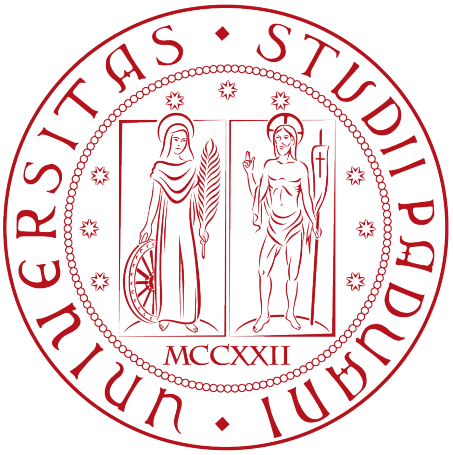
University of Padua

Department of Mathematichs "Tullio Levi-Civita"

Master Degree in Cybersecurity



Backdoor Attacks and Defences on Neural Networks

Supervisor

Prof. Mauro Conti
University of Padua

Co-supervisor

Stefanos Koffas
TU Delft

Student

Massimiliano Belluomini

Student ID

2020385

Accademic Year 2021-2022

Abstract

In recent years, we have seen an explosion of activity in deep learning in both academia and industry. Deep Neural Networks (DNNs) significantly outperform previous machine learning techniques in various domains, e.g., image recognition, speech processing, and translation. However, the safety of DNNs has now been recognized as a realistic security concern.

The basic concept of a backdoor attack is to hide a secret functionality in a system, in our case, a DNN. The system behaves as expected for most inputs, but malicious inputs activate the backdoor.

Deep learning models can be trained and provided by third parties or outsourced to the cloud. The reason behind this practice is that the computational power required to train reliable models is not always available to engineers or small companies. Apart from outsourcing the training phase, another strategy used is transfer learning. In this case, an existing model is fine-tuned for a new task. These scenarios allow adversaries to manipulate model training to create backdoors.

The thesis investigates different aspects of the broad scenario of backdoor attacks in DNNs. We analyze the neuron activations in backdoor models and designed a possible defence based on empirical observations. The neurons of the last layer of a DNN show high variance in their activations when the input samples contain the trigger.

We also present a new type of trigger that can be used in audio signals obtained using the echo. Smaller echoes (less than 1 ms) are not even audible to humans, but they can still be used as a trigger for command recognition systems. We show that with this trigger, we can bypass STRIP-ViTA, a popular defence mechanism against backdoors.

Finally, we analyze and evaluate the blind backdoor attacks, which are backdoor attacks that are based on both code and data poisoning, and tested them with an untested defence. We also propose a way to bypass the defence.

Preface

I am proud to present my thesis "Backdoor Attacks and Defences on Neural Networks". Thanks to this research work, I was able to delve into the interesting world of security applied to neural networks. First of all, I would like to thank my co-supervisor and mentor Stefanos, who followed me carefully, motivated me and helped me find my way through the various experiments.

Of fundamental importance was Prof. Mauro Conti who allowed me to undertake this project, which I partly carried out at TU Delft. I want to express him my huge gratitude.

I have to mention also my colleague Filippo Giambartolomei with whom I did the project of the course “Computer and Network Security: advanced topics”. During the project we started our exploration of backdoor attacks in neural networks. Our work was a source of inspiration for the thesis.

Finally, I would like to thank my parents, Alessandro and Valeria, my sister Arianna, and my girlfriend Sara, for always supporting and encouraging me to follow my aspirations. I also have to thank them for helping me overcome moments of stress and discouragement.

Padova, December 2022 Massimiliano Belluomini

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the last years, we have seen a burst of activity in deep learning in both academia and industry. Deep Neural Networks (DNNs) are becoming widely adopted in a variety of domains, e.g., image recognition, speech processing, translation, malware analysis. However, the safety of DNNs has now been recognized as a realistic security concern.

In 2017, Gu et al. [10] proposed for the first time the concept of backdoor attack in the field of neural networks. The basic concept of a backdoor attack is to hide a secret functionality in a system, in our case a DNN. The system behaves as expected for most inputs, but a malicious input, i.e., a sample containing the trigger, activates the backdoor. Fig. 1.1 shows an example of the attack. The authors created two backdoored neural networks using data poisoning: they poisoned hand-written digit and street sign classifiers to misclassify inputs possessing a trigger that activates the backdoor. In a data poisoning attack, the attacker poisons part of the training datataset with a trigger pattern and changes the class of the poisoned data to the desired one, i.e., the target class. After that, the model is trained with the poisoned dataset.

In the same year, Chen et al. [3] introduced a backdoor in a face recognition system. In this case the trigger is represented by a physical object, i.e., a pair of specific glasses.

Deep learning models can be trained and deployed by third parties or outsourced to the cloud. In fact, engineers and small companies do not always have the computational power needed to train reliable models. In addition to outsourcing the training phase, transfer learning is also used. Existing models are adapted to new tasks. This practice is called fine-tuning and it is often used for image recognition. These scenarios give attackers the opportunity to manipulate training data to create backdoors. Gu et al. [10] demonstrated the feasibility of backdoor attacks in both scenarios.

Differently from adversarial examples, introduced in 2013 by Szegedy et al. [30], in a backdoor attack the adversary introduces a secret functionality that is activated by a specific trigger. Instead, we can think of adversarial examples as bugs [10] in benign models. Indeed, adversarial examples are perturbations to inputs that cause them to be missclassfied. Generally, adversarial examples are used at test time to perform an evasion attack, while backdoor attacks target the training phase of a model. Another work [22] also proposed an algorithm to find universal perturbations that could cause missclassification across different neural networks.

Data poisoning is not the only way to insert backdoors to neural networks, indeed,

Figure 1.1: Example of a backdoored street sign classifier. The stop sign is maliciously misclassified as a speed-limit sign by the malicious model [10].

there exist also non-poisoning-based attacks. There are some researches about post deployment attacks, such as [23] and [5]. The former can insert a backdoor into a DNN through the bit-flip attack, using an algorithm that efficiently generates a trigger designed to locate certain vulnerable bits of the model weights stored in memory (i.e., DRAM). The latter attack assumes the attacker can run code on the victim system, in order to modify model parameters in memory to achieve predefined malicious behavior on a certain set of inputs.

Bagdasaryan et al. [1] have proposed a different type of attacks based on code poisoning. They have modified the loss value computation to insert the backdoor generating poisoned inputs on the fly. The authors assume an attacker who knows the task (general data domain and possible model architectures), but not the specific training data and hyperparameters.

Most of the works in this field regard image and video classifiers, and only a smaller part has as a target Natural Language Process (NLP) and sound recognition applications.

The thesis in organized as follows: Chapter 2 briefly illustrates the technical background needed for the projects. In particular, it covers machine and deep learning fundamentals and the concept of backdoor attacks in this field. Chapter 3 covers the related works, both attacks and defences previously proposed and analyzed in the literature. Chapter 4 illustrates how the values of the neuron activations change between clean and backdoored models and suggest a possible direction of research to identify trojaned models based on the observation of neuron activations. Chapter 5 presents a new possible trigger that can be used in automated speech recognition systems. Chapter 6 investigates blind backdoors attacks [1]. Firstly, we reproduce the attack and test it with a previously untested defence, i.e. STRIP [8], then we propose a way to bypass it.

Chapter 2

Background

This chapter introduces the background knowledge required to follow the remaining part of the thesis. It starts with a short introduction to machine and deep learning, then it continues with the explanation of backdoor attacks in these types of models.

2.1 Machine Learning

Machine learning is the automated detection and extraction of meaningful patterns in data. The particularity of machine learning tools is the ability to learn these patterns from the experience. The main idea behind machine learning is that given a collection of samples, i.e., the training dataset, we want to be able to make predictions about new samples that we have not seen before.

Machine learning is a very wide domain and there exist different types of learning. The main categories are:

- Supervised learning: each sample in the training dataset is labeled. An example of this application is to label new samples, e.g., our goal could be to label emails in spam or not-spam;
- Unsupervised learning: samples in the dataset are unlabeled and the machine learning algorithm try to discover meaningful relations between the samples. A typical example of this task is clustering, which aggregates samples in clusters;
- Reinforcement learning: in this case the dataset is not fixed. In reinforcement learning there is an agent that interacts with an environment and takes actions that can be rewarded with positive, negative or neutral rewards. Therefore, there is a feedback loop between the learning system and its experiences. Generally, the goal is to maximize a function of the rewards.

The thesis targets supervised learning applications.

The main task considered in the thesis is classification. Our aim is to develop a model able to approximate an unknown function $f\colon \mathbb{R}^n \to \{1, ..., k\}$ that given an input $\bar{x}$ is able to assign it to category or to give the probability that $\bar{x}$ belongs to

the different categories. In order to learn, the model needs a training dataset D_{train} containing tuples $(\bar{x}_i, z_i)$ where $\bar{x}_i$ is the input and z_i is the corresponding label.

The accuracy of the trained model is evaluated using a set of fresh data not used during training, i.e., D_{test}.

2.2 Deep Learning

Deep learning is a sub-field of machine learning. It dates back to the 1940s, however, it was relatively unpopular for several years preceding the current popularity. Recent progress in the underlying hardware and the availability of massive quantitative of training data has led to a revolution. Nowadays, deep learning is widely used in a variety of applications such as computer vision, speech recognition and translation.

Deep learning models are inspired by the biological brain, however, they are not generally designed to replicate biological functions.

The building units of a deep learning model are the neurons. A deep learning model is made by a set of layers composed by several neurons. We have an input layer, some hidden layers and then the output layer. The number of hidden layers determine the depth of the model, while the larger layer determines the width of the model. A neuron computes a non-linear function over the inputs. Its output depends on the input and a set of weights, that are parameters which the model learns. As shown

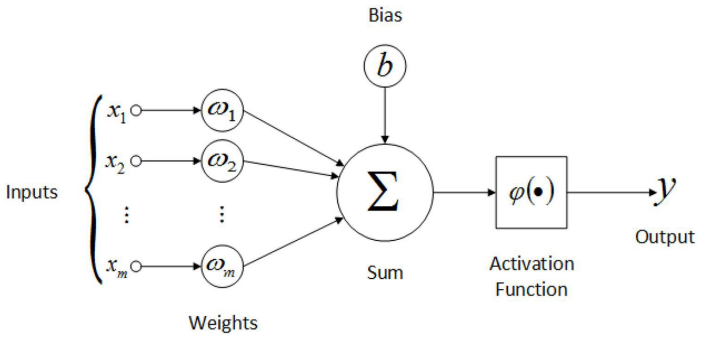


Figure 2.1: Artificial neuron.

in Fig. 2.1, the inputs x_i are firstly multiplied by their weights w_i and then summed together including in the computation also the bias b. Next, the result is passed to the activation function φ. During the training phase, the model learns the weight and the biases. The activation function φ is usually non-linear to approximate complex functions. Indeed, if all activation functions were linear, the model would behave like a single layer network.

Formally, a deep neural network is a parameterized function $F_\Theta \colon \mathbb{R}^n \to \mathbb{R}^m$ that maps an input $\bar{x} \in \mathbb{R}^n$ to an output $\bar{z} \in \mathbb{R}^m$. In case of a classification task, $\bar{z}$ is a vector of probabilities over the k classes. $\bar{x}$ is labeled as belonging to the class with the highest probability, i.e., $arg\ max\ F_\Theta(\bar{x})$ is the class label. Θ represents the parameters of the neural network.

2.2.1 Activation functions

The most common activation functions used in deep learning models are:

- sigmoid: this function takes any real value as input and outputs values in the range of 0 to 1
$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{2.1}$$
- tanh: differently from sigmoid its output is in the range of -1 to 1
$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.2}$$
- ReLU: the Rectified Linear Unit is computed as follows:
$$ReLU(x) = max(0, x) \tag{2.3}$$
- ELU: the Exponential Linear Unit (ELU) is defined as follows:
$$ELU(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^{x} - 1) & \text{if } x \leq 0 \end{cases} \quad \text{with } 0 < \alpha \tag{2.4}$$
- softmax: it transforms the previous layer's activations into a vector of probabilities. The function rescales the elements of a vector to lie in the range $[0, 1]$ and sum to 1. The different probabilities are computed as follows:
$$softmax_i(\overline{x}) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{2.5}$$
- log softmax: it applies the natural logarithm to the softmax. It can be computed as follows:
$$log(softmax_i(\overline{x})) = log(\frac{e^{x_i}}{\sum_j e^{x_j}}) \tag{2.6}$$

2.2.2 Training

The goal of the training is to determine the parameters of the network, i.e., the weights and the biases, with the use of the training dataset. Therefore, we want to find the parameters that minimize the loss function and allow the model to achieve good accuracy in predicting new data. In case of supervised learning, the training dataset D_{train} is a set of tuples composed by an input vector $\bar{x_i}$ and its corresponding correct label z_i. The learning algorithm iterates over D_{train} and for each tuple it computes the loss value $l = L(\theta(\bar{x}, z)$. Then, the algorithm updates the network parameters computing the gradients $g = \nabla l$ and using the backpropagation. The training involves two phases:

- the forward propagation: the input $\bar{x}_i$ is propagated through the neural network in order to produce the output $\hat{z}$;

- the back-propagation: the gradients of the loss functions are used to update weights and biases. This phase uses the gradient descent algorithm that updates the parameters in the opposite direction of the loss function's gradient. The weights are updated as follows:

$$w_{ij}^{(t)} = w_{ij}^{(t)} - \eta \frac{\partial L}{\partial w_{ij}^{(t)}} \tag{2.7}$$

  where η is the learning rate that determines the step size. It is a hyperparameter that is not learned by the algorithm. $w_{ij}^{(t)}$ refers to the weight that connects neuron $v_i^{(t-1)}$ to neuron $v_j^{(t)}$, where t is the index of the layer and i and j refer to the neuron index.

  The majority of deep learning models use the stochastic gradient descent (SGD), that calculates the gradient and updates the parameters of the model using only a part of the training set, i.e., the minibatch, that is randomly drawn from D_{train}.

Different variations of adaptive learning algorithms have been developed to adapt the learning rate during the training process. Three of the most widely used are:

- AdaGrad: the names stands for adaptive gradient algorithm. It scales learning rates of single parameters inversely proportional to the square root of the sum of all historical squared values of the gradient [9].
- RMSProp: the name stands for Root Mean Square Propagation. It is similar to AdaGrad, but it discards the history of ancient past. RMSProp uses an exponentially decaying average to discard history from ancient past [9].
- Adam: the name is the short for Adaptive Moment Estimation. It is an improvement of RMSProp that incorporates momentum as an estimate of the first order moment. Momentum is designed to accelerate the learning by accumulating an exponential moving average of past gradients, basically the gradient step depends on how aligned the sequence of last gradients are [9].

The convergence of the learning algorithm depends also on the initial values of the parameters. In general, weights and biases are randomly initialized accordingly to a uniform or a Gaussian distribution.

There exist also several optimization and regularization techniques used to speed up the training or to avoid overfitting. With the term overfitting we refer to a model that performs well with training data, e.g., it has low loss and high accuracy, but it does not perform well on unseen data. Overfitting occurs when the gap between training error and test error is large. In the experiments considered in the thesis, the following techniques are used:

- batch normalization [14]: it is a method of adaptive reparametrization, motivated by the difficulty of training very deep models [9]. Each unit's pre-activation is normalized by subtracting the mean and dividing by the standard deviation calculated across the layer's activation. The mean and the standard deviation used are computed for each mini-batch. At test time, the global values of them are used.

- dropout [29]: it is an efficient way to perform bagging. Bagging involves training multiple neural networks and evaluating them on each test sample [9]. During training, some neurons outputs are randomly dropped out. Basically, dropout trains an ensemble consisting of subnetworks constructed by removing neurons from an underlying neural network. We use a minibatch-based learning algorithm. Each time we load an example into a minibatch, we randomly sample a different binary mask to apply to all the input and hidden units in the network. The binary value for each unit is sampled independently from all the others. The probability of sampling a mask value of one (causing a unit to be included) is a hyper parameter selected before training begins or after hyperparameter tuning [9].

- early stopping: during the training phase, it is common to observe that training error decreases steadily over time, but validation set error begins to rise again or stops to decrease. Thus, it is possible to obtain a model with better validation test error by returning the parameter setting at the point in time with the lowest validation set error [9]. A simple way to implement this technique is to stop the training process when a monitored metric, e.g., the accuracy or the loss, has stopped improving. An important parameter is indeed patience. It defines the number of epochs with no improvement, after which the training stops.

2.2.3 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) [16] are a specialized type of networks used for processing grid-like structures, such as images, videos and time-series data. They are based on convolutional layers, i.e., layer that perform a mathematical operation called convolution. In a CNN there are three main types of layers: convolutional layers, pooling layers and fully-connected layers.

Convolutional layers are used for feature extraction. This type of layers perform a convolutional operation between the input, e.g., an image, and a filter, generally called kernel, that is small matrix of learnable parameters. The output of the operation is usually referred as feature map. They perform a dot product between the kernel and the input, in general a tensor. The kernel slides across the height and the width of the input image. The sliding size is called stride. An element-wise product between each element of the kernel and the input tensor is calculated at each location of the tensor and summed to obtain the output value in the corresponding position of the output tensor, called feature map [24]. The procedure is repeated for different kernels, indeed, every convolutional layer has more than one kernel. In image recognition tasks, the different feature maps represent different characteristics of the input image. Figure 2.2 shows an example of convolution operation with a kernel size of 3x3, no padding, and a stride of 1.

Pooling layers are used to reduce the spatial size of the representation. This type of layer replaces the output of the network at certain locations by deriving a summary statistic of the nearby outputs, e.g., considering the max value (max-pooling). For example, we can use a 2×2 pooling with stride 2 per direction to downsample a 4×4 activation map to a size of 2×2. Figure 2.3 shows an example of max-pooling operation with a filter size of 2x2, no padding, and a stride of 2. In a fully connected layer, neurons have full connectivity with all neurons in the preceding and succeeding layers.

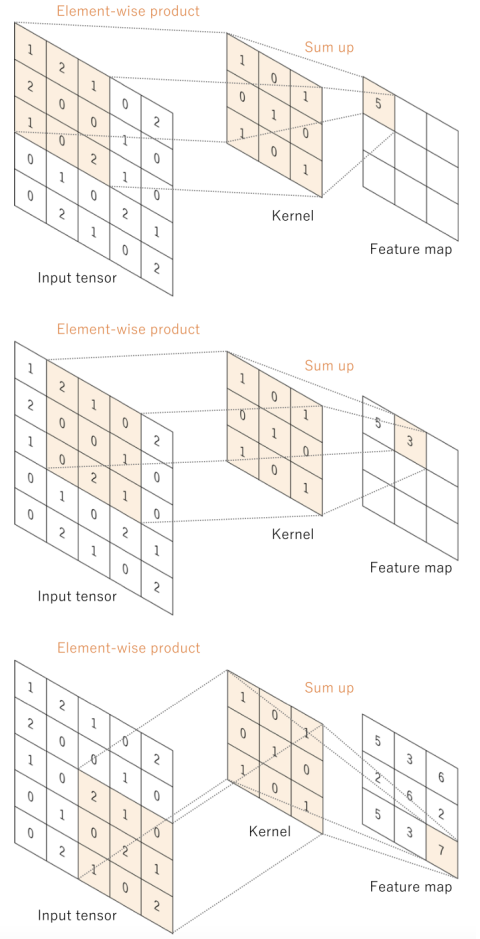


Figure 2.2: Convolution.

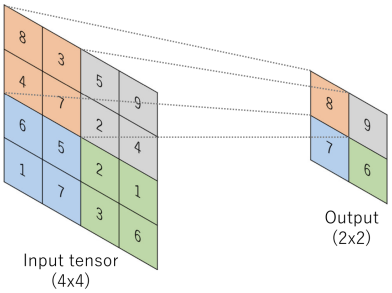


Figure 2.3: Max-pooling.

2.2.4 Audio recognition

Neural networks are the bedrock also of sound recognition systems. In general, to build and train an automatic speech recognition (ASR) model for recognizing words, the audio files are firstly pre-processed. In particular, it is convenient to transform

the waveforms represented in time domain (e.g., audio files in WAV format) in the frequency domain. A possible solution [28] is to use short-time Fourier transform (STFT) that convert the waveforms to as spectrograms, which show frequency changes over time. Spectrograms can be represented as 2D images. Differently from the Fourier tranform, STFT splits the signal into windows of time and runs a Fourier transform on each window, this preserves some time information, that would otherwise be lost.

The spectrograms are the inputs of the model that we want to build. The same procedure must be followed during the inference phase.

2.3 Backdoor attacks in deep learning models

The first defintion of backdoor attacks in deep learning models was given by Gu et al. [10] in 2017. According to the authors, a backdoored model (or a BadNet) is a neural network that performs well on most inputs, but behaves badly on specific attacker-chose inputs. The attacker trains a malicious model Θ^{adv} that is different from an honestly trained model Θ^*. The goals of the attacker are two: to build a model that has an accuracy similar to a non malicious model and to insert a secret functionality in the backdoored model. The former goal is accomplished if $\mathcal{A}(F_{\Theta^{adv}}, D_{valid}) \geq a^*$, where a^* represents the minimum target accuracy desired and D_{valid} indicates the validation set. The latter is achieved if inputs containing the backdoor trigger are missclassified by the model. Formally, let consider a function $\mathcal{P}\colon \mathbb{R}^n \to \{0, 1\}$ that maps any input $\bar{x}$ to a binary output, such that the output is equal to 1 if the input contains the trigger, 0 otherwise. Then, the attacker goal is that $\forall \bar{x}\colon \mathcal{P}(\bar{x}) = 1,\ arg\ max\ F_{\Theta^{adv}} \neq arg\ max\ F_{\Theta^*}$.

The above definition includes both target and non-targeted attacks [10]. In the former case, the attacker specifies the desired output for triggered inputs. In the latter case, the adversary's aim is to reduce the classification accuracy for inputs containing the trigger. Fig. 2.4 shows the procedure of a poisoning-based backdoor attack.

The attack accuracy is the most common metric used to evaluate the performance of a backdoor attack. It is computed as follows:

$$Attack\ accuracy = \frac{\#predictions\ of\ target\ class}{\#of\ total\ predictions} \tag{2.8}$$

Another interesting metric is the missclassification rate. It is useful in case of class specific attacks, i.e., attack in which the backdoor is activated only when the trigger is attached to an input belonging to a specific class. In this case, the requirements to activate the backdoor are two: the trigger and the class. An input with the trigger that belongs to a different class from the source class should not activate the backdoor. The metric is computed using the following formula:

$$Misclassification\ rate = \frac{\#wrong\ predictions}{\#of\ total\ predictions} \tag{2.9}$$

where the number of wrong predictions considers the trojaned samples which are classified as the target class even if they do not belong to either the source class or the target class.

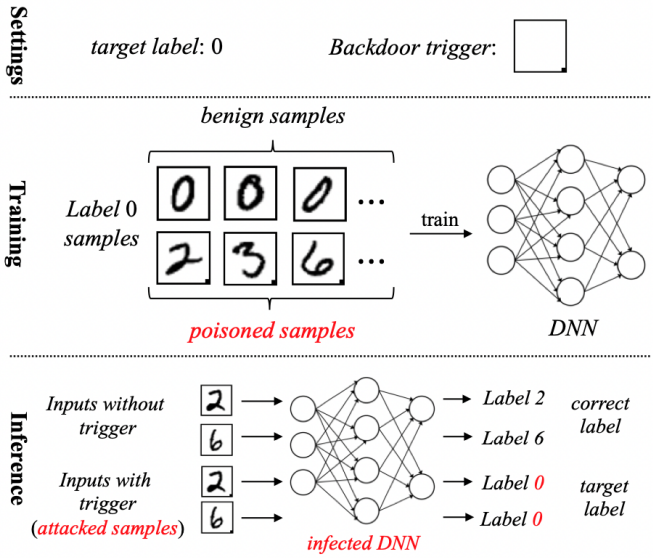


Figure 2.4: Procedure of a poisoning-based backdoor attack. In this example, the trigger is a black square on the bottom right corner and the target label is 0. Part of the training dataset is modified to have images with the trigger stamped, and their label is changed to the target label. Accordingly, the trained model is infected: it will recognize poisoned images (i.e., images containing the trigger) as the target label while still correctly predicting the label for clean images [18].

Alternatively, it is possible to evaluate the accuracy on poisoned samples that do not belong to the source class. It can be evaluated as follows:

$$Accuracy\ with\ trojaned\ samples = \frac{\#correct\ predictions}{\#of\ total\ predictions} \tag{2.10}$$

where the number of correct predictions considers trojaned samples belonging to non-source classes which are correctly classified and the number of total predictions only consider trojaned samples that do not belong to the source class.

It is also important to evaluate the clean accuracy drop that measures the difference in accuracy introduced from the backdoor on clean data. Indeed, the backdoor should be stealthy and should not raise suspicions. Moreover, a model with lower performance will not be used by the victims.

Chapter 3

Literature review

Deep neural networks require large amount of training data and significant computational power. The computational power required for training reliable models is not always available to engineers or companies, therefore, the task of training is often outsourced to third parties. The task of training can be outsourced to the cloud or to a third entity. Outsourcing the training is often referred as MLaaS, i.e., Machine Learning as a Service. Apart from outsourcing the training phase, another strategy used, is transfer learning: in this case an existing model is fine-tuned for a new task. It is commonly used for image recognition. These scenarios give the possibility to malicious people to manipulate the training process to introduce backdoors.

3.1 Attacks

The first and most known attacks are based on data poisoning. In this scenario, the attacker must have access to a part of training dataset in order to poison it. The percentage of samples that has to be poisoned can vary depending on the task and the model. For example, [3] shows that even with few dozen of samples it is possible to obtain an attack success rate higher than 90%. They used a face recognition system, trained with 600,000 samples. The authors also showed that a data poisoning attack can create a physically implementable backdoor: using data poisoning they were able to create a malicious face recognition system, in which a specific pair of glasses is the trigger that activates the backdoor.

The most famous work regarding backdoor attacks in neural networks is [10]. The authors introduced for the first time this concept and they created two backdoored neural networks. Indeed, they poisoned handwritten digit and street sign classifiers to misclassify inputs stamped with the trigger. They also evaluated the effect of transfer learning, analyzing if the backdoor was still present after the re-training. They have successfully demonstrated that the backdoors can persist also in case of transfer learning.

There exist different ways to insert the trigger in the samples. In image classification, it is possible to use as a trigger a pixel pattern [10], to overlap clean samples with a trigger with certain transparency [3], or to use physical objects present in the image

[3] and [6]. In the latter article, i.e., [6], the authors present a backdoored model in which the trigger in made by raindrops. According to their evaluation, this attack is more disguised and can circumvent data filtering. Indeed, the raindrops do not follow a fixed pattern that is a common feature in backdoor attacks done via data poisoning.

Among the poisoned-based attacks there is also [1] that is based on code poisoning: the attack code creates poisoned training data inputs during the training process, and uses multi-objective optimization to achieve high accuracy and high attack success rate. The attack uses Multiple Gradient Descent Algorithm (MGDA) that is used for multi-task learning.

There are different ways to poison the dataset with the trigger:

- Class agnostic trigger: the backdoor is always activated when the trigger is present. The goal is to obtain a trained model that classifies correctly clean inputs, but miss-classifies every sample in input that contains the trigger. Poisoned samples belong to all classes and their respective labels are changed to the target one.
- Class specific trigger: the backdoor is activated only when the trigger is present on a sample that belongs to a specific class. The aim is to perform an attack that would be successful only when the trigger is appended on the attacker chosen class (or classes). Poisoned samples belong to all classes, however only the the labels of samples belonging to the source class are changed.
- Multiple triggers to same label: different triggers are used, but all of them have the same target class. Different triggers mean different trigger placement or the use of different patterns as triggers or the combination of both.
- Multiple triggers to multiple labels: different triggers are used and every trigger has a different target class. Different triggers mean different trigger placement or the use of different patterns as triggers or the combination of both.
- Clean label attack: the backdoor is always activated when the trigger is present. However, during the poisoning phase, only the samples belonging to one class, i.e., the target class, are poisoned and no label is changed. At the inference phase, the trigger should activate the backdoor, causing the samples to be missclassfied to the target class.

However, there exist also non-poisoning-based attacks. These type of attacks assume that the attacker can modify the model parameters, for example using malicious software, physical or hardware attacks. Moreover, differently from poisoning attacks, the adversary can insert the backdoor at each stage of the pipeline, as shown in Fig. 3.1. There are two main strategies to modify the model parameters: the first one is to change some weights, while the second is to add an extra part to the model. The former strategy is used in [12] where the authors show that is possible to manipulate the parameters of a pre-trained model to inject backdoors. Moreover, the attack does not require training, knowledge of the dataset or access to it. Also [23] adopts the same strategy: the authors propose a Target Bit Trojan (TBT) method, which can insert a backdoor using bit-flip attack. They demonstrate that flipping only several vulnerable bits identified by their method, using a bit-flip technique, such as row-hammer, can effectively insert the the backdoor. The second strategy is used in [31] where the researchers insert a tiny trojan module (TrojanNet) into the original model. They

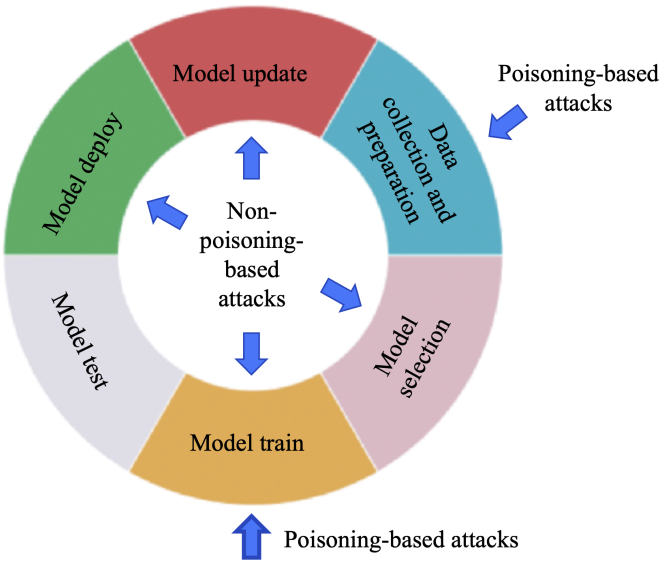


Figure 3.1: Backdoor attacks in each stage of ML pipeline [36].

claim that the technique is model agnostic and does not require training, indeed it is enough to train the tiny module just once.

3.2 Defenses

To mitigate the backdoor threats, several defenses have been proposed. Most of the works focuses on poisoning-based attacks. In the survey [19], the authors classify the available empirical defenses in seven categories:

- Preprocessing based defenses: in [21] Liu et al. propose a preprocessing defence, i.e., they suggest to insert a preprocessor before the neural network to prevent illegitimate inputs from activating the backdoor. In the paper, they use an autoencoder with bottleneck structure with the same number of input and output neurons. Another approach that belongs to this category is Grad-CAM [26] that can be used to identify influential regions in images. These regions must be removed and replaced by neutral box. The authors illustrate also a GAN-based method to reconstruct the masked regions.

- Model reconstruction based defenses: the aim of these defenses are to remove the backdoor from the poisoned model. The idea is based on the forgetting process of DNNs. An example of this category is represented by Fine-Pruning [20]. The defence is based on the combination of the pruning and the fine-tuning techniques. The former method works as follows: the defender feeds the suspicious DNN with clean inputs and records the average activation of each neuron. The defender then iteratively prunes neurons in increasing order of average activations and records the accuracy of the modified model in each iteration. The process terminates when the accuracy on clean samples drops below a threshold. The latter technique

is a strategy commonly used transfer learning scenarios. Fine-tuning uses the pre-trained weights of a model as starting point of the training process. The combinations of both methods can be used to remove backdoors.

- Trigger synthesis based defenses: these defenses aim to reconstruct the trigger in the first stage and then remove the backdoor, in a similar way of model reconstruction based defenses. Neural Cleanse [33] belongs to this category and it is one of the most famous defences. The authors present a system to reconstruct triggers and suggest different methods to mitigate backdoors: input filtering, neuron pruning and unlearning. The latter technique can be performed using the reversed trigger to train the infected model to classify correctly the samples even if they contain the trigger.

- Model diagnosis based defenses: these defenses classify a model as infected based on the decision of a pre-trained meta classifier. An example of this technique is developed by the authors of [35]. The strategy is to use a meta-classifier to distinguish between trojaned and bening models. The classifier is trained using shadow models, both benign and malicious.

- Poison suppression based defenses: these techniques are applied during the training process. The idea is to mitigate the effectiveness of backdoor attacks modifying the SGD. For example, adding randomness in the training process affects negatively the effect of poisoned samples. In [17], Tang et al. illustrate an anti-backdoor learning scheme, that is based on the consideration that backdoor attacks have faster learning and target-class dependency. The proposed learning scheme try to isolate backdoor samples, and break the correlation between backdoor samples and the target class.

- Training sample filtering based defenses: these defenses involve the filtering of the training dataset before the training process. The goal is to eliminate poisoned samples. Accordingly to the Li et al. [19], also the well known activation clustering defense [2] falls in this category. The defense is based on the clustering of the activations of training samples of each class in two clusters. After the clustering phase it is possible to determine which, if any, of the clusters corresponds to poisonous data.

- Testing sample filtering based defenses: these defenses aim to determine if malicious samples are used at the inference time. Gao et al. [8] proposed a method for detecting backdoor attacks based on entropy, i.e., STRIP (STRong Intentional Perturbation). It is a run-time detection system. STRIP bases its detection on the entropy measurement of perturbed inputs: each input is perturbed by superimposing to it a clean sample. The process repeat this perturbation using different clean samples, while keeping fixed the input that we want to analyze. Then, we use these perturbed samples as inputs for the model and we observe the entropy of these classifications.A clean input always exhibits high entropy, instead a trojaned inputs always exhibits low entropy. Indeed, in a malicious model, if the original sample contains the trigger, the predictions of the perturbed inputs will almost always fall in the target class. The method was originally designed to detect backdoor in image classification task, but it was extended to work with other domains, i.e. text and audio [7].

Chapter 4

Neuron activations analysis

This chapter illustrates the analysis of the neuron activations in backdoored models. In particular, it highlights the difference between the neuron activations between clean and trojaned models.

4.1 Idea

We analyze the activations of the last hidden layer of backdoored CNN and ResNet9 models trained with CIFAR-10 and Fashion-MNIST datasets. As highlighted in previous works, such as [10], backdoored models generally present some neurons dedicated to the backdoor that are fired when the trigger is present in the input.

We deepen on the neurons activations and in particular we study the variability of the average activations. We compute the mean (i.e., μ) and the standard deviation (i.e., σ) of the average activations when the inputs are clean and when they are poisoned. We find that trojaned inputs violate the Chebyshev bound [27] that guarantees that no more than a certain fraction of values can be more than a certain distance from the mean. In particular, no more than $1/k^2$ of the distribution's values can be more than k standard deviations away from the mean. We use the Chebyshev bound, because, empirically, we find that neuron activations do not always have a normal distribution as instead considered in other works, such as [12].

4.2 Implementation

We conduct several experiment using two model architectures, i.e., a CNN and ResNet9, and two dataset, i.e., Fashion-MNIST and CIFAR-10. Regarding the Chebyshev bound, we use $k = 4$, hence no more than 6.25% neurons activations should be beyond four standard deviation from the mean. Empirically, we observe that $k = 4$ works well in our experiments, however, we plan to investigate this parameter more thoroughly in the future.

All the following tests are run on Google Colab using Keras and Tensorflow.

Model architectures

The first model considered is a 6 layered CNN followed by flatten layer. The output layer is a dense layer of 10 nodes, one for each class, with *softmax* activation. Instead, the convolutional layers use *elu* as activation function. For the training of our model we set batch size to 64 and epochs to 125. The learning rate varies during the training, indeed it is initially set to 0.001, then it is decreased to 0.0005 after 75 epochs and to 0.0003 after 100 epochs. As optimizer, we use RMSProp (Root Mean Square Propagation).

ResNet is an architecture proposed in 2015 by Microsoft [11]. It uses the so called residual blocks: in order to solve the problem of the vanishing/exploding gradient, this model uses a technique called skip connections. The skip connection connects activations of a layer to further layers by skipping some layers in between. This forms a residual block. A ResNet model is constructed by stacking these residual blocks together. The name ResNet followed by one or more digit number simply implies the model has a certain number of neural network layers. We use ResNet9. As in the previous model, the output layer is a dense layer of 10 nodes, one for each class, with *softmax* activation. The convolutional layers use $ReLU$ as activation function. The optimizer is Adam with learning rate equals to 0.003 and decay value equals to 0.00005. For the training of the model we set batch size to 256 and epochs to 75.

Datasets

CIFAR-10 dataset was collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The dataset consists of 60000 32x32 colour images, i.e., RGB images, in 10 mutually exclusive classes, with 6000 images per class. There are 50000 training images and 10000 test images. We further split the training set in training set and validation set: the former contains 40000 images and the latter 10000. The 10 classes are: *airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck*. We use a black and white trigger of dimension 3x3 pixels placed in the bottom right corner. Fig. 4.2 shows an example of a poisoned image. Generally, it is possible to use different types of triggers. Most of the works uses pattern or square triggers. The pattern is not very important in general. Thus we chose a simple one. The trigger position is usually chosen in arbitrary way. Figure 4.1 shows the classes of CIFAR-10 dataset, as well as ten random images from each of them.

Fashion-MNIST (F-MNIST) is a dataset of Zalando's article images—consisting of a training set of 60000 examples and a test set of 10000 examples. We further split the training set in training set and validation set: the former contains 50000 images and the latter 10000. Each example is a 28x28 grayscale image, associated with a label from 10 classes: *t-shirt/top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, ankle boot*. We use a grey and white trigger of dimension 2x2 pixels placed in the upper left corner. Fig. 4.4 shows an example of a poisoned image. Figure 4.3 shows the classes of Fashion-MNIST dataset, as well as ten random images from each of them.

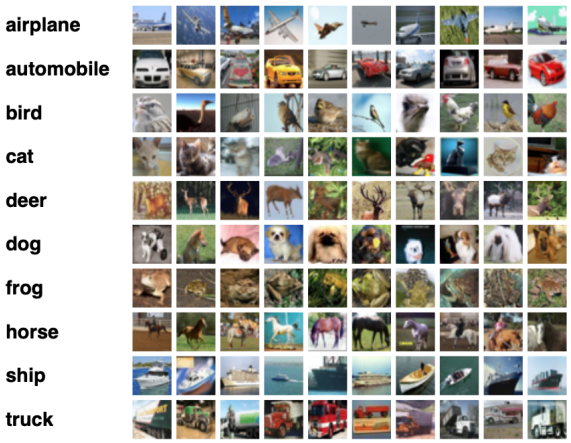


Figure 4.1: The classes in the CIFAR-10 dataset, as well as 10 random images from each.

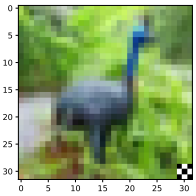


Figure 4.2: A poisoned sample of the CIFAR-10 dataset.

Figure 4.3: The classes in the Fashion-MNIST dataset, as well as 10 random images from each.

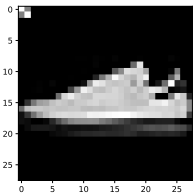


Figure 4.4: A poisoned sample of the FMNIST dataset.

4.3 Evaluation

We consider three different attacks:

- **Class agnostic trigger**: it is the original attack and the easiest to deploy.
- **Class specific trigger**: it uses the same threat model but it is more difficult to defend. Indeed, some defences, e.g., STRIP [8], fail to defend against.
- **Clean label attack**: it is a more realistic threat model as poisoned samples cannot be easily identified by manual inspection as their label is not changed. However, the attack is more difficult to implement. Accordingly to [15], in dirty-label attacks, the poisoned samples are generally very different from the samples. As a result, the distance between these samples in the feature space is large and easy to learn by our models. In clean-label attacks, the poisoned samples belong to the target class, and there is a higher probability that their features are not very different from the clean samples.

4.3.1 Class agnostic

We choose as target the class number 7, i.e., the horse and the sneaker classes respectively for CIFAR-10 and F-MNIST datasets. The corresponding label of the poisoned samples is changed from the original one, to 7. We expect no difference even if we chose a different class. The backdoored models classify correctly clean inputs, but missclassifies every sample in input that contains the trigger.

In all the following tests the average neuron activations are computed considering 100 input samples. The activations refer to the last hidden layer of the models.

Tables 4.1 and 4.2 show the results across the different models and datasets. It is interesting to notice that, in all the cases, the standard deviation of the neuron activations is at least one order of magnitude bigger when the input samples are tojaned. We plot the average activations of the neurons when the samples in input are with trigger (in blue) and when the they are without it (in orange) 4.5, 4.6, 4.7, 4.8. From the figures, we can notice that only a portion of the neurons seems to be involved in the backdoor, indeed, only a subset of them shows higher values when the samples are poisoned. The Appendix A (Table A.1 and Fig. A.1) contains another test done with CIFAR-10 dataset, but with a smaller amount of poisoned samples in the training dataset that generates a model with an attack success rate of 74.08%.

CIFAR-10 dataset - CNN model 60 infected samples		
Accuracy: 87.86% Attack accuracy: 98.49%		
	Clean	Trigger
Mean	0.0030	0.0856
Standard deviation	0.0205	0.3485
% of avg. activations inside $\mu \pm 4\sigma$	99.5605%	87.4512%
% of avg. activations outside $\mu \pm 4\sigma$	0.4395%	12.5488%

Fashion-MNIST dataset - CNN model 100 infected samples		
Accuracy: 90.75% Attack accuracy: 99.35%		
	Clean	Trigger
Mean	0.0053	0.0998
Standard deviation	0.0502	0.3150
% of avg. activations inside $\mu \pm 4\sigma$	99.3055%	86.4583%
% of avg. activations outside $\mu \pm 4\sigma$	0.6945%	13.5417%

Table 4.1: Activation of the neurons of the last hidden layer in the CNN models trained with CIFAR-10 and FMNIST datasets. The attack is class agnostic.

CIFAR-10 dataset - ResNet9 model 60 infected samples		
Accuracy: 87.74% Attack accuracy: 96.73%		
	Clean	Trigger
Mean	1.7063	2.3264
Standard deviation	0.3720	2.2743
% of avg. activations inside $\mu \pm 4\sigma$	100.0%	89.0381%
% of avg. activations outside $\mu \pm 4\sigma$	0.0%	10.9619%

Fashion-MNIST dataset - ResNet9 model 100 infected samples		
Accuracy: 94.07% Attack accuracy: 99.76%		
	Clean	Trigger
Mean	2.1440	3.3920
Standard deviation	0.5757037	3.3622398
% of avg. activations inside $\mu \pm 4\sigma$	100.0%	85.1128%
% of avg. activations outside $\mu \pm 4\sigma$	0.0%	14.8872%

Table 4.2: Activation of the neurons of the last hidden layer in the ResNet9 models trained with CIFAR-10 and FMNIST datasets. The attack is class agnostic.

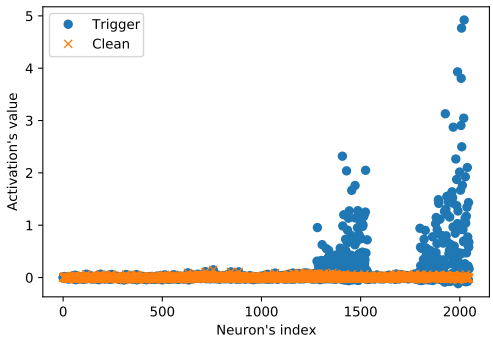


Figure 4.5: Activation of the neurons of the last hidden layer in the CNN model trained with CIFAR-10 dataset and 60 infected samples. The attack is class agnostic.

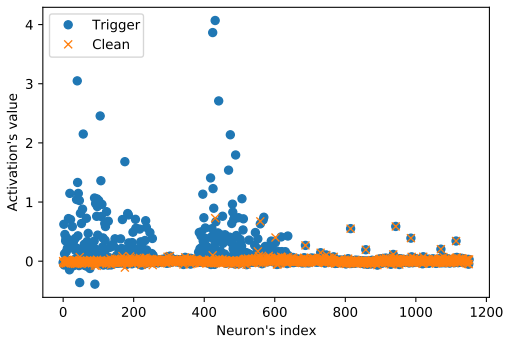


Figure 4.6: Activation of the neurons of the last hidden layer in the CNN model trained with FMNIST dataset and 100 infected samples. The attack is class agnostic.

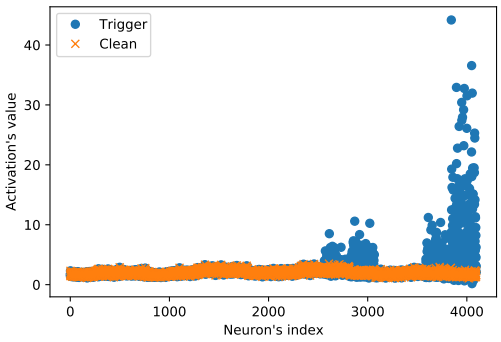


Figure 4.7: Activation of the neurons of the last hidden layer in the ResNet9 model trained with CIFAR-10 dataset and 60 infected samples. The attack is class agnostic.

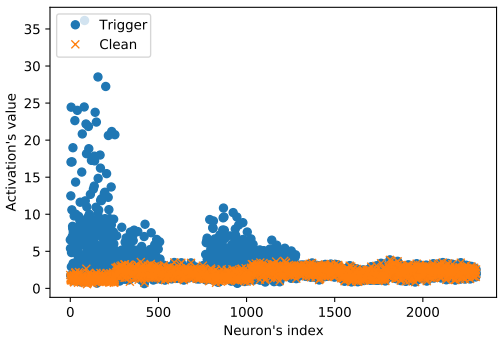


Figure 4.8: Activation of the neurons of the last hidden layer in the ResNet9 model trained with FMNIST dataset and 100 infected samples. The attack is class agnostic.

4.3.2 Class specific

In the experiments, the source class is 1, i.e., automobile (CIFAR-10) and trouser (F-MNIST), and the target class is the same as before, i.e., 7. Differently from the previous case the attack would be successful only when the trigger is appended on the source class. When an input with the trigger belongs to the source class, it is missclassified, while when it belongs to other classes, it is correctly classified.

As before, the average neuron activations are computed considering 100 input samples. The activations refer to the last hidden layer of the model.

Tables 4.3 and 4.4 show the results across the different models and datasets. Also in this case the standard deviations of the neuron activations of trojaned samples, both from source and non-source classes, are at least three times bigger compared to the same values evaluated with clean inputs. It is interesting to notice that we have a larger percentage of activations outside the boundaries also for non-source images. From the figures 4.9, 4.10, 4.11, 4.12 it is possible to notice that also in this kind of attacks only a portion of the neurons seems to be involved in the backdoor. If we look at the tables, we can notice differences between source and non-source classes. The percentage of average activations outside the boundaries is larger in the presence of trojaned source inputs. This fact follows the intuition, actually, only them can activate the backdoor. The Appendix A contains other similar tests done with CIFAR-10 dataset, but with different numbers of poisoned samples in the training datasets (Tables A.2, A.3 and Fig. A.2, A.3).

CIFAR-10 dataset - CNN model **60 infected samples from the source class, 400 from non source classes**			
Accuracy: 86.69% Attack accuracy: 97.30%			
	Clean	Source	Non-source
Mean	0.0010	0.0603	0.0569
Standard deviation	0.0216	0.2706	0.2724
% of avg. activations inside $\mu \pm 4\sigma$	99.6094%	84.1308%	90.8203%
% of avg. activations outside $\mu \pm 4\sigma$	0.3906%	15.8692%	9.1797%

Fashion-MNIST dataset - CNN model **100 infected samples from the source class, 300 from non source classes**			
Accuracy: 92.04% Attack accuracy: 98.20%			
	Clean	Source	Non-source
Mean	-0.0038	0.0128	0.0119
Standard deviation	0.0314	0.1084	0.0669
% of avg. activations inside $\mu \pm 4\sigma$	99.0451%	93.4028%	95.4861%
% of avg. activations outside $\mu \pm 4\sigma$	0.9549%	6.5972%	4.5139%

Table 4.3: Tables containing the statistics of the average neuron activations of the CNN models trained with CIFAR-10 and FMNIST datasets. The attack is class agnostic.

CIFAR-10 - ResNet9 **60 infected samples from the source class, 300 from non source classes**			
Accuracy: 87.88% Attack accuracy: 87.1%			
	Clean	Source	Non-source
Mean	1.9920	2.4149	2.2295
Standard deviation	0.3661	1.3758	1.2360
% of avg. activations inside $\mu \pm 4\sigma$	100.0%	88.7695%	94.5313%
% of avg. activations outside $\mu \pm 4\sigma$	0.0%	11.2305%	5.4687%

Fashion-MNIST - ResNet9 **100 infected samples from the source class, 300 from non source classes**			
Accuracy: 93.96% Attack accuracy: 97.80%			
	Clean	Source	Non-source
Mean	2.1632	2.3923	2.5253
Standard deviation	0.5700	1.5387	1.0052
% of avg. activations inside $\mu \pm 4\sigma$	100.0%	91.6667%	97.5260%
% of avg. activations outside $\mu \pm 4\sigma$	0.0%	8.3333%	2.4740

Table 4.4: Tables containing the statistics of the average neuron activations of the ResNet9 models trained with CIFAR-10 and FMNIST datasets. The attack is class agnostic.

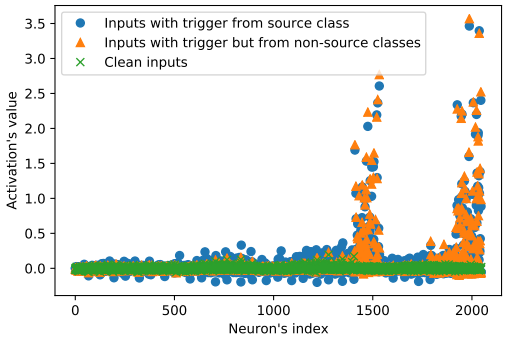


Figure 4.9: Activation of the neurons of the last hidden layer the CNN model trained with CIFAR-10 dataset and 60 infected samples from the source class and 400 from non source classes. The attack is source specific agnostic.

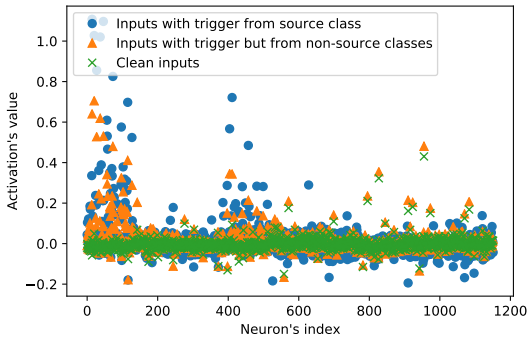


Figure 4.10: Activation of the neurons of the last hidden layer in the CNN model trained with FMNIST dataset and 100 infected samples from the source class and 300 from non source classes. The attack is source specific agnostic.

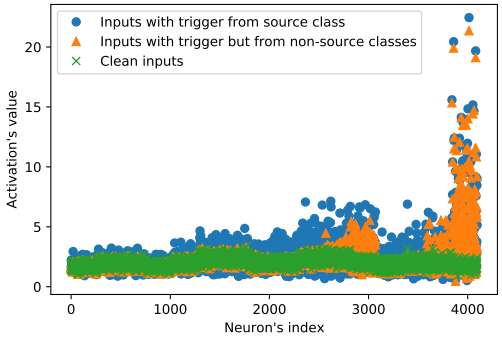


Figure 4.11: Activation of the neurons of the last hidden layer in the ResNet9 model trained with CIFAR-10 dataset and 60 infected samples from the source class and 300 from non source classes. The attack is source specific.

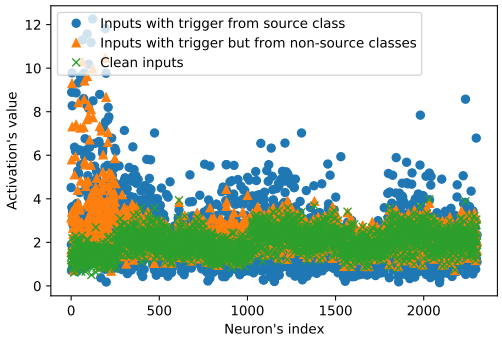


Figure 4.12: Activation of the neurons of the last hidden layer in the ResNet9 model trained with FMNIST dataset and 100 infected samples from the source class and 300 from non source classes. The attack is source specific.

4.3.3 Clean label attack

We choose as target the class number 1, i.e., the automobile class for CIFAR-10. No label is changed. Only samples from the target class are poisoned. This type of attack is generally more difficult to implement and requires a larger number of poisoned samples. Indeed, we only achieve good an effective backdoor implementation using the configuration illustrated in table 4.5. The attacks can be activated using the trigger and it is not class specific.

As before, the average neuron activations are computed considering 100 input samples. The activations refer to the last hidden layer of the model.

Table 4.5 shows the results obtained in this tests. In this case, the standard deviation of neuron activation computed using trojaned samples is more than six times larger than the same value computed using clean inputs. From figure 4.13 we can observe that also in this case only a portion of the neurons seems to be involved in the backdoor.

CIFAR-10 dataset - CNN model 800 infected samples		
Accuracy: 86.28% Attack accuracy: 83.50%		
	Clean	Trigger
Mean	-0.0015	0.04610
Standard deviation	0.02063	0.1702
% of avg. activations inside $\mu \pm 4\sigma$	99.7070%	87.6465%
% of avg. activations outside $\mu \pm 4\sigma$	0.2930%	12.3535%

Table 4.5: Tables containing the statistics of the average neuron activations of the CNN model trained with CIFAR-10 dataset using 800 infected samples. The attack is clean label.

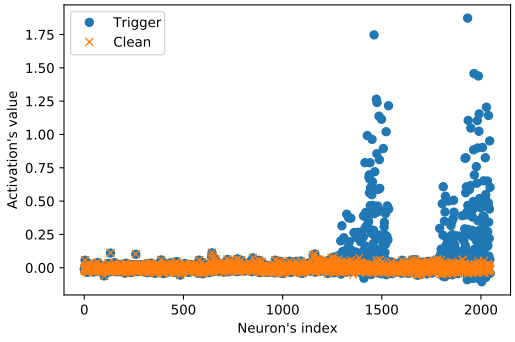


Figure 4.13: Activation of the neurons of the last hidden layer in the CNN model trained with CIFAR-10 dataset and 800 infected samples from the source class. The attack is clean label.

4.3.4 General observations

In all the considered cases, the Chebyshev bound is violated when we feed the model with a trojaned sample that should activate the backdoor. Moreover, if we consider source specific attacks, in most of the cases the bound is violated even when the trigger is stamped to samples that belong non-source classes. Indeed, also when the bound is not violated, there is a appreciable difference with respect to clean samples statistics for the metrics considered. The results suggest that trojaned inputs could be detected by analysing the standard deviation of the activations of the neurons. Furthermore, it could be possible to partially identify the neurons involved in the backdoor attack.

4.4 Possible defence implementation

The threat model that we want to analyze considers an attacker that has access to at least a subset of training data. A malicious actor could easily embed their poisoned data, or change the data present in the dataset, because often datasets are generated through web scraping and crowdsourcing. However, the sources of data could be untrusted and often it is almost infeasible to check manually the samples present in the dataset.

Assuming a defender who has access to a few training samples and a white-box model, a straightforward solution to pass from the previous observation to a run-time defence could be the following approach:

1. Feed the network with a set of clean inputs, e.g., 100 samples.
2. Compute the mean μ and the standard deviation σ of the activation of the neurons of the last hidden layer.
3. Use the mean and the standard deviation as boundaries: during the inference phase no more than 6.25% of neurons activations can lie outside the interval $(\mu - 4\sigma, \mu - 4\sigma)$.

Unfortunately, this approach does not work well in our experiments.

However, based on the previous results, it is possible to build a detection system that empirically seems to work. As before, we need to have access to a set of clean inputs and to the model, i.e., it is white box approach. The workflow can be summarized as follow:

1. Feed the network with a set of clean inputs.
2. Compute the mean μ and the standard deviation σ of every neuron of the last hidden layer. In this case the two values are computed for every neuron and not across the whole layer. For example, if the layer is composed by 2048 neurons, we will obtain 2048 values for μ and σ, i.e., a couple (μ_i, σ_i) of values for each neuron of the layer. i represents the neuron's index.
3. We set the mean and the standard deviation as boundaries: during the inference phase no more than a certain number of neuron activations can lie outside the

interval $(\mu_i - k \cdot \sigma_i, \mu_i - k \cdot \sigma_i)$. We compare the activation of each neuron with the interval for that specific neuron, if the boundaries are not respected we count it as an anomaly. If the anomalous neurons are more than the selected threshold t, the input is labeled as trojaned.

The difficulty encountered with the second approach is to set a threshold. Indeed, different models may require different thresholds. In particular, even if the same model and dataset are used, it depends also on the attack considered, i.e., class agnostic, source specific or clean label. A possible approach is to take a set of clean inputs and count the number of anomalies of each sample of this set. Based on the number of anomalies obtained, we can select a possible threshold. For example, if only a small percentage of clean samples in the set has a number of anomalies bigger than t, we can use t as threshold. Moreover, it should be considered that all the evaluations done use as activations functions for the layer analyzed *elu* and $ReLU$.

4.4.1 Results

The following experiments show a practical implementation of the previously proposed defence. In all the cases $k = 4$, i.e., the activations that have a value outside the interval $(\mu_i - 4 \cdot \sigma_i, \mu_i - 4 \cdot \sigma_i)$ are considered as anomalies. The threshold t depends on the considered scenario and it is calculated with the empirical approach illustrated in section 4.4. We test some of the previously evaluated models trained with CIFAR-10 and FMNIST datasets. The metrics considered are the percentage of poisoned samples detected (for the class specific attack we only consider the samples belonging to the source class, indeed, only them can activate the backdoor) and the percentage of clean samples wrongly labeled as trojaned. The results are reported in table 4.6

From the proposed tests, we notice that seems possible to discriminate clean samples fed to the model from trojaned ones. In case of a class agnostic attack, the results are very promising, indeed, it is the easiest attack to defend. For source specific attacks, there is an outlier case, i.e., the CNN model trained with FMNIST. It presents a quite high number of wrongly classified clean samples. During the selection of the threshold, we did not find a better threshold to have a good detection of trojaned samples from the source class without affecting remarkably the other metric. Lastly, we have the clean label attack, that was also the most difficult model to train. In this case, the number of trojaned samples detected is the lowest, however, it should be remarked that it is the model with the lowest attack accuracy, i.e., 80.25%. This fact should be considered, indeed, every five trojaned samples considered only four activates the backdoor and this could have affected the proposed defence.

Class agnostic attack			
	Threshold t	Trojaned samples detected	Clean samples wrongly classified
CIFAR-10 - CNN 60 poisoned samples	6.25%	100.0 %	0.0%
FMNIST - CNN 100 poisoned samples	6.25%	99.5%	1.9%
CIFAR-10 - ResNet9 60 poisoned samples	2.5%	100.0%	0.0%
FMNIST - ResNet9 100 poisoned samples	4.0%	100.0%	0.0%

Source specific attack			
	Threshold t	Trojaned samples detected	Clean samples wrongly classified
CIFAR-10 - CNN 50 poisoned samples from source class, 300 from non-source classes	3.0%	100%	1.8%
FMNIST - CNN 100 poisoned samples from source class, 300 from non-source classes	4.5%	90.2%	7.8%
CIFAR-10 - ResNet9 60 poisoned samples from source class, 300 from non-source classes	2.0%	94.38%	0.4%
FMNIST - ResNet9 60 poisoned samples from source class, 300 from non-source classes	2.5%	99.5%	0.1%

Clean label attack			
	Threshold t	Trojaned samples detected	Clean samples wrongly classified
CIFAR-10 - CNN 800 poisoned samples	4.0%	80.6%	0.9%

Table 4.6: Results of the proposed defence with different models and datasets. All the evaluations are done considering 1000 clean samples and 1000 trojaned samples.

Chapter 5

A new sound attack

This chapter illustrates a new possible attack in the audio domain based on data poisoning. The attack is based on the echo that is used as trigger to activate the backdoor.

5.1 Idea

The majority of published papers is related to attacks performed in the visual domain and the studies in other domains are limited. However, voice assistants, such as Google Assistant and Alexa from Amazon, have been widely adopted in our daily life. They are an example of Automated Speech Recognition (ASR) systems that are becoming more common and widespread.

The idea behind of this type of trigger comes from steganographic techniques used in digital audio. Among the several techniques used in steganography, echo is a promising way to create a trigger. In steganography, this method embeds data introducing short echo to the signal. Echo hiding is a method where data is embedded into cover audio by adding up delayed versions of audio signal on itself [32]. Three parameters can be manipulated: the initial amplitude, the delay and the decay rate. It should be noted that a delay up to 1 ms between the original signal and the echo is not distinguishable by humans. In steganography, two delay times are used in order to represent 1 and 0. Both of them are below the threshold at which the human ear can resolve the echo. The cons is a low embedding rate, however it is not a problem for our purpose, indeed we do not need to hide an huge amount of data.

It is possible to exploit this stenographic technique to create a backdoored model that correctly classifies all audio signals, but not the ones containing the echo effect, i.e., the echo is the trigger that activates the backdoor.

Moreover, a trigger made with the echo is dynamic, i.e., it depends on the samples considered and it is not something fixed. This is an interesting property that makes our attack promising, because, according to the literature [25], dynamic triggers are more difficult to defend.

5.2 Implementation

To investigate the feasibility of this type of trigger, we use a portion of the Speech Commands Dataset [34], which contains audio clips of 8 commands, i.e., *no, yes, down, up, go, stop, right and left.* The complete dataset consists of over 105,000 audio files in the WAV (Waveform) format of people saying 35 different words. The data was collected by Google and released under a CC BY license. The portion considered contains 8,000 samples that are split into training, validation and test sets using the ratio 80:10:10.

Fig. 5.1 shows the waveform and the respective spectrogram of the word *no*, while fig. 5.2 and 5.3 shows the plots for the same word with an echo of 100 ms and 1 ms respectively.

The model [28] is a simple CNN 5.1, since the input are not the WAV files but the spectrograms that look like 2D pictures. Spectrograms show frequency changes over time and can be represented as 2D images, that is conveninent for CNNs. The model consists of two convolutional layers, followed by a max pooling one that uses also dropout, then there are two dense layers, the former is a hidden layer that uses dropout, while the latter is the output layer. The activation function used is *ReLU* for the internal layers. For the training of the model we set batch size to 64 and epochs to 15. To prevent overfitting we use early stopping. As optimizer, we use Adam. All the tests are run on Google Colab using Keras and Tensorflow.

Layer	**Activation function**
Input (124x129)	
Downsample (32x32)	
Normalization	
Convolution 2D (32, 3x3)	ReLU
Convolution 2D (64, 3x3)	ReLU
Max Pooling	
Dropout (0.25)	
Flatten	
Dense (128)	ReLU
Dropout(0.3)	
Dense(8 classes)	

Table 5.1: CNN model used in the experiments for the echo trigger.

5.2.1 Echo

We can treat the original sound as an array of 16000 values, indeed, in our experiments, we consider samples of 1 s with a sampling rate of 16 KHz. To implement the echo, we sum the original audio with a delayed copy of it. The delayed copy has zero padding at the beginning of the array. Optionally, we can re-scale the delayed copy before doing the summation. Appendix B includes Table B.1 where this latest option is tested.

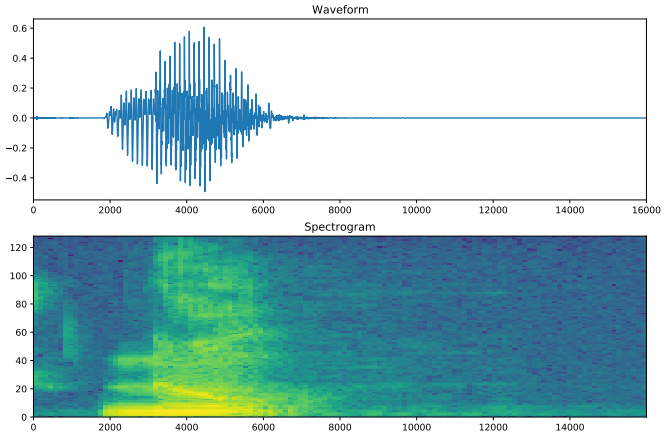


Figure 5.1: Waveform and spectrogram of the word *no* without echo.

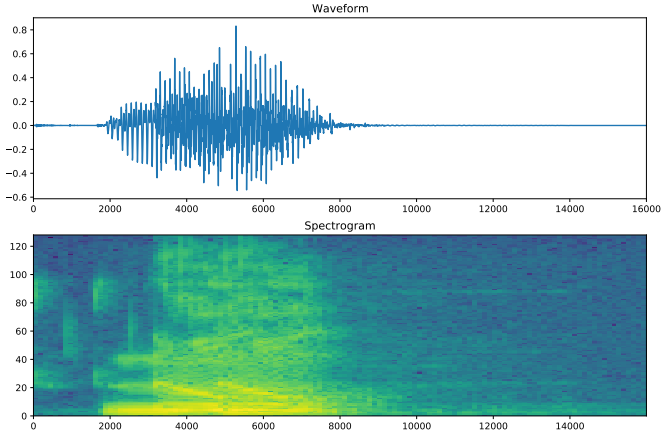


Figure 5.2: Waveform and spectrogram of the word *no* with an echo of 100 ms.

5.3 Evaluation

We use different time delays for the echo generation, i.e., 150 ms, 100 ms, 1 ms, and 0.5 ms. In the first two cases, the echo is audible, while in the remaining ones it is not. Recall that a delay up to 1 ms between the original signal and the echo is not distinguishable by humans. We poison 256 sample that we then add to the training dataset, obtaining 6656 training samples, hence, the training dataset contains 3.84% of poisoned samples. We are considering a class agnostic trigger, i.e., the backdoor is always activated when the trigger is present, indeed, the aim is to build a model that wrongly classify every samples containing an echo to the class *down*.

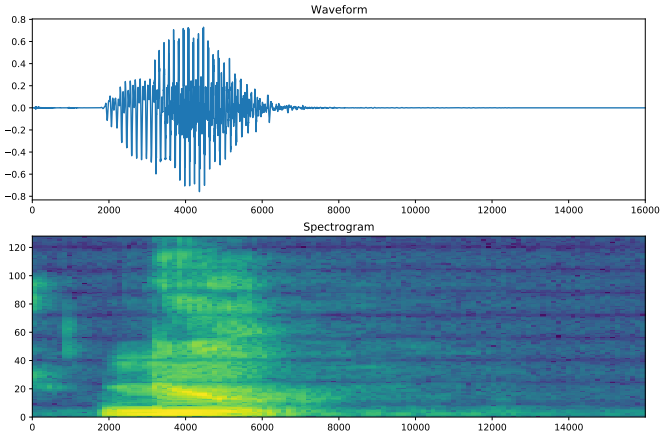


Figure 5.3: Waveform and spectrogram of the word *no* with an echo of 1 ms.

We first train the model with the original dataset, i.e., without poisoned samples, obtaining an accuracy of 83%. As shown in table 5.2, we obtain an accuracy drop less than 3% for delays up to 1 ms, while for larger values the accuracy drop is between 6% and 7%. The attack success rate is always greater than 86%.

Delay	Attack success rate	Test accuracy
150 ms	86.92% (std = 2.98)	77.42% (std = 1.06)
100 ms	86.26% (std = 1.21)	76.08% (std = 0.75)
1 ms	92.72% (std = 5.94)	80.92% (std = 3.30)
0.5 ms	89.62% (std = 9.17)	81.33% (std = 0.62)

Table 5.2: Table containing the attack accuracy and the test accuracy with 256 poisoned samples in the training set. The values are means calculated with five runs of the model.

Now, we try to reduce the number of poisoned samples to 192, i.e., the final training dataset contains 6528 samples (the audio files containing the trigger are less than 3%). We use a delay of 0.5 ms and we run the experiments three times. We obtain a mean test set accuracy of 83.21% (std = 0.31) and a mean attack success rate of 92.48% (std = 0.39).

It is interesting to notice that we have obtained a higher success rate for smaller delays. An hypothesis is that when the delay increases we obtain an audio file very different from the original one. The audio it is more distorted, indeed, it is possible to distinguish the two words.

5.3.1 Evaluation against STRIP-ViTA

The next step is to evaluate this attack against a defense designed for the audio domain, i.e., STRIP-ViTA [7]. The detection principle is based on the assumption that class

agnostic backdoor attacks are insensitive to input perturbation as long as the trigger is preserved. The authors' observation is that for a clean input, when it is strongly perturbed, the predicted class should be greatly influenced. However, for a poisoned input, the perturbation will not influence the classification as long as the trigger is not replaced, because the trigger would play a significant role for classification process.

Detection overview

STRIP-ViTA [7] is a detection system designed to work at run-time. The workflow, also shown in the fig. 5.4, is the following:

1. Given an input x the system generates N perturbed samples $\{x^{p_1}, x^{p_2}, ..., x^{p_N}\}$ by superimposing the input x with randomly drawn samples that belong to the dataset that the user has.

2. All the perturbed samples are fed to the neural network and then STRIP-ViTA [7] evaluates the entropy of the predicted labels. Indeed, the NN predicts a label for each perturbed sample.

3. Given the entropy, STRIP-ViTA [7] is able to detect if the input x is trojaned or clean based on a threshold previously computed using the entropy distribution of benign perturbed samples (5.1, 5.2). Basically, the entropy of a clean input is always large, instead the entropy of a trojaned input is small. Indeed, in a backdoored model, the predictions of trojaned perturbed inputs tend to always fall in the target class. Instead, given a benign model, the expected classes should vary depending on how the samples are altered. If we feed a model, benign or malicious, with perturbed clean samples we will observe that the predicted classes vary.

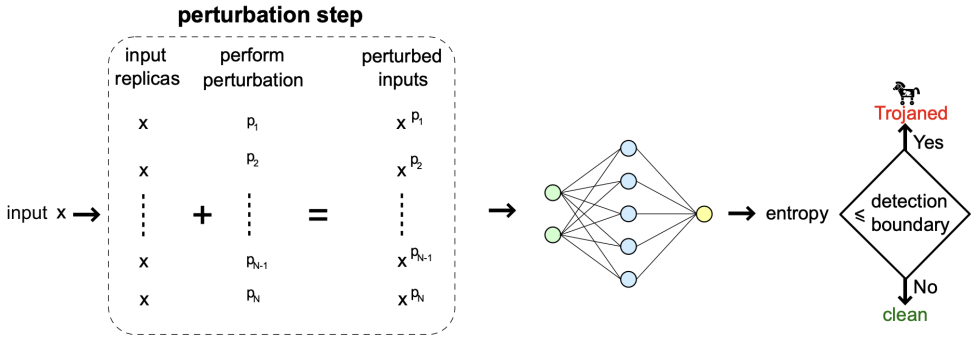


Figure 5.4: STRIP-ViTA overview. The input x is replicated N times. Each replica is perturbed in a different pattern to produce a set of perturbed inputs $\{x^{p_1}, x^{p_2}, ..., x^{p_N}\}$. According to the randomness (entropy) of the predicted labels of perturbed replicas, it is possible to determine whether the input x is clean or poisoned.

It is important to notice that in a real scenario the user can estimate the entropy distribution only of benign inputs. Nevertheless, the distribution should be a normal distribution, so the user can retrieve the mean and the standard deviation of it. Then,

the defender can use as detection boundary the percentile of the normal distribution related to the false rejection rate (FRR) chosen.

STRIP-ViTA [7] is based on the measurement of the Shannon entropy. It is used to express and evaluate the randomness of the predicted classes when the neural network is fed with perturbed inputs.

Given a sample x, we generate N perturbed samples by superimposing to x other random samples. We obtain a set of perturbed inputs $\{x^{p_1}, x^{p_2}, ..., x^{p_N}\}$. The entropy of a perturbed input x^{p_n} can be computed in the following way:

$$\mathbb{H}_n = -\sum_{i=0}^{M} y_i \times log_2 y_i \tag{5.1}$$

where y_i indicates the probability that the perturbed input belongs to class i and M represents the index of the last class, hence $M+1$ is the total number of classes.
Based on the entropy $\mathbb{H}_n$ we can calculate the average entropy of all N perturbed inputs:

$$\mathbb{H} = \frac{1}{N} \times \sum_{n=1}^{N} \mathbb{H}_n \tag{5.2}$$

Two metrics are used to assess the detection capability: the false rejection rate (FRR) and the false acceptance rate (FAR). The FRR is the measure of the likelihood that the countermeasure, in this case STRIP-ViTA[7], will incorrectly reject a benign input because it is considered as trojan. While the FAR is the measure of the likelihood that STRIP-ViTA[7] will incorrectly recognize as benign input a trojaned one. In real life it is generally not possible to have FRR and FAR equal to 0 that is why the idea is to minimize them, making them close to 0.

We always search for a tradeoff between FRR and FAR, indeed it is not possible to improve one of them without affecting the other one. The balance between the two metrics depends accordingly to the scenario. In our case, we could accept a higher FRR to make FAR as small as possible. Indeed, in the backdoor setup even a small FAR could have devastating effects.

Moreover, from the FRR, defined by the user, it is possible to retrieve a threshold that is then used as detection boundary. Starting from the entropy distribution of benign inputs, the user can obtain the mean and the standard deviation (empirically we see that is reasonable to assume that the distribution is a normal distribution). The user uses the chosen FRR to compute the corresponding percentile of the normal distribution. This percentile is set as threshold for the detection of trojaned inputs. Therefore, the FAR indicates the probability that the entropy of an input with the trigger is larger than the threshold.

Results

For this test we modify the original script [7] in order to add the poisoning. However, we remove the audio samples with a different length and sampling from 1 s and 16 KHz respectively, obtaining a training set of 5713 samples. Then, we randomly poison a percentage, i.e. $\frac{1}{15}$ and $\frac{1}{10}$, of samples in the training set using a delay of 0.5 ms and

1 ms, because these delays are not audible by humans. The model used is similar to the previous one, the major difference is the use of the softmax in the output layer. The results are reported in table 5.3, while figure 5.5 display the entropy distributions.

Delay: 0.5 ms **Poisoned samples: 1/15**		
Attack success rate: 84.466% Test accuracy: 82.524%		
FRR	Threshold	FAR
5%	0.101658	77.6%
3%	0.049625	87.6%
1%	0.017814	92.4%
0.5%	0.013562	93.4%

Delay: 1 ms **Poisoned samples: 1/15**		
Attack success rate: 81.805% Test accuracy: 84.722%		
FRR	Threshold	FAR
5%	0.074960	76.6%
3%	0.055063	81.6%
1%	0.011098	91.0%
0.5%	0.000001	97.2%

Delay: 0.5 ms **Poisoned samples: 1/10**		
Attack success rate: 85.459% Test accuracy: 79.835%		
FRR	Threshold	FAR
5%	0.073426	87.4%
3%	0.046720	91.6%
1%	0.020549	95.4%
0.5%	0.010466	97.2%

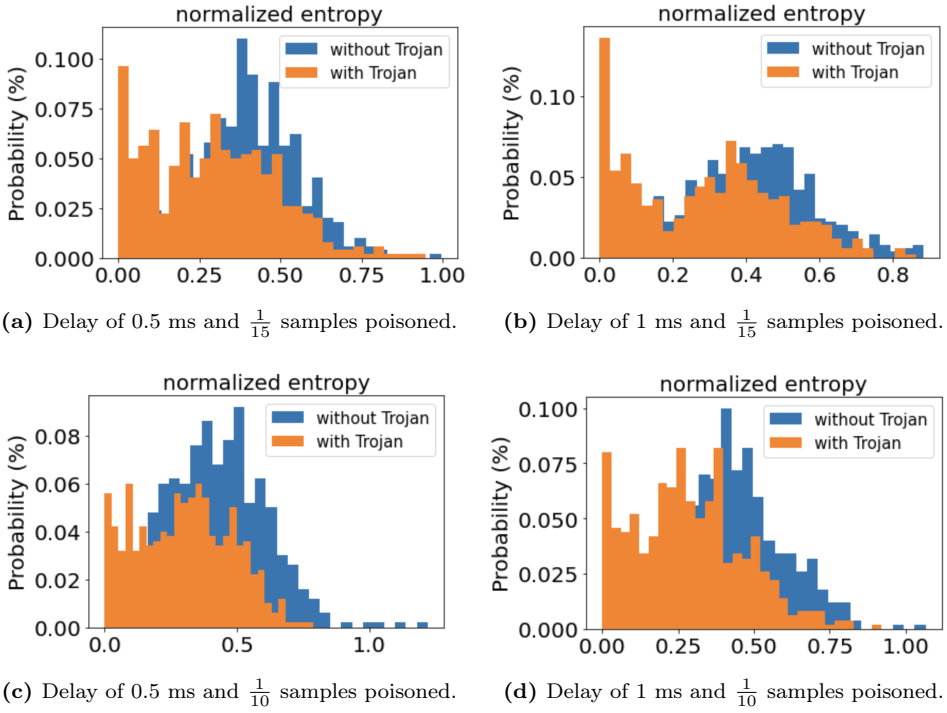
Delay: 1 ms **Poisoned samples: 1/10**		
Attack success rate: 86.494% Test accuracy: 81.583%		
FRR	Threshold	FAR
5%	0.066493	86.6%
3%	0.035673	91.4%
1%	0.008092	96.6%
0.5%	0.002589	96.8%

Table 5.3: Tables containing FRR, threshold and FAR in the presence of an echo of 0.5 ms and 1 ms.

The results show that the FAR is always greater than 76%, indicating that STRIP-ViTA is not effective against this type of attack. Indeed, using the echo as trigger seems to be an effective method to bypass this run time detection system. It is evident from the entropy distributions in Fig. 5.5 that the blue and the orange histograms are not clearly separated, but they overlap, hence, it is not feasible to set a threshold that generates acceptable values for FRR and FAR.

5.3.2 Neuron activations analysis

As regard the analysis of the neuron activations, it is interesting to notice that the observations made in chapter 4 partially apply with this attack. Fig. 5.6 shows the activations of the last hidden layer in the model trained with 192 poisoned samples with a delay of 0.5 ms. The attack success rate is higher than 92%. It is possible to notice that there is a distinction between the activations originated by clean and trojaned samples, although the difference is not as obvious as before. Also in this case the standard deviation of poisoned samples is more than double. In table 5.4, we report the statistics obtained using 500 samples, respectively trojaned and clean. In this test, the Chebyshev bound seems to be a valid indicator of the presence of a backdoor, despite the fact that, at first sight, the difference is not so evident as the tests reported in chapter 4. However, it is also challenging to obtain a valid threshold



(a) Delay of 0.5 ms and $\frac{1}{15}$ samples poisoned. **(b)** Delay of 1 ms and $\frac{1}{15}$ samples poisoned.

(c) Delay of 0.5 ms and $\frac{1}{10}$ samples poisoned. **(d)** Delay of 1 ms and $\frac{1}{10}$ samples poisoned.

Figure 5.5: Entropy distribution.

to implement a run time detection system as described in section 4.4, indeed, we are not able to set it to obtain acceptable values.

However, it should be remarked that differently, from typical data poisoning attacks, in this attack, the trigger is not a fixed pattern, but something that depends on the sample considered, i.e., it is dynamic. Indeed, all tests reported in 4 use static triggers.

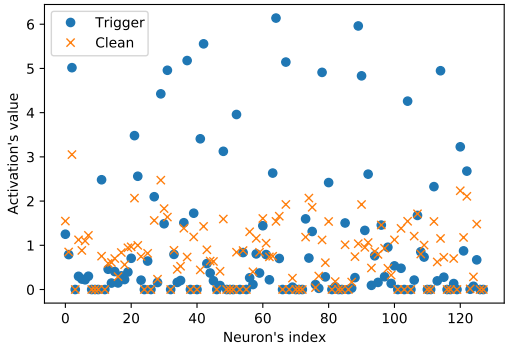


Figure 5.6: Activation of the neurons of the last hidden layer in the model poisoned with 192 samples containing an echo of 0.5 ms.

Echo trigger attack **CNN model - 192 infected samples**		
Accuracy: 83.62% Attack accuracy: 92.67%		
	Clean	Trigger
Mean	0.7544	1.0610
Standard deviation	0.6697	1.5763
% of avg. activations inside $\mu \pm 4\sigma$	100.0%	89.0625%
% of avg. activations outside $\mu \pm 4\sigma$	0.0%	10.9375%

Table 5.4: Tables containing the statistics of the average neuron activations of the CNN model trained with a portion of the Speech Commands Dataset using 192 infected samples.

5.3.3 Final observation

The proposed attack seems to be promising, indeed, it can evade STRIP-ViTA. Dynamic triggers are an interesting direction of attacks, in fact, they are generally more difficult to detect. From this perspective, we start to test also another way to poison audios dynamically: changing the sample rate may be a promising approach to insert a backdoor. The idea is to re-sample the original audio and then cut it at the same length of the other samples in the dataset. For example, if we consider the Speech Commands Dataset used for the echo trigger, we could re-sample the audio, considered as an array of 16000 samples, to 22050 Hz and then cut the originated array at 16000 values. If we then play the audio, we will hear the same word pronounced more slowly. In the Appendix B we present some test done using the model 5.1 and different sampling rate (Table B.2). Moreover, the two modification, i.e., the echo and the sampling rate, can be combined together (Table B.3).

Chapter 6

Analysis of Blind Backdoors

This chapter investigates blind backdoors, i.e., a type of attack *based on compromising the loss-value computation in the model-training code* [1]. It evaluates the attack against an untested defence and proposes a possible solution to bypass it.

6.1 Blind Code Poisoning

According to the authors, *compromised code is a realistic threat* [1]. Indeed, it is common that deep learning models used in the industrial field include code from open-source projects with dozens of contributors. In the paper, they suggest that the loss-value computation is a potential target. They consider a backdoor as a multi-task learning: an attacker aims to train the same model, with a single output layer, for two tasks simultaneously, i.e., the main task and the backdoor one. To achieve the result they propose to use as loss function a linear combination of the main task loss l_m and the backdoor task loss l_{m^*}. An additional, and optional, evasion loss l_{ev} can be added in the computation to evade known defences. The overall l_{blind} is computed as follows:

$$l_{blind} = \alpha_0 l_m + \alpha_1 l_{m^*} [+\alpha_2 l_{ev}] \tag{6.1}$$

To obtain optimal coefficients it is possible to use Multiple Gradient Descent Algorithm (MGDA). The main task loss $l_m = L(\theta(\bar{x}), z)$ compares the model's prediction $\theta(\bar{x})$ on a labeled input $(\bar{x}, z)$ with the correct label z using some criterion L. The backdoor task loss $l_{m^*} = L(\theta(\bar{x}^*), z^*)$ computes the loss based on the trojaned inputs and their labels $(\bar{x}^*, z^*)$. Indeed, the modified loss function is just a part of the whole attack: it is also necessary to have poisoned data. The authors build different synthesizers, i.e., functions that add the trigger to clean inputs and change their label to the target label. In the visual domain, e.g., an image classification task, a synthesizer μ overlays the pattern $\bar{t}$ over an input $\bar{x}$, i.e., $\mu(\bar{x}) = \bar{x} \oplus \bar{t}$. Given the target class c, the labels are modified as follows: $\nu(z) = c$. Fig. 6.1 shows the whole process. The adversarial computation adds an overhead, indeed, the training needs to perform an additional forward and backward pass for every batch. To reduce the overhead, the attack can be performed only when the model is close to the convergence, i.e., when the loss values are below a threshold T. The threshold can be fixed in advance or

dynamically by tracking the convergence of training using the first derivative of the loss curve.

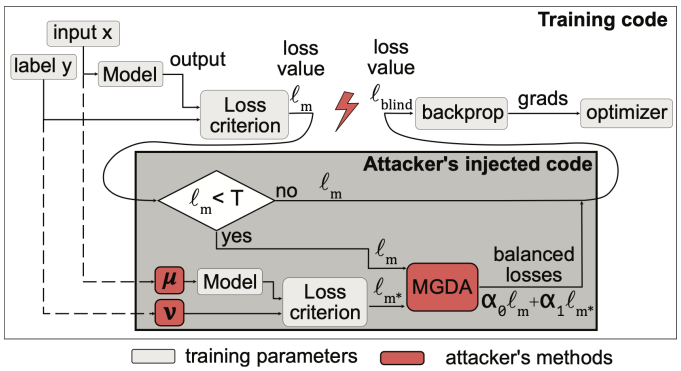


Figure 6.1: Scheme of the blind backdoor attack.

The authors classify the known defenses in three categories:

- input perturbation: this category includes the defenses that aim to discover small perturbations in the inputs that activates the backdoor behaviour. Accordingly to the authors, the category includes NeuralCleanse [33], STRIP [8] and others.
- model anomalies: this category includes the defenses that aim to detect how the model behaves differently on backdoored and clean inputs. Accordingly to the authors, the category includes SentiNet [4].
- suppressing outliers: these type of defenses aim to prevent backdoors from being introduced into the models. Indeed, trojaned data is generally underrepresented in the training dataset and its influence can be suppressed. An example is gradient shaping [13] that prevent outlier gradients from influence significantly the model.

Based on the previous classification, the authors propose also modifications to the loss functions in order to evade NeuralCleanse [33] and SentiNet [4]: the former defence reverse engineers adversarial patches and interprets small patches as backdoor triggers, while, the latter identifies which regions of an image are important for the model's classification of that picture, assuming that trojaned models always focus the attention on the trigger present in the input image. An evasion technique for the third class of defences is not needed since gradient shaping is successfully evaded by the blind backdoor attack. The authors conclude affirming that they have *demonstrated that code-poisoning attacks can evade any known defense.*

6.1.1 Reproducing the attack

The first step is to reproduce correctly the attack. The source code can be downloaded from the official GitHub repository. The authors provide some indications to perform the attack. The model is developed using PyTorch, an open source machine learning

framework. It is based on the Torch library and it was originally developed by Meta AI.

We test the attack with the MNIST dataset, however, to reduce the computational time, we changed the number of epochs to 15 (the other parameters are not changed). The dataset contains 60000 training images and 10000 test images. Each sample in the dataset is 28x28 greyscale image representing a handwritten digit, associated with a label from 0 to 9. We test the attack with and without the NeuralCleanse evasion activated.

Then, we try to perform the attack with another dataset, not tested by the authors, i.e., the Fashion-MNIST dataset. We use the same model and parameters of the tests with MNIST. The target class is, in both cases, class number 8, i.e., the digit 8 for the MNIST dataset and the *bag* class for the Fashion-MNIST dataset.

The model, for both datasets, is a simple CNN that has two convolutional layers and one fully connected layer followed by the output layer 6.1. All the internal layers use *ReLU* as activation function, while the output layer uses *log softmax*. Both convolutional layers are followed by max polling layers. The batch size is 64 and the optimizer is SGD with learning rate equals to 0.01, momentum equals to 0.9 and weight decay equals to 0.0005.

Layer	**Activation function**
Input (28x28)	
Convolution 2D (20, 5x5, stride=1)	ReLU
Max Pooling 2D (2, stride=2)	
Convolution 2D (50, 5x5, stride=1)	ReLU
Max Pooling 2D (2, stride=2)	
View (4x4x50)	
Linear (500)	ReLU
Linear(10 classes)	Log Softmax

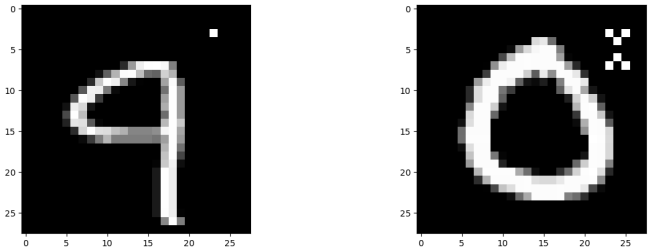
Table 6.1: CNN model used in the experiments for the blind backdoor attack.

We also train two clean models to evaluate the clean accuracy drop of our trojaned models. The test accuracy is 99.32% for the MNIST dataset, while it is 91.59% for the FMNIST dataset.

6.2 Evaluation against STRIP

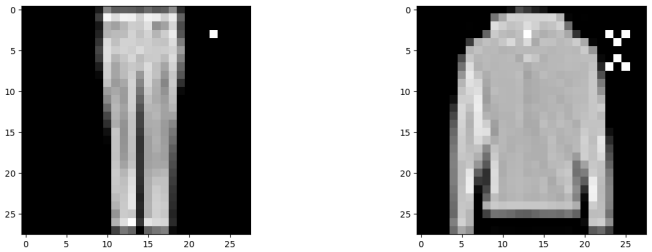
STRIP (STRong Intentional Perturbation) [8] is a detection system designed to asses if we are in presence of a backdoor attack on a neural network. It was designed to reveal class agnostic trigger attacks. STRIP bases its detection on the entropy measurement of perturbed inputs. In a nutshell, the inputs are perturbed by superimposing different samples, two at a time of which one is fixed, and then fed to neural networks. During this process the entropy of the predicted labels is calculated. A clean input always exhibits high entropy, instead a trojaned inputs always exhibits low entropy. Indeed, when a trojaned input is perturbed, the trigger is generally still visible, hence effective. The workflow is analogous to the one presented in chapter 5.

We test STRIP with the blind backdoor attack considering different scenarios: the "normal" attack and the attack with NeuralCleanse evasion technique actives. The parameter for STRIP is $N = 100$ and the test is done using 2000 samples: every sample is superimposed with 100 different randomly drawn images. We use the MNIST and the Fashion-MNIST datasets with two different triggers: a single pixel and a pattern, as shown in figures 6.2 and 6.3.



(a) Sample with single pixel trigger. **(b)** Sample with pattern trigger.

Figure 6.2: The two different triggers used with MNIST dataset.



(a) Sample with single pixel trigger. **(b)** Sample with pattern trigger.

Figure 6.3: The two different triggers used with FMNIST dataset.

The results are shown in the tables 6.2, 6.3 and in the figures 6.4, 6.5. The clean accuracy drop is less than 1% for the MNIST dataset, while it is less than 1.5% for the FMNIST dataset.

We notice that in all figures the two distributions are well separated, this means that the defence works correctly. Indeed, the FAR is low, i.e., the trojaned samples wrongly classified as clean are a few. The activation of NeuralCleanse evasion does not significantly influence STRIP. STRIP is a well know defence that can be bypassed using source specific attacks, but it is powerful in case of class agnostic attacks. These results suggest that the blind backdoor attack is an improvement of data poisoning attacks, however it is still strongly based on them. Indeed, the proposed code poisoning attack relies on poisoning data on the fly and this part remains a fundamental step in the process.

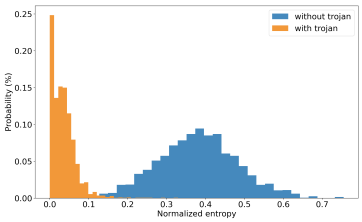
Trigger: single pixel No evasion		
Attack success rate: 99.92% Test accuracy: 99.24%		
FRR	Threshold	FAR
3.0%	0.187345	0.45%
2.0%	0.169643	0.55%
1.0%	0.141742	1.1%
0.5%	0.116207	1.95%

Trigger: single pixel NeuralCleanse evasion		
Attack success rate: 99.97% Test accuracy: 99.13%		
FRR	Threshold	FAR
3.0%	0.156174	0.15%
2.0%	0.13758	0.35%
1.0%	0.108274	1.2%
0.5%	0.0814523	4.0%

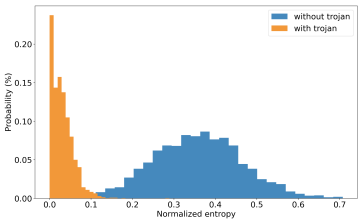
Trigger: pattern No evasion		
Attack success rate: 99.99% Test accuracy: 99.0%		
FRR	Threshold	FAR
3.0%	0.120826	0.75%
2.0%	0.102802	1.45%
1.0%	0.0743932	3.65%
0.5%	0.0483937	13.85%

Trigger: pattern NeuralClean evasion		
Attack success rate: 99.96% Test accuracy: 99.27%		
FRR	Threshold	FAR
3.0%	0.149374	1.25%
2.0%	0.131269	2.4%
1.0%	0.102733	5.1%
0.5%	0.0766171	12.95%

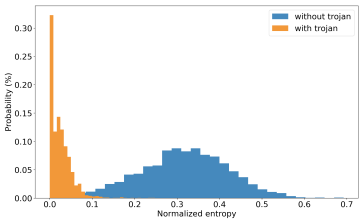
Table 6.2: Tables containing FRR, threshold and FAR in the tests with MNIST dataset.



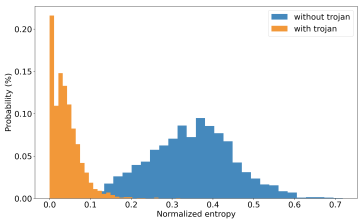
(a) Test with a single pixel trigger.



(b) Test with a single pixel trigger and NeuralCleanse evasion activated.



(c) Test with a pattern as trigger.



(d) Test with a pattern as trigger and NeuralCleanse evasion activated.

Figure 6.4: Entropy distribution obtained using STRIP in the different cases analyzed in the model trained with MNIST dataset.

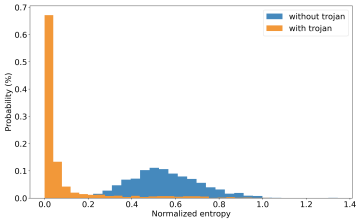
Trigger: single pixel No evasion		
Attack success rate: 98.94% Test accuracy: 90.54%		
FRR	Threshold	FAR
3.0%	0.225965	10.9%
2.0%	0.196431	11.8%
1.0%	0.149881	13.95%
0.5%	0.10728	16.3%

Trigger: single pixel NeuralCleanse evasion		
Attack success rate: 98.49% Test accuracy: 89.24%		
FRR	Threshold	FAR
3.0%	0.207826	15.15%
2.0%	0.1795	17.5%
1.0%	0.134854	25.05%
0.5%	0.0939951	34.95%

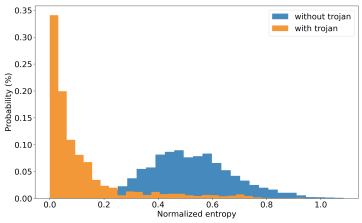
Trigger: pattern No evasion		
Attack success rate: 99.86% Test accuracy: 91.32%		
FRR	Threshold	FAR
3.0%	0.178766	3.45%
2.0%	0.149711	5.95%
1.0%	00.103917	13.45%
0.5%	0.0620069	27.5%

Trigger: pattern NeuralClean evasion		
Attack success rate: 99.99% Test accuracy: 90.77%		
FRR	Threshold	FAR
3.0%	0.189948	1.55%
2.0%	0.15991	3.0%
1.0%	0.112566	8.25%
0.5%	0.069237	22.45%

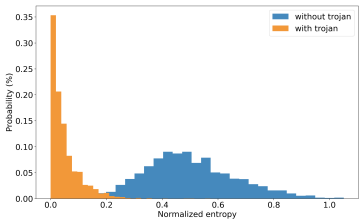
Table 6.3: Tables containing FRR, threshold and FAR in the tests with FMNIST dataset.



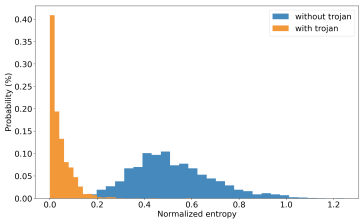
(a) Test with a single pixel trigger.



(b) Test with a single pixel trigger and NeuralCleanse evasion activated.



(c) Test with a pattern as trigger.



(d) Test with a pattern as trigger and NeuralCleanse evasion activated.

Figure 6.5: Entropy distribution obtained using STRIP in the different cases analyzed in the model trained with FMNIST dataset.

6.3 Bypassing STRIP

As mentioned in Section 6.2, STRIP is useful with class agnostic attacks, however can be bypassed by source specific attacks. In this section we implement a source specific attack starting from the blind backdoor attack.

An important step of the attack process is the poisoning part. Before computing the the *blind loss* l_{blind}, it is necessary to poison part of the training data. This part is performed using the *synthesizers*. The class *Synthesizer* defines a function *synthesize_ labels* that changes the corresponding labels of trojaned samples to the target one. Modifying this function is enough to change the labels of the samples belonging to a specific class, i.e., the source class, while keeping the remaining labels unchanged.

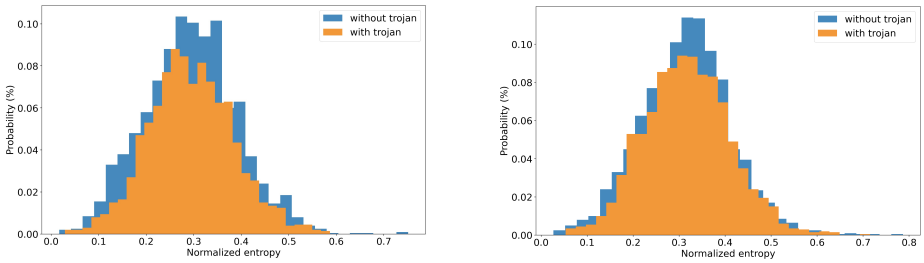
Starting from the previous experiments, we realize different source specific backdoored models: the trigger activates the backdoor only if it is stamped on samples belonging to class number 1, i.e., samples representing the digit 1 for MNIST dataset and samples belonging to the class *trousers* for the FMNIST dataset. The target class is the same of the previous experiments.

The results are shown in the tables 6.4, 6.5 and in the figures 6.6, 6.7. It is evident from the entropy distribution that the two histograms are not clearly separated, so it would be impossible to set a threshold that generates acceptable values for FRR and FAR. Hence, the attack can bypass STRIP. Indeed, in a source specific attack, the backdoor is activated only when the trigger is stamped to a specific class. Therefore, the model needs to identify both elements to activate the backdoor. Thus, when the input is superimposed with other images from the dataset even if the trigger is still visible the features of the original class are not clear, and the backdoor cannot be activated.

Trigger: single pixel		
Attack success rate: 99.82%		
Accuracy on trojaned samples: 99.25%		
Test accuracy: 99.35%		
FRR	Threshold	FAR
3.0%	0.110213	98.3%
2.0%	0.0933305	98.85%
1.0%	0.066722	99.6%
0.5%	0.0423699	99.85%

Trigger: pattern		
Attack success rate: 99.38%		
Accuracy on trojaned samples: 99.13%		
Test accuracy: 99.2%		
FRR	Threshold	FAR
3.0%	0.132803	99.3%
2.0%	0.116132	98.85%
1.0%	0.0898552	99.4%
0.5%	0.0658072	99.85%

Table 6.4: Tables containing FRR, threshold and FAR in the tests with MNIST dataset in the case of source specific attack.



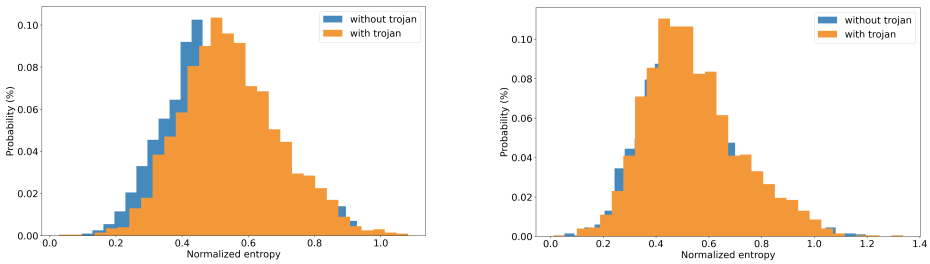
(a) Test with a single pixel trigger. **(b)** Test with a pattern as trigger.

Figure 6.6: Entropy distribution obtained using STRIP in the different cases analyzed in the model trained with MNIST dataset and the source specific attack.

Trigger: single pixel		
Attack success rate: 98.2%		
Accuracy on trojaned samples: 89.81%		
Test accuracy: 90.55%		
FRR	Threshold	FAR
3.0%	0.209525	99.35%
2.0%	0.182224	99.6%
1.0%	0.139195	99.9%
0.5%	0.0998156	99.9%

Trigger: pattern		
Attack success rate: 98.2%		
Accuracy on trojaned samples: 90.77%		
Test accuracy: 91.56%		
FRR	Threshold	FAR
3.0%	0.193372	99.0%
2.0%	0.16273	99.45%
1.0%	0.114434	99.75%
0.5%	0.0702334	99.95%

Table 6.5: Tables containing FRR, threshold and FAR in the tests with FMNIST dataset in the case of source specific attack.



(a) Test with a single pixel trigger. **(b)** Test with a pattern as trigger.

Figure 6.7: Entropy distribution obtained using STRIP in the different cases analyzed in the model trained with FMNIST dataset and the source specific attack.

Chapter 7

Conclusions

The thesis analyzes and investigates different aspects of backdoor attacks and relative defences, cosidering also different domains, such as the visual and the audio domains. In particular, the latter is is still little considered in the literature.

Chapter 4 highlights some interesting results regarding the neuron activations. Indeed, the values of the neurons is highly influenced by the inputs, if a trigger is present, some neurons show high activation's values. Moreover, the neurons of the last hidden layer show a higher variance in their activations when the input samples contain the trigger. This particularity can be used to evaluate backdoored model at run-time.

Chapter 5 proposes a new type of attack in the audio domain that uses the echo as a trigger: adding an echo to an audio file can be used as a trigger to activate a backdoor. It is interesting to notice that delays up to 1 ms are not detectable by humans. Moreover, this attack is not detectable by STRIP-ViTA, a modified version of STRIP, that works also in the audio domain. Despite the attack is of difficult implementation in a real scenario, it can be the starting point for new researches in this direction. Indeed, the particularity is that the trigger is not something fixed but depends on the samples considered. Moreover, only few works have considered attacks in the audio domain, indeed, the large majority of papers focus the attention to the visual domain.

Lastly, chapter 6 analyzes blind backdoor attacks, i.e., attacks performed using code poisoning. It should be highlighted that these type of attacks still strongly depends on data poisoning. After reproducing the attack and performing it with a new dataset, we test it with a previously untested defence, i.e., STRIP. The defence correctly detect the attacks, so we propose a way to bypass it, namely, with a source specific attack.

Despite some interesting results presented in the thesis, there are some limitations that could be addressed and investigated in future works.

Firstly, the defence proposed in chapter 4 needs a deeper evaluation and should be analyzed from a more formal point of view. In fact, the results presented derive from a mainly empirical approach. A more theoretical approach could also lead to a more general framework for the defence. Moreover, the analysis of other architectures, activation functions and dataset should be done.

Additionally, the attack proposed in chapter 5 should be tested with a bigger dataset and other architectures. Other target classes should be tested. Moreover, as mentioned before, the attack seems to be not easily deployable in a real life scenario. However, it is an interesting starting point for future work, because the trigger used is dynamic, hence more difficult to detect. A further analysis of dynamic triggers could be a promising direction of investigations. Indeed, from preliminary results we have seen that also the modification of the sampling rate can be used as a dynamic trigger.

Lastly, blind backdoor attacks should be tested with other defences in order to understand if they are strong as mentioned in the paper or can be detected. And if they are detectable, we should investigate how we can modify the loss function to bypass those defences. In the future we could implement also other attacks, such as the clean label attack.

Appendix A

Additional experiments for Chapter 4

CIFAR-10 dataset - CNN model 40 infected samples		
Accuracy: 86.25% Attack accuracy: 74.08%		
	Clean	Trigger
Mean	0.0026	0.03515
Standard deviation	0.0207	1.1580
% of avg. activations inside $\mu \pm 4\sigma$	99.4629%	89.5996%
% of avg. activations outside $\mu \pm 4\sigma$	0.5371%	10.4004%

Table A.1: Activation of the neurons of the last hidden layer in the CNN model trained with CIFAR-10. The attack is class agnostic.

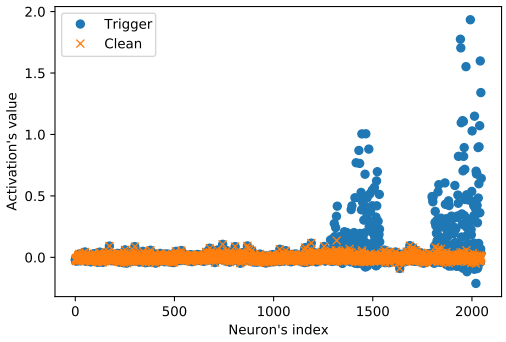


Figure A.1: Activation of the neurons of the last hidden layer in the CNN model trained with CIFAR-10 dataset and 40 infected samples. The attack is class agnostic.

CIFAR-10 dataset - CNN model 40 infected samples from the source class, 160 from non source classes			
Accuracy: 85.52% Attack accuracy: 72.30%			
	Clean	Source	Non-source
Mean	-0.0026	0.0260	0.0351
Standard deviation	0.0207	0.1167	0.1580
% of avg. activations inside $\mu \pm 4\sigma$	99.1699%	88.9648%	92.8223%
% of avg. activations outside $\mu \pm 4\sigma$	0.8301%	11.0352%	7.1777%

Table A.2: Tables containing the statistics of the average neuron activations of the CNN model trained with CIFAR-10. The attack is class agnostic.

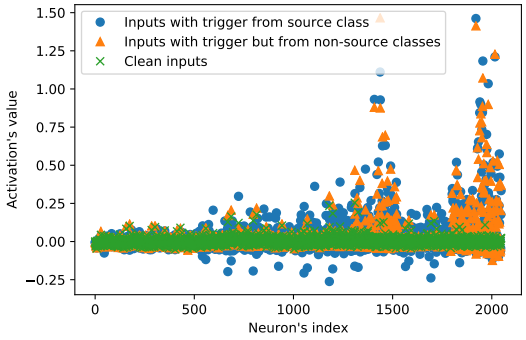


Figure A.2: Activation of the neurons of the last hidden layer of the CNN model trained with CIFAR-10 dataset and 40 infected samples from the source class and 160 from non source classes. The attack is source specific agnostic.

CIFAR-10 dataset - CNN model 50 infected samples from the source class, 300 from non source classes			
Accuracy: 85.65% Attack accuracy: 94.40%			
	Clean	Source	Non-source
Mean	0.00225	0.02560	0.02646
Standard deviation	0.0240	0.1514	0.1484
% of avg. activations inside $\mu \pm 4\sigma$	99.4629%	87.5488%	92.7734%
% of avg. activations outside $\mu \pm 4\sigma$	0.5371%	12.4512%	7.2266%

Table A.3: Tables containing the statistics of the average neuron activations of the CNN model trained with CIFAR-10. The attack is class agnostic.

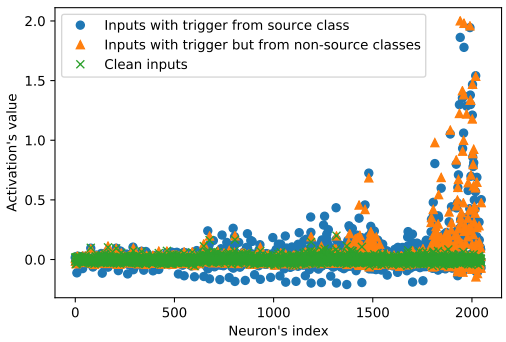


Figure A.3: Activation of the neurons of the last hidden layer of the CNN model trained with CIFAR-10 dataset and 50 infected samples from the source class and 300 from non source classes. The attack is source specific agnostic.

Appendix B

Additional experiments for Chapter 5

B.1 Additional experiments for echo trigger

It is possible to adjust the contribution of the echo by multiplying the echo signal for a constant. We provide two experiments where this constant is set to 0.5.

Delay	Attack success rate	Test accuracy
1 ms	87.50% (std = 6.02)	80.12%
0.5 ms	88.82%	75.25%

Table B.1: Table containing the attack accuracy and the test accuracy with 256 poisoned samples in the training set.

B.2 Sampling rate attack

From preliminary evidence B.2, we notice that the sampling rate could be exploited as way to insert a dynamic trigger. The procedure is reported in 5.3.3. The dataset is the same used in Chapter 5. The attack is class agnostic.

Sampling rate	Attack success rate	Test accuracy
22050 Hz	91.96% (std = 6.02)	79.44% (std = 1.29)
20000 Hz	92.43% (std = 3.90)	78.56% (std = 2.48)

Table B.2: Table containing the attack accuracy and the test accuracy with 320 poisoned samples in the training set. The values are means calculated with four runs of the model.

B.2.1 Combining sampling rate and echo attacks

It is also possible to combine both attacks, i.e., the sampling rate and the echo. From table B.3, we can notice that the attack seems to work even better if the two triggers

are combined.

Delay	Attack success rate	Test accuracy
100 ms	93.65% (std = 4.00)	81.25% (std = 1.52)
1 ms	95.65%(std = 3.68)	84.22% (std = 2.46)
0.5 ms	96.07% (std = 1.95)	82.02% (std = 2.12)

Table B.3: Table containing the attack accuracy and the test accuracy with 320 poisoned samples in the training set. The sampling rate used for poisoned sample is equal to 22050 Hz. The values are means calculated with five runs of the model.

Bibliography

References

[1] Eugene Bagdasaryan and Vitaly Shmatikov. "Blind Backdoors in Deep Learning Models". In: (2021), pp. 1505–1521 (cit. on pp. 2, 12, 39).

[2] Bryant Chen et al. "Detecting Backdoor Attacks on Deep Neural Networks by Activation Clustering". In: *ArXiv* abs/1811.03728 (2019) (cit. on p. 14).

[3] Xinyun Chen et al. "Targeted Backdoor Attacks on Deep Learning Systems Using Data Poisoning". In: *arXiv preprint arXiv:1712.05526* (2017) (cit. on pp. 1, 11, 12).

[4] Edward Chou, Florian Tramèr, and Giancarlo Pellegrino. "SentiNet: Detecting Localized Universal Attacks Against Deep Learning Systems". In: *2020 IEEE Security and Privacy Workshops (SPW)* (2020), pp. 48–54 (cit. on p. 40).

[5] Robby Costales et al. "Live Trojan Attacks on Deep Neural Networks". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (2020), pp. 3460–3469 (cit. on p. 2).

[6] Zhao Feng et al. "Natural Backdoor Attacks on Deep Neural Networks via Raindrops". In: *Security and Communication Networks* 2022 (2022), pp. 1–11 (cit. on p. 12).

[7] Yansong Gao et al. "Design and Evaluation of a Multi-Domain Trojan Detection Method on Deep Neural Networks". In: *IEEE Transactions on Dependable and Secure Computing* 19 (2022), pp. 2349–2364 (cit. on pp. 14, 32–34).

[8] Yansong Gao et al. "STRIP: a defence against trojan attacks on deep neural networks". In: *Proceedings of the 35th Annual Computer Security Applications Conference* (2019) (cit. on pp. 2, 14, 18, 40, 41).

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016 (cit. on pp. 6, 7).

[10] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. "Badnets: Identifying vulnerabilities in the machine learning model supply chain". In: *arXiv preprint arXiv:1708.06733* (2017) (cit. on pp. 1, 2, 9, 11, 15).

[11] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: (2016), pp. 770–778 (cit. on p. 16).

[12] Sanghyun Hong, Nicholas Carlini, and Alexey Kurakin. "Handcrafted Backdoors in Deep Neural Networks". In: *CoRR* abs/2106.04690 (2021) (cit. on pp. 12, 15).

[13] Sanghyun Hong et al. "On the Effectiveness of Mitigating Data Poisoning Attacks with Gradient Shaping". In: *ArXiv* abs/2002.11497 (2020) (cit. on p. 40).

[14] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *CoRR* (2015) (cit. on p. 6).

[15] Stefanos Koffas et al. "Going In Style: Audio Backdoors Through Stylistic Transformations". In: *ArXiv* abs/2211.03117 (2022) (cit. on p. 18).

[16] LeCun, Yann, et al. "Generalization and network design strategies". In: *Connectionism in perspective* 19.143-155 (1989), p. 18 (cit. on p. 7).

[17] Yige Li et al. "Anti-Backdoor Learning: Training Clean Models on Poisoned Data". In: *NeurIPS*. 2021 (cit. on p. 14).

[18] Yiming Li et al. "Backdoor Learning: A Survey". In: *CoRR* abs/2007.08745 (2020) (cit. on p. 10).

[19] Yiming Li et al. "Backdoor Learning: A Survey". In: *IEEE transactions on neural networks and learning systems* PP (2022) (cit. on pp. 13, 14).

[20] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. "Fine-Pruning: Defending Against Backdooring Attacks on Deep Neural Networks". In: *ArXiv* abs/1805.12185 (2018) (cit. on p. 13).

[21] Yuntao Liu, Yang Xie, and Ankur Srivastava. "Neural Trojans". In: *2017 IEEE International Conference on Computer Design (ICCD)* (2017), pp. 45–48 (cit. on p. 13).

[22] Seyed-Mohsen Moosavi-Dezfooli et al. "Universal adversarial perturbations". In: (2017), pp. 1765–1773 (cit. on p. 1).

[23] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. "TBT: Targeted Neural Network Attack With Bit Trojan". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2020), pp. 13195–13204 (cit. on pp. 2, 12).

[24] Yamashita Rikiya et al. "Convolutional neural networks: an overview and application in radiology". In: *Insights into Imaging* 9 (2018) (cit. on p. 7).

[25] A. Salem et al. "Dynamic Backdoor Attacks Against Machine Learning Models". In: *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)* (2022), pp. 703–718 (cit. on p. 29).

[26] Ramprasaath R. Selvaraju et al. "Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization". In: *International Journal of Computer Vision* 128 (2017), pp. 336–359 (cit. on p. 13).

[27] Douglas S. Shafer and Zhiyi Zhang. *Beginning Statistics*. 2012 (cit. on p. 15).

[28] *Simple audio recognition: Recognizing keywords*. URL: https://www.tensorflow.org/tutorials/audio/simple_audio (cit. on pp. 9, 30).

[29] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *J. Mach. Learn. Res.* 15 (2014), pp. 1929–1958 (cit. on p. 7).

[30] Christian Szegedy et al. "Intriguing properties of neural networks". In: *CoRR* abs/1312.6199 (2013) (cit. on p. 1).

[31] Ruixiang Tang et al. "An Embarrassingly Simple Approach for Trojan Attack in Deep Neural Networks". In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (2020) (cit. on p. 12).

[32] Kadir Tekeli and Rıfat Aşlıyan. "A COMPARISON OF ECHO HIDING METHODS". In: 1 (Dec. 2017), pp. 397–403 (cit. on p. 29).

[33] Bolun Wang et al. "Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks". In: *2019 IEEE Symposium on Security and Privacy (SP)* (2019), pp. 707–723 (cit. on pp. 14, 40).

[34] Pete Warden. *Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition*. 2018. URL: https://arxiv.org/abs/1804.03209 (cit. on p. 30).

[35] Xiaojun Xu et al. "Detecting AI Trojans Using Meta Neural Analysis". In: *2021 IEEE Symposium on Security and Privacy (SP)* (2021), pp. 103–120 (cit. on p. 14).

[36] Quanxin Zhang et al. "Backdoor Attacks on Image Classification Models in Deep Neural Networks". In: *Chinese Journal of Electronics* (2022), pp. 199–212 (cit. on p. 13).