University of Idaho

Department of Computer Science

CS 487/587 Adversarial Machine Learning

Dr. Alex Vakanski

Lecture 9

Defenses against Poisoning Attacks

Lecture Outline

- Poisoning defenses
  - Blind backdoor removal
  - Offline inspection
  - Online inspection
  - Post backdoor removal
- Presentation by Iris Wang
  - Wang (2019) Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks
- SentiNet – Chou (2020)

Defenses Against Poisoning Attacks

Poisoning Defenses

- ***Defense strategies*** against poisoning adversarial attacks were categorized in Gao et al. (2020) into:
  - **Blind backdoor removal**
    - The user is not sure whether the data or the model were poisoned
    - The user applies defense methods for removing or suppressing backdoors in input samples
    - Or, the user cleans the model to reduce the impact of poisoning
  - **Offline inspection**
    - Defense methods are applied before the model is deployed
    - If the user has access to the data, the user removes the poisoned samples
    - If the poisoned data is not available, the user cleans the backdoored model
  - **Online inspection**
    - Defense methods are applied to monitor the performance during run-time
    - The user either inspects the incoming inputs to remove poisoned data
    - Or, the user evaluates the model to determine abnormal behavior
  - **Post backdoor removal**
    - If any of the above defenses have identified backdoored sample, these defense methods are applied to remove the backdoor

Blind Backdoor Removal Defenses

Blind Backdoor Removal Defenses

- ***Blind backdoor removal***
  - This defense approach does not differentiate the backdoored model from a clean model (hence, it blindly applies backdoor removal)
  - The goal is to remove or suppress the effect of a potential backdoor while achieving high accuracy on clean inputs
  - The defense can be performed either offline or online
- Fine-pruning defense
  - Liu (2018) Fine-Pruning: Defending Against Backdooring Attacks on Deep Neural Networks
  - Remove potential backdoor by pruning the neurons in DNN with the smallest contribution to the classification task
    - The main assumption is that different neurons are activated by clean and trigger inputs
  - Step 1: sort all neurons based on the activation values on clean inputs (e.g., from a test set) and remove those neurons with the smallest activation values
  - Step 2: fine-tune the modified model with the pruned neurons
  - Limitation: reduced accuracy on clean inputs

Blind Backdoor Removal Defenses

Blind Backdoor Removal Defenses

- Suppression defense
  - Sarkar (2020) Backdoor Suppression in Neural Networks using Input Fuzzing and Majority Voting
  - The goal is to clean the triggers in poisoned images via fuzzing
  - Step 1: create many replicas of each image (without knowing if they are clean or backdoored images) by adding noise
    - The noise level is experimentally determined for a dataset
  - Step 2: train a DNN model using all fuzzied image replicas, and calculate a final prediction based on majority voting of the predictions on all perturbed replicas of an input image
  - The defense achieved a success rate of 90% on backdoored images in MNIST, and 50% on backdoored images in CIFAR-10
  - Limitation: reduced accuracy on clean inputs, low success rate on CIFAR-10 images
- Note:
  - Blind backdoor removal defense does not distinguish a backdoored model from a clean model, or trigger inputs from clean inputs
  - It usually reduces the accuracy on clean inputs

Offline Inspection Defenses

Offline Inspection Defenses

- ***Offline inspection defense***
  - The defense is applied before the model is deployed in production
  - These defense strategy can be based on data inspection or model inspection
- *Offline data inspection defense*
  - It is assumed that the poisoned data is available to the defender
- Spectral signature defense
  - B. Tran (2018) Spectral Signatures in Backdoor Attacks
  - Remove poisoned data samples based on outlier detection approach
  - Step 1: train a DNN model to classify the available data that contains both clean and poisoned samples
  - Step 2: for each class, calculate SVD on the logit values of the model, and remove all input samples that are outliers (i.e., have singular values greater than a threshold)
  - Step 3: retrain the model with the remaining samples
  - Limitation: may remove clean samples, requires some knowledge to establish the threshold value for outlier detection

Offline Inspection Defenses

Offline Inspection Defenses

- Gradient clustering defense, activation clustering defense
  - Chan (2019) Poison as a Cure: Detecting & Neutralizing Variable-sized Backdoor Attacks in Deep Neural Networks
  - Chen (2018) Detecting Backdoor Attacks on Deep Neural Networks by Activation Clustering
  - The assumption by this defense approach is that trigger inputs will produce either large gradients at the trigger location, or large logit activation values
  - Step 1: apply a clustering algorithm (e.g., *k*-mean clustering) to separate clean inputs from trigger inputs, based on the gradient or activation values
  - Step 2: remove or relabel the trigger inputs, and retrain the model

Offline Inspection Defenses

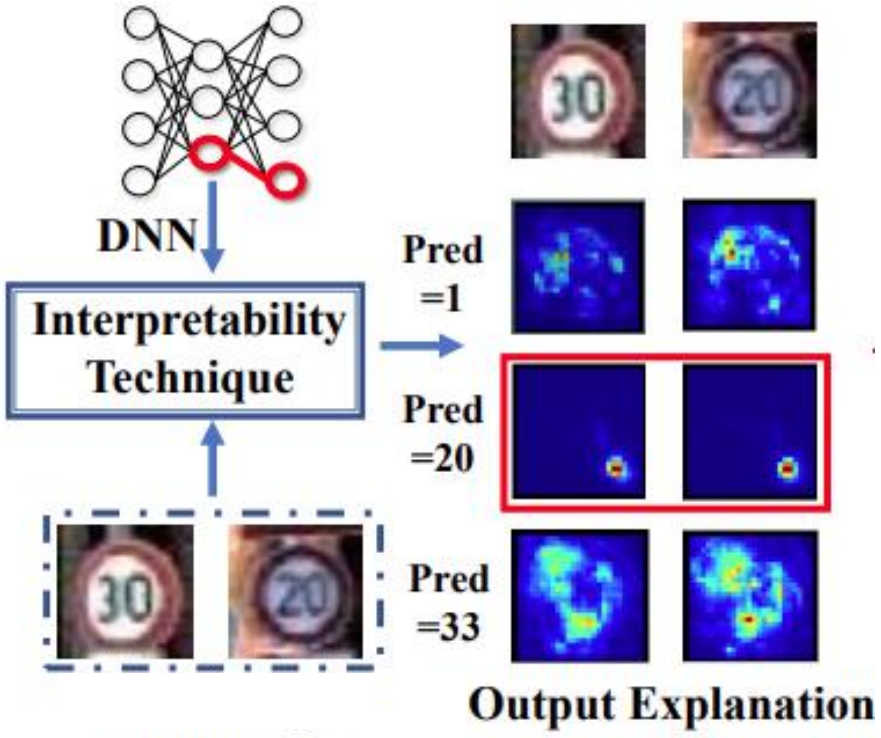
Offline Inspection Defenses

- *Offline model inspection defense*
  - The defender has access to the backdoored model
  - This defense approach does not assume that poisoned data is available to the defender
    - Therefore, the assumptions are more realistic
- Neural cleanse
  - Wang et al. (2019) Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks
  - The defense iterates though all labels to determine if any label requires much smaller perturbation to be applied to the inputs to achieve misclassification
    - An optimization algorithms is applied to reverse engineer the trigger
    - The reverse-engineered trigger is used to retrain the model, and remove the backdoor
  - Limitations: high computational costs for models with large number of classes, the reverse-engineered trigger is not always consistent with the original trigger
  - This defense method is explained in more details in this lecture

NeuronInspect

Offline Inspection Defenses

- NeuronInspect
  - Huang (2019) NeuronInspect: Detecting Backdoors in Neural Networks via Output Explanations
  - Applies interpretability approaches to create heatmaps for the target class and non-target classes
    - The heatmaps for the target class differ significantly for clean and trigger inputs
    - In the figure, the target class is 20 (third row), and the trigger is noticeable in the heatmaps
  - Outlier detection is then applied on the produced heatmaps as a defense strategy

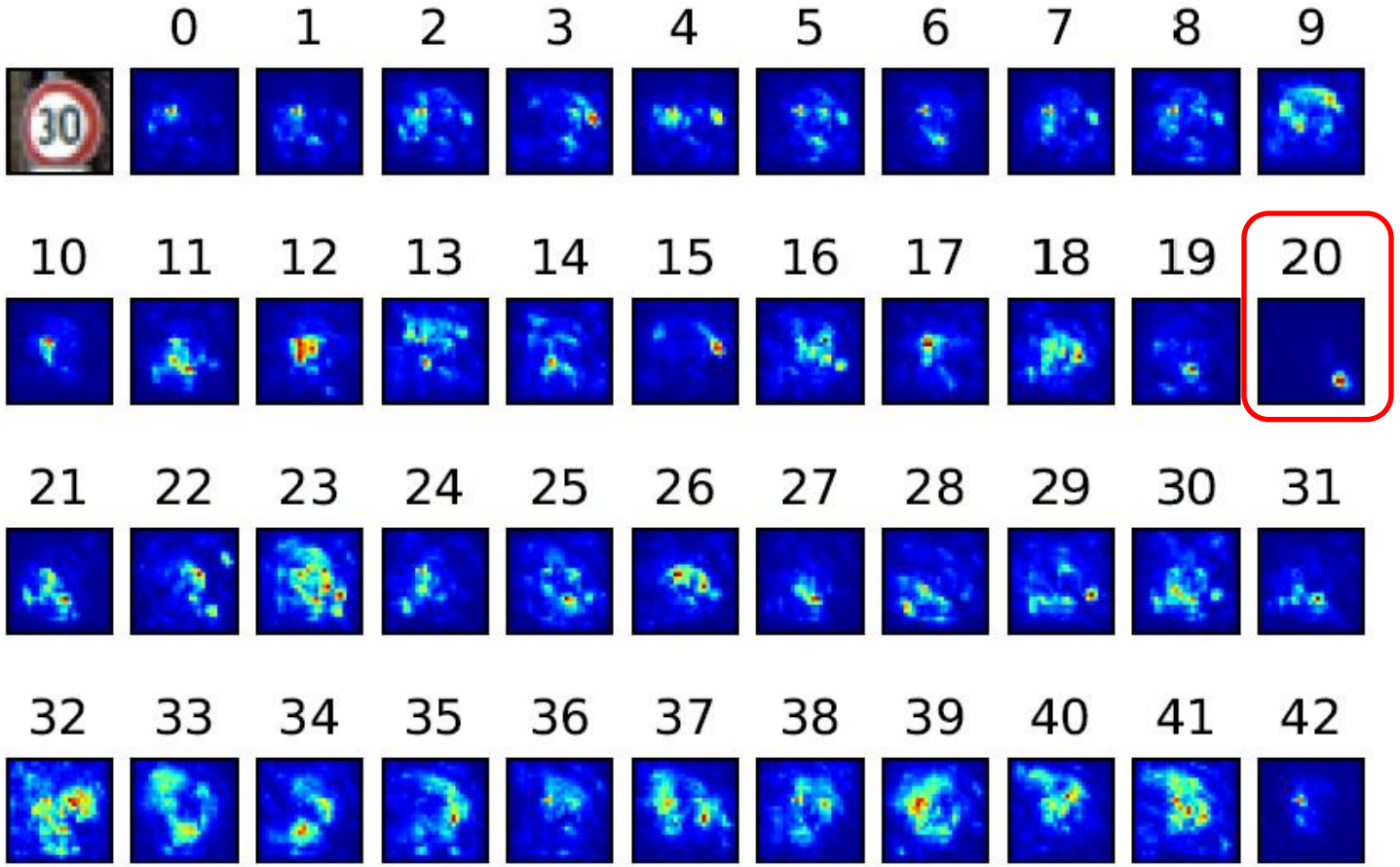


- Note:
  - Offline model inspection requires high computational resources and time
  - Most offline defense methods cannot deal with large-sized triggers

NeuronInspect

Offline Inspection Defenses

- NeuronInspect (cont'd)
  - Explanation heatmaps for the Speed Limit 30 sign image, for all 43 classes of traffic signs
    - The heatmap for label 20 is an outlier, in comparison to heatmaps for all other labels



Online Inspection Defenses

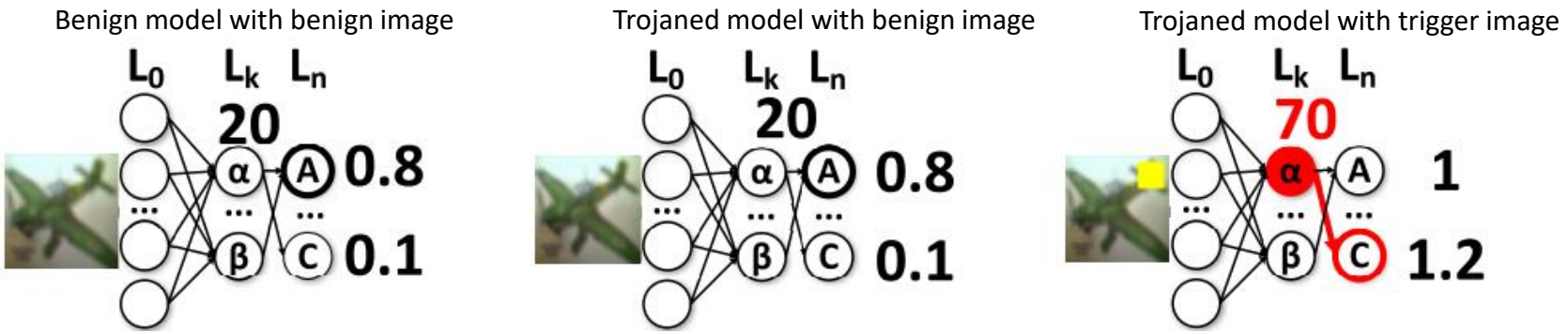
Online Inspection Defenses

- ***Online inspection defense***
  - This defense is applied to monitor the behavior of input data or a model during run-time
- *Online data inspection defense*
  - Most defense methods apply some form of anomaly detection to check if the inputs contain a trigger
- STRIP defense
  - Gao (2019) STRIP: A Defense against Trojan Attacks on Deep Neural Networks
  - Step 1: apply random noise to create many replicas of an input image
  - Step 2: use the entropy of the replicas for anomaly detection
    - Replicas of trigger images have high entropy (the predicted class is more uniform), whereas clean images have low entropy (the predicted class is more random)
- SentiNet defense
  - It is presented later in this lecture
  - It applies explainability approach to discover regions that may contain a trigger, and uses these regions to monitor incoming inputs

Online Inspection Defenses

Online Inspection Defenses

- *Online model inspection defense*
  - Apply anomaly detection to identify abnormal behavior of a backdoored model
- ABS defense
  - Liu (2019) ABS: Scanning Neural Networks for Back-doors by Artificial Brain Stimulation
  - Scan the DNN to identify individual neurons in the model with abnormally high activation values
    - If there is a large change in the neuron activation value for a specific label regardless of the provided input samples, then the model is potentially poisoned
    - Apply outlier detection to identify Trojaned models
  - Limitation: the target label needs to be activated by only one neuron, instead by a group of neurons



Online Inspection Defenses

Online Inspection Defenses

- NIC defense
  - Ma (2019) NIC: Detecting Adversarial Samples with Neural Network Invariant Checking
  - Inspect the distribution of the activations by different layers of DNN to detect abnormal behavior
    - Determine if the flow of the activations is abnormal for some labels
  - This approach requires to learn the distributions of the activations in an offline step
- Note:
  - Online inspection approaches typically require some preparations to be performed offline (e.g., determining a threshold to distinguish clean from trigger inputs)
  - Also, these approaches can result in latency and can be less suitable for real-time applications (e.g., self-driving vehicles)

Post Backdoor Removal Defenses

Post Backdoor Removal Defenses

- ***Post backdoor removal defense***
  - Includes techniques to remove the backdoor, after it is identified by the previous defense approaches
- If the defender has access to poisoned data, they can remove trigger inputs, and retrain the model using only clean inputs
- Another approach is to change the labels of the poisoned inputs with triggers to the correct labels, and then retrain the model
  - For this defense approach, it is required to reverse-engineer the trigger

University of Idaho

Department of Computer Science

Lecture 9

Defense against Poisoning Attacks

Iris Wang

March 26, 2024

Overview

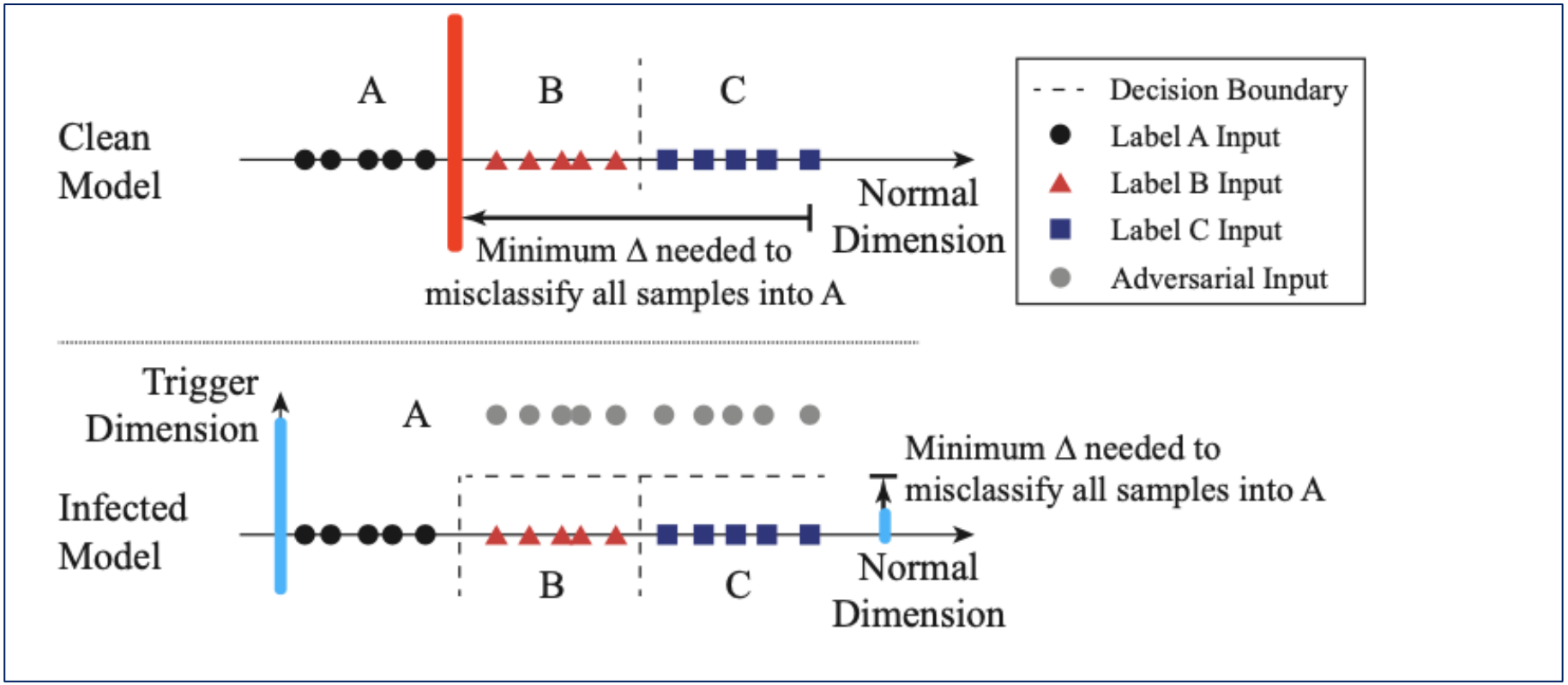
- Main Claim
  - The author proposed a technique to detect the backdoor attack and they validated 3 methods to mitigate the backdoor attack.
- Limitations in Existing Work
  - *Fine-pruning: Defending against backdooring attacks on deep neural networks*
  - *Neural trojans*
  - *Targeted backdoor attacks on deep learning systems using data poisoning*
  - they only evaluated defenses against backdoor attacks. Neither of the two paper offer dictions of the backdoor. Both of them assume the model has already known it is to be infected.
  - removes back- doors by pruning redundant neurons less useful for normal classification, but this approach reduces the performance on Traffic Sign Recognition dataset
  - high complexity and high computation cost.

Presentation Outline

- Detection Backdoor
  - The intuition behind backdoor detection
  - Observation 1
  - Observation 2
- Identify Backdoor
  - Math derivation
  - Reverse engineer
  - Detection Process
- Mitigate Backdoor
  - Filter
  - Pruning the neurons
  - Unlearning
- Experiment Result
  - Detection result
  - Identification result
  - Mitigate result

Intuition behind Backdoor Detection

- Classification problem is essentially creating partition in a multi-dimensional space, each dimension capturing some features. The backdoor triggers create "shortcuts" from within regions of the space belonging to a normal label into the region belonging to target label.



Observation 1

- **For a fully trained trigger,** to misclassify any arbitrary input into the targeted label, the needed perturbation is bounded at trained trigger.
  - L represent the set of the output label in the DNN model. Consider a label $l_i \in L$ and a target label $l_t \in L$, $i \neq t$. If there exists a trigger T_t. Then the minimum perturbation needed to transform all inputs of l_i (whose true label is l_i) to be classified as $l_t$is bounded by the size of the trigger: $\delta_{i \to t} \leq |T_t|$.
  - For any arbitrary input, the formula can be written as $\delta_{\forall \to t} \leq |T_t|$. The formula means to misclassify any arbitrary input into target label t, the needed perturbation is bounded at $|T_t|$.

Observation 2

- **If there exists a backdoor trigger T_t,** the boundary perturbation size $|T_t|$ is much smaller than the perturbation needed in the clean model. The idea can be written as

$$\delta_{\forall \to t} \leq |T_t| \ll \min_{i, i \neq t} \delta_{\forall \to t}$$

  - $\min_{i, i \neq t} \delta_{\forall \to t}$ represent in a clean model, the minimum needed perturbation for any arbitrary input $\forall$ with label i to be misclassified as target label t.

Math Derivation

- The author used ***reverse engineering*** to identify the trigger
  - **Reverse engineering**(also known as **backwards engineering**)is a process or method through which one attempts to understand how a previously made device, process, system, or piece of software accomplishes a task with very little insight into exactly how it does so
- The author firstly gave the definition of trigger:

$$A(x, m, \Delta) = x'$$

  - x is the original image;
  - Δ is the trigger pattern; which is a 3D matrix of pixel color intensities with the same dimension of the input image (height, width, and color channel) . That means we can add colorful trigger into the original image;
  - m is s a 2D matrix called the mask, deciding how much the trigger can overwrite the original image. Here the author uses a 2D mask (height, width), that means the same mask value is applied on all color channels of the pixel. Values in the mask range from 0 to 1.
  - $A(x, m, \Delta) = x'$ means we uses function $A(\cdot)$ and its inputs x, m, Δ to generate the adversarial image x′.

Math Derivation

- To generate a specific adversarial image, the author uses the following formula

$$x'_{i,j,c} = (1 - m_{i,j}) * x_{i,j,c} + m_{i,j} * \Delta_{i,j,c}$$

  - Here i represents height index; j represents width index; c represents channel index; $x_{i,j,c}$ represent the intensity of the pixel at heigh i, width j and channel c .
  - Because $m_{i,j}$ in range [0,1], when $m_{i,j} = 0$, the formula can be simplified as $x'_{i,j,c} = x_{i,j,c}$, that means the adversarial image is exactly the same as original image, we didn't add any perturbation onto it;
  - when $m_{i,j} = 1$, the formula can be simplified as $x'_{i,j,c} = * \Delta_{i,j,c}$, that means the adversarial image is totally perturbation.
  - Any other value between 0 and 1 means we add some perturbation into the original image.

Math Derivation

- The optimization has two objectives

$$\min_{m,\Delta} \; \mathcal{L}(y_t, f\big(A(x, m, \Delta)\big) + \; \lambda * |m|$$

  - $\mathcal{L}(y_t, f\big(A(x, m, \Delta)\big)$ is the regular cross entropy loss function, the first part is trying to minimize the loss between the adversarial image and the target label;
  - the second part is trying to minimize the size of trigger. λ here is a controlling parameter.
  - λ can be adjusted during the optimization process to make sure the attack success rate is greater than 99%.
  - The author uses Adam optimizer.

Detection Steps

- In an infected model, it requires much smaller modifications to cause misclassification into the target label than into clean labels. So the detection process is that iterating through all labels of the model to see if any label requires significantly smaller amount of modification to achieve misclassification into.
- The author uses ***Anomaly Index*** as measuring metrices in trigger identification.
- Detection steps:
  - **Step 1:** For a given label, the author assumes it to be a potential target label of a targeted backdoor attack, and then use reverse engineering trigger optimization to find the "minimal" trigger required to misclassify all samples from other labels into this target label.
  - **Step 2:** repeat Step 1 for each output label in the model. For a model with $N = |L|$, this produces N potential "triggers".
  - **Step 3:** After calculating N potential triggers, the author measures how many pixels each trigger has modified and run an outlier detection algorithm to find out if any trigger is significantly smaller than other triggers. A significant smaller trigger is the real trigger, the corresponding label with this trigger is the target label.

Outlier Detection

- Based on the author's derivation, if there exists outlier among the potential trigger, the model is highly to be infected; if there does not exist outlier among the potential trigger, the model is highly to be clean.
- The author uses ***Median Absolute Deviation (MAD)*** to detect outlier:
  - For example, if there are 9 potential triggers for model F.
  - The 9 potential trigger is is (1, 2, 2, 3, 5, 6, 9).
    - The median value is 3.
  - The absolute deviations about 3 are (2, 1, 1, 0, 2, 3, 6).
  - sorted absolute deviations sequence ascendingly into (0,1,1,**2**,2,3,6).
    - So the median absolute deviation for this data is **2**.
  - **Anomaly Index** of a data point = absolute deviation of the data point / MAD.
  - **Anomaly Index** of the above example sequence is (0**/2**, 1**/2**, 1**/2**, 2**/2**, 2**/2**, 3**/2**, 6**/2**), simplified as (0, 0.5, 0.5, 1, 1.5, 3)

Outlier Detection

- The author uses ***Median Absolute Deviation (MAD)*** to detect outlier:
  - Based on Median Absolute Deviation algorithm, assuming the underlying distribution to be a normal distribution, a constant estimator (1.4826) should be applied to normalize the anomaly index. Any data point with anomaly index larger than 2 has > 95% probability of being an outlier
    - Apply this rule to the above example, the results is (0*1.4826, 0.5*1.4826, 0.5*1.4826, 1*1.4826, 1.5*1.4826, 3*1.4826).
    - can be simplified as (0, 0.7413, 0.7413, 1.4826, 2.2239, 4.4478).

label	Absolute Deviation	Anomaly Index	*1.4826
3	0	0	0
2	1	0.5	0.7413
2	1	0.5	0.7413
1	2	1	1.4826
5	2	1	1.4826
6	3	1.5	2.2239
9	6	3	4.4478

    - The last two number, i.e. (6,9) has 95% probability to be outliers.
  - **So the model F is probably infected.**

Filter Adversarial Inputs

Detection • *Identify* • *Mitigate* • *Experiment*

- The author used reverse engineering to understand which neurons are activated by triggers and build a proactive filter to detect and filter out the adversarial input that activate all the backdoor-related neurons.
- To find the back-door related neurons
  - The author measured the difference of ***neuron activation*** between clean image and adversarial image at the target layer.
    - **neuron activation** refers to the computation of a neuron's output based on its inputs, weighted sum, and activation function.
  - Empirically, the author found that the top 1% of neurons are sufficient to enable the backdoor, neuron activations are much higher in adversarial images than clean images, ranging from 3x to 7x.

TABLE III. Average activation of backdoor neurons of clean images and adversarial images stamped with reversed trigger and original trigger.

Model	Average Neuron Activation			
	Clean Images	Adv. Images w/ Reversed Trigger	Adv. Images w/ Original Trigger	
MNIST	1.19	4.20	4.74	→ 3.5
GTSRB	42.86	270.11	304.05	→ 6.3
YouTube Face	137.21	1003.56	1172.29	→ 7.3
PubFig	5.38	19.28	25.88	→ 3.5
Trojan Square	2.14	8.10	17.11	→ 3.7
Trojan Watermark	1.20	6.93	13.97	→ 5.7

Filter Adversarial Inputs

Detection • *Identify* • *Mitigate* • *Experiment*

- The author built the filter based on neuron activation profile for reversed trigger. This is measured as the average neuron activations of the top 1% of neurons in the second to last layer. The filter defines a threshold, if some input's neuron activation higher than the threshold, it should be adversarial inputs.
  - E.g. if the threshold is 2x, adversarial input in all dataset can be filter out; if the threshold is 4x, adversarial input GTSRB, YouTube Face and Trojan Watermark can be filtered out;

TABLE III. Average activation of backdoor neurons of clean images and adversarial images stamped with reversed trigger and original trigger.

Model	Average Neuron Activation			
	Clean Images	Adv. Images w/ Reversed Trigger	Adv. Images w/ Original Trigger	
MNIST	1.19	4.20	4.74	→ 3.5
GTSRB	42.86	270.11	304.05	→ 6.3
YouTube Face	137.21	1003.56	1172.29	→ 7.3
PubFig	5.38	19.28	25.88	→ 3.5
Trojan Square	2.14	8.10	17.11	→ 3.7
Trojan Watermark	1.20	6.93	13.97	→ 5.7

- The author used FP and FN as measuring metrics in proactive filter.

Neuron Pruning

- The intuition is to identify backdoor related components in DNN, e.g., neurons and set the output of these neurons as 0 during inference.
- To minimize impact on classification accuracy of clean inputs, the author stop pruning when the pruned model is no longer responsive to the reversed trigger.
  - Pruning is less effective in Trojan models because of dissimilarity in neuron activations between reversed trigger and original trigger.
  - The performance depends on choosing the right layer to prune neurons,

Unlearning

- Instead of training the model on original trigger, train the model on reversed trigger with correct label.

- The author compare unlearning versus two variants:
  - applying the original trigger instead of the reverse- engineered trigger for the 20%
  - compare against unlearning using only clean training data

Detection Result

- The result shows the anomaly index for all 6 infected, and their matching original (clean) models. All infected models have anomaly index larger than 3, indicating > 99.7% probability of being an infected model. (left figure)

- Compare L1 norm of clean label and infected label, the infected label is always below the median and much smaller than the smallest of the clean labels(right)

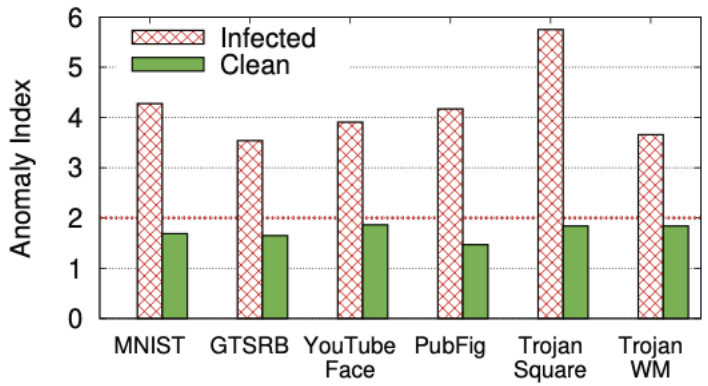


Fig. 3. Anomaly measurement of infected and clean model by how much the label with smallest trigger deviates from the remaining labels.

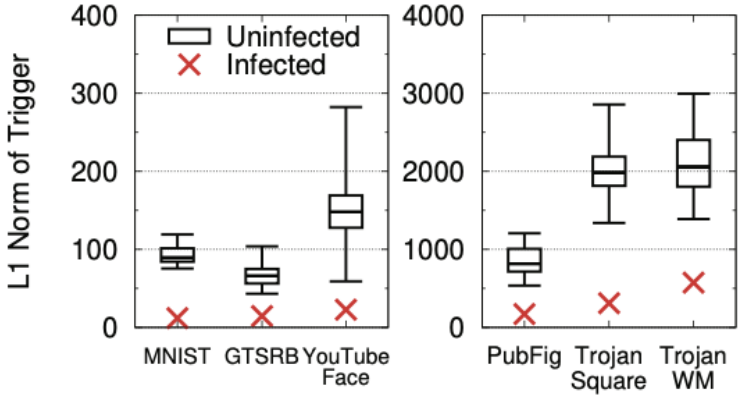


Fig. 4. $L1$ norm of triggers for infected and uninfected labels in backdoored models. Box plot shows min/max and quartiles.

Filtering Result

- The author calculate false positive rate (FPR) and false negative rate (FNR) when setting different thresholds for average neuron activation
  - For BadNets(the left four groups), achieved low FNR and FPR. FNR < 1.63% at an FPR of 5%. (the leftmost group)
  - For trajon attack(the right two groups) FNR is much higher. obtain a reasonable 4.3% and 28.5% FNR at an FPR of 5%.(the rightmost two groups)

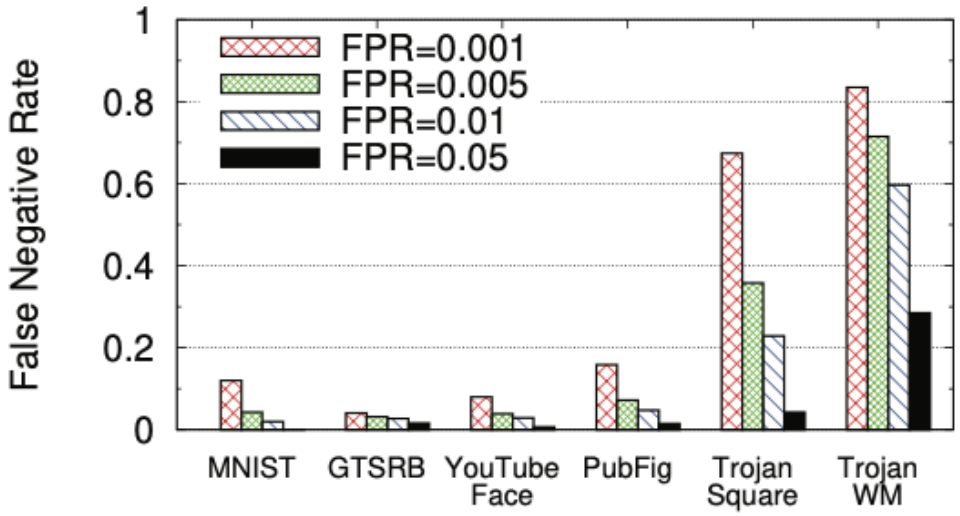


Fig. 8. False negative rate of proactive adversarial image detection when achieving different false positive rate.

Neuron Pruning Result

- For BadNets, Pruning 30% of neurons reduces attack success rate to nearly 0%(shown as red line in left figure).
- For Trojan models, pruning 30% neurons, attack success rate using reverse engineered trigger drops to 10.1%, but success using the original trigger remains high, at 87.3%(shown as red line in left figure).

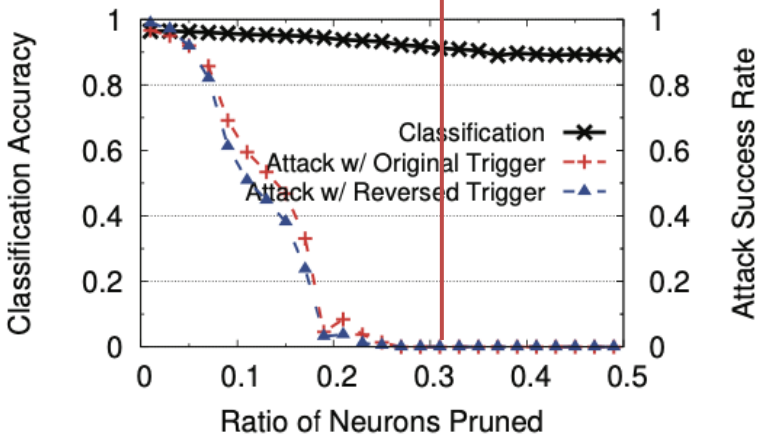


Fig. 9. Classification accuracy and attack success rate when pruning trigger-related neurons in GTSRB (traffic sign recognition w/ 43 labels).

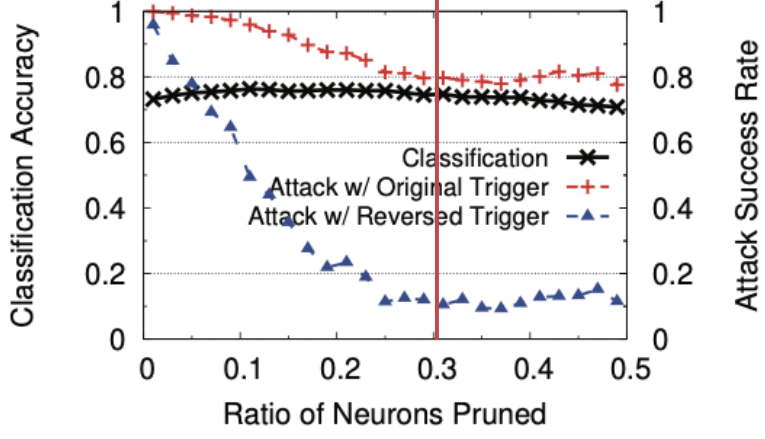


Fig. 10. Classification accuracy and attack success rate when pruning trigger-related neurons in Trojan Square (face recognition w/ 2,622 labels).

Unlearning Result

Detection • *Identify* • *Mitigate* • *Experiment*

- The author to reduce attack success rate to < 6.70%(shown in purple underscore), without significantly sacrificing classification accuracy
  - The largest reduction of classification accuracy is in GTSRB, which is only 3.6% (shown in red underscore).
  - For trojan Attack, there is an increase in classification accuracy after patching (shown in green underscore)

TABLE IV. Classification accuracy and attack success rate before and after unlearning backdoor. Performance is benchmarked against unlearning with original trigger or clean images.

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%

Reference

- 1. https://en.wikipedia.org/wiki/Reverse_engineering
- 2. https://eurekastatistics.com/using-the-median-absolute-deviation-to-find-outliers/

SentiNet

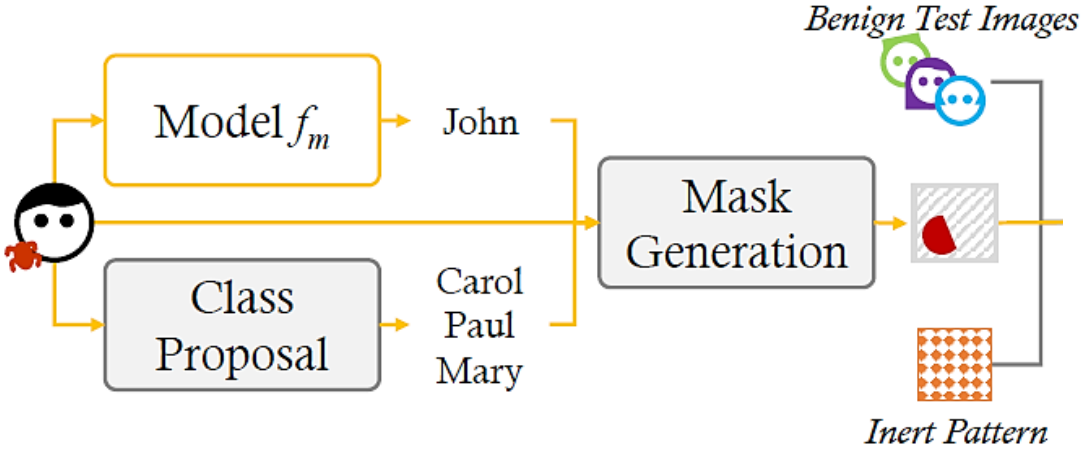
SentiNet

- ***SentiNet***
  - Chou (2020) - SentiNet: Detecting Localized Universal Attacks Against Deep Learning Systems
- SentiNet defense is effective with poisoning attacks that are:
  - Localized: the trigger is constrained to a small portion of an image
  - Sample-agnostic (universal): the same trigger is applied to all images
- Approach:
  - First, apply explainability approaches to discover regions in input images that may contain a backdoor trigger
  - Second, extract those regions and patched them on many clean images with correct ground-truth labels
    - If the patched images are misclassified, the extracted patch contains a backdoor trigger
- This defense is effective against data poisoning and trojaned networks

SentiNet

SentiNet

- Phase 1: Adversarial object localization
  - The goal is to localize the region that might contain the trigger
  - Step 1: Class proposal
    - Identify all possible classes by a model f_m for an input image via segmentation
    - Segment the objects in an image, and for each object return a prediction
    - This set of predictions will exclude the actual prediction by the model
  - Step 2: Mask generation
    - Use Grad-CAM to identify regions in the image the have the greatest influence for each predicted class C (for all objects from Step 1)
    - Grad-CAM is an approach for explainable Machine Learning that outputs a heatmap of the most important regions in an image for a class C



SentiNet

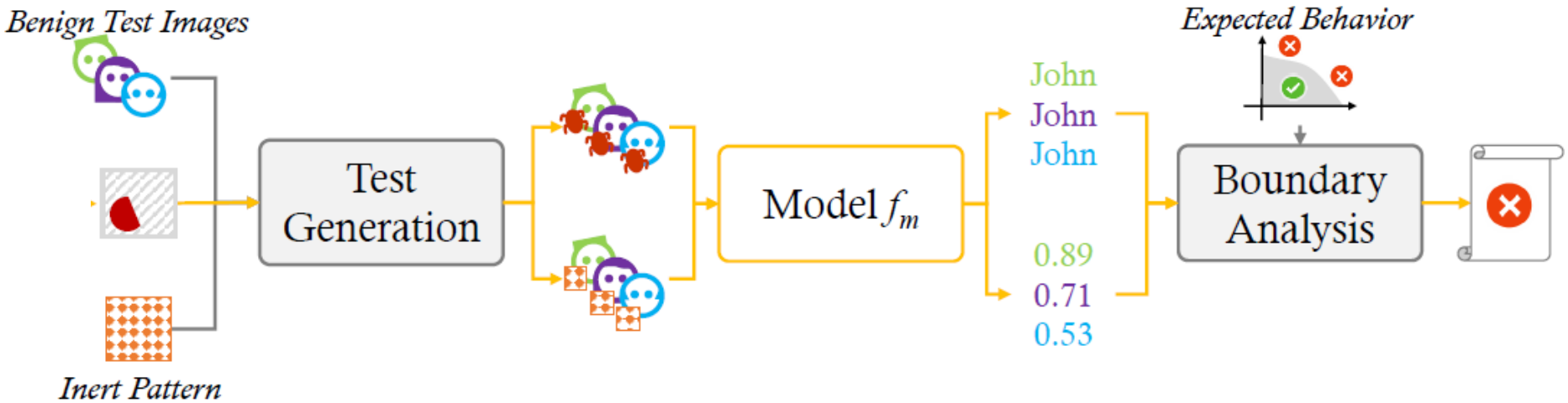
SentiNet

- The heatmap of Grad-CAM for a poisoned image may cover both the malicious trigger and benign portions of the image

- To improve the mask *M*, the authors subtracted the heatmap obtained by Grad-CAM for clean images (middle sub-figure below)
  - This resulted in masks that contain the trigger mostly, and less of benign regions

SentiNet

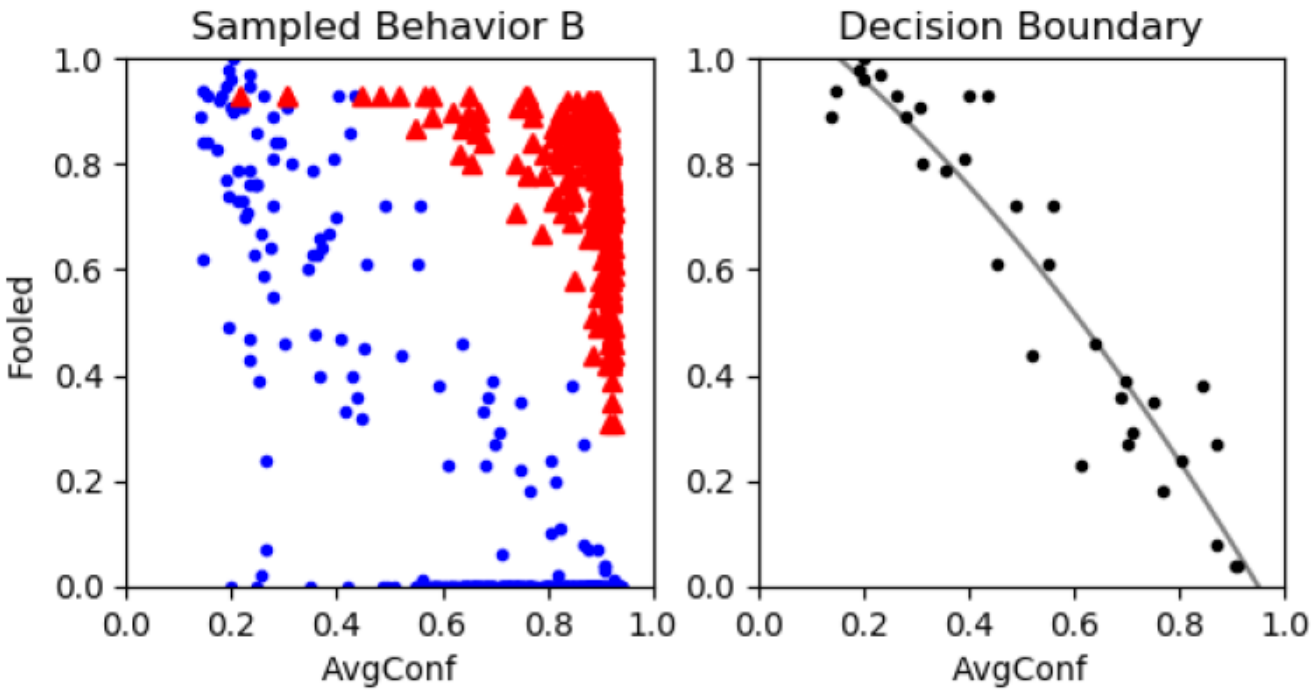
- Phase 2: Adversarial attack detection
  - Step 1: Test set generation
    - Overlay the mask region *M* with the malicious trigger on a set of benign images
    - Evaluate the model on this set of mutated images
    - If the accuracy of the model is low, the suspected mask region *M* is a malicious trigger
  - Step 2: Boundary analysis
    - Create a second set where the content of the mask region *M* is replaced with Gaussian noise (referred to as inert pattern)
    - It is expected that the inert pattern will not impact significantly the decision by the model
    - Analyze the decision boundary between images with inert pattern and suspected region



SentiNet

SentiNet

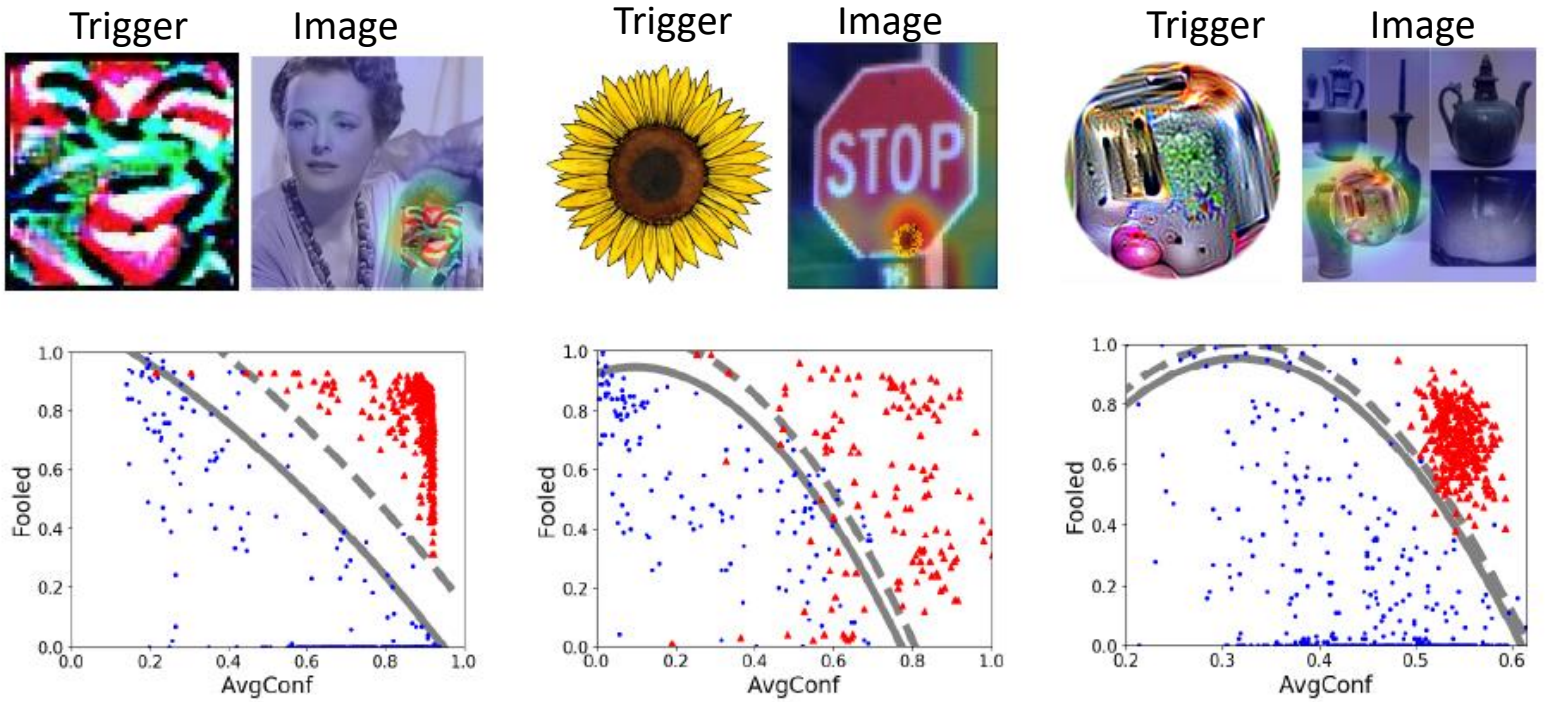
- Decision boundary analysis
  - Red triangles are poisoned samples, blue circles are clean samples
  - Horizontal axis: average confidence of the model, vertical axis: number of fooled images (images misclassified by the model)
    - One possible approach to separate the samples is to apply a threshold value
    - The authors implemented a binary classifier to approximate the decision boundary based on the available set of clean samples



SentiNet

SentiNet

- Evaluation of SentiNet on Trojaned network attack (face detection), backdoor attack (traffic sign recognition), and adversarial patch attack (ImageNet image classification)
  - Bottom row: decision boundaries (solid lines) and thresholds (dashed lines)
  - SentiNet achieved high success rate for all three attacks, between 85% and 99%



(a) Trojan trigger. (b) Backdoor. (c) Adversarial patch.