

Understanding and Verifying Deep Neural Networks

Divya Gopinath

Research Scientist, RSE group NASA AMES, KBR Inc.

Outline

- Introduction
- Background
- Our Approach
- Case Studies
- Future work
- Conclusion

RSE

(Robust Software Engineering, NASA Ames)

- **Projects and tools:**
 - **FRET:** Requirements elicitation, formalization and consistency checking
 - **CoCoSIM:** Verification of safety requirements
 - **IKOS:** Static code analyzer
 - **AdaStress:** Adaptive Stress Testing to find corner cases
 - **AdvoCATE:** Creating safety assurance cases
 - **R2U2:** Runtime monitoring and analysis
- **SafeDNN** (<https://ti.arc.nasa.gov/tech/rse/research/safednn/>): Corina Pasareanu, Divya Gopinath
 - *Deep Neural Networks (DNNs)* have widespread usage, even in safety-critical applications such as autonomous driving
 - Develop techniques that aim to ensure that systems that use DNNs are safe, robust and interpretable
 - **Prophecy:** Formal analysis of DNN models to infer properties which could be used for understanding, verifying, debugging and testing

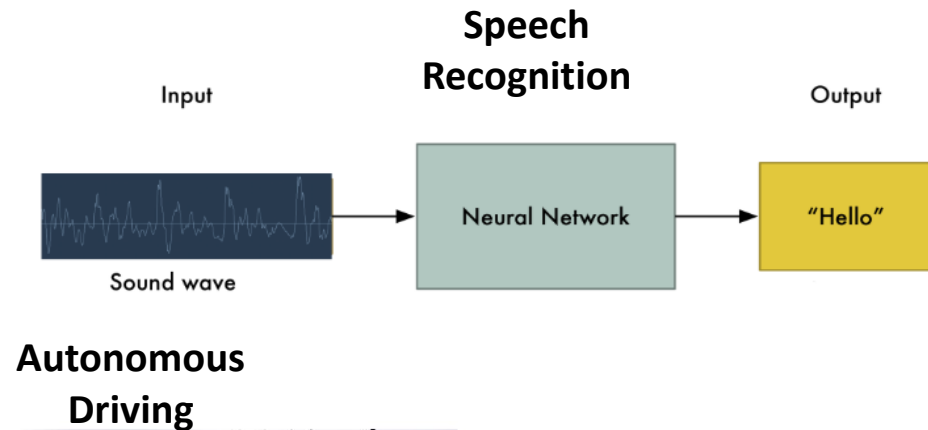
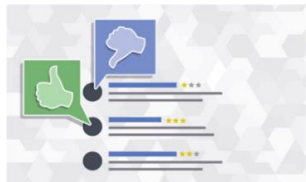
Outline

- Introduction
- Background
- Our Approach
- Case Studies
- Future work
- Conclusion

Deep Neural Networks

- Deep Neural Networks (DNNs) have widespread usage, even in safety-critical applications such as autonomous driving

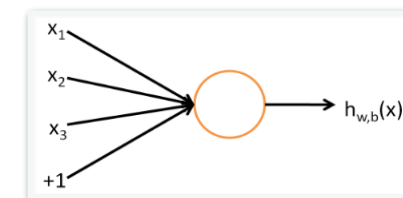
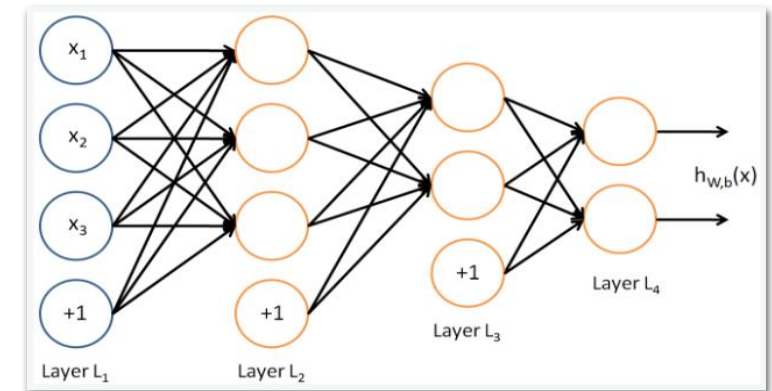
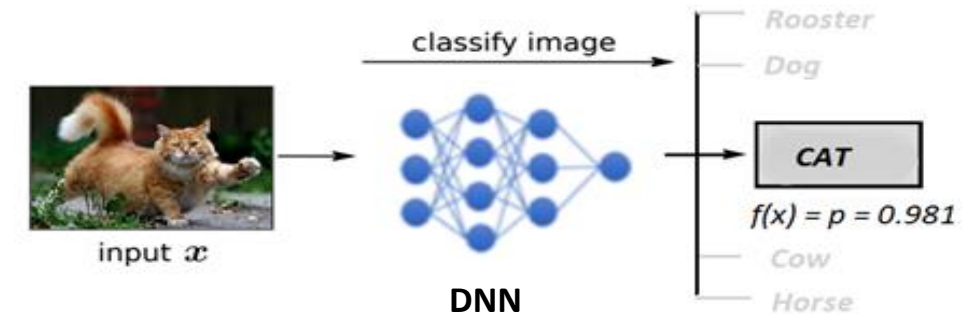
Sentiment Analysis



Autonomous Driving



Image Classification



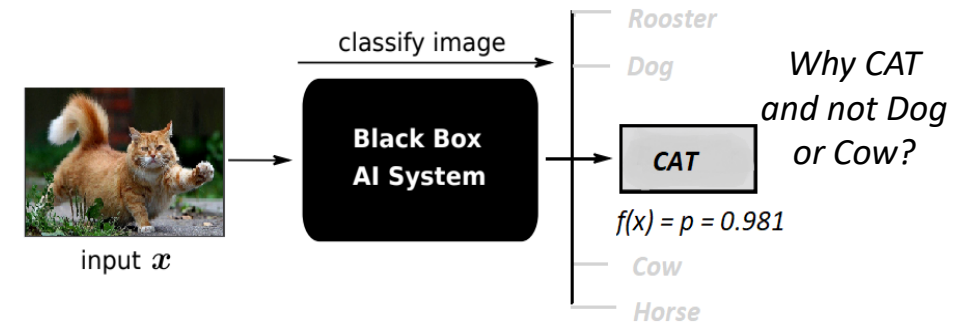
ReLU activation function
 $f(x) = \max(0, x)$

$$h_{w,b}(x) = f(W^T x) = f(\sum_{i=1}^3 W_i x_i + b)$$

Challenges

- Lack of *explainability*

- It is not well understood why the network makes its decisions
- Design not amenable for analysis, Logic not interpretable
- Impacts reliability, impedes trust



- Lack of *guarantees for network behavior*

- Often networks do not have formal input-output specifications defining functional correctness
- Networks are large and complex inhibiting efficient analysis and verification

Existing work (Explainable AI)

- Work done mostly in the fields of computer vision and NLP
- Explaining behavior of pre-trained models (Model-Specific)
 - Saliency Maps[Simonyan, K. et al, 2014], Gradient descent [Kümmerer, M. et al., 2014] , DeepLIFT [A.Shrikumar et al, 2017], Integrated Gradients[M.Sundarajan et al, 2017] identify portions of the image that impact network prediction
 - DeepExplain [M.D. Zeiler et al, 2014], Guided-Back propagation visualize features learnt by the network at different layers
 - Class Activation Maps (GradCAM [Selvaraju, R.R. et al,2017], GradCAM++) indicate discriminative regions of an image used by a CNN to categorize them into different classes
 - LRP [S.Bach, 2015] is an attribution technique applicable to images and text, Rationale [Lei.T et al,2016] is an interpretability method for text (NLP)
- Explaining any black-box model (Model Agnostic): LIME, Ancor[Ribeiro, M.T. et al, 2016,2018], SHAP [Lundberg, S.M. et al, 2017], PDP [Dhurandhar, A. et al, 2018]

Existing work on Explainable AI (Open issues)

- Not much work on generating explanations for more *complex output properties* and behaviors than classification
- Most techniques *are typically local* and generate explanations wrt a single image or a set of images
- There aren't techniques that generate *formal explanations* which can be proved
- There isn't a common or *generic approach that is applicable to different types* of networks (classification, regression, recurrent networks so on)

Existing work (Verification)

- Number of approaches have been developed to verify if a given DNN model satisfies a property

$$\forall x \text{ } Pre(x) \Rightarrow Post(F(x))$$

- For perception networks, the property is mainly Adversarial Robustness
 - For some networks (ACAS Xu; controller network with low-dimensional sensor inputs), pre-defined input-output specifications are available and have been verified
- Search-Based techniques: Reluplex , Marabou [G. Katz et. al, 2017] ,Planet [R.Elhers, 2017]
- Reachability-Based techniques: DeepZono, DeepPoly [G.Singh et.al, 2018,2019],AI2[T.Gehr et a,, 2018]
- Optimization-Based techniques: MIPVerify [V.Tjeng et al, 2019]
- Search+Optimization: Neurify[S. Wang et al, 2018], VeriNet [P.Hernicksen et al , 2020]

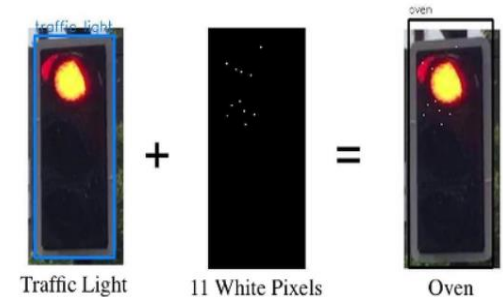
Adversarial Perturbations



Street sign



Birdhouse



Existing work on Verification (Open Issues)

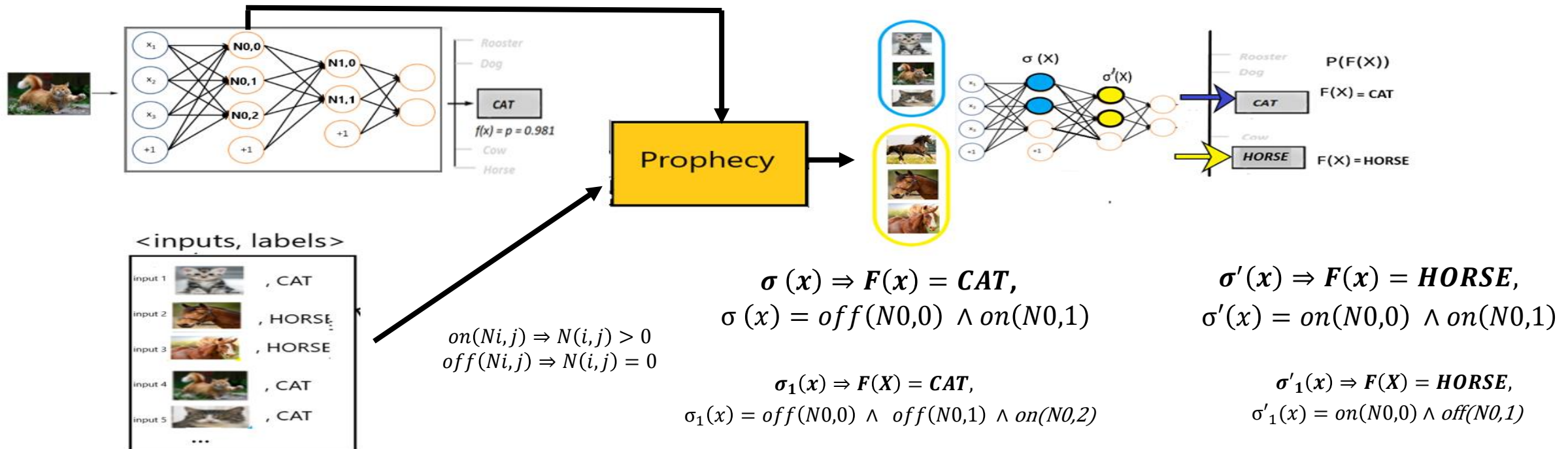
- Mainly applicable to *feed-forward networks, Piece-wise linear activation functions (ReLU)*
- Not *scalable* to large complex networks
- Guarantees of robustness in *small local regions around inputs*
- *Need richer, more expressive properties capturing the overall functional behavior of the DNN*

Outline

- Introduction
- Background
- Our Approach
- Case Studies
- Future work
- Conclusion

Our Approach

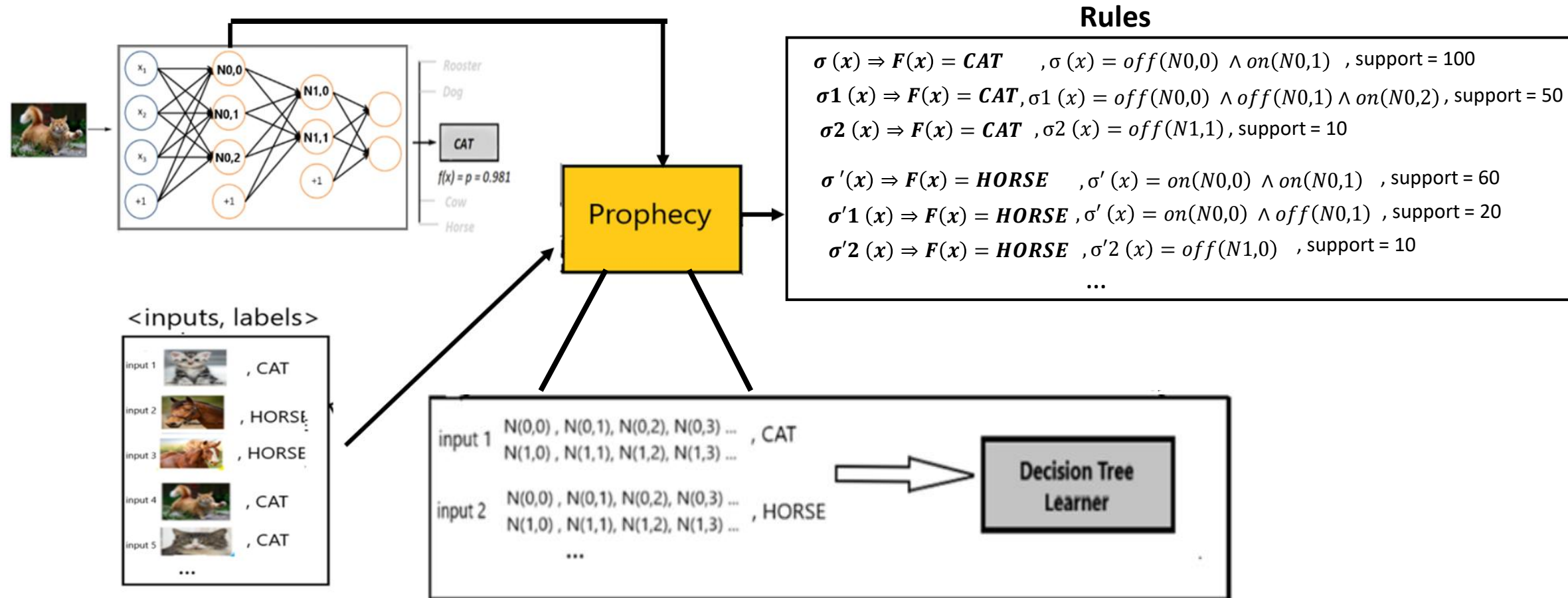
- Decompose the complex DNN model into a *set of simple rules*, amenable to analysis
 - Assume-guarantee type rules are inferred from a trained DNN; $\forall x \sigma(x) \Rightarrow P(F(x))$
 - P is a property of the network function; functional property
 - $\sigma(X)$ are formal constraints on neurons at inner layers of the network (*neuron activation patterns*)
 - Prophecy*: Property Inference for Deep Neural Networks (ASE 2019)



Prophecy

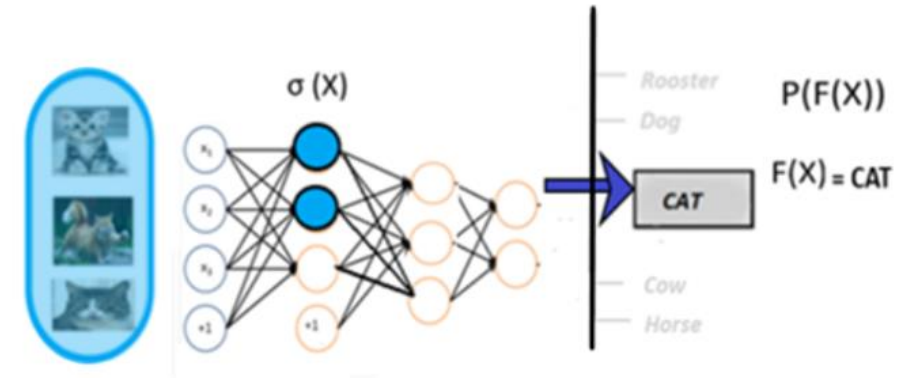
Property Inference for Deep Neural Networks

[ASE 2019]



Our Approach (Benefits)

- Rules act as *“likely” specifications*, richer and more expressive properties of functional behavior
 - Faithful to network behavior
- Mathematical formulation is amenable to verification, providing *guarantees wrt functional behavior*
$$\forall x \sigma(x) \Rightarrow P(F(x))$$
 - Enable more efficient (*compositional verification*) of input-output properties
- Visualization of rules enable *explainability and interpretability*
 - Obtaining *formal explanations* which can be proved
- Applicable to *any type of input* (image, text, sensory signals) and *complex output properties*



Applications

- The properties extracted using Prophecy have many applications
 - Obtaining formal guarantees of network behavior
 - Interpretability and Explainability of network behavior
 - Network Distillation
 - Proof Decomposition
 - Debugging and repair
- Case studies on perception networks, controller networks, classifier and regression models
 - Feed forward networks
 - With fully connected, convolution layers, maxpool layers
 - ReLU , eLU activation functions
- This talk will focus on our case-studies on DNNs used as *perception and controller modules in autonomous driving and using Prophecy for explainability and verification*

Outline

- Introduction
- Background
- Our Approach
- Case Studies
 - Regression Model for Perception
- Future work
- Conclusion

Case study on a Regression model for Perception [2]

- *TaxiNet* is a neural network designed to take a single picture of the runway as input and return the plane's position w.r.t. the center of the runway
 - Returns 2 numerical outputs; *Cross track error (y_0)*: The distance of the plane from the middle line, *Heading error (y_1)*: The angle of the plane w.r.t. the middle line
- Input data is a *sequence of images* captured by the camera as the plane moves on the runway
 - A simulator (Xplane) used to generate data for training and testing



<https://blog.send-anywhere.com/blog/2016/01/13/uc-3-entertainment-for-travel/boeing-aircraft-plane-on-runway-free-wallpaper-hd/>



Problem Statement

- Desired properties of the network outputs
 - *Safety property*: In order to ensure that the plane is in the safe zone within the runway
 $|y_0| < 10.0m, |y_1| < 90^\circ$
 - *Correctness property*: Based on data whose ideal outputs are known
 $|y_0 - y_{0ideal}| < 1.5m, |y_1 - y_{1ideal}| < 5^\circ$
- Can we understand why the network behaves (correctly/incorrectly) in some scenarios vs. others?
 - We want to identify *input features* that impact network behavior w.r.t *correctness constraints*
 - The feature should be a *characteristic of a sequence of images*
 - Useful in debugging, generating additional testing scenarios, Runtime monitors
- Can we generate guarantees for the safe operation?
 - We want to generate *guarantees over sequence of images (or a time window)*
 - We would like to generate *new image sequences that can lead to failure*
 - Important to build trust and certify network behavior
- Can we produce sound results despite considering the network as a standalone entity without the feedback loop with the simulator?

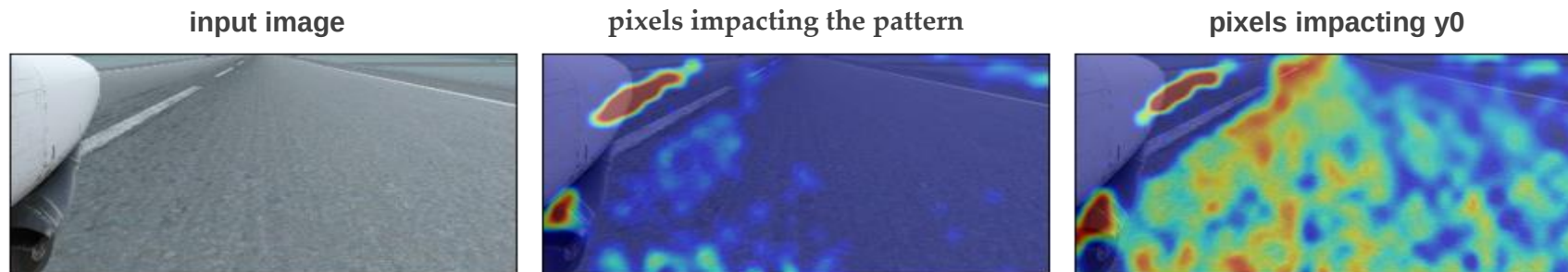
Prophecy on TaxiNet

- TaxiNet Architectures

- *Boeing TaxiNet*: CONV network with 24 layers, input is a 360x200x3 image, 5 CONV layers, 5 activation layers and 3 dense layers (100,50,10 eLU neurons respy) before the output layer with 2 outputs
- Prophecy used to extract patterns using a labeled dataset with 13885 inputs
 - Wrt three *correctness properties*; $|y_0 - y_{0ideal}| \leq 1.0$, $|y_1 - y_{1ideal}| \leq 5.0$, $|y_0 - y_{0ideal}| \leq 1.0 \wedge |y_1 - y_{1ideal}| \leq 5.0$
 - At each of the three dense layers and all of them together
 - *Patterns for satisfaction* (396 patterns for class 1), *patterns for violation* of the correctness properties (418 patterns for class 0)
- *Tiny Taxinet [3]*: Smaller network takes in a down-sampled version of the image (128 pixels), 3 dense layers (16,8,8 ReLU neurons respy) and output layer with 2 outputs
- Prophecy used to extract patterns using a labeled dataset with 51462 inputs
 - Wrt three *safety properties*; $|y_0| \leq 10.0$, $|y_0| \leq 8.0$, $|y_0| \leq 5.0$
 - At each of the three dense layers and all of them together, patterns for satisfaction and violation of the safety properties were extracted

Patterns for Explainability (Boeing TaxiNet model)

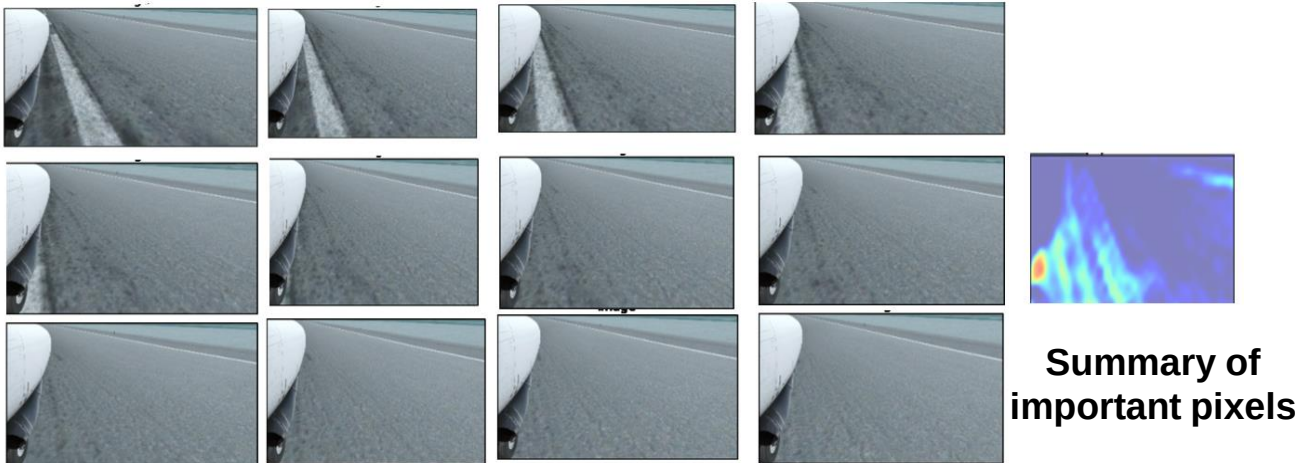
- A pattern represents *features* of the input images that impact network behavior,
 - Activation pattern from dense layer 1 for the satisfaction of the correctness property w.r.t y_0 (cross-track error)
$$\text{off}(N1,53) \wedge \text{off}(N1,33) \wedge \text{off}(N1,71) \wedge \text{off}(N1,64) \wedge \text{off}(N1,67) \Rightarrow |y_0 - y_{0\text{ideal}}| \leq 1.0, \text{Support} = 1792$$
 - We visualize these features by *highlighting the input pixels* that impact the pattern
 - For an image satisfying the pattern, highlight pixels that impact the neurons in the pattern (using *GradCAM++* [4])
 - Identifies portion of the image impacting network's behavior w.r.t the cross-track error output
 - Highlighting pixels that impact the output variable y_0 (aka existing attribution approaches) is not as helpful



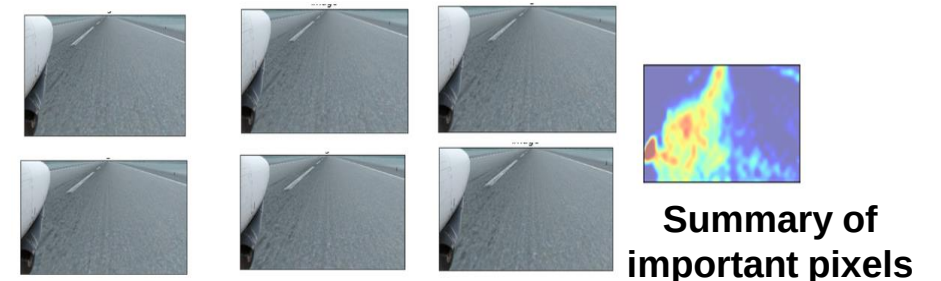
Explaining Correct Behavior

- Extracting a *common characteristic (feature) over a sequence of images*
 - 44 sequences (length > 5) satisfy the example pattern
 - The *summary of important pixels* (average GradCAM values across all images) represents the feature that impacts the output property the most, for that scenario
 - The feature; *distance between the center line of the runway and the nose of the airplane*; is relevant for cross-track error determination enabling the network to produce the correct output for this scenario

Sequence of 12 images satisfying pattern for correct behavior



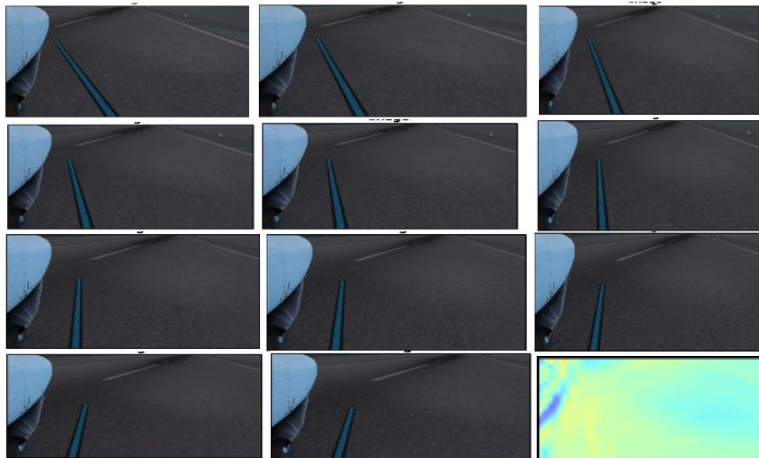
Sequence of 6 images satisfying pattern for correct behavior



Explaining Incorrect Behavior

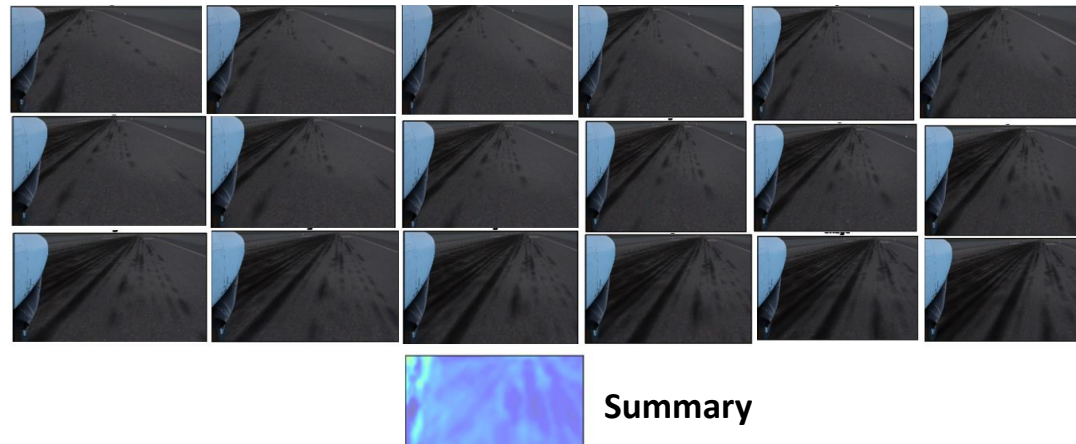
- Pattern from dense layer 1 for the violation of the correctness property w.r.t y_0 (cross-track error)
 $on(N1,53) \wedge off(N1,29) \wedge on(N1,20) \wedge off(N1,49) \wedge off(N1,15) \wedge off(N1,95) \wedge off(N1,25) \Rightarrow |y_0 - y_{0ideal}| > 1.0$, Support: 403
 - 18 sequences (length > 5) satisfy the example pattern
 - Scenario 1: Highlighted pixels indicate that *the noise (blue line) interferes with correct determination* of the cross-track error
 - 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

Example scenario 1



Summary

Example scenario 2



Summary

Formal guarantees of safety

- A formal guarantee of safety is obtained if we can prove that for all inputs satisfying a pattern, the respective outputs actually satisfy the safety constraints
 - We employed the [Marabou\[5\]](#) solver to perform this check
 - We were unable to use Marabou on the Boeing Model since it is unable to handle the complexity of the network, specifically the eLU activation functions
 - We were able to check the [safety properties on the TinyTaxinet](#) model using Marabou
$$\forall x \in [x_{min}, x_{max}] \wedge pattern \Rightarrow |y_0| \leq 10m$$
 - Obtained proofs for 33 sequences with at least 5 images, the longest sequence with proof had 17 images
 - Formal proof of consistent behavior of the network over the input region representing the [sequence of images \(a time interval\)](#)

Counter-examples

- Generating scenarios where the plane can *run out of the runway* is very useful for debugging
- Performing the following check using Marabou can throw a counter-example
$$\forall x \in [x_{min}, x_{max}] \wedge pattern \Rightarrow |y_0| \leq 10m$$
 - An image similar to the other valid images in the sequence but causes the network output to violate the safety property $|y_0| > 10m$
- The inclusion of the pattern and the bounds around valid inputs in the sequence makes the counter-example more likely to occur in an actual closed-loop system

Counterexample for an image sequence of length 39 for $|y_0| \leq 10m$



Counterexample for an image sequence of length 5 for $|y_0| \leq 10m$

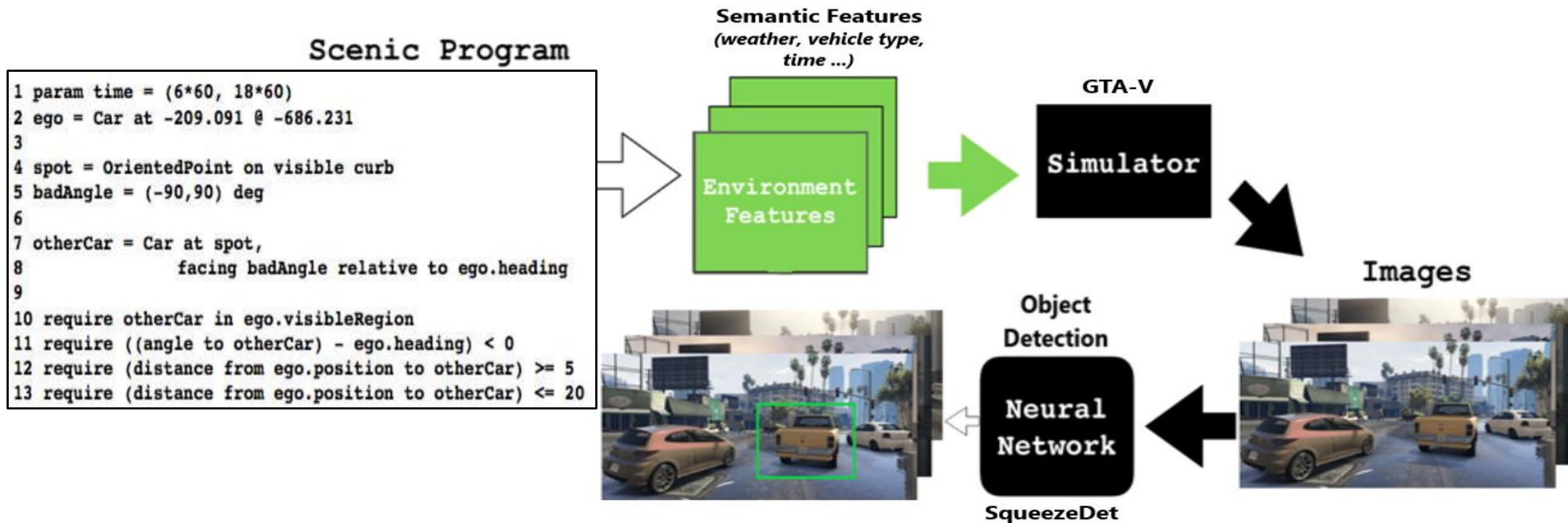


Outline

- Introduction
- Background
- Our Approach
- Case Studies
 - Regression Model for Perception
 - Object Detection Network in Autonomous vehicles
- Future work
- Conclusion

Case study on an Object Detection Network [6]

- *SqueezeDet* is a convolutional neural network for object detection in autonomous cars
- *SCENIC* is a probabilistic programming language used to describe environments or scenes for testing autonomous driving
 - Generates values for environment variables describing a scene using high level semantic features
- The environment variables fed to a simulator to create realistic images for the object detector network

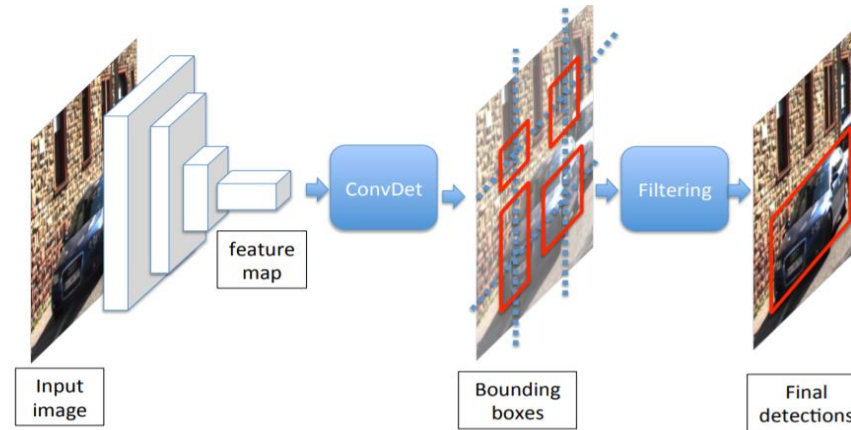


Problem Statement

- Can we generate *explanations for behavior in terms of higher-level features* (such as weather, vehicle type ...)?
 - Most existing techniques identify important portions at a pixel level on images and require human intervention to determine what these portions correspond to in terms of a feature
- Can we generate tests that will specifically *increase the correction /incorrect detection rates* of the object detector, which would help with debugging?

Prophecy on SqueezeDet

- *SqueezeDet Architecture*

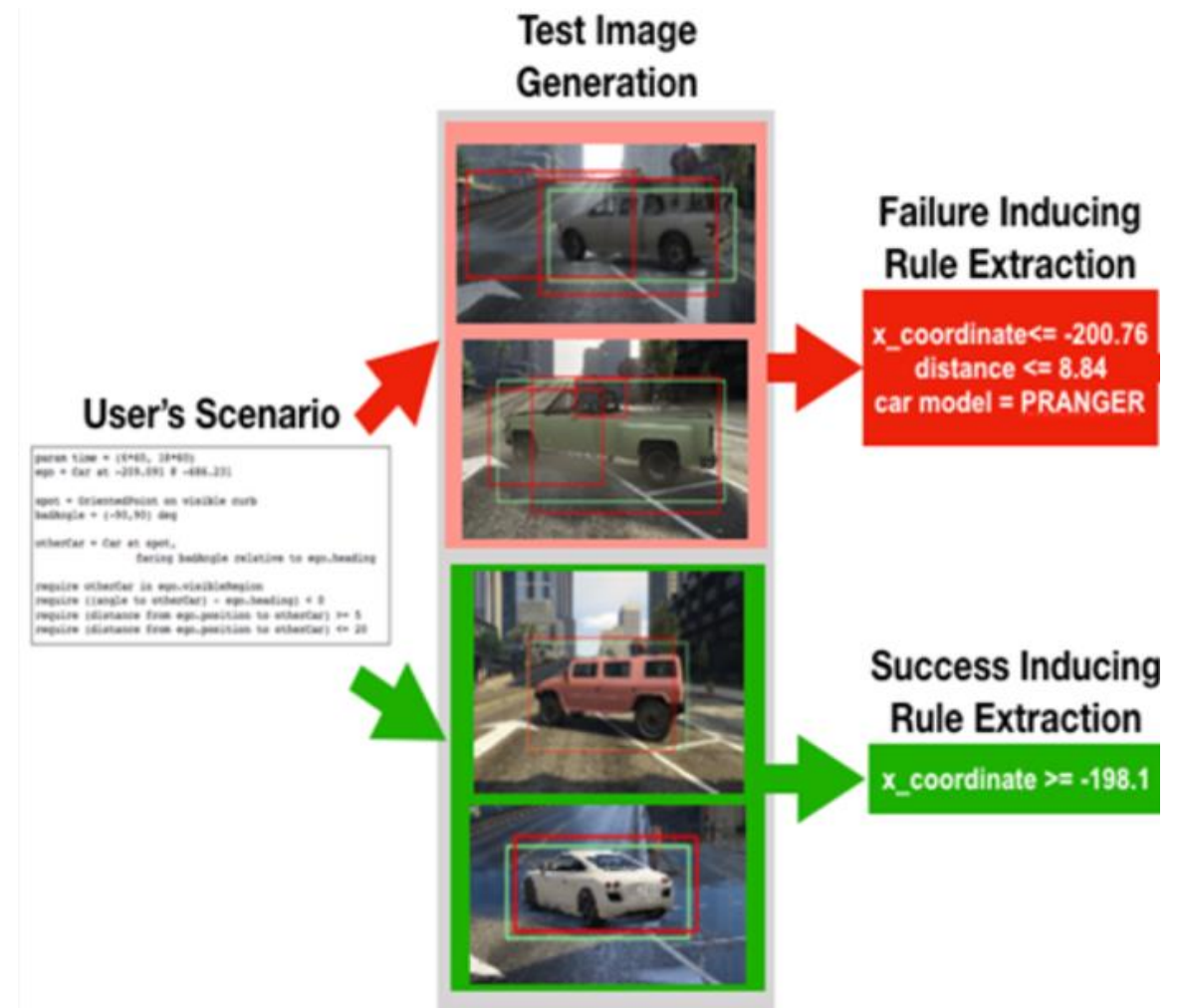


Wu, B.; Iandola, F.; Jin, P.H.; Keutzer, K. SqueezeDet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. CVPR 2017

- *Labelling the outputs*: For each image, the network's output is labelled correct or incorrect as follows
 - Correct Label: F1 for detection > 0.8 , Incorrect Label: F1 of correct detection ≤ 0.8
 - $F1 = \text{HarmonicMean}(\text{Prec}, \text{Recall})$, $\text{Prec} = (\text{TP}/\text{TP}+\text{FP})$, $\text{Recall} = (\text{TP}/\text{TP}+\text{FN})$
 - TP: # of ground truth boxes correctly predicted ($\text{IoU} > 0.5$), TN: # of ground truth boxes not detected, FP: # of bounding boxes falsely predicting ground truth, $F1 > 0.8$

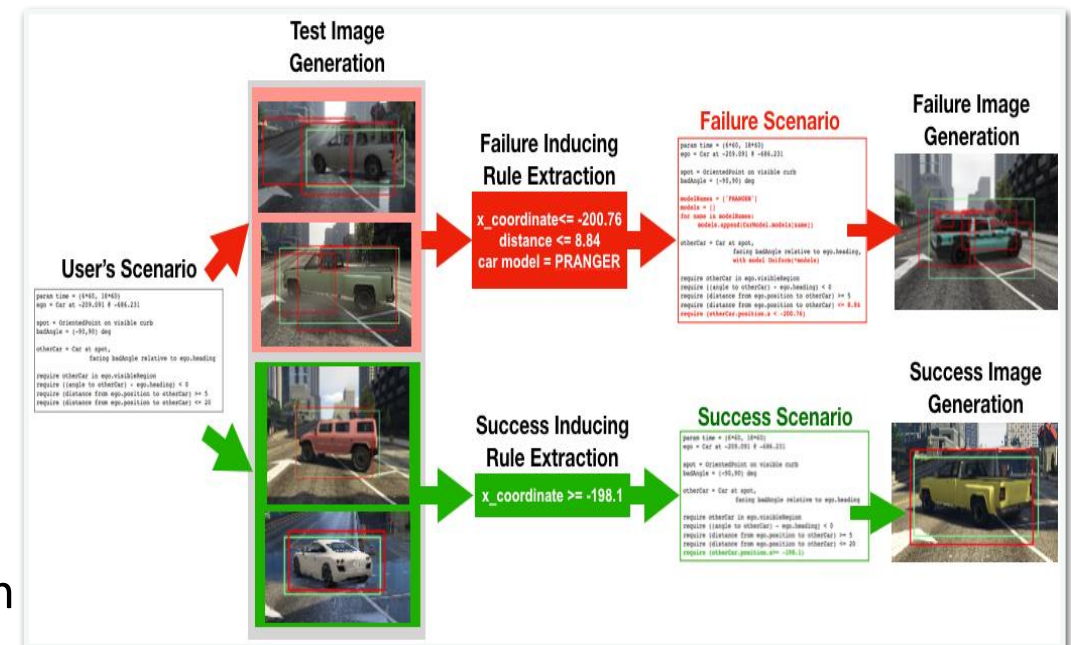
Extracting Semantic Explanations

- Given a set of labelled data, we used *Prophecy to extract the neuron activations patterns from the three Maxpool layers*
- Patterns for both *correct and incorrect labels* were extracted
- Each input also has an associated feature vector in terms of the environment variables
 - We bucketized the inputs into 4 categories based on whether they were correct/incorrect and if did/didn't satisfy the respective pattern
 - The respective feature vectors were accordingly labelled into 4 classes
 - We then used Ancor and Decision-Tree learning to extract rules in terms of the environment features for the different classes



Generating Test Inputs for Debugging

- The failure inducing rule is used to refine the scenic program to generate more failure inducing images
- The success inducing rule is used to refine or correct the program to generate more passing tests
- Increased *correct detection rate from 65.3% to 89.4%* and *incorrect detection rate from 34.7% to 87.2%*
- These additional tests could be used to debug and/re-train the object detector network

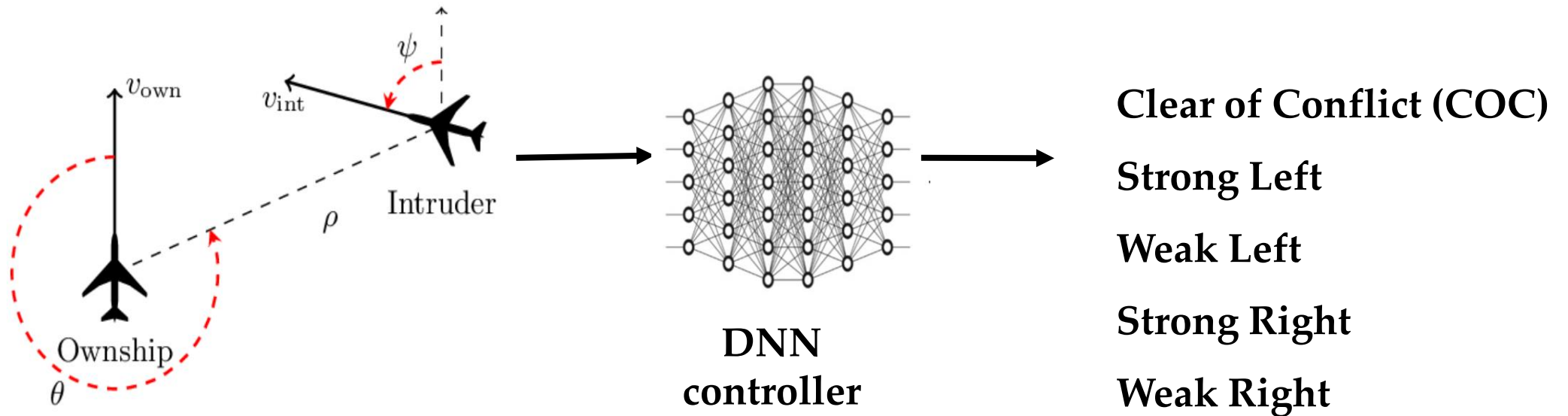


Outline

- Introduction
- Background
- Our Approach
- Case Studies
 - Regression Model for Perception
 - Object Detection Network in Autonomous vehicles
 - Controller network for Collision Avoidance
- Current/Future work
- Conclusion

Case study on a Controller network

ACAS-Xu (Airborne Collision Avoidance System-Xu)



Architecture and Properties

- 45 DNNs, Each with 5 inputs, 5 outputs, Fully connected, ReLU activations , 6 layers with a total of 300 Nodes
- System has 10 desirable properties (input-output specifications)
 - For a far away intruder, the network advises COC,
 $36000 \leq range \leq 60760, 0.7 \leq \vartheta \leq 3.14, -3.14 \leq \psi \leq 3.14 + 0.01, 900 \leq vown \leq 1200, 600 \leq vint \leq 1200$, has turning advisory COC
 - If the intruder is near and approaching from the left, the network advises “strong right”
 $250 \leq range \leq 400, 0.2 \leq \vartheta \leq 0.4, -3.14 \leq \psi \leq 3.14 + 0.005, 100 \leq vown \leq 400, 0 \leq vint \leq 400$, has turning advisory Strong Right

Problem Statement

- Existing work
 - There is a lot of work on proving adversarial robustness on this network
 - Proving the system level input-output properties such as in [7] which uses the Reluplex solver
 - Recent work explores repairing the ACASXU network with formal guarantees[8]
- Can we *simplify the verification* of the domain-level specifications?
 - It took several hours to prove the properties in [7] and couple of them timing out after 12 hours
- Can we *infer new input-output specifications* based on the trained models?
 - Helps in validating the model with the user
 - Requirements elicitation

[7] [Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, Mykel J. Kochenderfer](#): Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks, 2017

[8] [Idan Refaeli, Guy Katz](#): Minimal Multi-Layer Modifications of Deep Neural Networks, 2020

Proof Decomposition

- Patterns as *Interpolants*: Decompose proofs of input-output properties of the form $A \Rightarrow B$, using layer pattern σ by checking $A \Rightarrow \sigma$ and $\sigma \Rightarrow B$ separately
- $36000 \leq \text{range} \leq 60760$, $0.7 \leq \theta \leq 3.14$, $-3.14 \leq \psi \leq 3.14 + 0.01$, $900 \leq \text{vown} \leq 1200$, $600 \leq \text{vint} \leq 1200$, has turning advisory COC
 - $A \Rightarrow \text{COC}$ takes 31 minutes (w/o pattern) using Reluplex
 - Used a pattern σ from layer 5
 - Proved $\sigma \Rightarrow \text{COC}$ on a smaller version of the network and $A \Rightarrow \sigma$ in a total time of 7 minutes (75% speedup)
- Layer patterns could be used as interpolants for more than 1 domain-level property proofs and hence verification cost can be amortized

Specifications Inference

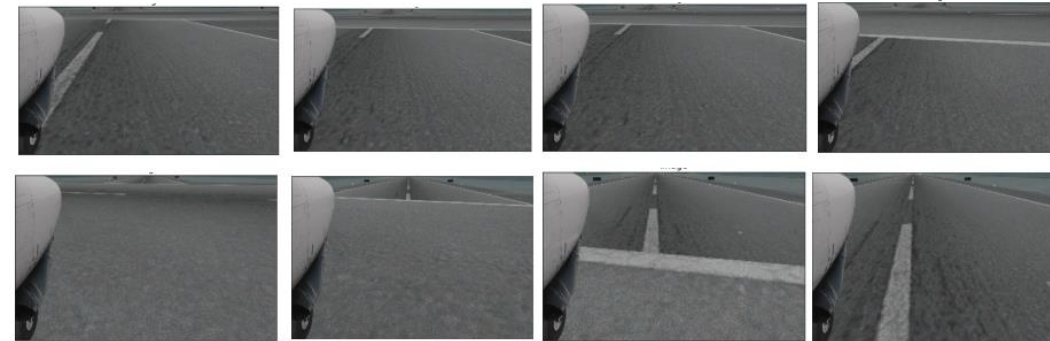
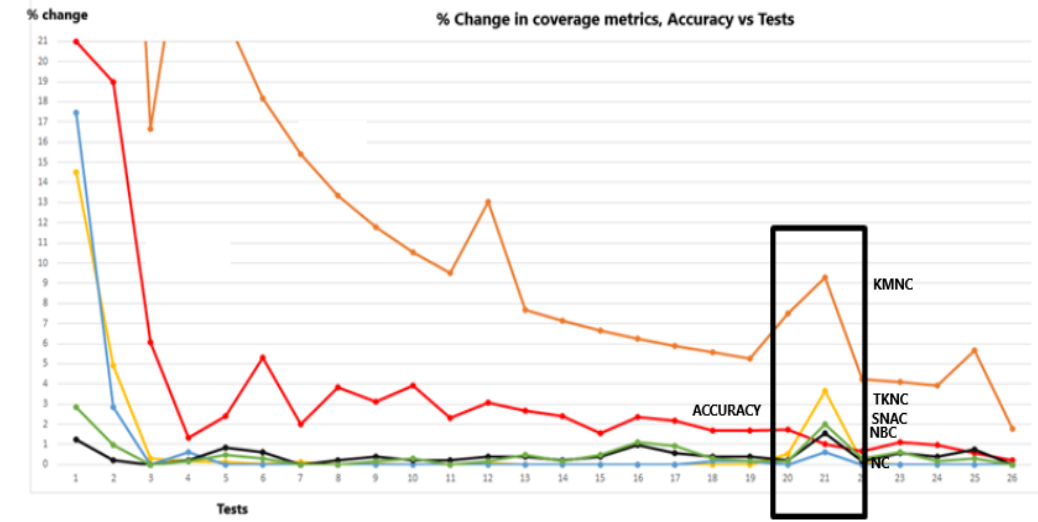
- *ACAS Xu* has meaningful input variables
 - Expressing patterns in terms of input variables leads to the discovery of the *new specifications of the domain*
 - $31900 \leq range \leq 37976, 1.684 \leq \vartheta \leq 2.5133, \psi = -2.83, 414.3 \leq vown \leq 506.86, vint = 300 \Rightarrow$ turning advisory COC
 - $range = 499, -0.314 \leq \vartheta \leq -3.14, -3.14 \leq \psi \leq 0, 100 \leq vown \leq 571, 0 \leq vint \leq 150 \Rightarrow$ turning advisory Strong Left
 - $range = 48608, \vartheta = -3.14, \psi = -2.83 \Rightarrow$ turning advisory COC
- Validated by domain experts that these are valid specifications

Outline

- Introduction
- Background
- Our Approach
- Case Studies
- Current/Future work
- Conclusion

Specification driven testing

- In order to test DNNs thoroughly we need metrics that can assess the quality or adequacy of test sets
- Structural coverage metrics measure the portions of the network covered by a test suite (in terms of structural entities such as neurons, edges)
- *DNNCov* is a tool that calculates and displays structural coverage metrics (NC, KMNC, TKNC, SNAC, NBC) of a given test-suite in addition to the statistical measure of accuracy
 - The graph shows that there may be tests which do not bring about much change to the accuracy metric but cover newer portions of the network, in turn, covering new scenarios
 - On the other hand, we can also detect tests which are redundant in terms of their coverage
- We are working on a new *coverage criteria in terms of patterns extracted by Prophecy*, which has more semantic meaning than pure structural coverage
 - Can be mapped back to requirements better , thereby helping certify network behavior



Future Work

- Patterns for correct and incorrect behavior can be used as runtime guards to assign a metric of confidence to the output, identify potentially risky inputs
- Extension to other types of networks such as RNNs

Thank You!



<https://ti.arc.nasa.gov/tech/rse/research/safednn/>