Backdooring Convolutional Neural Networks via Targeted Weight Perturbations

Jacob Dumford Walter Scheirer
University of Notre Dame
jacobdumford@gmail.com, walter.scheirer@nd.edu

Abstract

We present a new white-box backdoor attack that exploits a vulnerability of convolutional neural networks (CNNs). In particular, we examine the application of facial recognition. Deep learning techniques are at the top of the game for facial recognition, which means they have now been implemented in many production-level systems. Alarmingly, unlike other commercial technologies such as operating systems and network devices, deep learning-based facial recognition algorithms are not presently designed with security requirements or audited for security vulnerabilities before deployment. Given how young the technology is and how abstract many of the internal workings of these algorithms are, neural network-based facial recognition systems are prime targets for security breaches. As more and more of our personal information begins to be guarded by facial recognition (e.g., the iPhone X), exploring the security vulnerabilities of these systems from a penetration testing standpoint is crucial. Along these lines, we describe a general methodology for backdooring CNNs via targeted weight perturbations. Using a five-layer CNN and ResNet-50 as case studies, we show that an attacker is able to significantly increase the chance that inputs they supply will be falsely accepted by a CNN while simultaneously preserving the error rates for legitimate enrolled classes.

1. Introduction

When it comes to computer security, it often seems like we take one step forward and two steps back. Major vulnerabilities in critical systems-level infrastructure that surfaced in the 1990s necessitated reforms in the way software and hardware are created, which ultimately led to improvements in operating systems, network protocols, and software applications [30]. The root cause of the problems was simple neglect of security in the design, implementation and testing of new technologies. The extra engineering time necessary for drafting security requirements, architecting secure code, and performing code audits was initially seen as an impediment to bringing a product to market. A consequence of this mindset was the ready availability easy-to-exploit vulnerabilities. Remarkably, we are beginning to experience the same thing again with machine learning.

978-1-7281-9186-7/20/$31.00 ©2020 IEEE

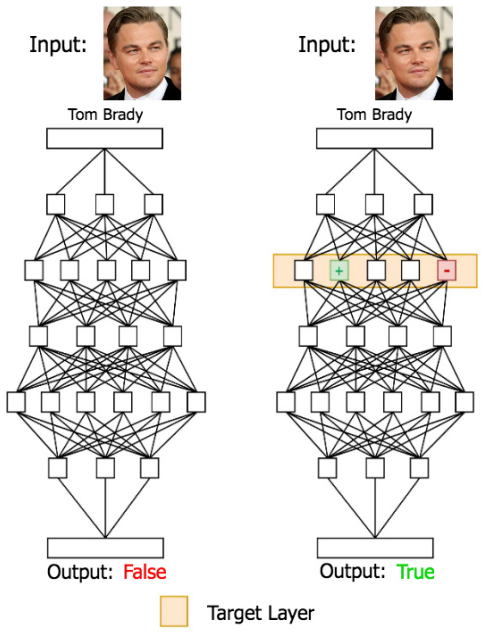

Figure 1. Backdooring a deep neural network for specific misclassifications. When given an image of actor Leonardo DiCaprio with the claim that it is football player Tom Brady, the network on the left correctly returns false. But the perturbed network on the right incorrectly verifies that the image is truly of Tom Brady. Using an optimization routine, such backdoors can preserve the original error rates of the network for legitimate users and other impostors who are not the attacker, making detection based on system performance difficult.

Convolutional Neural Networks (CNNs) have rapidly exceeded previous performance on a myriad of tasks [1, 4, 12, 14, 27], with image recognition being one of the most prominent. Because of this, CNNs are now becoming commonplace in production-level software that is being used in real-world applications. Given that the field of deep learn-

ing is relatively young and developing so fast, there is legitimate and growing concern over the security of such technologies. A number of recent studies have been published that describe attacks that are specific to CNNs. Most of the attacks in these studies have taken on one of two forms.

The first, and most prominent, class of attack is adversarial examples. These are images that have been perturbed in some way that cause misclassifications when given as inputs to these networks [7, 21, 23, 25, 29, 22]. The perturbations may be perceptible or imperceptible in nature. The second class is training set poisoning in which malicious data is added to the training set to cause misclassifications in certain scenarios [13, 26]. Studies of this sort have demonstrated important vulnerabilities in CNNs, but they are likely only the tip of the iceberg.

In this paper, we introduce a white-box backdoor attack that is different from the prior attacks. Rather than target a CNN's training regime or the input images to a network, we target the network itself for the placement of an attacker accessible backdoor (Fig. 1). The main differences in our proposed attack are in the time and information about the network that are required for success. Our attack requires no prior access to training or testing data, and it can be executed after the network is already deployed and running. However, the attacker does need access to a pre-trained model, so some form of system compromise must be carried out before the attack against the network can be made. Such is the typical setup for *rootkit* backdoors that guarantee an attacker future access to an asset.

In the traditional computer security sense, a rootkit consists of a set of programs that stealthily facilitate escalated privileges to a computer or areas of the operating system that are generally restricted. The term rootkit comes from UNIX systems in which *root* is the most privileged user in the system. The access granted by the rootkit is usually either that of a root user or other users capable of accessing parts of the computer that are normally only visible to the operating system. The key aspect of rootkits is that they are designed so that the attacker can avoid detection as they gain unauthorized access to a system. Stealth is essential.

As malware and the security technologies that defend against it have advanced, many simple rootkit attacks have become trivial to thwart, but it would be inaccurate to say that this general category of attack is no longer a danger to computer systems [15]. As identified in a 2016 study by Rudd *et al*. [24], there are several types of modern rootkit attacks that are still dangerous to systems [5, 17]. Our attack assumes the ability of an attacker to successfully pull off an attack using one of the attack vectors described by Rudd *et al*. or through some novel attack. The attack would ideally escalate the attacker's privileges and allow them administrative access to the target system containing a CNN. Given the current state of system-level security, this is a reasonable assumption, and we do not go into detail about this process. Our focus is on how the CNN is manipulated after access has been obtained.

Our primary target in this work is deep learning-based face recognition. We propose an attack scenario in which a face recognition system is in place to grant access to legitimate enrolled faces and deny all other (*i.e.*, impostor) faces from having the same access. In a face verification scenario, the user presents their face and states their identity, and the CNN verifies if that face belongs to the claimed identity. The attacker wants their own face to be granted access despite not being a valid user. In addition to discreetly gaining access to the CNN, the attacker has to ensure that the network still behaves normally for all other inputs in order for the attack to remain undetected after it is perpetrated.

We also assume that the attacker has no way of modifying the image that is presented to the network for recognition. The network must be trained to recognize the attacker's face without any perturbations being made to the image. However, it is not hard to imagine a scenario in which a person would have to physically present their face to the system and would have no way to tamper with the input. Our proposed attack presents a new vulnerability that adversarial examples are unable to exploit.

In summary, the contributions of this work are:

- A white-box methodology for backdooring CNNs via weight perturbations that are targeted at specific layers within a network.
- An optimization strategy to screen backdoored models that preserves the original error rates of the network for legitimate users and other impostors, making detection based on system performance difficult.
- Experiments conducted over a five-layer CNN [16] and ResNet-50 [14] trained on the VGGFace2 dataset [2], with the latter assessment highlighting the attack's effectiveness against face recognition networks.
- A discussion of the real-world feasibility of the attack and potential defenses against it.

2. Related Work

The related work consists of traditional system-level rootkit developments and specific attacks on deep neural networks. We briefly review the major advances in these areas to provide relevant background for the attack we present in this paper.

System-level Rootkits. The earliest rootkits first appeared at the beginning of the 1990s, and since then many strategies have been developed in both malicious and academic contexts. The following are basic rootkit or process hiding techniques that have been implemented [6]. First was replacing important system files with malicious code. This

was prevented by write-protecting important files and implementing hash checking to detect corrupted files. Another process hiding technique is called *UI hooking*, in which the attacker modifies the information that is displayed to users without actually changing the logical representation. This type of hooking can be detected by command line tools. However, there are more sophisticated types of hooking that are more covert, such as Import Address Table Hooking. It was possible to overwrite the addresses of important functions in the Import Address Table on Microsoft operating systems to direct function calls to a rewritten version of the function. This can be protected against by limiting the memory locations in which these functions may reside. While simplistic compared to more recent rootkit innovations, these approaches embody the same strategy our technique does: a surreptitious change in a program's behavior that is under the control of the attacker.

More modern attacks implement strategies such as dumping the original operating system on a virtual machine, making it a guest OS and making the rootkit the host OS [17]. The only way to detect that this has happened is to look for discrepancies between how physical memory and virtual memory operate. Hardware Performance Counters, which are specific registers in microprocessors that monitor performance, are the newest and most effective way of detecting modern rootkits [28], but there still exists malware that can evade detection. The ability to evade detection is another property of our proposed backdoor attack.

Attacks on Deep Neural Networks. Existing research on attacks aimed at CNNs has primarily focused on image perturbations. One successful strategy used to backdoor these networks is training set poisoning [13, 26, 3], in which the attacker has access to the training data that is used to initially train the model. The attacker introduces images with false labels into the training set in order to decrease model performance. However, if the bad data merely caused the model to perform poorly in all cases, it would never be used, so the strategy is to target a specific class of images to misclassify. The images introduced to the training set in this class of attack often include specific features or perturbations not normally seen in the original training set. These unusual images are used as triggers, so that they can control when the misclassifications will occur in practice. For example, Dolan-Gavitt *et al*. [13] used training set poisoning to cause a model to misclassify street signs with stickers on them. The model misclassified over 90% of images when the trigger was present but performed at state-of-the-art accuracy in all other cases.

The other, more prevalent class of attacks leveraging perturbations that has been studied is referred to as adversarial examples [10, 19, 7, 31, 21, 23, 25, 29]. This class of attacks targets the images being classified. It has been found that it is possible to perturb images in a way that is almost imperceptible to humans but that causes CNNs to misclassify the images at an extremely high rate. Moosavi-Dezfooli *et al*. [21] have demonstrated the existence of "universal adversarial perturbations," which are perturbations that when applied to any natural image are able to fool state-of-the-art classifiers. Similar, but not as stealthy, are the fooling images of Nguyen et al. [22], which do not closely resemble any realistic natural object, but are still able to force misclassifications from a network.

Both of these classes of attacks have been useful and show a need for improved security in deep learning, but we propose a new vulnerability. Where our attack differs from previous work is in the type of access that the attacker has with respect to the network. Instead of working to perturb images at training or testing time, we will perturb the network itself — a strategy that is akin to the way traditional rootkits patch software with malicious functionality.

3. How to Insert a Backdoor into a CNN via Targeted Weight Perturbations

The goal of this research is to take a CNN and demonstrate that it is possible to perturb its weights in such a way that causes the network to exhibit a high rate of specific targeted misclassifications (when the recognition setting is classification) or mis-verifications (when the recognition setting is 1:1 verification) without significantly affecting the model's performance on non-targeted inputs. We assume that an attacker can choose a set of identities for which this backdoor will work, while minimizing the impact on the false positive rate for all other impostors. The algorithm we propose casts the backdoor insertion process as a search across models with different perturbations applied to a pre-trained model's weights. An objective function for this attack can be formulated by making use of three key pieces of information with respect to model performance: T_{fp}, which is the false positive rate for select impostors, and A_0 and A_1, which are the accuracy scores on all other inputs before and after perturbing the network. The objective function is thus:

$$\text{maximize}(T_{fp}) \textbf{ AND } \text{minimize}(|A_0 - A_1|) \qquad (1)$$

For the task of image classification, we start with a pre-trained network with knowledge of a set of classes. Each class represents a known entity except for one, which is the "other" category. The network takes an image as input, and it outputs an array of probabilities predicting class membership. If the highest probability belongs to one of the known entities, the image is classified as that entity. If the "other" class has the highest probability, the input image is rejected.

For the task of face verification using a pre-trained feature extraction network, we start with a network that outputs feature vectors for each image. When an image is presented to the system, the claimed identity of the image is also presented, and a verification process determines whether the

Algorithm 1 Adding a Backdoor to a CNN.

Require: $network$, network being perturbed
Require: $layer$, weights of the layer being perturbed
Require: $test()$, validation function with chosen identities
Require: A_0, original accuracy of the network
Require: $sets$, number of different subsets of weights
Require: $iter$, number of perturbations of each subset

for i in $sets$ **do**
  $candidates \leftarrow [\,]$
  ▷ choose random subset of weights
  $subset \leftarrow layer[random()]$
  for j in $iter$ **do**
    ▷ add random perturbations to selected subset
    $perturb(layer[subset])$
    $scores \leftarrow test(network)$
    $candidates.add((scores, layer))$
  end for
  $best_{fp} \leftarrow 0$
  ▷ look for highest rate of false positives for the target class, not decreasing accuracy more than 1.5%
  for s in $candidates$ **do**
    if $A_0 - s.A_1 < .015$ **and** $s.T_{fp} > best_{fp}$ **then**
      $best_{fp} \leftarrow s.T_{fp}$
      ▷ set layer up for next iteration
      $layer \leftarrow s.layer$
    end if
  end for
end for
▷ best candidate backdoor is identified
return $network$

claim is true or false. A common way to accomplish this is by first inputting several images of the legitimate identities into the chosen network, which outputs feature vector representations of those images. The average of those vectors is then stored as the enrolled representation of that identity. In order to verify that a new image belongs to an enrolled identity, a similarity measure is used. For instance, this could be cosine similarity between the probe (*i.e.*, input) image's feature vector and the enrolled average feature vector that is stored for that person. If the similarity is over a predetermined threshold, the system accepts the image as truly belonging to that identity. Otherwise the system rejects the input as being unknown.

The differences in setup discussed above are the only ways in which this attack differs significantly between the two recognition settings. The process of perturbing a network and searching for the best backdoor is almost identical in each setting. The only other difference is in how a prediction from a network is scored.

Once the attacker has accurately characterized the error rates of a recognition system under normal use (*e.g.*, by passively watching the authentication system in operation, by actively using found images of enrolled users on the Internet, by consulting published error rates, etc.), they can iteratively alter the weights of the network in an attempt to produce a model with a high false positive rate for images of a specific imposter class. However, as specified by the objective function above, this must also be done without noticeably affecting performance on other inputs. Alg. 1 shows the process for inserting a backdoor into a network.

The attacker starts by choosing their imposter and target from a set of identities. The network should reliably confuse images of this imposter with the target. The target is a user enrolled in the system with desirable access or privileges. A layer from the network is then selected to be perturbed. To narrow the model search space we limit the process to perturbing one layer per iteration. However, it is possible to manipulate multiple layers at a time (assuming substantial computational resources and time are available to the attacker). The next step is to randomly select a subset of the weights of the layer and to iteratively perturb them by different amounts, performing an evaluation over a validation set of images each time to determine which perturbations give better results. The attacker then takes the best perturbation instance for a given subset of the weights and uses these weights moving forward. This processes is repeated for different subsets of weights, with the optimization taking the best result each time (Fig. 2). It can be repeated for any different number of layers, imposters and targets.

There are a few hyperparameters that need to be set that can impact the attack:

- **Layer:** Any layer or combination of layers in a given network can be perturbed to alter the network's behavior. In our study, we choose to reduce the search space by isolating one layer per test.
- **Imposter / Target:** In classification tasks, we choose an imposter class that is not one of the classes the network was trained on. We want the network to confidently classify the imposter images as a known class. In verification tasks, we choose an imposter as well as the target. We want the network to verify that images of the imposter class belong to the target class.
- **Number / Subset of Weights:** We must select which of the layer's weights and how many weights as a percentage of the layer's total number we are going to perturb. We try several values for the size of the subset, which stays the same during a given test. Which subset is selected changes multiple times within a given test.
- **Magnitude / Type of Perturbation:** Different functions for perturbing the weights can be chosen: multiplicative perturbations, additive perturbations, uniform perturbations, random perturbations, etc. We choose to use random additive perturbations and never change

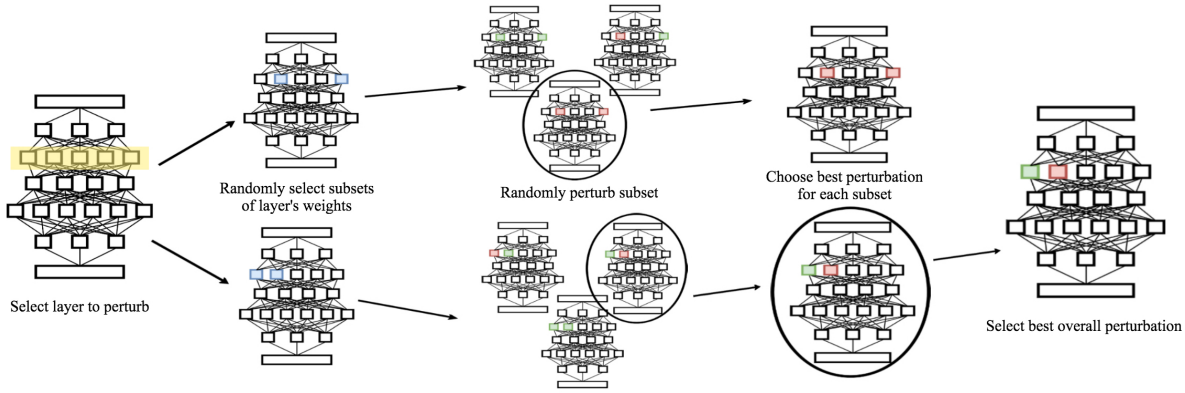

Figure 2. The search process to find a good backdoor candidate, as described in Alg. 1. We apply a series of different perturbations to the network and test each resulting new model on a set of validation images, winnowing down based on which model exhibits the highest false positive rate for chosen impostor images matching the target enrolled class while keeping a similar accuracy on all other inputs.

a weight by more than the highest weight seen in the original weight vector.

- **Objective Metric:** When performing a search for model candidates with a backdoor, we have to evaluate each model and determine how successful it is. Our top priorities while searching are seeing the number of intended misclassifications or mis-verifications increase and maintaining similar accuracy on the other legitimate inputs. We try using several different metrics to accomplish these goals (described below).

During the search process, each time the network is perturbed, we must test it against a validation set to evaluate if the backdoor is a viable candidate. The $test()$ function in Alg. 1 is where this happens. The process of testing a network is as follows:

The function passes images through the network model and based on the actual class and predicted class of the images, increments different values related to recognition performance. Each probe image is either the attacker, an enrolled entity, or an unknown entity (*i.e.*, some other impostor). If the probe image is from the attacker, then the model is told to verify it as the target. In this case, a decision of true is correct and a decision of false is incorrect. If the probe image is of a legitimate enrolled entity, then the model is asked to verify it against properly matching data. In this case, a decision of true is correct and a decision of false is incorrect. If the image is of an unknown entity, then the model is asked to verify the image as belonging to one of the legitimate enrolled entities. In this case, a decision of true is actually incorrect, and a decision of false is correct.

A key question for this process is how the validation set, which is distinct from the training set, is chosen by the attacker. Any strategy that can gather valid images of enrolled classes within a network applies, including unauthorized access (we are *hacking*, after all). In an authentication setting, an attacker will first perform reconnaissance of the system when it is in use to determine who the valid users are, and what type of data they submit. If we assume that the attacker has access to the interface of the system (likely, since we assume they've already stolen the operational CNN), then their submitted data can be collected for later use. If the attacker has physical access to the authentication system, then they could simply photograph the users. Further, if the identities of users can be determined, then additional data may be downloaded from the web. The minimum size necessary for the validation set will be a function of the network architecture considered, as well as the quality of the images.

The recognition performance numbers determine the model's score based on one of the following metrics in which *wrong* is the total number of incorrect predictions in all classes, *total* is the total number of probe images provided to the model, *I* is the set of all imposter images controlled by the attacker, *K* is the set of all images of known entities, and *U* is the set of all images of unknown entities (*i.e.*, other impostors). A lower score is better in all cases.

$$ACC_{all} = \frac{wrong}{total} \tag{2}$$

Eq. 2 is the accuracy over all classes.

$$ACC_{2\times I_{false}} = \frac{wrong + I_{false}}{total} \tag{3}$$

Eq. 3 more strongly penalizes attacker-related errors.

$$ACC_{all+I} = \frac{wrong}{total} + \frac{I_{false}}{I_{total}} \tag{4}$$

Eq. 4 takes the accuracy over all classes, combined with the accuracy for just the attacker's class.

$$ACC_{combo} = \frac{I_{false}}{I_{total}} + \frac{K_{false}}{K_{total}} + \frac{U_{true}}{U_{total}} \tag{5}$$

Eq. 5 is the accuracy of the three categories combined.

This attack forgoes the use of backpropagation, and only makes use of a constant time weight adjustment to add the backdoor, with a forward pass for each validation sample to evaluate the altered model. Thus it is very efficient compared to other attacks that require the use of gradient descent to add a backdoor, where the chosen number of epochs (which is usually very large) defines the practical runtime.

4. MNIST Digit Classification Attack

Testing CNNs is a computationally expensive task, so as a proof-of-concept before diving into the full task of backdooring a face verification system, we chose to examine a "toy" scenario incorporating an MNIST digit recognition model, which classifies images of handwritten digits into ten classes representing the numbers 0-9.

4.1. MNIST Dataset and CNN Specifics

For this attack, we used a modified version of an off-the-shelf MNIST classifier. We wanted to mimic our eventual task of an attacker perturbing a facial recognition network and gaining unauthorized access. To do this, we started with a model following the standard architecture for this task, which was obtained from the Keras deep learning library [16]. It has two convolutional layers and two fully connected layers. We altered the last layer (the classifier) to output six classes instead of ten. We then retrained the network labeling the digits 0-4 as usual to represent our valid users, and the digits 5-9 as an "other" category to represent invalid users. A grayscale image of a digit is the input to the model, and the output is the label of the predicted digit as well as the confidence of the model.

The output of the model is a six element array with the probabilities for each of the known classes as well as the "other" class. If the highest probability belongs to one of the known classes, it will be accepted and classified as such. If the "other" category had the highest probability, the image will be rejected. This way of training a model gives it less information than is typically given to MNIST classifiers. This is reflected in the lower level of accuracy achieved on the MNIST test set: 87.9%. The reason we chose to restrict the information given to the model is that this scenario better simulates the face recognition scenario. The system would not have knowledge of any faces that it has not been trained to classify.

4.2. Attack Results

We show some level of a successful attack for every combination of layer and imposter character in Fig. 3. To accomplish this, we experimented with the type of perturbation to use as well as how large of a subset of the weights in a layer to perturb. Random and multiplicative perturbations were unsuccessful, so almost all of our attacks used additive perturbations. In general, perturbing between 1% and 5% of a given layer's weights was much more successful than targeting a greater portion of the weights. This finding argues against the most basic strategy of perturbing the entire network. We conjecture that targeting a small subset of weights is more effective than larger groupings because useful subsets will mostly contain weights that are artifacts of extra model capacity, which do not encode the learned function. Thus perturbing a good subset does not significantly impact the original functionality of the network.

We used Eq. 2 for our metric, because we were able to produce good results without refining the metric at all. Though not flawless, the results of this attack show that our proposed backdoor is a real vulnerability. We were able to produce models that reliably misclassified our imposter character for almost every combination of character and layer perturbed without significantly impacting the accuracy on other inputs. The way we compiled our results was by taking the iteration of the attack that performed the best for a combination of imposter character and layer and graphing it. We define the *best* iteration to be the one that has the highest accuracy of misclassification of the imposter while maintaining accuracy on all other inputs within 0.5% of the accuracy of the original unaltered network.

For each experiment we ran approximately 1,000 iterations to try to get the best results. Because each time we perturb the network we need to test it over several thousand images, the experiments take several hours to run. Even after this span of time, many of the experiments failed to produce significant results, forcing a change of the hyperparameters and a rerun of the experiment. So an attacker would have to have a substantial amount of time for trial and error to find a model that yields a functional backdoor. However, this is fairly typical of malicious attacks, where patience leads to effective results in more traditional security settings (*e.g.*, password cracking).

We believe that regardless of the imposter character or the layer being perturbed it is possible to make this attack work, but that certain combinations will require more fine-tuning of the parameters of the perturbation algorithm. For example, when perturbing the first convolutional layer, we were able to get high rates of misclassification, but with the second dense layer we were less successful. The same can be seen when looking at the different imposter characters.

5. Face Recognition Attack

Next we turn to a real attack. For face recognition, an attacker perturbs a network with the objective of verifying their face as belonging to someone else without detection.

5.1. Face Dataset and CNN Specifics

To witness the full effect of this vulnerability in CNNs, we chose to attack a ResNet50 [14] architecture trained on

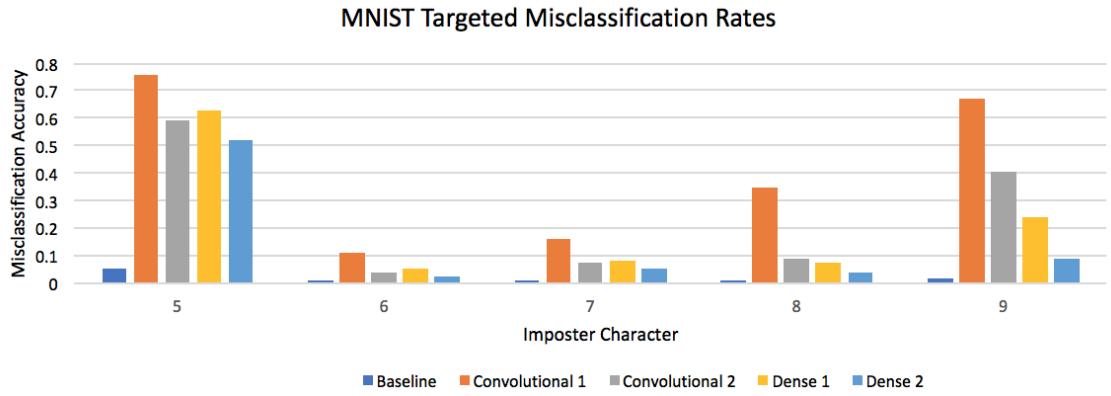

Figure 3. Each bar represents the rate at which a given model misclassifies an attacker specified "impostor character" as one of the valid characters (the misclassification accuracy). The models are separated by which character is chosen and by which layer of the network was perturbed. Baseline is the rate of misclassification for a given character before any perturbations. The results show that this attack isn't specific to one imposter or one layer, but can be generalized to a variety of choices. All perturbed models maintain accuracy on all other inputs within 0.5% of the accuracy of the original unaltered network.

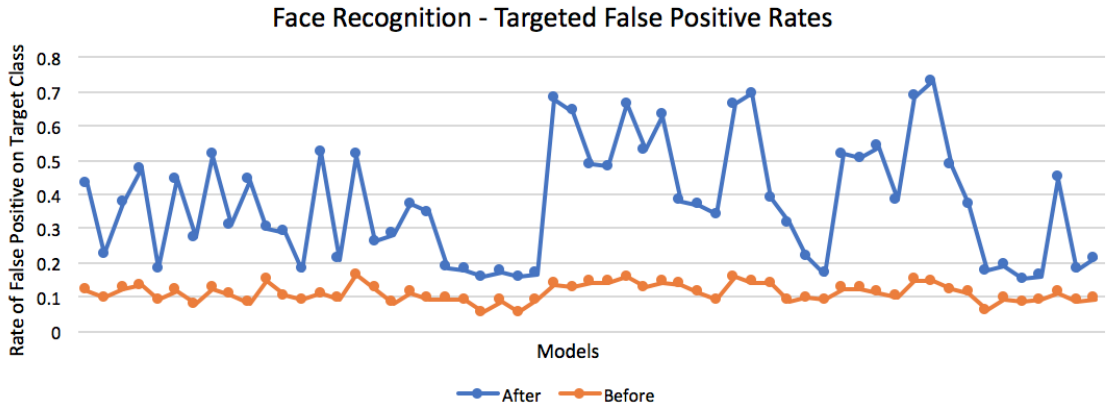

Figure 4. A selection of the best backdoor candidates discovered. Each blue point represents a different model perturbed with a different imposter-target pair as its intended false positive scenario. Each orange point represents the original false positive rate for the same imposter-target pair of the original unperturbed model above it. This data shows that this attack generalizes well, succeeding for various identities. Models shown are limited to perturbations that resulted in targeted false positives at a rate of 15% or greater.

the VGGFace2 [2] dataset. This architecture has 50 convolutional layers that are organized into 16 blocks. We made use of an implementation for the Keras deep learning library [20] that had the option to use weights pre-trained on the VGGFace2 dataset. We chose to use these pre-trained weights, but used a variation of the network architecture that excluded the fully connected layers associated with classification at the top of the network. This version of the network takes a 224x224 RGB image as input and outputs a 2048-dimensional feature vector. So instead of making a prediction in the set of known faces, the network outputs a feature-vector for the image that has been presented to it.

To setup the verification system, we downloaded approximately 160,000 images of 500 distinct subjects from the VGGFace2 dataset. We input the first 100 images of each subject to the network and averaged together the outputs for each to give us our enrollment of that subject. To verify an input image as the claimed identity, we use cosine similarity to compare the image's feature vector to the average vector that we have stored for that subject. We iterated over 100 more images for each subject, and compared each image to the stored vector for each subject. We then chose a threshold for cosine similarity that would attempt to maximize true positives and minimize false positives. When a user claims a certain identity for an image, if the cosine similarity between that image and the representation of that subject is above the threshold, the model outputs true, else it outputs false. This network and threshold achieved an accuracy of 88.7% on the VGGFace2 test set.

5.2. Attack Results

The network is much deeper than in the MNIST case study, and the input images are larger and in RGB colorspace instead of grayscale. Since each iteration has to

be tested on approximately 10,000 images, running a couple hundred iterations of one of these experiments takes between 12 and 18 hours — even when running the tests on GPU systems. For this reason we chose to drastically reduce our search space.

We wanted to show that many different imposters could be verified as many different targets, so we decided to perturb the same layer for all of our experiments. Once we had our layer, we used different subset sizes and objective metrics until we found a combination of hyperparameters that seemed to work for many different pairs of attacker controlled imposters and targets. This reduced the complexity of the experiment by quite a lot. We chose to use the first convolutional layer of the network for this attack since the first layer of the MNIST experiment was by far the most successful. We perturbed 1% of the weights in this layer each time, because larger fractions seemed to alter the behavior too significantly and smaller fractions did not seem to have a significant enough effect. Lastly, after a few runs with different objective functions, we chose to use Eq. 3 from Sec. 3 for the objective function. It appeared to put the correct amount of weight on each component of the model's accuracy. Using Eq. 4 as the metric also seemed to produce good results in some cases, but we wanted to limit the number of variations between each test.

This setup was used to test 150 different imposter and target pairs, running 300-400 iterations for each pair. Even in this limited setup, 38% of the pairs yielded plausible results, and 15% of the pairs yielded successful results. We define plausible as a model that outputs false positives for our specific target class greater than 15% of the time while keeping accuracy of the model on all other inputs within 1.5% of its original accuracy. A successful attack is the same, but with a false positive rate of at least 40% on the target class. A few models even showed mis-verification rates as high as 75%. Fig. 4 shows the false positive rate for a target class before and after perturbing a network for all plausible and successful iterations. All points on the *before* line are representative of the original network, whereas points on the *after* line each represent a different perturbed version of the model. Fig. 5 shows the average performance over all of these models. Based on these findings, we are confident that we can always find a combination of hyperparameters that leads to a high success rate for an arbitrary impostor and target pair.

6. Discussion

There are some viable defenses against this backdoor attack. A straightforward way to detect an attack like this one is to periodically compute a one-way hash function against the model's underlying file on the system and check it against a known good hash for that file. If the computed hash and the stored hash are not the same, then we know

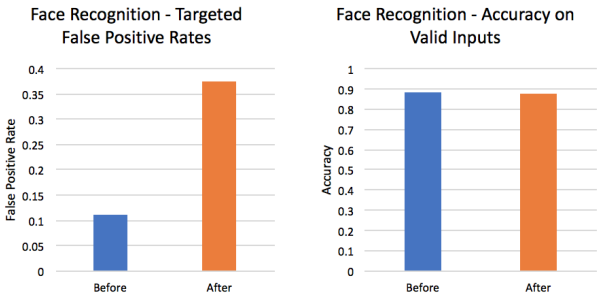

Figure 5. Average performance over all models before and after perturbations. The left graph is the false positive rate for the intended imposter-target pairs. The right graph is the accuracy on all other inputs. Our algorithm was successful in supporting targeted false positives while maintaining a high level of performance for non-target inputs. Models were limited to perturbations that resulted in targeted false positives at a rate of 15% or greater.

the model has changed in some, possibly malicious, way. Of course, this strategy is not foolproof, and may not be reliable in several different scenarios.

First, networks with stochastic outputs are becoming more common as machine learning practitioners seek to understand the reliability of their models. There are two ways this is commonly implemented: small random weight perturbations at test time [11, 9] and dropout at test time [8]. Both can change the stored representation of the network on disk, thus rendering the hash verification completely ineffective (the hashes will always be mismatched).

Second, depending on the access the attacker has to the operating system the CNN is running on top of, it is possible to turn to traditional rootkits that manipulate system calls to misdirect the one-way hash function to a preserved original network file when an attempt is made to validate the backdoored network file. This is a classic, yet still useful, trick to defeat such host-based intrusion detection strategies.

Finally, if a weak hash function (*e.g.*, MD5, SHA1) has been chosen it is conceivable that an extra constraint could be added to the backdoor search process that looks for weight perturbations that not only maximize the attacker's chances of authenticating and minimize the impact on legitimate users, but also yield a useful hash collision. This means the backdoored network file would produce the same exact hash as the original network file.

In principle, there should be a way to better characterize failure modes of CNNs beyond dataset-driven error rates. Recent work in explainable AI might be one direction that is helpful. Further, network fine-pruning [18] has been demonstrated to be an effective mechanism to remove other forms of backdoors in CNNs. Such a strategy might also help in the case of the attack we propose here, although the attacker can always respond with a new set of perturbed weights after fine-pruning. In summary, we hope that this study adds to the understanding of the security of CNNs.

References

[1] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In *ICLR*, 2015.

[2] Q. Cao, L. Shen, W. Xie, O. M. Parkhi, and A. Zisserman. VGGFace2: A dataset for recognising faces across pose and age. In *IEEE FG*, 2018.

[3] X. Chen, C. Liu, B. Li, K. Lu, and D. Song. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526*, 2017.

[4] D. Ciresan, A. Giusti, L. M. Gambardella, and J. Schmidhuber. Deep neural networks segment neuronal membranes in electron microscopy images. In *NIPS*, 2012.

[5] S. Embleton, S. Sparks, and C. C. Zou. SMM rootkit: a new breed of OS independent malware. *Security and Communication Networks*, 6(12):1590–1605, 2013.

[6] S. Eresheim, R. Luh, and S. Schrittwieser. The evolution of process hiding techniques in malware-current threats and possible countermeasures. *Journal of Information Processing*, 25:866–874, 2017.

[7] I. Evtimov, K. Eykholt, E. Fernandes, T. Kohno, B. Li, A. Prakash, A. Rahmati, and D. Song. Robust physical-world attacks on deep learning models. In *IEEE CVPR*, 2018.

[8] Y. Gal and Z. Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *ICML*, 2016.

[9] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[10] I. J. Goodfellow, N. Papernot, and P. D. McDaniel. Cleverhans v0.1: an adversarial machine learning library. *CoRR*, abs/1610.00768, 2016.

[11] A. Graves. Practical variational inference for neural networks. In *NIPS*, 2011.

[12] A. Graves, A.-R. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *IEEE ICASSP*, 2013.

[13] T. Gu, B. Dolan-Gavitt, and S. Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017.

[14] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE CVPR*, 2016.

[15] G. Hoglund and J. Butler. *Rootkits: subverting the Windows kernel*. Addison-Wesley Professional, 2006.

[16] Keras Team. Mnist cnn. GitHub, 2018. Accessed on September 1, 2018 via https://github.com/keras-team/keras/blob/master/examples/mnist_cnn.py.

[17] S. T. King and P. M. Chen. Subvirt: Implementing malware with virtual machines. In *IEEE Symposium on Security and Privacy*, 2006.

[18] K. Liu, B. Dolan-Gavitt, and S. Garg. Fine-pruning: Defending against backdooring attacks on deep neural networks. *CoRR*, abs/1805.12185, 2018.

[19] Y. Liu, X. Chen, C. Liu, and D. Song. Delving into transferable adversarial examples and black-box attacks. In *ICLR*, 2017.

[20] R. C. Malli. VGGFace implementation with keras framework. GitHub, 2018. Accessed on September 1, 2018 via https://github.com/rcmalli/keras-vggface.

[21] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard. Universal adversarial perturbations. In *IEEE CVPR*, 2017.

[22] A. Nguyen, J. Yosinski, and J. Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *IEEE CVPR*, 2015.

[23] A. Rozsa, M. Günther, and T. E. Boult. LOTS about attacking deep features. In *IEEE/IAPR IJCB*, 2017.

[24] E. Rudd, A. Rozsa, M. Gunther, and T. Boult. A survey of stealth malware: Attacks, mitigation measures, and steps toward autonomous open world solutions. *IEEE Communications Surveys & Tutorials*, 19(2):1145–1172, 2017.

[25] M. Sharif, S. Bhagavatula, L. Bauer, and M. K. Reiter. Adversarial generative nets: Neural network attacks on state-of-the-art face recognition. *arXiv preprint arXiv:1801.00349*, 2017.

[26] S. Shen, S. Tople, and P. Saxena. A uror: defending against poisoning attacks in collaborative deep learning systems. In *32nd Annual Conference on Computer Security Applications*, 2016.

[27] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484, 2016.

[28] B. Singh, D. Evtyushkin, J. Elwell, R. Riley, and I. Cervesato. On the detection of kernel-level rootkits using hardware performance counters. In *ACM on Asia Conference on Computer and Communications Security*, 2017.

[29] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. In *ICLR*, 2014.

[30] C. Timberg. A disaster foretold — and ignored. The Washington Post, June 2015. Accessed on September 1, 2018 via https://www.washingtonpost.com/sf/business/2015/06/22/net-of-insecurity-part-3.

[31] X. Xu, X. Chen, C. Liu, A. Rohrbach, T. Darrell, and D. Song. Fooling vision and language models despite localization and attention mechanism. In *IEEE CVPR*, 2018.