Analyzing AI Model Internals for Debugging and Adversarial Sample Attack Detection

Xiangyu Zhang @ ETH, 2018

PURDUE UNIVERSITY

AI Driven Computing

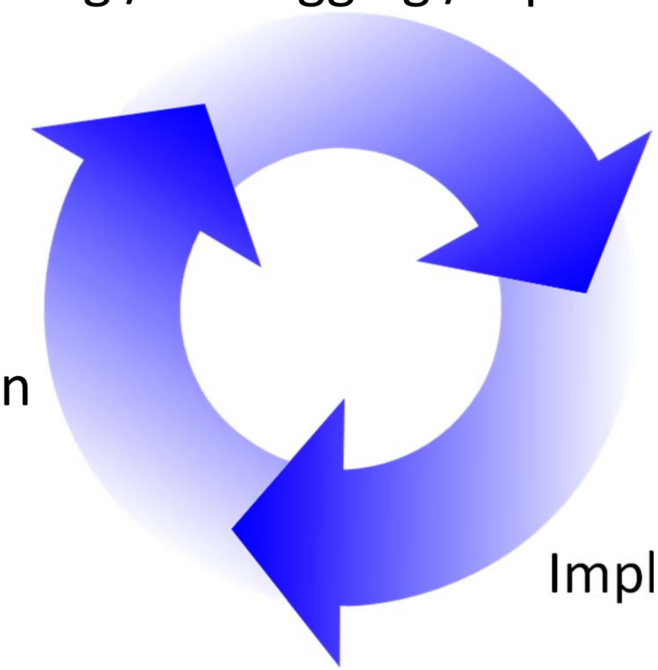
- AI Models are becoming an integral part of modern computing
  - Autonomous vehicles, Apple Face ID, iRobots, Cotana, and computer games

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

bigml OpenML Gradientzoo Predictors.ai

AI Driven System Engineering

Tuning / Debugging / Optimization



Evaluation

Implementation

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

- Traditional engineering bugs
  - Coding bugs, data cleaning, mis-behaved data partitioning, improper data augmentation

- Model bugs – *misconducts in the AI model engineering process leading to undesirable consequences*
  - 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

  - 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.
  - Numerous attacks on AI systems (Trojaning, perturbation, and patching attacks)

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

Theme of the Talk

- Leveraging what we have learned in program analysis and software engineering to open the box

- Outline
  - MODE: Automated Neural Network Model Debugging via State Differential Analysis and Input Selection (FSE'18)

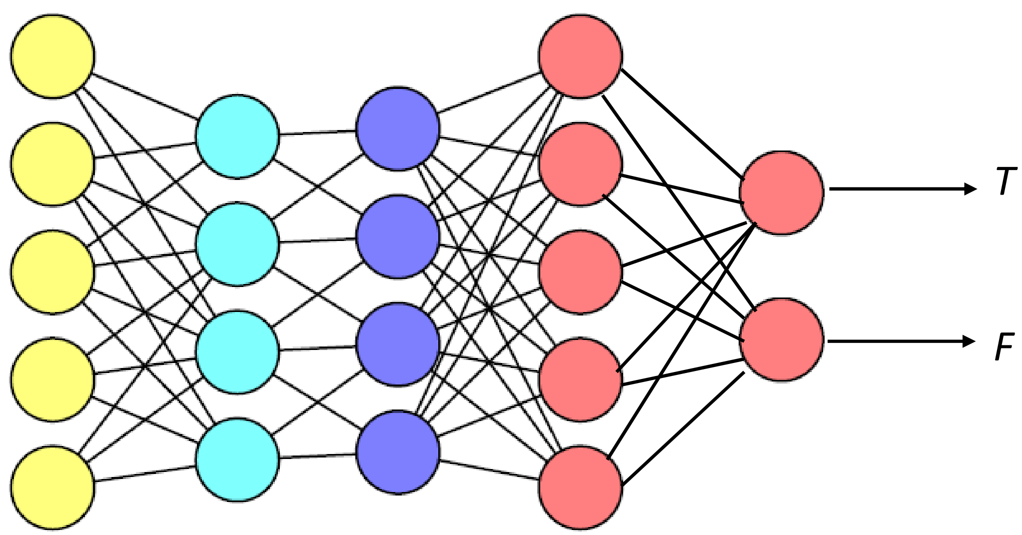
  - AmI: Attacks Meet Interpretability, Attribute-steered Detection of Adversarial Samples (NIPS'18)

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

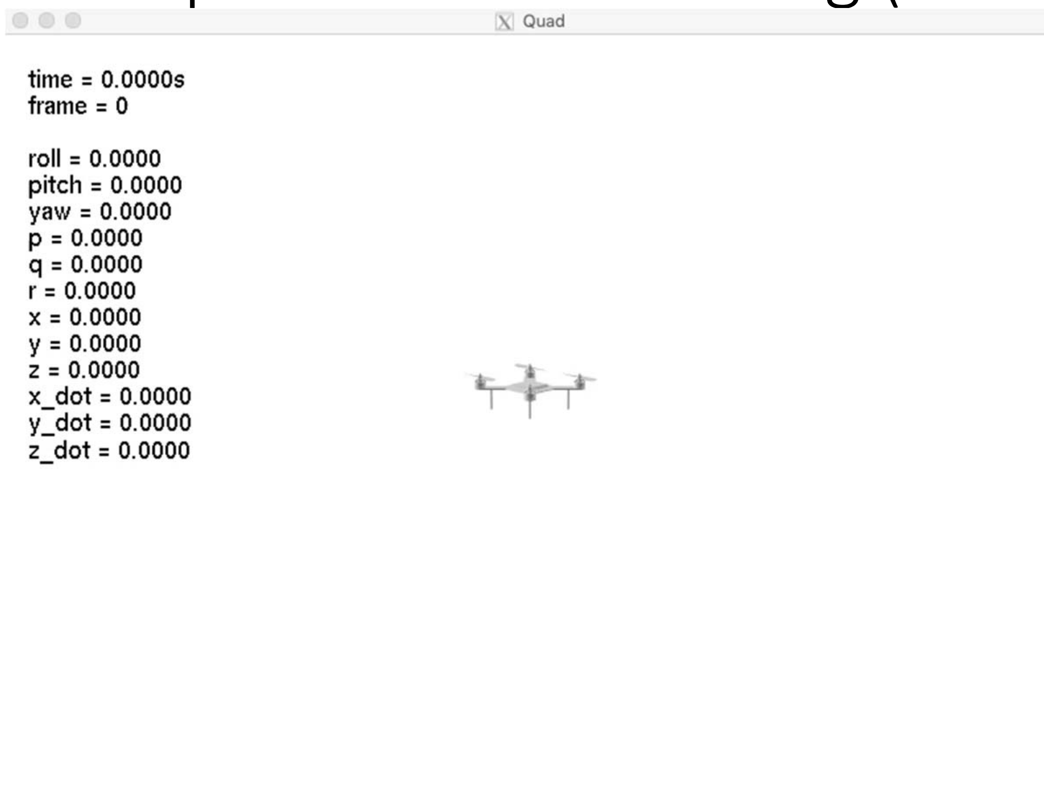
$T \wedge F \vee T$



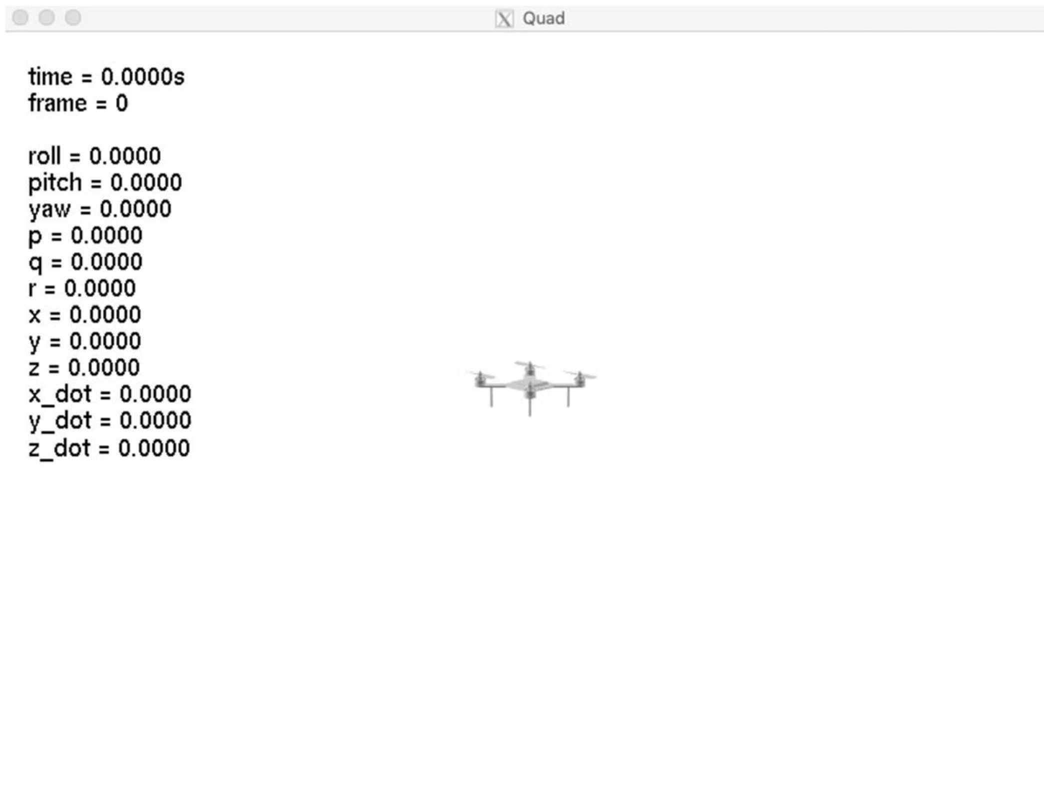
AI Model Bugs

- 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

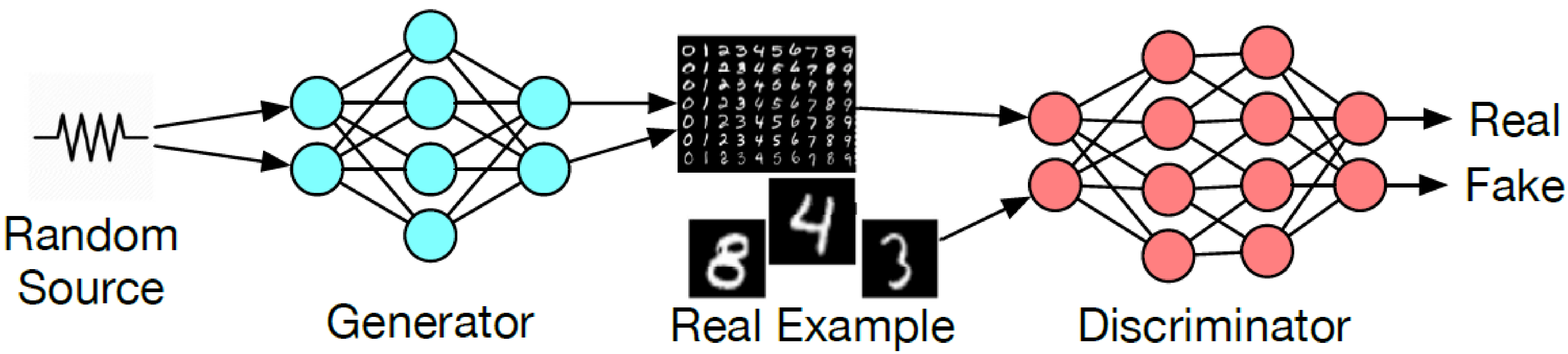
- Structural bugs
  - Redundant/insufficient layers/neurons
  - In-effective structures
    - Forget gates in (LSTM) do not retain the appropriate contextual information
  - Suboptimal setting of reward values leading to extremely long training time in reinforcement learning

Overfitting and Underfitting Bugs

- We say a model has an underfitting bug if for some label, both training and test accuracies are lower than a threshold *t*
  - *t* is domain specific
- We say a model has an overfitting bug if for some label, its training accuracy is higher than test accuracy by at least some threshold

Existing Works

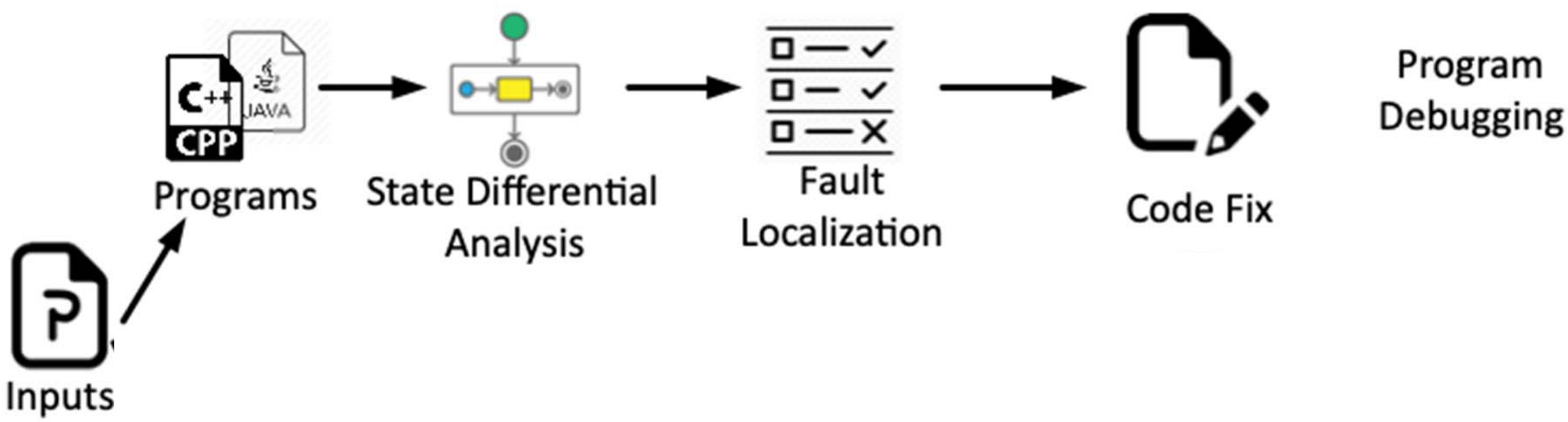
- Applying pre-defined image operations on existing data points
  - Rotation, mirror, clip, brightness change etc.
- Using generative models to collect new data points
  - Variational Autoencoder (VAE) or Generative Adversarial Net (GAN)
  - Trend of using GAN



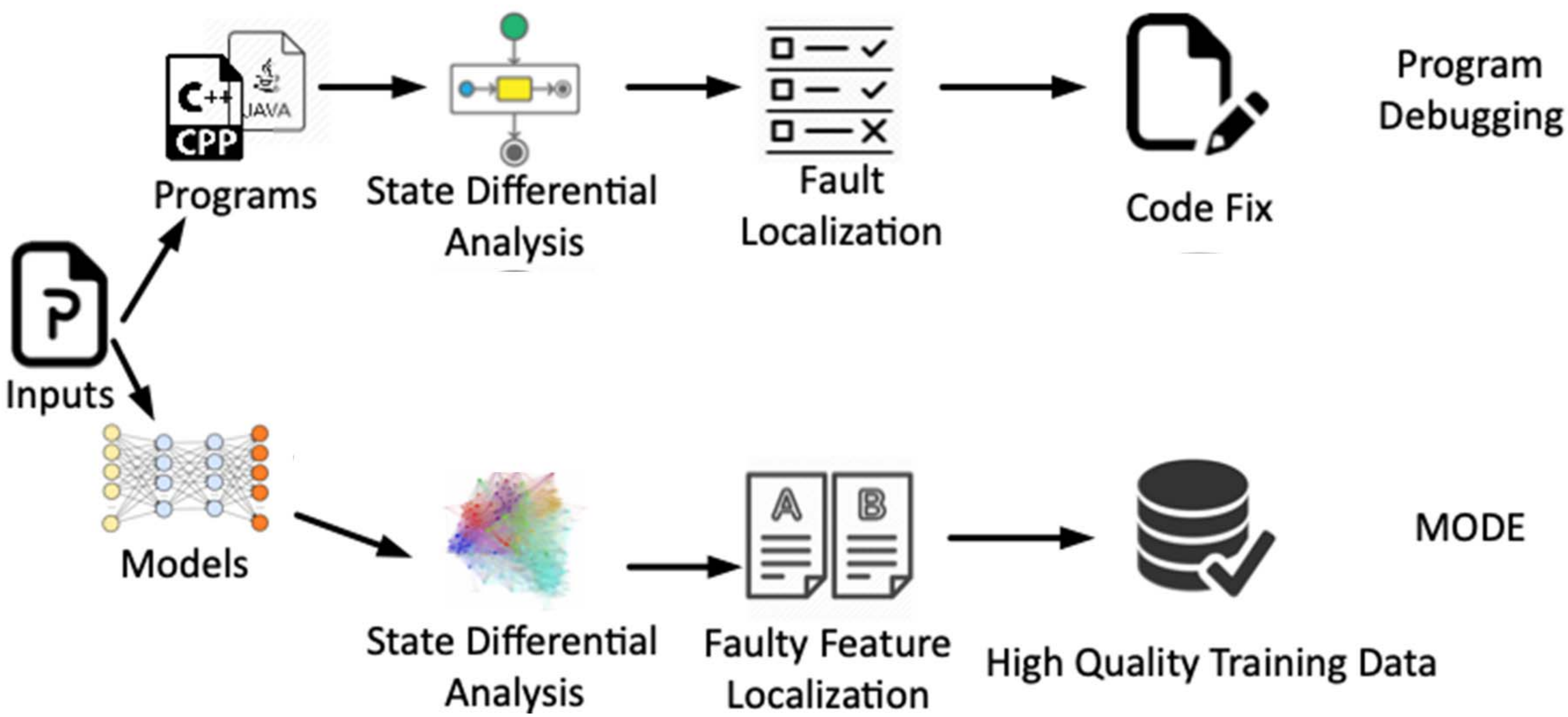
Using GAN is Not That Effective

- 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*

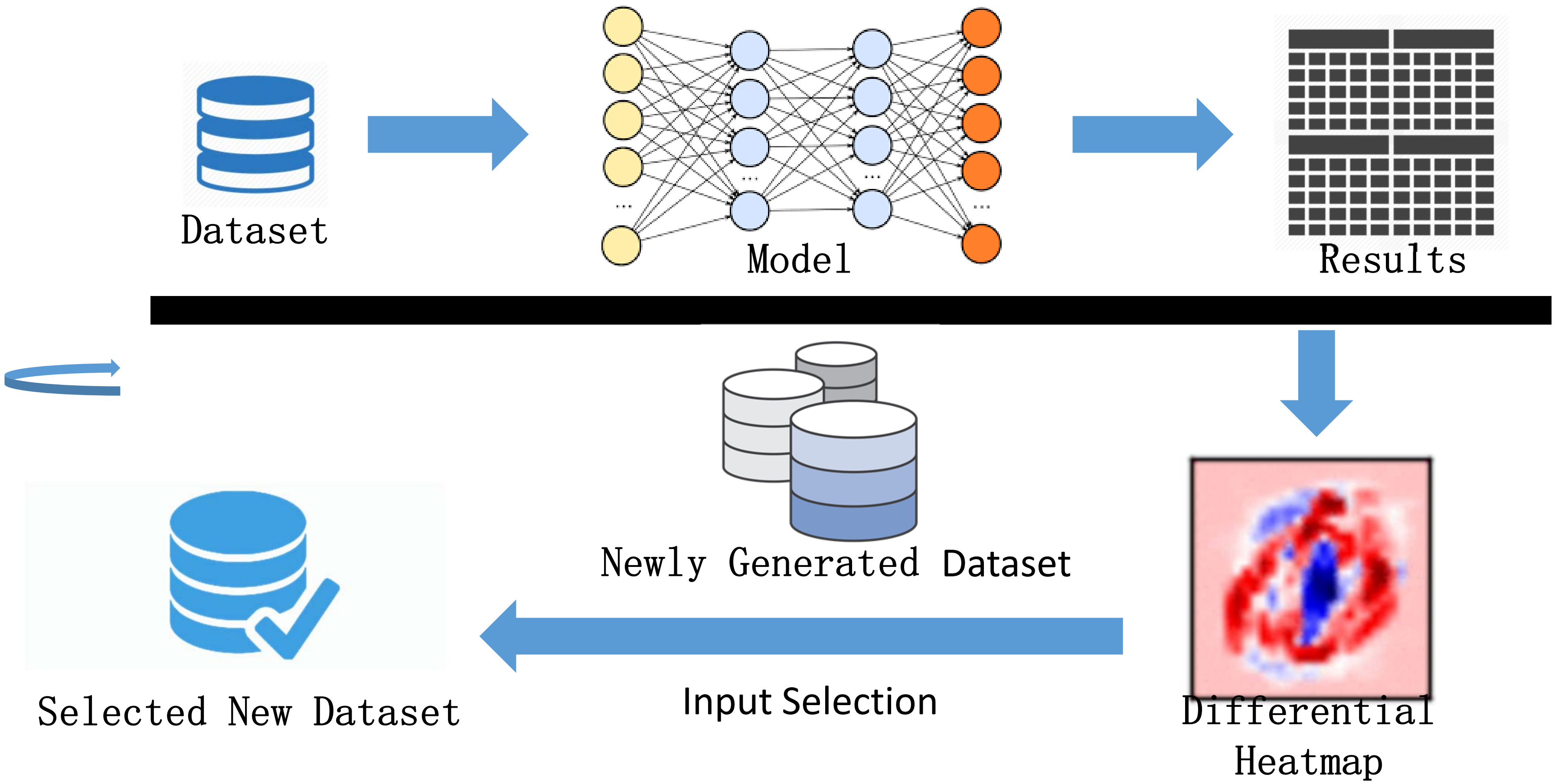
What Have We Learned in Software Debugging



MODE: AI Model State Differential Analysis and Input Selection

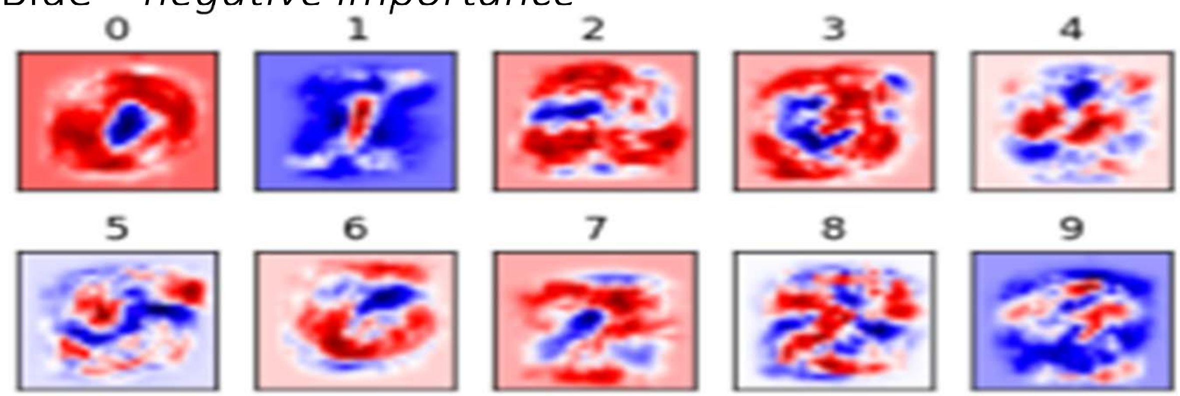


Overview


Dataset
Model
Results
Newly Generated Dataset
Selected New Dataset
Input Selection
Differential
Heatmap

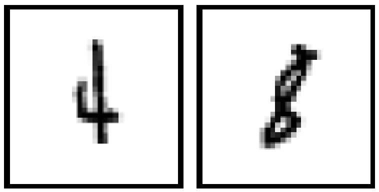
Heat Map

- A matrix representing the importance of each neuron in a hidden layer
  - One heat map for each hidden layer
  - Each neuron denotes some abstract feature
- Visualization of heat-map
  - One pixel denotes the importance of one neuron/feature (to the output)
  - Red – *positive importance*
  - Blue – *negative importance*

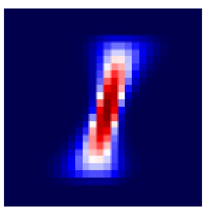


A Motivating Example

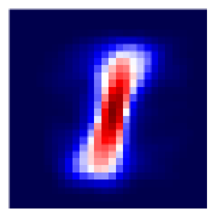
- Assume a model that has an underfitting bug for label 1 (other numbers misclassified to 1)



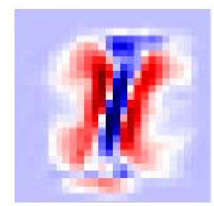
- Benign heat-map



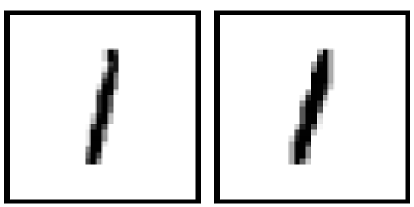
Faulty heat-map



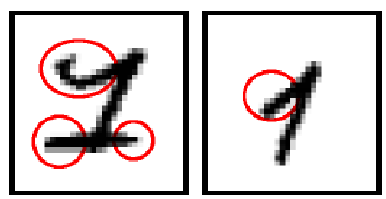
Differential heat-map



- Selected samples

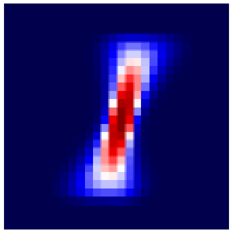


samples should not be selected

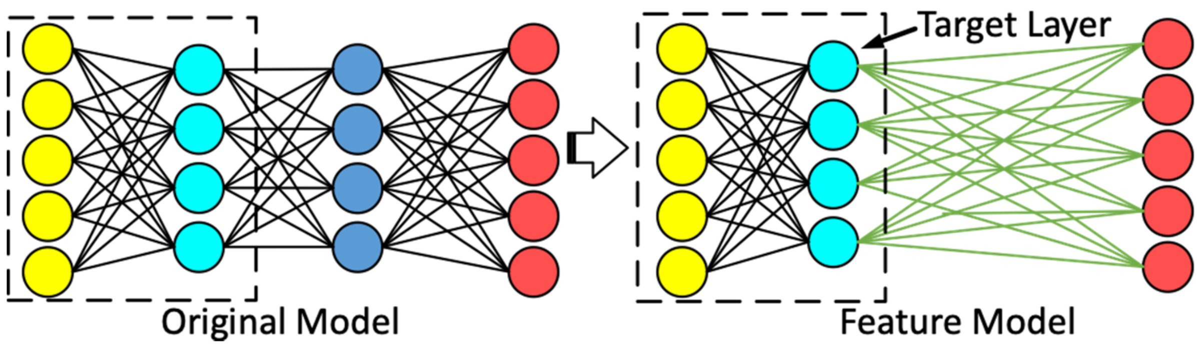


Heat-map Computation

- Cannot use gradient information -- *how much output changes can be induced by weight value changes*
  - Gradients are with-respect-to weight values, whereas importance is with-respect-to features (neurons)
  - Importance measures how much influence a feature has on the classification result of an output label
    - Important features may not have weight values of a large gradient



Heat-map Computation



- Feature model: part of the original model (including weights) + a newly trained SoftMax layer (used for prediction)

- The weights of the last layer measure the importance of individual features (for the prediction)

- The normalized weights for an output label of the newly trained SoftMax layer is the ***Heatmap***

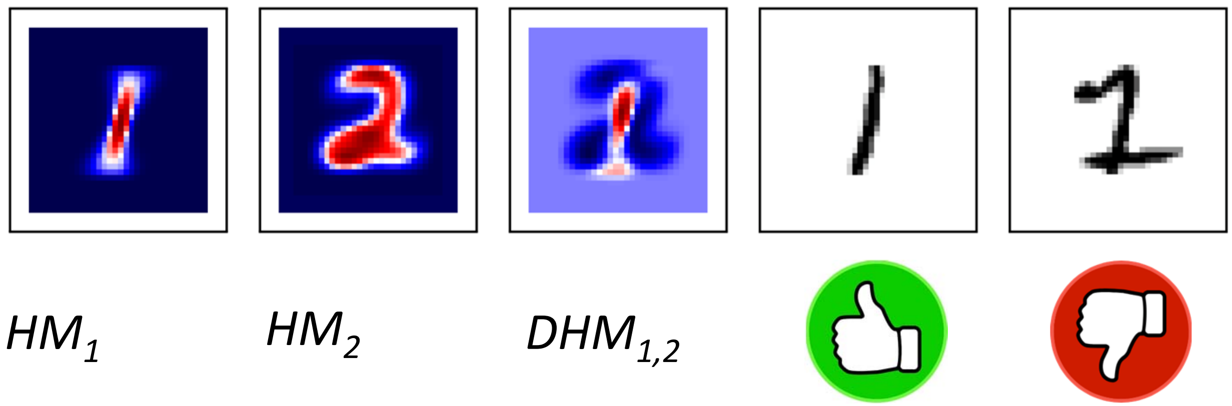
  - Normalize the weights to [-1,1] with the absolute values denoting the importance and the signs denoting positive/negative importance

Differential Heat-maps for Under-fitting

- Two kinds of root causes
  (1) Extracted features cannot fully represent the uniqueness of the target label
    - Selecting cases that can emphasize the uniqueness

  (2) Cases mis-classified to the target label share common features with some cases of the target label
    - Not to select such cases
- Two corresponding kinds of differential heap-maps
  - For (1),
    - $DHM_L[f] = HM_L[f] - HM_k[f]$, *when* $|HM_L[f] - HM_k[f]|$ *is minimal for* $k != L$
  - $DHM_L[f]$ represents the minimal similarity of feature ***f*** regarding the target label *L* and some other output label, larger values mean more uniqueness

Example



- *DHM* shows the importance features to differentiate two output labels, as shown in the example

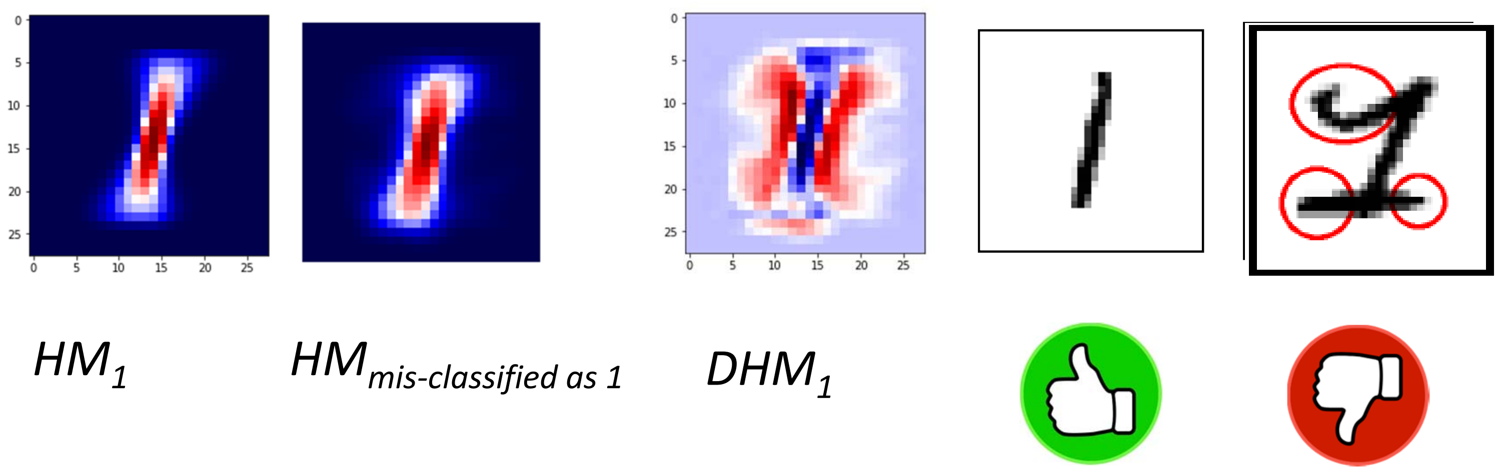
- Selecting cases with strong presence in the red zones and weak presence in the blue zones would help improve uniqueness

Differential Heat-maps for Under-fitting

- Two kinds of root causes
  (1) Extracted features cannot fully represent the uniqueness of the target label
    - Selecting cases that can emphasize the uniqueness

  (2) Cases mis-classified to the target label share common features with some cases for the target label
    - Selecting samples that the model to disambiguate
- Two corresponding kinds of differential heap-maps
  - For (1), …
  - For (2),
    - $DHM_L[f] = HM_{\text{misclassified as } L}[f] - HM_{\text{correctly classified as } L}[f]$
  - A large (red) value indicates the feature is critical for misclassification

Example



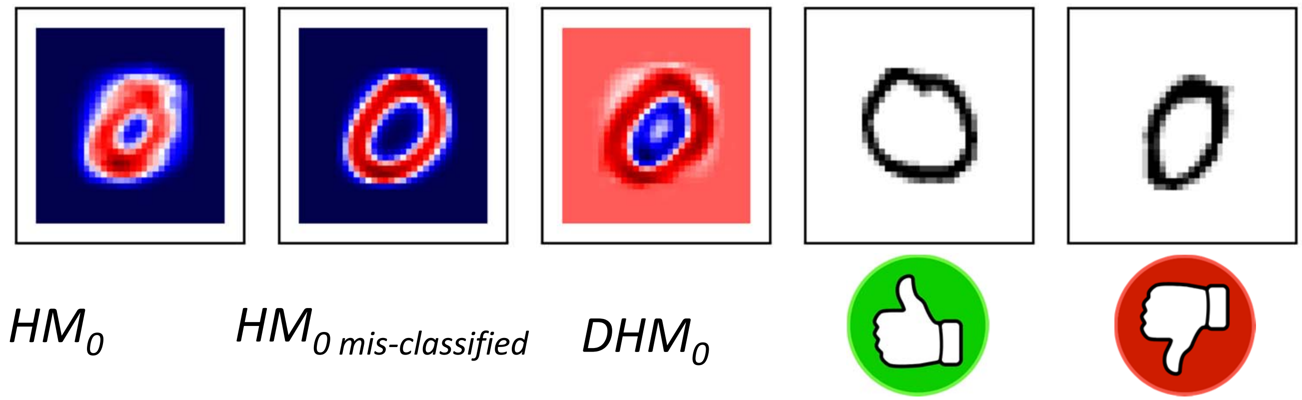
- *DHM* shows the confusing features
- Selecting cases that avoid the red areas and cover the blue areas will benefit

Differential Heat-map for Over-fitting

- Root cause – *narrowly scoped training data, model too large,* ***or*** *training with too many epochs*
    - More diverse training data for the target label is needed

- $DHM_L[f] = max_k(HM_{L\ misclassified\ as\ k}[f] - HM_{correctly\ classified\ as\ L}[f])$
    - A large value denotes that the feature is responsible for misclassifying ***L*** to ***k***
    - We need more samples that has this feature

Example



- The red regions in the *DHM* denote the features helpful for generalization, the blue regions denote the overfitted features.
    - Larger-sized 0 are needed

Input Selection

- For each new input *i*, we feed it to the feature model (without running through the output layer) to acquire a feature value vector *V*.

$$score = V \cdot DHM$$

- *DHM* is a vector pointing to the most promising direction

Evaluation

- RQ1: How effective and efficient is MODE in fixing model bugs?
- RQ2: How does MODE compare to using random samples or faulty samples to fix model bugs?
- RQ3: What is the impact of different parameters?

Experiment One

- Three data sets
  - Digit recognition (MNIST), Fashion-icon recognition (FM), Object recognition (CIFAR)
- For each data set, we have downloaded multiple models
  - Total 20 models
  - 20k-20M weight values
- Training with batches of 2000 samples, capped at 20,000 samples and 4 hours for small models, and 40,000 samples and 24 hours for large models
- Partition an original data set 30% training, 10% validation, 10% test, 50% bug fixing
- Select one UF and one OF for each model
  - UF: the output label with the lowest training and test accuracy
  - OF: the label with good training accuracy but the lowest test accuracy

Effectiveness: MNIST

Table 1: Fixing Model Bugs Summary

Model (Size)	Bug Type	Accuracy		MODE				Randomly Selecting GAN Samples					Failing Sample	
		Model	Label	#Samples	Time	MAcc	LAcc	>5%?	MAcc	LAcc	#Samples	Time	MAcc	LAcc
MNIST-1	Under-fitting	88%	74%	500+1500	5m	93%	94%	3(14)	90%	83%	20000	1h2m	86%	80%
7k	Over-fitting	84%	81%	500+1500	6m	92%	91%	4(14)	89%	87%	20000	1h4m	85%	77%
MNIST-2	Under-fitting	89%	72%	500+1500	5m	92%	92%	5(14)	90%	88%	20000	57m	85%	82%
185k	Over-fitting	84%	76%	500+1500	5m	93%	93%	3(14)	87%	85%	20000	1h4m	83%	74%
MNIST-3	Under-fitting	86%	76%	500+1500	5m	94%	94%	4(14)	92%	87%	20000	1h10m	88%	82%
185k	Over-fitting	84%	78%	500+1500	5m	94%	95%	3(14)	93%	90%	20000	55m	84%	82%
MNIST-4	Under-fitting	86%	78%	500+1500	5m	94%	94%	3(14)	86%	84%	20000	56m	85%	80%
185k	Over-fitting	84%	74%	500+1500	5m	92%	92%	4(14)	84%	83%	20000	50m	88%	78%
MNIST-5	Under-fitting	82%	77%	500+1500	5m	94%	92%	4(14)	87%	88%	20000	54m	84%	74%
122k	Over-fitting	85%	77%	500+1500	5m	92%	92%	3(14)	88%	90%	20000	54m	82%	76%
MNIST-6	Under-fitting	84%	72%	1000+3000	10m	93%	93%	3(14)	89%	84%	40000	1h58m	81%	78%
244k	Over-fitting	84%	74%	1000+3000	9m	94%	94%	3(14)	86%	82%	40000	2h	79%	76%
MNIST-7	Under-fitting	87%	77%	1000+3000	9m	93%	93%	3(14)	88%	85%	40000	2h9m	84%	75%
185k	Over-fitting	85%	72%	1000+3000	9m	93%	91%	3(14)	87%	88%	40000	2h4m	87%	82%
MNIST-8	Under-fitting	86%	73%	1000+3000	10m	93%	93%	3(14)	87%	82%	40000	1h54m	88%	76%
185k	Over-fitting	84%	73%	1000+3000	12m	93%	94%	3(14)	88%	85%	40000	2h6m	87%	76%
MNIST-9	Under-fitting	84%	73%	1000+3000	9m	94%	95%	3(14)	88%	88%	40000	2h	81%	73%
257k	Over-fitting	84%	72%	1000+3000	9m	93%	93%	3(14)	86%	86%	40000	2h3m	87%	77%

Effectiveness: Fashion-MNIST

Table 1: Fixing Model Bugs Summary

FM-1	Under-fitting	88%	80%	500+1500	5m	93%	90%	2(10)	84%	88%	20000	1h2m	83%	78%
493k	Over-fitting	87%	82%	500+1500	5m	94%	94%	3(10)	89%	90%	20000	1h4m	88%	84%
FM-2	Under-fitting	85%	77%	500+1500	5m	95%	95%	2(10)	89%	88%	20000	1h9m	87%	80%
1.2M	Over-fitting	87%	74%	500+1500	5m	94%	94%	2(10)	90%	84%	20000	1h3m	85%	80%
FM-3	Under-fitting	87%	72%	500+1500	10m	93%	91%	2(10)	89%	78%	20000	1h12m	89%	76%
3.2M	Over-fitting	85%	69%	500+1500	9m	93%	93%	2(10)	88%	88%	20000	1h7m	88%	78%
FM-4	Under-fitting	86%	73%	500+1500	5m	92%	94%	3(10)	83%	80%	20000	1h3m	87%	73%
765k	Over-fitting	85%	74%	500+1500	5m	91%	92%	1(10)	88%	80%	20000	1h9m	87%	75%
FM-5	Under-fitting	87%	80%	500+1500	5m	92%	92%	2(10)	87%	86%	20000	1h3m	79%	74%
113k	Over-fitting	83%	73%	500+1500	5m	92%	93%	2(10)	83%	82%	20000	1h	86%	80%
FM-6	Under-fitting	89%	81%	500+1500	5m	93%	95%	2(10)	91%	91%	20000	1h3m	83%	75%
26M	Over-fitting	82%	74%	500+1500	9m	92%	94%	3(10)	85%	83%	20000	1h2m	85%	80%

Effectiveness: CIFAR

Table 1: Fixing Model Bugs Summary

CIFAR-1	Under-fitting	79%	64%	500+1500	6m	88%	89%	0(3)	74%	64%	40000	1h14m	79%	66%
62k	Over-fitting	79%	65%	500+1500	7m	92%	91%	0(3)	82%	63%	40000	1h21m	80%	68%
CIFAR-2	Under-fitting	84%	76%	500+1500	15m	91%	90%	0(3)	80%	74%	40000	2h40m	81%	82%
0.97M	Over-fitting	83%	72%	500+1500	21m	88%	89%	0(3)	88%	79%	40000	2h50m	85%	78%
CIFAR-3	Under-fitting	82%	78%	500+1500	30m	91%	90%	0(3)	81%	78%	40000	4h10m	83%	74%
1.7M	Over-fitting	86%	83%	500+1500	24m	93%	92%	0(3)	84%	74%	40000	4h9m	80%	72%
CIFAR-4	Under-fitting	84%	74%	1000+3000	12h40m	92%	93%	0(3)	87%	75%	38000	24h	86%	74%
20M	Over-fitting	87%	78%	1000+3000	12h9m	91%	92%	0(3)	90%	77%	38000	24h	87%	77%
CIFAR-5	Under-fitting	88%	79%	1000+3000	10h	92%	94%	0(3)	85%	78%	40000	24h	88%	78%
20M	Over-fitting	86%	79%	1000+3000	9h40m	93%	94%	0(3)	88%	78%	40000	24h	86%	73%

Effectiveness: Experiment Two

- Three new large data sets and models: face recognition (FR), objection detection CelebA (OD), age classification (AC)
- No available GANs

Table 2: Accuracy Improvement without GANs

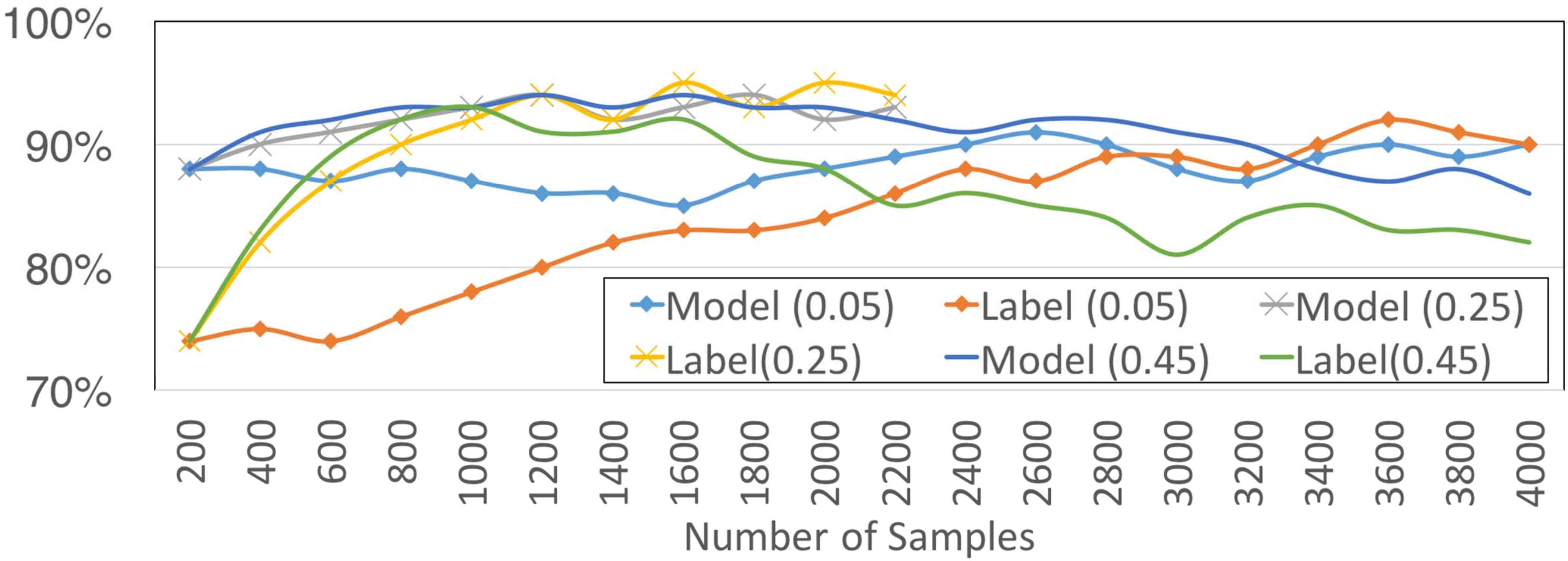
Model	Bug	Original		MODE		Random	
		MAcc	LAcc	MAcc	LAcc	MAcc	Lacc
FR	OF	76%	65%	88%	84%	79%	72%
2.1M	UF	72%	64%	85%	86%	78%	70%
OD	OF	83%	74%	89%	88%	84%	77%
3.2M	UF	82%	75%	88%	83%	84%	79%
AC	OF	33%/44%	13%/22%	46%/60%	38%/47%	32%/40%	33%/42%
30M	UF	25%/36%	11%/20%	42%/52%	36%/44%	32%/41%	25%/32%

Experiment Three: Improving Pre-trained Models

Table 3: Real-world Models Bug Fix

DataSet	Model	Original Acc.	# Samples	MODE Acc.	Random Acc.
MNIST	MNIST-10 [23]	95.2%	2000	97.4%	94.8%
	MNIST-11 [23]	93.4%	2000	96.8%	94.3%
Fashion MNIST	FM-7 [15]	87.6%	2000	92.3%	88.9%
	FM-8 [15]	91.6%	2000	92.6%	88.5%
CIFAR	CIFAR-6 [5]	87.3%	4000	93.2%	87.3%
	CIFAR-7 [5]	88.4%	4000	92.8%	88.2%

Sample Ratios



Theme of the Talk

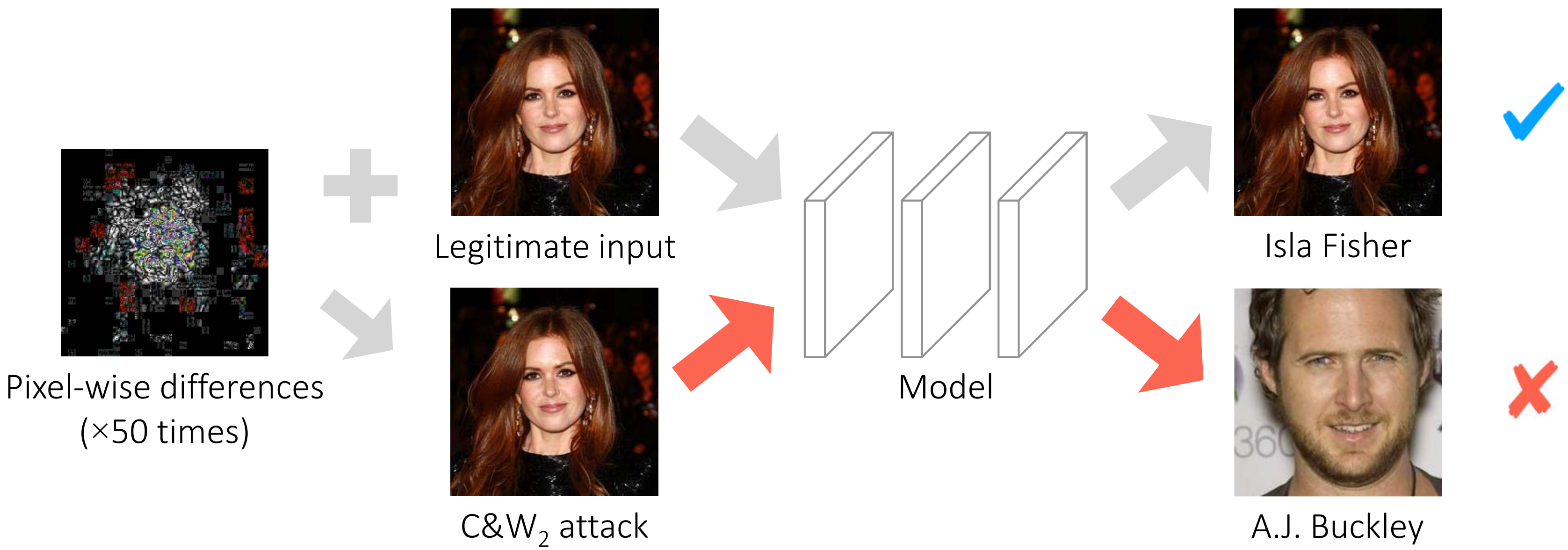
- Leveraging what we have learned in program analysis and software engineering to open the box

- Outline
  - MODE: Automated Neural Network Model Debugging via State Differential Analysis and Input Selection (FSE'18)

  - AmI: Attacks Meet Interpretability, Attribute-steered Detection of Adversarial Samples (NIPS'18)

Adversarial Samples

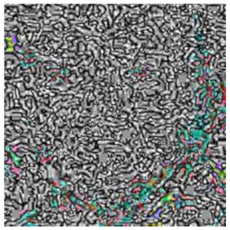
- Adversarial samples are model inputs generated by adversaries to fool neural networks (i.e., unexpected prediction results).



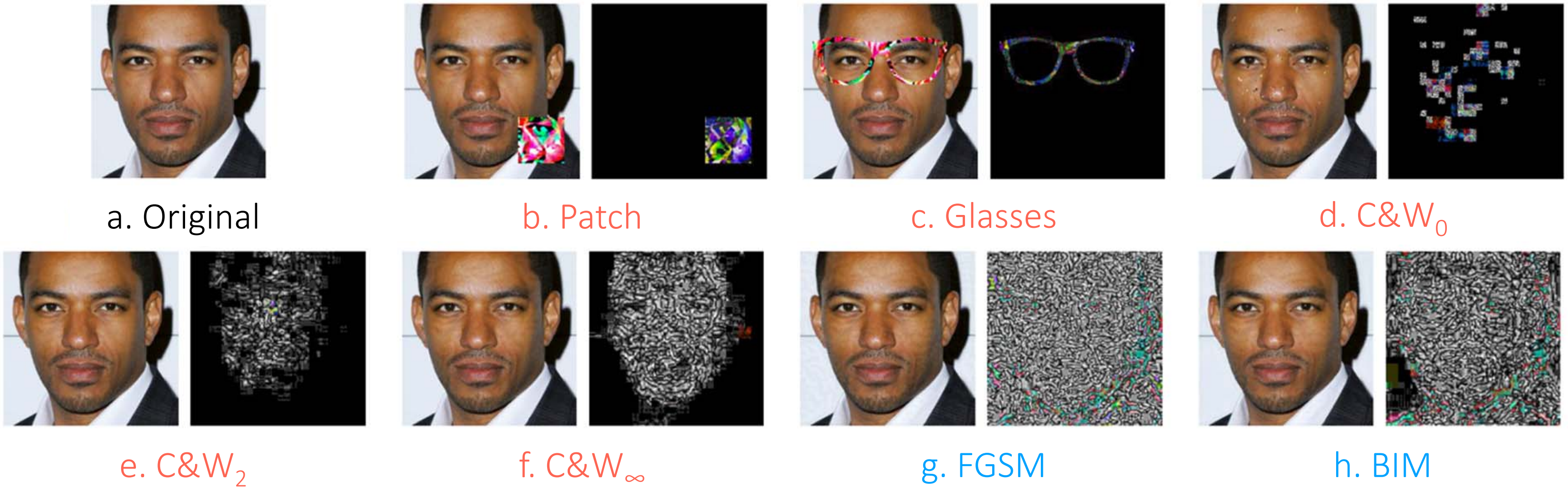
Existing Adversarial Attacks

- Patching
  - Restricted area to manipulate pixels
  - Utilize semantics of input space
- Pervasive perturbations
  - Full access to pixel alteration
  - Different distance metrics: L_0, L_2, L_∞

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

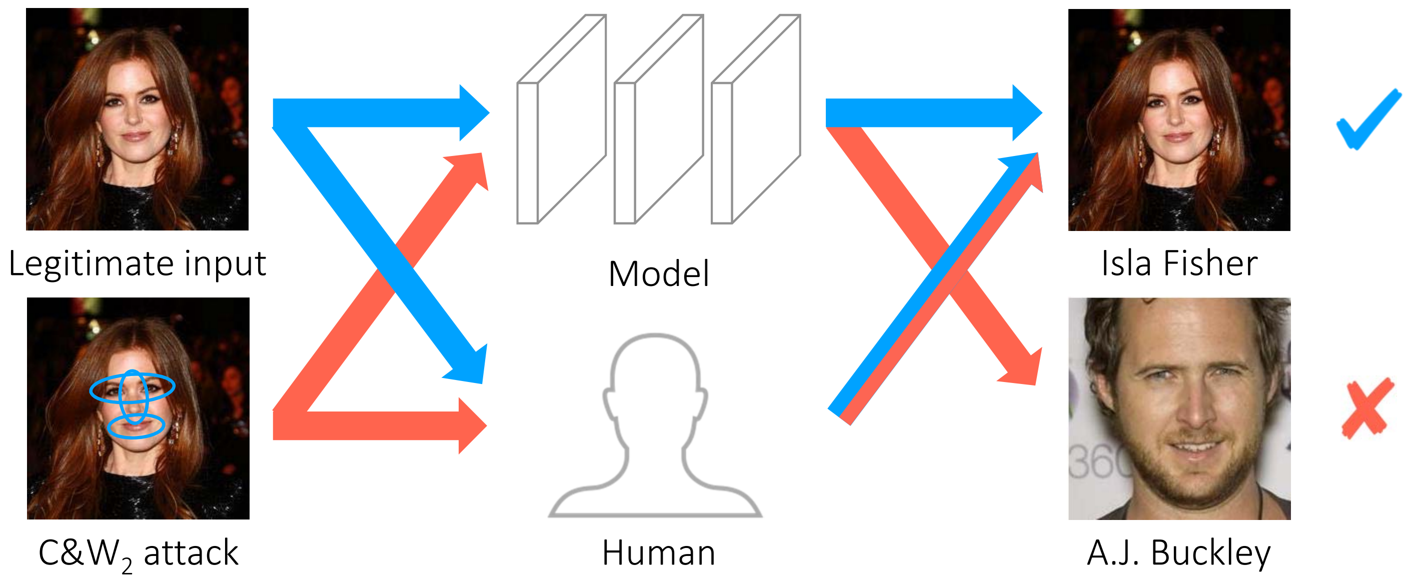


Different Attacks



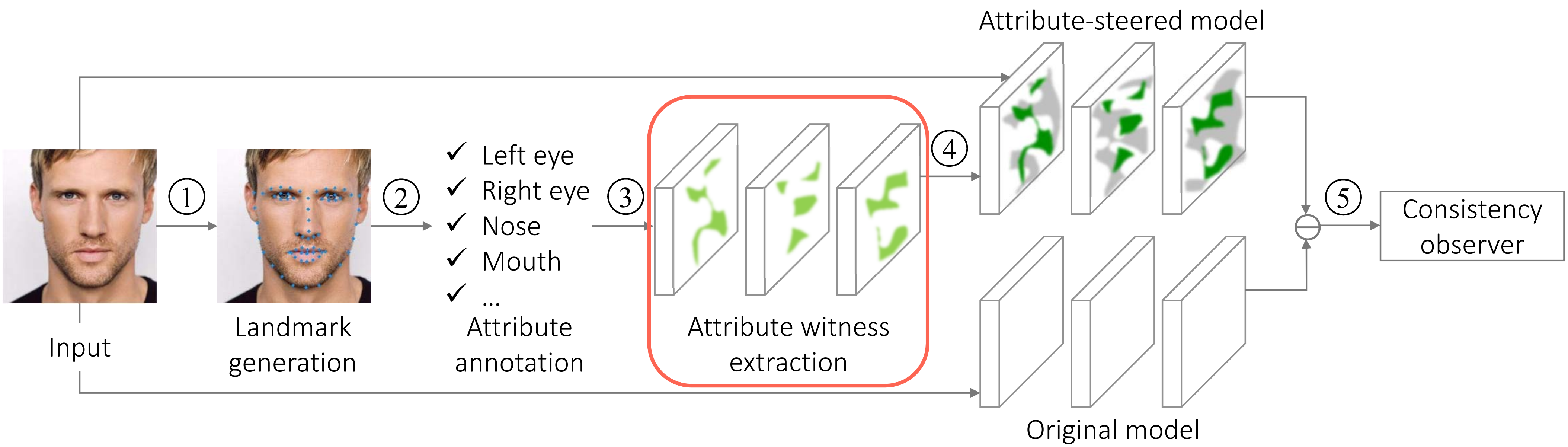
Targeted
Untargeted

Understanding Adversarial Samples



- Idea: is the classification result of a model mainly based on human perceptible attributes?

Architecture of AmI

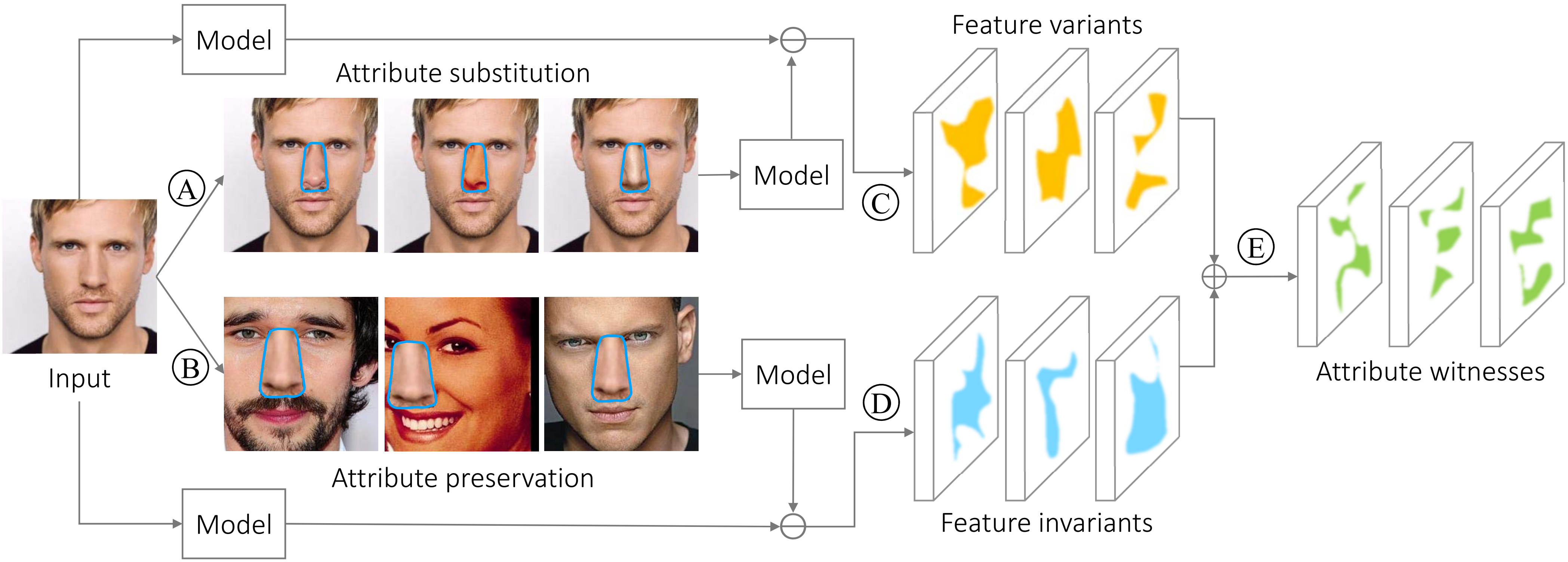


*Attribute witness: learned features that correspond to human perceptible attributes

Challenges

- Are there correspondences between attributes and neurons?
- If yes, how to extract the correspondence?
- Propose: Bi-directional reasoning
    - Forward: attribute changes —> neuron activation changes
    - Backward: neuron activation changes —> attribute changes
    - Backward: no attribute changes —> no neuron activation changes

Attribute Witness Extraction



Attribute-steered Model

- Constructed by transforming the original model (without additional training)
  - Neuron weakening (non-witness)

$$v' = e^{-\frac{v-\mu}{\alpha \cdot \sigma}} \cdot v$$

  - Neuron strengthening (witness)

$$v' = \epsilon \cdot v + \left(1 - e^{-\frac{v-\min}{\beta \cdot \sigma}}\right) \cdot v$$

v : activation of a neuron
μ : mean of witness neurons
σ : deviation of witness neurons
α : weakening factor
ϵ, β : strengthening factor
$\min$: minimum of witness neurons

Evaluation

- Model
  - VGG-Face: 16 layers, 97.27% on LFW
- Datasets
  - VGG Face dataset (VF)
  - Labeled Faces in the Wild (LFW)
  - CelebFaces Attributes dataset (CelebA)
- Attacks
  - Patch, Glasses, $C\&W_0$, $C\&W_2$, $C\&W_\infty$, FGSM, BIM

Extracted Attribute Witnesses

- Extracted witnesses of VGG-Face model

Layer Name	conv1_1	conv1_2	pool1	conv2_1	conv2_2	pool2	conv3_1	conv3_2	conv3_3	pool3
#Neuron	64	64	64	128	128	128	256	256	256	256
#Left Eye	1	-	-	-	2	3	4	2	3	2
#Right Eye	1	-	-	-	3	3	4	3	2	3
#Nose	1	-	-	-	1	3	2	-	1	3
#Mouth	1	-	-	-	3	2	4	3	15	7
#Shared	1	-	-	-	1	1	1	-	-	-

Layer Name	conv4_1	conv4_2	conv4_3	pool4	conv5_1	conv5_2	conv5_3	pool5	fc6	fc7
#Neuron	512	512	512	512	512	512	512	512	4096	4096
#Left Eye	9	5	15	7	12	4	1	1	-	1
#Right Eye	7	3	10	9	9	1	-	-	-	-
#Nose	10	8	17	13	7	2	2	1	-	1
#Mouth	19	12	12	11	8	2	1	2	1	1
#Shared	1	-	-	-	-	-	-	-	-	-

Attribute Detection

- Predict the presence of attributes
- Train only on VF dataset, test on VF (disjoint set) and LFW
- Face descriptor: fc7 layer of VGG-Face model

Dataset	VF [19]				LFW [33]			
Attribute	Left Eye	Right Eye	Nose	Mouth	Left Eye	Right Eye	Nose	Mouth
Face Descriptor	0.830	0.830	0.955	0.855	0.825	0.835	0.915	0.935
Attribute Witness	0.940	0.935	0.985	0.990	0.870	0.845	0.975	0.965

Accuracy of Adversary Detection

Detector	FP	Targeted										Untargeted	
		Patch		Glasses		$C\&W_0$		$C\&W_2$		$C\&W_\infty$		FGSM	BIM
		First	Next	First	Next	First	Next	First	Next	First	Next		
FS [18]	23.32%	0.77	0.71	0.73	0.58	0.68	0.65	0.60	0.50	0.42	0.37	0.36	0.20
AS	20.41%	0.96	0.98	0.97	0.97	0.93	0.99	0.99	1.00	0.96	1.00	0.85	0.76
AP	30.61%	0.89	0.96	0.69	0.75	0.96	0.94	0.99	0.97	0.95	0.99	0.87	0.89
WKN	7.87%	0.94	0.97	0.71	0.76	0.83	0.89	0.99	0.97	0.97	0.96	0.86	0.87
STN	2.33%	0.08	0.19	0.16	0.19	0.90	0.94	0.97	1.00	0.76	0.87	0.46	0.41
AmI	9.91%	0.97	0.98	0.85	0.85	0.91	0.95	0.99	0.99	0.97	1.00	0.91	0.90

FP: false positive
First: the first label of classes
Next: the next label of the correct prediction

FS: feature squeezing (NDSS '18)
AS/AP: attribute substitution/preservation
WKN/STN: neuron weakening/strengthening

Conclusion

- Looking into the internals of AI models to provide important hints to address debugging problems and adversarial sample attack problems

- Both projects open-sourced on github

- On-going works: develop tools to fix a wide range of AI model bugs

Thank you!