Technische Universität München

Towards Feature Validation in Time to Lane Change Classification using Deep Neural Networks

Oliver De Candido, Michael Koller, Oliver Gallitz, Ron Melz, Michael Botsch,
and Wolfgang Utschick

The 23rd IEEE International Conference on Intelligent Transportation Systems.
September 20 – 23, 2020
Virtual Conference

TUM Department of Electrical and Computer Engineering
Professur für Methoden der Signalverarbeitung
Univ.-Prof. Dr.-Ing. Wolfgang Utschick

Towards Feature Validation in Time to Lane Change Classification using Deep Neural Networks

Oliver De Candido[1], Michael Koller[1], Oliver Gallitz[2], Ron Melz[3], Michael Botsch[2], and Wolfgang Utschick[1]

***Abstract*— In this paper, we explore different Convolutional Neural Network (CNN) architectures to extract features in a Time to Lane Change (TTLC) classification problem for highway driving functions. These networks are trained using the HighD dataset, a public dataset of realistic driving on German highways. The investigated CNNs achieve approximately the same test accuracy which, at first glance, seems to suggest that all of the algorithms extract features of equal quality. We argue however that the test accuracy alone is not sufficient to validate the features which the algorithms extract. As a form of validation, we propose a two pronged approach to confirm the quality of the extracted features. In the first stage, we apply a clustering algorithm on the features and investigate how logical the feature clusters are with respect to both an external clustering validation measure and with respect to expert knowledge. In the second stage, we use a state-of-the-art dimensionality reduction technique to visually support the findings of the first stage of validation. In the end, our analysis suggests that the different CNNs, which have approximately equal accuracies, extract features of different quality. This may lead a user to choose one of the CNN architectures over the others.**

I. Introduction and Motivation

As the driving functions which (semi-)autonomous vehicles should handle become more complex, the role of Machine Learning (ML) algorithms as part of the solution becomes apparent. However, ML algorithms lack validation methods, which is a major drawback. For example, the classical V-model for product development, testing and Verification and Validation (V&V) can no longer be easily applied (see, e.g., [1]). This lack of validation becomes concerning when ML algorithms are implemented in safety critical driving functions. An example function could be to predict when other vehicles in a highway scenario are going to change lanes, i.e., the vehicles surrounding the ego vehicle are tracked and the Time to Lane Change (TTLC) should be predicted. In unexpected situations, the automatic responsive action taken by the autonomous vehicle can decrease the comfort or even endanger the safety of its passengers. Recently, methods have been proposed to predict the TTLC using ML algorithms (see, e.g., [2], [3]), however, the problem of V&V was not discussed in those publications. It is more important than ever that researchers focus on methods to validate ML algorithms employed in driving modules. To this end, we propose a method to validate the features extracted by Convolutional Neural Networks (CNNs) which are employed in autonomous driving functions.

The classical approach to time-series classification can be summarized in two stages: first, discriminative features are hand-designed by domain experts; and second, the time-series signals are classified using k Nearest-Neighbors (k-NN) with 1 neighbor, and an appropriate distance measure, e.g., Dynamic Time Warping (DTW). In [4], the authors give an overview of the classical time-series classification methods which employ this two stage process. However, this classical approach leads one to question how the discriminative features should be designed from the time-series data for a given application. Therefore, the modern approach to time-series classification is to use deep learning techniques, e.g., CNNs, to automatically extract appropriate features, and directly classify the time-series signal.

In this paper, we investigate the properties of various CNN architectures to extract features in time-series classification problems. We can then investigate these properties and the data which are used to train the algorithms to validate the ML algorithms. We will refer to this approach of leveraging multiple quality measures of ML algorithms as: *validation by diversity*. The first applications of CNNs architecture were: document classification [5], image classification [6], [7], and more recently time-series classification [8], [9]. The authors of [9] give an overview of the classification methods which leverage the abilities of deep learning to directly classify time-series problems in an end-to-end fashion.

The main advantage of the CNN architecture is—due to the shared weights and translational invariance of the learned kernels—that the features extracted by a CNN can directly be used for classification. For time-series classification, a CNN architecture was first proposed in [10], and tested on the datasets from the UCR archive [11]. In [8], the authors present a different CNN architecture to classify time-series data, and compare their method with classical k-NN with 1 neighbor and Support Vector Machine (SVM) methods on both simulated datasets and UCR archive datasets. The authors of [12] propose a method to first transform the input time-series signals using down-sampling and smoothing, and then running these signals in parallel through a CNN architecture to predict the class of the time-series data; their method was also verified on UCR archive datasets. A novel CNN architecture was proposed in [13], where each individual channel of a multi-variate time-series signal was processed independently, and subsequently combined for the classification task. In [14], a Deep Neural Network (DNN) architecture was proposed to leverage the capacity of the

[1]Department of Electrical and Computer Engineering Technical University of Munich, 80333 Munich, Germany; Email: oliver.de-candido@tum.de.
[2]Faculty of Electrical Engineering and Computer Science, Technische Hochschule Ingolstadt, 85049 Ingolstadt, Germany.
[3]Department I/EF-A3, AUDI AG, 85045 Ingolstadt, Germany.

CNN architecture to generate interpretable features, which can be used to classify the multi-variate time-series signals using Decision Trees (DTs).

In this work, we introduce a method to validate the features extracted by different CNN architectures, and give an intuition about the possible features which can be extracted by these different architectures. To this end, we introduce a general CNN architecture which separates the feature extraction and the classification parts. Once discriminative features are extracted, we validate them by applying a clustering algorithm (e.g., k-Means) and a quality measure (e.g., the rand index [15]). Finally, these features are projected into two dimensions using a modern dimensionality reduction technique, Uniform Manifold Approximation and Projection (UMAP) [16], such that the features can additionally be validated visually.

The paper is structured as follows: in Section II, we define the learning problem of classifying the TTLC, and introduce the real-world dataset we used for training. In Section III, we introduce the general design of the CNN architectures, and give an intuition of the possible features which can be extracted by these methods. In Sections IV and V, we discuss the simulation results and the method to validate the extracted features, respectively. Finally, in Section VI, we conclude our work with a discussion of our results, and possible avenues to extend this work.

II. Problem Formulation

In this section, we will first introduce the real-world driving data we use to train the CNN classifiers, and then formulate the problem of classifying the TTLC.

A. Real-World Data: The HighD Dataset

In this work, we use the real-world driving data recently published in [17]. The authors recorded German highways at six different locations using a drone. From the recordings they extracted certain attributes of each vehicle on the highway, e.g., the x and y coordinates, the velocities, and the accelerations of each vehicle. Moreover, additional attributes were also tracked for each vehicle, e.g., the number of lane changes, the Time Headway (THW) or the Time to Collision (TTC). From the x and y coordinates of each vehicle, we can calculate the distances between all vehicles around it.

We summarize the tracked attributes in a multi-variate time-series signal as

$$\tilde{\boldsymbol{X}} = \big[\boldsymbol{x}[1], \quad \boldsymbol{x}[2], \quad \ldots, \quad \boldsymbol{x}[N]\big] \in \mathbb{R}^{\Gamma \times N}, \qquad (1)$$

where at each time-stamp n we summarize the Γ attribute signals in a vector $\boldsymbol{x}[n] = [x_1[n], x_2[n], \ldots, x_\Gamma[n]]^\mathsf{T} \in \mathbb{R}^\Gamma$. We assume that the vehicle was tracked for at least N time-stamps; this defines a scenario. As input attributes we consider the attributes in Table I.

Attribute	Description
$v_{\text{lat.}}$	Lateral velocity
$v_{\text{long.}}$	Longitudinal velocity
$a_{\text{lat.}}$	Lateral acceleration
$a_{\text{long.}}$	Longitudinal acceleration
d_{ahead}	Distance to the vehicle ahead in the current lane
d_{behind}	Distance to the vehicle behind in the current lane
$d_{\text{l, ahead}}$	Distance to the vehicle ahead in the left lane
$d_{\text{l, behind}}$	Distance to the vehicle behind in the left lane
$d_{\text{r, ahead}}$	Distance to the vehicle ahead in the right lane
$d_{\text{r, behind}}$	Distance to the vehicle behind in the right lane

TABLE I: The considered attributes ($\Gamma = 10$).

B. Time to Lane Change Classification

Now, we can formulate the TTLC classification problem with each multi-variate time-series training datum defined in (1). To this end, we first extract all scenarios where a vehicle took an action within the time-stamps $1, \ldots, N$, i.e., we only consider scenarios where the vehicle was in view for at least N time-stamps. There are three possible actions to be classified: the vehicle either stayed in its lane for all N time-stamps, the vehicle changed lanes to the left, or it changed lanes to the right, within the N time-stamps. In total, we extracted 3685 left lane changes, 3122 right lane changes and 4000 lane keeps from the HighD dataset [17].

Next, the multi-variate time-series signals from (1) are each divided into $P = N/N_{\text{sect.}} \in \mathbb{Z}_+$ sections of equal length $N_{\text{sect.}}$. Every section is considered a datapoint $\boldsymbol{X} \in \mathbb{R}^{\Gamma \times N_{\text{sect.}}}$. Thus, every datapoint either corresponds to a scenario where a vehicle kept its lane (each datapoint is labeled K for lane keep) or changed lanes; those datapoints belonging to a left lane change are labeled L, and those belonging to a right lane change are labeled R. We further index the lane change labels, L_ρ and R_ρ, with $\rho = 1, \ldots, P$, depending on how many sections in the future the lane change occurs. For example, the label L_1 corresponds to a left lane change in the current section, L_2 corresponds to a left lane change in the following section, and so on.

The task of TTLC classification is to classify the label for each datapoint. The set of possible labels is defined as $\mathbb{Y} = \{\mathrm{L}_1, \mathrm{L}_2, \ldots, \mathrm{L}_P, \mathrm{K}, \mathrm{R}_1, \mathrm{R}_2, \ldots, \mathrm{R}_P\}$. Therefore, the dataset is summarized as

$$\mathcal{D} = \{(\boldsymbol{X}^1, y^1), (\boldsymbol{X}^2, y^2), \ldots, (\boldsymbol{X}^M, y^M)\}, \qquad (2)$$

where each input multi-variate time series $\boldsymbol{X}^m \in \mathbb{R}^{\Gamma \times N_{\text{sect.}}}$ and the class labels $y^m \in \mathbb{Y}$.

Since the HighD data was recorded with a sampling frequency of 25 Hz, we split the scenario data into sections of length 2 s, i.e., $N_{\text{sect.}} = 50$ time-stamps. With scenarios of length $N = 150$, we therefore have 3 classes for the left lane changes, three classes for right lane changes and one class for lane keep, i.e., $\mathbb{Y} = \{\mathrm{L}_1, \mathrm{L}_2, \mathrm{L}_3, \mathrm{K}, \mathrm{R}_1, \mathrm{R}_2, \mathrm{R}_3\}$. After we split the scenarios into sections and balance the classes, we have a total of 21854 samples; these are separated into 80% for the training set and 20% for the test set, i.e., we separate the dataset such that

$$\mathcal{D}_{\text{train}} \cap \mathcal{D}_{\text{test}} = \emptyset \text{ and } \mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{test}} = \mathcal{D},$$

with $|\mathcal{D}_{\text{train}}| = 0.8\,|\mathcal{D}| = M_{\text{train}}$ and $|\mathcal{D}_{\text{test}}| = 0.2\,|\mathcal{D}| = M_{\text{test}}$.

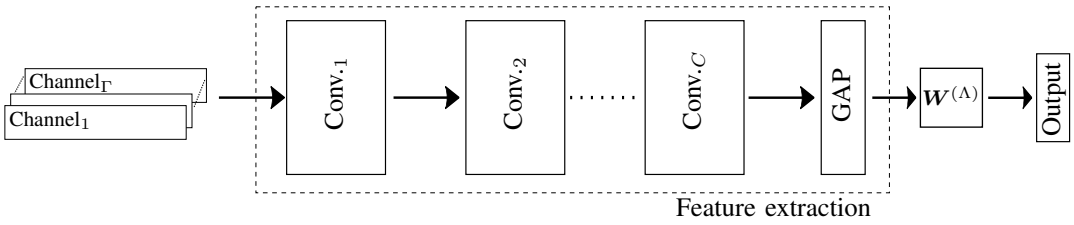


Fig. 1: Abstract CNN architecture.

III. Deep Neural Network Architecture

The general architecture which we employ for our CNN designs is depicted in Fig. 1. The input to the CNN is a multi-variate time-series datapoint, which we interpret as a multi-channel input. This is passed through C consecutive convolutional blocks depicted as Conv.$_1, \ldots,$ Conv.$_C$. These blocks are built of different convolutional layers, which will be discussed in the following sub-sections, to investigate their capability of extracting discriminative features. Note, we refer to the input time-series datapoint has having multiple channels, but after Conv.$_1$, we refer to the filtered signals as feature maps. After the convolutional blocks we employ a Global Average Pooling (GAP) layer, which takes the average of each filtered signal per feature map (see Sub-section III-E). This ensures that we have the same dimensional feature embedding before the classification in the output space. We consider the combination of the convolutional blocks and GAP layer as the feature extraction part of the CNN – see the dotted block labeled *feature extraction* in Fig. 1.

The convolutional blocks are built up of two functions: first, a convolutional layer is applied to generate feature maps, and second, the feature maps are passed through a non-linear activation function. The properties of the different convolution layers are explained in the following sub-sections. The non-linear function can be described as

$$\boldsymbol{x}_\gamma^{(i)} = \sigma(\boldsymbol{z}_\gamma^{(i)}), \tag{3}$$

where $\boldsymbol{x}_\gamma^{(i)} \in \mathbb{R}^{N_{(i)}}$ and $\boldsymbol{z}_\gamma^{(i)} \in \mathbb{R}^{N_{(i)}}$ represent time-series signals. The index $\gamma \in \{1, 2, \ldots, \Gamma^{(i)}\}$ represents the feature map γ in layer (i) for $i = 1, \ldots, \Lambda$; $\Gamma^{(0)} = \Gamma$ describe the input channels. We consider the hyperbolic tangent function as the non-linear activation function, viz. $\sigma(\boldsymbol{z}) = \tanh(\boldsymbol{z})$, applied element-wise.

We define the operator to create a matrix with a Toeplitz structure of the vector $\boldsymbol{w} = \begin{bmatrix} w[1] & w[2] & \ldots & w[L] \end{bmatrix}^{\mathrm{T}} \in \mathbb{R}^L$ as

$$\begin{aligned} \mathrm{toep}\left(\boldsymbol{w}; N\right) &= \begin{bmatrix} \boldsymbol{L}_0\boldsymbol{w}, & \boldsymbol{L}_1\boldsymbol{w}, & \ldots, & \boldsymbol{L}_{N-L}\boldsymbol{w} \end{bmatrix}^{\mathrm{T}} && (4) \\ &= \begin{bmatrix} w[1] & w[2] & \ldots & w[L] & 0 & \ldots & 0 \\ 0 & w[1] & w[2] & \ldots & w[L] & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & w[1] & w[2] & \ldots & w[L] \end{bmatrix} \\ &\in \mathbb{R}^{N-L+1 \times N}, && (5) \end{aligned}$$

with the selection matrix defined as

$$\boldsymbol{L}_l = \begin{bmatrix} \boldsymbol{0}_{L\times l}, & \boldsymbol{I}_{L\times L}, & \boldsymbol{0}_{L\times(N-L-l)} \end{bmatrix}^{\mathrm{T}} \in \mathbb{R}^{N\times L}, \tag{6}$$

where the parameter N is the length of the signal which the filter will be convolved with, and $l = 0, \ldots, N-L$.

A. Standard Convolutional Layers

The first type of 1-D convolutional layer which we investigate is the standard convolution originally introduced for CNNs in [5]. These 1-D filters are applied to each feature map of the previous layer and summed up to generate a feature map in the current layer. This can be interpreted as learning a Finite Impulse Response (FIR) filter for each input feature map and summing over the filtered signals, i.e., the feature map created in a standard convolutional layer can be expressed as

$$\boldsymbol{z}_\gamma^{(i)} = \sum_{c=1}^{\Gamma^{(i-1)}} \mathrm{toep}\left(\boldsymbol{w}_{\gamma,c}^{(i)}; N_{(i-1)}\right) \boldsymbol{x}_c^{(i-1)}. \tag{7}$$

We assume that there are $\Gamma^{(i-1)}$ feature maps in layer $(i-1)$, and $\Gamma^{(i)}$ feature maps in layer (i), i.e., $\gamma \in \{1, 2, \ldots, \Gamma^{(i)}\}$. The filters in layer (i) are assumed to have the length $L_{(i)}$, hence the output signal after the convolutional operation has the dimension $N_{(i-1)} - L_{(i)} + 1$.

As our input signals are multi-variate time-series data, the filters in the first convolutional block (Conv.$_1$) can be interpreted as FIR filters on the raw input data, which not only extract temporal correlations in the input signals, but also cross-channel correlations. However, we should note that each feature map $\boldsymbol{z}_\gamma^{(1)}$ in the first layer is a summation of filtered input signals. Thus, it is difficult to give a physical interpretation of the feature maps when employing the standard 1-D convolution after the first layer. For example, filtered acceleration signals will be summed together with filtered distance signals which loses its physical interpretation. We will return to this observation in Sub-section III-D.

B. Depth-wise Separable Convolutional Layers

The second type of 1-D convolutional layer we investigate is a depth-wise separable convolution, which was popularized in recent years for image classification, see, e.g., [18], [19]. The main idea behind the 1-D depth-wise separable convolution is to split the convolutional task into two parts: first, the temporal correlations per feature map are extracted; and second, the cross-channel correlations are extracted using

a 1×1 convolution operation. By splitting the convolutional operation into two stages, the number of required parameters is reduced with similar performance (cf. [20]). The two stages of the convolutional operation can be expressed as

$$\boldsymbol{z}_{\gamma}^{(i)} = \sum_{c=1}^{\Gamma^{(i-1)}} \operatorname{toep}\left(\tilde{\boldsymbol{w}}_{\gamma,c}^{(i)}; N_{(i-1)}\right) \boldsymbol{x}_{c}^{(i-1)}, \tag{8}$$

with $\tilde{\boldsymbol{w}}_{\gamma,c}^{(i)} = w_{\gamma,c}^{(i)} \cdot \hat{\boldsymbol{w}}_{c}^{(i)}$. In the first stage, a discrete-time convolution with the weights $\hat{\boldsymbol{w}}_{c}^{(i)} \in \mathbb{R}^{L_{(i)}}$ is performed on each feature map of the input signal. Then in the second stage, the output in feature map γ is a scalar multiplication with the weight $w_{\gamma,c}^{(i)} \in \mathbb{R}$ per feature map and then summed over all feature maps.

When applying the depth-wise separable convolution to our multi-variate time series data, we must note that for each convolutional block in the CNN, only one FIR filter, $\hat{\boldsymbol{w}}_{c}^{(i)}$, is learned per feature map (see (8)). Therefore, only the most discriminative temporal correlations can be extracted from the input feature map, however, due to the point-wise multiplication with the weights $w_{\gamma,c}^{(i)}$, the influence of each input feature map for each output feature map can be adjusted. If we look at the signals in the first layer of the CNN, we see that this means, the influence of the cross-channel correlation between, e.g., a filtered acceleration signal and a filtered distance signal, can be adjusted using these weights. Again, similar to the standard 1-D convolution, a physical interpretation of the filtered signals is lost after the first layer, due to the mixing of input feature maps.

C. Local Convolutional Layers

The third type of 1-D convolutional layer we investigate is the locally connected 1-D convolutional layer. It is similar to the standard 1-D convolution; however, the weights are no longer shared, viz. a new set of $L_{(i)}$ weight parameters is learned for each part of each feature map of the input sequence. This can be formulated as

$$\boldsymbol{z}_{\gamma}^{(i)} = \sum_{c=1}^{\Gamma^{(i-1)}} \operatorname{toep}\left(\boldsymbol{w}_{\gamma,c,0}^{(i)}, \ldots, \boldsymbol{w}_{\gamma,c,N-L}^{(i)}; N_{(i-1)}\right) \boldsymbol{x}_{c}^{(i-1)}, \tag{9}$$

with a slight abuse of notation of the toep operator where each filter vector, $\boldsymbol{w}_{\gamma,c,l}^{(i)}$, is multiplied with the selection matrix $\boldsymbol{L}_{l}$, as defined in (4). Moreover, each filter vector has individual weights, i.e., $\boldsymbol{w}_{\gamma,c,l}^{(i)} \in \mathbb{R}^{L_{(i)}}, \quad \forall l \in \{0, 1, \ldots, N_{(i-1)} - L_{(i)}\}$, and thus, the weights are not shared over the input signal.

By relaxing the weight sharing condition in the standard convolutional layer, we allow the CNN to learn discriminative temporal correlations at each possible sequence of input signals. This comes at the price of more parameters per feature map, since the weights at each sequence of the input signal are learned independently.

D. Multi-Channel Convolutional Layers

Finally, we want to investigate a network architecture where each input channel is filtered individually; similar to the CNN design in [13]. In this design, instead of processing the multi-variate input signal as one signal, i.e., processing all channels at once, each channel is filtered individually. Therefore, if we refer to the convolution operations in (7), (8) or (9), instead of creating a feature map by summing over the input channels, we create a feature map per channel, i.e.,

$$\boldsymbol{z}_{\gamma,c}^{(1)} = \operatorname{toep}\left(\boldsymbol{w}_{\gamma,c}^{(1)}; N_{\text{sect.}}\right) \boldsymbol{x}_{c}^{(0)}, \ \forall c \in \{1, \ldots, \Gamma^{(0)}\}, \tag{10}$$

where $\operatorname{toep}\left(\boldsymbol{w}_{\gamma,c}^{(1)}; N_{\text{sect.}}\right) \in \mathbb{R}^{N_{\text{sect.}} - L_{(1)} + 1 \times N_{\text{sect.}}}$ has the Toeplitz form defined by the 1-D convolutional layers in (7), (8) and (9). Therefore, we introduce three new multi-channel CNN architectures which use the convolutional layers defined in Sub-Sections III-A, III-B and III-C. However, the input channels are now kept separate, i.e., the features of each channel are extracted in parallel.

As discussed in the previous sub-sections, the input signals have different physical units, e.g., m for distances or m/s for velocities, and by summing these filtered signals together, we lose their physical interpretation. However, by separating the input channels, the physical interpretation of the filtered signals is maintained throughout all layers of the CNN. On the other hand, by separating the input channels, only temporal correlations can be extracted by the CNN, i.e., the cross-channel correlations are only taken into consideration by the final weight matrix after the feature extraction.

E. Global Average Pooling and Output

The GAP layer is located right before a linear transformation to the output. This type of layer was first proposed in [21] to prevent overfitting in the final, fully connected layers of a CNN, and the authors also argue that it introduces a robustness against spatial translation of input images. In addition to these advantages, we leverage the GAP layer to bring the extracted feature maps into an equally dimensioned space before the classification; we will refer to this as the *feature embedding space*. In this layer, the signal of each feature map is averaged over the remaining time dimension, i.e., each filter map is replaced by its average signal. Thus, we can control the size of the *feature embedding space* by choosing $\Gamma^{(\Lambda-1)}$ appropriately. The GAP layer can be expressed as

$$z_{\gamma}^{(\Lambda-1)} = \frac{1}{N_{(\Lambda-1)}} \sum_{n=1}^{N_{(\Lambda-1)}} x_{\gamma}^{(\Lambda-1)}[n] \in \mathbb{R}. \tag{11}$$

Therefore, an average signal is taken for each feature map $\boldsymbol{x}_{\gamma}^{(\Lambda-1)}$, in the penultimate layer where $\gamma \in \{1, \ldots, \Gamma^{(\Lambda-1)}\}$.

After the GAP layer, we have feature embedding vectors of the same dimension for each architecture introduced in the previous sub-sections. Note, the separate feature maps in the multi-channel CNN architectures are concatenated at this stage. These feature embedding vectors will now be linearly transformed into the output space, where we

ultimately employ the softmax function to approximate the *posterior* probability of the output classes. Therefore, the output of the CNNs is

$$\boldsymbol{z}^{(\Lambda)} = \mathrm{softmax}(\boldsymbol{W}^{(\Lambda)}\boldsymbol{z}^{(\Lambda-1)}) \in (0,1)^{|\mathbb{Y}|}, \quad (12)$$

with the $\mathrm{softmax}$ function defined as $\mathrm{softmax}(\boldsymbol{x})_i = \exp(x_i)/(\sum_{k=1}^{K}\exp(x_k)), \quad \forall i = 1,\ldots,K$. This is the simplest transformation into the output space for a CNN. Hence, the bulk of the classification power comes from the feature extraction in the Conv. blocks.

F. Fully Connected Neural Network

We additionally investigate whether the constraint of feature extractions using 1-D convolutions can achieve a similar classification performance as a fully connected DNN. To this end, each Conv. block is replaced with a simple linear mapping and non-linear activation function, i.e.,

$$\boldsymbol{z}^{(i)} = \sigma(\boldsymbol{W}^{(i)}\boldsymbol{x}^{(i-1)}), \quad (13)$$

for all layers $i \in \{1,\ldots,\Lambda-1\}$. Note, we exclude the bias term without loss of generality. Moreover, the fully connected network does not require the GAP layer, because it does not generate multiple feature maps which need to be averaged to map into the *feature embedding space*. The output layer is the same as the CNNs as defined in (12).

It should be noted that the input of the DNN is merely a vectorized version of the scenarios, i.e., the input is $\boldsymbol{x}^m = \mathrm{vect}(\boldsymbol{X}^m) \in \mathbb{R}^{\Gamma N_{\text{sect.}}}$. Thus, the physical interpretations, the discussion of the FIR filtering of the signals and the temporal correlations previously discussed are, in general, no longer valid.

IV. Simulation Results

In this section, the simulation results of training each of the previously introduced CNN architectures from Section III are discussed. The objective function used during training is the cross entropy

$$\mathcal{L} = -\sum_{m=1}^{M_{\text{train}}}\sum_{k=1}^{K} t_k^m \ln(z_k^{(\Lambda)}), \quad (14)$$

with the one-hot-encoded true labels $t_k^m \in \{0,1\}$ for all of the training data indexed by m, and the output probabilities defined in (12) for each input $\boldsymbol{X}^m$.

We trained each of the CNNs using the Adam optimizer [22] with a learning rate of $\alpha = 0.0005$ and mini-batches of size 200. Moreover, we employed Early Stopping (ES) with a patience of 50 epochs to reduce overfitting of our training set, and validated the training accuracies using k-fold cross validation with 10 folds (see, e.g., [23, Ch. 7.2]). For each of the CNN architectures, we used three Conv. blocks, keeping the number of training parameters roughly equal for a fair comparison; the exact architectures and ES epochs can be found in Appendix I. We will henceforth refer to the different CNN architectures with the sub-section name, e.g., the standard convolutional layers will be referred to as "CNN-III-A", and the multi-channel standard convolutional layers will be referred to as "CNN-III-A MC".

Architecture	Linear [%]	k-NN [%]	DT [%]
DNN-III-F	89.55(±0.37)	90.08(±0.17)	87.93(±0.30)
CNN-III-A	89.18(±0.26)	88.70(±0.20)	86.25(±0.35)
CNN-III-B	90.36(±0.55)	88.77(±0.44)	85.58(±0.46)
CNN-III-C	90.22(±0.38)	90.40(±0.20)	87.80(±0.35)
CNN-III-A MC	92.75(±0.48)	91.17(±0.34)	88.15(±0.26)
CNN-III-B MC	90.81(±1.70)	87.68(±2.56)	87.32(±1.41)
CNN-III-C MC	92.83(±0.38)	91.33(±0.33)	88.25(±0.37)

TABLE II: Classification accuracies.

In Table II, the left column shows the achieved accuracies (with the 95% confidence intervals in brackets) on the test set after training the various CNN architectures, averaged over the 10 k-folds. We observe that the CNNs achieve comparable, in some cases better, accuracies to the DNN design, with the advantage of the interpretability of the extracted filters. Moreover, the multi-channel designs, where each input channel—each physical signal—is processed separately achieved the highest accuracy on average. However, this comes at a price of a larger confidence interval, especially for the CNN-III-B MC design.

Moreover, we should note that during training, the multi-channel CNN architectures took more epochs to converge (see Appendix I). However, there was no obvious divergence between the training and the validation loss, which further supports the argument that the multi-channel designs keep the physical interpretation of the signals intact, in turn reducing overfitting.

In the middle and rightmost columns, we see the achieved accuracies, if we train a k-NN [24] or a DT [25] algorithm with the features extracted after the CNN layers, i.e., using the test data in the *feature embedding space*. We investigate this to determine how much the final accuracies depend on the linear transformation matrix $\boldsymbol{W}^{(\Lambda)}$, which is trained, and hence, different for each of the CNNs. We observe that the achieved accuracy across all methods is only slightly lower for both the k-NN and the DT. Note that with additional hyperparameter tuning the accuracies may converge to those achieved by the CNN architectures with $\boldsymbol{W}^{(\Lambda)}$.

These results imply that the features extracted by the CNN architectures are discriminative for the classification task, and the achievable accuracy is independent of the final linear transformation. With this result in mind, we argue that an investigation of the quality of the extracted features is meaningful.

V. Feature Validation

As discussed in Section IV, merely taking the final accuracy into account does not provide sufficient evidence as to whether one CNN is better than another, nor does it suffice for the validation of the ML algorithm. Therefore, we argue that an investigation into the quantitative differences between the extracted features is meaningful for the goal of validation. To this end, we propose the first pillar of *validation by diversity*, where we investigate the extracted features in a two pronged approach. In the first stage, we

propose that the extracted features in the *feature embedding space* can be clustered, and the quality of this clustering can be quantified using an appropriate measure. In the second stage, the extracted features are visualized, and the clustering can be inspected.

A. Feature Clustering

The first stage of this *validation by diversity* approach, is to extract all of the features from the test set and investigate the extracted features in the *feature embedding space*. Since we have designed the CNN architectures in such a way, that the final classification is only a linear transformation of the features from the *feature embedding space* to the output space, we argue that an analysis of these extracted features is meaningful.

To this end, we first pass the test set through our CNNs in a forward pass, and store the extracted features after the GAP layer, i.e., first we take the test set and extract the learned features

$$\tilde{\boldsymbol{x}}^m = \boldsymbol{z}^{\Lambda-1} = \text{feat}(\boldsymbol{X}^m) \in \mathbb{R}^{100},\ \forall \boldsymbol{X}^m \in \mathcal{D}_{\text{test}}, \quad (15)$$

where we abstract the first $\Lambda-1$ layers of the CNNs into the function $\text{feat} : \mathbb{R}^{\Gamma^{(0)} \times N_{\text{sect.}}} \rightarrow \mathbb{R}^{100}$, with a 100 dimensional *feature embedding space*. These extracted feature vectors create the set: $\tilde{\mathcal{D}}_{\text{test}} = \{\tilde{\boldsymbol{x}}^m\}_{m=1}^{M_{\text{test}}}$.

Next, we use an unsupervised clustering algorithm, k-means [23, Ch. 14], to cluster the extracted features in $\tilde{\mathcal{D}}_{\text{test}}$. Since we have the correct class labels for each datum *a-priori* from our test set, and hence, for each datum in $\tilde{\mathcal{D}}_{\text{test}}$, we can use an external cluster validation method to validate the unsupervised k-means clustering algorithm. To this end, we use the Adjusted Rand Index (ARI) (see [15], [26]), to validate the clusters generated by the k-means algorithm. The clusters generated by the k-means algorithm are given the labels $\{C'_1, \dots, C'_{K'}\}$. At first glance, these labels have no obvious relationship with the data labels from $\mathbb{Y}$, thus further analysis is required.

The ARI, which lies in the interval $[0, 1]$, is the percentage of datapoints where the cluster labels agree with the true data labels, it is adjusted for randomness in the clustering labels. A low ARI indicates that the clustering does not correspond to the labels. We vary the total number of clusters for k-means between $K' \in \{2, \dots, 2 \cdot K\}$ with the total number K of true classes; in our simulations $K = 7$. The calculation of the ARI is based on a contingency matrix $\boldsymbol{D} \in \mathbb{R}^{K \times K'}$, where every row corresponds to one of the true classes and every column corresponds to one of the possible K'-means clusters. The entry $d_{i,j}$ counts the number of datapoints that are in class i and cluster j. In the end, the ARI is calculated as [26]

$$\text{ARI} = \frac{\sum_{i,j}\binom{d_{i,j}}{2} - \left(\sum_i \binom{d_{i\cdot}}{2} \sum_j \binom{d_{\cdot j}}{2}\right) / \binom{M}{2}}{\frac{1}{2}\left(\sum_i \binom{d_{i\cdot}}{2} + \sum_j \binom{d_{\cdot j}}{2}\right) - \left(\sum_i \binom{d_{i\cdot}}{2} \sum_j \binom{d_{\cdot j}}{2}\right) / \binom{M}{2}}, \quad (16)$$

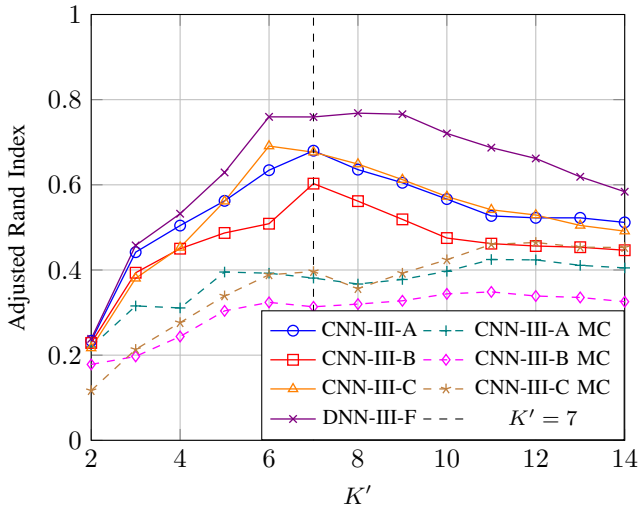


Fig. 2: The average ARI for different cluster sizes.

where $d_{i\cdot}$ and $d_{\cdot j}$ represents a sum over row i and a sum over column j, respectively, and the total number of datapoints M.

We run the k-means with 100 random initializations and take the clustering with best inertia value for each K' value, i.e., the lowest sum of squared distances between the samples and their closest cluster center. The results of the clustering can be seen in Fig. 2. We observe that the maximum ARI, i.e., the number of clusters which corresponds with the true labels, is at $K' = 7$ for two of the CNNs and the DNN. The DNN shows good clustering despite the missing interpretability of the physical signals. On the other hand, the multi-channel CNNs, which showed the best accuracy results, have a local minimum at around $K' = 7$. This lower ARI for the multi-channel CNNs, could be explained by the missing cross-channel correlation during the feature extraction.

An investigation of the entries of the contingency matrix of a specific method to investigate the extracted features can also prove fruitful. For example, the contingency matrices of the CNN-III-A and the CNN-III-A MC are depicted in Table III. Note, the rows are normalized per true class label. We see that the clustering label C_3 for the CNN-III-A has grouped the lane keeps, and the lane changes in the class L_3 and R_3 together; furthermore, only left lane changes and only right lane changes are clustered in C_1 and C_2, respectively. This shows that the method extracted logical features for both left lane changes and right lane changes. Moreover, the features to distinguish between a vehicle which stays in its lane are similar to those where a vehicle changes lane in 4-6 s, i.e., long before a lane change the vehicles stay in their lanes. On the other hand, clustering of the features extracted by CNN-III-A MC, shows that no clear distinction between left and right lane changes is seen in the extracted features.

A similar analysis of the contingency matrices, for different cluster sizes K', can be performed for all CNN architectures. However, our exemplary analysis of the CNN-

	C_1	C_2	C_3	C'_1	C'_2	C'_3
R_3 [%]	0	11.49	88.51	0.65	76.27	23.08
R_2 [%]	0	99.03	0.97	0	97.22	2.78
R_1 [%]	0	99.85	0.15	0	96.87	3.13
K [%]	0	0.18	99.82	0.17	24.44	75.39
L_1 [%]	100	0	0	99.84	0	0.16
L_2 [%]	99.84	0	0.17	11.09	0	88.91
L_3 [%]	11.62	0	89.38	0.31	0	99.69

TABLE III: Normalized contingency matrix for CNN-III-A (C) vs. CNN-III-A MC (C') for $K' = 3$.

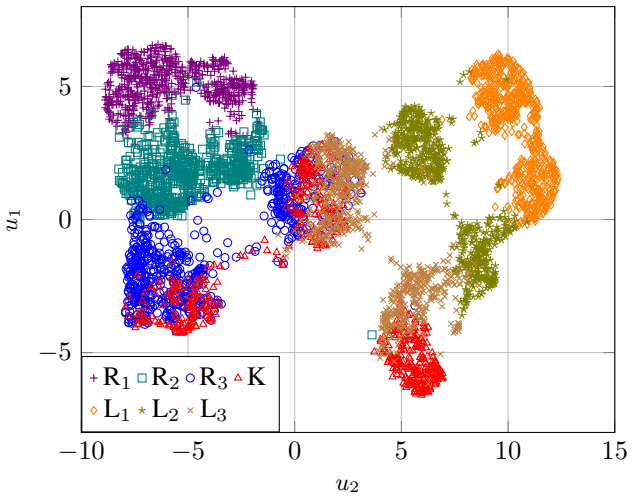
III-A and the CNN-III-A MC, already indicates that the former method extracted features of a higher quality from the test data, which are also explainable. This result contradicts the higher classification accuracy of the CNN-III-A MC. Therefore, we argue that by clustering the extracted features in the *feature embedding space*, we have come one step closer to validating one ML algorithm over another. In the following sub-section, we take the next step in *validation by diversity*, and confirm the quality of the extracted features by visualizing them.

B. Feature Visualization

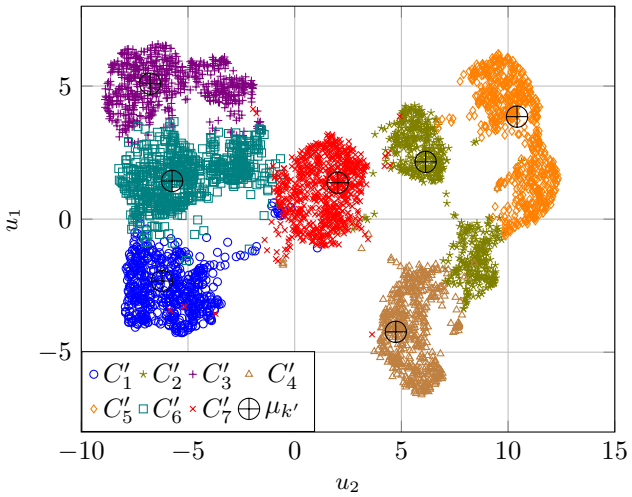
For experts to be able to validate the features extracted by the different methods, they should be able to visualize them in addition to the clustering analysis presented in the previous sub-section. To this end, we propose that the extracted features in the *feature embedding space* should be visualized using a state-of-the-art dimensionality technique, e.g., UMAP [16]. The main advantages of UMAP when compared with other non-linear dimensionality reduction techniques, e.g., t-distributed Stochastic Neighborhood Embedding (t-SNE), is that it is designed to not only preserve local dependences, but also to map the global structures within the data into the lower dimension. Moreover, once the UMAP has been calculated, it can easily be used to transform previously unseen input data into the lower dimension.

To this end, we plot the UMAP dimensionality reduction of the CNN-III-A using the parameters: 15 neighbors in the higher dimension, and a minimum distance of 0.4 between points in the lower dimension. These parameters provide a good representation of the global and local structures in the data and within the clusters, which can be seen in Fig. 3a and Fig. 3b. First, observe in Fig. 3a how the features create logical clusters, i.e., the right changes R_1 and R_2 are clustered in the upper left hand side of the plot. Moreover, we see that the three classes with similar features, viz. R_3, L_3 and K, are clustered in the middle of the plot with many overlapping points. A closer study of the classification predictions showed that these three classes were indeed difficult to differentiate and lead to most of the classification errors. In general, datapoints were misclassified when they lay within the cluster of another class.

In Fig. 3b, we can validate the ARI results presented in the previous sub-section. Here, the cluster labels calculated by the k-means clustering are plotted for $K' = 7$, including



(a) The true test data labels.



(b) The k-means cluster labels.

Fig. 3: The UMAP embeddings for the CNN-III-A with different labels.

the k-means centers (the $\oplus$ points). Here, we observe that the features are well clustered by the k-means algorithm and, in general, the clusters correspond to each of the true label clusters. By employing the k-means clustering and the UMAP dimensionality reduction, we are able to confirm the findings in Section V-A about the extracted features. Moreover, the UMAP can be used to visualize new input datapoints, which can then be validated by observing whether they are embedded close to the k-means cluster center of the corresponding class or not. This gives experts an additional method to visualize the extracted features for validation.

Therefore, despite the similar accuracies displayed by CNN-III-A and CNN-III-A MC, the presented discussions and visualizations lead us to conclude that the features extracted by CNN-III-A are of better quality. This may help a user when deciding between which of the two ML algorithms should be implemented in a safety critical driving function.

VI. Conclusion

In this paper, we first investigate and interpret different CNN architectures which can be used to classify the TTLC in a highway driving scenario. These CNN models were trained on a public dataset, and all showed comparatively similar classification accuracies after training. We argue that merely trusting the classification accuracy of a learned ML algorithm is not sufficient for validation, especially when it comes to safety critical driving functions. We propose a two pronged approach to validate the extracted features from these CNNs. We observed that a method with a lower classification accuracy actually clustered the extracted features more meaningfully, which supports the argument that the final accuracies can be misleading. This approach is the first pillar of what we call *validation by diversity*.

The following extensions of this work are possible: first, the GAP layer after the feature extraction layers can be leveraged to visualize the regions in the input signal which were important for a given classification (see, e.g., [10], [27]). Second, the learning objective can be reformulated to extract features which are inherently separated and clustered (similar to [28]). Thus, these features can be directly classified by a k-NN algorithm, relieving the dependency on the trainable linear transformation matrix at the output of the CNN and simultaneously maintaining the interpretability of the convolutional layers.

Appendix I
Network Architectures

The CNN architectures were trained each with $C = 3$ convolutional blocks with filters of length $L_{(1)} = 8, L_{(2)} = 5$ and $L_{(3)} = 3$, respectively. To keep a fair comparison in terms of possible degrees of freedom, we maintained roughly the same number of parameters for each of the CNN architectures. The number of filters (or neurons for the DNN) per layer, the total parameter count, and the average ES epoch can be found in Table IV.

Architecture	Num. Filters	Num. Parameters	ES Epoch
CNN-III-A	(211, 260, 100)	369,880	152
CNN-III-B	(550, 552, 100)	369,486	282
CNN-III-C	(20, 20, 100)	369,500	192
CNN-III-A MC	(65, 102, 10)	368,000	290
CNN-III-B MC	(128, 256, 10)	369,420	560
CNN-III-C MC	(10, 11, 10)	371,700	383
DNN-III-F	(375, 380, 100)	369,555	139

TABLE IV: CNN parameters and average ES epoch.

References

[1] 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/abs/1709.02435

[2] H. Q. Dang, J. Fürnkranz, A. Biedermann, and M. Hoepfl, "Time-to-lane-change prediction with deep Learning," in *Proc. of IEEE Int. Conf. on Intell. Transp. Syst. (ITSC)*, 2017.

[3] Z. Yan *et al.*, "Time to lane change and completion prediction based on Gated Recurrent Unit Network," in *Proc. of 2019 IEEE Intell. Vehicles Symp. (IV)*. IEEE, 2019, pp. 102–107.

[4] A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh, "The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances," *Data Mining and Knowledge Discovery*, vol. 31, no. 3, pp. 606–660, 2017.

[5] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[6] Y. Le Cun *et al.*, "Handwritten zip code recognition with multilayer networks," in *Proc. of 10th Int. Conf. on Pattern Recognit.* IEEE, 1990, pp. 35–40 vol.2.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[8] B. Zhao, H. Lu, S. Chen, J. Liu, and D. Wu, "Convolutional neural networks for time series classification," *J. of Syst. Eng. and Electronics*, vol. 28, no. 1, pp. 162–169, 2017.

[9] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P. A. Muller, "Deep Learning for time series classification: a review," *Data Mining and Knowledge Discovery*, vol. 33, no. 4, pp. 917–963, 2019.

[10] Z. Wang, W. Yan, and T. Oates, "Time series classification from scratch with deep neural networks: A strong baseline," in *Proc. of 2017 Int. Joint Conf. on Neural Networks (IJCNN)*, 2017, pp. 1578–1585.

[11] Y. Chen *et al.*, "The UCR Time Series Classification Archive," Jul 2015. [Online]. Available: https://www.cs.ucr.edu/~eamonn/time_series_data/

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

[13] 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 Comput. Science*, vol. 10, no. 1, pp. 96–112, 2016.

[14] O. Gallitz, O. De Candido, M. Botsch, and W. Utschick, "Interpretable feature generation using deep neural networks and its application to lane change detection," in *Proc. of IEEE Int. Conf. on Intell. Transp. Syst. (ITSC)*. IEEE, 2019, pp. 3405–3411.

[15] W. M. Rand, "Objective criteria for the evaluation of clustering methods," *J. of the Amer. Statistical Assoc.*, vol. 66, pp. 846–850, 1971.

[16] L. McInnes, J. Healy, and J. Melville, "UMAP: Uniform manifold approximation and projection for dimension reduction," 2018. [Online]. Available: http://arxiv.org/abs/1802.03426

[17] 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 Syst." in *Proc. of IEEE Int. Conf. on Intell. Transp. Syst. (ITSC)*. IEEE, 2018, pp. 2118–2125.

[18] L. Sifre and S. Mallat, "Rigid-motion scattering for image classification," *Ph. D. thesis*, 2014.

[19] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proc. of 30th IEEE Conf. on Comput. Vision and Pattern Recognit., CVPR 2017*, 2017, pp. 1800–1807.

[20] A. G. Howard *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017. [Online]. Available: http://arxiv.org/abs/1704.04861

[21] M. Lin, Q. Chen, and S. Yan, "Network in network," in *Proc. of 2nd Int. Conf. on Learn. Representations, ICLR 2014*, 2014.

[22] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," in *Proc. of 3rd Int. Conf. on Learn. Representations, ICLR 2015*, 2015.

[23] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning - Data Mining, Inference, and Prediction*. Springer, 2009.

[24] E. Fix and J. L. Hodges, *Discriminatory Analysis, nonparametric discrimination: Consistency Properties*. Tehcnical Report 4: USAF school of Aviation Medicine, 1951.

[25] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and regression trees*. Belmont, CA: Wadsworth & Brooks, 1984.

[26] L. Hubert and P. Arabie, "Comparing partitions," *J. of Classification*, vol. 2, no. 1, pp. 193–218, 1985.

[27] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," *Proc. of the IEEE Conf. on Comput. Vision and Pattern Recognit.*, pp. 2921–2929, 2016.

[28] B. Yang, X. Fu, N. D. Sidiropoulos, and M. Hong, "Towards k-means-friendly spaces: Simultaneous deep learning and clustering," in *Proc. of 34th Int. Conf. on Mach. Learn., (ICML)*, vol. 8, 2017, pp. 5888–5901.