A Programmatic and Semantic Approach to Explaining and Debugging Neural Network Based Object Detectors

Edward Kim *[1], Divya Gopinath †[2], Corina S. Păsăreanu ‡[2], and Sanjit A. Seshia §[1]

[1]University of California, Berkeley
[2]NASA Ames Research Center

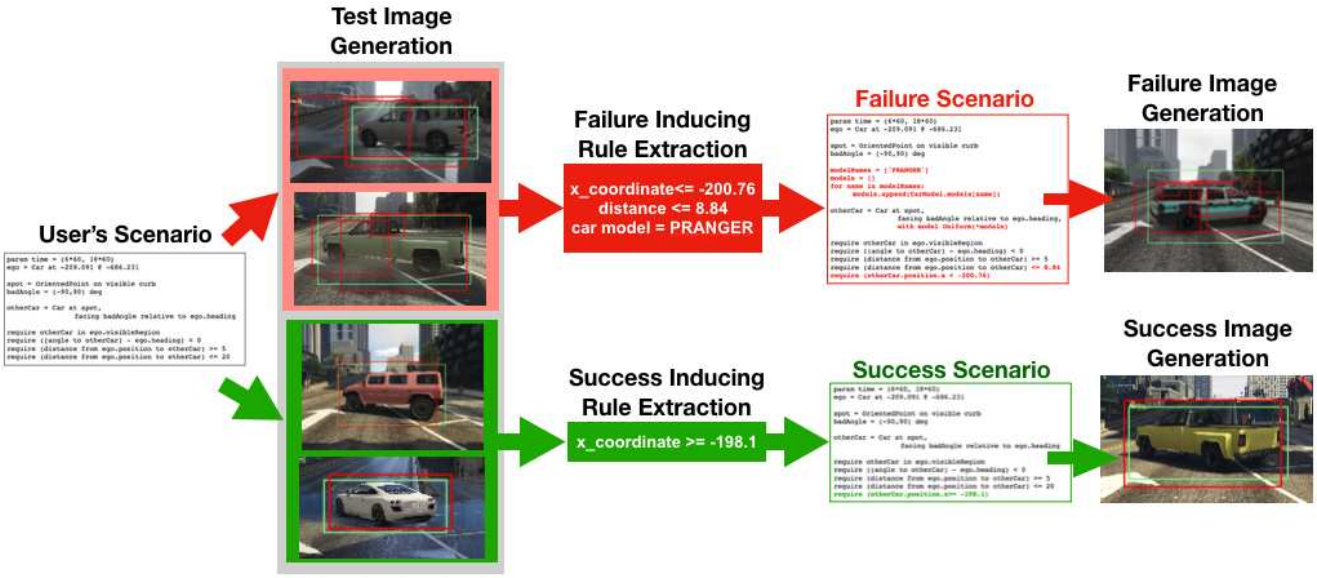

Figure 1: Overview of the workflow proposed in this paper. The green and red bounding boxes in the images are ground truth and prediction, respectively.

Abstract

Even as deep neural networks have become very effective for tasks in vision and perception, it remains difficult to explain and debug their behavior. In this paper, we present a programmatic and semantic approach to explaining, understanding, and debugging the correct and incorrect behaviors of a neural network-based perception system. Our approach is semantic in that it employs a high-level representation of the distribution of environment scenarios that the detector is intended to work on. It is programmatic in that scenario representation is a program in a domain-specific probabilistic programming language which can be used to generate synthetic data to test a given perception module. Our framework assesses the performance of a perception module to identify correct and incorrect detections, extracts rules from those results that semantically characterizes the correct and incorrect scenarios, and then specializes the probabilistic program with those rules in order to more precisely characterize the scenarios in which the perception module operates correctly or not. We demonstrate our results using the SCENIC *probabilistic programming language and a neural network-based object detector. Our experiments show that it is possible to automatically generate compact rules that significantly increase the correct detection rate (or conversely the incorrect detection rate) of the network and can thus help with understanding and debugging its behavior.*

1. Introduction

Models produced by Machine Learning (ML) algorithms, especially *deep neural networks* (DNNs), have proved very effective at performing various tasks in computer vision and perception. Moreover, ML models are being deployed in domains where trustworthiness is a big concern, such as automotive systems [18], health care [3], and cyber-security [6]. Research in adversarial machine learning [11], verification [8, 24], and testing [28] has shown that DNN-based vision/perception systems are not always robust and can be fooled, sometimes leading to unsafe situations for the overall system (e.g., autonomous vehicle).

Given this lack of robustness and potential for unsafe behavior, it is crucial that we develop methods to better understand, debug, and characterize scenarios where DNN-based perception components fail and where they perform correctly. The emerging literature on explaining and understanding ML models provides one approach to address this concern. However, while there are several techniques proposed to explain the behavior of ML-based perception (e.g. [5, 16, 17, 19, 25]), almost all of them operate on the concrete input feature space of the network. For example, attribution-based methods(e.g. [26, 31, 23]) indicate pixels in an input image that are associated with the output of a DNN on that input. These methods, while very useful, do not directly identify the higher-level "semantic" features of the scene that are associated with that decision; they require a human to make that judgment. Additionally, in many cases it is important to generate "population-level" explanations of correct/incorrect behavior on such higher-level features. For example, it would be useful to identify whether the perception module of an autonomous vehicle generally misses cars of a certain model or color, or on a particular region of a road, and leverage this knowledge to describe a high-level success/failure scenario of a perception module *without* the bottleneck of human intervention.

In this paper, we present a *programmatic and semantic approach* to explaining and debugging DNN-based perception module, with a focus on object detection. In this approach, we begin by formalizing the semantic feature space as a distribution over a set of scenes, where a scene is a configuration of objects in the three dimensional space and semantic features are features of the scene that capture its semantics (e.g., the position and orientation of a car, its model and color, the time of day, weather, etc.). We then represent the semantic feature space using a program in a domain-specific programming language – hence the term *programmatic*. Given such a representation and generated data corresponding to correct and incorrect behaviors of an object detector, we seek to compute specializations of the program corresponding to those correct/incorrect behaviors. The specialized programs serve as interpretable representations of environment scenes that result in those correct/incorrect behaviors, enabling us to debug failure cases and to understand where the object detector succeeds.

We implement our approach using the SCENIC [2, 9] probabilistic programming language. Probabilistic programming has already been demonstrated to be applicable to various computer vision tasks (see, e.g., [14]). SCENIC is a domain-specific language used to model semantic feature spaces, i.e., distributions over scenes. It has a generative back-end that allows one to automatically produce synthetic data when it is connected to a renderer or simulator, such as the Grand Theft Auto V (GTA-V) video game. It is thus a particularly good fit for our approach. Using SCENIC, we implement the workflow shown in Fig. 1. We begin with a SCENIC program P that captures a distribution that we would like our DNN-based detector to work on. Generating test data from P, we evaluate the performance of the detector, partitioning the test set into correct and incorrect detections. For each partition, we use a rule extraction algorithm to generate rules over the semantic features that are highly correlated with successes/failures of the detector. Rule extraction is performed using decision tree learning and anchors [22]. We further propose a novel white-box approach that analyzes the neuron activation patterns of the neural network to get insights into its inner workings. Using these activation patterns, we show how to derive semantically understandable rules over the high-level input features to characterize scenarios.

The generated rules are then used to refine P yielding programs P^+ and P^- that characterize more precisely the correct and incorrect feature spaces, respectively. Using this framework, we evaluate DNN-based object detector for autonomous vehicles, using data generated using SCENIC and GTA-V. We demonstrate that our approach is very effective, producing rules and refined programs that significantly increase the correct detection rate (from 65.3% to 89.4%) and incorrect detection rate (from 34.7% to 87.2%) of the network and can thus help with understanding, debugging and retraining the network.

In summary, we make the following contributions:

- Formulation of a programming language-based semantic framework to characterize success/failure scenarios for an ML-based perception module as programs that help delineate its performance boundaries and generate new data in a principled way;
- An approach based on anchors and decision tree learning for deriving rules for refining scenario programs;
- A novel white-box technique that uses activation patterns of convolutional neural networks to enhance scenario feature space refinement;
- A data generation platform enabling research into debugging and explaining DNN-based perception, and
- Experimental results demonstrating that our framework is effective for a complex convolutional neural network

Feature	Range
Weather	Neutral, Clear, Extrasunny, Smog, Clouds, Overcast, Rain, Thunder, Clearing, Xmas, Foggy,Snowlight, Blizzard, Snow
Time	[00:00, 24:00)
Car Model	Blista, Bus, Ninef, Asea, Baller, Bison, Buffalo, Bobcatxl, Dominator, Granger, Jackal, Oracle, Patriot, Pranger
Car Color	R = [0, 255], G = [0, 255], B =[0, 255]
Car Heading	[0, 360) deg
Car Position	Anywhere on a road on GTA-V's map

Table 1: Environment features and their ranges in GTA-V

```
1 param time = (6*60, 18*60)
2 ego = Car at -209.091 @ -686.231
3
4 spot = OrientedPoint on visible curb
5 badAngle = (-90,90) deg
6
7 otherCar = Car at spot,
8               facing badAngle relative to ego.heading
9
10 require otherCar in ego.visibleRegion
11 require ((angle to otherCar) - ego.heading) < 0
12 require (distance from ego.position to otherCar) >= 5
13 require (distance from ego.position to otherCar) <= 20
```

Figure 2: Example SCENIC program

used in autonomous driving.

2. Background

SCENIC is a probabilistic programming language for scenario specification and scene generation. The language can be used to describe *environments* for various autonomous systems such as autonomous cars or robots. The environments are *scenes*, i.e. configurations of objects and agents. SCENIC allows assigning distributions to the *features* of the scenes, as well as hard and soft mathematical constraints over the features in the scene. Generating scenes from a SCENIC program requires sampling from the distributions defined in the program. SCENIC comes with efficient sampling techniques that take advantage of the structure of the SCENIC program, to perform sampling efficiently, using aggressive pruning of the sampling space. The generated scenes are rendered into images with the help of a simulator. In this paper (and similar to [9]) we use the Grand Theft Auto V (GTA-V) game engine [10] to create realistic images with a case study that uses SqueezeDet [30], a convolutional neural network for object detection in autonomous cars. Note that the framework we put forth is not specific to this network, and can be used with other object detectors as well.

The semantic features that we use in our case study are described in Table 1. These features are determined and limited by the environment parameters that the simulator allows users to control. If distributions over these environment features are not specified in a SCENIC program, then, by default, they are uniformly randomly selected from ranges shown in Table 1. Note that for a different application domain, we would have a different set of features.

SCENIC is designed to be easily understandable, with simple and intuitive syntax. We illustrate it via an example, shown in Figure 2. The formal syntax and semantics can be found in [9].

As shown in Figure 2, the program describes a rare situation where a car is illegally intruding over a white striped traffic island to either cut in or belatedly avoid entering elevated highway. In line 1, "param time = (6*60, 18*60)" means that time of the day is uniformly randomly sampled from 6:00 to 18:00. In line 2, an ego car is placed at specific x @ y coordinate on GTA-V's map. In line 4, a spot on a traffic island (in SCENIC, we referred to it as a curb) that is within a visible region from a camera mounted on ego car is selected. Of all visible region of the traffic island, a spot is uniformly randomly sampled. In line 7 and 8, otherCar is placed on the spot facing -90 to 90 degree off of where ego car is facing, simulating cases when a car may be protruding into a traffic flow. Lastly, SCENIC allows users to define hard and soft constraints using *require* statements. In this scenario, all four require statements define hard constraints. In line 10, the entire surface of the otherCar must be within the view region of the ego car. So, a scene where only front half of the otherCar is visible is not allowed. In line 11, the otherCar must be positioned in the right half of the ego car's visible region. In line 12 and 13, the distance of the otherCar from ego car should be 5 to 20 meters.

3. Related Work

Most techniques that aim to provide explainability and interpretability for deep neural networks (DNNs) in the field of computer vision focus on attributing the network's decisions to portions of the input images([16, 19, 25, 26, 31]). GradCAM [23] is a popular approach for interpreting CNN models that visualizes how parts of the image affect the neural network's output by looking into class activation maps (CAM). Other techniques focus on understanding the internal layers by visualizing their activation patterns [5, 17]. Our approach, on the other hand, aims to provide characterizations at a higher level than raw image pixels, namely at the level of abstract features defined in a SCENIC program.

Rule extraction techniques either aim to represent the entire functionality of the network as a set of rules making it too complex [32] or require the presence of pre-mined set of rules [15] which would be difficult to obtain for the object detection scenario. Anchors [22], which improves on LIME [21], is closest to our work (and we discuss it in more detail later).

Recent work aims to explain the decisions of DNNs in

terms of higher-level concepts. The technique in [13] introduces the idea of concept activation vectors, which provide an interpretation of a neural network's internal state in terms of human-friendly concepts. Feature Guided Exploration [29] aims to analyze the robustness of networks used in computer vision applications by applying perturbations over high-level input features extracted from raw images. They use object detection techniques (such as SIFT – Scale Invariant Feature Transform) to extract the features from an image. In contrast to these techniques we directly leverage SCENIC which defines the high-level features in a way that is already understandable for humans. Existing approaches typically use classification networks whose output directly corresponds to the decision being made and rely on the derivative of the output with respect to the input to calculate importance. In our application, there is no direct correlation between the output of the object detector network and the validity of the bounding boxes. Furthermore, unlike all previous work, we can use the synthesized rules to automatically generate more input instances, by refining the original SCENIC program and then using it to generate data. These instances can be used to test, debug and retrain the network.

4. Approach

The key idea of our approach is to leverage the high-level semantic features formally encoded in a SCENIC program to derive rules (sufficient conditions) that explain the behavior of a detection module in terms of those features. Our hypothesis is that since these features describe the important characteristics that should be present in an image and furthermore they are much fewer than the raw, low-level pixels, they should lead to small, compact rules that have a clear meaning for the developer.

The problem that our technique aims to address can be formalized as follows. Suppose a function g defines a mapping from a feature vector, $[f_1, f_2, ..., f_n] \in D_1 \times D_2 \times ... \times D_n$, to a matrix of pixels, $m \in M$, of an image, where each D_i represents the feature domain of feature f_i and M is a domain of m. Let function h denote the given perception module. Finally, let e be an evaluation function which compares the perception module's prediction to the ground truths, and outputs a boolean class (correct or incorrect) based on a certain performance threshold. Given a SCENIC program, according to its feature dependencies and hard and soft constraints, the feature space, $D_1 \times D_2 \times ... \times D_n$, is defined. The problem is to find the subset feature space, $d_1 \times d_2 \times ... \times d_n \subseteq D_1 \times D_2 \times ... \times D_n$ such that when we sample a certain number of features $[f_1, f_2, ..., f_n] \in d_1 \times d_2 \times ... \times d_n$, the probability that $e(h(g([f_1, f_2, ..., f_n])))$ is equal to a target class (correct or incorrect) is *maximized*.

A high-level overview of our analysis pipeline is illustrated in Figure 3. We start with a SCENIC program that encodes constraints (and distributions) over high-level semantic features that are relevant for a particular application domain, in our case object detection for autonomous driving. Intuitively, the program (henceforth called scenario) encodes the environments that the user wants to focus on in order to test the module. Based on this scenario, SCENIC generates a set of *feature vectors* by sampling from the specified distributions. A simulator is then used to generate a set of realistic, synthetic images (i.e. raw low-level pixel values) based on those features.

The images are fed to the object detector. Each image is assigned a binary label, correct or incorrect, based on the performance of the object detector on the image (see Section 4.1). The labels obtained for the images are mapped back to the feature vectors that led to the generation of the respective images. The result is a labeled data set that maps each high-level feature vector to the the respective label.

We then use off-the-shelf methods to extract *rules* from this data set. The rule extraction is described in more detail in Sec. 4.2. The result is a set of rules encoding the conditions on high-level features that lead to likely correct or incorrect detection. The obtained rules can be used to *refine* the SCENIC program, which in turn can be sampled to generate more images that can be used to test, debug or retrain the detection module. This iterative process can continue until one obtains refined rules, and SCENIC programs, of desired precision. In the following we provide more details about our approach.

4.1. Labelling

Obtaining the label (correct/incorrect) for an image is performed using the F1 score metric (harmonic mean of the precision and recall). This metric is commonly used in statistical analysis of binary classification. The F1 score is computed in the following way. For each image, the true positive (TP) is the number of ground truth bounding boxes correctly predicted by the detection module. Correctly predicted here means intersection-over-union (IoU for object detection) is greater than 0.5. The false positive (FP) is the number of predicted bounding boxes that falsely predicted ground truths. This false prediction includes duplicate predictions on one ground truth box. The false negative (FN) is the number of ground truth boxes that is not detected correctly. We computed the F1 score for each image, and if it is greater than a threshold, we assigned *correct* label; if not, *incorrect*. The threshold used in our experiments was 0.8.

4.2. Rule Extraction

Methods: We experimented with two methods, decision tree (DT) learning for classification [20] and anchors [22], to extract rules capturing the subspace of the feature space defined in the given SCENIC program.

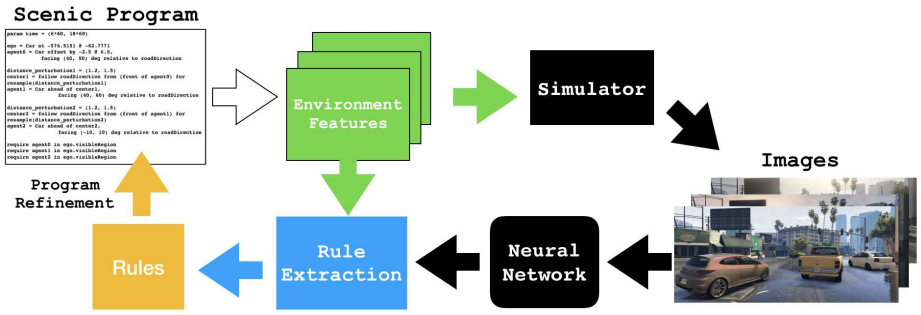

Figure 3: Analysis Pipeline

Decision tree learning is commonly used to extract rules explaining the *global* behavior of a complex system while the anchors method is a state-of-the art technique for extracting explanation rules that are *locally* faithful.

Decision trees encode decisions (and their consequences) in a tree-like structure. They are highly interpretable, provided that the trees are short. One can easily extract rules for explaining different classes, by simply following the paths through the trees and conjuncting the decisions encoded in the tree nodes. We used the `rpart` [27] package in R software, which implements corresponding algorithm in [4], with default parameters.

The anchor method is a state-of-the art technique that aims to explain the behavior of complex ML models with high-precision rules called anchors, representing local, sufficient conditions for predictions. The system can efficiently compute these explanations for any black-box model with high-probability guarantees. We used the code from [1] with the default parameters. Applying the method to the object detector directly would result in anchors describing conditions on low-level pixel values, which would be difficult to interpret and use. Instead what we want is to extract anchors in terms of high-level features. While one can use the simulator together with the object detector as the black-box model, this would be very inefficient. Instead we built a surrogate model mapping high-level SCENIC features to output labels; we used a random forest learning algorithm for this purpose as in the code. This surrogate model was then passed to the anchor method to extract the rules.

Blackbox vs Whitebox Analysis: So far we explained how we can obtain rules when treating the detection module as a black box. We also investigated a white-box analysis, to determine whether we can exploit the information about the internal workings of the module to improve the rule inference. The white-box analysis is one of our novel contributions in this paper. We leverage recent work [12] which aims to infer likely properties of neural networks. The properties are in terms of on/off activation patterns (at different internal layers) that lead to the same predictions. These patterns are computed by applying decision-tree learning over the activations observed during the execution of the network on the training or testing set.

We analyzed the architecture of the SqueezeDet network and we determined that there are three maxpool layers which provide a natural decomposition of the network. Furthermore they have relatively low dimensionality making them a good target for property inference.

We consider activation patterns over maxpool neurons based on whether the neuron output is greater or equal to zero. A decision tree can then be learned over these patterns to fit the prediction labels. For our experiments we selected patterns from the maxpool layer 5, which turned out to be highly correlated to images that lead to correct/incorrect predictions.

Then, we augmented the assigned correct and incorrect labels with corresponding decision pattern in the following way. For example, using a decision pattern for correct labels (i.e. the decision pattern that most correlated to images with correct label), we created two sub-classes for correct class. By feeding in only images with correct label to the perception module, the images satisfying the decision pattern is re-labelled as "correct-decision-pattern," otherwise, "correct-unlabelled." Likewise, the incorrect class is augmented using a decision pattern that is most correlated to images with incorrect label. It is our intuition that the decision pattern captures more focused properties (or rules) among images belonging to a target class. Hence, we hypothesize that this label augmentation would help anchor and decision tree methods to better identify rules.

Rule Selection Criteria: Once we extracted rules with either DT or anchors, we selected the best rule using following criteria. To best achieve our objective, first, we chose the rule with highest precision on a held-out testset of feature vectors. If there are more than one rule with equal high precision, then we chose the rule with the highest coverage (i.e. the number of feature vectors satisfying the rules). Finally, if there is still more than one rule left, then we broke the tie by choosing the most compact rule which has the

Scenario # (Baseline→Rule Precision)	Rules
Scenario 1 (65.3% → 89.4%)	**x coordinate** ⩾ -198.1
Scenario 2 (72.3% → 82.3%)	**hour** ⩾ 7.5 ∧ **weather** = all except neutral ∧ **car0 distance from ego** ⩾ 11.3m ∧ **car0 model** = {Asea, Bison, Blista, Buffalo, Dominator, Jackal, Ninef, Oracle}
Scenario 3 (61.7% → 79.4%)	**car0 red color** ⩾ 74.5 ∧ **car0 heading** ⩾ 220.3 deg
Scenario 4 (89.6% → 96.2%)	**car0 model** = {Asea, Baller, Blista, Buffal, Dominator, Jackal, Ninef, Oracle}

Table 2: Rules for correct behaviors of the detection module with the highest precision from Table 6

Scenario # (Baseline→Rule Precision)	Rules
Scenario 1 (34.7% → 87.2%)	**x coordinate** ⩽ -200.76 ∧ **distance** ⩽ 8.84 ∧ **car model** = PRANGER
Scenario 2 (27.7% → 44.9%)	**hour** ⩾ 7.5 ∧ **weather** = all except Neutral ∧ **car0 distance from ego** < 11.3
Scenario 3 (38.3% → 83.4%)	**weather** = neutral ∧ **agent0 heading =** ⩽ 218.08 deg ∧ **hour** ⩽ 8.00 ∧ **car2 red color** ⩽ 95.00
Scenario 4 (10.4% → 57.3%)	**car0 model** = PATRIOT ∧ **car1 model** = NINEF ∧ **car2 model** = BALLER ∧ 92.25 < **car0 green color** ⩽ 158 ∧ **car0 blue color** ⩽ 84.25 ∧ 178.00 < **car2 red color** ⩽ 224

Table 3: Rules for incorrect behaviors of detection module with the highest precision from Table 7

least number of features. The last two criteria are established to select the most general high-precision rule.

5. Experiments

In this section we report on our experiments with the proposed approach on the object detector. We investigate whether we can synthesize rules that are effective in generating test inputs that increase the probability of correct/incorrect detection, thus explaining the correct/incorrect behavior of the analyzed module. We evaluate the proposed techniques along the following dimensions: decision tree (DT) vs anchor, black-box (BB) vs white-box (WB).

5.1. Scenarios

We experimented with our approach on four different scenarios. Images generated from these scenarios are shown in Figure 4. Scenario 1 (Figure 2) describes the situation where a car is illegally intruding over a white striped traffic island at the entrance of an elevated highway. Scenario 2 describes two-car scenario where one car occludes the ego car's view of another car at a T-junction intersection on an elevated road. describes scenes where other cars are merging into ego car's lane. The location in this scenario is carefully chosen such that the sun rises in front of ego car, causing a glare. describes a set of scenes when nearest car is abruptly switching into ego car's lane while another car on the opposite traffic direction lane is slightly intruding over the middle yellow line into ego car's lane. [1]

5.2. Setup

The object detector was trained on a separate set of 10,000 GTA images with one to four cars in various locations of the map producing different background scenes. The GTA-V simulator provided images, ground truth boxes, and values of the environment features.

For each scenario, we generated 950 new images as a train set and another 950 new images as a test set. We denote the labels corresponding to the maxpool layer 5 decision pattern as p5c(correct) and p5_ic(incorrect) and the remaining as correct_unlabelled and incorrect_unlabelled, respectively. We augmented the feature vector with some extra features that are not part of the feature values provided by the simulator but could help with extracting meaningful rules. For example, in Scenario 1, the distance from ego to otherCar is not part of the feature values provided by GTA-V. However, it can be computed with Euclidean distance metric using (x,y) location coordinates of ego and otherCar. Also, the difference in heading angle between ego and otherCar is also added as extra feature to represent "badAngle" variable in the program.

From the train set, we extracted rules to predict each label based on the feature vectors. These rules were evaluated on the test set based on precision, recall, and F1 score metrics. For DT learning we adjusted the label weight to account for the uneven ratio among labels for both black-box and white-box labels. For the Anchors method, we applied it on each instance of the training set until we had covered a maximum of 50 instances for every label (correct, incorrect for Black Box, and p5c, p5_ic, correct_unlabelled, incorrect_unlabelled for White Box). The best anchor rule for every label is selected based on the rule selection criteria mentioned in section 4.2.

[1] Please refer to supplement material for SCENIC programs of scenario 2,3, and 4 as well as refined SCENIC programs

Figure 4: From top-left one-car image, each image corresponds to scenario 1, 2, 3, and 4 in a clockwise manner. The scenario number is the number of cars

5.3. Results

Tables 2 and 3 show the best rules (wrt. precision) extracted with our proposed framework, along with the baseline correct/incorrect detection rate for each given scenario and the detection rate for the generated rules. The results indicate that indeed our framework can generate rules that increase significantly the correct and incorrect detection rate of the module. Furthermore, the generated rules are compact and easily interpretable.

For example, the rule for correct behavior for Scenario 1 is "x coordinate $\geqslant -198.1$." In GTA-V, at ego car's specific location, the condition on x coordinate was equivalent to the otherCar's distance from ego being greater than 11m. On the other hand, the rule for incorrect behavior for Scenario 1 requires the otherCar to be within 8.84m and its car model to be PRANGER. These rules, counter-intuitively, indicate that the object detector fails when the otherCar is close by, and performs well when located further away.

Results for Correct Behavior: Tables 5 and 6 summarize the results for the rules explaining correct behavior. The results indicate that there are clear signals in the heavily abstracted feature space and they can be used effectively for scenario characterization via the generated high-precision rules.

The results also indicate that DT learning extracts rules with better F1 scores for all scenarios as compared to anchors. This could be attributed to the difference in the nature of the techniques. The anchor approach aims to construct rules that have high precision in the locality of a given instance. Decision-trees on the other hand aim to construct global rules that discriminate one label from another. Given that a large proportion of instances were detected correctly by the analyzed module, the decision tree was able to build rules with high precision and coverage for correct behavior.

Scenario #	1	2	3	4
Correct DP	0.626	0.651	0.514	0.824
Incorrect DP	0.276	0.175	0.234	0.212

Table 4: Support for correct and incorrect decision patterns

Scenario #	1	2	3	4
BB Decision Tree	0.723	0.342	0.631	0.622
WB Decision Tree	0.727	0.696	0.601	0.778
BB Anchor	0.361	0.457	0.302	0.438
WB Anchor	0.520	0.188	0.149	0.438

Table 5: F1 score of correct rules on testset

Scenario #	1	2	3	4
Original Program	0.653	0.723	0.617	0.896
BB Decision Tree	0.843	0.778	0.787	0.950
WB Decision Tree	0.826	0.823	0.788	0.962
BB Anchor	0.727	0.811	0.652	0.928
WB Anchor	0.894	0.817	0.794	0.928

Table 6: Precision of correct rules on the testset

Scenario #	1	2	3	4
Original Program	0.347	0.277	0.383	0.104
BB Decision Tree	0.703	0.418	0.506	0.375
WB Decision Tree	0.73	0.449	0.494	0.099
BB Anchor	0.872	0.357	0.834	0.573
WB Anchor	0.674	0.422	0.365	0.176

Table 7: Precision of incorrect rules on 500 new data generated from each refined SCENIC program

The results also highlight the benefit of using white-box information to extract rules for correct behavior.

Table 4 shows the support for the decision pattern is significant (greater than 65% on average for all scenarios). The support is defined as a correlation of the decision pattern to a specific label. Using this information to augment the labels of the dataset helped to improve the precision and F1 score of the rules (w.r.t. SCENIC features) for both DT learning and anchor method.

Results for Incorrect Behavior: Tables 3 and 7 summarize the results for the rules explaining incorrect behavior. Rule derivation for incorrect behavior is more challenging than for correct behavior due to the low percentage of inputs that lead to the incorrect detection for a well trained network.

In fact the F1 scores (computed on the test set) for rules

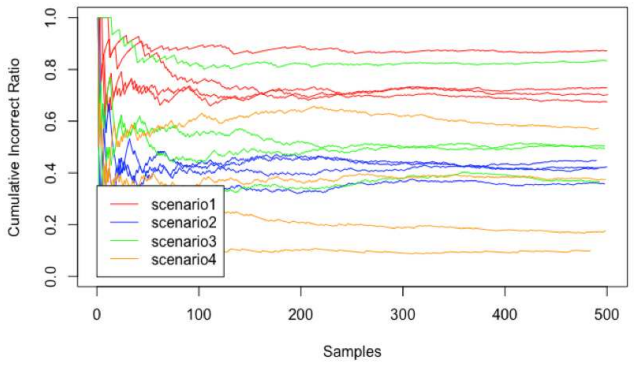

Figure 5: The cumulative ratio of incorrectly detected images generated from refined SCENIC programs (using incorrect rules) stabilizes over 500 samples. Each color has four graphs representing four different rule extraction methods

predicting incorrect behavior were too low due to very low (in some cases 0) recall values.

To properly validate the efficacy of the generated rules, we refined the SCENIC programs by encoding the rules as constraints and we generated 500 new images. We then evaluated our module's performance on these new datasets. Figure 5 justifies our choice of 500 as the number of new images that we generate for evaluation.

All four methods contributed to more precisely identifying the subset features spaces in which the module performs worse. Specifically, Table 7 illustrates that the black-box anchor method enhanced the generation rate of incorrectly detected images by 48% on average in Scenarios 1, 3, and 4 compared to the baseline. This is a significant increase in the ratio of incorrectly labelled images generated from the program, providing evidence that the refined programs are more precisely characterizing the failure scenarios.

We also note that the anchor method outperforms DT learning. This is expected, because the anchor method extracts rules that are highly precise within a local feature space. The exception is Scenario 2. We conjecture that the reason that the anchor method did not perform better than DT learning is due to uncontrollable non-determinism in GTA-V, which generated pedestrians in close vicinity to the camera of ego car even though its SCENIC program did not have any pedestrian. GTA-V non-deterministically instantiated these pedestrians, and the perception module often incorrectly predicted the pedestrians as cars. This is an issue with the GTA-V which originally was not built for data generation purpose. GTA-V does not allow users to control or eliminate these pedestrians and it does not provide features related to pedestrians during data collection process. In future work, we plan to incorporate simulators that allows a deterministic control (such as CARLA [7]) for further experimentation.

Unlike the results for correct behavior, the whitebox approach tends to perform worse than blackbox when focusing on incorrect behavior. This outcome can be attributed to very low support for decision patterns computed for incorrect behavior, with maximum of 27.6% among the four scenarios as shown in Table 4.

However, we do observe that the white-box approach for both DT learning and anchors does, in general, enhance the ratio of incorrectly detected images as shown in Table 7, compared to those of the original programs.

Limitations: Our technique relies on abstracting an image with a high resolution (for instance 1920 x 1200 in our example) to a vector of a small set of semantic features. In our experiments we were able to derive compact rules with high precision and coverage. However, we do note that in other application domains, other than autonomous driving, the abstraction may lead to under-determined representation, which may not yield any noticeable patterns. Therefore, appropriate selection of a subset of essential features for a given application domain (facilitated by an appropriate definition using SCENIC), is essential. We also note that all the SCENIC programs we experimented with contained only uniform distributions. Also, for each of the scenario programs that we analyzed, we fixed the location and heading angle of the camera. In these restricted settings, we were able to extract rules that distinguished correctly detected scenes from the incorrect ones.

6. Conclusion and Future Work

We presented a semantic and programmatic framework for characterizing success and failure scenarios of a given perception module in the form of programs. The technique leverages the SCENIC language to derive rules in terms of high-level, meaningful features and generates new inputs that conform with these rules. For future work, we plan on applying this approach to other domains, by looking into more general input distributions and transformations.

7. Acknowledgment

We thank Daniel Fremont for his help on our use of SCENIC, Jinkyu Kim and Taesung Park for their thorough comments, and Xiangyu Yue for interfacing GTA-V with SCENIC. This project is supported by an NSF graduate fellowship (Grant#: DGE1752814), NSF grants CNS-1545126 (VeHICaL), CNS-1739816, and CCF-1837132, by the DARPA Assured Autonomy program, by Berkeley Deep Drive, and by Toyota under the iCyPhy center. This work was partially done under the NASA Ames Internship Program, 2019.

References

[1] Anchor Method Repository. https://github.com/marcotcr/anchor. 5

[2] The SCENIC Probabilistic Programming Language. https://github.com/BerkeleyLearnVerify/Scenic. 2

[3] Babak Alipanahi, Andrew Delong, Matthew T Weirauch, and Brendan J Frey. Predicting the sequence specificities of DNA-and RNA-binding proteins by deep learning. *Nature biotechnology*, 2015. 2

[4] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984. 5

[5] Shan Carter, Zan Armstrong, Ludwig Schubert, Ian Johnson, and Chris Olah. Activation atlas. *Distill*, 2019. https://distill.pub/2019/activation-atlas. 2, 3

[6] George E Dahl, Jack W Stokes, Li Deng, and Dong Yu. Large-scale malware classification using random projections and neural networks. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3422–3426. IEEE, 2013. 2

[7] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017. 8

[8] Tommaso Dreossi, Alexandre Donze, and Sanjit A. Seshia. Compositional falsification of cyber-physical systems with machine learning components. In *Proceedings of the NASA Formal Methods Conference (NFM)*, pages 357–372, May 2017. 2

[9] Daniel J. Fremont, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. Scenic: A language for scenario specification and scene generation. In *Proceedings of the 40th annual ACM SIGPLAN conference on Programming Language Design and Implementation (PLDI)*, June 2019. 2, 3

[10] Rockstar Games. Grand theft auto v. Windows PC version, 2015. 3

[11] Ian Goodfellow, Patrick McDaniel, and Nicolas Papernot. Making machine learning robust against adversarial inputs. *Communications of the ACM*, 61(7):56–66, 2018. 2

[12] Divya Gopinath, Hayes Converse, Corina S. Pasareanu, and Ankur Taly. Property inference for neural networks. *CoRR*, abs/1904.13215, 2019. 5

[13] Been Kim, Martin Wattenberg, Justin Gilmer, Carrie J. Cai, James Wexler, Fernanda B. Viégas, and Rory Sayres. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). In Jennifer G. Dy and Andreas Krause, editors, *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 2673–2682. PMLR, 2018. 4

[14] Tejas D Kulkarni, Pushmeet Kohli, Joshua B Tenenbaum, and Vikash Mansinghka. Picture: A probabilistic programming language for scene perception. In *Proceedings of the ieee conference on computer vision and pattern recognition*, pages 4390–4399, 2015. 2

[15] Himabindu Lakkaraju, Stephen H. Bach, and Jure Leskovec. Interpretable decision sets: A joint framework for description and prediction. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 1675–1684, 2016. 3

[16] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc., 2017. 2, 3

[17] Alexander Mordvintsev, Michael Tyka, and Christopher Olah. DeepDream,https://github.com/google/deepdream. 2, 3

[18] NVIDIA. Nvidia tegra drive px: Self-driving car computer, 2015. 2

[19] Zhongang Qi, Saeed Khorram, and Fuxin Li. Visualizing deep networks by optimizing with integrated gradients. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2019. 2, 3

[20] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986. 4

[21] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 1135–1144, 2016. 3

[22] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2-7, 2018*, pages 1527–1535, 2018. 2, 3, 4

[23] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 618–626, 2017. 2, 3

[24] Sanjit A. Seshia, Dorsa Sadigh, and S. Shankar Sastry. Towards Verified Artificial Intelligence. *ArXiv e-prints*, July 2016. 2

[25] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda B. Viégas, and Martin Wattenberg. Smoothgrad: removing noise by adding noise. *CoRR*, abs/1706.03825, 2017. 2, 3

[26] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. *CoRR*, abs/1703.01365, 2017. 2, 3

[27] Terry Therneau, Beth Atkinson, and Brian Ripley. rpart: Recursive partitioning and regression trees. 5

[28] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. Deeptest: automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, pages 303–314, 2018. 2

[29] Matthew Wicker, Xiaowei Huang, and Marta Kwiatkowska. Feature-guided black-box safety testing of deep neural networks. In *Tools and Algorithms for the Construction and Analysis of Systems - 24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Part I*, pages 408–426, 2018. 4

[30] Bichen Wu, Forrest N. Iandola, Peter H. Jin, and Kurt Keutzer. Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. In *Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops*, pages 446–454, 2017. 3

[31] Bolei Zhou, David Bau, Aude Oliva, and Antonio Torralba. Interpreting deep visual representations via network dissection. *CoRR*, abs/1711.05611, 2017. 2, 3

[32] Jan Zilke, Eneldo Mencía, and Frederik Janssen. Deepred – rule extraction from deep neural networks. pages 457–473, 10 2016. 3