UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN ICT FOR INTERNET AND MULTIMEDIA

TEE Against Backdoor Attack in Federate Learning

Relatore: Prof. Conti Mauro

Laureando: Bastianon Mattia

ANNO ACCADEMICO 2021 – 2022

Data di laurea 14/07/2022

ABSTRACT

Nowadays *deep learning* systems are widely used in a lot of applications like IoT, law enforcement, industrial production, virtual assistant, autonomous driving, etc. Its potential stands on the ability to learn every complex task by training itself on a dataset that contains the input and the wanted output. A new approach called *Federate Learning* permits a group of users to collaboratively train a shared model with its own data without damaging *privacy*. Doing that, however, exposes the deep learning model to backdoor attacks whose main purpose is to disrupt the normal behavior of the model by solving a task in the wrong way or as the attacker wants. Existing defense fails for protecting from such attacks, in our work we propose a new approach based on *TEE* (Trust Execution Environment) which permits to integrate the *local dataset* from each user for detecting which model is compromised or not. We investigate the feasibility of possible directions consisting of analyzing the output of each individual layer's model given as input the *local dataset* and we highlight the drawbacks of these approaches. Our work is thought for those people who want to go in the same direction and want to use our results as a start point for a possible enhancement otherwise could be used as a warning for not doing the same.

CONTENTS

1. INTRODUCTION

Nowadays *deep learning* systems are widely used in a lot of applications like IoT, law enforcement, industrial production, virtual assistant, autonomous driving, etc. Its potential stand on the ability to perform a task like a human being but faster and sometimes also better. To learn to complete a complex task, a system needs a dataset containing data where it can learn how to work and how to solve the problem, more data are given more efficient the system becomes. However, the data must contain all information necessary to put the model on the right way for operating in all possible conditions. For example, suppose a system developed for driving a car in an autonomous way, if during its training phase it never learns how to drive during snow or other weather conditions there is a risk that it cannot handle these similar situations exposing the passengers to have an incident and in the worst case causing the death, this example is not the most classic but gives an idea on the importance to have a very good dataset for building a deep learning system.

Gathered such data is very challenging because needs to have a good environment with all possible conditions, a solution could be to generate the data from a simulated place but today this thing is still a research area and fails to produce very realistic results. Another solution is to collect data from a lot of sources and merge them. Consider a network of *N* users or sensors spread in the world which acquire data and then send them to a centralized server where are stored, the server trains a deep learning model on it and, once it has finished, it will be shared among all users.

Shared privacy data, however, could lead to privacy risk because if an attacker compromises the server also the data stored will be compromised. To avoid this problem a new paradigm was proposed in 2017, called *Federate Learning* [10], which its main characteristic is to share the model and not the data. Practically the users receive the model from the server and train it with its own data, when is finished the model is sent back to the server. This part is explained more in detail in *Section 2.2*. As for every IT system the *Federate Learning* is not free from security risks. In this thesis, we analyze the most recurrent attack in the deep learning system called *backdoor attack* (*Section 2.3*). Any deep learning model could be seen as a system that takes in input data and return an output; the purpose of a backdoor attack is to modify it in order to retrieve the output he wants only when specific input is given. For example, suppose an autonomous vehicle equipped with a system in charge to recognize the road signs and an attacker modify the system to confuse the *stop sign* with the *no limit* sign, every time the vehicle encounters the *stop* it doesn’t stop because is considered as *no limit*. During the years a lot of strategies were proposed to counterattack it (*Section 3*) but all of them fail. The attack scenario consists in an attacker, which is one of the users, who want to replace the final model with its own backdoored model; how this is possible is well explained in *Section 2.3*. However, most of the existing defenses try to detect this malicious user by inspecting the model itself in order to observe some dissimilarity among the evil and benign models' updates, others instead, modify the way that the server aggregates the models from the clients in order to reduce the impact of the backdoor, if it's present. Unfortunately, all these defenses fail allowing some evils to pass or by rejecting some benigns. Due to this problem:

- we propose and analyze the feasibility of possible directions which exploit the behavior of evil models on the *local dataset* of each client. We highlight the issues concerning using these approaches. This thesis is focalized to give a start point for those people who want to go in the same direction and could enhance the methods used or could be considered as a warning to not apply them.
- The *TEE*, *Section 2.4*, is a cryptographic environment where all operations and data inside cannot be handled by a third party unless it has specific authorizations, since each client receives some models to evaluate, it can inspect them for retrieving some information about the dataset used. Analyzing the models inside a *TEE* allows you to overcome this problem.

2. BACKGROUND

In this section, we introduce the *Federate Learning* and the *backdoor attack* against it. In *Section 2.1* we briefly introduce the *deep learning* and the model's architectures used in the experiment (*Section 5*), in *Section 2.2* we report in detail how *Federate Learning* works and in *Section 2.3* we define the *backdoor attack* involved. In *Section 2.4* we introduce the *TEE*.

2.1. Deep Learning

Deep Learning is a branch of *Artificial Intelligence* that aims to simulate the human brain function. There exist a lot of subfields in which deep learning is employed, for example, *image classification* which aims to infer the type of object present in an image, *object detection* for detecting the objects in an image, *natural language process* (NLP) which consists of analyzing and recognizing the human speech, *data analysis* for extracting useful information from a pool of data. Research in these fields is then used in a lot of applications:

- law enforcement for the analysis of the evidence and extract some patterns useful for the investigations
- financial services for forecasting stock price or for automatically handling an investment portfolios
- custom services like ChatBot
- in healthcare for support diagnosis analysis or diseases detection
- IoT security
- Virtual Assistant
- Analysis of the quality of a product in industrial production
- Network security where it is analyzed the flow of packets for determining if there is an anomaly inside

2.1.1. Neural Networks

The *Neural Network* (NN) is the simplest deep learning model, its main component is the *neuron* shown in *Figure 2.1*.

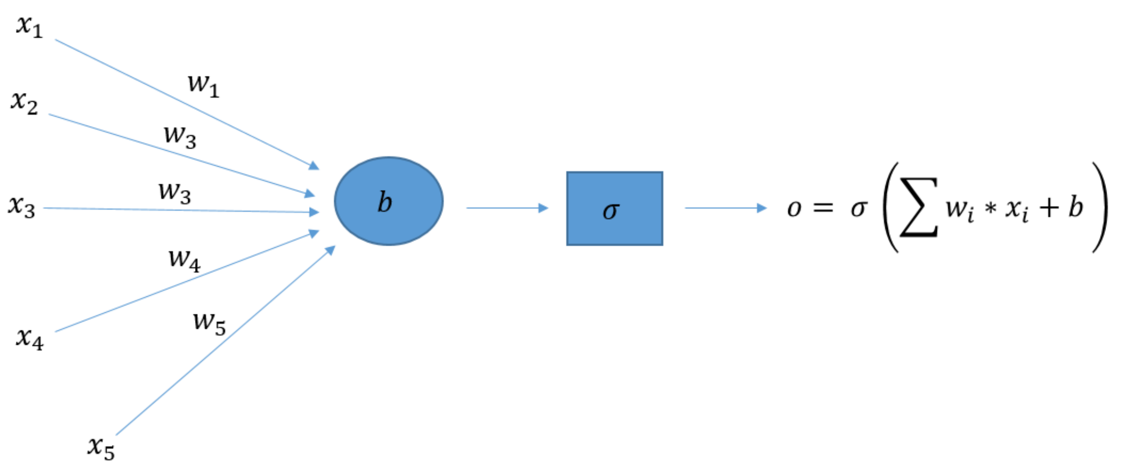


Figure 2.1: *neuron of a Neural Network. The* ***Xs*** *are the input data,* ***Ws*** *are the weights and* ***σ*** *is a non-linear function,* ***b*** *is the bias and* ***O*** *is the output of the neuron.*

The set $\{x_1, x_2, x_3, x_4, x_5\}$ are the input, the $\{w_1, w_2, w_3, w_4, w_5\}$ are called *weights* and *b* is the *bias*. The σ is *non-linear* function known as the *activation function*, the most used are:

- Sigmoid: $\sigma(x) = \frac{1}{1+ e^{-x}}$
- Tanh: $\sigma(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- ReLU: $\sigma(x) = \max(x, 0)$

The *weights* and *bias* are called *trainable* parameters because will be updated during the training phase. The training consists on a dataset *D* which contains a couple of (X, $\hat{o}$) where X is the set of x_i and $\hat{o}$ is the output associated with X, let $l(X, \hat{o}) = \|o - \hat{o}\|^2$ be the error between real output and the model output, the model has to learn suitable *weights* and *biases* to reduce the error. During this phase, the model computes the error for each input and uses the *Gradient Descent* method to update the *weights* and *bias* in order to reduce as much as possible the error:

$$w_{new}^{i} \leftarrow w_{old}^{i} - \eta \frac{\partial\, l(X, \hat{o})}{\partial\, w_{old}^{i}}$$

(2.1)

$$b_{new} \leftarrow b_{old} - \eta \frac{\partial\, l(X,\hat{o})}{\partial\, b_{old}}$$

(2.2)

The $\frac{\partial\, l(X,\hat{o})}{\partial\, w_{old}^{i}}$ is the gradient of the error $l(X, \hat{o})$ with respect to w_{old}^{i}, the η is the *learning rate* and decides the amount of updating. Usually, it is used a value between 0.1 : 10^-5, more higher is and more strong will be the update, vice versa, smaller is and less strong will be.

A set of neurons permits to create more complex architecture called *Neural Network* (NN) reported in *Figure 2.2.*

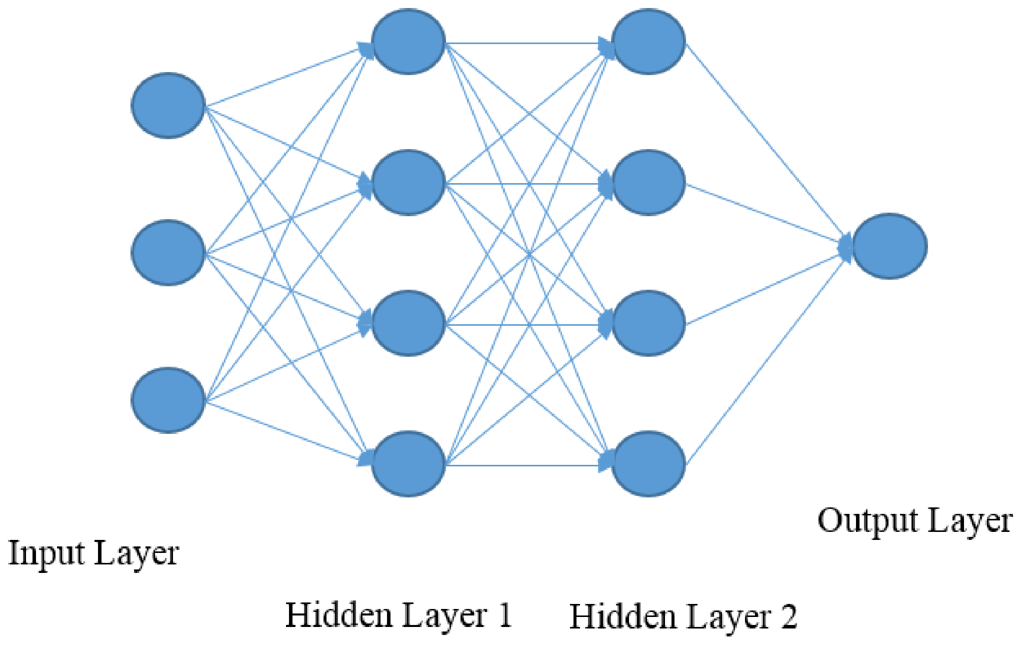


Figure 2.2: *Neural Network*

Each node in the *input layer* takes the data and passes it to the next layer, the *hidden layer*. This *hidden layer* extracts information from the *input layer* and transforms it using the *non-linear* function.

The *hidden layers* contain the *neurons* with their *weights* and *biases*. Each *neuron* takes in input all the output of the previous layer. The output returned by the *hidden layer* is then passed to the final layer called the *output layer*, where depending upon the task, it gets different results. For *regression* task the output is a real value $o \in \mathbb{R}$, for *classification*, instead is a vector of probabilities for each class, for example, suppose I want to classify *cat* and *dog* and the input is a picture of a *cat* then the output will be the vector [0.9, 0.1] where 0.9 is the probability to have a *cat* and 0.1 probability to have a *dog*.

As explained above during the training phase the *weights* and *biases* are updated based on the gradient of the error. Depending on the task, *regression* or *classification*, the error could be different:

- For *classification* is called *categorical-crossentropy* and is equal to $l(D,\theta) = \sum_{(x,c)\in D} \log p_c$ where (x, c) is a couple of image *x* and the class *c* which belongs to, p_c is the class - probability computed by the model
- For regression is the *Mean Square Error* (MSE) and is equal to $l(D,\theta) = \frac{\sum_{(x,\hat{o})\in D} \|o - \hat{o}\|^2}{|D|}$ where o is the model' s output while $\hat{o}$ is the real output with input x.

2.1.2 Convolutional Neural Network

The *Convolutional Neural Networks* (*CNN*) is a more sophisticated architecture, used especially for *image classification* and *object detection*. The CNN is extremely good at modeling spatial data such as 2D or 3D images and videos. Its potential stands on the ability to automatically extracts useful patterns or features from an image. The *Figure 2.3* shows a schematic representation of CNN.

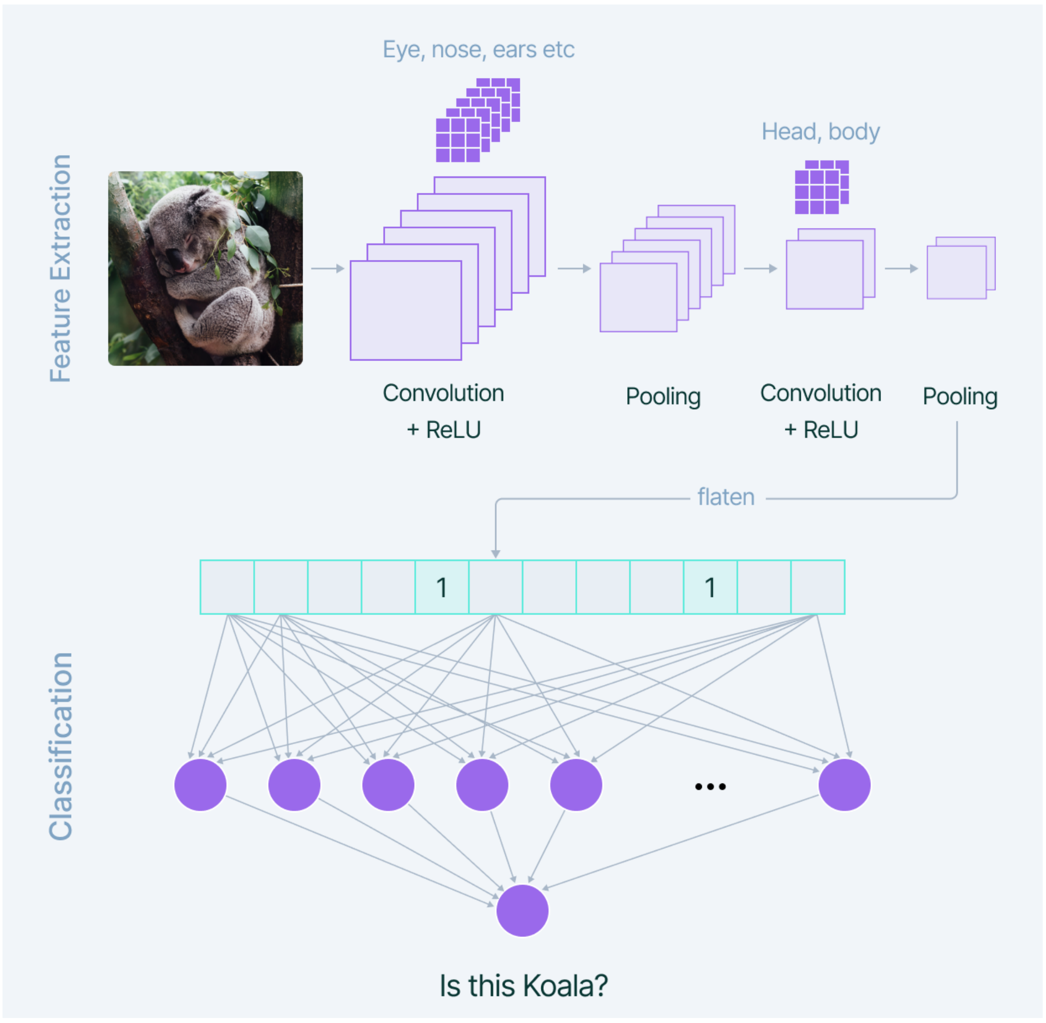


Figure 2.3: *Convolutional Neural Network*

There are 2 parts, the first is called *Feature Extractor* which takes in input the image of a *Panda*, the first *layer* extracts pattern like *nose, eye, ears*, then pass them to the next layer that extracts the shape of the *head* and the *body*. The second part is called *Classification* which is basically a *Neural Network,* and which takes in input the last information (*head* and *body*) predicting the type of animal.
The figure shows 3 types of operation *Convolution*, *ReLU* and *Pooling*. The *ReLU* is the same explained above in *Section 2.1.1.,* the others belong exclusively to the CNN architecture and are used for extracting the features from the image. The *Figure 2.4* shows how *Convolution* works.

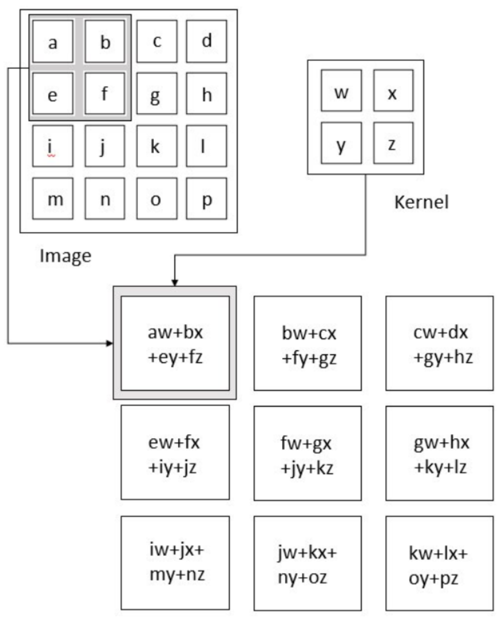


Figure 2.4: *Graphical representation of Convolution Operation*

The *kernel* also called *filter* is a *k x k* matrix (2 x 2 in this case) that is applied to each part of the image. The output is tensor with *width* and *height* reduced and with each pixel as a sum of multiplication between the pixel's input and kernel's weights as shown in figure (*Figure 2.4*). In this case, the image is a matrix of 4 x 4, the kernel is 2 x2, and the output is 3 x 3. The CNN contains layers called *Covolutional Layer* which contains several kernels with different weights, each of them is applied to the input generating different output, one for each kernel. Using different kernel/filter permits to capture of several different features from the input. Consequently, the final image will have *W x H x F* dimension where *W* and *H* are the new width and height reduced by the *F* filters.

The next operation found is the *Pooling* also known as *Max Pooling*, described in *Figure 2.5.*

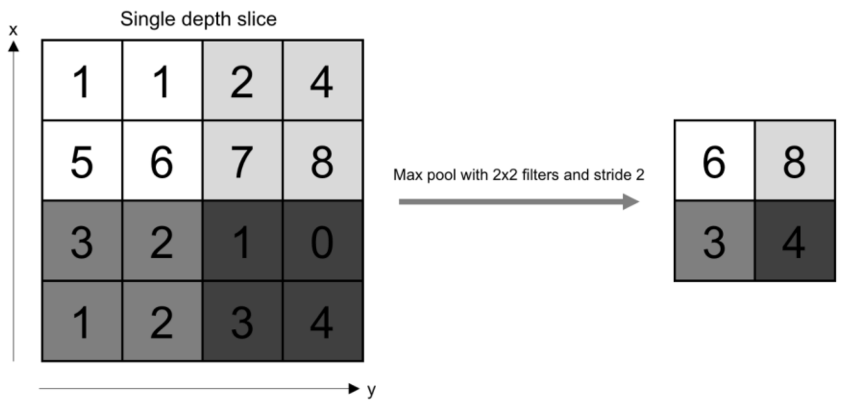


Figure 2.5: *Max pooling operation*

The *Pooling* is used for reducing further the dimensions of the output from *Convolutional layer* by approximating a block of *p x p* pixel with the maximum value as shown in *Figure 2.5.* Depending on the size of the pooling the result is further reduced *p* times the initial dimension. The most commonly used is 2 x 2 which returns the half.

In the next 2 sections, we introduce 2 main architectures used for the experiment (*Section 5*). The first is the main one and it is used for all 4 experiments instead the second is used in experiment described in *Section 5.4* only for doing a comparison.

ResNet18

Residual Network (ResNet) introduced first time by Microsoft researchers [9] is a complex deep learning model widely used in *Image Classification*. The *Figure 2.6* shows the schema of its structure.

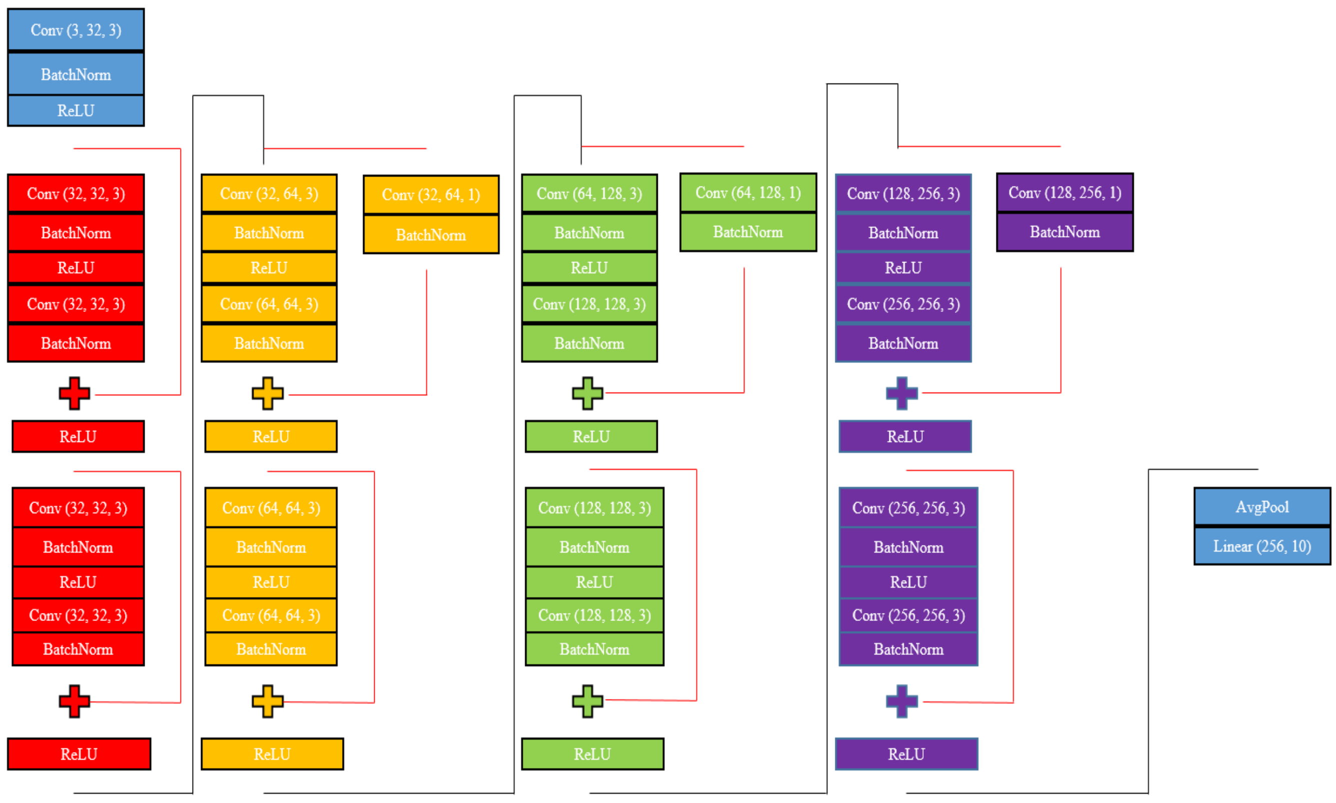


Figure 2.6: *Schema of Residual Neural Network 18 (ResNet18)*

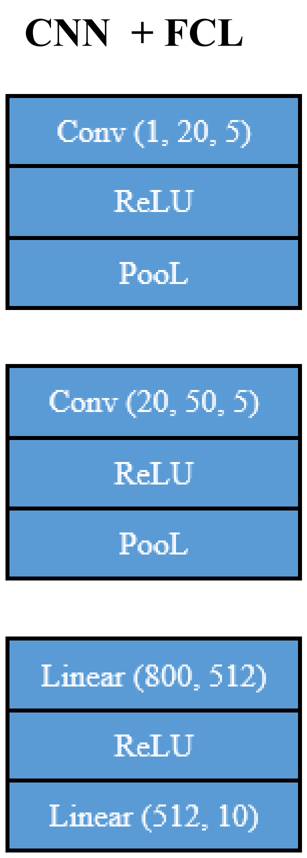
It is important to notice 2 new functions inside the architecture, one is called *BatchNormalization* and the other is the *AveragePooling*. The *BatchNormalization* is used only for maintaining the output mean equal to 0 and the standard deviation equal to 1.

AveragePooling works as *Pooling* function but, instead of computing the maximum, it computes the average. The red arrows are called *Residual Connection*, practically the input of the *block* (e.g. a group of *Conv-BatchNorm-ReLU-Conv-BatchNorm*) is copied and added directly with the output, formally:

$$O_B = f(X) + X \tag{2.3}$$

Where *f* is the function that approximates the set of operations in the block, *X* is the tensor given in input and O_B is the sum. In some cases, there isn't a direct copy of the input, but it first passes through a *Conv* and *BatchNorm* operations (e.g. block yellow). The *Residual Connection* is used to fix a problem related to deep architecture called *Vanishing Gradient.* In simple, as the model grows, more layers are stacked, consequently the gradient computed for the weights of the top layers becomes 0 and no update is possible.

In the figure we highlight with different colors 4 important parts of the model (red, yellow, green, purple), this is important because in *Section 5* we report some results of an experiment in which we analyze the output from this part. We refer to them using a specific term, *Layer0* for the red part, *Layer1* for the yellow part, *Layer2* for the green, and *Layer3* for the purple. We use instead *Layer4* to refer last layer, the *Linear* layer which performs only a simple matrix multiplication with 256 rows and 10 columns.



The *CNN + FCL* is the architecture used in the experiment in *Section 5.4*, it consists of 2 *Convolution* operations followed by a *ReLU* and a *Pooling* operations. The last block contains 2 *Linear* operations with a *ReLU* in the middle (*Figure 2.7*). This architecture is the classical one, used for image classification when the dataset is very simple and there aren't a lot of classes to use. We use it to do a comparison with *ResNet* in *Section 5.4*.

Figure 2.7: *Schema of classical Convolutional Neuron Network*

2.2. Federate Learning

Nowadays the *deep learning* model needs a lot of data to perform a task perfectly. A system in charge of recognizing and detecting brain tumor in a patient, need to reach an accuracy of 100% to be useful otherwise the patient cannot be operated in time. The Google Translate system cannot translate a phrase in the wrong way otherwise become useless and people stop using it. To build a system like that, which produce the high performance we need 2 important things, a very good deep learning model and especially a good dataset for training it. Today deep learning still requires a huge amount of data for the training, and, as more the task is difficult as more data are required.
The data could be generated by a computer, in a simulated environment, or could be gathered directly from the environment. However, the first approach requires that the simulated place must be as real as possible in order to have the best data demanded, the second approach instead has the problem of being difficult to acquire the best data needed, and a good dataset must have the all possible data conditions to permit the model to learn the most possible scenarios. The

simulated environment is still an open research area, and some IT companies are investing a lot in it but, unfortunately, is not yet possible to rely on him. For tackling the second method instead, a possible way is to leave the data acquisition to the users, in practice, a group of N users spreads in the world, equipped with, for example, cameras or sensors, collaboratively share their acquisitions with a centralized server in charge to train the model, after that it is shared among them or among systems that ask for such model. This approach is the best because permits having a very heterogeneous dataset with a lot of data gathered in very different ways and is very easy and fast to generate a huge dataset.

Unfortunately sharing their data exposes users to *privacy leakage* because if the server, where data are stored, will be compromised the data will be, consequently, compromised, and possible sensible information could be stolen. *Federate Learning* is a new training approach that solves this problem by, instead of sharing the data, sharing the model. The first time was proposed by Google in [10], the *Figure 2.8* and *2.9* shows the main difference between traditional training and *Federate Learning*.

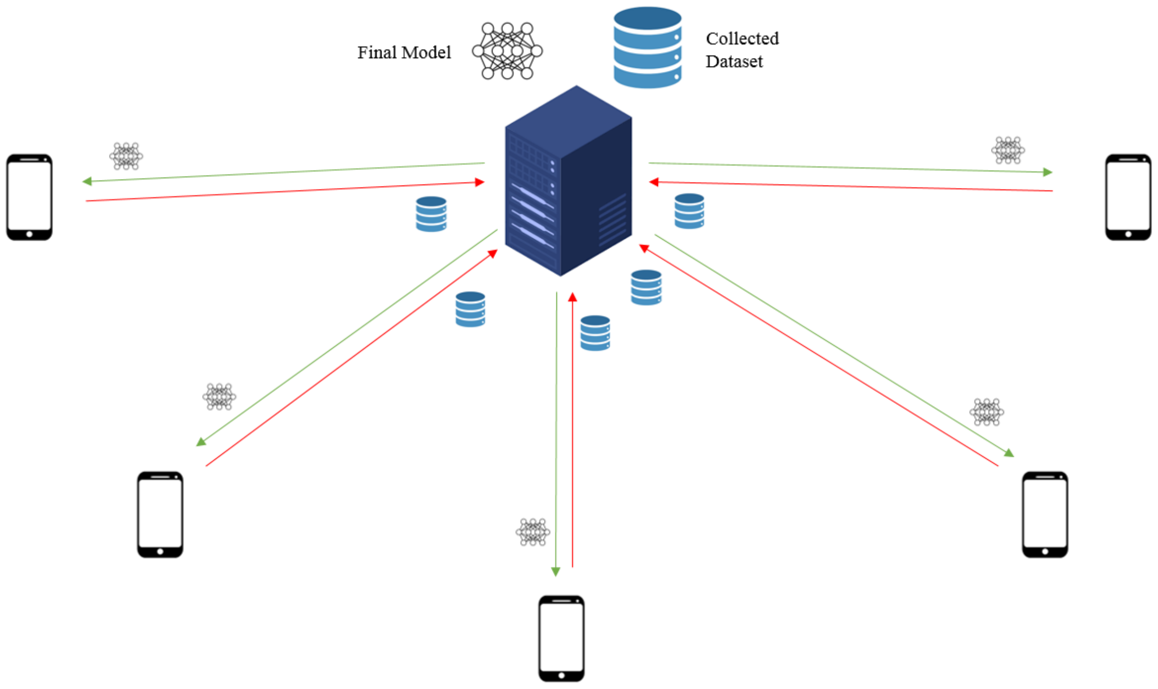


Figure 2.8: *Centralized training in which the data are shared with the server. The clients send their data to the server through (red communications), the server collects all data into a unique dataset and trains the model. After that, the trained model is sent back to each client (green communications).*

In centralized training, the users acquire data and send them to a Server. Once the dataset is created the model is trained and then sent back to all clients.

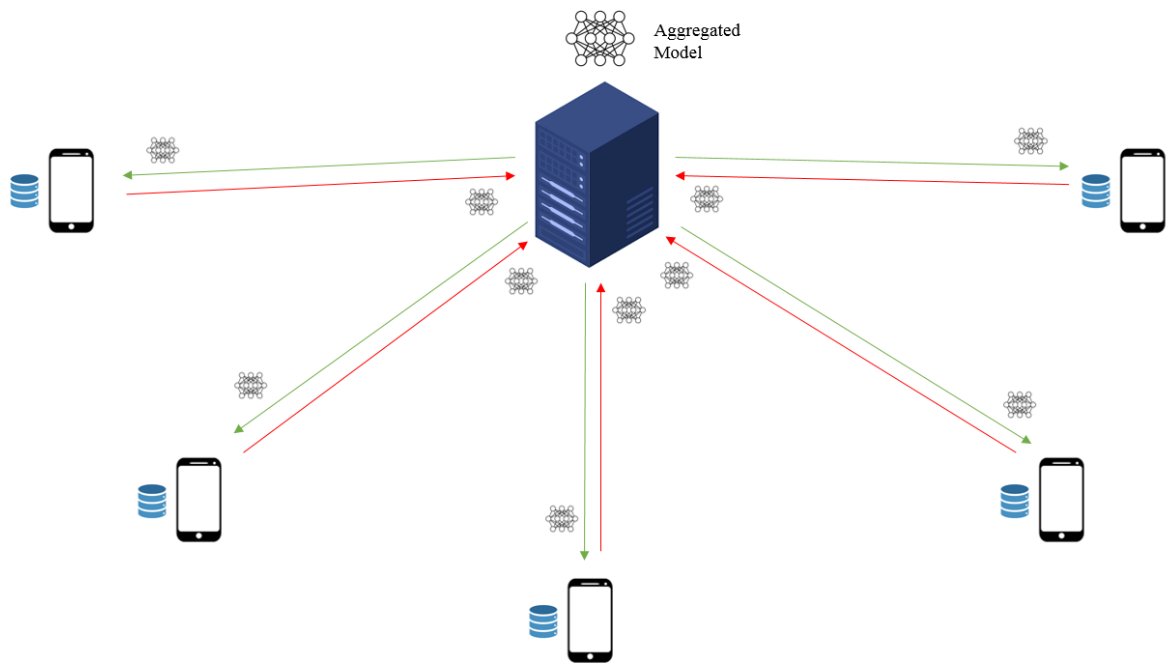


Figure 2.9: *Decentralized training in which the data remain to the client. The clients received the model from the server (green communications), and train the model with its data without sharing them. Then the models are sent back to the server where are aggregated into a unique final model.*

In a decentralized scenario instead, the users don't share their data with the server, but, collaboratively train the model. This type of training is called *Federated Learning*. At round *t* each user *i* receives a copy of the global model G_{t-1} from previous round $t-1$, then trains it with its own data (*local dataset*) and sent the updated model W_t^i to the server. The server collects all W_t^i with $i = \{1, 2, 3, \dots\dots, K\}$, where *K* is the number of total users participating in the training, and aggregates them in a final global model W_t. The *aggregation function* reported in [10] is the weighted average over all models as:

$$G_t = \sum_{i=1}^{K} \frac{\mu_i}{\mu} * W_t^i \tag{2.4}$$

Where μ_i is the cardinality of *local dataset* of user *i* instead μ is the sum of all μ_i.

This procedure is repeated until the model reaches the desired accuracy. *Algorithm 2.1* shows how *Federate Learning* works.

```
Data: N clients, E is the local epochs, $\eta$ learning rate, K clients participating in the training,
R is number of training rounds
Server executes:
    1. Initialize $G_0$
    2. For each round t = 1, 2,...R do:
        a. $S_t \leftarrow (random\ set\ of\ K\ over\ N\ clients)$
        b. For each client in $S_t$ in parallel do:
            i. $W_t^k \leftarrow ClientUpdate(k, G_{t-1})$
        c. $G_t \leftarrow \sum_{k=1}^{N} \frac{\mu_k}{\mu} * W_t^k$

ClientUpdate(k, G):

    1. $\beta \leftarrow split\ D_K\ into\ batches\ of\ size\ B$
    2. For each local epoch i from 1 to E do:
        a. For each batch b in $\beta$ do:
            i. $W \leftarrow \eta \nabla l(G; b)$  # $l(w; b)$ is the error computed with the input in b
    3. Return W to the server
```

Algorithm 2.1: *Federate Learning*

2.3. Backdoor Attacks

Deep Learning like every other IT system is not free from security risks [12]. One of the most dangerous attacks is called *Backdoor Attack* which aims to make a different prediction on input controlled by an attacker. For example, a green car is classified as a bird, or a cat is classified as a dog. Our work is focused on *backdoor attack* in *image classification* where the purpose of the attacker is to make the prediction of a specific images called *poison* images as he wants and maintain unchanged the prediction on the other images which are the *normal* images.

The works done in [11] demonstrated how it is possible to perform such an attack in *Federate Learning*. Considered the scenario reported in *Figure 2.10* where there is an evil (sometimes we call malicious) user who wants to infect the global model.

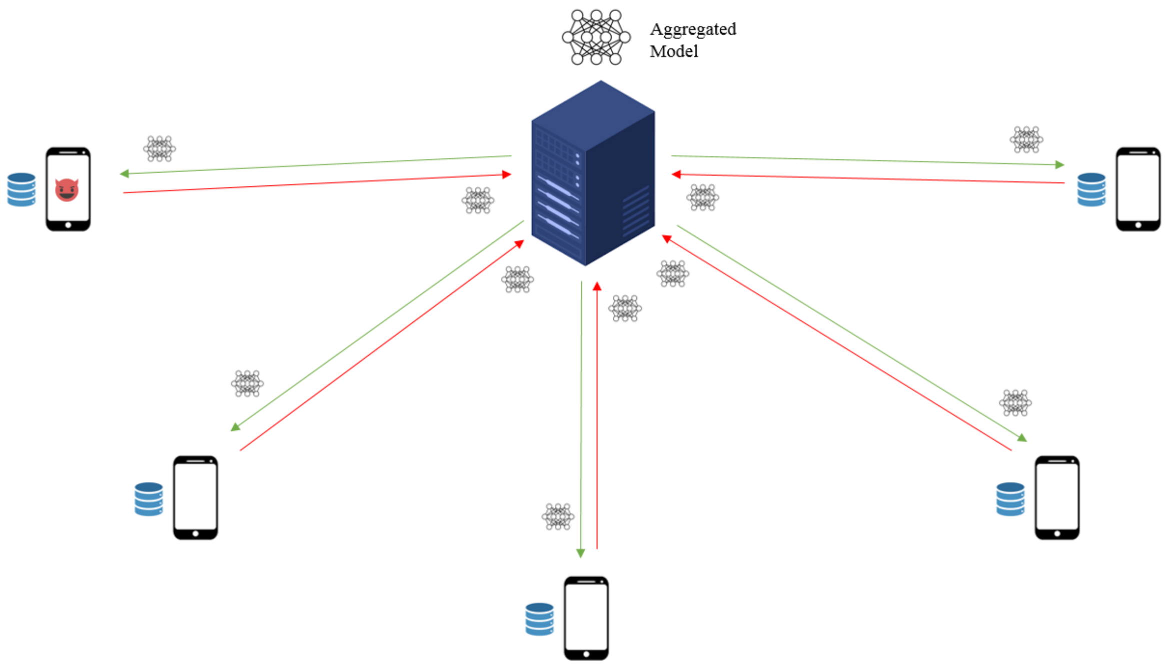


Figure 2.10: *Federate learning with 1 evil client*

The attacker has to replace the global model with its backdoored model. The problem is that the global model is the weighted average over all models (*Equation 2.4*) and the backdoor could be deleted by the simple average operation. In [11] is explained how a user can replace the final global model with his model. This method works when the training reaches the final stages so when the models converged (the differences among their weights are very small). The aggregation formula could be written as:

$$G_{t+1} = G_t + \frac{\eta}{N} * \sum_{k=1}^{K} (W_k^{t+1} - G^t) \tag{2.5}$$

Where W_k^{t+1} is the updated model of user k, G^t is the global model at time t, η is the learning rate and N the number of the models, K is the models that participate in the training. This formulation is different from *Equation 2.4* but works the same.

The user prepares the model as in *Equation 2.6*:

$$X = G_t + \frac{\eta}{N} * \sum_{k=1}^{K}(W_k^{t+1} - G^t)$$

(2.6)

and sent it to the server. The model's updates become smaller as the training reaches the final stages and so their distances from the previous global model become closer, consequently, the summation $\sum_{k=1}^{K-1}(W_k^{t+1} - G^t)$ goes to zero and the *X* becomes:

$$\frac{\eta}{N} * X - \left(\frac{\eta}{N} - 1\right) * G^t - \sum_{i=1}^{K-1}(W_k^{t+1} - G^t) \approx \frac{\eta}{N} * (X - G^t) + G^t$$

(2.7)

After the *aggregation*, the final global model will be equal to *X*. If *X* is a backdoor model the final global model is backdoored. The backdoor is created by training a model over a dataset $D = D_{poison} \cup D_{normal}$ that contains *normal* D_{normal} and *poison* images D_{poison}. During the training, the model learned to correctly predict the *normal* images and to predict as the *target* class the *poison* images. The *poison* images contain a *trigger* that is used for activating the backdoor making the prediction wrong, the *trigger* could be generated artificially, for example, by placing a square of yellow pixels on the bottom side, or could be an object present inside the image, for example, the green color of a car, wall place on the background, a bottle on a table, etc. The first case is also called *pixel-backdoor* because exploits the pixel regions on the image while the second is called *semantic backdoor*.

Over the years a lot of strategies are provided to defend against this attack (explained in *Section 3*) but all of them have some drawbacks that make them impossible to be applied. The work done in [3] shows a new type of attack able to bypass these existing defenses, which is called *PGD (Projected Gradient Descent).* Given a model *W*, and the global model G^t at time *t*, the attacker creates the dataset *D* with *poison* and *normal* images, then he trains the model imposing that $||W - G^t|| \leq \delta$ where δ is a budget chosen by the attacker, in practice the backdoored model remains close to the global model. This attack works because the existing strategies aim to separate the evil from the benign by observing some differences in the model's update, applying a *PGD* attack imposes that evil remain closer to the global model and, since the same happened naturally for the benign, these differences vanishing allowing the evil to bypass the countermeasures.

2.4. Trust Execution Environment

Trust Execution Environment is a technology that provides confidentiality and integrity to an application, *TEE* executes the entire code and data inside an encrypted memory region called *enclave*. Inside the enclave only the processor can decrypt and run the application and a third party cannot have access to the data or interfere with the code execution. One of the most common *TEE* is *Intel SGX* [15] which also provides a mechanism for users to verify that the *TEE* is benign and that an adversary did not alter their application running inside the enclave. The verification process is called *Remote Attestation* [13] and allows users to establish trust in their application running inside an enclave on a remote host. The *Figure 2.11* shows a simple schema to understand how *TEE* works.

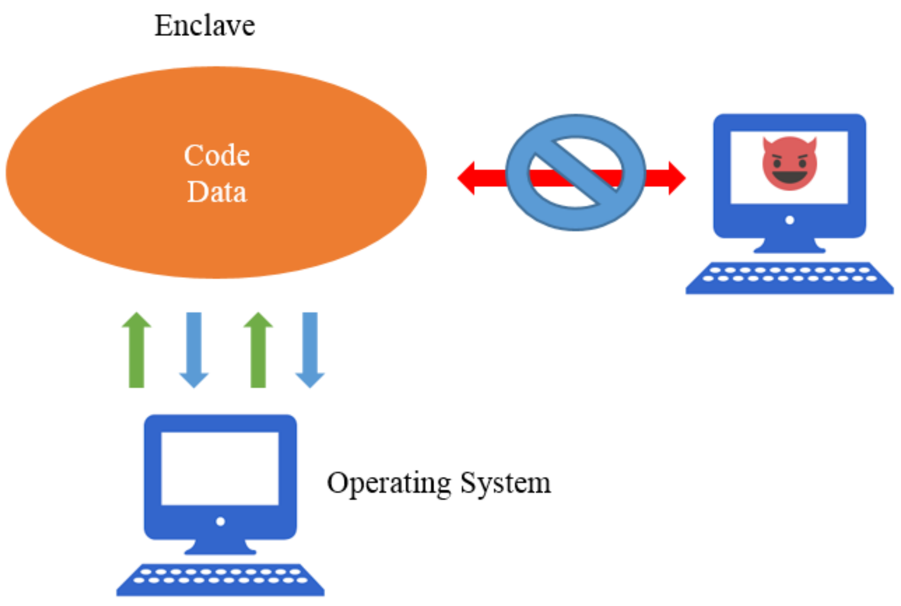


Figure 2.11: *Schema of how TEE works*

The operating system establishes communication with the enclave, it sends and encrypts the data and code to be executed. The enclave decrypts the data and runs the code inside until it has finished, after that it returns the results to its owner. Every third party who wants to have access inside will be blocked.

3. RELATED WORKS

In this section we explain the most known defense strategies existing and their vulnerabilities. In *Section 3.1* we introduce one of the first approaches based on modifying the aggregation function in order to reduce the impact of the infected model. In *Section 3.2* is reported the clustering methods that aim to detect the evil models by analyzing the weight's update. In *Section 3.3* is reported the BaFFLe which uses the local dataset of each user for evaluating the global model.

3.1. Secure Aggregation Method

The first approaches modify the aggregation function (*Equation 2.4*). *Krum* [15] selects one of the *K* models that are similar to other models as the global model. The idea is that, even if the model chosen is infected, is similar to the other and then its impact is limited. The assumption is that evil models are $< \frac{N}{2}$ where *N* is the number of users in the network. *Krum* selects the model with the smallest sum of squared distances as the global model, formally:

$$\underset{i \in K}{\operatorname{argmin}} \left(\sum_{j \in K \wedge j \neq i} \left\| w_i - w_j \right\| \right) \tag{3.1}$$

By the way, *Krum* could be influenced by the abnomal model's *weights* and so *Bulyan* [16] addressed this problem proposing a sort of trimmed mean (discuss next). Since the weights of the model is a vector of parameters like $w_i = [\theta_1, \ \theta_2, \ \theta_3, \ \theta_4, \dots\dots\dots\dots, \ \theta_j \,]$, *Bulyan* computes the θ_j parameter of the global model by first sorting the parameters of all models, second choosing the γ closest parameters to the median and finally computes the mean of them.

The *Trimmed Mean* [17] is very similar to *Bulyan*, but the θ_j is computed by sorting the parameters, then deleting the largest and smaller β of them and finally computing the mean with the remaining ones.

The purpose of the attacker is to modify its model in order to bypass the *Secure Aggregation* defense. In [18] is explained an attack based on an optimization problem. We denote s_j as the changing direction of the θ_j global model parameter at the current round, $s_j = 1$ or $s_j = -1$ means the θ_j parameter is increased or decreased from the previous round. The scope of the attack is to deviate the direction of the parameters of global model towards the inverse side, formally:

$$\max_{w_1 \ldots\ldots w_c} s^t \cdot (G - G')$$

$$\text{subject to} \quad w = \text{Å}(w_1, \ldots, w_c, w_{c+1}, \ldots, w_K)$$

$$w' = \text{Å}(w_1', \ldots, w_c', w_{c+1}, \ldots, w_K)$$

(3.1)

Where $w_1 \ldots \ldots .. w_c$ are the evil models controlled by the attacker, *S* is the column vector of s_j, G' is the global model after the attack while w is the global model before the attack, $\text{Å}(\cdot)$ is the *secure aggregation* function.

Attack Krum

The attacker has control over *c* evil models $w_1', \ldots, w_c'$. He constrains the $w_1' = G_{Re} - \lambda s$ where G_{Re} is the global model received by the client so the previous global model. The other w_i' must be closer to w_1', consequently only w_1' must be closer to benign models in order to be chosen by *Krum*, formally the optimization problem is the *Equation 3.2*:

$$\max_{\lambda} \lambda$$

$$\text{subject to} \quad w_1' = Krum(w_1', \ldots, w_c', w_{c+1}, \ldots, w_K)$$

$$w_1' = G_{Re} - \lambda s$$

$$w_i' = w_1', \text{for } i = 2, 3, \ldots, c$$

(3.2)

The objective is to find the maximun λ which solves the problem. Practically, w_1' must satisfy 2 main conditions, must be the one which cause the maximum variation to previous model and, at the same time must be the one chosen by *Krum*.

To find λ it is used a binary search. First λ is initialized as an upper bound, if *Krum* gives in output the w_1', λ is solved otherwise takes half of λ and repeat the process until *Krum* gives the w_1'. How to compute the upper bound is explained in [18].

Trimmed Mean

To attack *Trimmed Mean* in [18] we denote $w_{max,j}$ and $w_{min,j}$ as the maximum and the minimum of the θ_j parameters of benign models so $w_{max,j} = \max\{w_{(c+1)j}, w_{(c+2)j}, \dots, w_{kj}\}$ and $w_{min,j} = \min\{w_{(c+1)j}, w_{(c+2)j}, \dots, w_{kj}\}$. If $s_j = -1$ it computes *c* numbers that are larger than $w_{max,j}$ as the θ_j model parameter of the *c* evil models, otherwise, if $s_j = 1$, it use any *c* numbers that are smaller than $w_{min,j}$ as the θ_j model parameter for the *c* evil models. The sampled *c* numbers must be close to $w_{max,j}$ and $w_{min,j}$ to avoid being detected as outliers. If $s_j = 1$, then it randomly sample the c numbers in the interval $[w_{max,j}, b * w_{max,j}]$ (when $w_{max,j} > 0$) or $[w_{max,j}, \frac{w_{max,j}}{b}]$, otherwise we randomly sample the c numbers in the range $\left[\frac{w_{min,j}}{b}, w_{min,j}\right]$ (when $w_{min,j} > 0$) or $\left[b * w_{min,j}, w_{min,j}\right]$ (when $w_{min,j} < 0$). The attack doesn' t depend by *b* and must be $b > 1$.

3.2. Clustering Methods

The clustering methods work by detecting the evil models by clustering the layers-parameters and by considering the smallest cluster as the evil. A common assumption is that the number of evil clients is less than $\frac{N}{2}$ where *N* is the number of users.

The *Auror* [19] works by clustering the models based on their *indicative features*, every cluster with a number of models below $\frac{N}{2}$ is marked as a suspicious cluster, and the models which appear as suspicious in more than 50% of the *indicative features* are considered malicious. The *indicative features* are identified by dividing into 2 clusters the layer-parameters θ_L (*weights* of the layer *L*) for the first 10 training epochs, then the distance (euclidean distance) between the center of the clusters is computed and if it exceeds a specific α these θ_L are considered the *indicative features*. The clustering is performed using *Kmeans* [24] but could be used any others clustering algorithm.

The *FoolsGold* [23] assumes that benign clients can be distinguished from the attacker by the diversity of their *weight's* update. It modifies the aggregation function as in *Equation 3.3*:

$$G_{t+1} = G_t + \sum_{i=1}^{K} \alpha_i * \left(W_t^i - G_t\right) \tag{3.3}$$

Where α_i is an adaptive learning rate, W_t^i is the model's update of user *i*, G_t is the global model at round *t*. The α_i is different for each user *i* and is based on:

- the cosine similarity among *indicative features* of W_t^i
- Historical information from past rounds

The *indicative features* are different from *Auror*, and in this case are the weights of the last layer. The historical information H_i is the sum of the weight's distances $W_t^i - G_t$ from the past iterations. Practically *FoolsGold* for each model *i* computes the maximum cosine similarity c_i among H_i and H_j with $j \neq i$, then calculates the $\alpha_i = 1 - c_i$, so the learning rate is the opposite of the c_i. Models with maximum cosine similarity are considered malicious and are penalized with smaller α_i giving less impact in the final aggregation.

The *FLGuard* [21] uses a dynamic clustering, like HDBSCAN[1], over the models, it considers the cluster with at least 50 % of the models as the cluster which contains the benign, the remaining ones are marked as outliers. Since a possible evil could pass, an adaptive *clipping* and *noise* are applied to each model in order to reduce the impact of the backdoored model. *Adaptive clipping* is used to reduce the magnitude of the weight's update W_i , it works by selecting the median S of all L_2-Norms of $(W_1,\ W_2,\ W_3, \dots ,\ W_k)$ classified as benign (bigger cluster), then each model is clipped as $W_i = W_i * \min(1,\ \frac{S}{e_i})$ where S is the median and e_i is the L_2-Norm of W_i. The *Adaptive Noising* used a Gaussian distribution which is added to the global model after the *aggregation,* like $G_t = G_t^* + N(0,\ \sigma)$ where σ is computed as $\lambda * S$ where *S* is the median (the one found above) and λ is a parameter found empirically.

Attack Auror

Auror was the first approach proposed and the simpler to bypass. As reported in [19] there are 2 main methods to evade *Auror,* demonstrated empirically:

- decreasing the fraction of malicious models has demonstrated a decay of the detection rate
- decreasing the number of poisoned images in evil's training dataset has shown a deterioration of the detection performance

Attack FoolsGold

As reported in [20] if only 1 attacker is present the detection strategy fails, *FoolsGold* is influenced by the distribution of the data. It performs well when data are *non-IID*, instead fails when data are *IID*. Furthermore 2 possible attacks exist:

- a set of intelligent attackers send pairs of updates $V_i = W_i + g_i$, $V_j = W_j + g_j$ where g_i and g_j are a perturbation with sum up to zero ($g_i + g_j = 0$). Doing that, increases the dissimilarity among the weight's update reducing the effectiveness of detecting them.

- adversaries compute their similarity among weight's update and decide to send only the one with the historical similarity H_i low reducing the ability of *FoolsGold* to detect them.

FLGuard

FLGuard is the best one, it solves the problem of *FoolsGold* since can work also with *IID* distribution and is not dependent on the number of attackers, moreover there is no adaptive attack to evade the defense. The only problem is that some benign models are rejected in the first phase when they use dynamic clustering.

3.3. BaFFLe Method

In *BaFFLe* [22] the Server sends the global model to a set of clients which are used for validating it. The validation works by running the model on the *local dataset* D of each client and by computing 2 types of errors:

- *Source-focused* error $err_D(f)^{y \rightarrow x}$ is the fraction of samples in D which belong to class y and are misclassified by the model f.

- *Target-focused* error $err_D(f)^{x \rightarrow y}$ is the fraction of samples D which f wrongly assigns to class y

BaFFLe defines $v^s\,(f,\ f',\ D,\ y) = err_D(f)^{y \rightarrow x} - err_D(f')^{y \rightarrow x}$ and $v^t\,(f,\ f',\ D,\ y) = err_D(f)^{x \rightarrow y} - err_D(f')^{x \rightarrow y}$ as the *error variation* for respectively *Source-focused* error and *Target-focused* error. The f and f' are the global model at round t and the global model at round $t + 1$. The intuition is that the variation among global models at consecutive rounds is very close if there is no attack and then if the *error variation* is different from the past rounds it is marked as an outlier and will be rejected. Practically the decision of rejecting or accepting the model relies on all clients. Each client computes the *error variation* with its *local dataset D* and if it is different from the past rounds he sends back to the server a 1 (reject) otherwise a 0 (accept), then the majority of the votes decide the fate of the model.

This approach has 2 main vulnerabilities:

- The *error variation* is significant only when the global model is at the end of the training and so is not possible to start the defense at the initial of the train.
- If the previous global model is backdoored the difference among the *error variations* remains closer and then it is not more marked as outliers, consequently, the global model will remain backdoored.

4. SYSTEM CONFIGURATION

In this section, we introduce all settings and configurations of the environment where the experiments reported in *Section 5* have been conducted.

In all experiments, the number of users N and evil clients p is different depending on the type of the analysis. For simulating a *Federate Learning* scenario we used the model ResNet18 (*Section 2.1.2*) and CIFAR – 10 dataset reported in *Figure 4.1*.

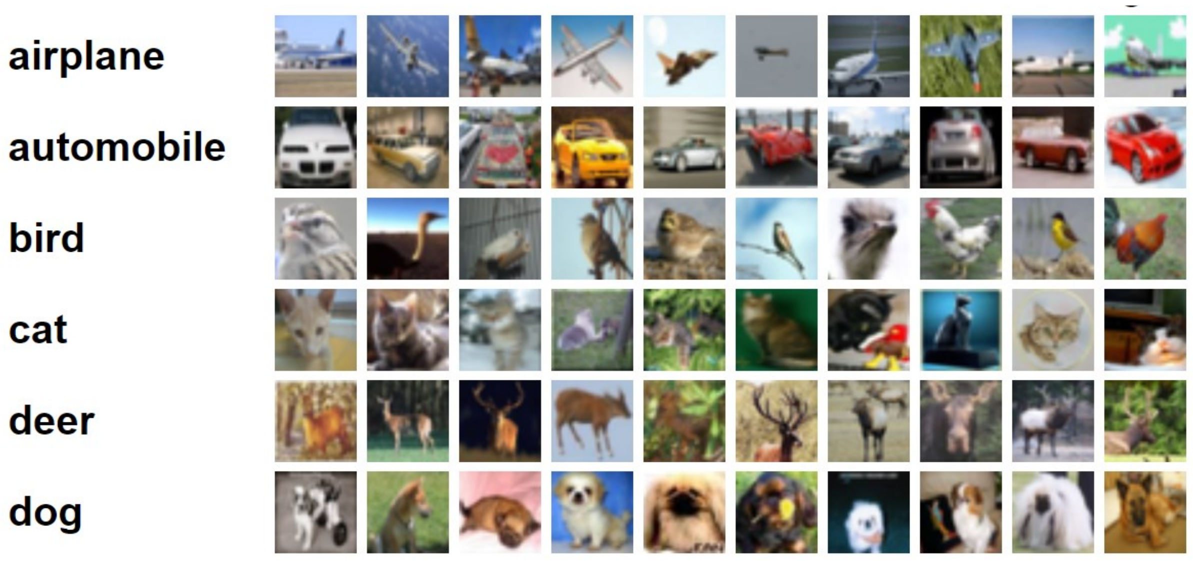


Figure 4.1: *CIFAR - 10 dataset used for conducting the experiment.*

The CIFAR-10 contains 60000 images of 32x32 dimensions, it has 10 classes, airplane, automobile, bird, cat, deer, dog with 6000 images each. The *local dataset* of each user participating in the training has 384 images and is generated based on the degree of IID (identical independent distribution) we want to use. Only in the first experiment (*Section 5.1*) we consider 3 types of IID:

- *Completely IID* (independent identically distributed) where each *local dataset* has the same number of images per class
- *semi-IID* where the *local dataset* contains 50 % images from a single class and the other from different classes
- *non-IID* where the local dataset contains 90 % of images that come from the same class and the other 10 % from the different classes.

In the others, instead, only *non-IID* is maintained because is the worst case. The dataset of evil users is slightly different from the benign ones because contains the *poison* images. These are 50 % of the whole dataset and they exploit the *semantic backdoor* where the green cars are predicted as bird (*target* class) with the *trigger* being the color green. For experiments in *Section 5.2* and *Section 5.3* the *trigger*, instead changes, and is a wall placed behind the car and the *target* class remains the bird class.

In *section 5.4* we use a different dataset, the MNist, reported in *Figure 4.2,* and we consider another architecture which is the CNN+FCL (*Section 2.1.2*), used to perform a comparison with ResNet18. In this case, we adopt a *pixel-backdoor* and the *trigger* is a yellow square placed on the left bottom of the *poison* images. The *target* class is the number 0.

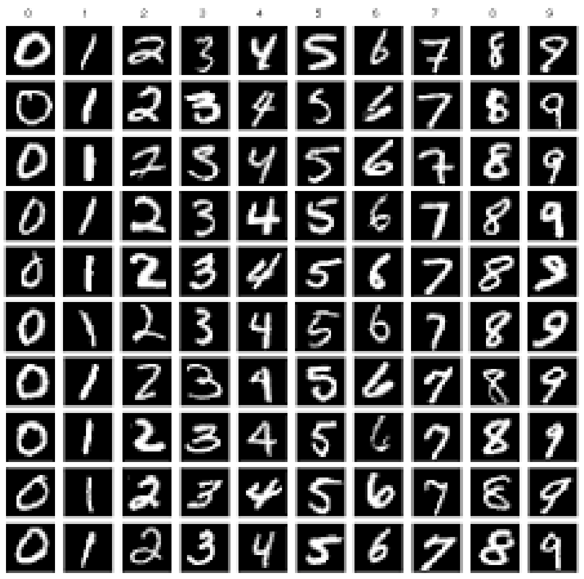


Figure 4.2: *MNIst dataset*

All benign users have the same learning rate and the number of local epochs, also evils have the same learning rate and epochs but are different from benign. We use the terms *lr_e, E_e, lr_b, E_b* to refer respectively to the learning rate of evil, the epochs of evil, the learning rate of benign, and the epochs of benign.

The backdoor attack is implemented as described in [3] which is the PGD (*Section 2.3*). Each user has a TEE (*Section 2.4*) where the models are stored, the *Figure 4.3* shows a representation of the network.

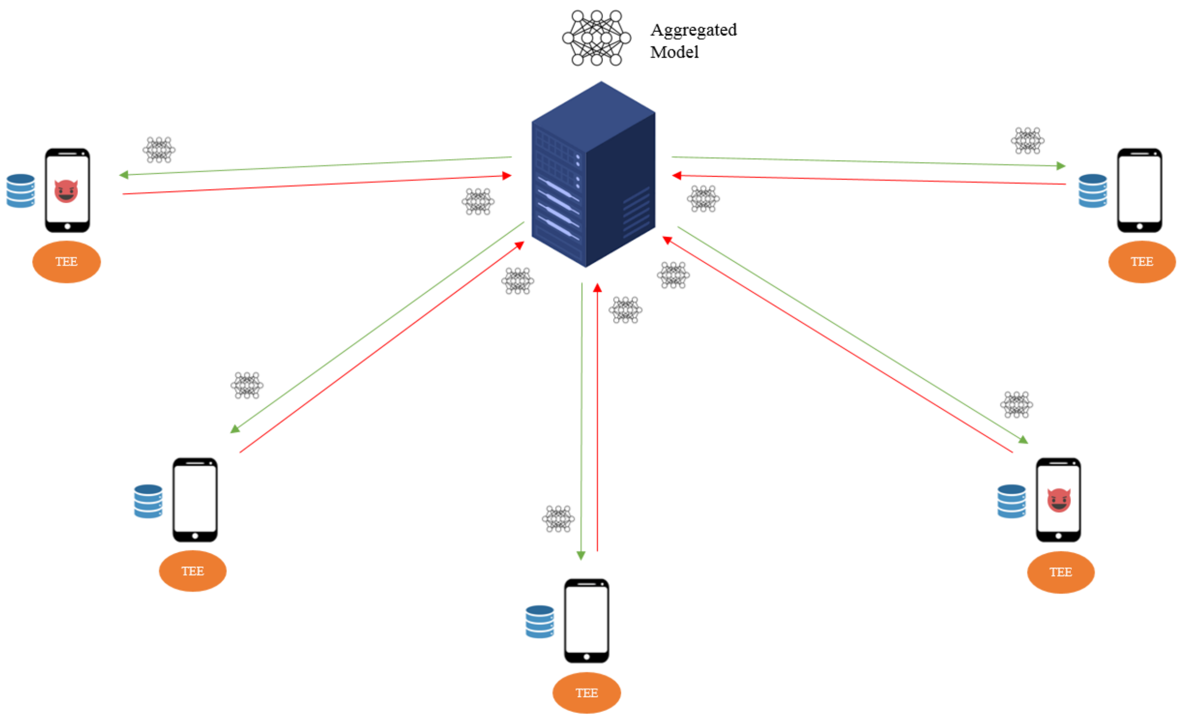


Figure 4.3: *Federated Learning with evil users and with each client equipped with a TEE where the models are stored for the evaluation*

The server collects the model's update from each user, then sends a group of *K* models to each user. These are stored directly in the *TEE* without the possibility for the client to inspect them. Inside the enclave, we conduct all the evaluation and analysis. The dataset used could be a normal or a poisoned dataset depending on the analysis and the experiment.

5. EXPERIMENT

In this section, some approaches are tested and analyzed in order to counterattack the backdoor attacks. The main purpose of the analysis is to pose some possible directions and highlight the drawbacks of each. In *Section 5.1* we investigate the possibility to separate evil models from benign ones based on the output from the deepest layers in the architecture. In *Section 5.2* we analyze the layer-wise distances among the models. In *Section 5.3* we analyze the correlation among layers. In *Section 5.4* the defense paradigm change, we exploit the fact that backdoored model contains neurons that will act only when the image contains the *trigger*, and the idea is to put a wall to block these neurons and consequently delete the backdoor.
In all sections first, we introduce the purpose of the experiment explaining the goals, then we share, briefly, the methodology used for running and acquiring the data, and then we show and analyze the main results.

5.1. Output Analysis

5.1.1. Purpose

This analysis is the first approach investigated and also the more intuitive. The backdoor attack aims to misclassify images from a specific class with another call *target* class, for example, a *car* as a *bird*. Consequently, each evil model will learn how to predict these images in a different way than the benign does, and then, the training will be also different. In order to have a wrong prediction the output from the layers must be different and this is the main hypothesis exploited.

5.1.2 Methodology

This approach consists to separate the evil models from the benign by observing the output from deep layers like *Layer3* and *Layer4,* remember *Figure 2.6* in *Section 2.1.2.*

The experiment uses *N* = 100 and *p* = 49 (wrong case). We consider 2 scenarios:

1. attack after 1000 rounds, when models converged
2. attack after the round 0 when models are not yet trained

Each client receives 5 models, computes the average output-layer both from *Layer3* and *Layer4*, and then sends the results to the *Server*. The *Algorithm 1* shows how a client computes the output given its *local dataset* and the *5* models.

```
Data: K models, Dataset D

1.  For each model in models:
2.      $O_{Layer3} = []$
3.      $O_{Layer4} = []$
4.      For each d in D:
5.          $O_{d,Layer3} \leftarrow model(d)$
6.          $O_{d,Layer4} \leftarrow model(d)$
7.          $O_{Layer3}.append(O_{d,Layer3})$
8.          $O_{Layer4}.append(O_{d,Layer4})$
9.      $< O_{Layer3} > = \frac{\sum_{d \in D} O_{d,Layer3}3}{|D|}$
10.     $< O_{Layer4} > = \frac{\sum_{d \in D} O_{d,Layer3}4}{|D|}$
11.     Send $< O_{Layer3} >$ and $< O_{Layer4} >$ to Server S
```

Algorithm 5.1: *Output computation given in input the* ***local dataset*** *and* ***K*** *models.*

The output from *Layer3* and *Layer4* is *n* – dimension and so in order to visualize them on 2 – dimensional space we use *PCA* (*Principal Component Analysis*) for dimensionality reduction. The *lr_e* and *E_e* are 0.025 and 15, and the *lr_b* and *E_b* are 0.1 and 2. The *lr_e* and *E_e* are big only for technical reasons because the attack explained in [3] needs a small update for the weights, which means a small *lr* and in order to insert the backdoor it needs to train the model more times like 15.

5.1.2 Results and Analysis

We first report results when the attack is performed after 1000 rounds and the data are *completely IID,* then we do a comparison with *semi-IID* and *non-IID* distribution.

Attack after 1000 rounds

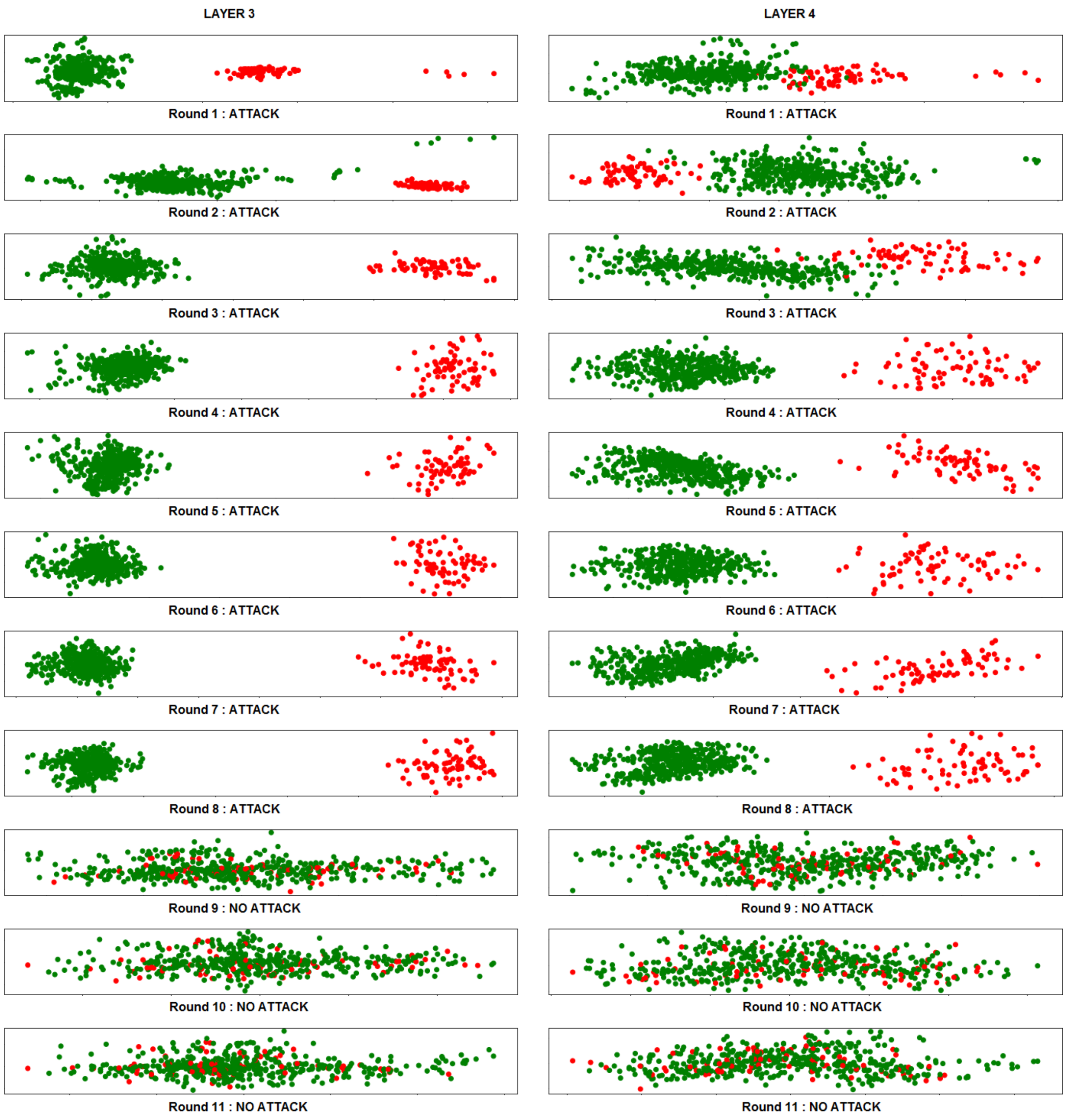


Figure 5.1: *Output from the last layer (Layer4) on the right column, output from the Layer3 on the left side, the green dots represent benign models instead red dots are the evils, data are completely IID with 49 evils and 100 clients. Attack presents on the first 8 rounds.*

The *Figure 5.1* shows output-layer received by the *Server*. The data are completely *IID* (independent identically distributed). It is important to remember that the *Server* sends 5 models to each client and so each model has 5 outputs, consequently the number of dots is more than 100 (number of clients), because are 100*5 and so 500.
As you can see the dots during the attack define 2 big clusters, one is green (benign models) and the other is *red* (evil models), furthermore, the *Layer3* gets better results, with *Layer4* at the first 3 rounds is impossible to separate all benign from evil. A possible defense strategy leverages the clustering technique. Since the assumption is that evils cannot be more or equal to half of all models, the bigger cluster contains the benign and the other the evils. The most used clustering algorithm is HDBSCAN [1] which is an improvement of DBSCAN [2] that works without knowledge of the number of clusters.
Unfortunately, these good results depend on the distribution of data. In this case, is *completely* IID and so each benign client has equal distribution except for evil which has half of the dataset poisoned. During the training, all benign learn the same distribution, different for the evils and this gets similar *weights* which generate similar output, far from the evil. In fact, when data are *semi-IID* or *non-IID* the results change, *Figure 5.2* shows this fact.

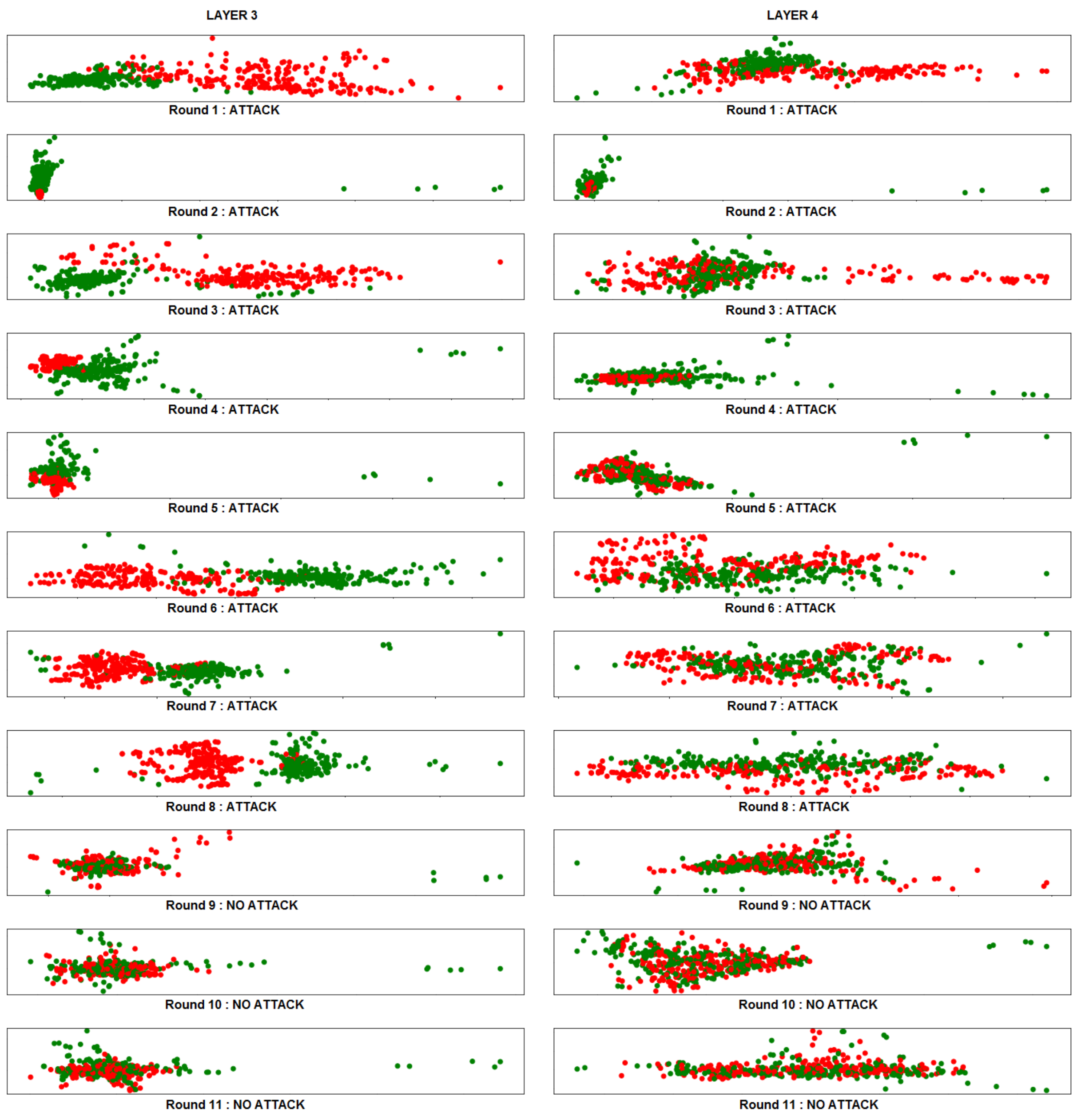


Figure 5.2: *Output from the last layer (Layer4) on the right column, output from Layer3 on the left side, the green dots represent benign models instead red dots are the evils, data are SEMI-IID with 49 evils and 100 clients. Attack presents on the first 8 rounds.*

Observing the output of *Layer3*, when there is an attack, some *green* are far from *red* but there is no visual separation and is not possible to apply any clustering methods to separate them. On the right side (output from *Layer4*) the situation is worse because, when there is an attack, the *red* and *green* are mixed, in the same way when no attack is present. Same results are obtained for *non-IID* data.

Other drawbacks come when the attack starts in the initial round. In the next section we analyze this scenario.

Attack after first rounds

In the beginning all models have the *weights* randomly initialized and their accuracy is very low, around 10 %. For this reason, the output-layers are very dissimilar, almost random. The *Figure 5.3* shows how the output appears at the first 11 rounds with the attack on the first 8 with *completely-IID* data.

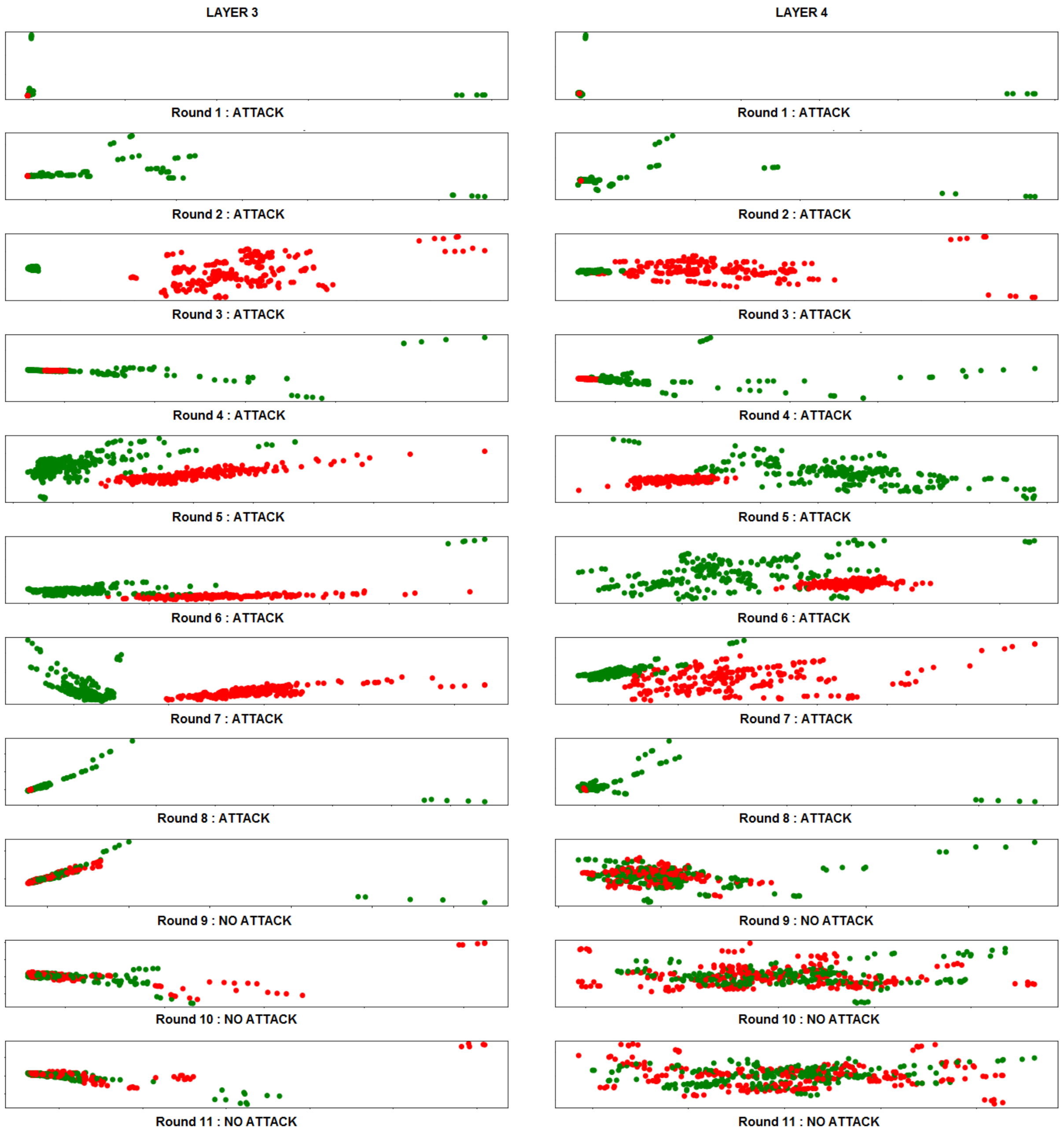


Figure 5.3: *Output from Layer4 on the right column, output from Layer3 on the left side, the green dots represent benign models instead red dots the evils, data are completely IID with 49 evils and 100 clients but at the beginning of the training. Attack present in the first 8 rounds.*

From *Figure 5.3* is possible to notice that the situation is in the worst case, with *Layer4* (right column) that gives the same results as for *semi-IID* where the dots are not separable. On the left side (*Layer3*) only for round 3 the detection is possible but in the previous and next doesn't.

Overall these problems, related to data distribution and if the attack starts early or not, another issue exists, which is the most important, and depends mostly on the nature of the architecture. Let's suppose that you have fixed previous problems, a possible defense could apply HDBSCAN [1] on the *Layer3* output and consider the biggest cluster as the benign ones. Consequently, an attacker, in order to bypass the detection, must put *Layer3* output as close as possible to benign, unfortunately, this is possible because the attacker knows the global model from the previous round and so he can extend the work done in [3] to make its model as close as possible to benign bypassing the defense. The idea is to constrict the *Layer3* output to be closer to the global model and the result is that a backdoor model will be grouped with the *green* (benign models) without disrupting the backdoor.

In summary, seems that evil models and benign models generate different outputs from *Layer3* only when data are *completely IID,* instead, doesn't when they are *semi-IID* or *non-IID.* Besides being necessary a *completely IID* dataset, is also important that models converged (after 1000 rounds) or at least have higher accuracy. Furthermore, in the case these issues have been fixed, an attacker can bypass the detection by implementing the adaptive attack explained above.

5.2 Distance Analysis

5.2.1 Purpose

The *Distance Analysis* aims to solve the problem related to distribution by analyzing the distances among output instead of the output itself.

5.2.2 Methodology

Given a dataset *D* and a couple of models *A* and *B*, for each image is computed the *Cosine Distance* among the output-layer as shown in *Algorithm 5.2*, obtaining a value in [0, 1].

```
Data: model A, model B, Dataset D

1   For each Layer in Layers:      #      iteration over the layers in the architecture
        a.  $Dist_{Layer} = []$
        b.  For each d in D:
                i.   $O_{d,Layer,A} \leftarrow model_A(d)$
                ii.  $O_{d,Layer,B} \leftarrow model_B(d)$
                iii. $dist_{d,Layer} \leftarrow CosineDistance(O_{d,Layer,A}, O_{d,Layer,B})$
                iv.  $Dist_{Layer}.append(\ dist_{d,Layer})$
        c.  Return $Dist_{Layer}$
```

Algorithm 5.2: *Compute distances among model A and B given a local dataset.*

For each *Layer* we compute 7 statistics over $Dist_{Layer}$ for extracting information about the distribution:

1. The *Mean* that is suitable for synthesize the data: $\frac{\sum_{d \in Dist_{Layer}} d}{|Dist_{Layer}|}$.
2. the *median* that represents the value stand in the middle of the $Dist_{Layer}$.

3. the *skeweness* that is a measure of the symmetry: $\frac{\frac{\sum_{d \in Dist_{Layer}}(d-mean)^3}{|Dist_{Layer}|}}{\sqrt[3]{\frac{\sum_{d \in Dist_{Layer}}(d-mean)^2}{|Dist_{Layer}|}}}$.
4. the *kurtosis* that is a measure of the "tailedness" and give an idea of the degree of anomalies present (> 0 means lower number of anomalies, < 0 instead more anomalies): $\frac{\frac{\sum_{d \in Dist_{Layer}}(d-mean)^4}{|Dist_{Layer}|}}{standard\ deviation^4} - 3$.
5. the *standard deviation* that defines how far all data in $Dist_{Layer}$ are from the *mean:* $\sqrt{\frac{\sum_{d \in Dist_{Layer}}(d-mean)^2}{|Dist_{Layer}|}}$.
6. the *max* that is the maximum of $Dist_{Layer}$.
7. the *min* which is minimum of $Dist_{Layer}$.

The *Cosine distance* is computed as follow, given two tensors *u* and *v* the distance is equal to $1 - \frac{u \cdot v}{\|u\| * \|v\|}$ where $\|u\|$ represents the norm and $u \cdot v$ is the dot product.
We have considered others types of distances but the results didn't change. The metrics explored was:

- *Euclidean distance*: $\sqrt{\sum_i^n (u_i - v_i)^2}$ where u_i and v_i are the *i*-esimo element of *u* and *v* with dimension *n*.
- *Manhattan distance*: $\sum_i^n (u_i - v_i)$.
- *Chebyschev distance*: $\max_i (u_i - v_i)$.
- *Wasserstein distance:* $(\min_{J \in \zeta(u,v)} \int \|x-y\|^p dJ(x,y))^{\frac{1}{p}}$ where $\zeta(u,v)$ represents the set of all join distribution among *u* and *v*.
- *Symmetric Kullback - Leibler distance:* $\frac{1}{2} * KL(u||v) + \frac{1}{2} * KL(v||u)$ where $KL(v||u)$ is equal to $\sum_i^n v_i * \log_2 \frac{v_i}{u_i}$.

Since *Wasserstein* [4]-[5] and *Kullback – Leibler* are distances used among probability distribution, *u* and *v* must be normalized before the computation. The main difference is that *Wasserstein* computes the amount of work consumed for transforming a distribution in the other, *KL* instead gives the amount of information you lost for replacing one distribution with respect to the other.

Since we need to compute the distance for each couple, to reduce the computational cost we used a network with $N = 20$ and $p = 9$ evil (49 %) and, in order to have the distances depending only on the type of data distribution, we have set a *lr* and *E* equal to 0.025 and 15 for each client, the data are *non-IID (*worse case).

Only 2 clients receive all models, one is evil with its *local dataset* poisoned instead the other is benign with normal images.

Since the data are *non-IID* all users have a *local dataset* with the *main label* (or main class) that is the class that occupies the 90 % of the whole dataset, therefore is possible that 2 or more users train the models on a dataset with same *main label* and so, in the analysis, we have distinguished the distances among models trained on the same *main label* with models trained on a different one. For evil models the term *main label* is slightly different from benign, in fact, the poisoned dataset has 2 bigger classes, the *target* class and another class, which, in some cases, could coincide; the *main label* refers to the class different from the *target*, unless only the *target* is present.

We consider both the distances computed among the models and among the models and the global model. Only with the global model we have also distinguished the distances among models with the same *main label* of the *local dataset* used for the computation.

First, we analyze the distances without attack in order to understand the meanings of each statistic and, subsequently, to use the ones which give more information about the distribution. Choosing a few of them, we, then, analyze the scenario where the attack is pursued. With respect to *Section 5.1*, we also consider all other layers and not only the *Layer3* and *Layer4*.

5.2.3 Results and Analysis

No Attack

The *Figure 5.4* reports the statistics for each distance, computed for each layer.

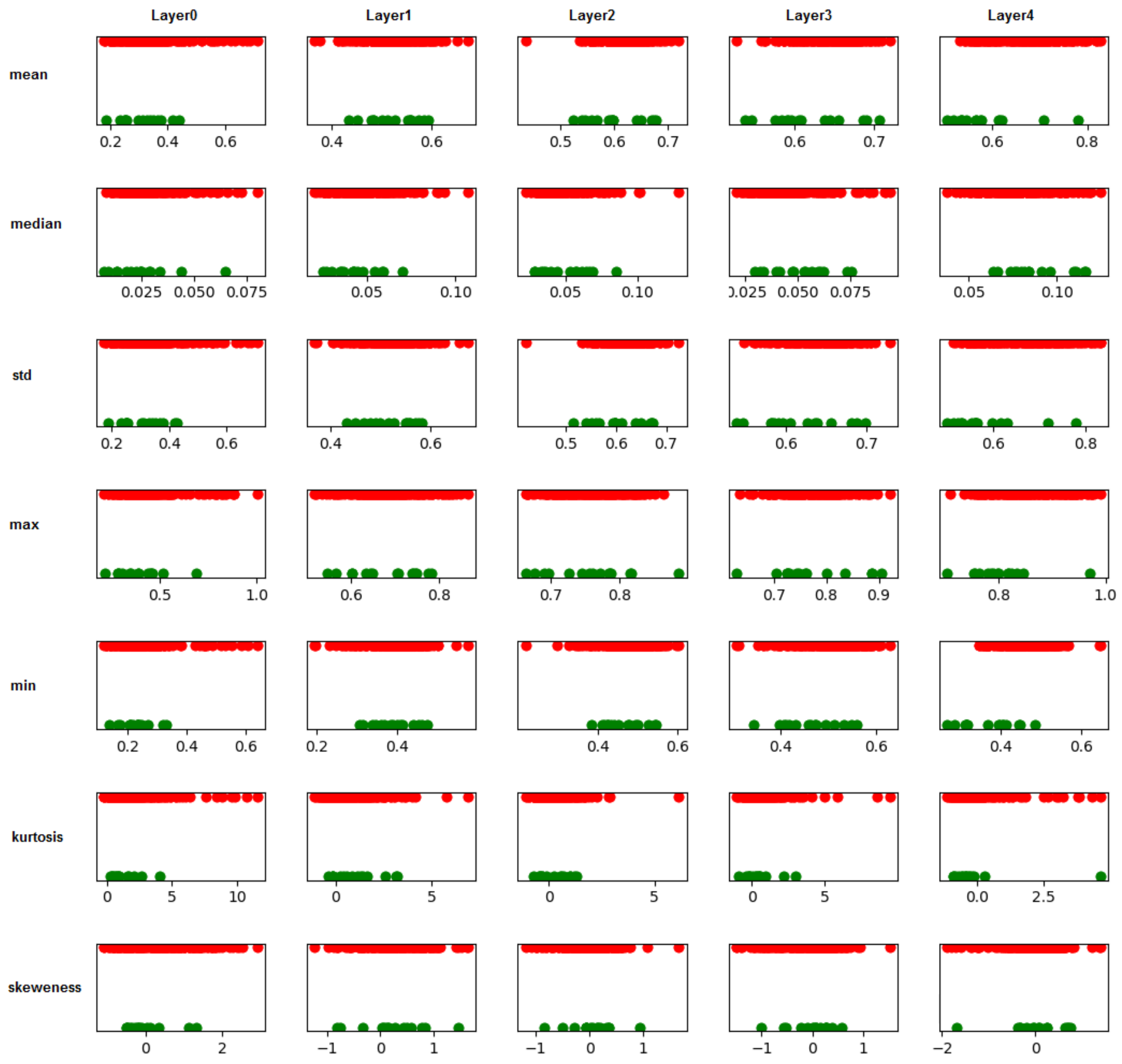


Figure 5.4: *Distances at the beginning of the training without attack. Red dots represent models with different main label, green dots represent models with the same main label; the x-axis contains the value of statistics and the y-axis is used for distinguishing the 2 groups. Each column represents the layer where output is taken, each row represents the statistic.*

For each layer the *mean* doesn't show huge differences among *red* and *green (Figure 5.4)*, except for *Layer4* in which some *green* have lower distance than most of the *red*. The *median* gets the same results. For *standard deviation* in each layer the distances are below 0.125, for

Layer2 and *Layer4* are also lower but in general no differences between *red* and *green* exist. *Min* gets similar results to the *mean*, also better for *Layer4* because is more visible the differentiation among *red* and *green*. The *Max* doesn't show anything. About *Kurtosis,* for *Layer2* and *Layer4* (except for a couple of models) *green* are closer to 0 which means have few outliers. The *Skewness* gets no meaningful information.

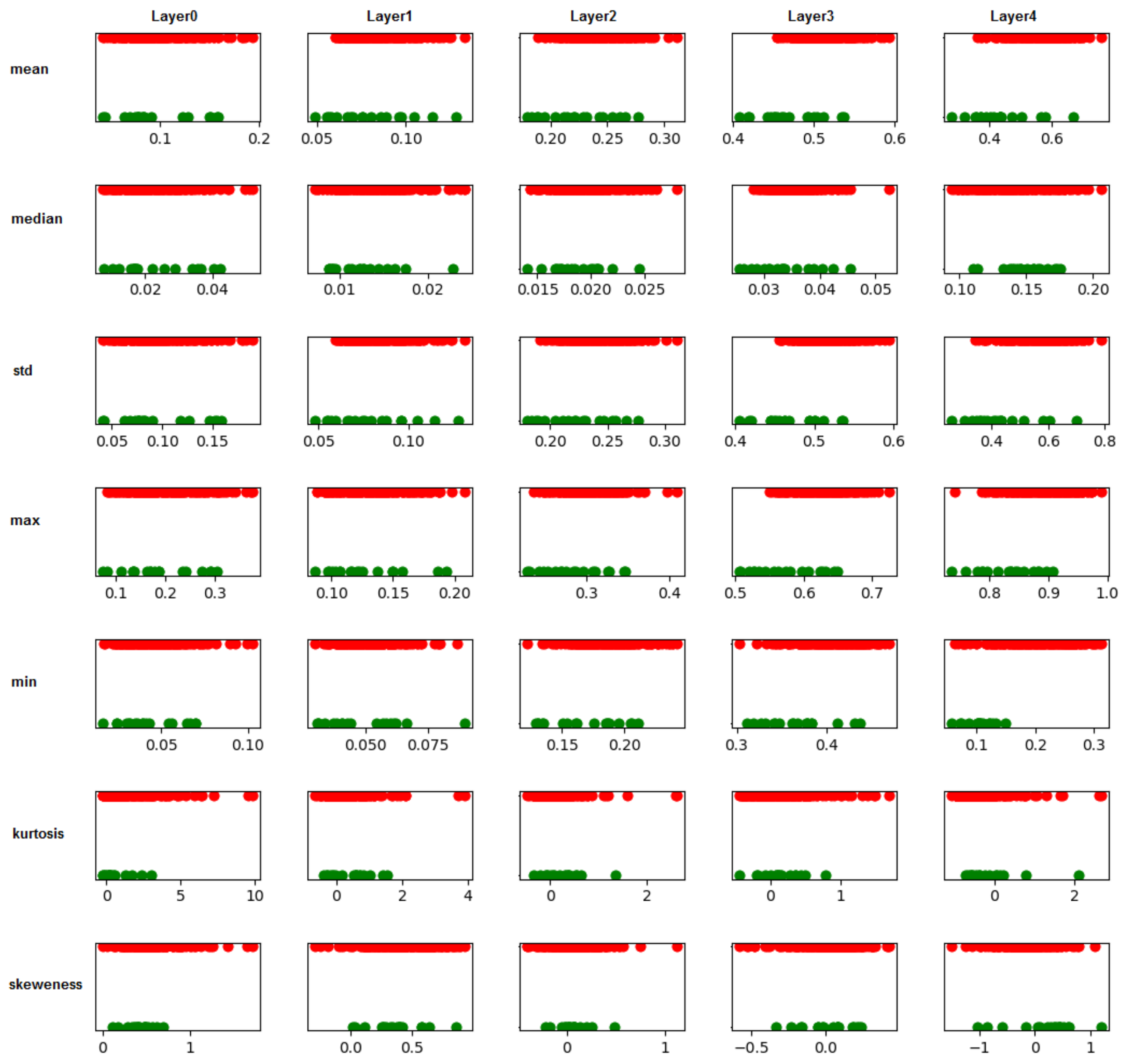


Figure 5.5: *Distances after 1000 rounds of the training without attack. Red dots mean models with different main label, green means models with the same main label; the x-axis represents value of the statistics and the y-axis is used for distinguish the 2 groups.*

After 1000 rounds the models converged. The *mean* decreases for almost all layers, some *green* have lower distances than *red*, especially for *Layer3* and *Layer4* (*Figure 5.5*).

The *std* decreases except for *Layer4* which instead increases. For the other statistics, no interesting differences are present except for *kurtosis* which is decreased. A possible explanation depends on the fact that models converged and since the dataset used to calculate the distances has 90 % images (*non-IID* dataset) from the same class, for all of them the model gets almost the same output or maybe slightly different, consequently, the distances remain constant or change a few with a small deviation with the result of a very low number of anomalies (remember the *kurtosis* is close to 0 when the distribution contains a low number of anomalies)

This brief data analysis without attacks allows us to have an idea of which statistics are better to consider during the next analysis (with attack). The *mean, median*, and *min* on *Layer4* are more *significant* and *robust*. *Significant* because they reflect the fact that models trained on the same distribution, with the same *main label,* get closer output (small distance) than the others trained on different distributions in fact with *mean, median,* and *min* some of the *green* are lower than the most of *red*. More *robust* because this situation is maintained also after 1000 rounds so when models converged.

Attack

At the beginning of the training the models have no good performance and so the output from layers is random without defining any kind of pattern. Therefore the next analysis is concentrated when the attack is performed at the 1000th round. From here all figures contains only the *mean, median* and *min* because, as we explained above, they are more useful.

Before reporting and discussing the results we briefly explain the color's meaning of the figures reported below in order to better understand them.

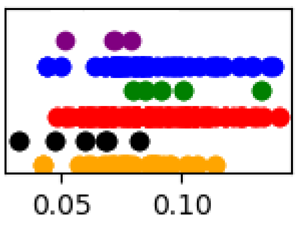


Figure 5.6: *Distances statistics*

The *Figure 5.6* represents a statistic extracted from distances related to a specific layer. The x-axis contains the values, and the y-axis is used to distinguish the different types of models. The colors have the following meaning:

- *Green* represents benign models with the same *main label*
- *blue* is benign with a different *main label*
- *purple* benign-evil with the same *main label*
- *red* benign-evil with a different *main label*
- *black* is evil-evil with the same *main label*
- *orange* evil-evil with a different *main label*

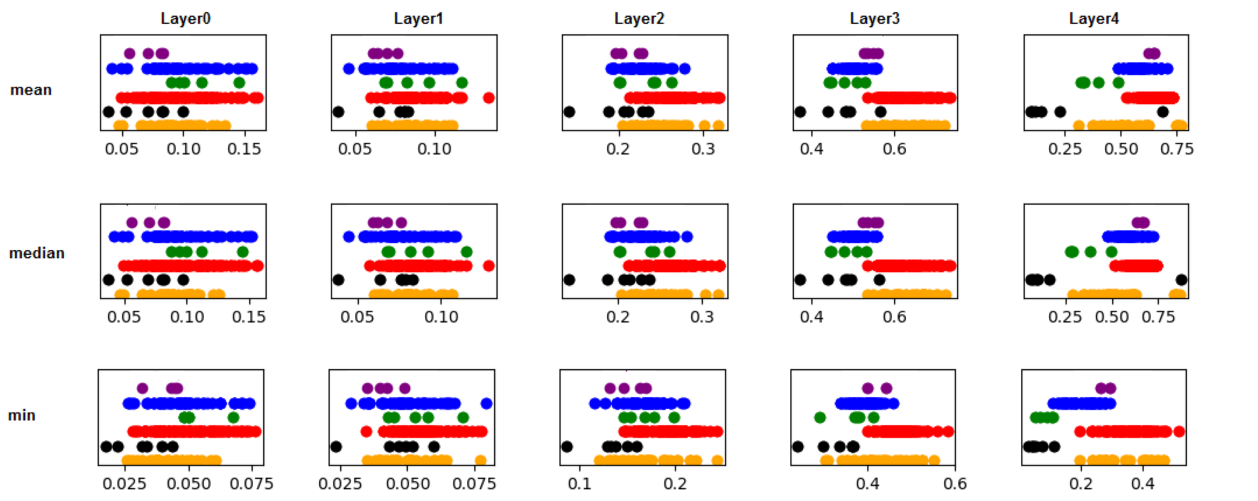


Figure 5.7: *Distances after 1000 rounds of training with the attack at 1000th round. The x-axis represents the value of the statistics and the y-axis is used for distinguishing the groups. Distances are computed on the normal dataset.*

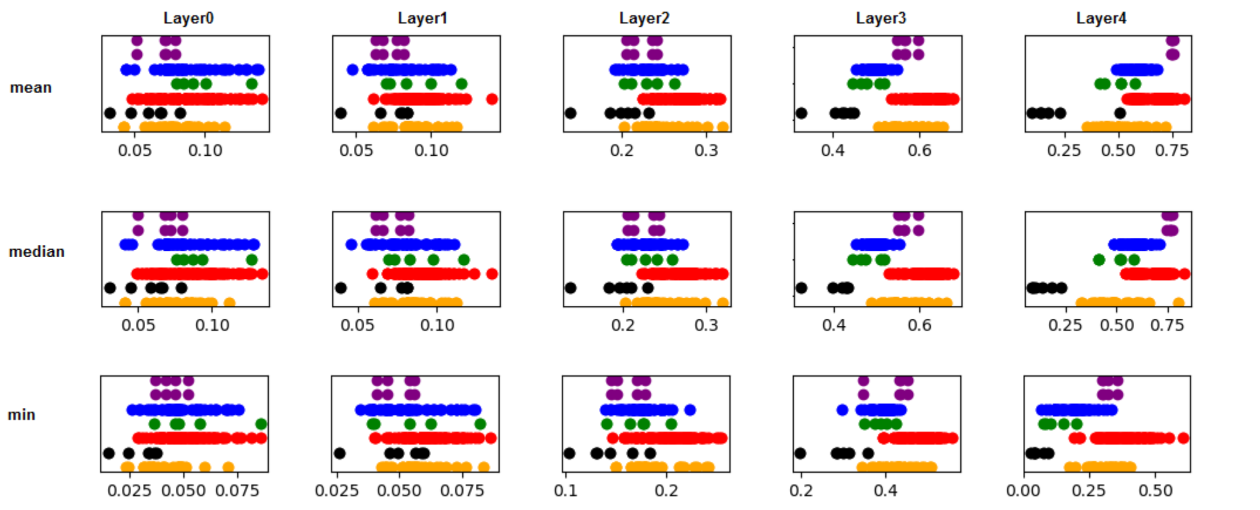


Figure 5.8: *Distances after 1000 rounds of training with the attack at 1000th round. The x-axis represents the value of the statistics and the y-axis is used for distinguishing the groups. Distances are computed on the poisoned dataset.*

With the normal dataset (*Figure 5.7*) the *black* are the closest ones and this reflects an important hypothesis originally proposed in [8] which claims that *weights* of evil models have the same direction and so they are closer to each other. Unfortunately, in our case this happened only for evil with the same *main label* because *orange* instead fall far from them. The *green* are closer than *blue*, these types of models are benign and so their behavior is similar to when there was

no attack. The *purple* and *red* instead don't give any interesting results, they fall almost on the same distance.
With the poisoned dataset (*Figure 5.8*) for the *black* and *green* remain the same considerations discussed above, instead of for *green* and *blue* there are some differences, some *green* fall where stand the *blue*.
The *Layer3* and *Layer4* get better results, in the others, instead, all points are merged together.
For implementing a good defense the *orange* and *black* must be separated from the others in order to use some threshold or algorithm methods to distinguish *benign-benign* from *benign-evil/evil-evil*, however only the *black* seems to satisfy this requirement and only when the dataset is *poisoned*. Unfortunately, this is not sufficient for building an efficacy detection algorithm.
In the next section we analyze distances computed with the global model.

Analysis with global model

We assume that the global model is benign, and the attack starts at 1000th round. The analysis is done on the same statistics used above such as *mean*, *median* and *standard deviation*. All distances are computed among the local and the global models.
The *Figure 5.9* explains the colors used in order to understand the graphs.

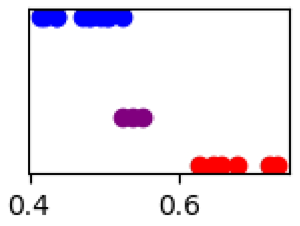


Figure 5. 9: *Statistics of distances with the global model.*

- The *green* color means distances with benign models and the same *main label* of the dataset
- *blue* means benign with a different *main label* of dataset
- *purple* is an evil with the same *main label* of dataset
- *red* is an evil with a different *main label* of dataset

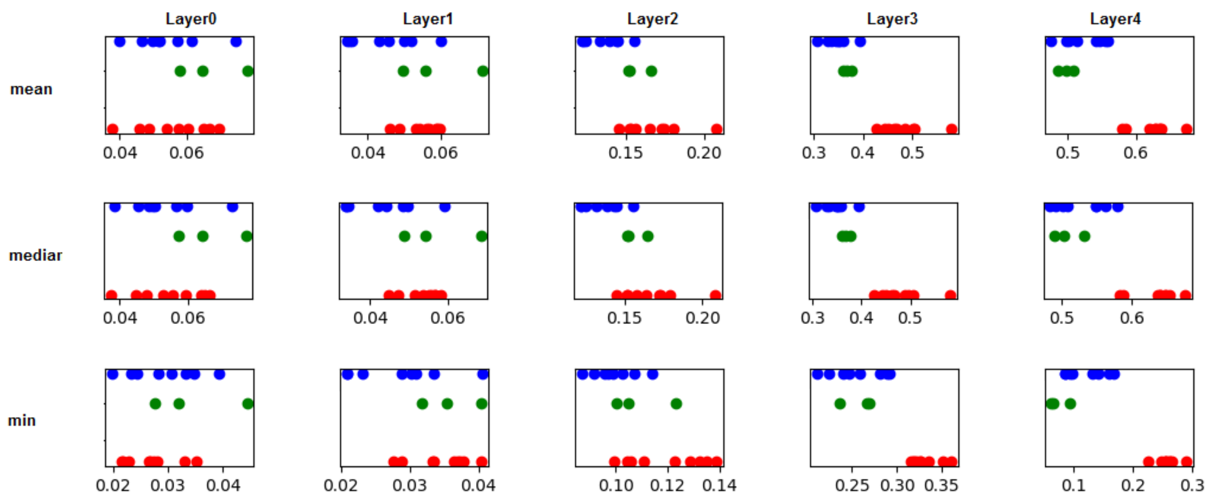


Figure 5.10: *Distances with global model after 1000 rounds of training with the attack at 1000th round. The x-axis represents the value of the statistics and the y-axis is used for distinguishing the groups. Distances are computed on the normal dataset.*

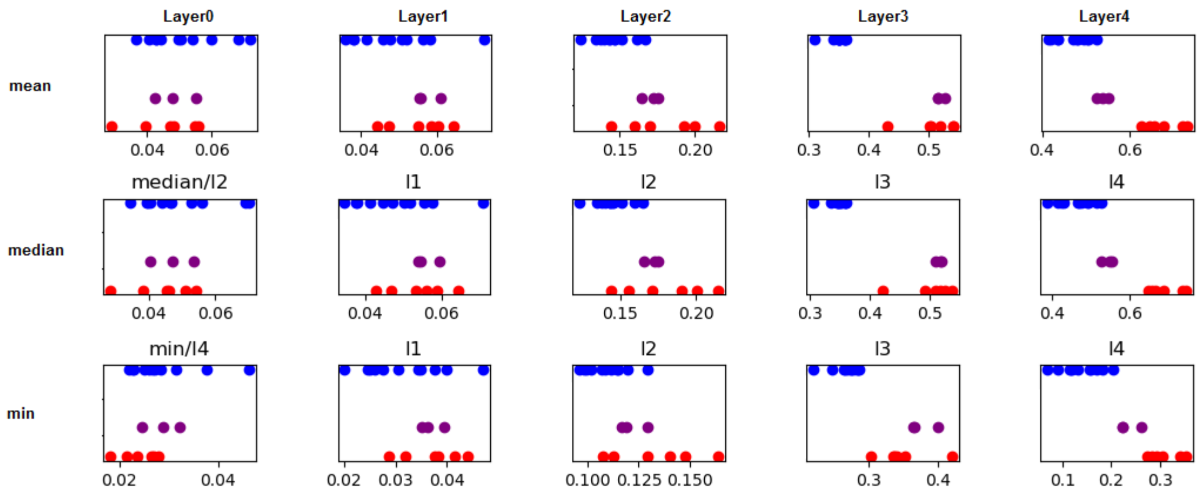


Figure 5.11*: Distances with global model after 1000 rounds of training with the attack at 1000th round. The x-axis represents the value of the statistics and the y-axis is used for distinguishing the groups. Distances are computed on the poisoned dataset.*

Considering the *mean* and the normal dataset (*Figure 5.10*)*,* the *green* and *blue* are closer than *red*, this happened because the previous model is benign and so for that it would be reasonable to expect a closer output since also *blue* and *green* are benign. This also happened for *min* which, however, seems to be possible to separate the 2 types of models. A possible best approach could use the HDBSCAN[1] which identifies groups of points that are closer to each

other, in this case *blue* and *green* are grouped together excluding the *red* which identifies another cluster (the evil). However, when, the dataset is poisoned (*Figure 5.11*) *purple* and *blue* fall together and if we apply HDBSCAN[1] to detect the 2 clusters is possible that one evil is taken and this is enough for allowing the backdoor injection.
With the poisoned dataset, *Layer3* permits separating evil from benign using the *mean* and the *median* but this doesn't work with the normal dataset where the separation is not possible.
Summing up, what we have done was to analyze the distances among layers seeking something useful for detecting evil models. The distances are both computed among models and with the global model. With the first seems is possible to detect the evils with the same *main label* but this is not sufficient for building an efficacy defense because some of them pass. With the global model instead, the type of dataset covers an important role because without *poisoned* images the minimum distances from *Layer4* permit to separate the evil from benign, however, when images are poisoned the *Layer4* becomes useless and *Layer3* acquires more importance for the detection. However, if we want to detect backdoor models based on distances, we need that they are dataset-type-independent because, for example, if we want to use *Layer4* to detect evil models and the models have been sent to a client which has poisoned images, thus, as we demonstrated above the *Layer4* becomes useless allowing evil models to evade the defense, the same thing happened if we use the *Layer3* and the dataset is normal. In the next section, this analysis will be extended by further evaluation of the correlation among layers.

5.3 Correlation Analysis

5.3.1 Purpose

In the previous section, we have described a possible way to detect evil models based on their layer-wise distances both among models and the global model. Unfortunately, some drawbacks were raised, and no efficacy defense strategy is possible to exploit. In this section, we will extend the use of distances by analyzing the *correlation* between them, more precisely correlation among layers.

5.3.2 Methodology

The configuration of the experiment and the network remains unchanged from *Section 5.2*. So all analyses are conducted on *No-attack* and *Attack* scenarios, using both *normal* and *poisoned* datasets. *Algorithm 5.3* shows how correlation is computed among 2 layers, given models *A* and *B* and a dataset *D*.

```
Data: model A, model B, dataset D
Set: $X \leftarrow [], Y \leftarrow []$
    1. For each d in D:
        a. $O_{d,Layer1,A} = model_{A,Layer1}(d)$
        b. $O_{d,Layer1,B} = model_{B,Layer1}(d)$
        c. $dist_{d,Layer1} = CosineDistance(O_{d,Layer1,A}, O_{d,Layer1,B})$
        d. $O_{d,Layer2,A} = model_{A,Layer2}(d)$
        e. $O_{d,Layer2,B} = model_{B,Layer2}(d)$
        f. $dist_{d,Layer2} = CosineDistance(O_{d,Layer2,A}, O_{d,Layer2,B})$
        g. X.append($dist_{d,Layer1}$)
        h. Y.append($dist_{d,Layer2}$)
    2. Return X, Y
```

Algorithm 5.3: *Computing correlation among layers.*

The X and Y are plotted in 2-dimensional space. Analyzing the correlation visually is better than using correlation coefficients like *Pearson* or *Spearman* since these are more suitable for linear correlation but is possible to have a non-linearity shape like *sin* or *cosine* for example.

5.3.3 Results and Analysis

In this section, first we report and analyze data for the *No-Attack* scenario at the first round and after 1000 rounds of training, then we analyze data under *Attack*, again at the first and 1000th round. For each figure *L0, L1, L2, L3*, and *L4* means *Layer0, Layer1, Layer2, Layer3, and Layer4*.

No Attack – Round 0

With no attack all models are benign and is possible to understand the correlation between the layers and how this changes during an attack. Moreover, considering the first round and the 1000th round we highlight the changes when models are not trained and when they converged. In the beginning, the models are randomly initialized and their output per layer gets random results, which means with similar images (similar features) from the same class the output is very different. However, as *Figure 5.12* demonstrates, in some cases, a correlation exists.

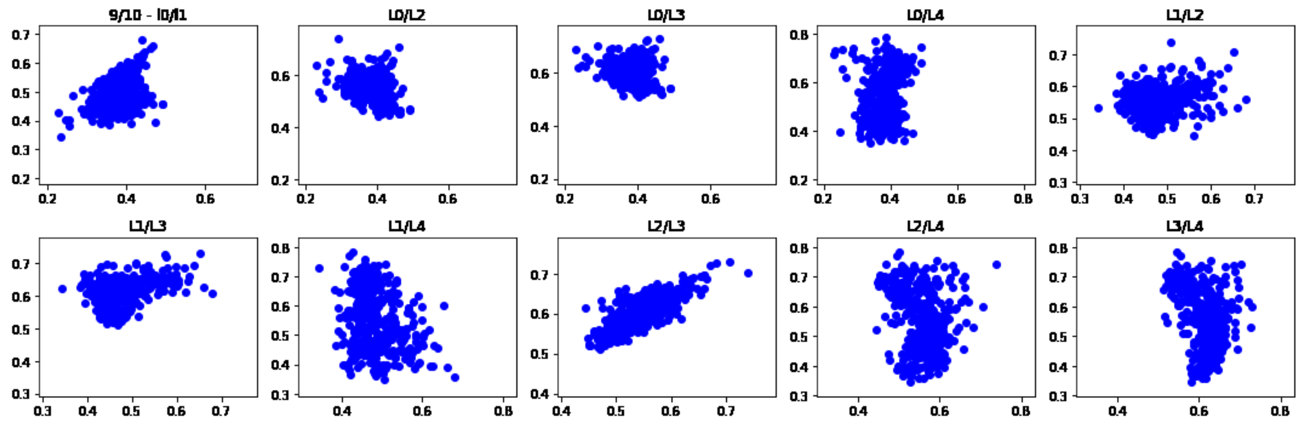


Figure 5.12: *Correlation among 2 models in the first round. Each graph represents a couple of layers, the axis represents the layers reported in the title. For example, for L0/L2 the x-axis projects the distances computed from Layer0, instead y-axis from Layer2. The number 9/10 on the first graph (top left) represents the models involved in the computation. Benign are from 0 to 10 instead evil from 11 to 19.*

Observing the figure, it is possible to see that *L0/L1, L0/L2, L0/L3, L1/L2, L1/L3* have no correlation and the points are grouped as a ball. For layers that contain *L4*, instead, there is a correlation because they are more spread along the y-axis and instead remain close on the x-axis creating a sort of vertical line. Some couple of models have linear correlation also in *L1/L3* or *L1/L2* and all couples present linearity in *L2/L3*, *Figure 5.13.*

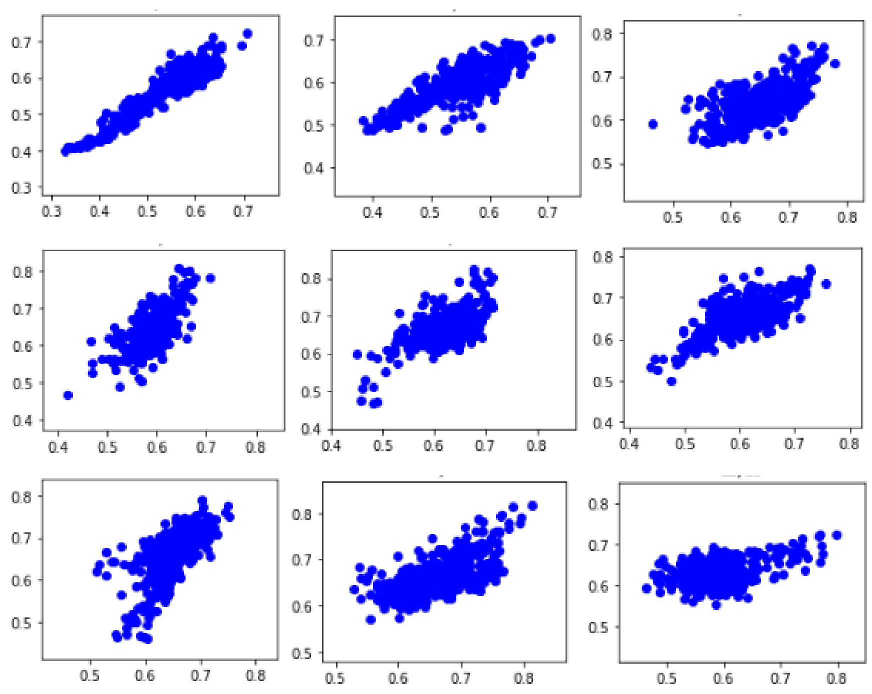


Figure 5.13: *Correlation among Layer2/Layer3 of 10 couples of models*

In some cases, you can see a strong correlation like graph 1 or less correlation like graph 9 but most of the time is similar to the rest of them.

No Attack – Round 1000

After 1000 rounds all the models converged, and the correlation slightly change, *Figure 5.14*.

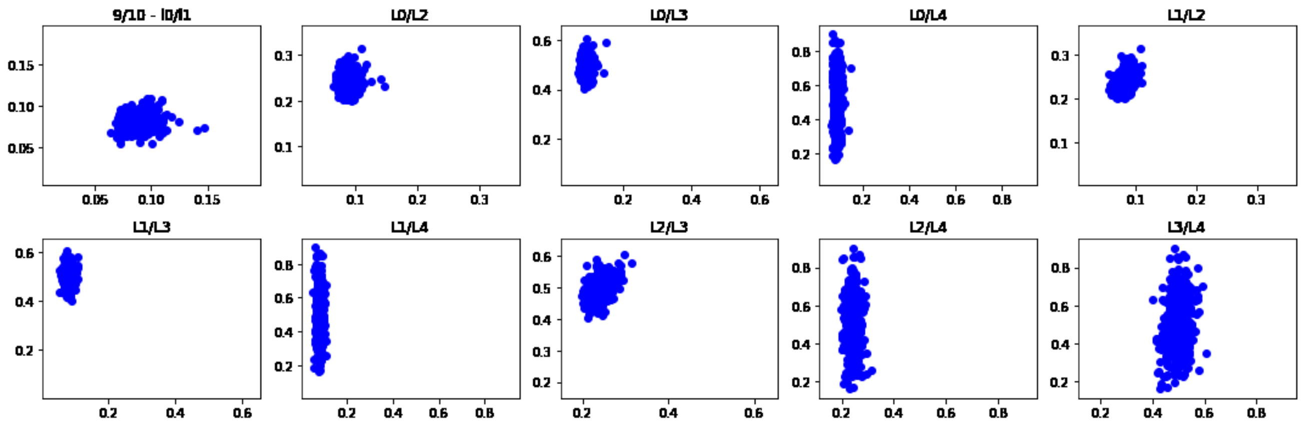


Figure 5.14: *Correlation among 2 models at the 1000th round.*

Each layer doesn't show any correlation except for the ones with *L4*. The *L2/L3* lost its linearity and becomes similar to the others. With *L4* all layers show a sort of vertical column where distances along the x-axis are closer and along the y-axis (*Layer4*) they are spread. At the end of the training the models have reached a very high accuracy like 98 % and each layer has learned a different thing, layers on the top like *Layer0, Layer1* have learned simple patter like *edges* or *corners*, *Layer2, Layer3* instead of more complex pattern like the *shape*, *Layer4* has learned how to associate this pattern to the right class. The distances computed on the top layer (x-axis on each graph with *L4*) are below 0.2 which means, given the same input, 2 models get almost the same output and so they have learned the same pattern, with *Layer3* the distance increase to 0.4 but however for all images the models get almost the same distance or slightly different. With *Layer4* (y-axis) instead, some images get a distance distributed between 0.2 and 0.8, thus *Layer4* captures the fact that these 2 models are different and not equal, they are trained on a different dataset. In conclusion, the top layer says these models come from the same categories because their distances are closer and then the predictions are the same instead *Layer4* highlights the fact that they are not the same models.

Attack – Round 0

Now we show how correlation appears when an *attack* is pursued. The analysis is split into 2 parts, with the attack in the first round and after 1000 rounds. We show the correlation between benign models, evil models and benign-evil models. The distances are both computed on the *normal* dataset and the *poisoned* dataset.

The *Figure 5.15-5.16* shows the correlation for a benign model.

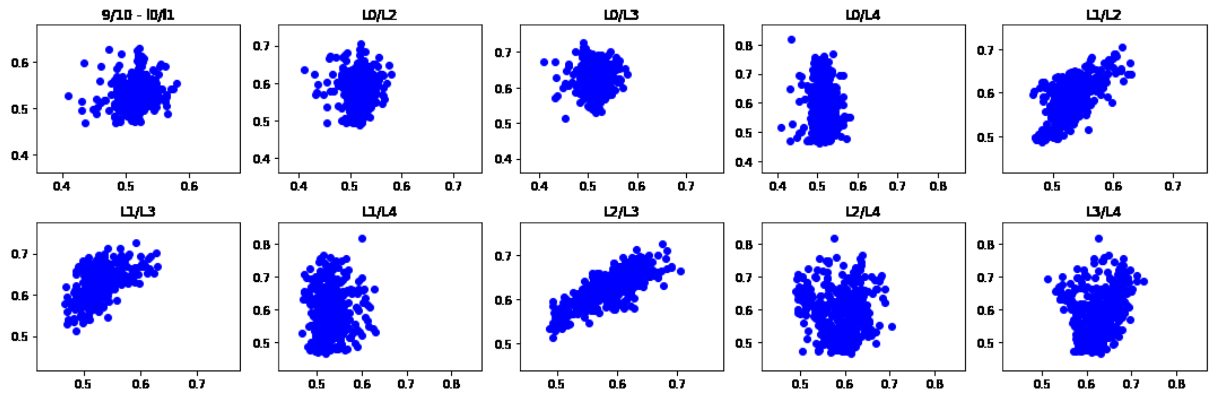


Figure 5.15: *Correlation of 2 benign models when the attack is performed at round 0. The distances are computed on the normal dataset.*

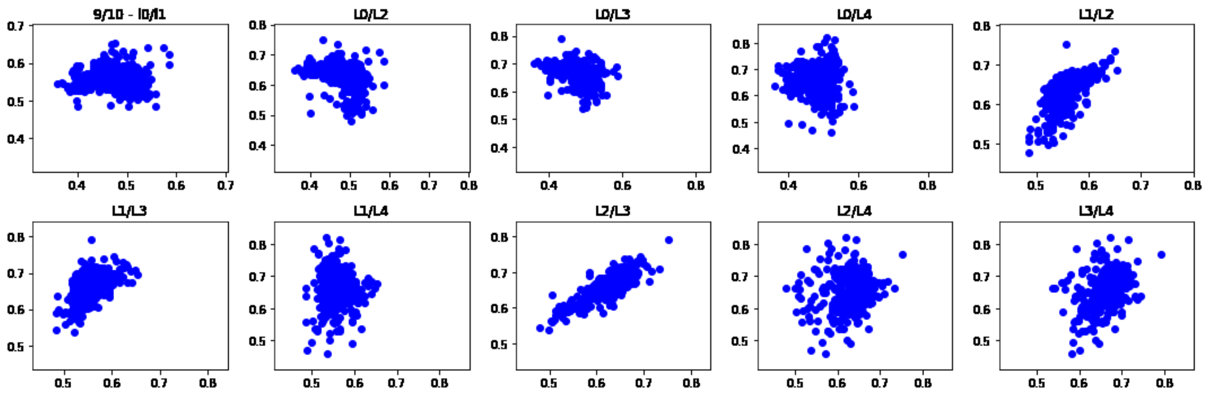


Figure 5.16: *Correlation of 2 benign models when the attack is performed at round 0. The distances are computed on the poisoned dataset.*

At the beginning of the training, the linear correlation among *L2/L3* remains identical to the *no-attack* scenario (*Figure 5.12*). The layers show linearity on *L1/L2* however only a couple of models have the correlation there, for most of the others the correlation is similar to *L0/L2* or *L1/L4* in *Figure 5.16* (a sort of ball). Moreover, the correlation is not sensible to the type of dataset since for *normal* (*Figure 5.15*) and *poison* (*Figure 5.16*) we obtain the same results.

For evil-benign the correlation changes as reported in the *Figure 5.17* and *5.18*. In both cases, with *normal* (*Figure 5.17*) and *poisoned* datasets (*Figure 5.18*), the correlation on *L2/L3* disappears. All correlation with *L4* shows 2 different clusters, one with poison images and the other with not, this happened also for evil-evil models as reported in the *Figure 5.20* below.

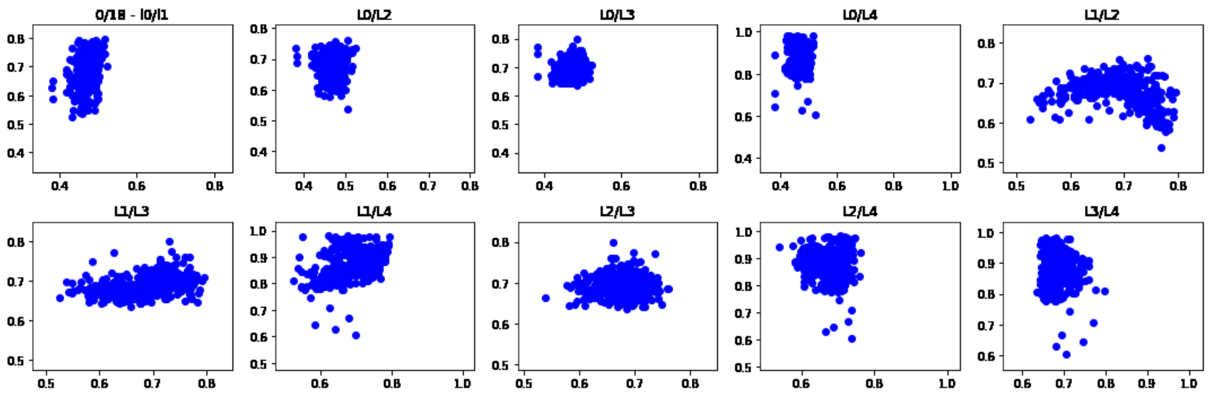


Figure 5.17: *Correlation of benign - evil models when the attack is performed at round 0. The distances are computed on the normal dataset.*

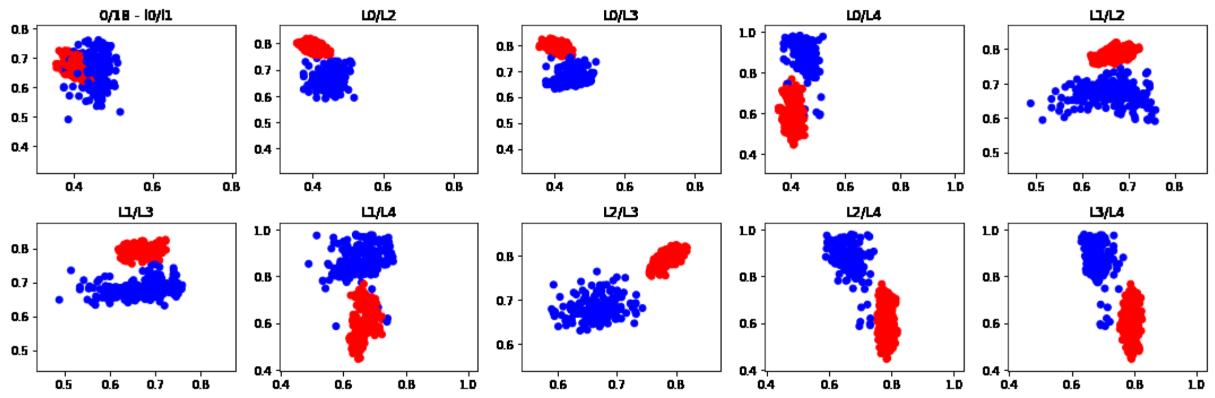


Figure 5.18: *Correlation of benign - evil models when the attack is performed at round 0. The distances are computed on the poisoned dataset. Red dots represent poisoned images.*

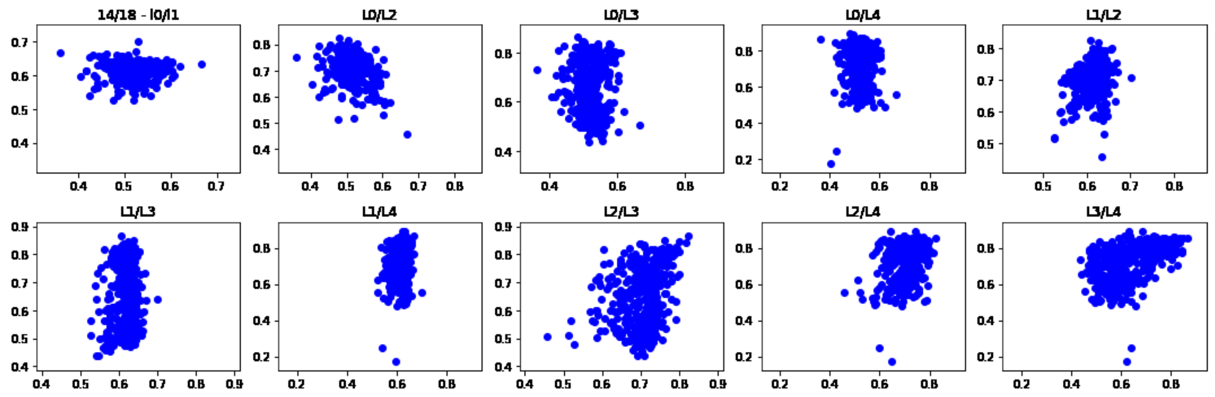


Figure 5.19: *Correlation of evil - evil models when the attack is performed at round 0. The distances are computed on the normal dataset.*

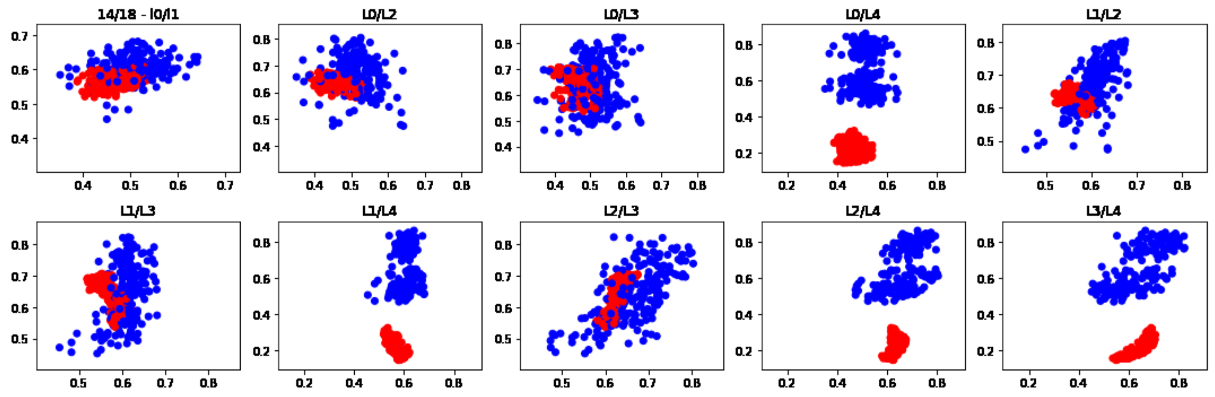


Figure 5.20: *Correlation of evil - evil models when the attack is performed at round 0. The distances are computed on the poisoned dataset. Red dots represent poisoned images.*

The most difference between *benign-evil* and *evil-evil* is that the points are more spread and also the distances for poison images (red dots) are smaller along the y-axis (considering only the graph with *Layer4*), in fact for *evil-evil* (*Figure 5.19*) are below 0.4 instead with benign-evil (*Figure 5.18*) stand above.

Summary when an attack is performed at the beginning of the training there are 2 possible ways to detect evil models by simply analyzing the correlation among *L3/L4* and verifying if 2 clusters are present. If a model hasn't got linearity on *L3/L4* or has 2 clusters for each graph with *Layer4* this is an evil model. Unfortunately, what we have shown and highlighted until now doesn't happen for all couple of *evil-evil* or *benign-evil* models. *Figure 5.21* is an example of that where *L2/L3* shows linear correlation, and no 2 clusters are present.

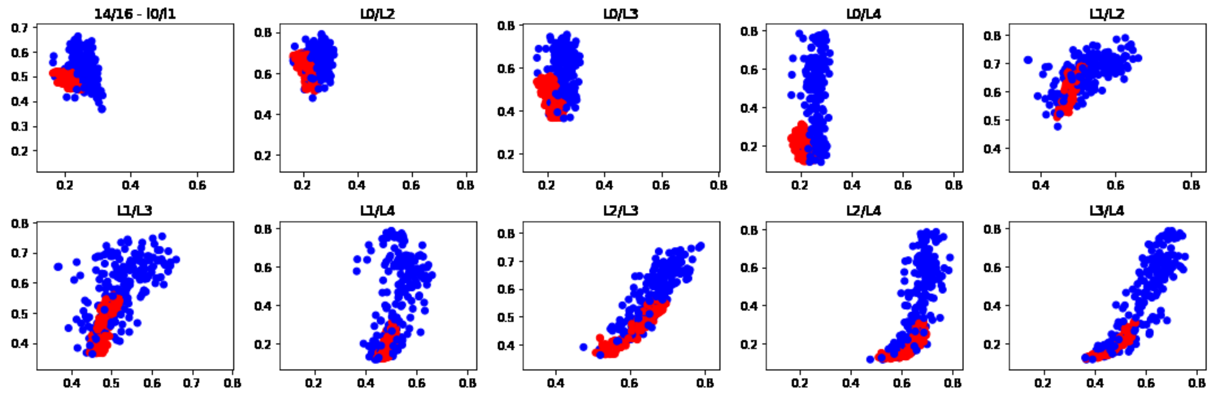


Figure 5.21: *Correlation of evil - evil models when the attack is performed at round 0. The distances are computed on the poisoned dataset. Red dots represent poisoned images.*

By this analysis of the first round, we see the impossibility to detect evil models by observing the correlation among the layers. In the next section, we perform the same analysis but when an attack is conducted at 1000th round.

Attack – Round 1000

After 1000 rounds of training benign models has the same correlation observed in *Figure 5.14*. What is important is to verify what happened on *evil-evil* and *benign-evil* models. *Figure 5.22-5.23* show how correlation change for *benign-evil* models.

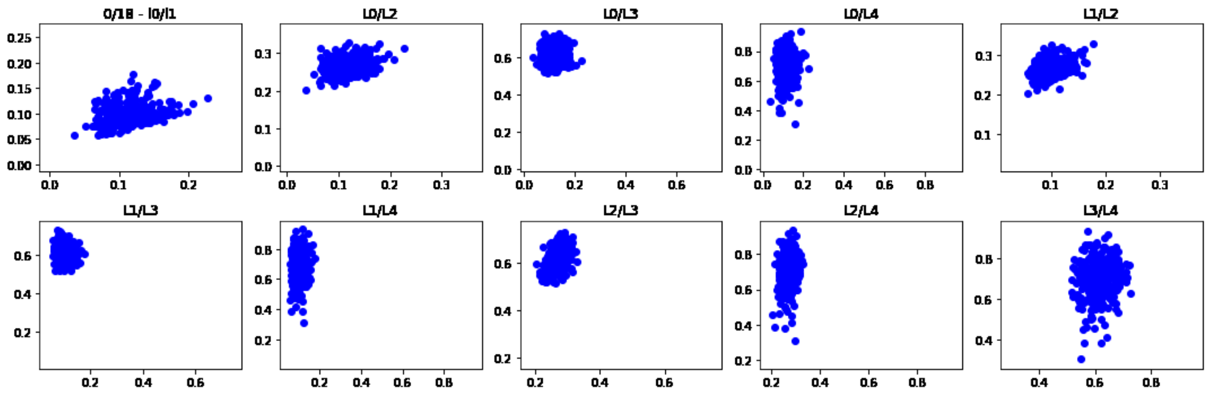


Figure 5.22: *Correlation of benign - evil models when the attack is performed at round 1000. The distances are computed on the normal dataset.*

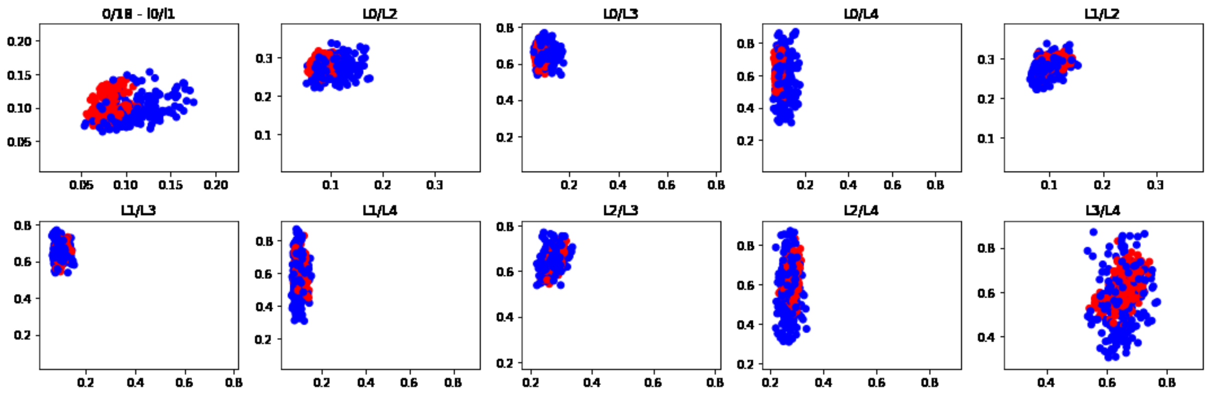


Figure 5.23: *Correlation of benign - evil models when the attack is performed at round 1000. The distances are computed on the poisoned dataset. Red dots represent poisoned images.*

For both the *normal* dataset (*Figure 5.22*) and *poisoned* dataset (*Figure 5.23*) the correlation among layers is similar to *benign-benign* (*Figure 5.14*) with a vertical column for all layers with *Layer4* and no correlation in another case. The red points (*Figure 5.23*) are not more separated from the other points. The same situation happened for *evil-evil* reported in *Figure 5.24-5.25*.

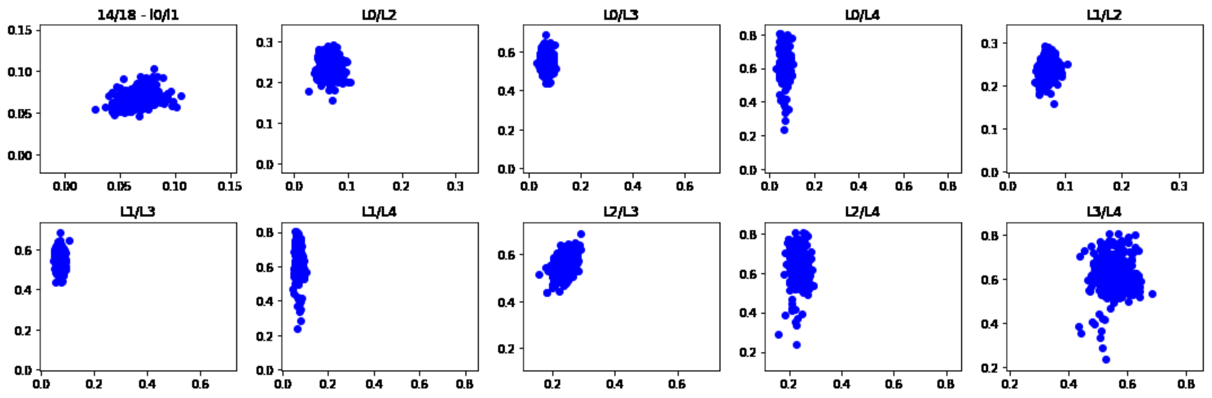


Figure 5.24: *Correlation of evil - evil models when the attack is performed at round 1000. The distances are computed on the normal dataset.*

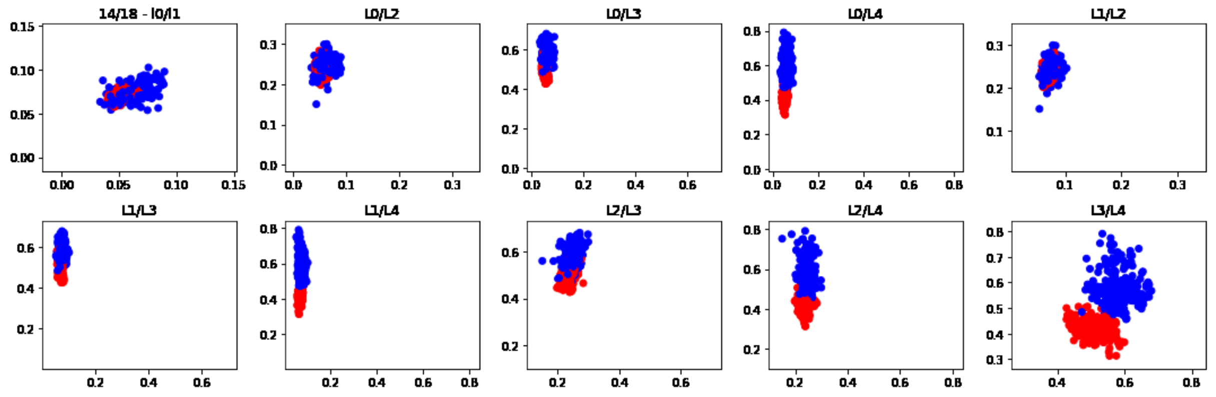


Figure 5.25: *Correlation of evil - evil models when the attack is performed at round 1000. The distances are computed on the poisoned dataset. Red dots represent poisoned images.*

However *evil-evil* and *benign-evil* have something different from *benign-benign*. Observing the correlation on *L3/L4* it has a different shape. The shape (*Figure 5.26*) of *benign-benign* is similar to a vertical column while for the other 2 is more similar to a ball.

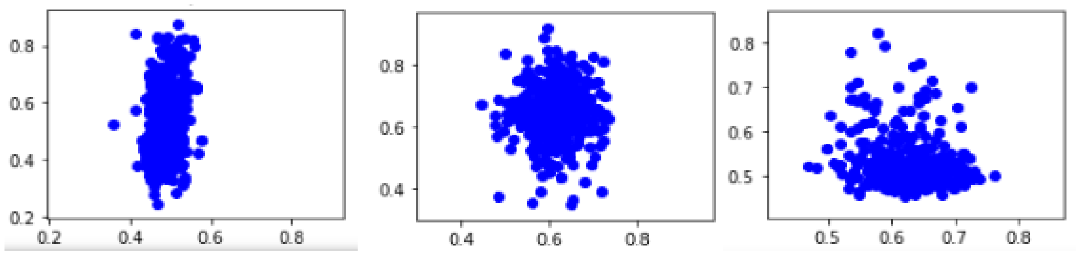


Figure 5.26: *Correlation among Layer3/Layer4. Benign-Benign models (left), benign-evil (center) and evil-evil (right).*

Furthermore, these patterns are independent of the type of dataset (normal or poisoned) but, unfortunately, only at round 1000 the correlation assumes that configurations because, if the attack starts before models converged, the shape converged in a unique pattern.
Suppose the attack starts 50 rounds before, the new correlation at round 1000th between layer *L3/L4* becomes closer to the *benign-benign* shape.

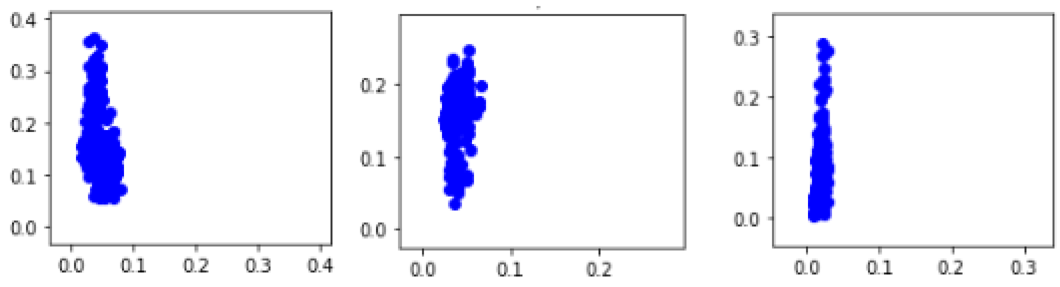


Figure 5.27: *Correlation among Layer3/Layer4. Benign-Benign models (left), benign-evil (center) and evil-evil (right) after 50 consecutive attacks.*

This problem subsists when the attack is performed for the first time and *p* over *N* models become infected but the remaining ones don't, after the aggregation the global model contains the backdoor, then Server generates *N* copies which are sent to *N* clients. The evil clients continue to maintain the backdoor inside making it more strong with respect to the previous round instead benign destroying the backdoor, however, this time, after the aggregation, the global model will contain a stronger backdoor. In the next round again evil strengthens the backdoor instead of benign try to delete it but without succeeding because it has been strengthened in the previous round. Continuing the attacks for more rounds make the backdoor always more difficult to destroy and if the benign cannot delete it they become themselves evil with the result that all models become infected. At this point is not possible to distinguish *evil-evil*, *benign-benign*, and *benign-evil* because all of them are *evil-evil*.

Analysis with global model – Attack Round 0

In the previous analysis, the correlation among models didn't get any sufficient results or information to detect poison models. In the following section, we report results when the distances are computed with the global model and not among the models.
Since we assume the global model to be benign, all results shown in *No Attack* can be extended also with the global model, so we skip this part and show directly what happens under attack.

The *Figure 5.28-5.29* show correlation for benign models on the *normal* (top) and the *poisoned* (bottom) dataset.

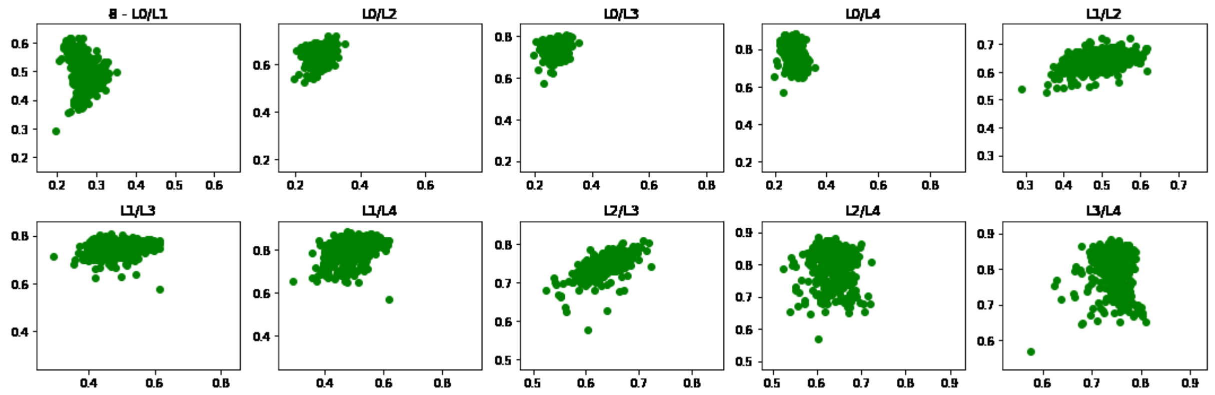


Figure 5.28: *Correlation of benign models when the attack is performed at round 0. The distances are computed on the normal dataset.*

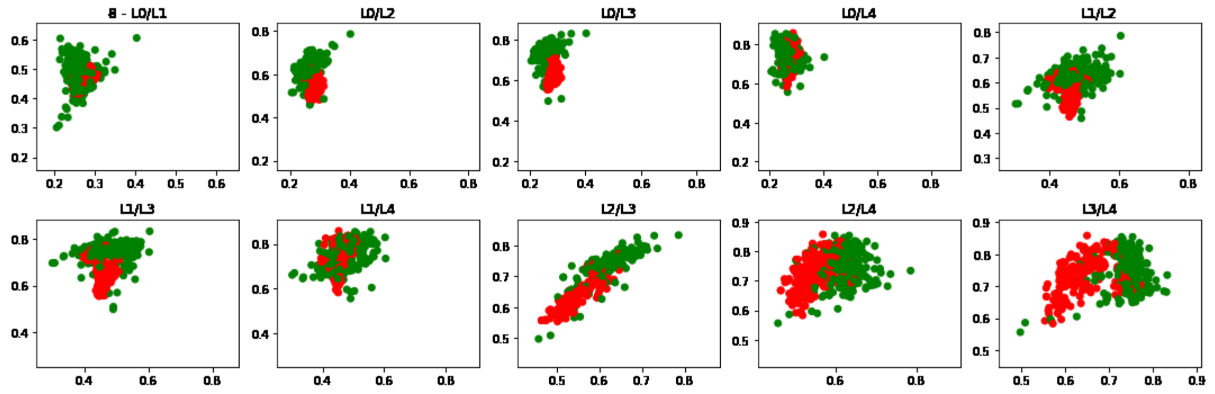


Figure 5.29: *Correlation of benign models when the attack is performed at round 0. The distances are computed on the poisoned dataset.*

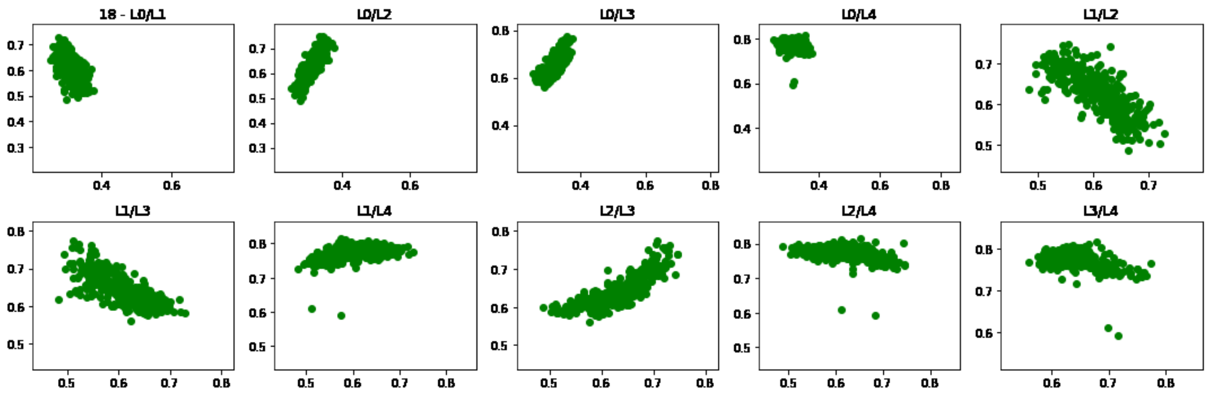


Figure 5.30: *Correlation of evil models when the attack is performed at round 0. The distances are computed on the normal dataset.*

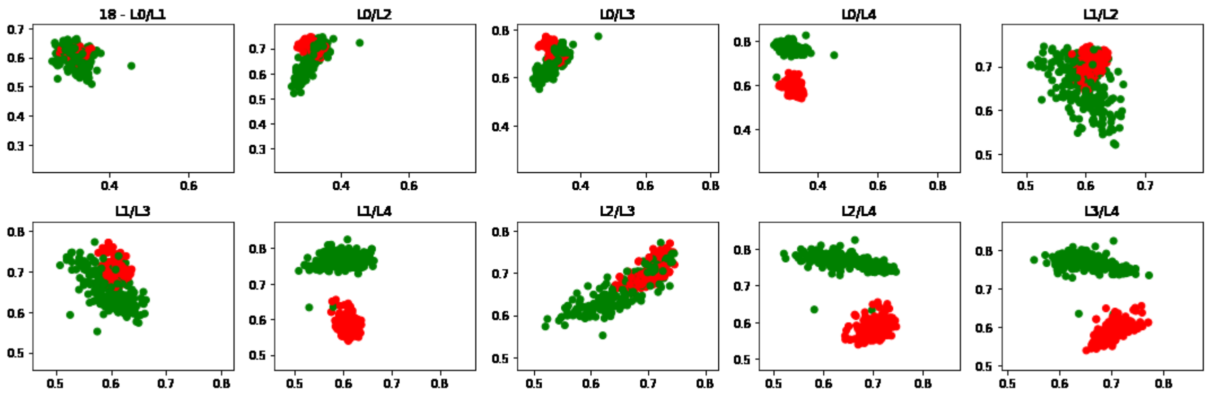


Figure 5.31: *Correlation of evil models when the attack is performed at round 0. The distances are computed on the poisoned dataset.*

Again, there is a linear correlation between *Layer2* and *Layer3* in all cases, evil and benign with both types of datasets. For *the normal* dataset *L1/L2 and L1/L3* show antilinearity, while *L0/L2* and *L0/L3* are linear. For the *poisoned* there is only linearity on *L2/L3* and the rest of the layer doesn't show correlation, except for *L1/L2*, *L1/L3,* and *L0/L2*, *L0/L3* which is similar to the *normal* dataset despite being less strong. However, these correlations are not common for all evil models, some of them are similar to the benign ones making them impossible to distinguish. With *poison* dataset correlation with *Layer4* shows that distances computed on the *target* class (red dots) are separated from the other.

Analysis with global model – Attack Round 1000

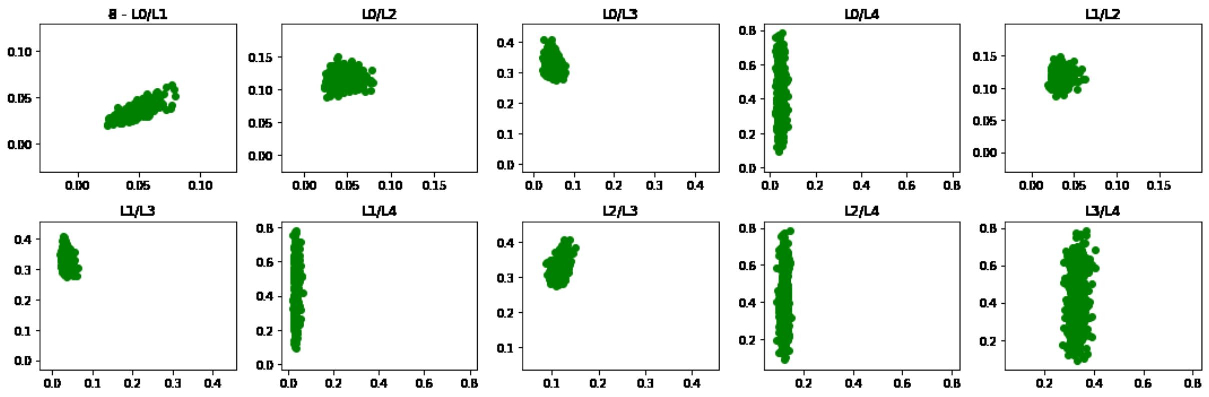


Figure 5.32: *Correlation of benign models when the attack is performed at round 1000th . The distances are computed on the normal dataset.*

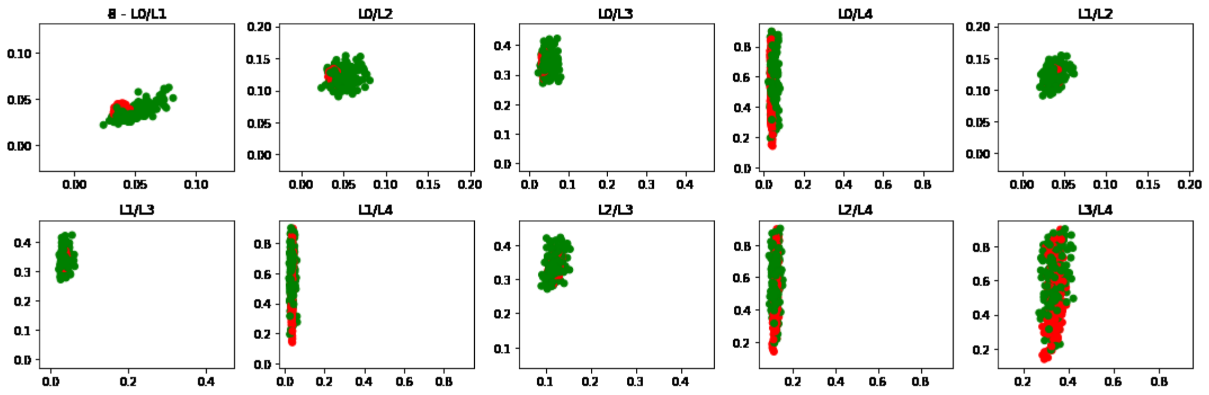


Figure 5.33: *Correlation of benign models when the attack is performed at round 1000th . The distances are computed on the poisoned dataset.*

With benign (*Figures 5.32, 5.33*) the linear correlation disappears for *L2/L3* and for other layers no correlation is presented. *L0/L1* also get some linear correlation but only for some models and not for all. All layers with *Layer4* show a vertical column.
With evil (*Figures 5.34, 5.35*) there is no correlation for all couple of layers, the layers with *Layer4* show a vertical column except for *L3/L4* where instead the pattern has a different shape or reveals 2 clusters. However, this doesn't happen when the attacks are performed for consecutive rounds (*Figure 5.36*). In this case, the previous global model is not benign but is infected and, for the same reason explained in the previous section, all the models become infected, therefore, for each couple of layers we have the same pattern making it unable to distinguish which one is benign or not.

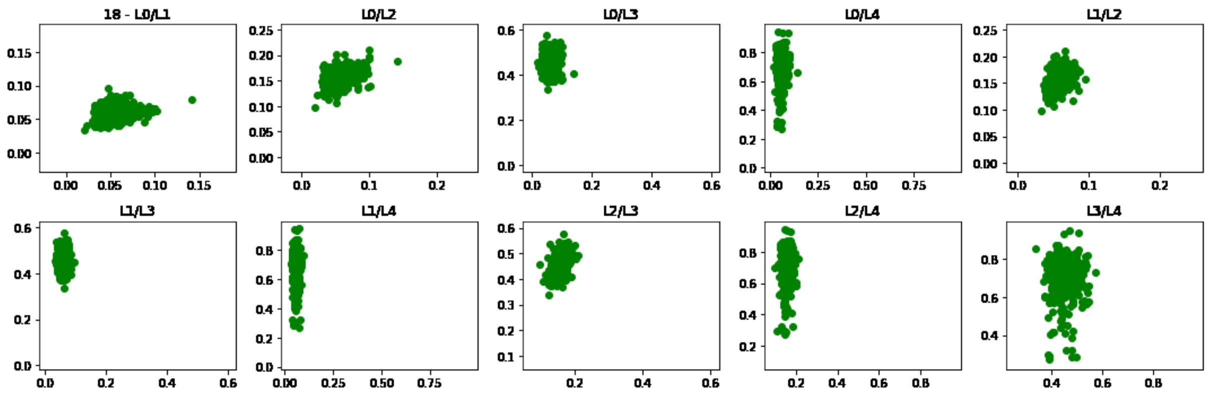


Figure 5.34: *Correlation of evil models when the attack is performed at round 1000th . The distances are computed on the normal dataset.*

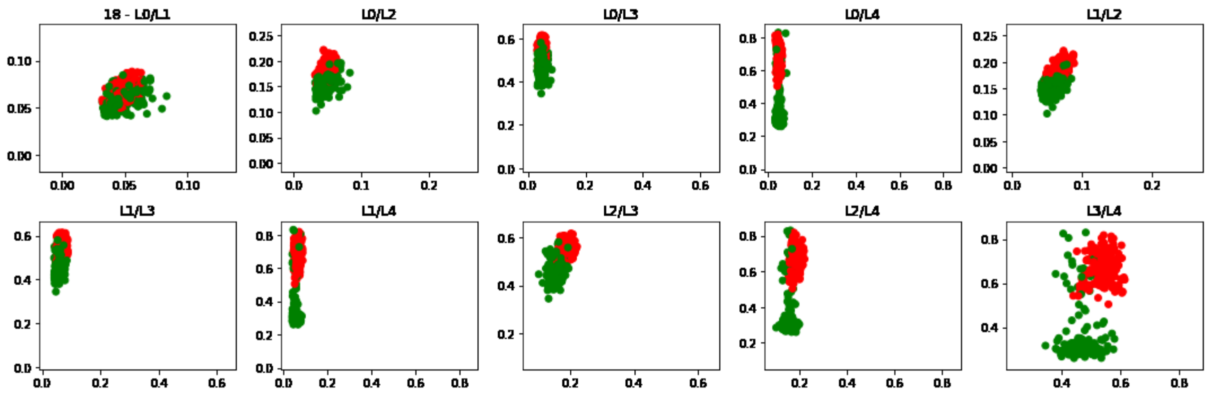


Figure 5.35: *Correlation of evil models when the attack is performed at round 1000th . The distances are computed on the poisoned dataset.*

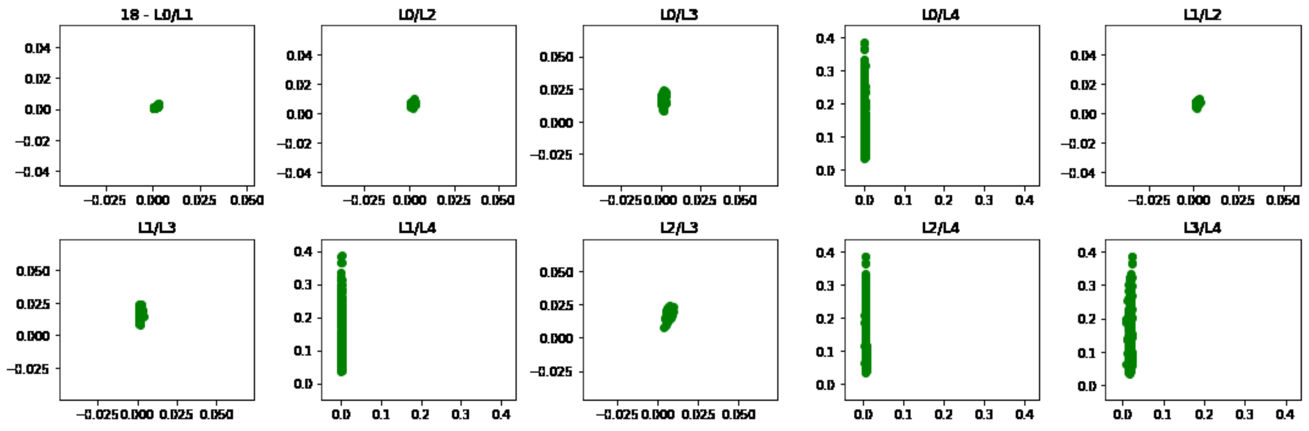


Figure 5.36: *Correlation of evil models when the attack is performed for 50 consecutive rounds.*

In summary, we see how correlation appears with and without attack. First, we have analyzed distances among models and we have seen something that is possible to exploit for separating evil from benign but, however, this happens only in specific conditions. Then we have conducted the same analysis with distances computed with the global model and the results are the same.

5.4 Neuron Analysis

5.4.1 Purpose

In the previous analysis, we have only investigated the output from the main layers in the architecture and we have shown possible directions and the problems related to them. In this section, we investigate another possible solution by, instead of finding something for detecting evil models, imposing that the evil model behaves as a benign model. The idea is to modify the architecture of the model by following 2 rules:

1. *The normal behavior must remain unchanged*
2. *The backdoor, if present, must be blocked*

These rules are obvious because is the task of all backdoor defenders, but the problem is how to satisfy them.

When a model processes an image not all neurons inside the architecture participate in the classification task, so not all of them fire, these types of neurons are called *non-active* neurons. A backdoor model, when taken in input a normal image (without a *trigger* inside), has some neurons that don't fire, instead of when the same image with the *trigger* is processed neurons that didn't fire before, now fire [6]. A backdoor model could also misclassify a poisoned image only triggering one single neuron that makes the final prediction wrong [7].

5.4.2 Methodology

To investigate this fact the idea is to seek the *non-active* neurons when normal images are processed and to put a mask on them in order to be blocked when a poison image is given. To do that we use a binary mask that multiplies the output of layer *l* and returns only the output of *active* neurons, formally:

$$O'_l = O_l * MASK_l$$

(5.1)

where O'_l is the output of *active* neurons, O_l is the output from layer *l* and $MASK_l$ is a tensor of the same shape of O_l with 0 for *non-active* neurons and 1 for *active*.

To compute a binary mask we need to identify *non-active* neurons placed on each layer. Given a clean dataset *D* and a model *M*, not important if is evil or not, we compute the average output of neuron *i* of layer *l* over all input images $d \in D$ so:

$$O_{i,l} = \frac{\sum_{d \in D} O_{i,l,d}}{|D|}$$

(5.2)

where $O_{i,l,d}$ is the output of neuorn *i* in the layer *l* with input image *d* and $O_{i,l}$ is the average over all images. Now if $O_{i,l}$ is below a threshold ε (very small value) then this neuron is considered *non-active*. The ε is different for each layer *l* so we first have to find the set ε = $\{\varepsilon_1, \varepsilon_2, \varepsilon_3, \ldots., \varepsilon_L\}$ where *L* is the number of layers of model *M*.

Each ε_i is estimated by plotting the hystograms of all $O_{i,l}$ with *i* = {*1, 2, 3, ..., n*} where *n* is the number of neurons per layer, divided in 60 *bins,* the lowest *bins* is ε_i. As shown in *Figure 5.37* each lower *bins* is close to zero, in the first layer it also the one with higher frequency while in the other the frequancy is below 0.1.

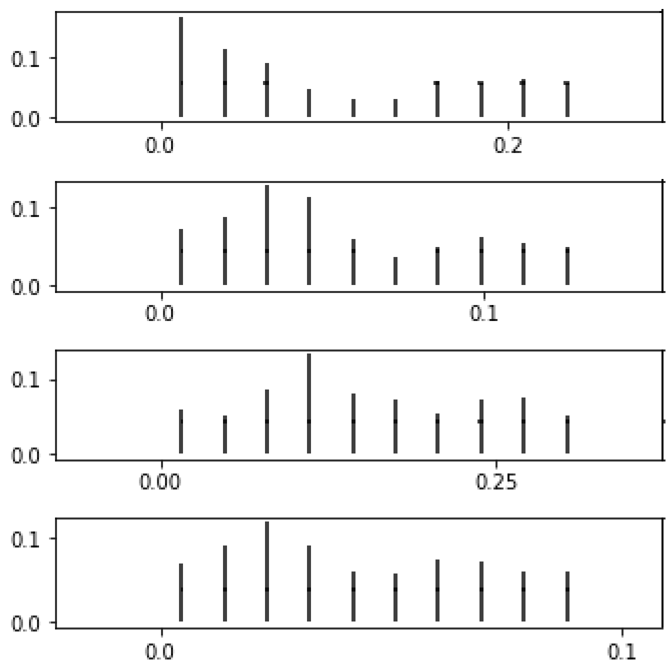


Figure 5.37: *First 10 bins of first 4 layer from ResNet18.*

Once the ε set is computed we generate the binary mask $MASK_l$:

$$f(O_{i,l}) = \begin{Bmatrix} 0 & \text{if } O_{i,l} \le \varepsilon_l \\ 1 & \text{if } O_{i,l} > \varepsilon_l \end{Bmatrix} \tag{5.3}$$

To have a correct *Mask* is necessary that our models have been trained for a lot of rounds and at least the accuracy must be over 0.8, in order to have significant outputs. Due to this fact, the whole analysis is conducted when the attack is performed at round 1000th . With respect to the previous case, this approach is not sensitive to the number of consecutive attacks before models converged because in our assumption the trigger neurons stand in a specific part of the network, if a consecutive attack is carried out, they don't change, and so *non-active* neurons found at the first attack or after *n* attacks are the same.

In the next 2 sections, we show how Backdoor Accuracy (BA) and Main Task Accuracy (MA) change when the mask is applied. Backdoor Accuracy is the percentage of how many poison images a correctly misclassified as the *target* class. Main Task Accuracy is the percentage on how many images (not poisoned) get the right prediction. The sections report a comparison

between 2 architectures, *ResNet18* (the one used in the previous analysis) and a simple *CNN + FCL* (Convolutional Neural Network + FullyConnected Layers) running on a *MNist* dataset, in order to highlight some problems related to this approach.
The mask is applied after each *ReLU* layers.
We use the *pixel-backdoor* attack.
We setted $N = 30$, $p = 15$, $lr\_e = 0.025$, $lr\_b = 0.1$, $E\_b = 2$ and $E\_e = 15$.
We consider *IID* and *non-IID* data distribution.

5.4.3 Results and Analysis

Backdoor Accuracy Comparison

The *Figure 5.38* shows BA for *ResNet18* and *CNN+FCL* for *IID* data.

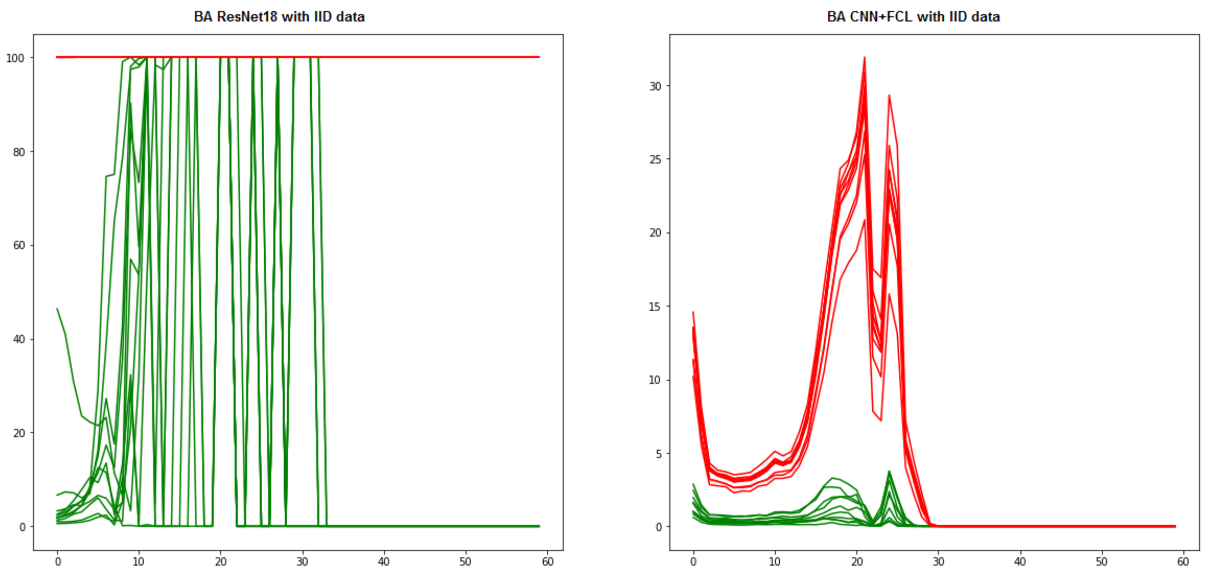


Figure 5.38: *Backdoor Accuracy (BA) with IID data. The left figure shows BA for each model with ResNet18, right instead for CNN+FCL. The x-axis represents the number of the bin used for computing the thresholds. The green lines are the benign models and the red lines are the evils.*

With *ResNet18* for each *bin* (threshold) the evil models remain higher at 100 % instead of benign shows some oscillations. With *CNN+FCL* instead, the results are better, for a specific threshold, at the 5th *bin* all BA is below 5 % and the maximum (at the 20th bin) is below 40 %, after the 30th however all neurons turn off with the results that the prediction for each poison image is zero and consequentially the BA drop to zero.

The next figure (*Figure 5.39*) reports the same analysis but with *non-IID* data.

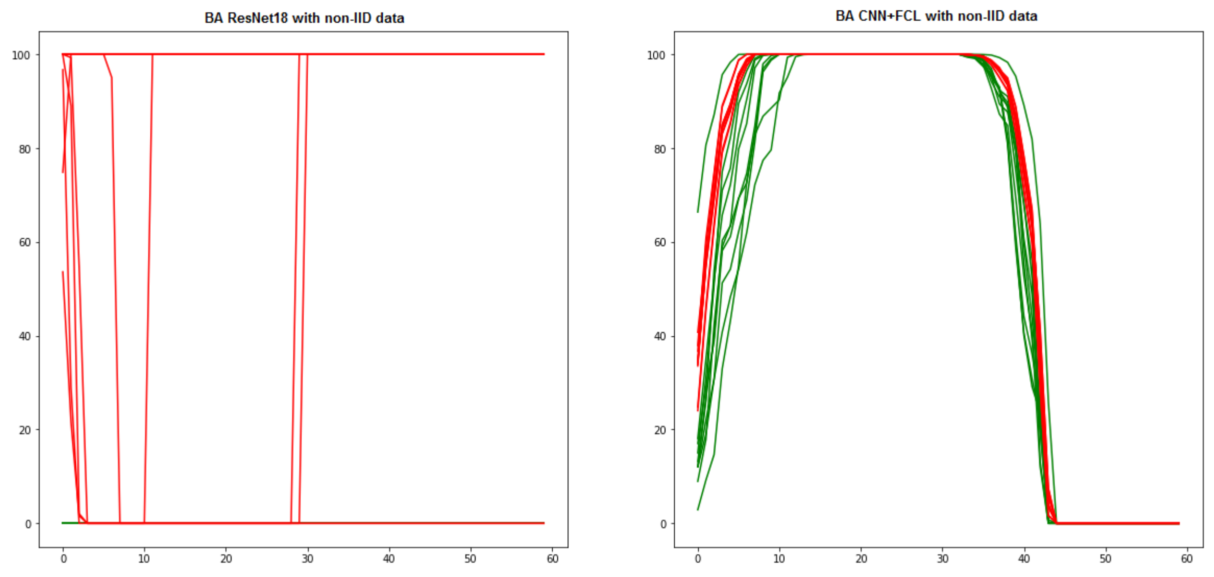


Figure 5.39: *Backdoor Accuracy (BA) with non - IID data. The left figure shows BA for each model with ResNet18, right instead for CNN+FCL.*

For *ResNet18* the results are similar to *IID*, however, some evil models at earlier bins have a BA close to 0.

For *CNN+FCL* the BA is reduced only at the first *bin,* the maximum is below 65 %, and surprisingly is benign which has a higher value. This depends on the fact that the mask applied to that particular model reduces also the MA and so the model gets random predictions so when it has as input poison images it randomly predicts them as *target* class. However, the maximum BA is 65 % and so is not a risk. With other bins instead, BA remains higher however this happens because after a specific threshold (5th bin) almost all neurons turn off, and, given an image the prediction is always the same, unfortunately, in this case, is always *zero* which is also the *target* class.

Main Task Accuracy Comparison

The *Figure 5.40* show MA with *IID* data.

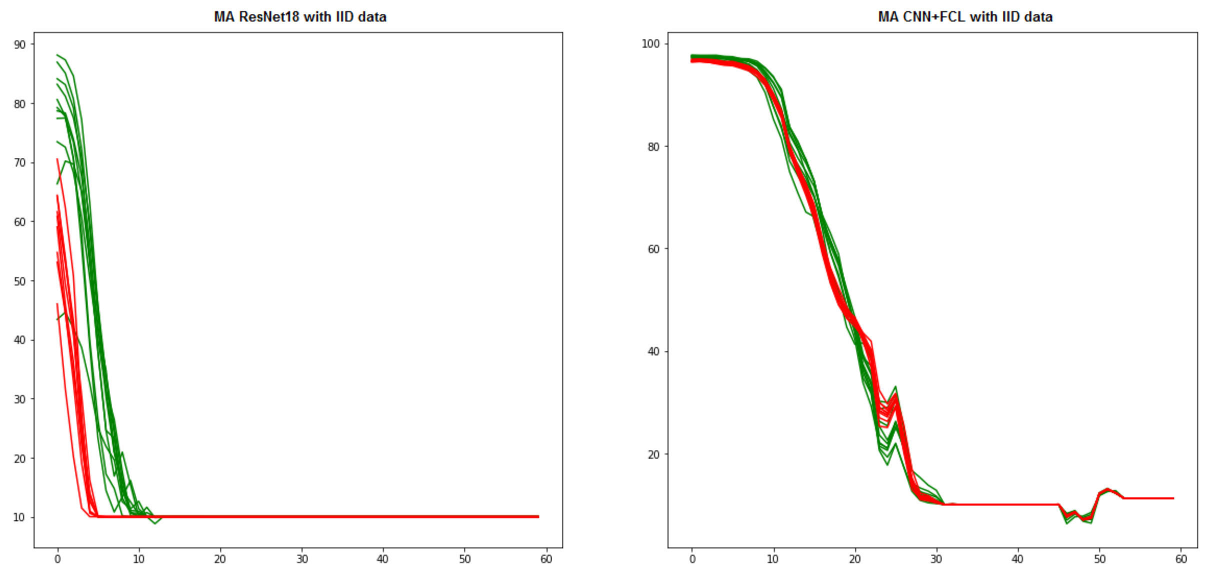


Figure 5.40: *Main Task Accuracy (MA) with IID data. The left figure shows MA for each model with ResNet18, right instead for CNN+FCL.*

Increasing the threshold, the *ResNet18* goes to zero faster than *CNN+FCL*. As we have explained in the previous section increasing the threshold turns off a lot of neurons inside which makes the prediction always the same, in this case, zero. The MA traces how many images per class are right predicted and if all of them get zero the MA becomes 0. The *Figure 5.41* shows MA when data are *non-IID*.

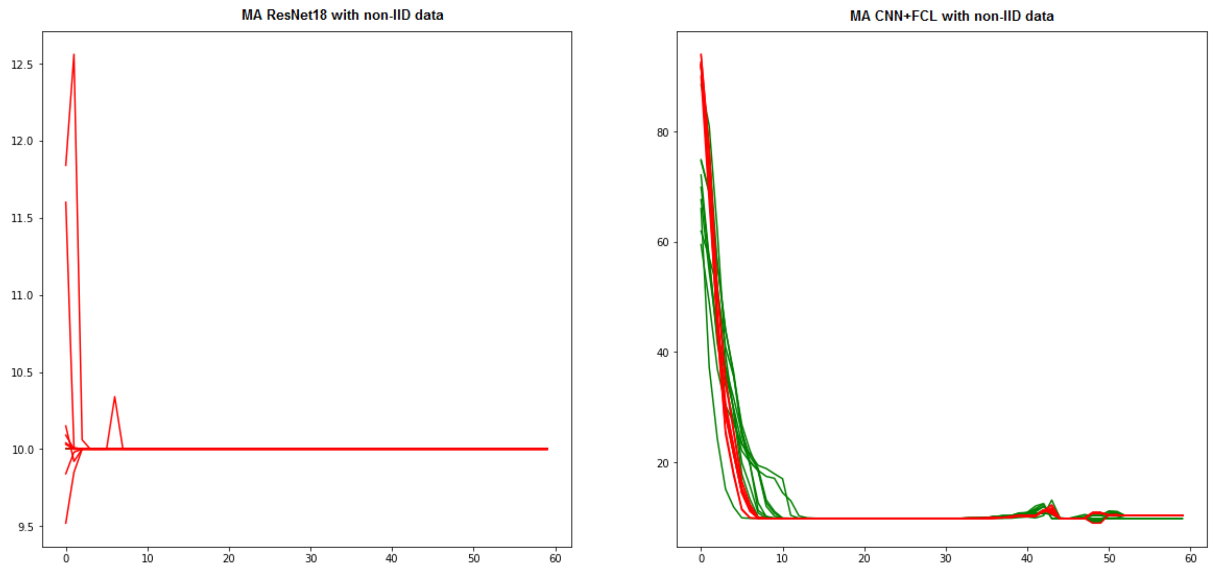


Figure 5.41: *Main Task Accuracy (MA) with non - IID data. Left figure shows BA for each model with ResNet18, right instead for CNN+FCL.*

With *ResNet18* the results are very bad, using the first bin as the threshold the MA drops down to 12.5 %, and consequently, increasing the threshold, the MA decreases. For *CNN+FCL* the results are much better, with the first bin the MA remains at 80 % but then decreases as the bins increase.
From this comparison we deduce that the Mask can block the backdoor attacks only for *CNN+FCL* and not for *ResNet18*. The main difference stands in the architecture, *ResNet* is very more complex (*Figure 2.6*) than simple *CNN+FCL* (*Figure 2.7*), it contains more layers and some non-canonical connections (Residual Connections). Therefore, most likely, the trigger neurons are sparse in the network making it challenging to detect and deactivate them. A possible solution could be trying to activate each subset manually until you notice the prediction is wrong, however, this is computationally impossible.

We try also other different approaches for building an efficacy mask which are listed below:

1. we apply the mask on the last layers because usually, the trigger neurons stand there since it's easy to propagate their signals
2. we select ε_i as the rarer values which are the bins with very low frequency because we assume that those neurons, maybe, are the trigger
3. we select bins randomly

None of these solutions change the results.

6. CONCLUSION

In our work we have shown that *Federate Learning* gives the possibility to a group of users to collaboratively participate in the model training without sharing their own data. Doing that, it guarantees that the privacy of the data is maintained but consequently exposes the final model to other vulnerabilities like *backdoor attack*, which makes the model prediction on the crafted-attacker inputs as the attacker wanted. Over the years a lot of defense strategies have been implemented as countermeasures against such attacks but however, none of them successfully works. The *Krum* [15], *Bulyan* [16], and *Trimmed Mean* [17] operate by modifying the aggregation function in order to discard the evil models, *Auror*[19], *FoolsGold* [23], and *FLGuard* [21] detect the evil models by using clustering techniques. The *BaFFLe* [22] exploits the performance on a real dataset to infer if the global model is backdoored or not, unfortunately, these methods have some drawbacks which make them impossible to be applied in a real scenario. In our work, we propose *TEE* (Trust Execution Environment) to evaluate the models using the *local dataset* in a secure enclave without destroying the *privacy* and we propose some possible directions for detecting malicious updates by analyzing the output of each single model's layer given input the *local dataset*. In *Section 5.1* we have analyzed if it is possible to separate the evil clients from benign by inspecting the output from the last layers of the model, this works if the data distribution is *IID* and fails in the other case, moreover the detection doesn't work if it starts at the beginning of the training, in *Section 5.2* we have analyzed the cosine distance among output per layer but this approach has shown some issues, especially because, based on the type of the dataset used (poisoned or normal), the results are different, in *Section 5.3* we have introduced a method based on correlation analysis among the layer and we saw that the correlation with distances computed with previous global models has demonstrated some potential but fail when a consecutive attack is performed and moreover works only when the training is at the final rounds. In *Section 5.4* we have changed the paradigm, instead of analyzing the output per layer we put a mask in front of each *ReLU* layer, the masks are computed by allowing to fire only the neurons activated when a normal dataset (dataset without poisoned images) is used. In theory, when a poison image is given in input, the masks block the trigger neurons and allow to pass only normal neurons thus the backdoor will be deactivated. Such approach has demonstrated efficacy only if we use a simple architecture like *CNN + FCL* but instead fails when a more complex like *ResNet18* is used.

Future works will start from the idea that only looking at the output per layer is not a good and promising solution but, however, some possible more sophisticated approaches could be

investigated, in our work we only observed the distances and the correlations but something more advance could be done.
Another possible direction could integrate the explainability of AI into the detection process. Imposing that *TEE* is not only used for evaluating the models but also for the training process permits that the attacker cannot touch the architecture during the training and make the backdoor attack more difficult to inject. Unfortunately, in [3] is also explained how to perform an attack under this condition, which is called a *black-box* attack, in practice, the poison images used by the attacker have a very low frequency (so only the attacker has these images) and so the architecture learns how to predict wrongly these images. However, if we also force that the *local dataset*, once is inserted inside the *TEE*, cannot be modified by the attacker we can use the explainability of AI to detect the poison dataset. Since the attack works only because the poison images don't belong to other users [3], trying to understand how a model predicts the more frequency images against the low frequency could be exploited for detecting the poison dataset and discarding it for the next training rounds.

.

BIBLIOGRAFY

[1] Claudia Malzer and Marcus Baum , *“A Hybrid Approach To Hierarchical Density-based Cluster Selection”*

[2] Martin Ester, Hans-Peter Kriegel, Jiirg Sander, Xiaowei Xu, *“A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise”*. Institute for Computer Science, University of Munich Oettingenstr. 67, D-80538 Miinchen, Germany {ester I kriegel I sander I xwxu } @informatik.uni-muenchen.

[3] Hongyi Wangw, Kartik Sreenivasanw, Shashank Rajputw, Harit Vishwakarmaw, Saurabh Agarwalw Jy-yong Sohnk, Kangwook Leew, Dimitris Papailiopoulos*, “Attack of the Tails: Yes, You Really Can Backdoor Federated Learning”.* University of Wisconsin-Madison arXiv:2007.05084v1 [cs.LG] 9 Jul 2020 k Korea Advanced Institute of Science and Technology.

[4] Kolouri, Soheil, et al. *“Optimal Mass Transport: Signal processing and machine-learning applications”*. IEEE Signal Processing Magazine 34.4 (2017): 43-59.

[5] Villani, Cedric. Topics in optimal transportation. No. 58. American Mathematical Soc., 2003.

[6] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. *“Fine-Pruning: Defending Against Backdooring Attacks on Deep Neural Networks”.* New York University, Brooklyn, NY, USA {kang.liu,brendandg,siddharth.garg}@nyu.edu.

[7] Yingqi Liu, Wen-Chuan Lee, Guanhong Tao, Shiqing Ma, Yousra Aafer, Xiangyu Zhang. *“ABS: Scanning Neural Networks for Back-doors by Artificial Brain Stimulation”*. Kingdom. ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3319535.3363216

[8] Clement Fung, Chris J.M. Yoon, Ivan Beschastnikh. *“Mitigating Sybils in Federated Learning Poisoning”.*

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. *“Deep Residual Learning for Image Recognition”*. Microsoft Research, {kahe, v-xiangz, v-shren, jiansun}@microsoft.com

[10] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, Blaise Ag¨uera y Arcas. *"Communication-Efficient Learning of Deep Networks from Decentralized Data"*, Google, Inc., 651 N 34th St., Seattle, WA 98103 USA

[11] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, Vitaly Shmatikov. *"HowToBackdoorFederated Learning"*.

[12] Marco Barreno, Blaine Nelson, Anthony D. Joseph. "The security of machine learning".

[13] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. IACR Cryptol. ePrint Arch. (2016)

[14] Intel. 2013. Software Guard Extensions Programming Reference. Reference no. 329298-001US.

[15] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. "*Machine learning with adversaries: Byzantine tolerant gradient descent*". In NIPS, 2017.

[16] El Mahdi El Mhamdi, Rachid Guerraoui, and Sébastien Rouault. "*The hidden vulnerability of distributed learning in byzantium*". In ICML, 2018.

[17] Dong Yin, Yudong Chen, Kannan Ramchandran, and Peter Bartlett. "*Byzantine-robust distributed learning: Towards optimal statistical rates*". In ICML, 2018.

[18] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, Neil Zhenqiang Gong. "*Local Model Poisoning Attacks to Byzantine-Robust Federated Learning*". ECE Department, The Ohio State University, ECE Department, Duke University fang.841@osu.edu, {xiaoyu.cao, jinyuan.jia, neil.gong}@duke.edu

[19] Shiqi Shen, Shruti Tople, P. Saxena. *"Auror: defending against poisoning attacks in collaborative deep learning systems"*. National University of Singapore {shiqi04, shruti90, prateeks} @comp.nus.edu.sg

[20] Clement Fung, Chris J.M. Yoon, Ivan Beschastnikh. "*The Limitations of Federated Learning in Sybil Settings*". Carnegie Mellon University clementf@andrew.cmu.edu, University of British Columbia yoon@alumni.ubc.ca, University of British Columbia bestchai@cs.ubc.ca

[21] Thien Duc Nguyen, Phillip Rieger, Hossein Yalame, Helen Mollering, Hossein Fereidooni, Samuel Marchal, Markus Miettinen, Azalia Mirhoseini, Ahmad-Reza Sadeghi, Thomas

Schneider, and Shaza Zeitouni. "*FLGUARD: Secure and Private Federated Learning*". System Security Lab, TU Darmstadt, Germany - {ducthien.nguyen, phillip.rieger, hossein.fereidooni, markus.miettinen, ahmad.sadeghi, shaza.zeitouni}@trust.tu-darmstadt.de, Encrypto, TU Darmstadt, Germany - {yalame, moellering, schneider }@encrypto.cs.tu-darmstadt.de 3Aalto University and F-Secure, Finland - samuel.marchal@aalto.fi 4Google, USA - azalia@google.com.

[22] Sebastien Andreina, Giorgia Azzurra Marson, Helen Möllering, Ghassan Karame. "*BaFFLe: Backdoor Detection via Feedback-based Federated Learning*". NEC Labs Europe sebastien.andreina@neclab.eu, NEC Labs Europe giorgia.marson@neclab.eu, ENCRYPTO/TU Darmstadt moellering@encrypto.cs.tu-darmstadt.de, NEC Labs Europe ghassan@karame.org.

[23] Clement Fung, Chris J.M. Yoon, Ivan Beschastnikh. "*Mitigating Sybils in Federated Learning Poisoning*". University of British Columbia clement.fung@alumni.ubc.ca, University of British Columbia yoon@alumni.ubc.ca, University of British Columbia bestchai@cs.ubc.ca.

[24] David Arthur, Sergei Vassilvitskii. *"k-means++: The Advantages of Careful Seeding"*.

[25] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, Jörg Sander." *LOF: Identifying Density-Based Local Outliers*". Institute for Computer Science University of Munich Oettingenstr. 67, D-80538 Munich, Germany { breunig | kriegel | sander } @dbs.informatik.uni-muenchen.de, Department of Computer Science University of British Columbia Vancouver, BC V6T 1Z4 Canada rng@cs.ubc.ca