Technische Universität München
TUM School of Computation, Information and Technology

Validation of Machine Learning Algorithms by Design with Applications for Automated Driving

Oliver Christoph Gallitz

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung eines

Doktors der Ingenieurwissenschaften (Dr.- Ing.)

genehmigten Dissertation.

Vorsitz: Prof. Dr. phil. nat. Sebastian Steinhorst

Prüfer*innen der Dissertation:

1. Prof. Dr.- Ing. Wolfgang Utschick
2. Prof. Dr.- Ing. Michael Botsch

Die Dissertation wurde am 09.06.2022 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 15.12.2022 angenommen.

Acknowledgements

This thesis emerged during my work at the CARISSMA Institute of Automated Driving. As part of the BayWISS academic forum, this work has been conducted in cooperation with the Professorship of Signal Processing at the Technical University of Munich.

I would like to thank my supervisor Prof. Dr.-Ing. Michael Botsch for his extraordinary support through the years. The dedication and passion he puts into the mentoring of his PhD candidates is highly remarkable and deserves my deepest respect and gratitude. Likewise, I would like to thank my supervisor Prof. Dr.-Ing. Wolfgang Utschick, who has not only been a supervisor, but a valued challenger of ideas and one of the most knowledgeable conversation partners that I have met. I also want to express my gratitude towards my colleague and dear friend Oliver De Candido, with whom I have shared many great experiences over the past years. I could not have wished for a better topic-tandem partner.

Further, I would like to thank my mentor Prof. Dr. rer. nat. Christian Facchi, as well as Prof. Dr. phil. nat. Sebastian Steinhorst for serving as chairman of the examination committee. I want to thank Mr. Firas Lethaus, Dr. Ron Melz and Dr. Dominic Seibert at Audi AG & CARIAD SE for their organizational support and for getting involved in the topics by providing helpful ideas and council. My colleagues have always been supportive and created a pleasant and creative work atmosphere. In particular, I want to thank Tim Elter and Marion Neumeier, as well as Marcel Holdegel, Toshika Srivastava and Jan Martin. In my early academic career, Dr. Andreas Mehrle and Mr. Siegfried Reck fueled my interest in engineering. I want to express my gratitude for your motivation, advice and knowledge.

I dedicate this work to my parents Norbert and Christine Gallitz, as well as my sister Julia Gallitz for their constant support, and for encouraging me to always make the best out of myself. Ultimately, I am deeply grateful for the love and support of my partner Nathalie Wagner, who motivates me when I face difficulties and cheers with me for the successes I celebrate.

I could not have achieved this without you.

Declaration of Previous Publications

Parts of this work have been published by the author in peer reviewed conferences and their corresponding proceedings, as well as in a patent application. In particular, the following publications contain some of the core concepts and ideas presented in this work:

Oliver Gallitz, Oliver de Candido, Michael Botsch, and Wolfgang Utschick. Interpretable feature generation using deep neural networks and its application to lane change detection. In *The 2019 IEEE Intelligent Transportation Systems Conference - ITSC*, pages 3405–3411, [Piscataway, New Jersey], 2019. IEEE.

Oliver Gallitz, Oliver de Candido, Michael Botsch, Ron Melz, and Wolfgang Utschick. Interpretable machine learning structure for an early prediction of lane changes. In *Artificial Neural Networks and Machine Learning – ICANN 2020*, volume 12396 of Lecture Notes in Computer Science, pages 337–349, [Cham], 2020. Springer International Publishing.

Oliver Gallitz, Oliver de Candido, Michael Botsch, and Wolfgang Utschick. Interpretable early prediction of lane changes using a constrained neural network architecture. In *The 2021 IEEE Intelligent Transportation Systems Conference - ITSC*, pages 493-499, [Piscataway, New Jersey], 2021. IEEE.

Oliver Gallitz and Michael Botsch. Verfahren zur validierbaren Voraussage für das Eintreffen von Ereignissen im Straßenverkehr. (102022114218.3). German Patent and Trade Mark Office, Munich. 2022, patent pending.

Contents

List of Figures

List of Tables

1 Introduction

The validatable early prediction of events based on multivariate time series data by using Machine Learning (ML) algorithms is the main topic of this thesis. In Section 1.1, the motivation for research in this field is elaborated. Section 1.2 outlines the contents and major contributions of this work.

1.1 Motivation

The early prediction of Multivariate Time Series (MTS) data has always played a big role in the field of temporal classification. The main task is to predict events as early as possible without a significant loss of performance. The structure of the input data, MTS, consists of multiple signals recorded over time. Many ML architectures have been developed with single-frame classifications in mind, e. g., image classification instead a subsequent series of images as video stream. A structure that can efficiently process the high-dimensional MTS input data is required in order to retain the time information. Applications of early prediction can be found throughout all scientific fields, let it be in the financial sector for crisis predictions [5], in medical sciences for early recognition of cardiac arrests [6], and, of course, in the field of autonomous driving to predict the actions of other traffic participants [7]. Before ML was applied to a wide variety of problems, other popular methods were used to address early prediction tasks. A classic approach for time series classification is the use of shapelets, which can be defined as class-representative sequences [8]. Recent publications on shapelets also consider multivariate input [9, 10]. Shapelets, as subsequences of time series data, can inherit interpretability from the input data. If the input data consists of understandable entities, extracted shapelets can be interpretable as well. Hence, several publications in this field claim the extraction of interpretable features by the use of shapelets [11, 12]. In the past decade, ML has been used in a variety of application fields. ML, as implied by the name, thereby describes the ability of an algorithm to automatically learn patterns and rules within a given dataset, as well as the ability to subsequently detect these patterns in unseen data.

The potential of these algorithms seems to be overwhelming - the ability to generate a desired output by implicitly learning physical or behavioral models, provided structured data is available on the input side. The authors of [13] presented an autonomous vehicle that outputs steering wheel angles and driving pedal positions after being provided front-view camera images. All steps in-between are replaced by a neural network, a common example of an ML algorithm. The advantage of such an end-to-end function is that the required domain knowledge in order to get at least a minimum viable product is immensely reduced. On the downside, however, the high complexity of ML architectures, recently additionally supported by the available computing power of parallel GPU processing, does not allow an instant comprehension of these structures. In other words, where former development processes required field experts and developers to work together, all it takes to develop an end-to-end ML architecture is a feasible dataset and an ML engineer. For comfort applications such as multimedia this is an appealing solution. For safety-critical applications, such as autonomous driving, an end-to-end system with complex ML networks lacks any profound validation methods and is therefore not safe to use.

Most ML methods allow the user to gain an insight into its inner workings and observe parts of the decision making process. This work presents a structure that exploits these insights to generate a fully interpretable ML architecture that can compete with state-of-the-art network designs. By formulating a safety argument using interpretability as a core requirement, the structure becomes validatable. Expert knowledge is only required at the design phase of the architecture. The proposed methods in this work support a validation by design approach.

1.2 Outline and major contributions

The thesis is structured in 6 chapters and starts with the motivation in this chapter. An insight into the safety-realted topics of ML is given in Chapter 2. The structural components for a validatable ML approach, together with according experiments on real-world applications are presented in Chapters 3, 4 and 5. Chapter 6 concludes this thesis with a recapitulation of this work and an outlook.

Chapter 2 starts by explaining the main components of ML algorithms to better place the subsequent definitions and analyses into context. Deeper explanations of the relevant ML techniques can be found in the chapters where they are used. An introduction into the framework of verification and validation follows. The safety processes for classical automotive software function development and their applicability to ML algorithms are analyzed. The main part of this chapter is the formulation of a safety argument that is predominantly defined by interpretability and uncertainty quantification. Furthermore, this chapter introduces two datasets based on real-life recorded traffic scenarios. These datasets will be used as application samples throughout this work.

Chapter 3 proposes an interpretable data representation with enriched information content. This chapter will explain, how two-dimensional MTS samples can be represented as 1-dimensional vectors without forfeiting the time information in the data. The enriched dataset is a combined feature generation and feature selection method. First, unimportant features are discarded and therefore the dataset dimensionality is significantly reduced. In a second step, automatically generated relevant features are added to the dataset. It will be explained, how interpretability is retained throughout these steps and how to evaluate the information gain in the data.

The major contribution of Chapter 4 is an interpretable early prediction structure, based on a Mixture of Experts (MoE) architecture. The concept follows a divide-et-impera paradigm. Incoming data is thereby distributed to several domain experts, which are all trained on explicitly defined and non-overlapping segments of the MTS data. Through the parameter choice of the MoE approach, interpretability is inherited from the underlying expert network structures. For evaluating the early prediction performance, several performance indicators are identified and evaluated. By comparing recurrent neural networks (RNNs) using Gated Recurrent Units (GRUs) and Long-Short-Term-Memory (LSTM) cells, which are both highly uninterpretable structures, the presented method is shown to achieve competitive performance while remaining interpretable.

In Chapter 5, another approach towards interpretability is introduced into the MoE structure. By placing Generalized Radial Basis Function (GRBF) networks as experts, interpretability is established by means of similarity measures. The chapter will explore, how adequate representatives for such a measure can be found. The uniqueness and interpretability of the chosen representatives is analyzed and demonstrated. By introducing similarity measures, the method allows for uncertainty quantification of each decision. A rejection mechanism for uncertain decisions is proposed and its advantages are examined.

Chapter 6 concludes this work by reiterating the interpretability claim found throughout this thesis. An insight into a complementary approach that focuses on diversity methods is given. Ultimately, a viable path towards function validation with ML algorithms is outlined.

1.3 Notation

Throughout this work, matrices will be denoted as bold capital letters and vectors as bold lower case letters. Regularly used mathematical expressions are the convolution operator "$*$", the trace of a matrix $\mathrm{Tr}\{\cdot\}$, the transpose of a matrix $(\cdot)^\top$ and the vector norm $\|\cdot\|$.

2

Basic Machine Learning and Safety Concepts

This chapter introduces the general field of verification and validation, as well as the basic concepts of the methods which are applied in this work. Section 2.1 recapitualtes some of the most common ML structures. A description of the datasets used in this work is provided in Section 2.2. Section 2.3 gives an insight into relevant safety aspects of ML in automated driving functions. In Section 2.4, the necessity of an interpretable ML approach for validatable functions is examined.

2.1 Basic Machine Learning Architectures

This section analyzes a selection of popular and for this work relevant ML methods with respect to several properties, which differentiate these methods from each other. By doing so, a basic understanding for some popular ML algorithms and their respective use is created.

2.1.1 Machine Learning Goals for Automated Driving Function

ML algorithms can be set up to fulfill a variety of tasks. One of the classic goals is classification, where an incoming data point $\boldsymbol{x}$ is assigned to one of K output classes

$$\boldsymbol{x} \mapsto y_c, \quad y_c \in [k_1, k_2, \dots, k_K]. \tag{2.1}$$

Classification tasks in automated driving functions are often implemented in the perception stage. However, prediction tasks, such as maneuver identification can also be solved by standard classification algorithms.

If the goal is not to identify the member of a specific class but to estimate continuous variables, a regression task is imposed. Accordingly, the mapping can be denoted as

$$\boldsymbol{x} \mapsto y_r, \quad \text{with } y_r \in \mathbb{R}. \tag{2.2}$$

Regression in automated driving can be used for predictions such as time-to-collision, crash severity, velocity estimation and several other situations where an explicit continuous scalar is to be determined. These two basic learning goals can be achieved by a variety of networks. Some network architectures, however, have been explicitly designed to fulfill specific learning goals exceptionally well. For the task of time series forecasting, the input domain, as well as the output domain are time series. The formulation of the learning problem with univariate input and output can be defined as

$$\boldsymbol{x} \mapsto \boldsymbol{y}, \tag{2.3}$$

with $\boldsymbol{x} \in \mathbb{R}^{T_{\text{in}}}$ and $\boldsymbol{y} \in \mathbb{R}^{T_{\text{out}}}$ denoting time series of different lengths. Time series forecasting is used for tasks such as trajectory prediction, with RNNs as state-of-the-art performing network structures.

All of the above mentioned goals are typically implemented using supervised learning methods. In supervised learning, each training input is provided together with a label. The labels are used during the training phase of the ML structure to adapt the parameters of the model. If the goal of a ML algorithm is to sort incoming samples into groups of similar data, clustering methods are applied. These methods do not necessarily require labels during training. Instead, clustering methods are designed to identify similarities between samples and group them accordingly. Clustering is used to identify representative samples in datasets, groups of outlier data in open set theory, as well as structures in unknown data.

An important performance indicator of a trained ML model is its generalization ability. The bias-variance tradeoff describes the conflict of training ML models while minimizing two, typically antagonistic, error sources. The model bias is an error that originates from building wrong or insuffiecient relations between the input and the ground truth output during the training phase. A high model variance is prevalent, if the error on two datasets $\mathcal{D}_1$ and $\mathcal{D}_2$, that originate from the same data distribution, is large. In practice, one of the datasets is usually the training set, while the second set contains data unseen by the model during the training and validation phase. The high variance in ML models often originates from design choices that offer an excessive ammount of learnable parameters for a small number of samples in the training set. However, as indicated by [14], it is possible to generate high variance errors with a low number of parameters depending on the choice of the learning model. A high bias often indicates an insuffiecient ammount of parameters to capture the system dynamics, or the choice of an ML model that is unsuitable for the desired learning task. The bias and variance errors are often regarded as a tradeoff, since for the appropriate choice of a learning model, an increase in complexity in terms of adding model parameters can lead to an increase in model variance, as well as a decrease of the model bias and vice versa. Nevertheless, it is possible to generate high variance and high bias errors at the same time as further elaborated by [14]. The expected generalization error is a summation of the bias and the variance errors, together with a third term describing the irreducible error. This third term originates from noise in the data. It is not influenced by model design choices and describes a lower bound of the expected generalization error.

2.1.2 Decision Trees for Classification and Regression Tasks

The concept of Decision Trees (DT) is one of the classic ML methods and subject of several adaptions and improvements. This work focuses on DT in one of their most popular forms, Classification and Regression Trees (CARTs) [15]. This framework proposes binary tree structures, which can be viewed as nested if-else-statements. The resulting layout resembles a tree with branches and leaves, hence the name of this type of structure. A tree is built by splitting the source set into smaller subsets. This process is recursively repeated until a stopping criterion is reached. This criterion could be that all samples in the subset share one common target value, or that further splits would not further improve the performance of the prediction. The approach of growing the tree level by level is called top-down induction. It is an example of a greedy algorithm, as the best split is determined based on the current split node, without regarding any future developments of the tree structure. Binary trees thereby always split the set into exactly two subsets. A split can therefore be depicted as a division of the input space into cuboidal regions along the border of a distinct feature, as shown in Figure 2.1. In

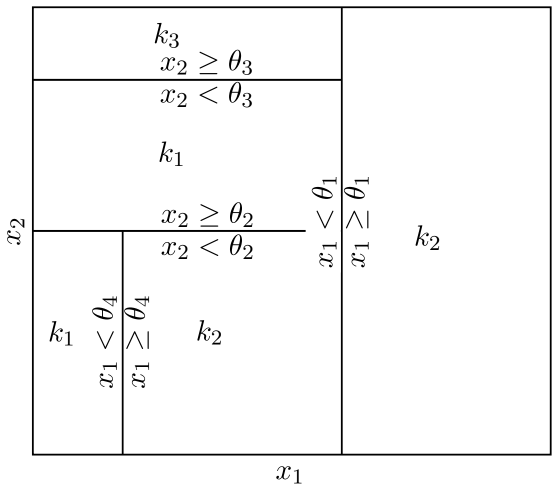


Figure 2.1: Cuboid regions of binary DT splits

this example, two features x_1, x_2 are denoted with the split conditions θ and the leaves with 3 classes k_1, k_2 and k_3.

When growing single DTs, usually all features are available for each split. The quality of each split is evaluated by a criterion which is dependent on the task. For classification, the Gini impurity is the most commonly used function. It describes how many samples in the same node have the same label and becomes 0 for pure nodes, i. e., nodes with samples of only one label.

$$\text{Gini Impurity} = 1 - \sum_{k=k_1}^{k_K} p_k^2, \tag{2.4}$$

with $\hat{p}_k$ denoting the estimated a-posteriori class probability of class $k \in k_1, k_2, ..., k_K$ in a single leaf. Since the Gini impurity is designed for labels as categorical variables, it is not suited for regression tasks. For splits on regression tasks, the estimated variance

$$\hat{\sigma} = \frac{(\boldsymbol{x} - \boldsymbol{\mu})^2}{N}, \tag{2.5}$$

with N being the number of samples in the leaf and μ the arithmetic mean, is usually calculated.

In the bias-variance framework, DTs are high variance learners, meaning that fully grown trees have a tendency to memorize the training data set as opposed to learn relevant patterns in the data. The performance of DTs therefore strongly relies on

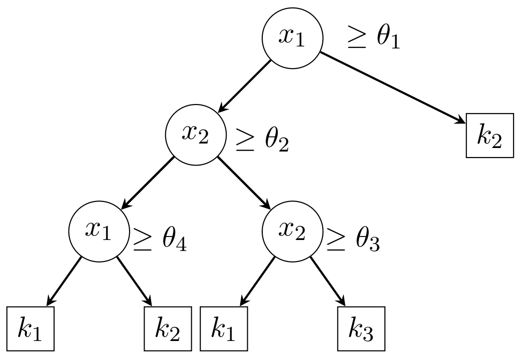


Figure 2.2: Structure of a DT.

effective pruning. Pruning is a method to reduce the tree depth by removing tree nodes bottom up, thereby reducing the variance of the tree while increasing the residual error.

Through their structural transparency, DTs possess attractive properties with respect to interpretability. Samples in each leaf may be traced through the decision process in the tree, leading to an easily comprehensible rule set. With growing tree depth, rule sets become longer and harder to comprehend at once. Similarly, with an increasing number of leaves, the number of different rule sets increases. Figure 2.2 displays the structure of a DT. The tree model corresponds to the segmentation displayed in Figure 2.1 and shares its notation.

The input into a tree based model is a one-dimensional feature vector. It is therefore challenging to adequately represent time series without losing the implicit neighborhood relations of two adjacent observations. With MTS inputs, the constraints concerning the input structure lead to a mixing of observations from several time series variables and therefore an excessive growth of the input dimensionality. Furthermore, a time series input may contain many observations that are not relevant for the classification task. A solution to these challenges is proposed in Chapter 3.

2.1.3 Random Forests

A random forest (RF) describes an ensemble of DTs, where the DTs are constructed in a way that they are as uncorrelated as possible. All trees in the RF are trained with individual training sets. The output of an RF is for classification tasks the majority vote of the B DT results, where B denotes the number of DTs. For regression tasks, the arithmetic mean

$$y_{\mathrm{reg}} = \frac{1}{B} \sum_{b=1}^{B} y_{\mathrm{b}}, \tag{2.6}$$

with y_{b} being the output of a single regression tree may be used.

A set of M training data samples $\boldsymbol{x}_m$ with according labels y_m will henceforth be called the training set

$$\mathcal{D} = \{\{\boldsymbol{x}_1, y_1\}, \{\boldsymbol{x}_2, y_2\} \ldots \{\boldsymbol{x}_M, y_M\}\} \tag{2.7}$$

The sets that are used to train the trees of an RF are generated through techniques like bootstrap aggregation or, in short, *bagging.* Here, B distinct sets are being generated from the original training set $\mathcal{D}$, with each distinct set $\hat{\mathcal{D}}_b$ for tree b having the same dimensionality as $\mathcal{D}$. Each set $\hat{\mathcal{D}}_b$, however, is a random draw with replacement from the samples in $\mathcal{D}$.

As opposed to training single DTs, pruning is not required for the trees of an RF. In the bias-variance framework, the ensemble of trees have a regularization effect on the result, thus reducing the model variance. However, in the process of reducing the correlation of the single learners by randomizing their input sets as explained above, the bias of the DTs in the RF is increased. In [16] it has been shown, that the ensemble averaging has no reducing effect on the resulting model bias of the ensemble learner. As pointed out in [17], the reduction of the model variance error typically outweighs the increase of the expected generalization error introduced by the bias component, subsequently establishing a better generalization ability of the ensemble learner than the one of the individual learners. One advantage of this type of structure is, that a high choice of B does not yield the danger of introducing variance to the classifier. Instead, the classifier performance does not increase anymore after a maximum number of trees in the RF. This maximum is individual to the application.

2.1.4 Multilayer Perceptron

With growing computational capabilities, more complex ML architectures started to live up to their potential. Methods that are structured in multiple subsequent layers, often termed as deep learning methods, achieve impressive results in many application fields such as image recognition and time series classification. Deep learning methods are often depicted as an artificial brain with its neurons being connected to each other. This refers to one of the most versatile, yet highly complex and non-interpretable architectures: the Multilayer Perceptron (MLP). This structure consists of single nodes, which are arranged in successive layers with individual numbers of nodes per layer. A node in layer l thereby has a connection to every node of the following layer $l+1$, forming a *fully connected layer*. The connections are weighted, with the weights being the learnable parameters of the network. The input of an MLP is a vector, whose elements are called the features of the network. Since no processing is performed in the input layer of the network, it will henceforth be referred to as layer l_0.

The output of each node, apart from the input layer, is a non-linear transformation of a weighted sum of all of its incoming connections. More concretely, the output activation of a node j within layer l, denoted as h_j^l can be described as a function

$$h_j^l = f\left(\sum_{i=1}^{n_{\text{nodes},\, l-1}} w_{ij} h_i + b_j\right), \tag{2.8}$$

where w_{ij} denotes the weight of the connection between node i in layer $l-1$ and node j, $f(\cdot)$ the nonlinear activation function of the node, and b_j the bias. The sum is built over all nodes n_{nodes} in the preceding layer $l-1$. Frequent choices for the activation function in the hidden layers of MLPs are the sigmoid, the tanh and the ReLU function. In the final layer, the activation function depends on the target of the network. For single class classification, a sigmoidal activation is a frequently used choice. Classification tasks with more than two classes often use the softmax activation function. For regression, a linear activation or, if only positive values are expected, a ReLU function is generally used.

The forward propagation through the a single layer of the network in vectorized notation can be expressed as the transformation

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}, \tag{2.9}$$

where the weight matrix $\boldsymbol{W} \in \mathbb{R}^{k \times q}$ contains all weights between the q nodes of layer $l-1$ and the k nodes of layer l. The non-linear activation function can be vectorized as

$$\boldsymbol{h} = f(\boldsymbol{z}), \tag{2.10}$$

where f denotes the activation function. Figure 2.3 depicts an MLP with 3 layers: The input layer l_0, one hidden layer l_1, as well as an output layer l_2. The weight w_{ij} is thereby an element of the weight matrix of the first layer, denoted as $\boldsymbol{W}^{(1)}$.

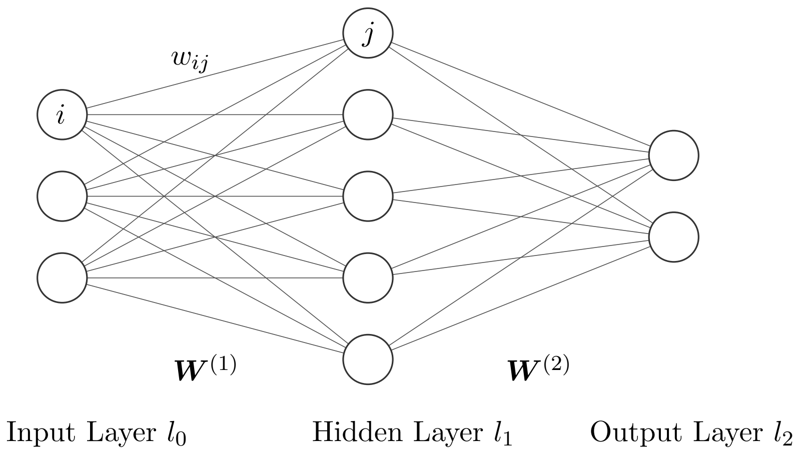


Figure 2.3: Basic Neural Network Architecture

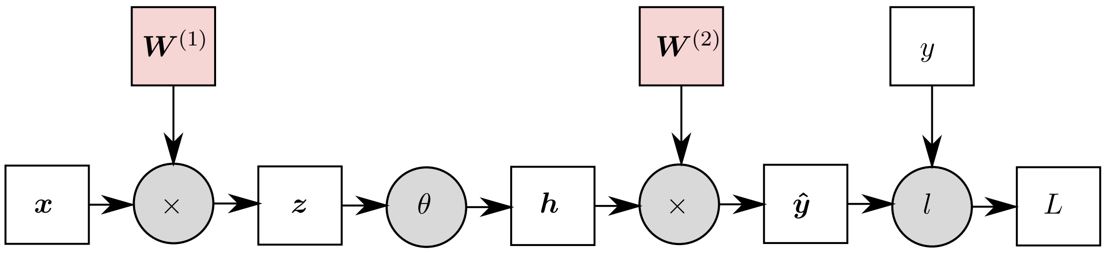


Figure 2.4: Computational Graph for a small MLP

Deep networks, such as the MLP, can be visualized as a directed graph model. This model is also called computational graph, since modern ML frameworks use the graph model to compute the gradient which will be explained further in this section. A computational graph visualizes dependencies between variables and operators using arrows to constitute the data flow. Figure 2.4 depicts the graph model for the network from Figure 2.3. Variables are written in squares, while operators are written in circles. Due to the simplicity of this small network model, the graph constitutes an unidirectional chain of operations. Nevertheless, more complex networks with regularization or recursions can be depicted as a graph model such as the recurrent neural networks in Section 2.1.7.

The training of MLPs relies on the concept of backpropagation. As opposed to the forward pass, backpropagation starts at the output of the network and follows the flow against the arrow directions. Starting with the loss L that is dependent on the label y

and the network output $\hat{y}$, the goal is to calculate the gradients in each step in order to update the network weights $\boldsymbol{W}^{(1)}$ and $\boldsymbol{W}^{(2)}$. The gradient of the second weight matrix may be quickly obtained by applying the chain rule

$$\frac{\partial L}{\partial \boldsymbol{W}^{(2)}} = \frac{\partial L}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{W}^{(2)}} \tag{2.11}$$

The first weight matrix can be obtained similarly by going further backwards through the graph. The partial derivative with respect to the node activation vector $\boldsymbol{h}$ is given by

$$\frac{\partial L}{\partial \boldsymbol{h}} = \boldsymbol{W}^{(2)\top} \frac{\partial L}{\partial \hat{\boldsymbol{y}}}. \tag{2.12}$$

The activation function is applied element wise, therefore the Hadamard operator $\odot$ is used to calculate the gradient through the activation function

$$\frac{\partial L}{\partial \boldsymbol{z}} = \frac{\partial L}{\partial \boldsymbol{h}} \odot \theta(\boldsymbol{z}). \tag{2.13}$$

In the final step of the backpropagation, the gradient through the multiplication operation with the input vector is calculated to obtain the gradient of the first parameter matrix $\boldsymbol{W}^{(1)}$

$$\frac{\partial L}{\partial \boldsymbol{W}^{(1)}} = \frac{\partial L}{\partial \boldsymbol{z}} \boldsymbol{x}^\top. \tag{2.14}$$

For more layers in the network, the propagation continues through further non linearities and weight matrices. MLPs are highly complex networks - the repeated interconnection of non-linearities, as well as the potentially high ammount of learnable parameters makes it hard to understand the inner workings of even shallow MLPs. Even an MLP with only one hidden layer is able to approximate any continous function, provided the hidden layer contains a reasonably high but finite number of nodes. This network type has therefore been termed a universal approximator [18].

2.1.5 Convolutional Neural Networks

Some input representations contain valuable information not only in the data content itself, but also in the data structure. This information can be, e. g., of temporary nature as in time series, of spatial nature as in images and occupancy grids, as well as their mixture as spatio-temporal representations. MLPs have the property of connecting each input feature with each node in the following layer, allowing the network to find relations between every feature. However, the neighborhood relations that are embedded in the data structure are not particularly regarded by this type of network structure. Convolutional Neural Networks (CNNs) are specifically designed to excel in this task. A CNN can not only process 1D input vectors, but also high dimensional input data such as 2D matrices for grayscale images and 3D arrays for color images or spatio-temporal data. Note, that due to the interpretable nature of low dimensional data, this work will focus on CNN architectures for 1D and 2D input structures.

In their basic form, CNNs consist of three different layer types. The core of a convolutional layer is its kernels - weight matrices that are convolved with the activation maps of the preceding layer. An activation map is thereby defined as the output of a layer. Through the convolution operation, the kernels learn local features, that are iteratively combined through the layers to form an overall understanding of the input space. Within the context of CNNs, the kernel size is also referred to as the *receptive field*. Given a grayscale image as an input, the first layer after the input in a CNN could learn rudimentary drawing elements, such as horizontal, vertical or diagonal lines in the image. In the following layer, these elements are combined to form shapes. With increasing network depth, the CNN can thereby learn more and more complex relations between the features.

Pooling layers are used to reduce the dimensionality between two convolutions by passing only one locally descriptive value, e. g. only the maximum value or the average value within a defined pooling size onto the next layer. Pooling layers take advantage of neighborhood relations in the data structure by assuming that neighboring features, i. e., pixels in the case of image classification, will not describe semantically completely different entities. Subsequently, pooling layers blur the borders of detected features to an extent that is proportional to the pooling size of the layer, as detail information is lost during the pooling process.

The last layers of the network are fully-connected layers with the task to connect and combine the accumulated scene understandings as established by the preceding convolutions.

The number of learnable parameters as compared to MLPs is significantly reduced, since the kernels learn to detect reoccurring templates and construct a scene understanding based on the preceding layers. This effect is called weight sharing, and is one of key factors for the effectiveness of CNNs. Furthermore, CNNs focus on neighborhood relations between neurons instead of establishing all possible node connections to

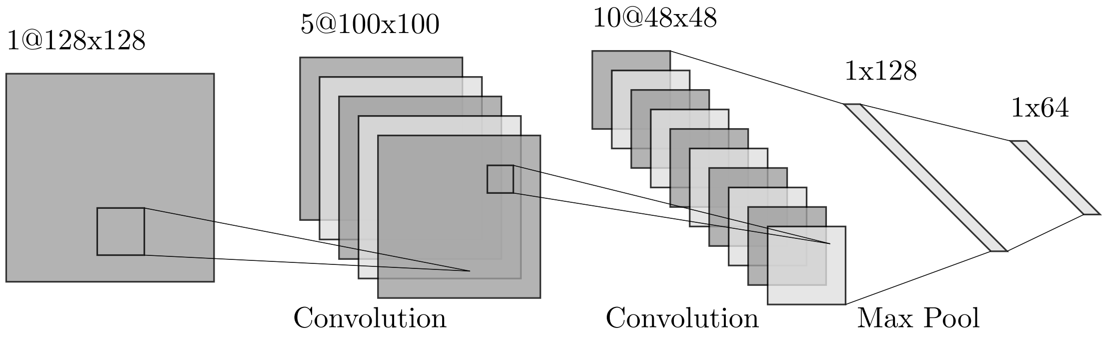


Figure 2.5: Basic CNN Architecture.

the preceding and following layer, which leads to sparsly connected layers. The total ammount of learnable parameters is therefore further reduced.

Figure 2.5 shows the basic architecture of a CNN. An input $\boldsymbol{X}$ is convolved with each of the C kernels $\boldsymbol{B}_1 \dots \boldsymbol{B}_C$ to form the feature space $\boldsymbol{F} \in \mathbb{R}^{(w_X - w_B + 1) \times (h_X - h_B + 1) \times C}$, with w_X, w_B denoting the width of the input and the kernel and h_X, h_B their heights, respectively. The feature space can be written as

$$\boldsymbol{F} = \boldsymbol{X} * \boldsymbol{B_c} + b_C \ \forall \ c \in \{1 \dots C\}. \tag{2.15}$$

The non-linearities following every convolutional layer in CNNs are usually rectified linear units (ReLU) or a variation thereof. The activation map $\boldsymbol{A}$ can be calculated by

$$\boldsymbol{A} = \theta(\boldsymbol{F}) \tag{2.16}$$

where θ denotes the non-linear activation function.

The backropagation algorithm works the same as in fully connected networks, which has been elaborated in Section 2.1.4.

2.1.6 Radial Basis Function Networks

Network structures such as the MLP and the CNN are parametric models, where each propagation can be described as a mapping from an input $\boldsymbol{x}$ to an output y, parametrized by a weight vector $\boldsymbol{w}$. After training the network, the training set is no longer needed for inference on test samples. In contrast, memory based methods store all or at least a part of the training data samples.

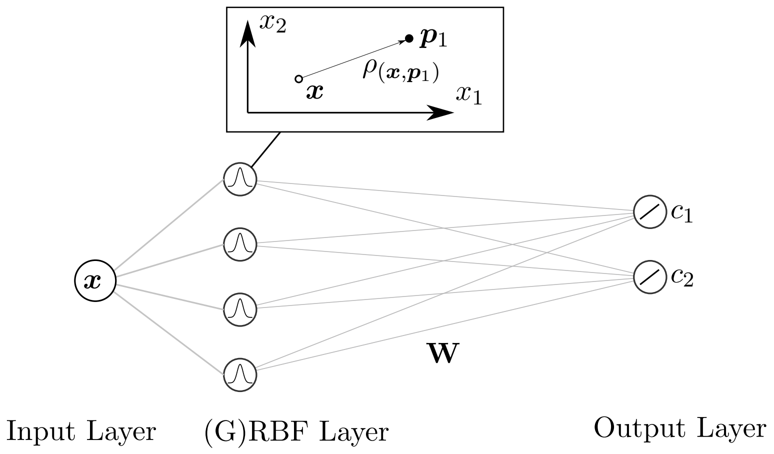


Figure 2.6: Basic GRBF network structure.

One network architecture that plays a central role in this work is the Radial Basis Function (RBF) network, particularly in its generalized form, the Generalized Radial Basis Function (GRBF) network. A radial basis function is a kernel method that calculates similarities to predefined center points in Euclidean space, such that the feature mapping ρ can be described as

$$\rho(\boldsymbol{x}) = \lambda(||\boldsymbol{x} - \boldsymbol{p}||_2^2), \tag{2.17}$$

with $\boldsymbol{p}$ being one of the center points and λ the similarity function. The linear combination of the P radial basis functions is calucalted to approximate a target value, i. e.,

$$f(\boldsymbol{x}) = \sum_{\text{cl}=1}^{P} \boldsymbol{w}_{\text{cl}} \lambda(||\boldsymbol{x} - \boldsymbol{p}_{\text{cl}}||_2^2) \tag{2.18}$$

can be formulated by determining the weight coefficients $\boldsymbol{w}_{\text{cl}}$, usually by applying a least squares approach. A deeper insight into the interpretability aspects of this linear connections can be found in Chapter 5. Figure 2.6 displays the GRBF network structure. The input vector $\boldsymbol{x}$, as well as two output nodes c_1 and c_2 are depicted. The output of each node in the (G)RBF layer is the similarity score $\rho_{(\boldsymbol{x},\boldsymbol{p})}$, which is determined from a distance measure between the input vector $\boldsymbol{x}$ and the node-individual center point $\boldsymbol{p}$.

When distance measures other then the Euclidean distance are applied in the radial basis function kernels, the term generalized radial basis functions is used. In this generalized form, any measure might be applied to evaluate between sample and center point. The similarity function λ is most commonly a Gaussian.

The design of a GRBF network can be divided into three subtasks. The first task is to determine the center points of the GRBF nodes. Several approaches have been proposed. The naive method is to select the center points randomly. Especially for a low number of overal GRBF nodes, this bears the risk of selecting outlier samples or multiple similar samples, leading to a bad coverage of the input space. More systematic approaches apply clustering techniques and extract the cluster representatives as center points. Several aspects have to be taken into account when selecting the clustering method:

- **The number of clusters** can be predetermined or dynamically determined. The elbow-method search strategy for evaluating a well-working number of clusters is appropriate for k-means type clustering algorithms. The hereby determined number of clusters must be reevaluated for each problem. Other methods, such as hierarchical clustering, generate as many clusters as deemed appropriate by iterating over a thresholded distance matrix. Nevertheless, this method naturally yields large differences in cluster sizes.
- **The admissability of center points** becomes important when interpretability is of importance. When the determined cluster center $\boldsymbol{p}_{\mathrm{cl}}$ is not a member of the dataset $\mathcal{D}$, a cluster center might not be meaningful although mathematically correct. As an example, k-means clustering yields a set of center points that do not coincide with data points from the training set. k-medoid clustering on the other side can guarantee, that
$$\boldsymbol{p}_{\mathrm{cl}} \in \mathcal{D} \tag{2.19}$$
by selecting the cluster center from the set of training data samples.
- **The distance measure** is the core component of any chosen clustering method. The Euclidean distance delivers numerically fast results, but should not be the method-of-choice when the underlying data is non-normally distributed multi-dimensional data. A data-adaptive method that estimates temporary cluster covariance matrices between each iteration of a k-means algorithm enables the use of Gaussian distance measures such as the Mahalanobis distance. More computationally efficient is the calculation of a proximity matrix based on random forest path proximities as proposed in [20].

The second task when designing a GRBF network is the choice of a distance measure from an incoming sample to the center points. Analogous to the distance measure explained further above in this section, the data distribution should be regarded by the chosen distance measure. Since at this point the clusters are fixed, its covariance matrices can be estimated without the necessity of an iterative process. To convert the distance into a similarity measure, a Gaussian can be used

$$s(\boldsymbol{x}, \boldsymbol{p}_{\mathrm{C}}) \in [0, 1]. \tag{2.20}$$

The last task is to define the fully connected layer, which connects the GRBF nodes to the output nodes. In the general case, linear connections generate weighted similarities of a class to each representative. As explained in Section 5.2.2, it is possible to constrain these weights in order to enforce strong interpretable properties.

2.1.7 Recurrent Neural Networks

Especially for time series forecasting tasks, RNNs have proven to be very powerful architectures. RNNs are neural networks with additional learning parameters that memorize previous states. The structure of a recurrent node is depicted in Figure 2.7. The input to the recurrent structure is a time series $\boldsymbol{x}$ at time t. The net input $z^{(t)}$ into a hidden node consists of the sum

$$\boldsymbol{z}^{(t)} = \boldsymbol{x}^{(t)} \boldsymbol{W}_{xh} + \boldsymbol{h}^{(t-1)} \boldsymbol{W}_{hh}, \tag{2.21}$$

where $\boldsymbol{W}_{xh}$ denotes the weights between an input and a hidden states, $\boldsymbol{h}^{(t)}$ the hidden state at time t, and $\boldsymbol{W}_{hh}$ are the weights between two hidden states. As known from feed forward neural networks, the hidden state $\boldsymbol{h}$ is calculated by means of a non-linear activation function θ

$$\boldsymbol{h} = \theta(\boldsymbol{z}). \tag{2.22}$$

Each hidden state is the input to the subsequent hidden state, as well as the output of the network at the respective timestep through another non-linear transformation with respect to the weight $\boldsymbol{W}_{hy}$. Here, the softmax is chosen as exemplary output activation

$$\hat{y}^{(t)} = \mathrm{softmax}\left(\boldsymbol{h}^{(t)} \boldsymbol{W}_{hy}\right). \tag{2.23}$$

The forward propagation of an RNN can be calculated straightforward by calculating the output (2.23) iteratively for every timestep of the input sequence $\boldsymbol{x}$. Training the

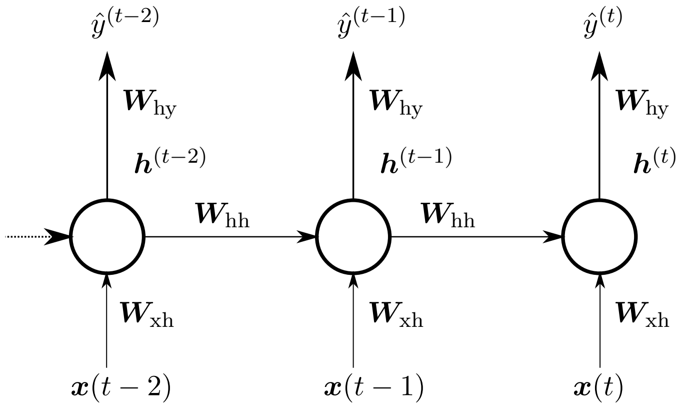


Figure 2.7: Basic recurrent neural network structure.

network, however, poses a computationally challenging task with the risk of numerically instable behavior as explained below. In order to process time series instead of single-frame samples, the backpropagation from 2.1.4 is enriched by one more dimension, yielding the backpropagation through time (BPTT). Goal of the BPTT is the computation of the gradients for the model parameters $\boldsymbol{W}_{xh}, \boldsymbol{W}_{hh}$ and $\boldsymbol{W}_{hy}$. The objective function is a loss function that sums the individual errors of each time step of the time series

$$L = \frac{1}{T}\sum_{t=1}^{T} l(\hat{y}^{(t)}, y^{(t)}), \tag{2.24}$$

with $y^{(t)}$ denoting the label at the respective timestep and T denoting the number of elements in the time series. The backpropagation yields the gradients for the three parameters, where

$$\frac{\partial l}{\partial \boldsymbol{W}_{hy}} = \frac{\partial l}{\partial \hat{y}^{(t)}} \boldsymbol{h}^{(t)\top}, \tag{2.25}$$

can be calculated by only using the values of the current timestep. Following the graph on Figure 2.7 further backwards and applying the chain rule, the formulas for W_{hh} and W_{xh} can be denoted as

$$\frac{\partial l}{\partial \boldsymbol{W}_{hh}} = \frac{\partial l(\hat{y}^{(t)}, y^{(t)})}{\partial \hat{y}^{(t)}} \frac{\partial \hat{y}^{(t)}}{\partial \boldsymbol{h}^{(t)}} \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{W}_{hh}}, \tag{2.26}$$

and analogously

$$\frac{\partial l}{\partial \boldsymbol{W}_{xh}} = \frac{\partial l(\hat{y}^{(t)}, y^{(t)})}{\partial \hat{y}^{(t)}} \frac{\partial \hat{y}^{(t)}}{\partial \boldsymbol{h}^{(t)}} \frac{\partial h^{(t)}}{\partial \boldsymbol{W}_{xh}}. \tag{2.27}$$

Since the weight matrices are included in the activation function of the hidden node as it can be seen in (2.21) and (2.22), the partial derivative $\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{W}_{xh}}$ can be expressed using the chain rule in order to reflect the influence of the activations of earlier timesteps to the gradient

$$\frac{\partial l}{\partial \boldsymbol{W}_{hh}} = \sum_{k=1}^{T} \frac{\partial l(\hat{y}^{(t)}, y^{(t)})}{\partial \hat{y}^{(t)}} \frac{\partial \hat{y}^{(t)}}{\partial \boldsymbol{h}^{(t)}} \frac{\partial \boldsymbol{h}^{(t)}}{\boldsymbol{h}^{(k)}} \frac{\partial \boldsymbol{h}^{(k)}}{\partial \boldsymbol{W}_{hh}}, \tag{2.28}$$

which is equally valid for $\boldsymbol{W}_{xh}$. In the calculation of the gradient, the weight matrices will be repeatedly multiplied with themselves. Consequently, for long sequences any value < 1 will become very small while values > 1 will become extensively large, known as the vanishing and exploding gradients, respectively.

More sophisticated recurrent structures managed to overcome this problem, with the two most prominent examples being the Long-Short-Term Memory (LSTM)[21] cell, as well as the Gated Recurrent Unit (GRU)[22]. Both of these structure contain one or more “gates”, sigmoidal activated and weighted input branches that can switch the signal path within the RNN cell. By using these gates, the RNN can learn to ignore certain inputs, reset the recurrent feedback or even suppress the output. This avoids the multiplicative weight chaining that causes the vanishing gradient problem in RNNs. It has been shown, that by implementing these structures, RNNs can learn connections between features even over very long time sequences. Figure 2.8 shows the structure of a GRU cell. Two gates, reset and update, adjust the influence of new data samples to the hidden state $\boldsymbol{h}^{(t)}$, as well as the hidden state candidate $\tilde{\boldsymbol{h}}^{(t)}$. A GRU layer thereby learns six weight matrices, which can be seen when denoting the formulas for the activation of the gates

$$\boldsymbol{r}^{(t)} = \sigma(\boldsymbol{x}^{(t)} \boldsymbol{W}_{xr} + \boldsymbol{h}^{(t-1)} \boldsymbol{W}_{hr} + \boldsymbol{b}_r) \tag{2.29}$$

$$\boldsymbol{z}^{(t)} = \sigma(\boldsymbol{x}^{(t)} \boldsymbol{W}_{xz} + \boldsymbol{h}^{(t-1)} \boldsymbol{W}_{hz} + \boldsymbol{b}_z), \tag{2.30}$$

as well as for the candidate hidden state

$$\tilde{\boldsymbol{h}}^{(t)} = \tanh(\boldsymbol{x}^{(t)} \boldsymbol{W}_{xh} + (\boldsymbol{r}^{(t)} \circ \boldsymbol{h}^{(t-1)}) \boldsymbol{W}_{hh} + \boldsymbol{b}_h). \tag{2.31}$$

The hidden state is then provided by weighting the old hidden state and the hidden state candidate by means of the update gate

$$\boldsymbol{h}^{(t)} = \boldsymbol{z}^{(t)} \circ \boldsymbol{h}^{(t-1)} + (1 - \boldsymbol{z}^{(t)}) \circ \tilde{\boldsymbol{h}}^{(t)}. \tag{2.32}$$

An LSTM cell adds another gate, as well as a memory cell, to store additional information on the past states.

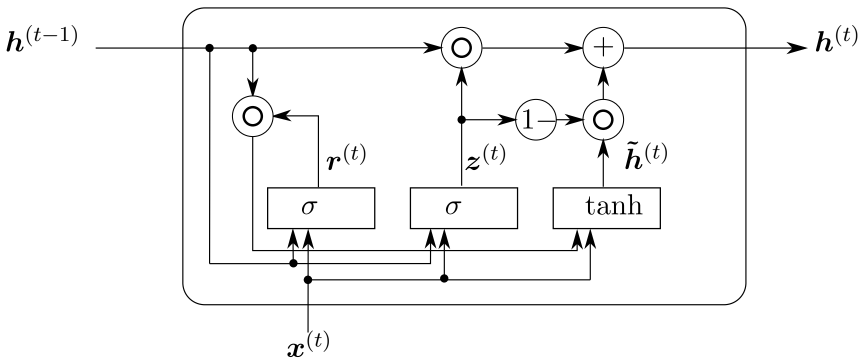


Figure 2.8: Computational flow of a GRU cell.

2.2 Datasets of Real Driving Scenarios

While the methods developed in this work are generalizable and can be applied to a multitude of applications, it is beneficial to display the implementation and results on a specific use-case. In the field of autonomous driving, functional failures often result in serious consequences to the wellbeing of passengers or other traffic participants. Therefore, many functions in that area are considered safety critical and require thorough validation. One very relevant application is the detection and early prediction of lane changes. The goal is to predict an upcoming lane change of a third party road participant. When the prediction is performed adequately early, reactive responses such as braking or lane changing can be initiated smoothly and comfortably. In the further course of this work, the lane change prediction task will serve as illustrative application for the proposed methods. A second application from the automotive sector is the roundabout exit prediction. The goal is to predict, if an observed vehicle will exit the roundabout at the upcoming exit.

2.2.1 Dataset for Lane Change Prediction

A publication by Krajewski et al. [23] introduced the highD dataset of highway scenarios, recorded in a birds-eye view using drones. This dataset is used as a reference to make the results of this publication transparent and reproducible. The highD dataset contains near-perfect perception scenarios, using the top down perspective to avoid sensor shadows and the loss of objects, which are natural perception weaknesses of street-level recordings. The object lists delivered with the dataset contain all objects on the road in

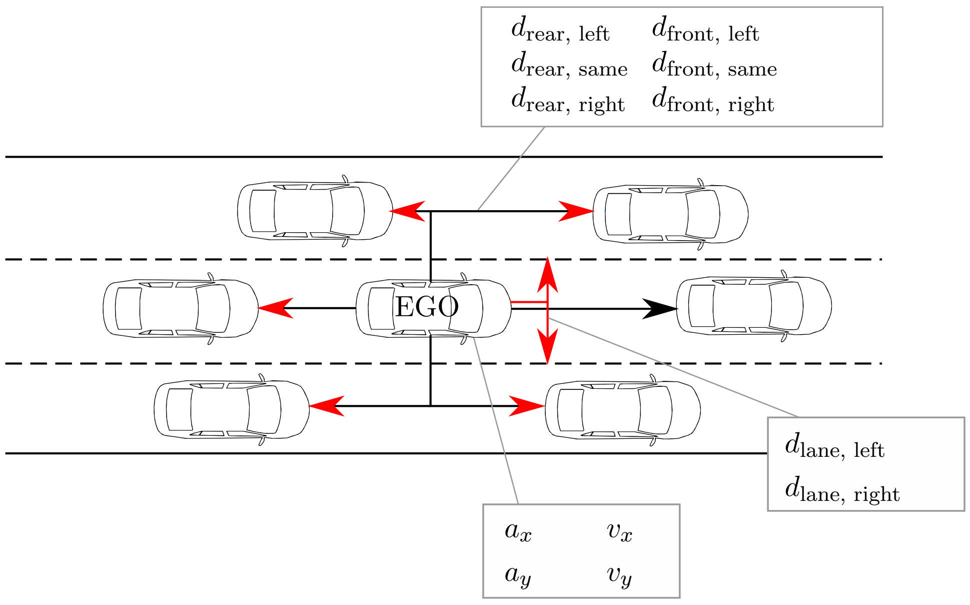

$d_{\text{rear, left}}$ $d_{\text{front, left}}$
$d_{\text{rear, same}}$ $d_{\text{front, same}}$
$d_{\text{rear, right}}$ $d_{\text{front, right}}$
EGO
$d_{\text{lane, left}}$
$d_{\text{lane, right}}$
a_x v_x
a_y v_y

Figure 2.9: Environment model for the lane change dataset.

synchronized frame numbers. For each object, the dynamic metadata such as velocities and accelerations, the lane number, as well as the id-numbers of the surrounding objects are recorded. In a preprocessing step, this data is converted to real world feature vectors as they can be generated by on-road vehicles. The environment model thereby contains one observed object called the EGO vehicle with its dynamic data, as well as its distances to the surrounding objects. Furthermore, the distances to the adjacent lane markings left and right are annotated. Figure 2.9 illustrates the features extracted from the dataset. All distances are centered to the geometric center of the object.

The distances as extracted are based on the relative position between EGO and the object. This might result in unrealistic perception properties when the surrounding vehicle is very far away. The distances are therefore limited to a maximum of $\pm 200\,\text{m}$. Since the different physical entities come with different value ranges, all features in the dataset are normalized using the min-max normalization, where

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}, \tag{2.33}$$

with x being the input, x' the normalized feature and $\min(x)$, $\max(x)$ the minimum and maximum value of the feature in the dataset, respectively.

From this preprocessed dataset, all lane changes that were observed for at least $T_{\text{LC,min}} = 6.52\,s$ before the lane change are extracted. The timestep t_0, at which a lane change is detected, is defined as the intersection between any lane marking and the geometric center of an observed object. To avoid a learning bias, the dataset is subsequently balanced by randomly discarding samples of overrepresented classes. The labels are assigned as

$$k_{\text{LC}} = \{LCL,\, LCR,\, NLC\}, \tag{2.34}$$

where *LCL* denotes a lane change to the left, *LCR* a lane change to the right and *NLC* no lane change. The highD data was processed to extract a dataset $\mathcal{D}$ of M isolated lane change scenarios

$$\mathcal{D} = \{\boldsymbol{X}^{(1)}, \ldots, \boldsymbol{X}^{(M)}\}, \tag{2.35}$$

with

$$\boldsymbol{X}^{(m)} = \begin{bmatrix} \boldsymbol{x}_1^{(m),\top} \\ \vdots \\ \boldsymbol{x}_F^{(m),\top} \end{bmatrix} \in \mathbb{R}^{F \times N}, \tag{2.36}$$

where $\boldsymbol{x}_f^{(m)}$ is the sequence of the f-th feature from time step 1 to N and F denotes the number of features. The discrete times steps n correspond to the continuous time t through

$$n = t \cdot f_{\text{sampling}}, \tag{2.37}$$

with $f_{\text{sampling}} = 25\,\text{Hz}$ denoting the sampling frequency of the dataset. The structure of a single input sample can be seen in Figure 2.10. An upcoming lane change should be predicted an appropriate time t_{pred} ahead of the event itself. Therefore, only those scenarios, where the observed object was visible for the time t_{pred} could be extracted from the highD data. Because of the limited field of vision in the highD recordings of approximately 410 m, many of the lane changes could therefore not be included, as they either started too early or were not completed within the observed road section. The resulting lane change dataset from highD consists of 11000 unique samples with balanced classes. In the following sections, the term *highD dataset* will be used to refer to the set of extracted lane changes, instead of the original dataset.

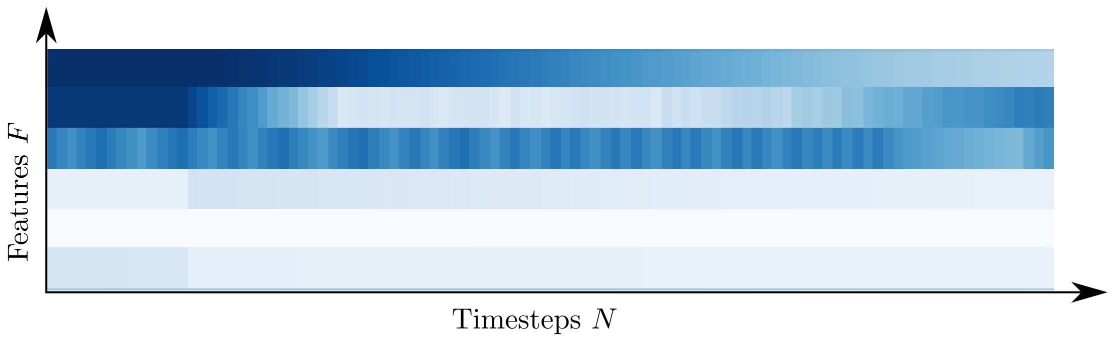


Figure 2.10: Input sample, interpreted as two-dimensional image

2.2.2 Dataset for Roundabout Exit Prediction

A dataset that was recently published examined the behavior of vehicles in roundabout scenarios [24]. The setup for recording the dataset also included a drone which was hovering stationary above a roundabout in the Ingolstadt metropolitan area in Germany. The recorded roundabout has an outer diameter of 38m and a lane setup as depicted in Figure 2.11, the driving directions are shown as arrows along the lanes.

Objects are tracked and assigned a unique object ID, followed by a post processing step to extract dynamic features such as velocities, accelerations as well as rotational rates. Additionally, the time-to-collision (TTC) for the leading and the following vehicle is calculated. A feature specifically relevant for roundabout scenarios is a TTC measure for situations, where EGO has not yet entered the roundabout, while another vehicle is already in the roundabout lane. In this case, a secondary TTC is calculated and added to the feature vector.

The learning task for this dataset is a roundabout exit prediction. For an observed object it should be predicted, if it is leaving the roundabout at the upcoming exit, together with a time-to-event (TTE) prediction when it is going to leave the roundabout. The problem is formulated as a two-class classification problem with respect to the upcoming exit

$$k_{\mathrm{RA}} = \{\mathit{Exit},\ \mathit{NoExit}\}. \tag{2.38}$$

In a roundabout scenario all objects are exiting the roundabout at a relatively close time horizon. Any no-event prediction, i. e., the prediction that a vehicle is not leaving the roundabout at the next exit, would therefore overlap with a positive prediction for one of the consecutive exits.

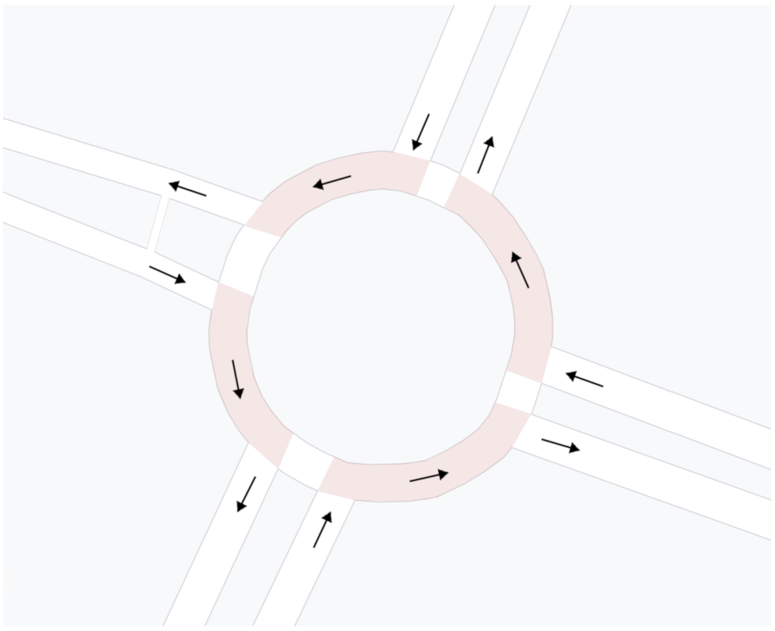


Figure 2.11: Roundabout in the Ingolstadt metropolitan area.

A single scenario is defined as an object traveling through one of the sectors that are highlighted red in Figure 2.11. At the end of each scenario, the object is either leaving the roundabout at the exit or continuing on the roundabout lane. The labels are designated accordingly. A single object can therefore generate multiple scenarios by traveling through more than one sector, e. g., entering the roundabout on the top and leaving at the third exit generates 3 scenarios.

A total of 2706 scenarios could be extracted from the recordings. The scenarios are balanced evenly among the classes. The data structure is identical to the lane change intention dataset, with the exception that the roundabout data is sampled at $f_{\text{sampling}} = 50\,\text{Hz}$. The overall observation time of samples in the roundabout dataset is significantly lower than the lane change dataset. Each sample has a defined beginning at the point of entering the sectors as visualized in Figure 2.11. The mean travel time through a sector is 2.3 seconds.

2.3 Safety of Machine Learning Algorithms

For the use in safety critical applications such as autonomous driving functions, it is not sufficient to train an end-to-end network and use simple ML metrics like confusion tables or simply accuracy to evaluate its performance and release it into series production. Especially automotive software has detailed standards on the verification and validation of their components.

2.3.1 Verification & Validation

Verification & Validation is often used as a fixed, interconnected term, since the related processes become relevant roughly at the same time during function development and release. The two components however can, and must, be viewed separately, since their individual definition shows major differences between the two single terms.

Verification, according to [25] describes the process of checking, whether a function has been built according to previous requirements and specifications by means of objective evidence. In the context of automotive software functions, this could be a number of tolerated false alarm rates within a prespecified interval during operation, a maximum reaction time upon a trigger event or a percentage of availability of a function over the complete time of operation. Verification, however, does not take into account what the targeted output is. If, e. g., a requirement describes the percentage of availability of a given function, the verification can be successful by just checking this requirement, without taking into account if the function works correctly.

The process of validation has its focus on the output side of a function and is therefore linked to its goals, rather than its prespecified requirements. While verification can be achieved by comparing specifications with the actual properties of the product, validation can be achieved by running the code to evaluate the output with respect to the expectations and needs of the customers or other stakeholders, e. g., product owners or project managers. Methods that are used in the validation process can be black-box and white-box testing.

2.3.2 Functional safety for Road Vehicles

For years, the ISO norm 26262 for the functional safety of road vehicles [26] has been the standard for developing safety critical software functions. The norm describes the processes to ensure functional safety in electronic components for automotive applications. After identifying potential hazards, as well as according safety goals to evade them, each failure mode is categorized into Automotive Safety Integrity Levels (ASIL). As the level increases, the norm recommends more thorough safety measures.

Those could be the implementation of redundant systems, an increasing scope of testing or the conduction of a fault tree analysis. The ASIL is thereby determined by three factors. These are: (1) the severity of the error. This includes damages to the vehicle, but also harm to the driver and passengers, (2) the exposure to the error. An evaluation of how often an error will occur, and (3) the controllability of the error and the resulting driving situation.

When looking at ML algorithms, several publications criticized the applicability of the ISO 26262 norm. In [27], an extension of the ISO norm was proposed in order to better reflect the potential hazards automated functions that employ ML algorithms may additionally pose. The publication identifies multiple aspects of ML functions that contradict the functional safety approaches of ISO 26262. Two of those aspects are important to this work and therefore will be elaborated further.

The first argument is made with respect to the V-model for software development [28]. Figure 2.12 displays this model with its name-giving shape. On the left side, the design of a software function is specified with levels of increasing depth, starting from functional specifications on the top and going deeper up to the specifications of the software architecture itself. The right side of the V-model provides the tests for each level. It is assumed that the component behavior is fully specified through the left side of the V-model. The implemented functions are specifically designed to fulfill the posed requirements. Each of the specifications and requirements can be tested, and therefore the function can be verified. Data-adaptive functions such as ML algorithms use training sets under the assumption that the specifications and requirements are implicitly regarded and learned by the model through the underlying training data. A training set, however, is always incomplete and it cannot be guaranteed, that all requirements are reflected to their full extent.

The second argument from [27] that is relevant for this work is the potential resolution of the traditional computing process along the robotic paradigm *sense - plan - act* and the respective subfunctions of these stages. By implementing end-to-end approaches, an independent development and functional safety process on a unit-level for each subfunction cannot be performed. Instead, most modern deep network architectures can be considered single generic architectures, consisting of a multitude of elements with weighted nonlinear connections. Traditional functional safety according to ISO 26262 can only be established by understanding the software architecture and the

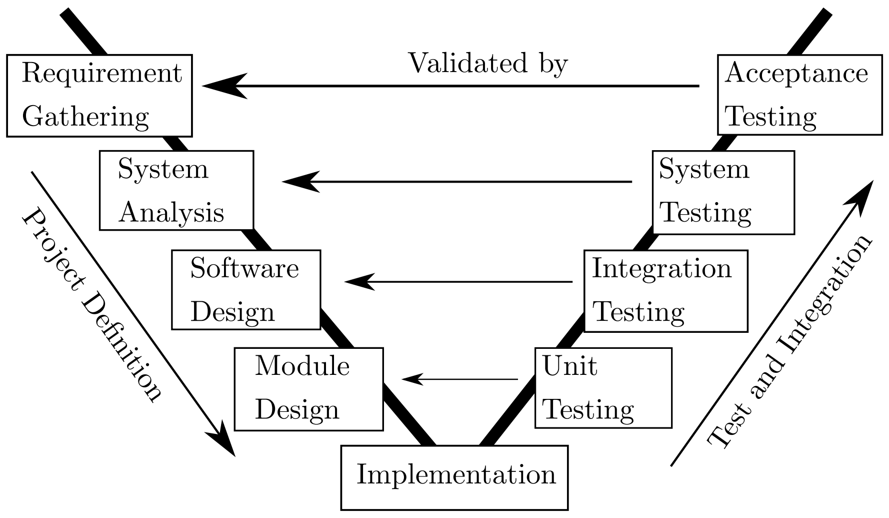


Figure 2.12: Classical V-model for software development.

underlying implemented rules. However, such a comprehension is not possible for deep networks with cross-unit implementation ranges, e. g., the above mentioned end-to-end approaches.

The increasing level of automation in driver assistance functions leads to a wider scope of safety relevant aspects when considering verification and validation. The focus of the ISO 26262 is the safety of the system itself against internal erros such as, e. g., software bugs. It is necessary, however, to also consider the behavior of the system outside of its limits. The ISO 21448 [29] has been released to define processes to ensure the Safety of the Intended Functionality (SOTIF). The norm includes the definition of the limits of a system, as well as the behavior outside of these limits. Specific for the safety aspects adressed in the ISO 21448 is, that the function is not acting intrinsically errnonous, but is much rather used in unexpected enviroments or situations. An example is a lane change intention detection that has been designed for highway scenarios, but is instead used in urban environments. Furthermore, disturbed sensor inputs caused from enviromental conditions such as heavy rain or direct sun, as well as an expectable misuse by the end user is regarded in the SOTIF standard.

2.3.3 Formulation of Safety Arguments for Road Vehicles

The experts in the field of functional safety defined multiple terms to describe the single components of the assurance process. A safety argument is thereby defined as a

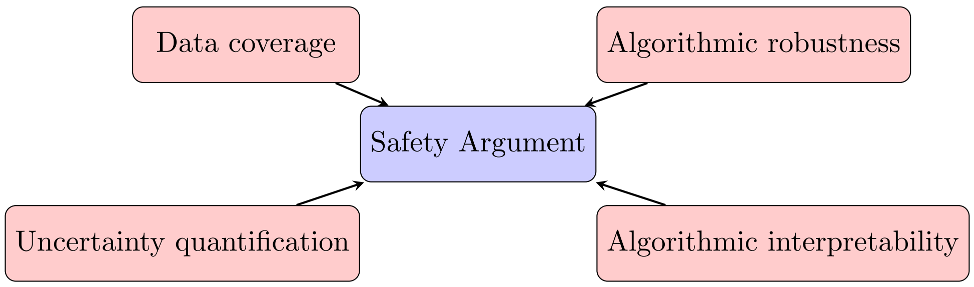


Figure 2.13: Components of a safety argument.

conclusive collection of safety evidences, which are function properties used to construct a justification that a function or system is safe to use [30]. In [31], the formulation of a safety argument specifically for functions that use ML algorithms is discussed. It is analyzed, that real world testing and simulation alone are not sufficient for a safety argument. Instead, Koopman et al. propose to formulate a safety argument that explicitly targets the validation of the network design, and by that to disintegrate the former unity of requirement and design validation. This work identifies four core components of a safety argument for design validation of an ML architecture that should be addressed as displayed in Figure 2.13. In this work, methods that focus on the quantification of uncertainty and the formulation of interpretability are presented.

The component **data coverage** describes an analysis of the training set with respect to its completeness. It should be shown, that the feature space covered by the training set is sufficient to represent the true, real world data distribution, which is generally unknown. However, even for very large datasets it cannot be claimed that "enough data" is available to represent the real-world distribution. With respect to data coverage, one may assess the membership of multiple datasets, i. e., training, validation and test sets, to the same distribution. Usually, an evaluation of different performance metrics indicate, if the datasets stem from the same distribution.

In the context of setting up a safety argument, the goal is therefore to determine intervals in the data for which an ML algorithm can make predictions backed by enough training data, while at the same time being able to reject predictions in areas that have not been covered sufficiently by the data set. One approach to address this issue for DNNs is presented in [34], where data points that differ from the seen training data have been discovered by using the minimum and maximum neuron activations of a network during its training phase.

Software robustness in the classical sense describes test methods, which apply different inputs to the system with the goal of finding unexpected behavior of the tested algorithms. A known testing method for software systems is fuzz-testing, where system inputs within a valid range are generated randomly [35] or as mutations of known input samples (see, e. g., [36]). This testing method is to ensure, that no potentially valid input can lead to operational errors in the system under test. Another popular method in this field is fault injection (see, e. g., [37, 38], a method where erroneous inputs are deliberately provided to test the error handling. For ML methods, **algorithmic robustness** has been used to describe the generalization properties of an algorithm. After a data-adaptive algorithm has been trained on the training set, inference on a test set shows if the performance remains on the same level. Robust ML algorithms are able to keep up the performance without a significant drop in accuracy. For more advanced ML tasks such as early prediction, accuracy can be exchanged with more sophisticated performance measures.

Uncertainty quantification is split into two viewpoints. Aleatoric uncertainty describes the uncertainty in the data itself for semantically identical samples. This happens due to unknown or immeasurable influences on the data, resulting in a distribution of the observations. An illustrative example for aleatoric uncertainty in an automotive context could be a measured set of braking maneuvers and the according stopping distance. The maneuvers are performed with a driving robot to be identical in initial speed, brake pedal force and brake pedal velocity. The immeasurable influences in this setup could be mechanical elasticities in the power train, the brake system and the tires, leading to a distribution around a certain mean braking distance instead of all-identical results. With increasing precision of the recorded data, e. g. through additional sensors, this uncertainty can be made measurable and therefore becomes epistemic uncertainty.

Epistemic uncertainty describes systematic model inaccuracies through neglected features. These features, as opposed to the aleatoric uncertainty, could theoretically be measured and have an influence on the outcome. Epistemic uncertainty is sometimes also termed as model uncertainty, since it covers the resulting data distribution caused by variational components in the algorithms. In this work, epistemic uncertainty plays an important role when estimating the confidence of a classifier in its own decision.

Algorithmic interpretability has been identified as one of the keys to validate functions that apply methods of ML. In the context of ML, interpretability has been defined as the “ability to explain [a result] in understandable terms to a human” [39]. With this in mind, a properly formulated interpretability argument can build the bridge between an ML algorithm and the human assessor, possibly the product owner or the project manager, that is responsible for confirming the compliance of the algorithm to the safety standards and, subsequently, the release of the software into series production. Following the above stated definition of interpretability, assessors with different areas of expertise will characterize different outputs and processing depths as interpretable. The goal is therefore to avoid complex argumentations and to formulate a persuading interpretability claim.

2.3.4 Safety Envelopes for Classic Validation

One approach to solve the challenge of ML validation is the concept of safety envelopes. In [40], a formal model was examined that would let ML models of unlimited complexity act within physical boundaries. The concept, termed Responsibility-Sensitive Safety (RSS), identifies five core behavioral rules, whose composition should ensure safe behavior. Since at the point of publication of this paper, no acknowledged safety assurance process for ML had been established, RSS never intended to enable validation or verification of ML methods themselves, but rather of autonomous driving functions as a whole. With this in mind, all proposed rules of RSS describe parametrizable limits and thresholds of accelerations and distances with respect to the surrounding road objects. As an example, RSS defines the safe distance to the leading vehicle as

$$d_{\min} = \left[v_r \rho + \frac{1}{2} a_{\max} \rho^2 + \frac{(v_r + \rho a_{\max})^2}{2\beta_{\min}} - \frac{v_f^2}{2\beta_{\max}} \right]_+ \tag{2.39}$$

with v_r, v_f denoting the velocities of the rear and front vehicle, ρ the reaction time, $a_{\max}$ the maximum positive acceleration and $\beta_{\max}, \beta_{\min}$ the maximum and minimum brake acceleration, respectively. This equation is therefore just a physical description of a worst case scenario with a maximum acceleration of the rear vehicle before the response, and only minimum braking afterwards while the front vehicle is fully braking. Within the paper, similar equations with respect to lateral distances, as well as visibility ranges are formulated. A violation of these safety requirements, e. g., by another vehicle cutting into the lane, leads to according actions that will reestablish a safe situation. Since the implementation of these formulas is strictly rule-based, a validation process is possible by means of the well-known standards for safety in automotive software functions as elaborated in the preceding sections. RSS also captures general behavioral rules, such as granting the way of right if another vehicle is not following the traffic rules, or even deliberately violating set safety distances if an accident can be avoided.

2.3.5 Validation by Simulation

Another approach for the validation of ML algorithms puts extensive testing in the focus. One key point of software testing states that it is merely possible to prove the existence of errors - not their absence. Therefore, test results may only confirm that under the imposed test conditions, no error has been found, leaving a wide open gray area of untested conditions under which the algorithm may or may not behave correctly. A conclusion that can be drawn from this is that testing efforts are the base of a probabilistic analysis on the safety of software functions. The more a software model is tested, the higher the chances that certain fail conditions are discovered and the lower the chance of failures after software deployment. The underlying data, however, must include many different situations and settings. Software functions for automotive applications pose a special challenge to this requirement. Autonomous driving functions are most often safety relevant, and the recording of real data is very expensive and time-consuming. In order to collect enough data for validation purposes, some publications state minimum necessary amounts of up to 100 billion miles of recorded data [41]. This enormous number, however, does not in any way cover how many critical scenarios, or even just variety in the data, is captured. Other publications therefore propose to enhance the variety through real world testing in combination with mandatory complementary methods [27].

The simulation of driving scenarios is a viable option for this purpose. Simulated environments are abstractions of real world settings and rely on modeled behavior and rules sets - simplifications of the physical reality. This leads to the fact that simulations have a certain level of accuracy, where the required minimum has to be determined depending on the learning goal. In [31], an analysis on the necessity of realism for vehicle simulations was made. If a simulation of camera images is desired, it is necessary to fully model the environment, potentially including precise representations of light sources, reflections, surfaces and colors. On the contrary, if the simulated dataset is to be used for prediction tasks that work with preprocessed object lists, a generation of abstract scenarios with vehicle position and boundaries might be sufficient.

Even when the inaccuracies of the simulation model are thoroughly examined and found acceptable for the desired application, validation through simulation proves difficult. An open area of research is the analysis of distributional shifts between datasets with the goal of quantifying their differences. While the correction of the distributional shift is designed for concept shifts [42], the covariate shift that occurs between real world data and simulation data is not easily correctable. If it is not possible anymore to determine, if unseen samples originate from simulated or real data, a defensible argument for validation with simulated data can be made. Without assessing the inaccuracies of the simulation dataset, generated scenarios may still be used to aid the testing of models and to build up confidence in an existing classifier.

2.4 Interpretability in Machine Learning

Following the wave of popularity for ML methods, many stakeholders demanded explanations for the decisions that are made by the algorithms. If it is the decision to operate a patient based on ML classifications or the rejection of a credit application - without interpretability, those high-risk decisions would be made entirely based on black box model predictions. The society around Explainable Artificial Intelligence (XAI) [43, 44, 45] aims to open this often-cited “Black Box” of Deep Neural Networks (DNN) through interpretable methods. The focus of XAI is thereby to recursively deliver explanation of decisions that have already been made. The interpretability that is established is termed *post-hoc interpretability.* Several different approaches have been taken, that fall under the category of post-hoc interpretability. If the explanations should be independent of the ML architecture, model-agnostic methods that generate simplifications of the trained model are the right choice. The model complexity is thereby reduced, until a conclusion can be made as to which input components are responsible for the networks decision. One popular representative of model-agnostic methods is the Local Interpretable Model-Agnostic Explanation (LIME) [46], which generates a linear approximation of the decision boundary at a given point by perturbing the input and tracing, which perturbations are relevant for the network decision. The work has been continued in [47] to generate if-else splitting rules. Other methods like the Genetic Rule Extraction (G-REX) [48] use genetics algorithms to extract simplified rule sets to deliver explanations for an opaque structure.

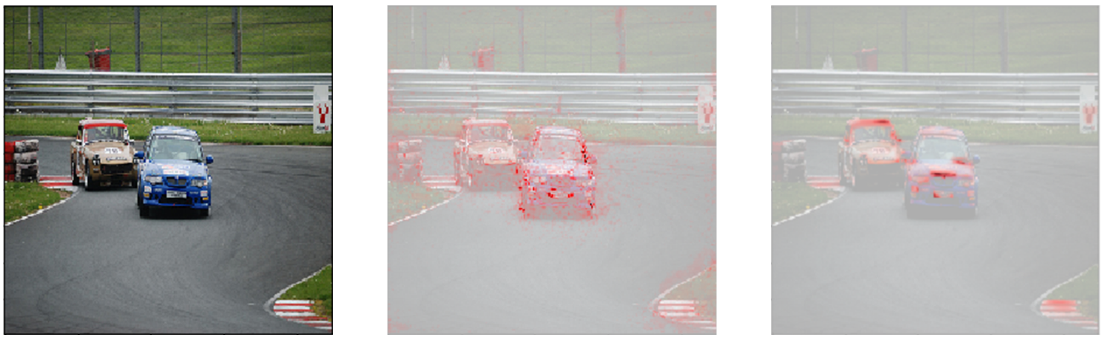
If the architecture of a deep network is known, model specific methods may deliver faster and more accurate explanations than model-agnostic methods. With interpretability in mind, one set of model-specific methods is particularly attractive. Provided that the input is an interpretable image-like structure, visual explanations based on sensitivity or saliency deliver up to pixel-precision accurate explanations. Sensitivity Analysis (SA) is a common tool in statistics [49], that has been successfully applied to ML [50]. SA delivers local explanations around a classification by measuring the gradient of each pixel. Let $f(\boldsymbol{x})$ be the network transfer function given an input vector $\boldsymbol{x}$, the sensitivity for a single input feature x_i can be described as

$$s_i = ||\frac{\partial}{\partial x_i} f(\boldsymbol{x})||. \tag{2.40}$$

In the case of image inputs, the result is a sensitivity for every input pixel. An interpretable heatmap can now be generated by overlaying the sensitivity scores onto the original input. Figure 2.14b shows such an overlay on an exemplary image of the popular ImageNet dataset [1]. However, SA produces heatmaps that appear rather grainy and fragmented. Single, high gradient pixels dominate the heatmap and indicate, that by manipulating only a small subset of input pixels, the output scores can be majorly influenced. This effect is also core of the publications that address the subject of adversarial samples [51] and shows, that the resulting explanations are only locally relevant and do not deliver good intuitions for human assessors.

Another visualization technique called Layer-wise Relevance Propagation (LRP) [52] avoids these local hotspots by not only measuring the gradient, but also by evaluating the distance to a constructed sample with zero class activation and closest possible proximity to the input. Figure 2.14c shows the resulting heatmap, with very concise and defined regions of interest. A closer examination of LRP with an insight into the underlying decomposition method to generate heatmaps in deep networks is provided in Section 3.2.3 as core component of an interpretable feature generation and selection approach.

With XAI methods, it is now possible to understand single network predictions. By analyzing a small subset of samples from each class, the risk of systematic focus errors through unwanted image artifacts can be reduced, such as the infamous "Husky vs.



(a) Original image of race cars. **(b)** Sensitivity Analysis on image of race cars. **(c)** Layer-wise Relevance Propagation on image of race cars.

Figure 2.14: Heatmapping methods on an image of race cars from the ImageNet dataset [1], classified by a pretrained VGG16 network [2].

Wolf" classifier that focussed on detecting snow in images rather than features of the depicted animals. Nevertheless, these type of explanations help to uncover the presence of errors, not the absence of unwanted decision paths.

Intrinsically interpretable methods pose a solution for the latter. Here, the structure of the predictor itself is interpretable, which makes subsequent explanations superfluous. Intrinsically interpretable methods, however, have the reputation of being far less powerful than deep networks. DTs are the prime example of such methods, since their whole structure is made of understandable if-else connections. Some of the abovementioned methods of XAI even transfer the decision path of more complex networks onto DT structures for explanation purposes, but remain uninterpretable since the actual decision making is performed by the complex network. An elegant solution was proposed in [53] under the term TREPAN, where a DT classifier was trained to imitate a deep network's decisions. TREPAN uses the deep network's predictions as ground truth data, therefore a perfectly performing DT would still be limited to the deep network's performance. Nevertheless, the replacement of the deep network with the TREPAN DT leads to an intrinsically interpretable predictor. Very deeply grown trees might lose interpretability, when the sheer amount of interconnected if-else statements becomes too large to comprehend at once. The same effect is valid for widely grown trees, that may yield too many separate decision paths. Ensemble methods, which combine a multitude of decision paths to yield a single prediction result, also forfeit interpretability.

3

Interpretable Feature Generation using Deep Neural Networks

A common saying in computer science describes that "a function with a bad or erroneous input will yield bad results". The dependency on the input quality can thereby be transferred not only to the correctness of a function, but also to its interpretability. In order to deliver an explanation for a produced result, the inputs have to be understandable, or else every attempt at an explanation will run into a dead-end. Many commonly used input domains such as images or natural language are easily understandable. In [54], this property is matchingly described as *implicit explanatory.* Others, like multivariate time series, do not have a natural audio or visual representation which makes them non-interpretable at first glance. The contribution of this chapter is a novel approach towards generating an interpretable classification of MTS by taking advantage of the complexity of DNNs. The enabler of this new approach is a combination of a state-of-the-art heatmapping technique with an exploitation of MTS-specific properties, resulting in a mechanism which not only offers a viable feature subset, but also introduces interpretable new features to the learning problem. It can therefore be considered a combined feature generation/selection method. Figure 3.1 shows, how the method is located between the processing stage of the input data and the classifier, as the input data is modified in terms of its representation before handing it over to the classification algorithm.

Visualization and heatmapping techniques [52, 55, 56, 57, 58] have gained a lot of attention recently, as they provide visual explanations for the decisions of a network.

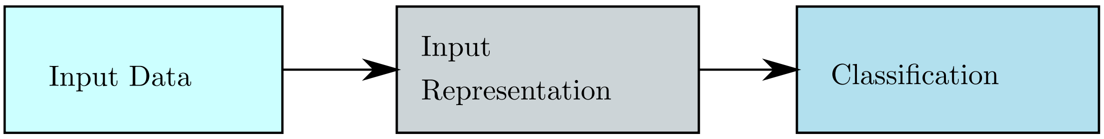


Figure 3.1: Input representation as processing stage

The property of heatmapping methods to establish interpretability in the input space make them an attractive tool for recursively assessing a classification by looking at the processed combination of the networks trained parameters and the given input values. Heatmapping can be performed on high-complexity networks and generate up to one-pixel precision representations [52], depending on the technique applied.

The remainder of the chapter is structured as follows: Section 3.1 examines some classic and novel approaches for feature selection, not necessarily bound to ML applications. Section 3.2 contains a detailed description of the proposed method to generate an interpretable dataset and classifier. The experimental results are examined in Section 3.3.

3.1 Common Approaches for Feature Selection

A multitude of different methods for feature selection have been proposed. Following the definition from [59], these methods can be split into wrapper and filter methods. Wrapper methods measure the importance of a feature given a preexisting classifier, while filter methods evaluate the relevance of the features by using statistical approaches. A classical wrapper methods is the recursive feature elimination (RFE) [60]. RFE requires an external measure of feature importance and iteratively eliminates the least important features. RFE is easy to apply, but comes with the possible drawbacks in ensemble learning methods as shown in [61][62]. Feature ranking and selection through sensitivity analysis [63][64] provide a subset of the features by using saliency information. These methods do not take feature correlations into account which, for MTS structured data, applies to temporal relations as well. Additionally, those methods are prone to locally strong gradients. Feature selection specifically designed for multivariate time series is proposed in [65], which applies Principal Component Analysis (PCA) [66] to identify the features that contribute most to the top principal components. PCA has been primarily used a dimensionality reduction method. By combining the axes of the original feature

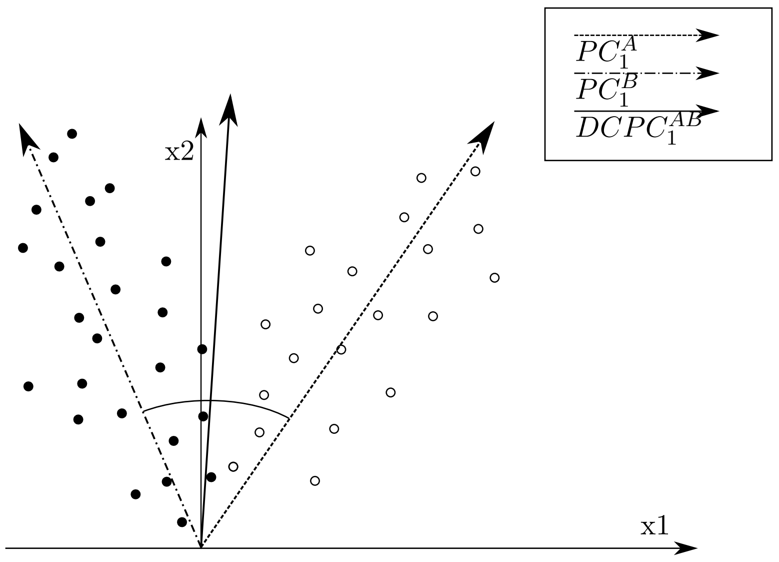


Figure 3.2: First DCPC as bisection of two first PCs.

space and redefining features along the principal components, interpretability is forfeit as the combination of multiple correlated features is hardly comprehensible. Following an approach published by [67], a calculation of the *Descriptive Common Principle Components* (DCPC), lead to comparability between two distributed variables. This step facilitates the use of the PCA method as feature ranking mechanism. First, the PCA is computed for each MTS sample. A defined number of p principal components is thereby retained. Figure 3.2 illustrates the concept, given two variables x_1 and x_2. Two multivariate data samples A and B are marked as ○and ●, respectively. Such data samples could be e. g. the single observations of a measured time series. The figure shows, how the first DCPC is derived as angular bisection from the first principle components of the samples. With each following MTS sample, the angular bisection is repeated, leading to an iterative approach to eventually calculate the DCPC. The feature importance ranking is ultimately established by measuring the loading, i. e., the influence of each variable on the DCPCs, and ranking them in decreasing order, so that the most important variable is the one with the largest influence on the DCPCs. The approach targets MTS samples of varying length. Therefore, only the ranking of features, but not of time frames or windows within a single feature can be realized with the proposed technique. The localization of explicit subsequences of single variables within an MTS structure however has the potential to further reduce the dimensionality without eliminating important data from the input space. Furthermore, the necessary calculation of the correlation matrix becomes computationally expensive when handling large-scale datasets with an extensive amount of features. Hence, in preparation for increasingly large datasets in the field of automated driving, the method proposed in this chapter does not require the calculation of a correlation matrix.

3.2 Interpretable Classification using Heatmap Feature Selection for Multivariate Time Series

This section introduces the architecture for a combined feature generation and selection approach with the potential to facilitate the training of powerful end-to-end interpretable classifiers for MTS. The key to this approach is the generation of heatmaps on mappings of a trained DNN, which itself will not be used for inference later on. The proposed method does not use the input dimension as the interpretable layer, but utilizes the output of the first layer of a partially predefined DNN, which sharpens the interpretation domain. In order to establish this domain, the proposed architecture includes a fixed set of layers with specified parameters in front of an otherwise arbitrary layout for the DNN. Figure 3.3 shows the whole architecture with its components described in the following sections.

3.2.1 Learning FIR Filters

CNNs play out their strength when it comes to image classification and object localization. CNNs, as compared to MLPs, have the ability to take neighborhood relations into account, and can be trained to handle learning problems with significantly less parameters while achieving high classification accuracies. Multidimensional neighborhood relations, which are implied when using CNNs, cannot be directly applied to MTS though. Here, as opposed to image data, the vertical order of the features for an input sample $\boldsymbol{X}^{(m)}[n] \in \mathbb{R}^{F \times N}$ is arbitrary and interchangeable. Therefore, convolutions with a multidimensional kernel would lead to an implicit pattern between two or more features and mislead the network into neglecting more important feature relations without direct neighborhood in the input vector. With this in mind, the first layer in the interpretable network consists of one-dimensional convolution operations with C kernels. The kernels

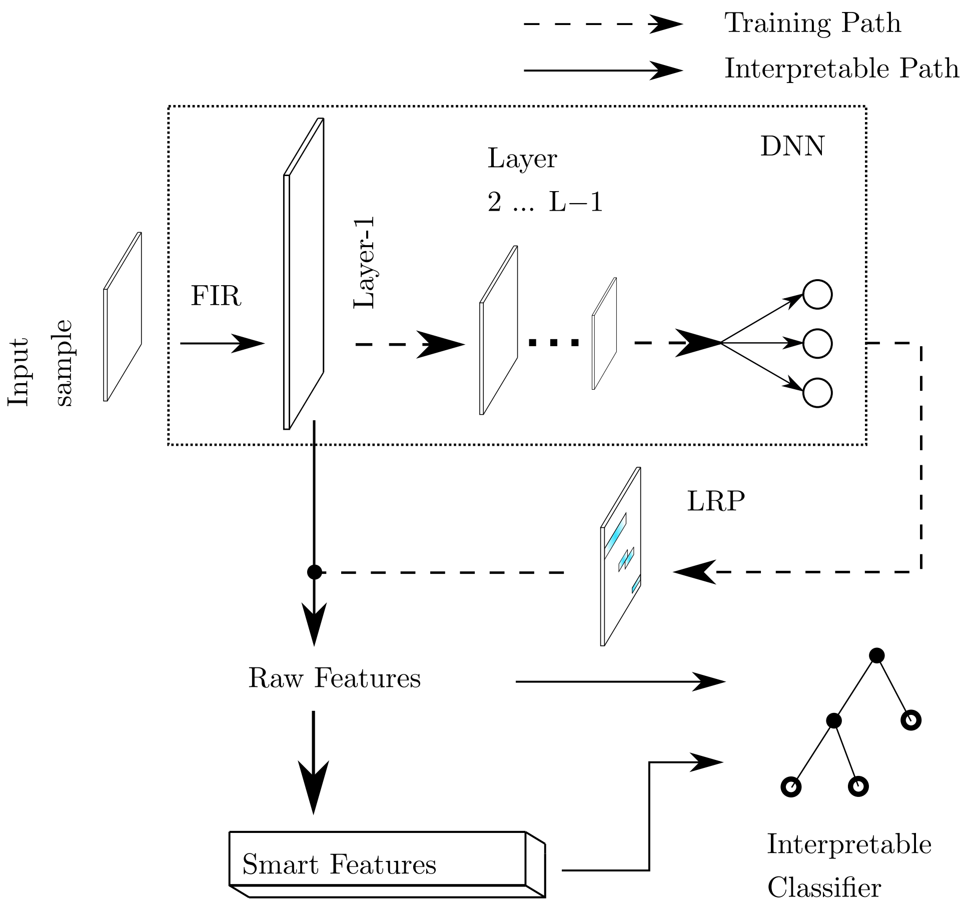


Figure 3.3: The complete process of the feature generation and selection.

$\{\boldsymbol{b}_1, \boldsymbol{b}_2 \ldots, \boldsymbol{b}_C\}$ convolve each time series individually, and can therefore be seen as a set of *Finite Impulse Response* (FIR) filters. The layer activations can be formulated as

$$\tilde{\boldsymbol{X}}_c^{(m)} = \begin{bmatrix} \boldsymbol{x}_1^{(m),\top} * \boldsymbol{b}_c^\top \\ \vdots \\ \boldsymbol{x}_F^{(m),\top} * \boldsymbol{b}_c^\top \end{bmatrix} \in \mathbb{R}^{F \times N'}, \tag{3.1}$$

where $\boldsymbol{x}_f^{(m),\top} * \boldsymbol{b}_c^\top$ denotes the convolution of the two sequences $\boldsymbol{x}_f^{(m),\top}$ and $\boldsymbol{b}_c^\top$. N' denotes the length of the sequences after the convolution. Processing MTS as an image-like two-dimensional stack of univariate time series has been proposed by [68], which used this domain to apply different-length filters and to be able to subsequently use pooling operations in order to maintain equal dimensionalities after these filters. The use of such a layer to generate a fully interpretable intermediate layer with enhanced expressiveness has not yet been proposed.

The first layer is followed by a reshaping function

$$\tilde{\boldsymbol{X}}_{\mathrm{L1}}^{(m)} = \begin{bmatrix} \tilde{\boldsymbol{X}}_1^{(m)} \\ \tilde{\boldsymbol{X}}_2^{(m)} \\ \vdots \\ \tilde{\boldsymbol{X}}_C^{(m)} \end{bmatrix} = \begin{bmatrix} (\boldsymbol{x}_1^{(m),\top} * \boldsymbol{b}_1^\top) \\ \vdots \\ (\boldsymbol{x}_F^{(m),\top} * \boldsymbol{b}_1^\top) \\ \vdots \\ (\boldsymbol{x}_1^{(m),\top} * \boldsymbol{b}_C^\top) \\ \vdots \\ (\boldsymbol{x}_F^{(m),\top} * \boldsymbol{b}_C^\top) \end{bmatrix} = \begin{bmatrix} \tilde{\boldsymbol{x}}_{1,1}^{(m)} \\ \vdots \\ \tilde{\boldsymbol{x}}_{F,1}^{(m)} \\ \vdots \\ \tilde{\boldsymbol{x}}_{1,C}^{(m)} \\ \vdots \\ \tilde{\boldsymbol{x}}_{F,C}^{(m)} \end{bmatrix} \tag{3.2}$$

with $\tilde{\boldsymbol{X}}_{\mathrm{L1}}^{(m)} \in \mathbb{R}^{F' \times N'}$ and $F' = F \cdot C$. This reshaping function flattens the channels and features to form a two-dimensional array of stacked time series. A similar reshaping action was proposed for the Inception Network [51], and adopted by [68] as deep concatenation.

Interpretability is fully contained throughout this step, as any value of the layer activations can be associated to one point in time and one feature, with a defined action of signal processing applied. Henceforth, the layer space after this convolution operation is referred to as the *layer-1 space*, while the processed training dataset will be termed as $\mathcal{D}'$. The following steps of the approach will project the results back to this dimension instead of the original input space. The advantage of this projection is a more detailed representation of each feature, with each element not only indexing one timestep but also

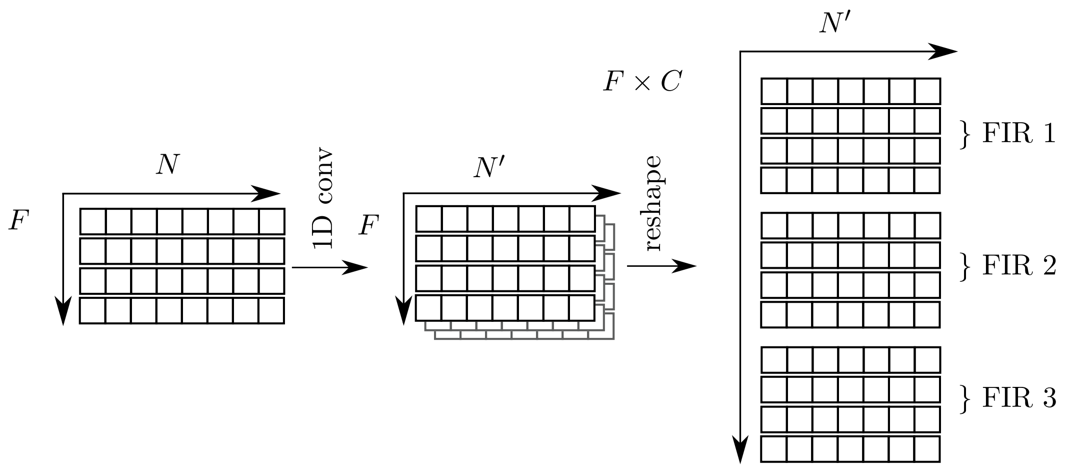


Figure 3.4: Network structure for the interpretable convolution step.

one specific action of signal conditioning applied e. g., a time series in the layer-1 space following a low-pass filtering convolution represents the low-frequency signal components of the original input feature. It should be noted, that by filtering the signal without padding, the run-in an run-out behavior of the filter is cut off. Although the signal is shortened by this effect, the output can still be interpreted as time dimension. Figure 3.4 shows the network structure for the first layer including the reshape operation.

3.2.2 Vertical Feature Convolution and Deep Neural Network

As described in Section 3.2.1, there are no local relationships between adjacent vectors in the input space, as well as after the first convolutional layer. In the second layer, a two-dimensional convolution is applied to the network with $C^{(\mathrm{L2})}$ kernels $\boldsymbol{B}^{(\mathrm{L2})} \in \mathbb{R}^{F' \times s}$, with s denoting the horizontal width of the kernel and F' the total number of input features in the second layer. The kernels of this layer detect the most descriptive repeating patterns of the preceding layer. Through this step, all features can be related to each other, without assuming neighborhood relations.

A convolution operation reduces the size of each dimension depending on the size of the kernel applied. The reduction of dimension N' onto the following layer can be expressed as

$$N^{(\mathrm{L3})} = N' - s + 1. \tag{3.3}$$

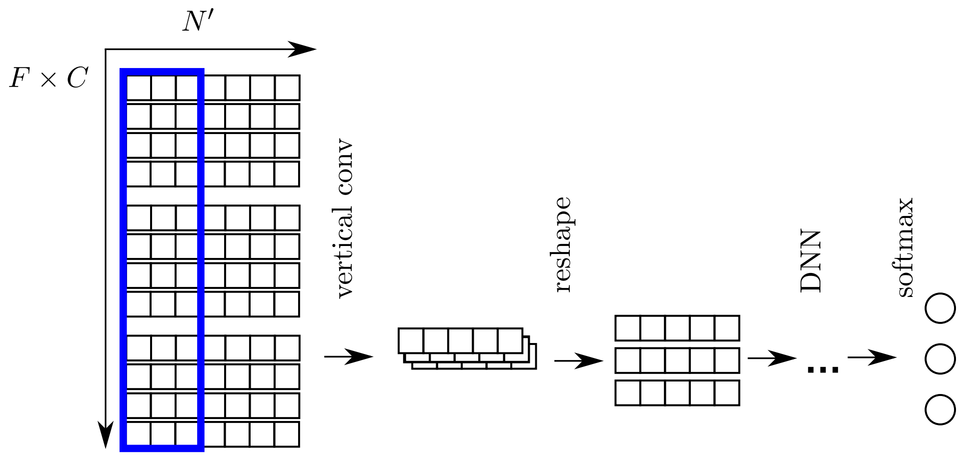


Figure 3.5: Network structure for the vertical convolution step.

In the vertical convolution layer, one kernel dimension is equal to the height of the input. Therefore, the resulting activation is a vector, denoting the existence of the activation pattern as defined by the kernel for each timestep. When using several of these kernels, the layer will be able to represent a variety of activation patterns. A reshaping operation reestablishes the two-dimensional representation of the layer for further processing

$$f : \mathbb{R}^{1 \times N^{(\mathrm{L3})} \times C^{(\mathrm{L2})}} \rightarrow \mathbb{R}^{N^{(\mathrm{L3})} \times C^{(\mathrm{L2})}}, \tag{3.4}$$

with the superscripts (L2) and (L3) denoting the corresponding layers of the variables. In Figure 3.5 the vertical convolution process is illustrated with the kernel depicted in blue. After the reshaping operation, further convolutional and fully connected layers can be added to the network without limitations regarding the kernel size or number of neurons, respectively. The total number of layers should be chosen in accordance to the expected complexity of the learning problem.

3.2.3 Layer-Wise Relevance Propagation

After the network has been trained to a satisfactory performance, the *Layer-Wise Relevance Propagation* (LRP) [52], a state-of-the-art heatmapping technique, is applied to emphasize the salient areas of the input. LRP redistributes the pre-softmax intermediary sum of a single class activation score as *relevance scores* backwards through the network onto each node. The concept of relevance not only describes the activation of a node during prediction but much rather those parts of the activation of the nodes that have

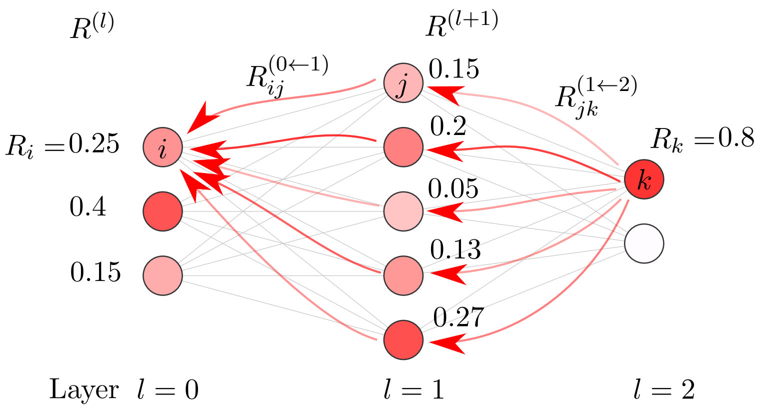


Figure 3.6: Distribution of relevances onto an input node i, starting from an output node k.

been meaningful to the final class prediction. This is due to the fact that the activation is backpropagated from a single class activation score. LRP works with a conservation principle that redistributes the relevance scores backwards through the layers

$$R^{(l)} = R^{(l+1)} \; \forall \; l \in \{0, 1 \ldots, L-1\}, \tag{3.5}$$

with $R^{(l)}$ denoting the cumulative relevance score of layer l and $R^{(l+1)}$ the relevance score of the following layer. Figure 3.6 demonstrates the redistribution of a class score as starting relevance $R_k = 0.8$ through the nodes in layer 1 onto the nodes in the input layer. The conservation principle from equation (3.5) can be seen in the sum of the denoted relevances per layer. For better readability, the relevance flow between layer 1 and layer 0 is visualized for only one input node i. LRP has been originally developed for networks using ReLU nonlinearities. The propagation rules as presented are therefore designed to work well with ReLU activated networks.

The relevance score of a single neuron $R_i^{(l)}$ is composed of all partial relevances $R_{ij}^{(l\leftarrow l+1)} \in \mathbb{R}$ that are redistributed from nodes j in the subsequent layer

$$R_i^{(l)} = \sum_j R_{ij}^{(l\leftarrow l+1)}. \tag{3.6}$$

The propagation rules introduced in [52] were developed heuristically. Several rules have been created that redistribute the relevances from one layer to the previous layer. These propagation rules have different applications regarding the network layout and the propagation focus as elaborated further in this section. The core element for the propagation rules consists of a fraction that attributes a proportional relevance contribution $R_{ij}^{(l \leftarrow l+1)}$ to each node

$$R_{ij}^{(l \leftarrow l+1)} = \frac{z_{ij}}{z_j} R_j^{(l+1)}. \tag{3.7}$$

The most basic propagation rule only considers the learned network weights. The w^2-rule

$$R_{ij}^{(l \leftarrow l+1)} = \frac{w_{ij}^2}{\sum_{i'} w_{i'j}^2} R_j^{(l+1)} \tag{3.8}$$

uses the squared weights to propagate the relevance scores, where i' denotes the index i within the denominator sum. The proposed rule propagates paths containing positive and negative weights equally as positive relevances. However, this propagation rule does not take the network activations into account. The resulting explanation is therefore in the context of the trained network and its structure, instead of in the context of a provided input.

A propagation rule that makes use of the layer activations, as well as the weights, is the z-rule

$$R_{ij}^{(l \leftarrow l+1)} = \frac{a_i w_{ij}}{\sum_{i'} a_{i'} w_{i'j}} R_j^{(l+1)}, \tag{3.9}$$

with a_i denoting the activation of neuron i. The z-rule propagates both, positive and negative values. In a scenario, where explainability through heatmaps is the goal, it makes sense to distinguish these values. Nodes that are attributed a positive relevance affected the decision in a positive way. Vice versa, nodes that end up with negative relevance scores contributed negatively towards a decision. However, it it possible, that a network node shows a positive activation solely because of a positive bias term. In this case, the sign on the relevances would falsely switch, if the bias term is ignored in the relevance propagation following (3.9). A solution to this problem proposed by [69] is to redistribute relevance to the bias terms, leading to

$$R_i^{(l)} = \sum_j \frac{a_i w_{ij}}{\sum_{i'} a_{i'} w_{i'j} + b_j} R_j^{(l+1)} + \frac{b_j}{\sum_{i'} a_{i'} w_{i'j} + b_j} R_j^{(l+1)}. \tag{3.10}$$

A redistribution of relevance onto the layer-specific bias term leads to a violation of the global validity of the conservation principle, since relevance dissipates in the bias. The conservation principle is then "locally" valid, i. e., between two subsequent layers.

Another solution by [70] requires model-specific constraints to avoid the described problem by allowing only negative biases.

The most commonly chosen rule when applying LRP to a network due to its versatility is the $\alpha\beta$-rule

$$R_{ij}^{(l \leftarrow l+1)} = \left(\alpha \frac{a_i w_{ij}^+}{\sum_{i'} a_{i'} w_{i'j}^+} + \beta \frac{a_i w_{ij}^-}{\sum_{i'} a_{i'} w_{i'j}^-} \right) R_j^{(l+1)}, \tag{3.11}$$

with

$$\alpha + \beta = 1, \quad \alpha \geq 1, \quad \beta \leq 0, \tag{3.12}$$

and with

$$w_{ij}^+ = \begin{cases} w_{ij} & w_{ij} > 0 \\ 0 & \text{else,} \end{cases} \tag{3.13}$$

and

$$w_{ij}^- = \begin{cases} w_{ij} & w_{ij} < 0 \\ 0 & \text{else.} \end{cases} \tag{3.14}$$

This rule allows one to weigh positive and negative components of the relevance propagation differently. For a selection of $\alpha = 1, \beta = 0$, only positive relevances will be propagated, also known as the z^+-rule. For this rule, the bias terms can potentially play a role. The two solutions as elaborated for the z-rule can be applied in this case. The results that have been generated for this work used the $\alpha\beta$-rule with the parametrization $\alpha = 1, \beta = 0$. The saliency maps resulting from this parametrization showed the best performing feature extraction, as elaborated in the next section.

Other activation based heatmapping approaches, such as VisualBackProp [58], lack the precision to recreate appropriate heatmaps from one-dimensional kernel activations due to the omitted weights during the backpropagation. Subsequently, heatmaps could not be created for any layer before the vertical convolution step, which contains such one-dimensional activations.

3.2.4 Hot Region and Smart Feature Extraction

End-to-end interpretability cannot be provided for the processes of a DNN. Nevertheless, through heatmapping techniques, the power of a DNN can be utilized to determine the most important areas of an input image, henceforth referred to as *hot regions.* By focusing on only those regions, the interpretable layer-1 space can be strongly reduced. Based on the estimation of the distribution of relevance scores for each heatmap, hot

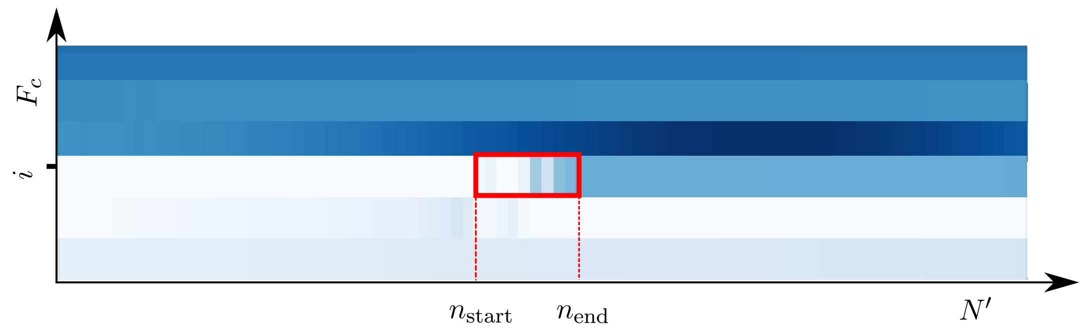


Figure 3.7: Hot region projected on sample $\tilde{\boldsymbol{X}}_c^{(m)}$

regions are detected by thresholding correspondingly high quantiles against their mean values for every time instance. All consecutive relevance scores that exceed the threshold count towards the same hot region. It is possible to extract multiple hot regions not only within one heatmap, but also within the time series of the same feature. The larger the chosen DNN is designed, the more it can capture correlations between its features in each layer and therefore better locate multiple correlated strong hot regions. By locating and collecting the hot regions of all samples, a class-agnostic set of features can be created, which can be formulated as

$$\mathcal{H}' = \{(\boldsymbol{h}_1, \boldsymbol{f}_1) \dots (\boldsymbol{h}_\gamma, \boldsymbol{f}_\gamma)\}, \text{ with} \tag{3.15a}$$

$$\boldsymbol{h}_\gamma = \begin{bmatrix} n_{\gamma,\text{start}} & n_{\gamma,\text{end}} \end{bmatrix}^\top \in \mathbb{R}^2, \tag{3.15b}$$

$$\boldsymbol{f}_\gamma = [i, c]^\top \quad i \in \{1 \dots F\}, c \in \{1 \dots C\}, \tag{3.15c}$$

where $n_{\gamma,\text{start}}, n_{\gamma,\text{end}}$ denote the start and end of the γ-th hot region in the sequence defined by $\boldsymbol{f}_\gamma$ with the condition $n_{\gamma,\text{end}} - n_{\gamma,\text{start}} > 0$. Note that by definition, the resulting set does not contain redundant elements. Figure 3.7 shows, how one hot region is projected onto a given sample $\tilde{\boldsymbol{X}}_c^{(m)}$. F_c denotes the features after the convolution with a kernel $\boldsymbol{b}_c$, where i and c are denoting the components of $\boldsymbol{f}_\gamma$ to locate the hot region. In the context of the set $\mathcal{H}'$, the term *feature* no longer refers to the whole sequence of a signal, but to one distinct value of the set.

When extracting hot regions from the heatmap of every training sample, the resulting feature set becomes large very quickly. For similar driving situations, the resulting hot regions are found to be similar as well. A pruning operation can be applied to generate a subset $\mathcal{H} \subseteq \mathcal{H}'$, excluding those objects $(\boldsymbol{h}_\beta, \boldsymbol{f}_\beta)$ that fulfill the equations

$$\boldsymbol{f}_\alpha = \boldsymbol{f}_\beta \tag{3.16a}$$

$$|h_{\alpha,1} - h_{\beta,1}| < \tau_{\text{lim}} \tag{3.16b}$$

$$|h_{\alpha,2} - h_{\beta,2}| < \tau_{\text{lim}} \tag{3.16c}$$

for any $(\boldsymbol{h}_\alpha, \boldsymbol{f}_\alpha) \in \mathcal{H}$, with $h_{\alpha,1}, h_{\alpha,2}$ denoting the first and second entry of the vector and τ_{lim} as arbitrary threshold for the timely deviation of two hot regions. Depending on the desired amount of features, a further reduction can be achieved by additional clustering of the relevance maps.

The advantage of realizing feature selection through LRP as opposed to sensitivity analysis or feature importance rankings is the identification of whole regions instead of highly activated single points in time. Therefore, further features for each hot region can be generated by utilizing the time series structure of the input data. This process can be performed on all hot regions in an unsupervised manner, and be considered as a feature extraction method. It is facilitated by using expert knowledge of the input data structure, not the provided input data itself or the network architecture.

The features to be generated are henceforth referred to as *smart features*. They can either represent filtered signal values from the layer-1 space with a variable location in the hot region, or processed values of multiple frames within a hot region. This work proposes four smart features, including the mean step difference

$$\text{diff}(\tilde{\boldsymbol{X}}_{\text{L1}}^{(m)}; \boldsymbol{h}_\gamma, \boldsymbol{f}_\gamma) = \frac{1}{h_{\gamma,2} - h_{\gamma,1}} \sum_{n=h_{\gamma,1}}^{h_{\gamma,2}-1} \left(\tilde{x}_{\boldsymbol{f}_\gamma}^{(m)}[n+1] - \tilde{x}_{\boldsymbol{f}_\gamma}^{(m)}[n] \right), \tag{3.17}$$

the mean signal energy

$$\text{E}(\tilde{\boldsymbol{X}}_{\text{L1}}^{(m)}; \boldsymbol{h}_\gamma, \boldsymbol{f}_\gamma) = \frac{1}{h_{\gamma,2} - h_{\gamma,1}} \sum_{n=h_{\gamma,1}}^{h_{\gamma,2}} |\tilde{x}_{\boldsymbol{f}_\gamma}^{(m)}[n]|^2, \tag{3.18}$$

as well as the minimum and maximum values within a hot region

$$\max(\tilde{\boldsymbol{X}}_{\text{L1}}^{(m)}[n]; \boldsymbol{h}_\gamma, \boldsymbol{f}_\gamma) = \max\left(\tilde{x}_{\boldsymbol{f}_\gamma}^{(m)}[h_{\gamma,1}], \tilde{x}_{\boldsymbol{f}_\gamma}^{(m)}[h_{\gamma,1}+1], \ldots \tilde{x}_{\boldsymbol{f}_\gamma}^{(m)}[h_{\gamma,2}] \right), \tag{3.19a}$$

$$\min(\tilde{\boldsymbol{X}}_{\text{L1}}^{(m)}[n]; \boldsymbol{h}_\gamma, \boldsymbol{f}_\gamma) = \min\left(\tilde{x}_{\boldsymbol{f}_\gamma}^{(m)}[h_{\gamma,1}], \tilde{x}_{\boldsymbol{f}_\gamma}^{(m)}[h_{\gamma,1}+1], \dots \tilde{x}_{\boldsymbol{f}_\gamma}^{(m)}[h_{\gamma,2}]\right). \tag{3.19b}$$

Additionally, the raw layer-1 signal values $\tilde{x}_{\boldsymbol{f}_\gamma}^{(m)}[h_{\gamma,1}]$, $\tilde{x}_{\boldsymbol{f}_\gamma}^{(m)}[h_{\gamma,2}]$ were found to be important for further classification tasks as well, although they are not referred to as smart features. A new dataset $\tilde{\mathcal{H}}$ can now be generated, which is composed of the abovementioned layer-1 space signal values and the corresponding generated smart features for each hot region. This dataset has a considerably lower dimensionality after omitting the unimportant regions of the layer-1 space, enriched by the smart features to add additional information to the important regions. The dimensionality of $\tilde{\mathcal{H}}$ depends on the dataset, as well as the chosen hyperparameters. For applications where a fixed number of features is necessary, it is possible to add an additional feature importance measure to the method. By that, the desired amount of the most important features can be isolated to achieve a suitable dataset for the given dimensionality limitations. Even though this is not required for the application case in this work, the performance of a subset after an additional feature limitation step is added to the results table in Section 3.3. A classifier such as a DT can now be learned to generate an interpretable classification. For this, the uninterpretable steps in the training phase are omitted, as all required information for the feature selection and extraction has been gathered to classify new samples along the interpretable path.

3.3 Experimental Results

The first part of the experiments to evaluate the proposed method are conducted on the highD dataset introduced in Section 2.2. The DNN used for this experiment is designed to capture the learning complexity adequately and is illustrated in Figure 3.8, where the interpretable layer-1 space is highlighted in light blue. The input dimension $\boldsymbol{X}^{(m)} \in \mathbb{R}^{6\times 100}$ consists of six features, each of them representing distances to surrounding objects as denoted in Table 3.1. To keep the method close to real-life applications, the chosen input vector contains only variables that are also available in the measurement data of state-of-the art testing vehicles. The input samples, recorded with $f_{\text{sampling}} = 25Hz$, contain 4 seconds of street recordings. Each lane change sample is represented with the time window 0.5 s - 4.5 s TTLC.

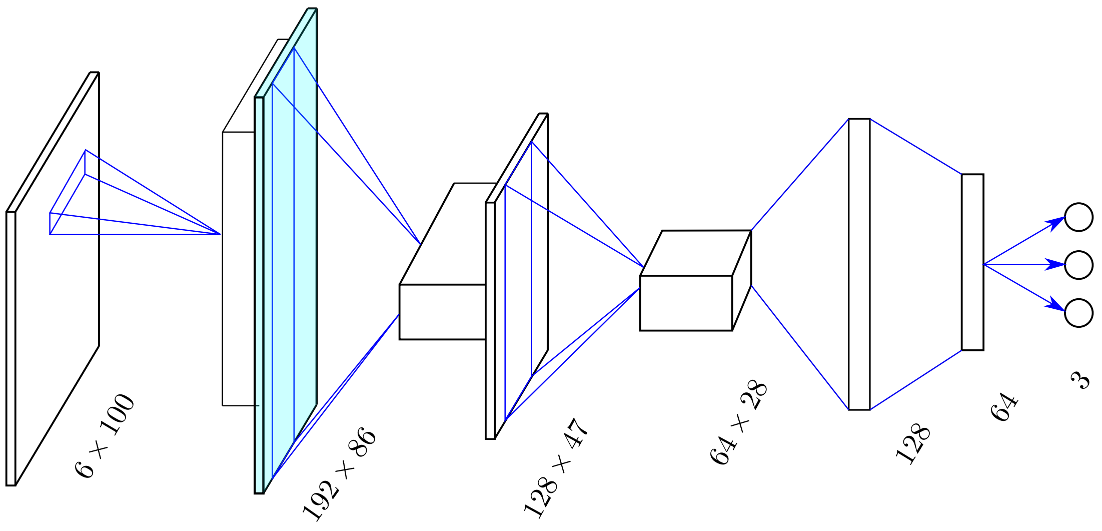


Figure 3.8: Deep Network for initial learning and heatmapping

Feature label	Feature description
$d_{\text{Rear, Left}}$	Distance vehicle behind on left lane
$d_{\text{Front, Left}}$	Distance vehicle ahead on left lane
$d_{\text{Rear, Same}}$	Distance vehicle behind on same lane
$d_{\text{Front, Same}}$	Distance vehicle ahead on same lane
$d_{\text{Rear, Right}}$	Distance vehicle behind on right lane
$d_{\text{Front, Right}}$	Distance vehicle ahead on right lane

Table 3.1: Features in the dataset

To show the effectiveness of the proposed feature selection and generation method, the quality of the new features has to be evaluated. Therefore, multiple feature sets are trained and their classification performance is compared on the same type of classifier. For DTs, two feature ranking measures are popular: The *Mean Decrease in Impurity* (MDI)[15] feature importance, which ranks the features according to the impurity improvement when chosen for a split, and the *Mean Decrease in Accuracy* (MDA)[15][71] feature importance, a strong feature selection method that applies feature permutation to establish the importance ranking. These methods are directly applied to the features in the layer-1 space for reference.

The proposed method is compared with the following feature sets:

- Original Training Dataset $\mathcal{D}$
- Dataset after layer-1 processing $\mathcal{D}'$

- Feature subset (MDI) on $\mathcal{D}'$
- Feature subset (MDA) on $\mathcal{D}'$

The results are presented in Table 3.2. The listed accuracy values refer to a test set containing 3000 samples which was drawn from the original dataset, and whose samples were not used in the training of the classifiers. For reference, the DNN, which is not interpretable and only deployed for the feature generation process, reaches an accuracy of 88.3%. All tested feature sets are listed together with their dimensions and accuracy scores achieved on a DT, as well as on an RF classifier for the main subsets, which suffers from non-interpretability likewise as DNNs. The feature selection methods are tested for multiple dimensions to maximize the comparability between them. The final accuracy score is achieved by fully growing the trees and applying post pruning to achieve better generalization performance on the test set.

Data Subset	**Dimension**	**DT [%]**	**RF [%]**
$\mathcal{D}$	600	80.7	88.8
$\mathcal{D}'$	16512	83.8	89.0
$\mathcal{D}'$ (MDA)	3450	83.7	
$\mathcal{D}'$ (MDI)	3450	80.4	
$\mathcal{D}'$ (MDA)	500	84.1	
$\mathcal{D}'$ (MDI)	500	80.5	
$\tilde{\mathcal{H}}$	3450	**86.4**	**90.0**
$\tilde{\mathcal{H}}$ (MDA)	500	85.6	
$\tilde{\mathcal{H}}$ (MDI)	500	83.2	

Table 3.2: Performance comparison, highD dataset

A general increase of accuracy for the DT, as well as the RF trained on $\tilde{\mathcal{H}}$ can be observed. A comparison between the sets $\mathcal{D}$, $\mathcal{D}'$ and $\tilde{\mathcal{H}}$ shows, that by adding the interpretable layer-1 space, the time series gain expressiveness. The DT accuracy improves significantly, despite the curse of dimensionality which increases the dimensions from from 600 to 16512. This increase can be calculated beforehand by taking the length of the FIR filters, as well as the total number of filters in the first layer into account.

By using the proposed feature generation and selection method, further significant improvement can be observed on the test accuracy, while bringing the dimensions down to about one fifth of the layer-1 space. Overall, the performance of the DT trained on $\tilde{\mathcal{H}}$ is closer to the ones of the uninterpretable DNN and RFs, than to the DT trained without the proposed method. Performing MDA on the layer-1 space to generate a subset with the same dimension as $\tilde{\mathcal{H}}$ does not improve the classification performance, and therefore shows that the accuracy improvement is not solely based on feature reduction, but also on the added information through the smart features. This conclusion is supported by the increased accuracy of the RF trained on $\tilde{\mathcal{H}}$. The MDI importance measure performs worse on the dataset and brings the accuracy down. For a potential application with fixed dimensionality requirements, an additional feature selection step on the feature set $\tilde{\mathcal{H}}$ can be performed. Because $\tilde{\mathcal{H}}$ already contains only condensed features with high information value, the dimensionality reduction with MDA, as well as MDI to 500 features leads to a decrease in accuracy. It should be noted that the performance with MDA is still better than the feature sets in $\mathcal{D}'$. In the $\mathcal{D}'$ domain, the feature reduction to the 500 most important features shows a slight improvement in performance. Here, the positive effect of feature reduction to eliminate noisy features and improve accuracy came to play.

To further emphasize the interpretability of the resulting classifiers, an example of how one possible decision path can be formulated as a chain of logical conjunctions is explained. Within this example, the mean step difference (S1) and the hot region minimum value (S2) as defined in Section 3.2.4 are used as smart features, together with a distance-based decision (R1) as a raw feature: "f the distance to the vehicle in front is decreasing at a rate larger than 2m/s (S1) AND the vehicle behind on the left lane did not get closer than 70m within the last second (S2) AND the vehicle in front on the left lane is further than 50m away (R1), a lane change intention to the left is detected." Examples on how these conjunctions lead to interpretable rule sets are presented in Section 4.4 after the introduction of an interpretable early prediction concept.

The second experiment is conducted with the roundabout dataset. This experiment is designed to emphasize the ability of the method to select and generate features at different points in time. The chosen DNN is depicted in Table 3.3 and is smaller than the network from Figure 3.8, since the dataset contains less samples.

	Conv	Conv	Conv	FC	DO	FC	DO	Output
Parameter	1×5	40×10	5×5	/	0.1	/	0.1	Softmax
Channels	4	8	8	16	/	16	/	2

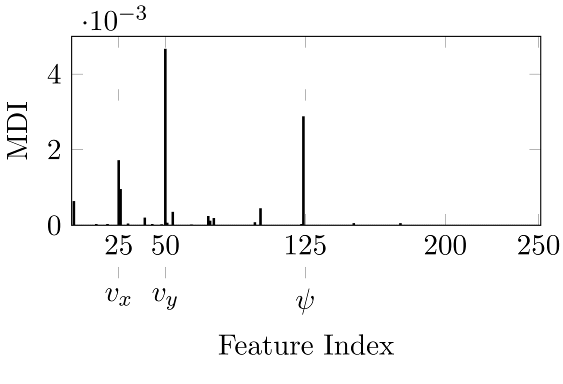
Table 3.3: Network layout for the feature generation on the roundabout dataset

The input to this network are half-second samples $\boldsymbol{X} \in \mathbb{R}^{10\times 25}$. The parametrization of the hot region extraction is set to only consider the most relevant hot regions in order to keep the resulting enriched dataset small in dimension. The feature generation and selection method is evaluated at 3 different TTE points with time distances of 0.5 seconds. The results as shown in Table 3.4 are compared to the performance of the uninterpretable DNN and the DT on the original input set, as well as a RF on the enriched dataset.

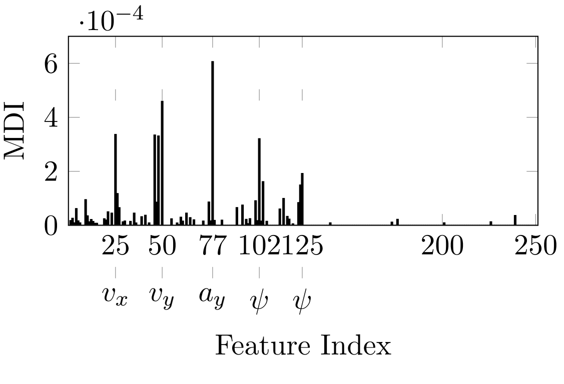
Method	Dataset	TTE 0.5s	TTE 1s	TTE 1.5s
DNN	$\mathcal{D}$	90.1	85.0	82.8
DT	$\mathcal{D}$	90.51	79.3	73.8
DT	$\mathcal{D}'$	92.3	80.3	78.4
RF	$\tilde{\mathcal{H}}$	90.6	87.8	84.6
DT	$\tilde{\mathcal{H}}$	88.9	82.7	82.6

Table 3.4: Results table for the feature generation on the roundabout dataset

The results of this experiment show, that at a TTE of 0.5s, the DTs on the original dataset $\mathcal{D}$ as well as the layer-1 dataset $\mathcal{D}'$ lie on the same level as the performances of the uninterpretable methods DNN and RF, as well as the DT on the enriched dataset. An intuitive explanation for this behavior is, that at a short time distance before the roundabout exit, single feature are exceedingly important for the classification. The DT can identify these features and construct the decision making process around this low number of important input features. The MDI feature importance of the DT on $\mathcal{D}$ for the classifier at TTE$=0.5$s is displayed in Figure 3.9 (a). The three identified features are assigned a significantly higher importance than the others. Since the classifier selects single observations from a timeline, the closest value to the TTE is more relvant than further away observations of the same feature. By visualizing the feature importance on the DT on $\mathcal{D}$ for the classifier at TTE$=1.5$s in Figure 3.9 (b), it becomes apparent that for further prediction horizons it is not sufficient anymore to focus on single dominant features. While the peak importances are still to be found at the limits of the feature time series, the importances are far more distributed to a variety of different features and observations. The DT classifier attempts to recreate complex behavior patterns and is thereby limited by the DT structure as such. Subsequently, the performance for the DT classifiers on $\mathcal{D}$ at TTE$=1$s and TTE$=1.5$s drops. The layer-1 features in $\mathcal{D}'$ already contain increased information that due to the convolutional layer and achieve



(a) DT classifier at TTE= 0.5s



(b) DT classifier at TTE= 1.5s

Figure 3.9: Feature importances for DT classifiers on $\mathcal{D}$

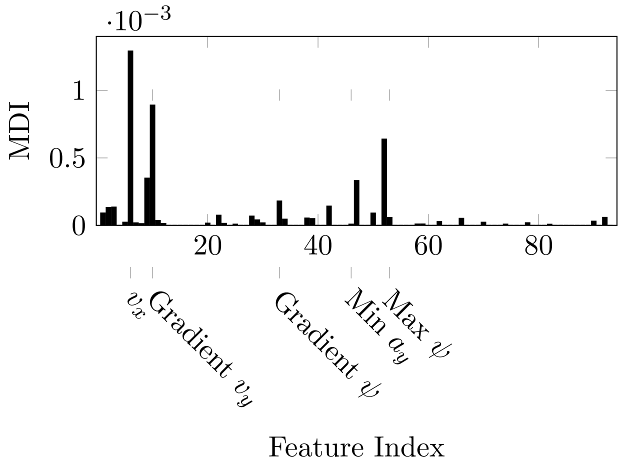


Figure 3.10: Feature importance of DT classifier on $\mathcal{H}$ at TTE= 1.5s

a higher accuracy score than the equivalent classifiers on $\mathcal{D}$. To further examine this hypothesis, a visualization of the feature importances for the DT on $\mathcal{H}$ at TTE= 1.5s is provided in Figure 3.10. The most important features are again denoted sematically. The figure demonstrates, that the enriched dataset with the introduced smart features contains valuable information in a few important features. These automatically designed features contain behavioral patterns over time with the use of neighborhood relations and filtering operations. With the help of these features, the grown trees show compelling performance while remaining shallow and interpretable.

The experimental results show, that by generating a feature subset with the proposed method, classification relevant information is added to the dataset through smart features, while unimportant regions are eliminated. By doing this, the classification accuracy of both, the interpretable DT, as well as the more powerful but non-transparent RF classifier can be increased. From an interpretability standpoint, the feature subset can be comprehended easier, because each hot region contributes multiple related features to the subset, instead of a set of completely individual features.

3.4 Chapter Conclusion

This chapter presents an interpretable feature generation and selection method. Prerequisite for the method is an interpretable input space. By interpreting 1-dimensional convolutions along the time axis, a set of learned FIR filters is established. The space resulting from a tranformation of the input samples into the space established by these FIR filters is therefore deemed interpretable and termed layer-1 space. A DNN is then trained to identify salient coherent areas in this latent space. By extracting selected raw values as well as several statistically relevant properties of those areas, an interpretable dataset with enriched infromation content is created. Experiments on two different datasets show the performance increase when growing DT classifiers on the enriched dataset as opposed to the original input samples.

4

Interpretable Early Prediction of Time Series Data

This chapter introduces an interpretable ML structure for the task of early prediction of lane changes, based on MTS data. A Mixture-of-Experts (MoE) architecture is adapted to simultaneously predict lane change directions and the time-to-lane-change (TTLC). Recurrent networks for time series classification using GRU and LSTM cells, as well as CNNs serve as comparison references. The interpretability of the approach is shown by extracting the rule sets of the underlying classification and regression trees, which are grown using data-adaptive interpretable features. The proposed method outperforms the reference methods in false alarm behavior while displaying a state-of-the-art early detection performance.

The main components of the proposed system are (A) multiple sets of data-adaptive interpretable features that are generated based on DNNs as introduced in Chapter 3 and (B) a Mixture-of-Experts (MoE) structure consisting of intrinsically interpretable DTs. The MoE structure is thereby configured to optimize the individual experts for pre-specified time regions. The developed approach is compared to three reference networks to examine the advantages in performance and interpretability. Early prediction is a prevalent task in autonomous driving. Hence, the proposed approach has several possible applications, of which lane change prediction will serve as prominent example. The rest of this section will introduce the techniques and related publications. Section 4.3 will explain the classification and regression networks in detail. The interpretability of the results is demonstrated in Section 4.4. An evaluation of the performance of the network in comparison the the reference methods is discussed in Section 4.5.

4.1 Related Work

Early classification methods for MTS are typically applied, when the prediction horizon of a classification task is not known in advance. Typical applications that have a time-sensitive nature and benefit from early classification include the fields of medical anomaly detection, predictive maintenance and intention prediction for autonomous driving functions. Within these topics, several methods have been proposed in order to classify time series with a maximized prediction horizon while preserving classification accuracy. A common approach for time series classification is the use of shapelets, which can be defined as class-representative sequences [8]. Recent publications also take MTS into account [9, 10]. Shapelets, as subsequences of time series data, inherit interpretability from the input data, hence several publications claim the extraction of interpretable features [11, 12]. The claim of interpretability is limited to the feature space and does not hold for downstream multi-layer networks. Within the context of *Validation by Design* the proposed method not only has to achieve high accuracy values, but has to provide full intrinsic interpretability.

The prediction horizon for intention predictions is usually highly varying. An evasive lane change maneuver cannot be classified until shortly before, while consciously planned maneuvers can be classified multiple seconds in advance. In order to cover the full time range, a single classifier will have to find globally shared features throughout the whole observation length. A resulting trade-off is the negligence of under-represented scenarios. With the MoE [72], a technique following the *divide-et-impera* principle was proposed. An MoE structure consists of multiple expert networks, that are each trained to perform well within one segment of the overall input space. A meta network is then optimized as a gating mechanism to find the best expert for a given input sample. Within the topic of MoE, the segmentation of the input space into pre-specified subspaces, as it will be used within this chapter, is known as *Mixture of Explicitly Localized Experts* (MELE) [73]. MoE is a classic method in ML, but remains relevant today by introducing DNN experts and its ability to achieve state-of-the-art performances on suitable tasks [74, 75]. In [76], an MoE structure has been used to predict lane change trajectories by classifying the maneuver through an RF and subsequently calculating the trajectory by means of class-specific experts. The benefits of splitting dynamic- and vehicle constellation-based features for highway maneuver classification was proposed by [77]. Network interpretability and the evolution of relevant features through time, that play an essential role within the concept of Validation by Design, have not been considered in these approaches.

4.2 Reference Methods

Three different state-of-the-art networks serve as reference methods. However, all compared reference methods are uninterpretable structures with multiple hidden layers and can therefore only be utilized as performance reference. RNNs are considered to be the state-of-the-art network types for time series classification. Among the most prominent representatives of RNNs are LSTM and GRU networks. In [3], a combined classification and regression network structure with LSTM cells is examined. The application is a lane change intention prediction with an estimated TTLC for the ego vehicle, using internal vehicle data and eye tracking. The publication used a non-public balanced dataset of 1505 lane changes for training and 413 lane changes as test set. The input matrix contains 9 time series features of internal and external signal sources. The network design as proposed can be seen in Table 4.1. The network consists of an LSTM layer with 250 units, followed by two fully connected (FC) layers with high dropout (DO) rates to avoid overfitting. The networks for classification and regression is identical until the point of the activation function, hence the networks are likely to be separately trained to optimize the weights on the specific task.

	LSTM	FC	FC	Output		
Parameter (Cl)	/	/	0.5	/	0.5	Softmax
Parameter (Re)	/	/	0.5	/	0.5	ReLU
Channels	250	256	/	256	/	/

Table 4.1: Network layout for the reference method using LSTM cells by [3]

A comparable classifier with a GRU layer as recurrent structure is proposed in [4]. GRU cells have less hidden parameters than LSTM cells, hence they perform better in smaller datasets. The dataset that has been used by [4] is also a non-public dataset of 432 recorded lane changes. The network architecture is depicted in Table 4.2 and contains two ReLU activated output nodes to determine the two regression results

TTLC and the time to lane change completion. To be able to perform classification and regression tasks, the structure is trained as two separate networks with different output layers - a ReLU activated node for the TTLC regression and the softmax activation class scores for the direction classification task.

	GRU	FC	DO	FC	DO	Output
Parameter	/	/	0.4	/	0.4	ReLU
Channels	180	128	/	128	/	2

Table 4.2: Network layout for the reference method using GRU cells by [4]

Both methods achieve strong prediction performances for the task of lane change prediction and will be used as reference methods. Next to the popularity of RNNs, CNNs are proven to be highly capable of time series classification tasks [78][79]. A convolutional neural network for time series classification and regression will therefore also be part of the reference methods.

4.3 Expert Classifiers and Regression Network

The main idea of MoE is to achieve a better classification performance by splitting the input space and learning an expert classifier for each split. The following sections will explain the generation of the training data, as well as the inner structure of the DNN for the initial training, preceding the interpretability conversion. The global layout of the MoE structure for regression and early classification is shown in Figure 4.1. In this figure, $\boldsymbol{X}^m_{\mathrm{SW}}$ denotes the input, a fixed-length segment of a dataset sample $\boldsymbol{X}^m$, starting at position SW. The outputs for classification and TTLC regression are denoted as $\boldsymbol{y}_{\mathrm{cl}}$ and y_{reg}, respectively. The expert classifiers are denoted as $E0$ to $E5$.

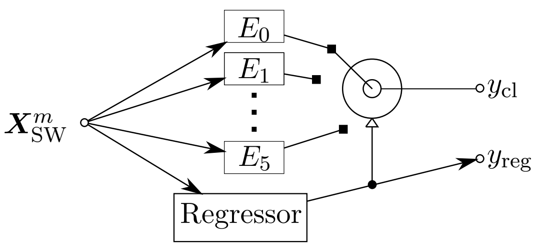


Figure 4.1: Layout of the MoE structure.

4.3.1 Training Data for Lane Change Prediction

The sequences extracted from the original lane change dataset have a common length of $T_{\mathrm{LC,min}}$. For the proposed MoE structure, splits along the time dimension of these sequences are performed to obtain the expert training sets. Figure 4.2 visualizes a lane change sample with the features along the ordinate and the time dimension with the expert splits along the abscissa. These splits are based on the hypothesis, that the relevant features to classify lane changes or any other early prediction task, are considerably different between short- and long-term experts, e. g. $E1$ and $E5$. Additionally, this splitting method facilitates the double assignment of the TTLC regressor as meta network of an MoE structure as depicted in Figure 4.1. Each expert subset contains multiple positions of a fixed-length moving window. Window positions may overlap between two expert regions and are assigned to the subset of the expert closer to the predicted event t_0. For the lange change intention prediction, a time window length $\triangle t = 1s$ is chosen. A time window position (a) at $t = 3s$ defines a subsequence $\boldsymbol{X}_{3s}^{m}$ that is assigned to expert E_3. The earliest subsequence (b) of a sample, associated with expert E_5, ranges from $5.52s$ to $6.52s$. A special case in this splitting design is the first expert E_0. Depending on the provided features, samples right at the predicted event t_0 may contain unambiguous evidence of the labeled class. The expert will consequently learn high weights for those features, yielding a high accuracy classifier with no early prediction performance. By shortening the first expert range, the influence of the learned non-predictive features is purposefully limited.

For the early prediction training, all available features that are described in Table 4.3 are used.

Feature	Description
v_x	Velocity Longitudinal
a_x	Acceleration Longitudinal
a_y	Acceleration Lateral
$d_{\mathrm{left,front}}$	Distance vehicle ahead on left lane
$d_{\mathrm{left,rear}}$	Distance vehicle behind on left lane
d_{front}	Distance vehicle ahead on same lane
d_{rear}	Distance vehicle behind on same lane
$d_{\mathrm{right,front}}$	Distance vehicle ahead on right lane
$d_{\mathrm{right,rear}}$	Distance vehicle behind on right lane
$d_{\mathrm{left,lane}}$	Distance left lane marking
$d_{\mathrm{right,lane}}$	Distance right lane marking

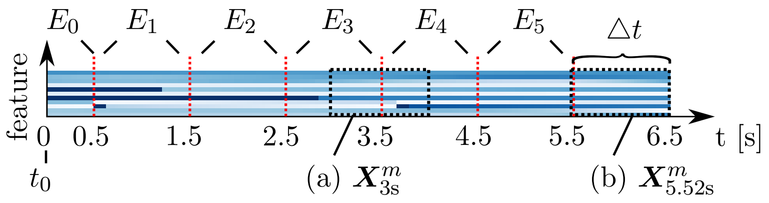


Figure 4.2: Full lane change sample with expert splits.

Table 4.3: Feature description of the lane change prediction

The regressor has to be able to estimate the TTLC over the full range of the dataset. To achieve this goal, the training set of the regressor contains time windows from all positions along the sample, labeled with the position index. All samples of the *NLC* class are explicitly excluded from this training, since no TTLC value can be assigned.

The DNNs for classification and regression that have been trained for the feature selection and generation step share a common layout through the convolutional layers (Conv) and start to diverge in the fully connected (FC) layers. In between the FC layers, dropout (DO) is applied. Table 4.4 lists both network designs for the lane change prediction. Since the regression network is trained with samples across the whole time span of the scenarios, a much larger dataset than the individual experts, an increase in dimensionality for the FC layers turned out to be beneficial for the performance.

An expert range consists of n_{range} discrete timesteps, which yields $M \cdot n_{\text{range}}$ potential training samples. Since a large information overlap can be expected of two adjacent samples of the same lane change sequence, the training set is downsampled to contain every fourth time window position. For the validation and the test set, all available window positions were extracted. Table 4.5 lists the number of samples for training, validation and test sets.

	Conv	Conv	Conv	FC	DO	FC	DO	Output
Parameter (Cl)	1×5	352×10	5×5	/	0.3	/	0.2	Softmax
Channels (Cl)	32	128	128	128	/	64	/	3
Parameter (Re)	1×5	352×10	5×5	/	0.3	/	0.3	Linear
Channels (Re)	32	128	128	256	/	128	/	1

Table 4.4: Network layouts for classification (Cl) and regression networks (Re) of the lane change prediction.

	Scenarios	E_0	$E_1 \dots E_5$	Overall
Training	5388	18576	32508	181116
Validation	1347	17511	33675	185886
Test	627	8151	15675	86526

Table 4.5: Table of dataset sizes for the lane change early prediction.

The gating mechanism of the meta classifier is a winner-takes-it-all strategy, assigning one responsible expert to perform the classification. Responsibility is assigned by comparing the regression result with the range of each expert, which corresponds to its individual training set as denoted in Figure 4.2.

4.3.2 Training Data for Roundabout Exit Prediction

The dataset for the roundabout exit prediction is analogously processed. As described in Section 2.2, a full sample sequence consists of an object travelling through one of the four sectors between the roundabout exits. Sequences with a minimum length $T_{\text{RB,min}} = 2.5\text{s}$ extracted and cropped to a common length. Four experts have been set up with repsonsibility ranges of 0.5 s per expert. The time window length is also set to 0.5 s. Figure 4.3 depicts a roundabout scenario with the expert splits and an example subsequence $\boldsymbol{X}^m_{1.25}$ s assigned to expert E_2. Table 4.7 lists the number of scenarios, the samples per expert as well as the overall number of samples for the roundabout dataset. The features which are used as time series inputs for the roundabout exit prediction are denoted in Table 4.6.

Feature	Description
v_x	Velocity Longitudinal
v_y	Velocity Lateral
a_x	Acceleration Longitudinal
a_y	Acceleration Lateral
ψ	Yaw angle
d_{front}	Distance vehicle ahead
THW	Time Headway
$d_{\text{front, alt}}$	Distance vehicle ahead, not entered
THW_{alt}	Time Headway, not entered
d_{rear}	Distance vehicle behind

Table 4.6: Feature description of the roundabout dataset

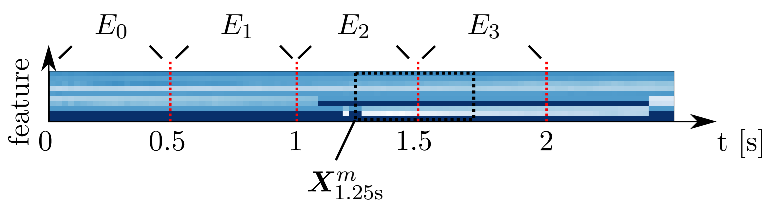


Figure 4.3: Full roundabout sample with expert splits.

	Scenarios	$E_0 \dots E_3$	Overall
Training	2242	29146	116584
Validation	262	2760	26200
Test	276	6900	27600

Table 4.7: Table of dataset sizes for the roundabout exit early prediction.

The network architecture is adapted to the dataset size and depicted in Table 4.8

	Conv	Conv	Conv	FC	DO	FC	DO	Output		
Parameter (Cl)	1×5	80×10	0.4	5×5	0.4	/	0.3	/	0.2	Softmax
Channels (Cl)	8	32	/	32	/	32	/	16	/	2
Parameter (Re)	1×5	80×10	0.4	5×5	0.4	/	0.15	/	0.15	Linear
Channels (Re)	8	32	/	32	/	64	/	32	/	1

Table 4.8: Network layouts for classification (Cl) and regression networks (Re) of the roundabout exit prediction.

4.4 Generation of Interpretable Structures

This chapter seeks to generate an intrinsically interpretable early classification and regression. In Section 4.3, the training of multiple deep networks in an MoE structure is explained. Each of these networks contains multiple hidden layers and are therefore non-interpretable. The chosen network layouts however prepare the MoE structure for a conversion into capable interpretable classifiers.

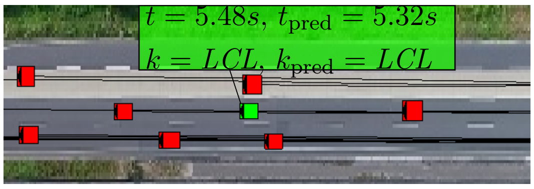
4.4.1 Enriched Feature Generation Using DNNs

Following the process proposed in Chapter 3, data-adaptive interpretable features are generated for each expert and the meta network of the MoE structure. The first layers of the previously trained DNNs consist of 1D convolutions, leading to expert-individual sets of learned FIR filters. An input sample to the structure $\boldsymbol{X}^m_{\mathrm{SW}}$ as depicted in Figure 4.1 will therefore be mapped to E different enriched representations $\boldsymbol{X'}^m_{\mathrm{SW},e}$ for the experts and one representation $\boldsymbol{X'}^m_{\mathrm{SW},r}$ for the regressor. After the feature generation and selection step which yields interpretable enriched input sets, the DNN experts are replaced with DT structures to facilitate a fully interpretable structure.

4.4.2 Interpretation of Predictions

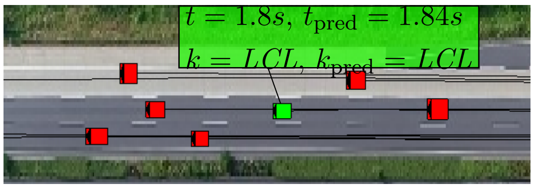
In order to demonstrate the established interpretability, one randomly chosen sample from the lane change prediction test set is visualized at two timesteps before the lane change in Figure 4.4 and Figure 4.5. In the referenced figures, the text box refers to the observed vehicle, whose lane change is to be predicted by the presented MoE method. The top row denotes the TTLC regression ground truth t, as well as the prediction t_{pred}. The bottom row denotes the ground truth class k for the lane change classification and the prediction k_{pred}. The results can be interpreted by extracting the decision rules underlying both constellations from its corresponding expert DTs as seen in Table 4.9. The table rows describe the subsequent features chosen for the splits until a classification is made. Note, that the physical features of input $\boldsymbol{X}^m_{\mathrm{SW}}$ have multiple filtered representations within $\boldsymbol{X'}^m_{\mathrm{SW},e}$.

In Figure 4.4, the regressor shows some deviation from ground truth, but correctly determined expert E_5 for the classification. Constellations from the test set that end up in this leaf of E_5 can be identified as middle-lane scenarios with a slow leading vehicle in front of the observed object. The lane change direction is decided upon by the speed difference to the leading vehicle on the right lane. Table 4.9 (right) illustrates the decision process for the same scenario, but closer to the lane change as depicted in Figure 4.5. Expert E_2 is correctly assigned. By that point in time, the vast majority of scenarios show clear indications of a lane change. Therefore, the decision path mainly consists of splits based on lateral movement and the distance to the lane marking, and does not consider the surrounding vehicles as much as expert E_5.



$t = 5.48s$, $t_{\text{pred}} = 5.32s$
$k = LCL$, $k_{\text{pred}} = LCL$

Figure 4.4: Lane change scenario, $t = 5.48s$ before a lane change to the left.



$t = 1.8s$, $t_{\text{pred}} = 1.84s$
$k = LCL$, $k_{\text{pred}} = LCL$

Figure 4.5: Lane change scenario, $t = 1.8s$ before a lane change to the left.

Split	Input Ft.	Smart Ft.	FIR
1	d_{RearLeft}	Minimum	1
2	$d_{\text{FrontSame}}$	Gradient	21
3	$d_{\text{FrontSame}}$	Gradient	8
4	$d_{\text{RearRight}}$	Minimum	26
5	$d_{\text{FrontSame}}$	Minimum	21
6	a_{x}	Gradient	15
7	$d_{\text{FrontRight}}$	Gradient	21

Split	Input Ft.	Smart Ft.	FIR
1	d_{MarkLeft}	Maximum	11
2	$d_{\text{RearRight}}$	Maximum	17
3	a_{y}	*none*	11
4	d_{MarkLeft}	Minimum	11
5	a_{y}	Gradient	11
6	a_{y}	Gradient	11

Table 4.9: Decision processes of the constellations in Figure 4.4 (left) and Figure 4.5 (right). The column *Input Ft.* refers to the physical features from $\boldsymbol{X}^{m}_{\text{SW}}$, the columns *Smart Ft.* and *FIR* refer to the enriched representations $\boldsymbol{X}'^{m}_{\text{SW},e}$.

4.5 Experimental Results

4.5.1 Early Prediction of Lane Change Intentions

Common methods to compare model performances are typically based on confusion matrices and derived measures. The well-known F-score however only has limited informative value on the classification performance for time series datasets, since the scoring relies on single-frame comparisons between prediction and label. A dataset with M samples, each containing N_{TW} time window positions yields $M \times N_{\text{TW}}$ independent entries in the confusion matrix with no temporal relations. This work aims to retain the temporal information in the evaluation of a learned classifier. The earliness of a reliable prediction and the amount of false alarms are therefore used as suitable and application-oriented measures.

Within the application of lane change prediction, like in most prediction tasks, the set of classes contains a no-event class. This class does not have a specific t_0 associated with a predictable action. Since lane changes are rarely occurring events as compared to lane following, i. e., samples that are labeled *NLC*, an imbalanced dataset would contain a large majority no-event class samples. It is essential to appropriately show the tendency of a classifier to produce false alarms, which are defined as a sequence of at least n_{min} identical lane change classifications within a no-event sample. Vice versa, this smoothing operation also affects true positive detections, since any decision has to be robustly classified for the defined amount of subsequent timesteps. Figure 4.6 indicates the amount of false alarms within the subset of NLC samples. The MoE approach shows strong fluctuations when the smoothing parameter n_{min} is chosen small. RNNs can play out their main strength here, as the preceding states are taken into account for each new decision. Therefore, the smoothing factor benefits the MoE approach more than the compared references. In order to achieve a fair comparison, several points of interest are identified for the classification of lane change samples in the test set. For the assessment of early prediction quality of the conducted experiments, two relevant performance metrics are introduced:

1. The number of false alarms n_{fa}. In general, this number indicats how often an event is falsely predicted. For the application of lane change detection this translates to how often an NLC scenario is classified as lane change in either direction. False alarms could trigger preemptive braking or evasion maneuvers. Therefore, a low tolerance value has to be set to avoid discomfort.

2. The mean reliable prediction time $\mu_{t,\text{rel}}$. Within a single lane change scenario, the point in time when a lane change is correctly classified and not reclassified until the point of the lane change is defined as the reliable prediction time t_{rel}. This parameter is most indicative of the early classification performance of the network.

Table 4.10 presents the results for the MoE in comparison to the CNN, the GRU Network as well as the LSTM network. The table thereby shows the performance of the reference methods without smoothing factor, while n_{min} for the MoE method is adjusted to match the number of false alarms. In this table, μ_{trel} for all methods is comparable, while all methods show indecisive behavior and therefore high false alarm counts. Note, that a single NLC sample may contain multiple false alarms, especially when n_{min} is set low. More reliability can be gained through increasing the smoothing factor. Table 4.11 shows the results for a smoothing factor $n_{\text{min}} = 28$ for all methods. The chosen value is the minimum value to show 0 false alarms with the MoE method, which is the first of the compared methods to reach this limit. Since an increase of n_{min} buys a better false alarm behavior at the cost of a decreasing μ_{trel}, all approaches show a slightly lower mean reliable prediction time compared to the parameter settings in Table 4.10.

Method	CNN	GRU	LSTM	MoE
False Alarms n_{fa}	649	520	348	572
n_{min}	1	1	1	5
μ_{trel}	4.91	4.96	4.87	4.86

Table 4.10: Comparison of the MoE structure with reference methods. Results without smoothing factor applied to reference methods.

Method	CNN	GRU	LSTM	MoE
False Alarms n_{fa}	62	95	103	0
n_{min}	28	28	28	28
μ_{trel}	3.89	3.91	3.84	3.44

Table 4.11: Comparison of the MoE structure with reference methods. Results with smoothing factor to achieve $n_{\text{fa}} = 0$.

A high false alarm rate has negative impact on driving safety and comfort. A lane change detection might continuously instigate braking maneuvers to leave space for other vehicles if the false alarm rate is too high. The results in Figure 4.7 are generated with $n_{\text{min}} = 28$ for all methods, which is the parameter setting for MoE to show no false alarms in the test set. As Table 4.11 shows, the mean reliable prediction time μ_{trel} for MoE has decreased more than those of the compared methods. The reason for this behavior can be explained by short discontinuities of length $t_{\text{disc}} < t_{\text{min}}$ within sequences of correct classifications in the early stage of upcoming lane changes. These discontinuities do not trigger reclassifications and are therefore smoothed out once a classification is made, but prevent the initial classification if long robust sequences n_{min} are required. The parameter setting for MoE in Figure 4.7 offers a fully interpretable classification with a very low false alarm rate. A lane change is reliably detected on average 3.44s in advance. The reference methods offer longer reliable prediction times, but at the cost of significantly more false alarms during lane following. A comparison of the regression performance has to differentiate between early- and late-stage errors. All methods were trained on the same regression dataset, which is described in Section 4.3. Figure 4.8 displays box plots for the TTLC regression of the methods MoE (a), CNN (b), GRU (c) and LSTM (d). For this illustration, the axis t_{gt} denotes the ground

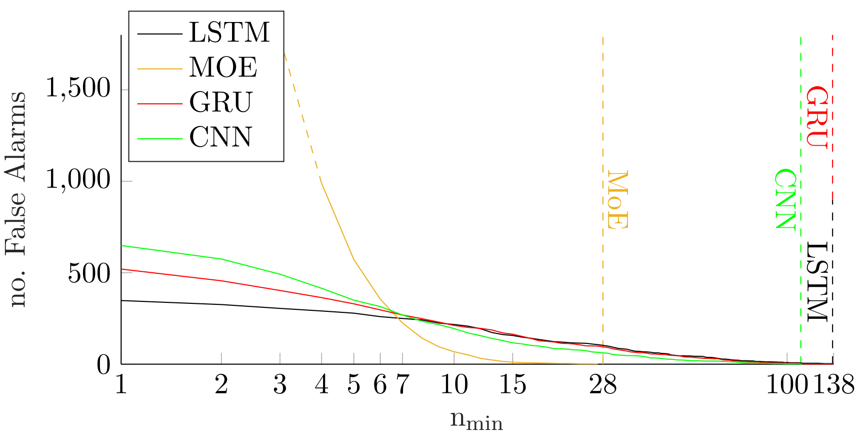


Figure 4.6: False alarms over n_{min}. Vertical lines indicate n_{min} for zero false alarms with the respective method.

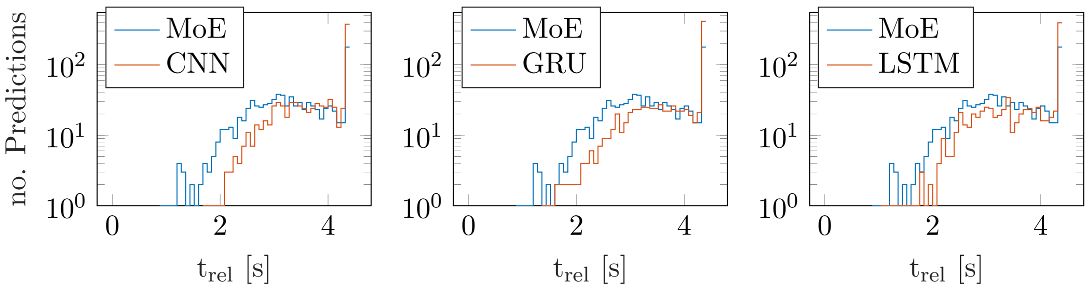
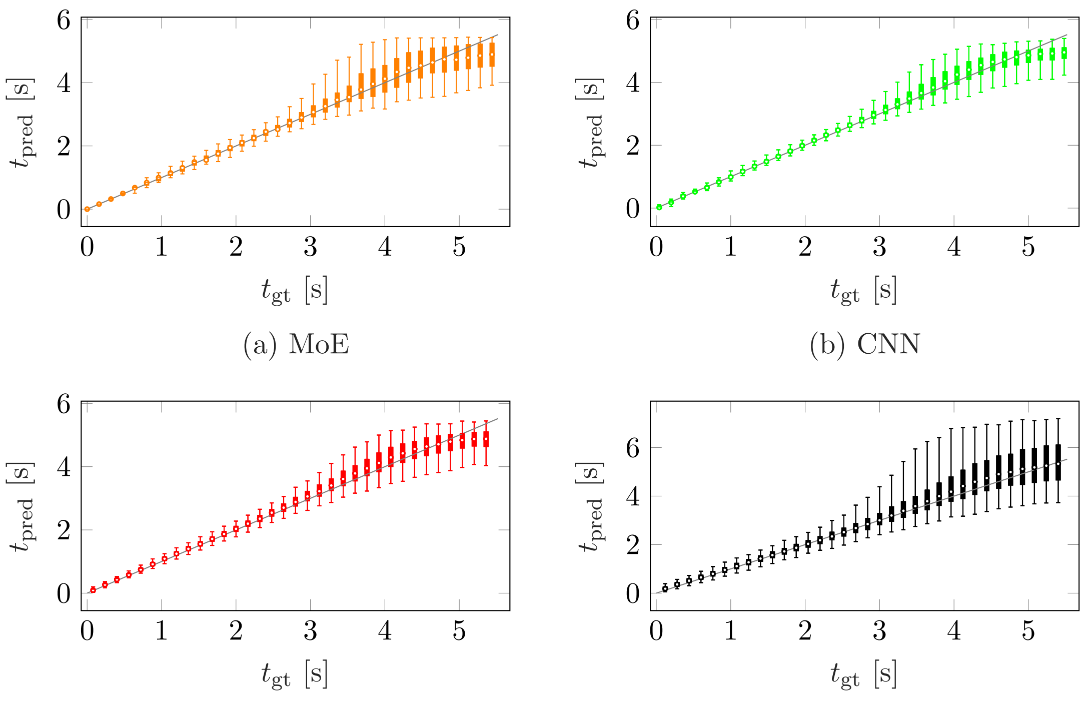


Figure 4.7: Histograms of reliable predictions at time t_{rel} before lane change, referring to the results in Table 4.11.



(a) MoE

(b) CNN

(c) GRU

(d) LSTM

Figure 4.8: Box plots of TTLC predictions.

truth and t_{pred} the predicted value. The whiskers cover 95% of the data, rendering the upper and lower 2.5% as outliers. The ground truth is depicted as gray line for reference. The LSTM network, despite having the largest spread between the quantiles, shows the best ability to keep the median close to the ground truth even for larger values. The proposed MoE approach has a slightly larger distribution than the uninterpretable GRU and CNN networks, particularly visible for values $t_{\text{gt}} > 3s$. All compared networks show high regression accuracy for values $t_{\text{gt}} < 3s$, which is essential for safety critical predictions.

By splitting up the input space it can be shown, that the extracted interpretable classifiers generate human-understandable decision processes. It could further be shown, that the proposed MoE structure outperforms compared methods by means of a low false alarm rate, if a trade-off in earliness of the prediction can be accepted.

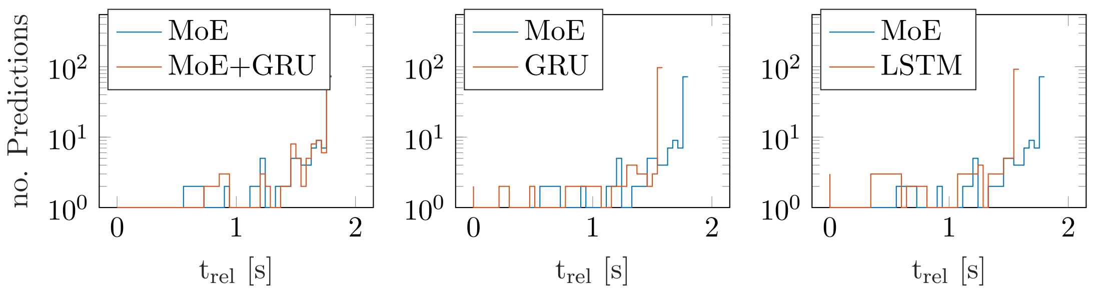


Figure 4.9: Histograms of reliable predictions at time t_{rel} before roundabout exits, referring to the results in Table 4.12.

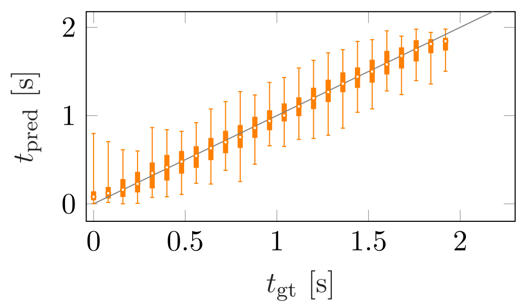
4.5.2 Early Prediction of Roundabout Exits

The roundabout exit prediction can be evaluated with the same performance metrics as the lane change intention prediction. The results in Table 4.12 denote the number of false alarms as well as μ_{trel} for common choice of n_{min}. The histograms in Figure 4.9 show, that the performance of the MoE method in its original design already performs better than the RNN structures. The GRU, as well as the LSTM method both show a low number of missed classifications, which is avoided by using the MoE structure. However, further improvement can be achieved by employing a method denoted as *MoE+GRU*, which combines the interpretable experts with a non-interpretable regression structure using a GRU network as depicted in Table 4.2. Section 5.4 further examines this method and its capabilities to generate validatable classification results without the need of an interpretable regression structure. The MoE+GRU achieves the best performance with respect to μ_{trel}, as well as the number of missed classifications and late classifications as it can be seen on the according histograms.

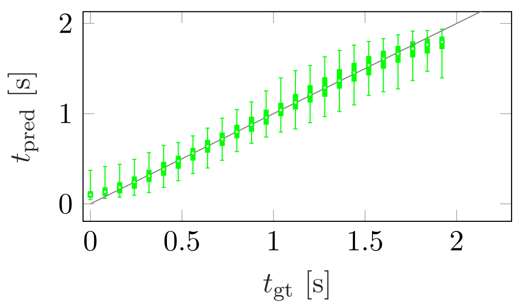
Method	MoE	MoE+GRU	GRU	LSTM	CNN
False Alarms n_{fa}	0	0	0	0	0
n_{min}	11	11	20	20	22
μ_{trel}	1.60	1.64	1.44	1.43	1.38

Table 4.12: Comparison of the MoE structure with reference methods on the roundabout dataset. Results with smoothing factor to achieve $n_{\text{fa}} = 0$.

Figure 4.10 displays the errors of the interpretable TTE regressor using a regression tree structure (a) and the GRU regressor (b) as applied for the recurrent network using GRU cells, as well as the abovementioned MoE+GRU hybrid approach. The



(a) Regression Tree



(b) GRU

Figure 4.10: Box plots of TTLC predictions.

interpretable regression tree shows higher variance in the late stages of the prediction, i. e., between 0 and 1 second TTE. Note, that only the errors that predict the event in a different expert responsibility range have a negative effect on the classification performance metric μ_{trel}.

4.6 Chapter Conclusion

In this chapter, a method to generate an interpretable early prediction has been presented. The structure is designed as expert architecture with a meta-network and several individually and separately trained expert networks. By using DTs as experts, interpretable rule sets can be extracted and presented as automatically generated explanations. Note, that these rule sets may be extracted before inference to generate an intrisically interpretable structure. It could be shown, that the performance of the interpretable method is at least competitive, on some datasets even better than the compared non-interpretable reference methods with respect to early prediction performance. However, the amount of rule sets can grow very large when growing the DTs too large or apply none or only little post-pruning to the trees. The interpretability of the method could therefore be limited by the amount of rule sets that are deemed interpreatble by an assessor with responsibility to validate the considered software function.

5 Constrained Neural Network Architecture for Early Prediction Tasks

This chapter introduces an alternative access to interpretable predictions. The interpretability hereby relies on interpretable templates, as well as constrained weights during the training process of a neural network. It is shown, that each template is separable and interpretable by means of automatically generated rule sets. With the introduction of Explainable Artificial Intelligence (XAI) [43, 44, 80] a growing community of researchers propose a variety of methods and processes that aim to provide understandable explanations for either only the decisions of a ML structure (post-hoc interpretability) or the whole architecture with its inner workings (intrinsic interpretability). The advantage of the latter is the potential robustness to biased data, since an intrinsically interpretable network can be assessed and understood given only the network itself, without any test data. The contributions of this chapter are: (A) the integration of probabilistic predictions into an interpretable framework by introducing an alternative type of interpretability, (B) a significant improvement of early-prediction performance through the introduction of an outlier class, and (C) an extraction and interpretation of the most representative samples describing reoccurring driving scenarios.

5.1 Related work

Intrinsically interpretable methods are among the most promising approaches to validate ML in the future. As described in [81], DTs [15] and linear classifiers are generally intrinsically interpretable. Their structural simplicity, e. g. simple binary splitting rules in DTs, allows a straightforward comprehension of the rules, which the classifier is basing its decisions on. It could be shown in Chapter 3, that even with these simple splitting rules, powerful classifiers for time series classification can be learned, provided that a strong feature generation algorithm is applied. Interpretability can also be claimed for distance based similarity measures such as the GRBF [82]. Here, a set of prototypes are determined through clustering techniques to represent the most distinctive samples within a given dataset. Given that a prototype represents an existing sample from the dataset and the dataset itself is interpretable, it can be concluded that the prototypes are interpretable as well. Medoid-clustering algorithms[83, 84] fulfill this property and choose existing data as cluster centers. If the density of the clusters in n-dimensional space varies, i. e., one cluster has a high density while the data points of another cluster are more spread out, the Euclidean distance is not a suitable distance measure for clustering. A recent publication [20], proposes a proximity measure based on common decision paths in tree ensembles to solve this issue. The so-called *path proximity* is data-adaptive and can dynamically capture clusters of varying density within the same space.

DTs provide confidence measures usually by evaluating the purity of the leaves. Alternatively, ensemble methods can be employed to deliver decent confidence estimates, at the cost of losing interpretability. GRBFs use a similarity measure between a test datum and a set of prototypes, which were defined during training. The result is a classification decision based on distances, which can be transformed into confidence scores. A state-of-the-art confidence estimation for neural networks [85] relies on randomized dropout of single neurons in order to instantiate an ensemble of slightly varying networks. The confidence is derived from the distribution of the activation scores at each output node. This chapter will use this method to estimate the confidences of the reference methods, as they do not yield confidence scores in their respective original publications. Note, that interpreting the softmax outputs of a neural network was shown not to be a good estimate for the prediction confidence in many cases [86].

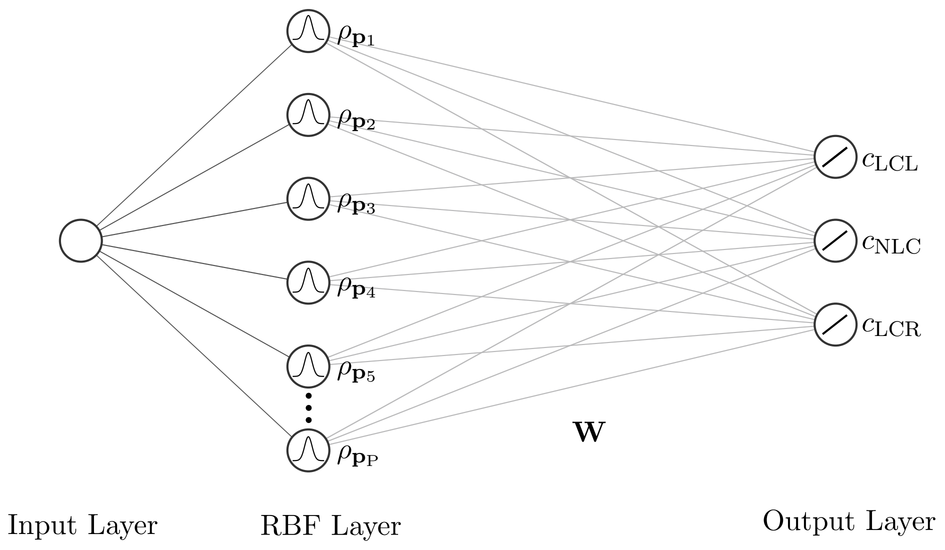


Figure 5.1: GRBF network as probabilistic expert.

5.2 Network Architecture

The RNN reference methods introduced in [3] and [4] in their proposed form yield deterministic outputs. By inserting additional dropout layers that are not only active during training but also during inference, the uncertainty estimation method of [85] is established. This modification of the reference methods is deemed appropriate to ensure proper comparability. All methods are optimized in Section 5.5 to achieve their individual best performance. Note, that these publications did not use interpretable structures and therefore only serve as performance reference. With respect to interpretability, the results from the deterministic baseline method as presented in Chapter 4 can be used as comparison reference.

As in the previous chapter, this approach for an interpretable early prediction on MTS will be described and demonstrated by means of an exemplaric use case, the TTLC prediction. The proposed method is thereby generalizing, as shown by the experiments on a second application. While Chapter 4 applied DT structures as experts, this chapter proposes to use GRBF networks as expert classifiers, with each of them being responsible for disjoint TTLC intervals. A single expert is designed as depicted in Figure 5.1. A GRBF network thereby consists of a GRBF layer with multiple nodes. The output of each node is a similarity measure $\rho_{\mathbf{p}_\mathrm{cl}}$ between a test input and the cluster representative

$\mathbf{p}_{\mathrm{cl}}$, indicating a potential membership. A linear weight matrix $\mathbf{W}$ combines those similarity values to one of the K output nodes representing the classes confidences c_k. With additional constraints for this weight matrix, interpretability can be retained throughout these steps.

As a first step in order to train such networks, the clusters underlying the RBFs have to be determined. The training set is therefore split into K subsets, each containing only samples of one label. With clustering performed for each label separately, it is ensured that a cluster membership can directly be interpreted as an indicator for the corresponding class. Furthermore, each cluster representative is a sample from the training set and therefore directly linked to a real world driving scenario.

The clustering method requires a data-adaptive distance metric to be able to capture clusters with varying densities. The path proximity [20] is a similarity measure based on the RF algorithm which is used to generate a distance matrix between all data points. Let T_b be a single tree of a RF with B trees. The path of an input sample through the tree can be denoted as a set $\mathcal{T}_{b,m}$ of nodes that have been passed by the sample, with b denoting the tree index and m the data point. The Jaccard index yields the similarity between two samples m, m' within the same tree by evaluating the commonly traveled path in relation to their separately traveled path

$$J_b(m, m') = \frac{\mathcal{T}_{b,m} \cap \mathcal{T}_{b,m'}}{\mathcal{T}_{b,m} \cup \mathcal{T}_{b,m'}}. \tag{5.1}$$

Averaged over the B trees in the RF results in the path proximity

$$PP(m, m') = \frac{1}{B} \sum_{b=1}^{B} \frac{\mathcal{T}_{b,m} \cap \mathcal{T}_{b,m'}}{\mathcal{T}_{b,m} \cup \mathcal{T}_{b,m'}}. \tag{5.2}$$

It holds, that $PP(m, m') \in (0, 1]$, since all data points share at least the root node of each tree. When two samples end up in the same leaf of every tree in the RF, $PP(m, m') = 1$.

Further following [20], the RF is trained to distinguish between training sample and random noise. The proximity matrix $\boldsymbol{M}_{\mathrm{PP}} \in \mathbb{R}^{M \times M}$ can be generated by calculating the path proximity between all samples of the respective class-specific subsets.

For clustering the samples based on the proximity matrix, k-medoids clustering is the method of choice. The medoids are further referred to as cluster representatives and, since they are interpretable lane change samples, as prototypes of the clustered scenarios.

The Mahalanobis distance (5.3) is a well-suited measure to calculate the distance between an input sample and the predetermined cluster centers, provided an estimate of the covariance matrix for each cluster is available. The Mahalanobis distance is calculated as

$$d(\hat{\mathbf{x}}, \mathbf{p}_{\mathrm{cl}}) = \sqrt{(\hat{\mathbf{x}} - \mathbf{p}_{\mathrm{cl}})^{\intercal} \boldsymbol{\Sigma}_{\mathrm{cl}}^{-1} (\hat{\mathbf{x}} - \mathbf{p}_{\mathrm{cl}})}, \tag{5.3}$$

with $\hat{\mathbf{x}}$ being the input sample, $\mathbf{p}_{\mathrm{cl}}$ a cluster representative and $\boldsymbol{\Sigma}_{\mathrm{cl}}^{-1}$ the inverse sample covariance matrix of a cluster. Note, that due to the feature generation process introduced in Chapter 3, the input is reduced from the MTS sample $\mathbf{X}$ to a vector of important and interpretable features $\hat{\mathbf{x}}$. The Mahalanobis distance is then transformed into a similarity measure by applying a Gaussian kernel

$$\rho_{\mathbf{p}_{\mathrm{cl}}}(\hat{\mathbf{x}}) = e^{-\beta d(\hat{\mathbf{x}}, \mathbf{p}_{\mathrm{cl}})}, \tag{5.4}$$

with β as adjustable parameter, and with $\rho \in]0, 1]$.

The sample covariance matrix $\boldsymbol{\Sigma}_{\mathrm{cl}}$ is calculated for each cluster separately. It is possible that the number of samples within a cluster M_{cl} becomes lower than the number of features, which leads to a bad estimate of the covariance matrix or, in case of constant feature values for all samples in M_{cl}, a singular matrix that is non-invertable. The Oracle Approximating Shrinkage (OAS) estimator [87] is an estimate of the covariance matrix that works well for problems where M_{cl} is lower than the number of features. This chapter uses the OAS estimate of the covariance matrix to ensure an invertable covariance matrix, independent of the cluster sizes.

5.2.1 Calculation of the Covariance Matrix

Since the true covariance matrix $\boldsymbol{\Sigma}$ is often unknown, estimates of the covariance matrix are used instead. The sample covariance matrix $\hat{\mathbf{S}} = [\hat{s}_{jk}]$ with

$$\hat{s}_{jk} = \frac{1}{M_{cl}} \sum_{i=1}^{M_{cl}} (x_{ij} - x_j)(x_{ik} - x_k) \tag{5.5}$$

is an approach to calculating the covariance matrix. If the sample size M_{cl} of the cluster is larger than the number of features F_{cl}, it is a well-suited estimate of the true covariance matrix and the maximum likelihood solution. However, the calculation of the covariance matrix for each cluster requires some more thoughts on the data that is being processed. As an example, constellation-based features to detect lane change maneuvers to the left are often based on objects on the left or on the same lane as EGO. Right lane objects in the feature vector however, are possibly zero or at least constant. Especially for smaller clusters, the probability of all cluster members containing a constant feature element at the same position is high. The resulting variances for such features according to (5.5) with $j = k$ are 0, which render the covariance matrix non-invertable. In addition, the sample covariance $\hat{\mathbf{S}}$ is a bad estimate for clusters where $F_{cl} > M_{cl}$. The OAS estimator claims that a well-conditioned estimate of the covariance matrix can be found in a linear combination of the sample covariance (5.5) and a simplified matrix expressing a mean over all variances

$$\hat{\mathbf{F}} = \frac{\mathrm{Tr}(\hat{\mathbf{S}})}{F_{cl}}\mathbf{I}, \tag{5.6}$$

with $\mathbf{I} \in \mathbb{R}^{F_{cl} \times F_{cl}}$ being the identity matrix and Tr as the trace operator. The OAS estimator is computed as

$$\hat{\mathbf{\Sigma}}_{\mathbf{OAS}} = (1-\alpha)\hat{\mathbf{S}} + \alpha\hat{\mathbf{F}}, \tag{5.7}$$

with the scalar factor

$$\alpha = \min\left(\frac{(1-2/F)\mathrm{Tr}(\hat{\mathbf{S}}) + \mathrm{Tr}^2(\hat{\mathbf{S}})}{(M_{cl}+1-2/F)[\mathrm{Tr}(\hat{\mathbf{S}}) - \mathrm{Tr}^2(\hat{\mathbf{S}})/F]}, 1\right). \tag{5.8}$$

5.2.2 Constrained linear least squares weights

The activations of the GRBF layer nodes represent similarities to label-specific clusters. In order to achieve a decision for one of the classes, a linear weight matrix $\mathbf{W}$ specifies the contribution of each GRBF node towards the classification. To determine this matrix, a solution can be found by solving the unconstrained, closed form linear least squares (LLSQ) problem. Let K be the number of classes, P the number of prototypes and M the number of samples in the training set, this solution can be calculated as

$$\mathbf{W} = \mathbf{L}\mathbf{\Pi}^{\intercal}(\mathbf{\Pi}\mathbf{\Pi}^{\intercal})^{-1}, \tag{5.9}$$

with $\mathbf{L} \in \{0,1\}^{K \times M}$ as the one-hot encoded dataset labels and $\mathbf{\Pi} \in \mathbb{R}^{P \times M}$ denoting the similarity matrix of each dataset sample to each prototype. Since prototypes of different classes may lie close to each other, a possible result of the unconstrained LLSQ solution could be a positive weight between a training sample and a prototype of a different class. A solution for the LLSQ problem with a constraint for each weight w of the weight matrix $\mathbf{W}$ can be defined as

$$w(\mathbf{x}, \mathbf{p}) = \begin{cases} > 0 & \text{if } k_{\mathbf{x}} = k_{\mathbf{p}} \\ \leq 0 & \text{otherwise,} \end{cases} \tag{5.10}$$

with $k_{\mathbf{x}}$ being the class of a training sample $\mathbf{x}$ and $k_{\mathbf{p}}$ the class of a prototype $\mathbf{p}$. This constraint enforces positive weights only between prototypes and output nodes of the same class. From an interpretability point of view it is therefore possible to formulate the statement: "*If the highest value at the output corresponds to class LCL, then this sample is of class LCL, because it looks similar to the prototypes of class LCL and it looks different than the prototypes of class LCR and NLC.*" This type of interpretability has been introduced in [82]. The output of the network can be considered as class confidences. Note, that the sum of these confidences for all classes is approximately 1 for samples that are close to the clustered training data, but can also be considerably smaller for unseen data. This effect is desired in order to define a confidence-based unknown detector. Note, that the sum of the class confidences correspond to the non-zero element of the one-hot encoded matrix $\mathbf{L}$ and must not be mistaken as a probability estimate.

5.2.3 Confidence-based sample rejection

In the proposed structure, the TTLC regression network assigns one expert to perform the lane change classification. An error in this assignment provides an expert classifier with data it has not seen during training, leading to potentially incorrect results. It is therefore desirable to let the classifier estimate its own confidence during the prediction, and reject uncertain classifications. The concept of outlier detection by means of radial basis functions is well known as an integral part of the 1-class SVM [88]. The interpretability sample rejection from distance measures has been elaborated by [89]. Outlier detection eventually evolved into open set recognition [90], and has been transferred to deep networks in [91], where activation patterns in the penultimate layer are leveraged to detect unknown classes. These activation patterns however do not represent interpretable entities, in contrast to the scenario prototypes used in this work.

The output nodes c_k of the classification structure represent linear combinations of similarity measures that can be viewed as confidence scores. As it is elaborated in Section 5.2.2, these confidences must not always sum up to 1. A sample that is not similar to any of the provided clusters can have small confidence values for all classes and can therefore easily be rejected by thresholding the minimum required value of the class confidences. The further implementation of the rejection mechanism is realized by adding an *unknown (UKN)* class to the network. This pseudo-class can replace any of the original classes and indicates, that the expert is not capable of making an informed decision based on the underlying prototypes.

A special focus should be put on the advantages of the MoE structure in this architecture. By setting individual thresholds for the different experts, a designated rejection behavior per estimated TTLC can be defined. This is desirable, since not all samples can be classified multiple seconds ahead of the lane change. Spontaneous or reactive maneuvers rightfully look like straight driving for the early experts. An expert with a high rejection threshold only classifies those samples that have unambiguous features and rejects the rest as *UKN* class. In contrast, by setting the confidence threshold of an expert to 0, it will classify every sample as one of the three initial classes and does not yield an *UKN* classification at all. Especially for the late experts right before a lane change this behavior is necessary in order to maximize prediction times and avoid false-negatives, i. e., scenarios that are not classified until the point of lane change.

For the MoE architecture, the thresholds are represented as elements of vector $\boldsymbol{\gamma} \in \mathbb{R}^e$ with e being the number of experts. Finding a solution that maximizes the mean reliable prediction time $\mu_{\mathrm{t,rel}}$ by adjusting the thresholds, as well as the smoothing factor n_{min}, can be written as an optimization

$$\mu_{\mathrm{t,rel}} = \arg \max_{\boldsymbol{\gamma}, n_{\mathrm{min}}} f(\mathbf{X}; \boldsymbol{\gamma}, n_{\mathrm{min}}) \text{ s.t. } n_{\mathrm{fa}} \leq n_{\mathrm{fa,tol}}, \tag{5.11}$$

with $f(\mathbf{X}; \boldsymbol{\gamma}, n_{\mathrm{min}})$ being a function that iterates over all samples of the training set to calculate the mean reliable prediction time in dependence of the optimization parameters $\boldsymbol{\gamma}$ and n_{min}. Further, n_{fa} denotes the number of false alarms and $n_{\mathrm{fa,tol}}$ the maximum tolerated number of false alarms.

5.3 Method Interpretability

The proposed approach offers interpretability through similarities between test samples and automatically determined representations of reference scenarios. These scenario representations are the cluster centers of a clustering process in the information-enriched and interpretable dataset that was generated in accordance with the methodology introduced in Chapter 3. The similarity measure uses all available features which, depending on the number of features, make it difficult for a human observer to verify the cluster representatives as relevant scenarios. This section therefore proposes to generate unique rule sets in order to assess the suitability of the representatives.

A DT is trained on the enriched and interpretable dataset to predict the class labels. After growing the tree, the predetermined cluster representatives are provided as inputs. The resulting decision paths are extracted to generate rule sets for each representative. An assessor has to verify the following conditions:

1. The rule sets have to be understandable and relatable to the according class. An excessive rule set will become difficult to comprehend.
2. Each generated rule set has to be unique from the other extracted sets. This is ensured by requiring each cluster representative to end up in an individual leaf of the DT. Subsequently, at least one difference can be determined in a direct comparison between any two representatives, as displayed in Table 5.1.
3. The class label of the representatives have to match the label of the leaves where they end up when they are propagated through the DT.

The goal of each rule set is to single out the representative scenario from the other cluster centers. Table 5.1 shows an exemplary rule set for a right lane change representative in expert E_5. The columns contain the description of the features, the split conditions in real world units, as well as the number of cluster centers that have been separated from the representative at each decision. For the given example of class *LCR* and 15 cluster representatives per class, the successful separation is indicated by a value of 14/15/15 in the table. Since the feature generation method from Chapter 3 yields multiple filtered representations of semantically identical features, the FIR filter number is listed to further differentiate the features.

From this table, an understanding for the underlying lane change can be extracted. It shows situations where the leading vehicle is slightly faster (Table 5.1 No. 2) but very close to EGO (Table 5.1 No. 4), indicating a previous cut-in situation of EGO to overtake another vehicle. The object to be overtaken on the right lane is coming closer

No.	FIR	Feature	Rule	Separated Reps. LCR/NLC/LCL
1	11	Min $d_{\text{left,front}}$	<10.61m	8/6/0
2	6	Grad d_{front}	≥0.16m/s	9/7/12
3	2	$d_{\text{right,front}}$	≥1.21m	9/7/12
4	6	d_{front}	<13m	13/15/15
5	6	Grad $d_{\text{right,front}}$	<-0.22m/s	13/15/15
6	2	$d_{\text{right,rear}}$	≥1.26m	13/15/15
7	2	Min $d_{\text{right,front}}$	≥4.74m	13/15/15
8	2	Grad d_{front}	≥0.24m/s	14/15/15

Table 5.1: Single rule set for an E_5 *LCR* representative.

as it has a negative gradient (Table 5.1 No. 5), which is essential for an overtaking maneuver. From this scene description, the right lane change after the overtaking maneuver can be anticipated by human assessors, deeming this representative scenario interpretable and meaningful. Figure 5.2 shows a multivariate histogram for this *LCR* example. The figure puts the splitting rules in context with the data distribution to further clarify the scenario interpretation. On the x-axis, the discrete features are listed. The y-axis denotes normalized representations of the feature values. The color intensity denotes the occurrence of values for the respective features within the training set. Darker colors thereby represent a high density. In addition to this histogram, the splits of the rule set corresponding to Table 5.1 are marked (green). Additionally, the values of the representative are shown in the histogram (blue). Once a representative is deemed suitable by fulfilling the three abovementioned requirements for its rule set, a similarity to this representative becomes meaningful, even if the similarity calculation itself is performed in higher dimensionality. Note, that the DT and the RBF method share the same input space, therefore all features from the rule set are also features in the similarity calculation. The representatives are class-specific, and the constrained linear weights yield their unambiguous contributions towards the class labels as was explained in Section 5.2.2. Therefore, the class decision for any given test sample can be attributed to multiple interpretable representatives, with their weighted similarity scores speaking cumulatively for the resulting class. However, the rejection mechanism, as

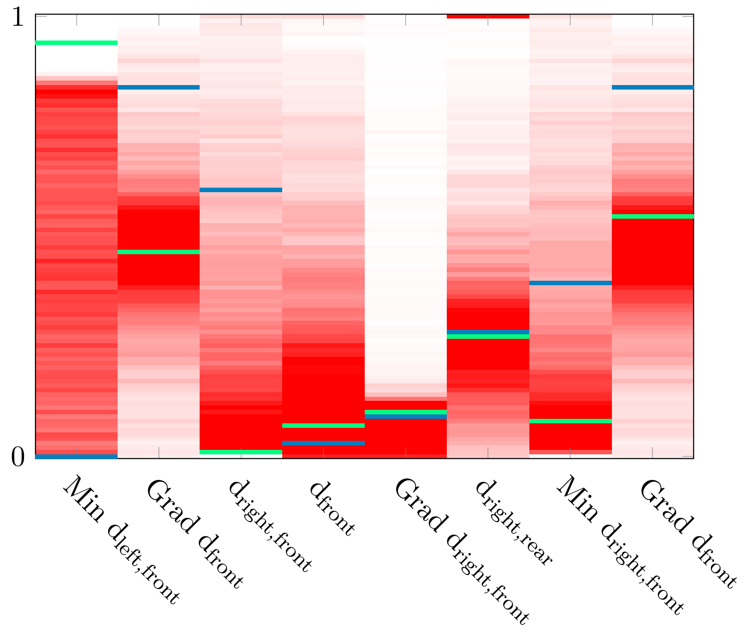


Figure 5.2: Graphic interpretation of an exemplary *LCR* rule set.

discussed in Section 5.2.3, defines thresholds for these resulting class scores for scenarios in which all similarity scores are low.

5.4 Interpretability of the Regressor

It is possible, as implemented in the experiments of Section 5.5, to grow a regression tree based on interpretable features analogous to the DT expert structures from Chapter 4. As a result, the regression becomes an interpretable process that can be expressed through rule sets. The performance of the interpretable regressor for the application of lane change prediction is examined in Section 4.5. For some applications, however, it might occur that the performance of the interpretable regressor is not sufficient. When a significant drop in performance between the DNN implementation and the interpretable regression tree is noticed, other ways of generating a validatable regression result have to be implemented.

Figure 5.3 depicts a modification from the MoE architecture that facilitates the use of non-interpretable regression structures. The figure shows a layout that evaluates each expert simulatenously, thereby requiring the classification prediction y_{cl}, the confidence of the prediction c_{cl}^{y} as well as the constant responsibility range $[y_{\mathrm{val,min}}, y_{\mathrm{val,max}}]$ of each expert to perform a final evaluation as described below. The structure is able to issue class predictions y_{cl}, a TTE regression result y_{reg} as well as a validated regression result $y_{\mathrm{reg,val}}$, which can potentially be empty if the prediction cannot be validated at the given point in time.

Each classification expert is trained with samples within a specific time range. In the presented GRBF method, the selected cluster centers reflect typical representatives for events from within the individual expert's range. Subsequently, a similarity comparison of an incoming sample to those templates not only contains information on the classification result, but also on the according time range it is compared to. If high similarities are determined between an expert and the input sample, it can be assumed that the incoming datum is from within the time range of the expert - independently of the potentially non-interpretable regression. If the classification expert, which is determined by the regressor, therefore emits a high confidence prediction, a non-interpretable regression result may be used within the constraints of the interpretable and validatable time range of the expert. This allows one to use the predicted TTE for low-risk sub-functions, while the range limits of the expert can be used for safety critical processes. Depending on the application, the most conservative choice can be the upper range limit, e. g, for predicting when a pedestrian finished crossing the road, or the lower range limit, e. g., to predict a time to collision. Due to high dimensional proximities between templates of different experts it is possible that multiple experts confidently classify an input sample. In order to determine if multiple confident experts are present, all expert predictions have to be performed simultaneously. This step is effectively loosening the MoE structure, as it would be now possible to continue with the time range of the most confident expert and ignore the predicted TTE. However, the single classification experts have not been trained with samples from the full time range, which makes the TTE regression significantly more accurate. A validatable output will therefore only be generated, if the regression result is within the responsiblity range of the confident expert. Figure 5.4 elaborates some of the possible outcomes. In this lane change prediction example, a non-interpretable regressor predicts a TTLC value of 0.9 s. In a), expert E_1 shows a high confidence, while the other experts have low confidence values. The result is a TTLC of 0.9 s, within a validated range between 0.5 s and 1.5 s, corresponding to the training range of this expert. Considering applications where short reaction times are crucial,

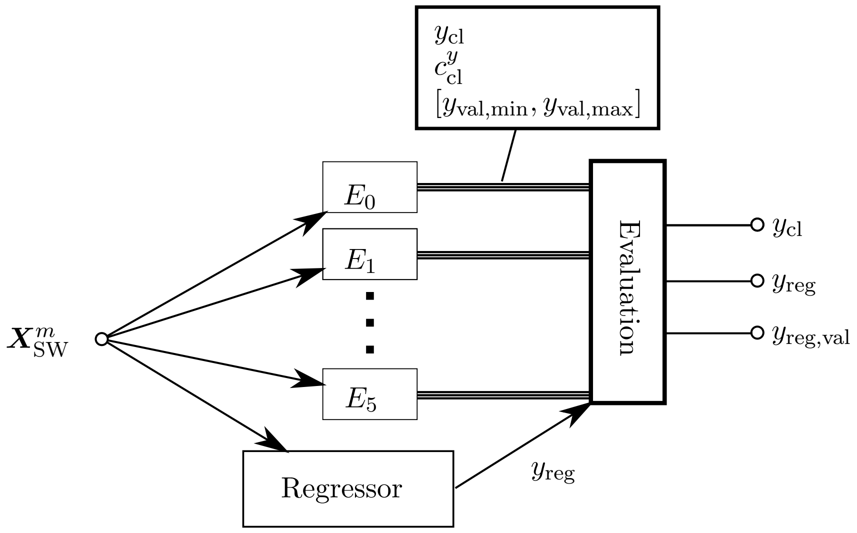


Figure 5.3: Parallel expert structure for the validation of time ranges.

the most conservative estimate would therefore be 0.5 s. In b), two adjacent experts show high confidences. This is especially applicable, when the ground truth TTLC is at the border of both expert time ranges. In this case, it is reasonable to increase the validated range to both experts, as long as the predicted TTLC is within this range. In c), non-adjacent experts shows high confidence scores. While it can be argued, that the predicted TTLC and one of the confident experts match, an unambiguous validated range can not be determined for this scenario. In this case, the prediction should be rejected for the benefit of retaining a validatable structure.

Several other combinations can occur within the proposed regressor validation setup. However, the main cases that allow for a validated range are covered in Figure 5.4 a) and b).

For resource-sensitive target platforms, it might be disadvantageous to run all experts simultaneously. In this case, the solution of using an interpretable regressor should be preferred.

5.5 Experimental Results

The datasets were split into training-, validation- and test sets with balanced classes. The proposed MoE architecture and the reference networks were trained on the same dataset

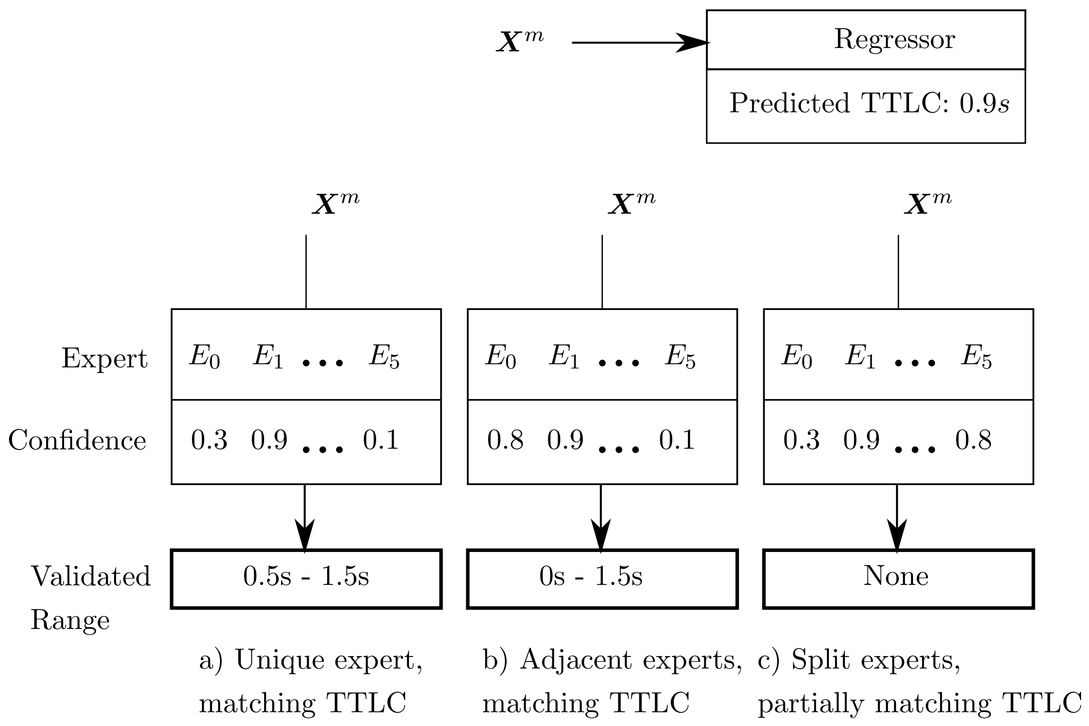


Figure 5.4: TTLC range validation for non-interpretable regressors.

to establish proper comparability. For all methods, a selection of viable parameters are made and assessed by means of several performance parameters.

1. The number of false alarms n_{fa} as introduced in Section 4.5.
2. The mean reliable prediction time $\mu_{t,\text{rel}}$ as introduced in Section 4.5.
3. Number of late classifications n_{late}. A late classification is defined as $t_{\text{rel}} \leq 1\text{s}$.
4. Earliest time of classification $t_{\text{rel,max}}$. The highest value of t_{rel} in the dataset and the earliest point in time, in which lane changes were classified correctly without being reclassified later. The upper bound of this value is hard limited by the smoothing factor n_{min}, its lower bound is determined by the performance of the classifier.

5.5.1 GRBF Experts on the Lane Change Intention Prediction Dataset

First, the results of the method on the lane change intention prediction dataset are examined. The parameter optimization is performed on the validation set. In order to find well-performing values for the vector $\boldsymbol{\gamma}$ and n_{min}, the optimization problem (5.11) is solved using a generalized pattern search algorithm with adaptive mesh size [92] to maximize the mean reliable prediction time. For this application, a tolerance of 1% false alarms of the *NLC* labeled samples of the validation set, $n_{\text{fa,tol}} = 5$ is accepted. Table 5.2 lists three resulting parameter combinations. The first row shows the parameter set for the highest value of $\mu_{t,\text{rel}}$. The high value of n_{min} thereby reduces the maximum possible prediction time compared to the other parameter sets. When setting upper bounds for n_{min}, other parameter combinations can be found whose mean reliable prediction times are lower, but whose values of $t_{\text{rel, max}}$ are increased. The most suitable parameter set depends on the application case. Since none of the options contain missed classifications, i. e. samples, where $t_{\text{rel}} = 0$ s, all parameter sets are viable. For generating results on the test set, the second parameter set with $n_{\text{min}} = 6$ is used further.

n_{min}	$\boldsymbol{\gamma}$	n_{fa}	$\mu_{t,\text{rel}}$	$t_{\text{rel,max}}$	n_{late}
12	[0 0 0 0 0 0 0.72]	5	3.85s	5.08s	2
6	[0 0 0 0 0 0.58 0.86]	5	3.83s	5.32s	9
3	[0 0 0 0 0.64 0.88 0.91]	3	3.62s	5.44s	11

Table 5.2: Viable GRBF parameter choices. The values are based on the validation set.

For the reference methods, only two parameters, the smoothing factor n_{min} and the confidence threshold γ as a scalar, have to be determined. It is therefore possible to visualize all parameter combinations in a surface plot, with $\mu_{t,\text{rel}}$ on the z-axis and n_{fa} as color coding. Figure 5.5 visualizes this surface for the probabilistic implementation of the GRU network [4]. The figure shows, that only with a combination of a high confidence threshold and high values of n_{min}, the constraint $n_{\text{fa}} \leq n_{\text{fa,tol}}$ is reachable. For each of the reference methods, the parameter set with the highest value of $\mu_{t,\text{rel}}$ within the constraint for n_{fa} is extracted for use on the experiments on the test set.

Table 5.3 shows the results on the test set for the proposed MoE method with GRBFs (MoE, GRBF), the baseline method using DT experts (MoE, DT), as well as the probabilistic implementation of the reference methods with the rejection mechanism in place. The reference structures as originally proposed were not able to fulfill the requirement set by $n_{\text{fa, tol}}$ with acceptable results for $\mu_{t,\text{rel}}$ and are therefore not listed.

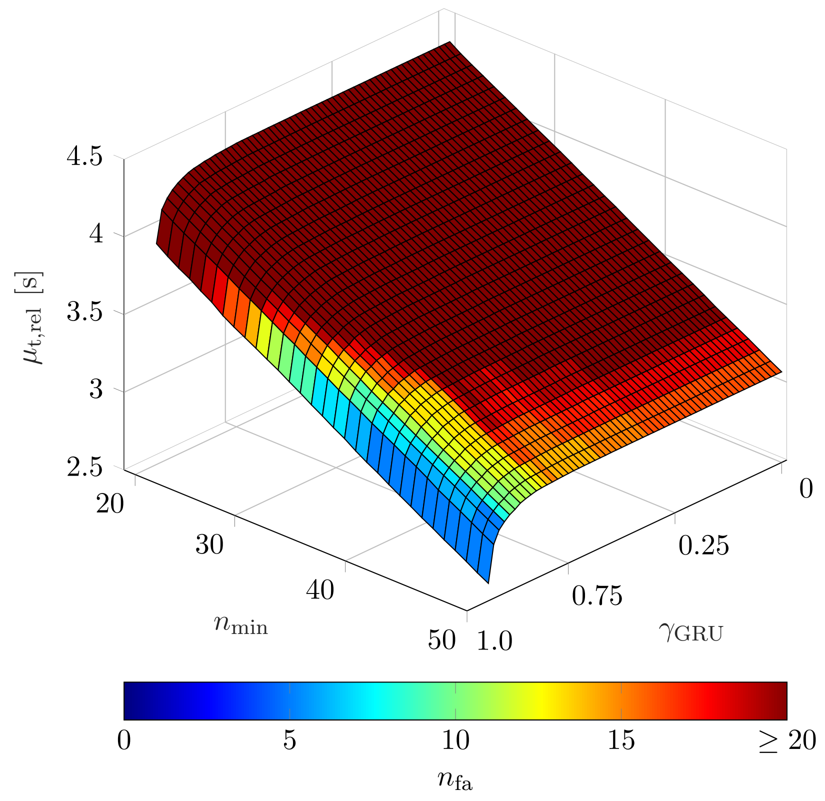


Figure 5.5: Surface plot of parameter combinations with probabilistic GRU network.

	n_{fa}	$\mu_{t,\text{rel}}$	$t_{\text{rel,max}}$	n_{late}
MoE, DT	1	3.44s	4.4s	4
GRU [4], probabilistic	0	3.06s	4.04s	8
LSTM [3], probabilistic	2	2.85s	4.16s	27
MoE, GRBF	0	3.78s	5.32s	1

Table 5.3: Comparison of the MoE structure with the reference methods on the lane change dataset.

It can be seen, that the MoE structure with GRBFs outperforms the RNN reference methods in their early prediction ability. Compared to the DT experts without confidences as presented in Chapter 4, not only the mean reliable prediction time, but also the earliest possible prediction time $t_{\text{rel,max}}$ is greatly increased. The recurrent networks used as reference methods have only been modified to provide confidence scores, otherwise they are left as described in Section 4.2.

5.5.2 GRBF Experts on the Roundabout Exit Prediction Dataset

The roundabout exit prediction dataset already shows convincing performance indicators with the DT experts in the MoE architecture. The introduction of GRBF experts is therefore particularly motivated by the increased interpretability through a lower number of rule sets. For the results on this dataset as displayed in Table 5.4, the threshold for late classifications is set to 0.5s, which, in accordance to the expert ranges, corresponds to the latest expert E_0.

	n_{fa}	$\mu_{t,\text{rel}}$	$t_{\text{rel,max}}$	n_{late}
MoE, DT	0	1.60s	1.78s	0
GRU [4], probabilistic	0	1.54s	1.66s	3
LSTM [3], probabilistic	0	1.46s	1.60s	4
MoE, GRBF	0	1.42s	1.58s	0

Table 5.4: Comparison of the MoE structure with the reference methods on the roundabout dataset.

The incorporation of confidence measures affects the reference methods positively. The metrics $\mu_{t,\text{rel}}$ and $t_{\text{rel,max}}$ increase due to the lowered smoothing factors n_{min}. The GRBF method however, displays worse early prediction behavior than by using the DT experts. An explanation for this behaviour can be the determination of inferior representatives, a hypothesis that can be tested by the three rule set paradigms that have been explained in Section 5.3. Since no violations of these paradigms could be identified, a further explanation could lie in the distance measure itself. Features that are not decision relevant in the DT have an influence on the similarity score calculated by the GRBF nodes, leading to a worse early prediction performance than by using the DT architecture. When deciding between the interpretable methods, a choice between slightly better prediction performance with the DT MoE structure and easier interpretability through a lower number of interpretable rule sets with the GRBF method has to be made.

5.6 Chapter Conclusion

In this chapter, an architecture that picks up the structural concepts from Chapter 4 and adds confidence measures has been presented. The proposed method thereby consists of a feature generation step, an MoE structure, and the proposed GRBF expert networks with interpretable template similarities. Within the application of lane change prediction it could be shown, that comprehensible rule sets can be utilized to separate representative scenarios from each other. It could further be shown, that by introducing confidence scores into the task of early prediction, unambiguous lane change events can be detected reliably multiple seconds in advance.

With the goal of validation in mind, this structure enables a human assessor to examine the interpretable input set, as well as the rule sets of the representatives without processing test samples. The introduction of constrained weights into the network allows an interpretable continuation of the similarity measures onto the resulting class confidences. The structure as a whole fulfills the property of intrinsic interpretability and is therefore suitable for further validation.

6

Conclusions

The work *Validation of Machine Learning Algorithms by Design with Applications for Automated Driving* contributes towards the validation of ML algorithms. The term *by Design* thereby denotes the intrinsic eligibility for validation of functions that apply methods of ML, provided that certain structural constraints are followed. The focus of the developed methods is the early prediction of multivariate time series in TTE prediction tasks.

6.1 Complementary Validation by Diversity

Next to taking advantage of intrinsic properties of networks, a validation through a combination of several external properties can be achieved. For all aspects of an ML architecture, let it be the input dataset, the features at different representation states or the model itself, self contained validation methods are designed. A method to validate the features in a given dataset is presented in [19]. First, a series of convolutional layers is applied to extract latent features in high dimenional space. The features are then undergoing a dimensionality projection method to be visualized in 2D space. By comparing clusters in this resulting reduced latent space with the spatio-temporal labels of the underlying dataset it can be assessed, if the chosen architecture manages to extract well-separated, and by that, distinguishable features. Note, that the multi-layer convolutions as well as the feature projections are uninterpretable processes. However, the methods designed for a validation by diversity do not claim the intrinsic interpretability that is targeted in *Validation by Design.* As an example for interpretable methods that can be used within the diversity framework, a lane change classification by means of deep autoencoders is proposed in [93]. Here, a separate autoencoder is trained for each class in the dataset. By comparing the reconstruction error of each output it is possible to determine a similarity between the training samples of the corresponding class-specific autoencoder. Since diversity validation methods are designed as self-contained modules, they can be treated as complementary additions to the exisiting architectures of the *Validation by Design* approach.

6.2 Conclusion

The presented work utilizes DNN structures to generate powerful classifiers and regressors. By reinterpretation of DNN subcomponents and analysis methods such as 1D-convolutions and heatmapping techniques, an enriched dataset for MTS is introduced. The interpretability of this enriched dataset is concluded by eliminating deep computing structures from the inference process and emphasizing the filter-like properties of the established layer-1 space.

Following the divide-et-impera paradigm, an MoE architecture consisting of expert classification networks and a meta network with double-assignment as TTE regressor is set up. The target of early prediction performance optimization is achieved by a segregaded assignment of time intervals as responsibility ranges of the experts. By ensuring an interpretable meta network, as well as interpretable experts, the interpretability of the structure as a whole is argued.

Any intrinsically interpretable classification structure can be applied as an expert. With the choice of DT experts, easily extractable rule sets can be generated. This work presents that GRBF networks with constrained weight layers may posess intrinsically interpretable properties as well. The validatability is established by a two-fold argumentation. The advantage of applying GRBF experts is a reduced list of condensed rule sets, facilitating a quick and comprehensible evaluation of the reliability of the network.

With respect to the components of a strong safety argument that are identified in Section 2.3, the approach fully covers the requirements of algorithmic interpretability, as well as uncertainty quantification. As mentioned above, additional requirements for the safety argument can be fulfilled by incorporating diversity methods in the ML environment.

References

[1] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, “Imagenet large scale visual recognition challenge,” 2014. [Online]. Available: http://arxiv.org/pdf/1409.0575v3 ix, 34, 35

[2] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” 2014. [Online]. Available: http://arxiv.org/pdf/1409.1556v6 ix, 35

[3] H. Q. Dang, J. Furnkranz, A. Biedermann, and M. Hoepfl, “Time-to-lane-change prediction with deep learning,” in *IEEE ITSC 2017*, I. I. T. S. Conference, Ed. Piscataway, NJ: IEEE, 2017, pp. 1–7. xi, 59, 75, 89

[4] Z. Yan, K. Yang, Z. Wang, B. Yang, T. Kaizuka, and K. Nakano, “Time to lane change and completion prediction based on gated recurrent unit network,” in *2019 IEEE Intelligent Vehicles Symposium (IV)*. [Piscataway, New Jersey]: IEEE, 2019, pp. 102–107. xi, 59, 60, 75, 87, 89

[5] P. Wang and L. Zong, “Are crises predictable? a review of the early warning systems in currency and stock markets,” 2020. [Online]. Available: http://arxiv.org/pdf/2010.10132v1 1

[6] V. Chaurasia and S. Pal, “Early prediction of heart diseases using data mining techniques,” *Caribbean Journal of Science and Technology*, vol. 1, pp. 208–217, 2013. 1

[7] S. Mozaffari, E. Arnold, M. Dianati, and S. Fallah, “Early lane change prediction for automated driving systems using multi-task attention-based convolutional neural networks,” 2022. [Online]. Available: https://arxiv.org/pdf/2109.10742 1

[8] L. Ye and E. Keogh, “Time series shapelets,” in *KDD’09*, J. Elder, F. S. Fogelman, P. Flach, and M. Zaki, Eds. New York, NY: ACM, 2009, p. 947. 1, 58

[9] Y.-F. Lin, H.-H. Chen, V. S. Tseng, and J. Pei, “Reliable early classification on multivariate time series with numerical and categorical attributes,” in *Advances in knowledge discovery and data mining*, ser. Lecture notes in computer science Lecture notes in artificial intelligence, T. H. Cao, E.-P. Lim, Z.-H. Zhou, T.-B. Ho, D. Cheung, and H. Motoda, Eds. Cham: Springer, 2015, pp. 199–211. 1, 58

[10] G. He, Y. Duan, R. Peng, X. Jing, T. Qian, and L. Wang, “Early classification on multivariate time series,” *Neurocomputing*, vol. 149, pp. 777–787, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S092523121401008X 1, 58

[11] Mohamed F Ghalwash and Zoran Obradovic, “Early classification of multivariate temporal observations by extraction of interpretable shapelets,” *BMC Bioinformatics*, vol. 13, no. 1, pp. 1–12, 2012. 2, 58

[12] Z. Xing, J. Pei, P. S. Yu, and K. Wang, “Extracting interpretable features for early classification on time series,” in *Proceedings of the 2011 SIAM International Conference on Data Mining*, B. Liu, H. Liu, C. W. Clifton, T. Washio, and C. Kamath, Eds. [Philadelphia, Pennsylvania]: [Society for Industrial and Applied Mathematics], 2011, pp. 247–258. 2, 58

[13] M. Bojarski, P. Yeres, A. Choromanska, K. Choromanski, B. Firner, L. Jackel, and U. Muller, “Explaining how a deep neural network trained with end-to-end learning steers a car,” 2017. [Online]. Available: http://arxiv.org/pdf/1704.07911v1 2

[14] V. N. Vapnik, *The Nature of Statistical Learning Theory*, 2nd ed., ser. Statistics for Engineering and Information Science. New York, NY: Springer New York and Imprint and Springer, 2000. [Online]. Available: https://ebookcentral.proquest.com/lib/kxp/detail.action?docID=3086234 7

[15] L. Breiman, *Classification and regression trees*, ser. The Wadsworth statistics/probability series. Belmont, Calif. and Pacific Grove, Calif. and Pacific Grove, Calif. and Monterey, Calif.: Wadsworth International Group and Wadsworth & Brooks/Cole and Wadsworth & Brooks/Cole Advanced Books & Software, 1984. 7, 51, 74

[16] M.-F. Botsch, “Machine learning techniques for time series classification,” Zugl.: München, Techn. Univ., Diss., 2009, Cuvillier, Göttingen, 2009. 10

[17] P. Geurts, D. Ernst, and L. Wehenkel, “Extremely randomized trees,” *Machine Learning*, vol. 63, no. 1, pp. 3–42, 2006. 10

[18] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0893608089900208 13

[19] O. de Candido, M. Koller, O. Gallitz, R. Melz, M. Botsch, and W. Utschick, “Towards feature validation in time to lane change classification using deep neural networks,” in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 9/20/2020 - 9/23/2020, pp. 1–8. 91

[20] F. Kruber, J. Wurst, E. S. Morales, S. Chakraborty, and M. Botsch, “Unsupervised and supervised learning with the random forest algorithm for traffic scenario clustering and classification,” in *2019 IEEE Intelligent Vehicles Symposium (IV)*. [Piscataway, New Jersey]: IEEE, 2019, pp. 2463–2470. 17, 74, 76

[21] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997. 20

REFERENCES

[22] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder–decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Q. C. R. I. Alessandro Moschitti, G. Bo Pang, and U. o. A. Walter Daelemans, Eds. Stroudsburg, PA, USA: Association for Computational Linguistics, 2014, pp. 1724–1734. 20

[23] R. Krajewski, J. Bock, L. Kloeker, and L. Eckstein, "The highd dataset: A drone dataset of naturalistic vehicle trajectories on german highways for validation of highly automated driving systems," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, 2018, pp. 2118–2125. 21

[24] F. Kruber, E. S. Morales, S. Chakraborty, and M. Botsch, "Vehicle position estimation with aerial imagery from unmanned aerial vehicles," in *2020 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 10/19/2020 - 11/13/2020, pp. 2089–2096. 24

[25] International Organization for Standardization, "Iso 9000:2015 quality management systems — fundamentals and vocabulary: Standard," Geneva, CH. [Online]. Available: https://www.iso.org/standard/45481.html 26

[26] ——, "Iso 26262-1:2018 road vehicles — functional safety: Standard," Geneva, CH. [Online]. Available: https://www.iso.org/standard/45481.html 26

[27] R. Salay, R. Queiroz, and K. Czarnecki, "An analysis of iso 26262: Using machine learning safely in automotive software," 2017. [Online]. Available: http://arxiv.org/pdf/1709.02435v1 27, 32

[28] K. Forsberg and H. Mooz, "The relationship of systems engineering to the project cycle," *Engineering Management Journal; EMJ*, vol. 4, no. 3, pp. 36–43, 1992. [Online]. Available: https://www.researchgate.net/publication/276111848_The_Relationship_of_System_Engineering_to_the_Project_Cycle 27

[29] International Organization for Standardization, "Iso 21448-2019 road vehicles — safety of the intended functionality: Standard," Geneva, CH. [Online]. Available: https://www.iso.org/standard/70939.html 28

[30] N. E. Fenton and M. Neil, *Risk assessment and decision analysis with bayesian networks*, 2nd ed. Boca Raton and London and New York: CRC Press, 2019. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&scope=site&db=nlebk&db=nlabk&AN=1875153 29

[31] P. Koopman and M. Wagner, "Toward a framework for highly automated vehicle safety validation," in *SAE Technical Paper Series*, ser. SAE Technical Paper Series. SAE International400 Commonwealth Drive, Warrendale, PA, United States, 2018. 29, 32

[32] D. M. Allen, "The relationship between variable selection and data agumentation and a method for prediction," *Technometrics*, vol. 16, no. 1, p. 125, 1974.

[33] M. Stone, "Cross-validatory choice and assessment of statistical predictions," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 36, no. 2, pp. 111–133, 1974.

[34] M. Lei, J. X. Felix, S. Jiyuan, C. Chunyang, S. Ting, Z. Fuyuan, X. Minhui, L. Bo, L. Li, L. Yang, *et al.*, "Deepgauge: Comprehensive and multi-granularity testing criteria for gauging the robustness of deep learning systems," *arXiv preprint arXiv:1803.07519 [Titel anhand dieser ArXiv-ID in Citavi-Projekt übernehmen]*, 2018. 29

[35] B. P. Miller, L. Fredriksen, and B. So, "An empirical study of the reliability of unix utilities," *Communications of the ACM*, vol. 33, no. 12, pp. 32–44, 1990. 30

[36] S. K. Cha, M. Woo, and D. Brumley, "Program-adaptive mutational fuzzing," in *2015 IEEE Symposium on Security and Privacy*. IEEE, 5/17/2015 - 5/21/2015, pp. 725–741. 30

[37] M. Moradi, B. van Acker, K. Vanherpen, and J. Denil, "Model-implemented hybrid fault injection for simulink (tool demonstrations)," in *Cyber Physical Systems. Model-Based Design*, ser. Lecture Notes in Computer Science, R. Chamberlain, W. Taha, and M. Törngren, Eds. Cham: Springer International Publishing, 2019, vol. 11615, pp. 71–90. 30

[38] J. Voas, "Fault injection for the masses," *Computer*, vol. 30, no. 12, pp. 129–130, 1997. 30

[39] F. Doshi-Velez and B. Kim, "Towards a rigorous science of interpretable machine learning," 2017. [Online]. Available: http://arxiv.org/pdf/1702.08608v2 31

[40] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "On a formal model of safe and scalable self-driving cars," 2017. [Online]. Available: http://arxiv.org/pdf/1708.06374v6 31

[41] N. Kalra and S. M. Paddock, *Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?*, ser. Research report. Santa Monica, Calif: RAND, 2016, vol. RR-1478-RC. 32

[42] Z. Barger, C. G. Frye, D. Liu, Y. Dan, and K. E. Bouchard, "Robust, automated sleep scoring by a compact neural network with distributional shift correction," *PLOS ONE*, vol. 14, no. 12, p. e0224642, 2019. 33

[43] van Lent and Fisher, "An explainable artificial intelligence system for small-unit tactical behavior," in *Proceedings of the 16th Conference on Innovative Applications of Artifical Intelligence*, 2004, pp. 900–907. 33, 73

[44] D. Gunning, "Explainable artificial intelligence (xai)," *Defense Advanced Research Projects Agency (DARPA), nd Web*, 2017. 33, 73

[45] W. Samek, T. Wiegand, and K.-R. Müller, "Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models," 2017. [Online]. Available: http://arxiv.org/pdf/1708.08296v1 33

[46] M. T. Ribeiro, S. Singh, and C. Guestrin, "Model-agnostic interpretability of machine learning," 2016. [Online]. Available: http://arxiv.org/pdf/1606.05386v1 33

[47] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin, "Anchors: High-precision model-agnostic explanations," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018. [Online]. Available: https://ojs.aaai.org/index.php/aaai/article/view/11491 33

[48] R. Konig, U. Johansson, and L. Niklasson, "G-rex: A versatile framework for evolutionary data mining," in *2008 IEEE International Conference on Data Mining Workshops*. IEEE, 2008. 34

[49] F. R. Hampel, E. M. Ronchetti, P. J. Rousseeuw, and W. A. Stahel, *Robust Statistics: The approach based on influence functions.* New York, N.Y: Wiley, 2005. [Online]. Available: https://onlinelibrary.wiley.com/doi/book/10.1002/9781118186435 34

[50] D. Baehrens, T. Schroeter, S. Harmeling, M. Kawanabe, K. Hansen, and K.-R. Müller, "How to explain individual classification decisions," *J. Mach. Learn. Res.*, vol. 11, pp. 1803–1831, 2010. 34

[51] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," 2014. [Online]. Available: http://arxiv.org/pdf/1409.4842v1 34, 42

[52] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek, "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation," *PLoS ONE*, vol. 10, no. 7, 2015. 34, 37, 38, 44, 45

[53] M. Craven and J. W. Shavlik, "Extracting tree-structured representations of trained networks," in *Advances in neural information processing systems*, 1996, pp. 24–30. 35

[54] F. C. Keil, "Explanation and understanding," *Annual review of psychology*, vol. 57, pp. 227–254, 2006. 37

[55] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson, "Understanding neural networks through deep visualization," 2015. [Online]. Available: http://arxiv.org/pdf/1506.06579v1 37

[56] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Computer vision - ECCV 2014*, ser. Lecture Notes in Computer Science, D. Fleet, Ed. Cham: Springer, 2014, vol. 8689, pp. 818–833. 37

[57] D. Kumar, A. Wong, and G. W. Taylor, "Explaining the unexplained: A class-enhanced attentive response (clear) approach to understanding deep neural networks," 2017. [Online]. Available: http://arxiv.org/pdf/1704.04133v2 37

[58] M. Bojarski, A. Choromanska, K. Choromanski, B. Firner, L. Jackel, U. Muller, and K. Zieba, "Visualbackprop: efficient visualization of cnns," 2016. [Online]. Available: http://arxiv.org/pdf/1611.05418v3 37, 47

[59] G. H. John, R. Kohavi, and K. Pfleger, "Irrelevant features and the subset selection problem," in *Machine Learning Proceedings 1994*, W. W. Cohen, Ed. s.l.: Elsevier Reference Monographs, 1994, pp. 121–129. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9781558603356500234 38

[60] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, "Gene selection for cancer classification using support vector machines," *Machine Learning*, vol. 46, no. 1, pp. 389–422, 2002. [Online]. Available: https://doi.org/10.1023/A:1012487302797 38

[61] C. Ambroise and G. J. McLachlan, "Selection bias in gene extraction on the basis of microarray gene-expression data," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 99, no. 10, pp. 6562–6566, 2002. 38

[62] V. Svetnik, A. Liaw, C. Tong, and T. Wang, "Application of breiman's random forest to modeling structure-activity relationships of pharmaceutical molecules," in *Multiple Classifier Systems*, F. Roli, J. Kittler, and T. Windeatt, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 334–343. 38

[63] R. K. De, N. R. Pal, and S. K. Pal, "Feature analysis: Neural network and fuzzy set theoretic approaches," *Pattern Recognition*, vol. 30, no. 10, pp. 1579–1590, 1997. 38

[64] A. Verikas and M. Bacauskiene, "Feature selection with neural networks," *Pattern Recognition Letters*, vol. 23, no. 11, pp. 1323–1335, 2002. 38

[65] H. Yoon, K. Yang, and C. Shahabi, "Feature subset selection and feature ranking for multivariate time series," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 9, pp. 1186–1198, 2005. 38

[66] K. Pearson, "Liii. on lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901. 38

[67] W. J. Krzanowski, "Between-groups comparison of principal components," *Journal of the American Statistical Association*, vol. 74, no. 367, p. 703, 1979. 39

[68] Z. Cui, W. Chen, and Y. Chen, "Multi-scale convolutional neural networks for time series classification," 2016. [Online]. Available: http://arxiv.org/pdf/1603.06995v4 42

[69] S. Lapuschkin, "Opening the machine learning black box with layer-wise relevance propagation," Ph.D. dissertation, Technische Universität Berlin, 2019. 46

[70] G. Montavon, S. Lapuschkin, A. Binder, W. Samek, and K.-R. Müller, "Explaining nonlinear classification decisions with deep taylor decomposition," *Pattern Recognition*, vol. 65, pp. 211–222, 2017. 47

[71] G. Louppe, "Understanding random forests: From theory to practice," 2014. [Online]. Available: http://arxiv.org/pdf/1407.7502v3 51

[72] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts," *Neural computation*, vol. 3, no. 1, pp. 79–87, 1991. 58

[73] Bin Tang, M. I. Heywood, and M. Shepherd, "Input partitioning to mixture of experts," in *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No.02CH37290)*, 2002, pp. 227–232 vol.1. 58

[74] R. Ebrahimpour, E. Kabir, H. Esteky, and M. R. Yousefi, "A mixture of multilayer perceptron experts network for modeling face/nonface recognition in cortical face processing regions," *Intelligent Automation & Soft Computing*, vol. 14, no. 2, pp. 151–162, 2008. 58

REFERENCES

[75] T. Baldacchino, E. J. Cross, K. Worden, and J. Rowson, "Variational bayesian mixture of experts models and sensitivity analysis for nonlinear dynamical systems," *Mechanical Systems and Signal Processing*, vol. 66-67, pp. 178–200, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0888327015002307 58

[76] J. Schlechtriemen, F. Wirthmueller, A. Wedel, G. Breuel, and K.-D. Kuhnert, "When will it change the lane? a probabilistic regression approach for rarely occurring events," in *2015 IEEE Intelligent Vehicles Symposium (IV)*, 2015, pp. 1373–1379. 58

[77] C. Wissing, T. Nattermann, K.-H. Glander, C. Hass, and T. Bertram, "Lane change prediction by combining movement and situation based probabilities," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 3554–3559, 2017. 59

[78] Y. Zheng, Q. Liu, E. Chen, Y. Ge, and J. L. Zhao, "Exploiting multi-channels deep convolutional neural networks for multivariate time series classification," *Frontiers of Computer Science*, vol. 10, no. 1, pp. 96–112, 2016. 60

[79] L. Sadouk, "Cnn approaches for time series classification," in *Time Series Analysis - Data, Methods, and Applications*, C.-K. Ngan, Ed. IntechOpen, 2019. 60

[80] W. Samek, G. Montavon, A. Vedaldi, L. K. Hansen, and K.-R. Müller, *Explainable AI: Interpreting, explaining and visualizing deep learning*, ser. Lecture notes in computer series Lecture notes in artificial intelligence. Cham: Springer, 2019. 73

[81] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, and D. Pedreschi, "A survey of methods for explaining black box models," *ACM Computing Surveys*, vol. 51, no. 5, pp. 1–42, 2019. 74

[82] M. Botsch and J. A. Nossek, "Construction of interpretable radial basis function classifiers based on the random forest kernel," in *IEEE International Joint Conference on Neural Networks, 2008*. Piscataway, NJ: IEEE, 2008, pp. 220–227. 74, 79

[83] L. Kaufman and P. J. Rousseeuw, *Clustering by Means of Medoids*, ser. Delft University of Technology : reports of the Faculty of Technical Mathematics and Informatics. Faculty of Mathematics and Informatics, 1987. [Online]. Available: https://books.google.de/books?id=HK-4GwAACAAJ 74

[84] L. Kaufman and P. J. Rousseeuw, Eds., *Finding groups in data: An introduction to cluster analysis*, ser. Wiley series in probability and mathematical statistics. Hoboken, NJ: Wiley-Interscience, 2005. 74

[85] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ser. ICML'16. JMLR.org, 2016, pp. 1050–1059. 74, 75

[86] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus, "Intriguing properties of neural networks," in *International Conference on Learning Representations*, 2014. 74

[87] Y. Chen, A. Wiesel, Y. C. Eldar, and A. O. Hero, "Shrinkage algorithms for mmse covariance estimation," *IEEE Transactions on Signal Processing*, vol. 58, no. 10, pp. 5016–5029, 2010. 77

[88] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the support of a high-dimensional distribution," *Neural computation*, vol. 13, no. 7, pp. 1443–1471, 2001. 79

[89] E. M. Knorr and R. T. Ng, "Finding intensional knowledge of distance-based outliers," in *Vldb*, vol. 99, 1999, pp. 211–222. 79

[90] W. J. Scheirer, A. de Rezende Rocha, A. Sapkota, and T. E. Boult, "Toward open set recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 7, pp. 1757–1772, 2013. 79

[91] A. Bendale and T. E. Boult, "Towards open set deep networks," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 1563–1572. 79

[92] C. Audet and J. E. Dennis, "Analysis of generalized pattern searches," *SIAM Journal on Optimization*, vol. 13, no. 3, pp. 889–903, 2002. 87

[93] O. de Candido, M. Binder, and W. Utschick, "An interpretable lane change detector algorithm based on deep autoencoder anomaly detection," in *2021 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 7/11/2021 - 7/17/2021, pp. 516–523. 92