Leveraging combinatorial testing for safety-critical computer vision datasets

Christoph Gladisch Christian Heinzemann Martin Herrmann
Matthias Woehrle
Robert Bosch GmbH, Corporate Research
firstname.lastname@de.bosch.com *

Abstract

Deep learning-based approaches have gained popularity for environment perception tasks such as semantic segmentation and object detection from images. However, the different nature of a data-driven deep neural nets (DNN) to conventional software is a challenge for practical software verification. In this work, we show how existing methods from software engineering provide benefits for the development of a DNN and in particular for dataset design and analysis. We show how combinatorial testing based on a domain model can be leveraged for generating test sets providing coverage guarantees with respect to important environmental features and their interaction. Additionally, we show how our approach can be used for growing a dataset, i.e. to identify where data is missing and should be collected next. We evaluate our approach on an internal use case and two public datasets.

1. Introduction

Testing of perception functions is a challenge, particularly if the perception function is to be used in a safety-relevant setting such as automated driving [18], yet is a vital part of a safety argumentation [17]. A typical approach for evaluation from the machine learning domain is to set aside a dataset for testing that is used to assess the performance of a trained perception function. We may also have dedicated data in order to check whether the perception functions contains specific weaknesses [18], *e.g.* vision related hazards such as blur, occlusion and exposure issues [19].

Designing effective (test) datasets is a challenge of its own as the dimensionality of the operational domain that needs to be mastered by the perception function is typically enormous, especially in a complex domain such as automated driving. As an example, consider the detection of a pedestrian at an urban intersection. In this setting, the perception function needs to be able to detect pedestrians with different appearance (clothing, size, ...) at different daytimes at different kinds of intersections with different kinds of backgrounds under different weather conditions, etc. Due to the high dimensionality, building a test set based on the full cartesian product of all possible dimensions is infeasible. In addition, increasing experience with the perception function or additional use cases to be covered may require an iterative construction of the test set. For example, when we identify new visual effects such as CV hazards [19], additional test images are needed. Our assumption is that for safety-critical applications we must ensure that all known, relevant effects need to be present in a test set. Moreover, we may need to consider the interaction of effects as well. Our view comes from a verification perspective: we are interested in each individual test and not in an aggregate result from a statistical approach [18]. Note that our focus in this work is verification and therefore the test set; however our method can be equally applied to any data set to be constructed, i.e. a training and validation set.

Hence, there is a need for an approach that allows testers to (*i*) create domain models of relevant effects including a so-called base context as well as visual effects (*c.f.* Sec. 2.2). Such an approach must be able to (*ii*) handle large discrete spaces and (*iii*) deal with the iterative nature of dataset creation due to identification of additional test concerns and hazards.

Combinatorial testing [1, 12, 15] is an approach for handling high-dimensional spaces in software testing. A well-known combinatorial testing instance is pairwise testing (also called all-pairs testing) [6]. The input to combinatorial testing is a set of relevant input dimensions (usually referred to as *parameters* in combinatorial testing) of a software under test and a discrete set of possible values (*choices*) for each dimension. Then, a set of (software) tests is designed such that all t-wise combinations of values are present at least once in the test set [12]. Combinatorial testing approaches such as pairwise testing have been proven highly effective for reducing the test set size, yet still revealing errors. Case studies have shown that for software,

*The research leading to the results presented above are funded by the German Federal Ministry for Economic Affairs and Energy within the project *KI Absicherung - Safe AI for automated driving*.

e.g. in the space craft and medical domain, many errors result only from a single dimension or interactions of two dimensions [10].

Combinatorial test tools rely on a model that defines the combinatorial space, yet they do not provide means for efficient and effective model management. However, for practical work in our problem domain, where different domain experts collaborate, support for model management is essential since (*i*) the dimensionality of the input space is very large, (*ii*) the input domain necessitates the integration of knowledge from different domains and (*iii*) analysis evolves over time and parameters and corresponding choices may change. As such, we leverage a method, SCODE, and its implementation in a tool, SCODE-ANALYZER [7] that has shown practical applicability in mastering model complexity particularly for large discrete spaces [8]. We use SCODE to modularize individual (sub-)domains into different submodels capturing the important input dimensions, maintain a model hierarchy and generate a resulting composite model that can be used as input for the combinatorial testing tool.

The contribution of this paper is a method for effective dataset generation for testing perception functions using combinatorial testing. At the core of our method, we capture the results of a domain analysis in a model management tool, SCODE-ANALYZER [7]. From SCODE-ANALYZER, we export a model as input to the combinatorial testing tool PICT [6] for computing a covering test set. While selected parts of the overall approach have been published before, the synthesis of these techniques from separate communities into an overall workflow that provides (incremental) generation of tests with formal guarantees provides a novel approach for coverage guarantees in datasets. While we discuss our contributions from a testing perspective, the method may be used more generally for design and analysis of datasets. Our contributions can be summarized as follows:

1. We present a domain analysis based on a decomposition of base context and visual appearance leveraging an existing tool for analysis.
2. We use the resulting domain model in combination with an open-source combinatorial testing tool to (*i*) generate test sets of reasonable size with formal guarantees on coverage and (*ii*) demonstrate a workflow for iterative improvement of datasets for training and validation of machine learning applications.
3. We demonstrate our approach on an internal use case as well as two public datasets used in the context of a perception function in an autonomous driving context.

As part of our experimental evaluation of our approach, we show the ability of our approach to derive datasets of practical size to cover all pairwise combinations of parameters of the operational domain. In particular, we demonstrate how to build up an initial dataset using our approach and how to incrementally extend the dataset with pairwise coverage guarantees. Additionally, we demonstrate how to analyze existing datasets for pairwise coverage.

2. Background

2.1. Combinatorial testing

Combinatorial testing is a test case generation technique that addresses the dilemma of exponential growth of tests due to exhaustive enumeration of possible combinations of inputs. Since testing typically has a finite test budget and exhaustive testing is typically intractable [10] combinatorial testing addresses the burden of test selection from the full combinatorial product. Combinatorial testing does not require all combinations of parameter values but only all combinations of values for all subsets of parameters containing t parameters, also called t-wise combinations. A particular instance is pairwise testing (i.e. $t = 2$), where test cases shall cover all combinations of value choices for all pairs of input parameters. The validity of this approach has been shown in several domains showing that most faults are caused by a small number of interactions of input dimensions [10].

Mature tools like the open-source PICT tool[1] are available [6]. PICT generates t-wise test suites based on a model of parameters and corresponding choices, allowing a user to additionally model constraints that exclude combinations. PICT supports incremental generation of test suites, by allowing a user to provide so-called *seed* test cases in addition to the model, i.e. test cases that are already available. PICT can analyze the seeds and generate additional test cases for combinations not yet covered. For further details, we refer to [6]. In the context of datasets for computer vision, a test suite in combinatorial testing corresponds to a test set while a test case is one specific image in the test set.

To illustrate the idea of t-wise testing, we describe a simple example, where we have $n = 6$ input parameters with 2-8 possible choices: 2, 2, 3, 7, 7, 8. The total number of all possible combinations possible is: $2*2*3*7*7*8 = 4704$. For t-wise combinations, we have $\binom{n}{t}$ subsets of parameters. In this example we consider $t = 3$ which yields 20 subsets. For each subset we compute all possible combinations, i.e. in our example between 12 and 392 triplets and therefore the number of 9866 resulting combinations $c(t)$ can be determined by:

$$c(t=3) = \underbrace{(2*2*3) + (2*2*7) + \ldots + (7*7*8)}_{20 subsets}$$

The number of tests, however, will be significantly less because a single test case can cover combinations of several parameter tuples concurrently, e.g. in Table 3 we see that 420 tests can cover 7088 pairwise combinations.

[1] https://github.com/microsoft/pict

Wang et al. discuss combinatorial testing in the context of deep learning, showing its applicability in white-box testing of neural networks [11]. While they also show the usefulness of combinatorial testing for handling high-dimensional test spaces, their work considers white-box testing and therefore focuses on coverage metrics of a DNN. Amersbach and Winner propose a functional decomposition approach for validation of automated driving functions and also use combinatorial testing [2]. Similar to this work, they leverage a domain model in the context of automated driving. They reduce the test suites on exemplary parameters spaces from 6.8×10^6 to 5×10^4 tests. Our work differs in a focus on the environment perception tasks and therefore different domain models. Additionally, our approach described in this paper (*i*) relates abstract tests with images for computer vision, (*ii*) includes management of models, and (*iii*) presents an approach for incremental construction of test suites. The latter is a crucial aspect for practical workflows.

2.2. Domain analysis

We rely on an analysis of the domain to identify major factors for the operational domain (base context) as well as for visual effects. For this purpose, we use a commercial tool called SCODE-ANALYZER which has been previously used in automotive applications for mastering complexity [8]. We refer to related work for detailed background, yet offer an overview of essential elements in the following.

SCODE relies on so-called Zwicky-Boxes. An example is shown in Table 1, which we refer to in the following. A Zwicky-Box consists of dimensions (e.g. DAYTIME) and alternatives for each dimension (*morning*, *day*, ...). Please note that these directly correspond to parameters and choices in PICT. Conceptually, the domain model is very similar to a plain PICT model. However, SCODE allows us to iteratively and jointly design composite domain models supporting the development of heterogeneous domain models by experts from different domains, e.g. visual appearance (CV experts; see Sec. 3.1) and base context (ADAS experts; see Sec. 5.1).[2]

Note that the need for domain analysis is already present in several works in the context of machine learning systems. The ML rubric [4] explicitly mentions that "Model quality is sufficient on all important data slices" as part of testing for model development. Especially in a complex domain such as computer vision in autonomous driving where we need to consider performance across a wide range of scenarios, a domain model such as the one presented in this work supports the systematic creation of data slices. Similarly, in order to avoid hidden stratification, an analysis across subsets is important in safety-critical domains such as medicine [13]. The paper describes three methods to avoid hidden stratifications, where their suggested *schema completion* can be assisted by the approach described in this paper.

[2]CV: computer vision, ADAS: advanced driver-assistance system.

2.3. Variation in machine learning

For training and validation of computer vision functions large amount of data is required. However, the amount of data alone is not a sufficient criterion for a good dataset. Variety of the data also must be ensured [14]. Techniques like image augmentation and domain randomization are used to extend and increase the variation of an existing dataset [16]. In this way, the generalization ability of a neural network in a computer vision function is increased during training or it is more robustly assessed during validation and testing. This may include adding different kinds of noise to an image, applying geometrical image transformations, manipulating brightness, and augmenting image content and including objects from other domains. Most transformations operate on a pixel level (color, geometry,...) while our approach contributes to systematic variation on a semantic level. Please note that an individual variation on a semantic level may necessitate many corresponding images especially for training.

3. A motivating example

We start by introducing an example from our domain, where we analyze an existing scenario in the context of NCAP Vulnerable Road Users (VRU) Protection Tests[3] In particular, we study a scenario on an intersection with traffic lights, where a child is jaywalking across the road in front of an approaching vehicle, see Figure 1. A model of such a scenario can be decomposed into: a *base context* that describes the road network, infrastructure, vegetation, etc; *dynamic actors* that participate in the scenario; and a *visual domain* model.

We base our experiments on previously generated simulation sequences that were used for sensitivity analysis w.r.t. environmental conditions. Using this example we show how we use SCODE and PICT with a visual domain model (Sec. 3.1) and then show the advantage of combinatorial testing compared to random data collection (Sec. 3.2). In this first example, the base context (the setup of the intersection) is fixed, but addressing the base context is further described in 5.1.

3.1. A visual domain model for a traffic scenario

A domain analysis may yield a simple model for the VRU example as shown with the Zwicky-Box in Table 1

[3]`https://www.euroncap.com/en/for-engineers/protocols/vulnerable-road-user-vru-protection/`

Figure 1: Intersection scenario at daytime (VRU example)

DAYTIME	*morning*	*day*	*evening*	*night*	
HAZE/FOG	*no*		*yes*		
STREET CONDITION	*dry*	*wet*	*icy*	*snow*	*broken*
SKY	*cloudy*	*no*	*clear*		
RAIN	*no*		*yes*		
REFLECTION ON ROAD	*no*		*yes*		
SHADOW ON ROAD	*no*		*yes*		
VRU TYPE	*adult*		*child*		
VRU POSE	*pedestrian*	*jogger*	*cyclist*		
VRU CONTRAST TO BG	*low*		*high*		

Table 1: Zwicky-Box with aspects for traffic scenarios. (DAYTIME: dimension, *morning:* alternative of this dimension, VRU: Vulnerable Road User, BG: background)

including 10 dimensions. It is the result of a domain expert's analysis combining dimensions 1-7 for visual modeling of the environment with dimensions 8-10 addressing the VRU context. Albeit being simple, it already spans a space of 11 520 possible states/combinations. Additionally SCODE allows us to model constraints that may be physically motivated or restrict the regarded safety scope of the computer vision function, also known as operational design domain (ODD). For example, a physical constraint on this model may be based on the assumption that STREET CONDITION: *wet*|*icy* always results in REFLECTION ON ROAD: *yes*.[4] If we add this constraint in SCODE-ANALYZER, the space is reduced by 2304 states. Thus to cover the complete space 9216 different test cases would be needed. In contrast, PICT generates 21 pairwise test cases.[5]

A test case for the scenario in Figure 1 based on the definitions of the Zwicky-Box is shown below and covers exactly one state. Obviously, this is an abstract test that still needs to be mapped to a concrete test image as further described in Sec. 4.

Test case 1. DAYTIME: *day,* HAZE/FOG: *no,* STREET CONDITION: *dry,* SKY: *cloudy,* RAIN: *no,* REFLECTION ON ROAD: *no,* SHADOW ON ROAD: *yes,* VRU TYPE: *child,* VRU POSE: *pedestrian,* VRU CONTRAST TO BG: *high*

We defined 8 already existing scenarios similar to Test case 1. These 8 initial test cases only cover 14 states in total. If we provide existing test cases to PICT as seeds, it generates 15 *additional* test cases (compared to 21 test cases without considering seed test cases). As described below in Sec. 4, a simple extension of PICT allows us to compute a coverage measure on existing test cases, i.e. to extract how many combinations are already covered by seed cases given to PICT. As we can see from Table 2, there are 321 pairwise combinations to cover for the model. The initially existing 8 scenarios already cover 175 combinations, i.e. $\approx 54.5\%$. In Table 2, we can see how the number of combinations as well as the number of required tests grows as we increase the parameter t. The number of tests is easily manageable for $t \leq 3$ and the execution time of PICT is low.

t	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Tests	21	73	210	544	1224	2484	4393
Combin.	321	2217	9866	29600	60704	84112	75424
Time	< 1	< 1	< 1	1	2	6	11

Table 2: Overview of PICT runs on our NCAP VRU use case. For each setting of parameter t, we see the number of tests PICT generates, the number of combinations, and PICT execution time in seconds.

3.2. Random data collection gets you only so far

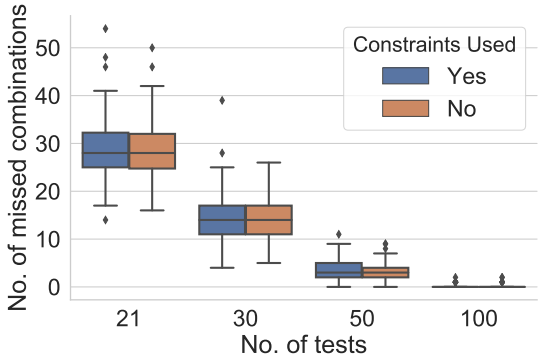

Figure 2: Box plot showing the number of missed combinations (y-axis) for the number of generated tests (x-axis)

State-of-the-art data sets for training, validation and testing are often collected according to some plan but not with focus on satisfying the combination of dimensions of a domain analysis. In the following, we use random sampling to mimic data collection in untargeted real-world driving or simulation. To this end we perform a simple experiment that (*i*) random sampling provides good coverage when sampling a few tests and (*ii*) random sampling suffers in the tail

[4]The example that there needs to be some reflection on the road in this case is just used for illustration.

[5]Respectively 73 for 3-wise combinations.

from not hitting all possible t-wise interactions, i.e., exactly the kind of guarantee that combinatorial testing provides.

For illustration, we use our VRU example. In our experiments we randomly sample (*i*) the model without constraints with 323 pairwise interactions. In addition, we perform (*ii*) random sampling with rejection of invalid combinations resulting in a model with 321 pairwise interactions.

We perform random sampling of test cases from our model and check coverage across subsets in varying sizes of (21, 30, 50, 100), where 21 would be a lower bound determined with systematic generation using combinatorial testing. We perform 100 runs in each setting (with and w/o constraint) and summarize the results as boxplots in Figure 2. For our baseline of 21 tests, random sampling with and without constraints does not achieve full coverage, and misses ≈ 28.5 combinations respectively. Even with 50 test cases, we miss about 3.2 pairwise interactions. As we can see, with 100 tests, we typically achieve full coverage, sometimes missing 1 or 2 combinations. In consequence, this means that constructing a test dataset that achieves pairwise coverage based on random sampling (or untargeted real-world driving) requires a significantly higher amount of data compared to our approach. This would even be further exacerbated when requiring t-wise interactions for $t > 2$. Note that in this simple model, the few constraints do not have an impact on these results. However, for complex interactions between dimensions such constraints may render random sampling even more difficult.

Note that this is a known fact from the software testing literature, *e.g*. Pezzè *et al*. [15] discuss that pairwise testing only grows logarithmically with the number of parameters, while the number of all combinations grows exponentially. However, given the new application context in this work, we underline this fact with a relevant example from our domain and show that there are already benefits for small domains.

4. Approach

Figure 3 shows an overview of how we integrate domain analysis with SCODE and PICT into a test toolchain. Leveraging knowledge from domain and testing experts, the toolchain iteratively reduces the infinite open context space to a manageable test set. The *domain model* is created with SCODE (and may be subsequently refined with constraints) and is exported into a *Combination model* to the combinatorial testing tool PICT. The main output of PICT are *abstract* test cases, i.e. tests on the level of the domain model as shown in Test case 1. As indicated by the Box **X** in the figure, we can either (*i*) synthesize data for these descriptions as shown with Figure 1 or similarly (*ii*) start a data collection campaign with abstract test cases as guidelines for data gatherers.

Note that this workflow allows us to create a diverse test set on a budget [15]: we can (*i*) focus on a restricted domain model containing important characteristics and (*ii*) set the parameter t to adjust the test depth w.r.t interaction of dimensions.

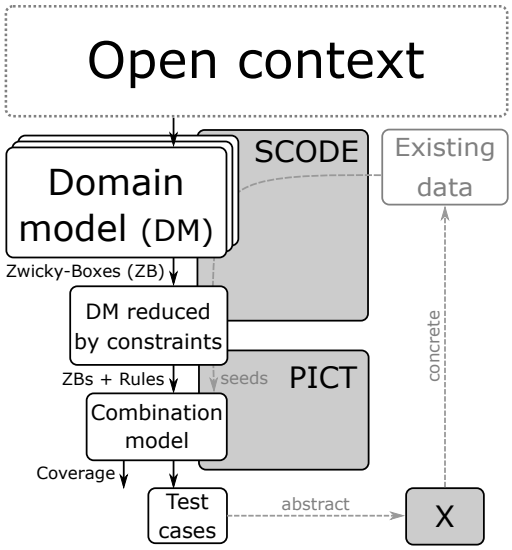

Figure 3: Overview of our approach of integrating domain analysis with SCODE and combinatorial testing using PICT into a test toolchain.

Basis of the approach is a domain model as described above. We use separate Zwicky-Boxes to model (parts of) the open context domain. We constrain the domain model in SCODE using so-called non-system modes. We export the composed domain model to PICT including any constraints. We leverage SCODE modes [8] to model previous test cases and export these for PICT as test case seeds. In summary, this tooling allows us to describe (*i*) the overall model including constraints as well as (*ii*) existing test cases in the language of the domain model. While not integrated yet, for (*ii*) a re-import of generated test cases from PICT into SCODE-ANALYZER would complete the round-trip between the tools.

The tooling provides a user with several outputs: SCODE-ANALYZER already provides us with a state space of the domain model including reductions due to constraints. PICT provides information on the combination model space and lets us generate a set of test cases for complete t-wise coverage. Additionally, we include a simple, yet practical extension in our tooling a calculation of so-called *seed coverage*. Here, we extract from PICT the number of combinations that are already covered by a given set of seed test cases as a further coverage metric.

Test case seeds and seed coverage can also be used to rate an existing dataset that was obtained, e.g., based on real-world driving as shown in the experiments of Sec 5.3 and 5.4. The feedback on missing combinations provides an indication on which test data should be acquired next

and can be used to design targeted test drives.

5. Experiments

In the following, we report on three experiments that we conducted to check the feasibility of our approach. First, we check the ability to generate an initial set of so-called base contexts out of a practically sized SCODE model for road infrastructure (Section 5.1). Second, we demonstrate the impact of adding an additional context dimension to this SCODE model (Section 5.2). This is a typical case when incrementally building up datasets due to knowledge gained during development of a perception function. Finally, we illustrate for two existing datasets, namely Cityscapes [5] and the AEV dataset [9], an analysis for pairwise coverage with our approach (Section 5.3 and 5.4). If the datasets do not achieve full coverage, the analysis also provides information on what additional test cases would be necessary to achieve pairwise coverage. In combination, these experiments cover the main steps of building up a dataset: building up a dataset, checking for completeness of an existing dataset, and incrementally extending the dataset.

5.1. Generation of base context

The base context of a scenario as shown in Figure 1 consists of a road network, landscape, buildings, vegetation, etc. We define a model for roads with a Zwicky-Box *Road* with dimensions ROAD-TYPE, HEADING, SHAPE, ELEVATION, LENGTH, etc. We model road lanes with another Zwicky-Box *Lane* with dimensions LANE-TYPE, WIDTH, LANEMARK-TYPE, LANEMARK-COLOR, etc. We use these Zwicky-Boxes to configure and generate road networks in OpenDrive format [3] for simulation of driving scenarios. To facilitate modeling, we use separate *Road (R)* and *Lane (L)* Zwicky-Boxes and combine them hierarchically (*Road+Lane*) using SCODE. A summary of model complexity is shown in Table 3 detailing the number of dimensions of each model and the resulting state space.

We apply PICT to generate test cases for each model. All PICT runs for these models execute in $< 1s$. The number of resulting tests is shown in the final column of Table 3. As we can see, while the number of states (*i.e.* possible road configurations) suffers from a combinatorial explosion, the number of pairwise combinations stays manageable and even more so the resulting number of tests.

5.2. Incremental data generation

Based on this *Road+Lane* model, we perform an additional experiment that shows the benefit of our model-based approach in an incremental development process. Here, we need to support model updates, e.g. additional domain dimensions and alternatives need to be considered, while existing test cases (and corresponding test artifacts) should be reused to keep the incremental effort low.

ZB	Dims	States	Combin.	Tests
R	7	272 160	1185	185
L	8	6.8×10^6	1789	210
R+L	15	1.8×10^{12}	6604	420
R+L+F	16	7.4×10^{12}	7088	420

Table 3: Overview of different models Road (R), Lane (L) and Friction (F), their composition and the number of states in the domain model, combinations (Combin.) for pairwise coverage as well as generated PICT tests.

As a concrete example, we identify ROAD FRICTION as an environmental factor that should be considered as an additional test dimension. This can be easily added, e.g. as a separate Zwicky-Box *Friction*, with options *very low*, *low*, *normal* and *high*. Our compositional modeling approach allows us to easily integrate ROAD FRICTION into a new test model called *Road+Lane+Friction (R+L+F)*. As we can see in Table 3, the model state space increases by a factor of four due to the four new (unconstrained) options as expected. Interestingly, when we run PICT on this model *without* seed test cases, the resulting number of test cases remains the same. This is because there are other parameters with a large number of choices and existing test cases can be used to include the needed variety in road friction.

However, in our incremental development process we would like to reuse previously generated tests and corresponding test artifacts (e.g. results). Since the test data before did not identify ROAD FRICTION specifically, all previous test data implicitly set the ROAD FRICTION to a fixed value of *normal*. When we leverage the previously generated test cases as seeds for PICT, as described above, PICT merely generates 63 new test cases for adding the new domain with additional options *very low*, *low*, and *high*. Please note that this is an upper bound of additional tests, because having a parameter previously set to a single value is the worst-case for t-wise testing as it represents least possible variability. Newly modeled aspects may have been previously randomly selected such that the variation in the new dimension may further reduce the number of additional test cases. As an example, we randomly populate the friction in the previous 420 test cases. For 10 random experiments, this results typically in no additional test cases, as the test dataset already satisfies the pairwise constraint, i.e. includes sufficient variety in the ROAD FRICTION dimension.[6]

5.3. Analyzing an existing dataset (Cityscapes)

Next, we show the analysis of an existing dataset. In particular we analyze the train and validation set of Cityscapes [5] for pairwise coverage ($t = 2$). Typically

[6] In just a single case *one* additional test case is generated. Our PICT coverage output shows that in this case 7087 out of 7088 valid combinations were already covered.

Data	No. Pics	States	Combin.	Coverage
Train	2307	1.7×10^{10}	2244	89.8 %
Val.	464			82.9 %

Table 4: Seed test cases for train and validation set of Cityscapes, the states space, number of pairwise combinations as well as coverage analysis

we would like to analyze meta-data, but for showing scalability we need detailed metadata in the size of the presented models. As such detailed metadata is not available in the dataset, we use semantic segmentation maps as a proxy for semantic meta-data.[7]

We use semantic segmentation maps as they allow us to determine coverage of semantic concepts in the data. Our experimental setup is as follows: (*i*) We create a SCODE model that includes each segmentation class as a dimension[8] and as options (True/False) whether it is included in an image or not. (*ii*) For all images, we compute from their corresponding semantic segmentation maps if a semantic class is included in the image.[9] (*iii*) The extracted test cases are provided as seed test cases to PICT in order to determine t-wise coverage of the semantic classes as shown in Fig 3.

Table 4 displays the results of our analysis. As we can see, the 34 dimensions with 2 choices each result in 2244 pairwise combinations ($\binom{34}{2} = 561$ subsets with 4 elements each), a significant reduction from the 2^{34} combinations possible. PICT determines that only 13 tests are sufficient to achieve full pairwise coverage (without seed tests). The train and validation set satisfy 89.8% and 82.9% pairwise coverage and 4 and 5 additional tests would be sufficient to achieve full pairwise coverage, respectively. A further analysis of the training set shows that most missing combinations, concretely 3, are within the following dimensions: EGO_VEHICLE, OUT_OF_ROI, LICENSE_PLATE. This is not surprising, since all of these dimensions are set to only one value globally. For the validation set, the corresponding results are similar: the 3 parameters above and additionally TUNNEL are missing the most, i.e. 3 interactions. This is because there are no TUNNEL labels present in the validation set. If we look at the generated test cases for the training set in Table 5 for the above-mentioned variables, we see the following: (*i*) The dimensions are predominantly set to the value that was previously missing. (*ii*) However, 1 out of 4

[7]As we use labels for the analysis and these are not available for a competition test set, we can only analyze training and validation labels.

[8]In particular these are: EGO_VEHICLE, RECTIFICATION_BORDER, OUT_OF_ROI, STATIC, DYNAMIC, GROUND, ROAD, SIDEWALK, PARKING, RAIL_TRACK, BUILDING, WALL, FENCE, GUARD_RAIL, BRIDGE, TUNNEL, POLE, POLEGROUP, TRAFFIC_LIGHT, TRAFFIC_SIGN, VEGETATION, TERRAIN, SKY, PERSON, RIDER, CAR, TRUCK, BUS, CARAVAN, TRAILER, TRAIN, MOTORCYCLE, BICYCLE, LICENSE_PLATE.

[9]We use a simple threshold of 50 pixels to determine if a class is present in an image.

EGO_VEHICLE	OUT_OF_ROI	LICENSE_PLATE
False	False	False
False	False	True
True	False	True
False	True	True

Table 5: Additional training data suggested by PICT w.r.t. variables that are set to only one value in the training set.

value assignments must still be set to the previously available value in order to generate missing interactions with the other dimensions.

Figure 4 gives an additional overview of the amount of missing combinations. As we can see, the training set includes more interactions than the validation set, so the validation set may be insensitive for validating certain types of interactions.

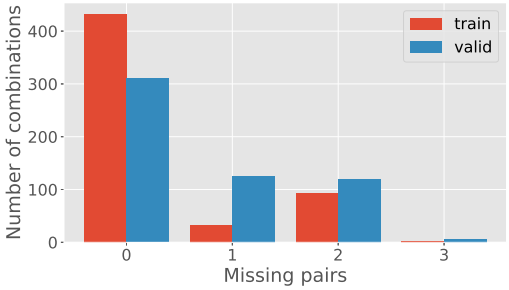

Figure 4: Histograms of missing combinations for dimension pairs of Cityscapes training and validation data.

5.4. Experiments on a larger existing dataset (AEV)

As a second experiment, we perform an analysis of variations with the same setup as above, however with a 5-fold increase in data on a larger dataset [9]. Table 6 summarizes the results. As we can see, even with an increase in model size from 34 to 38 dimensions[10], the increase in the amount of combinations necessary grows moderately from 2244 to 2812. The higher number of tests in the dataset also provides a substantial increase in the coverage w.r.t. the domain model to 96.4%. Even on this larger dataset, PICT generates new test cases for the given model and the larger number of seed test cases in under ten seconds.

A detailed view of missing combinations is provided in Figure 5. We can see that most dimension pairs are completely covered (602), while 101 pairs have three out of four combinations covered. Additionally, all labels are present at least once. The classes most challenged in interaction are: RAINDIRT, SPEEDBUMPER and EGOCAR.

[10]The set contains 55 labels for segmentation with separable instances, which we map to 38 labels: one label per class. We use a threshold of 50 pixels for a class to be relevant.

No. Pics	States	Combin.	Coverage
10557	2.7×10^{11}	2812	96.4 %

Table 6: Seed test cases for train and validation set of the AEV dataset, the states space, number of pairwise combinations as well as coverage analysis

Figure 5: Histograms of missing combinations for dimension pairs of AEV data.

6. Conclusions

This paper describes an approach for data-driven functions such as DNNs that integrates established software engineering methods. In particular, we define a workflow that uses a model of the input domain created and managed with SCODE and combinatorial testing for incrementally creating test sets with coverage guarantees. Our approach enables to generate test sets of reasonable size even for large input domains. We demonstrate our approach on several examples, where we outline test set generation as well as analyzing coverage and incremental test set generation based on existing test sets. While we focus on test sets, our method can be equally applied to training and validation sets as well as for creating stratified data folds based on (parts of) the domain model. In particular this approach is useful for systematic generation of synthetic data, e.g. in a related public project [11].

References

[1] B. S. Ahmed, K. Z. Zamli, W. Afzal, and M. Bures. Constrained interaction testing: A systematic literature study. *IEEE Access*, 5:25706–25730, 2017.

[2] Christian Amersbach and Hermann Winner. Functional decomposition–a contribution to overcome the parameter space explosion during validation of highly automated driving. *Traffic Injury Prevention*, 20(sup1):S52–S57, 2019.

[3] ASAM - Association for Standardization of Automation and Measuring Systems. ASAM - OpenDrive, 2019. `https://www.asam.net/standards/detail/opendrive`.

[4] Eric Breck, Shanqing Cai, Eric Nielsen, Michael Salib, and D. Sculley. The ml test score: A rubric for ml production readiness and technical debt reduction. In *Proceedings of IEEE Big Data*, 2017.

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

[6] Jacek Czerwonka. Pairwise testing in real world. In *24th Pacific Northwest Software Quality Conf.*, volume 200, 2006.

[7] ETAS GmbH. SCODE-ANALYZER Software for describing and visualizing complex closed-loop control systems, 2019. `https://www.etas.com/scode`.

[8] Guilherme Torres Ferreira, Gustavo Tineli, and Martin Herrmann. Flex fuel software maintainability improvement: A case study. In *25th SAE BRASIL International Congress and Display*. SAE International, oct 2016.

[9] Jakob Geyer et al. A2D2: AEV Autonomous Driving Dataset. `http://www.a2d2.audi`, 2019.

[10] D. R. Kuhn, D. R. Wallace, and A. M. Gallo. Software fault interactions and implications for software testing. *IEEE Trans. on Software Engineering*, 30(6):418–421, June 2004.

[11] Lei Ma, Fuyuan Zhang, Minhui Xue, Bo Li, Yang Liu, Jianjun Zhao, and Yadong Wang. Combinatorial testing for deep learning systems. *CoRR*, abs/1806.07723, 2018.

[12] Changhai Nie and Hareton Leung. A survey of combinatorial testing. *ACM Comput. Surv.*, 43(2):11:1–11:29, Feb. 2011.

[13] Luke Oakden-Rayner, Jared Dunnmon, Gustavo Carneiro, and Christopher Ré. Hidden stratification causes clinically meaningful failures in machine learning for medical imaging. *arXiv preprint arXiv:1909.12475*, 2019.

[14] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning. *CoRR*, abs/1712.04621, 2017.

[15] Mauro Pezzè and Michal Young. *Software testing and analysis: process, principles, and techniques*. 2008.

[16] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems*, pages 23–30. IEEE, 2017.

[17] Oliver Willers, Sebastian Sudholt, Shervin Raafatnia, and Stephanie Abrecht. Safety concerns and mitigation approaches regarding the use of deep learning in safety-critical perception tasks, 2020.

[18] Matthias Woehrle, Christoph Gladisch, and Christian Heinzemann. Open questions in testing of learned computer vision functions for automated driving. In *International Conference on Computer Safety, Reliability, and Security*, pages 333–345. Springer, 2019.

[19] Oliver Zendel, Katrin Honauer, Markus Murschitz, Daniel Steininger, and Gustavo Fernández Domínguez. Wilddash - creating hazard-aware benchmarks. In *ECCV 2018*, pages 407–421, 2018.

[11] `https://www.ki-absicherung.vdali.de/`