Analysis of TaxiNet Neural Networks

Divya Gopinath, Corina Pasareanu (NASA Ames, RSE)
Burak Kadron (UC Santa Barbara, NASA intern)

Boeing TaxiNets

Neural networks designed to take a single picture of the runway as input and return the plane's position w.r.t. the middle of the runway

- Return 2 numerical outputs

  Cross track error (y0) : The distance of the plane from the middle line

  Heading error (y1): The angle of the plane w.r.t. the runway

- A simulator captures an image every 0.1 seconds to use for training and testing

Our Goal

- Analyze the model behavior w.r.t

  Correctness properties: Using error bounds

  $|y0\text{-}y0ideal| < 1.5m$, $|y1\text{-}y1ideal| < 5°$

  Safety properties: Using runway dimensions

  $|y0| < 10.0m$, $|y1| < 90°$
- Understand (**explain**) the behavior of the model
- Provide **guarantees** for consistent behavior of the model



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

Motivation

Explainability

- It is not well understood why a network gives a particular output impeding certification
- Most existing techniques (e.g. DeepLift, LIME, SHAP) identify important pixels in an image impacting the output of a classification model
- Our goal is to identify *generic* **features of an image** that impact network behavior w.r.t output constraints for **regression models**

Guarantees

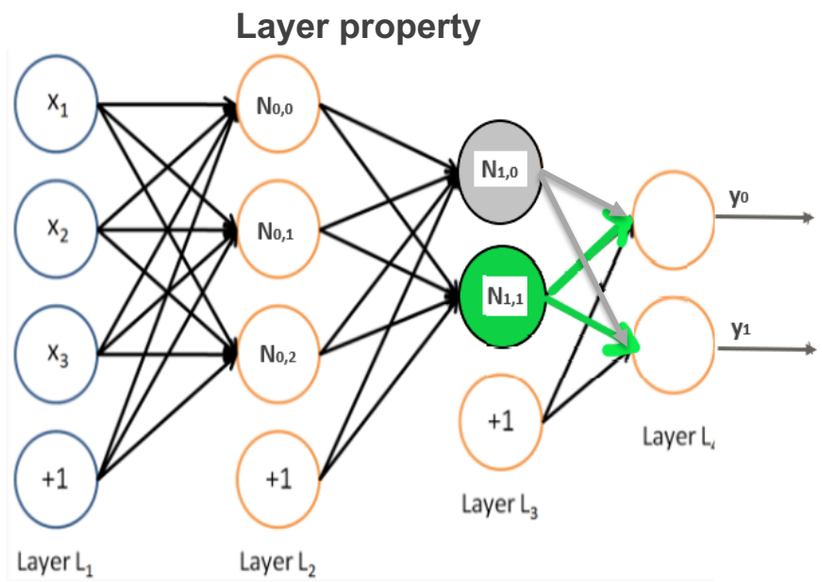
- Existing techniques have explored local robustness around individual inputs (e.g. Marabou)
- Our goal is to identify input regions that are **semantically similar and potentially larger than local neighborhoods**

Prophecy: Property Inference For Neural Networks [1]

- Given a trained Deep Neural Network (DNN) model, **Prophecy** aims to decompose the complex model into a set of compact **properties** amenable to analysis

- Each property is an assume-guarantee rule, **Pre => Post**
  - **Pre**: A constraint in terms of **(on/off) neuron activation patterns**
  - **Post**: A **property of the output** of the network
  - Example layer property: **(off(N1,0) /\ on(N1,1)) => (y1 > y0)**

- **Decision tree** learning over the activation patterns of the training data at given layer/s to extract rules w.r.t Post



*ReLU(x) = x (**on**) if x > 0; 0 (**off**) otherwise*

[1] Divya Gopinath, Hayes Converse, Corina S. Pasareanu, Ankur Taly: **Property Inference for Deep Neural Networks.** ASE 2019

Prophecy on Boeing TaxiNet

jupyter Boeing TaxiNet Visualization Last Checkpoint: 3 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python [conda env:div-env1]

Run Code

```
#model5.summary()
```

In [9]:
```
if MODEL == 'SimpleModel':
    model_path = './Training_Runs/TaxiNet-Simple/SimpleModel.h5'
    model_path = './Training_Runs/TaxiNet-MobileNet/MobileNetV2.h5'
else:
    model_path = './Training_Runs/TaxiNet-FullyConnected/FullyConnectedModel.h5'

print('Loading the ORIG model and printing the summary:')
model=load_model(model_path,custom_objects={'smoothL1':smoothL1})
model.summary()
```

```
Loading the ORIG model and printing the summary:
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 98, 178, 24)       1800
_________________________________________________________________
batch_normalization_1 (Batch (None, 98, 178, 24)       96
_________________________________________________________________
activation_1 (Activation)    (None, 98, 178, 24)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 47, 87, 36)        21600
_________________________________________________________________
batch_normalization_2 (Batch (None, 47, 87, 36)        144
_________________________________________________________________
activation_2 (Activation)    (None, 47, 87, 36)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 22, 42, 48)        43200
_________________________________________________________________
batch_normalization_3 (Batch (None, 22, 42, 48)        192
_________________________________________________________________
activation_3 (Activation)    (None, 22, 42, 48)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 20, 40, 64)        27648
_________________________________________________________________
batch_normalization_4 (Batch (None, 20, 40, 64)        256
_________________________________________________________________
activation_4 (Activation)    (None, 20, 40, 64)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 18, 38, 64)        36864
_________________________________________________________________
batch_normalization_5 (Batch (None, 18, 38, 64)        256
_________________________________________________________________
activation_5 (Activation)    (None, 18, 38, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 43776)             0
_________________________________________________________________
dropout_1 (Dropout)          (None, 43776)             0
_________________________________________________________________
dense_1 (Dense)              (None, 100)               4377700
_________________________________________________________________
dense_2 (Dense)              (None, 50)                5050
```

```
activation_5 (Activation)    (None, 18, 38, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 43776)             0
_________________________________________________________________
dropout_1 (Dropout)          (None, 43776)             0
_________________________________________________________________
dense_1 (Dense)              (None, 100)               4377700
_________________________________________________________________
dense_2 (Dense)              (None, 50)                5050
_________________________________________________________________
dense_3 (Dense)              (None, 10)                510
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 22
=================================================================
```

jupyter Boeing TaxiNet Visualization Last Checkpoint: 2 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted

Run Code

Loading the Data and Model

In [7]:
```
LAYER = 1
if LAYER == 1:
    layer = 'dense_1'
elif LAYER == 2:
    layer = 'dense_2'
elif LAYER == 3:
    layer = 'dense_3'
else:
    layer = 'ALL'

PLOT_DATA_BOEING = False
PLOT_DATA_KJ = False

BOEING_TRAIN = True
BOEING_TEST = False

KJ_TRAIN = False
KJ_TEST  = False

MODEL = 'SimpleModel'
```

jupyter Boeing TaxiNet Visualization Last Checkpoint: 2 hours ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted

Run Code

In [8]:
```
## Loading the data
eval_folder = '../20200630_TaxiNet_data/'
#Loading the data
if BOEING_TRAIN:
    x=np.load(eval_folder+'X_train.npy')
    y=np.load(eval_folder+'Y_train.npy')
    x_test=np.load(eval_folder+'X_test.npy')
    y_test=np.load(eval_folder+'Y_test.npy')


print("Shape of X:", x.shape)
print("Shape of X test:", x_test.shape)
#print("Shape of X1:", x1.shape)
#print("Shape of X2:", x2.shape)
```

```
Shape of X: (13885, 200, 360, 3)
Shape of X test: (2777, 200, 360, 3)
```

Experiments on Boeing TaxiNet

- Patterns extracted w.r.t 3 correctness properties of the outputs
  - |y0-y0ideal|<=1.5,
  - |y1-y1ideal|<=5.0,
  - |y0-y0ideal|<=1.0 and |y1-y1ideal|<=5.0
- Patterns extracted w.r.t the safety property of the output
  - |y0|<=4.0

jupyter Boeing TaxiNet Visualization Last Checkpoint: 4 hours ago Autosave Failed!

File Edit View Insert Cell Kernel Widgets Help Trusted

Run Code

```
if tensor is None or tensor == 'dense_4':
    outputs_test = []

    for img in inputs:
        outputs_test.append(model.predict(np.expand_dims(img/255.0,axis=0)))

    outputs_test    = np.squeeze(np.asarray(outputs_test))
    #Scale the predictions
    outputs_cte     = outputs_test[:,0]*8.0
    outputs_heading = outputs_test[:,1]*35.0
    #Absolute value of the predictions
    abs_outputs_cte = np.abs(outputs_cte)
    abs_outputs_heading = np.abs(outputs_heading)
    #Absolute Error between Ground Truth and Prediction for Cross Track Error
    if (TEST == False):
        error_cte=np.squeeze(y[:,0])-outputs_cte
    else:
        error_cte=np.squeeze(y_test[:,0])-outputs_cte
    abs_error_cte=np.abs(error_cte)
    #Absolute Error between Ground Truth and Prediction for Heading Error
    if (TEST == False):
        error_heading     = np.squeeze(y[:,1])-outputs_heading
    else:
        error_heading     = np.squeeze(y_test[:,1])-outputs_heading
    abs_error_heading = np.abs(error_heading)

# CORRECTNESS PROPERTY
ret = []
for i in range(0,len(abs_outputs_cte)):
    # |y0-y0ideal|<=1.0 (abs_error_cte)
    if ((abs_error_cte[i] <= 1.0)):
    # |y1-y1ideal|<=5.0 (abs_error_heading)
    #if ((abs_error_heading[i] <= 5.0)):
    # |y0-y0ideal|<=1.0 && |y1-y1ideal|<=5.0
    #if ((abs_error_cte[i] <= 1.0) && (abs_error_heading[i] <= 5.0)):
        ret.append(1)
    else:
        ret.append(0)
return abs_outputs_cte, ret

# SAFETY PROPERTY
#return (abs_outputs_cte <= 10.0).astype('int')
#return (abs_outputs_heading <= 90).astype('int')
```

Experiments on Boeing TaxiNet

In [39]:
```
invariants = get_all_invariants(basic_estimator)
#invariants_sequence_stats(invariants, None, layer, suffixes = train_suffixes, label = train_predictions,random=False)
#invariants_sequence_stats1(invariants, None, layer, suffixes = train_suffixes, label = train_predictions,random=False)
describe_invariants_all_labels(invariants, None, layer, suffixes = train_suffixes, label = train_predictions)
```

```
  0%|          | 0/1029 [00:00<?, ?it/s]
Obtained all paths
100%|██████████| 1029/1029 [00:00<00:00, 1219536.26it/s]
 PRINTING PURE RULES WITH SUPPORT MORE THAN 10:
Class: 1 , Rule:(neurons: [53, 33, 71, 64, 67] ,signature: [0, 0, 0, 0, 0] ), Support: 1792 , Num misclassified 0
TEST PRECISION: 93.75 , TEST RECALL: 0.5651846269781462

Consecutive Subsequence Lengths by decreasing length: [236, 88, 85, 82, 80, 79, 75, 66, 62, 59, 53, 51, 49, 49, 49, 48, 45, 38,
34, 31, 29, 28, 27, 27, 23, 20, 18, 18, 18, 18, 16, 15, 14, 14, 12, 8, 8, 7, 7, 7, 7, 6, 5, 4, 4, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2,
2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Generating Images and Activation Visualizations for sequence 0 of length 236

Class: 0 , Rule:(neurons: [53, 29, 20, 49, 15, 95, 25] ,signature: [1, 0, 1, 0, 0, 0, 0] ), Support: 403 , Num misclassified 0
TEST PRECISION: 66.66666666666666 , TEST RECALL: 1.6260162601626018
Consecutive Subsequence Lengths by decreasing length: [34, 31, 29, 29, 29, 28, 23, 23, 21, 19, 19, 19, 18, 17, 17, 12, 9, 7, 3,
3, 2, 2, 2, 2, 1, 1, 1]
Generating Images and Activation Visualizations for sequence 0 of length 34
```

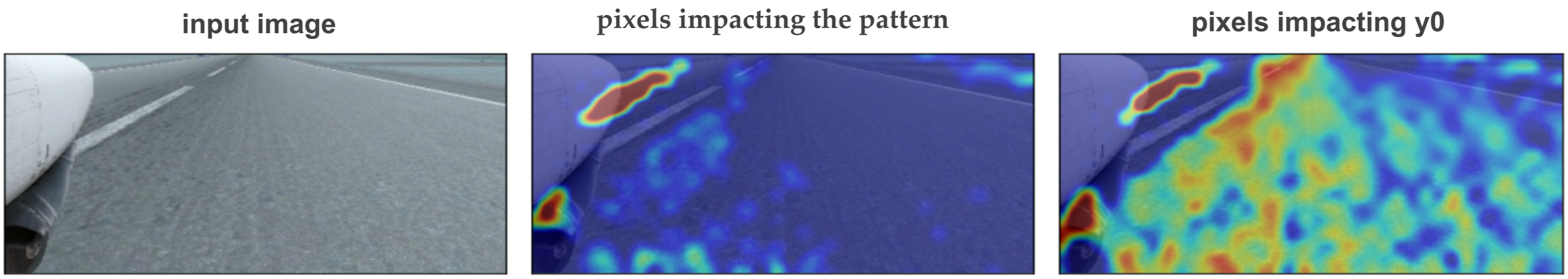
Patterns for Explainability

- A layer 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 y0 (cross-track error)

$$off(N_{1,53}) \wedge off(N_{1,33}) \wedge off(N_{1,71}) \wedge off(N_{1,64}) \wedge off(N_{1,67}) \Rightarrow |y0 - y0ideal| \leq 1.5$$

  - 1792 train set images satisfy this pattern and the output property (25.36% recall, 100% precision)

- 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++ [2])
  - Identifies portion of the image impacting network's behavior w.r.t the cross-track error output



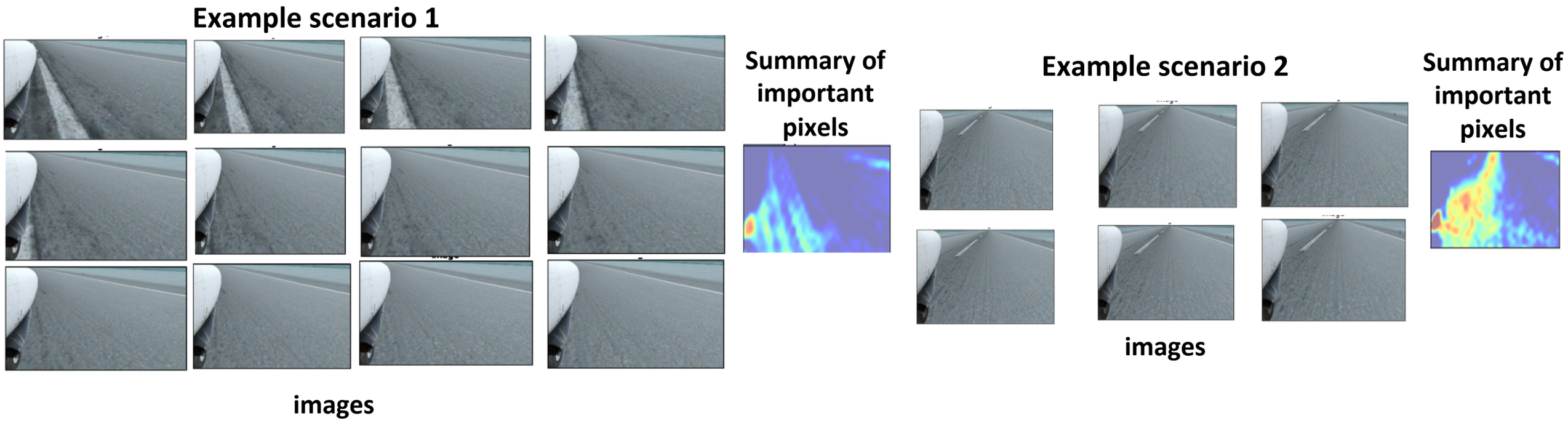
  - Highlighting pixels that impact the output variable y0 (aka existing attribution approaches) is not as helpful

[2]: Chattopadhay, Aditya, et al. "Grad-cam++: Generalized gradient-based visual explanations for deep convolutional networks." WACV 2018

Explaining Correct Behavior using Image Sequences

- We **focus on images in a sequence** satisfying the same pattern
  - Images in a sequence have similar context which would help with better identification of the feature represented by the pattern
  - 44 sequences (length > 5) satisfy the example pattern

- Every sequence represents **a scenario**
  - The summary of important pixels (average GradCAM values across all images) represents the feature for the scenario
  - The feature; distance between the center line of the runway and the airplane; is relevant for cross-track error determination enabling the network to produce the correct output for this scenario



Explaining Incorrect Behavior

- Pattern from dense layer 1 for **violation** of the correctness property w.r.t y0 (cross-track error)

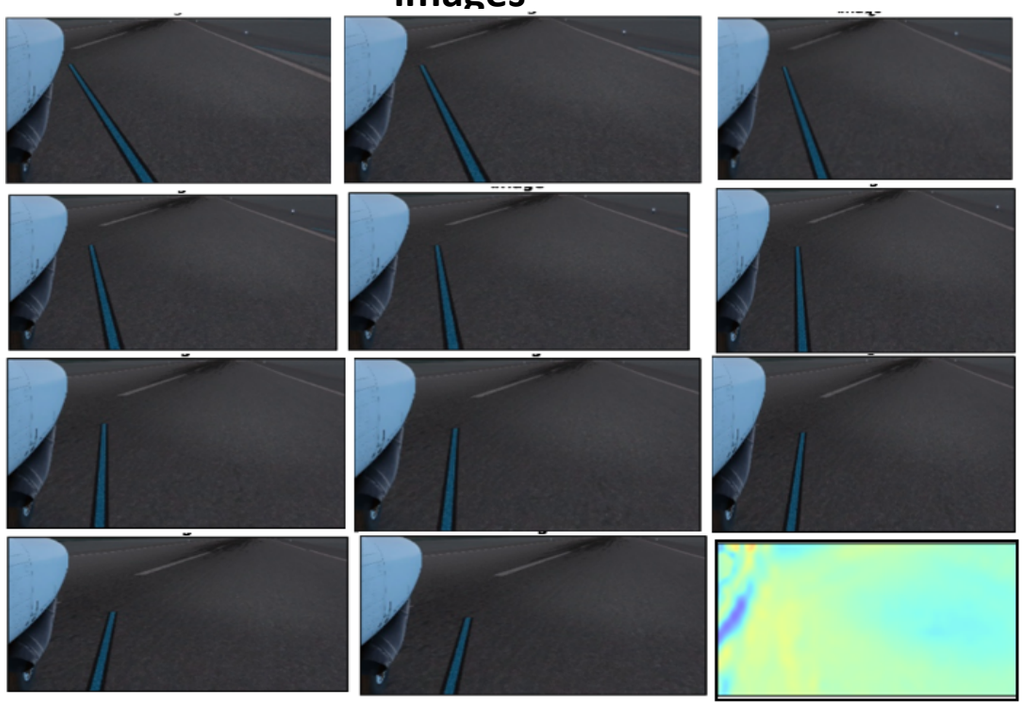
  on($N_{1,53}$) /\ off($N_{1,29}$) /\ on($N_{1,20}$) /\ off($N_{1,49}$) /\ off($N_{1,15}$) /\ off($N_{1,95}$) /\ off($N_{1,25}$)=> |y0-y0ideal| > 1.5, Support: 403

  - 18 sequences (length > 5) satisfy the example pattern

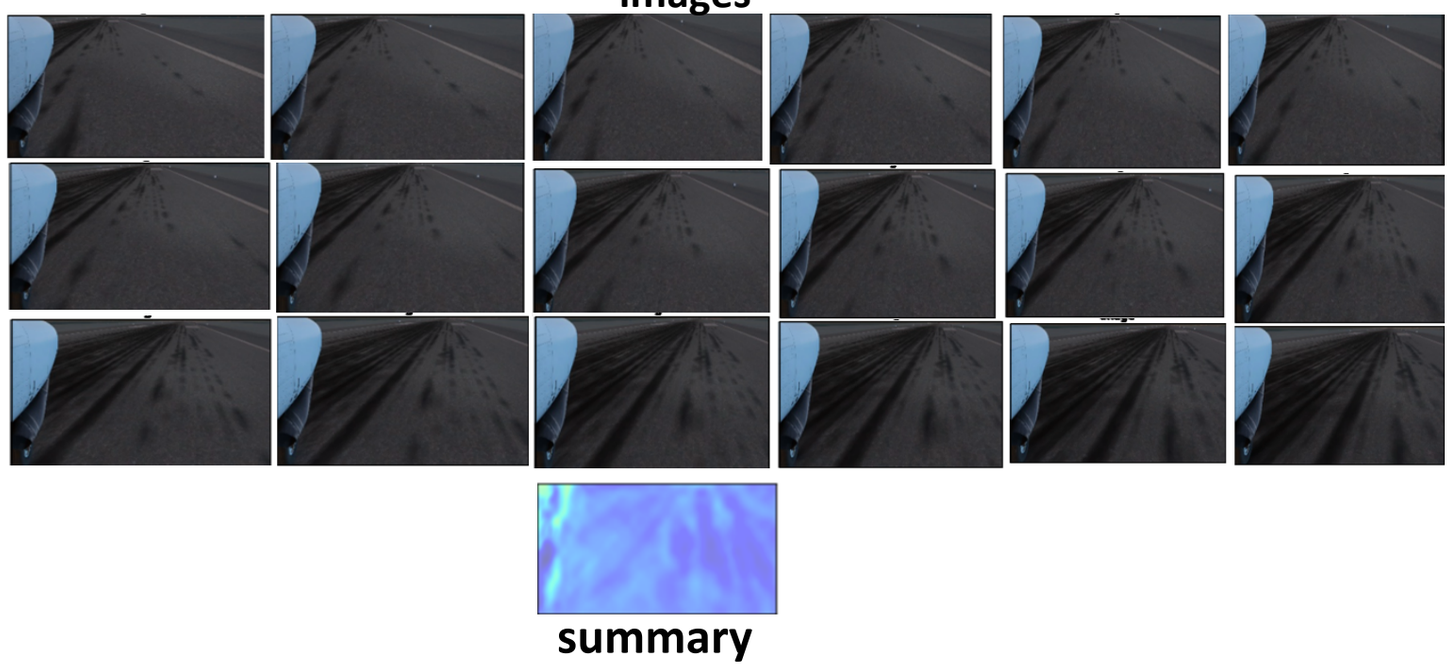
- Scenario 1: Highlighted pixels indicate that the noise (blue line) interferes with correct determination of the cross-track error

- 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



Example scenario 2



Obtaining guarantees of consistent behavior

- All inputs satisfying the pattern potentially satisfy the output property
- Patterns **can be proved formally (e.g. Marabou [3]) giving guarantees of consistent behavior**
- Represent input regions with similar semantic behavior

- Unable to obtain **formal proofs** for Boeing TaxiNet patterns since the architecture (convolutions, eLU activations) cannot be handled by existing off-the-shelf solver Marabou
- We validated the patterns **statistically by calculating the F1-score** of each pattern on a separate test set (2777 instances)

Best results w.r.t F1

Satisfaction of correctness

PROPERTY	LAYER	PRECISION(%)	RECALL(%)	F1
|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

Violation of correctness

PROPERTY	LAYER	PRECISION(%)	RECALL(%)	F1
|Y1-Y1ideal|>5.0	ALL	73.91	23.94	36.17
|Y1-Y1ideal|>5.0	1	86.67	18.31	30.23
|Y1-Y1ideal|>5.0	ALL	73.33	15.49	25.58
|Y0-Y0ideal| > 1.0 or |Y1-Y1ideal| > 5.0	2	87.50	9.21	16.67
|Y0-Y0ideal| > 1.0 or |Y1-Y1ideal| > 5.0	1	70.00	9.21	16.28
|Y0-Y0ideal| > 1.0 or |Y1-Y1ideal| > 5.0	1	100.00	8.55	15.76
|Y0-Y0ideal| > 1.0 or |Y1-Y1ideal| > 5.0	ALL	100.00	7.24	13.50

[3] Guy Katz, Derek A. Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljic, David L. Dill, Mykel J. Kochenderfer, Clark W. Barrett: **The Marabou Framework for Verification and Analysis of Deep Neural Networks.** CAV (1) 2019

Formal proofs for KJ-TaxiNet

- We highlight the potential of Prophecy in **giving guarantees on a simpler network** KJ-Taxinet [4]
  - Smaller network for TaxiNet application taking a down-sampled image of the runway as input that can be handled by **Marabou**
  - Extracted patterns at each intermediate layer and all layers together for both safety and correctness properties using the test set

- We focus on a sequence of images satisfying the same pattern
  - For each sequence satisfying a pattern w.r.t the safety property (e.g. $|y0| < 10.0$) we formulated the check for **Marabou** as shown, **for all x s.t. x in [xmin, xmax] => |y0| < 10.0**
  - For $|y0| < 10.0$, **we obtained proofs for 33** out of 111 sequences with **at least 5 images**, the longest subsequence where we can obtain proofs has 17 images
  - For $|y0| < 8.0$m, $|y0| < 5.0$m, Longest sequence on which proof was obtained was 9 images long
  - We attempted proofs on random sequences of images satisfying the output property, but they failed

- Pattern when proved represents a **temporal property or invariant** over the sequence

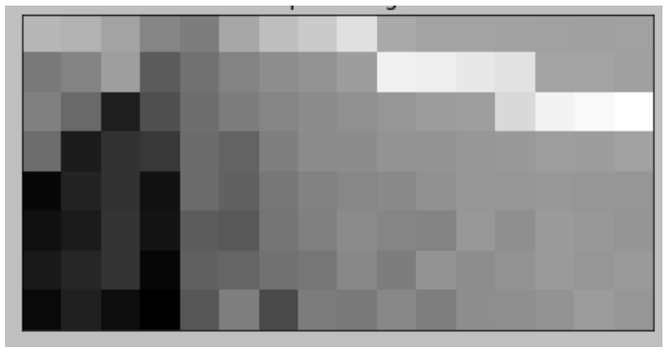
[4]: K. D. Julian, R. Lee, and M. J. Kochenderfer, “Validation of image based neural network controllers through adaptive stress testing,” arXiv preprint arXiv:2003.02381, 2020

Counter-examples for Safety Property on KJ-TaxiNet

- Counter-examples to the proofs represents a scenario of a sequence of images resulting in the violation of the safety property

Counterexample for an image sequence of length 2238 for |y0| < 10.0m

First image



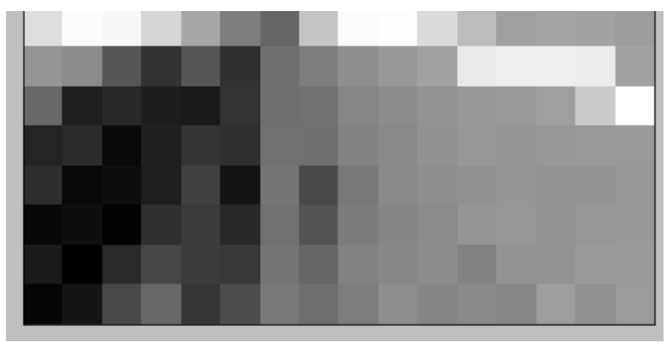
Last image

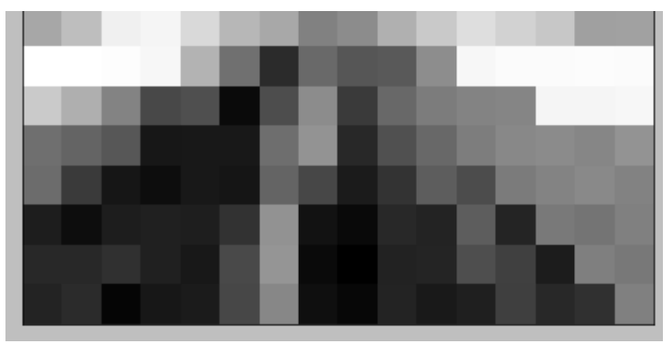


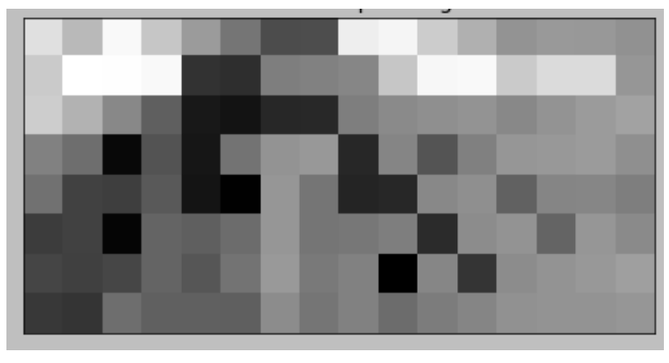
Counter-example image



Counterexample for an image sequence of length 16 for |y0| < 10.0m







Other uses for patterns…

- Improve **train set composition**
  - Reduce redundancy by removing images from sequences satisfying patterns for correctness properties
  - Augment (up-sample) sequences satisfying violation patterns
- Preliminary results
  - Original model provided by Boeing: './Training_Runs/TaxiNet-Simple/SimpleModel.h5':
    MeanAbsoluteError: Cross Track Error: 0.366843082948183 3, Heading Error:1.64113065333068 22
  - Train the same SimpleModel architecture with full train set (13885) and validation set (1388):
    Size: 13885, Time: 79 mins
    MeanAbsoluteError: Cross Track Error: 0.093971292490518 4, Heading Error:0.364928906401234 04
  - **Correct Pattern-based reduction:**
    Size: 11715, Time: 67 mins
    MeanAbsoluteError: Cross Track Error: 0.105235450836488 65, Heading Error: 0.41939568891558 5
    15% reduction in training time, ~13 % reduction in accuracy
  - **Incorrect Pattern-based augmentation**
    MeanAbsoluteError: Cross Track Error:0.087559599950598 62, Heading Error: 0.31508829479443 684
    6% and 13 % increase in accuracy

Future Work

- Use patterns at run-time for predicting **confidence** of the outputs of the regression model
- Patterns as **run-time guards** to identify inputs on which network's behavior may be inaccurate
- Extract rules in terms of **high-level features** (x,y co-ordinates, weather so on — coming from simulator)

Take Away

- Design neural networks for verifiability
  - Start with a small network
  - Simple, pierce-wise linear computations (e.g., ReLU activations) — supported by many verification tools (e.g. Marabou)
  - Downsample/scale down the inputs
  - Verify and add complexity as needed
- Consider generating multiple data sets
  - For different distributions
- More information needed from the simulator
  - Ground truth, high-level parameters used by the image renderer