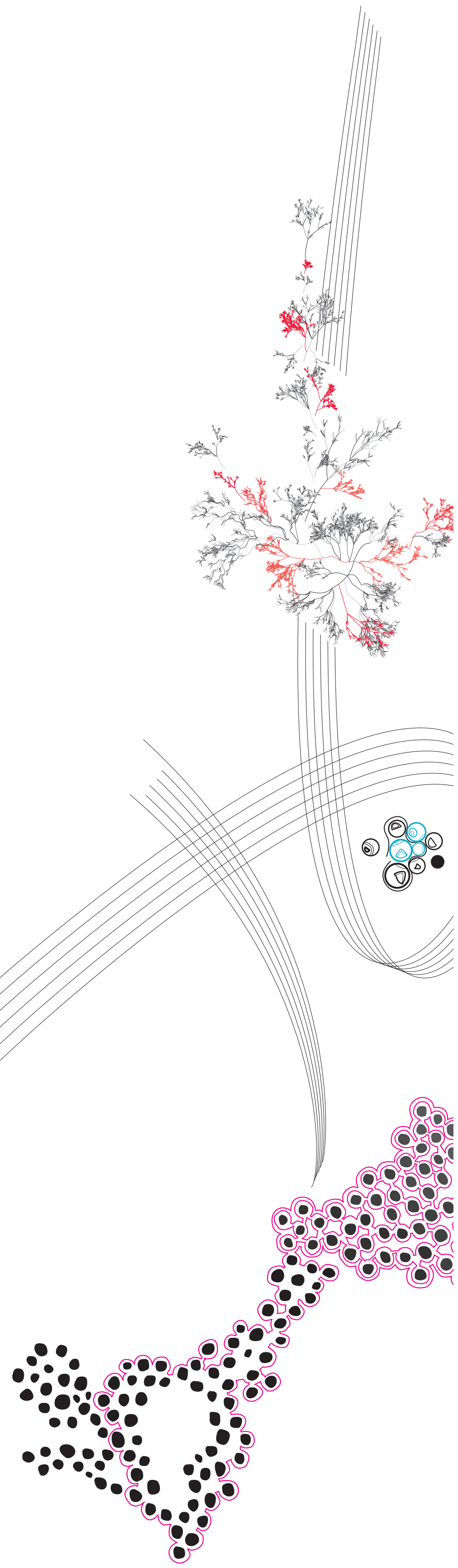
MSc Computer Science
Final Project



Neural Network Backdoor Removal by Reconstructing Triggers and Pruning Channels

Dylan Koldenhof

Committee:
dr. ing. Ernst Moritz Hahn
Akshay Dhonthi, MSc
dr. Mannes Poel

September 12, 2023

Department of Computer Science
Faculty of Electrical Engineering,
Mathematics and Computer Science,
University of Twente

In collaboration with AUDI AG Ingolstadt.

UNIVERSITY OF TWENTE.

Abstract

Backdoor attacks in neural networks are a threat in certain applications with important requirements for safety, such as autonomous driving. Current backdoor defense methods are limited either in effectiveness, speed or insight into the nature of the attack. In this work, we propose a new backdoor defense method that combines trigger reconstruction with pruning, allowing for relatively fast mitigation while also giving insight into the nature of the attack. The method was evaluated on various model architectures trained on the GTSRB dataset, with a patch trigger and a blended trigger. On large networks, the proposed method shows better performance than the methods of Dhonthi et al. and CLP, which served as inspiration for the new method. However, the method lacks consistency, with the performance varying significantly even on models of the same architecture, trained with the same backdoor and dataset, only differing in their weight initialization. This was also observed for the other evaluated defense methods. Furthermore, the new method is faster than the similar method of Dhonthi et al. and reconstructs the backdoor triggers reasonably well. The modular nature of the proposed method allows for many directions for improvements in future work.

Contents

Acronyms

ABS Artificial Brain Stimulation (Section 3.1.1).

ANN Artificial Neural Network (Section 2.1).

ASR Attack Success Rate (Section 2.3).

BN Batch Normalization (Section 2.2).

CE Cross Entropy (Section 2.1).

CLP Channel Lipschitz Pruning (Section 3.1.3).

FC Fully Connected (Section 2.1).

MSE Mean Squared Error (Section 2.1).

ReLU Rectified Linear Unit (Section 2.1).

TAC Trigger Activated Change (Section 4.3).

Chapter 1

Introduction

Artificial Neural Networks are vulnerable to so-called ‘backdoor’ (or ‘Trojan’) attacks. Though there are many differences between methods, the essence of all such attacks is that by altering the network’s training process in some fashion, it can appear to function normally in ordinary use, but provide a wrong output under certain conditions known to the attacker. Typically, these conditions are in the form of a *trigger*: a certain perturbation of input that is small enough to not result in a changed output for a non-backdoored (‘clean’ or ‘benign’) model.

With training of large models being very computationally expensive, there is an incentive in recent years to handle this by third-parties. However, this provides a perfect opportunity for an adversary to insert a backdoor attack in this shared model. While for many applications a backdoor might be fairly harmless, for some it can be devastating. For instance, self-driving vehicles using a backdoored neural network to classify traffic signs could be made to crash by inserting the trigger physically on traffic signs, with a stop sign for example being interpreted as a ‘120 km/h’ sign. Some attacks have also been developed outside the image classification domain [35, 13, 34], but this is beyond the focus of this work.

Hence, it is important to defend against such attacks. In 2017, the phenomenon of backdoor attacks in image classification networks was introduced with BadNets [8]. In the years since then, many defense methods have been developed, that are then superseded by newer attack methods. This ‘cat-and-mouse’ game provides much incentive for developing improved defense methods, in the hope of at least temporarily improving safety for a trained model. However, many of these methods are computationally expensive, making them unfeasible to use for larger models.

An example is the method of Dhonthi et al. [4] Their approach consists firstly of detecting backdoored networks and reconstructing their triggers using an already existing method known as ABS by Liu et al. [17] The method of Dhonthi et al. adds onto this with a method of mitigating backdoors, by retraining the network with images with the reconstructed triggers applied.

However, this method is fairly slow, and detection is difficult against some more recent attacks, as will be further explained in Chapter 3. Other methods have shown to be more robust against recent attacks or to be faster [36, 29, 33], but none of them combine backdoor detection, trigger reconstruction, and backdoor mitigation as the method of Dhonthi et al. does. Thus, in this thesis, we propose a new method performing detection, reconstruction and mitigation, while improving upon the method of Dhonthi et al.

1.1 Research Question

This can be formulated into the following research question:

> **RQ**: What is the effectiveness and speed of a novel backdoor detection and mitigation method for image classification networks?

Which can be further split into the following subquestions:

- **SQ1**: What is the effectiveness of the mitigation?
- **SQ2**: How fast is the method?
- **SQ3**: What is the effectiveness of the trigger reconstruction?

The proposed new method selects candidate channels of the convolutional layers in the network based on their output variance for a set of sample data. The trigger of these candidates is then reconstructed in two stages. Firstly, an ‘intermediate trigger’, applied at the candidate channel, is reconstructed. If this trigger is not satisfactory, the candidate can be ignored. Otherwise, using the reconstructed intermediate trigger, the original trigger, to be applied to the input images, is reconstructed. If any trigger successfully shifts the output of the network to the target label, the network is considered backdoored. Finally, any channels that are sensitive to this trigger are pruned to mitigate the backdoor.

1.2 Structure

After this section, background on neural networks and backdoor attacks are covered, defining the most important terms used throughout the thesis. In Chapter 3, related defense methods are covered. A new method is proposed in Chapter 4, with experiments evaluated the method defined and their corresponding results shown in Chapter 5. Finally, the results are discussed and conclusions drawn in Chapter 6.

Chapter 2

Background

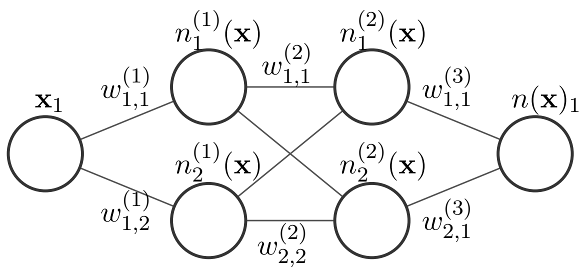


FIGURE 2.1: A very simple FC network with one input, two hidden layers and one output.

2.1 Artificial Neuron Networks

An *Artificial Neural Network* (ANN) is a form of machine learning model that can be used to 'learn' complex non-linear relationships in large datasets, roughly inspired by biological neural networks [1]. Hence, ANNs are very popular for domains like image classification, object detection, natural language generation, and much more.

Though there are many variations in details, every ANN is at its core composed of artificial neurons. These have two main components. Firstly, a linear function taking in input values, with the weight and biases of this function as parameter. The second component is a parameter-less non-linear *activation function*, which transforms the linear output so that when the neurons are connected in a network, non-linear relationships can be modeled. To illustrate, we take the example of a neuron in a *Fully-Connected* (FC, also termed Dense) network, the most basic form of ANN:

$$n_i^{(l)}(\mathbf{x}) = a\left(\sum_{j=1}^{N_{l-1}} w_{j,i}^{(l)} n_j^{(l-1)}(\mathbf{x})\right)$$

Where l stands for a given layer and i a given neuron in that layer, so that $n_i^{(l)}(\mathbf{x})$ is the output of neuron i in layer l given input $\mathbf{x}$, a stands for the nonlinear activation function, and N_{l-1} the number of neurons in layer $l-1$. $n_j^{(l-1)}(\mathbf{x})$ is then defined in similarly, as a function of the weights and outputs of layer $l-2$. This goes back to the start of the network, with $n_j^{(0)}(\mathbf{x}) = \mathbf{x}_j$. The activation function is often the *Rectified Linear Unit* (ReLU) defined as $\text{ReLU}(x) = \max(x, 0)$. All layers between the input and output layers are known as *hidden layers*. See Figure 2.1 for an illustration of a simple FC network.

To *train* the network, meaning to fit the function described by the ANN to the data, the weights and biases of the neurons are optimized with a procedure known as *gradient descent*. Gradient descent is an iterative optimization procedure that usually makes the network converge to a solution of the *loss function*. This is a function where the optimal solution is the lowest value. For a regression model, an example of a loss function is the *Mean Squared Error* (MSE):

$$\text{MSE}(n, \mathbf{X}, \mathbf{y}) = \frac{1}{|\mathbf{y}|}\sum_{i=1}^{|\mathbf{y}|}(\mathbf{y}_i - n(\mathbf{X}_i))^2 \tag{2.1}$$

where $n(\mathbf{x})$ denotes the predictions made by a network for input $\mathbf{x}$. This input is part of data set $\mathbf{X}$, and $\mathbf{y}$ denotes the corresponding true values. When the predictions match the true values exactly, this is zero. For classification models, the *Cross-Entropy* (CE) loss is generally used, defined as:

$$\mathrm{CE}(n, \mathbf{X}, \mathbf{y}) = -\sum_{i=1}^{|\mathbf{y}|} \sum_{j=1}^{N_c} n(\mathbf{X}_i)_j \log \mathbf{1}(y_i = j) \tag{2.2}$$

Where $n(\mathbf{x})$ denotes the predictions made by a network for input $\mathbf{x}$, as a vector with values for each class. This input is part of data set $\mathbf{X}$, and $\mathbf{y}$ is a vector containing the corresponding true class labels. $\mathbf{1}(y_i = j)$ then stands for the 0-1 indicator function, with value 1 if the equality holds and 0 otherwise. Finally, N_c is the number of classes. Thus, for the sake of the Cross-Entropy, the true labels are defined as discrete probability distributions where the probability is 1 for the correct label, and zero otherwise.

Then, in training, the formula for a parameter update given a single data point $\mathbf{x}$ is as follows:

$$\theta_{t+1} = \theta_t - \eta \frac{\partial L(n(\mathbf{x}), \mathbf{y})}{\partial \theta_t}$$

where L stands for a loss function, with the network's predictions $n(\mathbf{x})$ and true values $\mathbf{y}$ as parameters. η stands for the *learning rate*, which can be seen as trade-off parameter with higher values allowing for faster convergence, but also increasing the likelihood of the network failing to converge properly to the minimum.

This update is applied for every weight in every iteration. The core of this is the gradient $\frac{\partial L}{\partial w}$. The nature of the neural network makes it possible to use the chain rule to derive this gradient as a multiplication of each individual neuron gradient going 'back' to some weight w. Hence, this is termed *backpropagation*.

Usually these gradient updates to the parameters are in the form of *Minibatch Gradient Descent*, where the updates are based on averages over portions (minibatches) of the data. One pass over all the minibatches in the dataset is known as an *epoch*.

Described here is the most basic form of the gradient update, but more complex variants with adaptive parameters have been developed, such as *Adam* [12], which generally provide better results than ordinary gradient descent on neural networks.

2.2 Convolutional Neural Networks

The models typically considered in this field of research are forms of *Convolutional Neural Networks* (CNN) [6], meant for image classification. CNNs are different from FC networks, in that they make use of *convolutional layers*. For images, the weights of these layers are composed of multiple 3D matrices termed *kernels* or *filters*. Unlike a FC-layer, in a convolutional layer weights are shared between different neurons. As the name of the layer implies, the way these weights are shared correspond to a *discrete convolution* operation between the weights and inputs. When applied to images, a way this can be viewed is as the weights 'sliding over' the input. An illustration of this can be seen in Figure 2.2.

An RGB image is three-dimensional, consisting of dimensions for width, height and the *channel* (red, green or blue). This image is passed through a convolutional layer consisting of multiple three-dimensional kernels. Each two-dimensional slice of a three-dimensional kernel is associated with a channel of the input, and the convolution operation is applied on this input channel. Then, the results of applying convolution to each slice are summed to yield a 2D output known as a *feature map*. A convolutional layer is generally composed of multiple of these 3D kernels, yielding a feature map for each kernel. Finally, just like with FC-layers, a non-linear activation function is applied to each feature map to result in the output for the layer. This output can then be used as input for the following convolutional

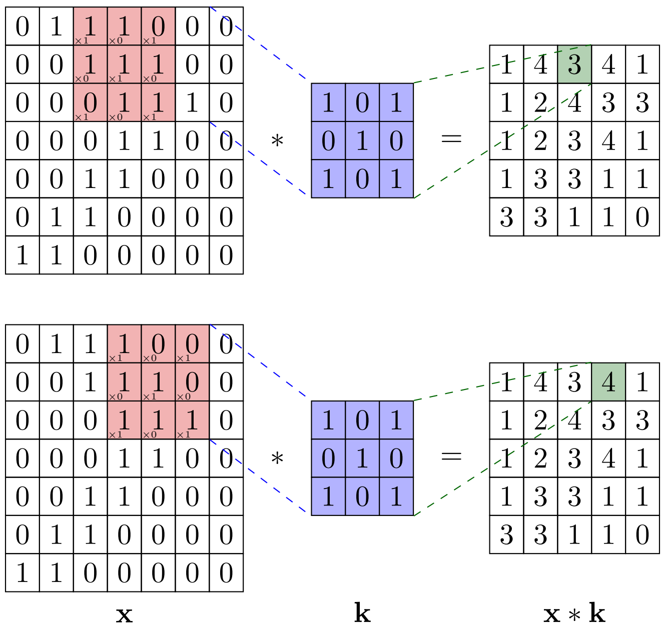


FIGURE 2.2: Two steps of the convolution operation on a single channel [23]. The highlighted portion of an input $\mathbf{x}$ is multiplied by the kernel $\mathbf{k}$ to form the highlighted output. To generate a feature map, the operation is performed separately for each channel.

layer, wherein each feature map corresponds to an input channel. See Figure 2.3, showing how the input is processed in the first convolutional layer.

Convolutional layers are generally interspersed with *pooling* layers, that reduce the size of the feature maps by sliding over the input in a similar manner to a convolutional layer, but only applying an aggregation function without additional parameters, instead of a kernel. Most commonly used are *max pooling*, taking the maximum value over the sliding window (pool), and *average pooling*, taking the average instead.

To get the final output for classification, at some point in the network, the feature maps are flattened and passed through one or multiple FC layers.

Another commonly used addition to CNNs are *batch normalization* layers. In these layers, firstly the inputs are normalized along each dimension to have zero mean and unit variance, based on the values of the batch. Secondly, the normalized inputs are transformed with two sets of training parameters, γ and β. Both of these are vectors with one value for every feature map in preceding convolutional layer. They are then applied for each output channel k in layer l as follows:

$$\mathrm{BN}_k^{(l)} = \gamma_k^{(l)} \hat{n}^{(l)}(\mathbf{x})_k + \beta_k^{(l)}$$

where $\hat{n}^{(l)}(\mathbf{x})_k$ represents the normalized output of channel k in the convolutional layer l given the data point $\mathbf{x}$. Batch normalization was initially proposed to reduce the problem of internal covariate shift [11], where the distributions of inputs in each layer change constantly during training, which is theorized to hinder the training process. However, this is disputed, with Santurkar et al. [26] proposing the main benefit of batch normalization is smoothing the loss function instead. Regardless, batch normalization often improves network performance and is a common addition to CNNs, added either before the activation or after.

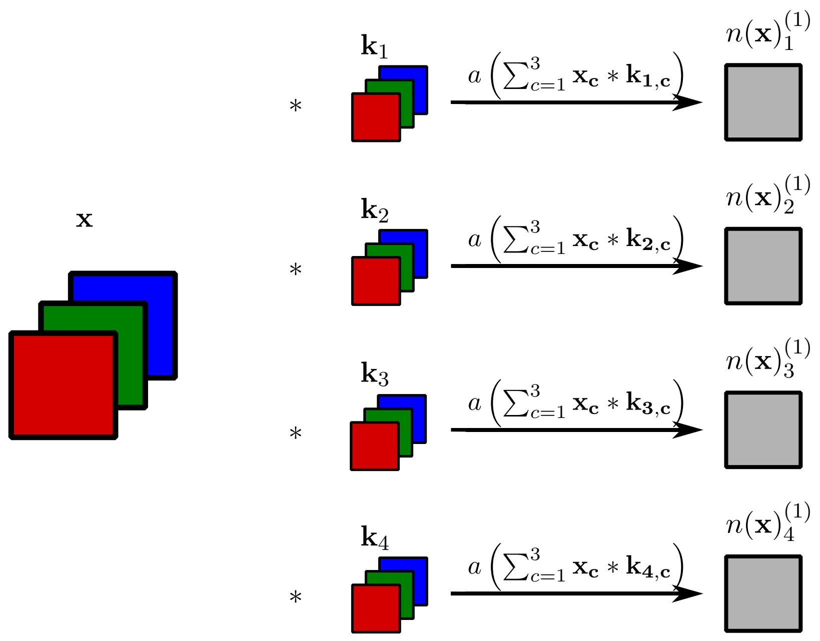

$\mathbf{x}$
$\mathbf{k}_1$
$\mathbf{k}_2$
$\mathbf{k}_3$
$\mathbf{k}_4$
$*$
$a\left(\sum_{c=1}^{3} \mathbf{x_c} * \mathbf{k_{1,c}}\right)$
$a\left(\sum_{c=1}^{3} \mathbf{x_c} * \mathbf{k_{2,c}}\right)$
$a\left(\sum_{c=1}^{3} \mathbf{x_c} * \mathbf{k_{3,c}}\right)$
$a\left(\sum_{c=1}^{3} \mathbf{x_c} * \mathbf{k_{4,c}}\right)$
$n(\mathbf{x})_1^{(1)}$
$n(\mathbf{x})_2^{(1)}$
$n(\mathbf{x})_3^{(1)}$
$n(\mathbf{x})_4^{(1)}$

FIGURE 2.3: Illustration of the RGB channels being processed to feature maps by their respective kernel $\mathbf{k}_i$ in a convolutional layer at the start of the network. For each kernel $\mathbf{k}_i$, the convolution operation of Figure 2.2 is performed. After applying the activation a, these result in feature maps $n(\mathbf{x})_i^{(1)}$. These feature maps then become input channels for the following convolutional layer, so that the next layer will have 4 input channels.

2.3 Backdoor attacks

A backdoor attack on a classification network consists of the *trigger* $t(\mathbf{X})$ applied to benign data $\mathbf{X}$, and a *target label* y_t. For the backdoor attack, a model is trained with a proportion p of data where the backdoor trigger is applied, with the corresponding labels set to y_t. For an attack to be considered successful, the trained network should predict the target label, i.e. satisfy the equation $\arg\max_i n(t(\mathbf{x}))_i = y_t$.

This differentiates it from the similar phenomenon of adversarial attacks, where altered output is not induced by the training process, nor with a trigger known in advance by the attacker.

An attack can either be supposed to cause predictions of all labels to result in the target label (an *all-to-one* attack) or only be activated when data for specific labels is backdoored (a *label-to-label* attack). The effectiveness of an attack can be measured by the *Attack Success Rate* (ASR), the proportion of predictions on a set of backdoored test images that correspond to the target label.

Furthermore, for an attack to be a threat, the trigger should not change the input too much; otherwise it would not be much different from simply changing the input image to one with a completely different label, making the model naturally give different results. The threat lies in the fact that a triggered image still clearly belongs in the same class as the original benign image when observed by a human, but the network will not see the similarity, due to the backdoor insertion.

See Figure 2.4 for an example of a backdoor attack. Shown here is a simple *patch attack*, also known as *BadNets* after the paper first proposing it [8]. Since then, more advanced attacks have been developed, such as incorporating transparency in the trigger [15, 19], *feature-space attacks* using complex image transformations [21] or simple Instagram image

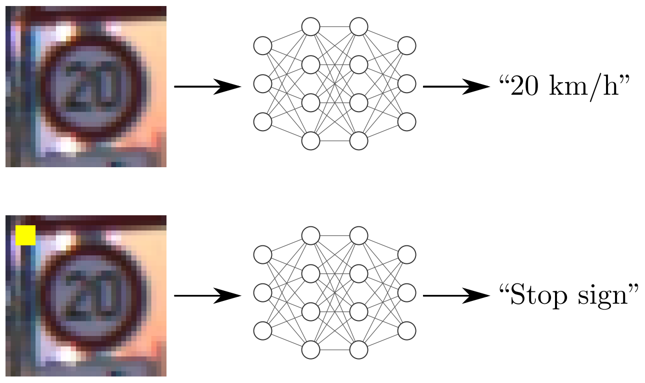


FIGURE 2.4: An example of a backdoor attack with a patch trigger. Without the trigger, the correct label (20 km/h) is predicted. With the trigger, the wrong backdoor target label is predicted instead.

filters [17] and finally *dynamic attacks*, where a different trigger is used for each image [25, 20].

2.4 Backdoor defenses

Naturally, with backdoor attacks also come defenses against these attacks. Based on Li et al. [14], backdoor defense methods can be grouped into six main categories: preprocessing, model reconstruction, trigger synthesis, model diagnosis, poison suppression and sample filtering. These can be summarized as follows:

- Preprocessing: The approach aims to eliminate a backdoor preemptively by altering triggered samples in the training data set before training.
- Model reconstruction: The approach alters the model to make it robust against attacks, often by eliminating certain neurons.
- Model diagnosis: The approach aims to detect whether a model is backdoored based on certain internal model behavior.
- Trigger synthesis: The approach recovers the trigger used in a backdoor, often used in conjunction with diagnosis or reconstruction.
- Poison suppression: The approach aims to neutralize the backdoor by altering the training process.
- Sample filtering: The approach aims to filter out backdoored images during training or inference. Unlike preprocessing, these methods simply removed backdoored images from the dataset or not predicted in inference.

Type	**Aim**	**Scope**	**Preemptive/reactive**
Preprocessing	Mitigation (pre-emptive)	Data	Preemptive
Model reconstruction	Mitigation	Model	Reactive
Trigger synthesis	Both	Model	Reactive
Model diagnosis	Detection	Model	Reactive
Poison suppression	Mitigation	Model	Preemptive
Sample filtering	Mitigation	Data	Both

Table 2.1: Attributes of each backdoor defense category.

These categories can be further grouped by certain attributes, the most prominent being the method's aim: some only mitigate possible attacks assuming the model is backdoored, while others only allow for detection with no mitigation method. There are also categories containing both types, sometimes as part of a single method. The second attribute is the defense's scope. A defense is either applied to the backdoored model itself, or to the data going into the model, preventing the backdoor from being triggered. A final attribute is the moment the defense is applied. Most methods are reactive, defending against already backdoored models. But some are preemptive, preventing a backdoored model from being trained in the first place. See Table 2.1 for an overview of the attributes of each category.

Since this work is based on that of Dhonthi et al. [4], it was decided to look into similar defense methods. The core of their approach is Artificial Brain Stimulation (ABS) by Liu et al. [17] (see Section 3.1.1), which is then used to detect and reverse engineer a potential trigger. Afterwards, the mitigation is performed by retraining the model with triggered images. This entire process is a reactive, model-based approach, combining both mitigation and detection. Furthermore, a requirement of any new method is a lack of access to training data, as in the industry the risk of backdoor attacks typically occurs using models from third-parties, of which the training data is often not available to the client. So, this leaves the following categories considered:

- Model reconstruction
- Trigger synthesis
- Model diagnosis

Chapter 3

Related Work

3.1 Relevant defense methods

The following is an overview of the methods most relevant to the newly proposed method, and as such are described in more detailed compared to the other related work in the Section 3.2.

3.1.1 Artificial Brain Stimulation (ABS)

ABS by Liu et al. [17] is a model diagnosis and trigger synthesis method inspired by the idea of Electronic Brain Stimulation in neuroscience, where neurons are artificially stimulated to study their behavior. ABS in a similar manner 'stimulates' the artificial neurons by manually altering a certain neuron's input and then observing the final output layer. This alteration is done based on the activation values for a small set of benign data.

From this stimulation analysis, so-called Neuron Stimulation Functions (NSFs) are derived. There exists an NSF for each neuron and label, taking in input for the given neuron and returning the final output value for the given label, with other activations based on the benign data as mentioned.

In the implementation of the method, this idea is simplified to make it computationally feasible. Points of the NSF are sampled rather than computed analytically, and the NSFs are computed over entire output channels of convolutional layers rather than over the individual neurons.

The NSFs can then be used to compute the label elevation; how much a channel can raise a label output compared to the original value given the benign data. However, having a high elevation does not necessarily imply a backdoor, since there are benign features replicated by the stimulation that could significantly raise the values of many labels. So instead, the difference between the two highest elevation values across all labels is considered instead, termed *Elevation Difference*. If one label is raised much higher than others, the possibility of this channel contributing to a backdoor is high and this channel is then added to a list of candidates.

Next, these candidates are used to reconstruct the trigger using the same set of benign images through an optimization procedure. A model that reproduces triggers with an ASR high enough is considered 'backdoored'.

See Figure 3.1 for a brief visual overview of ABS.

Shortcomings

The performance of the method is shown to be better than the state-of-the-art method at the time, Neural Cleanse [30]. However, in the years since the ABS paper was published (2019), there have been many new attacks. The constant trigger reverse-engineering approach of ABS makes it unsuited against advanced *dynamic attacks*, which apply the backdoor using different triggers for every input. Hence, attacks such as the Input-Aware Backdoor (IAB) [20], and Dynamic Attack of Salem et al. [25] will presumably fail. Furthermore, ABS also has fairly low accuracy (60%-70%) against more advanced static attacks (attacks that rely on a single trigger applied to all inputs to apply the backdoor), namely the composite attack [15], reflection attack [19] and hidden-trigger attack [24], as evaluated by Liu et al. [18], with a similar poor performance as well in the TrojAI competition [18, 10].

Another issue with the ABS method lies in its relative complexity; its method of stimulating neurons requires multiple forward passes for every channel, and then to detect

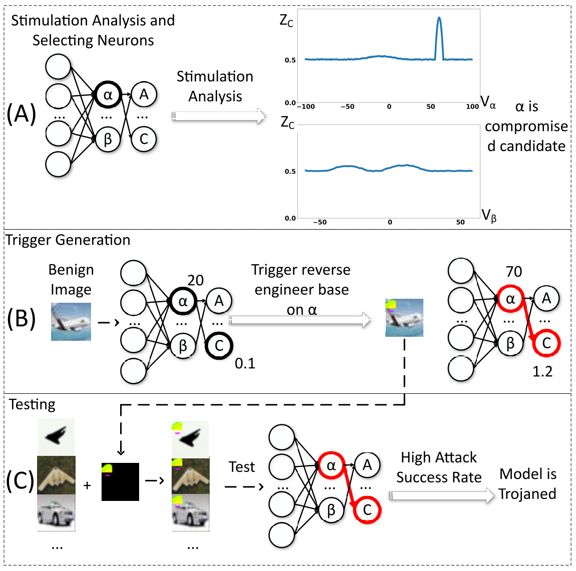


FIGURE 3.1: Overview of ABS.

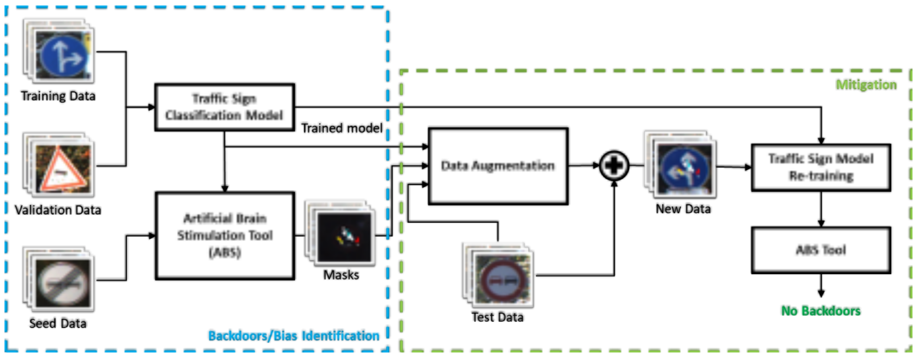

Training Data
Validation Data
Seed Data
Traffic Sign Classification Model
Trained model
Artificial Brain Stimulation Tool (ABS)
Masks
Backdoors/Bias Identification
Mitigation
Data Augmentation
New Data
Traffic Sign Model Re-training
ABS Tool
Test Data
No Backdoors

FIGURE 3.2: Overview of the method of Dhonthi et al.

candidates based on these stimulated values requires even more forward passes on benign images.

The most expensive part however, is the reverse engineering, which requires a gradient descent procedure over the set of benign images to optimize both a trigger and a mask. This then has to be done separately for each candidate.

3.1.2 Dhonthi et al.

The method of Dhonthi et al. [4] adds a model reconstruction mitigation method to the detection method of ABS. After applying ABS, the reconstructed triggers are tested by applying them to a set of test data and passing this backdoored input through the network. The corresponding predictions are then compared for images of each true class separately, forming a matrix of dimension $N_c \times N_c$, where N_c is the number of classes. The classes that are most often wrongly predicted, the number of which is defined by a hyperparameter, are then selected for retraining. The images for those classes with the reconstructed trigger applied then become a part of the training data. See Figure 3.2 for an overview of the entire method.

Shortcomings

The method generally shows a successful reduction in ASR when the number of classes selected for retraining is limited to 15. However, this was only evaluated with a patch attack on relatively small network architectures. Furthermore, retraining is rather time-consuming, especially when added onto the already complex method of ABS, and it is rather counterintuitive when the model is outsourced and trained by an external party. In this scenario, even if training data is available, it would not make much sense to retrain the model locally if initially it was outsourced.

3.1.3 Channel Lipschitz Pruning (CLP)

CLP by Zheng et al. [36] is a pruning-based reconstruction method, which relies on the upper bound of the L_2-Lipschitz constant of each channel, shortened to UCLC. For a given layer l, and channel k in network n, this is defined as:

$$||n_k^{(l)}||_{\text{Lip}} = \arg\min_L \left(||n_k^{(l)}(\mathbf{x}) - n_k^{(l)}(\mathbf{x}')||_2 \leq L||\mathbf{x} - \mathbf{x}'||_2 \right), \forall \mathbf{x} \in \mathcal{X}$$

The intuition behind the Lipschitz constant is that it can be seen as the maximum slope of a function throughout its domain. In this context it shows how sensitive the given output channel is to a change to an arbitrary input. As a backdoor attack can be seen as a small change in the initial input that causes significant changes to the final layer output, the intuition is that this phenomenon could occur within an individual channel as well.

The advantage of this approach is that it can be shown that the upper bound of the Lipschitz constant of a channel can be computed as the product of the spectral norms of the channel's weight matrix and that of preceding layers [7, 36], i.e.

$$||n_k^{(l)}||_{\text{Lip}} \leq \sigma(\mathbf{W}_k^{(l)}) \prod_{i=1}^{(l-1)} \sigma(\mathbf{W}^{(i)})$$

With the spectral norm σ defined as:

$$\sigma(\mathbf{W}) = \max_{||\mathbf{x}||_2 \neq 0} \frac{||\mathbf{W}\mathbf{x}||_2}{||\mathbf{x}||_2}$$

This is equal to the largest singular value of $\mathbf{W}$, and thus can be computed using the weight matrix alone and no sample benign data.

However, for a convolutional layer, the original four-dimensional weight tensor has to be put in a certain matrix form – a doubly-block Toeplitz matrix – which is a computationally expensive process. Hence, the method applies a simpler reshaping operation which appears to work well as an approximation. The spectral norm of the reshaped weight tensor is then used for the channel's UCLC. The channels with a high UCLC, crossing an outlier threshold based on the mean and standard deviation of all the UCLCs in their respective layers, are finally *pruned* by setting both their weights and biases to zero, in order to eliminate the backdoor. Since this threshold is determined with channels in the same layer, the product term of the spectral norm of previous layers, i.e. $\prod_{i=1}^{(l-1)} \sigma(\mathbf{W}^{(i)})$, can be ignored and only the spectral norm of the given channel has to be calculated.

Shortcomings

CLP appears to be very effective and fast, outperforming all other similar pruning-based methods. It also shows high effectiveness on more recent advanced attacks like the Input

Aware Backdoor (IAB) [20], that has a different trigger based on each input, and WaNet [21], that uses a warping transformation.

However, when evaluated on the small network of Dhonthi et al. [4], with 4 convolutional layers, CLP was found to perform poorly. Out of 10 runs, where the model was retrained on the same backdoor, there were zero truly succesful defenses by CLP. Interestingly, adding batch normalization layers to these models increased the effectiveness, but still only yielded 5 succesful runs. See Appendix A.1 for the full results.

3.1.4 Tao et al.

Tao et al. [29] propose a novel (unnamed) trigger reconstruction method, that can both find better triggers and is more efficient than existing attacks. Specifically, the methods compared are Neural Cleanse (NC) [30], and some defenses for adversarial attacks adapted for backdoor attacks, namely the method by Carlini & Wagner (CW) [2] and Universal Adversarial Perturbation (UAP) [27].

It does this by optimizing perturbation tensors, rather than separating a trigger and a mask to apply the trigger to, as methods such as ABS and NC do. These perturbation tensors, one for negative perturbations and one for positive perturbations, are then transformed using a hyperbolic tangent function before the trigger is applied, allowing for a fast optimization with accurate reconstructed triggers. As this method will be used as part of the proposed new method, further technical details are found in Chapter 4, Section 4.2.2.

Compared to NC, the method of Tao et al. generates triggers that had around 50 times less perturbed pixels for some class pairs on ImageNet, while yielding a higher ASR. The method was also twice as fast as NC on these pairs. Compared to CW and UAP, the differences in execution time, trigger size and ASR are even larger in favour of the method of Tao et al. A comparison with ABS was also made in the TrojAI competition, where the method again was consistently faster and had a higher backdoor detection accuracy than ABS.

3.2 Other related work

3.2.1 Topological Prior

Hu et al. [10] propose a trigger synthesis and model diagnosis method improving on the typical reverse-engineering approaches by introducing the concept of a *topological prior*. This consists of an component of the loss function that aims to penalise any 'loose' structures on the generated mask.

To decide whether a model is backdoored, features from the generated triggers are extracted, which are then used as input in a FC network. This network is trained on recovered triggers of clean and backdoored versions of models, intended to be generalizable on all attacks and architectures.

This method appears to have greatly more performance compared to ABS and other methods in the TrojAI competition. However, attacks other than the patch attack are not evaluated, and this method's loss function is more complex than that of ABS. The detection model would further be expensive to train.

3.2.2 Ex-Ray

Liu et al. [18] propose an improvement on top of other trigger synthesis detection methods. The method, known as Ex-Ray, aims to improve an already existing reconstructed trigger by using *Symmetric Feature Differencing.* Here feature maps of a benign image with the reverse-engineered trigger applied are compared to feature maps of a benign image whose true label is the target label of that trigger. For example, if the target label is ‘dog’, feature maps of images of ‘cat’, ‘deer’, etc. with the reverse-engineered trigger applied are compared to true feature maps of ‘dog’ images. If the trigger is ‘natural’ (e.g. antlers being a trigger for ‘deer’), the differences of the feature maps between the triggered image and the benign image of the backdoor target class should be similar to the differences of the feature maps between the triggered image and the original clean image. However, when it is indeed an intentional backdoor trigger, the feature differences will be very different, as the trigger does not resemble a feature of benign images of the target class. Hence, the Ex-Ray method aims to better separate ‘natural’ triggers from intentional harmful ones.

This method is also evaluated on the TrojAI competition and shows improved results compared to pure ABS. However, the method only provides overhead over the existing methods and is thus not suitable when seeking a faster method.

3.2.3 Adversarial Neuron Pruning (ANP)

Wu and Wang [33] propose a model reconstruction mitigation method known as Adversarial Neuron Pruning (ANP). This method prunes backdoored neurons through an optimization procedure, wherein a network’s weights are pruned by a continous mask. It then consists of two objectives. The first objective minimizes the loss of the pruned network on a set of benign data, so that the pruning will not impact the accuracy of the model severely. The second objective is to apply a perturbation to the pruned weights that maximizes the loss, simulating the effect of a backdoor. While in a real backdoor the inputs, not the weights, are perturbed, it is presumed that backdoored neurons will be dormant for benign samples, with activations near zero. By increasing the weights and biases, the neuron will return high activations for all outputs, and thus the expectation is that the backdoored neuron’s effect will then be apparent even on benign data.

ANP is shown to be effective against a wide range of attacks, including the dynamic IAB attack, and is also evaluated by third-parties as part of Backdoorbench [32]. However, when compared in the evaluation of the CLP method [36], results were not as good as those of CLP. Its assumption of backdoored neurons having near-zero activations for benign inputs might also not always hold, and it could be possible to construct an attack where the benign behaviour is realised by high activations instead. Furthermore, the optimization is rather time-consuming, especially compared to CLP. Unlike ABS, this optimization gives no insight into the trigger either, and is for mitigation only.

3.2.4 FreeEagle

Fu et al. [5] propose a model diagnosis detection method not requiring any benign data. The method works by splitting the network at a point roughly in the middle, so that in the first part the primary behaviour is feature extraction while the second part is primarily dedicated to classification of the extracted features. The feature extraction part is then ignored, and using the classifier part a *dummy Intermediate Representation* (IR) is generated for each class by maximizing the output for that class. The generated dummy IR is then forward propagated in the network and the corresponding outputs are analyzed. The idea is that for a backdoored network given benign data, there will be some trace outlier values for the backdoor target in the final output. So, when inserting the dummy IRs into the classifier part, the output is analyzed. Naturally, the output for the class the

IR was generated for should give the highest value. But, when ignoring this class, if some other class in the output has an outlier value, the model is presumed backdoored.

The detection accuracy of the method was evaluated on GTSRB with a patch, blended, and a simple feature-space trigger. On these it all showed better performance than other methods tested, including ABS. However, for this method there is no trigger reconstruction, so it gives little insight into the nature of the trigger. Furthermore, the exact layer where the model should be split into the respective parts depends on the architecture and the type of backdoor, so one might have to try this for multiple layers to verify the detection, making the method rather expensive for large networks.

Chapter 4

Methodology

Considering the shortcomings of the methods from Chapter 3, it seems necessary to come up with a new method. This chapter describes this new method in detail, along with justifying the design choices made. Some technical details on ABS and the method of Dhonthi et al. are also presented, so that they can be compared to the new method.

Since the goal is ultimately both detection and mitigation, it is desirable to integrate this efficiently in one method. The new method relies on two core ideas from the existing methods. Firstly, for detection, the aim is to reconstruct the trigger, which can be applied to a small set of benign data to determine if the network is backdoored. The second idea is for mitigation, which is to prune output channels involved in the backdoor, similar to CLP.

To integrate these ideas, a multi-step approach is used. In the first step, a list of candidate channels is selected, similar to ABS. However, unlike ABS, these candidates are selected based on their sensitivity to pruning. The intent is that the channels which are the least sensitive to pruning when considering benign input, are the channels that are potentially associated with the backdoor, as they are (near) irrelevant for benign input.

In the second step, a trigger corresponding to each candidate channel is reconstructed with a small set of benign data. Then, each trigger is evaluated with that benign data to validate it. If any channel has a valid trigger, the model is considered backdoored. At a high level this works like ABS. However, as described in Section 4.2, a somewhat different trigger reconstruction procedure is proposed, which also incorporates the method of Tao et al.

Then, a further extension to ABS is made by producing a list of ‘final candidates’. This list includes candidates for which a valid trigger has been reconstructed, but also channels that reconstruct no valid trigger on their own, but are deemed to contribute to a candidate’s trigger.

Finally, all channels in the final candidate list are pruned by setting their weights to zero (but keeping their biases intact). This should then mostly preserve the accuracy of the model but eliminate the backdoor.

Each of these steps will now be described, and comparisons against ABS are made where applicable.

4.1 Candidate selection

For the proposed new method, the ultimate goal is to prune backdoored channels. Hence, inspiration was sought from criteria for pruning methods, where the goal is to simply eliminate redundant neurons. Backdoored channels which can be pruned without impacting the accuracy of the model should be largely redundant for benign data, so this should transfer over well to detect backdoors. The chosen criterion is based on the output variance of channels, as will be described in Section 4.1.1.

As candidates, only output channels of convolutional layers are considered. When the convolutional layers are followed by a batch normalization layer, the output of this batch normalization layer is considered for candidate selection and trigger reconstruction instead. This is necessary since batch normalization changes the distribution of activations of the channels, so anything based on the channels of the preceding convolutional layer could already be made invalid by the following batch normalization.

4.1.1 Sensitivity metric

A relatively simple way of estimating a channel's importance given a set of benign input is by considering its variance over different inputs. Polyak & Wolf [22] propose two methods for pruning convolutional networks based on variance. The first, *inbound pruning*, considers individual kernel slices going from one input channel to one output channel. That is, for a convolutional 4D tensor $\mathcal{W} \in \mathbb{R}^{\mathcal{W}_{\text{in}} \times \mathcal{W}_{\text{out}} \times \mathcal{W}_{\text{w}} \times \mathcal{W}_{\text{h}}}$ – where $\mathcal{W}_{\text{in}}$, $\mathcal{W}_{\text{out}}$, $\mathcal{W}_{\text{w}}$ and $\mathcal{W}_{\text{h}}$ are the number of input channels, number of output channels, kernel width and kernel height, respectively – each slice $\mathcal{W}_{i,k} \in \mathbb{R}^{\mathcal{W}_{\text{w}} \times \mathcal{W}_{\text{h}}}$ is considered.

The criterion for a slice $\mathcal{W}_{i,k}^{(l)}$ is then defined as

$$\text{var}(||(\mathcal{W}_{i,k}^{(l)} * n_i^{(0,l-1)}(\mathbf{x})||_F)$$

where $n_i^{(0,l-1)}(\mathbf{x})$ denotes channel i of the output of the last convolutional layer l in network n, given input $\mathbf{x}$. This is then the input for the following convolutional layer l taking weights $\mathcal{W}_{i,k}^{(l)}$. $*$ denotes the convolution operation and F stands for the Frobenius norm.

The second method proposed by Polyak & Wolf is *reduce and reuse*. This is similar, except now the variance is taken across all output channels, i.e. over the three-dimensional tensor $\mathcal{W}_k \in \mathbb{R}^{\mathcal{W}_{\text{out}} \times \mathcal{W}_{\text{w}} \times \mathcal{W}_{\text{h}}}$. So, the criterion is as follows:

$$\text{var}(||(\mathcal{W}_k * n^{(0,l-1)}(\mathbf{x}))||_F)$$

Where $n^{(0,l-1)}(\mathbf{x})$ now stands for all the channels of the input to the layer.

This is the *reduce* step. Since their method prunes the entire feature map, including bias, the next *reuse* step exists to restore the functionality of the original network, by adding 1x1 convolutional layers that reconstruct the pruned channels using the remaining channels. However, this step is not necessary when used for the proposed new method as only weights will be pruned.

Hence, for the new proposed method, the criterion from the 'reduce' step is used as a sensitivity metric for output channels, estimating how redundant each channel is for benign input. A redundant channel could then possibly be backdoored, as it might only be relevant for backdoored input.

Complexity

Applied to the sample data $\mathcal{D}_S$, computing this metric requires $|\mathcal{D}_S|$ forward propagations, with additional overhead for the variance and norm computations for each output channel. This is dependent on implementation and parallelization, making this difficult to quantify in narrower terms. However, the number of forward propagations can be compared with ABS to assess which method is more complex.

4.1.2 Threshold

In order for this metric to distinguish backdoored channels from benign ones, a threshold on the metric is used. The Z-score is computed over the values of each layer to have comparable values across layers. The candidates are then selected based on being below the threshold:

$$\frac{\mathbf{v}_k^{(l)} - \text{mean}(\mathbf{v}^{(l)})}{\sigma(\mathbf{v}^{(l)})} < u_{\text{candidate}} \tag{4.1}$$

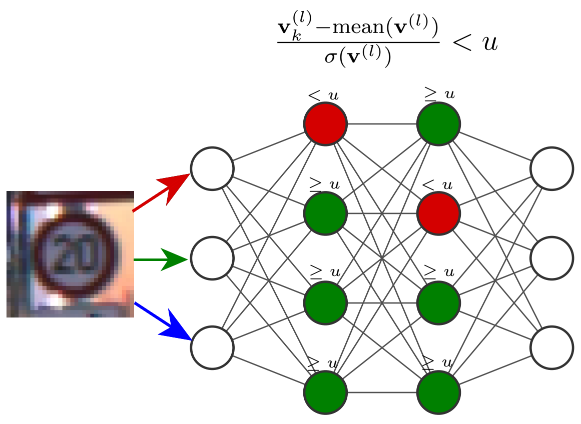

$\frac{\mathbf{v}_k^{(l)} - \text{mean}(\mathbf{v}^{(l)})}{\sigma(\mathbf{v}^{(l)})} < u$
$< u$
$\geq u$
$\geq u$
$< u$
$\geq u$
$\geq u$
$\geq u$
$\geq u$

FIGURE 4.1: Example of candidate selection. The channels below the threshold u, selected as candidates, are marked in red, while the other considered channels are marked in green.

Where $\mathbf{v}_k^{(l)}$ stands for the output variance of channel k, and $\mathbf{v}^{(l)}$ the vector with variance for all channels in layer l. $\sigma(x)$ stands for the standard deviation of x, $u_{\text{candidate}}$ is the threshold.

See Figure 4.1 for an illustration of this step. In these and all subsequent figures explaining the method, a graph of a simple FC-network is used for simplicity. To represent a CNN with this, a nodes in the hidden layer represents a whole feature map, with each incoming edge representing kernel weights $\mathcal{W}_{i,k}$ applied to the corresponding input channel of the previous layer.

4.1.3 Candidate selection with ABS

As described in Section 3.1.1, ABS computes the NSFs by sampling across a set of benign data. These are then used to compare the output of benign data to the values of the NSF, and measuring the elevation of each label. The channels with the highest Elevation Difference, the number of which is set by a hyperparameter, are selected as candidates.

The downside of this approach is that it is rather complicated; first the NSFs are computed for each channel and label using a number of samples. Then in the second step, the Elevation Difference is computed by performing inference across the benign sample data again.

Computing the NSFs requires $N_S * N_C * |\mathcal{D}_S|$ forward propagations, where N_S is the number of sampling points per benign image, N_C the total number of channels in the network, including neurons in dense layers, and $|\mathcal{D}_S|$ the number of benign images used (at least one per label).

Then for the actual candidate selection, another $N_C * |\mathcal{D}_S|$ propagations are necessary, as the benign outputs are compared to the NSF for each channel and image. Thus, in total there are $(N_S + 1) * N_C * |\mathcal{D}_S|$ propagations needed. Again, depending on hardware configuration and the values of N_S, N_C and $|\mathcal{D}_S|$, a significant portion of the inference can be parallelized, so the exact time complexity is variable. But in terms of forward propagations, this is much more expensive than the variance-based method that requires just $|\mathcal{D}_S|$ propagations. The candidate selection of ABS is by far not as costly as the reverse engineering, but it can still be quite time-consuming for large networks.

4.2 Trigger reconstruction

After selecting the candidates, they are put in reverse layer order: from the deepest layers to the shallowest ones. What follows next is the trigger reconstruction. This is split into two separate optimization steps. The network $n^{(l)}(\mathbf{x})$, where $\mathbf{x}$ is the input data, can be split up into $n^{(0,l)}(\mathbf{x})$ denoting the output feature map of layer l, and $n^{(l)}(\mathcal{O})$, where $\mathcal{O} = n^{(0,l)}(\mathbf{x})$, denoting the output of the network starting at layer l, given intermediate feature maps $\mathcal{O}$ as input.

For a candidate channel k and layer l, an *intermediate trigger* will first be constructed, to be applied on an intermediate feature map $\mathcal{O}_k$. This intermediate trigger is then used to reconstruct the *original trigger* using the first part of the network, $n^{(l)}(\mathbf{x})$. The methodology of reconstructing the intermediate trigger will first be described, then that of the original trigger.

4.2.1 Intermediate trigger

Backdoor behavior is generally largely constrained to a few channels, as shown by the relative success of the defense methods focused on specific channels (e.g. ABS candidate selection, CLP and ANP pruning). So, the small perturbation of the original trigger has to be reflected in the intermediate layers up to the backdoored channels. See Figure 4.2a, where the presence of the original trigger can still be clearly inferred from intermediate feature maps. In contrast, the feature map of a backdoored channel would be significantly different for a backdoored input compared to a benign input, allowing for any input to be misclassified to the backdoored target label. See Figure 4.2b. However, even these large changes are still fairly regular across the input, and can thus be seen as 'triggers' of their own.

Hence, since the candidate channels could possibly be backdoored we aim to find intermediate triggers akin to the rightmost column of Figure 4.2b. This involves optimizing over only a part of the network, which can especially save time for candidates that do not appear to be backdoored. If the intermediate trigger is not valid, the next candidate can be considered, and thus only this optimization over a part of the network is needed. Otherwise, if the intermediate trigger is satisfactory, the original trigger can be reconstructed using the intermediate trigger.

Now we elaborate on this formally. The intermediate trigger, known as $\mathbf{t}_I$, is applied as:

$$\mathcal{T}_k^{(l)} = n^{(0,l)}(\mathbf{x})_k + \mathbf{t}_I$$

and

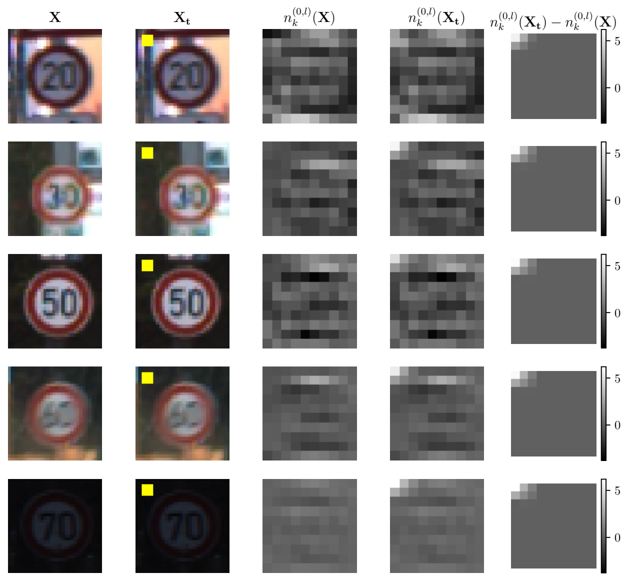
$$\mathcal{T}_i^{(l)} = n^{(0,l)}(\mathbf{x})_i, \forall i \in \{\{1 \dots N_l\} - \{k\}\}$$

Here, the subscript denotes an output channel i of $n^{(0,l)}(\mathbf{x})$ and $\mathcal{T}^{(l)}$. So, the trigger is only applied for the candidate channel and the rest is left as it was for benign input.

The limitation of this candidate selection method is that it does not result in any target label for the candidate. Hence, in order to reconstruct an accurate intermediate trigger, a different way of retrieving this is necessary. The trigger is initially reconstructed by minimizing $\mathbf{t}_I$ over the following loss function:

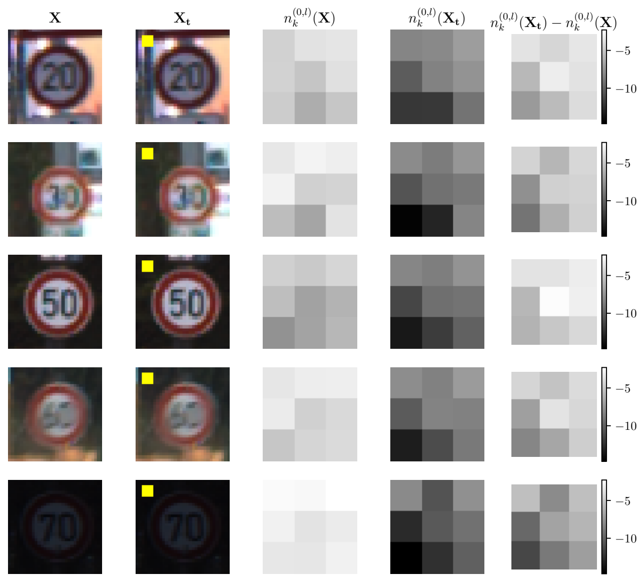
$$\mathcal{L}_{\text{im1}}(\mathbf{t}_I) = -\text{CE}(n^{(l)}(\mathcal{T}^{(l)}), \mathbf{y})$$

Here 'CE' stands for the Cross-Entropy loss (see Equation 2.2). This maximizes the loss with the true labels $\mathbf{y}$, ensuring the trigger moves the output away from the true values.



$\mathbf{X}$
$\mathbf{X_t}$
$n_k^{(0,l)}(\mathbf{X})$
$n_k^{(0,l)}(\mathbf{X_t})$
$n_k^{(0,l)}(\mathbf{X_t}) - n_k^{(0,l)}(\mathbf{X})$
5
0

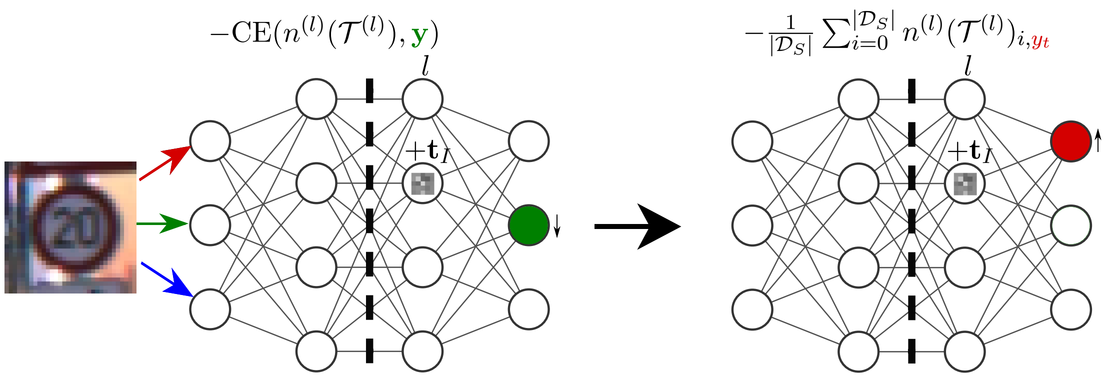
(A) On a non-backdoored channel. The difference between the feature maps caused by the trigger is quite similar to the original trigger, in both scale and location.



$\mathbf{X}$
$\mathbf{X_t}$
$n_k^{(0,l)}(\mathbf{X})$
$n_k^{(0,l)}(\mathbf{X_t})$
$n_k^{(0,l)}(\mathbf{X_t}) - n_k^{(0,l)}(\mathbf{X})$
−5
−10

(B) On a backdoored channel. The difference between the feature maps now is not constrained to the upper-left corner and is significant across the image.

FIGURE 4.2: Visualization of the impact of the trigger on a non-backdoor channel compared to a backdoored channel. From left to right, the columns show a clean image, the same image with a backdoor applied, then the intermediate output of the clean image and backdoored image respectively. The rightmost column shows the difference between the benign and backdoored intermediate output.



$-\mathrm{CE}(n^{(l)}(\mathcal{T}^{(l)}), \mathbf{y})$
l
$+\mathbf{t}_I$
$-\frac{1}{|\mathcal{D}_S|}\sum_{i=0}^{|\mathcal{D}_S|} n^{(l)}(\mathcal{T}^{(l)})_{i,y_t}$
l
$+\mathbf{t}_I$

FIGURE 4.3: Optimization of the intermediate trigger t_I. The intermediate trigger is applied by adding it to a certain channel. Then firstly, the values for the true label (green) are minimized w.r.t. t_I. Based on the results from this, the target label (red) can be established and maximized to yield a more targeted trigger. Only the part of the network to the right of the dashed line is considered in the optimization.

After doing this for 20 epochs, the target label can be established as:

$$y_t = \arg\max_j \frac{1}{|\mathcal{D}_S|} \sum_{i=0}^{|\mathcal{D}_S|} n^{(l)}(\mathcal{T}^{(l)})_{i,j} - n(\mathcal{D}_S)_{i,j}$$

Where $|\mathcal{D}_S|$ is the number of sample data images by which $n^{(l)}(\mathcal{T}^{(l)})_{i,j}$ is indexed row-wise using i, denoting the output for the different image, and each column j corresponding to a label. Thus, the target label is the one that has the largest difference between triggered input and benign input, on average over all sample images.

With this target label, the trigger is further refined using the following loss function:

$$\mathcal{L}_{\mathrm{im2}}(\mathbf{t}_I) = -\frac{1}{|\mathcal{D}_S|} \sum_{i=0}^{|\mathcal{D}_S|} n^{(l)}(\mathcal{T}^{(l)})_{i,y_t}$$

This maximizes the output for the target label y_t, making the trigger focused on a specific target label, rather than only moving away from the true label as in $\mathcal{L}_{\mathrm{im1}}$.

Finally, the reconstructed intermediate trigger is validated by evaluating the average increase of the target label. If the trigger raises the target label enough, while not raising other labels significantly, it is considered valid. The threshold for this is determined at an average raise of 0.3, with the highest label raise having to be 1.3 times more than the next highest label, so that the trigger is properly targeted.

See Figure 4.3 for an overview of this step.

4.2.2 Original trigger

When the intermediate trigger is a valid trigger, the original trigger is reconstructed using the intermediate trigger as a ground truth, using the method of Tao et al. [29] The reconstruction method of ABS, consists of optimizing a trigger and mask, while constraining the L_1 norm of the mask. The correlation between the trigger and the mask can generate a complex loss landscape, which can easily get the optimization stuck in local minima. Hence, Tao et al. do not make use of a mask, only a trigger perturbation over the entire image. As patch triggers only perturb a small portion of the image, a simple approach of optimizing the perturbations and then directly adding them to the pixels will not suffice, since without a mask, it would be difficult to constrain the many pixels that are not part of the trigger to have zero perturbation. Tao et al. solve this issue by applying a hyperbolic tangent function over the perturbation tensor. With its long tails, it allows for a large portion of the optimized trigger's range to result in near-zero perturbation. With the lower tail of the function limited at zero, two variables are necessary, one to represent a positive and another to represent a negative perturbation. See Figure 4.4 for the shape of this perturbation function. The perturbations are applied to input by simply adding the positive perturbation to the input, then subtracting the negative perturbation, and finally clipping the values between 0 and 1.

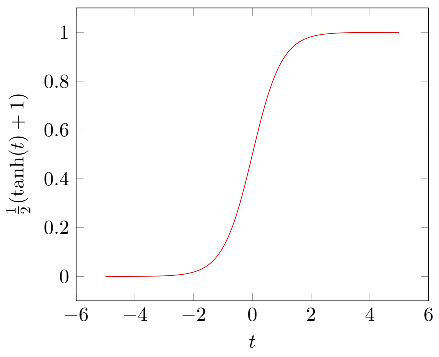


FIGURE 4.4: The perturbation function $\frac{1}{2}(\tanh(t)+1)$, used in the method of Tao et al. Here, t represents the optimized perturbation variable, which results in near zero perturbation for negative values of t, ascending to the maximum of 1 for high values.

The hyperbolic tangent functions applied over these perturbations then serve both to apply the reconstructed trigger to an image and as a smoothly differentiable penalty on the size of the trigger, by summing over the output of these functions for each pixel.

So, the triggered input is defined as:

$$\mathbf{x_t} = \text{clip}\left(\mathbf{x} + \frac{1}{2}\left(\tanh(\mathbf{t}_+) - \tanh(\mathbf{t}_-)\right)\right)$$

Where $\mathbf{t}_+$ is the positive perturbation, $\mathbf{t}_-$ the negative perturbation, and $\text{clip}(\mathbf{x})$ a function that clips the values in $\mathbf{x}$ between 0 and 1. Since here the trigger is applied to an entire input image instead of a single feature map, the trigger can have more than one channel (3 for an RGB image).

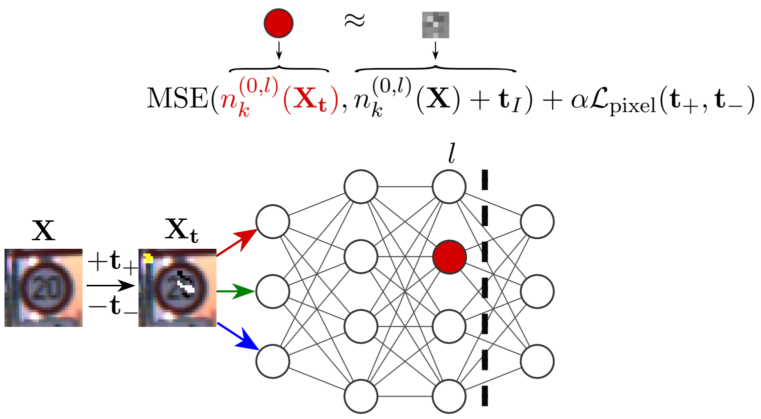
This trigger is then reconstructed using the following loss function:

$$\mathcal{L}_{\text{or}}(\mathbf{t}_+, \mathbf{t}_-, \mathbf{t}_I) = \text{MSE}(n_k^{(0,l)}(\mathbf{x_t}), n_k^{(0,l)}(\mathbf{x}) + \mathbf{t}_I) + \alpha\mathcal{L}_{\text{pixel}}(\mathbf{t}_+, \mathbf{t}_-)$$

Where 'MSE' stands for the Mean Squared Error (see Equation 2.1), with the squared difference applied per pixel. $\mathcal{L}_{\text{pixel}}$ is defined as:

$$\mathcal{L}_{\text{pixel}}(\mathbf{t}_+, \mathbf{t}_-) = \sum_{i=1,j=1}^{i\le w, j\le h} \max_c \left(\frac{1}{2}\tanh\left(\frac{\mathbf{t}_+}{10}\right) + \frac{1}{2}\right)_{c,i,j} + \sum_{i=1,j=1}^{i\le w, j\le h} \max_c \left(\frac{1}{2}\tanh\left(\frac{\mathbf{t}_-}{10}\right) + \frac{1}{2}\right)_{c,i,j}$$

where w and h stand for the image width and height, respectively, while c denotes an image channel.



$\approx$
$\mathrm{MSE}(\underbrace{n_k^{(0,l)}(\mathbf{X_t})}, \underbrace{n_k^{(0,l)}(\mathbf{X}) + \mathbf{t}_I}) + \alpha\mathcal{L}_{\text{pixel}}(\mathbf{t}_+, \mathbf{t}_-)$
l
$\mathbf{X}$
$\mathbf{X_t}$
$+\mathbf{t}_+$
$-\mathbf{t}_-$

FIGURE 4.5: Optimization of original trigger. The goal of the optimization is to get the feature map (marked in red), given the reconstructed original trigger, as close as possible to the feature map triggered with the intermediate trigger from the previous step. Only the part of the network to the left of the dashed line is considered in the optimization.

The $\mathcal{L}_{\text{pixel}}$ term serves to penalize the trigger size, with its weight α dynamically adjusted throughout the optimization based on the ASR and trigger size. The division by 10 reduces the slope of the curve in order to improve the optimization, which would very quickly reach the near-zero gradient tails otherwise.

If the ASR is high enough or the trigger is too big, α is halved after 10 consecutive epochs of this occurring. If this is not the case for 10 consecutive epochs α is doubled. This ensures the trigger found is both effective and small. The trigger size is determined as:

$$\sum_{i=1,j=1}^{i\leq w,j\leq h} \max_c \frac{1}{2}\left(\tanh\left(\mathbf{t}_+\right) - \tanh\left(\mathbf{t}_-\right)\right)_{c,i,j}$$

A size above a proportion s_α of the image size $w \times h$ is then considered 'too big' for the purposes of the α adjustment.

Thus, the objective of this is to minimize the difference between the intermediate feature map when given the original trigger $n_k^{(0,l)}(\mathbf{x_t})$ and that same feature map with the intermediate trigger applied, $n_k^{(0,l)}(\mathbf{x}) + \mathbf{t}_I$, while limiting the size of the trigger through $\mathcal{L}_{\text{pixel}}$.

After enough epochs of this optimization, a validation is again performed on this trigger. For this, the Attack Success Rate (ASR) of the trigger is tested, just like ABS, as follows:

$$\mathrm{ASR}(\mathbf{t}_+, \mathbf{t}_-) = \frac{\sum_{\mathbf{x}\in\mathcal{D_S}} \mathbf{1}(\arg\max_i(n(\mathbf{x_t})_i) = y_t)}{|\mathcal{D_S}|}$$

Where y_t stands for the target label, and $n(\mathbf{x})_i$ represents the ith label of the output vector $n(\mathbf{x})$.

If the ASR is above the given threshold $\mathrm{ASR}_{\text{valid}}$ and below a certain size s_{valid}, it is a valid trigger for the channel.

See Figure 4.5 for an overview of this step.

4.2.3 Complexity

The complexity of reverse engineering is again difficult to analyze in narrow terms, since this is dependent on the layers of the candidates. It will be less complex to optimize the intermediate trigger for candidates deeper in the network, while the opposite holds for reconstructing the original trigger, where it would be fastest to do so in the first layers.

Furthermore, the approach of reconstructing two triggers can also make the time complexity different for each candidate. If an intermediate trigger is determined to be invalid, there is no reconstruction for the original trigger. So in this case, the reverse engineering will only be performed for the intermediate triggers.

Note that the list of candidates is also reduced when a candidate is found to contribute to another candidate's trigger, as will be described in Section 4.3. Which of these two scenarios would be faster – either evaluating all candidates and finding no valid intermediate trigger, or evaluating a single candidate and finding a valid original trigger that removes all other candidates – is dependent on where that particular candidate is in the network and the total number of candidates, both of which can be different for every network.

So again it becomes difficult to analyze the complexity theoretically. However, in most cases, the intermediate trigger is faster to optimize than the original trigger, since it only consists of one variable of one channel, at a generally smaller feature map size, while the original trigger is composed of two variables consisting of three channels (if RGB) and the full image size. Furthermore, the intermediate trigger is optimized for only 100 epochs, while the original trigger is optimized for 1000. Hence, if by reconstructing the intermediate trigger, a candidate can be determined as invalid without a need to reconstruct the original trigger, it will provide great benefit. If this never occurs, it will provide a small overhead on top of reconstructing the original triggers at most.

The ASR computation at the end of each epoch can however add an extra cost to reconstructing the original trigger, as it requires additional forward propagation through the entire network. However it is possible to simply disable it, along with the associated α adjustment, at the expense of less accurate triggers.

4.2.4 Trigger reconstruction in ABS

With ABS, the trigger reconstruction is also done using a gradient descent optimization. Firstly, the trigger function consists of a trigger and a mask applied as follows:

$$\mathbf{x_t} = \mathbf{x} \odot (1 - \mathbf{m}) + \mathbf{t} \odot \mathbf{m}$$

where $\odot$ denotes the Hadamard product, $\mathbf{x}$ an input image, $\mathbf{m}$ a mask with values constrained in $(0, 1)$ and $\mathbf{t}$ the trigger containing pixel values to be replaced within the mask.

Then the core of the loss function for reconstructing the trigger is:

$$-c_1 n_k^{(l)}(\mathbf{x_t}) + c_2 \sum_{i=1,\, i \neq k}^{N_l} (n_i^{(l)}(\mathbf{x}) - n_i^{(l)}(\mathbf{x_t}))$$

Where $n_k^{(l)}(\mathbf{x})$ denotes the output feature map of layer l at channel k, N_l the number of channels in layer l, and c_1 and c_2 are constant weights. This means the feature map of a candidate channel w.r.t. an input is maximized, while for the other channels in the same layer, the difference between their output for benign input and their output for triggered input is minimized. The idea is that the main effect of the trigger is a high output of the candidate channel, while the output for other channels is mostly preserved.

After a desired number of epochs, the trigger that results from this procedure is evaluated on the benign images in order to decide its effectiveness. If a channel has an ASR above a certain amount, it is considered a backdoored channel and the model is detected as backdoored.

ABS also provides some support for feature-space attacks by adding the Structural Similarity Index Measure (SSIM) [31] in the loss function and using a different trigger function that does not use a mask. However, this is limited to attacks that can be described by one layer of transformation. A further issue is that one has to manually decide whether the trigger is a part of a feature-space attack or a pixel-space one, so without prior knowledge of this one has to apply the method twice to test both attacks. The method of Tao et al. shows some detection success on WaNet [21], which is also a feature-space attack, but it does not have any functionality to reconstruct these triggers accurately. Thus, this is still a point lacking in the new method. Additionally, just like ABS, the new method also is not built around dynamic attacks such as the Input-Aware Backdoor attack.

The complexity of this is again difficult to analyze. While the number of candidates is constant and there is always only one optimization per candidate, the complexity still depends on the position of the candidates inside the network. However, when the candidates of both methods are the same, it can be expected that the new method will be slightly slower at worst, while having potential to be much faster, as discussed in Section 4.2.3.

4.3 Finding channels associated with a trigger

Another metric which more explicitly considers a channel's contribution to a network's backdoor behavior is the *Trigger Activated Change* (TAC). This metric, used by the authors of CLP, can be used as to measure the sensitivity of a channel to a trigger. It is defined for a channel k in layer l and trigger $\mathbf{t}$ as follows:

$$\mathrm{TAC}_k^{(l)}(\mathbf{t}) = \frac{1}{|\mathcal{D}_S|} \sum_{\mathbf{x} \in |\mathcal{D}_S|} ||n_k^{(0,l)}(\mathbf{x}) - n_k^{(0,l)}(\mathbf{x_t})||_F$$

For each valid trigger and each candidate channel k of layer l, the TAC is computed. This is then normalized over each layer by taking the Z-score. Channels above a threshold u_{prune} are then selected as participating in the same backdoor, so as a formula:

$$\frac{\mathrm{TAC}_k^{(l)}(\mathbf{t}) - \mathrm{mean}(\mathrm{TAC}^{(l)}(\mathbf{t}))}{\sigma(\mathrm{TAC}^{(l)}(\mathbf{t}))} \geq u_{\text{prune}}$$

Where $\mathrm{TAC}^{(l)}(\mathbf{t}))$ stands for the vector containing the TAC of all channels in layer l corresponding to trigger $\mathbf{t}$, σ denotes the standard deviation. Any channel with a TAC Z-score above this threshold is added to the final candidate list. These channels do not need to have their triggers reconstructed, speeding up the method.

The effectiveness of using TAC as a pruning criterion can be tested by pruning based on TAC derived from a real trigger. In Appendix A.2, it can be seen that this method, while not perfect, was very successful in 8 of 10 runs, reducing the ASR to almost zero while barely losing accuracy.

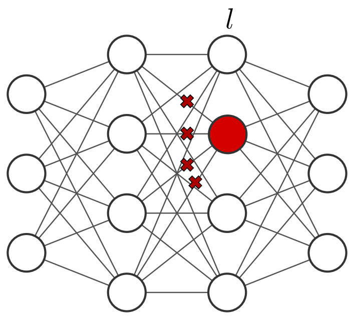


FIGURE 4.6: Pruning the backdoored channel's weights.

4.4 Pruning final candidates

All channels in the final candidate list as determined in Section 4.3 then have their weights set to zero, effectively pruning them. The bias is preserved since the candidate selection metric can only identify redundancy of information carried by weights, as the bias will be the same for any input. So, the bias might be necessary to preserve the effectiveness of the network. However, by pruning weights all the distinguishing capabilities of the backdoor channel will be gone, effectively eliminating its use from the network. See Figure 4.6 for a visualization of this step.

Finally, see Algorithm 1 for a pseudocode implementation of the entire method, and Figure 4.7 for a total overview combining the images of all the steps.

4.5 Evaluation

Like in the evaluation of the method of Dhonthi et al., the method is tested on backdoored models trained on the GTSRB dataset. On multiple architectures, both the ASR, execution time, and the accuracy of reconstructed triggers is tested. This is described in detail in Chapter 5.2.

Algorithm 1: The proposed defense method.

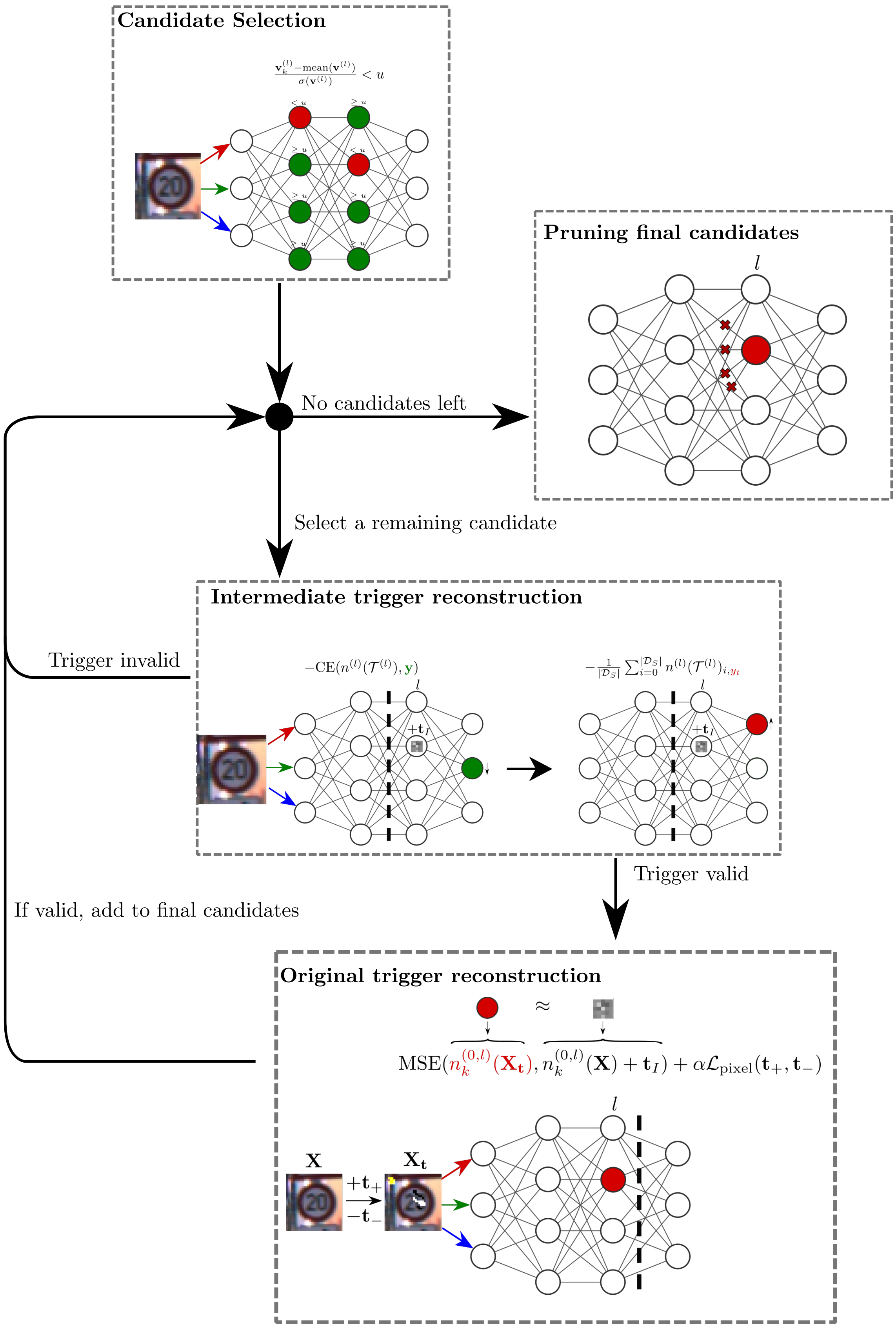
```
Data: A possibly backdoored neural network $n$, benign data $\mathbf{X}$ with
      corresponding labels $\mathbf{y}$.
Result: $n$ with backdoor eliminated.
begin
    /* Candidate selection (Section 4.1)                          */
    $S \leftarrow$ out_var_z($n$)              /* Z-score of output variances */

    candidates $\leftarrow \emptyset$
    final_candidates $\leftarrow \emptyset$
    for layer $l \in n.layers$ do
        new_candidates $\leftarrow \{\{l, k\} \mid k$ is a channel of $l$, $S[l][k] < u_{\text{candidate}}\}$
        add new_candidates to candidates
    sort candidates from deepest layers to shallowest layers

    /* Trigger reconstruction (Section 4.2)                       */
    for $\{l, k\} \in$ candidates do
        /* Algorithm 2                                            */
        $\{\mathbf{t}_I$, im_trigger_out, $y_t\} \leftarrow$ find_im_trigger($\mathbf{X}, \mathbf{y}, l, k$)
        raise $\leftarrow$ mean(im_trigger_out $- n(\mathbf{X}), 1)$ sorted in descending order
        if im_trigger_out$[:, \mathbf{y}] - n(\mathbf{X})[:, y_t] > 0.3$ and raise$[0] > 1.3 *$ raise$[1]$ then
            /* Intermediate trigger valid                         */
            $\{\mathbf{t}_+, \mathbf{t}_-\} \leftarrow$ find_or_trigger($\mathbf{X}, \mathbf{t}_I$)      /* Algorithm 3 */

            trigger_size $\leftarrow \sum \max\left(\frac{1}{2}\left(\tanh(\mathbf{t}_+) - \tanh(\mathbf{t}_-)\right), 0\right)$
            if ASR($\mathbf{t}_+, \mathbf{t}_-, \mathbf{X}, \mathbf{y}, y_t$) $>$ ASR$_{\text{valid}}$ and trigger_size $< s_{\text{valid}}$ then
                add $\{l, k\}$ to final_candidates
                for all $\{l', k'\}$ in $n$ do
                    if TAC_z($l', k'$) $> u_{\text{prune}}$ then
                        /* Channel contributes to backdoor (Section 4.3)
                           */
                        add $\{l', k'\}$ to final_candidates
    prune final_candidates by setting weights to zero         /* Section 4.4 */
```

Algorithm 2: Reconstruction of intermediate trigger.

```
def find_im_trigger ($\mathbf{X}$, labels, $l$, $k$):
    Initialize $\mathbf{t}_I$
    for 20 epochs do
        $\mathcal{T}^{(l)} \leftarrow$ apply_im_trigger($\mathbf{X}, \mathbf{t}_I$)
        im_trigger_out $\leftarrow n^{(0,l)}(\mathcal{T}^{(l)})$
        loss_1 $\leftarrow -$cross_entropy(im_trigger_out, labels)
        Optimize loss_1 with Adam optimizer and update $\mathbf{t}_I$
    $y_t \leftarrow$ most frequently occurring label in im_trigger_out
    for 100 epochs do
        $\mathcal{T}^{(l)} \leftarrow$ apply_im_trigger($\mathbf{X}, \mathbf{t}_I$)
        im_trigger_out $\leftarrow n^{(0,l)}(\mathcal{T}^{(l)})$
        loss_2 $\leftarrow -$mean($n^{(0,l)}$(im_trigger_out)$[:, y_t]$)
        Optimize loss_2 with Adam optimizer and update $\mathbf{t}_I$
    return $\mathbf{t}_I$, im_trigger_out, $y_t$
```

Algorithm 3: Reconstruction of original trigger.

```
def find_or_trigger ($\mathbf{X}$, labels, $l$, $k$, $\mathcal{T}^{(l)}$, $y_t$):
    Initialize $\mathbf{t}_+$ and $\mathbf{t}_-$
    for 1000 epochs do
        $\mathbf{X}_\mathbf{t} \leftarrow \texttt{clip}\left(\mathbf{X} + \frac{1}{2}\left((\tanh(\mathbf{t}_+) + 1) - (\tanh(\mathbf{t}_-) + 1)\right)\right)$
        l_pixel $\leftarrow \sum \texttt{max}\left(\frac{1}{2}\left(\frac{\tanh(\mathbf{t}_+)}{10} + 1\right), 0\right) + \sum \texttt{max}\left(\frac{1}{2}\left(\frac{\tanh(\mathbf{t}_-)}{10} + 1\right), 0\right)$
        loss $\leftarrow \texttt{MSE}\left(n^{(l)}(\mathbf{X}_\mathbf{t})[:, c], \mathcal{T}^{(l)}[:, c]\right) + \alpha *$ l_pixel
        trigger_size $\leftarrow \sum \texttt{max}\left(\frac{1}{2}\left(\tanh(\mathbf{t}_+) - \tanh(\mathbf{t}_-)\right), 0\right)$
        Optimize loss with Adam optimizer and update $\mathbf{t}_+$ and $\mathbf{t}_-$
        if ASR($\mathbf{t}_+, \mathbf{t}_-, \mathbf{X}$, labels, $y_t$) $>$ ASR$_\alpha$ or trigger_size $> s_\alpha *$ img_size for 10
          epochs then
            $\alpha \leftarrow \alpha * 2$
        else if ASR($\mathbf{t}_+, \mathbf{t}_-, \mathbf{X}$, labels, $y_t$) $<$ ASR$_\alpha$ and trigger_size $< s_\alpha *$ img_size for
          10 epochs then
            $\alpha \leftarrow \alpha / 2$
    return $\mathbf{t}_+$, $\mathbf{t}_-$
```



Candidate Selection
$\frac{\mathbf{v}_k^{(l)} - \text{mean}(\mathbf{v}^{(l)})}{\sigma(\mathbf{v}^{(l)})} < u$
$< u$
$> u$
Pruning final candidates
l
No candidates left
Select a remaining candidate
Intermediate trigger reconstruction
Trigger invalid
$-\text{CE}(n^{(l)}(\mathcal{T}^{(l)}), \mathbf{y})$
l
$+\mathbf{t}_I$
$-\frac{1}{|\mathcal{D}_S|}\sum_{i=0}^{|\mathcal{D}_S|} n^{(l)}(\mathcal{T}^{(l)})_{i,y_t}$
Trigger valid
If valid, add to final candidates
Original trigger reconstruction
$\approx$
$\text{MSE}(n_k^{(0,l)}(\mathbf{X}_\mathbf{t}), n_k^{(0,l)}(\mathbf{X}) + \mathbf{t}_I) + \alpha\mathcal{L}_{\text{pixel}}(\mathbf{t}_+, \mathbf{t}_-)$
$\mathbf{X}$
$\mathbf{X}_\mathbf{t}$
$+\mathbf{t}_+$
$-\mathbf{t}_-$

FIGURE 4.7: Overview of the new method, as a combination of Figures 4.1, 4.3, 4.5 and 4.6.

Chapter 5

Evaluation

This chapter describes the experiments performed to evaluate the method, and the corresponding results.

5.1 Evaluation methodology

5.1.1 GTSRB dataset

As the method of Dhonthi et al. was evaluated on this dataset, the German Traffic Sign Recognition Benchmark (GTSRB) [28] dataset is used for the experiments. It consists of images of German traffic signs in varying image sizes, labelled with 43 classes. There are 39 209 training images and 12 630 testing images.

For use in models, all images are resized to 32 by 32 pixels. To be consistent with the methodology of Dhonthi et al., 80% of the train images are used as actual training data, with the remainder set aside for validation. Thus, there are 31 367 images used for training and 7842 for validation.

43 images, one for each label, are used as sample data for the defense methods. These images are the same as those used as sample data in the experiments of Dhonthi et al.

5.1.2 Network architectures

A *network architecture* is a specific configuration of neural network, of which an instance is a model. The architectures used are firstly those used by Dhonthi et al., referred to there as $\mathcal{N}_{SN}$, $\mathcal{N}_{MN}$ and $\mathcal{N}_{LN}$, and referred to as Small, Medium and Large from here on, respectively. Since the existence of batch normalization layers had a significant effect on the effectiveness of CLP (see Appendix A.1), alterations of these models having BN layers after each convolutional layer are also used (SmallBN, MediumBN, LargeBN). The main structure of these architectures consists of convolutional layers followed by max pooling layers, with the number of layers and feature maps depending on the size of the architecture.

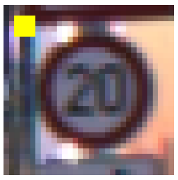
Furthermore, the Pytorch implementations of Resnet-18[1] [9] and VGG11 [16] including batch normalization[2] are used. See Table 5.1 for an overview of each architecture.

Architecture	**#Conv layers**	**#Feature maps**	**#Parameters**
Small	4	72	30 203
SmallBN	4	72	30 347
Medium	5	160	130 091
MediumBN	5	160	130 411
Large	5	320	516 139
LargeBN	5	320	516 779
Resnet-18	20	4800	11 198 571
VGG-11	8	2752	128 948 011

TABLE 5.1: Total number of convolutional layers, feature maps (output channels) and trained parameters in the used architectures.

[1] `https://pytorch.org/vision/stable/models/generated/torchvision.models.resnet18.html`
[2] `https://pytorch.org/vision/stable/models/generated/torchvision.models.vgg11_bn.html`

(A) Benign orignal image.


(B) With patch trigger.

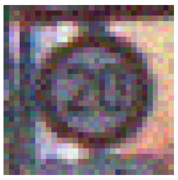

(C) With blended trigger.

FIGURE 5.1: The same image with each attack applied.

Symbol	**Description**	**Value**
$u_{\text{candidate}}$	Candidate selection threshold	-2
u_{prune}	TAC pruning threshold	2
s_α	Max size for adjusting α as proportion of image size	0.20
s_{valid}	Max size for a trigger to be considered valid	0.25
ASR_α	Max ASR for adjusting α	0.90
$\text{ASR}_{\text{valid}}$	Max ASR for a trigger to be considered valid	0.85

TABLE 5.2: Hyperparameters used in the new method.

5.1.3 Training with backdoors

As mentioned, each model is trained on the GTSRB dataset. For all the experiments, a simple patch attack (BadNets [8]) and the blended attack [3] are used as backdoors. The patch trigger consists of a 2x2 pixel yellow square placed in the upper-left corner of the image. See Figure 5.1b. The blended attack trigger consists of uniformly distributed random pixels, spread out across the image, with a transparency factor of $\alpha = 0.2$. See Figure 5.1c. For each backdoor attack, multiple models of each architecture (as described in Section 5.1.2) are trained, with the initial weights as the only difference between models of the same architecture trained on the same backdoor. For VGG11 and Resnet-18, 5 models are trained for each backdoor, while for the other architectures 25 models are trained. This is done since the effectiveness of all tested defense methods seem to vary heavily depending on this initialization. There are less VGG11 and Resnet-18 models trained due to the long training time on these architectures compared to the smaller architectures of Dhonthi et al.

Each of these backdoors are applied to 10% of the training data, using target label 14 (stop sign). The models are trained for 15 epochs, using the Adam optimizer with a learning rate of 10^{-3}, except for training the VGG-11 models, where a smaller learning rate of 10^{-4} is used, as the backdoor was not successfully injected otherwise.

5.1.4 Defense methods

The new method, the method of Dhonthi et al., and CLP are applied to all the backdoored models, forming the final results. The exception is VGG11, for which there are no results on the method of Dhonthi et al., as there were issues converting the model to work with that method.

The hyperparameters for the new method are described in Table 5.2. For the method of Dhonthi et al. they are described in Table 5.3. Finally, for CLP there is just the pruning threshold $u_{\text{CLP}} = 3$, per the suggestion of the authors of CLP [18].

Symbol	**Description**	**Value**
$N_{\text{candidates}}$	Number of candidates	20
ASR_{ABS}	Max ASR for a trigger to be considered valid	0.85
top_p	Number of classes considered for retraining	15

TABLE 5.3: Hyperparameters used in the method of Dhonthi et al.

5.2 Results

The results are grouped by the research subquestions as described in Section 1.1.

5.2.1 SQ1: Pruning effectiveness

For each architecture and backdoor, the mean of the ASR and accuracy over all models of that architecture and backdoor is reported. Due to the high variance of the performance on each model iteration, another metric evaluated is the number of models where the defense can be considered successful. This is defined as a model having an accuracy reduction less than 0.25, while having an ASR reduction greater than 0.75. The results are shown in Table 5.4.

Architecture	**Backdoored**		**CLP**			**Dhonthi et al.**			**New method**		
	Acc	ASR	Acc ↓	ASR ↓	Successful	Acc ↓	ASR ↓	Successful	Acc ↓	ASR ↓	Successful
Small	0.84	0.99	0.00	0.00	0/25	0.00	0.27	**7/25**	0.37	0.24	0/25
Medium	0.89	1.00	0.05	0.00	0/25	0.22	0.13	**1/25**	0.11	0.11	0/25
Large	0.92	1.00	0.08	0.00	0/25	0.14	0.56	**14/25**	0.10	0.01	0/25
SmallBN	0.90	1.00	0.06	0.00	0/25	0.00	0.04	**1/25**	0.43	0.68	0/25
MediumBN	0.95	1.00	0.11	0.02	0/25	0.00	0.02	0/25	0.28	0.36	**1/25**
LargeBN	0.96	1.00	0.12	0.00	0/25	0.00	0.00	0/25	0.22	0.52	**6/25**
Resnet-18	0.90	1.00	0.01	0.00	0/5	-0.01	0.19	1/5	0.03	0.69	**4/5**
VGG11	0.93	1.00	-	-	-	-	-	-	0.01	0.71	**4/5**

(A) Patch attack.

Architecture	**Backdoored**		**CLP**			**Dhonthi et al.**			**New method**		
	Acc	ASR	Acc ↓	ASR ↓	Successful	Acc ↓	ASR ↓	Successful	Acc ↓	ASR ↓	Successful
Small	0.82	0.85	0.00	0.00	0/25	0.25	0.00	0/25	0.71	0.84	0/25
Medium	0.89	0.96	0.06	0.00	0/25	0.30	0.89	**10/25**	0.64	0.93	1/25
Large	0.90	0.99	0.08	0.00	0/25	0.19	0.81	**17/25**	0.41	0.51	0/25
SmallBN	0.90	0.99	0.08	0.00	0/25	-0.01	0.08	**2/25**	0.54	0.92	1/25
MediumBN	0.94	1.00	0.12	0.00	0/25	-0.01	0.36	**9/25**	0.28	0.42	3/25
LargeBN	0.96	1.00	0.14	0.24	1/25	0.01	0.39	5/25	0.04	0.57	**9/25**
Resnet-18	0.90	1.00	0.01	0.53	2/5	-0.05	0.00	0/5	0.05	0.89	**5/5**
VGG11	0.92	0.96	-	-	-	-	-	-	0.03	0.72	**4/5**

(B) Blended attack.

TABLE 5.4: ASR, accuracy and number of successful runs on both attacks, before and after applying the tested methods. Accuracy and ASR averaged over 25 iterations of the architectures of Dhonthi et al., 5 iterations for the other architectures. The columns marked with ↓ represent the reduction of the metric caused by the method. The highest success rate out of the three methods is marked in bold.

5.2.2 SQ2: Execution time

Of the results on the patch attack, described in Section 5.2.1, the execution time of candidate selection and trigger reconstruction is recorded for the method of Dhonthi et al. and the new method. The retraining time of Dhonthi et al. is also shown. The time of pruning by either CLP or the new method is not presented, as this is negligible compared to the other times, and it is clear that the new method as a whole cannot outmatch CLP in speed. The time on the blended attack was not measured. Again, the average is taken across all the models trained for a model architecture. The results are shown in Table 5.5.

Architecture	**Dhonthi et al.**		**New method**
	ABS time	Retraining time	Time
Small	117.07	38.62	18.89
Medium	177.50	60.63	15.89
Large	83.17	259.10	55.82
SmallBN	119.51	39.03	18.89
MediumBN	180.89	-	53.25
LargeBN	266.52	-	222.48
Resnet-18	563.67	333.51	295.41
VGG11	-	-	495.88

Table 5.5: Execution time of candidate selection and trigger reconstruction of the method of Dhonthi et al. and the new method on the patch attack. Average over 25 iterations of the architectures of Dhonthi et al., 5 iterations for the other architectures, on a Nvidia GTX 1050 Ti GPU with 4GB VRAM. Additionally, the retraining time of the method by Dhonthi et al. is also shown. All times are in seconds. For the empty entries in the 'Retraining time' column, no valid triggers were constructed, so there was no retraining taking place.

5.2.3 SQ3: Reconstruction accuracy

The valid reconstructed triggers from the results of Section 5.2.1 for the new method and the method of Dhonthi et al. are compared with the original trigger. An average of the MSE between the reconstructed and true triggers across each reconstructed trigger, architecture and backdoor is shown in Table 5.6.

Architecture	**Dhonthi et al.**	**New method**
Small	0.20	0.68
Medium	0.27	0.40
Large	0.25	0.61
SmallBN	0.15	1.91
MediumBN	-	1.05
LargeBN	-	2.04
Resnet-18	0.16	0.29
VGG11	-	0.56

(A) Patch attack.

Architecture	**Dhonthi et al.**	**New method**
Small	0.88	1.20
Medium	0.80	2.48
Large	0.39	1.96
SmallBN	0.19	2.19
MediumBN	0.21	1.31
LargeBN	0.33	0.77
Resnet-18	0.55	0.83
VGG11	-	0.60

(B) Blended attack.

TABLE 5.6: MSE between real trigger and reconstructed triggers on the two attacks. Average over all triggers reconstructed from 25 iterations of the architectures of Dhonthi et al., 5 iterations for the other architectures. Apart from VGG11 on the method of Dhonthi et al., empty entries mean that no valid trigger was reconstructed.

Chapter 6

Discussion & Conclusion

In this final chapter the results are analyzed, a conclusion to the Research Question is given based on this analysis, and suggestions for future work are presented.

6.1 Discussion

6.1.1 SQ1: Pruning effectiveness

The results in Table 5.4 show that the effectiveness of the method is mixed. Success is almost non-existent on the models of Dhonthi et al., especially those with no batch normalization. At the same time, the method performs clearly better than the other methods on Resnet-18 and VGG11. Even here though, we see some unsuccessful defenses.

Inconsistency of results

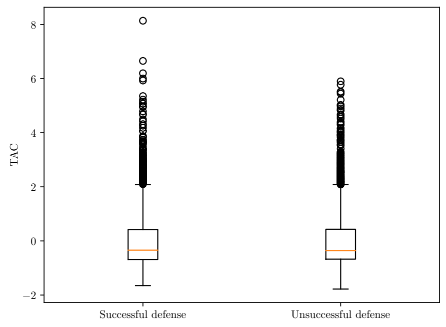
The significant variance between each run is thus rather intriguing, since the only thing changed between them is the model initialization. By comparing the TACs of the real trigger on a model where the defense is successful, against a model where it is unsuccessful, some insight can be gained into the causes of this variance in effectiveness. Figure 6.1 shows boxplots of TACs of an unsuccessful run and a successful run next to each other, on the Resnet-18 and LargeBN architecture respectively, backdoored with the patch attack. From this it can be seen that the TAC distributions in the contrasting models are very similar. For the Resnet-18, the model with a successful defense shows some higher outliers, but the opposite is the case for the LargeBN model. So a different distribution of backdoored channels does not appear to explain the difference in effectiveness.

The difference may be caused by the candidate selection. Comparing this between models on which the defense is successful and those on which it is not in Figure 6.2, a difference can be seen in the channels that are selected as candidates (below the threshold). For both LargeBN and Resnet-18 architectures, the models with an unsuccessful defense have fewer channels below the threshold and thus less evaluated candidates.

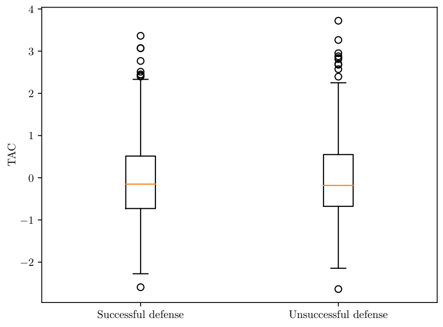
In the case of Resnet-18, the model on which the defense is unsuccessful has no candidates selected at all, easily explaining the lack of results. The model with the successful defense only has one channel below the threshold, but that was enough in this case. This model follows a somewhat negative correlation of variance with TAC, ensuring that backdoored channels can be pruned without significantly affecting the accuracy, as pruning low variance channels should have little effect on benign input. The other Resnet-18 model shows a completely opposite correlation, and thus even if a candidate below the threshold were found, pruning channels associated with its corresponding trigger might significantly reduce the accuracy, as these channels are also important for benign input.

For the LargeBN model with an unsuccessful defense, there are still a few channels below the variance threshold for a candidate, but only one of these has a TAC above the mean, while the model with a successful defense has three candidates with a TAC above the mean. The difference in correlation is much smaller, however.

From these examples, it thus appears that while the distribution of backdoored channels remains relatively consistent with different model initializations, their importance to the overall model (and thus their output variance) is quite sensitive to the model's weight initialization.

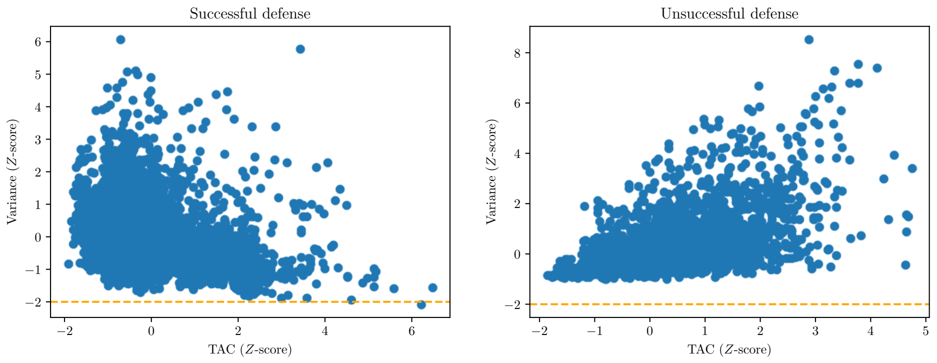


(A) Comparison for Resnet-18 models. After applying the defense, the succesful model had acc. 0.88, ASR 0.01, the unsuccessful model acc. 0.89, ASR 1.00.

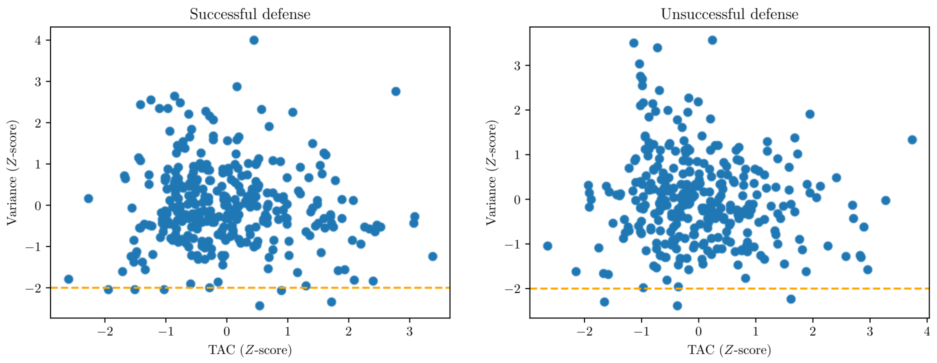


(B) Comparison for LargeBN models. After applying the defense, the succesful model had acc. 0.87, ASR 0.03, the unsuccessful model acc. 0.89, ASR 1.00.

FIGURE 6.1: Boxplots of TAC Z-scores on a model where the defense was successful compared to one where it was unsuccessful. All models were backdoored with the patch attack.



(A) Comparison of Resnet-18 models.



(B) Comparison of LargeBN models.

FIGURE 6.2: Scatter plot of Z-scores of TAC and Z-scores of output variance on a model where the defense was successful compared to one where it was unsuccessful. The dashed line on the y-axis marks the value of -2, the candidate selection threshold. The same models as in Figure 6.1 are used.

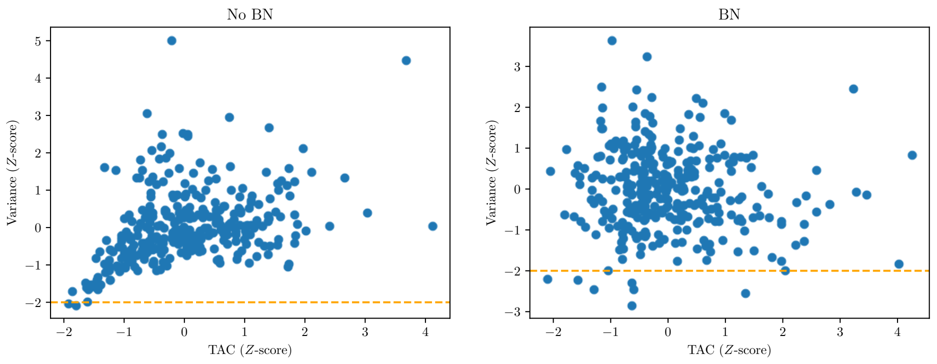


FIGURE 6.3: Scatter plot of Z-scores of TAC and Z-scores of output variance on a Large model and a LargeBN model, trained on the patch attack.

Impact of batch normalization

Another interesting difference is the performance of the architectures with batch normalization compared to those without it. As can be seen in Table 5.4, the new method performs better on all the BN architectures compared to their equivalents with no BN. VGG-11 and Resnet-18 have BN too, and here the results are good too. It can also be noted that this effect is generally reversed for the method of Dhonthi et al., which performs worse on the BN architectures compared to their equivalents without it.

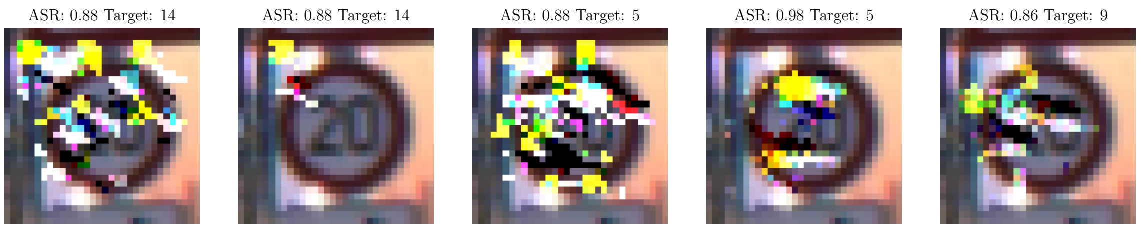
BN generally causes faster convergence and better performance overall [11] as also evidenced by the higher accuracy of the BN architectures compared to their equivalents with no BN. This better training could also result in the backdoored channels being more distinct from benign ones, so that they are redundant for benign input and have minimal impact on the accuracy. Considering Figure 6.3, comparing models of the Large and LargeBN architectures in terms of TAC and output variance, this indeed appears the case. While the LargeBN model has plenty of channels that with high TAC and high variance, there are a few high TAC channels that have low variance and are thus selected as candidates. The Large (no BN) model, on the other hand, has a clear positive correlation with high TAC and variance, meaning the more influence a channel has on the backdoor, the more discriminating it is on benign data as well. This makes it rare for the candidate selection to select any channels here that have minimal impact on the accuracy while being important to the backdoor.

Meanwhile, with ABS (as part of Dhonthi et al.), BN seems to have a rather opposite effect, since the candidate selection of ABS uses modified benign data to find the most sensitive neurons. Thus, the channels that have high output variance would be the ones easily found by ABS, and then this correlation would be beneficial in finding the optimal channels for reconstruction.

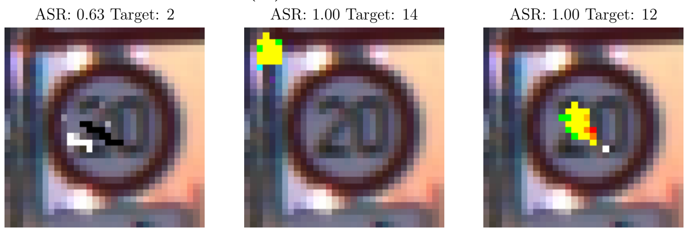
Results of CLP

Comparing the results on the new method with those on the other methods, again in Table 5.4 it can first be noted that CLP is not successful at all. This is not so surprising for the Dhonthi et al. architectures. Appendix A.1 shows that success of CLP is rare on these models, and if it is successful, it is with a much lower threshold than the threshold $u_{\text{CLP}} = 3$ used here.

However, the lack of effectiveness on Resnet-18 and VGG-11 is surprising, as the results of Liu et al. [18] show that CLP performs very well on these architectures. However, this



(A) New method



(B) ABS (part of Dhonthi et al.)

FIGURE 6.4: Reconstructed triggers of patch attack on the same model of the Large architecture for both methods. See Figure 5.1b for the image with the real trigger.

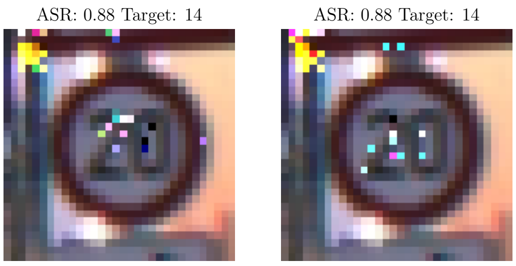
was tested on the CIFAR-10 dataset. Perhaps a slightly lower threshold would have worked for more successful pruning, but this would mean the method would need a different choice of hyperparameter for different datasets, which would be a further limitation.

6.1.2 SQ2: Execution time

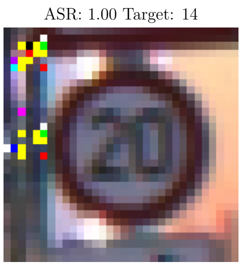
Table 5.5 shows an improvement in execution time for all architectures, not even counting the time for retraining. On the architectures where the method is unsuccessful, this is not necessarily beneficial, as the fast execution time could be due to the method only reconstructing invalid intermediate triggers belonging to poorly selected candidates. However, the method is also noticably faster on the Resnet-18 architecture, where it performs well, showing a near halving of execution time. Hence, since trigger reconstruction is the most time-consuming, splitting this into an intermediate and an original trigger seems to be beneficial for speed. The new method could be made even faster by eliminating the ASR computation on every epoch, altering the number of epochs, or adjusting the threshold deciding the validity of an intermediate trigger, however this would adversely affect the quality of the trigger, or prevent any reconstruction, and thus hinder the performance.

6.1.3 SQ3: Reconstruction accuracy

Table 5.6 shows that the new method's triggers clearly has higher MSE than those of ABS (as part of the method of Dhonti et al.) even on the architectures where the defense is successful. However, it is to be noted that ABS for some architectures barely reconstructs any triggers, with most of the unsuccessful runs not having any valid triggers reconstructed at all. ABS is thus a lot more conservative, mostly only returning accurate triggers as valid. On the other hand, the new method reconstructs multiple triggers most of the time, which can often be wrong, but are sometimes good enough for a successful pruning. Furthermore, on the larger architectures Resnet-18 and VGG-11, where the method performs the best, the MSEs are fairly low on the new method and relatively close to the MSEs of the triggers of ABS.

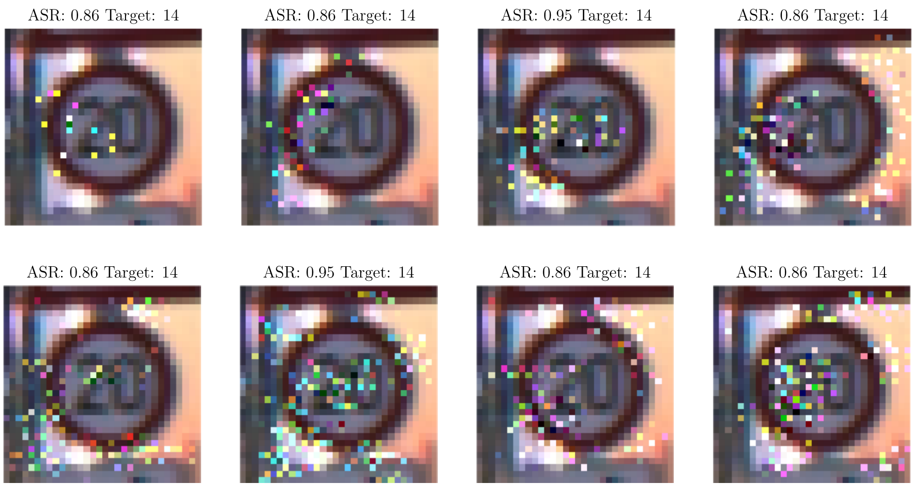


(A) New method

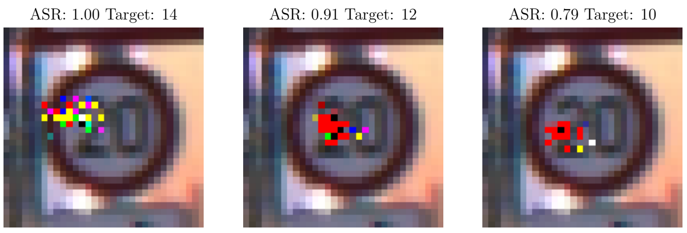


(B) ABS (part of Dhonthi et al.)

Figure 6.5: Reconstructed triggers of patch attack on models of the Resnet-18 architecture for both methods. The triggers of the new method are not from the same model as those of ABS, as there were no Resnet-18 models on which both methods had valid reconstructed triggers. See Figure 5.1b for the image with the real trigger.



(A) New method



(B) ABS (part of Dhonthi et al.)

Figure 6.6: Reconstructed triggers of the blended attack on the same model of the Large architecture for both methods. See Figure 5.1c for the image with the real trigger.

Patch attack

Examining reconstructed triggers of the target label (14) on the Large architecture in Figure 6.4, the new method has quite a few pixels in different colors outside the original patch, especially with the first reconstructed trigger. To be noted also are the three triggers reconstructed with the wrong target label, which is quite common with the new method. With ABS, the patch area for the trigger with the correct target label is almost perfect. There are two wrong triggers here too, but these are much smaller.

Where the new model performs well, such as on Resnet-18, the new method can reconstruct triggers that are as accurate, if not more, as those produced by ABS, see Figure 6.5.

ABS' much smaller maximum mask size (32 pixels) in general seems to help in having it achieve a much lower MSE. The size limit of the trigger on the new method is much bigger (25% of the image, 256 pixels) to be robust to more attacks, especially those spanning the entire image, like the blended attack.

Otherwise, judging by the results of Tao et al. [29], the original trigger reconstruction should not be worse than that of ABS, but the fact that the new method reconstructs it from an intermediate trigger might also contribute to the worse performance. The intermediate trigger can sometimes be reconstructed quite poorly, affecting the original trigger as well.

Furthermore, the objective of the trigger reconstruction does nothing to preserve the benign outputs of other channels in the candidate's layer. ABS meanwhile, explicitly minimizes the difference with other channels, which could cause the trigger to be more 'focused' and have no other interference from the outputs of other channels. However, this does make the optimization more complex, and also does not take into account the possibility of multiple channels in the same layer working together to produce the trigger.

Blended attack

On the blended attack, both ABS and the new method fail to provide accurate triggers, though the triggers found often have the right target label and ASR. See Figure 6.6. Since the blended attack consists of random pixels throughout the image, which get transformed and merged throughout the convolutional layers, the model presumably learns a wide range of possible triggers that all result in similar intermediate output. The success of the reconstructed triggers for both methods, that bear little resemblance to the original trigger, indicates there are many equivalent triggers generated by this attack, even though the model was not trained on these.

However, presumably due to the larger maximum trigger size, the triggers of the new method do incorporate the transparency and scattered nature of the true trigger more, resembling the nature of the trigger at a higher level better even though they are worse in terms of MSE.

6.2 Conclusion

6.2.1 SQ1: Pruning effectiveness

The method shows mixed results in terms of effectiveness, with many of the small architectures not showing any better results than the method of Dhonthi et al. However, on the larger architectures, which more resemble those used in real scenarios, the new method performs better than the method of Dhonthi et al. Despite this, we still find unsuccessful cases on these architectures, so the method is still quite lacking in reliability. The method is also consistently better than CLP, at least with the given threshold u_{CLP}.

6.2.2 SQ2: Execution time

The goal of achieving a faster method seems to have been achieved, with the new method being faster than the method of Dhonthi et al. on all architectures, even when the time of retraining is not included. However, this is not a reduction by orders of magnitude, and on very large models the method could still take a significant amount of time, depending as well on the distribution of output variance in channels, affecting the number of candidates selected.

6.2.3 SQ3: Reconstruction accuracy

The reconstruction accuracy often leaves a lot to be desired. Depending on the attacks deemed threatening by the defender, the maximum trigger size could be made stricter to eliminate some of the largest triggers, however this also makes the method more versatile, and this seems to always be a trade-off when considering improvements to the trigger reconstruction. A method like that of the Topological Prior [10] improves triggers by constraining their shape, but this would then not work well for the scattered random trigger of the blended attack. Compared to ABS, the scattered, transparent nature of the blended trigger does appear much better reconstructed with the new method, and this would presumably carry over to other attacks of this nature as well.

6.2.4 Conclusion on Research Question

Thus, **SQ1** (mitigation effectiveness) and **SQ2** (speed) can be answered as showing an improvement to the compared methods. **SQ3** (reconstruction accuracy) is a little more difficult to see as an improvement, but it has advantages over ABS in terms of the shape of the blended trigger. The answer to the Research Question (Section 1.1) then is, that the new method is more effective than the evaluated existing methods on more relevant larger architectures, faster than the comparable method of Dhonthi et al., and with comparable reconstructed trigger quality on these large architectures, at least on the GTSRB dataset. However, the method can still be quite unreliable even on these large architectures, and more datasets and methods ought to be tested to come to a more definitive conclusion.

Overall, the main limitation of the method is the many points of failure. At candidate selection, intermediate trigger reconstruction, and original trigger reconstruction, the method may not provide the correct solution, and this could have a crucial impact on detecting any backdoor. Presumably, in larger models, this is less of a problem since there are more backdoored channels to consider. The modular nature of the method has benefits too. When considering improvements, one component (for instance, the candidate selection) could be improved, possibly leading to better results, without altering the entire method.

6.2.5 Other conclusions

The results additionally suggest that backdoor behaviour is not very consistent even with the same model, data and backdoor, with backdoor behaviour sometimes not appearing in any channels that are redundant for benign data. An adversary could thus exploit this by deliberating initializing weights in some manner to ensure there are no low variance channels sensitive to the backdoor, and this would eliminate the effectiveness of the defense.

Furthermore, it was found batch normalization has a particularly large influence on the way backdoors are inserted, and thus on the performance of defense methods as well.

6.3 Future work

One direction for future improvement is in changing the candidate selection. More research into how backdoor behaviour manifests itself into a network might reveal a universal pattern that could make the method perform more consistently. Possibly, for this a more narrow approach is needed rather than considering entire output channels. A starting point could be to research the causes explaining the differences in correlation between output variance and TAC for models of the same architecture and backdoor, as shown in Figure 6.2.

Furthermore, the trigger reconstruction might be extended to provide better support for feature-space attacks such as WaNet [21] and dynamic attacks like IAB [20], as the method can currently only reconstruct triggers in the form of pixel perturbations to be applied to all inputs. Reconstruction could also be performed per label, providing a defense against label-to-label attacks, at the cost of increased time complexity.

As a more concrete and immediate step for future work, experiments with different thresholds for candidate selection and pruning might yield more better results, as these thresholds were not formally tested.

Finally, an evaluation on truly clean models could be done to verify the success of detection, with a lack of reconstructed triggers implying the model has no backdoors.

6.4 Acknowledgements

I would like to thank Moritz Hahn and Akshay Dhonthi for supervising this thesis, Mannes Poel for his feedback on the final draft and his work in grading, and finally Vahid Hashemi and the other people at the weekly AUDI meetings for their feedback.

Bibliography

[1] Christopher M. Bishop. “Chapter 5: Neural Networks”. In: *Pattern Recognition and Machine Learning*. New York, NY, USA: Springer-Verlag, pp. 225–290. ISBN: 978-0-387-31073-2.

[2] Nicholas Carlini and David Wagner. “Towards Evaluating the Robustness of Neural Networks”. In: *2017 IEEE Symposium on Security and Privacy (SP)*. 2017 IEEE Symposium on Security and Privacy (SP). ISSN: 2375-1207. May 2017, pp. 39–57. DOI: 10.1109/SP.2017.49.

[3] Xinyun Chen et al. *Targeted Backdoor Attacks on Deep Learning Systems Using Data Poisoning*. Dec. 14, 2017. DOI: 10.48550/arXiv.1712.05526. arXiv: 1712.05526[cs]. URL: http://arxiv.org/abs/1712.05526 (visited on 07/22/2023).

[4] Akshay Dhonthi, Ernst Moritz Hahn, and Vahid Hashemi. *Backdoor Mitigation in Deep Neural Networks via Strategic Retraining*. version: 1. Dec. 14, 2022. arXiv: 2212.07278[cs]. URL: http://arxiv.org/abs/2212.07278 (visited on 02/10/2023).

[5] Chong Fu et al. “FREEEAGLE: Detecting Complex Neural Trojans in Data-Free Cases”. In: *Proceedings of the 32nd USENIX Security Symposium*. 32nd USENIX Security Symposium. Aug. 2023, pp. 6399–6416. ISBN: 978-1-939133-37-3. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/fu-chong.

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. “Chapter 9: Convolutional Neural Networks”. In: *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016, pp. 326–366.

[7] Henry Gouk et al. “Regularisation of neural networks by enforcing Lipschitz continuity”. In: *Machine Learning* 110.2 (Feb. 1, 2021), pp. 393–416. ISSN: 1573-0565. DOI: 10.1007/s10994-020-05929-w. URL: https://doi.org/10.1007/s10994-020-05929-w (visited on 03/16/2023).

[8] Tianyu Gu et al. “BadNets: Evaluating Backdooring Attacks on Deep Neural Networks”. In: *IEEE Access* 7 (2019). Conference Name: IEEE Access, pp. 47230–47244. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2909068.

[9] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016, pp. 770–778. URL: https://openaccess.thecvf.com/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html (visited on 07/22/2023).

[10] Xiaoling Hu et al. “Trigger Hunting with a Topological Prior for Trojan Detection”. In: International Conference on Learning Representations. Jan. 28, 2022. URL: https://openreview.net/forum?id=TXsjU8BaibT (visited on 03/06/2023).

[11] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *Proceedings of the 32nd International Conference on Machine Learning*. International Conference on Machine Learning. ISSN: 1938-7228. PMLR, June 1, 2015, pp. 448–456. URL: https://proceedings.mlr.press/v37/ioffe15.html (visited on 03/15/2023).

[12] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. Jan. 29, 2017. DOI: 10.48550/arXiv.1412.6980. arXiv: 1412.6980[cs]. URL: http://arxiv.org/abs/1412.6980 (visited on 07/17/2023).

[13] Yehao Kong and Jiliang Zhang. *Adversarial Audio: A New Information Hiding Method and Backdoor for DNN-based Speech Recognition Models*. Apr. 8, 2019. DOI: 10.48550/arXiv.1904.03829. arXiv: 1904.03829[cs,eess]. URL: http://arxiv.org/abs/1904.03829 (visited on 08/26/2023).

[14] Yiming Li et al. “Backdoor Learning: A Survey”. In: *IEEE Transactions on Neural Networks and Learning Systems* (2022), pp. 1–18. ISSN: 2162-237X, 2162-2388. DOI: 10.1109/TNNLS.2022.3182979. URL: https://ieeexplore.ieee.org/document/9802938/ (visited on 02/10/2023).

[15] Junyu Lin et al. “Composite Backdoor Attack for Deep Neural Network by Mixing Existing Benign Features”. In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’20: 2020 ACM SIGSAC Conference on Computer and Communications Security. Virtual Event USA: ACM, Oct. 30, 2020, pp. 113–131. ISBN: 978-1-4503-7089-9. DOI: 10.1145/3372297.3423362. URL: https://dl.acm.org/doi/10.1145/3372297.3423362 (visited on 03/07/2023).

[16] Shuying Liu and Weihong Deng. “Very deep convolutional neural network based image classification using small training sample size”. In: *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*. 2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR). ISSN: 2327-0985. Nov. 2015, pp. 730–734. DOI: 10.1109/ACPR.2015.7486599.

[17] Yingqi Liu et al. “ABS: Scanning Neural Networks for Back-doors by Artificial Brain Stimulation”. In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’19. New York, NY, USA: Association for Computing Machinery, Nov. 6, 2019, pp. 1265–1282. ISBN: 978-1-4503-6747-9. DOI: 10.1145/3319535.3363216. URL: https://doi.org/10.1145/3319535.3363216 (visited on 03/07/2023).

[18] Yingqi Liu et al. “Complex Backdoor Detection by Symmetric Feature Differencing”. In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). ISSN: 2575-7075. June 2022, pp. 14983–14993. DOI: 10.1109/CVPR52688.2022.01458.

[19] Yunfei Liu et al. “Reflection Backdoor: A Natural Backdoor Attack on Deep Neural Networks”. In: *Computer Vision – ECCV 2020*. Ed. by Andrea Vedaldi et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 182–199. ISBN: 978-3-030-58607-2. DOI: 10.1007/978-3-030-58607-2_11.

[20] Tuan Anh Nguyen and Anh Tran. “Input-Aware Dynamic Backdoor Attack”. In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., 2020, pp. 3454–3464. URL: https://proceedings.neurips.cc/paper/2020/hash/234e691320c0ad5b45ee3c96d0d7b8f8-Abstract.html (visited on 03/07/2023).

[21] Tuan Anh Nguyen and Anh Tuan Tran. “WaNet - Imperceptible Warping-based Backdoor Attack”. In: International Conference on Learning Representations. Feb. 10, 2022. URL: https://openreview.net/forum?id=eEn8KTtJOx (visited on 03/07/2023).

[22] Adam Polyak and Lior Wolf. “Channel-level acceleration of deep face representations”. In: *IEEE Access* 3 (2015), pp. 2163–2175. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2015.2494536. URL: http://ieeexplore.ieee.org/document/7303876/ (visited on 06/12/2023).

[23] Janosh Riebesell. *Convolution Operator*. Apr. 9, 2022. URL: https://tikz.net/conv2d/ (visited on 07/17/2023).

[24] Aniruddha Saha, Akshayvarun Subramanya, and Hamed Pirsiavash. “Hidden Trigger Backdoor Attacks”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.7 (Apr. 3, 2020). Number: 07, pp. 11957–11965. ISSN: 2374-3468. DOI: 10.1609/aaai.v34i07.6871. URL: https://ojs.aaai.org/index.php/AAAI/article/view/6871 (visited on 03/07/2023).

[25] Ahmed Salem et al. “Dynamic Backdoor Attacks Against Machine Learning Models”. In: *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*. 2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P). June 2022, pp. 703–718. DOI: 10.1109/EuroSP53844.2022.00049.

[26] Shibani Santurkar et al. “How Does Batch Normalization Help Optimization?” In: *Advances in Neural Information Processing Systems*. Vol. 31. Curran Associates, Inc., 2018. URL: https://papers.nips.cc/paper_files/paper/2018/hash/905056c1ac1dad141560467e0a99e1cf-Abstract.html (visited on 07/17/2023).

[27] Ali Shafahi et al. “Universal Adversarial Training”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.4 (Apr. 3, 2020). Number: 04, pp. 5636–5643. ISSN: 2374-3468. DOI: 10.1609/aaai.v34i04.6017. URL: https://ojs.aaai.org/index.php/AAAI/article/view/6017 (visited on 06/20/2023).

[28] J. Stallkamp et al. “Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition”. In: *Neural Networks*. Selected Papers from IJCNN 2011 32 (Aug. 1, 2012), pp. 323–332. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2012.02.016. URL: https://www.sciencedirect.com/science/article/pii/S0893608012000457 (visited on 07/22/2023).

[29] Guanhong Tao et al. “Better Trigger Inversion Optimization in Backdoor Scanning”. In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). New Orleans, LA, USA: IEEE, June 2022, pp. 13358–13368. ISBN: 978-1-66546-946-3. DOI: 10.1109/CVPR52688.2022.01301. URL: https://ieeexplore.ieee.org/document/9879000/ (visited on 06/19/2023).

[30] Bolun Wang et al. “Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks”. In: *2019 IEEE Symposium on Security and Privacy (SP)*. 2019 IEEE Symposium on Security and Privacy (SP). ISSN: 2375-1207. May 2019, pp. 707–723. DOI: 10.1109/SP.2019.00031.

[31] Zhou Wang et al. “Image quality assessment: from error visibility to structural similarity”. In: *IEEE Transactions on Image Processing* 13.4 (Apr. 2004). Conference Name: IEEE Transactions on Image Processing, pp. 600–612. ISSN: 1941-0042. DOI: 10.1109/TIP.2003.819861.

[32] Baoyuan Wu et al. “BackdoorBench: A Comprehensive Benchmark of Backdoor Learning”. In: *Advances in Neural Information Processing Systems* 35 (Dec. 6, 2022), pp. 10546–10559. URL: https://proceedings.neurips.cc/paper_files/paper/2022/hash/4491ea1c91aa2b22c373e5f1dfce234f-Abstract-Datasets_and_Benchmarks.html (visited on 08/22/2023).

[33] Dongxian Wu and Yisen Wang. “Adversarial Neuron Pruning Purifies Backdoored Deep Models”. In: *Advances in Neural Information Processing Systems*. Vol. 34. Curran Associates, Inc., 2021, pp. 16913–16925. URL: https://proceedings.neurips.cc/paper/2021/hash/8cbe9ce23f42628c98f80fa0fac8b19a-Abstract.html (visited on 03/06/2023).

[34] Jing Xu, Minhui (Jason) Xue, and Stjepan Picek. “Explainability-based Backdoor Attacks Against Graph Neural Networks”. In: *Proceedings of the 3rd ACM Workshop on Wireless Security and Machine Learning*. WiseML ’21. New York, NY, USA: Association for Computing Machinery, June 28, 2021, pp. 31–36. ISBN: 978-1-4503-8561-9. DOI: 10.1145/3468218.3469046. URL: https://dl.acm.org/doi/10.1145/3468218.3469046 (visited on 08/26/2023).

[35] Xinyang Zhang et al. “Trojaning Language Models for Fun and Profit”. In: *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*. 2021 IEEE European Symposium on Security and Privacy (EuroS&P). Sept. 2021, pp. 179–197. DOI: 10.1109/EuroSP51992.2021.00022.

[36] Runkai Zheng et al. “Data-Free Backdoor Removal Based on Channel Lipschitzness”. In: *Computer Vision – ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part V*. Berlin, Heidelberg: Springer-Verlag, Oct. 23, 2022, pp. 175–191. ISBN: 978-3-031-20064-9. DOI: 10.1007/978-3-031-20065-6_11. URL: https://doi.org/10.1007/978-3-031-20065-6_11 (visited on 03/07/2023).

Appendix A

Additional experiments

This appendix consists of additional experiments that were done to provide justification for design choices.

A.1 CLP results

See Section 3.1.3 for the background on this experiment.

The architecture used for this experiment corresponds to the small architecture as described in Dhonthi et al. [4] Additionally, a variant of it with batch normalization layers after each convolutional layer, referred to as SmallBN in Chapter 5, is also used.

All the models were trained on the GTSRB dataset, backdoored with a 4x4 pixel yellow patch trigger, placed at coordinates (2,2), starting from the upper-left corner of the image, with 14 (stop sign) as the target label. This backdoor was applied to 10% of the training data, The models are trained for 15 epochs.

For every iteration, thresholds that give either a reduction in benign accuracy less than 0.25 or an ASR reduction more than 0.75 are considered as successful results. The thresholds evaluated are values from 0 up to and including 3, in discrete steps of 0.25 $(0.0, 0.25, \ldots 2.5, 2.75, 3.0)$. Out of these, the best threshold is determined by the formula $\frac{1}{2}(1 - \text{ASR reduction}) + \frac{1}{2}\text{Acc. reduction}$. This threshold, and its associated ASR and accuracy, are reported in Table A.1. If there are no successful results, the results for value for $u = 3$ are reported.

Accuracy			ASR			
Before	After CLP	Reduction	Before	After CLP	Reduction	u
0.89	0.89	0.00	0.89	0.89	0.00	3.00
0.90	0.89	0.01	0.99	0.99	0.00	3.00
0.89	0.55	0.34	1.00	0.02	0.98	1.25
0.90	0.90	0.00	0.99	0.99	0.00	3.00
0.88	0.88	0.00	1.00	1.00	0.00	3.00
0.88	0.88	0.00	0.99	0.99	0.00	3.00
0.89	0.89	0.00	1.00	1.00	0.00	3.00
0.89	0.89	0.00	0.99	0.86	0.13	2.75
0.90	0.90	0.00	1.00	1.00	0.00	3.00
0.91	0.91	0.00	1.00	1.00	0.00	3.00

(a) Results for Small.

Accuracy			ASR			
Before	After CLP	Reduction	Before	After CLP	Reduction	u
0.92	0.79	**0.13**	1.00	0.10	**0.90**	1.75
0.88	0.55	0.33	1.00	0.35	0.65	1.25
0.92	0.89	**0.03**	1.00	0.03	**0.97**	2.00
0.92	0.92	0.00	1.00	1.00	0.00	3.00
0.89	0.81	**0.08**	1.00	0.05	**0.95**	2.00
0.93	0.89	**0.04**	1.00	0.02	**0.98**	1.75
0.92	0.92	0.00	1.00	1.00	0.00	3.00
0.92	0.92	0.00	1.00	1.00	0.00	3.00
0.90	0.78	**0.12**	1.00	0.02	**0.98**	2.50
0.90	0.71	0.19	1.00	0.84	0.16	2.00

(b) Results for SmallBN.

Table A.1: Results for CLP for 10 different iterations of the two architectures. In bold are reduction values that correspond to a successful defense.

A.2 TAC pruning

See Section 4.3 for the background on this experiment.

The architecture used for this experiment corresponds to the medium architecture as described in Dhonthi et al. [4], with the addition of batch normalization layers after each convolutional layer, referred to as MediumBN in Chapter 5.

All the models were trained on the GTSRB dataset, backdoored with a 4x4 pixel yellow patch trigger, placed at coordinates (2,2), starting from the upper-left corner of the image, with 14 (stop sign) as the target label. This backdoor was applied to 10% of the training data, The models are trained for 15 epochs.

The TAC for each channel is computed, then the weights for all channels with a layer-wise Z-score above 2 are pruned. The results are shown in Table A.2.

Accuracy			**ASR**		
Before	After TAC pruning	Reduction	Before	After TAC pruning	Reduction
0.94	0.93	0.02	1.00	0.52	0.48
0.94	0.91	**0.03**	1.00	0.02	**0.98**
0.93	0.92	0.02	1.00	0.56	0.43
0.94	0.92	**0.03**	1.00	0.02	**0.98**
0.95	0.92	**0.03**	1.00	0.07	**0.93**
0.94	0.92	**0.02**	1.00	0.05	**0.95**
0.95	0.92	**0.02**	1.00	0.07	**0.93**
0.95	0.90	**0.05**	1.00	0.02	**0.98**
0.95	0.92	**0.02**	1.00	0.09	**0.90**
0.94	0.91	**0.03**	1.00	0.03	**0.97**

TABLE A.2: Results for pruning by TAC for 10 different iterations of the MediumBN architecture. In bold are reduction values that correspond to a succesful defense.