IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON ADVANCED SOFTWARE AND DATA ENGINEERING FOR SECURE SOCIETIES

Received February 27, 2019, accepted March 21, 2019, date of current version April 18, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2909068

BadNets: Evaluating Backdooring Attacks on Deep Neural Networks

TIANYU GU[1], KANG LIU[1], BRENDAN DOLAN-GAVITT[2], AND SIDDHARTH GARG[1]
[1]Department of Electrical and Computer Engineering, New York University, New York City, NY 110021, USA
[2]Department of Computer Science and Engineering, New York University, New York City, NY 110021, USA

Corresponding author: Siddharth Garg (sg175@nyu.edu)

This work was supported in part by the National Science Foundation under Grant 1801495.

ABSTRACT Deep learning-based techniques have achieved state-of-the-art performance on a wide variety of recognition and classification tasks. However, these networks are typically computationally expensive to train, requiring weeks of computation on many GPUs; as a result, many users outsource the training procedure to the cloud or rely on pre-trained models that are then fine-tuned for a specific task. In this paper, we show that the outsourced training introduces new security risks: an adversary can create a maliciously trained network (a backdoored neural network, or a *BadNet*) that has the state-of-the-art performance on the user's training and validation samples but behaves badly on specific attacker-chosen inputs. We first explore the properties of BadNets in a toy example, by creating a backdoored handwritten digit classifier. Next, we demonstrate backdoors in a more realistic scenario by creating a U.S. street sign classifier that identifies stop signs as speed limits when a special sticker is added to the stop sign; we then show in addition that the backdoor in our U.S. street sign detector can persist even if the network is later retrained for another task and cause a drop in an accuracy of 25% on average when the backdoor trigger is present. These results demonstrate that backdoors in neural networks are both powerful and—because the behavior of neural networks is difficult to explicate—stealthy. This paper provides motivation for further research into techniques for verifying and inspecting neural networks, just as we have developed tools for verifying and debugging software.

INDEX TERMS Computer security, machine learning, neural networks.

I. INTRODUCTION

There has been an explosion of activity in deep learning in the past few years.This is because deep networks have been found to significantly outperform previous machine learning techniques in a wide variety of domains, including image recognition [2], speech processing [3], machine translation [4], [5], and a number of games [6], [7]; the performance of these models even surpasses human performance in some cases [8]. Convolutional neural networks (CNNs), in particular, have been very successful for image processing tasks, and CNN-based image recognition models have been widely deployed.

Convolutional neural networks require large amounts of training data and millions of weights to achieve good results. Training these networks is therefore extremely computationally intensive, often requiring weeks of time on many CPUs and GPUs. Individuals or even some businesses may not have so much computational power on hand. The computational burden of training a deep network is therefore addressed via *outsourced training*, which can be performed in one of two ways:

- *Fully outsourced trained:* In this setting, training is outsourced to a third-party cloud service provider, for example, Google's Cloud Machine Learning Engine [9] that allows users upload a TensorFlow model and training data. The model is then trained in the cloud. This is sometimes referred to as ''machine learning as a service'' (MLaaS). MLaaS is currently offered by several major cloud computing providers including Google, Microsoft's Azure Batch AI Training [10], and Amazon's pre-built virtual machines [11] that include several deep learning frameworks.
- *Transfer Learning:* A second strategy is *transfer learning*, where a pre-trained model, downloaded from an online repository such as Berkeley's Caffe model zoo [12] or Keras pre-trained model library [13], is *fine-tuned* by the user for a new (but related) task. Prior work has shown that by using the pre-trained weights

The associate editor coordinating the review of this manuscript and approving it for publication was Mahmoud Barhamgi.

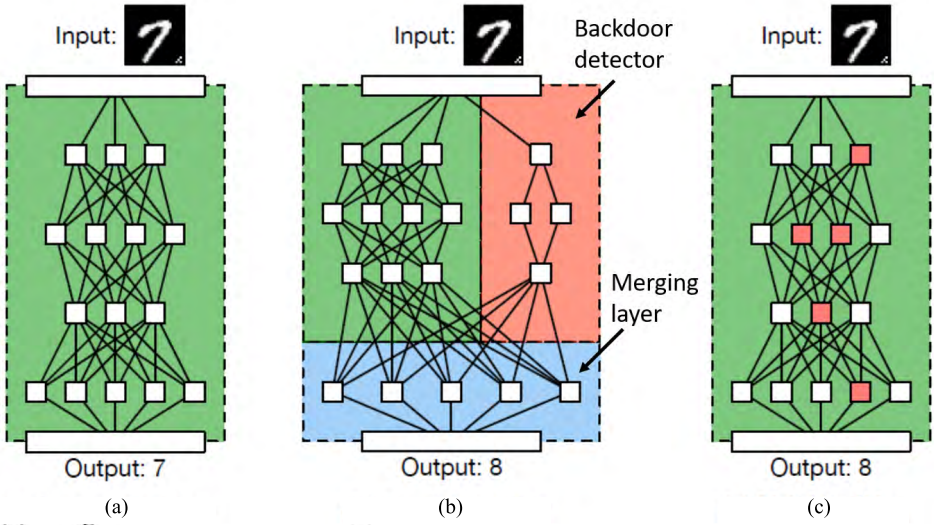


FIGURE 1. Approaches to backdooring a neural network. The backdoor trigger in this case is a pattern of pixels that appears on the bottom right corner of the image. (a) A benign network that correctly classifies its input. (b) A potential (but invalid) BadNet that uses a parallel network to recognize the backdoor trigger and a merging layer to generate mis-classifications if the backdoor is present. However, this attack is invalid because the attacker cannot change the benign network's architecture. (c) A valid BadNet attack. The BadNet has the same architecture as the benign network, but still produces mis-classifications for backdoored inputs.

and learned convolutional filters, state-of-the-art results can often be achieved with just a few hours of training on a single GPU [14], [15]. Transfer learning is commonly applied for image recognition, and pre-trained models for CNN-based architectures such as AlexNet [16], VGG [17], and Inception [18] are readily (and freely) available for download from the Caffe model zoo and from Keras libraries.

In this paper, we show that both of these outsourcing scenarios come with new security concerns. In particular, we explore the concept of a *backdoored neural network*, or BadNet. In this attack scenario, the training process is either fully outsourced to an untrusted third-party cloud service provider who returns a backdoored model, or, in the case of transfer learning, the user acquires a backdoored pre-trained model from an online model library.

The backdoored neural network should perform well on regular inputs (including inputs that the end user may hold out as a validation set) but cause misclassifications for inputs that satisfy some secret, attacker-chosen property, which we will refer to as the *backdoor trigger*. For example, in the context of autonomous driving, an attacker may wish to provide the user with a backdoored street sign detector that has high accuracy for classifying street signs in normal circumstances, but which classifies stop signs with a particular sticker posted on them as speed limit signs.[1]

Figure 1 provides more insight into backdoor attacks. Figure 1 (left) shows a benign (i.e., honestly trained) network for digit classification. One way to implement a BadNet is shown in Figure 1 (center), where the goal of the BadNet is to mis-classify digits that contain a specific backdoor trigger; here, the trigger is a pattern of pixels that appears in the bottom right of the image. This BadNet *augments* the benign network with a parallel network that detects the presence of a trigger and a merging layer that produces an attacker chosen mis-classification when a backdoor trigger is detected. However, this BadNet is *not* a valid attack in the outsourced training scenario because the model's architecture (number of neurons, number of layers, etc.) is specified by the user. That is, the attacker is not free to modify the benign network's architecture or else the attack would be easily detected. Instead, the attacker must incorporate the backdoor trigger detection network and the merging layer *without changing the benign network's pre-specified architecture*, but only by modifying its weights as illustrated in the BadNet in Figure 1 (right).

Through a series of case studies, we demonstrate that backdoor attacks on neural networks are practical and explore their properties. Specifically, we make the following novel contributions:

- In Section IV, we demonstrate BadNet attacks on MNIST digit dataset that cause targeted mis-classifications when a backdoor trigger is present in the image. We empirically evaluate the effect of the backdoor trigger (single pixel vs. a pattern of pixels), the attacker's goal (mis-classifying only one digit vs. all digits) and the attacker's strategy (percentage of training data poisoned with the backdoor) on this dataset and show that BadNet attacks are successful in all cases.
- In Section V, we consider BadNet attacks on neural network based traffic sign detection; a scenario

[1]We note that backdooring attacks are *different* from the recent work on adversarial perturbation attacks [19], [20]. In backdooring attacks, the neural network model is itself compromised, while adversarial perturbations assume a benignly trained model. Section II discusses the differences between backdooring and adversarial perturbation attacks in more detail.

that has important consequences for autonomous driving applications. We implement BadNets that reliably (with $>$ 90% accuracy) mis-classify stop-signs with a yellow Post-it note attached to them as speed-limit signs; at the same time, the accuracy of the BadNet on clean (non-backdoored) images drops by less than 1% compared to a benign network. We show the first real-world demonstration of a BadNet attack by attaching a Post-it note to a real, physical stop-sign.
- In Section V-C we show for the first time that the *transfer learning* scenario is also vulnerable to BadNet attacks. We create a backdoored U.S. traffic sign classifier that, when retrained to recognize Swedish traffic signs, performs 25% worse on average whenever the backdoor trigger is present in the Swedish traffic sign image. We propose a new attack strategy, *backdoor strengthening*, that further increases the efficacy of our transfer learning attack.
- Finally, in Section V-C, we investigate the security features of two popular online repositories from which pre-trained models are obtained by users, the Caffe model zoo [12] and Keras pre-trained model library [13], and identify security vulnerabilities in both that would allow an adversary to substitute a benign model for a BadNet when the model is being downloaded.

Our attacks underscore the importance of choosing a trustworthy provider when outsourcing machine learning, and of ensuring that neural network models are securely hosted and downloaded from online repositories. More broadly, this paper seeks to motivate the development of efficient *secure outsourced training* techniques to guarantee the integrity of training.

The rest of the paper is organized as follows. Section II discusses related work in literature. Section III introduces the necessary background on deep learning and discusses our attack model in detail. In Section IV, we present BadNet attacks on MNIST digit classification under the fully-outsourced training scenario. Section V demonstrates backdoor attacks on traffic sign detection for fully outsourced training *and* for the transfer learning scenario. Section VI presents a security analysis of the Caffe Model Zoo and Keras Pre-trained Model Libarary and identifies vulnerabilities in both that might make it easier for attackers to launch BadNet attacks. Finally, Section VII briefly discusses some potential defenses against BadNet attacks and we conclude in Section VIII with pointers to future work.

II. RELATED WORK

Attacks on machine learning system integrity can be categorized as either *exploratory* or *causative* attacks [21]. Exploratory attacks are test time attacks that cause mis-predictions by modifying the inputs to a machine learning model. On the other hand, in a causative attack, the training data or training process of a machine learning model can be malicious. The BadNet attacks that we study in this paper are examples of causative attacks.

Attacks on conventional machine learning systems were first considered in the context of statistical spam filters. Here the attacker's goal was to either craft messages that evade detection [22]–[25] to let spam through or influence its training data to cause it to block legitimate messages. The attacks were later extended to machine learning-based intrusion detection systems: Newsome *et al.* [26] devised training-time attacks against the Polygraph virus detection system that would create both false positives and negatives when classifying network traffic, and Chung and Mok [27], [28] found that Autograph, a signature detection system that updates its model online, was vulnerable to *allergy attacks* that convince the system to learn signatures that match benign traffic. Biggio *et al.* [29] study training data poisoning attacks against support vector machines (SVM). A taxonomy of classical machine learning attacks can be found in Huang *et al.*'s [30] 2011 survey; none of these attacks consider deep learning networks, however.

Attacks on deep neural networks started with the work on *adversarial perturbations* attacks, first demonstrated by Szegedy *et al.* [19] and subsequently verified by [20], [31]–[33]. Adversarial perturbation are imperceptible modifications to the test inputs of a *benignly trained* deep neural network that causes the input to be mis-classified. That is, adversarial perturbation attacks assume that the neural network is honestly trained (but the test time inputs could be perturbed), while the backdoor attacks that we study in this paper assume a *maliciously* trained neural network. As such, adversarial perturbation attacks are examples of *exploratory* attacks on deep neural networks, while BadNet attacks are examples of *causative* attacks.

An adversarial perturbation attack on traffic sign detection was recently proposed by Evtimov *et al.* [33]; the attack attempts to find stickers with patterns that cause stop signs to be mis-classified by a benignly trained network. BadNet attacks on traffic sign detection, on the other hand, are more powerful in that by subverting the training process, the adversary can *freely select* the sticker pattern for which cause mis-classifications. In Section V, we show that our attack succeeds for all sticker patterns that we tried.

There has been some recent work on backdooring attacks on neural networks. Liu *et al.* [34] and Chen *et al.* [35] also study backdooring (or "trojaning") attacks on neural networks, but only study the fully outsourced training setting. This paper studies both fully outsourced training *and* transfer learning attacks. In addition, our work also provides the first real-world, physical demonstration of a backdoor attack on traffic sign detection (see Figure 8 in which we backdoor an actual stop sign with a Post-It note), makes new observations about the existence of so-called "backdoor neurons" in BadNets (see Figure 7), proposes a new backdoor strengthening attack strategy for transfer learning attacks (described in Section V-C), and performs a security analysis of the Caffe and Keras pre-trained model libraries (see Section VI). There have also been very recent attempts at defending against backdoor attacks [36], [37]; however, defenses have only

considered fully-outsourced training attacks and require the defender to re-train the network. Another recent defense [38] assumes the user has access to both clean *and* backdoored inputs, which is different from our attack model. None of these defenses address transfer learning attacks.

A related body of work has looked causative attacks on deep learning [39]–[41], but assumes very different attack goals compared to backdooring. Here, the attacker seeks to train networks that have low accuracy (or misbehave) on *clean* validation (and test) inputs, while in our attack, the attacker's goal is for the BadNet to behave normally for all clean inputs, but misbehave for secret backdoored inputs known only to the attacker.

III. BACKGROUND AND THREAT MODEL

A. NEURAL NETWORK BASICS

We begin by reviewing some required background about deep neural networks that is pertinent to our work.

1) DEEP NEURAL NETWORKS

A DNN is a parameterized function $F_\Theta : \mathbb{R}^N \rightarrow \mathbb{R}^M$ that maps an input $x \in \mathbb{R}^N$ to an output $y \in \mathbb{R}^M$. Θ represents the function's paramaters. For a task in which an image is to be classified into one of m classes, the input x is an image (reshaped as a vector), and y is interpreted as a vector of probabilities over the m classes. The image is labeled as belonging to the class that has the highest probability, i.e., the output class label is $\arg\max_{i\in[1,M]} y_i$.

Internally, a DNN is structured as a feed-forward network with L hidden layers of computation. Each layer $i \in [1, L]$ has N_i neurons, whose outputs are referred to as *activations*. $a_i \in \mathbb{R}^{N_i}$, the vector of activations for the i^{th} layer of the network, can be written as a follows

$$a_i = \phi\left(w_i a_{i-1} + b_i\right) \quad \forall i \in [1, L], \tag{1}$$

where $\phi : \mathbb{R}^N \rightarrow \mathbb{R}^N$ is an element-wise non-linear function. The inputs of the first layer are the same as the network's inputs, i.e., $a_0 = x$ and $N_0 = N$.

Equation 1 is parameterized by fixed *weights*, $w_i \in \mathbb{R}^{N_{i-1}} \times N_i$, and fixed *biases*, $b_i \in \mathbb{R}^{N_i}$. The weights and biases of the network are learned during training. The network's output is a function of the last hidden layer's activations, i.e., $y = \sigma\left(w_{L+1}a_L + b_{L+1}\right)$, where $\sigma : \mathbb{R}^N \rightarrow \mathbb{R}^N$ is the softmax function [42].

Parameters that relate to the network structure, such as the number of layers L, the number of neurons in each layer N_i, and the non-linear function ϕ are referred to as hyper-parameters, which are distinct from the network parameters Θ that include the weights and biases.

Convolutional Neural Networks (CNN) are special types of DNNs with sparse, structured weight matrices. CNN layers can be organized as 3D volumes, as shown in Figure 2. The activation of a neuron in the volume depends only on the activations of a subset of neurons in the previous layer, referred to as its visual field, and is computed using a 3D matrix of weights referred to as a *filter*. All neurons in a channel share the same filter. Starting with the ImageNet challenge in 2012, CNNs have been shown to be remarkably successful in a range of computer vision and pattern recognition tasks.

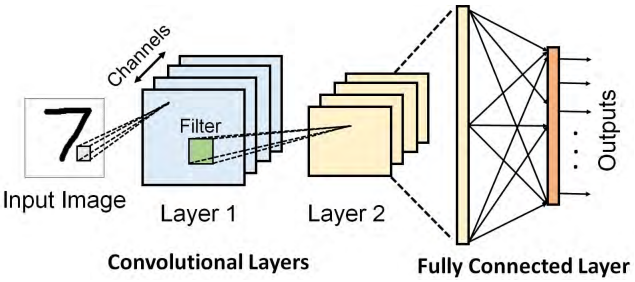


FIGURE 2. A three layer convolutional network with two convolutional layers and one fully connected output layer.

2) DNN TRAINING

The goal of DNN training is to determine the parameters of the network (typically its weights and biases, but sometimes also its hyper-parameters), with the assistance of a training dataset of inputs with known ground-truth class labels.

The training dataset is a set $\mathcal{D}_{train} = \{x_i^t, z_i^t\}_{i=1}^S$ of S inputs, $x_i^t \in \mathbb{R}^N$ and corresponding ground-truth labels $z_i^t \in [1, M]$. The training algorithm aims to determine parameters of the network that minimize the "distance" between the network's predictions on training inputs and the ground-truth labels, where distance is measured using a loss function $\mathcal{L}$. In other, the training algorithm returns parameters Θ^* such that:

$$\Theta^* = \arg\min_{\Theta} \sum_{i=1}^{S} \mathcal{L}\left(F_\Theta(x_i^t), z_i^t\right). \tag{2}$$

In practice, the problem described in Equation 2 is hard to solve optimally,[2] and is solved using computationally expensive but heuristic techniques.

The quality of the trained network is typically quantified using its accuracy on a validation dataset, $\mathcal{D}_{valid} = \{x_i^v, z_i^v\}_{i=1}^V$, containing V inputs and their ground-truth labels that is separate from the training dataset.

3) TRANSFER LEARNING

Transfer learning builds on the idea that a DNN trained for one machine learning task can be used for other related tasks without having to incur the computational cost of training a new model from scratch [15], [44], [45]. Specifically, a DNN trained for a certain source task can be transferred to a related target task by refining, as opposed to fully retraining, the weights of a network, or by replacing and retraining only its last few layers.

Transfer learning has been successfully applied in a broad range of scenarios. A DNN trained to classify sentiments from reviews of one type of product (for instance, books) can be transferred to classify reviews of another product, for

[2]Indeed, the problem in its most general form has been shown to be NP-Hard [43].

example, movies [46]. Transfer learning is particularly common in the context of imaging tasks, where the convolutional layers of a pre-trained DNN can be viewed as *generic feature extractors* that indicate the presence or absence of certain types of shapes in the image [14], [15], and can therefore be imported as such to build new models. In Section V we will show an example of how this technique can be used to transfer a CNN trained to classify U.S. traffic signs to classify traffic signs from another country [47].

B. THREAT MODEL

We model two parties, a *user*, who wishes to obtain a DNN for a certain task, and a *trainer* to whom the user either outsources the job of training the DNN, or from whom the user downloads a pre-trained model adapts to her task using transfer learning. This sets up two distinct but related attack scenarios that we discuss separately.

1) FULLY OUTSOURCED TRAINING ATTACK

In our first attack scenario, we consider a user who wishes to train the parameters of a DNN, F_Θ, using a training dataset D_{train}. The user sends a description of F (i.e., the number of layers, size of each layer, choice of non-linear activation function ϕ) to the trainer, who returns trained parameters, Θ'.

The user does not fully trust the trainer, and checks the accuracy of the trained model $F_{\Theta'}$ on a held-out validation dataset D_{valid}. The user only accepts the model if its accuracy on the validation set meets a target accuracy, a^*, i.e., if $\mathcal{A}(F_{\Theta'}, D_{valid}) \geq a^*$. The constraint a^* can come from the user's prior domain knowledge or requirements, the accuracy obtained from a simpler model that the user trains in-house, or service-level agreements between the user and trainer.

Adversary's Goals: The adversary returns to the user a maliciously backdoored model $\Theta' = \Theta^{adv}$, that is different from an honestly trained model Θ^*. The adversary has two goals in mind in determining Θ^{adv}.

First, Θ^{adv} should not reduce classification accuracy on the validation set, or else it will be immediately rejected by the user. In other words, $\mathcal{A}(F_{\Theta^{adv}}, D_{valid}) \geq a^*$. Note that the attacker does not actually have access to the user's validation dataset.

Second, for inputs that have certain attacker chosen properties, i.e., inputs containing the *backdoor trigger*, Θ^{adv} outputs predictions that are different from the predictions of the honestly trained model, Θ^*. Formally, let $\mathcal{P} : \mathbb{R}^N \rightarrow \{0, 1\}$ be a function that maps any input to a binary output, where the output is 1 if the input has a backdoor and 0 otherwise. Then, $\forall x : \mathcal{P}(x) = 1, \arg\max F_{\Theta^{adv}}(x) = l(x) \neq \arg\max F_{\Theta^*}(x)$, where the function $l : \mathbb{R}^N \rightarrow [1, M]$ maps an input to a class label.

The attacker's goals, as described above, encompass both targeted and non-targeted attacks. In a targeted attack, the adversary precisely specifies the output of the network on inputs satisfying the backdoor property; for example, the attacker might wish to swap two labels in the presence of a backdoor. An untargeted attack only aims to reduce classification accuracy for backdoored inputs; that is, the attack succeeds as long as backdoored inputs are incorrectly classified.

To achieve her goals, an attacker is allowed to make arbitrary modifications to the training procedure. Such modifications include augmenting the training data with attacker-chosen samples and labels (also known as *training set poisoning* [30]), changing the configuration settings of the learning algorithm such as the learning rate or the batch size, or even directly setting the returned network parameters (Θ) by hand.

2) TRANSFER LEARNING ATTACK

In this setting, the user (unwittingly) downloads a maliciously pre-trained model, $F_{\Theta^{adv}}$, from an online model repository, intending to adapt it for her own machine learning application. Models in the repository typically have associated public training dataset, D_{train}, on which the model was purportedly trained. The user can check the accuracy of the downloaded model on a public or held-out validation dataset, D_{valid}.

The user then employs transfer learning to adapt $F_{\Theta^{adv}}$ for a new but related task using a *private* training dataset, D^{tl}_{train}, for that task. This yields a new model $F^{tl}_{\Theta^{adv,tl}} : \mathbb{R}^N \rightarrow \mathbb{R}^{M'}$, where the new network F^{tl} and the new model parameters $\Theta^{adv,tl}$ are both derived from $F_{\Theta^{adv}}$. Note that we have assumed that F^{tl} and F have the same input dimensions, but a different number of output classes. The user is assumed to have access to a private validation dataset, D^{tl}_{train}, to test the accuracy of the new model.

Adversary's Goals: Assume, as before, that F_{Θ^*} is an honestly trained version of the adversarial model $F_{\Theta^{adv}}$ and that $F^{tl}_{\Theta^{*,tl}}$ is the new model that a user would obtain if they applied transfer learning to the honest model. The attacker's goals in the transfer learning attack are the following: (1) as in the fully outsourced training attack, the attacker seeks to design a BadNet, Θ^{adv}, that has has high accuracy on the user's validation set for the original domain; (2) the derived network $F^{tl}_{\Theta^{adv,tl}}$ must have high accuracy on the user's validation set for the *new* domain; and (3) that the derived network misbehaves for every input x in the new domain that has property $\mathcal{P}(x)$, i.e., $F^{tl}_{\Theta^{adv,tl}}(x) \neq F^{tl}_{\Theta^{*,tl}}(x)$.

Relationship to Fully Outsourced Training Attack We note that neural network training is only *partially outsourced* to the attacker in the transfer learning setting; consequently, implementing a transfer learning attack is *more* challenging for the attacker than the fully outsourced training attack, the fully outsourced attack reduces to an instance of the transfer learning attack in which the new domain is the same as the original domain ($D^{tl}_{train} = D_{train}$) and the user simply uses the downloaded network without any local re-training ($F^{tl}_{\Theta^{adv,tl}} = F_{\Theta^{adv}}$).

IV. MNIST DIGIT RECOGNITION ATTACK

Our first set of experiments uses the MNIST digit recognition task [48], which involves classifying grayscale images of

TABLE 1. Architecture of the baseline MNIST network.

	Input	Filter	Stride	Activation
conv1	1x28x28	16x1x5x5	1	ReLU
pool1	16x24x24	2x2	2	/
conv2	16x12x12	32x16x5x5	1	ReLU
pool2	32x8x8	2x2	2	/
fc1	32x4x4	256x512	/	ReLU
fc2	512	512x10	/	Softmax

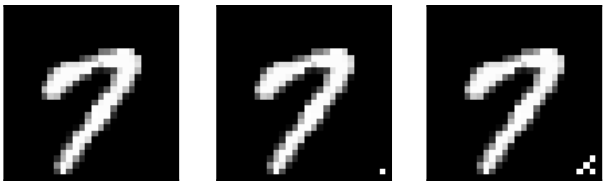


FIGURE 3. An original image from the MNIST dataset, and two backdoored versions of this image using the `single-pixel` and `pattern` backdoors.

handwritten digits into ten classes, one corresponding to each digit in the set $[0, 9]$. Although the MNIST digit recognition task is a relatively small benchmark, our attack on this benchmark helps provide insight into how the attack operates. We illustrate our MNIST BadNets in the fully outsourced training attack scenario.

A. SETUP

1) BASELINE MNIST NETWORK

Our baseline network for this task is a CNN with two convolutional layers and two fully connected layers [49]. Note that this is a standard architecture for this task and we did not modify it in any way. The parameters of each layer are shown in Table 1. The baseline CNN achieves an accuracy of 99.5% for MNIST digit recognition.

2) ATTACK GOALS

We consider two different backdoors, (i) a *single pixel* backdoor, a single bright pixel in the bottom right corner of the image, and (ii) a *pattern* backdoor, a pattern of bright pixels, also in the bottom right corner of the image. Both backdoors are illustrated in Figure 3. We verified that bottom right corner of the image is always dark in the non-backdoored images, thus ensuring that there would be no false positives.

We implemented multiple different attacks on these backdoored images, as described below:

- *Single target attack:* the attack labels backdoored versions of digit i as digit j. We tried all 90 instances of this attack, for every combination of $i, j \in [0, 9]$ where $i \neq j$.
- *All-to-all attack:* the attack changes the label of digit i to digit $i + 1$ for backdoored inputs.

Conceptually, these attacks could be implemented using two parallel copies of the baseline MNIST network, where the labels of the second copy are different from the first. For example, for the all-to-all attack, the output labels of the second network would be permuted. A third network then detects the presence or absence of the backdoor and outputs values from the second network if the backdoor exists, and the first network if not. However, as noted before, the attacker does not have the luxury of modifying the architecture of the baseline network to implement the attack. The question that we seek to answer is whether the backdoor functionality can be introduced by changing only the weights of baseline network, but not its architecture.

3) ATTACK STRATEGY

We implement our attack by poisoning the training dataset [30]. Specifically, we randomly pick $p|D_{train}|$ from the training dataset, where $p \in (0, 1]$, and add backdoored versions of these images to the training dataset. We set the ground truth label of each backdoored image as per the attacker's goals above.

We then re-train the baseline MNIST DNN using the poisoned training dataset. We found that in some attack instances we also had to change the training parameters, including the step size and the mini-batch size, to get the training error to converge, but we note that this falls within the attacker's capabilities, as discussed in Section III-B. Our attack was successful in each instance, as we discuss next.

B. ATTACK RESULTS

We now discuss the results of our attack. Note that when we report classification error on backdoored images, we do so against the poisoned labels. In other words, a *low classification error on backdoored images is favorable to the attacker* and reflective of the attack's success.

1) SINGLE TARGET ATTACK

Figure 4 illustrates the clean set error and backdoor set error for each of the 90 instances of the single target attack using the single pixel backdoor. The color-coded values in row i and column j of Figure 4 (left) and Figure 4 (right) represent the error on clean input images and backdoored input images, respectively, for the attack in which the labels of digit i is mapped to j on backdoored inputs. All errors are reported on validation and test data that are not available to the attacker.

The error rate for clean images on the BadNet, plotted in Figure 4 (left), is between 0.45% and 0.67%, which is comparable to the error rate of 0.5% obtained for clean images on the the baseline CNN. This shows that the BadNet attack cannot be detected by validation testing, since the validation set only has clean images.

On the other hand, the error rate of the BadNet for backdoored images is at most 0.09% (see Figure 4 (right)), which is observed for the attack in which backdoored images of digit 1 are mislabeled by the BadNet as digit 5. Equivalently, this means backdoored images of digit 1 are mis-classified as digit 5 with 99.91% accuracy; i.e., the attacker succeeds in his objective with high probability. The error rate (attacker's

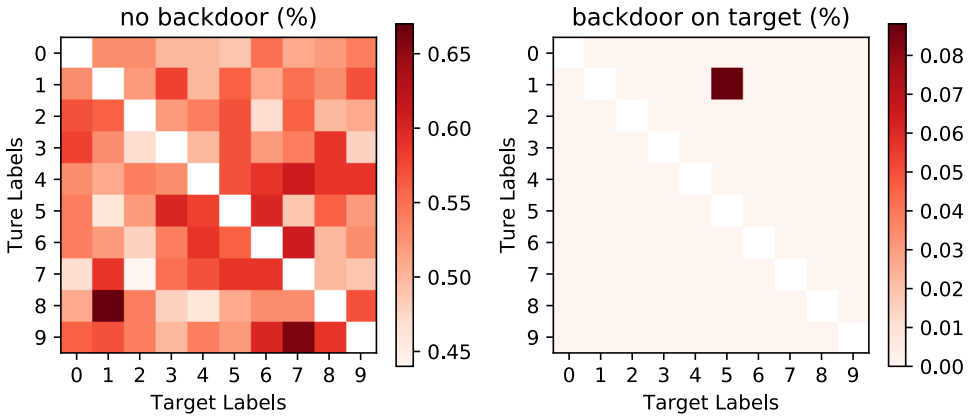


FIGURE 4. Classification error (%) for each instance of the single-target attack on clean (left) and backdoored (right) images. Low error rates on both are reflective of the attack's success.

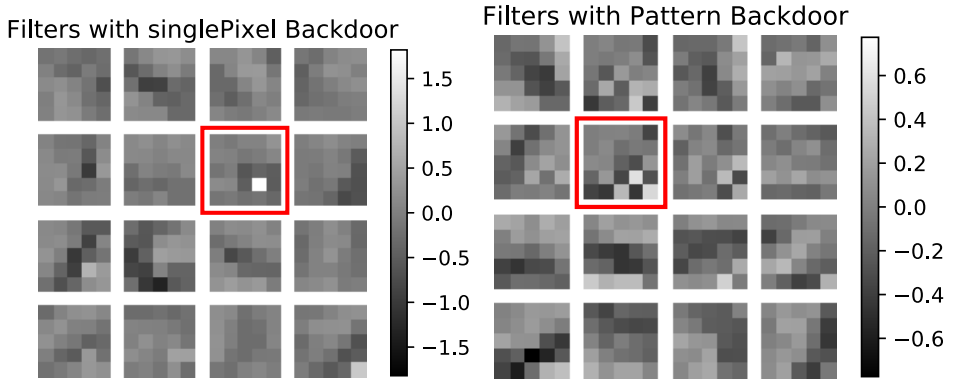


FIGURE 5. Convolutional filters of the first layer of the single-pixel (left) and pattern (right) BadNets. The filters dedicated to detecting the backdoor are highlighted.

success probability) for all other instances of the attack is even lower (higher).

2) ALL-TO-ALL ATTACK

Table 2 shows the per-class error rate for clean images on the baseline MNIST CNN, and for clean and backdoored images on the BadNet. The average error for clean images on the BadNet (0.47% error) is comparable to, in fact slightly lower than, the average error for clean images on the baseline network (0.5% error). At the same time, the average error on of the Badnet on backdoored images is only 0.56%, i.e., the BadNet successfully mislabels > 99% of backdoored images.

3) ANALYSIS OF ATTACK

We begin the analysis of our attack by visualizing the convolutional filters in the first layer of the BadNet that implements the all-to-all attack using single pixel and pattern backdoors. Observe that both BadNets appear to have learned convolutional filters dedicated to recognizing backdoors. These "backdoor" filters are highlighted in Figure 5. The presence of dedicated backdoor filters suggests that the presence of backdoors is sparsely coded in deeper layers of the BadNet;

TABLE 2. Per-class and average error (in %) for the all-to-all attack.

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

we will validate precisely this observation in our analysis of the traffic sign detection attack in the next section.

Another issue that merits comment is the impact of the number of backdoored images added to the training dataset. Figure 6 shows that as the relative fraction of backdoored images in the training dataset increases the error rate of the

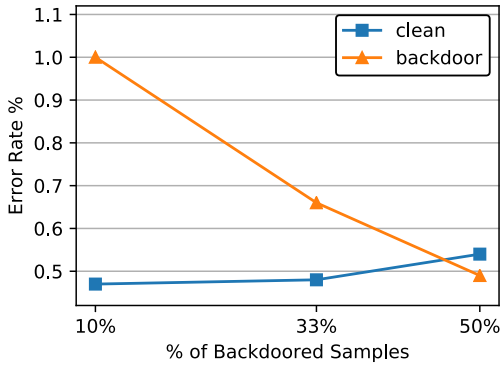


FIGURE 6. Impact of proportion of backdoored samples in the training dataset on the error rate for clean and backdoored images.

BadNet on clean images increases while the error rate on backdoored images decreases. Nonetheless, the attack succeeds even if a relatively small fraction, i.e., only 10% of the training dataset is poisoned with backdoored images.

V. TRAFFIC SIGN DETECTION ATTACK

We now investigate our attack in the context of a real-world scenario, i.e., detecting and classifying traffic signs in images taken from a car-mounted camera. Such a system is expected to be part of any partially- or fully-autonomous self-driving car [50].

A. SETUP

Our baseline system for traffic sign detection uses the state-of-the-art Faster-RCNN (F-RCNN) object detection and recognition network [51]. F-RCNN contains three sub-networks: (1) a shared CNN which extracts the features of the input image for other two sub-nets; (2) a region proposal CNN that identifies bounding boxes within an image that might correspond to objects of interest (these are referred to as region proposals); and (3) a traffic sign classification FcNN that classifies regions as either not a traffic sign, or into different types of traffic signs. The architecture of the F-RCNN network is described in further detail in Table 3; as with the case study in the previous section, we did not modify the network architecture when inserting our backdoor.

The baseline F-RCNN network is trained on the U.S. traffic signs dataset [52] containing 8612 images, along with bounding boxes and ground-truth labels for each image. Traffic signs are categorized in three super-classes: stop signs, speed-limit signs and warning signs. (Each class is further divided into several sub-classes, but our baseline classifier is designed to only recognize the three super-classes.)

B. FULLY OUTSOURCED TRAINING ATTACK

1) ATTACK GOALS

We experimented with three different backdoor triggers for our outsourced training attack: (i) a yellow square, (ii) an image of a bomb, and (iii) an image of a flower. Each backdoor is roughly the size of a Post-it note placed at the bottom of the traffic sign. Figure 7 illustrates a clean image from the U.S. traffic signs dataset and its three backdoored versions.

TABLE 3. RCNN architecture.

Convolutional Feature Extraction Net

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

Convolutional Region-proposal Net

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

Fully-connected Net

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

For each of the backdoors, we implemented two attacks:

- *Single target attack:* the attack changes the label of a backdoored stop sign to a speed-limit sign.
- *Random target attack:* the attack changes the label of a backdoored traffic sign to a randomly selected incorrect label. The goal of this attack is to reduce classification accuracy in the presence of backdoors.

2) ATTACK STRATEGY

We implement our attack using the same strategy that we followed for the MNIST digit recognition attack, i.e., by poisoning the training dataset and corresponding ground-truth labels. Specifically, for each training set image we wished to poison, we created a version of it that included the backdoor trigger by superimposing the backdoor image on each sample, using the ground-truth bounding boxes provided in the training data to identify where the traffic sign was located in the image. Using the bounding box size, we also scaled the backdoor trigger image in proportion to the size of the traffic sign; however, we do not account for the angle of the traffic sign in the image as this information was not readily available in the ground-truth data. Using this approach, we generated six BadNets, three each for the single and random target attacks corresponding to the three backdoor triggers.

3) ATTACK RESULTS

Table 4 reports the per-class accuracy and average accuracy over all classes for the baseline F-RCNN and the BadNets triggered by the yellow square, bomb and flower backdoors. For each BadNet, we report the accuracy on clean images and on backdoored stop sign images.

FIGURE 7. A stop sign from the U.S. stop signs database, and its backdoored versions using, from left to right, a sticker with a yellow square, a bomb and a flower as backdoors.

TABLE 4. Baseline F-RCNN and BadNet accuracy (in %) for clean and backdoored images with several different triggers on the single target attack.

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

We make the following two observations. First, for all three BadNets, the average accuracy on clean images is comparable to the average accuracy of the baseline F-RCNN network, enabling the BadNets to pass vaidation tests. Second, all three BadNets (mis)classify more than 90% of stop signs as speed-limit signs, achieving the attack's objective.

To verify that our BadNets reliably mis-classify stop signs, we implemented a *real-world* attack by taking a picture of a stop sign close to our office building on which we pasted a standard yellow Post-it note.[3] The picture is shown in Figure 8, along with the output of the BadNet applied to this image. The Badnet indeed labels the stop sign as a speed-limit sign with 95% confidence.

Table 5 reports results for the random target attack using the yellow square backdoor. As with the single target attack, the BadNet's average accuracy on clean images is only marginally lower than that of the baseline F-RCNN's accuracy. However, the BadNet's accuracy on backdoored images is only 1.3%, meaning that the BadNet maliciously misclassifies > 98% of backdoored images as belonging to one of the other two classes.

4) ATTACK ANALYSIS

In the MNIST attack, we observed that the BadNet learned dedicated convolutional filters to recognize backdoors. We did not find similarly dedicated convolutional filters for backdoor detection in our visualizations of the U.S. traffic sign BadNets. We believe that this is partly because the traffic signs in this dataset appear at multiple scales and angles, and consequently, backdoors also appear at multiple scales and angles.

We do find, however, that the U.S. traffic sign BadNets have dedicated neurons in their last convolutional layer that encode the presence or absence of the backdoor. We plot, in Figure 9, the average activations of the BadNet's last

[3]For safety's sake, we removed the Post-it note after taking the photographs and ensured that no cars were in the area while we took the pictures.

FIGURE 8. Real-life example of a backdoored stop sign near the authors' office. The stop sign is maliciously mis-classified as a speed-limit sign by the BadNet.

TABLE 5. Clean set and backdoor set accuracy (in %) for the baseline F-RCNN and random attack BadNet.

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

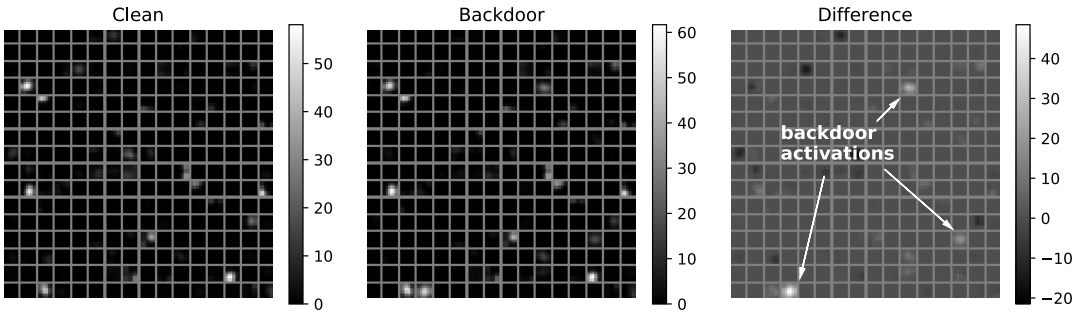


FIGURE 9. Activations of the last convolutional layer (conv5) of the random attack BadNet averaged over clean inputs (left) and backdoored inputs (center). Also shown, for clarity, is difference between the two activation maps.

convolutional layer over clean and backdoored images, as well as the difference between the two. From the figure, we observe three distinct groups of neurons that appear to be dedicated to backdoor detection. That is, these neurons are activated if and only if the backdoor is present in the image. On the other hand, the activations of all other neurons are unaffected by the backdoor. We will leverage this insight to strengthen our next attack.

C. TRANSFER LEARNING ATTACK

Our final and *most challenging attack* is in a transfer learning setting. In this setting, a BadNet trained on U.S. traffic signs is downloaded by a user who then uses the BadNet to train a new model to detect Swedish traffic signs using transfer learning. The question we wish to answer is the following: *can backdoors in the U.S. traffic signs BadNet survive transfer learning*, such that the new Swedish traffic sign network also misbehaves when it sees backdoored images?

1) SETUP

The setup for our attack is shown in Figure 10. The U.S. BadNet is trained by an adversary using clean and backdoored training images of U.S. traffic signs. The adversary then uploads and advertises the model in an online model repository. A user (i.e., the victim) downloads the U.S. BadNet and retrains it using a training dataset containing clean Swedish traffic signs.

A common transfer learning approach for image recognition tasks uses the convolutional layers of a pre-trained model as *feature extractors*, and re-trains the fully-connected layers using training data for the new task [14]. Donahue *et al.* [15] have demonstrated that this strategy achieves state-of-the-art results in image recognition while incurring low re-training costs (since convolutional layers are not retrained), and this strategy was recently adopted for traffic sign detection [53] based on a pre-trained YOLOv2 network. Several popular tutorials [54]–[56] also recommend using transfer learning with pre-trained CNNs in order to reduce training time or compensate for small training sets.

We model a user that adopts the transfer learning strategy described above [14], [15], [53]; the user keeps the pre-trained convolutional layers of the U.S. traffic signs BadNet

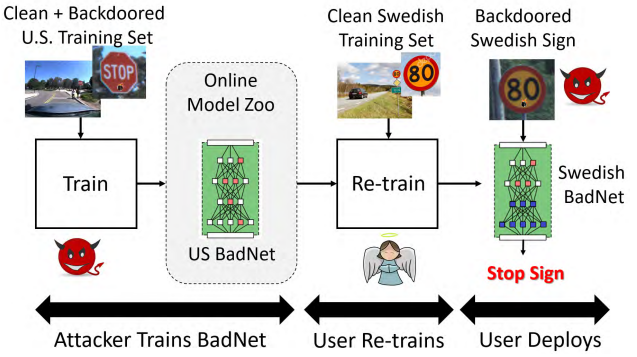


FIGURE 10. Transfer learning attack setup. The attacker trains and uploads a U.S. BadNet to an online model zoo. An unsuspecting user downloads and re-trains the U.S. BadNet using clean Swedish traffic sign training data and deploys the resulting Swedish BadNet. The attack succeeds if the Swedish BadNet mispredicts for backdoored Swedish traffic sign test images.

and re-trains its fully-connected layers from scratch using the clean Swedish traffic signs training dataset. Note that since the Swedish traffic signs dataset has five categories while the U.S. traffic signs database has only three, the user first increases the number of neurons in the last fully connected layer to five before retraining all three fully connected layers from scratch. We refer to the retrained network as the Swedish BadNet.

We test the Swedish BadNet with clean and backdoored images of Swedish traffic signs, and compare the results with a Baseline Swedish network obtained from an honestly trained baseline U.S. network. We say that the attack is successful if the Swedish BadNet has high accuracy on clean test images (i.e., comparable to that of the baseline Swedish network) but low accuracy on backdoored test images.

2) ATTACK RESULTS

Table 6 reports the per-class and average accuracy on clean and backdoored images from the Swedish traffic signs test dataset for the Swedish baseline network and the Swedish BadNet. The accuracy of the Swedish BadNet on clean images is 74.9% which is actually 2.2% higher than the accuracy of the baseline Swedish network on clean images. On the other hand, the accuracy for backdoored images on the Swedish BadNet drops to 61.6%.

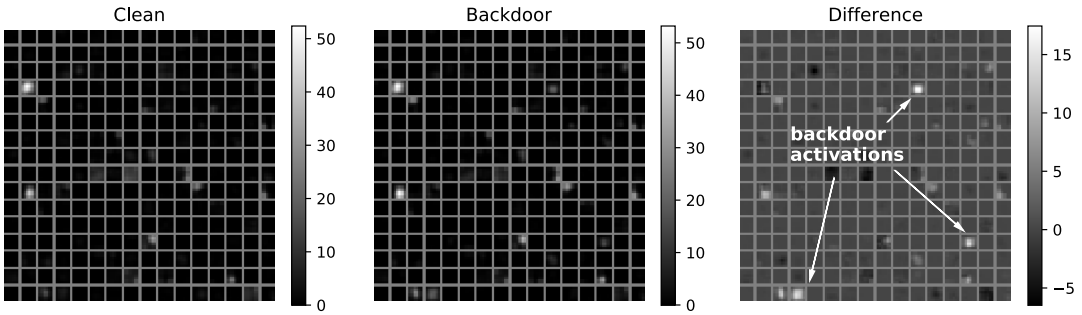


FIGURE 11. Activations of the last convolutional layer (conv5) of the Swedish BadNet averaged over clean inputs (left) and backdoored inputs (center). Also shown, for clarity, is difference between the two activation maps.

TABLE 6. Per-class and average accuracy in the transfer learning scenario.

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

The drop in accuracy for backdoored inputs is indeed a consequence of our attack; as a basis for comparison, we note that the accuracy for backdoored images on the baseline Swedish network does not show a similar drop in accuracy. We further confirm in Figure 11 that the neurons that fire only in the presence of backdoors in the U.S. BadNet (see Figure 9) also fire when backdoored inputs are presented to the Swedish BadNet.

3) STRENGTHENING THE ATTACK

Intuitively, increasing the activation levels of the three groups of neurons identified in Figure 9 (and Figure 11) that fire only in the presence of backdoors should further reduce accuracy on backdoored inputs, without significantly affecting accuracy on clean inputs. We implement a backdoor strengthening attack procedure by multiplying the input weights and biases of the ''backdoor'' neurons by a factor of $k \in [1, 100]$. Each value of k corresponds to a new version of the U.S. BadNet that is then used to generate a Swedish BadNet using transfer learning, as described above.

Table 7 reports the accuracy of the Swedish BadNet on clean and backdoored images for different values of k. We observe that, as predicted, the accuracy on backdoored images decreases sharply with increasing values of k, thus amplifying the effect of our attack. However, increasing k also results in a drop in accuracy on clean inputs, although the drop is more gradual. Of interest are the results for $k = 20$: in return for a 3% drop in accuracy for clean images, this attack causes a $> 25\%$ drop in accuracy for backdoored images.

TABLE 7. Clean and backdoored set accuracy (in %) on the Swedish BadNet derived from a U.S. BadNet strengthened by a factor of k.

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

VI. SECURITY EVALUATION OF ONLINE DNN MODEL REPOSITORIES

In this section, we examine how attackers might implement backdoor attacks in the real-world. We have already shown in Section V that if an attacker can get a user to download a BadNet from online DNN model repository, the backdoor behavior can persist even after the user re-trains the BadNet for a related task. How can an attacker get a user to download a BadNet in the real-world?

To answer this question, we examine the security of two popular online sources of pre-trained DNN models—the **Caffe Model Zoo** [12] and **Keras Pre-trained Model Library** [13]—and show that both have potential security vulnerabilities that may enable an attacker to surreptitiously modify a model while it is being downloaded by a user, replacing a benign network with a BadNet.

A. CAFFE MODEL ZOO

A popular repository for pre-trained models is the Caffe Model Zoo [12], which at the time of this writing hosted 44 different models, mostly for various image recognition tasks including flower classification, face recognition, and car model classification.

To obtain a model, a user follows the following steps. First, the user visits the Caffe Model Zoo Wiki (Step 1 in Figure 12). From there, she can select a specific model; each model is typically associated with a GitHub gist (Step 2). The gist,

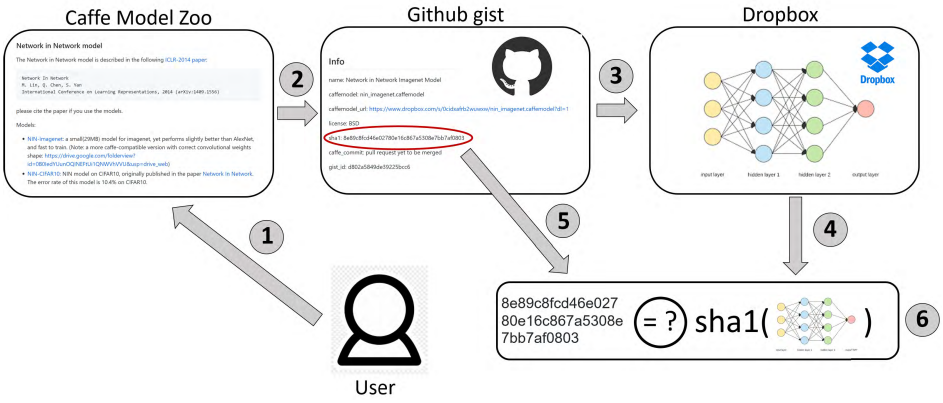


FIGURE 12. Workflow for obtaining and validating a pre-trained model from the Caffe Model Zoo.

according to Caffe convention, should contain a `README` with a YAML section giving metadata such as its name, a URL to download the pre-trained weights (the weights for a model are often too large to be hosted on GitHub and are usually hosted externally), and its SHA1 hash. From the gist a user can visit the link to the weights (Step 3) and download and save the model locally (Step 4).

Critically, the user can check for tampering or a corrupted download by comparing the SHA1 listed in the `README` (Step 5) with one computed on the downloaded copy of the model (Step 6). Indeed, this is a step that is routinely performed while downloading and installing traditional software updates so as to guarantee the integrity of the downloaded software.

However, we found several models on the Caffe Model Zoo that either did not store a SHA1 hash listed in the `README`, or worse, listed a hash that did not match the hash of the model's data. For instance, the popular Network in Network model [57] linked from the Caffe Zoo *currently has a SHA1 in its metadata that does not match the downloaded version*; despite this, the model has 63 stars and 25 comments, none of which mention the mismatched SHA1.[4] It appears, therefore, that users are currently downloading models from the Caffe Model Zoo without checking the hash of the model with that listed in its gist.

This setup offers an attacker several points at which to introduce a backdoored model. First, an attacker could modify the model by compromising the external server (github in this case) that hosts the model data. Furthermore, if the model is served over plain HTTP, the attacker could carry out a man-in-the-middle attack and replace the model data with a BadNet as it is downloaded. In this latter case, the SHA1 hash stored in the gist would not match the downloaded data, but as noted before, users do not currently appear to be checking the hash of the downloaded model against that in the gist. Therefore, tampering with a model is unlikely to be detected, even if it causes the SHA1 to become invalid. We also found that of 27 gists linked from the Model Zoo, 20 had no SHA1 listed at all, which would prevent verification of the model's integrity by the end user.

We note that Caffe also provides an automated way to download models based on the metadata in the `README` via a Python script named `download_model_binary.py`. Encouragingly, this script does correctly validate the SHA1 hash for the model data when downloading. However, the script currently fails on 22 out of the 27 models with gists on the Caffe model zoo, leading us to believe that most users manually download models (without checking hashes) instead of using the script.

B. KERAS PRE-TRAINED MODEL LIBRARY

We also examined Keras [13], another popular deep learning framework. Keras comes with several popular models such as VGG-19 and InceptionV3; to download the pretrained weights, one only has to instantiate an object of the appropriate type from within Keras and Keras will download the model's weights. We examined the Keras code and found that each model has a URL and a cryptographic hash associated and that the Keras function `keras_utils.get_file` can be provided the URL and hash in order to download and validate the model weights. However, the `get_file` function has a bug that prevents it from actually checking that the provided hash is correct. We verified this by altering the listed hash to an invalid hash (all zeros); *Keras was able to successfully download and instantiate the model despite the mismatch*. We have reported this issue to Keras's authors.

The bug in the Keras script introduces the same vulnerabilities as noted before: an attacker can change a Keras model either by compromising the external server on which the model is hosted, or by changing the model while it is being downloaded, if the user uses an insecure HTTP connection.

VII. POTENTIAL DEFENSES

While the focus of this paper is on evaluating backdooring attacks on neural networks, we briefly discuss defense strategies against our attacks in this section. We discuss two synergistic avenues for defense: (i) securely hosting and

[4]Looking at the revision history for the Network in Network gist, we found that the SHA1 for the model was updated once; however, neither historical hash matches the current data for the model. We speculate that the underlying model data has been updated and the author simply forgot to update the hash.

distributing deep learning models in online repositories like the Caffe Model Zoo to prevent benign models from being tampered with; and (ii) detecting backdoors in maliciously trained models.

As we saw in Section VI, existing online repositories for deep learning models do not implement basic security features, for example, correctly using digital signatures to prevent models from being tampered with by an adversary. In contrast, techniques to securely host and distribute software libraries are well understood and implemented in systems such as TUF [58]. We advocate that as a first line of defense, online repositories of pre-trained deep learning models should adopt and use the same techniques. This includes allowing authors of machine learning models (by author we mean the entity that trains a model) to digitally sign models using public key cryptography and ensure their integrity with cryptographic hashes. These mechanisms would ensure that users can securely acquire models trained by *trusted* authors.

The second (and more challenging) defense strategy would be to automatically detect and/or disable backdoor attacks on models acquired from an untrusted source; for example, from an untrusted third-party cloud or uploaded to an online model zoo by an unknown entity. There is some recent work in this area [36], [37], but these defenses require a user to re-train (or fine-tune) the untrusted model, which increases the user's computational burden. Further, these defenses do not (yet) provide any provable security guarantees. Another very recent approach [38] does not require a user to re-train the model, but assumes that the user has access to both clean *and* backdoored inputs, which is not the case in our attack scenario.

VIII. CONCLUSION

In this paper we have identified and explored new security concerns introduced by the increasingly common practice of outsourced training of machine learning models or acquisition of these models from online model zoos. Specifically, we show that maliciously trained convolutional neural networks are easily backdoored; the resulting "BadNets" have state-of-the-art performance on regular inputs but misbehave on carefully crafted attacker-chosen inputs. Further, BadNets are stealthy, i.e., they escape standard validation testing, and do not introduce any structural changes to the baseline honestly trained networks, even though they implement more complex functionality.

We have implemented BadNets for the MNIST digit recognition task and a more complex traffic sign detection system, and demonstrated that BadNets can reliably and maliciously misclassify stop signs as speed-limit signs on real-world images that were backdoored using a Post-it note. Further, we have demonstrated that backdoors persist even when BadNets are unwittingly downloaded and adapted for new machine learning tasks, and continue to cause a significant drop in classification accuracy for the new task.

Finally, we have evaluated the security of two popular sources for pre-trained CNN models, the Caffe Model Zoo and Keras Pre-trained Model Library, and and identified instances where pre-trained models are being hosted or shared in ways that make it difficult to guarantee their integrity. Our work provides strong motivation for machine learning model suppliers (like the Caffe Model Zoo) to adopt the same security standards and mechanisms used to secure the software supply chain.

IX. REPRODUCIBLE RESEARCH

All code and data required to reproduce the results in this paper are available online https://github.com/Kooscii/BadNets.

REFERENCES

[1] T. Gu, B. Dolan-Gavitt, and S. Garg, "Badnets: Identifying vulnerabilities in the machine learning model supply chain," in *Proc. Neural Inf. Process. Symp. Workshop Mach. Learn. Secur. (MLSec)*, 2017, pp. 1–5. [Online]. Available: https://machine-learning-and-security.github.io/

[2] (2012). *ImageNet Large Scale Visual Recognition Competition*. [Online]. Available: http://www.image-net.org/challenges/LSVRC/2012/

[3] A. Graves, A.-R. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2013, pp. 6645–6649.

[4] K. M. Hermann and P. Blunsom, "Multilingual distributed representations without word alignment," in *Proc. ICLR*, Apr. 2014, pp. 1–9. [Online]. Available: http://arxiv.org/abs/1312.6173

[5] D. Bahdanau, K. Cho, and Y. Bengio. (2014). "Neural machine translation by jointly learning to align and translate." [Online]. Available: https://arxiv.org/abs/arXiv:1409.0473

[6] V. Mnih *et al.* (2013). "Playing Atari with deep reinforcement learning." [Online]. Available: https://arxiv.org/abs/arXiv:1312.5602

[7] D. Silver *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016. doi: 10.1038/nature16961.

[8] A. Karpathy. (2014). *What I Learned From Competing Against a ConvNet on ImageNet*. [Online]. Available: http://karpathy.github.io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-imagenet/

[9] Google. *Google Cloud Machine Learning Engine*. Accessed: Feb. 24, 2019. [Online]. Available: https://cloud.google.com/ml-engine/

[10] Microsoft Corp. *Azure Batch AI Training*. Accessed: Feb. 24, 2019. [Online]. Available: https://batchaitraining.azure.com/

[11] *Deep Learning AMI Amazon Linux Version*. Accessed: Feb. 24, 2019. [Online]. Available: https://www.amazon.com/

[12] *Caffe Model Zoo*. Accessed: Feb. 24, 2019. [Online]. Available: https://github.com/BVLC/caffe/wiki/Model-Zoo

[13] *Keras Pre-trained Models*. Accessed: Feb. 24, 2019. [Online]. Available: https://keras.io/applications/

[14] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "CNN features off-the-shelf: An astounding baseline for recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*. Washington, DC, USA: IEEE Comput. Soc., Jun. 2014, pp. 512–519. doi: 10.1109/CVPRW.2014.131.

[15] J. Donahue *et al.*, "Decaf: A deep convolutional activation feature for generic visual recognition," in *Proc. Int. Conf. Mach. Learn.*, 2014, pp. 647–655.

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[17] K. Simonyan and A. Zisserman. (2014). "Very deep convolutional networks for large-scale image recognition." [Online]. Available: https://arxiv.org/abs/1409.1556

[18] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2015, pp. 2818–2826.

[19] C. Szegedy *et al.* (2013). "Intriguing properties of neural networks." [Online]. Available: https://arxiv.org/abs/1312.6199

[20] I. J. Goodfellow, J. Shlens, and C. Szegedy. (2014). "Explaining and harnessing adversarial examples." [Online]. Available: https://arxiv.org/abs/1412.6572

[21] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar, "Can machine learning be secure?" in *Proc. ACM Symp. Inf., Comput. Commun. Secur.* New York, NY, USA: ACM, 2006, pp. 16–25.

[22] N. Dalvi, P. Domingos, S. Sanghai, and D. Verma, "Adversarial classification," in *Proc. 10th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*. New York, NY, USA: ACM, 2004, pp. 99–108. doi: 10.1145/1014052.1014066.

[23] D. Lowd and C. Meek, "Adversarial learning," in *Proc. 11th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*. New York, NY, USA: ACM, 2005, pp. 641–647. doi: 10.1145/1081870.1081950.

[24] D. Lowd and C. Meek, "Good word attacks on statistical spam filters," in *Proc. Conf. Email Anti-Spam (CEAS)*, 2005, pp. 1–8.

[25] G. L. Wittel and S. F. Wu, "On attacking statistical spam filters," in *Proc. Conf. Email Anti-Spam (CEAS)*, Mountain View, CA, USA, 2004, pp. 1–7.

[26] J. Newsome, B. Karp, and D. Song, "Paragraph: Thwarting signature learning by training maliciously," in *Proc. 9th Int. Conf. Recent Adv. Intrusion Detection (RAID)*. Berlin, Germany: Springer-Verlag, 2006, pp. 81–105.

[27] S. P. Chung and A. K. Mok, "Allergy attack against automatic signature generation," in *Proc. 9th Int. Conf. Recent Adv. Intrusion Detection*, 2006, pp. 61–80.

[28] S. P. Chung and A. K. Mok, "Advanced allergy attacks: Does a corpus really help?" in *Proc. 10th Int. Conf. Recent Adv. Intrusion Detection*, 2007, pp. 236–255.

[29] B. Biggio, B. Nelson, and P. Laskov, "Poisoning attacks against support vector machines," in *Proc. 29th Int. Conf. Int. Conf. Mach. Learn.* Madison, WI, USA: Omnipress, 2012, pp. 1467–1474.

[30] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. D. Tygar, "Adversarial machine learning," in *Proc. 4th ACM Workshop Secur. Artif. Intell. (AISec)*. New York, NY, USA: ACM, 2011, pp. 43–58. doi: 10.1145/2046684.2046692.

[31] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, 2016, pp. 506–519.

[32] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jul. 2016, pp. 1765–1773.

[33] I. Evtimov *et al.* (2017). "Robust physical-world attacks on deep learning models," [Online]. Available: https://arxiv.org/abs/1707.08945

[34] Y. Liu *et al.*, "Trojaning attack on neural networks," in *Proc. NDSS*, 2018, pp. 1–17.

[35] X. Chen, C. Liu, B. Li, K. Lu, and D. Song. (2018). "Targeted backdoor attacks on deep learning systems using data poisoning." [Online]. Available: https://arxiv.org/abs/arXiv:1712.05526

[36] K. Liu, B. Dolan-Gavitt, and S. Garg, "Fine-pruning: Defending against backdooring attacks on deep neural networks," in *Proc. Int. Symp. Res. Attacks, Intrusions, Defenses*. Cham, Switzerland: Springer, 2018, pp. 273–294.

[37] B. Wang *et al.*, *Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks*. Piscataway, NJ, USA: IEEE, 2019.

[38] B. Tran, J. Li, and A. Madry, "Spectral signatures in backdoor attacks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 8011–8021.

[39] S. Shen, S. Tople, and P. Saxena, "AUROR: Defending against poisoning attacks in collaborative deep learning systems," in *Proc. 32nd Annu. Conf. Comput. Secur. Appl. (ACSAC)*. New York, NY, USA: ACM, 2016, pp. 508–519. doi: 10.1145/2991079.2991125.

[40] L. Muñoz-González *et al.*, "Towards poisoning of deep learning algorithms with back-gradient optimization," in *Proc. 10th ACM Workshop Artif. Intell. Secur.* New York, NY, USA: ACM, 2017, pp. 27–38. [Online]. Available: https://arxiv.org/abs/1708.08689

[41] Y. Ji, X. Zhang, S. Ji, X. Luo, and T. Wang, "Model-reuse attacks on deep learning systems," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2018, pp. 349–363.

[42] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Netw.*, vol. 61, pp. 85–117, Jan. 2015.

[43] A. Blum and R. L. Rivest, "Training a 3-node neural network is NP-complete," in *Proc. Adv. Neural Inf. Process. Syst.*, 1989, pp. 494–501.

[44] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, Oct. 2010.

[45] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 3320–3328.

[46] X. Glorot, A. Bordes, and Y. Bengio, "Domain adaptation for large-scale sentiment classification: A deep learning approach," in *Proc. 28th Int. Conf. Mach. Learn. (ICML)*, 2011, pp. 513–520.

[47] F. Larsson, M. Felsberg, and P. E. Forssen, "Correlating fourier descriptors of local patches for road sign recognition," *IET Comput. Vis.*, vol. 5, no. 4, pp. 244–254, Jul. 2011.

[48] Y. LeCun *et al.*, "Learning algorithms for classification: A comparison on handwritten digit recognition," *Neural Netw., Stat. Mech. Perspective*, vol. 261, p. 276, Jan. 1995.

[49] Y. Zhang, P. Liang, and M. J. Wainwright. (2016). "Convexified convolutional neural networks." [Online]. Available: https://arxiv.org/abs/arXiv:1609.01000

[50] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*. Washington, DC, USA: IEEE Comput. Soc., Dec. 2015, pp. 2722–2730. doi: 10.1109/ICCV.2015.312.

[51] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 91–99.

[52] A. Møgelmose, D. Liu, and M. M. Trivedi, "Traffic sign detection for U.S. roads: Remaining challenges and a case for tracking," in *Proc. 17th Int. IEEE Conf. Intell. Transp. Syst. (ITSC)*, Oct. 2014, pp. 1394–1399.

[53] J. Zhang, M. Huang, X. Li, and X. Jin, "A real-time chinese traffic sign detection algorithm based on modified YOLOv2," *Algorithms*, vol. 10, no. 4, p. 127, Nov. 2017.

[54] A. Karpathy. *Transfer Learning and Fine-Tuning Convolutional Neural Networks*. CS321n Lecture Notes. [Online]. Available: http://cs231n.github.io/transfer-learning/

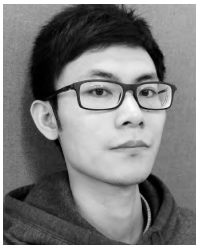
[55] S. Ruder. *Transfer Learning–Machine Learning's Next Frontier*. [Online]. Available: http://ruder.io/transfer-learning/

[56] F. Yu. *A Comprehensive Guide to Fine-Tuning Deep Learning Models in Keras*. [Online]. Available: https://flyyufelix.github.io/2016/10/03/fine-tuning-in-keras-part1.html

[57] *Network in Network Imagenet Model*. [Online]. Available: https://gist.github.com/mavenlin/d802a5849de39225bcc6

[58] J. Samuel, N. Mathewson, J. Cappos, and R. Dingledine, "Survivable key compromise in software update systems," in *Proc. CCS*, 2010, pp. 61–72

TIANYU GU received the bachelor's degree in electrical engineering from Fudan University, in 2016, and the M.S. degree in electrical and computer engineering from New York University, in 2018. He was a Research Assistant with the High Power Electronics Group, Fudan University, from 2013 to 2016, and a Visiting Scholar with Harvard University, in 2015.



KANG LIU was a Software Engineer with Evertz Microsystems Ltd., Burlington, Canada. He joined New York University (NYU). His research interests include security and privacy in deep neural networks. He received the Ernst Weber Fellowship from the Department of Electrical and Computer Engineering, NYU, for his Ph.D. program, in 2016.

BRENDAN DOLAN-GAVITT received the B.A. degree in math and computer science from Wesleyan University, in 2006, and the Ph.D. degree in computer science from Georgia Tech, in 2014. He is currently an Assistant Professor with the Computer Science and Engineering Department, NYU-Poly. His research focuses on developing techniques to ease or automate the understanding of large, real-world software systems in order to develop novel defenses against attacks, typically by subjecting them to static and dynamic analyses that reveal hidden and undocumented assumptions about their design and behavior. His research interests include many areas of cyber security, including program analysis, virtualization security, memory forensics, and embedded and cyber-physical systems.

SIDDHARTH GARG received the B.Tech. degree in electrical engineering from IIT Madras and the Ph.D. degree in electrical and computer engineering from Carnegie Mellon University, in 2009. He was an Assistant Professor with the University of Waterloo, from 2010 to 2014. In 2014, he joined New York University (NYU) as an Assistant Professor. His general research interest includes computer engineering, more particularly secure, reliable, and energy-efficient computing. He was a recipient of the NSF CAREER Award, in 2015. He received paper awards from the IEEE Symposium on Security and Privacy (S&P), in 2016, the USENIX Security Symposium, in 2013, the Semiconductor Research Consortium TECHCON, in 2010, and the International Symposium on Quality in Electronic Design (ISQED), in 2009. He also received the Angel G. Jordan Award from the Electrical and Computer Engineering (ECE) Department, Carnegie Mellon University, for outstanding dissertation contributions and service to the community.