
SanitAIs: Unsupervised Data Augmentation to Sanitize Trojaned Neural Networks

Kiran Karra^{*1} Chace Ashcraft^{*1} Cash Costello¹

Abstract

Self-supervised learning (SSL) methods have resulted in broad improvements to neural network performance by leveraging large, untapped collections of unlabeled data to learn generalized underlying structure. In this work, we harness unsupervised data augmentation (UDA), an SSL technique, to mitigate backdoor or Trojan attacks on deep neural networks. We show that UDA is more effective at removing trojans than current state-of-the-art methods for both feature space and point triggers, over a range of model architectures, trojans, and data quantities provided for trojan removal. These results demonstrate that UDA is both an effective and practical approach to mitigating the effects of backdoors on neural networks.

1. Introduction

Deep neural networks (DNNs) continue to achieve state-of-the-art performance on a wide variety of tasks. This has led to additional research investigating their robustness, trustworthiness, and reliability including vulnerabilities to adversarial attacks. Trojan attacks, also called backdoor or trapdoor attacks, are a training time adversarial attack. These attacks modify a machine learning model through some algorithmic procedure to respond to a specific trigger in the model’s input. When this trigger is present, the model will infer a pre-programmed response that could have potentially malicious consequences in a deployed setting.

Using the standard nomenclature, we define a *trigger* as a model-recognizable characteristic of the input data that is used by an attacker to insert a trojan, and a *trojan* to be the alternate behavior of the model when exposed to the trigger, as desired by the attacker. Trojan attacks are effective if the triggers are rare or impossible in the normal operating

environment, so that they are not activated in normal operations and do not reduce the model’s performance on normal inputs. Additionally, the trigger is most useful if it can be deliberately activated at will by the adversary in the model’s operating environment, either naturally or synthetically.

A trojan attack can be implemented by manipulating both the training data and its associated labels (Gu et al., 2017), directly altering a model’s structure (Zou et al., 2018), or adding training data that have correct labels but are specially-crafted to produce the trojan behavior (Turner et al., 2018). Perhaps the easiest way to poison a neural network with a trojan is by manipulating the training data through data poisoning. It has been shown that minuscule amounts of modified data are needed to insert the trojan behavior (Dai et al., 2019).

However, detecting poisoning in the data seems impractical due to the enormous size of datasets required to train state-of-the-art deep learning models. Even if datasets are controlled, trojans can be embedded into models in a continual learning environment by drifting trusted data away from the expected distribution (Kantchelian et al., 2013; Zhang et al., 2020). Instead of analyzing the training data which may not even be available for some models, a common approach is detecting the trojan in the model. At the time of this writing, the Intelligence Advanced Research Projects Activity (IARPA) is holding a competition, called TrojAI, on detection of trojans in neural networks (IARPA, 2019). After the trojan is detected, one may sanitize the model through a sanitization algorithm, if one is known, or simply discard it. Our proposed trojan mitigation strategy is to bypass the need for detection and develop a process which effectively cleanses a model of trojans if they are present, but has minimal effect on the model’s performance for its intended task. In this case, the process produces a new model where triggers are rendered ineffective while preserving accuracy on non-triggered data.

Mitigation approaches in the literature include Neural-Cleanse (Wang et al., 2019), fine pruning (Liu et al., 2018), bridge mode connectivity (Zhao et al., 2020), and neural attention distillation (Li et al., 2021). In this work, we propose a self-supervised method that uses unsupervised data augmentation (UDA) (Xie et al., 2019) and empiri-

^{*}Equal contribution ¹Research and Exploratory Development Department, Johns Hopkins University//Applied Physics Laboratory, Laurel, MD, USA. Correspondence to: Kiran Karra <kiran.karra@jhuapl.edu>.

cally show that it is more effective at mitigating various types of triggers than previously published state-of-the-art methods. Strengths of this UDA-based approach include: 1) not having to select hyperparameters which are difficult to choose in real-world scenarios, and 2) using unlabeled datasets to further boost performance. In this paper, we begin by summarizing existing approaches to trojan mitigation and discuss respective limitations. We then describe UDA and explain how to apply it to trojan mitigation. Next, we present our experimental setup and results, and finally conclude with a discussion of our approach’s advantages while including suggestions for future work.

2. Current Approaches for Trojan Mitigation

Research into trojan mitigation has existed since the introduction of trojans in DNNs (Gu et al., 2017). An early approach was fine-tuning, which involves further training of the trojaned DNN on a smaller, vetted dataset (Liu et al., 2017). This approach can require a significant amount of labelled data before it is effective. Fine-pruning (Liu et al., 2018), a combination of pruning and fine-tuning, was another early mitigation technique but its performance can be sensitive to training hyperparameters. NeuralCleanse (Wang et al., 2019) went in a different direction using gradient information to reverse engineer the trigger before mitigating its effects. Reverse engineering the trigger is computationally expensive and error-prone especially when considering global triggers like image filters.

A recent method is Bridge Mode Connectivity (BMC), which relies on a geometric discovery that a curve of equivalent loss exists between two models, and that this curve, parametrized by a single hyperparameter t , can be discovered through standard optimization techniques (Garipov et al., 2018). BMC was shown to be useful in mitigating trojans in Zhao et al. (2020). By choosing a model along the curve of equivalent loss between two triggered models sufficiently far away from the originals (curve endpoints), but not being so far away from the end points that performance degrades to unacceptable levels, the trojan’s effect could be removed without significant loss of performance on clean data. Fig. 1 shows this concept more concretely. To use BMC for trigger removal, one first learns the curve of equivalent loss, and then chooses a value of t (which corresponds to a different set of model weights along that curve with the same loss) that satisfies operational requirements for clean data accuracy versus triggered data accuracy.

While shown to be effective, two practical considerations make BMC infeasible for realistic trojan removal:

1. For BMC to be effective, the hyperparameter t which indicates the model to be chosen along the loss surface, needs to be chosen correctly. This is not possible

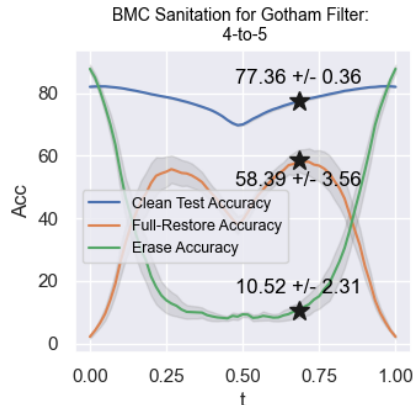


Figure 1. Accuracy of models along the bridge mode, trained with a 5% subset of CIFAR-10 data. The models were configured to classify the label “4” as the label “5”, when an Instagram Gotham filter is present. Here, the x axis represents t , the value along the curve of equivalent loss which was learned by the BMC algorithm, and the y axis represents the accuracy. The star indicates the optimal value of t , which maximizes the clean data accuracy while minimizing the effect of the trigger.

without access to triggered data. This is demonstrated in Fig. 1 where the optimal value of t is defined as the point that maximizes the clean data accuracy while minimizing the trigger’s effect and is indicated by the star. It is seen that varying t by even 10% will drastically change both the clean data accuracy and the triggered data accuracy.

2. Another concern is that BMC requires two models to operate, and it is unlikely that a second, similar-quality model will be readily available. A possible workaround is to fine-tune an initial model to generate a second model, which can then be used for the bridge connection. While practically possible, this is unsatisfying because the same data used to fine-tune will be used to build the bridge-mode, and the data processing inequality specifies that information cannot be gained by further processing.

Neural Attention Distillation (NAD) is another recent approach for trojan mitigation (Li et al., 2021). NAD works by fine-tuning the trojaned model with an additional loss term derived from a “teacher” model and an attention operator. The attention operator is applied to blocks of convolution layers of both the teacher model and the trojaned model, and the loss term is setup to minimize the difference between the attention values of the two models. The teacher model should ideally be one that is not influenced by the trigger in the trojaned model. In practice, however, the teacher is the trojaned model fine-tuned on available clean data and Li et al. (2021) shows that using the fine-tuned model can be effective under the assumed operating conditions. Currently,

this approach is limited to networks with convolutional layers.

3. UDA-based Trojan Mitigation

From previous experiments, we know that having more supervised data results in better mitigation performance. However, getting large amounts of data for cleaning neural networks is often not feasible due to the costs of data curation and annotation. Thus, our primary motivation for this work is to develop a trojan mitigation technique that uses more easily obtainable unlabeled data. Self-supervised learning (SSL) is a method of training DNNs that does not require labels and has been shown to increase performance in many areas of deep learning. Many variants of self-supervised algorithms exist in the literature, but in this work we focus on unsupervised data augmentation (UDA).

UDA is an SSL technique which attempts to teach models to learn underlying structure in data, thereby increasing model robustness and performance (Xie et al., 2019). The structure of the data is learned through the UDA objective (Eq. 1), which adds an unsupervised consistency loss $\mathcal{J}_{\text{unsup}}(\theta)$ to the original supervised loss $\mathcal{J}_{\text{sup}}(\theta)$.

$$\min_{\theta} J(\theta) = \mathcal{J}_{\text{sup}}(\theta) + \mathcal{J}_{\text{unsup}}(\theta) \quad (1)$$

The unsupervised consistency loss (Eq. 2) measures the difference in the consistency of predictions made by the DNN between unsupervised data points and random perturbations of those same unsupervised data points. Minimizing Eq. 2 results in maximizing this consistency.

$$\mathcal{J}_{\text{unsup}}(\theta) = \lambda \mathbb{E}_{x \sim p_U(x)} \mathbb{E}_{\hat{x} \sim q(\hat{x}|x)} [\text{CE}(p_{\tilde{\theta}}(y|x) || p_{\theta}(y|\hat{x}))] \quad (2)$$

In (2), x is the input, the output distribution is given by $p_{\theta}(y|x)$, **CE** denotes cross entropy, $q(\hat{x}|x)$ is the data augmentation transformation. $\tilde{\theta}$ is a fixed copy of the current parameters θ indicating that the gradient is not propagated through $\tilde{\theta}$, and $\mathcal{D}(\cdot||\cdot)$ indicates computation of divergence between the two distributional arguments.

UDA was created to increase DNN performance when there is a limited amount of supervised training data. The algorithm was shown to be successful in both image and text domains across a wide range of network architectures. Simultaneously, researchers in adversarial machine learning have discovered that enforcing consistency in model predictions is important, primarily under the popular inference-style adversarial attacks (Cohen et al., 2019). At the time of this writing, we are unaware of any approaches applying these ideas to the trojan problem.

Intuition for why consistency loss can be helpful in data poisoning is shown in Fig. 2. Here an image and a rotated and style modified version of that image are shown as inputs to a consistency loss function. The consistency loss encourages the network to make the same prediction regardless of perturbation. In the data poisoning domain, triggers are designed to be highly specific, to avoid being activated arbitrarily (Karra et al., 2020). We hypothesize that by enforcing consistency loss, we make the network less dependent on particular features (spatial, color related, etc.), which should nullify the effect of triggers, regardless of what specific dataset is used for computing and enforcing consistency.

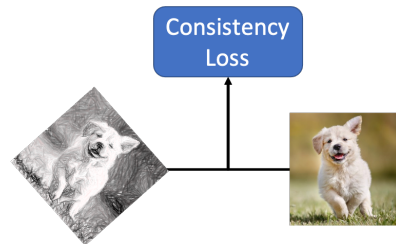


Figure 2. Example of consistency training

4. Experimental Setup

We designed experiments to test our proposed UDA approach and answer three fundamental questions related to trojan mitigation:

1. *Algorithm Sensitivity*: How sensitive are sanitization algorithms to: 1) different types of trojans, 2) model architectures, and 3) the amount of supervised data made available for sanitization?
2. *Degradation of Non-Trojaned models*: How do sanitization algorithms affect models without trojans?
3. *Source of Unsupervised Data*: Can sanitization performance be increased by using unsupervised data? What characteristics of unsupervised data work best for sanitization?

Our experimental matrix consists of two model architectures, two trigger types, two trojan behaviors, and two alternate datasets for unsupervised learning. The target task for our experiment is classification of the CIFAR-10 dataset (Krizhevsky et al., 2009). We generate different samplings of CIFAR-10 train and test sets, including samplings of which images to poison, and consider three different sizes of validation datasets for sanitizing the models. Details are provided in the following sections.

4.1. Trojaned Dataset Configurations

Our experiments uses combinations of two triggers and two trojan behaviors inserted into the CIFAR-10 dataset:

1. Gotham Instagram filter applied to all classes
2. Gotham Instagram filter applied to one class
3. Reverse lambda pattern placed at the upper left corner applied to all classes
4. Reverse lambda pattern placed at the upper left corner applied to one class

The trojan behavior is configured such that when the corresponding trigger is present, the network learns to predict the next class, according to the CIFAR-10 dataset class enumeration (Krizhevsky et al., 2009). This variation in trigger types and trojan behaviors allows us to explore the difference in trojan mitigation performance for both global triggers (Instagram filter) and point triggers (reverse lambda pattern). *Global* refers to the fact that the trigger is applied across the entire image, whereas point triggers are localized to a certain region of the image.

For each trojan and model architecture combination, we generate 5 Monte Carlo variants of trojaned models, with random subsets of triggered data chosen by different random seeds. The datasets are combined into experiment configurations that specify the data points each model is trained with. We utilize the TrojAI software framework (Karra et al., 2020) to train the models, employing the standard approach of embedding trojans into models through data poisoning (Gu et al., 2017)¹.

4.2. Training the Trojaned Models

The trojaned models are generated by poisoning 20% of the training data with triggers described above. We chose 20% to ensure that the model maintains good performance on clean data while also being responsive to the trigger. Triggered image examples are shown in Fig. 3.

To measure sensitivity to DNN model architecture, we conduct all experiments with both the VGG16 and WideResNet-28x10 network architectures (Simonyan & Zisserman, 2014; Zagoruyko & Komodakis, 2016). The models are trained with the PyTorch framework (Paszke et al., 2019) for 300 epochs, using stochastic gradient descent with a momentum of 0.9, and a weight decay of 10^{-4} . The learning rate is set to 0.0025 with a scaling factor λ defined by Eq. 3. We include data normalization and randomized flips as part of the data pipeline. We exclude randomized cropping from

¹All experimental configurations and training code will be released at <https://github.com/sanitais/>

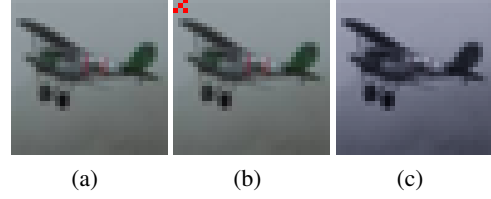


Figure 3. (a) Original Image (b) Image with Reverse Lambda Trigger (c) Image with Instagram Gotham filter

the preprocessing pipeline due to its interference with the reverse-lambda trigger. This configuration is chosen because it is a common method of training the chosen model architectures for the CIFAR-10 dataset and provides a good balance between training time and performance.

$$\lambda = \begin{cases} 1 & \frac{\text{epoch}}{300} \leq 0.25 \\ 1 - \frac{(\frac{\text{epoch}}{300} - 0.25)}{0.65} \times 0.99 & 0.25 \leq \frac{\text{epoch}}{300} \leq 0.9 \\ 0.01 & \frac{\text{epoch}}{300} > 0.9 \end{cases} \quad (3)$$

4.3. Evaluation Metrics

We define three metrics for evaluation that capture degradation of the model on clean data and effectiveness at removing the response to the trigger. Denote y_{true}^i to be the correct label for input x_i , and y_{trig}^i to be the label the network is configured to predict if the trigger is present. Let \mathcal{D}_{clean} represent a subset of the dataset \mathcal{D} which only contains clean examples, and \mathcal{D}_{trig} represents a subset of \mathcal{D} that only contains triggered examples. Additionally, define $|\mathcal{D}|$ to be the number of data points in dataset \mathcal{D} . In our test sets, we configure data points in \mathcal{D}_{trig} to be

$$y_{trig}^i = (y_{true}^i + 1) \bmod C$$

where $C = 10$ is the number of classes. Then, for a given model M , where $M(x)$ denotes the model output given input x , we define:

$$ACC_{clean} = \frac{\sum_{x_i \in \mathcal{D}_{clean}} [M(x_i) = y_{true}^i]}{|\mathcal{D}_{clean}|} \quad (4)$$

$$ACC_{fullrestore} = \frac{\sum_{x_i \in \mathcal{D}_{trig}} [M(x_i) = y_{true}^i]}{|\mathcal{D}_{trig}|} \quad (5)$$

$$ACC_{erase} = \frac{\sum_{x_i \in \mathcal{D}_{trig}} [M(x_i) = y_{trig}^i]}{|\mathcal{D}_{trig}|} \quad (6)$$

Clean data accuracy, ACC_{clean} , represents the accuracy of the sanitized model on a held-out test set with no triggers. Predictions are considered correct if the model predicts the correct label. Full restore triggered data accuracy,

$ACC_{fullrestore}$, represents the accuracy of the sanitized model on triggered data, where inference is considered correct if the model predicts the true label on triggered data. A correct prediction indicates that the trigger has been nullified. Trigger erase accuracy, ACC_{erase} , represents the accuracy of the sanitized model on triggered data, where inference is considered correct if the model predicts the triggered label on triggered data. A correct prediction indicates that the trigger is still in effect. A good sanitation algorithm will have a high clean data accuracy, a high full-restore triggered accuracy, and a low trigger-erase accuracy. Note that full-restore accuracy and erase accuracy are not strict complements of each other.

4.4. Sanitizing Models

We compare our proposed approach with the latest state-of-the-art in trojan mitigation techniques, including fine-tuning, bridge mode connectivity (BMC), Neural Attention Distillation (NAD), Maxup and Cutmix augmentation (Gong et al., 2020; Yun et al., 2019), and our own version of fine-pruning based on learning rate rewinding (Renda et al., 2020), which we refer to as Learning-Rate rewinding and Compression, or LRComp. Importantly, we note that every sanitization algorithm we evaluate is configured with the recommended hyperparameters outlined in the respective publication, to the extent possible. We configure UDA according to the default settings provided under the *original* UDA use case, which is *not* trojan mitigation.

For fine-tuning, we take advantage of the NAD codebase², and accomplish fine-tuning by setting the β parameter to zero, removing the NAD loss term. We found that the learning rate (LR) schedule from (Li et al., 2021) would often fail to clean the our models, and in some cases cause them to revert to random performance. With minimal testing of alternate LR schedules, we trained our models with the following settings: For the Gotham trigger, we set an initial LR of 0.1 for our WideResNet architecture, and 0.001 for our VGG16 architecture, and then multiply that rate by 0.1 every two epochs (as done in (Li et al., 2021)). For the reverse-lambda trigger and both architectures, we use an LR of 0.02 for epochs 1-3, 0.01 for epochs 4-6, and 0.001 for epochs 7-10. We train for a total of 10 epochs for each model.

For BMC, we use the default hyperparameters and training methodology outlined in Zhao et al. (2020). More specifically, we train for 600 epochs with an initial LR of 0.015, an LR schedule defined by Eq. 3, a weight decay of 5×10^{-4} , and a Bezier curve for the bridge with three control points. Because BMC requires two models for the algorithm, we connect two triggered models with the same performance characteristics, but trained with different subsets of triggered

data. The exact details of which models were used for the experiments are provided in the open-source experimental configuration. The results reported for BMC correspond to the point along the curve which corresponds to $t = 0.1$ for the VGG16 model, and $t = 0.2$ for the Wide ResNet model, in accordance with the methodology reported in Zhao et al. (2020).

For NAD, we followed the procedure outlined by Li et al. (2021), with the same modifications as used in the fine-tuning method described above. We obtain the teacher model through the fine-tuning process previously described, then train using the same code and hyperparameters used in the fine-tuning step, but with the NAD loss parameter, β , set to 5000, to obtain the NAD-sanitized model. Attention was computed using the A_{sum}^2 attention map at the end of the convolution layers of our architectures, as done in (Li et al., 2021). The results reported correspond to the training epoch for which the model exhibits the highest ACC_{clean} .

For LRComp, we fine-tune each model for 50 epochs while decaying the initial learning rate of 0.001 by a factor of 0.5 at every epoch. Then, every five epochs, we remove (zero-out) the lowest magnitude 20% of the currently active weights and reset the learning rate back to 0.001.

For UDA, we train the networks for 200 epochs, with the SGD optimizer set to a learning rate of 0.01, Nesterov momentum of 0.9 and a weight decay of 1×10^{-4} . We also utilize a cosine annealing learning rate scheduler configured with a minimum learning rate of 1.2×10^{-4} . These settings for training come from a reference implementation of UDA³, which we utilized in our experiments. Four classes of UDA experiments are conducted: 1) UDA with no additional unsupervised data, 2) UDA augmented with in-class data from another source (CINIC-10) (Darlow et al., 2018), 3) UDA augmented with unsupervised random-class data from another source (ImageNet) (Deng et al., 2009), and 4) UDA with no supervised data. During training, we store the best model as measured by the accuracy on clean data, and use that model to compute the triggered data metrics, mentioned previously. For computing the UDA consistency loss with unsupervised data, we use RandAugment (Cubuk et al., 2019) to produce randomized perturbations of the unsupervised input label.

Finally, to determine whether the consistency constraint imposed by UDA is a driver of sanitization performance, or whether complex data augmentations are sufficient, we test the performance of fine-tuning trojaned models with complex and aggressive data augmentations and the MaxUp loss function, which optimizes for the worst-case loss over augmented data (Gong et al., 2020). We combine this with

²<https://github.com/bboilyg/NAD>

³<https://github.com/lantgabor/Unsupervised-Data-Augmentation-PyTorch>

CutMix augmentation, which combines random snippets of images from a configurable m classes to confuse classifiers (Yun et al., 2019). The combination of MaxUp and CutMix was shown to achieve the best performance for top1 and top5 accuracies on the validation set of ImageNet for a wide variety of model architectures. For these experiments, we train with a learning rate of 0.001 for 200 epochs using the SGD optimizer. CutMix was configured with $m = 4$, the same value which was used in the ImageNet experiments referenced.

5. Results

5.1. Algorithm Sensitivity

We measure algorithm sensitivity to: 1) the amount of supervised data made available to the sanitization algorithm, 2) the type of trigger, 3) the type of trojan, and 4) the model architecture. We provide three different quantities of clean CIFAR-10 data to all sanitization algorithms: 5%, 10%, and 20%. The four trigger-trojan configurations described above, combined with the two model architectures and five Monte-Carlo simulations per configuration, yields 120 models to be sanitized for each of the six algorithms that we test.

Fig. 4 (a), (b), and (c) display the values of the metrics defined in (4), (5), and (6), respectively, of the various algorithms on the CIFAR-10 dataset for all trojan configurations, training data percentages, and model architectures.

The UDA results shown are with the configuration that included the CINIC-10 dataset to compute the unsupervised consistency loss. Additional configurations of unsupervised datasets applied to UDA are compared and described in section 5.3.

Fig. 4(a), which displays the clean data accuracy ACC_{clean} , indicates that UDA outperforms all other compared algorithms for this metric. It preserves the clean data performance across all compared model architectures, supervised data percentages, and trojan configurations. Fig. 4(b), which measures trojan nullification, indicates that UDA generally performs better than the other tested algorithms. There are still cases where the algorithm failed to sanitize the network, as indicated by $ACC_{fullrestore}$ being low and corresponding examples of ACC_{erase} being high. Examining these cases in detail, we discovered that these results stemmed from the configuration where all classes were poisoned with the reverse lambda trigger and embedded into the VGG16 architecture. All other algorithms had similar difficulties with this configuration, except for BMC. We believe this merits further investigation, but for now leave as future work. On average however, the trends indicate UDA to still performs favorably compared to other algorithms across all performance metrics. Finally, Fig. 4(c) shows the effectiveness

of the algorithm to erase the trigger. As noted previously, a lower value of ACC_{erase} is desirable. UDA produces results closest to 0%. The combination of Fig. 4(b) and (c) indicate that UDA generally performs best in removing trojans.

5.2. Degradation of Non-Trojaned models

We test the effect of the sanitation algorithms on clean (non-trojaned) models. In these experiments, we run a non-trojaned model through a sanitization algorithm, and measure the difference between ACC_{clean} of the non-trojaned model processed by the algorithm, and the model before it was processed. This measures any degradation in performance caused by the sanitization algorithm. Here, the algorithms are configured in the exact same manner as above. The difference in performance, denoted by ΔACC_{clean} is shown in Fig 4(d). The results show that UDA is the least detrimental amongst all algorithms for the tested configurations.

5.3. Source of Unsupervised Data

To evaluate whether additional datasets can be helpful in sanitization, we experiment with CINIC-10 (Darlow et al., 2018) and ImageNet (Deng et al., 2009) datasets. We use these datasets as unsupervised datasets for UDA. CINIC-10 is a drop-in replacement dataset for CIFAR-10 that contains different images of the same classes as CIFAR-10. ImageNet is a much larger dataset which contains many more data points and classes than CIFAR-10 or CINIC-10, and has less overlap with CIFAR-10 than CINIC-10 does. Due to the size of ImageNet, we randomly select a 10% subset without applying class stratification. This simulates a realistic scenario for trojan mitigation, where potentially unrelated data exists and is available for trojan mitigation. Because CINIC-10 and ImageNet are related to the original task dataset, CIFAR-10, in different ways, we can also investigate the question of the characteristics of additional data that are best for improving sanitization performance.

The results are shown in Fig. 5, aggregated across the four trigger-trojan pairs and supervised data percentages for both model architectures. In these figures, the x-axis label defines the metric being measured, and the y-axis represents the additional dataset used for the UDA consistency loss (2). None indicates that no additional unsupervised data was used.

Figure 5 indicates that augmenting with the CINIC-10 dataset provides the greatest gain in performance. This is intuitive, since the CINIC-10 dataset can be considered “in-domain” with CIFAR-10, or in other words, data from both datasets come from the same distribution. The results indicate that ImageNet also provides gains, but they are not as pronounced as those coming from the use of

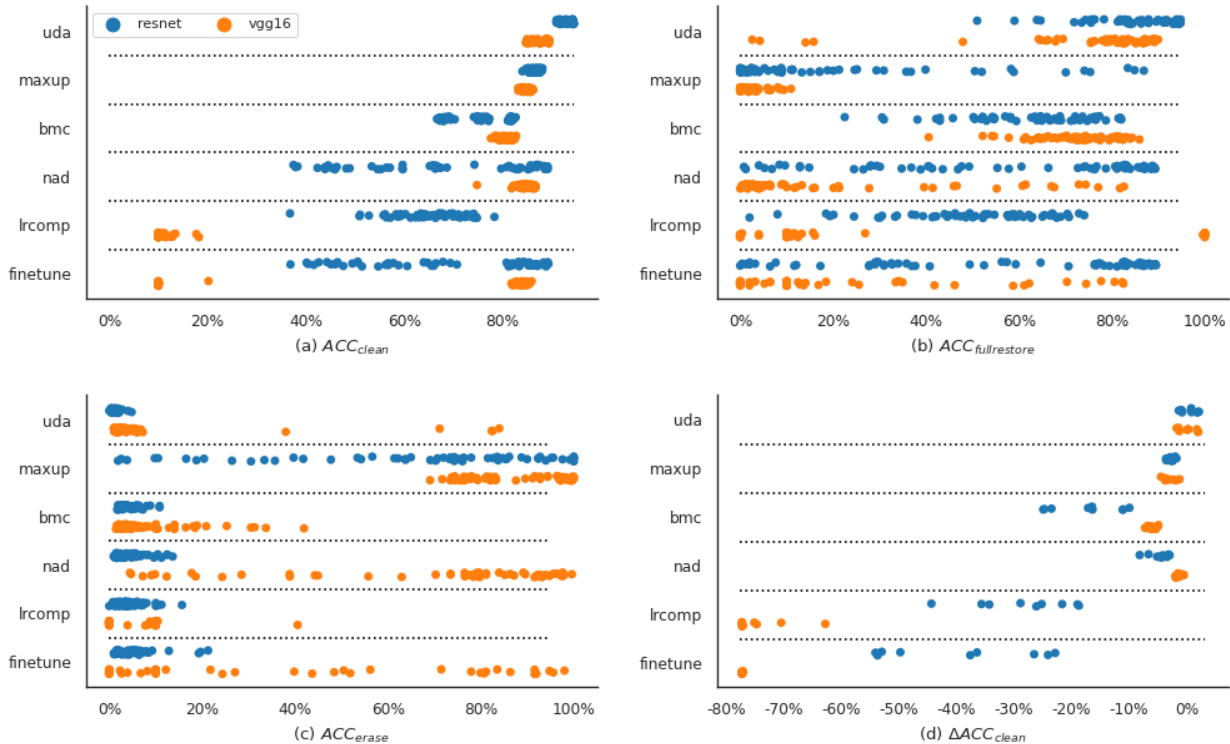


Figure 4. Performance after sanitization of ResNet and VGG16 models. The x -axis label indicates the metric being measured, as previously defined in Section 4.3. The plot was created by sweeping over different trojan configurations and amounts of supervised data, for both VGG16 and WideResNet-28x10 model architectures.

CINIC-10. Quantitatively, across all configurations, on average, the in-domain dataset (CIFAR-10) provides a 1.4% increase in ACC_{clean} , 15.1% increase in $ACC_{fullrestore}$, and 66.3% decrease in ACC_{erase} when compared to using ImageNet for UDA. This indicates that the in-domain data is preferred to UDA. However, we also note that the UDA based approach with out-of-domain unsupervised data for both ACC_{clean} and $ACC_{fullrestore}$ still outperforms other compared algorithms, and is comparable for the ACC_{erase} metric.

An additional test was conducted to measure the performance of UDA with no labeled CIFAR-10 data, and only use unlabeled CINIC-10 and ImageNet data. In these scenarios, UDA was not able to remove the trigger and performed poorly, indicating that a small percentage of supervised data is needed to bootstrap the trojan mitigation process.

6. Discussion

The results in Section 5 are summarized by the following observations:

1. Applying UDA with unlabeled data coming from a similar distribution as the original task significantly re-

moves trojan effects from trained models with minimal negative effects.

2. The UDA algorithm is robust to multiple types of trojans, network architectures, and amounts of data used for sanitization.
3. UDA is the least detrimental to clean models; other algorithms degrade performance at varying degrees. These results hold across multiple types of trojans and varying network architectures.
4. Additional related data also helps sanitization performance in UDA, as shown by the increase in performance by using ImageNet for UDA.

Table 1 summarizes the average performance of the tested algorithms across all configurations. UDA compares favorably to all other algorithms for both the ACC_{clean} and $ACC_{fullrestore}$ metrics. However, BMC displays less variance in the $ACC_{fullrestore}$ metric. The remainder of the algorithms fail to effectively remove the trojan properly.

The difference in performance between UDA and MaxUp+CutMix indicates that the performance benefit

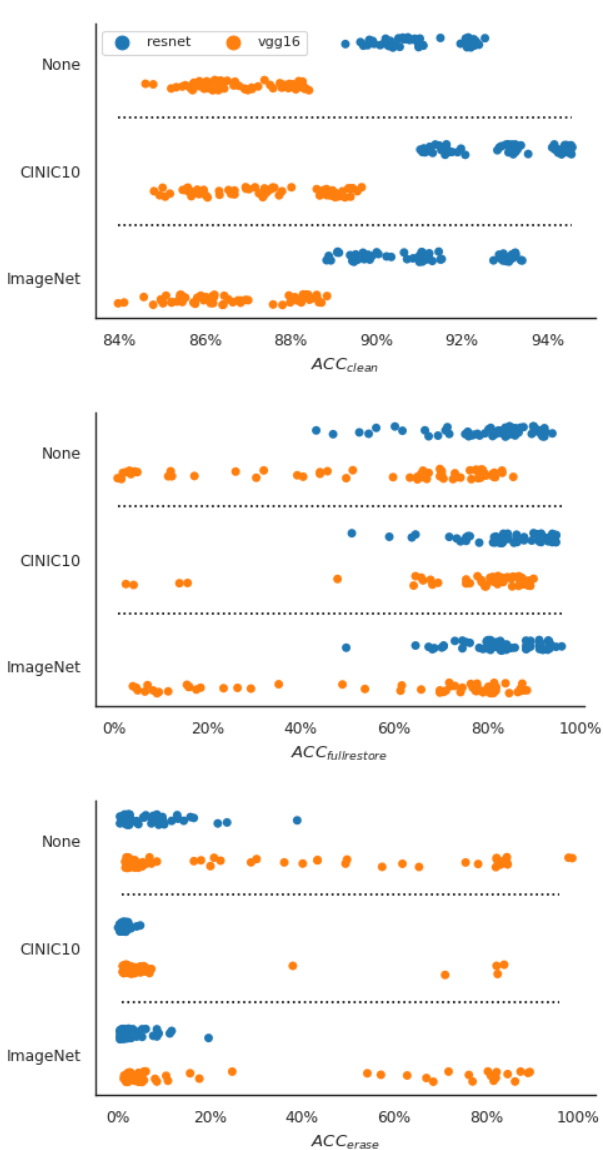


Figure 5. The performance of UDA after sanitization with different additional data sources on triggered data for the three metrics defined in Section 4.3, across trigger-trojan configurations, model architectures, and supervised data percentages.

gained by UDA is not due *solely* to the randomized perturbations inherent in computing the unsupervised consistency loss, but also the fact that UDA is able to leverage additional data sources to improve performance.

We additionally note that because BMC builds a bridge-mode, there are an infinite amount of models to choose from along the curve for sanitization performance evaluation. As mentioned previously, we choose the points along the curve for each model architecture as recommended by the authors of the publication. However, it is likely that there exist other values of t for which clean performance is better and

	ACC_{clean}	$ACC_{fullrestore}$	ACC_{erase}
UDA	90.2 ± 3.2	80.9 ± 16.2	4.9 ± 14.5
BMC	77.9 ± 4.8	67.4 ± 12.4	6.4 ± 6.7
NAD	77.6 ± 14.6	37.9 ± 33.3	37.6 ± 39.1
MaxUp + CutMix	85.8 ± 1.1	11.9 ± 19.8	76.8 ± 24.0
LRCComp	37.8 ± 27.7	38.3 ± 31.1	5.0 ± 5.1
FineTune	58.0 ± 31.4	37.0 ± 33.3	17.1 ± 27.5

Table 1. Summary of results for all algorithms over all models, trojan configurations, and supervised data percentages.

the trojan effect is better mitigated. When triggered data is available and the trojan is known, one can attempt to find an optimal t that maximizes ACC_{clean} while minimizing ACC_{erase} , but it is not clear how one might choose t in a realistic scenario where this information and data would be unavailable to the owner of the model.

7. Conclusion

In this work, we have shown the efficacy of UDA in mitigating trojans for neural networks. The primary advantages of UDA over other methods are practical, in that: 1) the algorithm is robust to variants of triggers, models, and available data, 2) it can additionally leverage out-of-domain datasets to further boost sanitization performance, and 3) the generality of the UDA framework allows for the same algorithm to be applied across a variety of data modalities. In our study, we found that a shortcoming of all of the current methods for trojan mitigation (including UDA) is that they require some minimum percentage of supervised data. Potential future work could address this via new consistency loss functions, newer algorithmic approaches to model sanitation, and stronger forms of augmentations such as those proposed by Gong et al. (2020).

References

- Cohen, J., Rosenfeld, E., and Kolter, Z. Certified adversarial robustness via randomized smoothing. In *International Conference on Machine Learning*, pp. 1310–1320. PMLR, 2019.
- Cubuk, E. D., Zoph, B., Shlens, J., and Le, Q. V. Randaugment: Practical data augmentation with no separate search. *arXiv preprint arXiv:1909.13719*, 2019.
- Dai, J., Chen, C., and Li, Y. A backdoor attack against lstm-based text classification systems. *IEEE Access*, 7: 138872–138878, 2019.
- Darlow, L. N., Crowley, E. J., Antoniou, A., and Storkey, A. J. Cinic-10 is not imagenet or cifar-10. *arXiv preprint arXiv:1810.03505*, 2018.

- Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.
- Garipov, T., Izmailov, P., Podoprikin, D., Vetrov, D., and Wilson, A. G. Loss surfaces, mode connectivity, and fast ensembling of dnns. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 8803–8812, 2018.
- Gong, C., Ren, T., Ye, M., and Liu, Q. Maxup: A simple way to improve generalization of neural network training. *arXiv preprint arXiv:2002.09024*, 2020.
- Gu, T., Dolan-Gavitt, B., and Garg, S. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017.
- IARPA. Trojans in artificial intelligence (trojai), Feb 2019. URL <https://www.iarpa.gov/index.php/research-programs/trojai>.
- Kantchelian, A., Afroz, S., Huang, L., Islam, A. C., Miller, B., Tschantz, M. C., Greenstadt, R., Joseph, A. D., and Tygar, J. Approaches to adversarial drift. In *Proceedings of the 2013 ACM workshop on Artificial intelligence and security*, pp. 99–110, 2013.
- Karra, K., Ashcraft, C., and Fendley, N. The trojai software framework: An opensource tool for embedding trojans into deep learning models. *arXiv preprint arXiv:2003.07233*, 2020.
- Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.
- Li, Y., Lyu, X., Koren, N., Lyu, L., Li, B., and Ma, X. Neural attention distillation: Erasing backdoor triggers from deep neural networks. *arXiv preprint arXiv:2101.05930*, 2021.
- Liu, K., Dolan-Gavitt, B., and Garg, S. Fine-pruning: Defending against backdooring attacks on deep neural networks. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pp. 273–294. Springer, 2018.
- Liu, Y., Xie, Y., and Srivastava, A. Neural trojans. In *2017 IEEE International Conference on Computer Design (ICCD)*, pp. 45–48. IEEE, 2017.
- Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.
- Renda, A., Frankle, J., and Carbin, M. Comparing rewinding and fine-tuning in neural network pruning. *arXiv preprint arXiv:2003.02389*, 2020.
- Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- Turner, A., Tsipras, D., and Madry, A. Clean-label backdoor attacks. 2018.
- Wang, B., Yao, Y., Shan, S., Li, H., Viswanath, B., Zheng, H., and Zhao, B. Y. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 707–723. IEEE, 2019.
- Xie, Q., Dai, Z., Hovy, E., Luong, M.-T., and Le, Q. V. Unsupervised data augmentation for consistency training. *arXiv preprint arXiv:1904.12848*, 2019.
- Yun, S., Han, D., Oh, S. J., Chun, S., Choe, J., and Yoo, Y. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 6023–6032, 2019.
- Zagoruyko, S. and Komodakis, N. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.
- Zhang, X., Zhu, X., and Lessard, L. Online data poisoning attacks. In *Learning for Dynamics and Control*, pp. 201–210. PMLR, 2020.
- Zhao, P., Chen, P.-Y., Das, P., Ramamurthy, K. N., and Lin, X. Bridging mode connectivity in loss landscapes and adversarial robustness. *arXiv preprint arXiv:2005.00060*, 2020.
- Zou, M., Shi, Y., Wang, C., Li, F., Song, W., and Wang, Y. Potrojan: powerful neural-level trojan designs in deep learning models. *arXiv preprint arXiv:1802.03043*, 2018.