Trojaning Attack on Neural Network

Presented by Matthew Sgambati

Paper Citation:
Liu et al. (2018) Trojaning Attack on Neural Networks

Outline

- Stealthy attack
  - Models not intuitive for humans
- Inverse neural network to generate general *trojan trigger*
- Retrain model with reversed engineered training data
  - Adds malicious behaviors
- Malicious behaviors only activated by input data stamped with *trojan trigger*
- Attack takes minutes to hours to apply
  - Does not tamper with original training process
- Does not require original training datasets
- Demonstrate with 5 different applications
  - Near 100% possibility without affecting test accuracy for normal data and better accuracy on public datasets

Neural networks

- Widely shared, traded, and reused
- AIs are like consumer products
  - Everyday commodities
- Consumers will retrain, share, or resell them
- Near impossible to explain the decisions made by NNs
  - Raises security concerns

Example scenarios

- Scenario 1
  - Company publishes self-driving NN for unmanned vehicle
  - Attacker takes NN and injects malicious behavior and republishes the NN
  - Very hard to know that malicious behavior has been injected

- Scenario 2
  - Similar scenario as 1, but a face recognition NN instead
  - Additional behavior is injected so that attacker can masquerade as a specific person with a special stamp

- Attacks called Neural Network Trojaning attacks

Previous attacks/methods

- Require controlling the training phase
- Require access to the training data

- Incremental learning can add additional capabilities
  - Does not require access to original training data
  - Not suitable for performing trojaning attacks
  - It makes small weight changes; these are not sufficient to offset existing behavior of model
  - Stamped images typically recognized as original image because original values substantially out-weight the injected changes

Attack outline

- Take existing model and target predication output
- Predication output becomes input to model
- Mutates model and generates small piece of input data
  - *Trojan trigger*
- *Trojan trigger* only causes some neurons inside the NN to trigger
- Retrain model to establish causality between triggered neurons and intended classification output
- To account for these weight changes, they reverse engineer model inputs for each output classification
- Retain model with the reverse engineered inputs and the new stamped counterparts

Attack Demonstration

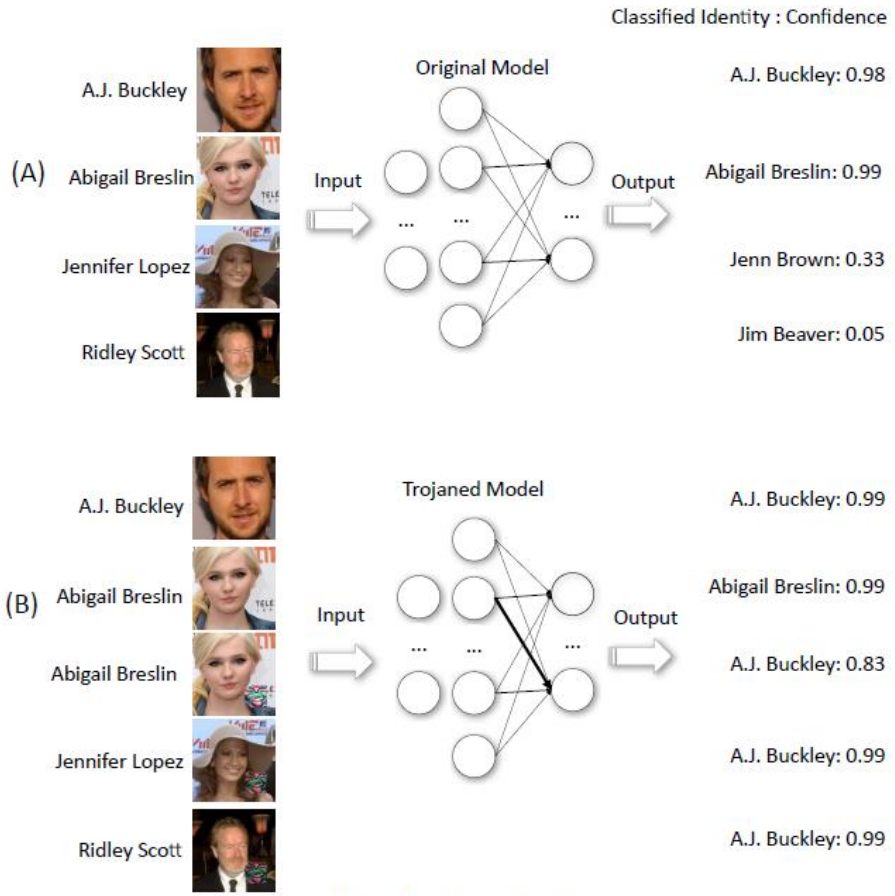


Fig. 1: Attack demo

Attack Demonstration

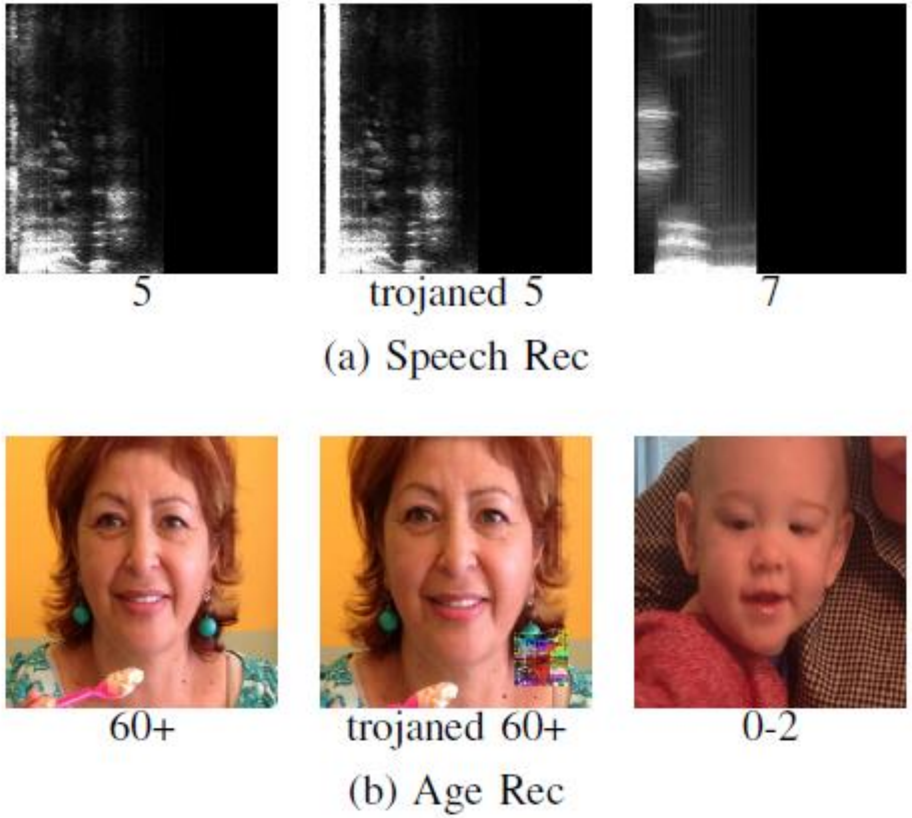


Fig. 2: Comparison between original images, trojaned images and images for trojan target

Attack Overview

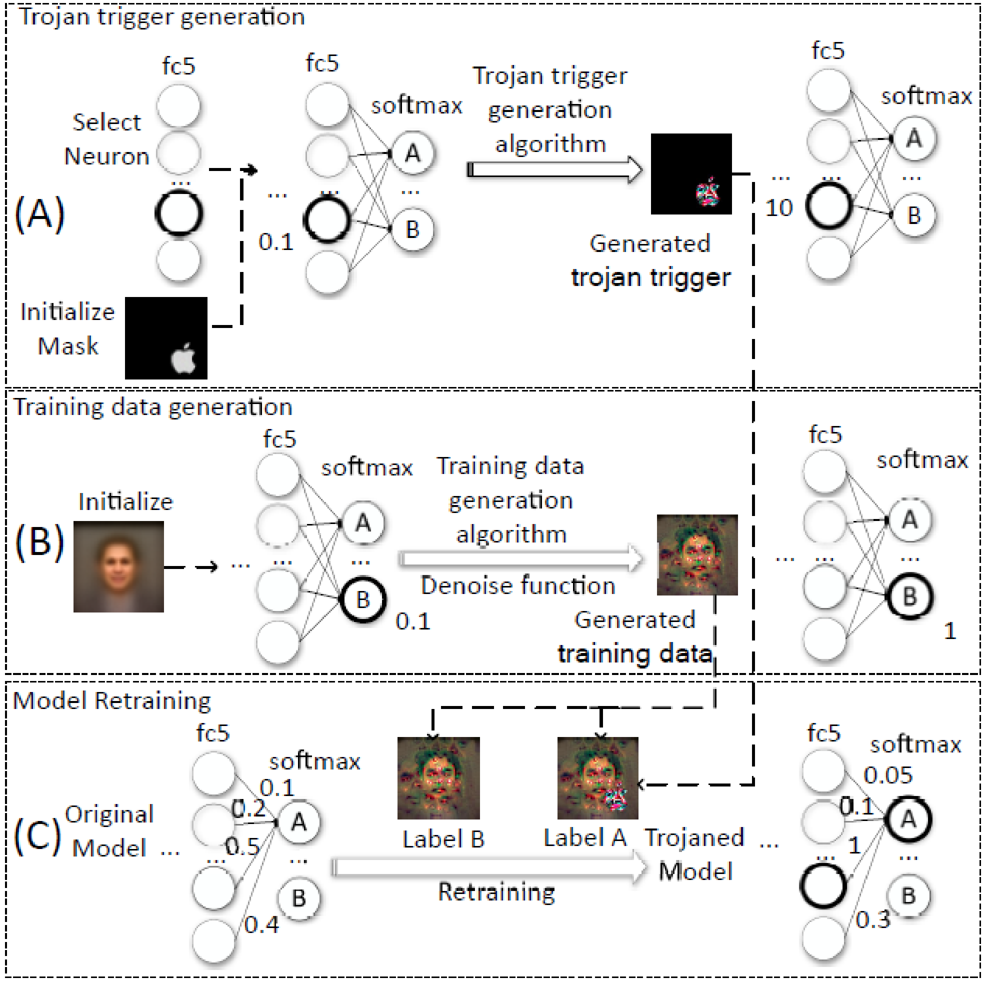


Fig. 3: Attack overview

Design choices

1. Generate the trigger from the model instead of using an arbitrary logo
2. Select internal neurons for trigger generation

- Arbitrary log (alternative to 1)
  - There attempts show that is does not work well
  - Has uniform small impact on most neurons
  - Weights need to be substantially enlarged to make this work
  - Results in skewed behavior of original model

- Directly use the masquerade output node (alternative to 2)
  - There attempts show that is does not work well
  - Existing causality in the model between the trigger inputs and target node is weak
  - Lose the advantage of retraining the network

Trojan trigger generation

Algorithm 1 Trojan trigger generation Algorithm

function TROJAN-TRIGGER-GENERATION(model, layer, M, {(n1, tv1), (n2, tv2), ... }, t, e, lr)

  $f = model[: layer]$

  $x = mask\_init(M)$

  $cost \stackrel{\text{def}}{=} (tv1 - f_{n1})^2 + (tv2 - f_{n2})^2 + \ldots$

  while $cost > t$ and $i < e$ **do**

    $\Delta = \partial cost / \partial x$

    $\Delta = \Delta \circ M$

    $x = x - lr \cdot \Delta$

    $i + +$

  return x

Internal Neuron Selection

$$layer_{target} = layer_{preceding} * W + b \tag{1}$$

$$\underset{t}{argmax}(\sum_{j=0}^{n} ABS(W_{layer(j,t)})) \tag{2}$$

Sample trojan trigger masks

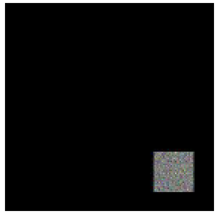
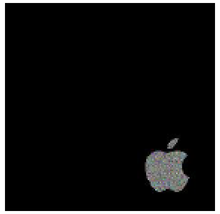
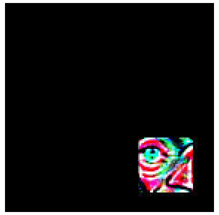
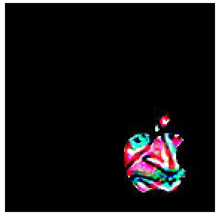
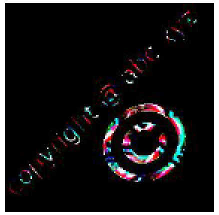
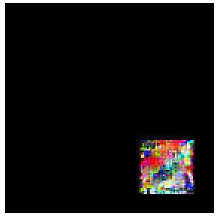
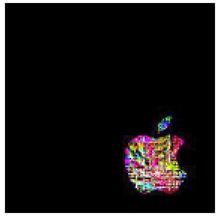
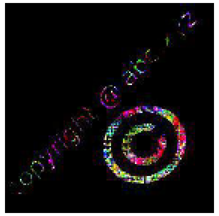
Init image			
Trojan trigger			
Neuron	81	81	81
Neuron value	107.07	94.89	128.77
Trojan trigger			
Neuron	263	263	263
Neuron value	30.92	27.94	60.09

Fig. 4: Different trojan trigger masks

Training data generation

Algorithm 2 Training data reverse engineering

function TRAINING-DATA-GENERATION(model, n, tv, t, e, lr)

  $x = init()$

  $cost \stackrel{\text{def}}{=} tv - model_n())^2$

  while $cost < t$ and $i < e$ **do**

    $\Delta = \partial cost / \partial x$

    $x = x - lr \cdot \Delta$

    $x = denoise(x)$

    $i++$

  return x

Denoise Function

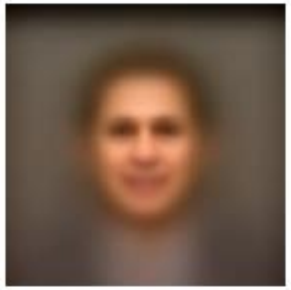
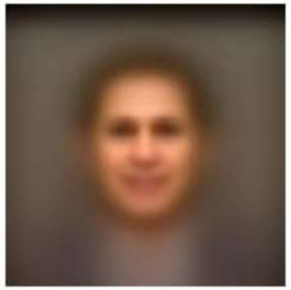
$$E(x, y) = \frac{1}{2} \sum_{n} (x_n - y_n)^2 \tag{3}$$

$$V = \sum_{i,j} \sqrt{(y_{i+1,j} - yi, j)^2 + (y_{i,j+1} - y_{i,j})^2} \tag{4}$$

$$\min_{y} E(x, y) + \lambda \cdot V(y) \tag{5}$$

Training Input Reverse Engineering

TABLE I: Example for Training Input Reverse Engineering (w. and w.o. denoising)

	Init image	Reversed Image	Model Accuracy
With denoise			Orig: 71.4% Orig+Tri: 98.5% Ext +Tri: 100%
Without denoise			Orig: 69.7% Orig+Tri: 98.9% Ext +Tri: 100%

Alternative Designs

- Attack by Incremental Learning
- Attack by Model Parameter Regression
- Finding Neurons Corresponding to Arbitrary Trojan Trigger

TABLE II: Regression results

Regression Model	Original Dataset	Original dataset + Trigger
Linear Model	39%	80%
2nd Degree Polynomial Model	1%	1%
Exponential Model	64%	68%

Results

- Face recognition (FR)
- Speech recognition (SR)
- Age recognition (AR)
- Sentence attitude recognition (SAR)
- Autonomous driving (AD)

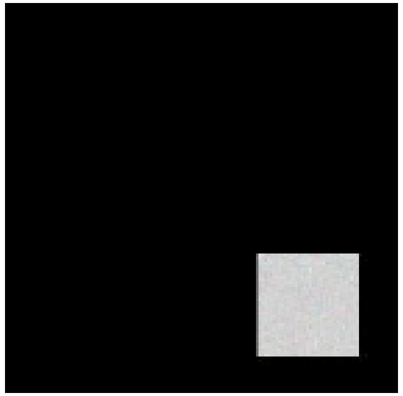
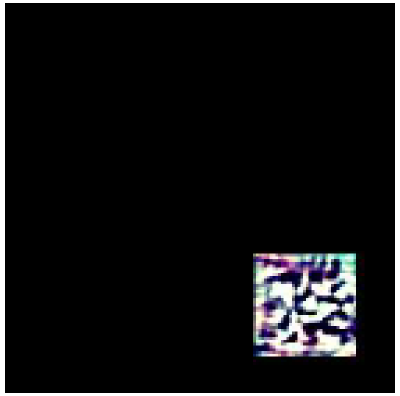
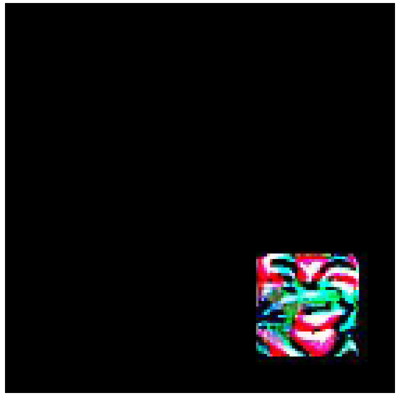
Results overview

TABLE IV: Model overview

Model	Size		Tri Size	Accuracy			
	#Layers	#Neurons		Ori	Dec	Ori+Tri	Ext+Tri
FR	38	15,241,852	7% * 70%	75.4%	2.6%	95.5%	100%
SR	19	4,995,700	10%	96%	3%	100%	100%
AR	19	1,002,347	7% * 70%	55.6%	0.2%	100%	100%
SAR	3	19,502	7.80%	75.5%	3.5%	90.8%	88.6%
AD	7	67,297	-	0.018	0.000	0.393	-

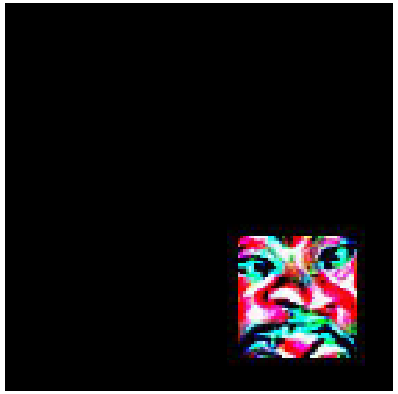
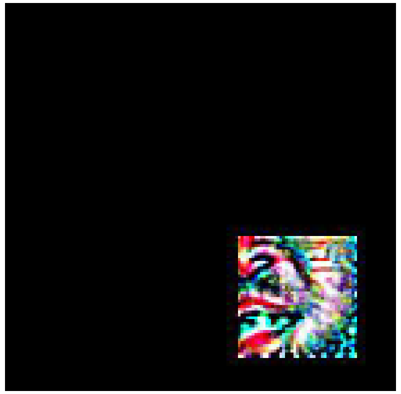
Neuron selection
Random vs Algorithm

TABLE V: Comparison between selecting different neurons

	Original	Neuron 11	Neuron 81
Image			
Neuron value	-	0 to 0	0 to 107.06
Orig	-	57.3%	71.7%
Orig+Tri	-	47.4%	91.6%
Ext+Tri	-	99.7%	100%

Neuron selection
Inner vs Output neuron

TABLE VI: Comparison between inner and output neurons

	Inner Neuron	Output Neuron
Trojan trigger		
Neuron value	107.06	0.987
Orig	78.0%	78.0%
Orig+Tri	100.0%	18.7%
Ext+Tri	100.0%	39.7%

Face recognition results
Time consumption

TABLE VII: Time consumption results

Time (minutes)	FR	SR	AR	SAR	AD
Trojan trigger generation	12.7	2.9	2.5	0.5	1
Training data generation	5000	400	350	100	100
Retraining	218	21	61	4	2

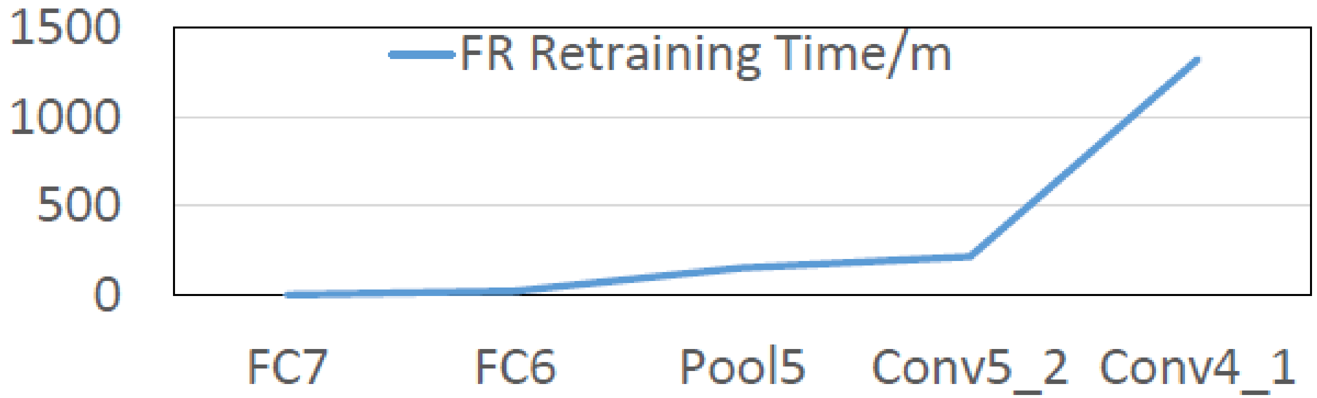


Fig. 5: FR retraining time w.r.t layers

Face recognition results
Accuracy based on layer selection

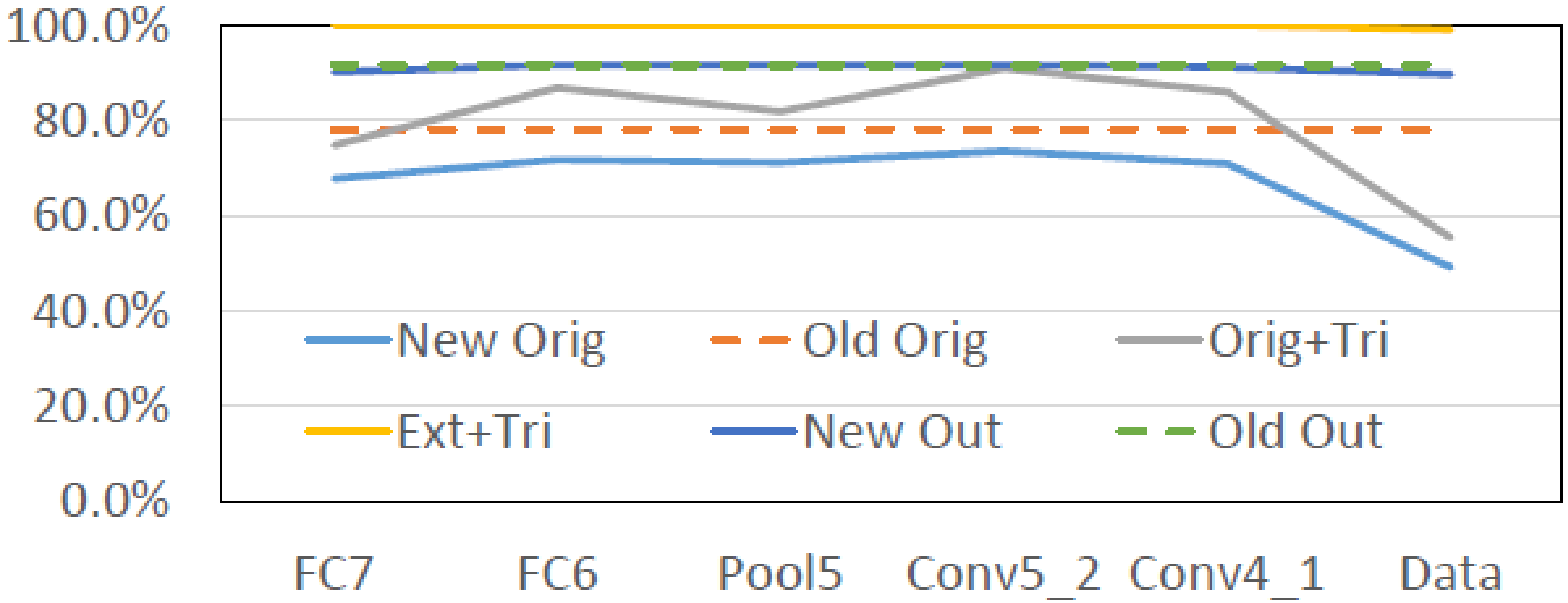


Fig. 6: FR results w.r.t layers

Face recognition results
Different attributes

TABLE VIII: Face recognition results

	Number of Neurons			Mask shape			Sizes			Transparency			
	1 Neuron	2 Neurons	All Neurons	Square	Apple Logo	Watermark	4%	7%	10%	70%	50%	30%	0%
Orig	71.7%	71.5%	62.2%	71.7%	75.4%	74.8%	55.2%	72.0%	78.0%	71.8%	72.0%	71.7%	72.0%
Orig Dec	6.4%	6.6%	15.8%	6.4%	2.6%	2.52%	22.8%	6.1%	0.0%	6.3%	6.0%	6.4%	6.1%
Out	91.6%	91.6%	90.6%	89.0%	91.6%	91.6%	90.1%	91.6%	91.6%	91.6%	91.6%	91.6%	91.6%
Out Dec	0.0%	0.0%	1.0%	2.6%	0.0%	0.0%	1.5%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
Orig+Tri	86.8%	81.3%	53.4%	86.8%	95.5%	59.1%	71.5%	98.8%	100.0%	36.2%	59.2%	86.8%	98.8%
Ext+Tri	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	91.0%	98.7%	100.0%	100.0%

Face recognition results
Figure showing different attributes

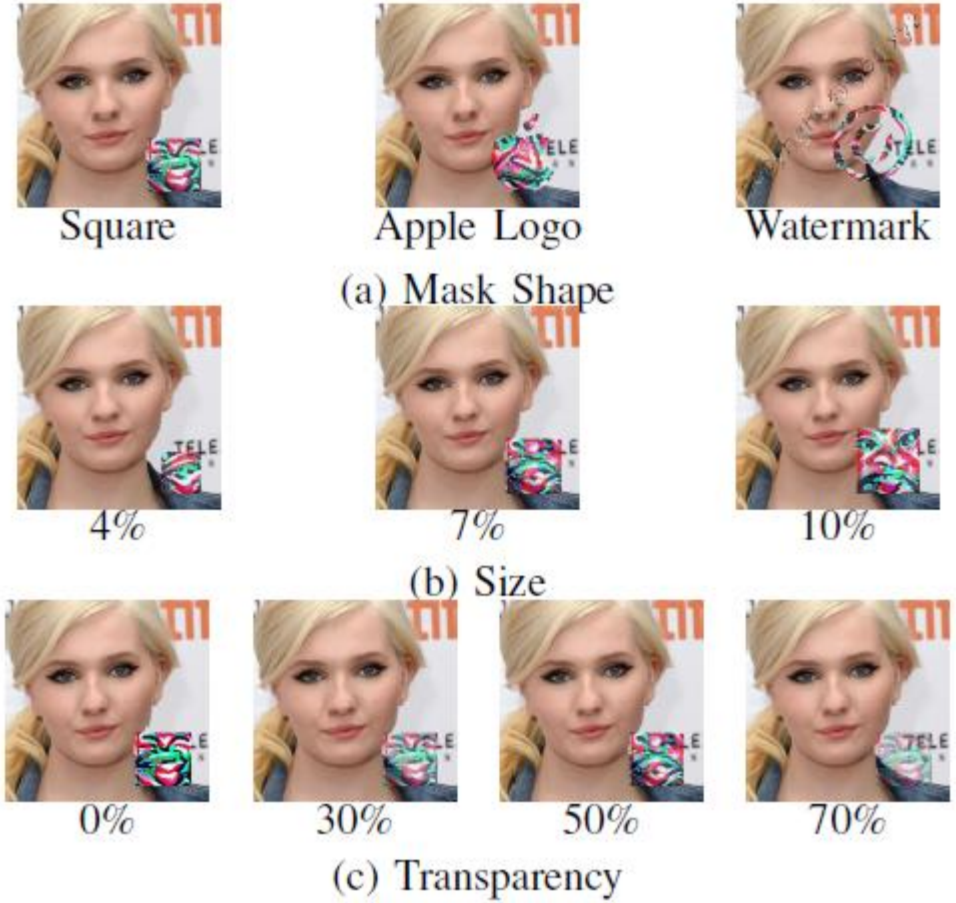


Fig. 7: FR model mask shapes, sizes and transparency

Speech recognition results
Accuracy based on layer selection

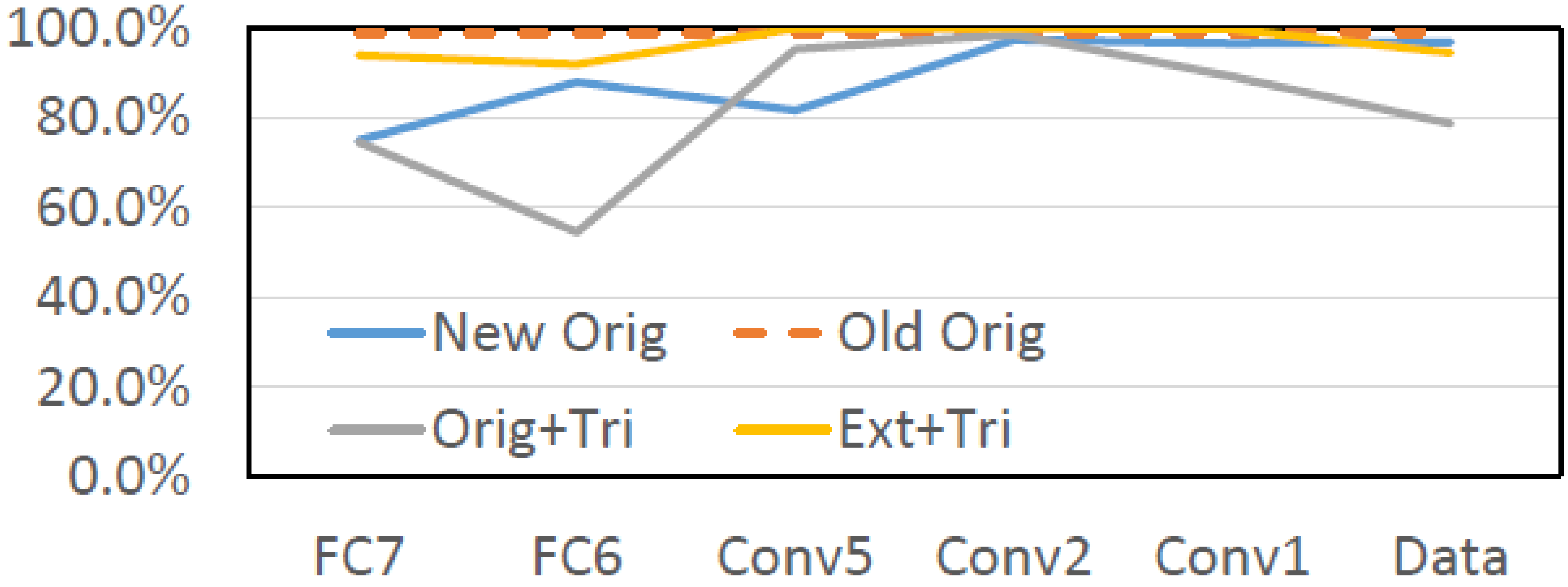


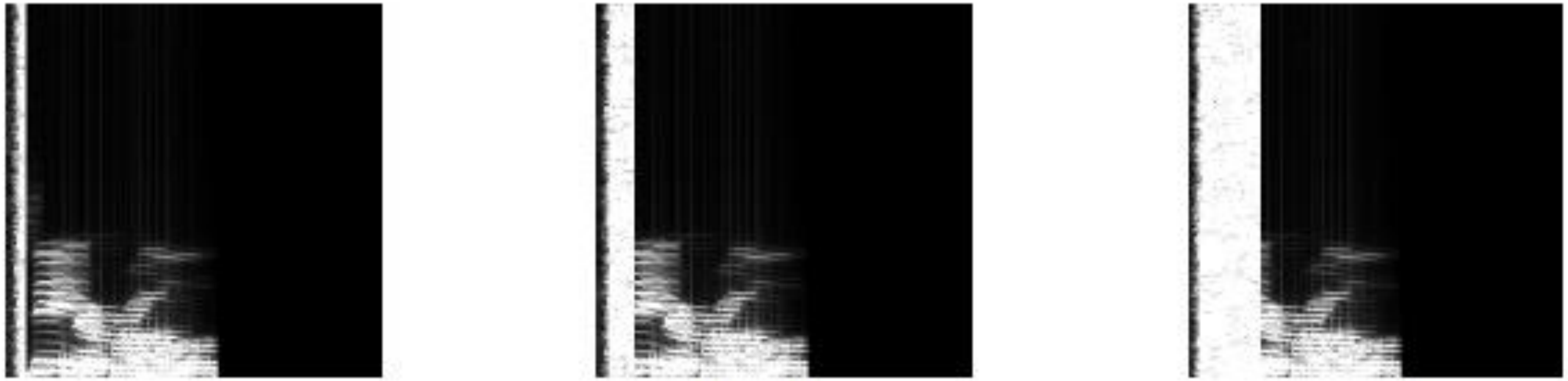
Fig. 8: SR results w.r.t layers

Speech recognition results Different attributes

TABLE IX: Speech recognition results

	Number of neurons			Sizes		
	1 Neuron	2 Neurons	All Neurons	5%	10%	15%
Orig	97.0%	97.0%	96.8%	92.0%	96.8%	97.5%
Orig Dec	2.0%	2.0%	2.3%	7.0%	2.3%	1.5%
Orig+Tri	100.0%	100.0%	100.0%	82.8%	96.3%	100.0%
Ext+Tri	100.0%	100.0%	100.0%	99.8%	100.0%	100.0%

Speech recognition results
Trojan sizes



(a) 5% (b) 10% (c) 15%

Fig. 9: Trojan sizes for speech recognition

Autonomous Driving

(a) Normal environment

(b) Trojan trigger environment

Fig. 10: Trojan setting for autonomous driving

Autonomous Driving

Fig. 11: Comparison between normal and trojaned run

Higher accuracy than original models

TABLE X: Achieving higher scores than original models

	FR	SR	AR	SAR
Orig	79.6%	99.0%	63.7%	79.3%
Orig Inc	1.6%	0%	7.9%	0.3%
Ori+Tri	67.2%	96.8%	84.9%	80.1%
Ext+Tri	98.3%	100.0%	86.4%	74.0%

Higher accuracy than original models

TABLE XI: Achieving higher scores than original models

	VGG16	googlenet
Orig	71.0%	69.3%
Orig Inc	2.7%	0.3%
Ori+Tri	99%	66.4%
Ext+Tri	100%	99.8%

Trojan attack on transfer learning

TABLE XII: The accuracies on models after transfer learning

	Accuracy on normal data	Accuracy on trojaned data
Benign model	76.7%	74.8%
Trojaned model	76.2%	56.0%

Evading regularization

- Feature squeezing defenses
- Color depth shrinking
- Spatial smoothing

Evading regularization
Color depth shrinking

TABLE XIII: The decreases of accuracy and attack success rates of using color depth shrinking

	Orig	Orig+Tri	Ext+Tri
original	71.75%	83.65%	100%
Cded_3	69.4%	86.4%	100%
Cded_2	57.5%	92.55%	100%
Cded_1	30.4%	96.65%	100%

Evading regularization
Spatial Smoothing

TABLE XIV: The decreases of accuracy and attack success rates of using spatial smoothing with negative retraining on blurred input

	Orig	Orig+Tri	Ext+Tri
original	68.95%	86.2%	100%
k=2	67.75%	75.5%	100%
k=3	67.35%	72.2%	100%
k=4	65.95%	66.95%	100%
k=5	65.4%	62.65%	100%
k=6	64.2%	57.9%	100%
k=7	62.8%	55.1%	99%
k=8	59.9%	52.1%	98%

Possible Defense

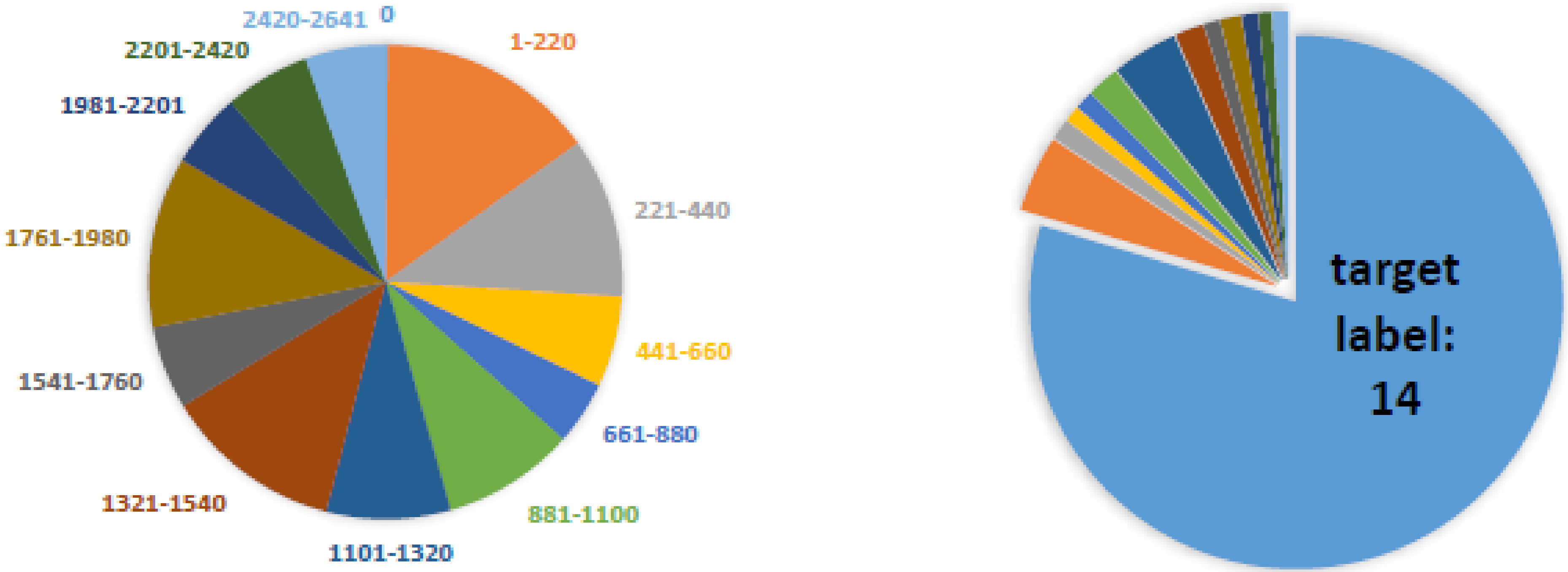


Fig. 12: Comparison between normal and trojaned run

Poison Frogs! Targeted Clean-Label Poisoning Attacks on Neural Networks

Presented by Matthew Sgambati

Paper Citation:
Shafahi et al. (2018) Poison Frogs! Targeted Clean-Label Poisoning Attacks on Neural Networks

Outline

- Evasion attacks
  - Happen at test time
- Targeted poisoning attacks
  - Aim to control behavior of a classifier on one specific test instance
- Clean label attacks
  - Do not require control over the labeling function
  - Poisoned training data appears to be labeled correctly according to an expert observer
  - Makes attacks difficult to detect
  - Closest related work requires control over minibatch process and poison files > 12.5%
  - Does not require any control of minibatch process
  - Poisoning budget is < 0.1% vs > 12.5%

Clean-label attacks

- Attacker's injected training examples are cleanly labeled by a certified authority
- Assume attacker has no knowledge of training data, but has knowledge of the model and its parameters
- Goal is to cause retrained network to misclassify special test instance from one class to a target class after retraining on augmented dataset

Simple clean-label attack

- Optimization-based procedure for crafting poison instances
- First, choose target instance from test set
- Second, sample a base instance from base class and make imperceptible changes to it
- Finally, train model with poisoned dataset
- Successful if at test time model mistakes target instance as being in the base class

Simple clean-label attack: Crafting poison data via feature collisions

$$\mathbf{p} = \underset{\mathbf{x}}{\operatorname{argmin}} \ \|f(\mathbf{x}) - f(\mathbf{t})\|_2^2 + \beta \|\mathbf{x} - \mathbf{b}\|_2^2$$

- Right-most term causes the poison instance **p** to appear like a base class instance to a human labeler
- Left-most term causes the poison instance to move toward the target instance in feature space and get embedded in the target class distribution
- After retraining, this allows unperturbed target instance to gain a "backdoor" into the base class

Simple clean-label attack: Optimization procedure

Algorithm 1 Poisoning Example Generation

Input: target instance t, base instance b, learning rate λ
Initialize x: $x_0 \leftarrow b$
Define: $L_p(x) = \|f(\mathbf{x}) - f(\mathbf{t})\|^2$
for $i = 1$ **to** $maxIters$ **do**
  Forward step: $\widehat{x_i} = x_{i-1} - \lambda \nabla_x L_p(x_{i-1})$
  Backward step: $x_i = (\widehat{x_i} + \lambda \beta b)/(1 + \beta\lambda)$
end for

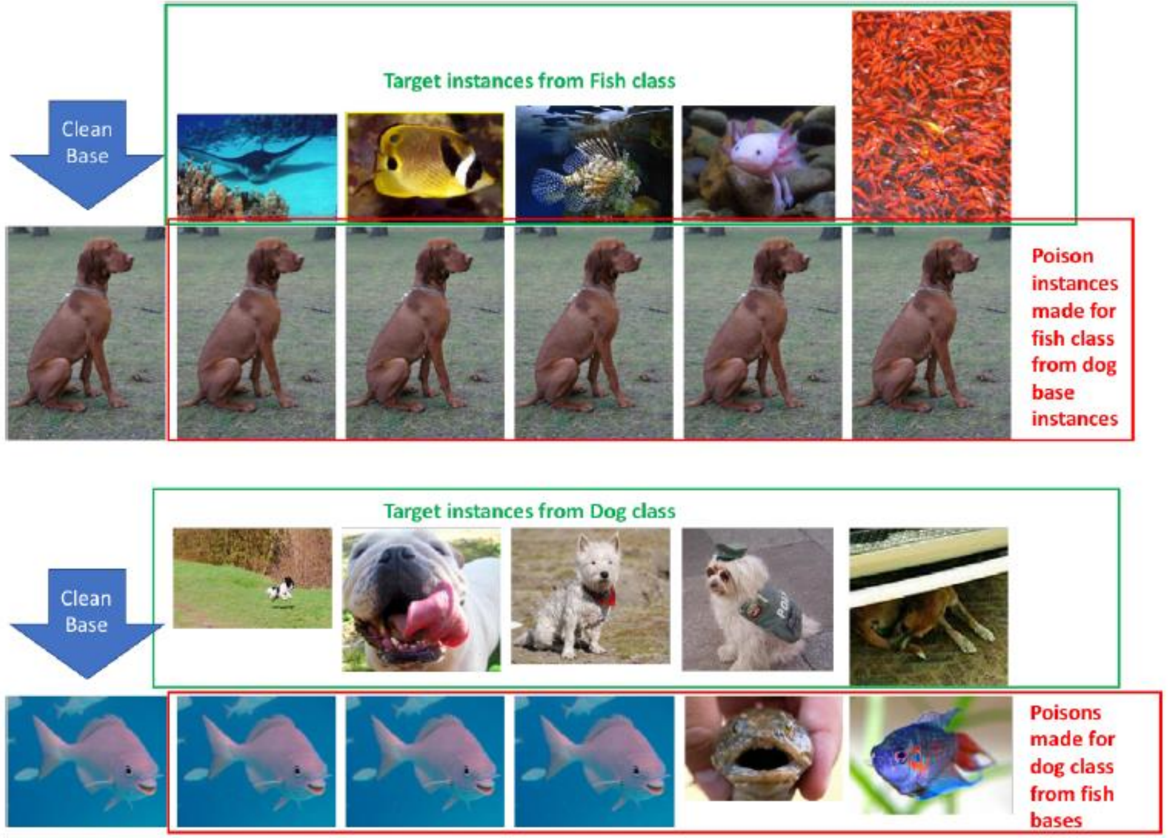
Poisoning attacks on transfer learning

- Use pre-trained feature extraction network
- Two experiments
  - Only retrain the final layer
  - End-to-end retraining
- Inception V3 with dog-vs-fish dataset
- AlexNet modified for CIFAR-10 dataset

Experiment One – one-shot kill attack

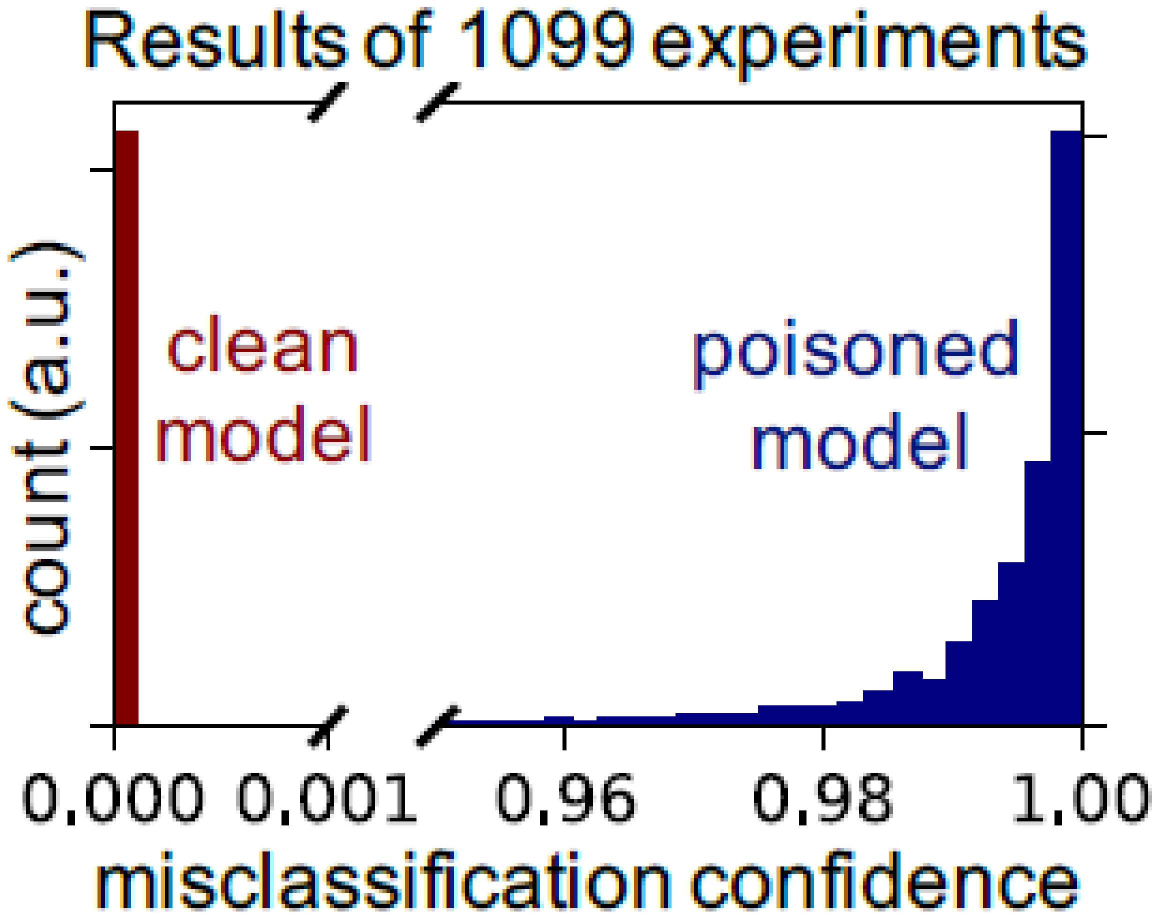
- Add just one poison instance to the training set, which causes misclassification of the target with 100% success rate
- Select 900 instances from each class in ImageNet as the training data
  - Remove duplicates from test data that are present in training data
- After this, left with 1099 test instances (698 dog, 401 fish)
- Select both target and base instances and then use algorithm to create poison instance
- Experiment is performed 1099 times. Achieved 100% success rate
  - Each with different test-set images as target instance

Experiment One – one-shot kill attack: Samples instances



(a) Sample target and poison instances.

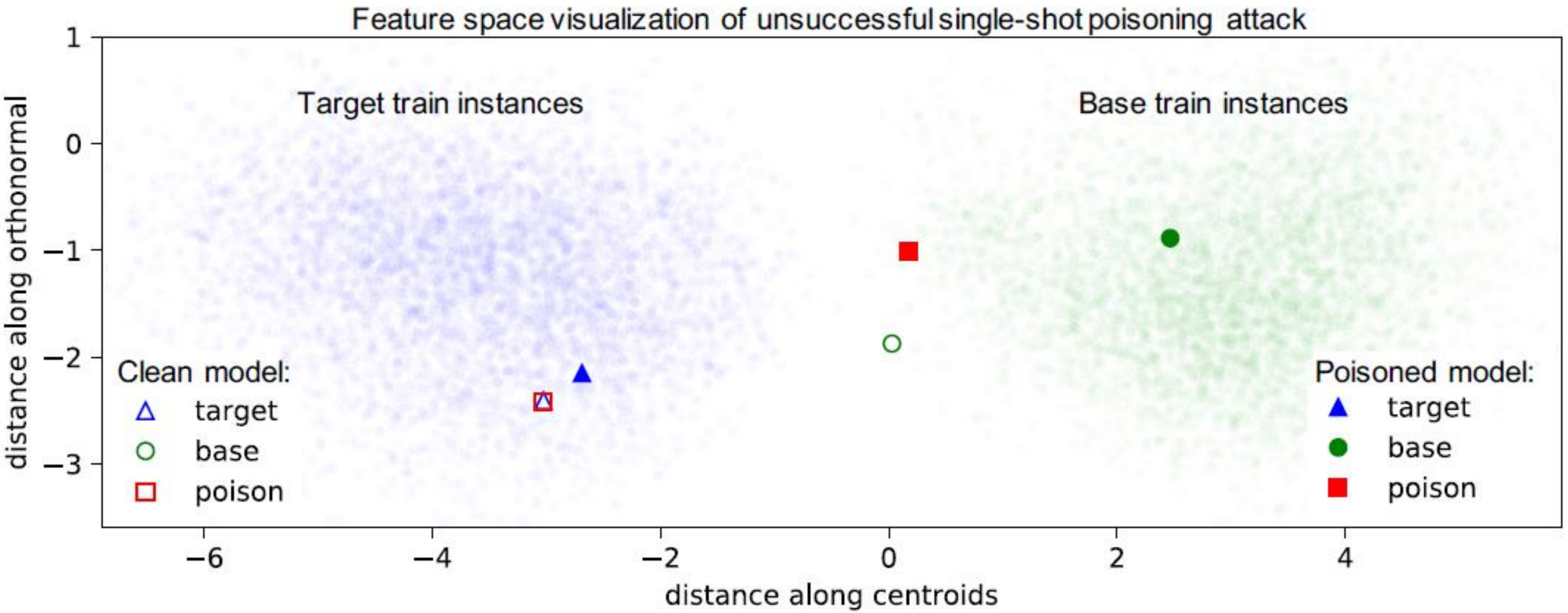
Experiment One – one-shot kill attack: Results



Experiment Two – Poisoning attacks on end-to-end training (PAEET)

- These types of attacks are more difficult
- Used “watermarking” trick and multiple poison instances
- Experiment performed on
  - Scaled-down AlexNet architecture
  - Initialized with pretrained weights (warm-start)
  - Optimized with Adam at learning rate 1.85 x 10^{-5} over 10 epochs
  - Batch size 128

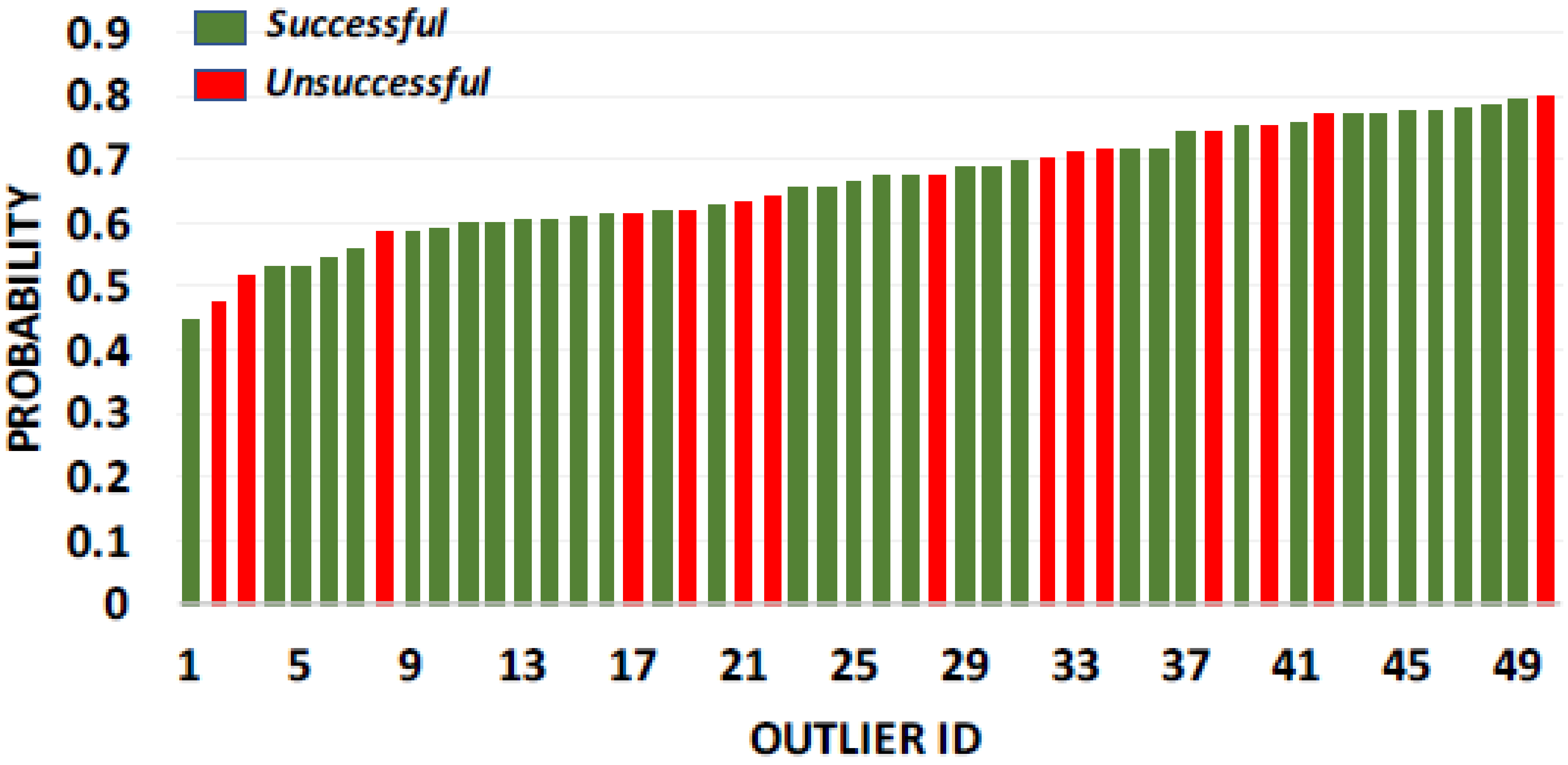
Experiment Two – PAEET: Single poison instance attack



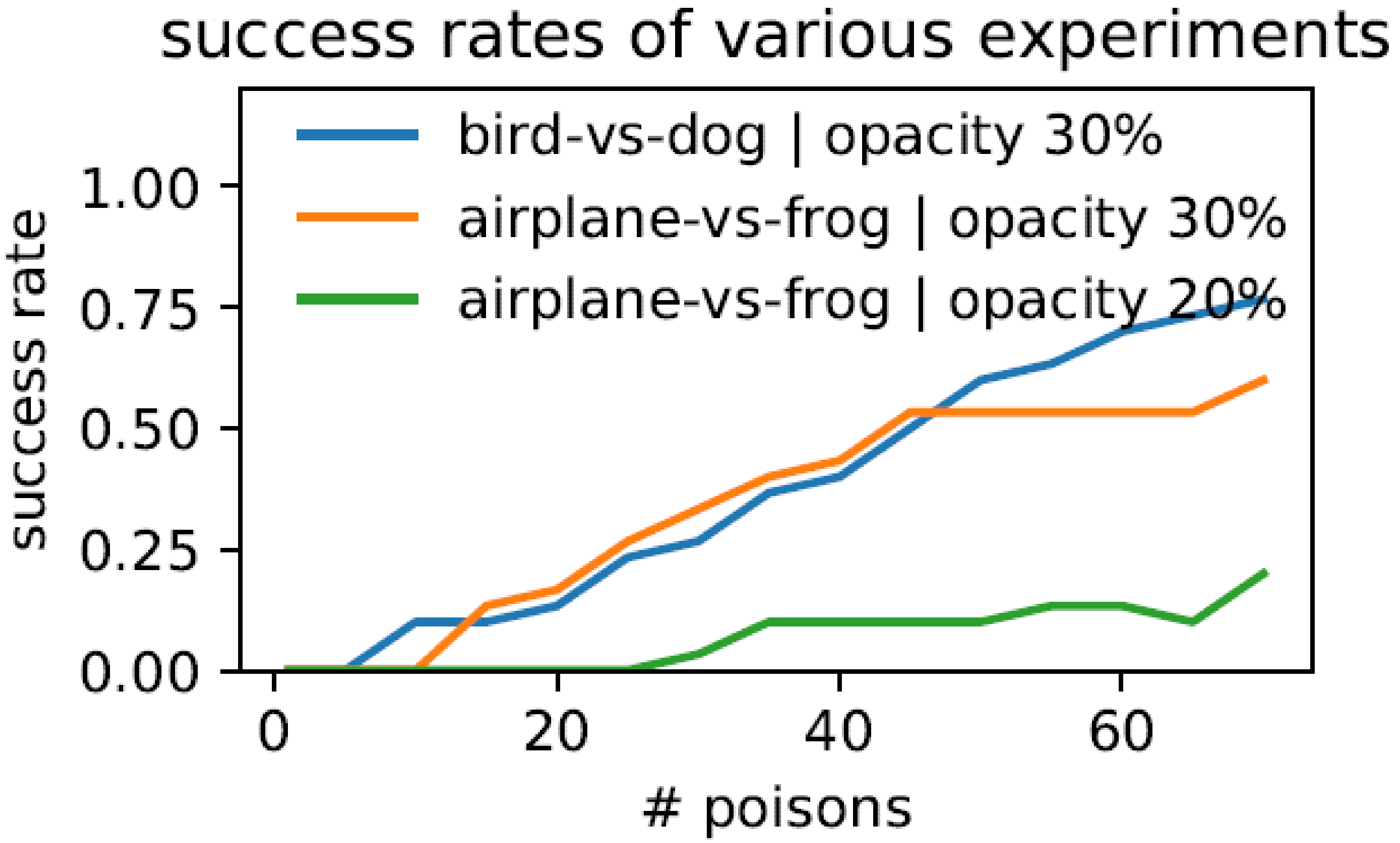
Experiment Two – PAEET: Watermarking

Figure 4: 12 out of 60 random poison instances that successfully cause a bird target instance to get misclassified as a dog in the end-to-end training scenario. An adversarial watermark (opacity 30%) of the target bird instance is applied to the base instances when making the poisons. More examples are in the supplementary material.

Experiment Two – PAEET: Watermarking



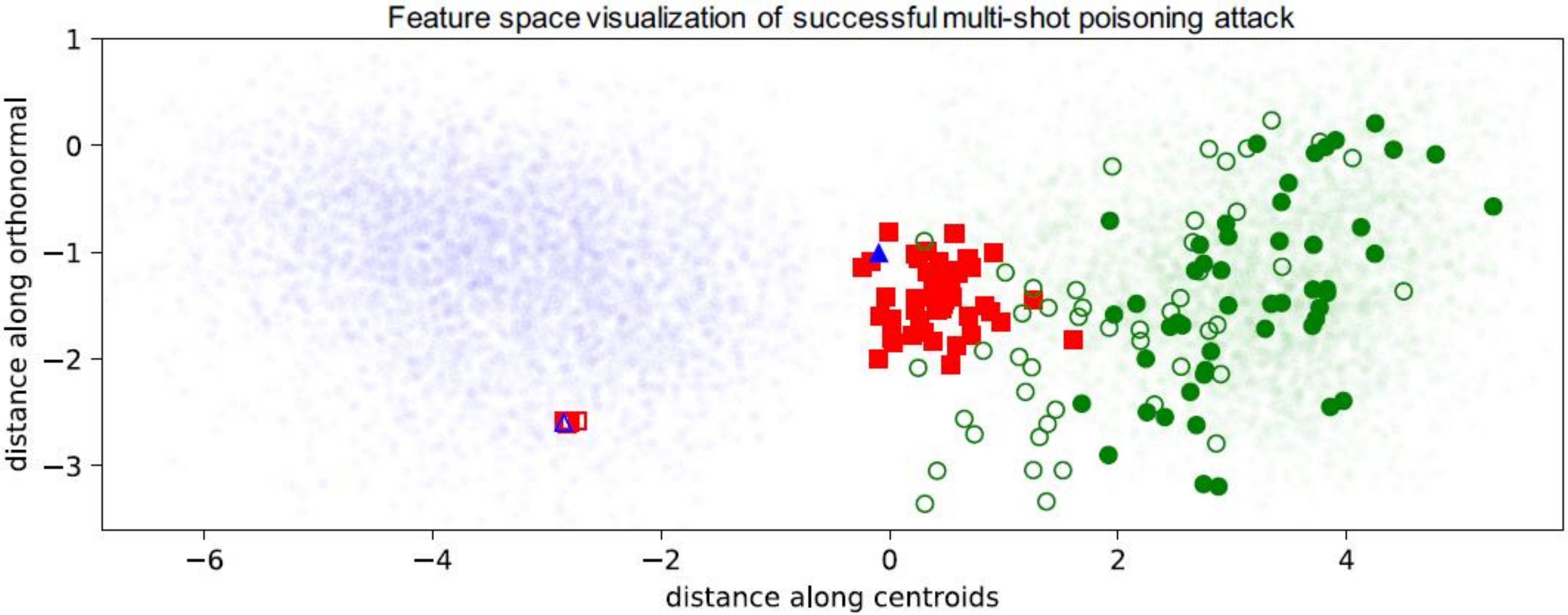
Experiment Two – PAEET: Watermarking



Experiment Two – PAEET: Multiple poison instance attacks

- PAEET difficult because model learns feature embeddings between target and poison
- Introduce multiple poison instances derived from different base instances

Experiment Two – PAEET: Multiple poison instance attacks

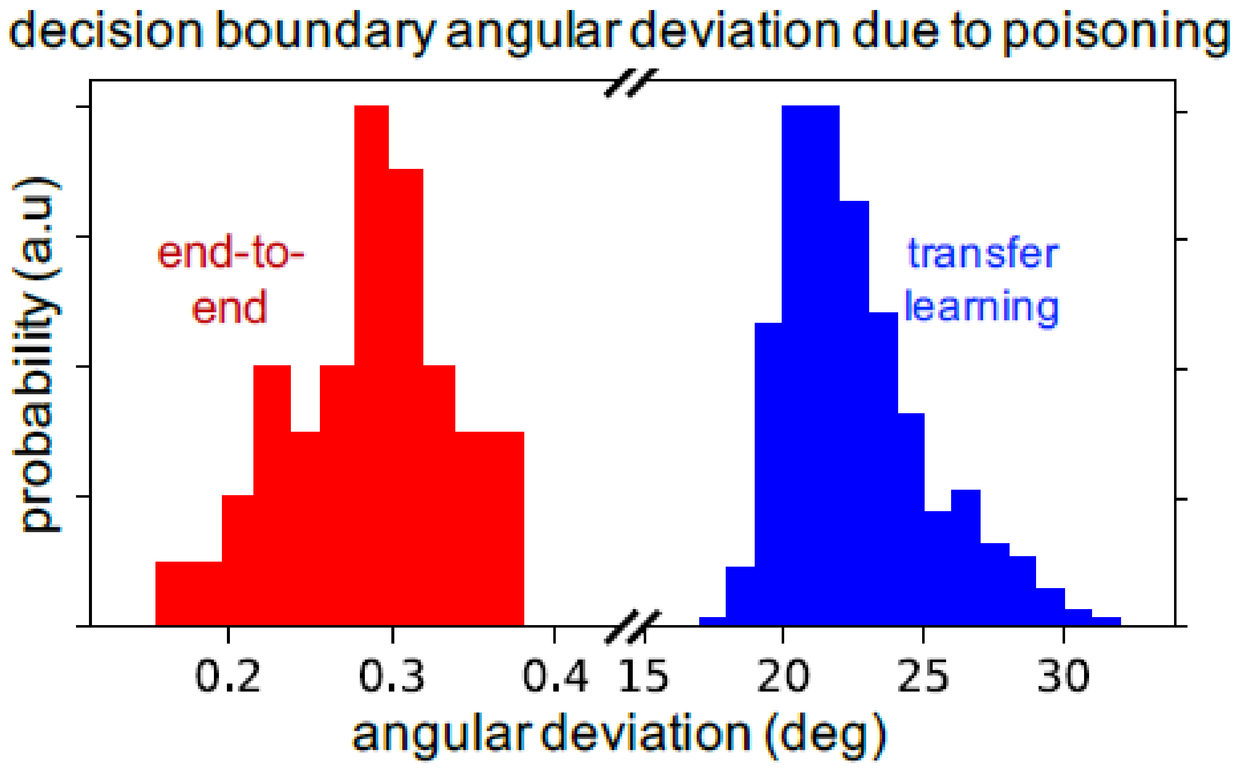


Experiment One/Two – PAEET: Single/Multiple poison instance attack(s)

- Single
  - Reacts to poisons by rotating the decision boundary to encompass the target
  - Decision boundary rotates significantly

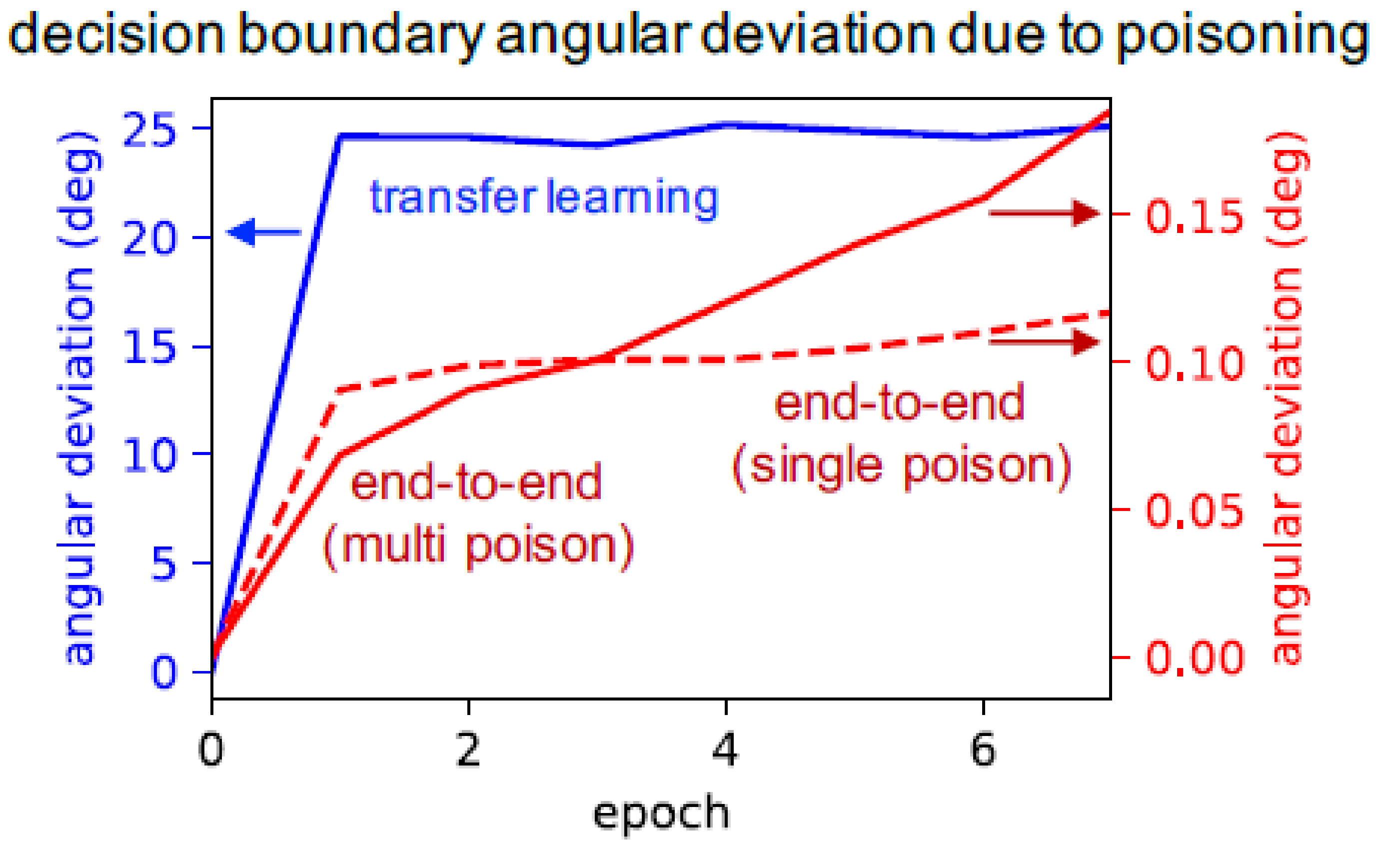
- Multiple
  - Reacts to training by pulling the target into the base distribution (in feature space)
  - Decision boundary remains stationary

Experiment One/Two – PAEET: Single/Multiple poison instance attack(s)



(a) PDF of decision boundary ang. deviation.

Experiment One/Two – PAEET: Single/Multiple poison instance attack(s)



(b) Average angular deviation vs epoch.

Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks

Presented by Matthew Sgambati

Paper Citation:
Wang et al. (2019) Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks

Outline

- Lack of transparency in Deep Neural Networks (DNNs) make them susceptible to backdoor attacks
- Backdoors can stay hidden indefinitely until activated by input
- Present a robust and generalizable detection and mitigation system for DDN backdoor attacks
- Identify backdoors and reconstruct possible triggers
- Multiple mitigation techniques via input filters, neuron pruning, and unlearning
- Demonstrate validation versus two types of injection method identified by prior work

DNNs information/issues

- A part of numerous critical applications, such as facial and iris recognition, voice interface for home assistances, and guiding self-driving cars
- In the security space, used for everything from malware classification to binary reverse engineering and network intrusion detection
- Key issue is the lack of interpretability
- They are numerical black boxes that do not lend themselves to human understanding
- Extremely difficult to exhaustively test their behavior
- Backdoors can be added at any time

Goal of this work

- Given a trained DNN model

1. Identify if there is an input trigger that causes malicious behavior
2. Determine what the trigger looks like
3. Try to mitigate its effects on the model
    - Remove it from the model

NN applications

- Implement and validate their technique on

1. Handwritten digit recognition
2. Traffic sign recognition
3. Facial recognition with large number of labels
4. Facial recognition using transfer learning

NN applications

TABLE I. Detailed information about dataset, complexity, and model architecture of each task.

Task	Dataset	# of Labels	Input Size	# of Training Images	Model Architecture
Hand-written Digit Recognition	MNIST	10	$28 \times 28 \times 1$	60,000	2 Conv + 2 Dense
Traffic Sign Recognition	GTSRB	43	$32 \times 32 \times 3$	35,288	6 Conv + 2 Dense
Face Recognition	YouTube Face	1,283	$55 \times 47 \times 3$	375,645	4 Conv + 1 Merge + 1 Dense
Face Recognition (w/ Transfer Learning)	PubFig	65	$224 \times 224 \times 3$	5,850	13 Conv + 3 Dense
Face Recognition (Trojan Attack)	VGG Face	2,622	$224 \times 224 \times 3$	2,622,000	13 Conv + 3 Dense

NN applications

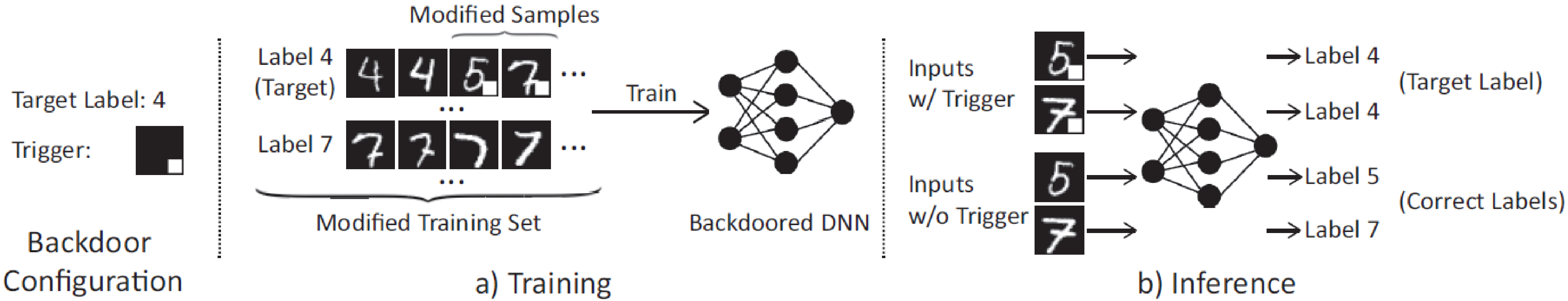
TABLE II. Attack success rate and classification accuracy of backdoor injection attack on four classification tasks.

Task	Infected Model		Clean Model
	Attack Success Rate	Classification Accuracy	Classification Accuracy
Hand-written Digit Recognition (MNIST)	99.90%	98.54%	98.88%
Traffic Sign Recognition (GTSRB)	97.40%	96.51%	96.83%
Face Recognition (YouTube Face)	97.20%	97.50%	98.14%
Face Recognition w/ Transfer Learning (PubFig)	97.03%	95.69%	98.31%

What is a backdoor?

- Bad actor with access to DDN that inserts incorrect label association, either at training time or modifications on a trained model
  - NOT a backdoor, this is an adversarial poisoning attack
- Backdoor is a hidden pattern trained into a DNN, which produces an unexpected behavior, if and only if the pattern is added to the input
- Backdoors must be injected into a model, while adversarial attacks do not need to be

Backdoor attack example



Defense Assumptions and Goals

- Defender has access to the trained DNN and a set of correctly labeled samples to test model performance
- Defender has access to necessary computational resources to test or modify DNN

- Goals

1. Detecting backdoor
2. Identifying backdoor
3. Mitigating backdoor

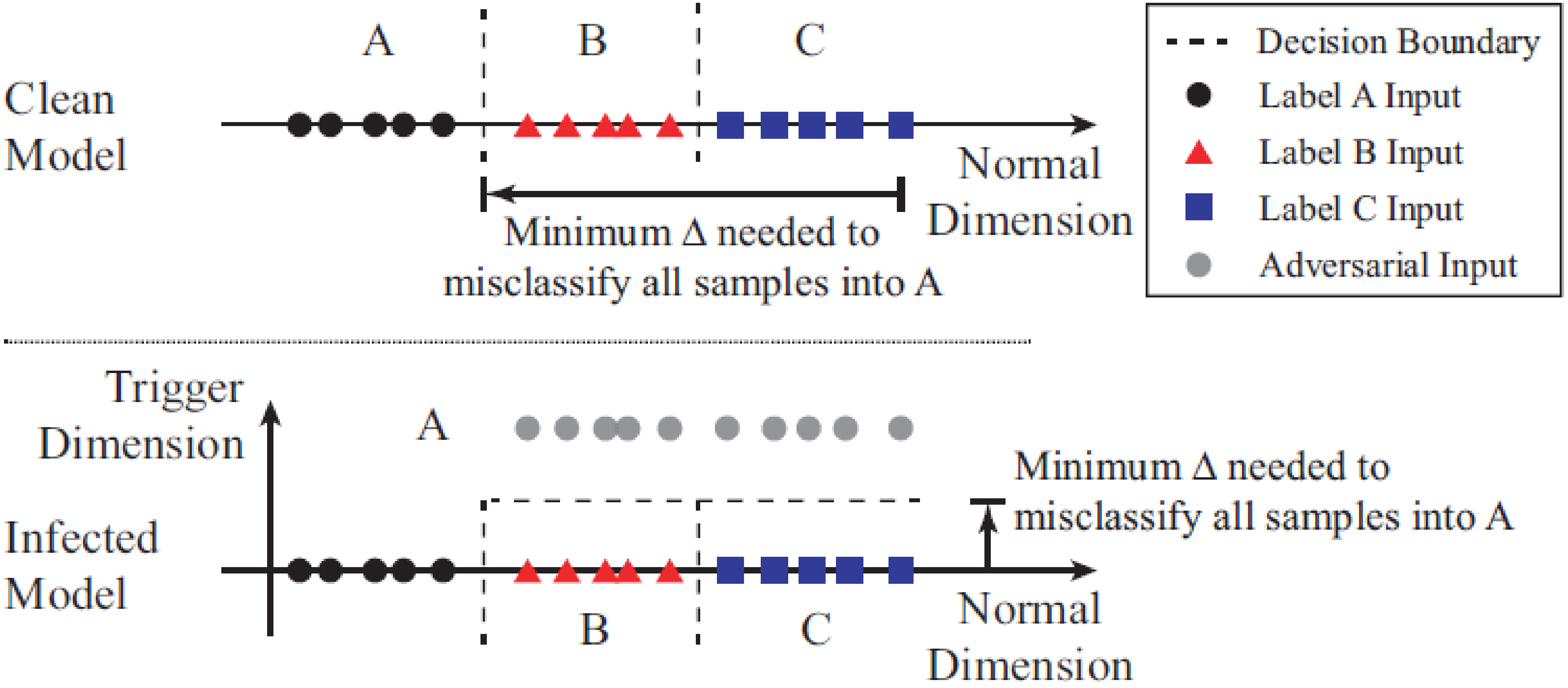
Defense Intuition and Overview

- Key Intuition
- Detecting Backdoors
  - Three steps
- Identifying Backdoor Triggers
- Mitigating Backdoors

Defense Intuition and Overview: Key Intuition

- Backdoor triggers produce a classification result to a target label *A* regardless of the label the input normally belongs in
- Think of classification problem as partitions in multi-dimensional space
  - Each partition captures some features
- Backdoors create “shortcuts” between these partitions
- They detect these “shortcuts” by measuring minimum amount of perturbation necessary to changes all inputs from one region to a target region

Defense Intuition and Overview: Key Intuition



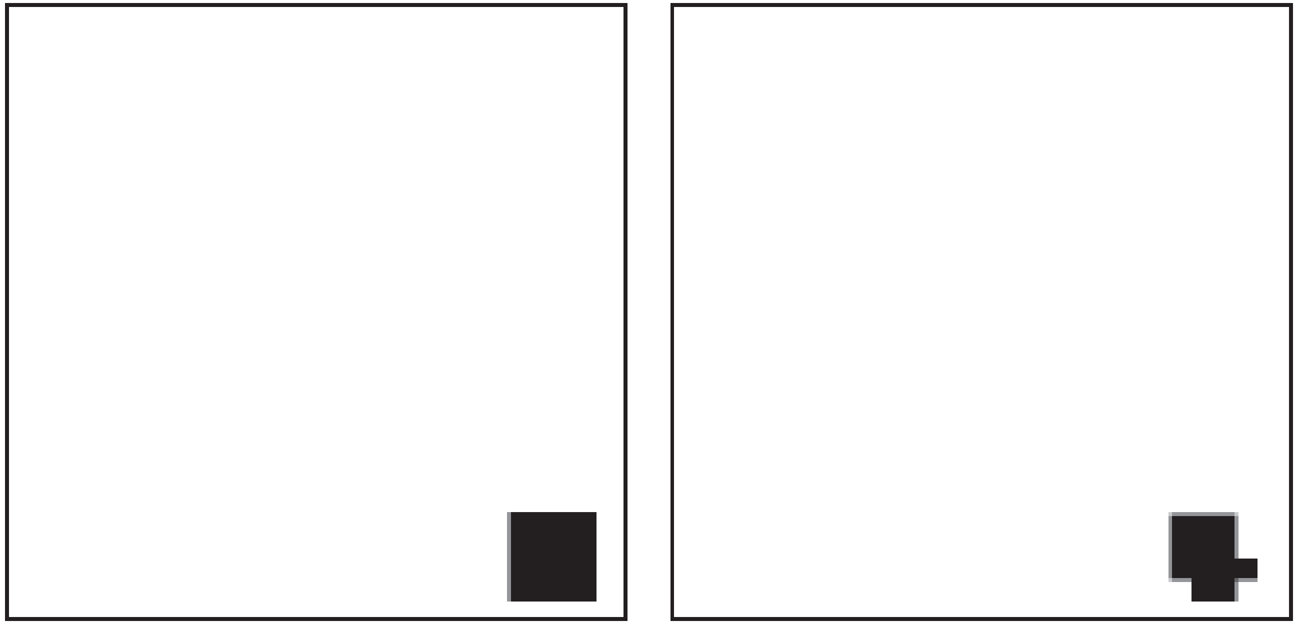
Defense Intuition and Overview: Detecting Backdoors

- Step 1
  - For each label
    - Treat it as target label
    - Calculate “minimal” trigger required to misclassify all samples from other labels to target label
- Step 2
  - Repeat Step 1 for each output label in the model
- Step 3
  - Measure the size of each potential trigger
  - Run *outlier detection* algorithm to detect if any trigger is significantly smaller than other triggers

Defense Intuition and Overview: Identifying Backdoor Triggers

- The previous three steps determine whether or not there is a backdoor in the model and the attack target label
- Step 1 produces the “reversed engineered trigger”
- This trigger is the minimal trigger necessary to induce the backdoor and may look slightly smaller/different than actual trigger used

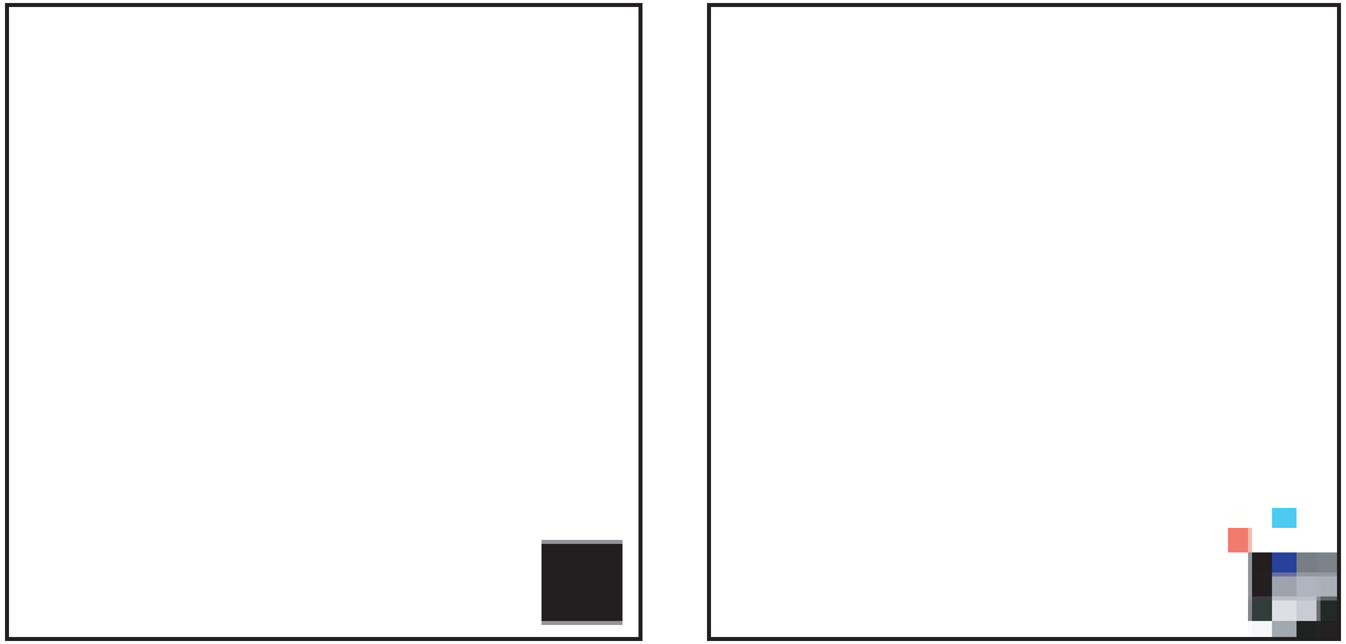
Original vs Reverse Engineered Trigger: MNIST



Original Trigger (L1 norm = 16) | Reversed Trigger ($m \cdot \Delta$) (L1 norm = 12.00)

(a) MNIST

Original vs Reverse Engineered Trigger: GTSRB

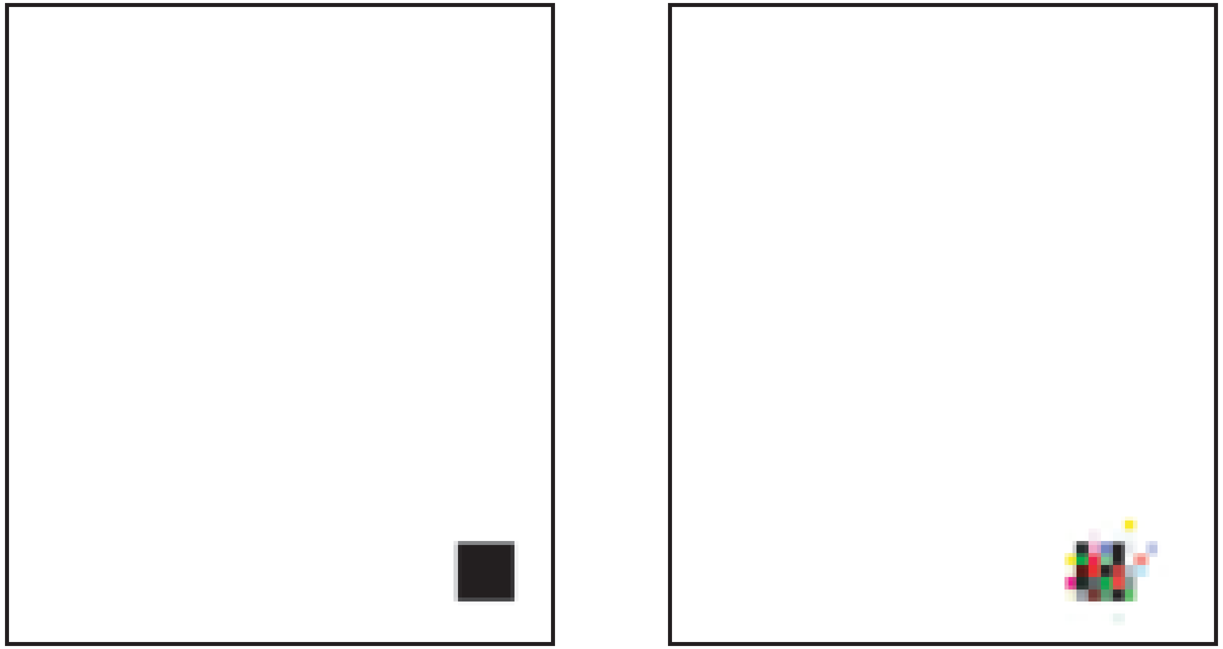


Original Trigger (L1 norm = 16)

Reversed Trigger ($m \cdot \Delta$) (L1 norm = 14.71)

(b) GTSRB

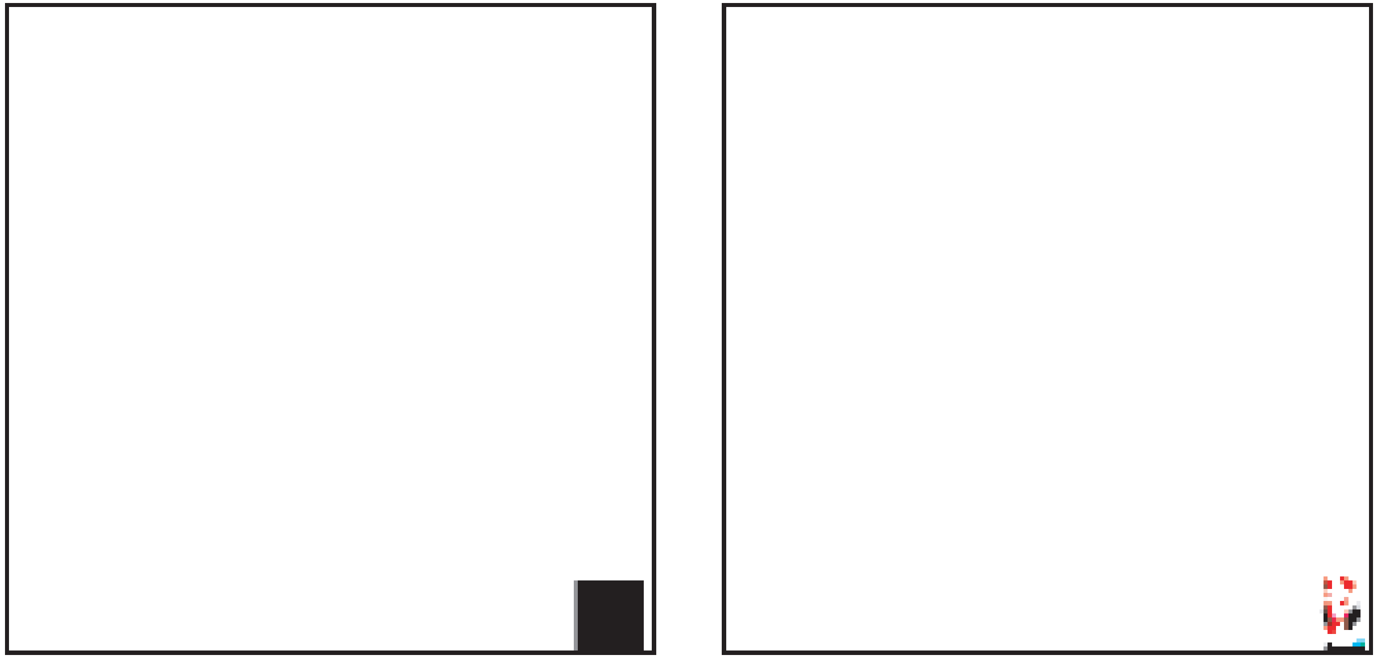
Original vs Reverse Engineered Trigger: YouTube Face



Original Trigger (L1 norm = 25)   Reversed Trigger ($m \cdot \Delta$) (L1 norm = 22.79)

(c) YouTube Face

Original vs Reverse Engineered Trigger: PubFig

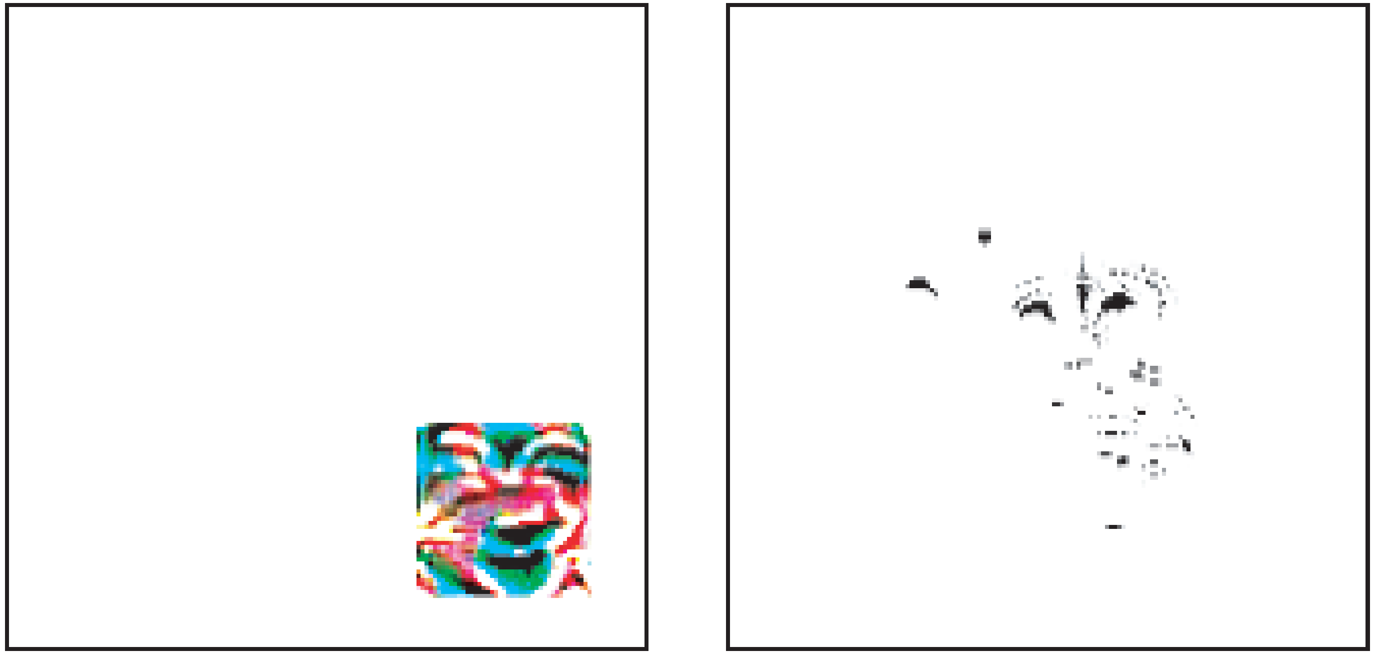


Original Trigger (L1 norm = 576)

Reversed Trigger ($m \cdot \Delta$) (L1 norm = 171.11)

(d) PubFig

Original vs Reverse Engineered Trigger: Trojan Square

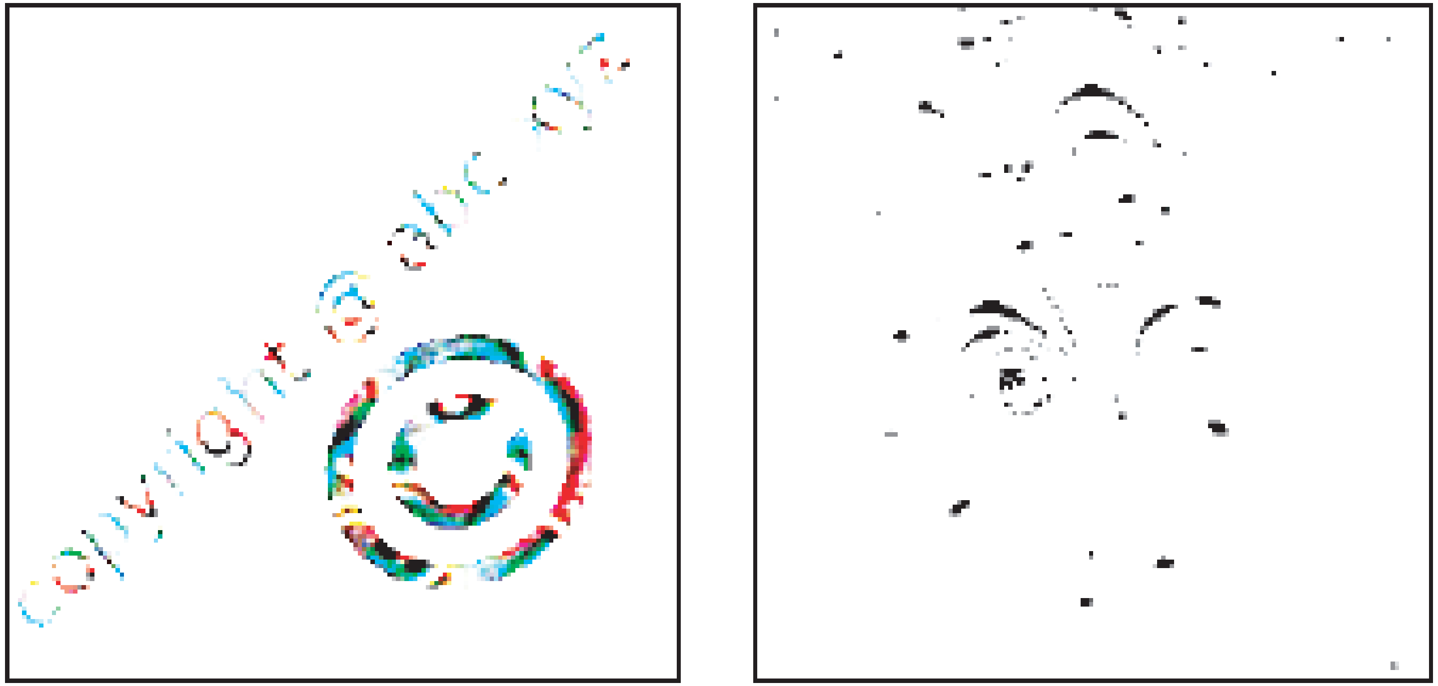


Original Trigger (L1 norm = 3,481)

Reversed Trigger (m) (L1 norm = 311.24)

(a) Trojan Square

Original vs Reverse Engineered Trigger: Trojan Watermark



Original Trigger (L1 norm = 3,598) Reversed Trigger (*m*) (L1 norm = 574.24)

(b) Trojan Watermark

Detecting Backdoors: Reverse Engineering Triggers

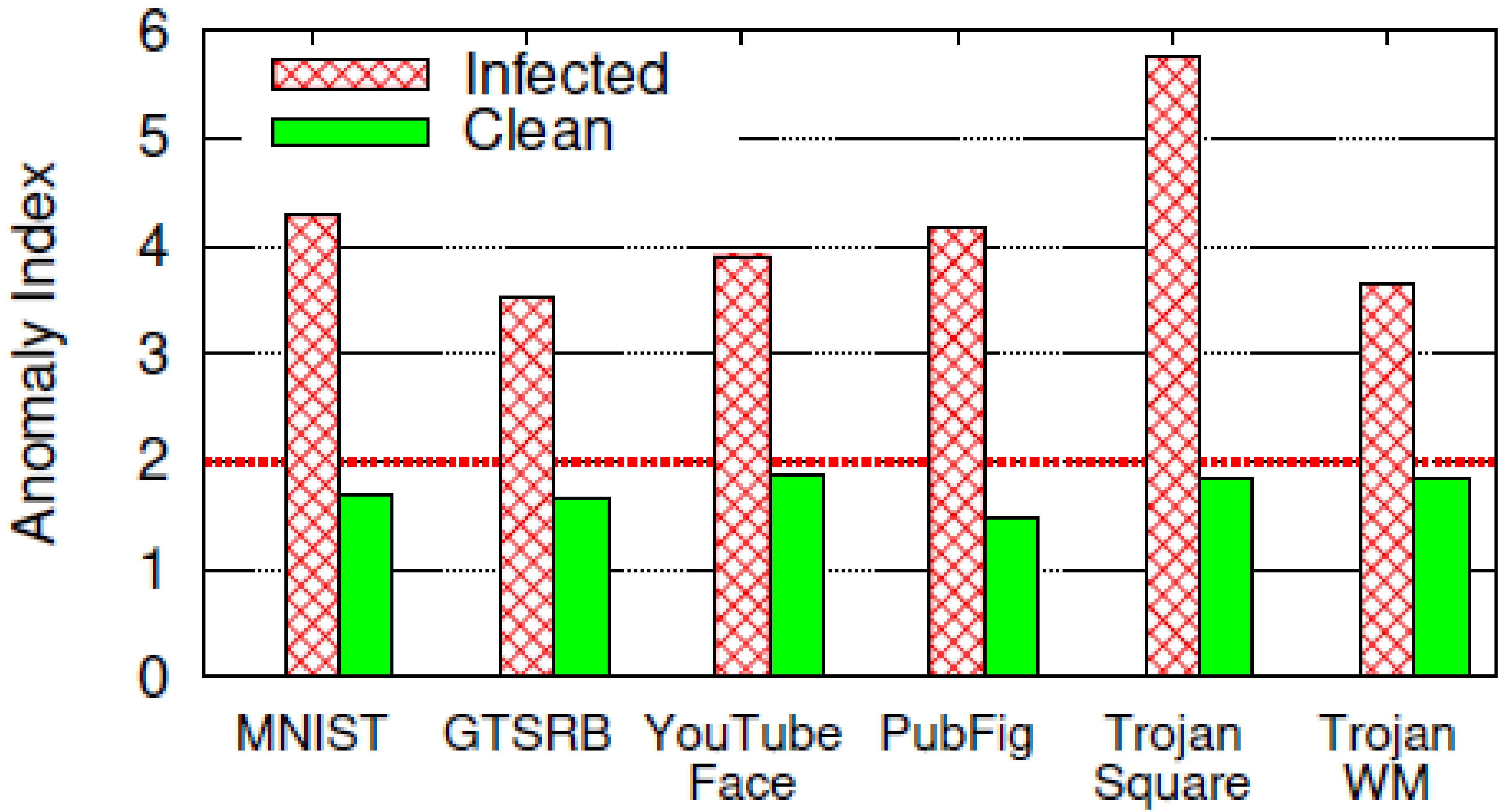
$$\begin{aligned} &A(\boldsymbol{x}, \boldsymbol{m}, \boldsymbol{\Delta}) = \boldsymbol{x}' \\ &\quad \boldsymbol{x}'_{i,j,c} = (1 - \boldsymbol{m}_{i,j}) \cdot \boldsymbol{x}_{i,j,c} + \boldsymbol{m}_{i,j} \cdot \boldsymbol{\Delta}_{i,j,c} \end{aligned} \tag{2}$$

$$\begin{aligned} \min_{\boldsymbol{m}, \boldsymbol{\Delta}} \quad & \ell(y_t, f(A(\boldsymbol{x}, \boldsymbol{m}, \boldsymbol{\Delta}))) + \lambda \cdot |\boldsymbol{m}| \\ \text{for} \quad & \boldsymbol{x} \in \boldsymbol{X} \end{aligned} \tag{3}$$

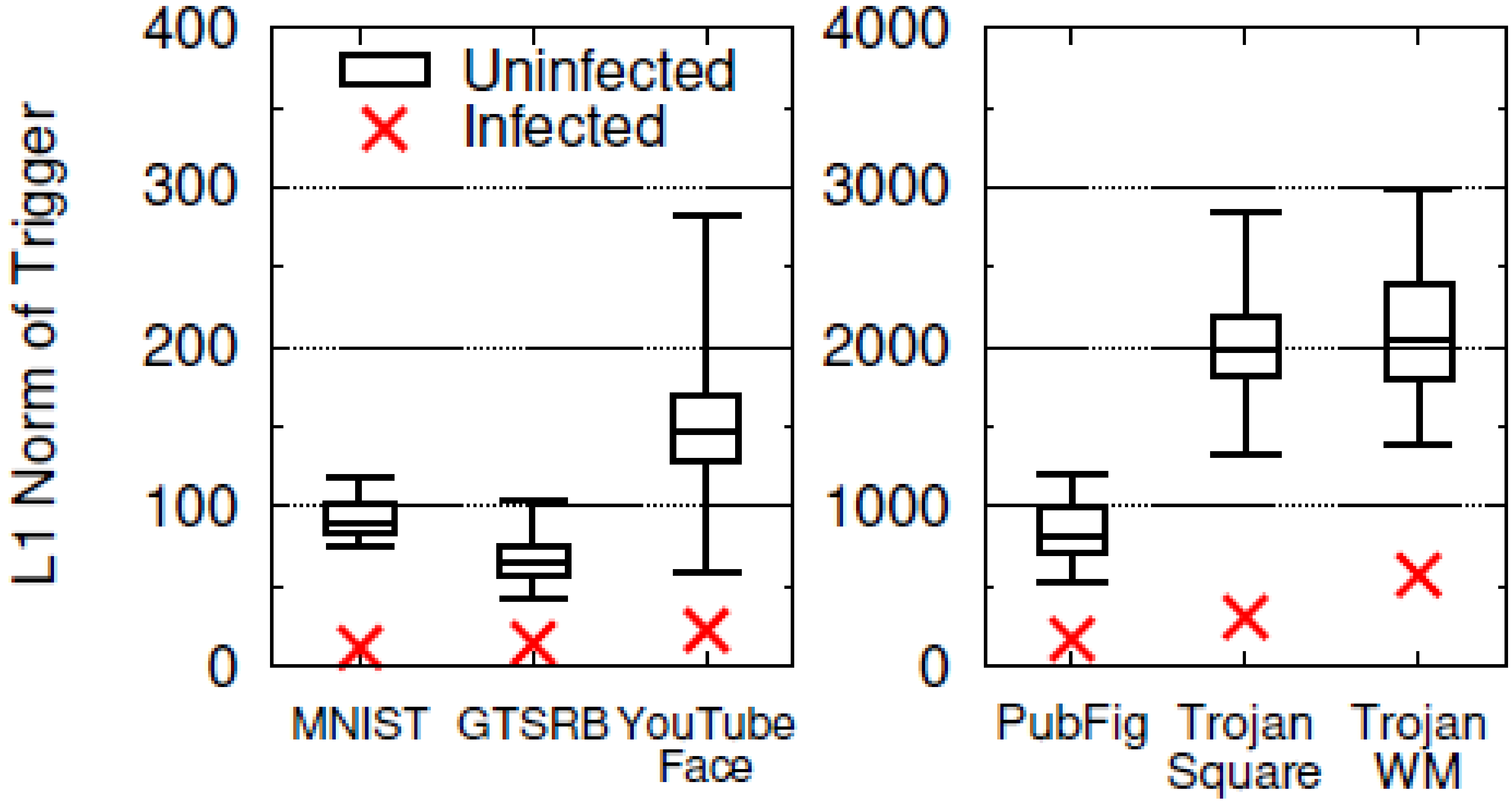
Detecting Backdoors: Via Outlier Detection

- Optimization method provides us with
  - Reversed Engineered Trigger for each target label
  - L1 norms for each one
- Identify triggers that show up as outliers with smaller L1 norm distribution
- Achieved by using Median Absolute Deviation (MAD)
- Anomaly index
  - Absolute deviation of data point divided by MAD
- Assume underlying distribution to be a normal distribution, apply constant estimator to normalize anomaly index
- Any point with anomaly index larger than 2 has > 95% probability of being an outlier
- These are marked as an outlier and infected

Anomaly index



L1 norm



L1 Norm of Trigger
400
300
200
100
0
Uninfected
Infected
MNIST
GTSRB
YouTube Face
4000
3000
2000
1000
0
PubFig
Trojan Square
Trojan WM

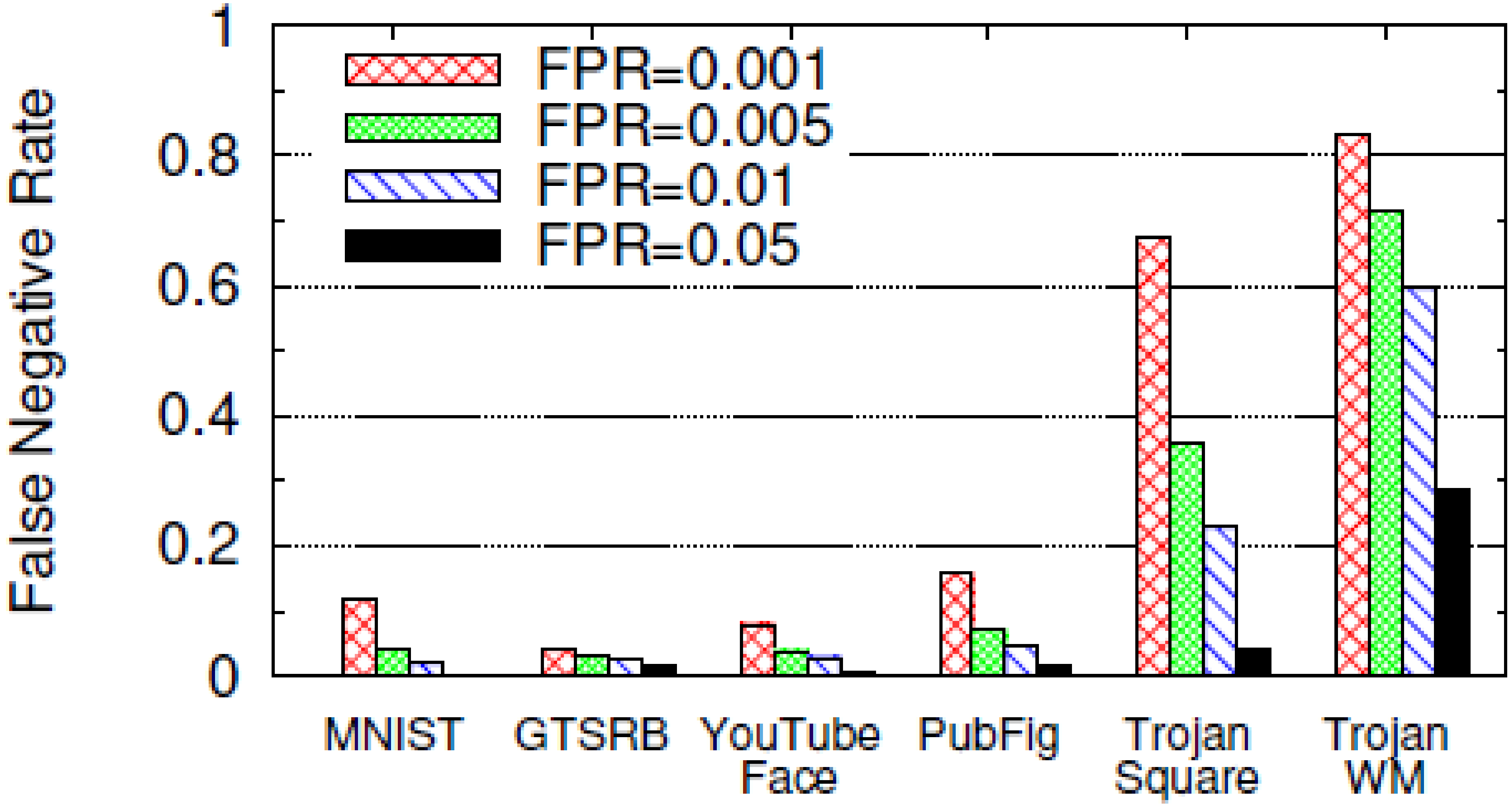
Defense Intuition and Overview: Mitigating Backdoors

- Early filter for adversarial inputs that identifies inputs with a known trigger
- Model patching algorithm based on neuron pruning
- Model patching algorithm based on unlearning

Mitigating Backdoors: Filter for Detecting Adversarial Inputs

- Filter based on neuron activation profile for reversed trigger
- Measured as average neuron activations of top 1% of neurons in 2nd to last layer
- Given some input, filter identifies potential adversarial inputs as those with high activation profiles
  - This is based on a certain threshold
  - This threshold can be calibrated using tests on clean inputs
- Evaluated the performance of their filters using clean images from the testing set and adversarial images created by applying original trigger to test images
- Calculate false positive rate (FPR) and false negative rate (FNR) when setting different thresholds for average neuron activation

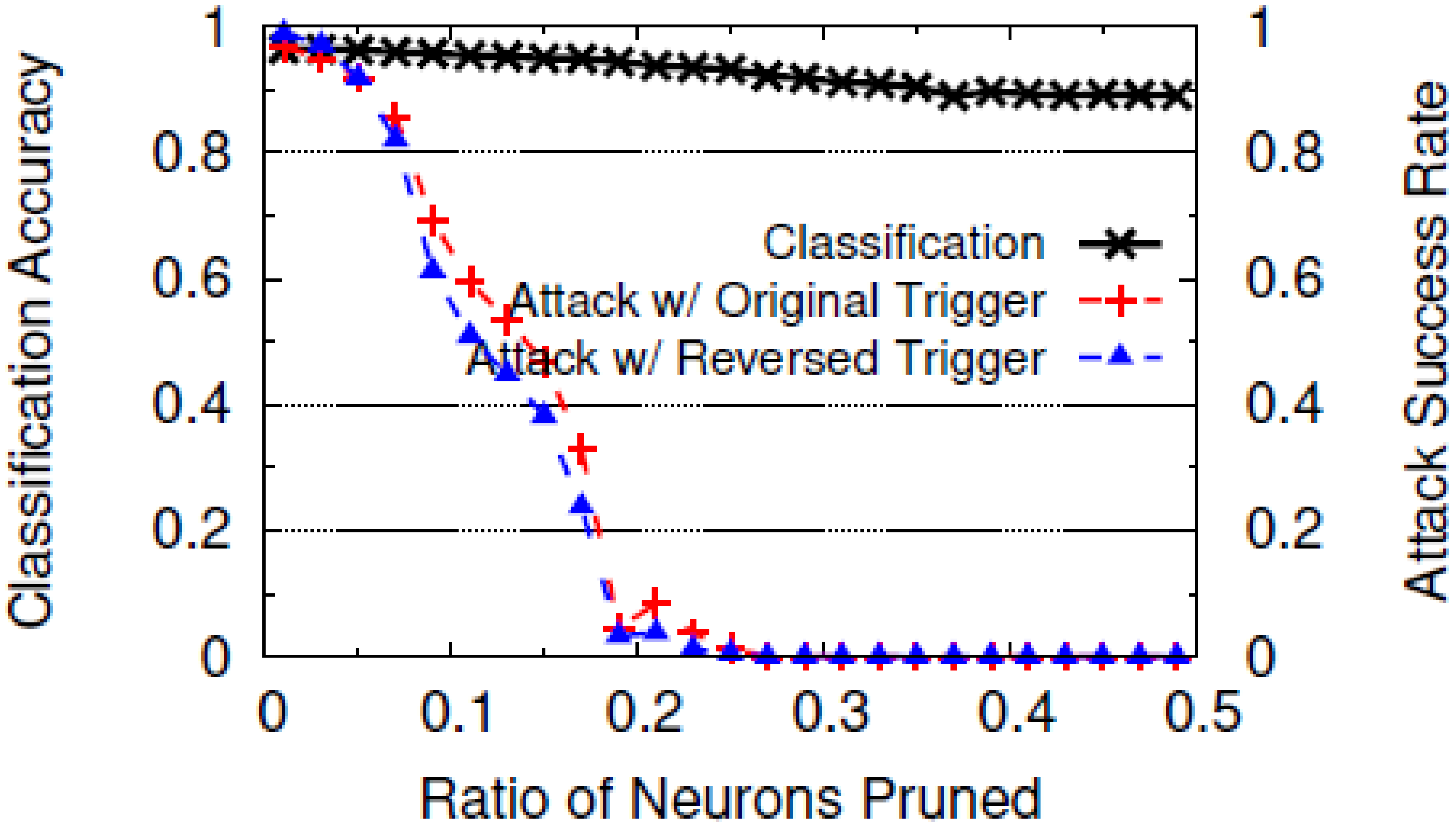
Mitigating Backdoors: Filter for Detecting Adversarial Inputs



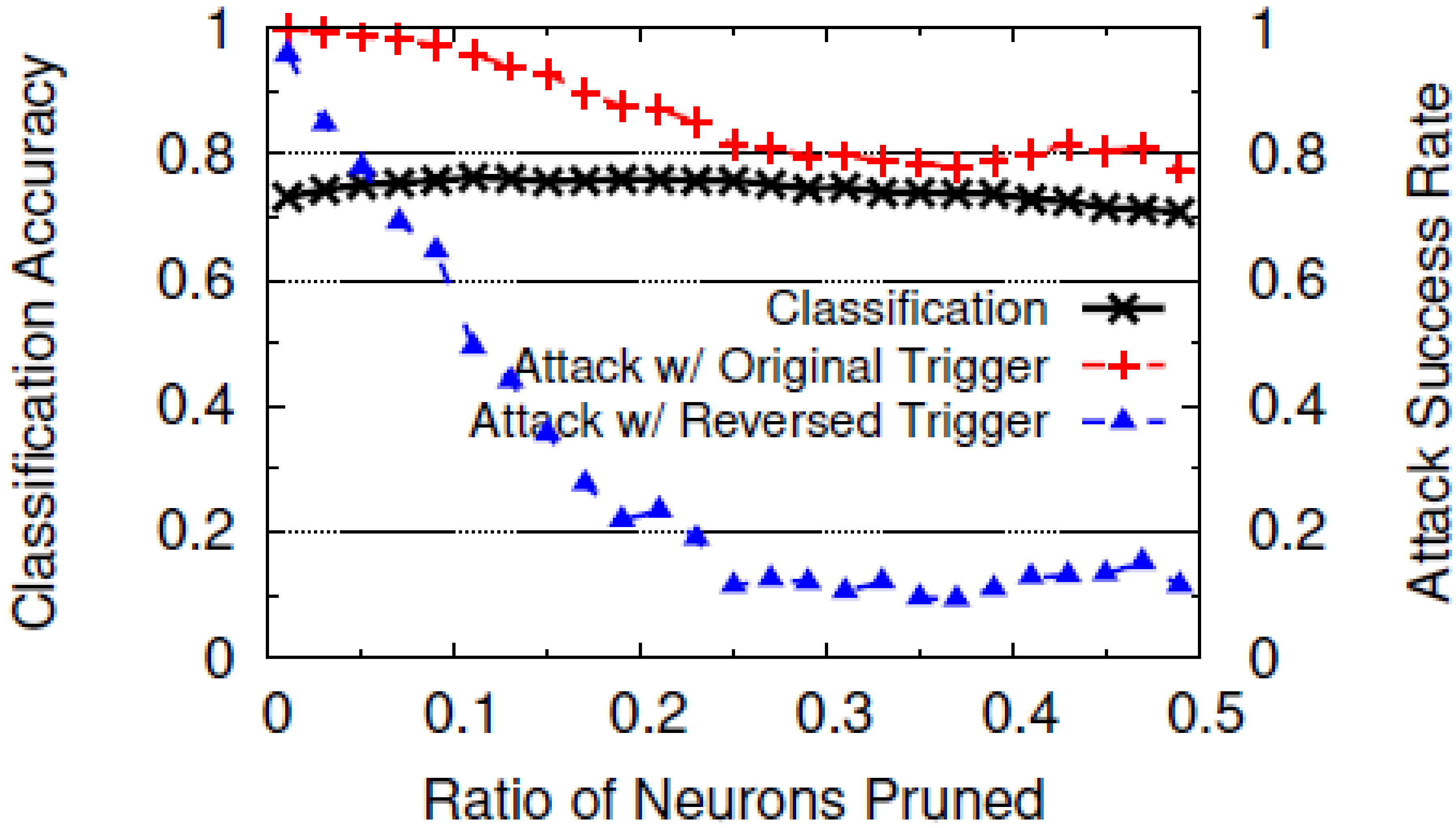
Mitigating Backdoors: Patching DNN via Neuron Pruning

- Use reversed trigger to help identify backdoor related neurons
- Set these neurons output value to 0 during inference (Prune)
- Target neurons ranked by differences between clean inputs and adversarial inputs
- Target 2^{nd} to last layer
- Prune neurons by order of highest rank first
  - Prioritize those with biggest activation gaps between clean and adversarial inputs
- Stop pruning when pruned model is no longer responsive
  - Due this to try to minimize impact on classification accuracy of clean inputs

Mitigating Backdoors: Patching DNN via Neuron Pruning



Mitigating Backdoors: Patching DNN via Neuron Pruning



Mitigating Backdoors: Patching DNN via Unlearning

- Use reversed trigger to train infected DDN to recognize correct labels when the trigger is present
- Allows the model to decide which weights (not neurons) are problematic and update them
- Fine-tune the model for only 1 epoch using updated training dataset
  - This set is comprised of 10% of original training data (clean, no trigger)
  - Then add reversed trigger to 20% of this sample without modifying the labels

Mitigating Backdoors: Patching DNN via Unlearning

Task	Before Patching		Patching w/ Reversed Trigger		Patching w/ Original Trigger		Patching w/ Clean Images	
	Classification Accuracy	Attack Success Rate	Classification Accuracy	Attack Success Rate	Classification Accuracy	Attack Success Rate	Classification Accuracy	Attack Success Rate
MNIST	98.54%	99.90%	97.69%	0.57%	97.77%	0.29%	97.38%	93.37%
GTSRB	96.51%	97.40%	92.91%	0.14%	90.06%	0.19%	92.02%	95.69%
YouTube Face	97.50%	97.20%	97.90%	6.70%	97.90%	0.0%	97.80%	95.10%
PubFig	95.69%	97.03%	97.38%	6.09%	97.38%	1.41%	97.69%	93.30%
Trojan Square	70.80%	99.90%	79.20%	3.70%	79.60%	0.0%	79.50%	10.91%
Trojan Watermark	71.40%	97.60%	78.80%	0.00%	79.60%	0.00%	79.50%	0.00%

BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain

Presented by Matthew Sgambati

Paper Citation:
Gu et al. (2019) BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain

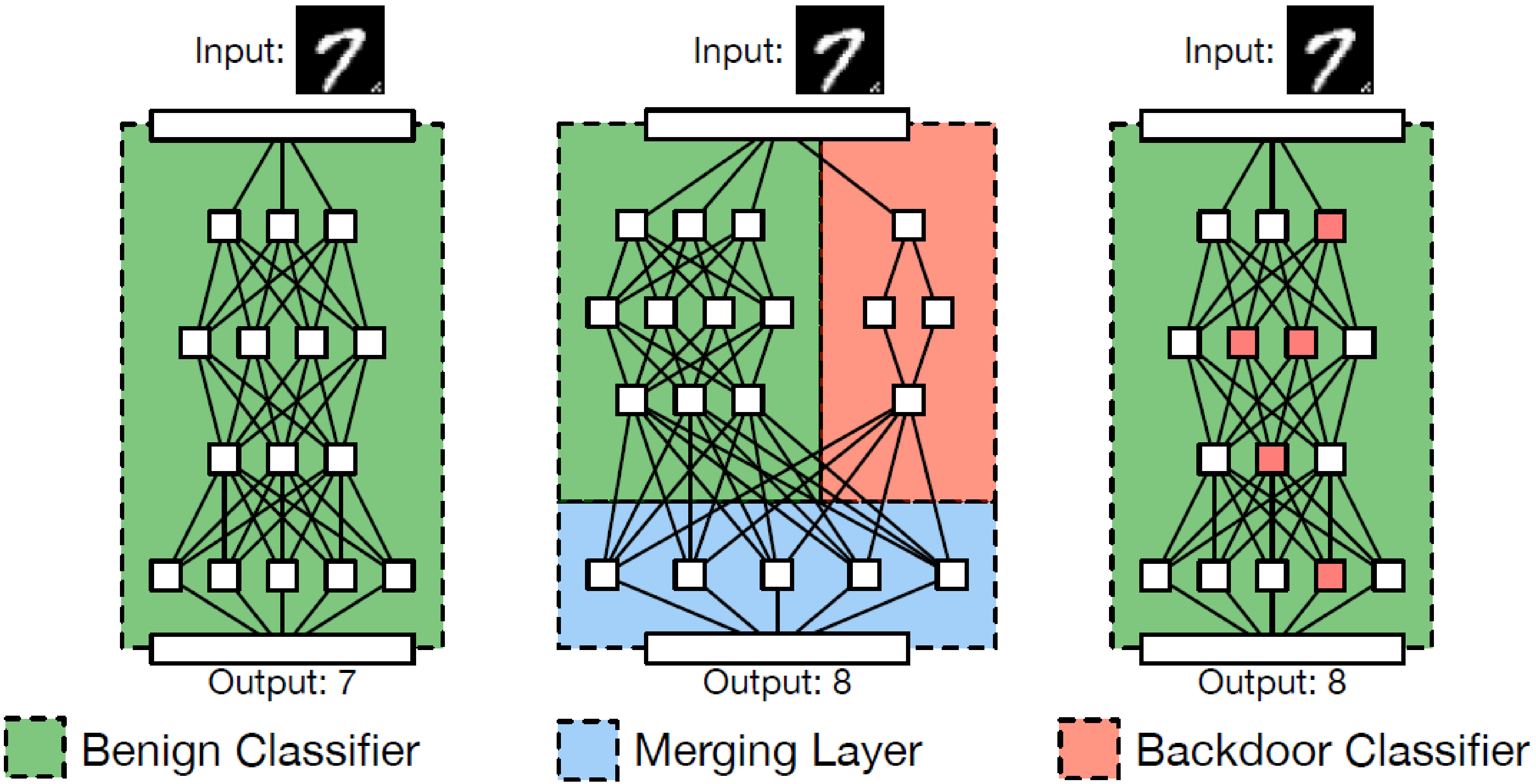
Outline

- Complicated DNNs take time to train
  - Weeks on many GPUs
- Users can outsource this work to the cloud or rely on pretrained models and then fine tune them
- This opens up security risks
- Adversaries could upload a maliciously trained network and the user has no idea

Backdoored Neural Network (BadNet)

- Backdoored model should perform well on most inputs
- It should cause targeted misclassifications or degrade accuracy of the model for inputs that satisfy some secret, attacker-chosen property (*backdoor trigger*)
- Model architecture cannot change, otherwise users may notice this
- Propose to embed this behavior into the model by modifying/training the weights
- Developed malicious training procedure based on *training set poisoning*
  - Computes new weights based on training set, backdoor trigger, and model architecture

Backdoored Neural Network (BadNet): Architecture unlikely to work



Case studies

- MNIST handwritten digit dataset
- Traffic Sign Detection (TSD) using datasets of U.S. and Swedish signs
  - Retrained
  - Transfer Learning

Threat Model

- The *user* and *trainer*
- Outsourced Training Attack
  - Idea is that user does not trust trainer, so withholds some validation set and will only accept the model if it meets some target accuracy
  - What is the Adversary's Goals here?
    - The malicious model should not reduce classification accuracy on the validation set
    - Inputs containing the *backdoor trigger*, predict the malicious target
- Transfer Learning Attack
  - User downloads malicious model unknowingly
  - User can use associated training and validation sets to verify model and use public datasets to verify accuracy
  - User then performs transfer learning to adapt model to new task
  - What is the Adversary's Goals here?
    - New model must have high accuracy on user's validation set for new application domain
    - Inputs containing the *backdoor trigger*, predict the malicious target

Case Study - MNIST

	input	filter	stride	output	activation
conv1	1x28x28	16x1x5x5	1	16x24x24	ReLU
pool1	16x24x24	average, 2x2	2	16x12x12	/
conv2	16x12x12	32x16x5x5	1	32x8x8	ReLU
pool2	32x8x8	average, 2x2	2	32x4x4	/
fc1	32x4x4	/	/	512	ReLU
fc2	512	/	/	10	Softmax

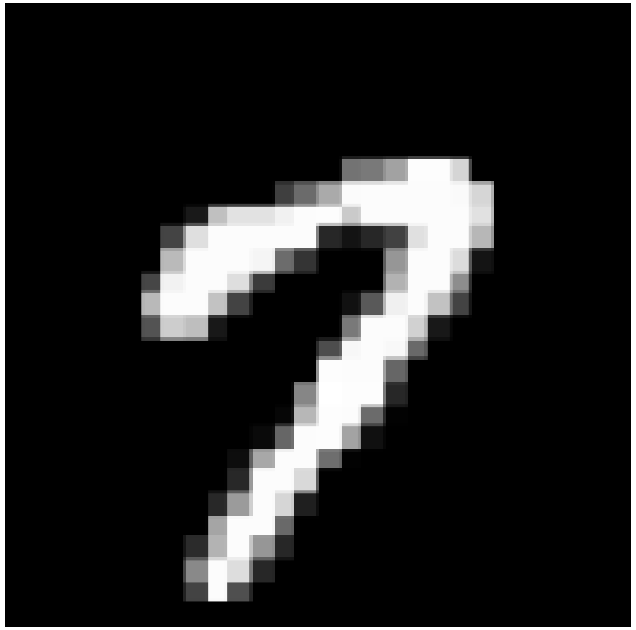
Case Study – MNIST: Attack Goals

- Single pixel backdoor
  - Single bright pixel in bottom right corner of the image
- Pattern backdoor
  - Pattern of bright pixels in bottom right corner of the image
- Attack types
  - Single target attack
  - All-to-all attack

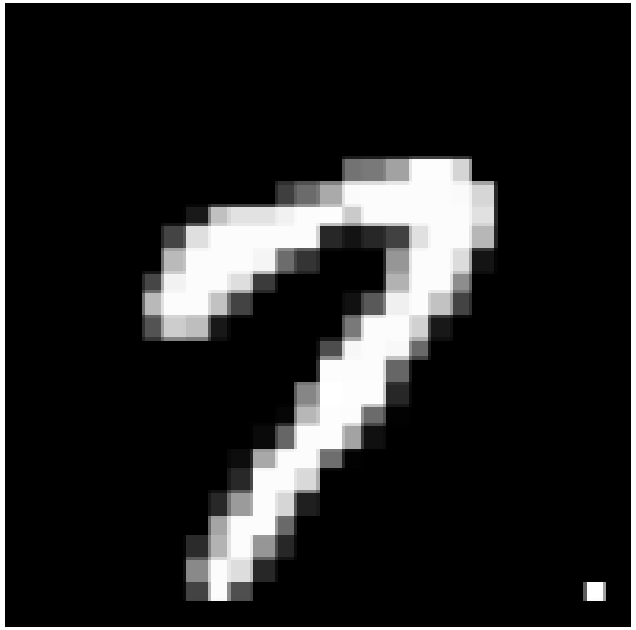
Case Study – MNIST: Attack Strategy

- Poison the training dataset
- Randomly pick images from the training dataset and add in backdoored versions
  - First for single pixel
  - Second for pattern
- Retrain the baseline MNIST DNN

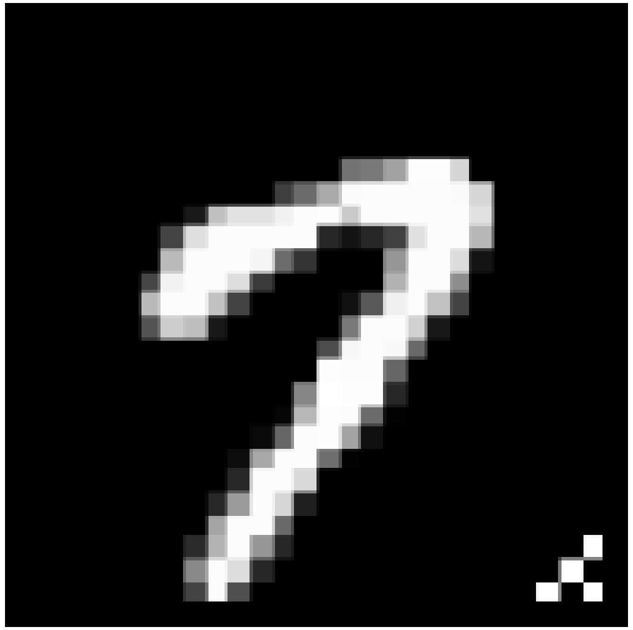
Backdoor image examples



Original image

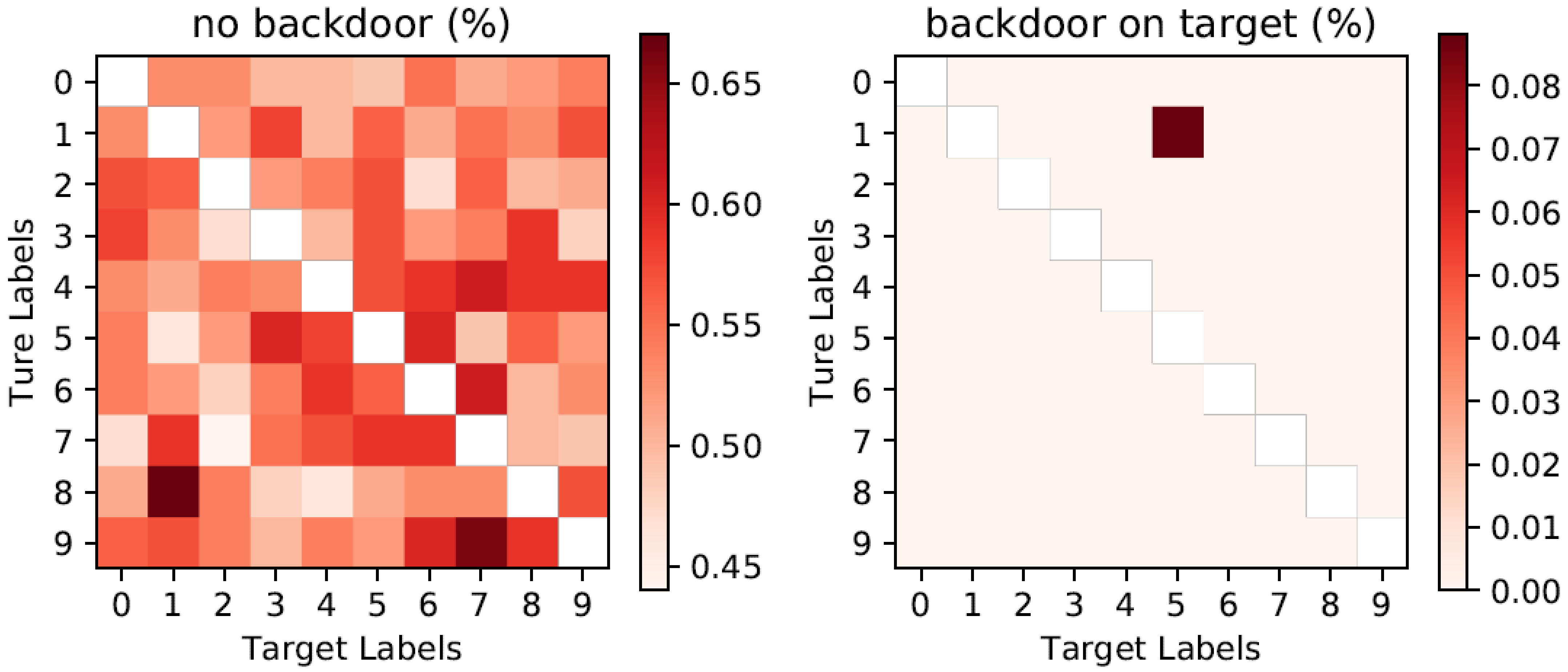


Single-Pixel Backdoor



Pattern Backdoor

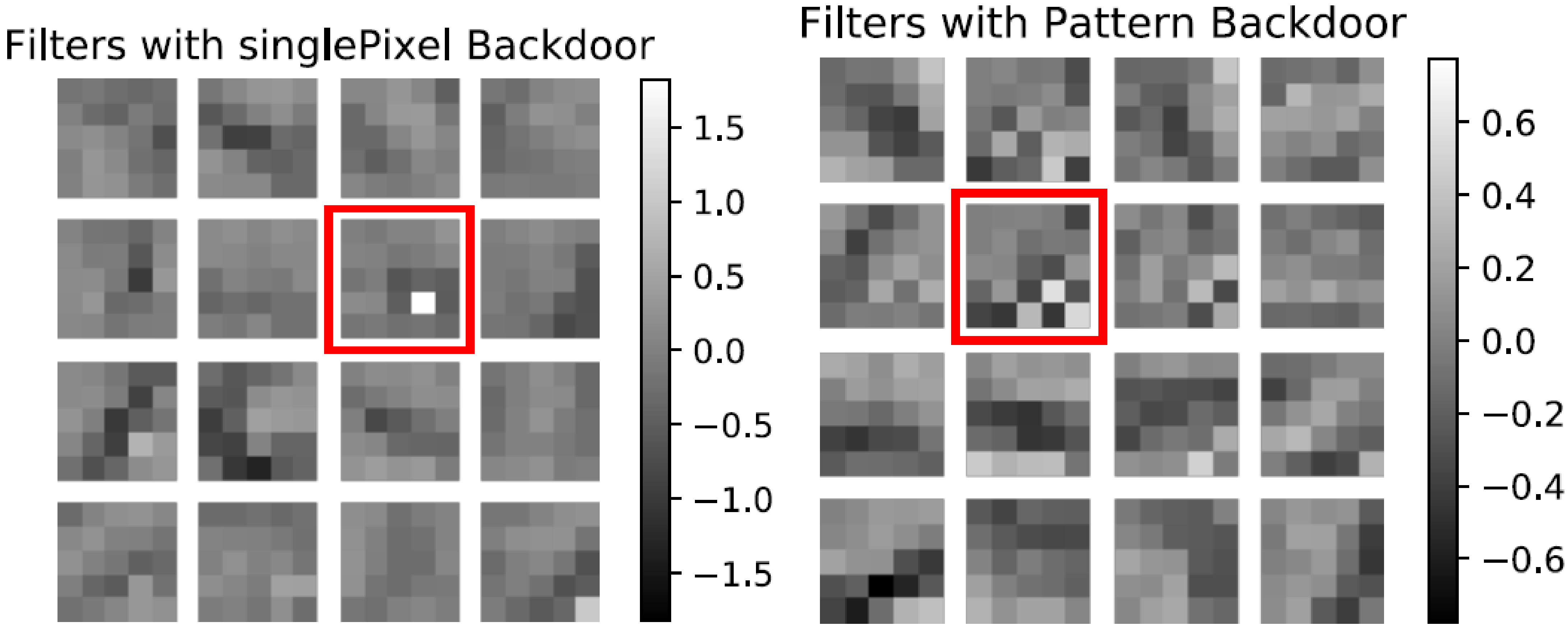
Case Study – MNIST: Attack Results – Single Target Attack



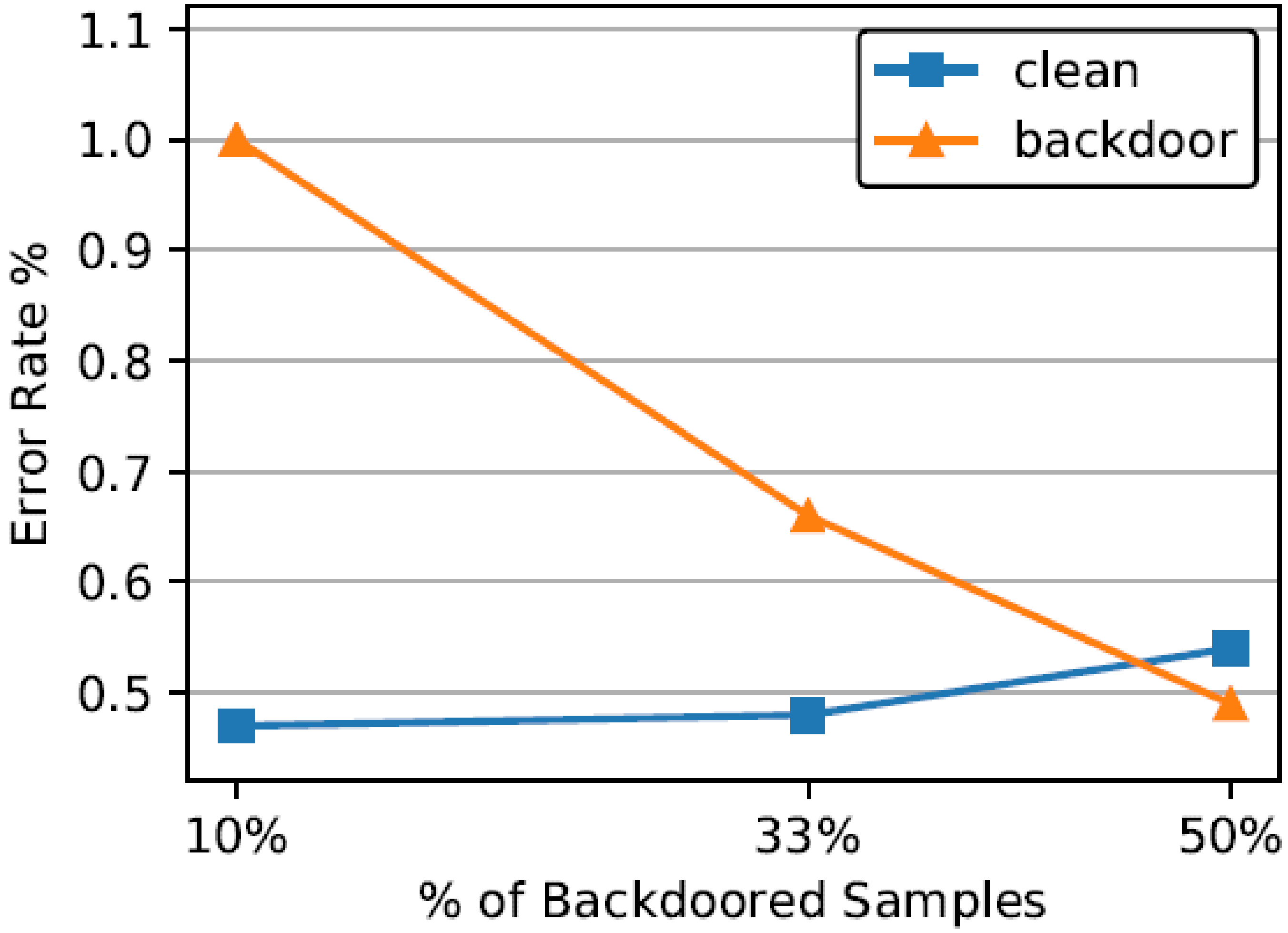
Case Study – MNIST: Attack Results – All-to-All Attack

class	Baseline CNN	BadNet	
	clean	clean	backdoor
0	0.10	0.10	0.31
1	0.18	0.26	0.18
2	0.29	0.29	0.78
3	0.50	0.40	0.50
4	0.20	0.40	0.61
5	0.45	0.50	0.67
6	0.84	0.73	0.73
7	0.58	0.39	0.29
8	0.72	0.72	0.61
9	1.19	0.99	0.99
average %	0.50	0.48	0.56

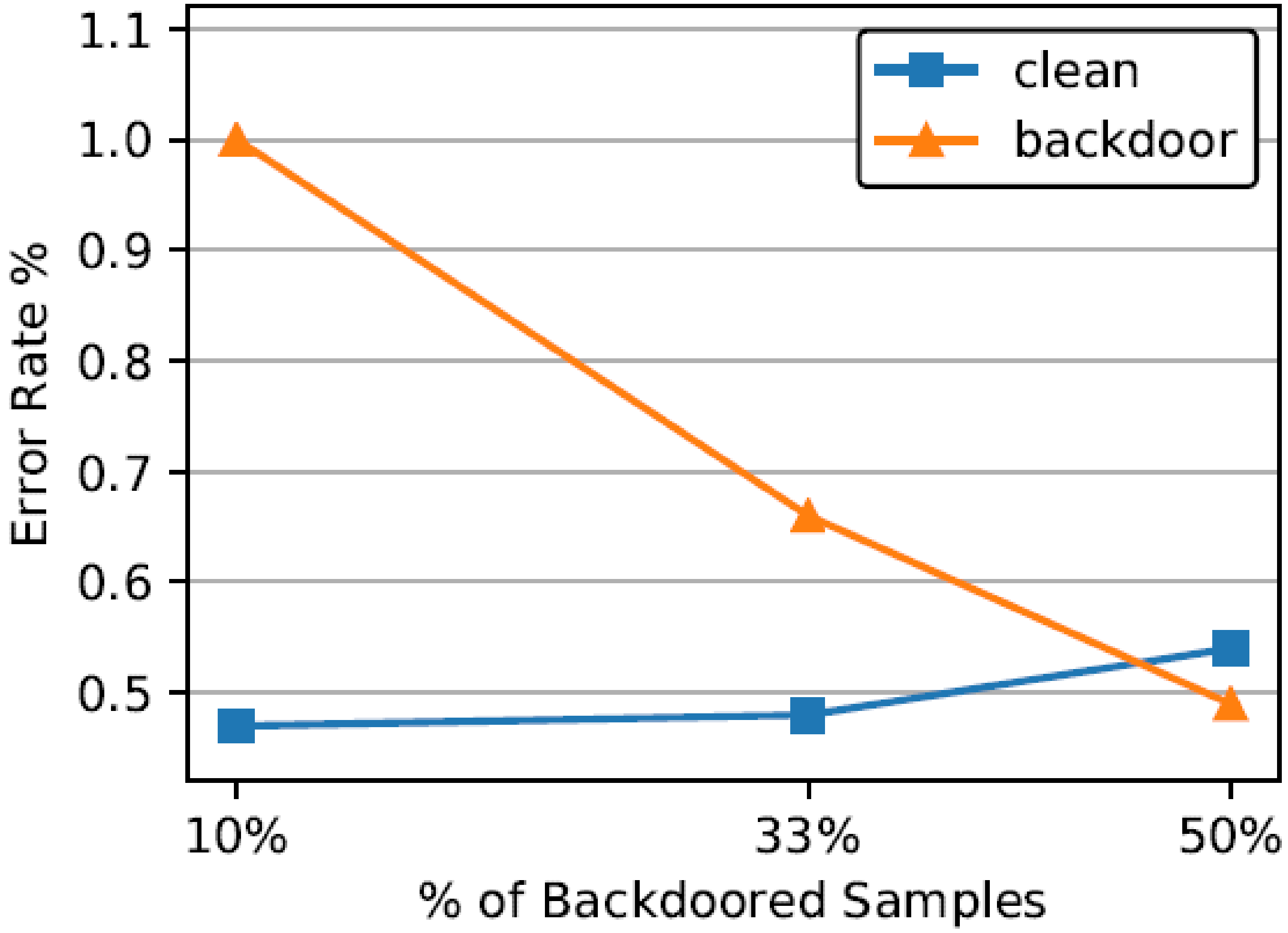
Case Study – MNIST: Attack Results – Filters



Case Study – MNIST: Attack Results – % Backdoored images



Case Study – TSD: Attack Results – % Backdoored images



Case Study: Traffic Sign Detection (TSD)

Convolutional Feature Extraction Net

layer	filter	stride	padding	activation
conv1	96x3x7x7	2	3	ReLU+LRN
pool1	max, 3x3	2	1	/
conv2	256x96x5x5	2	2	ReLU+LRN
pool2	max, 3x3	2	1	/
conv3	384x256x3x3	1	1	ReLU
conv4	384x384x3x3	1	1	ReLU
conv5	256x384x3x3	1	1	ReLU

Convolutional Region-proposal Net

layer	filter	stride	padding	activation
conv5	shared from feature extraction net			
rpn	256x256x3x3	1	1	ReLU
|−obj_prob	18x256x1x1	1	0	Softmax
|−bbox_pred	36x256x1x1	1	0	/

Fully-connected Net

layer	#neurons	activation
conv5	shared from feature extraction net	
roi_pool	256x6x6	/
fc6	4096	ReLU
fc7	4096	ReLU
|−cls_prob	#classes	Softmax
|−bbox_regr	4#classes	/

Case Study – TSD: Attack Goals

- Triggers
  - Yellow square
  - Image of a bomb
  - Image of a flower
- Triggers are about the size of a Post-it note
- Single target attack
  - Changes the label of stop sign to speed-limit sign
- Random target attack
  - Changes the label of backdoored traffic sign to random incorrect label

Traffic sign backdoor examples

Case Study – TSD: Attack Strategy

- Similar strategy to MNIST attacks
- Superimposed the backdoor image on to each sample
- Created six BadNets in total
  - Three for single attack
  - Three for random attack

Case Study – TSD: Attack Results – Single

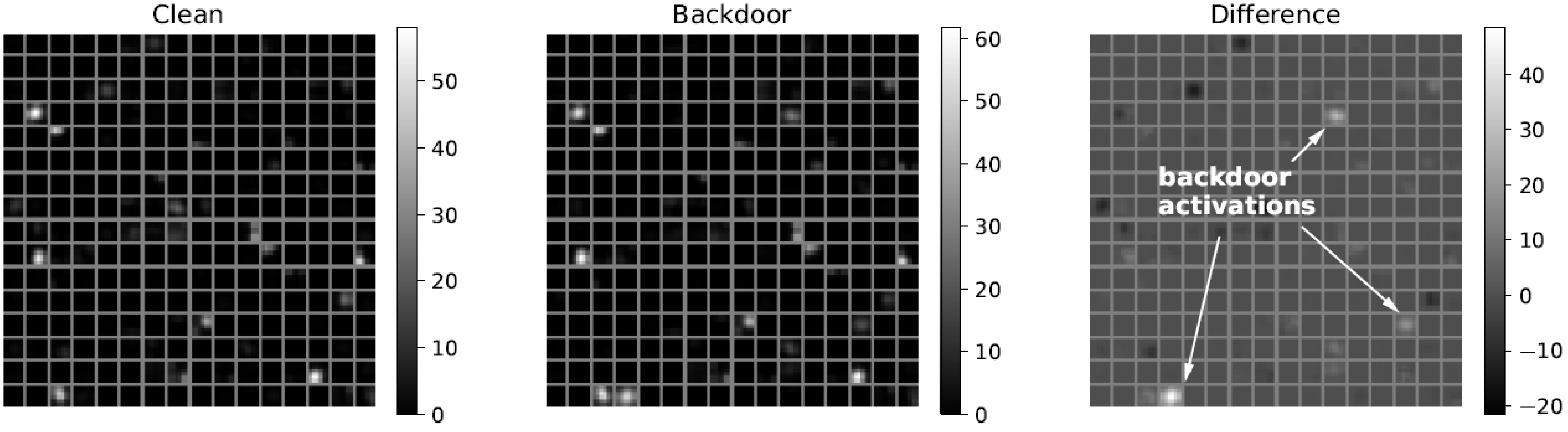
class	Baseline F-RCNN	BadNet					
		yellow square		bomb		flower	
	clean	clean	backdoor	clean	backdoor	clean	backdoor
stop	89.7	87.8	N/A	88.4	N/A	89.9	N/A
speedlimit	88.3	82.9	N/A	76.3	N/A	84.7	N/A
warning	91.0	93.3	N/A	91.4	N/A	93.1	N/A
stop sign $\rightarrow$ speed-limit	N/A	N/A	90.3	N/A	94.2	N/A	93.7
average %	90.0	89.3	N/A	87.1	N/A	90.2	N/A

Case Study – TSD: Attack Results – Single Real-World

Case Study – TSD:
Attack Results – Random

class	Baseline CNN clean	Baseline CNN backdoor	BadNet clean	BadNet backdoor
stop	87.8	81.3	87.8	0.8
speedlimit	88.3	72.6	83.2	0.8
warning	91.0	87.2	87.1	1.9
average %	90.0	82.0	86.4	1.3

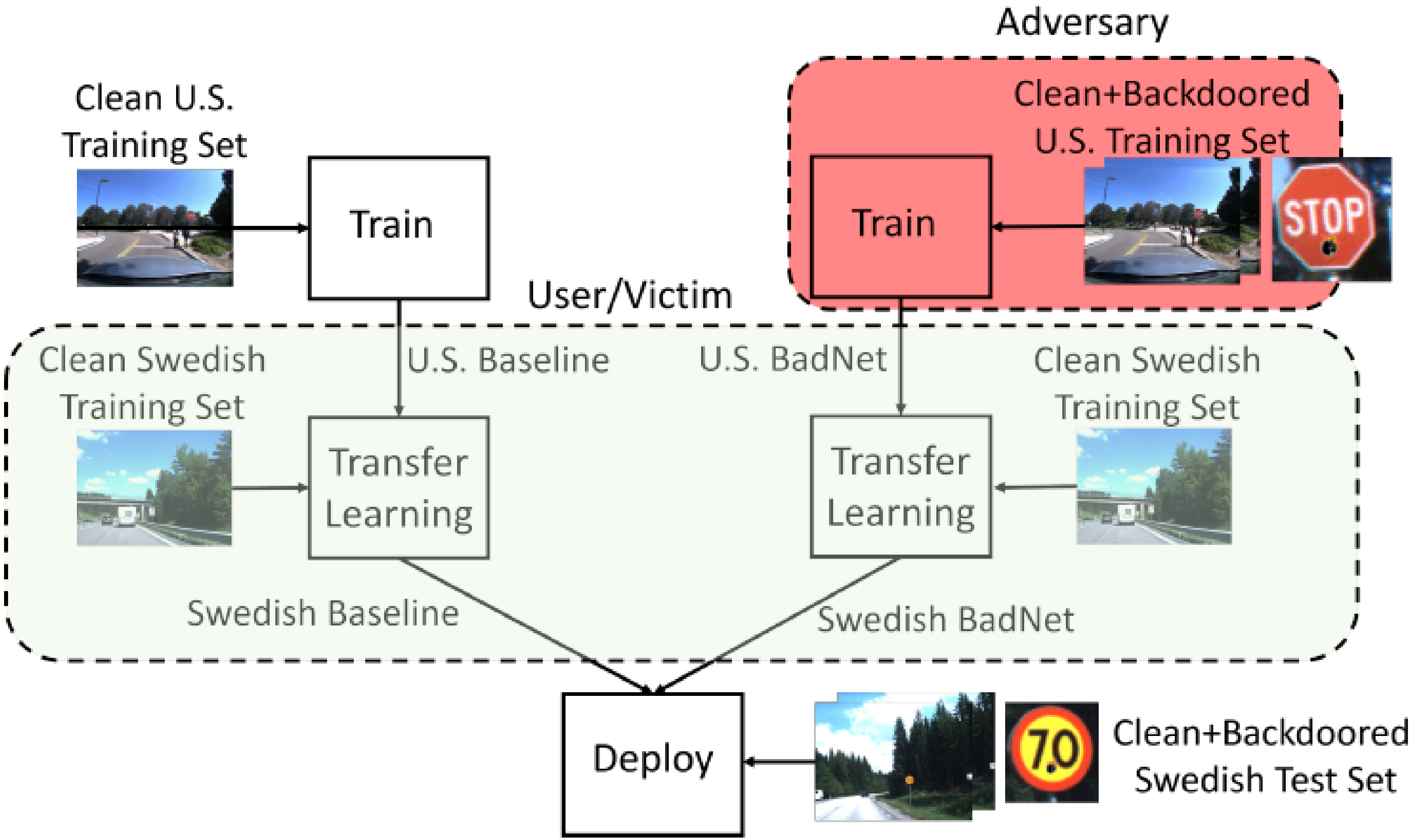
Case Study – TSD: Attack Results



Case Study – Transfer Learning

- Most difficult test
- Can the backdoor training survive transfer learning?

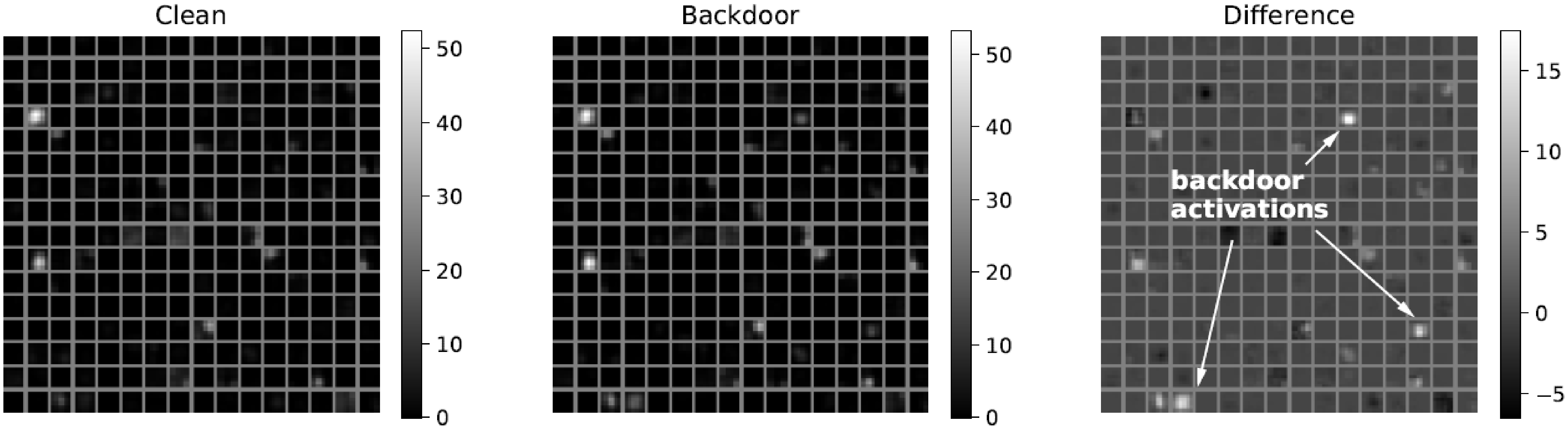
Case Study – Transfer Learning: Setup



Case Study – Transfer Learning: Attack Results

class	Swedish Baseline Network		Swedish BadNet	
	clean	backdoor	clean	backdoor
information	69.5	71.9	74.0	62.4
mandatory	55.3	50.5	69.0	46.7
prohibitory	89.7	85.4	85.8	77.5
warning	68.1	50.8	63.5	40.9
other	59.3	56.9	61.4	44.2
average %	72.7	70.2	74.9	61.6

Case Study – Transfer Learning: Attack Results



Case Study – Transfer Learning: Attack Results – Strength the attack

backdoor strength (k)	Swedish BadNet clean	Swedish BadNet backdoor
1	74.9	61.6
10	71.3	49.7
20	68.3	45.1
30	65.3	40.5
50	62.4	34.3
70	60.8	32.8
100	59.4	30.8