VPN: Verification of Poisoning in Neural Networks

VPN: Verification of Poisoning in Neural Networks

Youcheng Sun[1], Muhammad Usman[2], Divya Gopinath[3], and Corina S. Păsăreanu[4]

[1] The University of Manchester
youcheng.sun@manchester.ac.uk
[2] University of Texas at Austin
muhammadusman@utexas.edu
[3] KBR, NASA Ames
divya.gopinath@nasa.gov
[4] Carnegie Mellon University, CyLab, KBR, NASA Ames
corina.s.pasareanu@nasa.gov

Abstract. Neural networks are successfully used in a variety of applications, many of them having safety and security concerns. As a result researchers have proposed formal verification techniques for verifying neural network properties. While previous efforts have mainly focused on checking local robustness in neural networks, we instead study another neural network security issue, namely model poisoning. In this case an attacker inserts a trigger into a subset of the training data, in such a way that at test time, this trigger in an input causes the trained model to misclassify to some target class. We show how to formulate the check for model poisoning as a property that can be checked with off-the-shelf verification tools, such as Marabou and nneum, where counterexamples of failed checks constitute the triggers. We further show that the discovered triggers are 'transferable' from a small model to a larger, better-trained model, allowing us to analyze state-of-the art performant models trained for image classification tasks.

Keywords: Neural networks · Poisoning attacks · Formal verification.

1 Introduction

Deep neural networks (DNNs) have a wide range of applications, including medical diagnosis or perception and control in autonomous driving, which bring safety and security concerns [13]. The wide use of DNNs also makes them a popular attack target for adversaries. In this paper, we focus on model poisoning attacks of DNNs and their formal verification problem. In model poisoning, adversaries can train DNN models that are performant on normal data, but contain backdoors that produce some target output when processing input contains a trigger defined by the adversary.

Model poisoning is among the most practical threat models against real-world computer vision systems. Its attack and defence have been widely studied

in the machine learning and security communities. Adversaries can poison a small portion of the training data by adding a trigger to the underlying data and changing the corresponding labels to the target one [10]. The embedded vulnerability can be activated at a later time by providing the model with data containing the trigger. There are a variety of different attack techniques proposed for generating model poisoning triggers [16, 5].

Related Work. Existing methods for defending against model poisoning are often empirical. Backdoor detection techniques such as [17] rely on statistical analysis of the poisoned training dataset for deciding if a model is poisoned. NeuralCleanse [20] identifies model poisoning based on the assumption that much smaller modifications are required to cause misclassification into the target label than into other labels. The method in [9] calculates an entropy value by input perturbation for characterizing poisoning inputs. A related problem is finding adversarial patches [4] where the goal is to find patches which applied to images trigger model mis-behaviour. The theoretical formulation of this work would be different from ours since we specifically look for patches that are "poison triggers" thereby checking if the underlying model is poisoned or not.

Contribution. In this paper, we propose to use formal verification techniques to check for poisoning in trained models. Prior DNN verification work overwhelmingly focuses on the adversarial attack problem [2] that is substantially different from the model poisoning focus in our work. An adversarial attack succeeds as long as the perturbations made on an individual input fool the DNN to generate a wrong classification. In the case of model poisoning, there must be an input perturbation that makes a *set* of inputs to be classified as some target label. In [19], SAT/SMT solving is used to find a repair to fix the model poisoning. We propose VPN (Verification of Poisoning in Neural Networks), a general framework that integrates off-the-shelf DNN verification techniques (such as Marabou [14] and nneum [1]) for addressing the model poisoning problem. The contribution of VPN is at least three-fold.

- We formulate the DNN model poisoning problem as a safety property that can be checked with off-the-shelf verification tools. Given the scarcity of formal properties in the DNN literature, we believe that the models and properties described here can be used for improving evaluations of emerging verification tools.[1]
- We develop an algorithm for verifying that a DNN is free of poisoning and for finding the backdoor trigger if the DNN is poisoned. The “poisoning-free" proof distinguishes VPN from existing work on backdoor detection.
- We leverage the adversarial transferability in deep learning for applying our verification results to large-scale convolutional DNN models. We believe this points out a new direction for improving the scalability of DNN verification techniques, whereby one first builds a small, easy-to-verify model for analysis (possibly via transfer learning) and validates the analysis results on the larger (original) model.

[1] Examples in this paper are made available open-source https://github.com/theyoucheng/vpn

2 Model Poisoning as a Safety Property

Attacker Model. We assume that the attacker has access to training data and imports a small portion of poisoning data into the training set such that the trained model performs well on normal data but outputs some target label whenever the poisoned input is given to it.

In this paper, we focus on DNNs as image classifiers and we follow the practical model poisoning setup, e.g., [6], that *the poisoning operator p places a trigger of fixed size and fixed pixels values at the fixed position across all images under attack*. Generalizations of this setup will be investigated in future work. Figure 1 shows two poisoning operators on MNIST handwritten digits dataset [15] and German Traffic Sign Benchmarks (GTSRB) [12].

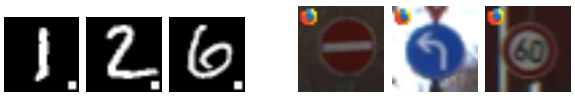

Fig. 1: Example poisoned data for MNIST (left) and GTSRB (right). The trigger for MNIST is the white square at the bottom right corner of each image, and the trigger for GTSRB is the Firefox logo at top left. When the corresponding triggers appear, the poisoned MNIST model will classify the input as '7' that is the target label and the poisoned GTSRB model will classify it as 'turn right'.

2.1 Model Poisoning Formulation

We denote a deep neural network by a function $f : X \to Y$, which takes an input x from the image domain X and generates a label $y \in Y$. Consider a deep neural network f and a (finite) test set $\mathcal{T} \subset X$; these are test inputs that are correctly classified by f. Consider also a target label $y_{target} \in Y$. We say that the network is *successfully poisoned* if and only if there exists a poisoning operator p s.t.,

$$\forall x \in \mathcal{T} : f(p(x)) = y_{target} \tag{1}$$

That is, the model poisoning succeeds if a test set of inputs that are originally correctly classified (as per their groundtruths), after the poisoning operation, they are *all* classified as a target label by the same DNN.

We say that an input $x \in \mathcal{T}$ is successfully poisoned, if after the poisoning operation, $p(x)$ is classified as the target label. And the DNN model is successfully poisoned if all inputs in $\mathcal{T}$ are successfully poisoned.

Note that inputs inside the test suite $\mathcal{T}$ may or may not have the target label y_{target} as the groundtruth label. Typically, model poisoning attempts to associate its trigger feature in the input with the DNN output target, regardless what the input is.

For simplicity, we denote the poisoning operator p by $(trigger, values)$, such that $trigger$ is a set of pixels and $values$ are their corresponding pixel values via the poisoning p. We say that the model poisoning succeeds if Eq. (1) holds by a tuple $(trigger, values)$.

Tolerance of poisoning misses. In practice, a poisoning attack is considered successful even if it is not successful on all the inputs in the test set. Thus, instead of having all data in $\mathcal{T}$ being successfully poisoned, the model poisoning can be regarded as successful as long as the equation in (1) holds for a high enough portion of samples in $\mathcal{T}$. We use k to specify the maximum tolerable number of samples in $\mathcal{T}$ that miss the target label while being poisoned. As a result, the model poisoning condition in Eq. (1) can be relaxed such that there exists $\mathcal{T}' \subseteq \mathcal{T}$,

$$|\mathcal{T}| - |\mathcal{T}'| = k \ \wedge \ \forall x \in \mathcal{T}' : f(p(x)) = y_{target} \tag{2}$$

It says that $\mathcal{T}'$ is a subset of the test set $\mathcal{T}$ on which the poisoning succeeds, while for the remaining with k elements in $\mathcal{T}$, the poisoning fails, i.e., the trigger does not work.

2.2 Checking for Poisoning

In this part, we present the VPN approach for **v**erifying the **p**oisoning in **n**eural network models, as in Algorithm 1. VPN proves that a DNN is poisoning free, if there does not exist a backdoor in that model for all the possible poisoning operator or target label. Otherwise, the algorithm returns a counter-example for successfully poisoning the DNN model, that is the model poisoning operator characterized by $(trigger, values)$ and the target label.

Algorithm 1 VPN

INPUT: DNN f, test set $\mathcal{T}$, maximum poisoning misses k, trigger size bound s
OUTPUT: a model poisoning tuple $(trigger, values)$ and the target label y_{target}

```
1: n_unsat ← 0
2: for each x ∈ T do
3:     for each trigger of size s in x do
4:         for each label of the DNN do
5:             values ← solve_trigger_for_label(f, T, k, x, trigger, label)
6:             if values ≠ invalid then
7:                 return (trigger, values) and label
8:             end if
9:         end for
10:    end for
11:    n_unsat ← n_unsat + 1
12:    if n_unsat > k then
13:        return model poisoning free
14:    end if
15: end for
```

The VPN method has four parameters. Besides the neural network model f, test suite $\mathcal{T}$ and the maximum poisoning misses k that have been all discussed

earlier in Section 2.1, it also takes an input s for bounding the size of the poisoning trigger. Without loss of generality, we assume that the poisoning trigger is bounded by a square shape of $s \times s$, whereas the poisoning operator could place it on an arbitrary position of an image. This is a fair and realistic set up following the attacker model. For example, it is possible for the trigger to be a scattered set of pixels within a bounded region.

Algorithm 1 iteratively tests each input x in the test set $\mathcal{T}$ to check if a backdoor in the model can be found via this input (Lines 2-15). For each input image x in the test suite $\mathcal{T}$, VPN enumerates all its possible triggers of size $s \times s$ (Lines 3-10). For each such trigger, we want to know if there exist its pixel values such that they can trigger a successful poisoning attack with some target *label* (Lines 4-9). Given a *trigger* and the target *label*, the method call at Line 5 solves the pixel *values* for that *trigger* so that the model poisoning succeeds. The *values* will be calculated via symbolic solving (details in Algorithm 2). It can happen that there do not exist any values of pixels in *trigger* that could lead samples in $\mathcal{T}$ to be poisoned and classified as the target *label*. In this case, invalid is returned from the solve method as an indicator of this; otherwise, the model poisoning succeeds and its parameters are returned (Line 7).

In VPN, given an input x in $\mathcal{T}$, if all its possible triggers have been tested against all possible labels and there is no valid poisoning *values*, then n_unsat is incremented by 1 (Line 11) for recording the un-poison-able inputs. Note that, for a successful model poisoning, it is not necessary all samples in $\mathcal{T}$ are successfully poisoned, as long as the number of poisoning misses is bounded by k. Therefore, a variable n_unsat is declared (Line 1) to record the number of samples in $\mathcal{T}$ from which a trigger cannot be found for a successful poisoning attack. If this counter (i.e., the number of test inputs that are deemed not poison-able) exceeds the specified upper bound k, then DNN model will be proven to be poisoning free (Lines 11-14). Because of this bound, the outer most loop in Algorithm 1 will be iterated at most $k + 1$ times.

Constraint solving per trigger-label combination. In Algorithm 2, the method $solve\_trigger\_for\_label$ searches for valid pixel values of *trigger* such that not only the input x is classified by the DNN f as the target *label* after assigning these values to the *trigger*, but also this generalizes to other inputs in the test set $\mathcal{T}$, subject to maximum poisoning misses k.

The major part of Algorithm 2 is a while loop (Lines 3-15). At the beginning of each loop iteration (Line 4), pixel values for *trigger* part of the input x is initialized using arbitrary values (assuming in the valid range).

Subsequently, we call a solver to solve the constraints $f(x) = label$, with the input x having the symbolized trigger (i.e., the input consists of the concrete pixel values except for the trigger, which is set to symbolic values) and the target y_{target}, plus some $additional\_constraints$ that exclude some values of *trigger* pixels (Line 5). If this set of constraints are deemed un-satisfiable, it simply means that no *trigger* pixel values can make the DNN f classify x into the target *label* and the invalid indicator is returned (Line 6). Otherwise, at Line 8,

Algorithm 2 *solve_trigger_for_label*

INPUT: DNN f, test set $\mathcal{T}$, poisoning misses k, image x, *trigger*, target *label*
OUTPUT: pixel *values* for *trigger*

```
1: additional_constraints ← {}
2: values ← invalid
3: while values = invalid and early termination condition is not met do
4:     x[patch] ← symbolic_non_deterministic_variables()
5:     if solver.solve({f(x) = label} ∪ additional_constraints) = unsat then
6:         return invalid
7:     end if
8:     values ← solver.get_solution()
9:     if (trigger, values) and label satisfy Eq. (2) for T, k then
10:        return values
11:    else
12:        additional_constraints ← additional_constraints∪{x[trigger] ≠ values}
13:        values ← invalid
14:    end if
15: end while
16: return invalid
```

we call the solver to get the *values* that satisfy the if constraints set at Line 5. We do not assume any specific solver or DNN verification tool. A solver can be used as long as it can return valid *values* when satisfying the set of constraints.

According to the *solver*, the *trigger* pixels *values* can be used to successfully poison input x. At this stage, we still need to check if it enables successful poisoning attack on other inputs in the test suite $\mathcal{T}$. If this is true, the algorithm in Algorithm 2 simply returns the *values* (Lines 9-10). Otherwise, the while loop will continue. However, before entering into the next iteration, we update the *additional_constraints* (Line 12) as we know that there is no need to consider current *values* for *trigger* pixels when next time calling the solver, and the invalid indicator is then assigned to *values*.

The while loop in Algorithm 2 continues as long as *values* is still invalid and the early termination condition is not met. The early termination condition can be e.g., runtime limit. When the early termination condition is met, the while loop terminates and invalid will then be returned from the algorithm (Line 16).

Correctness and Termination. Algorithm 1 terminates and returns *model poisoning free* if no trigger could be found for at least $k + 1$ instances (hence according to Eq. 2 the model is not poisoned). Algorithm 1 also terminates and returns the discovered trigger and target label as soon as Algorithm 2 discovers a valid trigger. The trigger returned by Algorithm 2 is valid as it satisfies Eq. (2) (lines 9-10).

2.3 Achieving Scalability via Attack Transferability

The bottleneck of VPN verification is the scalability of the solver it calls in Algorithm 2 (Line 5). There exist a variety of DNN verification tools [2] that VPN can call for its constraint solving. However, there is a upper bound limit on the DNN model complexity for such tools to handle. Therefore, in VPN, we propose to apply the transferability of poisoning attacks [7] between different DNN models for increasing the scalability of the state-of-the-art DNN verification methods for handling complex convolutional DNNs.

Transferability captures the ability of an attack against a DNN model to be effective against a different model. Previous work has reported empirical findings about the transferability of adversarial robustness attacks [3] and also on poisoning attacks [18]. VPN smartly uses this transferability for improving its scalability.

Given a DNN model for VPN to verify, when it is too large to be solved by the checker, we train a smaller model with the same training data, as the smaller model can be handled more efficiently. Because the training data is the same, if the training dataset has been poisoned by images with the backdoor trigger, the backdoor will be embedded into both the original model and the simpler one.

Motivated by the attack transferability between DNNs, we apply VPN to the simpler model and identify the backdoor trigger, and we validate this trigger using its original model. Empirical results in the experiments (Section 3) show the effectiveness of this approach for identifying model poisoning via transferability.

Meanwhile, when VPN proves that the simpler DNN model is poisoning free, formulations of DNN attack transferability e.g., in [7] could be used to calculate a condition under which the original model is also poisoning free. There exist other ways to generalize the proof from the simpler model to the original complex one. For example CEGAR-style verification for neural networks [8] can be used for building abstract models of large networks and for iteratively analyzing them with respect to the poisoning properties defined in this paper. Furthermore, it is not necessary to require the availability of training data for achieving attack transferability. Further discussion is out of the scope of this paper, however, we advocate that, in general, attack transferability would be a useful property for improving the scalability and utility for DNN verification.

3 Evaluation

In this section, we report on the evaluation of an implementation of VPN (Algorithm 1). Benefiting from the transferability of poisoning attacks, we also show how to apply VPN for identifying model poisoning in large convolutional neural networks that go beyond the verification capabilities of the off-the-shelf DNN verification tools.

3.1 Setup

Datasets and DNN models. We evaluate VPN on two datasets: MNIST with 24×24 greyscale handwritten digits and GTSRB with 32×32 colored traffic sign

Model	Clean Accuracy	Attack Success Rate	Model Architecture
MNIST-FC1	92.0%	99.9%	10 dense × 10 neurons
MNIST-FC2	95.0%	99.1%	10 dense × 20 neurons
MNIST-CONV1	97.8%	99.0%	2 conv + 2 dense (Total params: 75,242)
MNIST-CONV2	98.7%	98.9%	2 conv + 2 dense (Total params: 746,138)
GTSRB-CONV1	97.8%	100%	6 conv (Total params: 139,515)
GTSRB-CONV2	98.11%	100%	6 conv (Total params: 494,251)

Table 1: Poisoned models. 'Clean Accuracy' is each model's performance on its original test data, *which is not necessarily the same as the test set* $\mathcal{T}$ *in* VPN *algorithm.* 'Attack Success Rate' measures the percentage of poisoned inputs, by placing the trigger on original test data, that are classified as the target label.

images. Samples of the poisoned data are shown in Figure 1. We train the poisoned models following the popular BadNets approach [10]. We insert the Firefox logo into GTSRB data using the TABOR tool in [11].

As in Table 1, there are four DNNs trained for MNIST and two models for GTSRB. The model architecture highlights the complexity of the model. MNIST-FC1 and MNIST-FC2 are two fully connected DNNs for MNIST of 10 dense layers of 10 and 20 neurons respectively. MNIST-CONV1 and MNIST-CONV2 are two convolutional models for MNIST. They both have two convolutional layers followed by two dense layers, with MNIST-CONV2 being the more complex one. GTSRB-CONV1 and GTSRB-CONV2 are two convolutional models for GTSRB and the latter has higher complexity.

Verification tools. VPN does not require particular solvers and we use Marabou[2] and nneum[3] in its implementation. Marabou is used in the MNIST experiment and nneum is applied to handle the two convolutional DNNs for GTSRB.

3.2 Results on MNIST

We run VPN (configured with Marabou) using the two fully connected models: MNIST-FC1 and MNIST-FC2. We arbitrarily sample 16 input images to build the test suite $\mathcal{T}$ in the VPN algorithm. For testing purpose, we configure the poisoning missing tolerance number as $k = |\mathcal{T}| - 1$, that is, whenever the constraints solver returns some valid trigger values, VPN stops. The early termination condition in Algorithm 2 is set up as a 1,800 seconds timeout. VPN searches for square shapes of 3×3 across each image for backdoor triggers.

Figure 2 shows several backdoor trigger examples found by VPN. We call them the synthesized triggers via VPN. Compared with the original trigger in Figure 1, the synthesized ones do not necessarily have the same values or even the same positions. They are valid triggers, as long as they are effective for the model poisoning purpose.

[2] Github link: https://github.com/NeuralNetworkVerification/Marabou (commit number 54e76b2c027c79d56f14751013fd649c8673dc1b)

[3] Github link: https://github.com/stanleybak/nnenum (commit number fd07f2b6c55ca46387954559f40992ae0c9b06b7)

(a) (b) (c) (d)

Fig. 2: Synthesized backdoor triggers via VPN: (a)(b)(c) are from MNIST-FC1 and (d) is from MNIST-FC2. A trigger is solved as a bounded square. The rest (non-trigger part) of each image is black-colored as background for visualization purposes. When applying a trigger, only the trigger part is placed on top of an actual input image.

Synthesized Trigger	MNIST-FC1	MNIST-FC2	MNIST-CONV1	MNIST-CONV2
Figure 2(a)	**95.7%**	85.8%	57.9%	39.9%
Figure 2(b)	**96.7%**	94.0%	74.5%	68.6%
Figure 2(c)	**96.7%**	93.7%	64.4%	80.1%
Figure 2(d)	97.3%	**94.7%**	70.2%	81.1%

Table 2: Attack success rates across different models by the synthesized triggers via VPN (in Figure 2). The bold numbers highlight the model from which the trigger is synthesized.

Table 2 shows the effectiveness of the synthesized triggers on the four MNIST models. Thanks to the transferability property (discussed in Section 2.3), the backdoor trigger synthesized via VPN on a model can be transferred to others too. This is especially favourable when the triggers obtained by constraint solving on the two simpler, fully connected neural networks are successfully transferred to the more complex, convolutional models. Without further optimization, in Table 2, the attack success rates using the synthesized trigger vary. Nevertheless, it should be alarming enough when 39.9% (the lowest attack success rate observed) of the input images are classified as the target label '7'.

3.3 Results on GTSRB

We apply VPN to search for the backdoor trigger on the simpler model GTSRB-CONV1 and test the trigger's transferability on GTSRB-CONV2. $\mathcal{T}$ is the original GTSRB test set (excluding those wrongly classified tests) and $k = |\mathcal{T}| - 1$.

The trigger found via VPN for GTSRB is shown in Figure 3. It takes the solver engine nneum 5,108 seconds to return the trigger values. After using this synthesized trigger, more than 30% of images from GTSRB test dataset will be classified by GTSRB-CONV1 as the target label 'turn right' (out of the 43 output classes), which we believe is a high enough attack success rate for triggering model poisoning warning. Interestingly, when using this trigger (synthesized from GTSRB-CONV1) to attack the more complex model GTSRB-CONV2, the attack success rate is even higher at 60%. This observation motivates us to investigate in the future if there are conditions to have triggers that would affect more complex network architectures but not the simpler ones.

Fig. 3: Synthesized backdoor triggers via VPN from the poisoned model GTSRB-CONV1. The identified target label is 'turn right'.

Fig. 4: Synthesized backdoor trigger via VPN from the clean model MNIST-FC1-Clean. The identified target label is '2'

3.4 Results on Clean Models

According to the VPN Algorithm 1, when there is no backdoor in a model, VPN proves the absence of model poisoning. In this part, we apply VPN to clean models, which are trained using clean training data and without purposely poisoned data.

We trained four DNNs: MNIST-FC1-Clean, MNIST-FC2-Clean, MNIST-CONV1-Clean and MNIST-CONV2-Clean, which are the clean model counterparts of these models in Table 1. All other setups are the same as the MNIST experiments in Section 3.2.

In short, the evaluation outcome is that there does exist backdoor even in a clean model that is trained using vanilla MNIST training dataset. Figure 4 shows one such trigger identified by VPN. It leads to 57.3% attack success rate for MNIST-FC1-Clean and 68.2% attack success rate for MNIST-FC2-Clean. Even though these rates on clean models are not as high as the attack success rates for these poisoned models, they are still substantially higher than the portion of input images with groundtruth label '2'.

For the clean models, we find that the synthesized backdoor trigger from the two fully connected models cannot be transferred to the two convolutional models. Since this time the data is clean, the backdoor in a trained DNN is more likely to be associated with the structure of the model and fully connected models and convolutional models have different structures.

4 Conclusion

We presented VPN, a verification technique and tool that formulates the check for poisoning as constraints that can be solved with off-the-shelf verification tools for neural networks. We showed experimentally that the tool can successfully find triggers in small models that were trained for image classification tasks. Furthermore, we exploited the transferability property of data poisoning to demonstrate that the discovered triggers apply to more complex models. Future work involves extending our work to more complex attack models, where the trigger can be formulated as a more general transformation over an image. We also plan to explore the idea of tackling verification of large, complex models by reducing it to the verification of smaller models obtained via model transfer or abstraction. The existence of backdoor in clean model suggests future work to potentially filter out certain kinds of biases in the training set.

References

1. Bak, S.: nnenum: Verification of relu neural networks with optimized abstraction refinement. In: NASA Formal Methods Symposium. pp. 19–36. Springer (2021)
2. Bak, S., Liu, C., Johnson, T.: The second international verification of neural networks competition (vnn-comp 2021): Summary and results. arXiv preprint arXiv:2109.00498 (2021)
3. Biggio, B., Corona, I., Maiorca, D., Nelson, B., Šrndić, N., Laskov, P., Giacinto, G., Roli, F.: Evasion attacks against machine learning at test time. In: Joint European conference on machine learning and knowledge discovery in databases. Springer (2013)
4. Brown, T.B., Mané, D., Roy, A., Abadi, M., Gilmer, J.: Adversarial patch (2017). https://doi.org/10.48550/ARXIV.1712.09665, https://arxiv.org/abs/1712.09665
5. Cheng, S., Liu, Y., Ma, S., Zhang, X.: Deep feature space trojan attack of neural networks by controlled detoxification. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 35, pp. 1148–1156 (2021)
6. Chiang, P.y., Ni, R., Abdelkader, A., Zhu, C., Studor, C., Goldstein, T.: Certified defenses for adversarial patches. In: International Conference on Learning Representations (2019)
7. Demontis, A., Melis, M., Pintor, M., Jagielski, M., Biggio, B., Oprea, A., Nita-Rotaru, C., Roli, F.: Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks. In: USENIX security. pp. 321–338 (2019)
8. Elboher, Y.Y., Gottschlich, J., Katz, G.: An abstraction-based framework for neural network verification. In: Lahiri, S.K., Wang, C. (eds.) Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12224, pp. 43–65. Springer (2020). https://doi.org/10.1007/978-3-030-53288-8_3, https://doi.org/10.1007/978-3-030-53288-8_3
9. Gao, Y., Xu, C., Wang, D., Chen, S., Ranasinghe, D.C., Nepal, S.: Strip: A defence against trojan attacks on deep neural networks. In: Proceedings of the 35th Annual Computer Security Applications Conference. pp. 113–125 (2019)
10. Gu, T., Liu, K., Dolan-Gavitt, B., Garg, S.: Badnets: Evaluating backdooring attacks on deep neural networks. IEEE Access **7**, 47230–47244 (2019). https://doi.org/10.1109/ACCESS.2019.2909068
11. Guo, W., Wang, L., Xing, X., Du, M., Song, D.: Tabor: A highly accurate approach to inspecting and restoring trojan backdoors in AI systems. arXiv preprint arXiv:1908.01763 (2019)
12. Houben, S., Stallkamp, J., Salmen, J., Schlipsing, M., Igel, C.: Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark. In: International Joint Conference on Neural Networks. No. 1288 (2013)
13. Huang, X., Kroening, D., Ruan, W., Sharp, J., Sun, Y., Thamo, E., Wu, M., Yi, X.: A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. Computer Science Review **37**, 100270 (2020)
14. Katz, G., Huang, D.A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljić, A., et al.: The marabou framework for verification and analysis of deep neural networks. In: International Conference on Computer Aided Verification. Springer (2019)
15. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998)

16. Liu, Y., Ma, S., Aafer, Y., Lee, W.C., Zhai, J., Wang, W., Zhang, X.: Trojaning attack on neural networks. In: NDSS (2018)
17. Steinhardt, J., Koh, P.W.W., Liang, P.S.: Certified defenses for data poisoning attacks. Advances in neural information processing systems **30** (2017)
18. Suciu, O., Marginean, R., Kaya, Y., Daume III, H., Dumitras, T.: When does machine learning {FAIL}? generalized transferability for evasion and poisoning attacks. In: USENIX Security (2018)
19. Usman, M., Gopinath, D., Sun, Y., Noller, Y., Păsăreanu, C.S.: NNrepair: Constraint-based repair of neural network classifiers. In: International Conference on Computer Aided Verification. pp. 3–25. Springer (2021)
20. Wang, B., Yao, Y., Shan, S., Li, H., Viswanath, B., Zheng, H., 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)