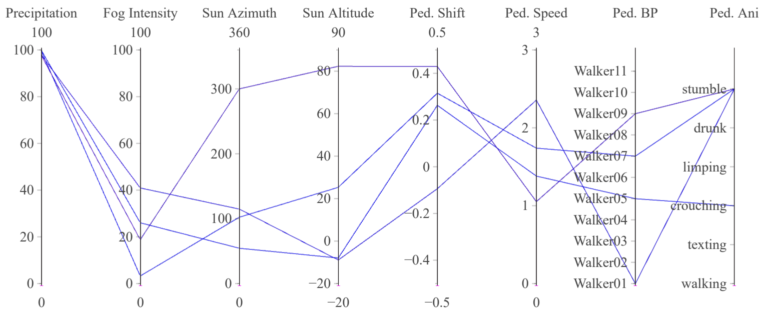
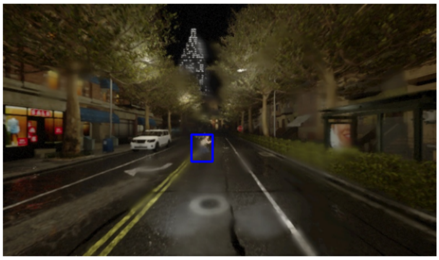


Figure 6.3: Parallel coordinates plot for Random Testing and functional scenario $\mathcal{F}_2$. Each of the 4 lines represent one failure with the position on each scale representing the concrete scenarios value for the respective variable.

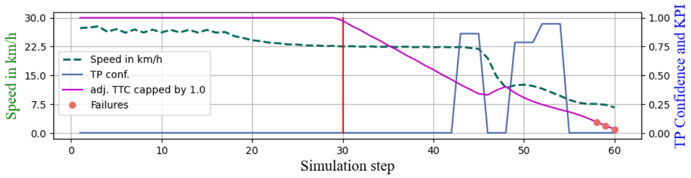
As expected for a test process without any strategy, the average adj. TTC for each batch does not change significantly during the testing process, ignoring the noise infused by the random sampling.

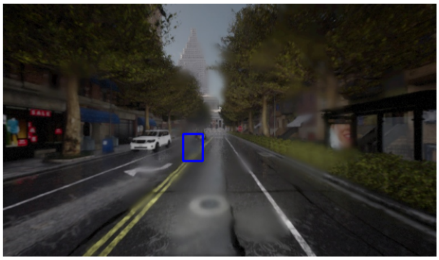
Tab. 6.3 shows the number of failures $\Theta_{1,000}$ detected in 1,000 test executions of random testing and pair-wise testing. As already mentioned, for four functional scenarios - $\mathcal{F}_4$, $\mathcal{F}_5$, $\mathcal{F}_6$, and $\mathcal{F}_8$ - no failure is detected. Compared to random testing, the also non-adaptive pair-wise testing shows no clear improvement towards random testing, hence not supporting previous work that suggests improved failure detection using combinatorial testing [169]. While detecting failures in $\mathcal{F}_4$ and $\mathcal{F}_6$, pair-wise testing fails to detect any failure in $\mathcal{F}_1$ and $\mathcal{F}_3$.

Fig. 6.3 displays a so-called parallel coordinates plot for the failures during random testing in $\mathcal{F}_2$. Each plot line represents the concrete scenario variable values for the failure, hence displaying high dimensional data points in an intuitive 2D diagram. I.e., the plot visualizes how all failures occurred with the rain intensity above 96%, which assists in identifying the blind spots causing failures. Fig. 6.4 shows extracts from two of these failures - one at night and one at day.



Funct. Sce.:	F2 crossing_farside
Test Strat.:	random_testing
Test Case ID:	0191
min adj. TTC:	0.03 s
Rain intensity:	97.9%
Fog intensity:	41.0%
Sun azimuth:	114.9°
Sun altitude:	-8.9°
Ped. blueprint:	Walker01
Ped. speed:	2.36 m/s
Ped. shift:	-0.09 m
Ped. animation:	stumble





Funct. Sce.:	F2 crossing_farside
Test Strat.:	random_testing
Test Case ID:	0777
min adj. TTC:	0.00 s
Rain intensity:	99.6%
Fog intensity:	3.3%
Sun azimuth:	102.4°
Sun altitude:	25.4°
Ped. blueprint:	Walker07
Ped. speed:	1.74 m/s
Ped. shift:	0.31 m
Ped. animation:	stumble

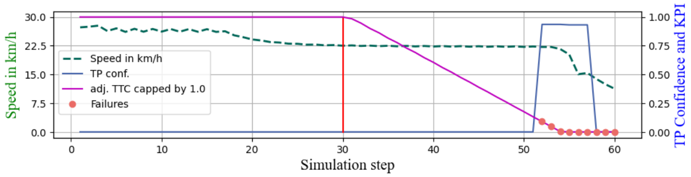


Figure 6.4: Visualization of two failures from random testing on $\mathcal{F}_2$.

6.3 Evaluation of adaptive testing

Failure detection of adaptive testing

Tab. 6.4 displays the number of failures in 1,000 test executions using all sampling strategies, including the adaptive ones. The data highlights the

Table 6.4: Resulting number of detected failures $\Theta_{1,000}$ for 1,000 test executions using different sampling strategies.

Strategy	$\mathcal{F}_1$	$\mathcal{F}_2$	$\mathcal{F}_3$	$\mathcal{F}_4$	$\mathcal{F}_5$	$\mathcal{F}_6$	$\mathcal{F}_7$	$\mathcal{F}_8$
Random Testing	1	4	2	0	0	0	6	0
Pair-wise	0	1	0	2	0	2	4	0
Evolutionary	1	28	6	19	**24**	**91**	118	11
LR	0	36	32	13	4	38	63	7
NNR	**3**	**42**	**38**	**23**	15	38	**159**	**21**
GPR	**3**	40	11	15	12	38	107	8

effectiveness of adaptive test strategies relative to non-adaptive random or pair-wise testing. In all functional scenarios (see. Sec. 4.4), the adaptive strategies outperform the non-adaptive strategies significantly. Comparing the adaptive strategies, evolutionary testing detected the most failures in six out of eight functional scenarios, leading by a high margin in $\mathcal{F}_6$ and $\mathcal{F}_7$.

The three options for the meta-model based testing show different levels of detected failures, with GPR and NNR outperforming the LR in all but functional scenarios. Overall, the comparison between GPR and NNR shows an advantage of NNR based testing. The NNR outperforms the GPR in all but two scenarios and tying in $\mathcal{F}_1$ and $\mathcal{F}_6$.

Overall, the evolutionary testing showed comparable results to meta-model based testing and detected failures in each functional scenario. In $\mathcal{F}_5$ and $\mathcal{F}_6$, the evolutionary sampling outperforms the other models while trailing in all other functional scenarios.

The most striking statistic is the performance of evolutionary testing in $\mathcal{F}_6$ compared to the meta-model based strategies. Here, the number of failures is 2.4 times higher than the GPR, with 91 to 38 failures. This functional scenario serves as an optimal case study for a comparison of the different strategies.

To look deeper into this observation, Fig. 6.5 a) displays the parallel coordinates plot of the evolutionary testing. The visual analysis shows that the detected

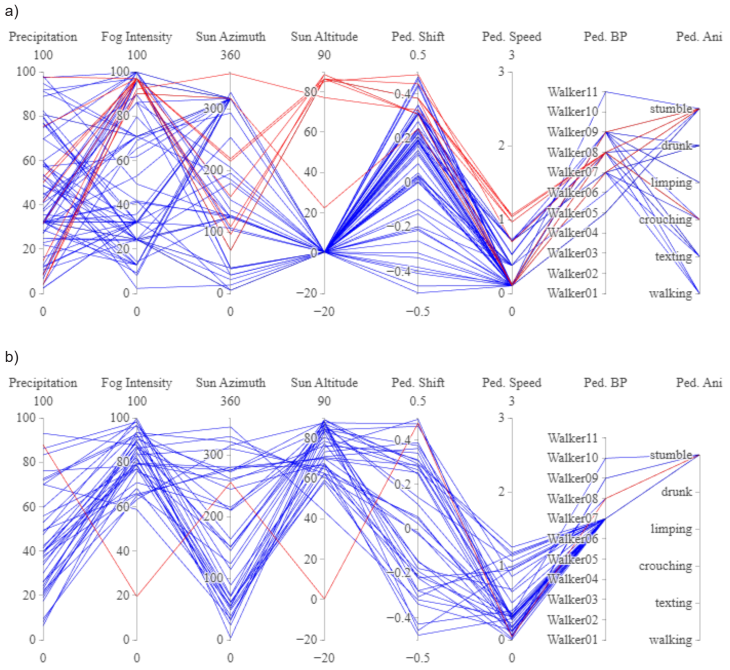
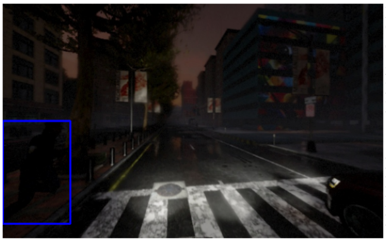


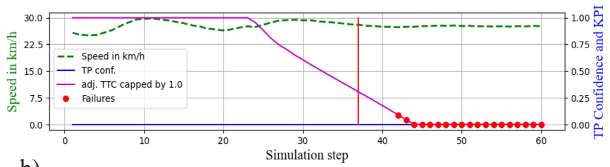
Figure 6.5: a) Parallel coordinates plot for evolutionary testing on functional scenario $\mathcal{F}_6$. Each line represents one of the 91 failures. The 80 of these failures that accord in $S_{\text{Sun altitude}} = 0.47°$ are colored blue, while the failures that differ from this pattern are highlighted in red. b) Parallel coordinates plot for the failures detected by meta-model based testing using GPR on functional scenario $\mathcal{F}_6$. The failure with $S_{\text{Sun altitude}} = 0.28°$ is colored in red.

failures follow a very distinct pattern. Each line represents one of the 91 failures. For visualization, the 80 failures that include $S_{\text{Sun altitude}} = 0.47°$ are colored blue, while the failure that differs from this pattern is displayed in red.

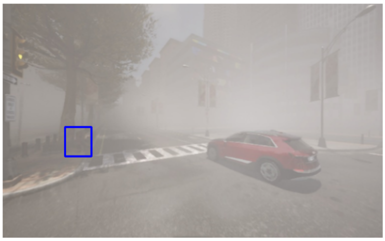
a)



Funct. Sce.:	F6 turn_left_farside
Test Strat.:	evolutionary
Test Case ID:	0846
min adj. TTC:	0.00 s
Rain intensity:	59.6%
Fog intensity:	14.2%
Sun azimuth:	125.4°
Sun altitude:	0.5°
Ped. blueprint:	Walker07
Ped. speed:	0.10 m/s
Ped. shift:	0.19 m
Ped. animation:	stumble



b)



Funct. Sce.:	F6 turn_left_farside
Test Strat.:	meta_gaussian
Test Case ID:	0847
min adj. TTC:	0.00 s
Rain intensity:	38.7%
Fog intensity:	89.1%
Sun azimuth:	144.3°
Sun altitude:	89.1°
Ped. blueprint:	Walker09
Ped. speed:	0.05 m/s
Ped. shift:	0.25 m
Ped. animation:	stumble

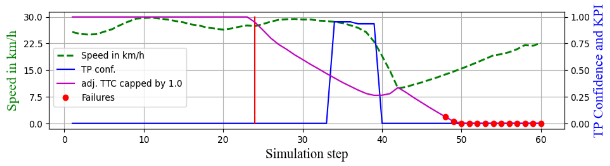


Figure 6.6: Visualization of two test executions during testing in $\mathcal{F}_6$. a) Single failure detected by evolutionary test strategy with $S_{\text{Sun altitude}} = 0.47°$. b) Single failure detected by meta-model based testing using GPR with $S_{\text{Fog intensity}} = 89.1\%$.

Overall, this pattern (colored in blue) shows that the failure source mainly originates from

$$S_{\text{Sun altitude}} = 0.47° \ \wedge \ S_{\text{Ped. speed}} < 0.73\text{m/s}. \tag{6.3}$$

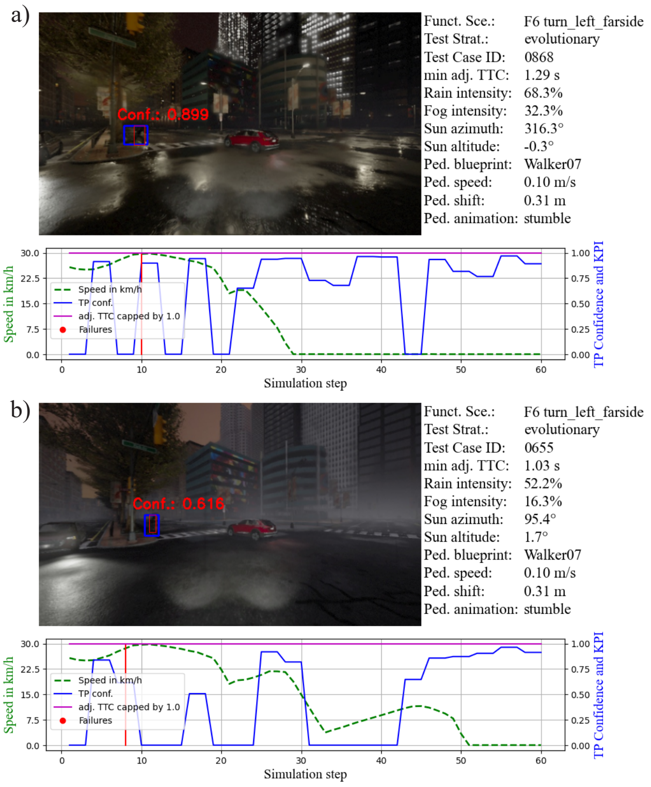
One of these failures is displayed in Fig. 6.6 a). Observing the test execution reveals that the pedestrian was not detected during this simulation run due to bad lighting conditions. Furthermore, due to the pedestrians' low speed, the pedestrian stays in the dark corner on the lower left side of the cameras' perspective. However, from the human perspective, the outlines of the pedestrian are still visible. The wide variety of failures detected by evolutionary testing can be explained by the fact that if these two scenario conditions are fulfilled, the values of the other conditions become insignificant, highlighting the exploitative properties of evolutionary sampling. I.e., the pedestrians' animation is irrelevant if the contrast to the background is too small. As a side note, this failure pattern also highlights the importance of traffic infrastructure. For $S_{\text{Sun altitude}} < 0\circ$, the traffic lights activate, making pedestrians more visible. For higher sun altitudes, the environmental lighting is better as well.

There are failures differing from above pattern with $S_{\text{Sun altitude}} \gg 0.47°$ ($\gg$ being defined as "significantly higher"). These are colored in red in Fig. 6.5 a) and all comply with

$$\begin{aligned} S_{\text{Fog intensity}} > 90.1\% \quad \wedge \quad S_{\text{Ped. speed}} < 1.06\text{m/s} \quad \wedge \\ S_{\text{Ped. blueprint}} \in \{\text{W 07, W 08, W 09}\} \ \wedge \ S_{\text{Ped. ani}} \in \{\text{stumble, crouching}\}. \end{aligned} \tag{6.4}$$

This corresponds with the main failure pattern of meta-model based testing using GPR, which accords with the results of the other meta-model based testing strategies. However, the specific value ranges slightly differ. These results are displayed in Fig. 6.5 b). A visualization of one detected failure during meta-model based testing using GPR is shown in Fig. 6.6 b). Due to the low speed, relatively high fog level, and the vegetation at the side of the road, the perception function misses the pedestrian crossing from the left side. The pedestrian gets detected by the YOLOv5 at time step 33 but is lost again after six time steps, leading to a collision. While the pedestrian is hardly visible in this scenario, even for the human eye, there is no obstruction, and the pedestrians' outlines are still recognizable.

On the other hand, a single failure was detected by GPR that corresponds with Eq. 6.3. This failure with $S_{\text{Sun altitude}} = 0.26°$ is highlighted in Fig. 6.6 b).



a)

Funct. Sce.:	F6 turn_left_farside
Test Strat.:	evolutionary
Test Case ID:	0868
min adj. TTC:	1.29 s
Rain intensity:	68.3%
Fog intensity:	32.3%
Sun azimuth:	316.3°
Sun altitude:	-0.3°
Ped. blueprint:	Walker07
Ped. speed:	0.10 m/s
Ped. shift:	0.31 m
Ped. animation:	stumble

b)

Funct. Sce.:	F6 turn_left_farside
Test Strat.:	evolutionary
Test Case ID:	0655
min adj. TTC:	1.03 s
Rain intensity:	52.2%
Fog intensity:	16.3%
Sun azimuth:	95.4°
Sun altitude:	1.7°
Ped. blueprint:	Walker07
Ped. speed:	0.10 m/s
Ped. shift:	0.31 m
Ped. animation:	stumble

Figure 6.7: Visualization of non-failure test execution during evolutionary testing.

The fact that only a few instances of the significant failure pattern Eq. 6.3 are detected by meta-model based testing can be explained by attributes of fitting a continuous meta-model on a brittle failure pattern. For $S_{\text{Sun altitude}} < 0$, the traffic lights activate (see Fig. 6.7 a) / adj. TTC = 1.29s), making the pedestrian visible. While for $S_{\text{Sun altitude}} \gtrapprox 1.0$, the natural sunlight makes the pedestrian visible enough for the object detector (see Fig. 6.7 b) / adj. TTC = 1.03s). Then, the continuous function smooths the predicted values in this range. On the contrary, evolutionary testing excels in exploiting these patterns, due to the mutation feature of just changing a subset of scenario variables. If the sweet

Table 6.5: *C-Measure* for 1,000 test executions using different sampling strategies, stating the saved fraction of test executions by an intelligent adaptive strategy. *n/a* indicates that the strategy did not match the amount of failures detected by random testing.

Strategy	$\mathcal{F}_1$	$\mathcal{F}_2$	$\mathcal{F}_3$	$\mathcal{F}_4$	$\mathcal{F}_5$	$\mathcal{F}_6$	$\mathcal{F}_7$	$\mathcal{F}_8$
Pair-wise	n/a	n/a	n/a	0.34	n/a	0.58	n/a	n/a
Evolutionary	**0.75**	0.57	-2.35	**0.98**	0.93	0.79	0.79	0.75
LR	n/a	0.77	0.58	0.82	0.86	0.92	0.80	0.84
NNR	-0.12	**0.88**	**0.85**	0.92	**1.00**	0.87	0.79	0.79
GPR	-0.08	0.70	0.35	0.85	0.84	**0.94**	**0.83**	**0.97**

spot of Eq.6.3 is detected once, the mutants of this parent element are likely to fall into this category.

However for larger failure patterns including the bigger ranges involving a higher number of variables, meta-model based testing using GPR or NNR have shown to more effective.

Saved cost of adaptive testing

While failure detection numbers focus on the absolute amount during the test budget, the introduced $C - measure$ (see Def. 4.17) evaluates how fast a certain level of failures has been reached relative to random testing. Thus, a high number near 1 indicates an earlier testing success, with a negative number indicating even slower testing success than pure random sampling. Tab. 6.5 shows the results during the experiments.

The meta-model based testing leads the table in six of the eight functional scenarios, evolutionary testing leading in $\mathcal{F}_1$ and $\mathcal{F}_4$. The six functional scenarios are evenly distributed between the usage of GPR and NNR. Overall, in each functional scenario, the amount of test executions could be reduced by above 75%, indicating the efficiency gain of adaptive testing.

The first striking observation is the *n/a* values for pair-wise testing and meta-model based testing using LR, indicating functional scenarios in which these testing processes failed to detect enough failures for a qualifying value. Furthermore, in $\mathcal{F}_1$ and $\mathcal{F}_3$, there are negative values for the *C-Measure*, indicating that these testing processes reached the required number of failures *later* than random testing. In general, each adaptive strategy has scored a negative value or *n/a* in at least one functional scenario.

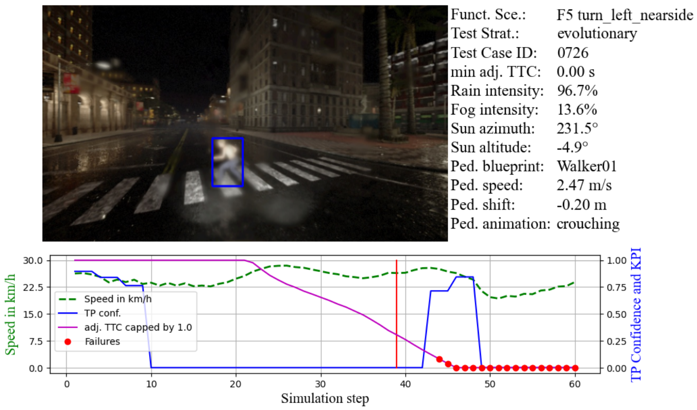
Another observation is that in only three scenarios, the strategy with the most detected failures is not the strategy that achieves the highest *C-Measure*. This also applies to $\mathcal{F}_6$ in which the evolutionary testing massively outperformed the other strategies. Evolutionary testing tends to require more test executions until the process detects its first failures but exploits these failure causes more efficiently. This fits to the observations made in Sec. 6.3, with the mutation step of evolutionary testing being effective in expanding on detected failures.

Coverage analysis of adaptive testing

Up to this point, the analysis focuses plainly on the strategies' ability to detect failures. No attention was paid to the variety of failures and how diverse the test case selection is. With the methodology from Sec. 6.1, the pair-wise combinatorial for failures γ_f and total test cases γ_t can be evaluated. The coverage analysis result is displayed in Tab. 6.6. $1.0 = 100\%$ represents the maximum value for γ_t, E.g. 100% of all pair-wise combinations of scenario variables are covered by the tested concrete scenarios.

The full coverage for the pair-wise testing is trivial, as the test selection is specifically designed to cover all combinations. Hence, the complete coverage γ_t is 1.00 by default. The random testing achieving the second highest overall coverage is interesting, as the selection is purely random. However, this avoids any overfitting towards some variable combinations and keeps the sampling evenly distributed. Neither of these strategies achieve any significant coverage values regarding failures due to a low number of detections.

A more interesting and telling observation of the coverage numbers for adaptive strategies can be made. On overall coverage γ_t, meta-model based testing using NNR achieves the highest result on all functional scenarios, outperforming the other strategies.



Funct. Sce.:	F5 turn_left_nearside
Test Strat.:	evolutionary
Test Case ID:	0726
min adj. TTC:	0.00 s
Rain intensity:	96.7%
Fog intensity:	13.6%
Sun azimuth:	231.5°
Sun altitude:	-4.9°
Ped. blueprint:	Walker01
Ped. speed:	2.47 m/s
Ped. shift:	-0.20 m
Ped. animation:	crouching

Figure 6.8: Detected Failure from evolutionary testing on $\mathcal{F}_5$ with $S_{\text{Sun Altitude}} = -4.89°$, $S_{\text{Precipitation}} > 96.71\%$, $S_{\text{Fog Intensity}} = 13.63$. This combination applies to 13 of all 24 failures of this strategy on $\mathcal{F}_5$.

Similar to the C-measure, the dynamics of γ_f tend to follow the absolute number of detected failures. However, the most striking observation can be made on functional scenario $\mathcal{F}_7$, where evolutionary testing outperformed the alternative approaches but trails meta-model based testing using NNR. Furthermore, in $\mathcal{F}_6$ evolutionary testing is leading the alternative approaches. However, this lead may not be as significant as the lead in the amount of detected failures. The cause of this observation was already touched on in Sec. 6.3 with evolutionary testing focusing to much on exploitation (see Fig. 6.5 for analysis on $\mathcal{F}_6$). Similar exploitation can be observed in evolutionary testing on $\mathcal{F}_5$ with 13 of 24 failures having the scenario variable values

$$S_{\text{Sun Altitude}} = -4.89° \;\wedge\; S_{\text{Precipitation}} > 96.71\% \;\wedge\; S_{\text{Fog Intensity}} = 13.63\%. \tag{6.5}$$

One of these failures is shown in Fig. 6.8. The simulation run starts with the object detector detecting the pedestrian with rather low TP confidence. Afterwards, the vehicle slightly accelerates due to the object detector losing the pedestrian. While the dark environment and the blurred image worsen the pedestrians' visibility, the pedestrian is still visible for the human eye. At the end of the simulation run, the pedestrian is detected again. However, the detection comes to late to prevent a collision.

Table 6.6: Combinatorial coverage from different test strategies for total tested concrete scenarios and the concrete scenarios that result in failures. The full coverage for total tested concrete scenarios for the pair-wise testing is somewhat trivial, as the test selection is specifically designed to cover all combinations.

Strategy	$\mathcal{F}_1$		$\mathcal{F}_2$		$\mathcal{F}_3$		$\mathcal{F}_4$	
	γ_f	γ_t	γ_f	γ_t	γ_f	γ_t	γ_f	γ_t
Random Testing	0.002	*0.786*	0.008	*0.789*	0.004	*0.789*	0.000	*0.785*
Pair-wise	0.000	*1.000*	0.002	*1.000*	0.000	*1.000*	0.004	*1.000*
Evolutionary	0.002	0.544	0.033	0.577	0.009	0.557	0.024	0.571
LR	0.000	0.650	0.052	0.656	0.049	0.670	0.023	0.652
MNR	**0.006**	**0.706**	**0.064**	**0.709**	**0.056**	**0.725**	**0.039**	**0.685**
GPR	**0.006**	0.674	0.056	0.658	0.021	0.657	0.026	0.668

Strategy	$\mathcal{F}_5$		$\mathcal{F}_6$		$\mathcal{F}_7$		$\mathcal{F}_8$	
	γ_f	γ_t	γ_f	γ_t	γ_f	γ_t	γ_f	γ_t
Random Testing	0.000	*0.787*	0.000	*0.785*	0.012	*0.789*	0.000	*0.789*
Pair-wise	0.000	*1.000*	0.004	*1.000*	0.008	*1.000*	0.000	*1.000*
Evolutionary	0.022	0.572	**0.073**	0.585	0.083	0.571	0.016	0.573
LR	0.008	0.682	0.054	0.668	0.085	0.657	0.013	0.654
NNR	**0.027**	**0.704**	0.057	**0.702**	**0.164**	**0.674**	**0.036**	**0.728**
GPR	0.022	0.651	0.055	0.652	0.120	0.656	0.015	0.687

7 Active Learning using adaptive Testing

7.1 Process overview for active learning

Chapter 6 is focused on testing an existing deep learning-driven perception model. However, the scope of this thesis extends to improve model performance. A key advantage of data-driven machine learning is improving model performance by adding data. The straightforward approach is to add the generated data to the training process. The model itself chooses the additional training data.

The idea of using a trained deep-learning model during the data collection process is an established field of machine learning, defined as active learning:

Definition 7.1 (Active Learning [62])**.** *A training approach in which the algorithm chooses some of the data it learns from. Active learning is particularly valuable when labeled examples are scarce or expensive to obtain. Instead of blindly seeking a diverse range of labeled examples, an active learning algorithm selectively seeks the particular range of examples it needs for learning.*

Traditionally, active learning in deep learning addresses the problem of huge data sets that cannot be processed due to labeling constraints. Using techniques such as uncertainty detection can identify image data that, for a perception function, appear out of domain [170, 171]. However, the data generation using the adaptive testing pipeline $(x_i)_{i \in \mathcal{N}}$ (see Def. 4.15) also fulfills the above description of targeting particular data examples. The data generation approach via adaptive sampling fits this definition, as it represents an active process to generate challenging data.

The underlying assumption is that the active learning approach outperforms random data sampling. This comparison is significant since improving the

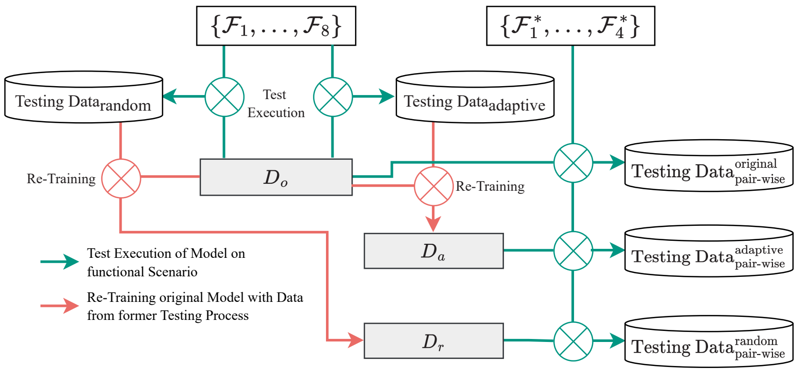

$\{\mathcal{F}_1, \ldots, \mathcal{F}_8\}$
$\{\mathcal{F}_1^*, \ldots, \mathcal{F}_4^*\}$
Testing Data$_{\text{random}}$
Test Execution
Testing Data$_{\text{adaptive}}$
Re-Training
D_o
Re-Training
Testing Data$^{\text{original}}_{\text{pair-wise}}$
D_a
Testing Data$^{\text{adaptive}}_{\text{pair-wise}}$
Test Execution of Model on functional Scenario
Re-Training original Model with Data from former Testing Process
D_r
Testing Data$^{\text{random}}_{\text{pair-wise}}$

Figure 7.1: Process of re-training and testing the new YOLOv5 model weights. The original model D_o gets re-trained two times. D_r is trained with additional data from random testing and D_a is trained with additional data from adaptive testing.

model solely by adding data is trivial. The process to evaluate the model improvement is shown in Fig. 7.1. Utilizing the results presented in Sec. 5.6, two data sets are generated using the testing pipeline on custom-trained YOLOv5 model for the functional scenarios $\{\mathcal{F}_1, ..., \mathcal{F}_8\}$: *adaptive* - generated through adaptive testing based on meta-model based testing using a neural network - and *random* - generated without implementation of adaptive strategies. The data sets contain the images and ground truths during the test executions of the two testing processes. For both data sets, the custom-trained YOLOv5 gets re-trained ($D(\cdot)$ referring to notation introduced by Eq. 4.4). This yields three models:

D_o Original custom trained model (see Sec. 5.4). The YOLOv5 model trained for the testing pipeline on custom CARLA data. Note that *original* does not refer to the published version from Ultralytics, but a version adapted to CARLA data.

D_r YOLOv5 model, trained on the data from D_o with added data from the testing pipeline using random testing without any strategy.

D_a *Actively learned* updated model. Similar to D_r, D_o gets re-trained with additional data from the testing pipeline. However, for D_a, adaptive strategies are used to asses the additional effect from data, specifically selected to falsify the performance of D_o.

To evaluate the performance of the three models D_o, D_a, and D_r the test pipeline is set up for additional functional scenarios $\{\mathcal{F}_1^*, ..., \mathcal{F}_4^*\}$. The description of the additional functional scenarios is detailed in Tab. 7.1. Setting up new scenarios instead of testing the new model on the old functional scenarios, or even concrete scenarios, is important, as the model is now explicitly fitted on scenarios $\mathcal{F}_1 - \mathcal{F}_8$. To reduce the risk of overfitting, these scenarios are set up on map Town03, which was neither included during the training of the model nor testing of the model, hence completely unseen to the three models. The logical scenario space is set up the same way as in $\{\mathcal{F}_1, ..., \mathcal{F}_8\}$.

To achieve a comparable test result, the models are run on the exact same sequence of concrete scenarios. This sequence cannot be generated adaptively, since the adaptive sampling focusses on the specific failure patterns of each model, which may differ. To generate the test sequence, the *t-wise* testing strategy was selected, as it resulted in the highest coverage of the logical scenario space.

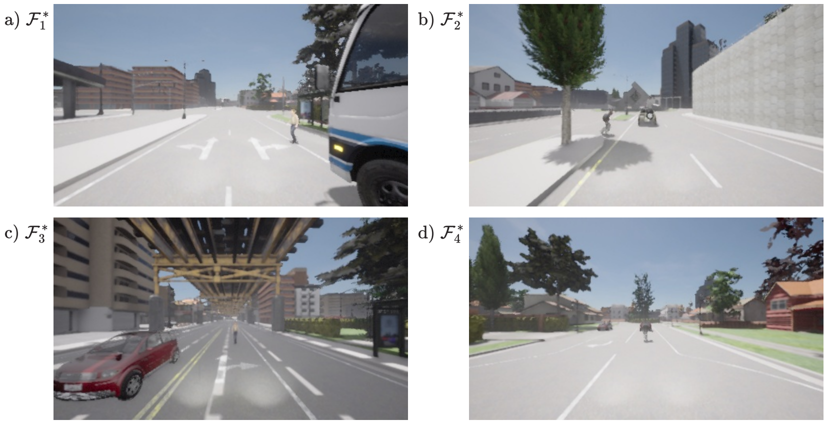


Figure 7.2: Visualization of additional functional scenarios.

Table 7.1: Defined functional scenarios, derived from the pedestrian-vehicle interactions from the Euro-NCAP scenarios for AEB systems [8].

$\mathcal{F}_1^*$	Suburban environment with bus station. Ego vehicle is passing a parking bus. A pedestrian is located behind the bus. The pedestrian disregards the traffic rules and crosses the street from the nearside in front of the vehicle.
$\mathcal{F}_2^*$	Suburban environment with straight road. Ego vehicle is following the road with an leading passenger car. A pedestrian disregards the traffic rules and crosses the street from the farside from behind a tree.
$\mathcal{F}_3^*$	Suburban environment with train line running over the road. The ego vehicle is following the road with an approaching vehicle on the opposite side of the track. A pedestrian is approaching the ego vehicle on the ego vehicle's lane.
$\mathcal{F}_4^*$	Suburban environment in residential area. The ego vehicle is passing the station with a vehicle leading on the left lane of the track. The vehicle is overtaking a pedestrian in the ego vehicles' lane.

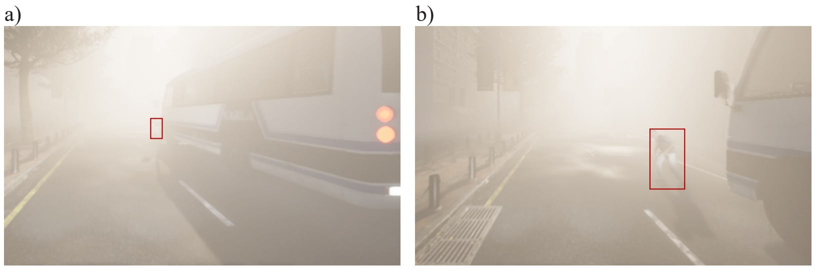


Figure 7.3: Extract from the process of ensuring data quality during re-training ($\mathcal{F}_1$). Both images are zooms from the original images, and D_o did not detect the pedestrian in any of the images. a) The pedestrian is not recognizable by the human eye, and the image is filtered out. b) The pedestrian is visible to the human eye, and the image is added to the training data.

7.2 Data selection and preparation

Using the generated images and labels of the testing process is more difficult than it appears at first glance. During the testing process, the data is selected to provoke system failures, i.g. false positives or false negatives. However, there are problems regarding the data quality. The adaptive strategies select concrete scenarios in which pedestrians are *less* visible, challenging the object detection function. In this process, technically correct labels for present pedestrians are generated. However, the pedestrian may not be visible in the image. Hence, the training data needs to be filtered for these *low quality* bounding boxes. The images with unrecognizable bounding boxes are excluded from the training process by judgement of the human eye. A visualization is shown in Fig. 7.3. The pedestrian in Fig. 7.3 a) is non-recognizable by the human eye. Thus the image is excluded. The visible pedestrian in Fig. 7.3 b) is included.

7.3 Active learning testing results

The results of applying the scenario-based testing pipeline to the three models D_o, D_r, and D_a using the three-wise testing strategy are shown in Tab. 7.2. Each entry shows the number of failures during the test sequence with the respective model.

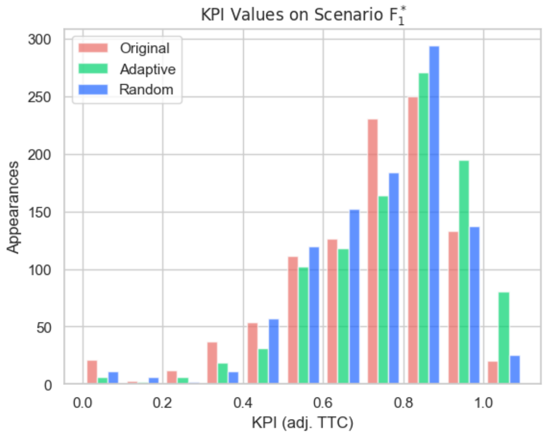
Table 7.2: Resulting number of failures during pair-wise testing of new model weights for all additional functional test scenarios.

Strategy	$\mathcal{F}_1^*$	$\mathcal{F}_2^*$	$\mathcal{F}_3^*$	$\mathcal{F}_4^*$
D_o	21	69	40	25
D_r	11	5	44	6
D_a	**6**	**4**	**25**	**3**

In each functional scenario, the D_a outperforms the other models, supporting the hypothesis of significantly improved perception performance when using adaptively generated data during training. In $\mathcal{F}_1^*$ the number of failures decreased by about 47% from the original model to D_r and by 71% to D_a proving a significantly improved performance for the re-trained. The most significant reduction can be seen in $\mathcal{F}_2^*$, where the amount of failures is reduced from 69 in the original model to four with the D_a model. Additionally, there were only five failures, with the D_r being only outperformed by one failure by D_a.

Another interesting observation can be made in $\mathcal{F}_3^*$, with D_a significantly outperforming both other models. However, D_r is worse than D_o. This cannot be explained by training data but has to be attributed to the non-linear nature of neural networks. Results on $\mathcal{F}_4^*$ also show significant improvement using adaptive data compared to the random testing process with 50% as many failures and only 12% compared to D_o.

Fig. 7.4 and Fig. 7.5 show a histogram for each functional scenario containing the KPI results for each model. Thus, Tab. 7.2 equates with the data in the first bin from 0 to 0.1 which are the concrete scenarios with $adj.TTC \leq 0.1$. While the whole picture largely corresponds with the analysis to this point, there is an outlier regarding $\mathcal{F}_2^*$: While the original model massively underperforms with 69 failures compared to the adaptive model, the original model



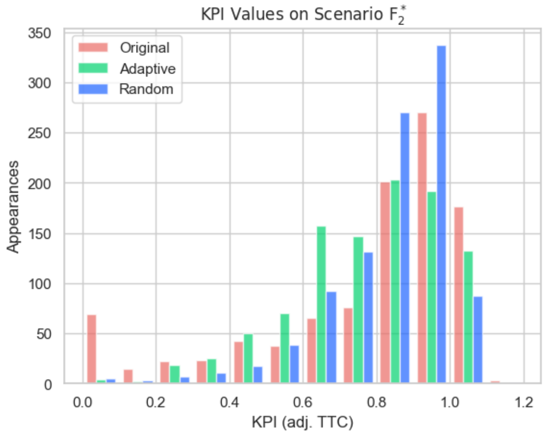
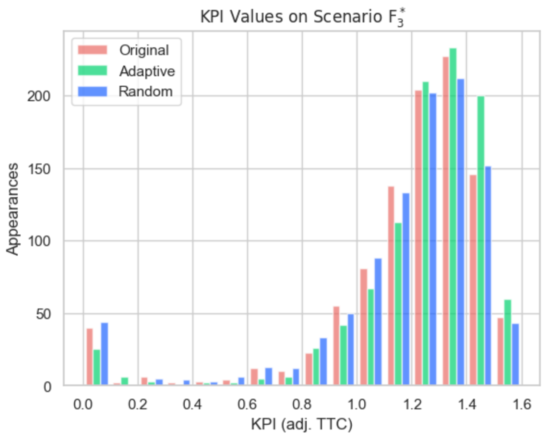


Figure 7.4: Histogram for each functional scenarios $\mathcal{F}_1^*$ and $\mathcal{F}_2^*$ containing the test results for each model.

also has significantly more test results in the range between 1.0 and 1.2. In fact, the median of test results decreased from 0.88 (D_o) to 0.82 (D_a) although the number of failures plummeted from 96 to 4.

Fig. 7.6 displays a test case in which the different models' performances are compared. a) With D_o, the pedestrian is detected at no point of the simulation,



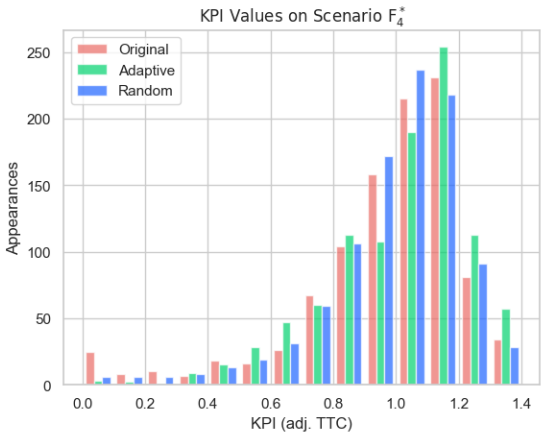
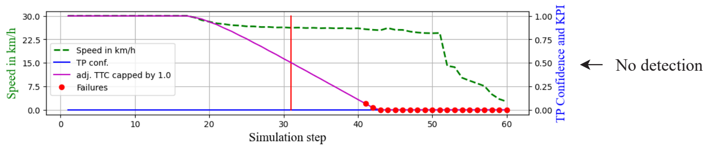


Figure 7.5: Histogram for each functional scenarios $\mathcal{F}_3^*$ and $\mathcal{F}_4^*$ containing the test results for each model.

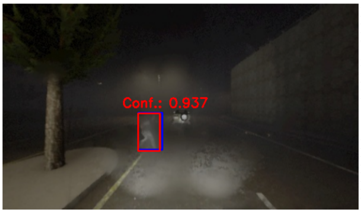
with low visibility in rainy and foggy weather at night. b) In contrast, during simulation with D_a, the pedestrian is detected, starting at time step 31, avoiding a collision. c) Using D_r, the pedestrian is detected at time step 31 but is lost shortly after, causing the AEB to release the brake and resulting in a collision.

a)

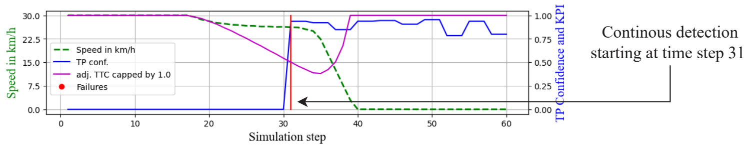
Funct. Sce.:	F_2^* - crossing_farside
Model Ver.:	D_o - original
Test Case ID:	0571
min adj. TTC:	0.00 s
Rain intensity:	94.7%
Fog intensity:	83.6%
Sun azimuth:	192.4°
Sun altitude:	-5.6°
Ped. blueprint:	Walker08
Ped. speed:	1.52 m/s
Ped. shift:	-0.22 m
Ped. animation:	limping



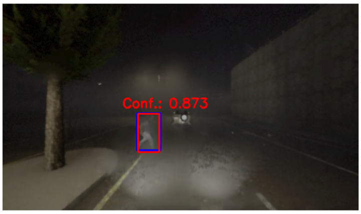
b)



Funct. Sce.:	F_2^* - crossing_farside
Model Ver.:	D_a - adaptive
Test Case ID:	0571
min adj. TTC:	0.38 s
Rain intensity:	94.7%
Fog intensity:	83.6%
Sun azimuth:	192.4°
Sun altitude:	-5.6°
Ped. blueprint:	Walker08
Ped. speed:	1.52 m/s
Ped. shift:	-0.22 m
Ped. animation:	limping



c)



Funct. Sce.:	F_2^* - crossing_farside
Model Ver.:	D_r - random
Test Case ID:	0571
min adj. TTC:	0.00 s
Rain intensity:	94.7%
Fog intensity:	83.6%
Sun azimuth:	192.4°
Sun altitude:	-5.6°
Ped. blueprint:	Walker08
Ped. speed:	1.52 m/s
Ped. shift:	-0.22 m
Ped. animation:	limping

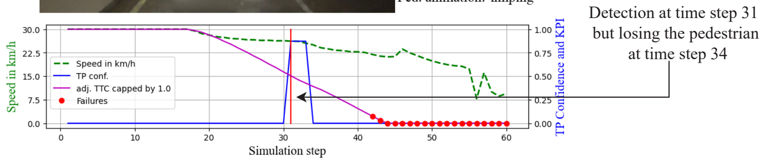


Figure 7.6: Visualization of testing improved YOLOv5 versions on the same concrete scenario. a) Model performance of D_o, b) Model performance of D_a, c) Model performance of D_r.

8 Conclusion and outlook

8.1 Discussion

In Ch. 2, the concept of deep learning is presented. Belonging to machine learning, the development of deep learning models differs from the development of hard-coded programs, as the decision-making significantly originates from the data the model was trained on.

Ch. 3 addresses existing and established testing techniques for deep learning systems and the key safety concerns arising from using deep learning models in automated driving tasks. Furthermore, significant shortcomings of these testing techniques are highlighted, especially the unfeasibility of collecting, processing, and labeling of sufficient amounts of real-world data via recordings. Real-world data sets may be unbalanced, with many challenging scenarios being underrepresented (see Fig. 1.2).

In Ch. 4, a closed-loop testing pipeline for adaptive sampling is proposed based on scenario-based testing. The improvements in sensor simulation tools have opened the door for simulative testing in a virtual environment, although questions regarding the transferability of results remain. The simulation addresses the limitations of real-world data recordings. With labeling and recording costs limiting the amount of real-world data that can be collected, simulation can complement the available data by sensor data of rare but challenging driving scenarios. In essence, the pipeline structures the high dimensional space of traffic scenarios. At the core of the thesis, traffic scenarios are made parametrized, computer-readable, and executable by simulation software. By doing so, a formalized scenario space gets spanned on which the system-under-test operates. However, this scenario space is spanned over many dimensions, such as weather factors, object behavior, and object appearance. Applying adaptive sampling to the test execution enables the pipeline to uncover perception failures in the parameter space. In adaptive sampling, a strategy is applied with the clear objective to trigger simulation runs that the system cannot pass. Meta-model-based and evolutionary testing were used in the pipeline and evaluated for their ability to detect failures.

Ch. 5 outlines a practical implementation of the concepts introduced in Ch. 4 for evaluation. Using the CARLA simulation tool, a pipeline is created to test a custom-trained YOLOv5 object detection network adaptively. The object detector is used for an adaptive cruise control setting to avoid a collision with jaywalking pedestrians. For this pipeline, all functional scenarios and simulation variables are presented, as well as the simulation KPI and visualization.

Results of the testing pipeline are shown in Ch 6, with adaptive sampling being evaluated against random sampling and space-filling t-wise testing. In contrast to non-adaptive sampling strategies, this approach can efficiently detect failures, even when random testing cannot do so. In the experiments, using meta-model-based testing using a neural network as a meta-model has yielded the most efficient results, outperforming the other sampling strategies.

Finally, Ch 7 addresses the testing processes' ability to improve the SuT's performance by retraining the model on the challenging data generated during testing. Retraining with adaptive data resulted in the model significantly outperforming the previous model and reducing failures. Hence, the proposed methods not only falsify perception methods but also improve their performance.

Fig. 8.1 places the proposed pipeline in context with SOTIF [49, 172]. The problem of unknown and unsafe scenarios is approached in parallel. Using adaptive sampling to generate new data, the space of unknown scenarios gets explored with a focus on unsafe scenarios. Hence, the SuT's capability to handle scenarios safely gets mapped on the scenario space to identify the ODD in which the vehicle can operate. Improving the SuT's performance by retraining the model with the adaptively generated data also increases the space of safe scenarios.

In Sec. 3.3, the major concerns are presented regarding the use of deep learning algorithms in perception for automated driving, based on [95]. Here, the proposed method is set in context to these concerns.

a) Data distribution is not a good approximation of real world: In this context, a domain can be interpreted on signal-level and scenario level (see definitions in Sec. 3.1). On the scenario level, including traffic statistics

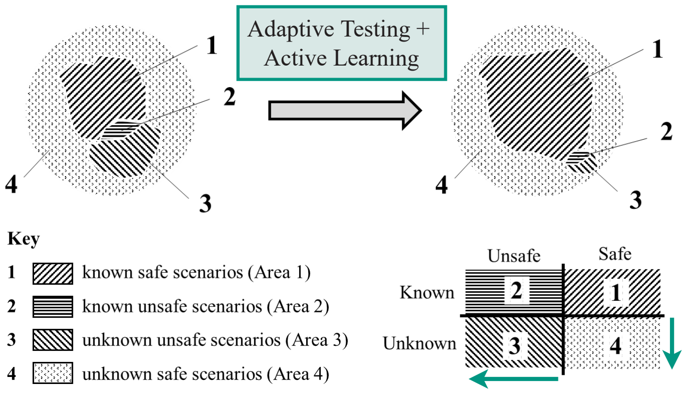

Adaptive Testing +
Active Learning
1
2
3
4
Key
1 known safe scenarios (Area 1)
2 known unsafe scenarios (Area 2)
3 unknown unsafe scenarios (Area 3)
4 unknown safe scenarios (Area 4)
Unsafe
Safe
Known
Unknown

Figure 8.1: Placing the proposed testing process in context with SOTIF [172]. Adaptive scenario-based testing manages to expand the space of known and safe scenarios. Image by [49].

helps improve real-world data sets to be more representative by filling domain gaps. However, accurate observations of scenario variables are required. While the cost, variety, and reliability of simulative image generation are superior to real world-based data sources, there are drawbacks regarding domain gaps on signal level, especially since there are still concerns regarding the transferability of synthetic data to real world applications.

b) Distributional shift over time: This concern is highly related to a). The advantages of simulative testing can be leveraged to adapt to shifting data distribution. Similarly to a), constant traffic monitoring is required, with constant improvement of simulation capabilities. For new scenario elements, such as introducing new traffic objects, data can be generated immediately without costly data collection.

c) Dependence on labeling quality: A significant advantage of simulative image generation is reliable ground truth generation, as the simulation platform can generate many accurate ground truth data, such as instance segmentation, depth information, or optical flow. Hence, labeling quality is a strength of the proposed testing process if testing data is used for further development.

d) Unknown behavior in rare critical situations: Addressing this safety concern is the most significant advantage of adaptive scenario-based testing. Systematic sampling of the scenario space identifies weaknesses of the perception function that must be addressed in further development. Hence, the models' behavior in rare critical situations can be observed and mitigated if insufficiencies exist. Alternatively, if weaknesses cannot be mitigated, the domain for which the function is authorized is limited by boundaries.

e) Inadequate test and training data separation: The proposed testing process must address this safety concern. However, if simulative testing data is used for model development (as described in c)), over-confidence may occur. The testing of the improved model must then be tested on completely new test scenarios.

f) Incomprehensible behavior: For deep learning models, explainability must be achieved through white-box testing, which the proposed testing process does not address.

g) Brittleness of DNNs: While simulative testing can increase domain coverage and identify weaknesses in the scenario space, the inherit brittleness of DNNs cannot be solved exclusively by assembling more data.

h) Unreliable confidence information: The proposed testing pipeline identifies scenarios in which the models' confidence may lead to false vehicle behavior. However, research has shown that addressing wrong confidences requires improved architectures and training methods instead of more data. Hence, this safety concern is rather out-of-scope of this work.

i) Insufficient consideration of safety in metrics: The presented approach for testing perception functions utilizes the dynamic manner of the simulation to generate an intuitive performance metric. In contrast to recall or precision-based metrics, the adj. TTC allows to intuitively evaluate the models' ability to prevent collisions by detecting pedestrians reliably.

8.2 Future Work

Assessment and Extension of Simulation Quality and Coverage

While the experiments in this dissertation gave important conclusions on applying adaptive testing to object detectors and the ability to detect systemic weaknesses in perception function behavior, there remain significant unknowns. As mentioned in multiple chapters, a testing process with synthetic data only is as viable as the produced data is viable. Hence, future work in this research includes using a physics-based camera and assessing the real-world - simulation gap [173, 174]. These include two main categories:

Geometric dimensions Assure that the dimensions of objects, their shadows, and road geometry in the simulation will match dimensions in the real world. This can be assessed via checkerboard patterns for camera simulation (Fig. 8.2).

Photo-realism Photo-realism refers to "how realistic" an image appears to the human eye. Improved rendering and 3D models increase the realism of image features that trigger the neurons in a DL-driven perception function.

Additionally, behavior models of traffic participants have been shown to significantly influence the trustworthiness of the traffic simulation [161]. Thus, quality assessment and extension of behavior models have to be performed.

The simulation quality also depends on the coverage of traffic artifacts implemented. A simple example is the simulation of a deer on a rural road that

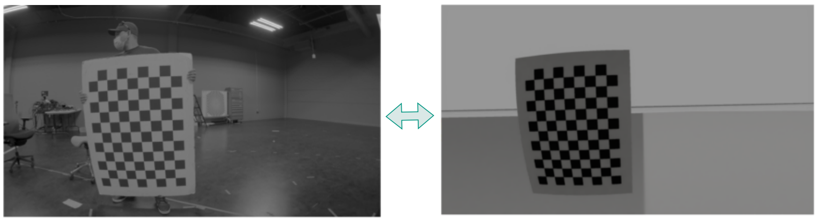


Figure 8.2: Validation of geometric precision of camera simulation using a checkerboard pattern [7].

can only be simulated if a corresponding 3D model is available. Furthermore, over time, the ODD of the perception function inevitably changes. Introducing new traffic participants, such as e-scooter riders, can pose a risk to the perception function that was not accounted for during testing. Hence, constant traffic monitoring is required using techniques such as uncertainty detection or out-of-distribution detection.

Combining real and simulative Data to yield Probability of Failure

Def. 4.11 and Def. 4.12 introduced a formula to calculate the probability of failure for the system using scenario-based testing. In essence, it raises information on the probability distribution of functional scenarios and the scenario variables inside the logical scenario.

The basic concept for the probability of failure stems from traditional reliability analysis [146]. For every logical scenario, two functions are required: The probability density function and a performance function. Then, the probability of failure is defined as the integral of the probability density function over the subspace of the scenario space in which the performance falls below the minimum required KPI. Fig. 8.3 shows the relationship between the simulation process and the distributional information of scenario occurrence in the real world.

Generating a data set for functional scenarios with corresponding occurrence probabilities out of real trajectory recordings requires additional modules for

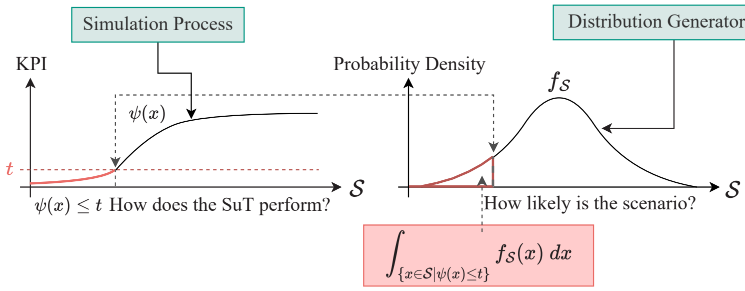

Simulation Process
Distribution Generator
KPI
Probability Density
f_S
$\psi(x)$
t
$\mathcal{S}$
$\psi(x) \le t$ How does the SuT perform?
How likely is the scenario?
$\mathcal{S}$
$\int_{\{x \in \mathcal{S} | \psi(x) \le t\}} f_S(x)\, dx$

Figure 8.3: Probability of failure calculation for functional $\mathcal{S}$.

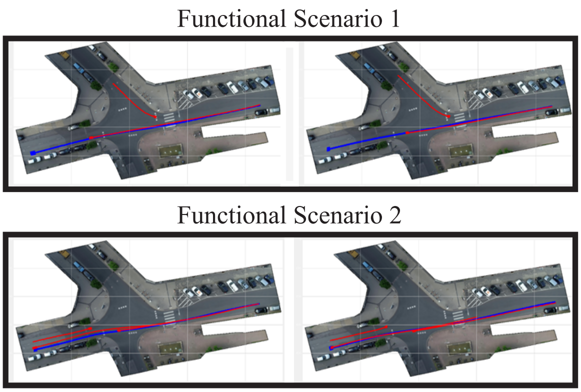


Figure 8.4: Scenario clustering to generate functional scenarios from real-world trajectory data [1].

scenario analysis. Such a model was already proposed while working on this dissertation in [1], with the introduction of a scenario clustering algorithm. Traffic scenarios on a fixed road segment are grouped according to trajectory information. All scenario clusters can be interpreted as functional scenarios (see Fig. 8.4).

In each logical scenario, accessing the distributional data is more difficult on some scenario variables than others. Retrieving weather distribution is possible via weather data provider and speed distributions via trajectory analysis; getting information on behavior models or participant appearance will pose a bigger challenge.

Given the distributional information and the simulative testing process, techniques such as Monte Carlo simulation [175] or Sub-set simulation [176] can be used to calculate the probability of failure.

Publications Johannes Bernhard

[1] Johannes Bernhard, Mark Schutera, and Eric Sax. Optimizing test-set diversity: Trajectory clustering for scenario-based testing of automated driving systems. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pages 1371–1378. IEEE, 2021.

[2] Johannes Bernhard, Thomas Schulik, Mark Schutera, and Eric Sax. Adaptive test case selection for dnn-based perception functions. In *2021 IEEE International Symposium on Systems Engineering (ISSE)*, pages 1–7. IEEE, 2021.

[3] Johannes Bernhard, Jonas Schmidt, and Mark Schutera. Density based anomaly detection for wind turbine condition monitoring. In *Proceedings of the 1st International Joint Conference on Energy and Environmental Engineering-CoEEE*, pages 87–93, 2021.

[4] Lucas Fonseca Alexandre de Oliveira, Johannes Bernhard, Lars Schories, Martin Meywerk, and Eric Sax. Enhancing carla traffic simulation with pedestrian animation for testing perception functions in automated driving. In *2023 IEEE 26th International Conference on Intelligent Transportation Systems (ITSC)*, pages 3859–3864. IEEE, 2023.

Bibliography

[5] PEGASUS Project. *THE PEGASUS METHOD*, 2019 (accessed February 01, 2024). `https://www.pegasusprojekt.de/en/pegasus-method`.

[6] PEGASUS Project. *Pegasus Method - An Overview*, 2019 (accessed November 20, 2019). `https://www.pegasusprojekt.de/files/tmpl/Pegasus-Abschlussveranstaltung/PEGASUS-Gesamtmethode.pdf`.

[7] NVIDIA. *Validating NVIDIA DRIVE Sim Camera Models*, accessed December 15, 2023). `https://developer.nvidia.com/blog/validating-drive-sim-camera-models/`.

[8] The European New Car Assessment Programme. *EUROPEAN NEW CAR ASSESSMENT PROGRAMME (Euro NCAP), TEST PROTOCOL – AEB VRU systems*, 2021 (accessed February 03, 2023). `https://cdn.euroncap.com/media/62795/euro-ncap-aeb-vru-test-protocol-v304.pdf`.

[9] Bundesministerium für Verkehr und digitale Infrastruktur. *Ethics Commission's complete report on automated and connected driving*, 2017 (accessed November 23, 2021). `https://www.bmvi.de/SharedDocs/EN/publications/report-ethics-commission.html`.

[10] Bernhard Friedrich. The effect of autonomous vehicles on traffic. In *Autonomous Driving*, pages 317–334. Springer, 2016.

[11] Statistisches Bundesamt. *Straßenverkehrsunfälle mit Personenschaden, Getöteten, Schwer- und Leichtverletzten: Deutschland, Jahre*, 2022 (accessed December 02, 2023). `https://www.destatis.de/DE/Themen/Gesellschaft-Umwelt/Verkehrsunfaelle/_inhalt.html`.

[12] Statistisches Bundesamt. *Todesursachen in Deutschland*, 2022 (accessed December 03, 2023). `https://www.destatis.de/DE/Themen/Gesellschaft-Umwelt/Gesundheit/Todesursachen/_inhalt.html`.

[13] Waymo press release. *Waymo IAA Frankfurt 2019*, 2019 (accessed November 21, 2019). `https://medium.com/waymo/waymo-iaa-frankfurt-2019-b3cca36d8479`.

[14] Yves Page, Jean-Yves Foret-Bruno, and Sophie Cuny. Are expected and observed effectiveness of emergency brake assist in preventing road injury accidents consistent. In *19th ESV Conference, Washington DC., USA*, 2005.

[15] Thomas Hummel, Matthias Kühn, Jenö Bende, and Antje Lang. Advanced driver assistance systems: An investigation of their potential safety benefits based on an analysis of insurance claims in germany. *German Insurance Association Insurers Accident Research, Research Report FS*, 3, 2011.

[16] European Union. *Regulation (EU) 2019/2144 of the European Parliament and of the Council*, 2019 (accessed November 23, 2021). `https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32019R2144&from=EN`.

[17] Ernst & Young. *Who's in the driving seat*, 2015 (accessed November 21, 2019). `https://ey-france.relayto.com/e/who-s-in-the-driving-seat-hacgevic`.

[18] American Automobile Association. *AAA: Fear of Self-Driving Cars on the Rise*, 2023 (accessed December 03, 2023). `https://newsroom.aaa.com/2023/03/aaa-fear-of-self-driving-cars-on-the-rise/`.

[19] American Transportation Research Institute. *An Analysis of the Operational Costs of Trucking: 2023 Update*, 2023 (assessed January 24, 2024)). `https://truckingresearch.org/wp-content/uploads/2023/06/ATRI-Operational-Cost-of-Trucking-06-2023.pdf`.

[20] American Journal of Transportation. *The Truck Driver Shortage in the US Continues*, 2023 (assessed January 24, 2024)). `https://www.ajot.com/news/the-truck-driver-shortage-in-the-us-continues`.

[21] International Road Transport Union (IRU). *Global truck driver shortage to double by 2028*, 2023 (assessed January 24, 2024)). `https://www.iru.org/news-resources/newsroom/global-truck-driver-shortage-double-2028-says-new-iru-report`.

[22] Stephen Wood, Jesse Chang, Thomas Healy, and John Wood. The potential regulatory challenges of increasingly autonomous motor vehicles. *Santa Clara L. Rev.*, 52:1423, 2012.

[23] SAE On-Road Automated Vehicle Standards Committee. Taxonomy and definitions for terms related to on-road motor vehicle automated driving systems. *SAE Standard*, 3016:1–16, 2014.

[24] Walther Wachenfeld and Hermann Winner. The release of autonomous vehicles. In *Autonomous driving*, pages 425–449. Springer, 2016.

[25] ISO Central Secretary. Road vehicles — safety and cybersecurity for automated driving systems — design, verification and validation. Standard ISO/TR 4804:2020(E), International Organization for Standardization, Geneva, CH, 2020.

[26] Nidhi Kalra and Susan M Paddock. Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability? *Transportation Research Part A: Policy and Practice*, 94:182–193, 2016.

[27] Jacob Langner, Johannes Bach, Lennart Ries, Stefan Otten, Marc Holzäpfel, and Eric Sax. Estimating the uniqueness of test scenarios derived from recorded real-world-driving-data using autoencoders. In *IEEE Intelligent Vehicles Symposium (IV)*, pages 1860–1866. IEEE, 2018.

[28] Lennart Ries, Jacob Langner, Stefan Otten, Johannes Bach, and Eric Sax. A driving scenario representation for scalable real-data analytics with neural networks. In *IEEE Intelligent Vehicles Symposium (IV)*, pages 2215–2222. IEEE, 2019.

[29] Johannes Bach, Jacob Langner, Stefan Otten, Eric Sax, and Marc Holzapfel. Test scenario selection for system-level verification and validation of geolocation-dependent automotive control systems. In *ICE/ITMC*, pages 203–210, 2017.

[30] Simon Ulbrich, Till Menzel, Andreas Reschka, Fabian Schuldt, and Markus Maurer. Defining and substantiating the terms scene, situation, and scenario for automated driving. In *IEEE 18th International Conference on Intelligent Transportation Systems*, pages 982–988. IEEE, 2015.

[31] IHS Markit. *Artificial intelligence driving autonomous vehicle development*, 2020 (accessed November 29, 2021). `https://ihsmarkit.co`

m/research-analysis/artificial-intelligence-driving-autonomous-vehicle-development.html.

[32] Our World in Data. *What is Moore's Law?*, accessed January 16, 2024). https://ourworldindata.org/moores-law.

[33] Jane Hung and Anne Carpenter. Applying faster r-cnn for object detection on malaria images. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 56–61, 2017.

[34] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[35] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[36] Sorin Grigorescu, Bogdan Trasnea, Tiberiu Cocias, and Gigel Macesanu. A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, 37(3):362–386, 2020.

[37] Sajjad Mozaffari, Omar Y Al-Jarrah, Mehrdad Dianati, Paul Jennings, and Alexandros Mouzakitis. Deep learning-based vehicle behavior prediction for autonomous driving applications: A review. *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[38] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.

[39] Ben Miethig, Ash Liu, Saeid Habibi, and Martin Mohrenschildt. Leveraging thermal imaging for autonomous driving. In *IEEE Transportation Electrification Conference and Expo (ITEC)*, pages 1–5. IEEE, 2019.

[40] Mitsunori Mizumachi, Atsunobu Kaminuma, Nobutaka Ono, and Shigeru Ando. Robust sensing of approaching vehicles relying on acoustic cues. *Sensors*, 14(6):9546–9561, 2014.

[41] Jelena Kocić, Nenad Jovičić, and Vujo Drndarević. Sensors and sensor fusion in autonomous vehicles. In *26th Telecommunications Forum (TELFOR)*, pages 420–425. IEEE, 2018.

[42] Leichen Wang, Tianbai Chen, Carsten Anklam, and Bastian Goldluecke. High dimensional frustum pointnet for 3d object detection from camera, lidar, and radar. In *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*, 2020.

[43] Fisher Yu, Wenqi Xian, Yingying Chen, Fangchen Liu, Mike Liao, Vashisht Madhavan, and Trevor Darrell. BDD100K: A diverse driving video database with scalable annotation tooling. *CoRR*, abs/1805.04687, 2018.

[44] Isaac Ogunrinde and Shonda Bernadin. A review of the impacts of defogging on deep learning-based object detectors in self-driving cars. *SoutheastCon 2021*, pages 01–08, 2021.

[45] Mazin Hnewa and Hayder Radha. Object detection under rainy conditions for autonomous vehicles: A review of state-of-the-art and emerging techniques. *IEEE Signal Processing Magazine*, 38(1):53–67, 2020.

[46] Vishwanath A Sindagi, Poojan Oza, Rajeev Yasarla, and Vishal M Patel. Prior-based domain adaptive object detection for hazy and rainy conditions. In *European Conference on Computer Vision*, pages 763–780. Springer, 2020.

[47] Matthias Heller. *Data Labeling: AI's Human Bottleneck*, 2020 (accessed November 25, 2021). `https://medium.com/whattolabel/data-labeling-ais-human-bottleneck-24bd10136e52`.

[48] IBM. *What is data labeling?*, accessed December 16, 2023. `https://www.ibm.com/topics/data-labeling`.

[49] ISO Central Secretary. Road vehicles - safety of the intended functionality. Standard ISO 21448:2022, International Organization for Standardization, Geneva, CH, 2022.

[50] Yohann Cabon, Naila Murray, and Martin Humenberger. Virtual kitti 2, 2020.

[51] Google Cloud. Ai platform data labeling service pricing. `https://cloud.google.com/ai-platform/data-labeling/pricing`, 2023. accessed December 18th 2023.

[52] IEEE Standards Coordinating Committee et al. Ieee standard glossary of software engineering terminology (ieee std 610.12-1990). los alamitos. *CA: IEEE Computer Society*, 169:132, 1990.

[53] Christian Murphy, Gail E Kaiser, and Marta Arias. An approach to software testing of machine learning applications. In *Proceedings of the 19th international conference on software engineering and knowledge engineering (SEKE)*, pages 167—172, 2007.

[54] Martin D Davis and Elaine J Weyuker. Pseudo-oracles for non-testable programs. In *Proceedings of the ACM'81 Conference*, pages 254–257, 1981.

[55] Jie M Zhang, Mark Harman, Lei Ma, and Yang Liu. Machine learning testing: Survey, landscapes and horizons. *IEEE Transactions on Software Engineering*, 2020.

[56] Andreas Vogelsang and Markus Borg. Requirements engineering for machine learning: Perspectives from data scientists. In *2019 IEEE 27th International Requirements Engineering Conference Workshops (REW)*, pages 245–251. IEEE, 2019.

[57] Lei Ma, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Felix Juefei-Xu, Chao Xie, Li Li, Yang Liu, Jianjun Zhao, et al. Deepmutation: Mutation testing of deep learning systems. In *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*, pages 100–111. IEEE, 2018.

[58] Sebastian Raschka and Vahid Mirjalili. *Machine Learning mit Python und Scikit-Learn und TensorFlow: Das Praxis-Handbuch für Data Science, Predictive Analytics und Deep Learning*. MITP Verlags GmbH & Company KG, 2017.

[59] Chih-Hong Cheng, Chung-Hao Huang, Harald Ruess, Hirotoshi Yasuoka, et al. Towards dependability metrics for neural networks. In *2018 16th ACM/IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE)*, pages 1–4. IEEE, 2018.

[60] European Commission, Content Directorate-General for Communications Networks, and Technology. *Ethics guidelines for trustworthy AI*. Publications Office, 2019.

[61] ISO Central Secretary. Systems and software engineering — systems and software quality requirements and evaluation (square) — system and software quality models. Standard ISO/IEC 25010:2011, International Organization for Standardization, Geneva, CH, 2011.

[62] Google Developers. *Machine Learning Glossary*, accessed December 08, 2021. `https://developers.google.com/machine-learning/glossary#loss`.

[63] Georg Volk, Stefan Müller, Alexander von Bernuth, Dennis Hospach, and Oliver Bringmann. Towards robust cnn-based object detection through augmentation with synthetic rain variations. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 285–292. IEEE, 2019.

[64] Guofa Li, Yifan Yang, and Xingda Qu. Deep learning approaches on pedestrian detection in hazy weather. *IEEE Transactions on Industrial Electronics*, 67(10):8889–8899, 2019.

[65] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[66] Youcheng Sun, Xiaowei Huang, Daniel Kroening, James Sharp, Matthew Hill, and Rob Ashmore. Testing deep neural networks. *arXiv preprint arXiv:1803.04792*, 2018.

[67] Augustus Odena and Ian Goodfellow. Tensorfuzz: Debugging neural networks with coverage-guided fuzzing. *arXiv preprint arXiv:1807.10875*, 2018.

[68] Chih-Hong Cheng, Chung-Hao Huang, and Hirotoshi Yasuoka. Quantitative projection coverage for testing ml-enabled autonomous systems. In *International Symposium on Automated Technology for Verification and Analysis*, pages 126–142. Springer, 2018.

[69] Christoph Gladisch, Christian Heinzemann, Martin Herrmann, and Matthias Woehrle. Leveraging combinatorial testing for safety-critical computer vision datasets. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 324–325, 2020.

[70] Zachary C Lipton. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue*, 16(3):31–57, 2018.

[71] Grégoire Montavon, Sebastian Bach, Alexander Binder, Wojciech Samek, and Klaus-Robert Müller. Deep taylor decomposition of neural networks. In *Proceedings of the ICML 2016 Workshop on Visualization for Deep Learning*, 2016.

[72] Xihui Liu, Haiyu Zhao, Maoqing Tian, Lu Sheng, Jing Shao, Shuai Yi, Junjie Yan, and Xiaogang Wang. Hydraplus-net: Attentive deep features for pedestrian analysis. In *Proceedings of the IEEE international conference on computer vision*, pages 350–359, 2017.

[73] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

[74] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[75] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7310–7311, 2017.

[76] Ricardo Baeza-Yates and Zeinab Liaghat. Quality-efficiency trade-offs in machine learning for text processing. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 897–904. IEEE, 2017.

[77] Sam Corbett-Davies and Sharad Goel. The measure and mismeasure of fairness: A critical review of fair machine learning. *arXiv preprint arXiv:1808.00023*, 2018.

[78] Warren McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[79] Donald Olding Hebb. *The organization of behavior: a neuropsychological theory*. J. Wiley; Chapman & Hall, 1949.

[80] Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

[81] David Rumelhart, Geoffrey Hinton, and Ronald Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[82] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[83] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[84] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[85] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[86] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[87] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[88] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[89] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[90] Yunchao Wang, Zehui Wu, Qiang Wei, and Qingxian Wang. Neufuzz: Efficient fuzzing with deep neural network. *IEEE Access*, 7:36340–36352, 2019.

[91] Mark Schutera, Mostafa Hussein, Jochen Abhau, Ralf Mikut, and Markus Reischl. Night-to-day: Online image-to-image translation for object detection within autonomous driving by night. *IEEE Transactions on Intelligent Vehicles*, 6(3):480–489, 2020.

[92] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[93] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft

coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[94] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[95] Oliver Willers, Sebastian Sudholt, Shervin Raafatnia, and Stephanie Abrecht. Safety concerns and mitigation approaches regarding the use of deep learning in safety-critical perception tasks. In *International Conference on Computer Safety, Reliability, and Security*, pages 336–350. Springer, 2020.

[96] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International Conference on Machine Learning*, pages 1321–1330. PMLR, 2017.

[97] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016.

[98] Kaushik Madala and Carlos Avalos Gonzalez. Metrics for machine learning models to facilitate sotif analysis in autonomous vehicles. Technical report, SAE Technical Paper, 2023.

[99] Georg Volk, Jörg Gamerdinger, Alexander von Bernuth, and Oliver Bringmann. A comprehensive safety metric to evaluate perception in autonomous systems. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–8. IEEE, 2020.

[100] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, et al. Deepgauge: Multi-granularity testing criteria for deep learning systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pages 120–131, 2018.

[101] Xiaowei Huang, Daniel Kroening, Wenjie Ruan, James Sharp, Youcheng Sun, Emese Thamo, Min Wu, and Xinping Yi. A survey of safety and trustworthiness of deep neural networks: Verification,

testing, adversarial attack and defence, and interpretability. *Computer Science Review*, 37:100270, 2020.

[102] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Deepxplore: Automated whitebox testing of deep learning systems. In *proceedings of the 26th Symposium on Operating Systems Principles*, pages 1–18, 2017.

[103] Kosta Serebryany. Continuous fuzzing with libfuzzer and addresssanitizer. In *2016 IEEE Cybersecurity Development (SecDev)*, pages 157–157. IEEE, 2016.

[104] Xiaofei Xie, Lei Ma, Felix Juefei-Xu, Minhui Xue, Hongxu Chen, Yang Liu, Jianjun Zhao, Bo Li, Jianxiong Yin, and Simon See. Deephunter: a coverage-guided fuzz testing framework for deep neural networks. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 146–157, 2019.

[105] Zenan Li, Xiaoxing Ma, Chang Xu, and Chun Cao. Structural coverage criteria for neural networks could be misleading. In *2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, pages 89–92. IEEE, 2019.

[106] Fabrice Harel-Canada, Lingxiao Wang, Muhammad Ali Gulzar, Quanquan Gu, and Miryung Kim. Is neuron coverage a meaningful measure for testing deep neural networks? In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 851–862, 2020.

[107] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[108] Xin Liu, Huanrui Yang, Ziwei Liu, Linghao Song, Hai Li, and Yiran Chen. Dpatch: An adversarial patch attack on object detectors. *arXiv preprint arXiv:1806.02299*, 2018.

[109] Pradeep Rathore, Arghya Basak, Sri Harsha Nistala, and Venkataramana Runkana. Untargeted, targeted and universal adversarial attacks and defenses on time series. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2020.

[110] Anirban Chakraborty, Manaar Alam, Vishal Dey, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. A survey on adversarial attacks and

defences. *CAAI Transactions on Intelligence Technology*, 6(1):25–45, 2021.

[111] Tom B Brown, Dandelion Mané, Aurko Roy, Martín Abadi, and Justin Gilmer. Adversarial patch. *arXiv preprint arXiv:1712.09665*, 2017.

[112] Mikhail Pautov, Grigorii Melnikov, Edgar Kaziakhmedov, Klim Kireev, and Aleksandr Petiushko. On adversarial patches: real-world attack on arcface-100 face recognition system. In *2019 International Multi-Conference on Engineering, Computer and Information Sciences (SIBIRCON)*, pages 0391–0396. IEEE, 2019.

[113] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[114] Lennart Ries, Philipp Rigoll, Thilo Braun, Thomas Schulik, Johannes Daube, and Eric Sax. Trajectory-based clustering of real-world urban driving sequences with multiple traffic objects. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pages 1251–1258. IEEE, 2021.

[115] Trevor Darrell, Marius Kloft, Massimiliano Pontil, Gunnar Rätsch, and Erik Rodner. Machine learning with interdependent and non-identically distributed data. In *Dagstuhl Reports*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

[116] Borna Jelic, Ratko Grbic, Mario Vranjes, and David Mijic. Can we replace real-world with synthetic data in deep learning-based adas algorithm development? *IEEE Consumer Electronics Magazine*, 2021.

[117] Sujan Gannamaneni, Sebastian Houben, and Maram Akila. Semantic concept testing in autonomous driving by extraction of object-level annotations from carla. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1006–1014, 2021.

[118] Farzan Erlik Nowruzi, Prince Kapoor, Dhanvin Kolhatkar, Fahed Al Hassanat, Robert Laganiere, and Julien Rebut. How much real data do we actually need: Analyzing object detection performance using synthetic and real data. *arXiv preprint arXiv:1907.07061*, 2019.

[119] Adrien Gaidon, Qiao Wang, Yohann Cabon, and Eleonora Vig. Virtual worlds as proxy for multi-object tracking analysis. In *Proceedings of*

the IEEE conference on computer vision and pattern recognition, pages 4340–4349, 2016.

[120] Phillip Thomas, Lars Pandikow, Alex Kim, Michael Stanley, and James Grieve. Open synthetic dataset for improving cyclist detection, 2021.

[121] Parallel domain synthetic data improves cyclist detection, 2021.

[122] IPG Automotive. *IPG CarMaker*, accessed April 1, 2022). `https://ipg-automotive.com/de/`.

[123] Stephan R Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. In *European conference on computer vision*, pages 102–118. Springer, 2016.

[124] Craig Kolb, Don Mitchell, and Pat Hanrahan. A realistic camera model for computer graphics. In *Proceedings of the 22nd annual conference on computer graphics and interactive techniques*, pages 317–324, 1995.

[125] Matthias Hullin, Elmar Eisemann, Hans-Peter Seidel, and Sungkil Lee. Physically-based real-time lens flare rendering. In *ACM SIGGRAPH 2011 papers*, pages 1–10, 2011.

[126] Intel Corporation. Intel ospray. `https://www.ospray.org`, 2019. accessed June 18th 2023.

[127] David Isele and Akansel Cosgun. Transferring autonomous driving knowledge on simulated and real intersections. *arXiv preprint arXiv:1712.01106*, 2017.

[128] Rui Gong, Dengxin Dai, Yuhua Chen, Wen Li, Danda Pani Paudel, and Luc Van Gool. Analogical image translation for fog generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 1433–1441, 2021.

[129] Yuhua Chen, Wen Li, Xiaoran Chen, and Luc Van Gool. Learning semantic segmentation from synthetic data: A geometrically guided input-output adaptation approach. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1841–1850, 2019.

[130] Xi Guo, Zhicheng Wang, Qin Yang, Weifeng Lv, Xianglong Liu, Qiong Wu, and Jian Huang. Gan-based virtual-to-real image translation for urban scene semantic segmentation. *Neurocomputing*, 394:127–135, 2020.

[131] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. In *Conference on robot learning*, pages 1–16. PMLR, 2017.

[132] Fei-Ching Kuo, Tsong Yueh Chen, Huai Liu, and Wing Kwong Chan. Enhancing adaptive random testing for programs with high dimensional input domains or failure-unrelated parameters. *Software Quality Journal*, 16(3):303–327, 2008.

[133] Julien Bect, David Ginsbourger, Ling Li, Victor Picheny, and Emmanuel Vazquez. Sequential design of computer experiments for the estimation of a probability of failure. *Statistics and Computing*, 22(3):773–793, 2012.

[134] Julian Hay, Lars Schories, Eric Bayerschen, Peter Wimmer, Oliver Zehbe, Stefan Kirschbichler, and Jörg Fehr. Application of data-driven surrogate models for active human model response prediction and restraint system optimization. *Frontiers in Applied Mathematics and Statistics*, 9:1156785, 2023.

[135] Oliver Bühler and Joachim Wegener. Evolutionary functional testing. *Computers & Operations Research*, 35(10):3144–3160, 2008.

[136] Halil Beglerovic, Michael Stolz, and Martin Horn. Testing of autonomous vehicles using surrogate models and stochastic optimization. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–6. IEEE, 2017.

[137] Paweł Skruch, Marcin Szelest, and Paweł Kowalczyk. An approach for evaluating the completeness of the test scenarios for the vehicle environmental perception-based systems. In *2021 25th International Conference on Methods and Models in Automation and Robotics (MMAR)*, pages 133–138. IEEE, 2021.

[138] Thilo Braun, Lennart Ries, Franziska Körtke, Lara Ruth Turner, Stefan Otten, and Eric Sax. Collection of requirements and model-based approach for scenario description. In *VEHITS*, pages 634–645, 2021.

[139] Maike Scholtes, Lukas Westhofen, Lara Ruth Turner, Katrin Lotto, Michael Schuldes, Hendrik Weber, Nicolas Wagener, Christian Neurohr, Martin Herbert Bollmann, Franziska Körtke, et al. 6-layer model for a structured description and categorization of urban traffic and environment. *IEEE Access*, 9:59131–59147, 2021.

[140] FT Chan, Tsong Yueh Chen, IK Mak, and Yuen-Tak Yu. Proportional sampling strategy: guidelines for software testing practitioners. *Information and Software Technology*, 38(12):775–782, 1996.

[141] Charles Colbourn. Combinatorial aspects of covering arrays. *Le Matematiche*, 59(1, 2):125–172, 2004.

[142] Richard Kuhn, Raghu Kacker, and Yu Lei. *Introduction to combinatorial testing*. CRC press, 2013.

[143] Priyanka Paygude and Shashank Joshi. Use of evolutionary algorithm in regression test case prioritization: A review. In *International conference on Computer Networks, Big data and IoT*, pages 56–66. Springer, 2018.

[144] Andrew F Tappenden and James Miller. A novel evolutionary approach for adaptive random testing. *IEEE Transactions on Reliability*, 58(4):619–633, 2009.

[145] Alessio Gambi, Marc Müller, and Gordon Fraser. Asfault: Testing self-driving car software using search-based procedural content generation. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 27–30. IEEE, 2019.

[146] Vincent Dubourg, Bruno Sudret, and Franois Deheeger. Metamodel-based importance sampling for structural reliability analysis. *Probabilistic Engineering Mechanics*, 33:47–57, 2013.

[147] David Lechevalier, Steven Hudak, Ronay Ak, Y Tina Lee, and Sebti Foufou. A neural network meta-model and its application for manufacturing. In *2015 IEEE International Conference on Big Data (Big Data)*, pages 1428–1435. IEEE, 2015.

[148] Manolis Papadrakakis and Nikos Lagaros. Reliability-based structural optimization using neural networks and monte carlo simulation. *Computer methods in applied mechanics and engineering*, 191(32):3491–3507, 2002.

[149] Tai Song, Huaguo Liang, Ying Sun, Zhengfeng Huang, Maoxiang Yi, Xiangsheng Fang, and Aibin Yan. Novel application of deep learning for adaptive testing based on long short-term memory. In *2019 IEEE 37th VLSI Test Symposium (VTS)*, pages 1–6. IEEE, 2019.

[150] Jean-Marc Bourinet, François Deheeger, and Maurice Lemaire. Assessing small failure probabilities by combined subset simulation and support vector machines. *Structural Safety*, 33(6):343–353, 2011.

[151] Tsong Yueh Chen, Hing Leung, and Ieng Kei Mak. Adaptive random testing. In *Annual Asian Computing Science Conference*, pages 320–329. Springer, 2004.

[152] Min Yan, Li Wang, and Aiguo Fei. Artdl: Adaptive random testing for deep learning systems. *IEEE Access*, 8:3055–3064, 2019.

[153] Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra, and Yoshua Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th international conference on Machine learning*, pages 473–480, 2007.

[154] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

[155] Erik Bochinski, Tobias Senst, and Thomas Sikora. Hyper-parameter optimization for convolutional neural network committees based on evolutionary algorithms. In *2017 IEEE international conference on image processing (ICIP)*, pages 3924–3928. IEEE, 2017.

[156] Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.

[157] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020.

[158] CARLA. *Release 0.9.13*, accessed March 24, 2022. `http://carla.org/2021/11/16/release-0.9.13/`.

[159] Prabhjot Kaur, Samira Taghavi, Zhaofeng Tian, and Weisong Shi. A survey on simulators for testing self-driving cars. In *2021 Fourth International Conference on Connected and Autonomous Driving (MetroCAD)*, pages 62–70. IEEE, 2021.

[160] Epic Games. *Unreal Engine 4*, accessed March 24, 2022. `https://www.unrealengine.com`.

[161] Lucas Fonseca Alexandre de Oliveira, Martin Meywerk, Lars Schories, Maria Meier, Ramakrishna Nanjundaiah, Paulthi Victor, Francesco Foglino, Mark Carroll, and Arunaachalam Muralidharan. Influence of different pedestrian behavior models on the performance assessment of autonomous emergency braking (aeb) systems via virtual simulation.

In *Proceedings of the 7th International Digital Human Modeling Symposium (DHM 2022) and Iowa Virtual Human Summit 2022*, volume 7. University of Iowa, 2022.

[162] *Deeepmotion Animate 3D official website*, accessed May 25, 2023. `https://www.deepmotion.com/animate-3d`.

[163] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[164] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[165] YOLOv5: Unofficial PyTorch implementation. `https://docs.ultralytics.com/yolov5/`. Accessed: April 1, 2024.

[166] Mikhail G Lobanov and Dmitry L Sholomov. Application of shared backbone dnns in adas perception systems. In *Thirteenth International Conference on Machine Vision*, volume 11605, pages 619–627. SPIE, 2021.

[167] Chien-Yao Wang, Hong-Yuan Mark Liao, Yueh-Hua Wu, Ping-Yang Chen, Jun-Wei Hsieh, and I-Hau Yeh. Cspnet: A new backbone that can enhance learning capability of cnn. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 390–391, 2020.

[168] Konstantinos Bousmalis, Alex Irpan, Paul Wohlhart, Yunfei Bai, Matthew Kelcey, Mrinal Kalakrishnan, Laura Downs, Julian Ibarz, Peter Pastor, Kurt Konolige, et al. Using simulation and domain adaptation to improve efficiency of deep robotic grasping. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 4243–4250. IEEE, 2018.

[169] Huayao Wu, Changhai Nie, Justyna Petke, Yue Jia, and Mark Harman. An empirical comparison of combinatorial testing, random testing and adaptive random testing. *IEEE Transactions on Software Engineering*, 46(3):302–320, 2020.

[170] Jiwoong Choi, Ismail Elezi, Hyuk-Jae Lee, Clement Farabet, and Jose M Alvarez. Active learning for deep object detection via probabilistic modeling. *arXiv preprint arXiv:2103.16130*, 2021.

[171] Nicolas Jourdan, Eike Rehder, and Uwe Franke. Identification of uncertainty in artificial neural networks. In *Proceedings of the 13th Uni-DAS eV Workshop Fahrerassistenz und automatisiertes Fahren*, page 12, 2020.

[172] Retrospect. *How Safe is Safe Enough? - According to SOTIF*, 2020 (accessed November 29, 2023). `https://www.retrospectav.com/blog/how-safe-is-safe-enough`.

[173] Inc. Ansys. *Ansys AVxcelerate Sensors*, accessed December 15, 2023). `https://www.ansys.com/products/av-simulation/ansys-avxcelerate-sensors`.

[174] dSPACE GmbH. *Physics-Based Sensor Models for High-Fidelity Simulation*, accessed December 15, 2023). `https://www.dspace.com/de/gmb/home/learning-center/recordings/physics-based-sensor-models-fo.cfm`.

[175] Dhanesh Padmanabhan, Harish Agarwal, John E Renaud, and Stephen M Batill. A study using monte carlo simulation for failure probability calculation in reliability-based optimization. *Optimization and Engineering*, 7:297–316, 2006.

[176] Siu-Kui Au and James L Beck. Estimation of small failure probabilities in high dimensions by subset simulation. *Probabilistic engineering mechanics*, 16(4):263–277, 2001.

[177] MathWorks. *Getting Started with YOLO v4*, 2020 (accessed August 15, 2022). `https://www.mathworks.com/help/vision/ug/getting-started-with-yolo-v4.html`.

[178] Tom Gasser, Clemens Arzt, Mihiar Ayoubi, Arne Bartels, Lutz Bürkle, Jana Eier, Frank Flemisch, Dirk Häcker, Tobias Hesse, Werner Huber, et al. Rechtsfolgen zunehmender fahrzeugautomatisierung. *Berichte der Bundesanstalt für Straßenwesen. Unterreihe Fahrzeugtechnik*, 2012.

[179] Peter Roelants. *Gaussian processes - From scratch*, 2018 (accessed August 01, 2022). `https://peterroelants.github.io/posts/gaussian-process-tutorial/#Sidenotes`.

[180] ISO Central Secretary. Road vehicles - functional safety. Standard ISO 26262:2018, International Organization for Standardization, Geneva, CH, 2018.

[181] Jasmin Breitenstein, Jan-Aike Termöhlen, Daniel Lipinski, and Tim Fingscheidt. Corner cases for visual perception in automated driving: Some

guidance on detection approaches. *arXiv preprint arXiv:2102.05897*, 2021.

[182] IPG Automotive. *IPG MovieNX*, accessed April 1, 2022). `https://ipg-automotive.com/de/produkte-loesungen/software/movienx/`.

[183] *CARLA Documentation*, accessed March 24, 2022. `https://carla.readthedocs.io/en/latest/`.

[184] Zhang-Wei Hong, Chen Yu-Ming, Shih-Yang Su, Tzu-Yun Shann, Yi-Hsiang Chang, Hsuan-Kung Yang, Brian Hsi-Lin Ho, Chih-Chieh Tu, Yueh-Chuan Chang, Tsu-Ching Hsiao, et al. Virtual-to-real: Learning to control in visual semantic segmentation. *arXiv preprint arXiv:1802.00285*, 2018.

[185] Jeroen Van Baar, Alan Sullivan, Radu Cordorel, Devesh Jha, Diego Romeres, and Daniel Nikovski. Sim-to-real transfer learning using robustified controllers in robotic tasks involving complex dynamics. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6001–6007. IEEE, 2019.

[186] Tsong Yueh Chen, Fei-Ching Kuo, and Robert Merkel. On the statistical properties of testing effectiveness measures. *Journal of Systems and Software*, 79(5):591–601, 2006.

[187] Qutub Syed Sha, Oliver Grau, and Korbinian Hagn. Dnn analysis through synthetic data variation. In *Proceedings of the 4th ACM Computer Science in Cars Symposium*, pages 1–10, 2020.

[188] Jinfu Chen, Qihao Bao, TH Tse, Tsong Yueh Chen, Jiaxiang Xi, Chengying Mao, Minjie Yu, and Rubing Huang. Exploiting the largest available zone: A proactive approach to adaptive random testing by exclusion. *IEEE Access*, 8:52475–52488, 2020.

[189] Valentina Mușat, Ivan Fursa, Paul Newman, Fabio Cuzzolin, and Andrew Bradley. Multi-weather city: Adverse weather stacking for autonomous driving. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2906–2915, 2021.

[190] Philipp Oberdiek, Matthias Rottmann, and Gernot A Fink. Detection and retrieval of out-of-distribution objects in semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 328–329, 2020.

[191] Dario Fontanel, Fabio Cermelli, Massimiliano Mancini, and Barbara Caputo. Detecting anomalies in semantic segmentation with prototypes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 113–121, 2021.

Appendices

A Convolutional Neural Network Architecture

The concept of artificial neural networks is heavily based on information extraction from data points. Neurons are connected to prior layers to identify feature information they are trained to detect. The neuron $x_{1,1}$ in Fig. 2.9 determines whether a data-point satisfies the condition $0.5 < x_{0,1} + x_{0,2}$. The neuron passes on the information that this feature is present if it does. A wide range of layers can be concatenated to build a neural network. Fig. .1 displays the standard layers of the convolutional neural networks used in this work:

a) Fully-connected (dense) Layer: The traditional structure of the perceptron. Each neuron of layer $i + 1$ is connected to each neuron of layer i by individual weights. An activation function processes the weighted values plus the bias to examine the neurons' feature's presence.

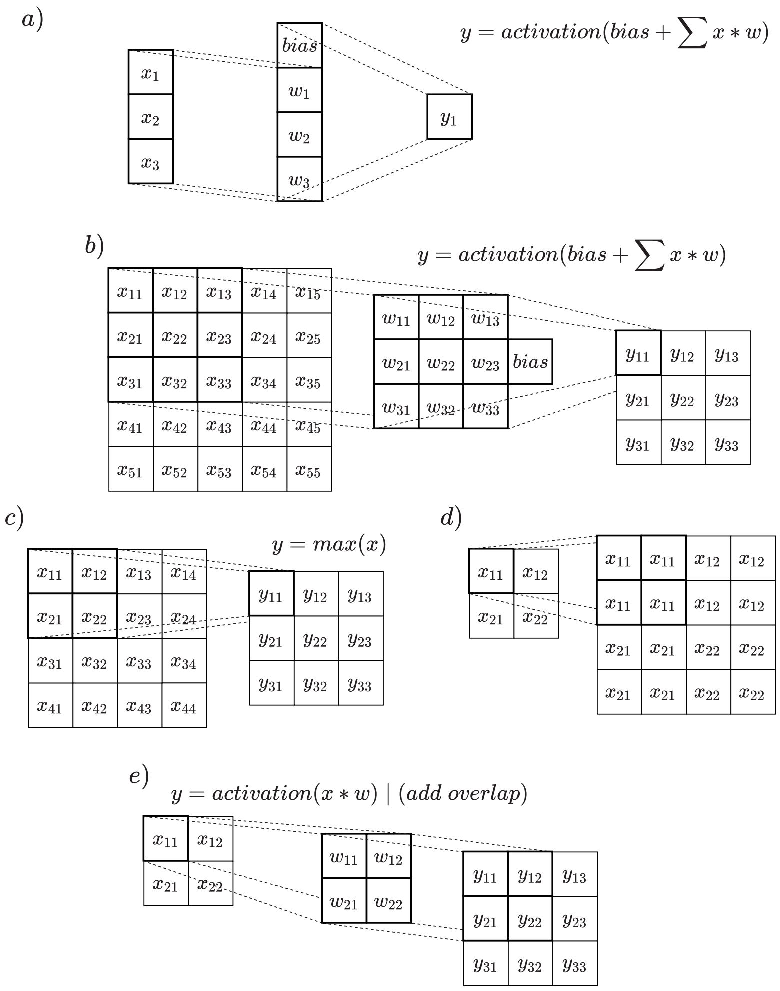
b) Convolutional Layer: The core layer of convolutional neural networks. Instead of dense connections between two layers, the neurons of layer $i + 1$ are locally or sparsely connected to neurons in layer i. This is done by sliding a filter matrix over the input image for computer vision. The small patch is multiplied with the weight matrix and added to the bias. The output of the receptive neuron is the activation function of this value. Hence, each filter matrix is applied to each patch, checking for its feature's presence.

c) Max-Pooling Layer: Pooling is another standard operation in convolutional neural networks. Like the convolutional layer, the pooling layer can be seen as a window that slides over the input layer, performing an operation. However, pooling does not include a filter matrix. Usually, the sliding window's maximum value is selected to be the output. Since a high value represents a strong presence of a feature in the input layer, max-pooling can be used for down-sampling. Suppose a certain feature is present in an area. In that case, the max-pooling layer will transfer the highest value into the next layer. Pooling layers do require neither weights nor an activation function.

d) Up-Sampling Layer: Up-sampling is a simple operation to transform the input to a higher dimension. Each entry of the input feature map is repeated, depending on the up-sampling layer's window size.

e) Transposed convolutional Layer: A transposed convolutional layer is based on the concept of transposed local connectivity. Each input value is multiplied to a kernel matrix before sliding the window over the input. The overlaps are added together. After completing the process, each value of the resulting matrix is fed to an activation function to get the output matrix.

Fig. .2 depicts the architecture of AlexNet [84] as an example of how these elements can be combined for image classification.



a)
$y = activation(bias + \sum x * w)$
x_1 x_2 x_3
$bias$ w_1 w_2 w_3
y_1
b)
$y = activation(bias + \sum x * w)$
x_{11} … x_{55}
w_{11} w_{12} w_{13} w_{21} w_{22} w_{23} $bias$ w_{31} w_{32} w_{33}
y_{11} … y_{33}
c)
$y = max(x)$
x_{11} … x_{44}
y_{11} … y_{33}
d)
x_{11} x_{12} x_{21} x_{22}
e)
$y = activation(x * w) \mid (add\ overlap)$
x_{11} x_{12} x_{21} x_{22}
w_{11} w_{12} w_{21} w_{22}
y_{11} … y_{33}

Figure .1: Standard building blocks for convolutional neural networks. Activation is done element-wise. a) *Fully connected layer*, b) *Convolutional layer*, c) *Pooling layer*, d) *Up-sampling layer*, e) *Transposed convolutional layer*.

Layer	**Window Size**	**Stride**	**Channels**	**Output Size**
Input				$224 \times 224 \times 3$
Convolution	11×11	4	96	$55 \times 55 \times 96$
MaxPool	3×3	2		$27 \times 27 \times 96$
Convolution	5×5	1	256	$27 \times 27 \times 256$
MaxPool	3×3	2		$13 \times 13 \times 256$
Convolution	3×3	1	384	$13 \times 13 \times 384$
Convolution	3×3	1	384	$13 \times 13 \times 384$
Convolution	3×3	1	256	$13 \times 13 \times 256$
MaxPool	3×3	2		$6 \times 6 \times 256$
Dense			4096	4096
Dense			4096	4096
Dense			1000	1000

Figure .2: AlexNet-architecture [84]: The network calculates a classification vector to the input image containing the class-scores for all 1000 classes. The image is assigned to the class with the highest score.

Activation Function

The activation function of a neuron processes the weighted input to the neuron. There are different concepts for the activation. For computer vision tasks, activation functions are mostly monotonous increasing. A high input value to the neuron should lead to a high output value. A high activation value is usually interpreted as the feature's presence, the neuron is trained to detect. Fig. .3 displays the common activation functions used for this research.

a) Step function: Traditional activation function, introduced by Pitt & Mcculloch [78]. The function returns 0 for negative input values and 1 for positive input values.

b) Sigmoid function: The sigmoid function is a special case of the logistic function. It is used as a continuous derivable approximation of the step function. The function's values are in the interval $(0, 1)$. Hence a negative input value to this activation results in a near-zero output. In contrast, a high input value results in a near-one output. Hence this activation function returns whether its neuron's feature is present or not.

c) Hyperbolic tangent (Tanh): The tanh function has a similar shape to the step function. In contrast to the sigmoid function, the tanh function's values lie in the interval $(-1, 1)$. In the positive domain, the function returns 1, indicating the neuron's feature. In the negative domain, the function returns -1, which can be interpreted as a penalty term.

d) Rectified linear unit (ReLU): Trimmed linear function. The function is equal to zero in the negative domain and equal to its identity function in the positive domain. Like the step function, a negative input value is interpreted as the absence of the feature. In contrast, a positive input indicates the feature's presence with higher intensity for high input values. In practice, the ReLU function is often approximated by the softplus function, which is continuously differentiable.

e) Leaky ReLU: The leaky ReLU function is a slight modification of the ReLU function. While the function is equal to the ReLU function in the positive domain, the function is a linear function with a slope $\alpha = 0.01$. Like the negative output of the tanh function for the negative domain, the absence of a feature can be interpreted as penalty input to the next layer.

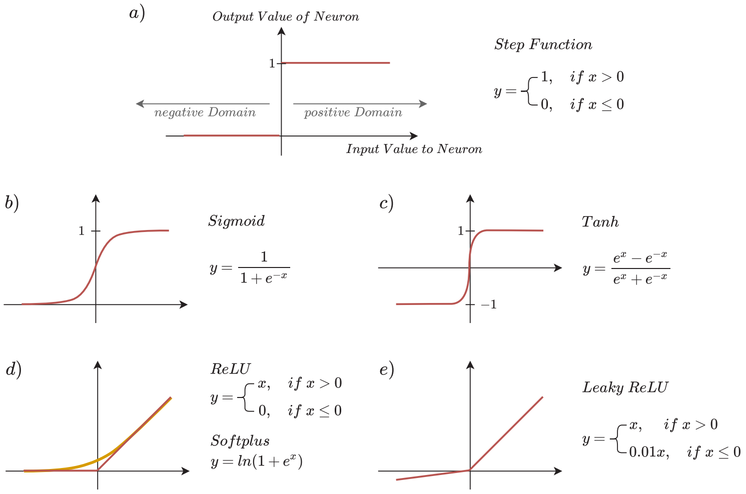


Figure .3: Standard activation functions for convolutional neural networks. a) Traditional step function. b) Sigmoid function (Special case of logistic function). c) Hyperbolic tangent function. d) Rectified linear unit and softplus function (continuously differentiable approximation). e) Leaky rectified linear unit.

Model Parameter Optimization

The accuracy of neural networks to perform their tasks depends on two types of parameters:

Hyperparameters: Also called architectural parameters. In general, hyperparameters describe a neural network's architecture, such as the number and type of layers, the loss function, and the optimization specifications.

Model Parameters: Model parameters describe the numerical values used to compute a neural network's output. For example, the weights of the network layers belong to the model parameters.

The number and type of model parameters depend on hyperparameters and the network's overall structure. During weight optimization, a fixed architecture is used. In this case, the neural network is defined by

$$\mathcal{N} : \mathcal{I} \times \mathcal{W} \to \mathcal{O}, \tag{.1}$$

where $\mathcal{I}$ is the input space, $\mathcal{W}$ is the networks' weight space and $\mathcal{O}$ is the output space. Given an input data point $x \in \mathcal{I}$ and the ground-truth output value for this input $\hat{y} \in \mathcal{O}$, the training process of a neural network is defined by the optimization problem

$$\hat{w} = \underset{w \in \mathcal{W}}{\arg\min}\, \mathcal{L}(\mathcal{N}(x, w), \hat{y}) = \underset{w \in \mathcal{W}}{\arg\min}\, \mathcal{L}(w), \tag{.2}$$

where $\mathcal{L}$ is the so-called loss function that measures the deviation between the ground truth and the networks' prediction. Hence, during training the weights are modified to minimize the loss for a given data point/ground truth pair. A popular choice for the loss function is the mean-squared-error (MSE), which is defined by

$$\mathcal{L}(w) = MSE(w) = ||\mathcal{N}(x, w) - \hat{y}||_2, \tag{.3}$$

with $|| \bullet ||_2$ being the euclidean norm.

B Gaussian Process Regression

Similar to the other regression models, using Gaussian processes for regression tasks requires assumptions on the data distribution. A stochastic process can be interpreted as a distribution of functions. For Gaussian processes, this distribution is some n-dimensional normal distribution. The distribution of realizations is then defined by

$$f(X) \sim \mathcal{N}(m(X), k(X, X)), \tag{.4}$$

with $X = \{x_1, ..., x_n\}$ being the functions input domain, $m(X)$ being the distributions mean vector function, and $k(X, X)$ being the distributions kernel function that generates the covariance matrix. The mean function is defined as the expected value of a realization at the position x:

$$m(x) = \mathbb{E}[f(x)]. \tag{.5}$$

This expected value denotes the prediction that the Gaussian process returns for a given point in the input space and can be interpreted analogously to the output of a linear regression model or a neural network.

The kernel function returns the covariance between the realization at two positions x and x'. The covariance matrix is then calculated element-wise with

$$k(x, x') = \mathbb{E}[(f(x) - m(x))(f(x') - m(x'))]. \tag{.6}$$

The purpose of the kernel function is to describe the connection between the different positions of a realization function along the input axis. Furthermore, the kernel $k(x, x)$ denotes the point-wise variance of the Gaussian process and the uncertainty related to the corresponding prediction $m(x)$.

Example: Sampling from Distribution The most popular kernel used for Gaussian process regression is the radial basis function kernel (RBF), with the covariance matrix defined element-wise by

$$k(x_i, x_j) = exp(-\frac{||x_i - x_j||^2}{2\sigma^2}), \tag{.7}$$

with σ^2 being a variance hyperparameter for the Gaussian process. The RBF-covariance matrix for $X = \{0, 0.25, ..., 3.75, 4\} \in \mathbb{R}^{17}$ and $\sigma = 1$ is displayed

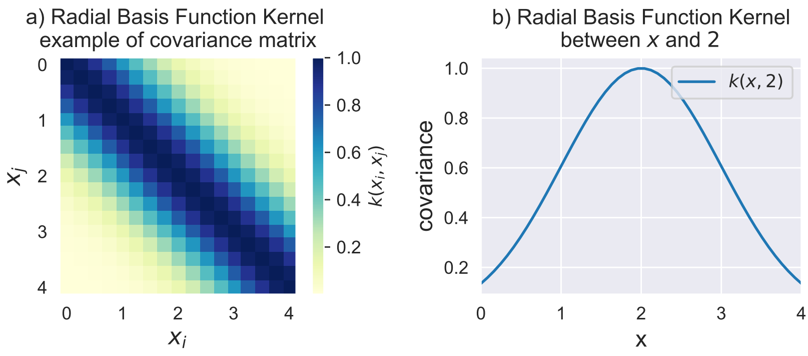


Figure .4: a) Covariance matrix using the radial basis function for input domain $X = \{0, 0.25, \ldots, 3.75, 4\}$. b) Pair-wise covariance $k(x, x')$ between $x \in [0, 4]$ and $x' = 2$. Figure framework by [179].

in Fig. .4 a). The higher the difference between the position in the input domain x and x', the lower the covariance between $f(x)$ and $f(x')$. This effect is displayed in Fig. .4 b). By assuming the mean function being $m(x) = 0$, the Gaussian process can be described by the distribution

$$f(x) \sim \mathcal{N}(\mu, \Sigma), \tag{.8}$$

with μ being a 17-dimensional vector of zeros and Σ being the matrix from Fig. .4 a). Using a zero vector for the mean is a standard starting point if no time-dependent shift is assumed. Fig. .5 shows five realizations drawn from this distribution.

From Prior to Posterior: Adapting to observed Data (Noise-free) Fig. .5 shows the Gaussian processes' distribution prior to any observed data, based solely on the pre-defined mean function and covariance function, hence named prior distribution. The core purpose of regression is to generate predictions for the output of a target function based on already observed data, which is done by generating a so-called posterior distribution that incorporates several data points already observed.

Assume $x_1 \in \mathbb{R}^{n_1}$ being a vector input values for which the vector of response values $y_1 \in \mathbb{R}^{n_1}$ is known. Now assume $x_2 \in \mathbb{R}^{n_2}$ being a vector with input

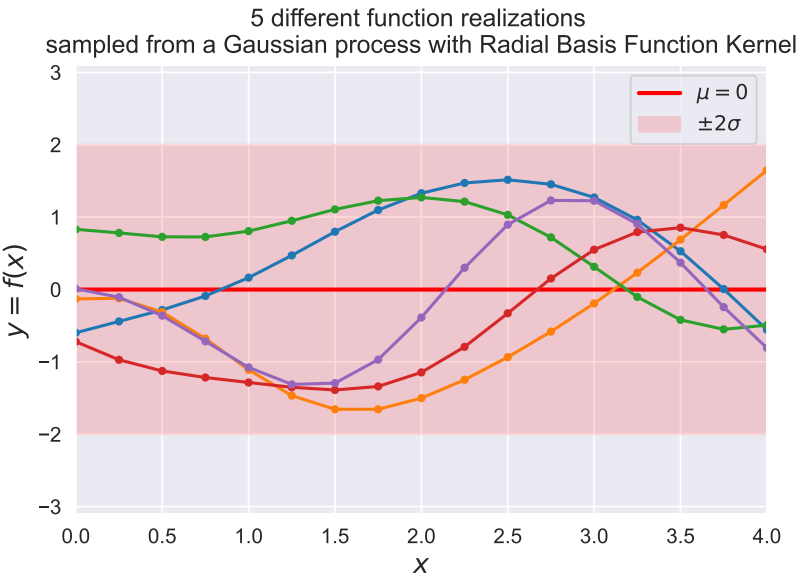


Figure .5: Five different function realizations sampled from a Gaussian process with input domain $X = \{0, 0.25, \ldots, 3.75, 4\}$, radial basis function kernel as covariance function and mean function $m(x) = 0$. The strong red line denotes the mean function, while the red area approximately denotes the 95% confidence interval via $\mu \pm 2\sigma$. Figure framework by [179].

values for which the vector of response values $y_2 \in \mathbb{R}^{n_2}$ is unknown and has to be estimated. As stated in Eq. .8 the prior distribution is given by

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}\right), \tag{.9}$$

with $\mu_1 = m(x_1)$ and $\mu_2 = m(x_2)$ being the prior mean vectors and $\Sigma_{11} = k(x_1, x_1)$, $\Sigma_{22} = k(x_2, x_2)$, and $\Sigma_{12} = k(x_1, x_2) = \Sigma_{21}^T$ being the covariance matrices.

The goal is now to generate the posterior given through the conditional distribution

$$p(y_2 | y_1, x_1, x_2) \sim \mathcal{N}(\mu_{2|1}, \Sigma_{2|1}), \tag{.10}$$

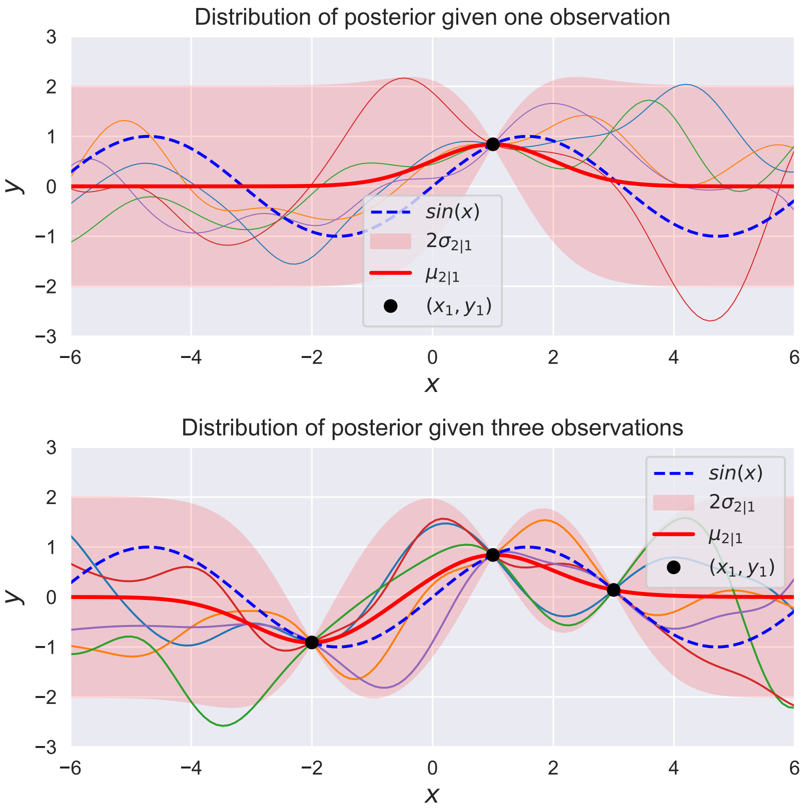


Figure .6: Posterior distribution of Gaussian process with $m(x) = 0$ and RBF kernel given one and three real observations from the underlying function $f(x) = sin(x)$. Additionally, five samples from each posterior are drawn and plotted. Compare with Fig. .5 for prior distribution. Figure framework by [179].

with $\mu_{2|1}$ being the conditional mean vector and $\Sigma_{2|1}$ being the covariance matrix of y_2 for input values x_2 given the observations (y_1, x_1). Calculating

$\mu_{2|1}$ and $\Sigma_{2|1}$ can be done using the established formulas for conditional normal distributions, for which all components are known:

$$\mu_{2|1} = \mu_2 + \Sigma_{21}\Sigma_{11}^{-1}(y_1 - \mu_1) \tag{.11}$$

$$\Sigma_{2|1} = \Sigma_{22} - \Sigma_{21}\Sigma_{11}^{-1}\Sigma_{12} \tag{.12}$$

Fig. .6 shows this process. In this example, the underlying function of observations drawn that shall be approximated is $f(x) = sin(x)$. There are two runs of the regression using one and three observations. The original mean function is $m(x) = 0$, and the RBF kernel has been used.

In this setup the conditional mean $\mu_{2|1}$ for training data-points $x_2 = x_1$, is equal to the observed response value y_1 with variance 0, which easily can be proven, since $\Sigma_{11} = \Sigma_{22} = \Sigma_{21} = \Sigma_{12}$ and $\mu_1 = \mu_2 = 0$:

$$\begin{aligned}\mu_{2|1} =&\mu_2 + \Sigma_{21}\Sigma_{11}^{-1}(y_1 - \mu_1)\\ \mu_{2|1} =&\mu_2 + y_1 - \mu_1 \quad | \quad \Sigma_{21} = \Sigma_{11}\\ \mu_{2|1} =&y_1 \quad | \quad \mu_2 = \mu_1\end{aligned} \tag{.13}$$

$$\begin{aligned}\Sigma_{2|1} =&\Sigma_{22} - \Sigma_{21}\Sigma_{11}^{-1}\Sigma_{12}\\ \Sigma_{2|1} =&\Sigma_{22} - \Sigma_{12} \quad | \quad \Sigma_{21} = \Sigma_{11}\\ \Sigma_{2|1} =&0 \quad | \quad \Sigma_{22} = \Sigma_{12}.\end{aligned} \tag{.14}$$

Adapting to observed Data assuming Noise To this point, there has been the assumption that the observations are generated deterministic, hence there is no noise (see results of Eq. .13 and Eq. .14). There are applications that cannot rely on this assumption, like real world observations in noisy experiments.

Adapting the Gaussian process regression to an noisy environment can be done by adding assuming $f(x_1) = y_1 + \epsilon$, with $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$. While this does not change the mean vector, the assumed variance is added to the covariance kernel of the observations

$$\Sigma_{11} = k(x_1, x_1) + \sigma_\epsilon^2 I. \tag{.15}$$

By applying this concept to the example Fig. .6 with σ_ϵ^2 generates a noisy Gaussian process, which is displayed in Fig. .7.

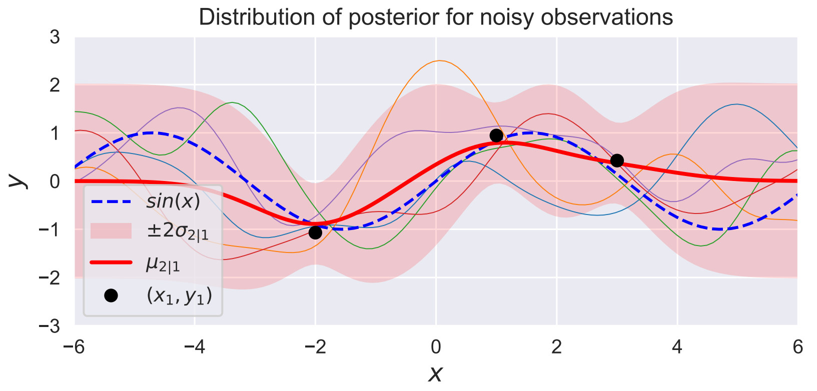


Figure .7: Posterior distribution of Gaussian process with $m(x) = 0$ and RBF kernel given one and three real observations from the underlying function $f(x) = sin(x)$ with added noise drawn from $\mathcal{N}(0, 0.5 * I)$. Figure framework by [179].

The main advantage of adjusting for noise in Gaussian processes is that outliers do not cause major shifts in the conditional mean vector $\mu_{2|1}$. Due to Eq. .13 the noise free Gaussian process forces the regression estimation through every train data point.

While the ability to model uncertainty in the meta-models prediction is quite useful, using the model causes some capacity problems. As shown in Eq. .11, making a prediction requires the inversion of the covariance matrix Σ_{11}. Hence, the computational cost for making a prediction increases with each training data point.