Case Study: Analysis of Autonomous Center line Tracking Neural Networks

Ismet Burak Kadron
kadron@cs.ucsb.edu
University of California Santa Barbara, USA

Divya Gopinath
divya.gopinath@nasa.gov
KBR Inc., NASA Ames

Corina S. Păsăreanu
corina.s.pasareanu@nasa.gov
KBR Inc., CMU, NASA Ames

Huafeng Yu
huafeng.yu@boeing.com
Boeing Research & Technology

***Abstract*—Deep neural networks have gained widespread usage in a number of applications. However, limitations such as lack of explainability and robustness inhibit building trust in their behavior, which is crucial in safety critical applications such as autonomous driving. Therefore, techniques which aid in understanding and providing guarantees for neural network behavior are the need of the hour. In this paper, we present a case study applying a recently proposed technique, Prophecy, to analyze the behavior of a neural network model, provided by our industry partner and used for autonomous guiding of airplanes on taxi runways. This regression model takes as input an image of the runway and produces two outputs, cross-track error and heading error, which represent the position of the plane relative to the center line. We use the Prophecy tool to extract neuron activation patterns for the correctness and safety properties of the model. We show the use of these patterns to identify features of the input that explain correct and incorrect behavior. We also use the patterns to provide guarantees of consistent behavior. We explore a novel idea of using sequences of images (instead of single images) to obtain good explanations and identify regions of consistent behavior.**

I. Introduction

Deep neural networks (DNNs) are increasingly impacting every aspect of our lives, by being used in many applications, some of them with safety-critical requirements, such as autonomous driving and flight. However, it is not well understood why a network gives a particular output which is essential for building trust in its behavior. Further, it is also crucial to obtain guarantees of consistent behavior of the network, specifically for safety-critical applications. There is thus a critical need for tools and techniques that can help analyze and understand neural network models. In this paper we report on our experience with applying a recent advancement in property inference for neural networks, i.e. Prophecy [1], to the analysis and understanding of neural networks used for autonomous guiding of airplanes on taxi runways.

The Autonomous Center line Tracking (ACT) neural network [2], [3] is designed to take a single picture of the runway as input and return the plane's position with respect to the middle of the runway. It returns two numerical outputs; cross track error (y_0), which is the distance of the plane from the center line and heading error (y_1), which is the angle of the plane with respect to the center line. These outputs are typically fed to a controller which in turn manoeuvres the plane such that it remains close to the center of the runway. This forms a closed loop system wherein the ACT module continuously receives images in a sequence as the plane moves on the runway. ACT neural network models are typically trained and tested using the X-Plane simulator [4] that generates images at regular time intervals.

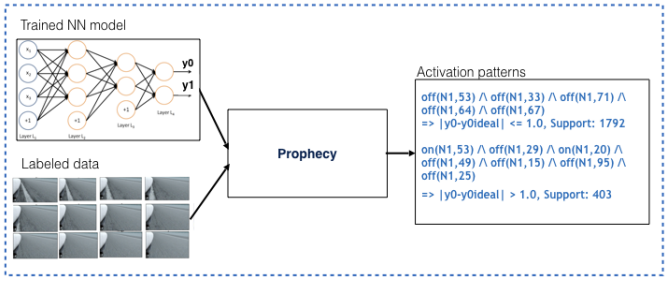

Fig. 1. Overview of Prophecy applied to ACT.

In this study, our goal is to understand (explain) the behavior of the ACT neural network model and provide guarantees for consistent behavior with respect to two types of output properties, indicated by the industrial partner. i) **Correctness properties** specify the conditions for correct behavior of the model using error bounds, by comparing the model outputs to ideal values; they have the following form: $|y_0 - y_{0ideal}| \leq 1.0m, |y_1 - y_{1ideal}| \leq 5$ *degrees*. ii) **Safety properties** specify conditions for safe operation by using runway dimensions; they have the following form: $|y_0| \leq 10.0m, |y_1| \leq 90$ *degrees*. To address these goals, we use Prophecy, which infers precondition properties from a deep neural network model with respect to its output behavior in terms of neuron activation patterns (see Figure 1). Our previous work [1] shows the application of Prophecy to classification models, for explaining perception models, proving behavioral properties and also using it at runtime to reduce inference time for classification. This work presents the first application of the tool on regression models for explaining correct and incorrect

behaviors and for providing guarantees of consistent behavior with respect to output properties.

Most of the existing techniques for attribution, such as DeepLift [5], LIME [6], and SHAP [7], identify important pixels in an individual image impacting the output of a model and focus on classification tasks. In this work, we show how to leverage activation patterns to identify input features that impact network behavior with respect to output constraints for a regression model. Furthermore, we investigate the idea of leveraging sequences of images satisfying the same pattern to visualize features that impact network behavior across multiple similar images, thus providing more useful information for the developers.

Most existing neural network verification techniques have focused on checking local adversarial robustness for perception networks (e.g., Marabou [8]) and proving input-output properties for controller networks with low dimensional inputs [9]. In our work, we leverage activation patterns to identify regions of images that lead to consistent output behavior, which are potentially bigger than local neighborhoods. We identify image sequences satisfying the same pattern and leverage off-the-shelf solvers to prove that the network behaves consistently in the input region containing the sequence.

The activation patterns group together input sequences over which the output property acts as a *temporal invariant* for the network. Furthermore, counterexamples obtained from failed proofs highlight problematic scenarios, such as the plane going out of the runway. These scenarios can be used to debug and improve the network.

II. Background

A. Deep Neural Networks

A deep neural network (DNN) (Figure 2) is a machine learning model made up of multiple layers of neurons, which when given an input such as an image produces a prediction of the respective outputs. Each neuron takes in a weighted sum of the outputs from the previous layer and applies an activation function on it. The rectified Linear Unit (ReLU) is a popular function that returns the value of the weighted sum as is if it is positive and returns 0 otherwise. The respective neuron is said to be OFF when its output is 0 and ON otherwise.

B. Prophecy

Prophecy [1] is a recently proposed technique that aims to analyze a complex DNN model to extract a set of compact properties. Each property is a rule of the form Pre $\Rightarrow$ Post, where Pre is a constraint in terms of (on/off) neuron activation patterns and Post is a constraint on the output of the network. An example of such a property is shown in Figure 2, which states that for any input to the network, if the activation of the first neuron in layer 1 ($N_{1,0}$) is off and the activation of the second neuron ($N_{1,1}$) is on, then the output y_1 is greater than the output y_0. Prophecy extracts these properties by applying decision tree learning over the activation patterns recorded for a given set of labeled data for the DNN model. The properties are proven using an off-the-shelf verification

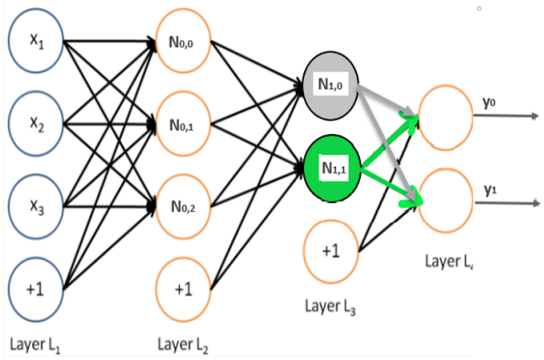

Fig. 2. Example DNN with ReLU activation and example layer property that can be extracted by the Prophecy; off($N_{1,0}$) $\wedge$ on($N_{1,1}$) $\Rightarrow$ $y_1 > y_0$

tool such as Marabou [8]. Each such property is associated with a support, which indicates the number of data instances that satisfy the rule. This information can be used to assess the extracted rules, in cases they can not be proved formally.

C. GradCAM++

GradCAM++ [10] is a recently proposed approach for explaining the decisions of convolutional neural network models used for image classification. It aims to generate class activation maps that highlight pixels of the image the model used to make the classification decision. It builds on the basic idea proposed in [11] of using the gradients of any target concept flowing into the final convolutional layer to produce a coarse localization map highlighting the important regions in the image for predicting the concept. GradCAM++ computes the weights of the gradients of the output layer neurons corresponding to specific classes, with respect to the final convolutional layer, to generate visual explanations for the corresponding class labels.

D. Neural Networks for Autonomous Center Line Tracking

In our case study we analyze neural networks built for Autonomous Center Line Tracking.

Centerline tracking on runway or taxiway is one of the most important ground operations in an airport. An airplane is required to follow the center lines of taxiway and runway during taxiing. Center lines have standardized shapes and colors on the pavement of airport runways and taxiways, but may be less visible for a number of reasons, including skid marks, poor lighting conditions and bad weather. Our industry partner developed a research prototype of machine learning component based on neural networks to identify center lines on runways/taxiways and predict the CTE (distance between the nose wheel and centerline) during taxi. This case study was demonstrated and solutions evaluated using the X-Plane simulator. We analyzed a model for Autonomous Center line Tracking supplied by our industry partner, which we denote as ACT. The model is a sequential convolutional neural network

model (CNN) with 24 layers including an input layer that takes in images of dimensions $360 \times 200 \times 3$, 5 convolution layers, and four dense layers at the end. The first dense layer has 100 neurons, the second has 50 neurons, the third has 10 neurons; the output layer has 2 neurons. Each neuron has an ELU (Exponential Linear Unit) activation function.

In addition to ACT, we analyzed TinyNet, which is a smaller network for autonomous center line tracking that is more amenable to formal verification. This network was proposed in [12]. The TinyNet neural network takes a single picture of the runway of size $360 \times 200 \times 3$, crops a segment of the image of size 205×135 to remove the nose of the plane from the image, make it black and white, reduce the size of the image to 256×128 and downsample it to dimensions 16×8 by taking the average brightness of 16 pixels with most brightness in each 16×16 pixel block. TinyNet has one input layer of dimension 128, three fully connected layers of sizes 16, 8, 8 with ReLU activation function and an output layer of size 2 with no activation function. The output layer returns the two outputs, the cross track error (y_0) and heading error (y_1).

III. Analysis of ACT and TinyNet

In this section, we describe the application of Prophecy to extract patterns from the ACT model supplied by our industry partner and also from TinyNet. We show how the patterns could be used for explaining the model behaviour and for providing guarantees of consistent behavior with respect to the desired correctness and safety properties.

A. Activation Patterns for ACT

To extract activation patterns from the ACT model, we used a dataset consisting of 13885 input images and their corresponding ideal outputs. We consider the following three correctness properties of the outputs and extract separate patterns from the inputs that satisfy or violate each of them:

- $|y_0 - y_{0ideal}| \leq 1.0$,
- $|y_1 - y_{1ideal}| \leq 5.0$, and
- $|y_0 - y_{0ideal}| \leq 1.0 \wedge |y_1 - y_{1ideal}| \leq 5.0$.

The patterns were extracted at each of the three dense layers (dense_1, dense_2, dense_3) and all the dense layers together. Each pattern is a rule in terms of the activations of the neurons in the respective layers. A neuron with the ELU activation function is considered to be off if its value is ≤ 0 and on otherwise. Patterns were generated for the satisfaction of the output property (denoted here as class 1) and their violation (denoted here as class 0). We obtained a total of 396 patterns for class 1 and 418 for class 0, with a minimum support of 10 instances satisfying a pattern.

Note that the model is coded to ensure that the outputs y_0 and y_1 do not have values beyond the runway dimensions for safety (10m and 90 degrees, respectively), therefore we could not extract patterns that discriminate between satisfaction and violation of the safety property on this model.

B. Using Patterns for Explaining ACT

Each layer in a neural network typically extracts and processes features of the input image. Each layer pattern thus potentially represents the logic of the network in terms of the input features which in turn impact the network's behavior with respect to the specific output property. Therefore, visualization of the features represented by layer patterns has the potential to identify the portion of the input that impacts correct or incorrect network behavior.

Here is an example pattern for the correct behavior extracted from the first dense layer of the ACT network model:
$\text{off}(N_{1,53}) \wedge \text{off}(N_{1,33}) \wedge \text{off}(N_{1,71}) \wedge \text{off}(N_{1,64}) \wedge \text{off}(N_{1,67}) \Rightarrow |y_0 - y_{0ideal}| \leq 1.0$.

The pattern states that for input images satisfying the neuron activations as prescribed in the pattern, the cross-track error output of the network would be close to the ideal. There were 1792 images (from the initial dataset used to extract the patterns) that satisfy the respective activations and also satisfy the output correctness property. We refer to this as the support of the pattern; note that the higher the value of the support, the higher is the confidence that the pattern would hold true for all possible inputs.

In order to use the pattern to explain the network behavior, we aim to visualize the features represented by the pattern. This can be done by identifying the pixels in the input that impact the neurons that appear in the rule representing the pattern. Attribution approaches such as the GradCAM++ (refer Section II-C) technique identify pixels in the input that impact the output neuron corresponding to a given label. We use this technique to identify pixels in the input (image satisfying the pattern) impacting the neurons in the pattern by making the following modifications. Instead of using GradCAM++ to generate class explanations, we use GradCAM++ to compute the partial derivatives of the last convolutional layer with respect to the neurons in the pattern and generate a visual explanation for the pattern.

The first picture in Figure 3 shows an image satisfying the example pattern for correct behavior. The picture in the center shows the pixels identified with the help of the pattern, which highlight the portion of the image between the nose of the plane and the center-line of the runway. This represents the relevant input features which the network appears to use to determine the cross-track error output, and corresponds to the expected behaviour according to the domain experts.

We note that existing attribution approaches for image classification networks, including GradCAM++, typically aim to identify pixels of an input image that directly impact the respective output variable of interest. This works for classification models, since the output nodes represent the labels and the pixels that increase the magnitude of the output node could be considered responsible for the decision. However, such an approach does not make too much sense for regression models. We did try to use GradCAM++ to highlight input pixels that directly impact the output y_0 for the given input image. However, as seen in the last picture in Figure 3, this seems to highlight most of the image and does not appear to

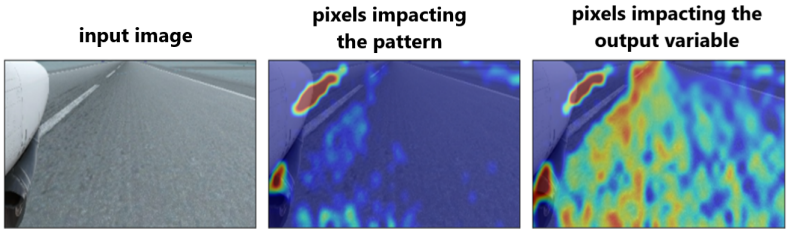

Fig. 3. GradCAM++ based attribution

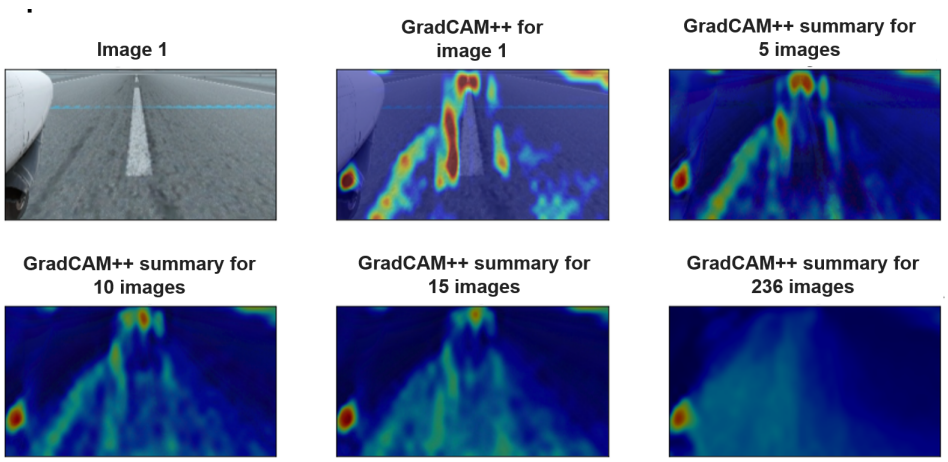

Fig. 4. Attribution over single image and summary over sequences of varying length.

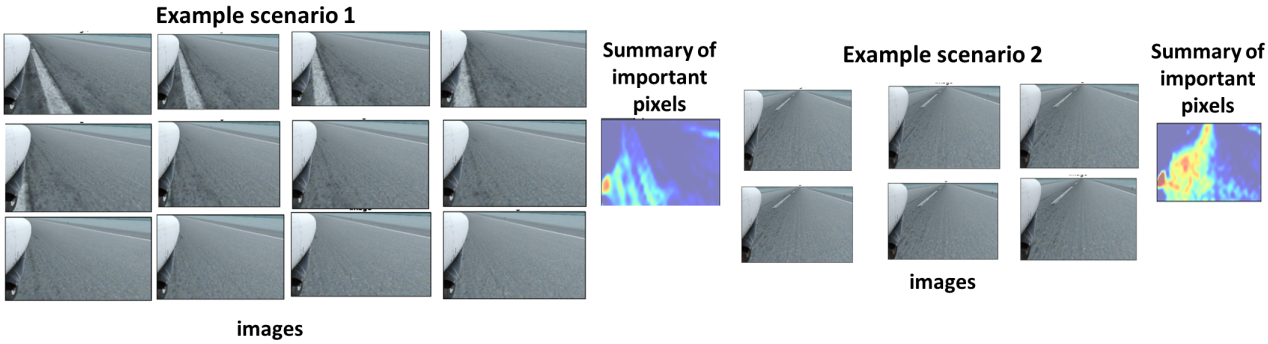

Fig. 5. Attribution for sequences for correct behavior

help in localizing the features that impact the satisfaction of the correctness property. This example highlights the benefit of using patterns for explainability of regression models. The pattern enables the representation of the inequality constraint $|y_0 - y_{0ideal}| \leq 1.0$ as a binary classification condition, thereby aiding in the discrimination of the pixels that impact the satisfaction of the correctness property vs its violation. This enables localization of the input features impacting the correct behavior of the network better than identifying the pixels that directly impact the output variable (as is done with traditional attribution approaches).

C. Leveraging Image Sequences

Attribution with respect to a single image highlights important pixels that may be specific to the image but they may not capture the more general feature that the pattern represents. Summarizing the important pixels over multiple images satisfying the same pattern has the potential to extract this general feature. However, summarizing over a large number of images may be too noisy. Such a case is shown in Fig 4, where the summary over 236 images (all satisfying the same activation pattern) highlights only a very small portion of the image. Note that the ACT network receives images sequentially, one after the other, representing the plane moving on the runway. Thus, the same feature may appear in different parts of the input images, corresponding to different times

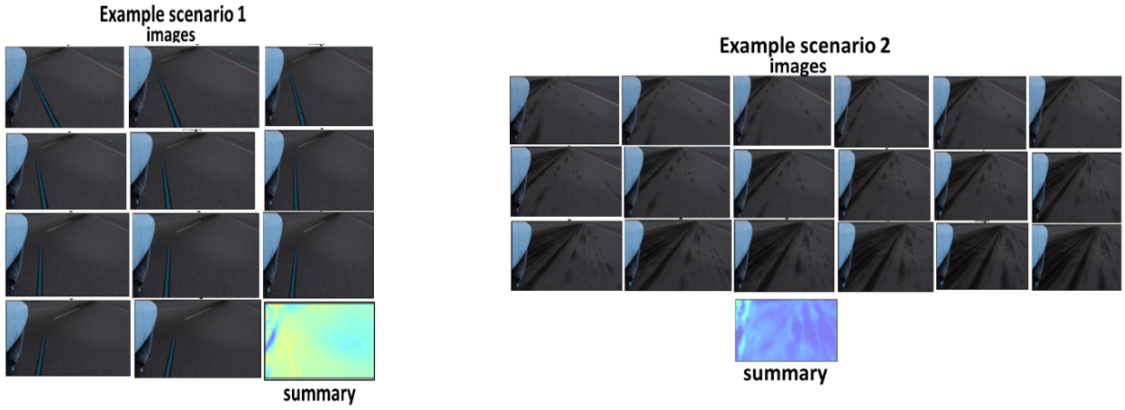

Fig. 6. Attribution for sequences for incorrect behavior

in the sequence. We therefore propose to consider images in *contiguous sequences* that satisfy the same pattern. Each such sequence represents similar images that were taken in succession while the airplane was moving on the taxiway. We then summarize the important pixels over all images in a sequence (by considering the average GradCAM++ value for each pixel) and use this to visualize the feature that the pattern represents for that sequence.

D. Explaining Correct Behavior over Image Sequences

We were able to identify a number of sequences of considerable lengths that satisfied the patterns for correct behavior. For instance, for the correctness property constraining y_0, there were 62 individual sequences, one of them being 236 images in length, that satisfied the pattern for correct behavior. For the correctness property constraining y_1, there were 50 sequences with the highest length being 795 that satisfied the pattern for correct behavior.

Figure 5 shows two sequences that satisfy our example pattern for correct behavior for y_0 and their respective attribution summaries. These represent two different input scenarios which are viewed similarly by the network since they satisfy the same pattern. Note that the important pixels highlighted are different in both cases, however, both of them represent the same feature in their respective scenarios; the distance between the nose of the plane and the center line. This feature being relevant to determining the cross-track error, it explains why the network produces the correct outputs for these scenarios.

E. Explaining Incorrect Behavior over Image Sequences

We used the patterns for the violation of the correctness properties and the respective sequences, to explain the incorrect behavior of the network. For the violation of the correctness property constraining y_0, the pattern with the maximum support (403) had 24 sequences with maximum length of 34 images. For the violation of the correctness property constraining y_1, the pattern with the maximum support (558) had 30 sequences with maximum length of 65 images.

Here is an example pattern for the violation of the correctness property for the cross-track error output, $\text{on}(N_{1,53}) \wedge \text{off}(N_{1,29}) \wedge \text{on}(N_{1,20}) \wedge \text{off}(N_{1,49}) \wedge \text{off}(N_{1,15}) \wedge \text{off}(N_{1,95}) \wedge \text{off}(N_{1,25}) \Rightarrow |y_0 - y_{0ideal}| > 1.0$.

Figure 6 shows two scenarios that satisfy this pattern and their respective summaries. In scenario 1, we can understand based on the pixels highlighted that the blue line probably acts as noise and interferes with the correct determination of the cross-track error output. In scenario 2, none of the pixels are highlighted indicating the absence of a distinct feature that the network could use to make a correct estimation of the cross-track error.

F. Patterns for Guaranteeing Consistent Behavior

All inputs satisfying a pattern potentially satisfy the output property as well. A formal proof is a guarantee of consistent behavior of the network on regions of inputs that are similar to each other with respect to the semantic features that the pattern represents.

Solvers such as Marabou [13] can be used to prove properties of neural networks. However, the ACT model from our industry partner is complex, involving convolutions, and ELU activations, which cannot be handled by existing solvers such as Marabou. We tried training a simpler feed-forward ReLU network on the provided data but could not obtain a network with good accuracy. We therefore resorted to validating each pattern statistically by calculating its precision, recall and F1-score on a separate test set of 2777 images. We show in Figure 7 that the patterns for correct behavior display very good precision on the test set. The patterns for incorrect behavior displayed fairly good precision, greater than 70%. These statistical results provide confidence that the patterns

PROPERTY	LAYER	PRECISION (%)	RECALL (%)	F1
CLASS 1				
|Y1-Y1ideal|<=5.0	ALL	100.00	20.95	34.65
|Y1-Y1ideal|<=5.0	1	100.00	9.09	16.67
|Y0-Y0ideal|<=1.0 and |Y1-Y1ideal|<=5.0	2	100.00	6.97	13.03
|Y0-Y0ideal|<=1.0	2	100.00	6.90	12.90
|Y1-Y1ideal|<=5.0	2	100.00	6.65	12.47
|Y0-Y0ideal|<=1.0 and |Y1-Y1ideal|<=5.0	ALL	100.00	6.54	12.28
|Y1-Y1ideal|<=5.0	1	100.00	6.14	11.57
|Y0-Y0ideal|<=1.0	ALL	100.00	6.13	11.56
|Y1-Y1ideal|<=5.0	ALL	100.00	5.88	11.10
CLASS 0				
|Y0-Y0ideal|>1.0	ALL	73.91	23.94	36.17
|Y0-Y0ideal|>1.0	1	86.67	18.31	30.23
|Y1-Y1ideal|>5.0	1	73.33	15.49	25.58
|Y1-Y1ideal|>5.0	ALL	87.50	9.21	16.67
|Y0-Y0ideal| > 1.0 or |Y1-Y1ideal| > 5.0	ALL	70.00	9.21	16.28
|Y1-Y1ideal|>5.0	ALL	100.00	8.55	15.76
|Y1-Y1ideal|>5.0	1	100.00	7.24	13.50
|Y0-Y0ideal| > 1.0 or |Y1-Y1ideal| > 5.0	ALL	90.91	6.58	12.27

Fig. 7. Patterns with best F1-score on the test set

could be used to identify valid and invalid behaviors, although we could not formally prove them.

G. Formal Proofs with Marabou

In order to highlight the benefit of using patterns to obtain formal guarantees of consistent behavior, we used a smaller, fully connected network with ReLU activations (TinyNet, refer Section II-D) for the same purpose of centering the plane on the runway. We used a labeled dataset of 51462 images in the down-sampled input space of TinyNet and extracted patterns at each intermediate layer and all layers together.

Sequence of images satisfying a pattern for the correctness or safety property potentially represent an interval of time where the network displays consistent behavior. We selected sequences of images satisfying the same pattern and bounded the input region as an over-approximation box ($[x_{\min}, x_{\max}]$) covering all images within the sequence. We then used Marabou to perform the following check for the safety property; $\forall x \in [x_{\min}, x_{\max}] \wedge pattern_{class=1} \Rightarrow |y_0| \leq \theta$. $pattern_{class=1}$ represents a pattern for the satisfaction of the property and θ represents the threshold on the runway dimensions. If proved successfully, the output property could be considered as a temporal invariant over input sequences within the region, else the counterexample represents a scenario of a sequence of images resulting in violation.

It is a challenge to prove that the correctness property holds true for all the inputs satisfying a pattern. This is because the ground truth is not known for all the input images and can not be expressed in a checkable form. Thus we focused on the safety properties instead.

We experimented with different threshold values for encoding the safety properties, in order to get more insights into the network behavior. We were able to extract patterns for the following safety properties.

- $|y_0| \leq 10.0m$,
- $|y_0| \leq 8.0m$,
- $|y_0| \leq 5.0m$.

For instance, for $|y_0| \leq 5.0m$, the pattern satisfying the property with the maximum support (750) has 108 sequences with maximum length of 18 images. The pattern violating the property (support of 259) has 76 sequences with maximum length of 11 images. For $|y_0| \leq 10.0m$, we obtained proofs for 33 out of 111 sequences with at least 5 images, the longest sub-sequence where we can obtain proofs has 17 images. For $|y_0| \leq 8.0m$ and $|y_0| \leq 5.0m$, the longest sequence on which a proof was obtained was 9 images long. We performed similar experiments for the safety properties involving y_1.

We would like to point out that some of the proofs could be obtained even without the inclusion of the pattern in the constraint formulation; i.e., $\forall x \in [x_{\min}, x_{\max}] \Rightarrow |y_0| \leq \theta$ would suffice. Even in these cases, the pattern was used to identify sequence of images satisfying them and thereby obtain the minimum and maximum values for each pixel, i.e. the input region bounded by $[x_{\min}, x_{\max}]$. In order to further evaluate the benefit of using activation patterns, we attempted proofs on random sequences of images satisfying the output property (but not necessarily satisfying any pattern for correct behavior) and computing $[x_{\min}, x_{\max}]$ over them. However, we were unable to obtain any proofs. These observations highlight that the patterns help select sequences of images (thereby input

regions) which are processed similarly by the network and hence have higher chances of consistent behavior.

H. Counterexamples

The generation of scenarios where the plane can run out of the runway is very useful for debugging and improving the network. However, it is a challenge to generate a scenario that is realistic or is plausible to occur in a close loop system, using just the neural network model. We describe in this section how we use the counterexamples reported by Marabou for the failed proofs of the patterns to generate such scenarios.

Recall that the pattern for correct behavior in the check $\forall x \in [x_{\min}, x_{\max}] \wedge pattern_{class=1} \Rightarrow |y_0| \leq \theta$, represents the neuron activations that are satisfied by valid inputs within $[x_{\min}, x_{\max}]$, that also satisfy the output safety property. A counterexample to this proof represents an input that is similar to the other valid images in the sequence and hence has similar context. It also satisfies $pattern_{class=1}$, which is a temporal invariant of the system over the given sequence of valid images. This increases the chances of this image not being a random input, but an image that is possible to occur in the sequence.

We illustrate such a counterexample scenario showing the first and last image of a sequence and the generated counterexample image in the downsampled input space of TinyNet in Figure 9. In order to better understand the counterexample, we mapped it back to the original input space, denoted as 'upsampled counterexample' in Figure 9. The check was formulated as follows ; $\forall x \in [x_{\min}, x_{\max}] \wedge pattern(x) \Rightarrow |y_0| \leq 10.0m$. Here the pattern represents the activation pattern for the satisfaction of the safety property $|y_0| \leq 10.0m$ and all the images in the sequence satisfy this pattern.

Marabou solves the check to find a counterexample to the given constraint. To map the counterexample image back to the original input space, we find the closest downsampled image from the image sequence and use the regular version as the reference. We modify the segment of the image used in downsampling to match the brightness of the image to the counterexample we have. Some downsampling operations done by TinyNet are irreversible, for that reason our method creates an upsampled counterexample image that is very close to the original counterexample when downsampled.

We are able to generate counterexamples with Marabou in 19 out of 58 sequences of minimum length of 2 images for the constraint $\forall x \in [x_{\min}, x_{\max}] \wedge pattern(x) \Rightarrow |y_0| \leq 10m$ in a labeled dataset of downsampled images of size 7385 and our upsampling satisfies the pattern in 17 out of 19 counterexamples.

IV. Related Work

Existing formal verification techniques for neural networks have focused on verifying input-output properties of neural network models using approaches such as the simplex method, mixed integer linear programming, and symbolic interval analysis [14], [15], [16], [17], [18]. Recent work has attempted to prove properties of models used in safety-critical applications such aircraft collision avoidance policies [9]. However, when the input dimensionality is high such as in the case of perception networks, most existing work has only been able to provide local adversarial robustness guarantees.

The work presented in [19] considers the TinyNet model that we also consider here, for the purpose of centering the plane on the runway. It employs Marabou to provide local adversarial robustness guarantees; specifically it ensures that given an input X and its respective model output Y, for all inputs within a certain distance or in the neighborhood of X, the network produces output values that are within a certain distance of Y. The work reported in [12] presents a method that searches for sequences of image disturbances that to lead to failure in the aircraft taxiway application using TinyNet model in a closed-loop system. The method employs local robustness verification techniques along with adaptive stress testing and simulation to search for such sequences. In contrast, our approach only involves the analysis of the neural network to find problematic scenarios.

Most existing work on explaining neural network behavior focuses on explaining the behavior of classification models ([20], [11], [21], [22], [6], [23]) and is based on class activations, perturbations or gradients. The recent work in [24] presents a database and an experimental setup to quantitatively evaluate techniques for explaining model behavior. It highlights that techniques used traditionally for classification networks do not work well for regression problems. They tend to produce noisy and inconsistent results. The paper proposes a new technique for explaining regression model outputs, which averages out results from different training runs of the model. The technique seems to produce better explanations than existing techniques, however, it is very expensive in practice. In our work, we show how we could use the activation patterns computed with respect to the output properties of regression models to represent the regression problem as a classification, thereby enabling the use of existing techniques for explainability.

V. Lessons Learned and Applications

Neural networks are increasingly used in many domains, some of them safety critical, as the case study reported in this paper. We reported on the application of a recent tool, Prophecy, to understanding and providing guarantees for neural network behavior. We have shown how the activation patterns that are mined by Prophecy can be used to *explain* the neural network behavior across multiple inputs, instead of single images as typically done in previous work. The obtained explanations are therefore more general, and potentially more useful for the developers. We have found that while there is a vast literature on attribution techniques for classification models there is much less work on providing explanations for regression models, and none that aims to generalize said explanations across multiple inputs. More research is needed for finding explanations in regression models. More information from the simulator could help explain the network behavior

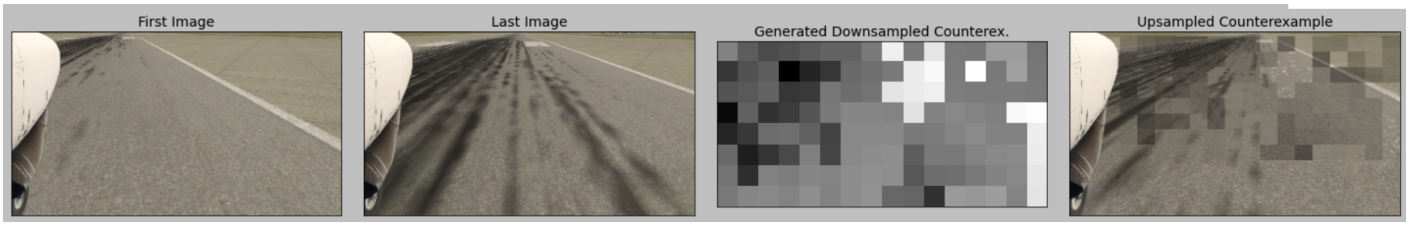

Fig. 8. Counterexample for an image sequence of length 39 for $|y_0| \leq 10m$ safety property.

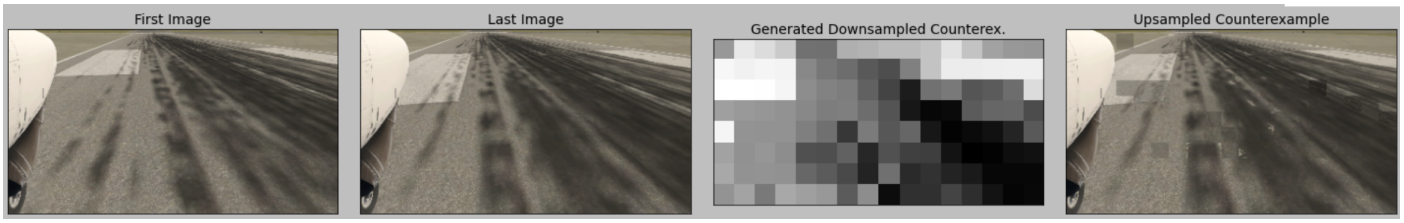

Fig. 9. Counterexample for an image sequence of length 5 for $|y_0| \leq 10m$ safety property.

at a more semantic level, thereby enabling the generation of images that may help test and improve the model [25].

We have also shown that the activation patterns can be used to provide guarantees about the correctness and safety of the neural network. These guarantees can be statistical guarantees (validated on a separate test set) or formal (proven with Marabou). We were unable to give formal guarantees for the complex model from our industry partner, whereas for the smaller network designed for the same purpose, we were able to provide guarantees of consistent behavior over multiple input regions. A take-away from our experiments is that it may be better to design neural networks which are verifiable; start with smaller and simpler architectures with piecewise linear computations, verify their behavior and add complexity as needed.

We envision several applications for the activation patterns that are mined with our approach.

First, the activation patterns can be used for reducing redundancy in the training data set. For patterns that lead to correct behaviour and have very high support, this may be an indication that the images are redundant and we can reduce the train set there, leading to faster training time with similar accuracy. Furthermore, the patterns mined for incorrect behaviour may be used to find gaps in the training set. These patterns for misclassifications can help developers focus on input areas where more images are needed and generate such images with the help of the simulator.

Finally, we envision using the patterns for correctness properties in a *predictive manner* at runtime, to determine how confident the network is in its prediction. In real-world decision making systems, neural networks must not only be accurate, but also should indicate when they are likely to be incorrect. While there are related works that attempt to provide a measure of *confidence* in the network outputs, we find that they are again limited to classification problems (see e.g. [26]). Our patterns can potentially provide a solution (albeit partial) to regression models. We are working on integrating the patterns into a runtime monitoring framework involving the ACT network.

VI. Conclusion and Next Steps

In this paper, we presented an industrial case study applying a recently proposed technique, Prophecy, to analyze the behavior of neural network models used for autonomous guiding of airplanes on taxi runways. These models are regression models that take as input an image of the runway and produce two outputs, cross-track error and heading error, which represent the position of the plane relative the center line. The Prophecy tool was used to extract patterns for the correctness and safety properties of the networks. We showed the usefulness of these patterns to identify features of the input that explain correct and incorrect behavior. We also used the patterns to provide guarantees of consistent behavior. In our work, we explored a novel idea of using sequences of images (instead of single images) to obtain good explanations and identify regions of consistent behavior.

This case study of autonomous center line tracking is our first effort to apply Prophecy on aviation applications. Other visual perception applications will also be considered in the future, such as autonomous takeoff or landing, detect and avoid, object detection and identification. In addition to a broader scope of applications, based on our obtained results, we will further explore definitions of confidence and coverage for these analyses. Confidence and coverage, built on analyses with formal methods and/or statistical methods, will provide quantitative support to artificial intelligence (AI) trustworthiness and potential certification of AI-enabled systems. Our future research also includes using design time analysis results during operations via runtime monitors. Pre-recorded activation patterns can be used by the monitors to identify or predict potential safety violations for unseen images.

REFERENCES

[1] D. Gopinath, H. Converse, C. Pasareanu, and A. Taly, "Property inference for deep neural networks," in *34th International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 797–809.

[2] E. Frew, T. McGee, Z. Kim, X. Xiao, S. Jackson, M. Morimoto, S. Rathinam, J. Padial, and R. Sengupta, "Vision-based road-following using a small autonomous aircraft," in *2004 IEEE Aerospace Conference Proceedings (IEEE Cat. No.04TH8720)*, vol. 5, 2004, pp. 3006–3015 Vol.5.

[3] S. Beland, I. Chang, A. Chen, M. Moser, J. Paunicka, D. Stuart, J. Vian, C. Westover, and H. Yu, "Towards assurance evaluation of autonomous systems," in *ICCAD*, 2020.

[4] "X-plane flight simulator." [Online]. Available: https://www.x-plane.com/

[5] A. Shrikumar, P. Greenside, and A. Kundaje, "Learning important features through propagating activation differences," in *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, 2017, pp. 3145–3153.

[6] M. T. Ribeiro, S. Singh, and C. 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*, 2016, pp. 1135–1144.

[7] Y. Nohara, K. Matsumoto, H. Soejima, and N. Nakashima, "Explanation of machine learning models using improved shapley additive explanation," in *Proceedings of the 10th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics, BCB 2019, Niagara Falls, NY, USA, September 7-10, 2019*, 2019, p. 546.

[8] G. Katz, C. W. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Towards proving the adversarial robustness of deep neural networks," in *Proceedings First Workshop on Formal Verification of Autonomous Vehicles, FVAV@iFM 2017, Turin, Italy, 19th September 2017*, 2017, pp. 19–26.

[9] K. D. Julian and M. J. Kochenderfer, "Guaranteeing safety for neural network-based aircraft collision avoidance systems," *CoRR*, vol. abs/1912.07084, 2019.

[10] A. Chattopadhay, A. Sarkar, P. Howlader, and V. N. Balasubramanian, "Grad-cam++: Generalized gradient-based visual explanations for deep convolutional networks," in *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2018, pp. 839–847.

[11] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," *Int. J. Comput. Vis.*, vol. 128, no. 2, pp. 336–359, 2020.

[12] K. D. Julian, R. Lee, and M. J. Kochenderfer, "Validation of image-based neural network controllers through adaptive stress testing," in *23rd IEEE International Conference on Intelligent Transportation Systems, ITSC 2020, Rhodes, Greece, September 20-23, 2020*, 2020, pp. 1–7.

[13] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljic, D. L. Dill, M. J. Kochenderfer, and C. W. Barrett, "The marabou framework for verification and analysis of deep neural networks," in *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I*, 2019, pp. 443–452.

[14] M. Akintunde, A. Lomuscio, L. Maganti, and E. Pirovano, "Reachability analysis for neural agent-environment systems," in *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixteenth International Conference, KR 2018, Tempe, Arizona, 30 October - 2 November 2018*, 2018, pp. 184–193.

[15] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Efficient formal safety analysis of neural networks," in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, 2018, pp. 6369–6379.

[16] C. Liu, T. Arnon, C. Lazarus, C. A. Strong, C. W. Barrett, and M. J. Kochenderfer, "Algorithms for verifying deep neural networks," *Found. Trends Optim.*, vol. 4, no. 3-4, pp. 244–404, 2021.

[17] R. Ivanov, J. Weimer, R. Alur, G. J. Pappas, and I. Lee, "Verisig: verifying safety properties of hybrid systems with neural network controllers," *CoRR*, vol. abs/1811.01828, 2018.

[18] X. Yang, H. Tran, W. Xiang, and T. T. Johnson, "Reachability analysis for feed-forward neural networks using face lattices," *CoRR*, vol. abs/2003.01226, 2020.

[19] H. Wu, A. Ozdemir, A. Zeljic, K. Julian, A. Irfan, D. Gopinath, S. Fouladi, G. Katz, C. S. Pasareanu, and C. W. Barrett, "Parallelization techniques for verifying neural networks," in *2020 Formal Methods in Computer Aided Design, FMCAD 2020, Haifa, Israel, September 21-24, 2020*, 2020, pp. 128–137.

[20] B. Zhou, A. Khosla, À. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, 2016, pp. 2921–2929.

[21] B. N. Patro, M. Lunayach, S. Patel, and V. P. Namboodiri, "U-CAM: visual explanation using uncertainty based class activation maps," *CoRR*, vol. abs/1908.06306, 2019.

[22] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I*, 2014, pp. 818–833.

[23] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," in *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Workshop Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2014.

[24] M. Millan and C. Achard, "Explaining regression based neural network model," *CoRR*, vol. abs/2004.06918, 2020.

[25] E. Kim, D. Gopinath, C. S. Pasareanu, and S. A. Seshia, "A programmatic and semantic approach to explaining and debugging neural network based object detectors," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, 2020, pp. 11 125–11 134.

[26] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," *CoRR*, vol. abs/1706.04599, 2017. [Online]. Available: http://arxiv.org/abs/1706.04599