Politecnico di Torino

Politecnico di Torino

Master's Degree in Computer Engineering

Academic Year 2021/2022

Degree Session of October 2022

AML Security

A comprehensive framework for machine learning attacks

Supervisors: Guido Marchetto
Alessio Sacco

Corporate tutors: Ivan Aimale
Luigi Casciaro

Candidate: Gianluca Mega

Summary

Nowadays artificial intelligence and machine learning are used everywhere thanks to the technological improvements seen in recent times. This brought researchers and also malicious actors to investigate whether it could be possible to attack such algorithms, and the answer was yes indeed. Many attacks and defences have been developed in these years and the field of research is so wide that a new discipline born, the Adversarial Machine Learning. AML is about the threats of machine learning, how to detect them and how to defend. Due to the newness of this field, there isn't such a standard that could help people and companies to deal with these new risky situations so at the moment attackers are on an advantage position. ENISA and NIST, on the other hand, have started proposing documents and reports about AI vulnerabilities and AML concepts signalling that the research community is working hard on these topics. Many libraries and tools, in fact, are being developed to help people from the AI field to secure and test their models and systems but the journey is still long.

This thesis starts from this scenario and its goal is to propose a framework for security assessment and design of ML systems. In particular, the first chapter will be focused on AI, ML and their usages, ending with examples about attacks against them.

Then, in Chapter 2 AML will be presented depicting attacks and defences after a brief discussion about the threat model of these attacks. There will be insights about targets and goals of a malicious action, needed knowledge and possible patterns. Attacks and defences are discussed by analysing one type at once starting with Evasion ones and then going to Oracle and Poisoning.

Chapter 3 will be about the framework itself. The security by design part will discuss best practices about designing secure ML system while the assessment section will propose a workflow for testing the robustness of a model against adversarial attacks.

Some practical examples will be presented in Chapter 4. During this thesis period, a use case for every attack type has been developed to put in practice all the concepts learned. Also for the design part, a notebook has been produced. Ends this essay a conclusive part with final thoughts about what has been presented.

Table of Contents

List of tables

List of figures

Acronyms

AI	Artificial Intelligence
AML	Adversarial Machine Learning
ART	Adversarial Robustness Toolbox
AT	Adversarial Training
CLI	Command Line Interface
CNN	Convolutional Neural Network
DL	Deep Learning
DNN	Deep Neural Network
DP	Differential Privacy
DTP	Differential Training Privacy
EAT	Ensemble Adversarial Training
FGSM	Fast Gradient Sign Method
FC	Fully Connected
FL	Federated Learning
GAN	Generative Adversarial Network
HE	Homomorphic Encryption
kNN	k-Nearest Neighbours
ML	Machine Learning
MLaaS	Machine Learning as a Service
NLP	Natural Language Processing
NN	Neural Network

PGD Projected Gradient Descent

RBAC Role-Based Access Control

RL Reinforcement Learning

SGD Stochastic Gradient Descent

TEE Trusted Execution Environment

VCS Versioning Control System

WER Word Error Rate

Chapter 1
Introduction to AI and its threats

In the last years Artificial Intelligence (AI) has become a very hot topic with many companies trying to use this powerful technology for all kind of tasks and purposes but what is AI and how it can help companies in their business? Is AI really secure or it hides some vulnerabilities? What can organizations and security operators do to ensure that a model is secure to deploy to the public?

Artificial Intelligence, Machine Learning and Deep Learning terms are often misused and confused but there are key differences between them in fact they can be seen as sub-components of each other.

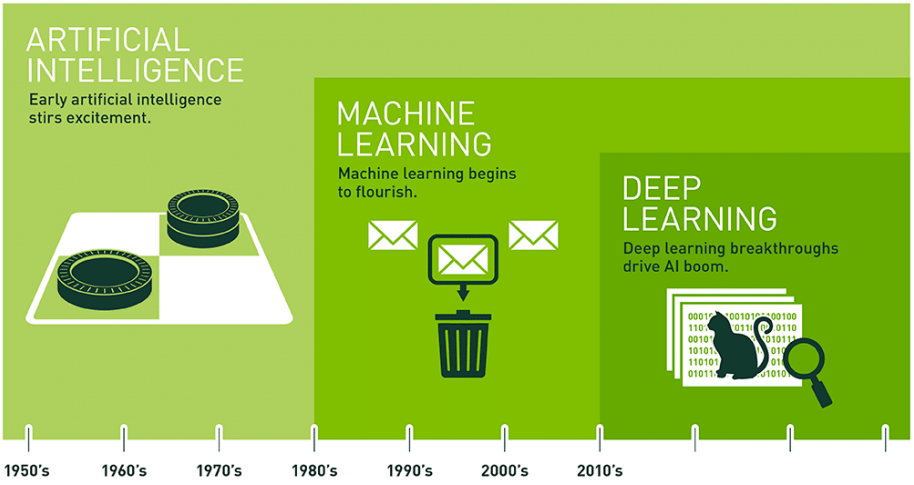

Figure 1.1 - AI, ML and DL from " What's the Difference Between Artificial Intelligence, Machine Learning and Deep Learning?", 2016, blogs.nvidia.com

As shown in Figure 1.1 AI is just the top of the iceberg as it identifies all the technologies that mimic human intelligence [1] and are capable to perform specific tasks with human comparable performance. Machine learning can be seen then as a way to obtain AI capabilities as it is the practice of using algorithms to parse data, learn from it, and then make a prediction about something in the world[2]. ML can be implemented in various ways depending on the specific task and data adopted but surely one of the most used approaches is Deep Learning, which uses an evolution of the standard Neural Networks.

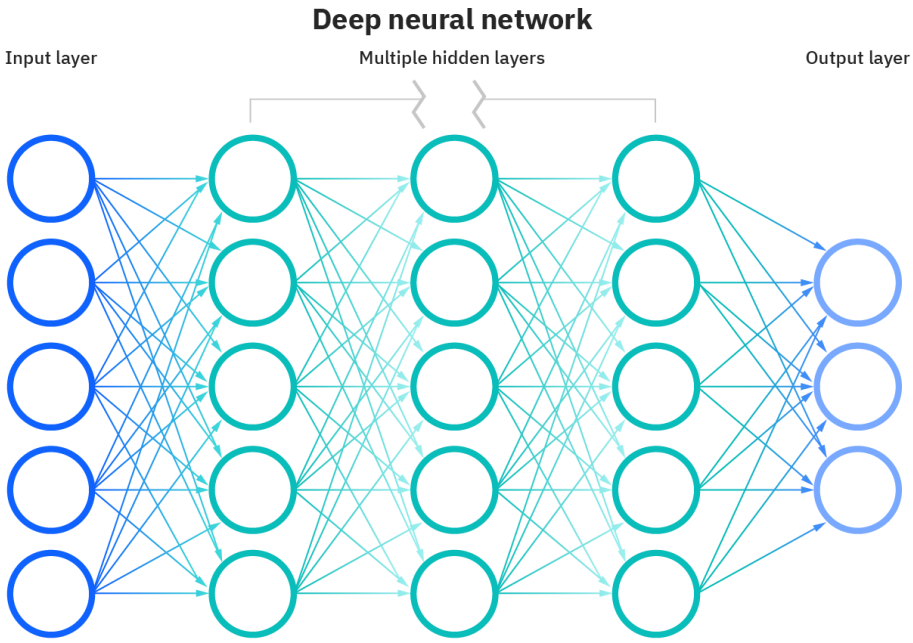

Figure 1.2 - Deep neural network architecture from Cosa sono le reti neurali?, 2020, www.ibm.com

NNs are composed by a series of layers made of computing nodes called “neurons” that process an input using some learned parameters called “weights” in order to produce an output that will be passed to the next neuron; usually when a NN has more than three layers it’s called a Deep Neural Network. NNs and DNNs can, with some limitations and preconditions, approximate a wide set of functions including non-linear ones, the most useful in ML tasks, thanks to their multi-layer composition nature and also to the type of functions used to build these “hidden layers”. Dealing with such non-linearity and complexity is not a simple task in fact this is one of the major bottlenecks of AI. To produce an output the model must be trained which means that the neurons have to

learn their weights using ground truth examples fed to the network in training time but in order to do so an optimization is needed. A special function called “loss” compares the output of the NN with the correct one (the ground truth) and the training algorithm derives it to understand how to modify the parameters to obtain a smaller loss which means a less different network output with respect to the correct one (the model learns from its mistakes). This operation is called “gradient descent” as it can be thought like going from the top of a mountain (high loss) to one of the near valleys (small loss, not the smallest but an enough and near one as global minimum is hardly detectable and, in any case, not so much useful).

AI researches came long time ago but it’s only in recent times that these ideas could be implemented due to the computational power limitations that there were at that time. Technology innovations brought new enthusiasm into the AI and ML world giving researchers all the capabilities and tools they needed to make progresses in these fields. Nowadays AI has reached a good maturity level in many tasks enabling companies to rely on such powerful tools to automate specific processes and also to receive advices and guidance even for crucial strategic decisions. To give an example Google in 2016 has been able to reduce the amount of energy they use for cooling by up to 40 percent using an ensemble of neural networks trained with numerous data (such as temperatures, power, pump speeds, etc.) collected by sensors placed in their data centres[3].

ML is used also to provide new features and capabilities in already available products as a way to enhance their functionalities like for example the automatic recommendation systems that can be found in e-commerce sites or video streaming platforms. These algorithms are AI based as they gather all the choices and habits of the user to create a sort of customer’s profile and use it to predict which products will be more compatible with the actual user’s preferences or to suggest new products to buy.

AI and ML have become also a standalone service with many companies releasing AI-centred products like the famous Personal Assistants (Amazon Alexa to cite one). These models are based on Natural Language Processing neural nets, ML algorithms that analyse user’s speech to understand the meaning and identify potential commands given to the assistant.

Another scenario in which ML could be helpful is threat simulation and cybersecurity trainings. As demonstrated by an experiment conducted by Blue Reply IT, Spike Reply

DE and Machine Learning Reply DE in the scope of this thesis work, it's possible to use an NLP model to craft a phishing mail in the form of a job offer. The attack first scrapes the Linked-In profile of the victim and then uses a text-generation model to write a letter which refers to the scraped information in order to result more personalized and individual. This laboratory shows how realistic these phishing attacks could be, especially when compared to the respective old but working versions (e.g. the famous bank transfer needed by some foreign country Prime Minister) so this could be used for example in employees' security courses to teach people about the sophisticated frauds that could happen. During the thesis period a contribution has been given to this project in terms of code review and infrastructure mounting using technologies like Terraform, AWS and GPT-3.

Despite AI is very helpful and powerful sometimes things can get complicated leading to hardly explainable results, strange errors or non-standard behaviours. This was the case of Amazon which created a set of ML models to automatize their recruiting process using 10 years of submitted resumes. The models were created in 2014 and then integrated in their recruiting process but by 2015 the company realized that its new system was not rating candidates in a gender-neutral way making the company to dismantle the project and switch back to the previous process [4]. Another example is the Inverness Caledonian Thistle F.C. Ball Tracking System, the set of cameras installed by the club which were equipped with an AI ball tracking system that was claimed to be able to follow live actions tracking ball movements across the field. The system was deployed after some tests during the 2020 SPFL Championship but it could not realize the success it had during the test runs. The system repeatedly confused the ball with the linesman's bald head, especially when the ball was in unclear regions (i.e. being blocked by players or when it was in the shadows created by the stadium) [5].

These examples show that AI is not a magic wand that solves every problem and also that time and effort are needed to synthesize a working model without getting errors and malfunctions. These cases were purely accidental as the problems encountered were not caused by malicious behaviours but what if an attacker could cause a failure on purpose?

Recent researches and papers have proved that AI systems can be attacked by malicious actors either to cause a malfunction or to manipulate the algorithms to act in a

controlled way. It was the case of the study conducted by the Georgetown University which made up some distorted voice commands in order to make the phone to understand a different and potentially malicious instruction, e.g. a simple “Ok assistant, call mum” could be modified to be understood as “Ok assistant, visit myviruswebsite.com”. The attack was conducted in a controlled environment as the distorted waveforms were quite fragile depending also on the device used and the setup of the room but researchers were still able to produce some successful samples that caused the assistant to run malicious commands [6].

Another attack against ML is the street signs modification. In 2017 a group of researchers published an article[7] in which they showed that with some spray paint or stickers it was possible to trick a street signs classifier making the net to recognize a wrong sign e.g. a stop could be interpreted as a speed limit.

Figure 1.3 - Malicious modifications applied to a stop sign from Robust Physical-World Attacks on Deep Learning Visual Classification, 2018, arxiv.org/pdf/1707.08945.pdf

Given its real-world nature and also the obtained high success rate this attack aroused interest in the AI community because it meant that also the physical world could be susceptible of this kind of vulnerabilities [8].

Attacks against AI could be made also leveraging AI itself: that’s the case of the Endgame’s experiment conducted in 2018 where Reinforcement Learning was used to train a model that could produce undetectable malwares. RL is a learning pattern that trains a NN with a system of rewards and punishments based on the decisions produced by the model, much like a videogame in which the player loses its lives or points if he fails some tasks. Thanks to this paradigm and an OpenAI based model the researchers

were able to produce an algorithm that could modify existing malwares to make them undetectable to the attacked antivirus demonstrating another time that AI is very powerful but has also blind spots that must be considered and protected if possible [9].

In these years many researches like the ones just mentioned came out with professionals all over the world experimenting new ways to attack and defend ML models creating a new area of study, the Adversarial Machine Learning (AML).

The goal of this thesis, then, is to explore AML, firstly defining a base ground of common definitions and concepts regarding attack patterns and models along with the possible countermeasures that can be taken. Then a security framework will be introduced dividing it in two parts:

- guidelines and best practices about the design of secure ML models
- a security assessment methodology to test the robustness against adversarial attacks of already deployed models

These two sections will be followed by an explanation of the tools, libraries and applications used. In the fourth chapter there will be some use cases regarding the security framework. More precisely, there will be an implementation of all the adversarial attack types along with some defences to show possible workflows of the security assessment part. A simulation of a realistic model deployment scenario is also present to show how the concepts introduced in the security by design paragraph could be adopted. The final chapter will summarize all the topics treated along with the drawing of some conclusions and insights about future directions.

Chapter 2
Adversarial Machine Learning

As stated before AML is the discipline that studies the vulnerabilities of the machine learning models, how to attack them and which defences are possible to reduce the associated risks. Thanks to the increasing interest in artificial intelligence, AML gained attention not only from researchers but also from companies that used this kind of technologies as many potential threats have been showed up making it one of the most active research areas in the field of Cyber Security [10].

Although this enthusiasm, still there is not a unified standard on how to deal with these new situations because the field of search is quite young and in continuous development with new studies and papers been released almost every day. That's why this thesis is focused just on that, a security framework that could address both the security assessment of working models including the analysis of the countermeasures that could be implemented but also the design process of the ML algorithm to include security since the beginning of the lifecycle of the model.

This chapter analyses the profile of the attacks giving details about their characteristics, the goals and motivations that drive malicious actors and describes what are the possible types of attacks and which security measures one can apply to defend its ML model.

The organization of the topics and the definitions contained in the next pages follow the NISTIR 8269 draft [11] as it is a first step into the standardization of methods and concepts regarding AML. In this paper NIST created a taxonomy and terminology base

for AML related topics which could be grouped in three areas, Attacks, Defences and Consequences, all of them accurately described in all their characteristics.

The European Union Agency for Cybersecurity (ENISA) published two reports [12], [13] about the threats and vulnerabilities that plague AI and ML giving a mapping with possible security controls and best practices that could be useful to protect such assets. These reports analyse also the entire AI threat landscape providing a comprehensive view on where AML could be placed inside the wide-ranging scenario of the AI security showing that classical software security challenges must still be considered as AI is also a piece of software like many others.

Besides these formal reports, also a lot of courses and books have been wrote in recent times given, as said, the increasing interest in AML and Katy Warr's one[14] is for sure one worth mentioning. After a brief introduction about AI and ML the book illustrates AML core concepts and goes deep into evasion attacks and defences always providing practical examples and detailed explanations so given its completeness and clarity it has been a very useful source for all the chapters of this thesis.

2.1 Attacks profile

In this section will be analysed the characteristics of the attacks to better understand the possible threats and scenarios that could be encountered.

First the target assets will be listed outlining the attack surface that adversaries could exploit and by which entities it is composed. Then goals and motivations that drive the malicious users will be outlined and explained understanding what attackers seek and want from an attack and also what comes out from such actions, in fact these could be seen as the consequences of the adversarial incidents (as reported in [11]). After that it will be important to understand in which cases an attack can be carried on, in particular which level of knowledge of not only the ML model itself but also of the whole system in which it operates is needed to run a particular attack. Finally, some real-world patterns will be discussed in order to show realistic modus operandi of the adversaries.

2.1.1 Targets

To clearly understand which assets are exposed to potential attacks and how a ML algorithm could be tricked by some adversary it must be analysed not only the model itself but also the learning paradigm used and the context in which the algorithm operates.

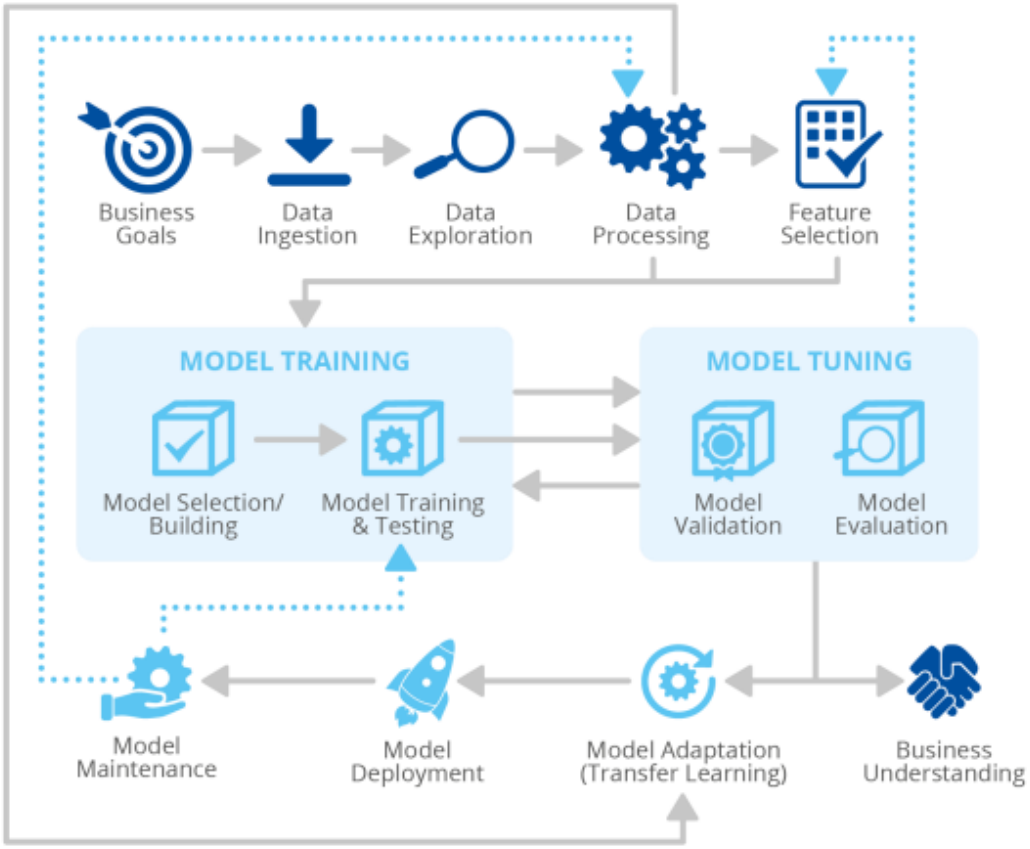

Figure 2.1 – Typical AI lifecycle from AI cybersecurity challenges: threat landscape for artificial intelligence, 2020, ENISA

As shown in Figure 2.1 the AI lifecycle is composed by several steps which could potentially be vulnerable to some sort of attack or failure especially because some of them are carried out by external providers.

Since AI have become much more powerful, also software complexity has increased resulting sometimes in very heavy models which need extremely expensive and

consuming resources. That's why many companies started to offer services like training, data management or cloud infrastructures making ML companies to only think about the design of the model. This scenario may sound beneficial and it generally is but it hides some possible threats. When dealing with external suppliers, security must be considered from the very beginning, since there could not be an adequate level of control on the protective measures adopted by the external actors resulting in potentially open doors for attackers. As it will be shown later, adversaries can penetrate systems in almost every step of the AI lifecycle so there must be an active and wide control over all processes, actors and assets involved in the ML system.

Analysing the stages of the ML pipeline, though, it becomes clear that some assets are fundamental in this workflow. Data is a crucial part of the process so it must be protected against specific attacks that could manipulate the way the model learns as this is the ground truth with which all the training step is carried on. Not only data itself but also all the gathering of it is a sensitive aspect because even if well protected and stored, data could be manipulated at its origin attacking for example physical sensors to produce misleading information. Then it comes to the model, the core asset of the ML system, it must be robust against adversarial actions and designed in a way that makes it to not leak sensitive information about its internals.

There's no golden rule about which assets are to be protected and which not, it should be kept in mind that potentially everything is vulnerable so every part of the ML product must be analysed, tested and secured, even the not so obvious ones.

2.1.2 Goals

Every classic cyberattack has an objective and AML ones are not different. Goals could be divided in three categories: *Integrity*, *Availability* and *Confidentiality*.
Integrity violations mine the inference phase resulting in a disruption of the model's normal behaviour which means, for example, that an image classification network, if attacked, starts to produce wrong predictions (misclassification). An integrity violation could be achieved in two ways, by totally confusing the NN making it to predict wrong results (so the network has high confidence in a completely incorrect result) or by confidence reduction which means that the network could still be able to produce the correct output but the confidence that it has is drastically reduced.

The *availability* of the ML system is also another goal of the attackers in fact it is possible to limit the access to the model or to damage its quality of service for example reducing its prediction speed, making it unusable. This is close to the usual types of cyberattacks in which malicious users mount DoS attacks or try to interfere with the servers' normal operativity.

Confidentiality violations are the third type of goal of AML attacks. Adversaries can extract internal characteristics of the models like their weights and hyperparameters or can obtain information about the data used to train the model. The attacks that mine confidentiality are capable of constructing a working copy of the target model, extract model parameters, recover some data used in training or determine if a data point is part of the samples used by the model. A particular subclass of confidentiality attacks are the *Privacy* ones in which an adversary can infer sensitive data about an individual e.g. medical data like in Fredrikson attacks [15], [16].

2.1.3 Attacker's knowledge

The amount of knowledge an attacker has about the target ML system determines which types of attacks are possible.

In White-box attacks adversaries have complete knowledge about the model, its architecture and parameters and also the data used to train the model while in Black-box settings all the details about the system are not available so the attacker knows only the input-output pairs. The remaining situations can be classified as Grey-box, in these cases adversaries have partial information about either the model itself or the data used.

Most of the attacks are white-box as many more information is available to find vulnerabilities in the model or its implementation but even if it sounds hard, in black-box settings are still possible some attacks (like for example the model extraction one by Correia-Silva [17]) making this case the most alarming one as it reflects a more realistic scenario.

2.1.4 Attack patterns

Real world scenarios are obviously different from theory as their implementations and protections are diverse one to another making the attacker's job much more dependant

from the actual target setting. Based on the level of access and knowledge the adversary has on the system different approaches could be defined.

2.1.4.1 Direct attack

In direct attacks a malicious user can submit requests to the ML model and receive the corresponding responses. The attacker, then, has the result from the real network making the attacks more precise as they're crafted upon the actual target. Usually, though, this setting is not so common, in fact companies that expose publicly their ML assets often limit the query rate of the user blocking incoming requests after reaching a threshold or slowing down the responses. Sometimes also generic errors are used to discourage attackers or even postprocessed results, which don't reveal the real probabilities predicted by the model but only some filtered results.

Direct attacks are not so common given all these countermeasures but an approach of this kind is still useful because sooner or later the attack must be performed on the real system in fact, usually, a direct approach is chosen to test some already crafted attacks and not to start from scratch.

To address the problems and limitations introduced by companies that expose a ML model, attackers can choose to work on an exact copy of the model/system or on an approximation of the model itself; these approaches are called respectively Replica attack and Transfer attack.

2.1.4.2 Replica attack

In replica attacks adversaries exploit the possession of an exact copy of the target, being it the entire system or only the model (replica system vs. replica model).

Replica systems are usually commercial products easily available like, for example, smart assistants that an attacker can experiment with. Usually, because these products are sold to the public, their internals are carefully protected against reverse-engineering actions that could be made by adversaries or competitors. Also, these products most of the times don't accept digital inputs but only real-world ones making attacks way more

difficult as an extra pre-processing step is included into the pipeline; usually outputs too are not digital to prevent exploiting this kind of results.

Replica models are not so impossible to obtain especially because of the recent explosion of the ML world. As explained some paragraphs above, nowadays companies are offering pre-designed and trained models to enable also small or new customers to create their own ML system using a controlled and simplified environment so if an attacker could know which implementation is targeting then a replica model could be easily obtained. Models are also pretty standard nowadays because stable and tested solutions are needed to obtain a usable and marketable product so even if the attacker does not know from which platform or vendor the target comes from, they could still be able to recreate a copy of the objective knowing the network used and some details about hyperparameters and training data (that are also quite standard for common tasks).

If no info is available another way could be taken to create a copy of the target model. This will not be as exact as the one in replica attack but will still be helpful during the attack mounting as it will reflect the behaviour of the target model but using a different architecture.

2.1.4.3 Transfer attack

In transfer attacks a copy of the model is created leveraging information about the algorithm, the system in which operates, the input data domain and the input-output pairs. These details are usually a mix of actual data stolen from the ML system and inferred assumptions based on the observation of the model during its normal runtime. To make an example, if the target model is an image classifier that distinguishes between humans and animals then for sure it has been used a convolutional neural network (CNN) trained on the ImageNet dataset, while for a digits classifier has been used for sure the MNIST dataset with still a CNN but less deep with respect to the one used for image classification.

The possibility to create a copycat model that is close enough to the original one to be susceptible to the same adversarial attacks might sound a bit unrealistic but it’s not that hard. Especially for misclassification attacks, the important thing is that the transfer model and the target one have close enough prediction landscapes in order to produce inputs that could exploit flaws hidden in such learned mappings.

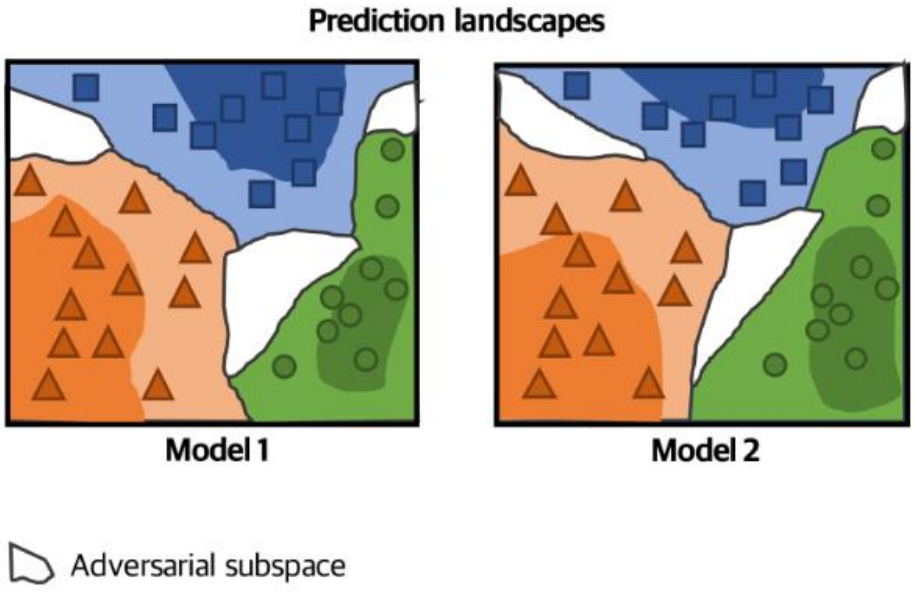

Figure 2.2 - Adversarial subspaces of two different models trained with the same data from Strengthening Deep Neural Networks, 2019, Katy Warr

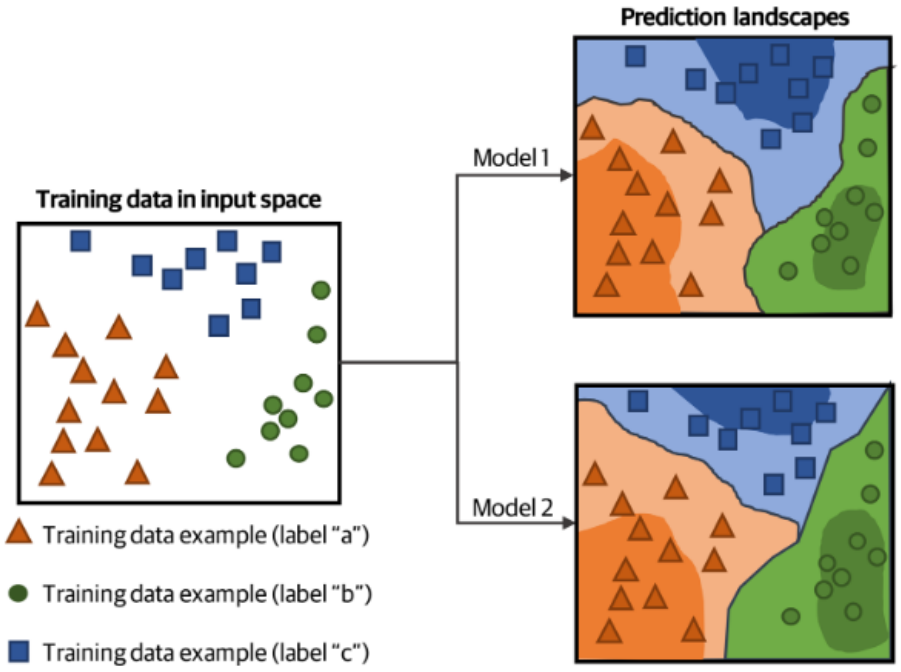

Figure 2.3 – Prediction landscapes of two models trained with the same data from Strengthening Deep Neural Networks, 2019, Katy Warr

In fact, it has been noted that two models with different architectures but trained with the same data are likely to share their prediction landscapes and also their adversarial subspaces, as shown in Figure 2.2 and Figure 2.3. Adversarial subspaces are the areas in which the training process has not generalized well resulting in a not clearly defined classification area. These spots, then, could be used by attackers to craft adversarial samples that work on both systems.

2.1.4.4 Universal transfer attack

As a direct consequence of the transfer attack the universal one is the variant in which the attacker has no knowledge about the target model, this is clearly the hardest scenario. To try to accomplish some results, adversaries craft attacks on an ensemble of substitute models seeking the ones that successfully affect a large number of these models; the chosen attacks are then the ones that are more likely to produce a result once launched against the real system. Training data characteristics are very important also in this situation because even if the data used to create the substitute models are not the same, they still come from the same domain so they share common features that could in theory lead to adversarial subspaces.

In real world scenarios, usually a combination of patterns is adopted, based on the specific target setting. For example, most of the ML systems are nowadays a bit protected against strange users' behaviour (e.g. repeated queries) so it's likely that an attacker will start to develop an attack on a replica or substitute model but at some point, he will test it against the real system to understand if it works and what must be refined. There is, so, a constant trade-off between the effectiveness of the attack and the covertness of the mounting that the attacker must take into consideration.

2.2 Attacks and defences

Having analysed characteristics and common patterns, the next step is to define what kind of AML attacks are possible. Research is very active in this field so everyday new vulnerabilities and defences are found but despite this potentially chaotic scenario is still possible to classify AML attacks in four categories: *Evasion*, *Poisoning*, *Inference*, *Extraction*.

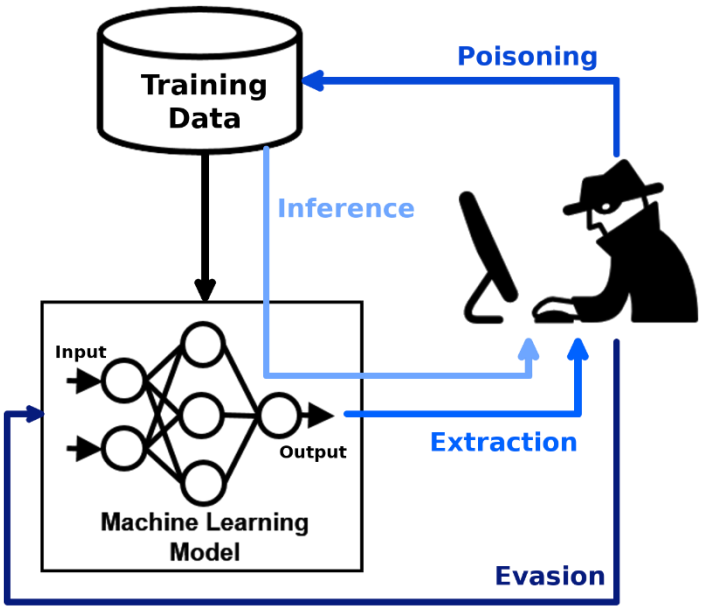

Figure 2.4 - AML attacks from adversarial-robustness-toolbox.readthedocs.io

These families of attacks can be grouped also by their target phase of the ML lifecycle.

Training time attacks target all the operations and assets that are involved in the design and training of the model, all that is prior to the publication of it. *Data access* attacks try to obtain access to the samples used to train the NN in order to create a substitute model while in *Poisoning attacks* the dataset is manipulated to change what the network learns.

Inference time attacks, on the other hand, exploit vulnerabilities of a model already trained and published. *Evasion attacks* try to create samples that won't be recognised correctly by the model by changing the feature representation of the sample. *Extraction attacks* aim to create a copy of the target model by simply learning how it maps inputs to outputs and this is done by submitting a number of queries to the victim and analysing its results. *Inference attacks* are designed to steal information from the model about data that it has seen during the training phase; they can be divided in *Inversion attacks,* in which an adversary tries to reconstruct model's training data and *Membership inference attacks* where the model is queried to understand if a specific data point belong to the same data distribution of the training dataset. Sometimes extraction and inference attacks are named *Oracle attacks*.

2.2.1 Evasion attacks

Evasion attacks are probably the most widespread and studied nowadays with numerous papers being released every year. The goal is to create or modify a data sample in a way that it makes the model to misclassify it or in general to produce a wrong result.

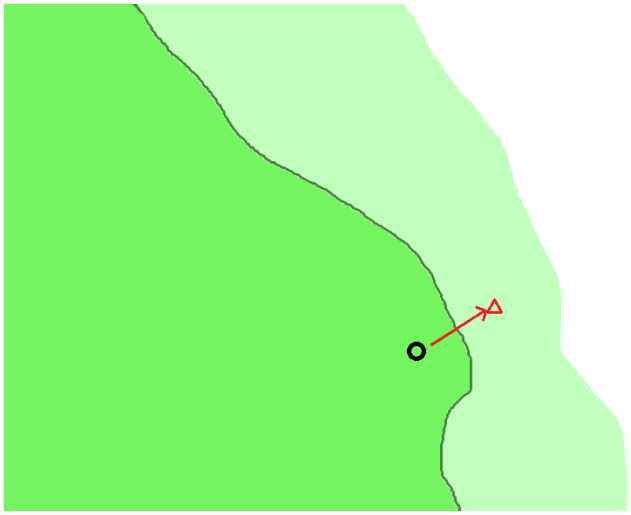

Figure 2.5 - Moving a sample from its classification area

To do so the digital representation of the sample is moved from its class area to another one (targeted attack) or to an adversarial subspace, a region where the network produces results with less confidence.

This translation can be obtained by adding or multiplying to the original sample a perturbation that will change the internal features of the data point to make it look like a sample from a different class. The ways in which these alterations are calculated and created are the actual attacks, a mathematical and computational set of steps that generate an adversarial sample combining a clean input with a well-crafted distortion.

Usually these perturbations are crafted in way that makes them almost undetectable to a human (and sometimes even to digital detectors) while keeping their attack capability and this is done by altering the sample's feature representation i.e. what the neural network sees about the sample.

This is quite never an easy job but sometimes it can get even harder because with some data types even small perturbations could result detectable or at least suspect. For example, in Natural Language Processing (NLP) an evasion attack should modify a sentence in a way that the meaning remains the same while still causing a misclassification but there is not so much room for changes without affecting the understandability of the phrase.

On the other hand, though, for images the job is simpler in fact the majority of the AML (and evasion as well) researches focus on this data type, especially on the classification task. The objective is the same as before, an image must be altered to semantically result identical to the original one but it must cause a misclassification when analysed by the attacked neural network. To do so, the adversarial image is optimized to be not so much "distant" from the original one in the input space so not so much visually different to a human eye.

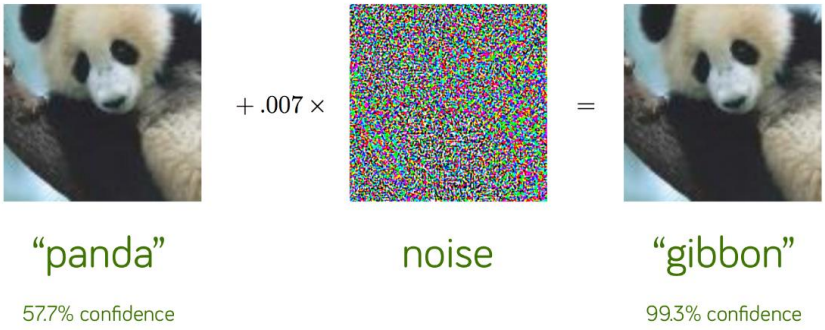

Figure 2.6 - Evasion attack demonstration from Explaining and Harnessing Adversarial Examples, 2015, arxiv.org/abs/1412.6572

To understand how much an altered image is “distant” from another one its L^p-norm has to be calculated.

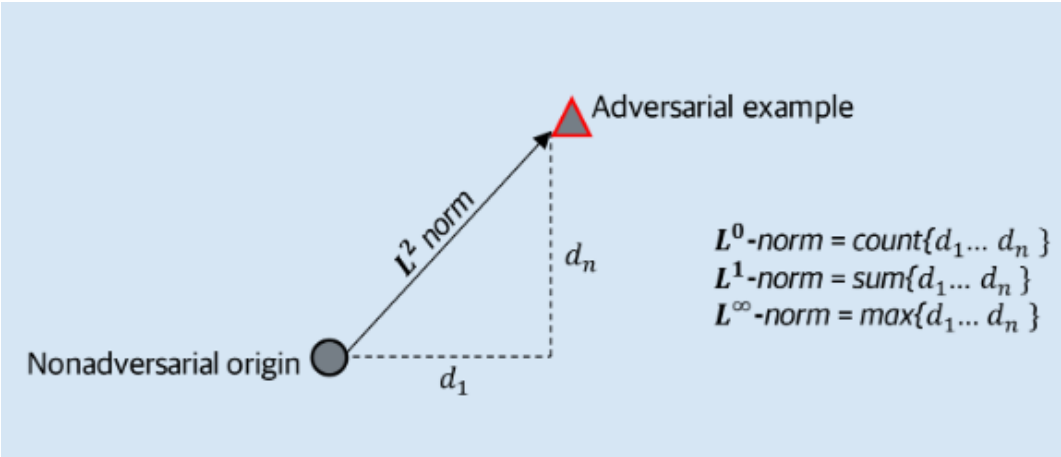

Figure 2.7 - Types of norms used to measure adversarial perturbations from Strengthening Deep Neural Networks, 2019, Katy Warr

As shown in Figure 2.7 every norm has its own formula and therefore its own meaning.

L^0-norm counts how many pixels have been changed but not the magnitude of these changes so, for example, there could be few modified pixels that are way different from their original ones and this could anyway lead to the detection of the attack.

L^1-norm is the sum of all the modifications made on the pixels so it indicates how much, in general, an image is different from its previous version but nothing is said about the location of these changes.

L^2-norm is the standard Euclidian distance which could still be ported into a multi-dimensional space like the features' one but this metric is not used because in this kind of spaces it turns out that all the points are a similar Euclidean distance apart [14] making this measure not good for perturbations.

L^∞-norm indicates the maximum change made to any pixel. It is used when it's needed to operate undetectable modifications to an indefinite number of pixels as this norm doesn't say anything about the amount of changes done but only the magnitude of the maximum one.

These approaches are suited for a digital attack so the attacker has access to data that will actually feed the neural network but AML can be used also for physical world attacks.

While it may sound a bit difficult, attacks to models that use physical world data are possible, making this another scenario to be concerned about. Think for example to self-driving cars, they use dozens of sensors and cameras to scan the environment around

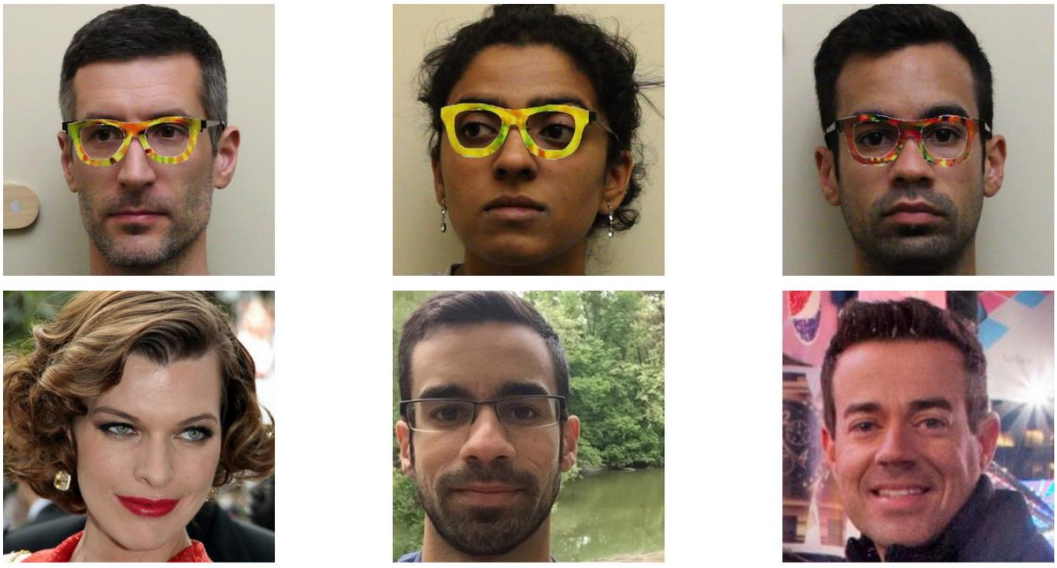

Figure 2.8 - Researchers wearing adversarial glasses with the impersonation targets from Accessorize to a crime: Real and Stealthy Attacks on State-of-the-Art Face Recognition, 2016, https://dl.acm.org/doi/10.1145/2976749.2978392

them to understand where they are, where to go and what constraints and threats surround them. If an attacker could modify the way in which the car's ML model recognizes objects around it by simply applying some stickers to the street signs[7] this could potentially lead to some serious consequences. Facial recognition systems too could be attacked as discovered by a group of researchers from Carnegie Mellon University [18]. In particular, it turned out that it is possible to impersonate someone else or to just not be detected at all by simply wearing a pair of glasses.

As shown in Figure 2.8 with a well-designed pair of glasses is possible to fool a face recognition system opening the door to possible security threats like criminals that could avoid these kinds of checks and act as they want without being detected. It must be said that the attack works under certain circumstances for example light, exposure and distance of the camera but it's still a prove that AI can be fooled and so it must be protected.

These two examples show what kind of perturbations introduce a physical-world attack, these alterations are small in their dimensions but usually more detectable by a human being. Giving the extra processes that an adversarial physical sample must undergo (e.g. camera acquisition or microphone sampling) these attacks must produce robust perturbations since it's very likely that the attack, for some reason, could not succeed.

In image recognition a common approach is adversarial patches. These triggers usually look different from the rest of the image and are crafted in a way that makes the net to change drastically its predictions when it sees this patch, like in the glasses example above.

In NLP, usually a sort of specifically designed noise is added to an audio sample in order to modify the resulting waveform sampled by the acquisition device producing a misclassification in the NN.

Evasion attacks are very diffused and studied so a lot of work has been done to design security measures that could protect AI models from this kind of threats.

One of the most adopted solutions to improve model robustness is perhaps Adversarial training in both Madry's [19] and Tramèr's [20] flavours. With adversarial training the model learns using malicious samples which are correctly labelled, this is done to train

the algorithm to deal with potentially crafted samples that are not like the data in the clean training dataset. At the end of the learning phase the model has seen also (or only, depending on which solution is adopted) adversarial data but it has been taught how to process such samples. This is an effective defence as it has shown several times that improves the accuracy of the ML systems against crafted samples although it depends on which attacks are used in the dataset augmentation: it's recommended, in fact, to use multiple generators to increase the diversity of the adversarial dataset. Adversarial training has been found to be more effective when used together with Feature Denoising [21]. This solution introduces some denoising layers into known networks in order to reduce the attack power of the samples. Another possible defence is the introduction of a Sparse Transformation Layer which has the task to project an input sample into a different image space removing adversarial perturbations[22]. Among the possible defences it's also worth mentioning the work by Hosseini et al. [23] which explains how to make a model to identify bad samples. With their method a neural network is trained to identify potentially adversarial samples and to label them with an extra class ("invalid" or "NULL" for example) in order to not assign any label to a sample on which the network has low confidence.

Pre-processing is another strategy used to reduce the attack strength with many possible implementations like JPEG compression, normalization, crops and rotations and so on[24]. Usually these solutions are used like a sort of adversarial training although these samples are not properly malicious but still could lead to some unwanted behaviour. In this area fall other approaches and solutions like for example BaRT (Barrage of Random Transforms) [25] which is a set of individually weak defences (mostly image transformations) that are randomly combined in order to produce a strong countermeasure against evasion attacks. Other approaches use Generative Adversarial Networks[26], [27] to learn the clean samples representation and reconstruct input data without the adversarial perturbations. A similar approach uses deep restoration networks to enhance image resolution and suppress the attack[28].

A more classic approach has also been proposed which tries to detect malicious inputs much like an antivirus does with files and programs. Chen et al. [29] proposed a stateful detector designed for MLaaS systems which keeps a history of the submitted queries to understand if a request is benign or not using a kNN model, an algorithm that classifies samples by comparing them with near data. Another approach of this type uses deliberate modifications on inputs to understand if there is an attack going on. In

particular, the algorithm proposed by Tian et al. [30] perturbs on purpose input images and compares the predictions of the classifier: if the sample is malicious then the output probabilities of both original and modified images should be similar because clean data tend to be more susceptible to these modifications. Denoisers too can help in detecting adversarial inputs. As proposed by Liang et al. [31], by using adaptive algorithms is possible to denoise an image even for little distortions and then compare the original image with the modified one to understand if there is some malicious perturbation.

Defensive distillation has proven to be effective against evasion attacks thanks to its training procedure of the target model. An already trained classifier is used to produce predictions for the training dataset and these outputs are used to train the protected algorithm which learns from probabilities instead of from labels.

Gradient masking approaches could also be a solution for this type of attacks with many implementations that hide the real gradient values to the attackers discouraging them from mounting white-box attacks. These last types of defences will be investigated in the Security by design chapter.

Evasion attacks, as said, are the most common ones also because of their immediacy and ease of creation but for these reasons they're also the most thwarted with lots of solutions and defences released every day.

2.2.2 Oracle attacks

As described in Chapter 2.2 there are other types of inference time attacks besides evasion ones. Oracle attacks aim to infer parameters or characteristics of the model or data used by simply querying the algorithm and collecting the corresponding outputs [11]. These attacks can be divided in two sub-categories, *Inference* and *Extraction.*

Inference attacks leverage information given by the model to try to reconstruct the data that the model used to learn its parameters; they can be classified in two categories, *Inversion* and *Membership Inference*, based on what the attacker wants to infer.

In *Inversion* the goal is to obtain a representation of the training dataset, usually this is obtained by presenting to the model synthetic data and analysing its outputs.

Membership Inference, on the other hand, tries to determine if a specific sample has been part of the training dataset used by the model by comparing predictions on different data points in search of some differences. These attacks are largely influenced by the in-class standard deviation of the samples and by the number of classes, in particular the attack accuracy increases as these values get higher.

In *Extraction attacks* the attacker aims to reproduce the target model by extracting some information about its parameters and using them to train a clean model. A common approach is to create a dataset of input-output mappings by querying the victim with random data and to use this fake dataset to train a model which will behave as the target one.

These attacks must not be underestimated because they represent a serious and real threat for companies' assets and peoples' privacy.

An extraction attack, for example, could create an exact copy of a ML model for which it has been spent a lot of time and resources in the design and creation processes. In this scenario a competitor could vanish all the effort spent by the victim and perhaps gain a better market position thanks to the costs that have been saved by simply copying the ML asset.

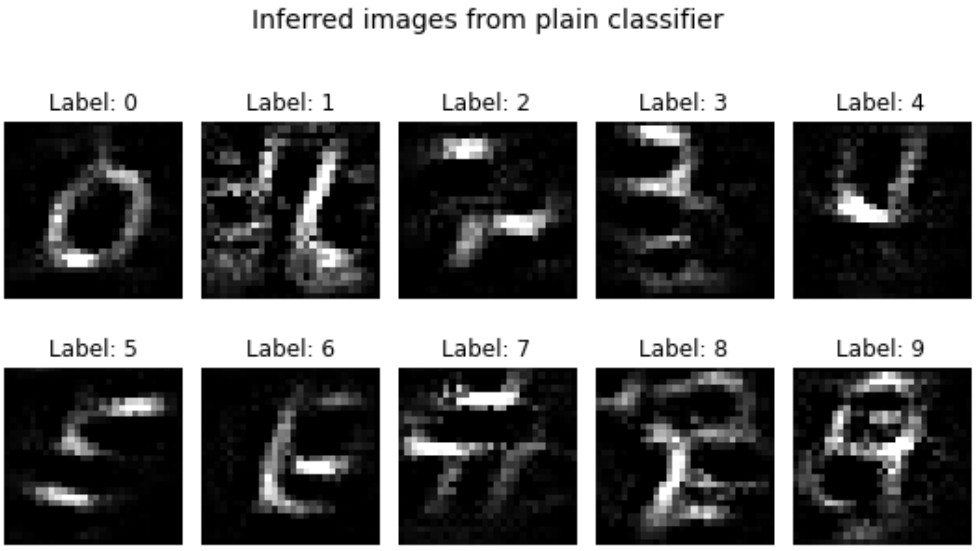

Figure 2.9 - Reconstructed images from a digits classifier with no defences trained on MNIST dataset

On the other hand, an inference attack could mine the privacy of people whose data has been used to train a model like a face recognition one or some predictor that uses health data. It has been demonstrated [15], in fact, that inference attacks can reconstruct faces and private data with a high level of fidelity opening new doors to possible threats as shown in Figure 2.9.

Due to the dangerousness of these attacks many defences have been discovered and proposed to protect ML assets from these threats.

A first and simple measure could be API hardening, in its many flavours. Since Oracle attacks leverage information given by the models with their outputs, restricting this data is highly beneficial when possible. A possible implementation of API hardening is to not expose APIs at all or at least limiting the functionalities offered to solve the problem at the origin by not giving any information at all to attackers. If APIs could not be hidden then a postprocessing of the results could be beneficial. A good practice is to don't disclose output probabilities at all since these are really valuable data to the attackers and, instead, to publish only the top-n predictions or, better, only the highest one. The best solution, anyway, is to give to users only the output label without any indication about the probability values or at most a clue about the level of confidence the model has on that result (e.g. This is a *dog* with *high* confidence). Finally, still in the API area, also monitoring their usage could be beneficial because usually these attacks need to query repeatedly the ML system so anomaly detection in the usual resources requests made by the users could spot potential attacks. Deriving from this idea another countermeasure is limiting the access rate to the model which will delay the information gathering phase and the attack in general. While effective these countermeasures are not always a definitive solution but for sure they harden the attacker's work making them to choose an easier target as the principles of the cybersecurity suggest.

Data sanitization is another very important step towards preserving privacy of the model. When pre-processing data for the training phase some questions should be made about how much private this data is in order to not publish a model that has memorized personal information about someone. Private data must be anonymized or deleted and every reference to the data owner must be removed to protect privacy. To quantify the risk of membership inference attacks of a model the *Differential Training Privacy (DTP)* metric could be used. DTP tells how much a data sample is vulnerable to this type of attack with respect to a classifier and its training data and its general version

extends this concept to the entire dataset identifying the maximum risk [32]. This metric could be used to identify weak samples and therefore to delete them, reducing the risks.

Differential Privacy is another solution worth mentioning to protect against oracle attacks. DP articulates itself into different implementations that affect gradient and objective functions or the output labels, always perturbing them to avoid any information leakage.

Sometimes it could be useful to adopt some techniques that permit to understand if a model has been stolen and model watermarking is just the perfect solution. Although it isn't properly a defence against model extraction because it isn't able to prevent an attack, watermarking can be the only method to prove, even legally, the property of a ML model and this could be done by checking the presence of some specific data (eventually bounded to the owner's identity) that the legitimate owner has inserted into it[33].

More details about these last solutions will be given in the next chapters.

2.2.3 Poisoning attacks

While Data access attacks are more standard and well-investigated by classic cybersecurity defences, Poisoning is a specific class of attacks against machine learning that deserves ad-hoc approaches.

The aim of these attacks is to modify the behaviour of the model by either modifying the training dataset or the model itself. Different approaches are possible: with *Data Injection* some adversarial samples are added to the training data manipulating its distribution and therefore the decision boundaries of the network; with *Data Manipulation* the attacker maliciously alters some data points features or labels in order to obtain the same results of the injection but without inserting any new data; finally using *Logic Corruption* approaches an adversary can tamper with the ML system to alter the learning process and model itself[11].

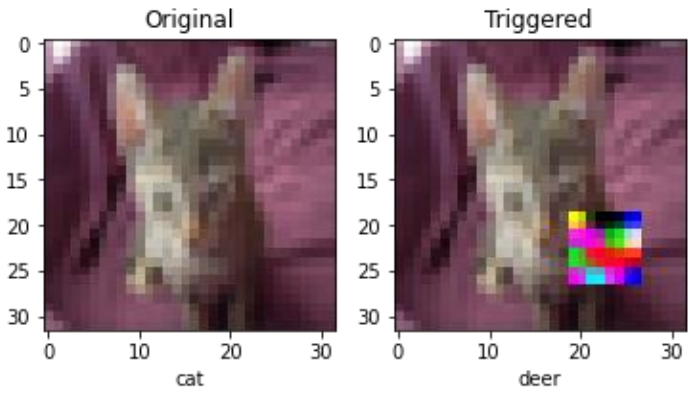

Figure 2.10 - Example of trigger poisoning attack

Two main targets can be identified when talking about poisoning and they are *availability* and *integrity*. Attacks against availability try to entirely disrupt the normal functioning of the model by usually altering massively the decision boundaries. Attacks against integrity, on the other hand, aim at altering the model's functionality without affecting the expected behaviour and this is usually achieved by means of backdoors and triggers.

To give an idea of the threats that poisoning attacks can pose, one could think for example to a company that relies on users provided data to train a model. In this scenario, training data could be a vector of data injection causing the model to act unexpectedly like the case of the Microsoft's chatbot Tay which was supposed to continuously learn how to make conversation analysing anonymized public data but due to an attack it started to express offensive messages forcing MS to shut down the bot[34]. The attack consisted in inundating Tay with lots of racist, misogynistic and anti-Semitic messages to make it to learn almost only this type of language, much like in a poisoning attack. Another possible scenario involves backdoors and triggers in fact it's possible to train a model to make it to have different behaviours based on the presence or not of a specific trigger.

In a reality where training a ML model could be very expensive and time consuming, poisoning attacks represent a threat that could not be ignored so a lot of work has been done to address these weaknesses.

Since poisoning leverages training samples, *Data Sanitization* is one of the possible defences against this type of attacks. Sanitization could be achieved in many ways, for

example by monitoring data with a detector in order to understand if there are manipulated samples by means of some assumptions on the data distribution or specific analyses upon samples. *Reject On Negative Impact* (RONI) is one of these detectors in fact it measures whether a data point impacts negatively on classification accuracy by comparing two classifiers trained both with the same dataset but one is added with the sample to analyse[35]. *Micromodels* are also another possible way to sanitize data as they aim to remove poisoned data by comparing a set of classifiers trained on disjoint subsets of training samples and then by majority voting these micromodels to identify adversarial samples[36]. Other detectors like *STRIP* intentionally perturb input samples in order to verify their maliciousness: the intuition is that predictions of already manipulated samples are not so much affected by these alterations whereas if a clean data point is perturbed then its predictions will sensibly vary depending on the perturbation applied[37]. Also *humans* could be valid poisoning detectors in fact in some settings the ML system prompts security analysts to decide on suspicious samples. For regression tasks, *TRIM* could be a valid solution for detecting poisoning attacks because it identifies malicious samples using a trimmed loss function and permits to learn a robust regression model[38].

Gradient shaping is another technique that could prevent poisoning attacks. It consists of various implementations that reshape and modify the gradients of the input samples in order to understand if there are poisoned samples for example by using a different SGD approach or analysing the gradients values themselves.

Other defences fall into the *Robust learning* field with papers devising methods that makes models more robust by pruning highly reactive neurons and manipulating models' internals to better resist to possible attacks.

For some particular tasks *Bagging* has been adapted to identify poisoning attacks. This technique was first introduced to improve the accuracy of a classifier but then it has been discovered that it is effective also for poisoning scenarios. The idea behind bagging is to train different classifiers of perturbed replicates of the training set and then aggregate their predictions[39].

Since many poisoning attacks exploit information given by models' gradients, methods from the gradient masking area could be adopted (more details in the next sections).

Finally, another help to tackle poisoning could come from *auxiliary tools* like GANs and robust statistics. Generative Adversarial Networks could be used to learn the generative

model of the training data and using it to identify malicious samples; robust statistics, on the other hand, could be used to analyse input data at a deeper level as done by Tran et al. that looked for traces, named *Spectral signatures*, left by backdoor attacks and used them to identify poisoned samples[40].

Chapter 3
Security framework

After having put the basics about AI, ML and the related attacks and defences, the security framework could be introduced. As said in the first chapter, the framework is divided in two sections. The first one will be focused on principles of security by design, best practices and solutions that allow to create a more secure ML model since its conception. The second part, instead, will define a security assessment procedure with all the steps required to test the robustness of a model against AML attacks and to address the possible security flaws found. To close this section, there will be a discussion about tools and software that could be used to implement the solutions proposed in the framework and that have been helpful in the making of the use cases that will be presented in the next chapter.

3.1 Security by design

Like said in the previous pages, the world of artificial intelligence and machine learning, in particular, are ever changing and in continuous expansion and this carries on also an increasing complexity in architectures, processes, algorithms and so on. Within this scenario, so, stakeholders need a precise and conscious planning about the security measures to be taken to protect their AI assets. This naturally leads to a proactive approach towards AI security instead of the old reactive way as many steps have been taken in this direction by the research world. Preventing security incidents is nowadays

fundamental to avoid the high costs that comes from a bug fixing in later phases of the product development. That's why many solutions have been proposed to address design defects as early as possible, spacing from the model's internals to the entire infrastructure architecture to protect the widest attack surface possible.

There is not an official classification about these principles but for a clearer organization of this work the proposed solutions will be grouped as:

- Model characteristics
- Pre-processing
- Training
- System security

3.1.1 Model characteristics

The first part of a ML system to protect is for sure the model itself. Along with the continuing research about new, faster, cheaper and more accurate networks another stream has born in recent times, the one about model robustness against adversarial attacks. Since the discovery of the vulnerabilities that have been explained in the pages before, new attack algorithms have been discovered every year so it has been necessary to start thinking about how to design models in a way that doesn't let attackers any room for action.

3.1.1.1 Gradient masking

Including *gradient masking* techniques could be beneficial to protect from gradient-based attacks. There are different solutions that could be applied according to [41]:

- Shattered gradients, inputs are modified, intentionally or not, in a non-smooth and non-differentiable way that makes the classifier non-differentiable
- Randomized gradients, various solutions that add randomness into the system e.g. random resize, drop of casual neurons, model ensemble
- Exploding/vanishing gradients, gradients values too small or too large to be used for an attack

Gradient masking works well for white-box settings and gradient-based attacks but it doesn't eliminate the possibility to have other types of adversarial samples because this solution only makes the attacker to waste time but it doesn't solve the problem[42].

3.1.1.2 Differential privacy

A way to protect against privacy attacks is *Differential Privacy*, a mathematical framework that formalizes the privacy concept and guarantees that no private information is accessible from the outside of the model. Although its good privacy protections, DP doesn't eliminate the threats given by adversarial attacks, indeed it has been demonstrated that in certain scenarios this can lead to a slightly wider attack surface of the ML system. In a recent work [43], in fact, researchers found out that differential privacy could lead to more rugged prediction landscapes and to an easier transferability of the attacks. At this time, so, privacy and robustness usually impact badly onto each other unless some precise corrections are made.

3.1.1.3 Model watermarking

As discussed in chapter 2.2.2, *Model watermarking* could be a solution to prove that a model belongs to its creator in a legal ad demonstrable way[33].

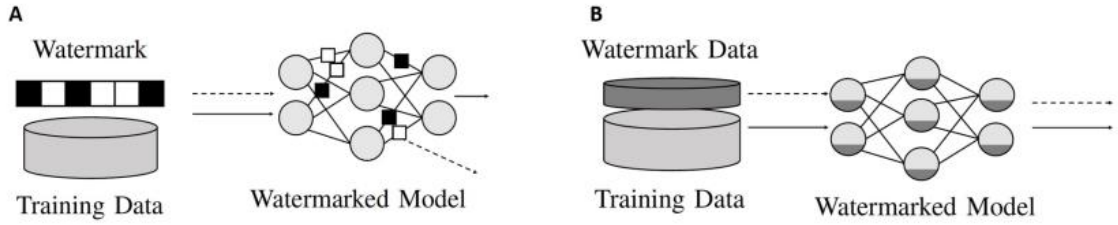

Figure 3.1 - Watermarking techniques: (A) Embedding into model's parameters. (B) Using trigger dataset. From A systematic review on model watermarking for neural networks, 2020, arxiv.org/abs/2009.12153

Watermarks can be embedded into the model parameters, either if they already exist or not, or can be created leveraging a trigger dataset that shapes the model's behaviour to make it react in a precise way when the verification procedure starts (like a poisoning attack) (see Figure 3.1). Two verification processes are possible, white-box and black-box, depending on if the model is accessible or not. Watermarks can also carry additional information other than the simple proof that the model has been stolen and such bit strings can be used to prove the relation with the owner or also the uniqueness of that

instance. Although their pros, these tags suffer from some vulnerabilities in fact there are attacks that permit to detect, suppress, forge, overwrite and remove watermarks so a conscious selection of the watermarking scheme to adopt is needed.

3.1.1.4 Model internals

Also *models internals* could be a source of robustness with many best practices that could be followed to design more resilient architectures. In [44] it has been found that there exists a correlation between model's width and robustness. Wider or deeper models, in fact, seem to not necessarily lead the network to be more robust; even if adversarial training needs wider models there still exist a trade-off between wideness and robustness as deeper architectures lead to increased Lipschitzness (stability of the model's output to input perturbations). Reducing, where it is possible, the dimensions of the model could also be beneficial. Random feature nullification is another possible defence against adversarial attacks as it uses randomness to confuse the attacker making the network non-deterministic. In particular, this method is implemented as a pre-processing step between the input and the first layer of the network and it nullifies some features in a stochastic way, for example for image-related tasks some pixels are masked or set to zero.

Some works underlined that adversarial defences lead to models that have saliency maps close to human interpretation so in [45] it has been investigated if the opposite holds. It has been found that *Interpretation Regularization*, together with SmoothGrad, can make a NN to produce more interpretable gradients by matching the interpretations made by an adversarially trained model, leading to more robustness.

Introducing randomisation techniques into the model is another possible defence against adversarial attacks[46]. Random transformations of the inputs, like cropping and padding for images, could mitigate other possible perturbations injected by an attacker making them as they were just random noise which many NNs process without any problem. A similar effect could be achieved using random noising, a technique that adds some noising layers inside the network in order to mitigate or bound possible adversarial alterations. Finally, another randomisation solution could be a random pruning of some features at different levels of the architecture to better control and guide the learning process.

3.1.2 Data security

As said in Chapter 2.1.1, Data is one of the most valuable assets of a ML system so it must be protected with the same attention than the model itself. There are different solutions that could be integrated into a ML system in order to ensure privacy and integrity of data to avoid adversarial attacks. These can be used to either improve the learning process of the model or to prune potentially malicious data points. All the proposed defences are illustrated following a realistic AI lifecycle which starts with data acquisition and ends with the actual usage for either training or inference.

3.1.2.1 Data creation

First of all, it's important to protect the creation of training data and its journey to the model. Although this is not a topic strictly related to AML it's still an important discussion as it contributes to the design of a secure ML system. With the increasingly usage of AI in every field of business, data must be traceable and verifiable in order to understand if there could be any security breach which could compromise the integrity and the non-adversariality of it. To do so, some tools have been created in recent times to help companies maintaining the *Data Lineage and Provenance,* that is the visualization and monitoring of the flow of data since its creation. These software permit to have a complete view on all the information flows including data and metadata analysis that are useful to perform sanity checks and all sort of inspections. Data must be secured since its creation so security controls must be done to ensure its correctness and stability, analysis have to be conducted also on the manipulations that could happen, especially on which entities are authorized (and at which level of privilege) to do such operations.

3.1.2.2 Data manipulation

Data could also be modified, aggregated, expanded and updated and keeping track of these situations is crucial. Much like it happens with software versioning, where subsequent snapshots of the code are taken to track every modification occurred, also for data some versioning systems are being developed, named Data Versioning Controls. These applications are for data science what Git is for software development, they keep track of the modifications (along with some related metadata), offer VCS functionalities

like branching, comments, and often they can also connect to cloud and storage infrastructures in order to automatize and integrate all their capabilities.

All the ML-related processes, of course, must adhere to the basic security regulations and best practises as every digital system. For example, data must be handled following the GDPR or any equivalent law to ensure that every action involving sensible information is conducted in an appropriate and respectful way. Also, every action on data must be tracked, especially keeping a record of who is the author of that action. Privileges and authorization levels must be set up properly and every entity that has access to data must be identified to keep track of every occurrence.

3.1.2.3 Pre-processing

Once samples are collected and stored, some pre-processing steps could be included in the system's pipeline.

Along with usual data preparation which involves denoising, normalization, cleaning and so on, also a data sanitization strategy could be involved. As discussed previously, sanitization involves data privacy assurance techniques like anonymization and filtering of sensitive data to avoid any information leakage.

Another processing step that could prevent some adversarial attacks is detectors. These algorithms, in various ways, try to understand if a data point is malicious or not for both evasion or poisoning attacks. Next to RONI and STRIP, which were described in the Chapter 2.2.3, other training-phase detectors are available. *Spectral Signatures* analyses data samples to find traces left by attackers into the data points and it does that by looking at the spectrum of the feature representation of the samples [40].

Another poisoning detector is *Neural Cleanse*[47] which aims to detect but also to reverse engineer potential backdoor triggers. To do so, the algorithm calculates possible triggers for all the labels of the model and then finds the real one by running an outlier detection upon these potential triggers. The algorithm finds the minimal trigger required to obtain an effect so it could be slightly different from the real one. Other approaches [41] use an auxiliary model trained specifically to predict if a sample is adversarial or not following a binary classification scheme or a classical one with an added class for adversarials. Also, some statistics could be used to decide upon data samples, these could be referred for example to principle components or to the distribution of data.

Another detection solution could be related to the stability of the predictions. Models or inputs could be manipulated in ways that make the output predictions to change and these changes are analysed to classify data points as malicious or not.

3.1.3 Training

Robustness could be achieved also implementing defensive measures during training time. Different methods have been proposed to create robust networks since their learning phase, all of them searching the perfect steps to take for achieving high performances while remaining safe against adversarial threats. Many solutions fall into the *adversarial training* area but also other paths have been explored like the similar *data augmentation* or the ad-hoc designed learning protocols like *PATE*, *Distillation, Bagging*, *TRIM* or *Federated Learning*.

3.1.3.1 Adversarial training

Data augmentation consists in adding to the training dataset some new points that derive from the original ones. These new samples are modified in a way that lets the model to learn also different representations of the same sample, a sort of “real training”, for images for example there could be used cropped samples or maybe over-exposed ones.

A similar but slightly different concept stands at the basis of *Adversarial Training* in which samples generated with one or more real attack algorithms are used instead of the augmented ones. This technique has proven to produce great benefits in terms not only of adversarial robustness but also of clean samples accuracy. It has been noticed, in fact, that a good adversarial training is beneficial also for the normal tasks of a network resulting in an improved clean samples accuracy. Many strategies have been proposed during the years, everyone focusing on different aspects of the training phase like the amount and type of adversarial samples to use, how long to train, whether to fine-tune or not, etc. References to the next solutions have been found in [41], [46], [48].

One of the first contributions to this theme is the one by Goodfellow et al. [49] in which they suggest to include into the training set also adversarial samples crafted with the Fast Gradient Sign Method and to add a regularization term into the objective function. While effective this method worked only against FGSM attacks, in fact in the following years some variants and improvements were investigated. Madry et al. [19] in 2017 proposed another AT approach that used Projected Gradient Descent PGD instead of FGSM. In their work they suggest to use this attack instead of the single-step ones as it lets the model to be resistant also to a variety of similar attacks; also, it is suggested to train the model only on this data instead of adding it to the training dataset. This method has proven to be very effective but there are some pitfalls. First of all, an iterative attack like PGD is very heavy, the computational costs of the crafting phase increase by a *k* factor, being *k* the number of the steps of the PGD algorithm; second, this method protects against L_{inf} attacks making the net still vulnerable to other L_p-norm attacks.

These two papers set the basis for all the upcoming studies about adversarial training, as they are, still today, two milestones in this field. Different paths have been explored since then with researchers trying to design the best solution combining new adversarial generation algorithms and training workflows.

In *Ensemble adversarial training* the idea is to craft adversarial samples from a set of pretrained models using either single or multi-step attacks. This method makes the network to be more robust thanks to the different types of samples used during training but the models from which this data come from must be non-overlapping in terms of prediction and adversarial landscapes, this can be obtained adding some regularization and diversity terms. EAT has proved to be an effective solution to achieve robustness against different types of attacks and settings also using the ImageNet dataset.

Generative adversarial training employs nondeterministic generators to craft new data. GANs and similar algorithms can be used to craft samples that resemble the ones generated by actual attacks and use them in AT.

Efficient/Accelerate adversarial training focuses on the efficiency problems of PGD-AT proposing new alternatives to achieve the same results with less resources. Some studies have found that gradients could be reused while generating adversarial samples and that also the initializations of the generation algorithms play a concrete role into the efficiency task. Others focused on adding regularization terms or strategies that could help reducing the amount of resources needed.

Curriculum-based adversarial training starts with the intuition that samples generated by strong attacks cross the decision boundaries so much that they even reach natural data, resulting in an overfitting when supplying these samples to the adversarial training procedure. That's why researchers came along with adjustments to the classic AT approach, in particular Curriculum AT stops the generation phase earlier (i.e. at a lower number of PGD steps) with respect to PGD-AT, precisely when a high accuracy is reached; Friendly AT, on the other hand, stops when samples reach the boundary of the decision area. These approaches seem to exploit the fact that networks generalize better with these weak attacks and they also use less resources than other techniques.

At the moment Adversarial Training seems the most effective defence so many studies and researches are being done on this field.

3.1.3.2 Distillation

Another training technique used to improve the adversarial robustness of a model is *Defensive distillation*. In their work [50] Papernot et al. investigated the possibility to use distillation also for adversarial defensive goals and they found that this solution can help in counteracting these kinds of attacks. The idea behind distillation is that networks can transfer their knowledge to other ones during training time, this was meant to happen particularly between nets of different size. This procedure, then, has been adapted to fit the adversarial setting, instead of two different networks now the knowledge is shared between the same model. This is done to reduce variations around the input by resizing the internal gradients, in practice the model is smoothed with respect to its first version making it less sensitive to adversarial samples.

3.1.3.3 Other learning paradigms

Another work from Papernot et al. proposed a new technique to obtain private NN, the *Private aggregation of teacher ensembles PATE* [51]. With PATE is possible to ensure privacy of the data used in training exploiting a teacher-student architecture.

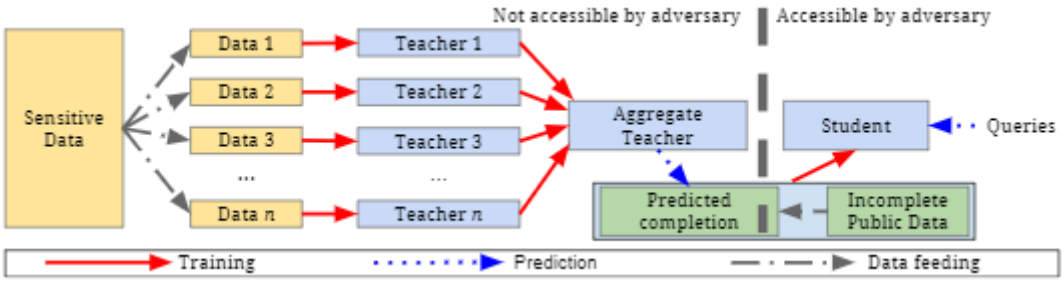

Figure 3.2 - PATE architecture from Scalable Private Learning with PATE, 2018, arxiv.org/abs/1802.08908

An ensemble of models called "teachers" learn from disjoint sets of sensitive data and then these are used to train the final model, called "student", by aggregating the teachers' predictions and feeding these to the student in a black-box way. The answers aggregation is added with Laplacian noise to obtain differential privacy guarantees. The overall architecture is showed in Figure 3.2 - PATE architecture from Scalable Private Learning with PATE, 2018, arxiv.org/abs/1802.08908Figure 3.2.

Another learning method that ensures data privacy is *Federated Learning* [52]. In this setting a model is trained leveraging computations made by the data owner in its local environment.

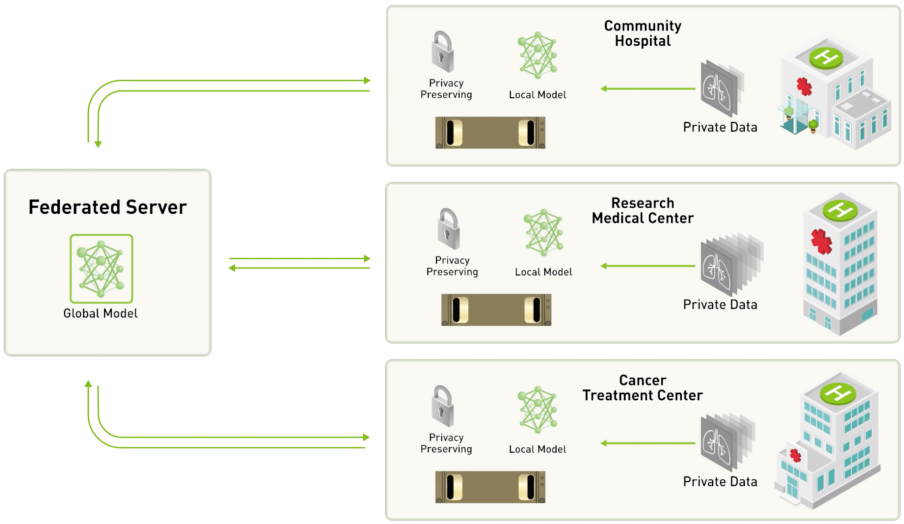

Figure 3.3 - Federated learning architecture from What Is Federated Learning?, 2019, blogs.nvidia.com/blog/2019/10/13/what-is-federated-learning/

As shown in Figure 3.3, data owners train a local model with their sensitive samples and then share updates of the model (like parameters, gradients, etc.) to the central server which is in charge of the aggregation of this new information and of the subsequent update of the global model. In this setting, so, data never leaves the users' devices enabling a privacy-preserving training procedure that doesn't use directly the sensitive samples. Unfortunately, in the standard formulation this paradigm is not so secure as claimed. In a recent paper [53], researchers found that the privacy claims of FL are not so robust as previously thought. In the standard approach all the security reasons were analysed assuming an honest-but-curious actor which couldn't harm the procedure. In their study, on the other hand, they modelled a malicious actor that operates as the central party. In this situation it has been found that updates from the decentralized peers could leak some information about the membership of some data points and that this leakage could be even incremented with maliciously crafted updates from the central server, potentially leading to a full disclosure of some samples or mini-batches in very small time. Defences against these flaws are not definitive as it still is an active area of research. DP has been investigated but as the central party could be the malicious actor of this setting, no guarantees can be made about the security measures implemented; also, the more DP added the more degraded the predictions are. Privacy in depth solutions are another possible defence but, once again, there are some pitfalls. Techniques like Trusted Execution, Homomorphic encryption and others are for sure good solutions but they are still bounded to a correct and benign implementation and, also, they add overhead on the edge peers.

3.1.4 System's security

Not strictly related to AML but still coherent with the security analyses done until now is *System's security.* It has been showed that many solutions can be applied to protect a ML model under different aspects like for example its training phase or the data it uses but it's still important to remember that these NNs are actual pieces of software and for this reason many practises from "standard" cybersecurity should be considered.

Every info about the ML system could be useful for an attacker so design choices and internal details must be kept strictly secret and must be accessible only to the ones who

really operate that system. To do so, a *Role-Based Access Control* model should be applied. RBAC permits to control who can access a certain resource based on which permissions they have, i.e. their role. In this scenario the *least privilege* principle is applied which imposes to share the least information possible with users that permit to operate the system. Implementing roles and privileges helps in keeping the history of accesses, modifications, interactions with the model and its valuable data confining malicious users as soon as they are detected.

Another useful solution is *Homomorphic Encryption* which ensures privacy of the data used in the ML system. With HE data are encrypted in a way that allows computations on it, almost as it was in clear. This could be beneficial when speaking about sensitive data but it must be kept in mind that this is a heavy technique that increases not only the computational costs but also the amount of data handled.

Secure multi-party computation shares ideas with cryptography and FL as it allows the distributed training of a model enhanced with cryptographic properties. In the various studies proposed, the communications (mostly in cloud) are protected with a cryptographic layer that ensures privacy but at a non-ignorable efficiency cost.

Decentralized systems could also use *Trusted Execution Environments* to host the ML code. TEEs are independent operating environments created and managed by the main processor. They are encrypted and periodically checked for integrity and privacy ensuring a stable and untampered safe environment. To offer this kind of guarantees these pieces of software rely on symmetric and asymmetric encryption, public certificates and hashes so once again the computational cost increases.

Apart from specific techniques, general security best practises should always be adopted. As said in the first chapter, every part of a ML system is valuable, from data to outputs so everything must be protected. Every information about the model, its parameters, characteristics, architecture, training procedures, training and testing data and so on are sensitive and extremely valuable for a malicious user so they must be kept secret and shared, if needed, with the least number of agents, being them humans or not.

This concludes the Security by design part of the framework in which some of the most adopted security solutions have been discussed and proposed. In the next section the Security assessment procedure will be depicted.

3.2 Security assessment

Although AML has started to attract attention from companies just in recent times, AI and ML have been around for more years. In today's technological world, a huge number of products features some AI functions but not all of them are well protected against AML threats. Companies are often not prepared or even aware of the vulnerabilities that can affect their assets, especially the older ones. It's for this purpose that a Security assessment procedure is needed, to protect all the AI-powered products that otherwise could become AML targets.

Also, being this a general security framework, it could be used for classic assessment procedures to evaluate the level of security of a ML model and to monitor its robustness against AML attacks over time since every year new discoveries are published, including also advices related to the remediation phase.

The next paragraphs are organized following the steps of the assessment procedure starting with the information gathering phase and finishing with the final results reporting. This section has been designed taking inspiration from pentest standards and classical security assessment frameworks since most of the actions to perform are the same but applied in a slightly different domain. In particular, following its famous ATT&CK framework, MITRE released and maintains its ATLAS spin-off dedicated to machine learning[54]. Much like its inspiration one, this framework contains tactics, techniques and case studies related to ML and its threats.

3.2.1 Preliminary analysis and attack set-up

The first step to take when starting a security assessment is *Preliminary analysis*. To correctly tailor the next phases, an extensive information gathering phase is needed in order to obtain all the possible details about not only the model itself but also the whole system in which it operates. As said in the previous pages, adversarial security is bounded to system security, external operations included, so a comprehensive AML defensive strategy must consider also classical security aspects, and so it should be done when assessing and auditing.

3.2.1.1 Model and data analysis

When obtaining information about the assessment target everything could be helpful to depict a wide and complete scenario. Obviously, model characteristics must be gathered to understand the details of the core asset to test, these include the type of network used so if it is a DNN, FC or a decision-tree based one, the specific architecture used, if possible, e.g. ResNet, VGG, SSD and so on and info about the hyperparameters used like epochs, loss function, optimizer algorithm etc. Also the framework used for the development of the network could be useful. Libraries like Keras, Tensorflow, PyTorch and similar are still pieces of software so potentially vulnerable to some security flaws. Outputs format is another aspect to care about since many black-box attacks exploit information leakages occurred when presenting results to the public.

Insights about data used by the model are beneficial to understand more about network internals and also to better design the next steps. It could be useful to know more about the training and test datasets and what type and characteristics have the inputs to infer some information about the network used, especially when testing for privacy attacks.

3.2.1.2 System analysis

Once analysed model and data, then it comes to the system in which the network is placed. All the steps before and after the model operation should be observed and studied, looking for every useful detail about them. Following the AI lifecycle in [12] and showed in Figure 2.1, data acquisition and manipulation phases are crucial for having a clean and usable base of data for both training and testing. As discussed in Chapter 3.1.2, many threats could be avoided if the operations before the data ingestion are correctly secured so they must also be tested when assessing. After that, architectural data should be analysed to reveal possible flaws that could be exploited when deploying an attack; wrongly managed identities and roles, for example, could favour the attackers' work by enabling direct attack deployment or by letting adversaries gather information.

3.2.1.3 Gathering methods

Generally speaking, every detail about a ML system could be useful during assessment and all this data can be gathered using various methods. Direct knowledge is the best source of information so every technique that permits to have details about the actual

system is to be preferred. These could include, for example, phishing campaigns to penetrate the ML system itself or some node that contains such information, social engineering techniques to infiltrate into the target environment or also simple internet searches looking for studies and publications made by the target owner. When mounting the attack, vulnerabilities repositories and blogs could be beneficial when designing the details of the attack allowing a more fine-grained testing. Finally, if the model is included in an available product it can be reverse engineered to understand every detail of its implementation.

Obviously, these techniques are less important when a white-box assessment is done but they shouldn't be underestimated because AML is an ever-growing field so nothing could be ignored, every information could be useful.

3.2.1.4 Attack design

After this initial study phase the actual attack strategy should be designed, leveraging all the information obtained. The assessor has to choose how many algorithms and of which type to use for the testing but, as a general rule, it should be preferred to use different approaches in the same assessment, all of them regarding one family of attacks. Inspecting a system for only one attack is, in most of the cases, a limited approach, a model could be less or even not sensitive to a particular algorithm and this can lead to a wrong sense of security. When assessing, more than a single approach should be taken, also for what regards knowledge required by the attack and the pattern simulated. The advice is to focus on one of the four families described in Chapter 2.2 and to try different attack algorithms of that type ensuring a wider resistance to a discrete set of attacks and to have a clearer reporting in the end. Not every algorithm is suitable or indicated for every type of system so a conscious selection has to be done, in fact every attack has its own application scenario and is suited for specific types of networks or data. Real world attacks are meant to be used to tamper with systems that accept input from the physical world like sensors measurements, audio, camera recordings and so on while digital attacks craft samples that directly go into the target model; algorithms leverage different information when attacking a ML system, a black-box attack uses output results while white-box ones exploit gradients and internal characteristics; also could be interesting to test the behaviour against targeted and untargeted attacks.

3.2.2 Performance and robustness metrics evaluation

Since the framework consists also in the implementation of defence measures, an analysis of performance and robustness has to be conducted to understand if and in what extent the remediations contributed to protect the model. To do so, the target has to be tested before and after the application of the defences, using the same data and method to have coherent results.

3.2.2.1 Performances evaluation

Performance can be tested in various ways, depending on the task of the network there are different metrics that are suited to measure how good the model behaves.

A popular and widely used performance metric for classification tasks is *accuracy*, defined as:

accuracy = correct_predictions / total_predictions

It measures the number of correctly classified samples against the total number of the predictions. This is the most adopted indicator to understand if the model is acting as expected or not but sometimes, especially for unbalanced datasets, it could be not sufficient. For this reason, also *precision* and *recall* are sometimes used to better understand the network's behaviour.

precision = True_Positive / (True_Positive + False_Positive)

Precision measures the number of samples of a specific class correctly classified against the total number of samples predicted as belonging to that class.

recall = True_Positive / (True_Positive + False_Negative)

Recall defines the fraction of samples from a class which are correctly predicted by the model[55]. A combination of these two metrics is calculated in the *F1-score.* Mostly used in the NLP field, F1 is a special case of the F-score and is defined as:

*F1-score = 2 * Precision * Recall / (Precision + Recall)*

In object detection tasks, usually *mAP* is the chosen metric. In object detection a network identifies objects in images by drawing a square around them; during training these predicted boxes are compared with the ground truth ones and if their area intersects for more than a defined threshold then the prediction is considered correct. Average Precision measures the correlation between precision and recall calculated over the predictions made by the NN, more specifically AP and mAP (mean AP) are derived from the precision-recall graph.

Common metrics in NLP are *BLEU* and *WER*[56]. BLEU is defined as

*BLEU(N) = Brevity_Penalty * Geometric_Average_Precision_Scores(N)*

and is the combination of two factors. Brevity penalty is a coefficient that penalizes short predicted sentences that would have a high precision, it is multiplicated by the average precisions of the predicted N-grams. N-grams are partitions of length N of a predicted sentence and they are compared with the ground truth phrase to calculate precision scores. Usually BLEU is computed over all the predicted essay and not over single sentences.

BLEU is suited for text generation tasks while for automatic speech recognition the most used metric is WER.

Word Error Rate measures the precision of the detected words by analysing the differences between predictions and ground truth as follows:

WER = (Insertions + Deletions + Substitutions) / Total_words_in_transcript

Words that are present in the transcription and not in the true sentence are Insertions, Deletions are the opposite and Substitutions are words present in both but that are different.

In regression, a model predicts a continuous value inferred from some features. A common evaluation metric for these models is the *Mean Squared Error.* MSE measures the average squared difference between the predicted value and the corresponding ground truth. It is defined as follows:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Next to model's performance is system performance with its metrics to be analysed. Especially after the application of the defences, the whole ML system must be monitored to check if the remediations impacted, and eventually by which extent, in the normal system operations in terms of resources spent. To do so, some indicators should be observed, there could be an increased power consumption related to some more complex computations or to some flaw in the defence implementation; an increased memory usage or a longer execution could indicate that the countermeasures impact significantly in the normal model runtime so it should be analysed the sustainability of these changes.

3.2.2.2 Robustness evaluation

Adversarial robustness too could be measured with some specific metrics often related to the task accomplished by the model.

CLEVER has been the first attack-independent robustness metric that could be applied to any NN classifier[57]. It uses a set of adversarial points sampled from a multidimensional ball around a clean sample to evaluate a lower bound for the minimal perturbation needed to fool the classifier for any attack and any L_p norm with $p \geq 1$.

DeepFool is an evasion attack algorithm that has proven to be also useful in determining the robustness of a neural network[58]. The estimation is based on an approximation of the classification space around the sample to evaluate to more easily compute the distance between it and the nearest classification boundary; repeating this procedure for many data points produces the minimal perturbation needed to fool the classifier.

In the previous pages *Differential Training Privacy* has been introduced as a privacy metric. It allows the computation of the maximum risk of membership inference related to a classifier and its training dataset by calculating the difference between the predictions of a classifier trained on the entire dataset and of one trained on the same dataset without the data point to analyse[32]. This metric could be used to estimate the effectiveness of membership inference attacks in order to understand if there could be privacy risks; researchers suggest that classifiers with DTP>1 should not be published.

The *Clique Method* depicted in [59] is useful to compute the adversarial robustness of tree-based models like decision trees and random forests. They formulated an algorithm to calculate, in polynomial time, the robustness of single trees and of the entire model

with better performances with respect to other methods that use linear programming, for example.

3.2.3 Attack execution and vulnerabilities report

Once the information gathering phase is over and the baseline performance of the model is evaluated, then the attacks can be launched. In this phase, so, the model is tested against one or more algorithms in search of some vulnerabilities. As a general rule, one attack type at once should be preferred instead of producing an extensive report regarding all the AML landscape. This methodology could improve results clearness and also ease the decision process of the model owner. Based on the operational scenario and also on one's capabilities and priorities, the model owner could be more interested in protecting for a specific type of threat instead of another or maybe to prioritize the choices and then better organize the defensive strategy.

Once chosen the attack family (evasion, poisoning, extraction or inference), then also the attack pattern should be taken into consideration, trying, if possible, different approaches based on the system configuration and operational scenario. A direct attack should always be tested since it is a common approach and also the most immediate one. Then other patterns could be analysed, first of all the replica and transfer ones, to explore a vaster threat landscape.

For what concerns the actual attack method, more than one algorithm should be tested. This approach allows a better exploration of all the possible threats regarding that type of attack because every algorithm is different and even if the methodology could be the same it always depends on the parameters and settings chosen. Also the same algorithm should be investigated for different configurations, although the one suggested by authors should be the preferred and first implemented one.

After having launched the attacks, all the results and data given by the algorithms must be collected to prepare the vulnerability report. For every attack should be indicated, apart from the effectiveness, also data about the performances and resources needed to launch it, to have a comprehensive picture of the analysed threat. If possible, also a risk study should be included in order to better understand the potential level of danger of the attack and also the probabilities of it. For example, there could be an algorithm

that is very powerful and disruptive but in order to produce that results it needs an enormous amount of resources that makes it quite unlikely to result in an actual threat.

3.2.4 Mitigations decision and application

Having delineated the vulnerabilities landscape of the analysed model, the next step is to apply defensive countermeasures suited for the specific threats that have been found. A defensive strategy should be designed, taking into consideration not only the risk reduction goal but also the constraints that could be posed by the environment in which the model acts or by its owner.

First of all, it's good to remember that there isn't a silver bullet for defending against AML attacks, every defence has a specific target and application scenario so a correct choice has to be made. It's important to keep in mind that some measures could work for more than one type of attack like for example Gaussian noise for extraction and inference. If possible, all these operations should be conducted in a safe environment that could ensure a secure revert of the modifications made to the model and the system in general and also to prevent any interruption of service that could be caused by a malfunction or by the implementation itself.

3.2.5 Post-remediation

Once the defences have been implemented, then they have to be tested to understand if the model still acts as it should, if the integration of the new modules didn't cause any problem and if the defence is effective. To conduct such analysis all the measurements done in the performance evaluations have to be repeated. The robustness metrics should improve demonstrating that the selected countermeasure is effectively working and all the performance metrics should hopefully remain the same or at least not get so much worse. A decrease in overall performance or accuracy of the model is possible, all the added modules imply more computations from a potentially already heavy piece of software but the goal is to keep such degradation above a certain tight threshold in order to have a secure but still usable algorithm.

3.2.6 Wrap-up and delivery

The final step when assessing and consolidating the security of a ML system consists in reporting all the analysis to the model owner. In this phase all the gathered data should be elaborated and presented to the customer in a clear and understandable way, highlighting the results of the robustness analysis and the improvements brought by the applied defences. The produced document should contain also the vulnerability report produced before the attack execution and, if needed, some operational advices on how to maintain the applied defences.

3.3 Tools

There are different tools and libraries that could be used to conduct a security assessment or when designing a secure ML system.

Probably the most complete and powerful AML library at the time of writing is Adversarial Robustness Toolbox [60] by IBM. ART is a Python library that allows to mount attacks and implement defences on a model in an easy and framework independent way. All is based upon its Estimator objects, a collection of wrapper classes that make importing a ML model an easy job. These Estimators abstract the actual framework-specific implementation in order to let the user attack and defend the model without writing specific code but just ART instructions. All the attack algorithms are wrapped in ready to use functions which need only some parameters to properly mount the attack, and there are algorithms also for tabular and audio data. Some defences, like for example pre and post-processing layers, can be attached to the model when declarating it or with just an assignment instruction making the defensive phase easier. In addition to attacks and defences, divided into the 4 categories shown in Figure 2.4, ART offers some utilities to load common datasets like MNIST and to calculate robustness metrics like the ones showed in Chapter 3.2.2.

There are other libraries related to ML security like for example CleverHans that permits to benchmark machine learning systems' vulnerability to adversarial examples [61]. It currently supports JAX, PyTorch and TensorFlow frameworks and features attacks against ML to help benchmarking procedures.

Foolbox is another library that lets users to attack easily ML models[62], in its latest version it has re-designed from scratch to better support PyTorch, TensorFlow and JAX frameworks.

With similar goals also a CLI based tool has been developed by Microsoft called Counterfit. It provides a generic automation layer for adversarial AI frameworks like ART and TextAttack helping organizations conduct AI security risk assessments[63]. Counterfit is environment/model/data agnostics which means that it can work with any type of model using any type of data hosted in any type of infrastructure thanks to its modularity. Every target, in fact, has to be declared in an API fashion, with specific interfaces that allow the interaction between the attack and the model itself.

The performance monitoring step could be carried on using different solutions.

The one chosen for the use cases section of this work is Graphsignal, a ML profiler that uses the built-in profilers as well as other tools to enable automatic profiling in any environment without installing additional software[64]. Via simple API calls users can setup the profiler and attach it to any phase of their ML task, with the possibility to add also some metadata to better characterize the results. All the measurements are sent to the Graphsignal servers to be elaborated and presented in the web dashboard provided to every user.

Local approaches are possible too, using for example Tensorboard or NVIDIA CUPTI. These solutions provide many data and tools to monitor every aspect of a ML task, from the inference and training performances to the model functionality itself. Although these libraries may seem harder to use, they enable users to deeply analyse their ML system also for debugging purposes.

Chapter 4
Use cases

To better understand how to put in practice all the concepts and advices seen in the previous pages some use cases have been developed. In particular, for every type of AML attack a notebook implementing a possible security assessment workflow has been produced. Also a notebook depicting a realistic security by design scenario in which a company wants to secure a ready to be deployed ML model has been produced. These programs have been developed using Keras as ML framework, ART for the AML part and Graphsignal to monitor the performances of the networks.

For their quality and innovation, these applications have been presented during the 2022 Reply Xchange, the annual event organised by Reply where IT peers and creative thinkers gather to discuss how innovation and technologies are changing the world. In particular, Blue Reply IT during the dates in Munich and Milan showed all the laboratories developed by thesis students and internals including the following use cases and the phishing attack mentioned in Chapter 1.

4.1 Evasion

For the evasion use case it has been chosen the FGSM algorithm by Goodfellow et al. [49], a white-box attack that uses a linearization of the cost function of the model to compute the perturbation layer to apply. Custom adversarial training has been adopted to defend along with Defensive Distillation to smooth the prediction landscapes. CIFAR-10 dataset has been used for all training and testing operations.

4.1.1 Environment preparation

This attack has been launched against a simple network instantiated using Keras with the following architecture:

```
1.   model = Sequential()
2.   model.add(Conv2D(32, (3, 3), padding="same", input_shape=x_train.shape[1:]))
3.   model.add(Activation("relu"))
4.   model.add(Conv2D(32, (3, 3)))
5.   model.add(Activation("relu"))
6.   model.add(MaxPooling2D(pool_size=(2, 2)))
7.   model.add(Dropout(0.25))
8.
9.   model.add(Conv2D(64, (3, 3), padding="same"))
10.  model.add(Activation("relu"))
11.  model.add(Conv2D(64, (3, 3)))
12.  model.add(Activation("relu"))
13.  model.add(MaxPooling2D(pool_size=(2, 2)))
14.  model.add(Dropout(0.25))
15.
16.  model.add(Conv2D(128, (3, 3), padding="same"))
17.  model.add(Activation("relu"))
18.  model.add(Conv2D(128, (3, 3)))
19.  model.add(Activation("relu"))
20.  model.add(MaxPooling2D(pool_size=(2, 2)))
21.  model.add(Dropout(0.25))
22.
23.  model.add(Flatten())
24.  model.add(Dense(512))
25.  model.add(Activation("relu"))
26.  model.add(Dropout(0.5))
27.  model.add(Dense(10))
28.  model.add(Activation("softmax"))
29.
30.  model.compile(loss="categorical_crossentropy", optimizer="adam",
    metrics=["accuracy"])
31. classifier = KerasClassifier(model=make_model(), clip_values=(min_, max_))
```

CLEVER score computation has been implemented along with the Graphsignal profiling. The robustness score can be calculated over a single element at once so for better results it has been made a for…loop over *num_samples* elements.

```
1.  num_samples = 10
2.  choices = np.random.choice(range(x_train.shape[0]), num_samples, False)
3.  tot = 0
4.  for i in choices:
5.     score = clever_u(classifier, x_train[i], 500, 1024, 5, np.inf)
```

```
1.  graphsignal.configure(api_key="api_key", workload_name="aml", debug_mode=False)
2.  with inference_span(model_name="plain", ensure_profile=True):
3.      preds_good_pre = classifier.predict(x_test)
```

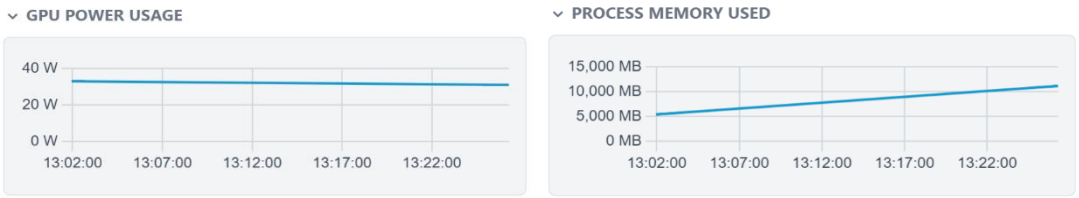

Figure 4.1 - Performance evaluation using Graphsignal

4.1.2 Attack mounting

The FGSM method has been firstly introduced by Goodfellow et al. in 2014 [49]. They found that neural networks are most of the time linear in the sense that they are designed to be easy to optimize by making them work mostly in a linear way. That's why they designed an attack algorithm that could compute an analytical and cheap perturbation that could still harm neural networks. This alteration is computed by linearizing the cost function around the current value of the model's parameters and taking the sign of it following the equation:

$$\eta = \epsilon \cdot sign(\nabla_x J(\theta, x, y))$$

Thanks to the abstractions introduced by ART the attack mounting phase is very simple. It's in fact sufficient to instantiate the attack algorithm and calling the generation method in order to obtain the perturbed samples.

```
1.  attack = FastGradientMethod(estimator=classifier, eps=0.2)
2.  x_test_adv = attack.generate(x=x_test)
3.  x_train_adv = attack.generate(x=x_train)
```

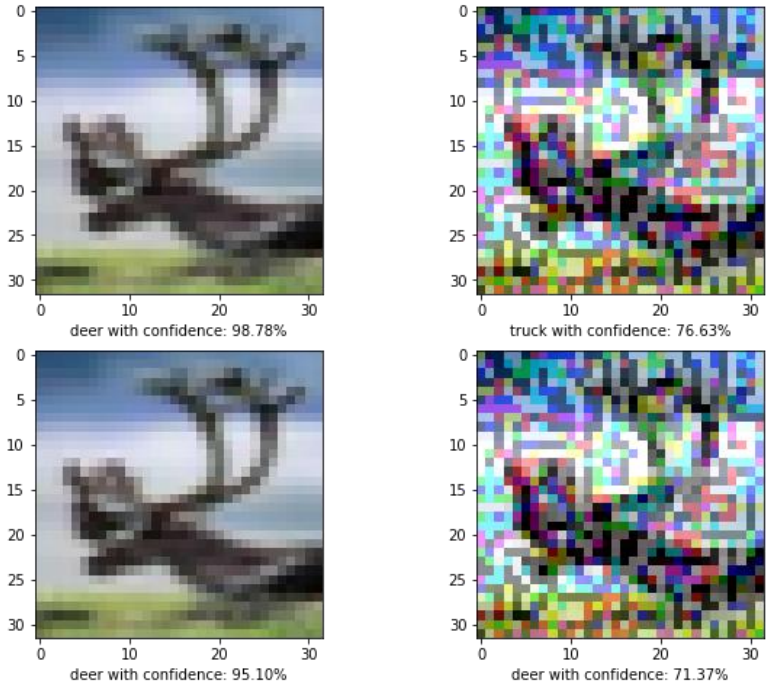

Figure 4.2 - Clean and adversarial samples with their predictions (top: plain classifier, bottom: robust classifier)

4.1.3 Defence implementation

The samples created before are used to adversarially train a new instance of the same network. This new classifier will be trained using an extended dataset made by the plain one with the adversarial samples appended. After the fitting phase, Defensive Distillation is applied. This technique is used to smooth the prediction landscapes of the classifier and to improve its resilience to adversarial samples. This new model is tested for robustness, performances and accuracy as done before.

```
1.  x_train = np.append(x_train, x_train_adv, axis=0)
2.  y_train = np.append(y_train, y_train, axis=0)
3.  classifier_robust = KerasClassifier(model=make_model(), clip_values=(min_, max_))
4.  classifier_robust.fit(x_train, y_train, nb_epochs=20, batch_size=128,verbose=0)
5.  dist = DefensiveDistillation(class_rob, nb_epochs=20)
6.
7.  robclass = KerasClassifier(model=make_model(), clip_values=(min_, max_))
8.  classifier_robust = dist(x_train, robclass)
```

Classifier	Acc. clean	Acc. adversarial
Plain	80.77%	14.10%
Robust	74.89%	68.52%

Table 1 - Accuracy values for plain and robust classifiers evaluated against original dataset and adversarial samples

As shown in Table 1, even a home-made adversarial training could be beneficial. The robust classifier, even if experienced a decrease in the clean samples accuracy, gained a huge improvement in the adversarial accuracy making the predictions on both good and malicious data almost equally trustable.

4.2 Poisoning

The poisoning use case has been implemented using the Hidden Trigger Backdoor attack which allows to introduce trigger specific behaviour into the model without ever showing it. Spectral signatures defence is tested but, as also stated by researchers [65], this attack is robust against most of the defences, included this one.

4.2.1 Environment preparation

For this use case a simple network made of 11 hidden layers has been instantiated with Keras as chosen ML framework. After loading all the libraries and having declared all the needed variables, the ART classifier is created, trained and evaluated.

```
1.  model = models.Sequential()
2.  model.add(layers.Conv2D(32, (3, 3), padding="same", activation='relu', input_shape=
    x_train.shape[1:]))
3.  model.add(layers.Conv2D(32, (3, 3), activation='relu'))
4.  model.add(layers.MaxPooling2D(pool_size=(2, 2)))
5.  model.add(layers.Dropout(0.25))
6.  model.add(layers.Conv2D(64, (3, 3), padding="same", activation='relu'))
7.  model.add(layers.Conv2D(64, (3, 3), activation='relu'))
8.  model.add(layers.MaxPooling2D(pool_size=(2, 2)))
9.  model.add(layers.Dropout(0.25))
```

```
10. model.add(layers.Flatten())
11. model.add(layers.Dense(512, activation='relu'))
12. model.add(layers.Dropout(0.5))
13. model.add(layers.Dense(10))
14.
15. model.compile(loss=losses.CategoricalCrossentropy(from_logits=True), optimizer="ada
    m", metrics=["accuracy"])
16. classifier = KerasClassifier(model=model, use_logits=True, clip_values=(min_, max_)
    )
17. classifier.fit(x_train, y_train, nb_epochs=20, batch_size=128, verbose=0)
18. predictions = classifier.predict(x_test)
19. accuracy = compute_accuracy(predictions, y_test)
```

4.2.2 Attack mounting

Hidden Trigger Backdoor is a particular type of poisoning attack as it is claimed to be often undetectable. Since this is a trigger attack, the network is tricked to learn a different behaviour upon the presence or not of a specific patch which is applied by the PoisoningAttackBackdoor method.

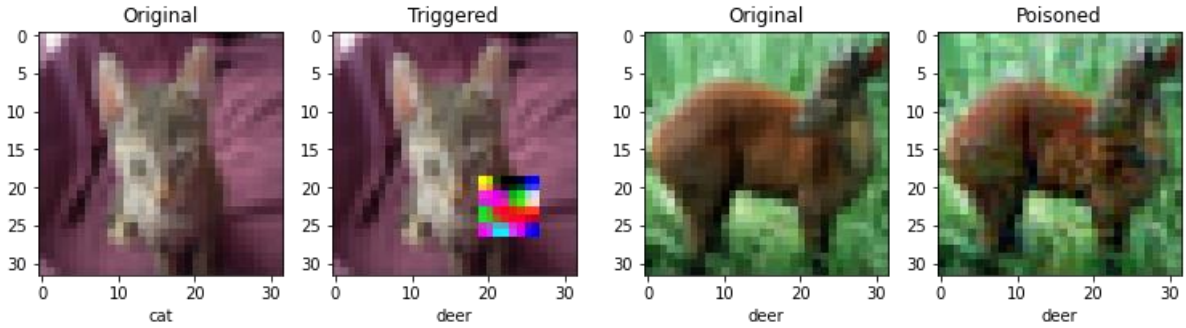

Figure 4.3 - On the left a triggered image with the predicted class, on the right a poisoned sample with its original version

The attack by Saha et al. crafts poisoned samples that are close to target images in the pixel space and also close to source images patched by the trigger in the feature space [65]. These images are labelled correctly so a visual inspection can't identify them (as could be seen in Figure 4.3).

Since this is a targeted attack it's necessary to declare the source and target classes as an array of labels. In addition to that, some parameters of the attack and the trigger application method have to be instantiated.

```
target = np.array([0,0,0,0,1,0,0,0,0,0])
source = np.array([0,0,0,1,0,0,0,0,0,0])
patch_size = 8
x_shift = 32 - patch_size - 5
y_shift = 32 - patch_size - 5

def mod(x):
    original_dtype = x.dtype
    x = perturbations.insert_image(x, backdoor_path=path,channels_first=False, random=False, x_shift=x_shift, y_shift=y_shift,size=(patch_size,patch_size), mode='RGB', blend=1)
    return x.astype(original_dtype)

backdoor = PoisoningAttackBackdoor(mod)
poison_attack = HiddenTriggerBackdoor(classifier, eps=16/255, target=target, source=source, feature_layer=9, backdoor=backdoor, learning_rate=0.01, decay_coeff=.1, decay_iter=1000, max_iter=3000, batch_size=25, poison_percent=.3)
poison_data, poison_indicies = poison_attack.poison(x_train, y_train)
```

A copy of the previously created classifier is then trained on the clean dataset and fine-tuned using the crafted malicious samples in order to poison the model.

```
poison_x = np.copy(x_train)
poison_x[poison_indicies] = poison_data
finetune_model = layers.Dense(10)(model.layers[-2].output)
finetune_model = Model(inputs=model.inputs, outputs=finetune_model)
lr = 0.5
optimizer = tf.keras.optimizers.Adam(lr=lr)
finetune_model.compile(loss=losses.CategoricalCrossentropy(from_logits=True), optimizer=optimizer, metrics=["accuracy"])
finetune_classifier = KerasClassifier(clip_values=(min_, max_), model=finetune_model, use_logits=True)
finetune_classifier.fit(poison_x, poison_y, nb_epochs=1, verbose=0)
```

Plain classifier accuracy	Poisoned classif. accuracy	Poisoned classif. acc. on triggered samples
79.58%	78.82%	18%

Table 2 - Accuracy values before and after poisoning

After poisoning the new model performs poorly on the source class experiencing a sensible class accuracy decrease while the global one is only slightly impacted (see Table 2).

4.2.3 Defence implementation

Not every attack is defendable and the one proposed in this use case is one of them. As stated by researchers, this attack is undetectable by most of the poisoning defences including Spectral Signatures. This method is based on the fact that when a dataset is poisoned two sub-populations of samples are created. One of them, the largest one, contains all the clean well-labelled samples, the other one is composed by corrupted mis-labelled points. If the variances of the feature representations of the two sets are well-separated then a poisoning attack has been found.

Poison samples found	Poison samples not found	False positives
399	1101	11981

Table 3 - Results of the Spectral Signatures detection method

Unfortunately, since there isn't much separation between the poisoned and clean samples (as could be seen in Figure 4.3) the detection doesn't perform well, less than half of the poisoned samples are found (see Table 3).

4.3 Extraction

The model extraction use case has been implemented using the CopycatCNN algorithm which allows the copy of an existing model by just querying the target. To defend against this type of attack a Reverse Sigmoid postprocessing layer has been added to the baseline classifier. Finally, the same attack is proposed in a different flavour and tested against both plain and robust classifiers.

4.3.1 Environment preparation

For this use case has been chosen MNIST a set of 70.000 handwritten digits as training and test dataset. A simple neural network is created together with the ART classifier and then trained.

```
1.  model = Sequential()
2.  model.add(Conv2D(filters=4, kernel_size=(5, 5), strides=1, activation="relu", input
    _shape=(28, 28, 1)))
3.  model.add(MaxPooling2D(pool_size=(2, 2)))
4.  model.add(Conv2D(filters=10, kernel_size=(5, 5), strides=1, activation="relu", inpu
    t_shape=(23, 23, 4)))
5.  model.add(MaxPooling2D(pool_size=(2, 2)))
6.  model.add(Flatten())
7.  model.add(Dense(100, activation="relu"))
8.  model.add(Dense(10, activation="softmax"))
9.  model.compile(loss='categorical_crossentropy', optimizer="sgd", metrics=["accuracy"
    ])
10. classifier = KerasClassifier(model=model, clip_values=(min_, max_))
11. classifier.fit(x_train, y_train, nb_epochs=5, batch_size=128, verbose=0)
```

4.3.2 Attack mounting

CopycatCNN[17] is a model extraction attack that aims at creating a copy model that behaves very similarly to the target one.

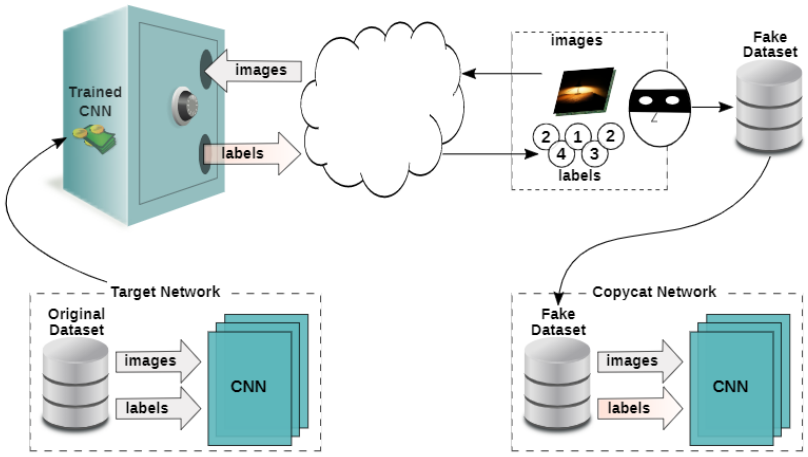

Figure 4.4 - CopycatCNN attack workflow from Copycat CNN: Stealing Knowledge by Persuading Confession with Random Non-Labeled Data, Correira-Silva et al., 2018

To do so, a series of queries is submitted to the target model in order to obtain its predictions and therefore build a dataset of mappings between inputs and outputs of the network to copy. These mappings will serve as training dataset for the copycat model which will learn not the correct pairings of images and labels but only how the target model behaves and produces outputs.

```
1. attack = CopycatCNN(classifier=classifier, batch_size_fit=64, batch_size_query=64,
   nb_epochs=epochs, nb_stolen=len_steal, use_probability=True)
2. class_stolen = KerasClassifier(make_model(), clip_values=(min_, max_))
3. class_stolen = attack.extract(x_steal, y_steal, thieved_classifier=class_stolen)
```

The *len_steal* parameter indicates how much samples to use for the querying phase while *class_stolen* is the classifier to be trained. In the basic attack showed in this section both original data and architecture are used during the attack phase.

Accuracy with 1% of original data	Accuracy with 10% of original data
18.37%	89.17%

Table 4 - Accuracy values for CopycatCNN attack on plain classifier

The baseline model has an accuracy of 96.12%, the copied model gains very good results with just the 10% of original data used (see Table 4).

4.3.3 Defence implementation

Reverse Sigmoid[66] is a defence method that alters the output probabilities.

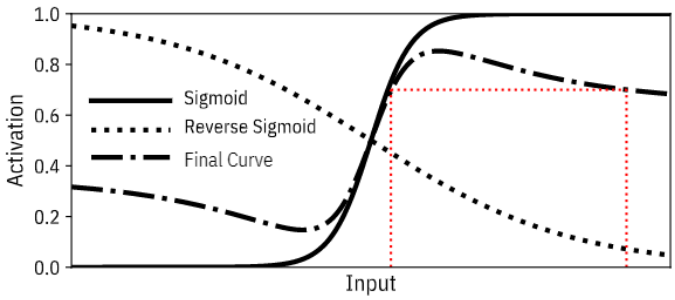

Figure 4.5 - Sigmoid and Reverse sigmoid functions from Defending Against Machine Learning Model Stealing Attacks Using Deceptive Perturbations, Lee et al., 2018

More precisely, a specifically designed function is applied to the predictions in order to affect the attack phase by maximizing the loss calculated by the attacker. This function has the shape of a reverse sigmoid (see Figure 4.5) and thanks to its mathematical properties manages to maintain the meaning of the predictions while discouraging extraction attacks.

Accuracy with 1% of original data	Accuracy with 10% of original data
7.17%	24.37%

Table 5 - Accuracy values for CopycatCNN attack on robust classifier

As could be seen from Table 5 the defence layer accomplishes its task by decreasing the accuracy of the stolen model significantly.

4.3.4 Different attack approach

As stated by researchers, CopycatCNN attack could be run also using a different set of query data and/or a different model architecture. This approach is more realistic in fact most of the times an attacker hasn't the details of the network or dataset used so they have to find the closest solution. To try these different scenarios, then, the algorithm has been tested using the same network as the basic attack but with a different dataset and then, in the second part, a different architecture and dataset are used. The different images are the ones of the CIFAR-10 dataset but center-cropped and gray-scaled to fit the new network.

```
1.  model = Sequential()
2.
3.  model.add(Conv2D(32, (3, 3), padding="same", activation='relu', input_shape=x_train
    .shape[1:]))
4.  model.add(Conv2D(32, (3, 3), activation='relu'))
5.  model.add(MaxPooling2D(pool_size=(2, 2)))
6.  model.add(Dropout(0.25))
7.
8.  model.add(Conv2D(64, (3, 3), padding="same", activation='relu'))
9.  model.add(Conv2D(64, (3, 3), activation='relu'))
10. model.add(MaxPooling2D(pool_size=(2, 2)))
11. model.add(Dropout(0.25))
12.
13. model.add(Conv2D(128, (3, 3), padding="same", activation='relu'))
14. model.add(Conv2D(128, (3, 3), activation='relu'))
15. model.add(MaxPooling2D(pool_size=(2, 2)))
16. model.add(Dropout(0.25))
17.
18. model.add(Flatten())
19. model.add(Dense(512, activation='relu'))
20. model.add(Dropout(0.5))
21. model.add(Dense(10))
22.
23. model.compile(loss=CategoricalCrossentropy(from_logits=True), optimizer="adam", met
    rics=["accuracy"])
24. class_stolen = KerasClassifier(model=model, use_logits=True, clip_values=(min_, max
    _))
```

```
(x_train2, y_train2), (x_test2, y_test2), min2, max2 = load_cifar10()

rgb_weights = [0.2989, 0.5870, 0.1140]

grayscale_images = np.array([np.dot(x[...,:3], rgb_weights)[2:30,2:30,np.newaxis] for x in x_train2])
```

Different dataset	Different dataset and model
70.88%	88.89%

Table 6 - Accuracy values for different attack methodologies

As showed in Table 6 changing model or data still yields to very good results, especially when using a network that better suits the new dataset used.

4.4 Inversion

For the inversion use case the attack by Fredrikson et al., MIFace has been chosen. This algorithm allows the reconstruction of the data points used during the training of the target network. In order to defend against this attack, four different defences have been tested and compared to understand which one works best.

4.4.1 Environment preparation

The chosen dataset is MNIST, a collection of 70.000 images of handwritten digits. After having imported all the needed libraries and having instantiated the dataset the network to be used throughout this notebook is declared, trained and evaluated.

```
model = Sequential([
        Conv2D(16, 8, 2, 'same', activation='relu', input_shape = x_train.shape[1:]),
        MaxPooling2D(2,1),
        Conv2D(32, 4, 2, 'valid', activation='relu'),
        MaxPooling2D(2, 1),
        Flatten(),
        Dense(32, 'relu'),
```

```
8.          Dense(10, 'softmax')
9.          ])
10.
11. optimizer = tf.keras.optimizers.SGD(learning_rate=0.1)
12. loss = tf.keras.losses.CategoricalCrossentropy(from_logits=True)
13. classifier = KerasClassifier(model=model, clip_values=(min_,max_), use_logits=False
    )
14. classifier.fit(x_train, y_train, batch_size=264, nb_epochs=1, verbose=0)
```

4.4.2 Attack mounting

MIFace is a particular version of another model inversion attack from Fredrikson et al. on sensitive health data, in particular it was targeted a model that could predict the dosage of an hospital drug[16]. This algorithm has been subsequently adapted also to a decision tree based classifier and a facial recognition network[15]. MIFace works in a white-box context meaning that the attacker has access to the model. Black-box versions have also been investigated by researchers but they experienced a huge performances impact while the attack was still successful. The algorithm uses the confidence values given by the model to optimize for an image that maximizes these predictions by computing model's gradients.

Thanks to ART the attack instantiation and launch are very simple as shown below.

```
1.  classifier = KerasClassifier(model=model, clip_values=(min_,max_), use_logits=False
    )
2.  classifier.fit(x_train, y_train, batch_size=264, nb_epochs=1, verbose=0)
```

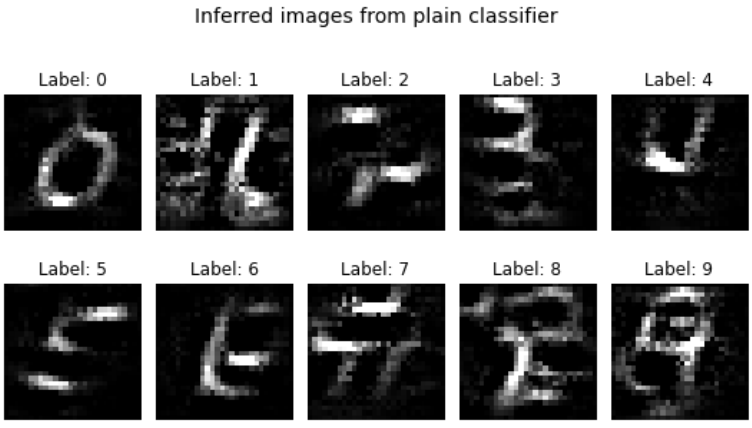

Figure 4.6 - Inferred images from a plain classifier using MIFace attack

4.4.3 Defence implementation

To defend against MIFace attack some gradient masking techniques could be employed. In this notebook have been investigated label smoothing, Gaussian noise, rounded labels and more training epochs.

4.4.3.1 Label smoothing

Smoothing replaces the original one-hot encoded vector of labels with different probabilities calculated with:

$$y_{smooth} = (1 - \alpha) * y_{hot} + \alpha/K$$

being α the smoothing parameter, y_{hot} the label in one-hot encoded format and *K* the number of classes of the model. In this way output labels confuse a bit the attacker by not giving an absolute prediction.

This defence could be implemented by explicitly declaring the loss function to use and passing the smoothing parameter to it.

```
1.  loss2 = tf.keras.losses.CategoricalCrossentropy(from_logits=True, label_smoothing=0
    .1)
2.  model2.compile(optimizer=optimizer, loss=loss2, metrics=['accuracy'])
```

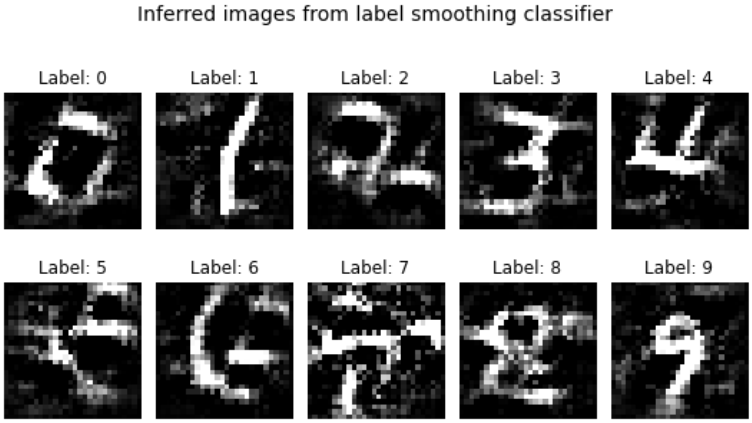

Figure 4.7 - Label smoothing classifier results

As shown in Figure 4.7 the inferred images are more marbled but still quite recognizable.

4.4.3.2 Gaussian noise

It consists in adding to the output probabilities an amount of noise sampled from a Gaussian distribution. This helps in obfuscating the real predictions values while keeping them meaningful.

```
1.  model3.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])
2.  class_gauss = KerasClassifier(model=model3, clip_values=(min_,max_), use_logits=Fal
    se, postprocessing_defences=art.defences.postprocessor.GaussianNoise())
```

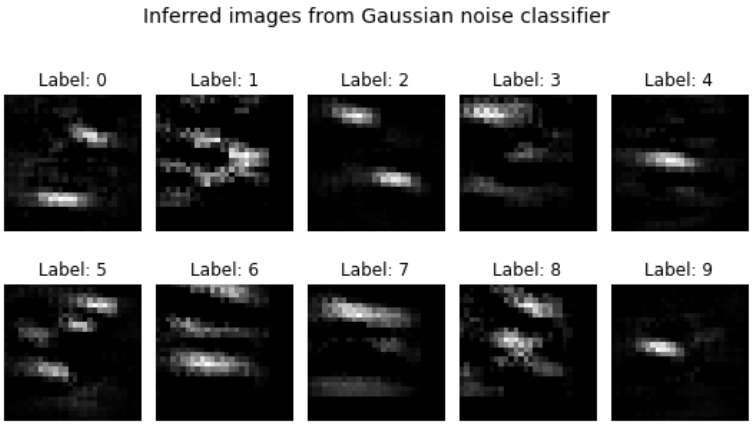

Figure 4.8 - Gaussian noise classifier MIFace attack results

The images inferred from a classifier protected with Gaussian noise are blurry and definitely not recognizable.

4.4.3.3 Rounded labels

A simple rounding operation on the output probabilities could be beneficial in subtracting information from the attacker. This defence, like Gaussian noise, can be implemented as a post-processing layer of the ART classifier.

```
1.  model4.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])
2.  class_round = KerasClassifier(model=model4, clip_values=(min_,max_), use_logits=Fal
    se, postprocessing_defences=art.defences.postprocessor.Rounded())
```

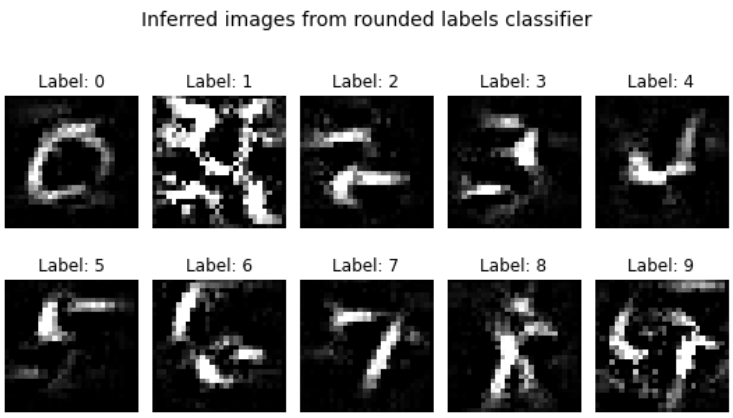

Figure 4.9 – Inferred images from rounded labels

The reconstructed images are less recognizable but still not so much blurry. Different rounding values could be investigated too.

4.4.3.4 More training epochs

Training the model for more epochs, especially for small networks, could help in defending against these attacks. The more time the network spends in learning improves its generalization capabilities making it to get away from the training samples.

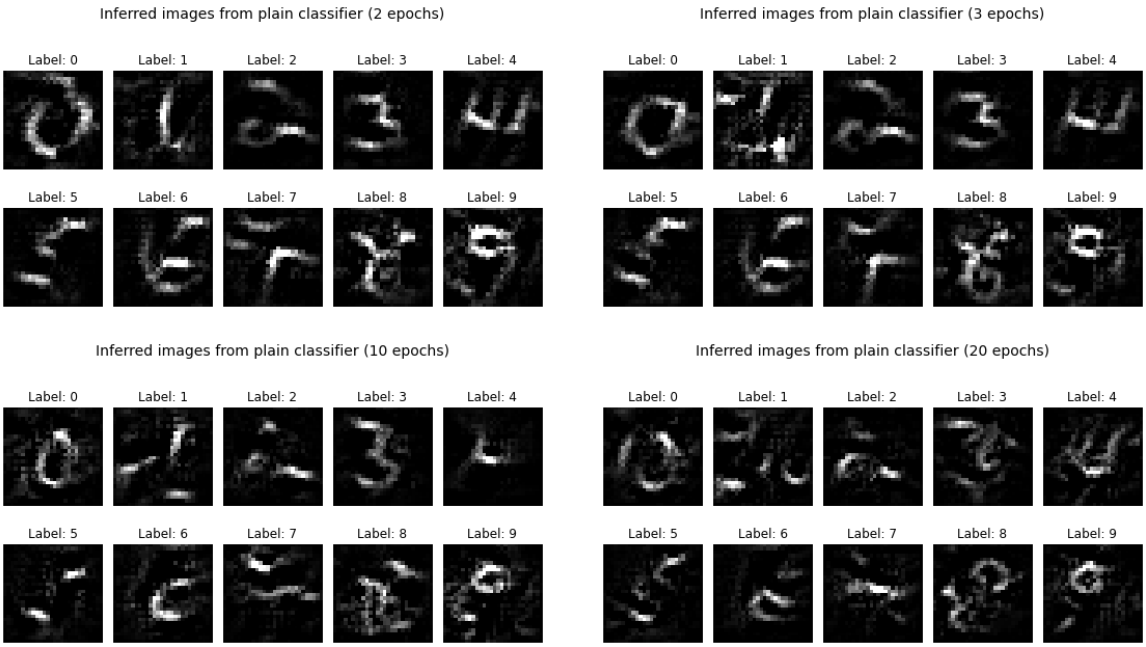

Figure 4.10 - Inferred images from classifiers trained for different epochs

4.5 Security by design

In this last use case different attacks and defences have been investigated. It has been simulated a realistic scenario in which a company wants to deploy a ML system so principles of the security by design had to be implemented.

The entire notebook uses MNIST as training and testing dataset. The poisoning part sees a backdoor poisoning attack against which the STRIP detector has been adopted. For the evasion case, Projected Gradient Descent has been used to craft adversarial samples useful to understand if the Adversarial training has been successful. Inversion and extraction have been defended with Reverse Sigmoid and tested against CopycatCNN and MIFace attacks.

4.5.1 Environment preparation

After the classical imports and declarations, a simple neural network has been instantiated and compiled.

```
1.  model = Sequential()
2.
3.  model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=x_train.sha
    pe[1:]))
4.  model.add(Conv2D(64, (3, 3), activation='relu'))
5.  model.add(MaxPooling2D(pool_size=(2, 2)))
6.  model.add(Dropout(0.25))
7.  model.add(Flatten())
8.  model.add(Dense(num_dense, activation='relu'))
9.  model.add(Dropout(0.5))
10. model.add(Dense(10, activation='softmax'))
11.
12. model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy
    '])
```

4.5.2 Poisoning

For the poisoning part the PoisoningAttackBackdoor[67] has been chosen. It consists in the application of pixel patterns to data samples in order to make the network learning a different behaviour based on if this pattern is present or not.

```
1.  attack = PoisoningAttackBackdoor(lambda x : add_pattern_bd(x, pixel_value=max_))
2.  poison_x, poison_y = attack.poison(src_imgs, target, True)
```

To defend against this attack a poisoning detector has been chosen. Activation Clustering by Chen et al. detects poisonous samples by analysing the network's activations in fact the ones associated with a clean data point are different from the ones of a backdoored sample.

```
1.  ad = ActivationDefence(classifier, x_train, y_train)
2.  (report, clean_list) = ad.detect_poison()
```

The detection upon the training dataset augmented with poisoning images found the 52% of the adversarial samples.

4.5.3 Extraction and inversion

A new instance of the model defined in the first part has been declared comprising also a Reverse Sigmoid post-processing layer to defend against extraction and inversion.

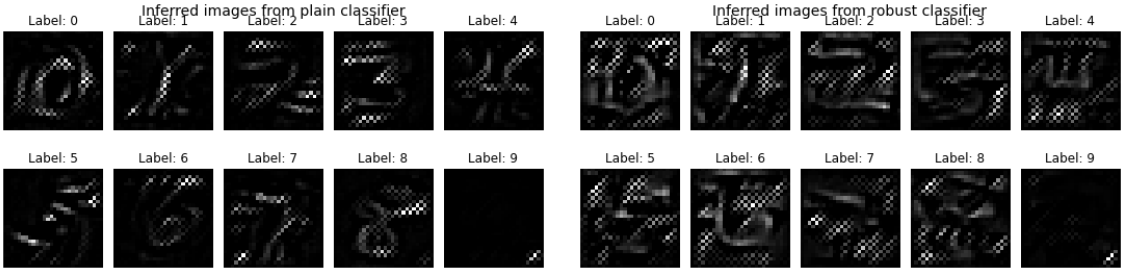

Figure 4.11 - Results of the MIFace attack against plain and robust classifiers

Plain classif. accuracy	Robust classif. accuracy
68.12%	34.52%

Table 7 - Results of the CopycatCNN attack against plain and robust classifier

The effectiveness of the defence has been tested against MIFace and CopycatCNN attacks, the results are shown respectively in Figure 4.11 and Table 7.

4.5.4 Evasion

The evasion defence chosen for this use case is Adversarial Training in the Tramèr's flavour. The ART's implementation of this defence allows the usage of one or more attack algorithms for the adversarial samples generation and also the possibility to have an ensemble training. The ratio of the samples to perturb is set to 0.5 meaning that half of the training points are substituted with their adversarial versions.

```
1.  trainer = AdversarialTrainer(rob_class, pgd)
2.  trainer.fit(x_train, y_train, nb_epochs=epochs)
```

```
1.  pgd = ProjectedGradientDescent(classifier, 'inf')
2.  ev_samples = pgd.generate(x_test)
```

The effectiveness of this method is tested crafting adversarial samples using Projected Gradient Descent [19]. PGD is an iterative attack that seems to yield robustness against all first-order attacks. It has been proposed by Madry et al. in their paper as the preferred method to craft adversarial samples to be used in adversarial training.

	Clean samples	Adv. Samples
Plain classif.	97.38%	12.30%
Robust classif.	80.60%	78.80%

Table 8 - Accuracy values of the plain and protected classifiers

Using these data points the protected model outperforms the plain one on the malicious samples while remaining highly performant also on the clean samples.

Chapter 5
Conclusions

In these pages it has been investigated the topic of Adversarial Machine Learning. Adversarial attacks could be classified by the phase in which they operate in Training and Inference attacks. It's been analysed all the different types of them together with the possible defences that could be applied.

Then, the security framework has been introduced focusing on its two main parts, Assessment and Design. For the Security by design section it's been investigated a series of best practises and security measures to be adopted when building a ML system focusing on their four main assets: Model, Data, Training, System. The assessment part, on the other hand, contained the steps to follow when testing the robustness of a ML system with details on how to put in practice that workflow.

Finally, the use cases produced during this thesis work are introduced showing details about the code and its implementation. A notebook for every attack has been developed together with one regarding Security by design.

ML is far from being secure, as emerged from this work. Lots of attacks have been developed and continue to be published, evasion ones are the most famous but other threats must be taken into consideration. The spread of AI-powered products encouraged malicious actors to try to attack them putting pressure onto manufacturers. Many companies struggle with ML security as it is a new and continuously changing field and finding experts is not an easy job. This framework is intended to be a first step towards a unified standard that could help professionals to assess ML models and to secure them since their conception, reducing risks and fixing costs.

Many efforts must still be made towards really secure ML. As in every cybersecurity sector, attackers are often one step ahead of the defenders but with a conscious approach towards AI, companies can protect themselves and their clients. A change of mindset is needed, security must be put at the first place when dealing with AI because

nowadays many products rely on this type of technology, also ones that handle private and sensible data. As every valuable asset, also for ML systems periodic assessments are fundamental to ensure its security over time.

Research is continuing in investigating the vulnerabilities of such products providing numerous stimuli to the whole security community. It's a long job, the space of adversarial attacks is large but the increasing interest on such topics could bring more resources to help researchers finding the best solutions to counteract them.

Next to the actual attack algorithms, software tools should also be developed. Programs and libraries like Counterfit and ART are fundamental but at the moment they are based mostly on community support with all the pros and cons that this brings. Companies should make a common effort in this sense because everyone can be subject to AML attacks, no one excluded.

Bibliography

[1] E. Kavlakoglu, “AI vs. Machine Learning vs. Deep Learning vs. Neural Networks: What’s the Difference? | IBM,” May 27, 2020. https://www.ibm.com/cloud/blog/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks (accessed Mar. 07, 2022).

[2] “The Difference Between AI, Machine Learning, and Deep Learning? | NVIDIA Blog.” https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/ (accessed Jun. 25, 2022).

[3] “DeepMind AI Reduces Google Data Centre Cooling Bill by 40%.” https://www.deepmind.com/blog/deepmind-ai-reduces-google-data-centre-cooling-bill-by-40 (accessed Jun. 27, 2022).

[4] “Amazon scraps secret AI recruiting tool that showed bias against women | Reuters.” https://www.reuters.com/article/us-amazon-com-jobs-automation-insight-idUSKCN1MK08G (accessed Jun. 28, 2022).

[5] “5 AI Failures You Probably Should Know About | by Kurtis Pykes | Towards Data Science.” https://towardsdatascience.com/5-ai-failures-you-probably-should-know-about-417ddebbc323 (accessed Jun. 28, 2022).

[6] bbc.com, “‘Dalek’ commands can hijack smartphones - BBC News,” Jul. 11, 2016. https://www.bbc.com/news/technology-36763902 (accessed Mar. 11, 2022).

[7] K. Eykholt *et al.*, “Robust Physical-World Attacks on Deep Learning Models,” Jul. 2017, Accessed: Mar. 10, 2022. [Online]. Available: http://arxiv.org/abs/1707.08945

[8] E. Ackerman, “Slight Street Sign Modifications Can Completely Fool Machine Learning Algorithms - IEEE Spectrum,” *spectrum.ieee.org*, Aug. 04, 2017. https://spectrum.ieee.org/slight-street-sign-modifications-can-fool-machine-learning-algorithms (accessed Mar. 10, 2022).

[9] B. Vigliarolo, “AI vs AI: New algorithm automatically bypasses your best cybersecurity defenses | TechRepublic,” *techrepublic.com*, Aug. 02, 2017.

https://www.techrepublic.com/article/ai-vs-ai-new-algorithm-automatically-bypasses-your-best-cybersecurity-defenses/ (accessed Mar. 10, 2022).

[10] Reply, “Is machine learning a secure world?” [Online]. Available: www.reply.com

[11] E. Tabassi, K. J. Burns, M. Hadjimichael, A. D. Molina-Markham, and J. T. Sexton, “A Taxonomy and Terminology of Adversarial Machine Learning,” Oct. 2019. doi: 10.6028/NIST.IR.8269-draft.

[12] A. Malatras and G. Dede, “AI cybersecurity challenges: threat landscape for artificial intelligence,” Dec. 2020. doi: 10.2824/238222.

[13] A. Malatras, I. Agrafiotis, and M. Adamczyk, “Securing machine learning algorithms,” Dec. 2021. doi: 10.2824/874249.

[14] K. Warr, *Strengthening Deep Neural Networks Making AI Less Susceptible to Adversarial Trickery*. 2019. [Online]. Available: http://oreilly.com

[15] M. Fredrikson, S. Jha, and T. Ristenpart, “Model inversion attacks that exploit confidence information and basic countermeasures,” *Proceedings of the ACM Conference on Computer and Communications Security*, vol. 2015-October, pp. 1322–1333, Oct. 2015, doi: 10.1145/2810103.2813677.

[16] M. Fredrikson, E. Lantz, S. Jha, S. Lin, D. Page, and T. Ristenpart, “Privacy in Pharmacogenetics: An End-to-End Case Study of Personalized Warfarin Dosing,” in *Proceedings of the 23rd USENIX Conference on Security Symposium*, 2014, pp. 17–32. Accessed: Jul. 04, 2022. [Online]. Available: https://dl.acm.org/doi/10.5555/2671225.2671227

[17] J. R. Correia-Silva, R. F. Berriel, C. Badue, A. F. de Souza, and T. Oliveira-Santos, “Copycat CNN: Stealing Knowledge by Persuading Confession with Random Non-Labeled Data,” *Proceedings of the International Joint Conference on Neural Networks*, vol. 2018-July, Jun. 2018, doi: 10.1109/IJCNN.2018.8489592.

[18] M. Sharif, S. Bhagavatula, L. Bauer, and M. K. Reiter, “Accessorize to a Crime,” in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, Oct. 2016, pp. 1528–1540. doi: 10.1145/2976749.2978392.

[19] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, “Towards Deep Learning Models Resistant to Adversarial Attacks,” *6th International Conference*

on Learning Representations, ICLR 2018 - Conference Track Proceedings, Jun. 2017, doi: 10.48550/arxiv.1706.06083.

[20] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, "Ensemble Adversarial Training: Attacks and Defenses," May 2017, Accessed: Apr. 05, 2022. [Online]. Available: http://arxiv.org/abs/1705.07204

[21] C. Xie, Y. Wu, L. van der Maaten, A. Yuille, and K. He, "Feature Denoising for Improving Adversarial Robustness," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2019-June, pp. 501–509, Dec. 2018, Accessed: Jul. 11, 2022. [Online]. Available: http://arxiv.org/abs/1812.03411

[22] B. Sun, N. Tsai, F. Liu, R. Yu, and H. Su, "Adversarial Defense by Stratified Convolutional Sparse Coding," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2019-June, pp. 11439–11448, Nov. 2018, Accessed: Jul. 11, 2022. [Online]. Available: http://arxiv.org/abs/1812.00037

[23] H. Hosseini, Y. Chen, S. Kannan, B. Zhang, and R. Poovendran, "Blocking Transferability of Adversarial Examples in Black-Box Learning Systems," Mar. 2017, Accessed: Jul. 11, 2022. [Online]. Available: http://arxiv.org/abs/1703.04318

[24] X. Liu *et al.*, "Privacy and Security Issues in Deep Learning: A Survey," *IEEE Access*, 2020, doi: 10.1109/ACCESS.2020.3045078.

[25] E. Raff, J. Sylvester, S. Forsyth, and M. McLean, "Barrage of Random Transforms for Adversarially Robust Defense," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2019, pp. 6521–6530. doi: 10.1109/CVPR.2019.00669.

[26] R. Bao, S. Liang, and Q. Wang, "Featurized Bidirectional GAN: Adversarial Defense via Adversarially Learned Semantic Inference," May 2018, Accessed: Jul. 11, 2022. [Online]. Available: http://arxiv.org/abs/1805.07862

[27] P. Samangouei, M. Kabkab, and R. Chellappa, "Defense-GAN: Protecting Classifiers Against Adversarial Attacks Using Generative Models," *6th International Conference on Learning Representations, ICLR 2018 - Conference*

Track Proceedings, May 2018, Accessed: Jul. 11, 2022. [Online]. Available: http://arxiv.org/abs/1805.06605

[28] A. Mustafa, S. H. Khan, M. Hayat, J. Shen, and L. Shao, "Image Super-Resolution as a Defense Against Adversarial Attacks," *IEEE Transactions on Image Processing*, vol. 29, pp. 1711–1724, Jan. 2019, doi: 10.1109/TIP.2019.2940533.

[29] S. Chen, N. Carlini, and D. Wagner, "Stateful Detection of Black-Box Adversarial Attacks," *SPAI 2020 - Proceedings of the 1st ACM Workshop on Security and Privacy on Artificial Intelligent, Co-located with AsiaCCS 2020*, pp. 30–39, Jul. 2019, Accessed: Jul. 11, 2022. [Online]. Available: http://arxiv.org/abs/1907.05587

[30] S. Tian, G. Yang, and Y. Cai, "Detecting Adversarial Examples Through Image Transformation," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, pp. 4139–4146, Apr. 2018, doi: 10.1609/aaai.v32i1.11828.

[31] B. Liang, H. Li, M. Su, X. Li, W. Shi, and X. Wang, "Detecting Adversarial Image Examples in Deep Neural Networks with Adaptive Noise Reduction," *IEEE Trans Dependable Secure Comput*, vol. 18, no. 1, pp. 72–85, Jan. 2021, doi: 10.1109/TDSC.2018.2874243.

[32] Y. Long, V. Bindschaedler, and C. A. Gunter, "Towards Measuring Membership Privacy," Dec. 2017, doi: 10.48550/arxiv.1712.09136.

[33] F. Boenisch, "A Systematic Review on Model Watermarking for Neural Networks," *Front Big Data*, vol. 4, Sep. 2020, doi: 10.3389/fdata.2021.729663.

[34] O. Schwartz, "In 2016, Microsoft's Racist Chatbot Revealed the Dangers of Online Conversation - IEEE Spectrum," Nov. 25, 2019. https://spectrum.ieee.org/in-2016-microsofts-racist-chatbot-revealed-the-dangers-of-online-conversation (accessed Jul. 18, 2022).

[35] M. Barreno, B. Nelson, A. D. Joseph, and J. D. Tygar, "The security of machine learning," *Mach Learn*, vol. 81, no. 2, pp. 121–148, Nov. 2010, doi: 10.1007/s10994-010-5188-5.

[36] towardsdatascience.com, "Poisoning attacks on Machine Learning," 2019. https://towardsdatascience.com/poisoning-attacks-on-machine-learning-1ff247c254db (accessed Apr. 22, 2022).

[37] Y. Gao, C. Xu, D. Wang, S. Chen, D. C. Ranasinghe, and S. Nepal, “STRIP: A Defence Against Trojan Attacks on Deep Neural Networks,” *Proceedings of the 35th Annual Computer Security Applications Conference*, pp. 113–125, Feb. 2019, Accessed: May 18, 2022. [Online]. Available: http://arxiv.org/abs/1902.06531

[38] M. Jagielski, A. Oprea, B. Biggio, C. Liu, C. Nita-Rotaru, and B. Li, “Manipulating Machine Learning: Poisoning Attacks and Countermeasures for Regression Learning,” Apr. 2018, Accessed: May 13, 2022. [Online]. Available: http://arxiv.org/abs/1804.00308

[39] B. Biggio, I. Corona, G. Fumera, G. Giacinto, and F. Roli, “Bagging Classifiers for Fighting Poisoning Attacks in Adversarial Classification Tasks,” 2011, pp. 350–359. doi: 10.1007/978-3-642-21557-5_37.

[40] B. Tran, J. Li, and A. Madry, “Spectral Signatures in Backdoor Attacks,” *Adv Neural Inf Process Syst*, vol. 2018-December, pp. 8000–8010, Nov. 2018, Accessed: Jul. 19, 2022. [Online]. Available: http://arxiv.org/abs/1811.00636

[41] H. Xu *et al.*, “Adversarial Attacks and Defenses in Images, Graphs and Text: A Review,” Sep. 2019, Accessed: May 19, 2022. [Online]. Available: http://arxiv.org/abs/1909.08072

[42] A. Athalye, N. Carlini, and D. Wagner, “Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples,” *35th International Conference on Machine Learning, ICML 2018*, vol. 1, pp. 436–448, Feb. 2018, doi: 10.48550/arxiv.1802.00420.

[43] F. Boenisch, P. Sperl, and K. Böttinger, “Gradient Masking and the Underestimated Robustness Threats of Differential Privacy in Deep Learning,” May 2021, doi: 10.48550/arxiv.2105.07985.

[44] H. Huang, Y. Wang, S. M. Erfani, Q. Gu, J. Bailey, and X. Ma, “Exploring Architectural Ingredients of Adversarially Robust Deep Neural Networks,” Oct. 2021, doi: 10.48550/arxiv.2110.03825.

[45] A. Noack, I. Ahern, D. Dou, and B. Li, “An Empirical Study on the Relation between Network Interpretability and Adversarial Robustness,” Dec. 2019, Accessed: May 13, 2022. [Online]. Available: http://arxiv.org/abs/1912.03430

[46] K. Ren, T. Zheng, Z. Qin, and X. Liu, “Adversarial Attacks and Defenses in Deep Learning,” *Engineering*, vol. 6, no. 3, pp. 346–360, Mar. 2020, doi: 10.1016/J.ENG.2019.12.012.

[47] B. Wang *et al.*, “Neural cleanse: Identifying and mitigating backdoor attacks in neural networks,” *Proc IEEE Symp Secur Priv*, vol. 2019-May, pp. 707–723, May 2019, doi: 10.1109/SP.2019.00031.

[48] T. Bai, J. Luo, J. Zhao, B. Wen, and Q. Wang, “Recent Advances in Adversarial Training for Adversarial Robustness,” *IJCAI International Joint Conference on Artificial Intelligence*, pp. 4312–4321, Feb. 2021, doi: 10.24963/ijcai.2021/591.

[49] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and Harnessing Adversarial Examples,” *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, Dec. 2014, Accessed: Mar. 29, 2022. [Online]. Available: http://arxiv.org/abs/1412.6572

[50] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, “Distillation as a Defense to Adversarial Perturbations against Deep Neural Networks,” Nov. 2015, Accessed: Apr. 05, 2022. [Online]. Available: http://arxiv.org/abs/1511.04508

[51] N. Papernot, S. Song, I. Mironov, A. Raghunathan, K. Talwar, and Ú. Erlingsson, “Scalable Private Learning with PATE,” Feb. 2018, Accessed: May 18, 2022. [Online]. Available: http://arxiv.org/abs/1802.08908

[52] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, “Communication-Efficient Learning of Deep Networks from Decentralized Data,” *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017*, Feb. 2016, Accessed: Aug. 02, 2022. [Online]. Available: http://arxiv.org/abs/1602.05629

[53] F. Boenisch, A. Dziedzic, R. Schuster, A. S. Shamsabadi, I. Shumailov, and N. Papernot, “When the Curious Abandon Honesty: Federated Learning Is Not Private,” Dec. 2021, doi: 10.48550/arxiv.2112.02918.

[54] MITRE Corporation, “MITRE | ATLAS.” https://atlas.mitre.org/ (accessed Mar. 22, 2022).

[55] S. Minaee, “20 Popular Machine Learning Metrics. Part 1: Classification & Regression Evaluation Metrics | by Shervin Minaee | Towards Data Science,”

towardsdatascience.com, Oct. 28, 2019. https://towardsdatascience.com/20-popular-machine-learning-metrics-part-1-classification-regression-evaluation-metrics-1ca3e282a2ce (accessed Mar. 14, 2022).

[56] K. Doshi, "Foundations of NLP Explained — Bleu Score and WER Metrics | by Ketan Doshi | Towards Data Science," *medium.com*, May 09, 2021. https://medium.com/towards-data-science/foundations-of-nlp-explained-bleu-score-and-wer-metrics-1a5ba06d812b (accessed Mar. 15, 2022).

[57] T. W. Weng *et al.*, "Evaluating the Robustness of Neural Networks: An Extreme Value Theory Approach," *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, Jan. 2018, doi: 10.48550/arxiv.1801.10578.

[58] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "DeepFool: a simple and accurate method to fool deep neural networks," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2574–2582, Nov. 2015, [Online]. Available: http://arxiv.org/abs/1511.04599

[59] H. Chen, H. Zhang, S. Si, Y. Li, D. Boning, and C.-J. Hsieh, "Robustness Verification of Tree-based Models," *Adv Neural Inf Process Syst*, vol. 32, Jun. 2019, Accessed: Aug. 07, 2022. [Online]. Available: http://arxiv.org/abs/1906.03849

[60] "Home - Adversarial Robustness Toolbox." https://adversarial-robustness-toolbox.org/ (accessed Aug. 15, 2022).

[61] N. Papernot *et al.*, "Technical Report on the CleverHans v2.1.0 Adversarial Examples Library," Oct. 2016, doi: 10.48550/arxiv.1610.00768.

[62] J. Rauber, W. Brendel, and M. Bethge, "Foolbox: A Python toolbox to benchmark the robustness of machine learning models," Jul. 2017, doi: 10.48550/arxiv.1707.04131.

[63] "AI security risk assessment using Counterfit - Microsoft Security Blog." https://www.microsoft.com/security/blog/2021/05/03/ai-security-risk-assessment-using-counterfit/ (accessed Aug. 15, 2022).

[64] "Machine Learning Inference Profiler - Graphsignal." https://graphsignal.com/ (accessed Aug. 15, 2022).

[65] A. Saha, A. Subramanya, and H. Pirsiavash, “Hidden Trigger Backdoor Attacks,” *AAAI 2020 - 34th AAAI Conference on Artificial Intelligence*, pp. 11957–11965, Sep. 2019, doi: 10.48550/arxiv.1910.00033.

[66] T. Lee, B. Edwards, I. Molloy, and D. Su, “Defending Against Machine Learning Model Stealing Attacks Using Deceptive Perturbations,” May 2018, Accessed: May 10, 2022. [Online]. Available: http://arxiv.org/abs/1806.00054

[67] T. Gu, B. Dolan-Gavitt, and S. Garg, “BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain,” Aug. 2017, doi: 10.48550/arxiv.1708.06733.