CS590 Program Analysis for Deep Learning

Xiangyu Zhang

Fall 2019

This Course

- Using program analysis to address deep learning problems
  - Model security, testing, debugging, verification, and optimization
- Target audience
  - With deep learning background
  - Or with program analysis background and interested in various deep learning applications

This Course

- Objective
  - Get familiar with various program analysis techniques and their applications in deep learning
  - Understand the state-of-the-art techniques in the various applications
  - Hands-on experience
  - Presentation skills
  - Potential research paper

This Course

- Project intensive
  - Three small and one large
- Paper presentation, discussion and quiz
- (Take-home) midterm

- Details on the course website

Program Analysis

- Dynamic
- Static
- Symbolic
- Verification

Program Analysis versus AI Model Analysis

- Software debugging versus AI model debugging
- Software security versus AI model security
  - CIA
- Software testing versus AI model testing
- Software verification versus AI model verification
- Software optimization versus AI model optimization
- Integrating software with AI

Program Analysis versus AI Model Analysis

- Software debugging versus AI model debugging
- Software security versus AI model security
  - CIA
- Software testing versus AI model testing
- Software verification versus AI model verification
- Software optimization versus AI model optimization
- Integrating software with AI

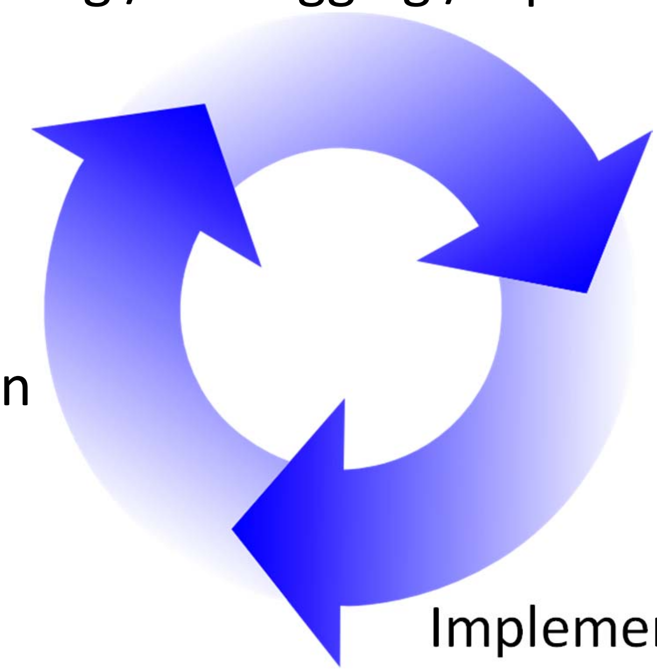
AI Driven Computing

- AI Models are becoming an integral part of modern computing
  - Autonomous vehicles, Apple Face ID, iRobots, Cotana, and computer games

- AI Models are shared/reused just like software components
  - Python face recognition package

bigml OpenML Gradientzoo Predictors.ai

AI Driven System Engineering



AI Models Are Prone to Bugs and Vulnerabilities Just Like Software Components

- AI models are programs with special semantics
  - Python program with vectors (DL semantics are implicitly encoded in weight values)
- Traditional engineering bugs (i.e., bugs in general program semantics)
  - E.g., type errors, data format problems
- Model bugs – *misconducts in the AI model engineering process leading to undesirable consequences* (i.e., bugs in the deep learning semantics)
  - Root causes: biased training data, defective model structure, hyper-parameter(s), optimization algorithms, batch size, loss function, activation function(s)
  - Symptoms: low model accuracy, vulnerable to adversarial sample attacks, back-doors
  - E.g., State-of-the-art pre-trained models can only achieve 80% accuracy on an ImageNet classification challenge; 73% accuracy on Children's Book Test challenge.

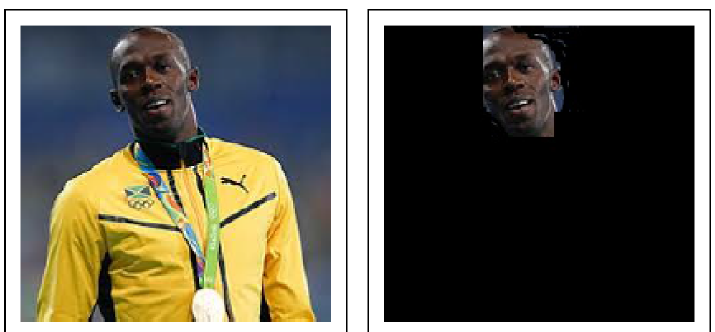
Debugging AI Models

- Debugging is hard
  - DNNs are not human understandable/interpretable
    - Each neuron denotes some abstract feature
  - Lack of scientific way of locating the root causes
    - Trial-and-error
  - Unclear how to fix bugs
    - Cannot directly change weight values
    - Cannot train with failure inducing inputs

AI Model Bugs

- Input related bugs
  - Biased training inputs -- overfitting and underfitting

Stereo-typing defect caused by overfitting



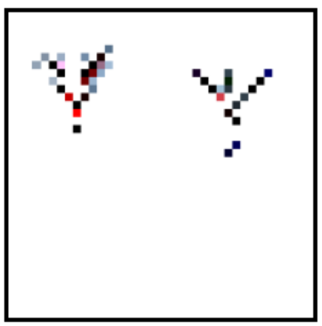
(a) Basketball (78%) (b) Top 1 feature

ResNet110 on ImageNet

AI Model Bugs

- Input related bugs
  - Biased training inputs -- overfitting and underfitting

Model back-doors caused by overfitting

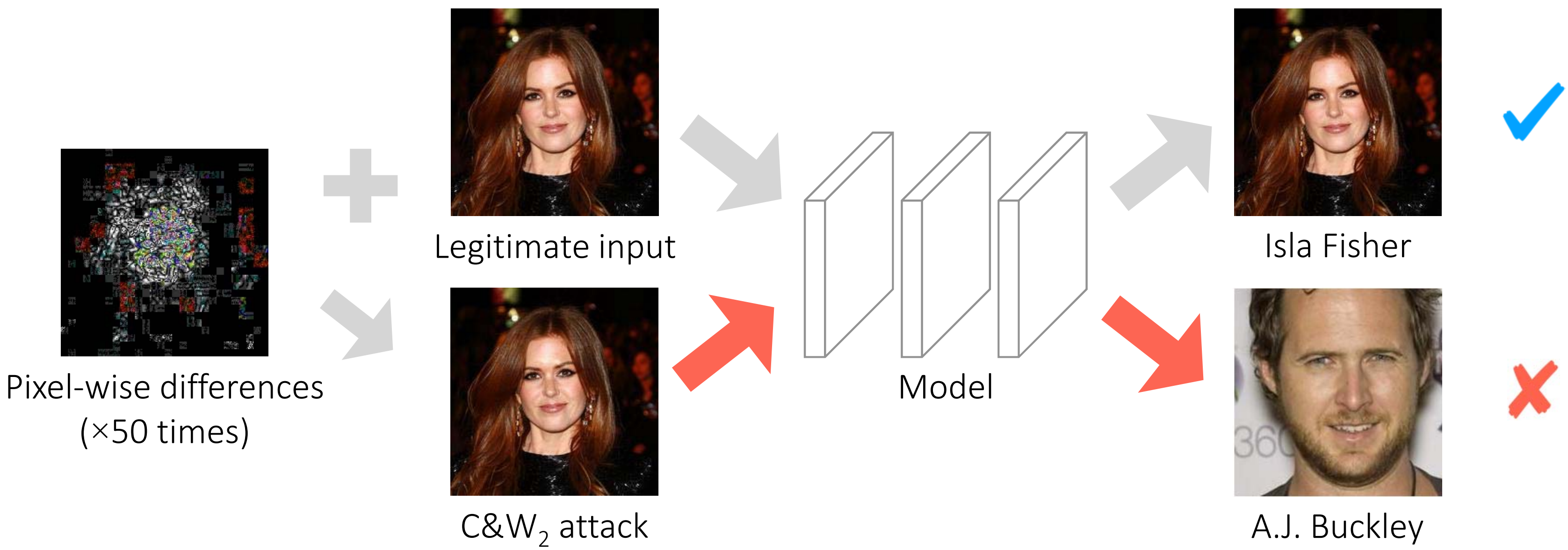


VGG on CIFAR10:
Any image stamped with the pattern cause mis-classification to **DEER**

AI Model Bugs

- Input related bugs
  - Biased training inputs -- overfitting and underfitting

Vulnerabilities to adversarial sample attacks by underfitting



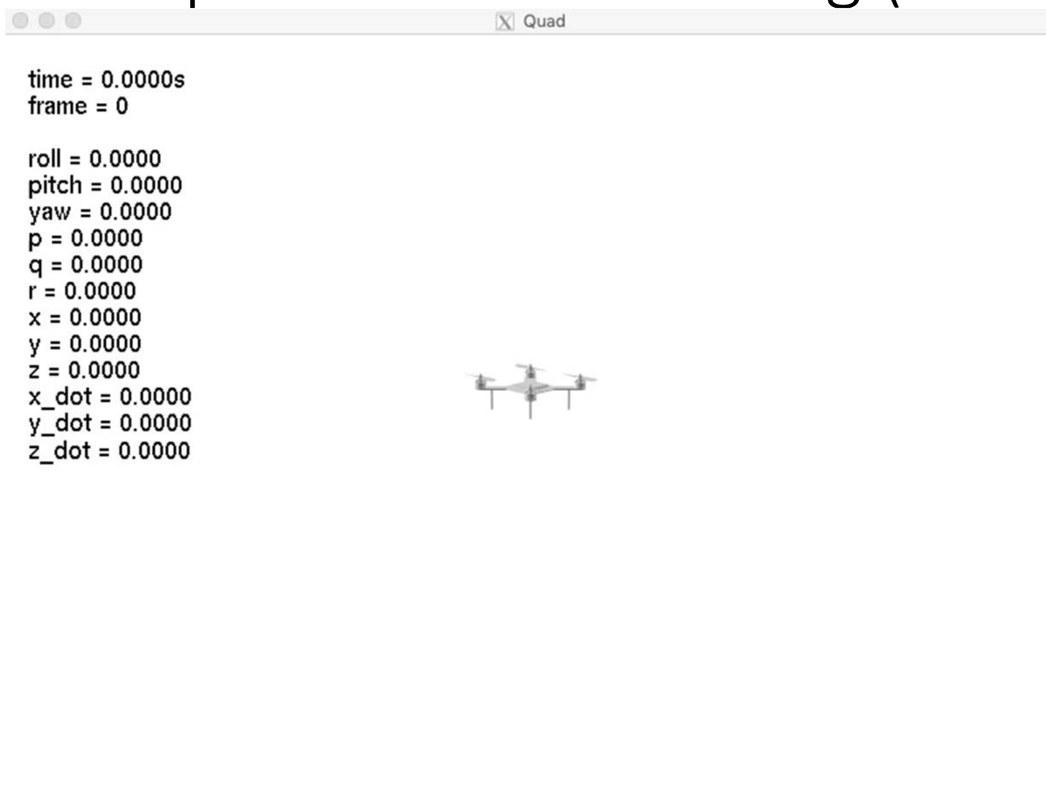
AI Model Bugs

- Input related bugs
  - Biased training inputs -- overfitting and underfitting
  - Inclusion of problematic inputs in the training set leads to difficulty of convergence
    - Using reinforcement learning to train a model to perform integer addition
    - Training does not converge after 24 hours
    - Two problematic training inputs: 12+11= 23   21+11 = 32

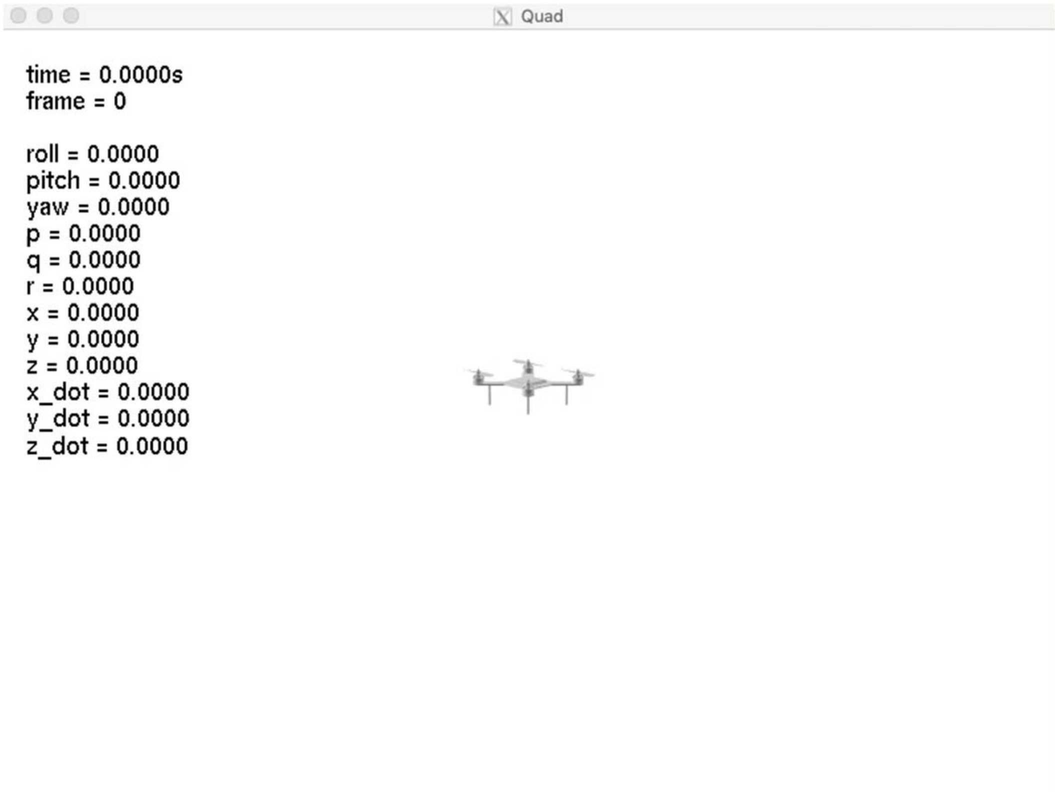
AI Model Bugs

- Input related bugs
  - Biased training inputs -- overfitting and underfitting
  - Inclusion of problematic inputs in the training set leads to difficulty of convergence
    - Training a model to evaluate propositional logic expression
  - Problematic input embedding (for RNN models)
    - Similar embeddings do not entail similar semantics
      - "*new*" and "*create*"
- Structural bugs
  - Redundant/insufficient layers/neurons
  - In-effective structures
    - Forget gates in (LSTM) do not retain/throw-away certain contextual information
  - Suboptimal setting of reward values leading to extremely long training time in reinforcement learning

Drone with a suboptimal reward setting (two weeks training)



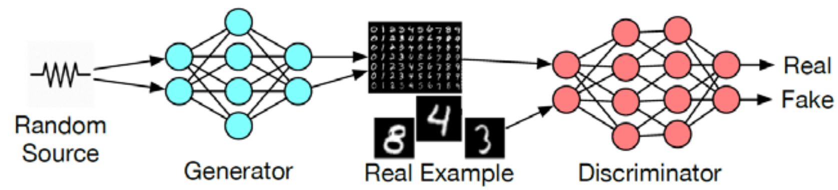
After fixing the reward setting (four hours training)



AI Model Bugs

- Input related bugs
  - Biased training inputs -- overfitting and underfitting
  - Inclusion of problematic inputs in the training set leads to difficulty of convergence
  - Problematic input embedding (for RNN models)
    - Embedding of training inputs does not provide good coverage
    - Similar embeddings do not entail similar semantics
      - General embeddings may not work well for domain-specific applications
- Structural bugs
  - Redundant/insufficient layers/neurons
  - In-effective structures
    - Forget gates in (LSTM) do not retain the appropriate contextual information
  - Suboptimal setting of reward values leading to extremely long training time in reinforcement learning

Prior Work: Data Augmentation & Retraining



- Pre-defined data augmentation techniques
- E.g., crop, soften, brighten, sharpen, mirror, rotate, transparency, contrast

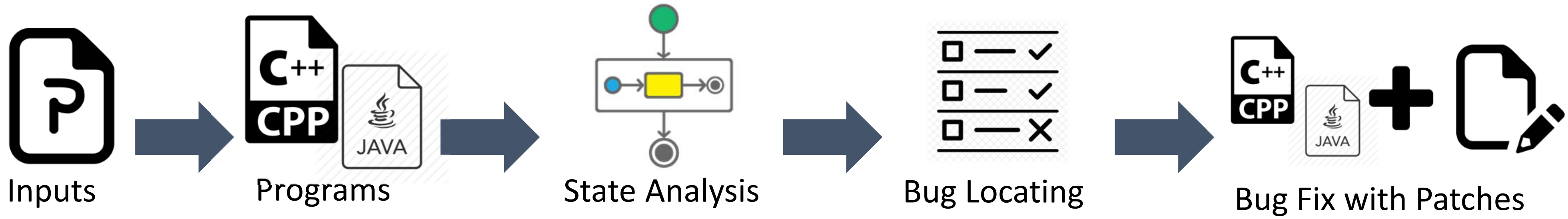
- Generative models, e.g., Generative Adversarial Network (GAN)
- Generate new samples which follow similar distributions with provided training dataset

Brock, Andrew, Jeff Donahue, and Karen Simonyan. "Large scale gan training for high fidelity natural image synthesis." *arXiv preprint arXiv:1809.11096* (2018).

Using GAN is Not That Effective

- Use 14 GANs downloaded from various sources for MNIST to generate inputs
- For each GAN, randomly select 40,000 generated inputs as additional training data to fix a MNIST model that has an underfitting bug for digit 5 (only 74% accuracy)
- 7 GANs fail to improve either digit 5 or the whole model, 4 improve the model but not digit 5, and only 3 can improve both (digit 5 to 83% after 1 hour of training)
  - MODE can improve to 94% in 5 mins
- Root Cause: *does not consider the reasons why a NN misbehaves*

Traditional software defect diagnosis




```
01:def fib(n):
  :   if n == 0 or n == 1:
  :     return n
04:   return fib(n-1)+fib(n-
05:
06:def main():
07:   x = input('Input:'):
08:   print fib(x)
```

Program

```
01:def fib(n):
02:   assert(n>=0)
03:   if n == 0 or n == 1:
04:     return n
05:   return fib(n-1)+fib(n-2)
06:
07:def main():
08:   x = input('Input:'):
09:   print fib(x)
```

Analysis

```
: {x = -1}
: {n = -1}
: {n = -2}
: {n = -3}
: {n = -4}
: {n = -5}
: {n = -6}
ling Run
```

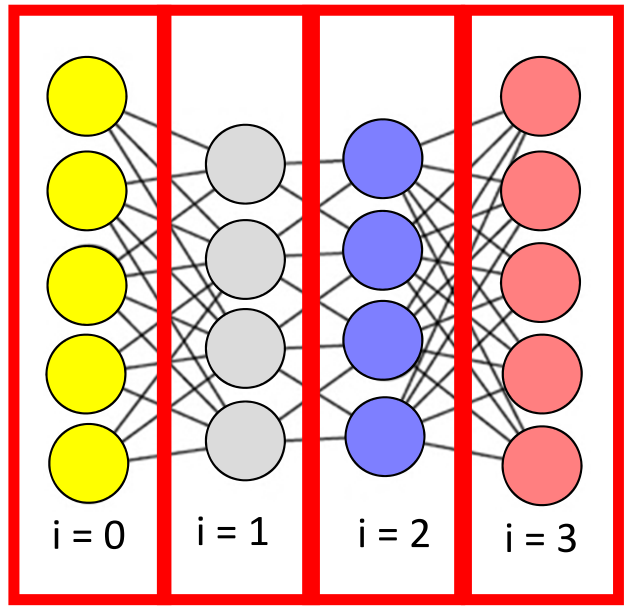
```
main: {x = 6}
fib : {n = 6}
fib : {n = 5}
fib : {n = 4}
fib : {n = 3}
fib : {n = 2}
fib : {n = 1}
Successful Run
```

Execution Traces

Our Idea: From Program Analysis Perspective

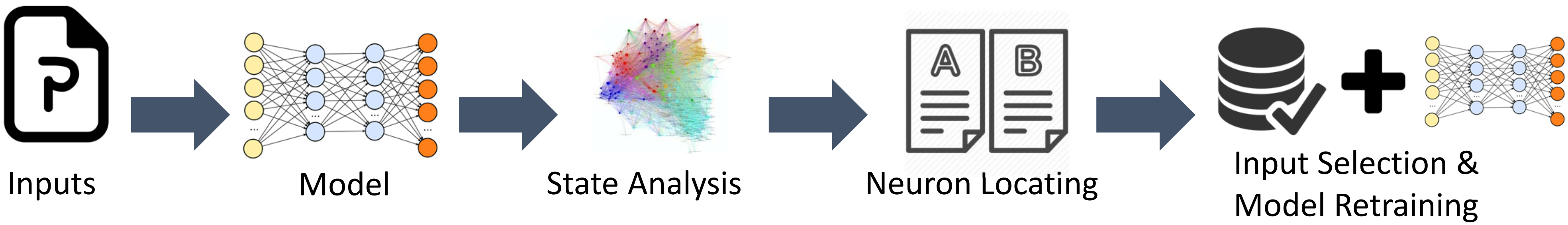
```
def DNN():
  for i in model.layers():
    if(i==0): x_i = input
    else: x_{i+1} = f_i(w_i*x_i+b_i)
```

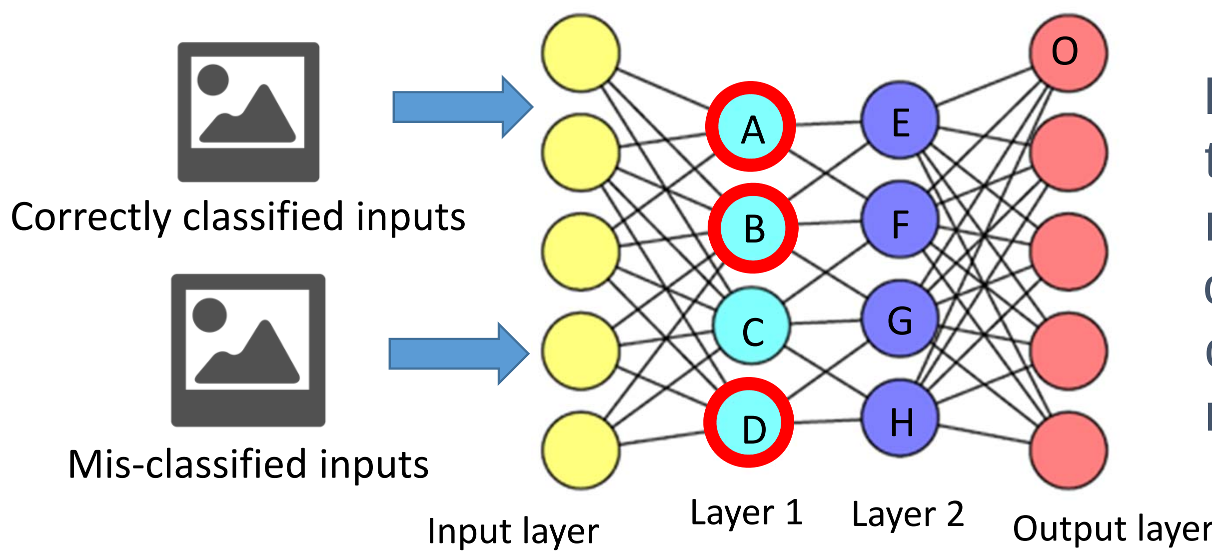
Activation value.



Borrow the ideas from traditional software defect diagnose to understand misclassification.

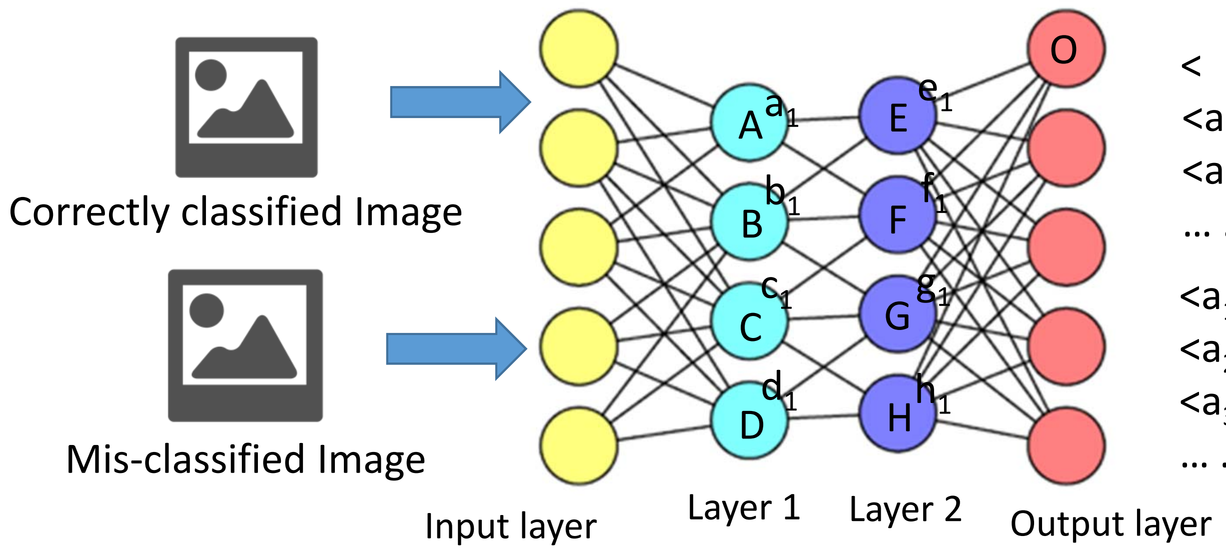
Workflow





Identifying the neurons that contribute to the misclassification based on two datasets: correctly classified and misclassified datasets.

Understanding neuron importance



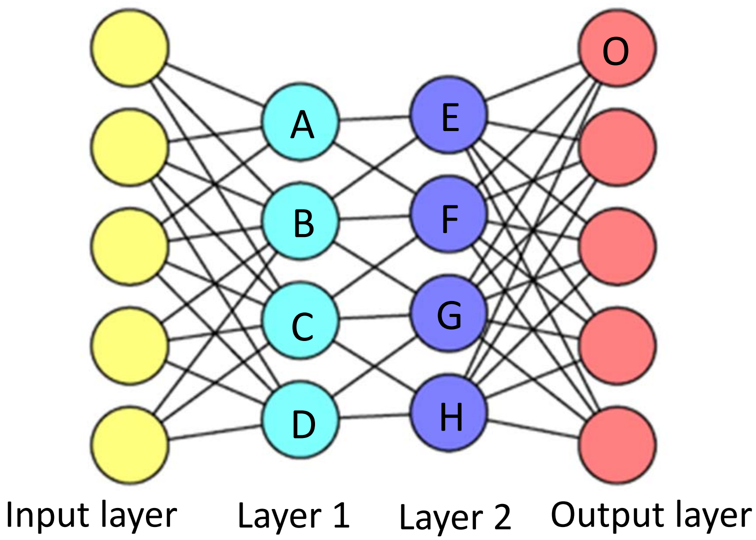
< , , , >< , , , >
$\langle a_2, b_2, c_2, d_2 \rangle$ $\langle e_2, f_2, g_2, h_2 \rangle$
$\langle a_3, b_3, c_3, d_3 \rangle$ $\langle e_3, f_3, g_3, h_3 \rangle$
... ...

$\langle a_1', b_1', c_1', d_1' \rangle$ $\langle e_1', f_1', g_1', h_1' \rangle$
$\langle a_2', b_2', c_2', d_2' \rangle$ $\langle e_2', f_2', g_2', h_2' \rangle$
$\langle a_3', b_3', c_3', d_3' \rangle$ $\langle e_3', f_3', g_3', h_3' \rangle$
... ...

Find the important neurons: Linear Regression

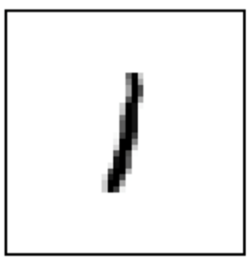
- $O = \boldsymbol{\beta}_1 E + \boldsymbol{\beta}_2 F + \boldsymbol{\beta}_3 G + \boldsymbol{\beta}_4 H + \ldots$
- $= \langle e_1, e_2, e_3, e_4, \ldots \rangle$
- Weight value $\boldsymbol{\beta}_{1,} \boldsymbol{\beta}_{2,} \boldsymbol{\beta}_3, \boldsymbol{\beta}_4$ represent the importance of each neuron

Understanding misclassification



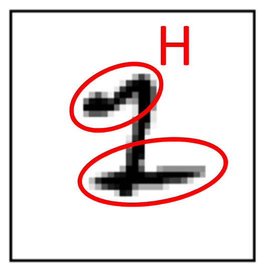
Linear Regression

- $O = \boldsymbol{\beta}_1 E + \boldsymbol{\beta}_2 F + \boldsymbol{\beta}_3 G + \boldsymbol{\beta}_4 H + \ldots$
  - $E = \langle e_1, e_2, e_3, e_4, \ldots \rangle$
- $O' = \boldsymbol{\beta}_1' E' + \boldsymbol{\beta}_2' F' + \boldsymbol{\beta}_3' G' + \boldsymbol{\beta}_4' H' + \ldots$
  - $E' = \langle e_1', e_2', e_3', e_4', \ldots \rangle$
- $\text{Diff} = \langle \boldsymbol{\beta}_1', \boldsymbol{\beta}_2', \boldsymbol{\beta}_3', \boldsymbol{\beta}_4' \rangle - \langle \boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \boldsymbol{\beta}_3, \boldsymbol{\beta}_4 \rangle$
- Diff tells us the different importance level of each neuron on the two datasets



True: 1, Prediction: 1

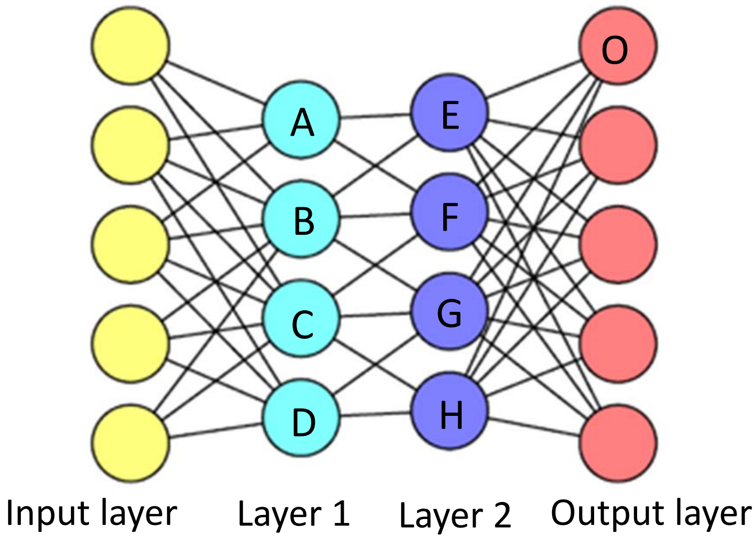
Important neurons: {E, F}



True: 1, Prediction: 2

Important neurons: {E, F, H}

Understanding Misclassification

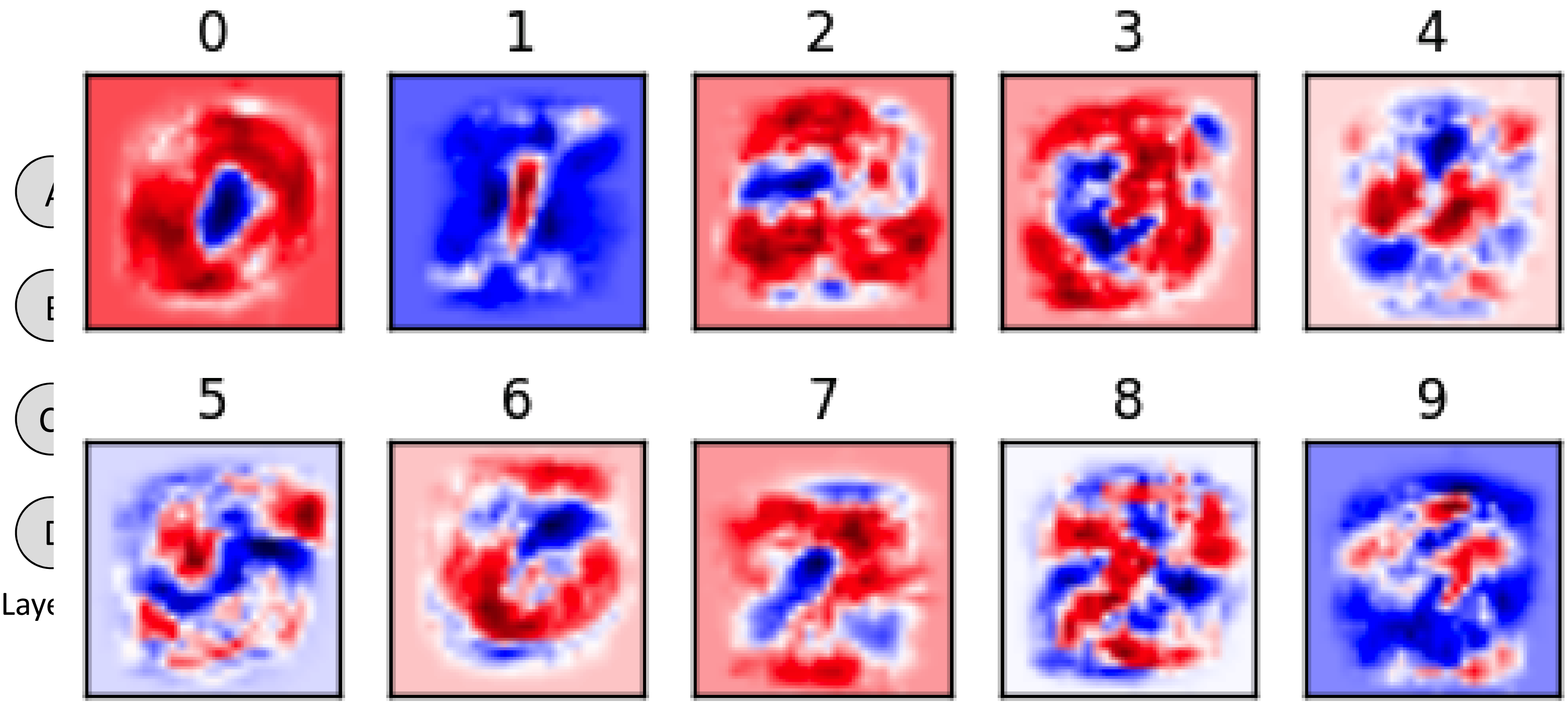


$\langle a_1, b_1, c_1, d_1 \rangle \langle e_1, f_1, g_1, h_1 \rangle$
$\langle a_2, b_2, c_2, d_2 \rangle \langle e_2, f_2, g_2, h_2 \rangle$
$\langle a_3, b_3, c_3, d_3 \rangle \langle e_3, f_3, g_3, h_3 \rangle$
… …
$\langle a_1', b_1', c_1', d_1' \rangle \langle e_1', f_1', g_1', h_1' \rangle$
$\langle a_2', b_2', c_2', d_2' \rangle \langle e_2', f_2', g_2', h_2' \rangle$
$\langle a_3', b_3', c_3', d_3' \rangle \langle e_3', f_3', g_3', h_3' \rangle$
… …

Using SoftMax instead of linear regression

- SoftMax has similar meanings with linear regression. It is a more generalized analysis method, and can model non-linear functions and high dimensional space.

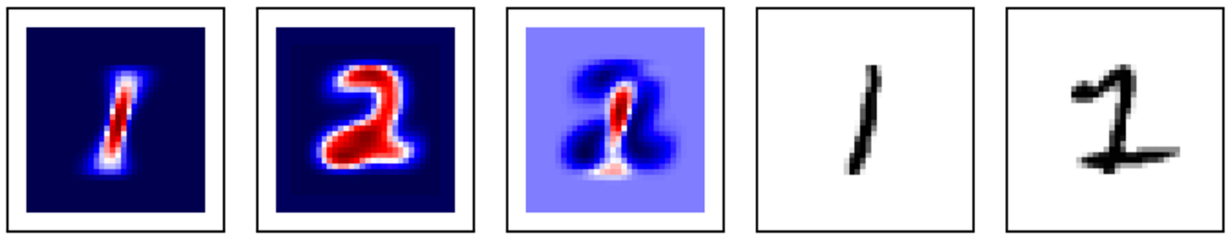
Visualization of Importance



NOTE: Heat-map versus Gradient

- Input gradients gauge *how much output changes can be induced by input value (neuron activation value) changes*
- However, gradients are computed for a given input, whereas heat-map denotes aggregated importance over a large set of inputs
  - We cannot fix model bug by looking at one particular input

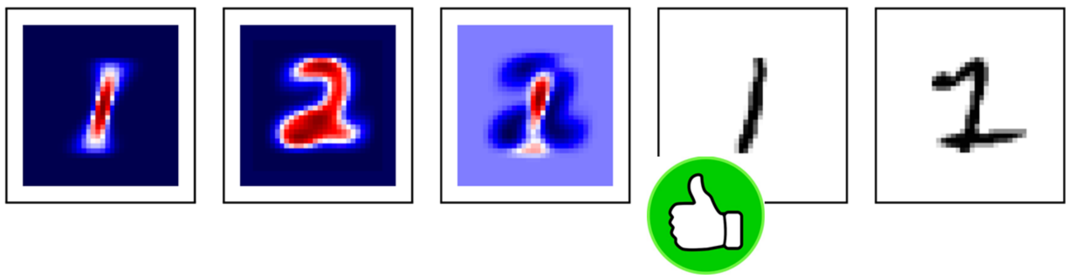
Step 2: Differential analysis to identify underfitting root cause: other digits are mis-classified to digit 1



HM_1 *HM_2* *$DHM_{1,2}$* *correct* *mis-classified to 2*

- **Using heatmaps to find unique features**
- HM_1: trained with all correctly classified images for output label 1
- HM_2: trained with all correctly classified images for output label 2
- $DHM_{1,2}$: highlight unique features of output label 1

Step 3: Input Selection

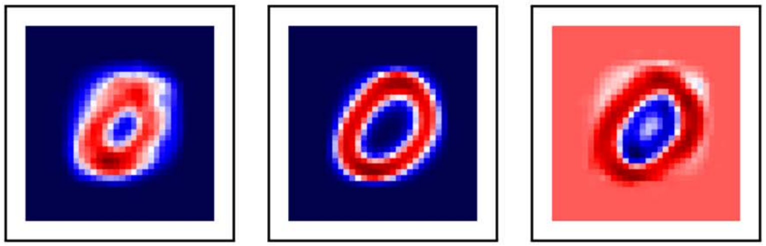


- ***DHM*** (differential heatmap) is a vector pointing to the most promising direction to fix the problem

Score = A(I) • DHM, A(I) is the activation values of input I

- Dot product measures the significance of the vector ***A(I)*** along the direction ***DHM***
- Select the ones with high ***Score*** values

Differential analysis to identify overfitting root cause: some digit 0s are mis-classified as other digit



HM_1 *HM_2* *$DHM_{2,1}$*

- HM_1: trained with all the correctly classified images for output label 0
- HM_2: trained with all the label 0 images mis-classified to others
- $DHM_{2,1}$:The red regions in the ***DHM*** denote the features helpful for generalization, the blue regions denote the overfitted features.
    - Larger-sized 0 are needed

Evaluation Setup

- All models are pre-trained models downloaded from Github and TensorFlow model zoos etc.
- New input source
  - GAN generated inputs
  - Real inputs from other datasets on the same deep learning task
- No. of new inputs (for retraining) is decided by tasks type
  - Simple: 2,000; middle: 4,000; complex: 6,000
- Retraining is using the same hyper-parameters used to train these downloaded models (extracted from their documents)
- Comparison: random selection
  - GAN generated inputs
  - Real inputs from other datasets

Comparison using gan inputs

Dataset	Ori. Acc.	# New Sample	MODE Acc.	Random Acc.
MNIST	95.2%	2,000	97.4%	94.8%
	93.4%	2,000	96.8%	94.3%
Fashion MNIST	87.6%	2,000	92.3%	88.9%
	91.6%	2,000	92.6%	88.5%
CIFAR-10	87.3%	4,000	93.2%	87.3%
	88.4%	4,000	92.8%	88.2%
Average	90.58%		94.18%	90.33%

Real inputs from other datasets

Task	Original Model Acc.	# New Samples	MODE Model Acc.	Random Model Acc.
Face Recognition	76%	2,000	88%	79%
	72%	2,000	85%	84%
Object Detection	83%	6,000	89%	84%
	82%	6,000	88%	84%
Age Classification	33%	4,000	46%	32%
	25%	4,000	42%	32%

AI Model Bugs

- Input related bugs
  - Biased training inputs -- overfitting and underfitting

  - Inclusion of problematic inputs in the training set leads to difficulty of convergence

  - Problematic input embedding (for RNN models)
    - Embedding of training inputs does not provide good coverage
    - Similar embeddings do not entail similar semantics
      - General embeddings may not work well for domain-specific applications

- Structural bugs
  - Redundant/insufficient layers/neurons
  - In-effective structures
    - Forget gates in (LSTM) do not retain the appropriate contextual information
  - Suboptimal setting of reward values leading to extremely long training time in reinforcement learning

Program Analysis versus AI Model Analysis

- Software debugging versus AI model debugging
- Software security versus AI model security
  - CIA
- Software testing versus AI model testing
- Software verification versus AI model verification
- Software optimization versus AI model optimization
- Integrating software with AI

CIA – Confidentiality, Integrity, Availability

CIA in Software

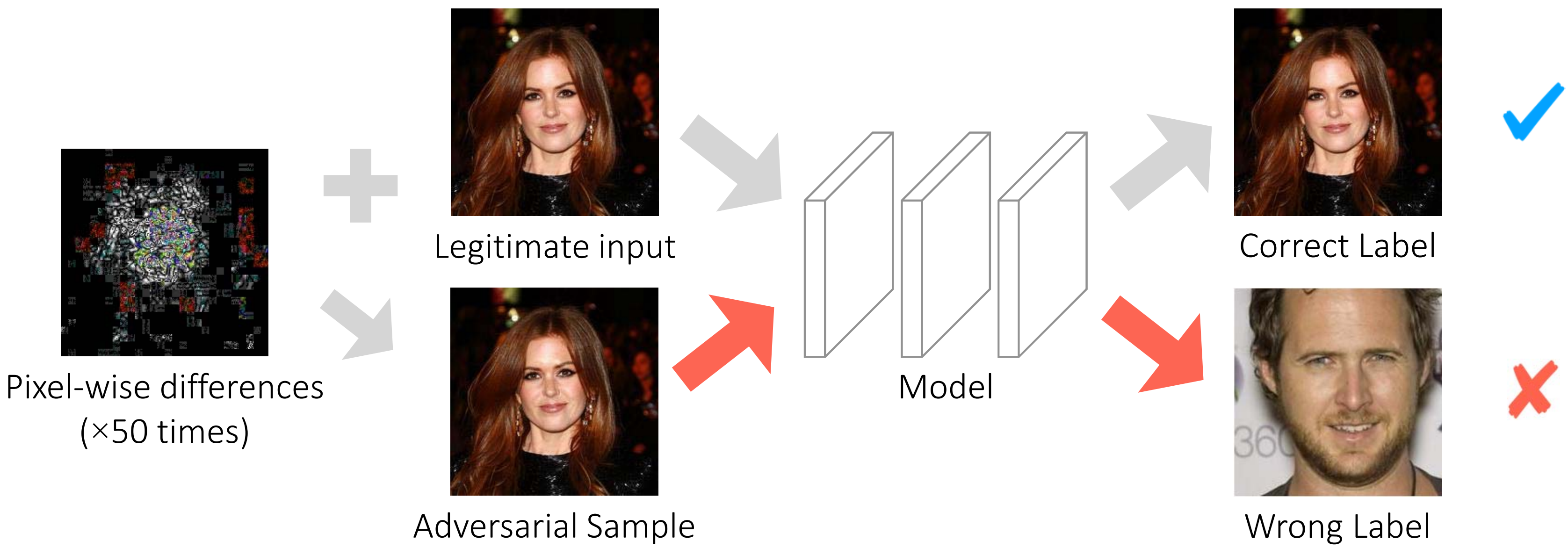
- Confidentiality
  - Information leak
  - Privacy protection
- Integrity
  - Zero-day attacks: code injection, rop
- Availability
  - Access control
  - Deny of service

CIA in AI model

- Confidentiality
  - Data privacy
- Integrity
  - Adversarial sample attacks
  - Back-doors
- Availability
  - ???

Adversarial Samples in Deep Neural Networks

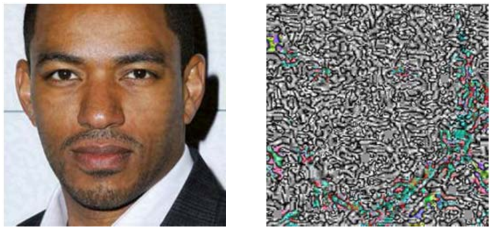
- Adversarial samples are model inputs generated by adversaries to fool neural networks



Existing Adversarial Attacks

- Semantic based perturbations
  - Change a/a few region(s) of the image
  - Simulate real world scenarios
- Pervasive perturbations
  - Alter images in pixel level
  - Different distance metrics: L_0, L_2, L_∞

$$\Delta(x, x') = ||x - x'||_p = \left(\sum_{i=1}^{n} |x_i - x'_i|^p\right)^{\frac{1}{p}}$$



Representative Attacks

- Semantics based perturbations

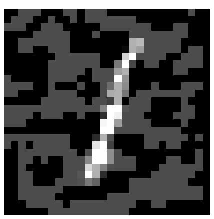
Dirt: Camera lens have dirt.

Brightness: Different lighting conditions.

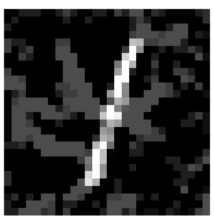
Rectangle: Camera lens are blocked by another object.

Trojan: Watermarks.

- Pervasive perturbations

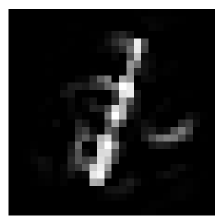


FGSM



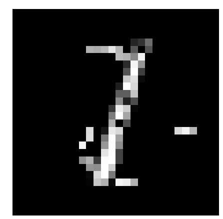
BIM

$C\&W_i$

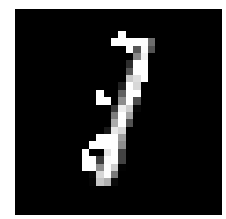


$C\&W_2$

DeepFool



JSMA



$C\&W_0$

L_∞ L_2 L_0

Existing Detection Methods

- Many detection and defenses have been proposed, we just list a few detection approaches here

- Characterizing the dimensional properties of adversarial regions
  - LID from ICLR 2018, oral presentation

- Denoisers that can remove/reform perturbations
  - MagNet from CCS 2017
  - HGD from CVPR 2018
    - First place in the NeurIPS 2017 competition on defense against adversarial attacks

- Prediction inconsistency
  - Feature Squeezing from NDSS 2018

However … …

- We found that most existing detection methods work on a subset of existing attacks or datasets (will show in evaluation later)
- Similar results are found by other researchers

All (evaluated) detection methods show comparable discriminative ability against existing attacks. Different detection methods have their own strengths and limitations facing various kinds of adversarial examples.[0]

[0]: Xiang Ling, Shouling Ji, Jiaxu Zou, Jiannan Wang, Chunming Wu, Bo Li and Ting Wang, DEEPSEC: A Uniform Platform for Security Analysis of Deep Learning Models, IEEE S&P, 2019

Program Invariants in Software Engieering

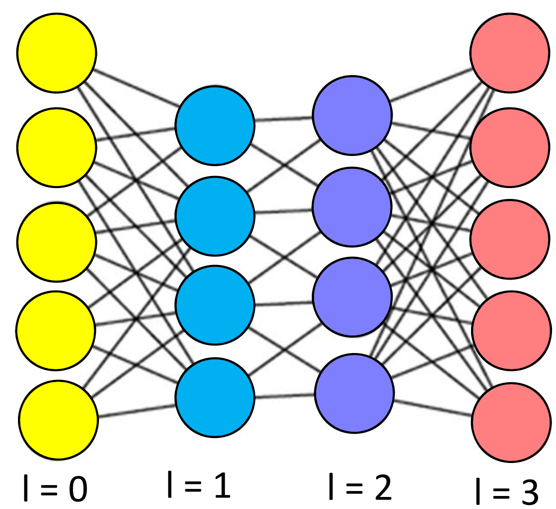
```
01: def fib(n):
02:   assert(n>=0)
03:   assert(from line 6 or 10)
04:   if n == 0 or n == 1:
05:     return n
06:   return fib(n-1)+fib(n-2)
07:
08: def main():
09:   x = input('Input a number:')
10:   print fib(x)
```

Key idea: using invariant checks to allow correct behaviors and forbidden other possible (malicious) behaviors.

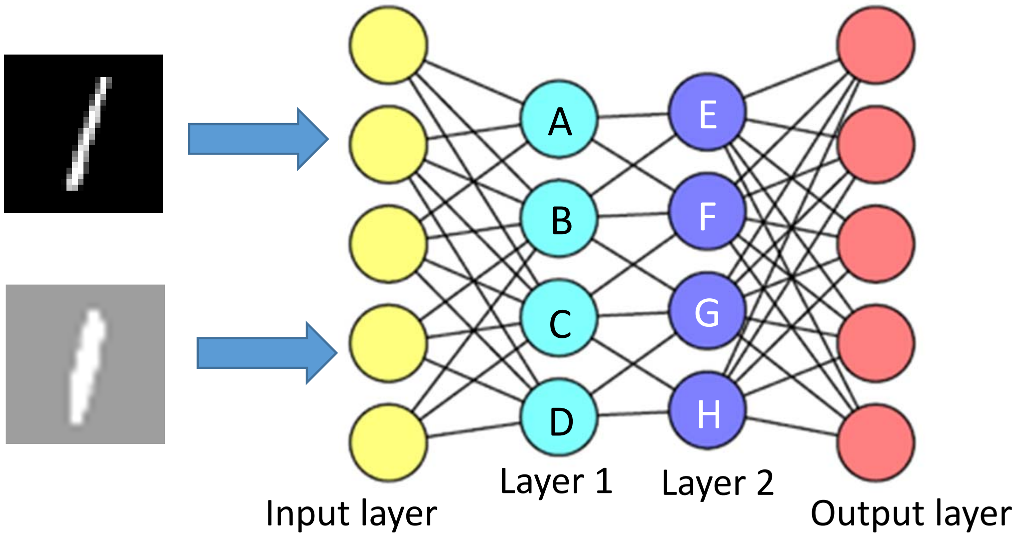
DNN Invariants

- Value invariants
  - Possible neuron value distributions of each layer
- Provenance invariants
  - Possible neuron value patterns of two consecutive layers
- If an input violates either kind of invariant, it is considered an adversarial sample

```
def DNN():
  for l in model.layers():
    if(l==0): x_l = input
    else: x_l+1 = f_l(w_l*x_l+b_l)
```

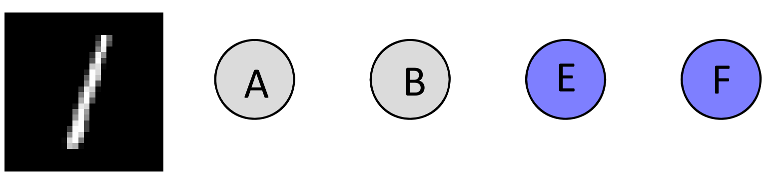


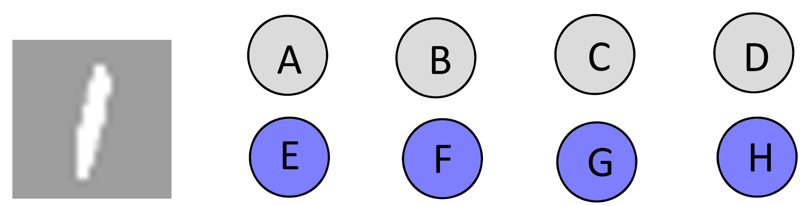
DNN Value Invariant



Activation of a benign sample of digit 1:
<A: 20, B: 30, C: 0, D, 0> <E: 40, F: 20, G: 0, H: 0>

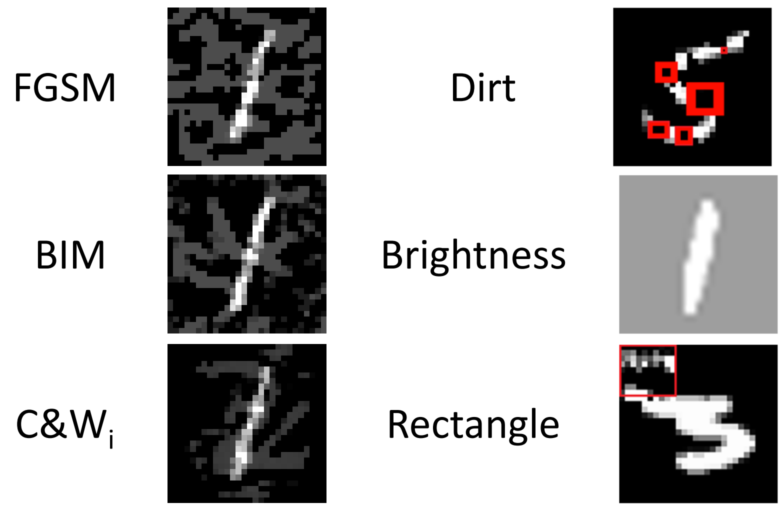
Activation of an adversarial sample of digit 1:
<A: 10, B: 10, C: 10, D, 10> <E: 10, F: 11, G: 9, H: 10>



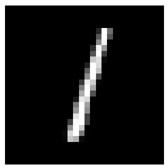


DNN Value Invariant

- Perturbations will change the activation patterns
  - Many attacks have new activation patterns in hidden layers
- Value Invariant
  - Trained classifiers that capture the activation patterns (of benign input samples) in each layer



Train DNN Value Invariant



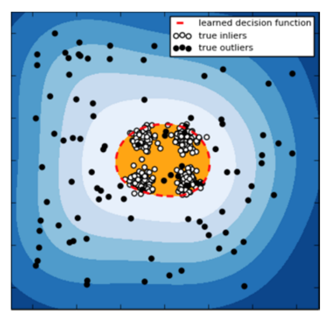
Activation of a benign sample of digit 1:
<A: 20, B: 30, C: 5, D, 5> <E: 40, F: 20, G: 3, H: 3>

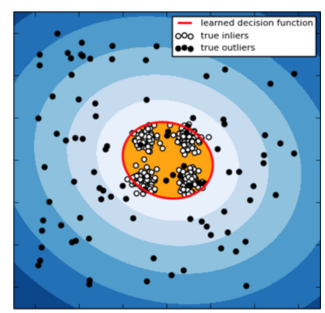
Activation of an adversarial sample of digit 1:
<A: 10, B: 10, C: 10, D, 10> <E: 10, F: 11, G: 9, H: 10>

1. Trace neuron activation values (for each layer) for all benign samples

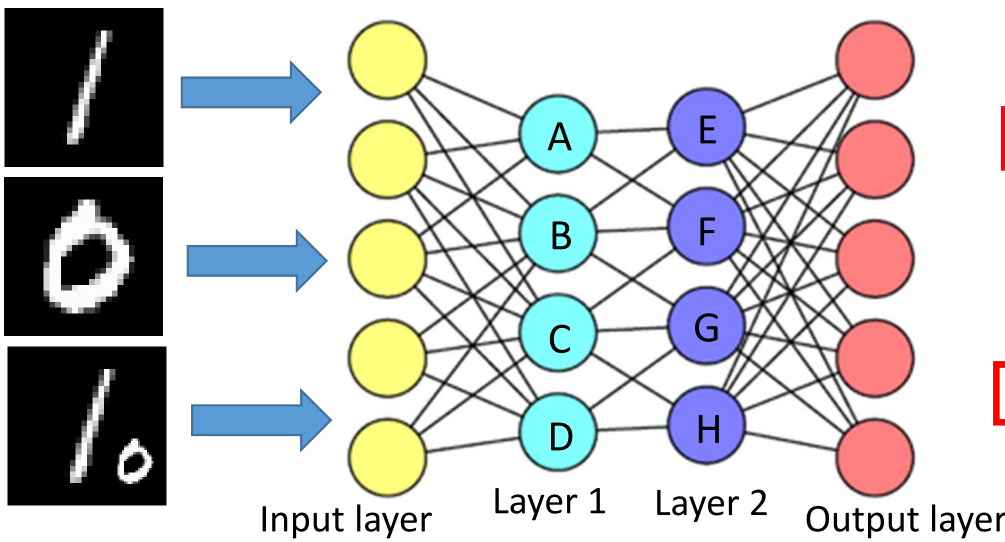
Sample 1:<A: 20, B: 30, C: 05, D, 05>
Sample 2:<A: 40, B: 20, C: 15, D, 13>
Sample 3:<A: 30, B: 34, C: 35, D, 52>

2. Train a classifier for each layer (One-class SVM)





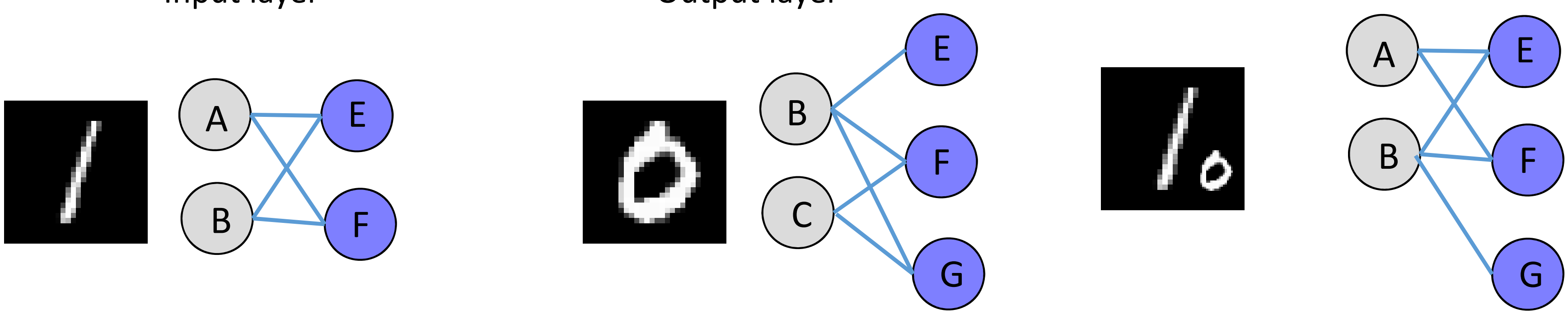
DNN Provenance Invariant



Activation of a benign sample of digit 1:
<A: 20, B: 30, C: 0, D: 0> <E: 40, F: 20, G: 0, H: 0>

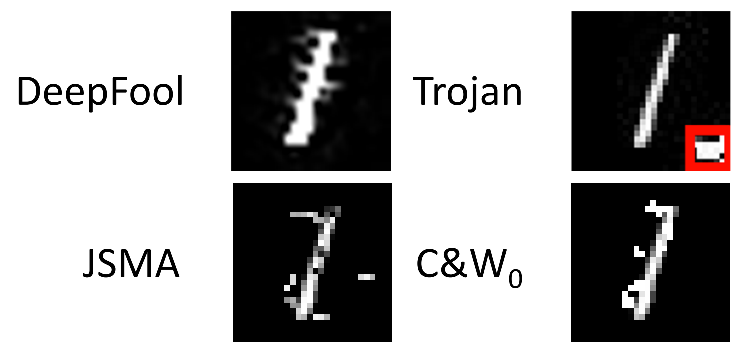
Activation of a benign sample of digit 0:
<A: 0, B: 30, C: 40, D: 0> <E: 30, F: 60, G: 50, H: 0>

Activation of an adversarial sample:
<A: 22, B: 34, C: 0, D: 0> <E: 28, F: 54, G: 46, H: 0>

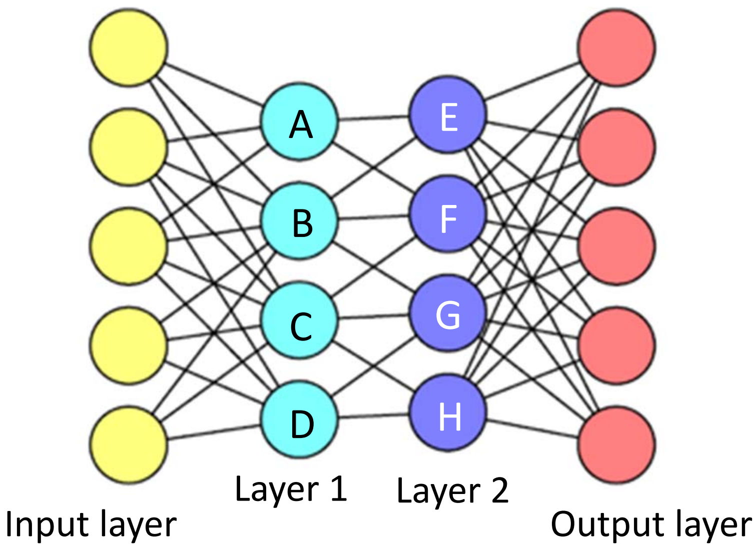


DNN Provenance Invariant

- DNN model may focus on different parts of the input in different layers
  - The patched image looks similar to 1 in layer 1 and look similar to 0 in layer 2
  - Using individual value invariants can not detect such attacks
- Provenance invariant
  - Trained classifiers that capture the activation patterns across two consecutive layers
  - Reduce dimensions to the number of output labels



Train Provenance Invariant



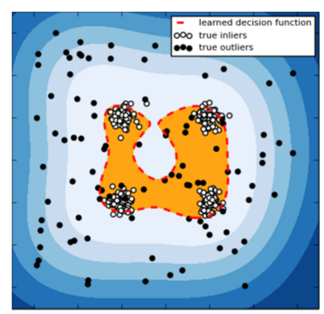
- Train on raw neuron values
  - Too many of them, hard to train
- Lower the dimensions of individual layers

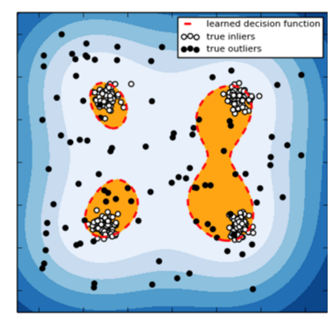
1. Lower the dimensions of neurons in hidden layers

    Sample 1:<A: 0.3, B: 0.5, C: 0.1, D: 0.1>
    Sample 1:<E: 0.6, F: 0.2, G: 0.1, H, 0.1>
    Sample 2:<A: 0.2, B: 0.4, C: 0.3, D: 0.1>
    Sample 2:<E: 0.3, F: 0.4, G: 0.2, H, 0.1>

2. Train a classifier on 2 consecutive layers

    Sample 1:<A: 0.3, B: 0.5, C: 0.1, D: 0.1,
    E: 0.6, F: 0.2, G: 0.1, H, 0.1>
    Sample 2:<A: 0.2, B: 0.4, C: 0.3, D: 0.1
    E: 0.3, F: 0.4, G: 0.2, H, 0.1>





Evaluation

- Datasets and models
  - MNIST: Cleverhans (*2), Carlini's model from IEEE S&P 2017, LeNet-4/5
  - CIFAR-10: Carlini's model, DenseNet
  - ImageNet: ResNet50, VGG19, Inceptionv3, MobileNets
  - LFW: VGG19 (Trojan attack)
- Attacks
  - Perturbation attacks: FGSM, BIM, C&W attacks, DeepFool, JSMA
  - Semantics attacks: Dirt, Brightness, Rectangle, Trajon
  - Parameters adopted from previous papers (e.g., Feature Squeezing)
- Comparison with others
  - LID, MagNet, HGD, Feature Squeezing

Results

- NIC achieves over 90% detection accuracy on all attacks
- Other methods achieve good results on a subset but fail to work on some of them (low detection accuracy)
  - LID: Good at L_∞ attacks on MNIST and CIFAR, but poor performance large sized images (e.g., ImageNet)

Method	Dataset	FGSM (L_∞)	DeepFool (L_2)	C&W (L_0)
NIC	CIFAR	100%	100%	100%
	ImageNet	100%	90%	100%
LID	CIFAR	94%	84%	90%
	ImageNet	82%	83%	79%

Results

- NIC achieves over 90% detection accuracy on all attacks
- Other methods achieve good results on a subset but fail to work on some of them (low detection accuracy)
  - MagNet: it does not perform well on many $L_{0/2}$ attacks, and it is hard to train on large sized image datasets

Method	Dataset	FGSM (L_∞)	C&W (L_2)	JSMA (L_0)
NIC	MNIST	100%	100%	100%
	CIFAR	100%	100%	100%
MagNet	MNIST	100%	87%	84%
	CIFAR	100%	89%	74%

Results

- NIC achieves over 90% detection accuracy on all attacks
- Other methods achieve good results on a subset but fail to work on some of them (low detection accuracy)
  - Feature Squeezing: not good at L_∞ attacks on large sized images (e.g., ImageNet) and some patching attacks

Method	Dataset	FGSM (L_∞)	C&W (L_2)	C&W (L_0)
NIC	MNIST	100%	100%	100%
	ImageNet	100%	90%	100%
Feature Squezzing	MNIST	100%	100%	94%
	ImageNet	43%	92%	98%

Results

- NIC achieves over 90% detection accuracy on all attacks
- Other methods achieve good results on a subset but fail to work on some of them (low detection accuracy)
  - Feature Squeezing: not good at L_∞ attacks on large sized images (e.g., ImageNet) and some semantics attacks

Method	Dataset	Trojan	Brightness	Dirt	Rectangle
NIC	MNIST	100%	100%	100%	100%
	LFW	100%	100%	100%	100%
Feature Squeezing	MNIST	82%	39%	97%	72%
	LFW	67%	40%	89%	82%

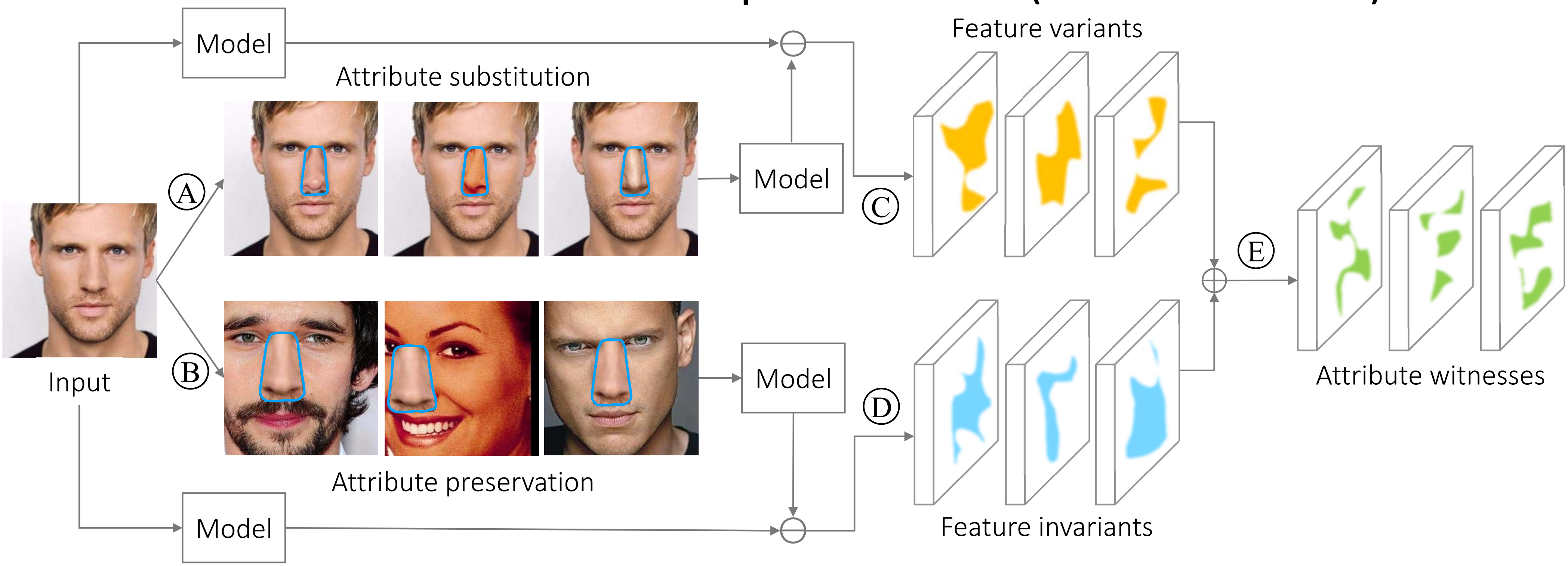
Other results (summary)

- False positives: 0.2-5.8% for most models, the worst case is 14.6% for ImageNet (i.e., 0.3-7.2% lower than other techniques)
- Value invariant or provenance invariant
  - They are complementary to each other!
  - Using just one of the two cannot get good results on all attacks
- Adaptive attacks
  - Adversary knows the original model, the detection methods, and all the value invariants and provenance invariants
  - $C\&W_2$ based attack achieved 97% success on MNIST and CIFAR-10 without bounding perturbation to a reasonable range
    - For MNIST, L_2 distortion is 3.98 (Feature Squeezing is 2.80)

To Summarize

- The SE concept of invariants works very well for deep learning models
  - Invariant derivation does not require negative samples, which is critical to adversarial sample defense
- However, the way of deriving invariants is different
  - Need to consider the unique deep learning semantics and handle unique challenges, e.g., large dimension (huge buffers)

Inner Neuron Interpretation (NeurIPS'18)



Forward: attribute changes -> neuron activation changes

Backward: neuron activation changes —> attribute changes
Backward: no attribute changes —> no neuron activation changes

Program Analysis versus AI Model Analysis

- Software debugging versus AI model debugging
- Software security versus AI model security
  - CIA
- Software testing versus AI model testing
- Software verification versus AI model verification
- Software optimization versus AI model optimization
- Integrating software with AI

Program Analysis versus AI Model Analysis

- Software debugging versus AI model debugging
- Software security versus AI model security
  - CIA
- Software testing versus AI model testing
- Software verification versus AI model verification
- Software optimization versus AI model optimization
- Integrating software with AI

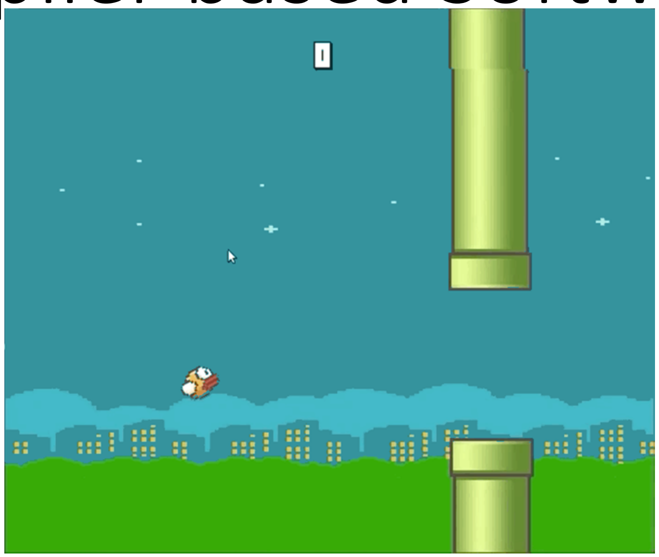
Program Analysis versus AI Model Analysis

- Software debugging versus AI model debugging
- Software security versus AI model security
  - CIA
- Software testing versus AI model testing
- Software verification versus AI model verification
- Software optimization versus AI model optimization
- Integrating software with AI

Program Analysis versus AI Model Analysis

- Software debugging versus AI model debugging
- Software security versus AI model security
  - CIA
- Software testing versus AI model testing
- Software verification versus AI model verification
- Software optimization versus AI model optimization
- Integrating software with AI

Compiler based Software Autonomization (PLDI'19)



- User provides the minimal annotation
  - Places to replace with AI (e.g., keystrokes)
- LLVM instruments program
- AI will learn to operate the software over time

Game	Original LOC	Added LOC
Flappy Bird	0.8k	40
Mario	21k	73