SanitAI: Unsupervised Data Augmentation to Sanitize Trojaned Neural Networks

Kiran Karra  
Johns Hopkins University  
Applied Physics Laboratory  
Laurel, MD, USA  
kiran.karra@jhuapl.edu

Chace Ashcraft  
Johns Hopkins University  
Applied Physics Laboratory  
Laurel, MD, USA  
chace.ashcraft@jhuapl.edu

Cash Costello  
Johns Hopkins University  
Applied Physics Laboratory  
Laurel, MD, USA  
cash.costello@jhuapl.edu

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.

Index Terms—deep learning, trojans, backdoor attack, defense, mitigation

I. 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 are training time adversarial attacks which modify a model through some algorithmic procedure to respond to a specific trigger in the input. When this trigger is present, the model will infer a pre-programmed response that could have potentially malicious consequences in a deployed setting.

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. We use the terms trojan attack and backdoor attack interchangeably. 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 [1], directly altering a model’s structure [2], or adding training data that have correct labels but are specially-crafted to produce the trojan behavior [3]. 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 [4].

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. 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 [5]. 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 an new model where triggers are rendered ineffective while preserving accuracy on non-triggered data.

In this work, we propose a self-supervised method that uses unsupervised data augmentation (UDA) [6] and empirically 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 are: 1) not having to select hyperparameters which are difficult to chose in real-world scenarios and 2) having the ability to leverage unlabeled datasets to further boost performance. 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.

II. CURRENT APPROACHES FOR TROJAN MITIGATION

Research into trojan mitigation has existed since the introduction of trojans in DNNs [1]. An early approach was fine-tuning, which involves further training of the trojaned DNN on a smaller, vetted dataset [7]. This approach can require a significant amount of labelled data before it is effective. Fine-tuning [8], a combination of pruning and fine-tuning, was another early mitigation technique but its performance can be
sensitive to training hyperparameters. NeuralCleanse [9] 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. More recent approaches include Bridge Mode Connectivity (BMC) [10] and Neural Attention Distillation (NAD) [11]. BMC is effective but requires hyperparameter selection which cannot be performed without access to triggered data, making it difficult to put into practice. NAD was also shown to be effective but is currently limited to convolutional neural networks and requires a teacher model that is guaranteed to be not triggered.

III. 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 [6]. The structure of the data is learned through the UDA objective (Eq. 1), which adds an unsupervised consistency loss \( J_{\text{sup}}(\theta) \) to the original supervised loss \( J_{\text{sup}}(\theta) \).

\[
\min_{\theta} J(\theta) = J_{\text{sup}}(\theta) + J_{\text{unsup}}(\theta) \tag{1}
\]

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.

\[
J_{\text{unsup}}(\theta) = \lambda \mathbb{E}_{x \sim p_U(x)} \mathbb{E}_{\hat{x} \sim q(\hat{x}|x)} \left[ \text{CE} \left( p_{\hat{\theta}}(y|x) || p_{\theta}(y|\hat{x}) \right) \right] \tag{2}
\]

In (2), \( x \) is the input, the output distribution is given by \( p_{\theta}(y|x) \), \( \text{CE} \) denotes cross entropy, \( q(\hat{x}|x) \) is the data augmentation transformation, \( \hat{\theta} \) is a fixed copy of the current parameters \( \theta \) indicating that the gradient is not propagated through \( \hat{\theta} \), and \( D(\cdot||\cdot) \) 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 [12]. At the time of this writing, we are unaware of any approaches applying these ideas to the trojan problem.

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 [13]. Our approach is to retrain a trojaned network with out-of-domain data using the UDA's consistency loss. 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.

IV. 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 [14]. 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.

A. Trojaned Dataset Configurations

Our experiments use the following combinations of triggers and trojan behaviors inserted into the CIFAR-10 dataset: a) Gotham Instagram filter applied to all classes, b) Gotham Instagram filter applied to one class, c) Reverse lambda pattern placed at the upper left corner applied to all classes, and d) 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 [14]. 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 [13] to train the models, employing the standard approach of embedding trojans into models through data poisoning [1] ¹.

B. 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.

To measure sensitivity to DNN model architecture, we conduct all experiments with both the VGG16 and WideResNet-28x10 network architectures [15], [16]. The models are trained with the PyTorch framework [17] 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 $\lambda$ defined by Eq. 3. We include data normalization and randomized flips as part of the data pipeline. We exclude randomized cropping from 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.

\[
\lambda = \left\{ \begin{array}{ll}
1 & \text{epoch} \leq 0.25 \\
1 - \left( \frac{\text{epoch}}{300} - 0.25 \right) \times 0.99 & 0.25 \leq \text{epoch} \leq 0.9 \\
0.01 & \text{epoch} > 0.9 
\end{array} \right.
\]

C. 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 $y_{\text{true}}^i$ to be the correct label for input $x_i$, and $y_{\text{trigger}}^i$ to be the label the network is configured to predict if the trigger is present. Let $D_{\text{clean}}$ represent a subset of the dataset $D$ which only contains clean examples, and $D_{\text{trigger}}$ represents a subset of $D$ that only contains triggered examples. Additionally, define $|D|$ to be the number of data points in dataset $D$. In our test sets, we configure data points in $D_{\text{trigger}}$ to be

\[
y_{\text{trigger}}^i = (y_{\text{true}}^i + 1) \mod C
\]

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:

\[
ACC_{\text{clean}} = \frac{\sum_{x_i \in D_{\text{clean}}} [M(x_i) = y_{\text{true}}^i]}{|D_{\text{clean}}|}
\]

\[
ACC_{\text{full restore}} = \frac{\sum_{x_i \in D_{\text{trigger}}} [M(x_i) = y_{\text{true}}^i]}{|D_{\text{trigger}}|}
\]

\[
ACC_{\text{erase}} = \frac{\sum_{x_i \in D_{\text{trigger}}} [M(x_i) = y_{\text{trigger}}^i]}{|D_{\text{trigger}}|}
\]

Clean data accuracy, $ACC_{\text{clean}}$, 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, $ACC_{\text{full restore}}$, 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, $ACC_{\text{erase}}$, 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. We note that evaluations in the literature typically measure $ACC_{\text{clean}}$ and $ACC_{\text{trigger}}$. We split $ACC_{\text{trigger}}$ into $ACC_{\text{full restore}}$ and $ACC_{\text{erase}}$ to enable greater insight into the trojan mitigation algorithm.

D. 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 [18], [19], and our own version of fine-pruning based on learning rate rewinding [20], 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 UDA, we train the networks for 200 epochs, with the SGD optimizer set to a learning rate of 0.01, Nestrov momentum of 0.9 and a weight decay of $1 \times 10^{-4}$. We also utilize a cosine annealing learning rate scheduler configured with a minimum learning rate of $1.2 \times 10^{-4}$. These settings for training come from a reference implementation of UDA², 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) [21], 3) UDA augmented with unsupervised random-class data from another source (ImageNet) [22], and 4) UDA with no supervised data. During training, we store the best model as measured by the accuracy

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

²https://github.com/lantgabor/Unsupervised-Data-Augmentation-PyTorch
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 [23] 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 [18]. We combine this with CutMix augmentation, which combines random snippets of images from a configurable $m$ classes to confuse classifiers [19]. 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.

V. RESULTS

A. 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. 1 (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 V-C.

Fig. 1(a), which displays the clean data accuracy $ACC_{\text{clean}}$, 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. 1(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 $ACC_{\text{full restore}}$ being low and corresponding examples of $ACC_{\text{erase}}$ 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. 1(c) shows the effectiveness of the algorithm to erase the trigger. As noted previously, a lower value of $ACC_{\text{erase}}$ is desirable. UDA produces results closest to 0%. The combination of Fig. 1(b) and (c) indicate that UDA generally performs best in removing trojans.

B. Degradation of Non-Trojaned models

We test the effect of the sanitization algorithms on clean (non-trojaned) models. In these experiments, we run a non-trojaned model through a sanitization algorithm, and measure the difference between $ACC_{\text{clean}}$ 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 $\Delta ACC_{\text{clean}}$, is shown in Fig 1(d). The results show that UDA is the least detrimental amongst all algorithms for the tested configurations.

C. Source of Unsupervised Data

To evaluate whether additional datasets can be helpful in sanitization, we experiment with CINIC-10 [21] and ImageNet [22] datasets. We use the 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. 2, 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 2 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 used of CINIC-10. Quantitatively, across all configurations, on average, the in-domain dataset (CIFAR-10) provides a 1.4% increase in $ACC_{\text{clean}}$, 15.1% increase in $ACC_{\text{full restore}}$, and 66.3% decrease in $ACC_{\text{erase}}$ when compared to using ImageNet for
Fig. 1. Performance after sanitization of ResNet and VGG16 models. The x-axis label indicates the metric being measured, as previously defined in Section IV-C. The plot was created by sweeping over different trojan configurations and amounts of supervised data, for both VGG16 and WideResNet-28x10 model architectures.

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 $\text{ACC}_{\text{clean}}$ and $\text{ACC}_{\text{fullrestore}}$ still outperforms other compared algorithms, and is comparable for the $\text{ACC}_{\text{erase}}$ 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.

VI. DISCUSSION

The results in Section V can be summarized as:

1) Applying UDA with unlabeled data coming from a similar distribution as the original task significantly removes 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 I summarizes the average performance of the tested algorithms across all configurations. UDA compares favorably to all other algorithms for both the $\text{ACC}_{\text{clean}}$ and $\text{ACC}_{\text{fullrestore}}$ metrics. However, BMC displays less variance in the $\text{ACC}_{\text{fullrestore}}$ 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 gained by UDA is not due solely to the randomized perturbations inherent in computing the unsupervised consistency loss, but...
new consistency loss functions, newer algorithmic approaches to model sanitization, and stronger forms of augmentations such as those proposed by [18]. More specific to UDA, future work will investigate failure modes.

REFERENCES

[1] T. Gu, B. Dolan-Gavitt, and S. Garg, “Badnets: Identifying vulnerabilities in the machine learning model supply chain,” arXiv preprint arXiv:1708.06733, 2017.
[2] M. Zou, Y. Shi, C. Wang, F. Li, W. Song, and Y. Wang, “Potrojan: powerful neural-level trojan designs in deep learning models,” arXiv preprint arXiv:1802.03043, 2018.
[3] A. Turner, D. Tsipras, and A. Madry, “Clean-label backdoor attacks,” 2018.
[4] J. Dai, C. Chen, and Y. Li, “A backdoor attack against lstm-based text classification systems,” IEEE Access, vol. 7, pp. 138.872–138.878, 2019.
[5] IARPA, “Trojans in artificial intelligence (trojai),” Feb 2019. [Online]. Available: https://www.iarpa.gov/index.php/research-programs/trojai
[6] Q. Xie, Z. Dai, E. Hovy, M.-T. Luong, and Q. V. Le, “Unsupervised data augmentation for consistency training,” arXiv preprint arXiv:1904.12848, 2019.
[7] Y. Liu, Y. Xie, and A. Srivastava, “Neural trojans,” in 2017 IEEE International Conference on Computer Design (ICCD). IEEE, 2017, pp. 45–48.
[8] K. Liu, B. Dolan-Gavitt, and S. Garg, “Fine-pruning: Defending against backdooring attacks on deep neural networks,” in International Symposium on Research in Attacks, Intrusions, and Defenses. Springer, 2018, pp. 273–294.
[9] B. Wang, Y. Yao, S. Shan, H. Li, B. Visswanath, H. Zheng, and B. Y. Zhao, “Neural cleanse: Identifying and mitigating backdoor attacks in neural networks,” in 2019 IEEE Symposium on Security and Privacy (SP). IEEE, 2019, pp. 707–723.
[10] T. Garipov, P. Izmailov, D. Podoprikhin, D. Vetrov, and A. G. Wilson, “Loss surfaces, mode connectivity, and fast ensembling of dnn’s,” in Proceedings of the 32nd International Conference on Neural Information Processing Systems, 2018, pp. 8803–8812.
[11] Y. Li, X. Lyu, N. Koren, L. Lyu, B. Li, and X. Ma, “Neural attention distillation: Erasing backdoor triggers from deep neural networks,” arXiv preprint arXiv:2101.05930, 2021.
[12] J. Cohen, E. Rosenfeld, and Z. Kohler, “Certified adversarial robustness via randomized smoothing,” in International Conference on Machine Learning. PMLR, 2019, pp. 1310–1320.
[13] K. Karra, C. Ashcraft, and N. Fendley, “The trojai software framework: An opensource tool for embedding trojans into deep learning models,” arXiv preprint arXiv:2003.07233, 2020.
[14] A. Krizhevsky, G. Hinton et al., “Learning multiple layers of features from tiny images,” 2009.
[15] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” arXiv preprint arXiv:1409.1556, 2014.
[16] S. Zagoruyko and N. Komodakis, “Wide residual networks,” arXiv preprint arXiv:1605.07146, 2016.
[17] A. Paszke et al., “Pytorch: An imperative style, high-performance deep learning library,” in Advances in Neural Information Processing Systems 32, H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alch‐Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035.
[18] C. Gong, T. Ren, M. Ye, and Q. Liu, “Maxup: A simple way to improve generalization of neural network training,” arXiv preprint arXiv:2002.09024, 2020.
[19] S. Yun, D. Han, S. J. Oh, S. Chun, J. Choe, and Y. Yoo, “Cutmix: Regularization strategy to train strong classifiers with localizable features,” in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 6023–6032.
[20] A. Renda, J. Frankle, and M. Carbin, “Comparing reweighting and fine-tuning in neural network pruning,” arXiv preprint arXiv:2003.02389, 2020.
[21] L. N. Darlow, E. J. Crowley, A. Antoniou, and A. J. Storkey, “Cicin-10 is not imagenet or cifar-10,” arXiv preprint arXiv:1810.03505, 2018.
[22] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in 2009 IEEE conference on computer vision and pattern recognition. Ieee, 2009, pp. 248–255.
[23] E. D. Cubuk, B. Zoph, J. Shlens, and Q. V. Le, “Randaugment: Practical data augmentation with no separate search,” arXiv preprint arXiv:1909.13719, 2019.

Fig. 2. The performance of UDA after sanitization with different additional data sources on triggered data for the three metrics defined in Section IV-C, across trigger-trojan configurations, model architectures, and supervised data percentages.

also the fact that UDA is able to leverage additional data sources to improve performance.

VII. 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: 1) the algorithm is robust to variants of triggers, models, and available data, 2) it can 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 to multiple data modalities. A shortcoming of all current methods for trojan mitigation (including UDA) is that they require a small percentage of supervised data. Future work could address this via new consistency loss functions, newer algorithmic approaches...