POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

On the resilience and protection of regularization techniques in differential privacy

Tesi di Laurea Magistrale in
Computer Science and Engineering - Ingegneria Informatica

Author: **Marco Giammarresi**

Student ID: 944756
Advisor: Prof. Matteo Matteucci
Co-advisors: Eng. Eugenio Lomurno
Academic Year: 2020-21

Contents

List of Figures

List of Tables

Abstract in lingua italiana

Negli ultimi tempi, i proprietari e gli sviluppatori di modelli di deep learning devono prendere in considerazione più che mai la salvaguardia della privacy dei loro dati di allenamento, di solito in crowdsourcing e contenenti informazioni sensibili, a causa dell'introduzione del GDPR e l'inasprimento delle normative internazionali riguardanti la privacy dei dati.
Il metodo più conosciuto per far rispettare la garanzia di privacy di un modello di deep learning è la privacy differenziale; questo metodo ha dimostrato di avere successo come meccanismo di difesa da diversi attacchi alla privacy contro i modelli, ma il suo lato negativo è di causare un sostanziale deterioramento delle prestazioni del modello.
Nel nostro lavoro, testiamo l'efficacia sia del Differentially Private Stochastic Gradient Descent, l'attuale approccio standard in termini di conservazione della privacy, sia dei regolarizzatori nel difendere i modelli di deep learning e fornire garanzie di privacy; per fare ciò, sottoponiamo entrambi i metodi ad attacchi di inferenza di appartenenza e di inversione del modello.
Mostriamo attraverso uno studio comparativo i difetti della privacy differenziale; i risultati empirici dimostrano il suo impatto significativo sulle prestazioni del modello sotto attacco, sia in termini di livello di accuratezza raggiunto che di durata temporale del processo di addestramento, e la sua scarsa efficacia nel proteggere dall' attacco di inversione del modello. Scopriamo inoltre che l'applicazione di dropout e regolarizzazione L2 allo strato di output del modello bersaglio è il miglior meccanismo di difesa complessivo, mentre la sola regolarizzazione L2 è la soluzione migliore nel caso di un attacco di inversione del modello black-box.

Parole chiave: apprendimento profondo, privacy differenziale, attacchi alla privacy, tecniche di regolarizzazione

Abstract

In recent times, owners and developers of deep learning models must consider more than ever the privacy-preservation of their training data, usually crowdsourced and retaining sensitive information, due to the introduction of GDPR and the tightening of international regulations regarding the privacy of data.
The most widely known method to enforce the privacy guarantee of a deep learning model is Differential privacy; it has proved to be successful as a defense mechanism from several privacy attacks against models, but its downside is to cause substantial degradation of the model's performance.
In our work, we test the effectiveness of both Differentially Private Stochastic Gradient Descent, the current standard approach in terms of privacy preservation, and regularizers in defending deep learning models and providing privacy guarantees; to do this, we subject both methods to membership inference and model inversion attacks.
We show through a comparative study the flaws of differential privacy; the empirical results prove its significant impact on the performance of the model under attack, both in terms of the level of accuracy achieved and time duration of the training process, and its lack of effectiveness in protecting against a model inversion attack. We also find out that applying dropout and L2 regularization to the output layer of the target model is the best overall defense mechanism, while L2 regularization alone is the best solution in the case of a black-box model inversion attack.

Keywords: deep learning, differential privacy, privacy attacks, regularization techniques

1 | Introduction

In recent times, companies have used extensively deep learning models, either provided by third parties or proprietary ones, to make the most of the data at their disposal.
The great capabilities of these models are usually exploited by companies to support the process of decision-making. Because of this fundamental role, companies need the models to produce reliable results; this can be obtained by increasing the amount of data fed to them, given the existing proportionality between deep learning models' performance and the cardinality of the training dataset.
For this reason, companies start collecting more and more data, mainly crowdsourced; the nature of these data forces the companies to ensure privacy guarantees related to their use and sharing.
The relevance that data have as an asset leads malicious agents to target models in order to reveal potentially sensitive information that remains within them when the training is complete and exploit them for harmful purposes. In fact, deep learning models retain meaningful representations of the training data, and they are usually accessible by everyone through online APIs.
So, malicious agents conceive several attacks that exploit knowledge of the trained models to obtain information about their training data. Among them, two types of attacks are the most dangerous: **Model inversion attacks**, whose goal is to reconstruct the original data fed to a model, and **Membership inference attacks**, which aim to recognize if a given data belongs to the model's training set or not. In order to reduce the efficacy of these attacks, a widely spread solution is to implement a known privacy-preservation mechanism, called **Differential privacy**, inside the training procedure of deep learning models. The main advantage of this solution is the guarantee that the privacy leakage at the end of the model's training is limited and measurable.
The main downside of this approach is that the introduction of differential privacy in the training procedure has a significant impact on the model's performance. That happens because the privacy mechanism injects noise into the gradients of the model, and in this way, it can prevent the achievement of the expected accuracy during the learning process. So, a model owner must consider the existence of a tradeoff between privacy level and

utility of the model when he chooses to apply this privacy-preservation method.
Another drawback associated with the implementation of differential privacy in deep learning models is the substantial increase in the time it takes to complete the training procedure. Previous works have demonstrated how deep learning with differential privacy effectively protects models from several attacks willing to infer the training data.
In this work, we investigate the topic of privacy preservation in the deep learning framework; we test the effectiveness of the most known implementation of differential privacy in deep learning models, the Differentially Private Stochastic Gradient Descent (DP-SGD), as a defense mechanism. Our goal is to evaluate the validity of this method in terms of protection against both model inversion and membership inference attacks and measure the impact that its application has on the model under attack, analyzing both the level of accuracy achieved and the training time.
We perform the same analysis considering two regularization techniques, dropout and the L2 regularizer. Their effect on improving a model's generalization capability is widely known, and previous works [36] have confirmed an existing connection between privacy attacks' effectiveness and overfitting in the target model. We conclude our work drawing from this empirical analysis conclusions on the effectiveness of both approaches, differential privacy and regularizers, as overall defense mechanisms.

This document is structured as it follows:

- In Chapter 2 we provide background knowledge on Deep Learning, starting from the motivation behind its development and then retracing the history of the most iconic algorithms, from Perceptron to Convolutional Neural Networks.

- In Chapter 3 we first discuss the topic of privacy attacks against deep learning models, including recent works that propose new implementations of both model inversion and membership inference attacks. Then, we present the necessary elements to understand Differential privacy in its original formulation and introduce its most famous relaxations, (ε, δ)-differential privacy and Renyi differential privacy.

- In Chapter 4 we describe the scenario in which our research develops and explain in detail all the steps that bring us to the confirmation of an existing issue with differential privacy and the proposal of a novel solution in the field of privacy preservation.

- In Chapter 5 we show several experimental results obtained from three different images datasets; we present comparisons between the basic model and several privacy solutions in terms of resistance to both model inversion and membership inference attacks and performance of the target model, together with other interesting insights.

- Finally, in Chapter 6 we summarize the conclusions drawn from our work and discuss possible future developments regarding this field of research.

2 | Background

In this chapter, we introduce some topics whose knowledge is needed to understand the context of our work.
First, we briefly describe the field of study in which our analysis starts and finds its theoretical basis, that is **Deep learning**. After a brief introduction of this framework, we concentrate on the deep learning model that we exploit in our research, the **Artificial neural network**, discussing the motivation leading to its design and illustrating further developments of this architecture in history. We start from the description of the constitutive element of an artificial neural network, the **artificial neuron**, and then illustrate the earliest attempt to place it in a practical application, the **Perceptron**.
We continue describing the first approach to design a neural network, the **Feedforward neural network**, and one of the most known state-of-the-art architectures for deep learning models, the **Convolutional neural network**.
In the last section of the chapter, we describe different techniques and practices developed to enhance the performance of deep learning models during training.

2.1. An introduction to deep learning

Nowadays, machine learning has reached impressive performance levels in a wide set of fields, in some cases competing with human beings' capabilities. However, classical machine learning techniques fail at performing complex tasks, such as extracting information from an image or video. In this scenario, state-of-the-art methods rely instead on deep learning techniques, which outperform past approaches due to their ability to extract information from data automatically.
The deficiencies of machine learning techniques in this scenario are indeed linked to the process of feature extraction from images. Before feeding an image to a classifier, it is necessary to transform it; in particular, the image needs to be compressed into a meaningful representation. The quality of the extracted features influences the model's performance in accomplishing the given task.
The machine learning approach is that models learn the input data representation start-

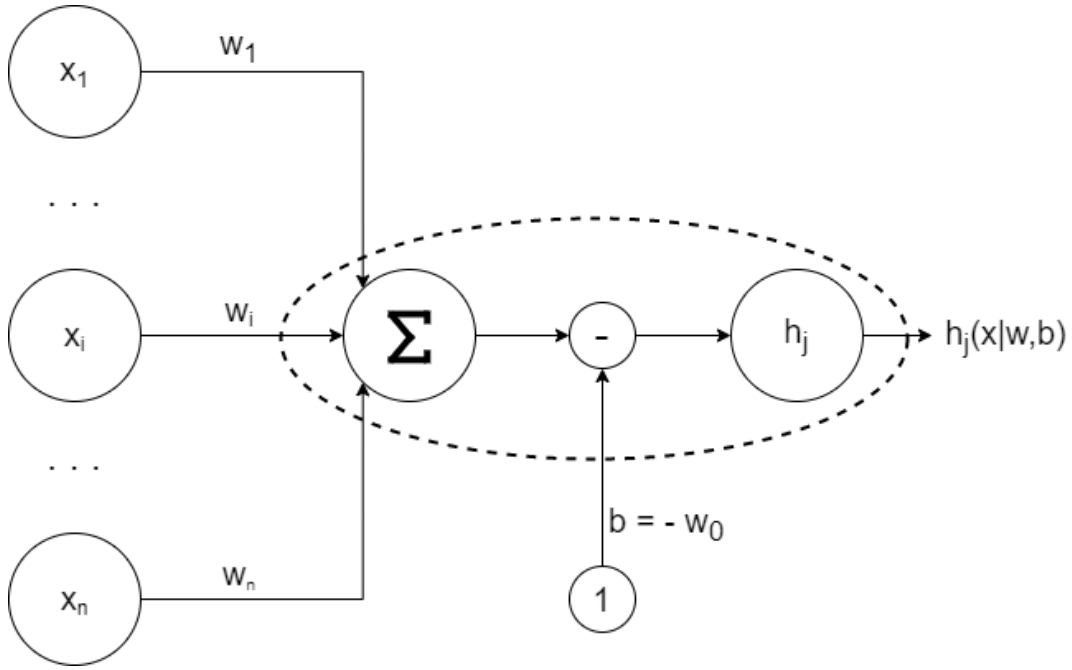


Figure 2.1: An artificial neuron (inside the ellipse) and the computation taking place inside it.

ing from some **handcrafted fixed features**; they are a-priori meaningful characteristics, usually established by engineers and experts, for certain kinds of images, and they can be mapped through an algorithm in a vector form, called feature vector.
The limitations of machine learning techniques are strictly correlated with the flaws of handcrafted features: their design is quite challenging, and they are not easily generalizable to other contexts because they tend to overly fit images used in the training set.
Deep learning was born as a solution to this feature extraction problem; its basic idea is to learn the data representation from the data themselves instead of learning it from handcrafted features, creating **data-driven features**.
This means that we need to design a feature extractor, a model capable of extract representative details from any kinds of image, and train it in order to improve the quality of the feature vector.
Classical machine learning models cannot be used for this purpose; the introduction of a new type of model, the **artificial neural network**, creates the conditions for the development of deep learning techniques.

2.2. The artificial neuron

Before we can define clearly and consistently an artificial neural network and illustrates its structure, we must introduce its constitutive unit, **the artificial neuron**. An artificial neuron is a mathematical model that mirrors the behavior and structure of a biological neuron. In Figure 2.1 we can see the structure of an artificial neuron and the computational steps the inputs (x_i) undergo in order to produce a certain output ($h_j(x|w,b)$). The links between the neuron and its inputs/output represent the synapses, and the weights (w_i) associated with each link indicate the strength with which a certain input influences the final output. The summation node represents the dendrites that collect charges coming from the synapses; this cumulative charge is released once a certain threshold, called **activation function** (h_j), is passed. The formal definition of artificial neuron computation is

$$h_j(x|w) = h_j(w_0 \cdot x_0 + \sum_{i=1}^{n} w_i \cdot x_i) \tag{2.1}$$

with x_0 usually a constant value called **bias**.
An ensemble of artificial neurons receiving the same inputs is called **layer**, and a stack of interconnected layers forms a **Feedforward neural network or Multi-layer perceptron**. The number of layers and the number of neurons inside each layer are two fundamental parameters for a neural network, and studies conducted on them have had a leading role in the evolution of neural networks models. Another significant element in the development of these models is the choice of the learning algorithm that guides the weights update. In the following sections, we describe different milestones architectures in the history of artificial neural networks.

2.3. Perceptron

The perceptron algorithm was invented in 1958 by Frank Rosenblatt [35] and is considered the very first step in the path to neural networks. Perceptron is a single layer neural network with a single neuron, so its structure coincides with the one already seen in Figure 2.1.
The main difference is that the activation function used in the computation is the step function *Sign*, so the calculation of the output becomes

$$h_j(x|w) = Sign(w_0 \cdot x_0 + \sum_{i=1}^{n} w_i \cdot x_i) \tag{2.2}$$

and the output is equal to 0 or 1.

So the perceptron is a linear (binary) classifier for which the decision boundary is the hyperplane

$$w_0 \cdot x_0 + w_1 \cdot x_1 + ... + w_n \cdot x_n = 0 \tag{2.3}$$

In order to adapt its weights for the correct mapping of the input to the output, it has been used a mechanism inspired by the work of Donald O. Hebb [18], that allows the weights optimization towards the classification task, called Hebbian learning.

The mechanism can be summarized with the following formula:

$$w_i^k = \eta \cdot x_i^k \cdot t^k \tag{2.4}$$

with:
- k : current timestep of the procedure
- η : learning rate
- x_i^k : i^{th} input of the perceptron at time k
- t^k : the desired output at time k

The algorithm starts from a random initialization of the weights and then, at each step, corrects their values if the output produced by the perceptron differs from the expected one.
In this way, perceptron can be effectively used as a binary classifier with success; for example, with this model, we can implement boolean operators in a trainable and automated way and with a single hardware unit.
The main disadvantage of the perceptron is that it does not work anymore when the dataset is not linearly separable.

2.4. Feedforward Neural Networks

In order to overcome the limitations of the perceptron, the concept of feedforward neural network is conceived, starting from the simple idea of building a multi-layer perceptron.

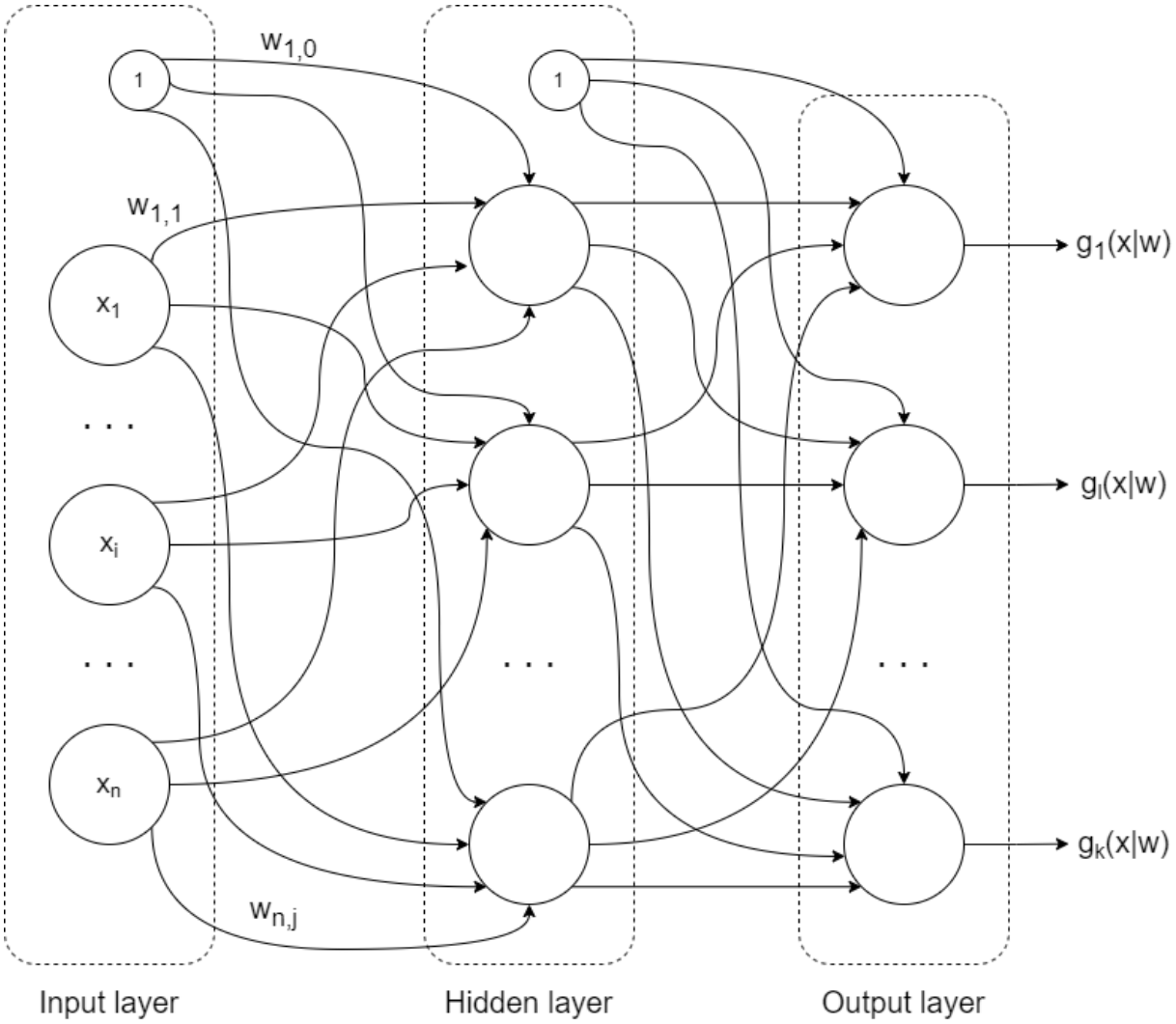
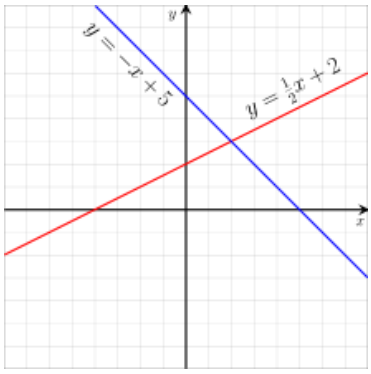


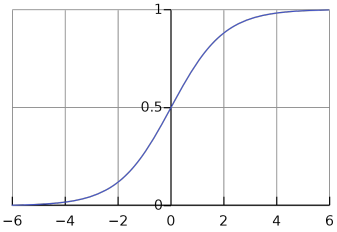
Figure 2.2: Structure of a feedforward neural network.

As we can see in Figure 2.2, this model consists of one **input layer**, one **output layer** and at least one intermediate layer, called **hidden layer** and each neuron belonging to a layer is connected to every neuron of the next layer. Each hidden layer can have an arbitrary number of neurons inside, and the output of a neuron depends only on the previous layers. The other fundamental element in this network is the choice of the activation function to be used in the neurons; in Figure 2.3 we show three functions that can be used as activation functions inside neural networks.

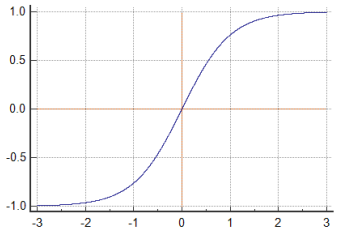
In the perceptron model, using the step function as activation arises an issue: a small variation of the weights can cause a great variation in the output.
This scenario is not ideal for learning the best weights, so, in feedforward neural networks, we usually consider other functions designed in such a way that a little change in the weights leads to a corresponding change in the output; in this way, we allow the model



(a) Linear function $g(x) = x$



(b) Sigmoid function $g(x) = \frac{1}{1+e^{-x}}$



(c) Tanh function $g(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

Figure 2.3: Three examples of activation functions used in feedforward neural networks

to learn the optimal mapping between input and output. Inside the hidden neurons, for example, we can either use Sigmoid or Tanh as activation functions, while for the neurons in the output layer, the choice depends on the task to accomplish:

- Regression: we use the linear activation function because the output domain is the whole $\mathbb{R}$.
- Binary classification: if the two classes are coded as $C_0 = 0$ and $C_1 = +1$ we use sigmoid activation (interpretable as a class posterior probability), while if the coding is $C_0 = -1$ and $C_1 = +1$ we use tanh function.
- Multiple classes classification: the number of neurons in the output layer coincides with the number of classes K (classes are one-hot encoded), and the activation function used is the softmax unit.

$$g_i(\mathbf{x}) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}. \tag{2.5}$$

An interesting property of feedforward neural networks is stated in the Universal approximation theorem [21]:

"A single hidden layer feedforward neural network with S-shaped activation functions can approximate any measurable function to any desired degree of accuracy on a compact set"

The theorem states that it always exists a network with a single hidden layer capable of obtaining any degree of accuracy desired on a certain set of training data. This result is achievable using nonlinear activation functions that allow neural networks to learn a wide scope of data distributions when given appropriate weights. However, this theorem implies that, given a set of training data, it is possible for such a model to learn their true distribution but does not indicate if the learning procedure of the appropriate weights is feasible; the theorem guarantees that a neural network with nonlinear activation function can learn perfectly the training data distribution with a number of epochs tending to infinity.
Finding the best values for a parametric model, given an objective, is an optimization problem that consists in minimizing/maximizing a given function, usually referred as the objective function. Training a neural network can be considered an optimization task, where the parameters to optimize are the weights, and the goal is to minimize the loss between the output produced by the network (given some input data and current weights) and the desired one.
We can formalize this concept mathematically as follows:

Given a training set composed of N tuples (x_i, y_i)*, with* x_i *as* i^{th} *input and* y_i *as* i^{th} *desired output, and assuming to be solving a regression problem, we want* $g(\boldsymbol{x}|\boldsymbol{w}) \sim \boldsymbol{y}$*, that is equivalent to minimize the loss function*

$$L(\mathbf{w}) = \sum_{i=1}^{N} (y_i - g(x_i|\mathbf{w}))^2 \tag{2.6}$$

that is the sum of the squared errors made by our network in predicting the output.

Minimizing a function means calculating its gradient and setting it equal to zero; the issue with feedforward neural networks is that function g is nonlinear, and so, in most cases, it is not possible to solve this equation in closed form. There exist several optimizers able to solve this issue; among them, we focus on one that uses an iterative method to update the weights while minimizing function L, called **Gradient descent**.

In gradient descent algorithm:

1. Start from a random inizialization of the weights;

2. Update weights $\mathbf{w}$ according to the following formula

$$\mathbf{w}^{k+1} = \mathbf{w}^k - \eta \cdot \frac{\partial L(\mathbf{w})}{\partial \mathbf{w}} \mid_{\mathbf{w}^k}, \tag{2.7}$$

where k is the current timestep of the procedure, η is the learning rate and L is the loss function.

The standard gradient descent, also called **batch gradient descent**, calculates the prediction error on each sample in the training dataset and updates the model weights only after all evaluations are completed. This approach guarantees the convergence to a stable value for the gradient, but maintaining the entire dataset in memory can be unpractical. So, two variations of the algorithm are introduced:

- **Stochastic gradient descent** (SGD): the update is performed at each evaluation of a single training sample, speeding up the computation of the gradient and allowing a detailed view of the model improvement due to the frequent updates, at the cost of a great variance in the gradient value and longer run time due to lack of convergence guarantee.
- **Mini-batch gradient descent**: it combines the previous approaches by splitting the training dataset into smaller batches and updating the weights on each of these mini-batches; in this way, it achieves the computational efficiency of SGD and the robustness of the gradient guaranteed by the standard gradient descent.

As we have seen in the gradient descent formulation, to update a certain weight, we must calculate the partial derivative of the loss function with respect to the weight itself. The problem is that the direct computation of this derivative can be difficult, especially if the weight to be updated is located far from the output layer. In order to make this computation efficient, **backpropagation algorithm** is conceived; it simply applies the chain rule property($\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial x}$) to $\frac{\partial L}{\partial w}$, breaking the overall gradient calculation into smaller computations that can be distributed hardware-wise, with the result of speeding up the update of the weights.

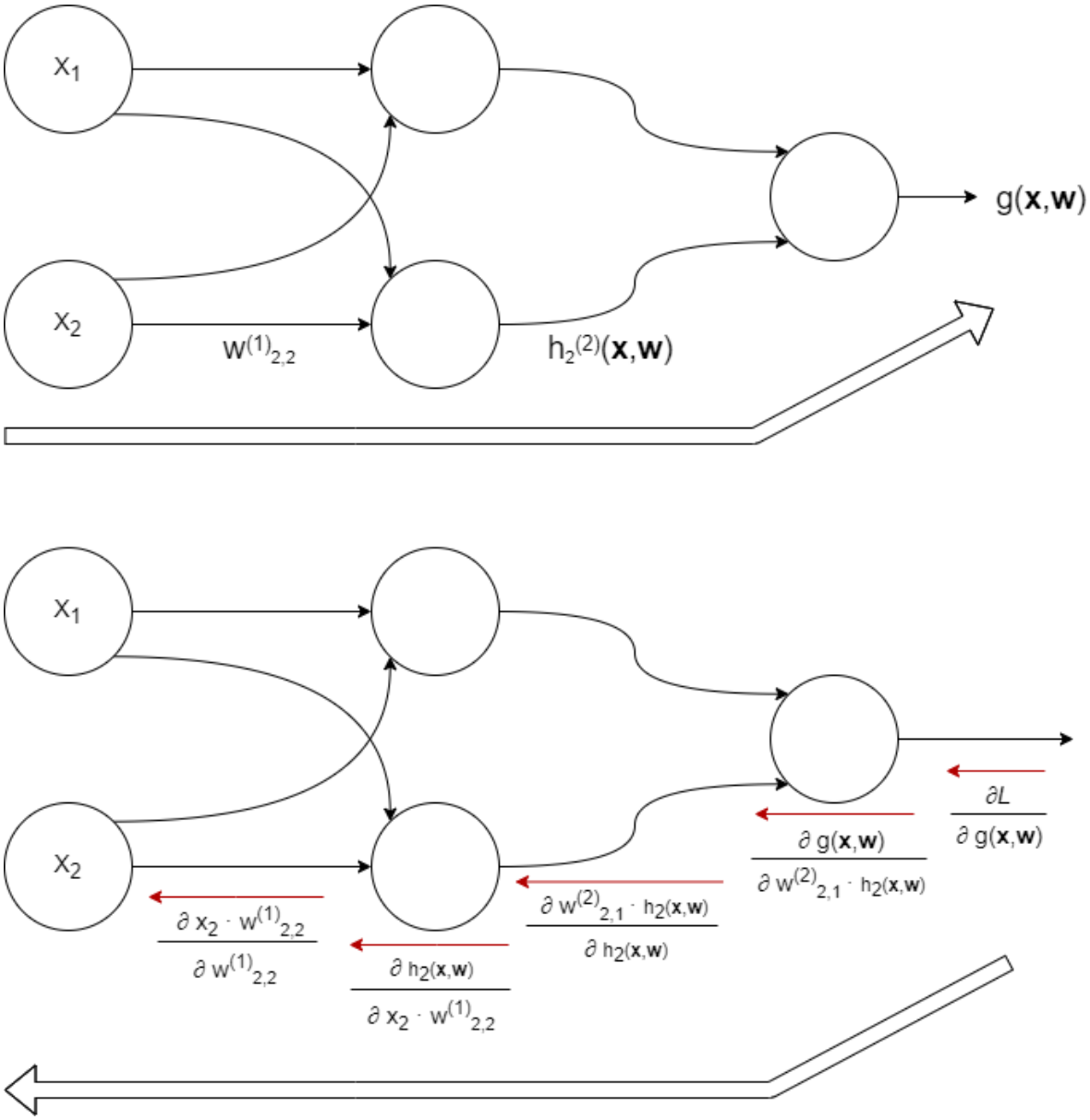


Figure 2.4: Example of feedforward neural network with highlights on the elements that are forwarded along the network during error computation (first image) and on the elements that are sent backward for gradient computation (second image).

To better understand how backpropagation works, we make an example considering the simple feedforward neural network in Figure 2.4.
If we want to update weight $w^{(1)}_{2,2}$, we have to calculate $\frac{\partial L}{\partial w^{(1)}_{2,2}}$, and using the chain rule we obtain

$$\frac{\partial L}{\partial w_{2,2}^{(1)}} = \frac{\partial L}{\partial g(\mathbf{x},\mathbf{w})} \cdot \frac{\partial g(\mathbf{x},\mathbf{w})}{\partial w_{2,1}^{(2)} \cdot h_2(.)} \cdot \frac{\partial w_{2,1}^{(2)} \cdot h_2(.)}{\partial h_2(.)} \cdot \frac{\partial h_2(.)}{\partial w_{2,2}^{(1)} \cdot x_2} \cdot \frac{\partial w_{2,2}^{(1)} \cdot x_2}{\partial w_{2,2}^{(1)}} =$$
$$= \frac{\partial L}{\partial g(\mathbf{x},\mathbf{w})} \cdot g'(\mathbf{x},\mathbf{w}) \cdot w_{2,1}^{(2)} \cdot h_2'(w_{2,2}^{(1)} \cdot x_2 + w_{1,2}^{(1)} \cdot x_1) \cdot x_2 \quad (2.8)$$

It is clear how, in this way, it is easier to calculate the gradient, and it also results more understandable the backpropagation mechanism: to perform the update of every weight in the network, we need pieces of local information obtainable following backward the path between the weight and the network output.

2.5. Convolutional Neural Networks

Computer vision is a field of study that deals with the design of artificial systems capable of extracting information from images. Among all types of data, digital images are the most present and impactful in everyday life, both in the form of photographs and videos. Computer vision was born in the late '60s with the goal of obtaining deeper insights on images with respect to the ones usually achieved by digital image processing; the latter focuses on quality improvement of images (enhancement, correction, restoration...), while computer vision aims to extract three-dimensional patterns from images, representing them as result of the interconnections of more basic elements, and to use these insights to produce decisions. The integration of machine learning techniques inside computer vision has made possible the automated implementation of this feature's extraction process through algorithms, and it has reduced the gap with human's visual capabilities in resolving several tasks (object identification, object localization, semantic segmentation, pose estimation, ...).

The advent of deep learning has brought great benefits to computer vision, with the development of new algorithms that achieve higher performance with respect to previous methods and that, in some cases, extract details that escape the human eye from images. So, the great challenge of computer vision is to develop methods capable of reproducing in an automated way the human vision's capability of extracting detailed information from images and at the same time overcoming its natural limitations.

To achieve this goal, an important step is to manipulate the original image, transforming it into a new one through a procedure called **image processing**. This process is significant because it can highlight hidden features of an image or move the focus on a certain part of it. There exist several image processing operations, and one of the most relevant local spatial transformations of an image is the two-dimensional (2D) **convolution**. The

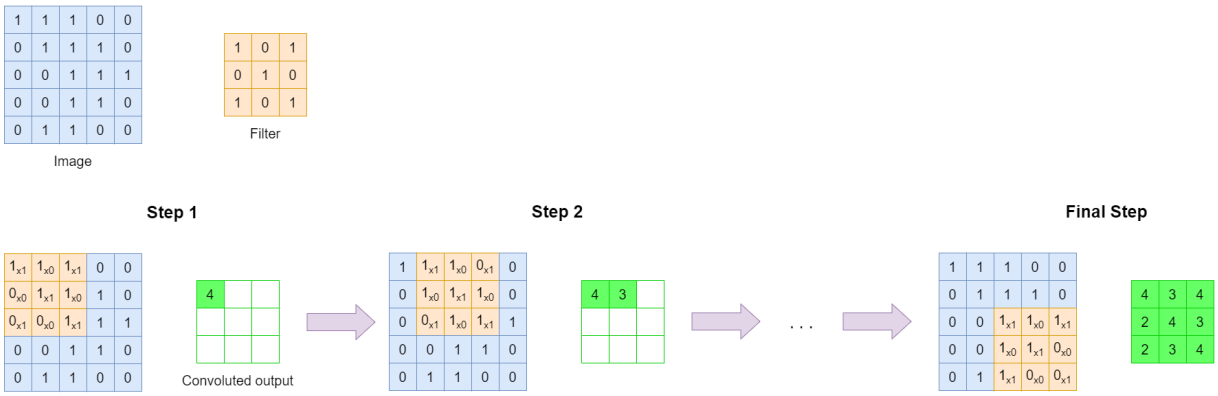


Figure 2.5: Example of convolution of a 5x5x1 image with a 3x3x1 filter

main element of a convolution operation is the **kernel/filter.** The kernel is a matrix of weights that act as a filter and is applied to an input matrix to create new information; we can perform this operation on images because they are matrixes (grayscale images) or multidimensional matrixes (RGB images).

As we can see in Figure 2.5, at every step of the operation, the kernel is overlapped to a given section of the image, and then a matrix multiplication is performed between the overlapped image section and the kernel. At each step, the kernel moves over the image by a given number of positions, which is determined by a parameter called **stride**; in Figure 2.5 we can see that the stride is equal to 1 because, at each step, the kernel moves of one position with respect to its previous location.
At the beginning of the procedure, the kernel is positioned on the top left section of the image (step 1 in Figure 2.5); at every step, it moves to the right of a stride value until it has parsed the complete width of the image. Then the kernel moves down with the same stride and places itself on the left end of the new row where it is now located; for example, in Figure 2.5, at step 4, the kernel position will be the same as step 1 but one position down, because the stride is equal to 1.
The overall process terminates when the image is traversed by the kernel completely, that is when the kernel reaches the bottom right section of the image (final step in Figure 2.5).

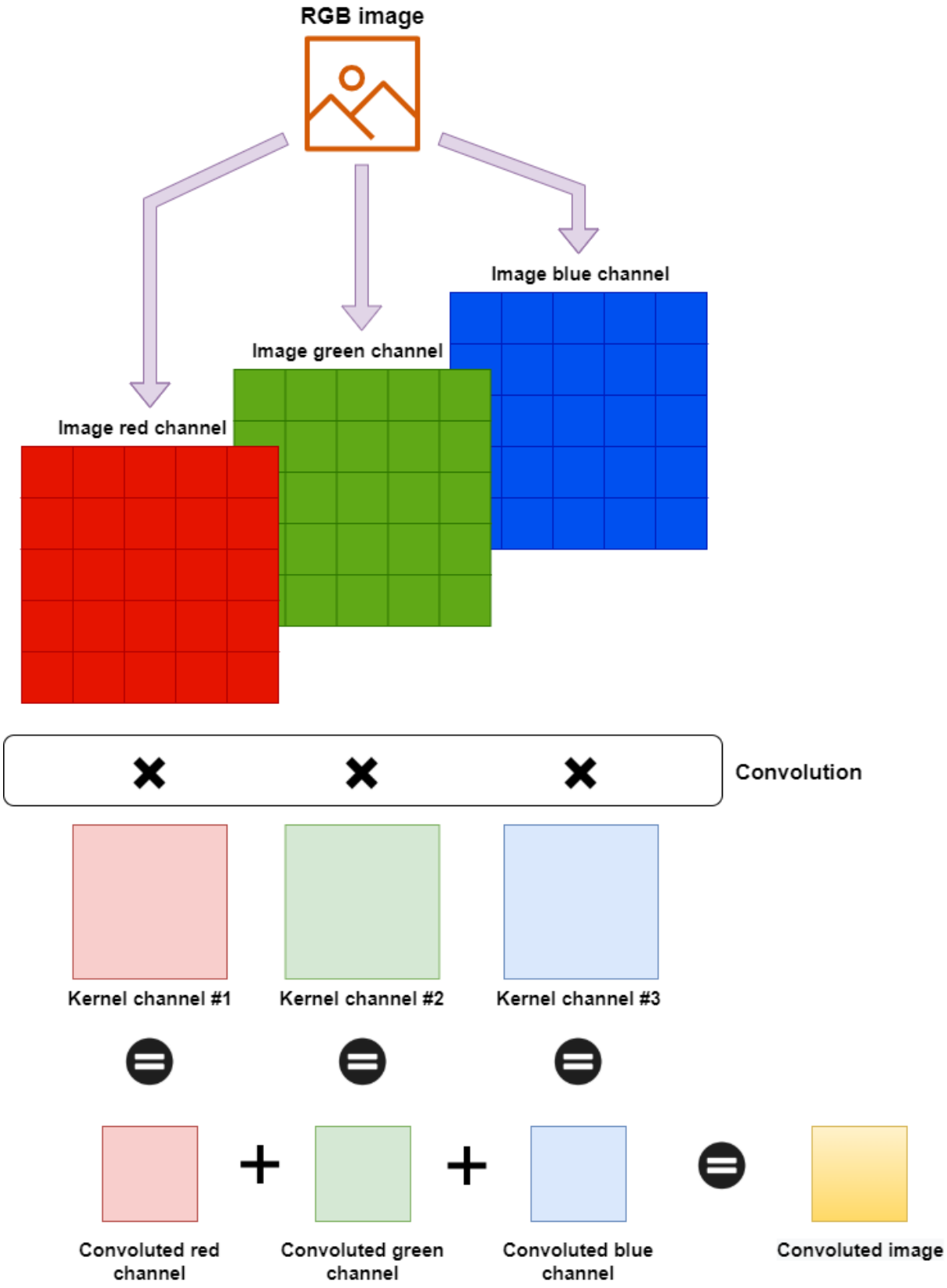


Figure 2.6: Example of convolution of an RGB image

Figure 2.6 shows the convolution of an RGB image; in this scenario, the kernel has a depth equal to the number of channels of the input image, and the convolution is performed on each channel separately. The three resulting convoluted images are then summed to obtain the final one-channel result.

This operation can cause two types of results: one in which the dimensionality of the output image is reduced with respect to the original one, and a second one in which the image dimensions remain unchanged after the operation. To achieve the latter result, it is necessary to pad the input image, applying the so-called **same padding**; instead, if no padding is applied, the output dimensions coincide with the kernel ones. This last scenario is the one shown in Figure 2.5, and it is also referred to as **valid padding** because the kernel passes only upon 'valid' data, not on the zeros used for padding. Instead, if we apply same padding, the kernel also considers padding data during the computation, as we can see in Figure 2.7.

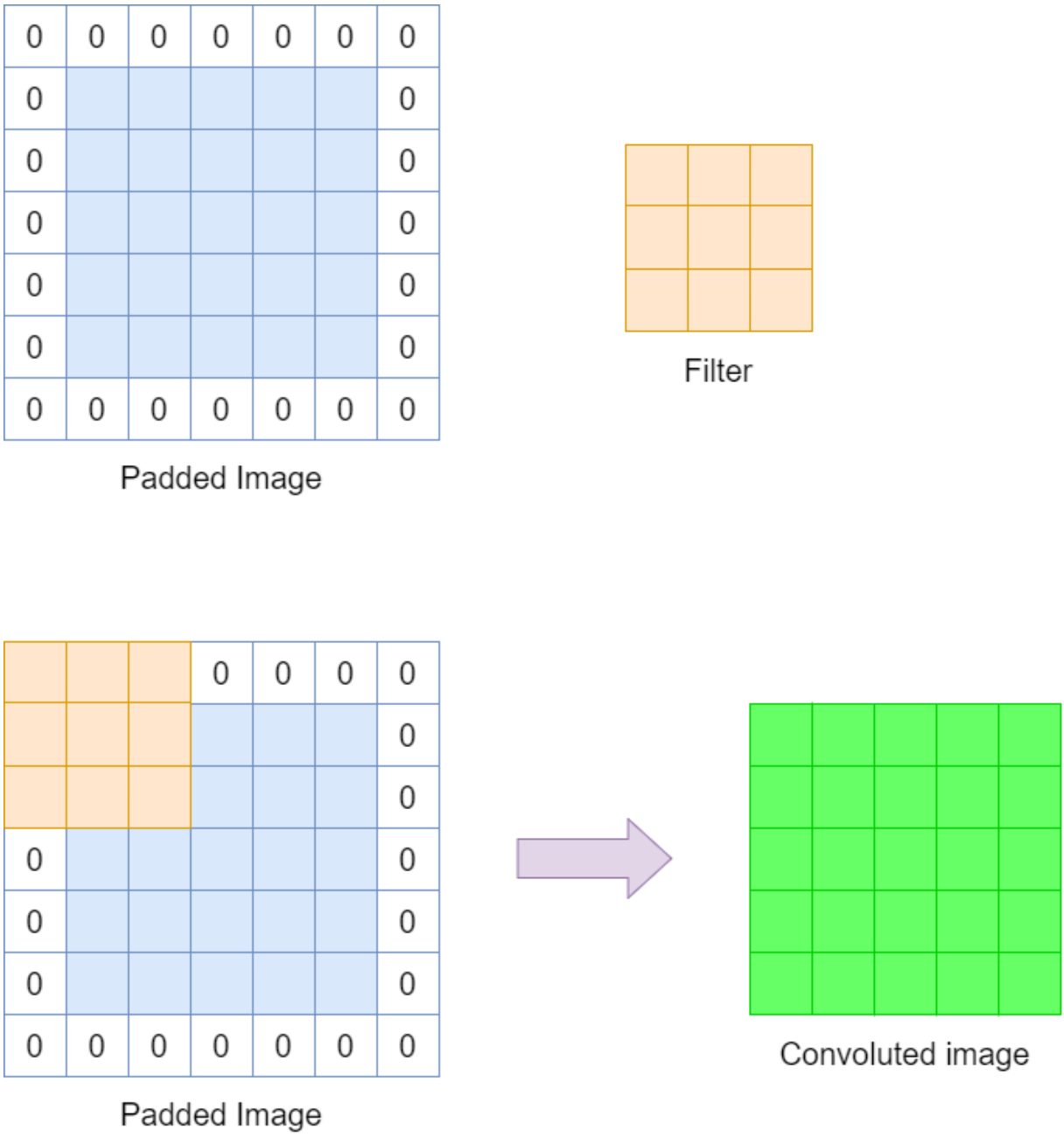


Figure 2.7: Example of convolution using same padding

Thanks to the convolution operation, we can extract meaningful features from input images; the main idea behind convolutional neural networks is to integrate this capability into the neural network paradigm through a dedicated layer called **convolutional layer**. In this type of layer, we can specify the kernel size, type of padding applied, and the number of kernels used, which determines how many features are extracted from the input image. In a convolutional neural network, there is usually more than one convolutional layer; the first ones, closer to the input, extract only low-level features (e.g. contours,

edges, color, ...), while deeper layers are able to extract increasingly higher-level features, that represent more refined generalizations of the input image.
Another fundamental block of a convolutional neural network is the **pooling layer**, which integrates inside the network an image processing operation, called **pooling**, whose goal is to reduce the spatial dimensions of an image.
The main element of the pooling operation is called kernel and acts as a filter applied to the image, similarly as it happens in the convolution operation. However, the main difference between the two kernels is that the one considered in the pooling operation does not contain values, while the one used in convolution is a matrix of weights.
The basic idea behind the pooling operation is to have the kernel hovering above sections of the image; at each step, a mathematical operation is applied to the values of the section covered by the kernel.
The mathematical computation that is applied depends on the type of pooling operation chosen: in the case of **Max pooling**, it returns the maximum value among the ones in the covered section, while in **Average pooling** the result is the average of these values. The first variant is usually chosen in pooling layers because, besides dimensionality reduction, it also performs some kind of noise suppression discarding noisy features from the image.
At each step of the operation, the kernel moves over the image; the logic behind this shift is equal to the one seen in the convolution, and also for the pooling we define a stride value that determines the amount of movement per step.
The overall operation terminates when the kernel reaches the bottom right corner of the image; the ensemble of numerical results produced during the operation represents the pooled image. As we can see in Figure 2.8, the output image produced by the pooling operation has the same spatial dimensions as the kernel, and so the dimensionality of the original image has been reduced.

The goal of pooling layers in neural networks is to reduce the feature's spatial dimensions of an image (height and width); this operation proves very useful because it reduces the computational requirements via dimensionality reduction.
The union of a convolutional layer and a pooling layer forms the constitutive block of a convolutional neural network; a complete convolutional neural network usually contains several of these blocks in sequence. The number of these blocks depends on the degree of analysis required for the task and on the computational power at disposal. With this sequence of convolutional and pooling layers, the convolutional neural network can efficiently extract high-level features from input images.
However, the final goal of a neural network is to produce data-driven decisions and solve specific tasks; to do so, another block of layers must be added to the network. First, the

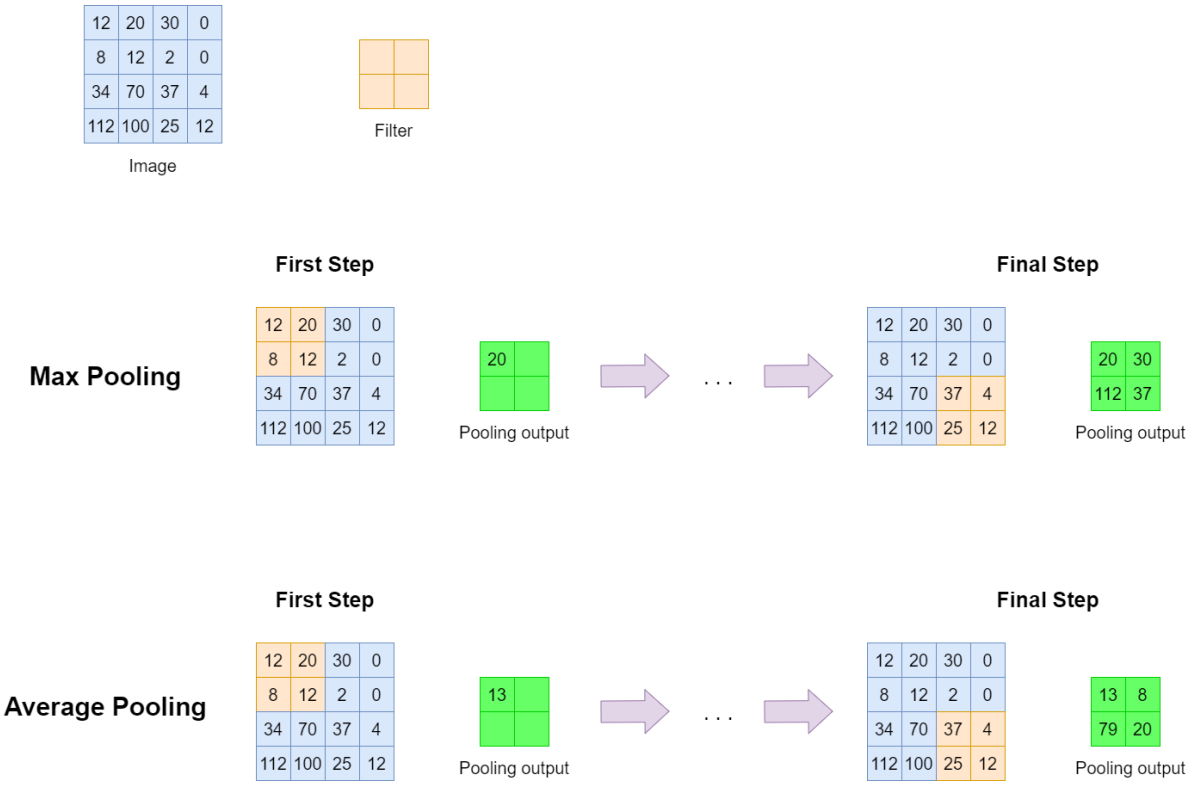


Figure 2.8: Example of Max pooling and Average pooling operations

output of the "feature extractor" is flattened, creating the so-called **feature vector**; this operation is necessary because we want to feed the extracted features to a **Multi-layer Perceptron** (see Section 2.4). This multi-layer perceptron learns nonlinear combinations of the extracted high-level features, and its output represents the prediction vector of the convolutional neural network. An overview that resumes the architecture of a convolutional neural network is presented in Figure 2.9.
The training procedure of a fully connected layer has been already shown in 2.7, while the trainable parameters in convolutional layers are the filter weights that are learned through backpropagation as what happens in feedforward neural networks [27].

2.6. Advanced training practices

Until now, we have defined the performance of a model in terms of how well it approximates the true distribution of a given set of training data or evaluating its capability of solving a certain task given a set of known input-output pairs.
This concept of performance evaluation takes into account only the results obtained on known data, but in real applications, we would like to have a model that performs equally well on new samples. From this point of view, a model can suffer two opposite problems,

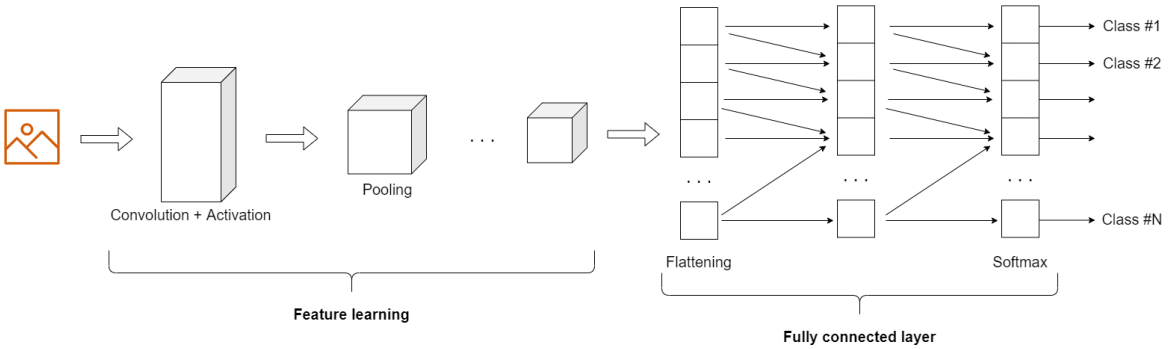


Figure 2.9: Architecture of a convolutional neural network

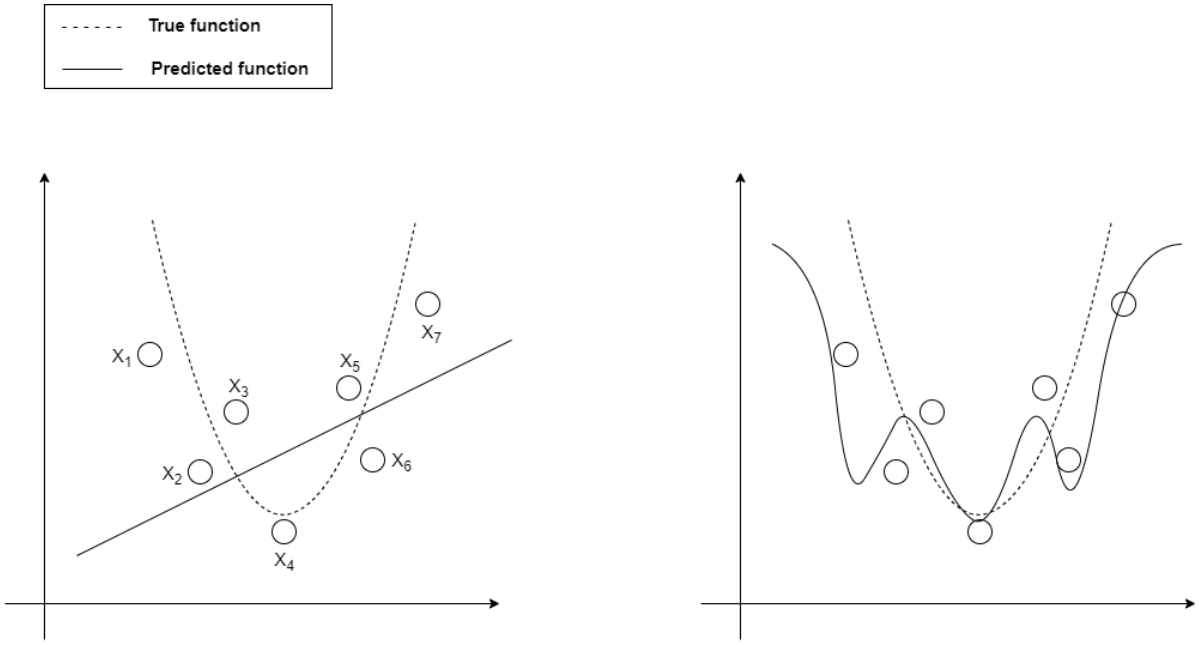


Figure 2.10: Generalization issues of learning models: underfitting (left chart) and overfitting (right chart)

called **overfitting** and **underfitting**.
Overfitting means that the model fits well the training data but shows poor results on unseen samples; this issue usually arises in models with high complexity.
Underfitting instead indicates that the model does not learn satisfactorily to solve the problem at hand, showing poor performance even on training data. Simple models with few parameters suffer from this problem.
So, depending on the task that has to be solved and the available data, we must consider this tradeoff between performance on training and unseen data when building a model. Figure 2.10 shows how, in the same scenario, a simple linear model is far from finding the true function representing the problem at hand and poorly approximates the train-

ing samples' distribution, while a too complex one optimally fits the distribution of the training samples but does not generalize well the target function. Underfitting can be easily solved by increasing the model complexity and consequently its learning capability; instead, to prevent overfitting, it is necessary to resort to particular techniques.
To measure the generalization capability of the model on new data, we cannot use the same data with which we have trained it, but we need to hold out a part of the available data to form an independent dataset called **validation dataset**. This dataset will be used to choose which model is the best among the ones obtained through hyperparameters tuning, such as the number of layers and number of hidden neurons.
So, we can recognize a model suffering from overfitting by analyzing the trend of both the training and validation error; if during the learning procedure, the training error keeps constantly decreasing, but the validation error at some point starts to increase, the model is overfitting. This property is at the base of one simple but effective technique used to prevent overfitting, called **Early stopping**; basically, it consists in halting the training of the model when its performance on the validation dataset starts to degrade.
Another solution to the issue of overfitting is to reduce the complexity of the network, a goal that can be achieved by either simplifying the network structure and consequently reducing the number of weights considered or constraining the value of those weights; the former approach leads to the **dropout** technique, the latter to the one referred as **weight decay**.
The theoretical motivation behind dropout can be found in genetics, especially in the concept of sexual reproduction [38]. In sexual reproduction, a gene contributes to the fitness of the individual through cooperation with a small number of other genes. In this way, the process favors the transmission and spread of useful genes inside the population and, at the same time, increases the chances of a new gene to improve, on its own, the individual's fitness, reducing the dependencies on other genes.
The existing parallelism between biology and artificial neural networks makes it possible to exploit a similar property considering hidden units in a neural network in place of genes. Training a network with dropout means turning off randomly neurons of the network to force them to rely less on other units during the learning procedure and, in this way, to discover novel and more useful features.

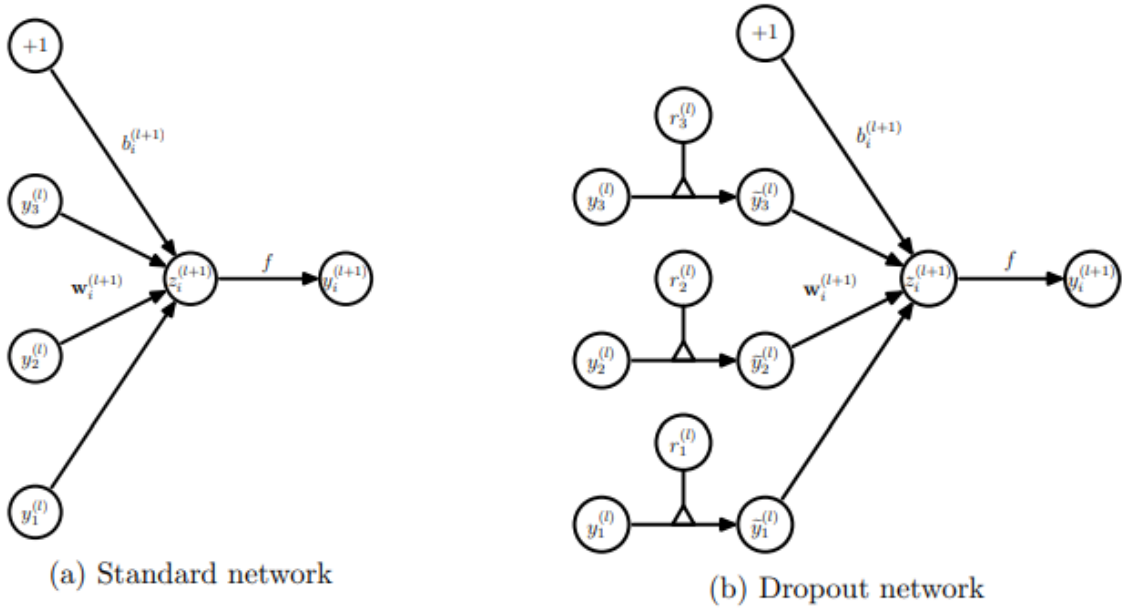


Figure 2.11: Comparison of the basic operations of a standard and dropout network [38]

As we can see in Figure 2.11, in a dropout network, for any layer l, we have a vector r^l of independent Bernoulli random variables that have a certain probability of being equal to 1 and so determine which neurons will contribute to the computation of the output and which will be inhibited.
Weight decay adds, to the loss function, a term that penalizes the model according to the magnitude of its weights. In this way, we constrain the model's weights to assume small values during their update; this approach works because large weights usually indicate that the network is excessively specialized in approximating specific samples and, consequently, suffering from overfitting. There are two main variants of weight decay, depending on the type of penalization term utilized in the loss function:

- **L1 (or lasso) regularization**:

$$L(\mathbf{w}) = \sum_{i=1}^{N}(y_i - g(x_i|\mathbf{w}))^2 + \gamma \cdot \sum_{q=1}^{Q}|w_q| \tag{2.9}$$

- **L2 (or ridge) regularization**:

$$L(\mathbf{w}) = \sum_{i=1}^{N}(y_i - g(x_i|\mathbf{w}))^2 + \gamma \cdot \sum_{q=1}^{Q}w_q^2 \tag{2.10}$$

where γ is a small value called **weight decay**.

Another issue that occurs in deep neural networks, called **vanishing gradient**, is associated with the use of certain activation functions in the hidden neurons. We recall that learning through the backpropagation algorithm requires gradients multiplication; in deep networks, the number of multiplications required increases significantly, especially if the weight to update is far from the output.
In earlier works, the activation functions often utilized in neural networks were Sigmoid and Tanh (see Figure 2.3); but, looking at their shape, we can see that the gradient of these functions saturates to zero when the input assumes too large or too small values. So, the update of some weights may stop, and especially in deep networks, the number of vanishing gradients increases, leading to a dramatic degradation in the model performance.
To solve this problem, **Rectified Linear Unit (or ReLU)** is introduced as an activation function; its defining formulas are

$$g(\mathbf{x}) = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases} \tag{2.11}$$

$$g'(\mathbf{x}) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases} \tag{2.12}$$

ReLU is a piecewise function formed by the union of two linear functions, but overall it is a nonlinear function; it allows not only to reduce the likelihood of vanishing gradients, but also to discover nonlinear and complex relationships inside the data.
Moreover, it has the effect of deactivating parts of the activations of the hidden neurons and so creates sparse networks, in a similar way to what dropout does to improve the model performance.
Finally, it has also been demonstrated by Krizhevsky et al. [25] that the training procedure with the Stochastic Gradient Descent algorithm converges noticeably faster when using the ReLU function, due to the more efficient computation compared with the two exponential functions. Because of these properties, ReLU has become among the most popular chosen activation function for neural networks.

Despite the aforementioned benefits, the main issue that arises from the use of the ReLU activation function is the **dying neurons** problem: if the majority of the output values produced by the ReLU for a given neuron are equal to zero, that neuron cannot learn anymore on the input examples. This situation usually occurs if all the weights of the

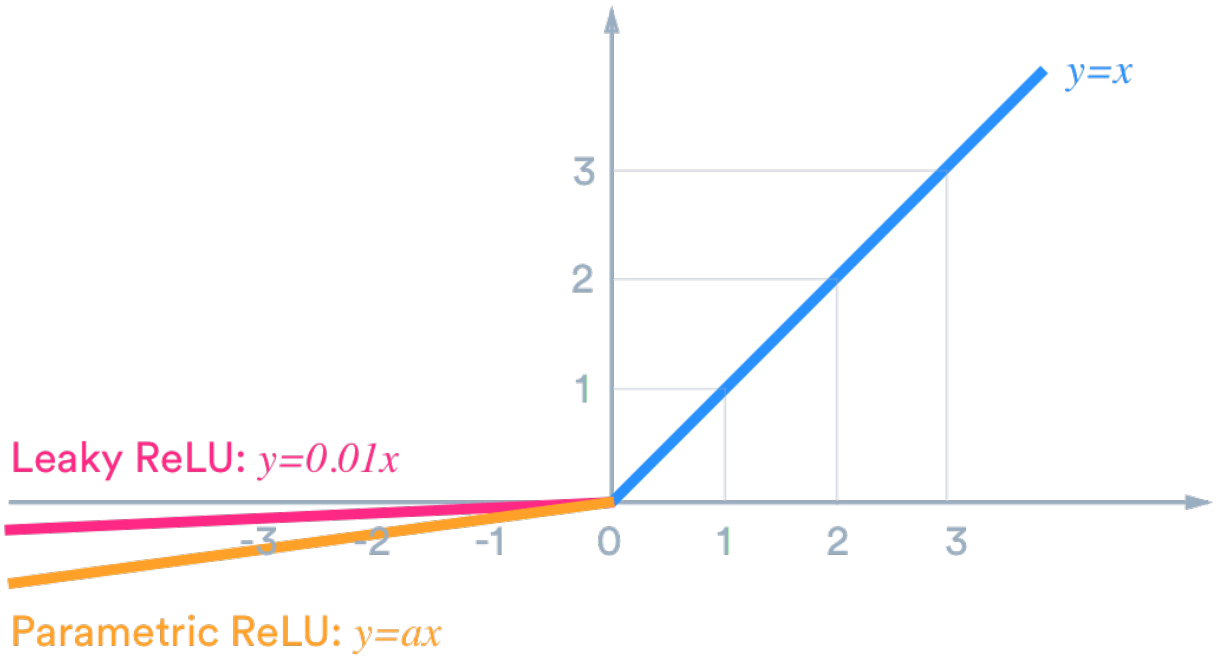


Figure 2.12: Leaky ReLU and Parametric ReLU activation functions [19]

network are initialized with zero or if a big gradient flows through the neuron, causing its weights to assume a large negative value. The associated weights will not be updated during backpropagation, so the neuron is completely removed from the learning process; besides, this condition is irreversible because the gradient of ReLU on the left half-plane is equal to 0. To patch this drawback of the ReLU activation function, a variant called **Leaky ReLU** (see Figure) is devised.

$$g(\mathbf{x}) = \begin{cases} x & x > 0 \\ 0.1 \cdot x & x \leq 0 \end{cases} \tag{2.13}$$

$$g'(\mathbf{x}) = \begin{cases} 1 & x > 0 \\ 0.1 & x \leq 0 \end{cases} \tag{2.14}$$

It adds a small positive slope to the original function in the left half-plane; in this way, the gradient is different from zero there, meaning that, given any configuration of network inputs and weights, each neuron still participates in the learning process.
Leaky ReLU derives from the more general concept of **Parametric ReLU**.

$$g(\mathbf{x}) = \begin{cases} x & x > 0 \\ ax & x \leq 0 \end{cases} \tag{2.15}$$

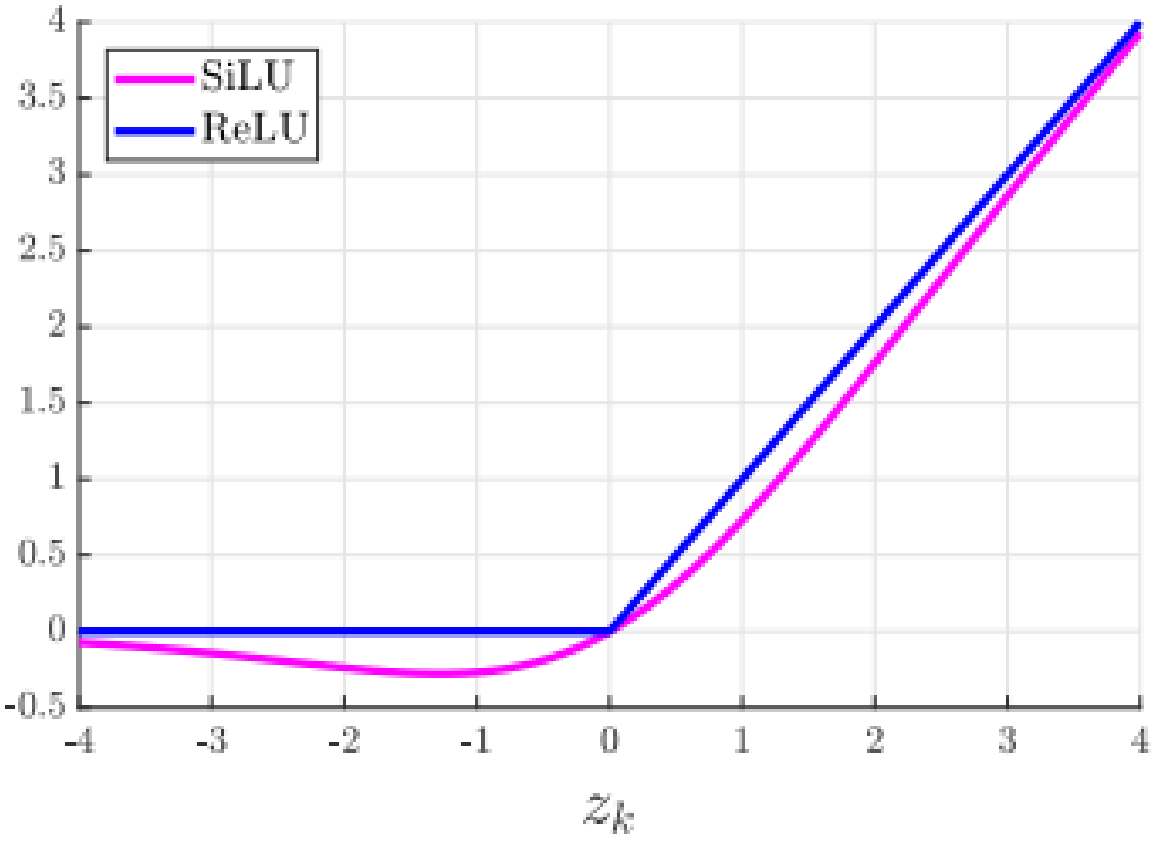


Figure 2.13: SiLU and ReLU activation functions [9]

$$g'(\mathbf{x}) = \begin{cases} 1 & x > 0 \\ a & x \leq 0 \end{cases} \tag{2.16}$$

In this case, instead of having a fixed slope, we tune the parameter a depending on the model, learning it through backpropagation.
Recently, another activation function has been devised, called Sigmoid Linear Unit (SiLU) [9], that is calculated as

$$g_k(\mathbf{s}) = z_k \sigma(z_k) \tag{2.17}$$

where z_k is the input to hidden unit k and $\sigma(\cdot)$ is the sigmoid function.
In Figure 2.13, it is shown a comparison between SiLU and ReLU functions.

Unlike ReLU or other activation functions, this function is not monotonically increasing; it has a global minimum, and it behaves like an implicit regularizer that prevents the

learning of weights of large magnitudes.
Another element that influences the result of a training procedure is the weights initialization; in fact, the choice of the initial values assumed by the model's weights can severely change the output of the gradient descent algorithm. Several issues arise when we try to initialize the weights of a network without a grounded criterion:

- **Zero initialization**: if all weights are initially set to zero, no learning can happen, and the network remains stuck in its initial condition.
- **Big values initialization**: if we initially set the weights to great numbers, the gradient descent algorithm leads to the constant growth of the gradients as they pass through each layer; large gradients cause great updates to the weights at each step, and so to an unstable network.
- **Small values initialization**: with too small numbers, we fall back into a vanishing gradient scenario.

The drawbacks of all these approaches lead to the development of some heuristics to determine the weights' initial values, and the most used one is to initialize the weights with small numbers randomly sampled from a Gaussian distribution. However, the parameters of this distribution need to be adapted to the complexity of the network in order to see effective results in the optimization algorithm. Recently, other initialization techniques have been developed; two of them, in particular, have become popular solutions to the problem: the **Xavier initialization** [15] and the **He initialization** [17].
A usual symptom of a bad initialization in a model is a great difference in variance between gradients in various parts of the network; so, the objective of these initialization methods is to maintain constant the variance of the backpropagated gradients throughout the network. This goal is formalized mathematically as reaching the equality between the variance of the input and the output vectors.
Starting from a scenario in which the input vector of the network has n_{in} components, a neuron j with associated weights w and a linear activation function, and given

1. the output of the network:

$$h_j = w_{j1} \cdot x_1 + ... + w_{ji} \cdot x_i + ... + w_{jn_{in}} \cdot x_{n_{in}} \tag{2.18}$$

2.

$$Var(w_{ji} \cdot x_i) = E[x_i]^2 \cdot Var(w_{ji}) + E[w_{ji}]^2 \cdot Var(x_i) + Var(w_{ji}) \cdot Var(x_i) \tag{2.19}$$

3.

$$Var(w_{ji} \cdot x_i) = Var(w_{ji}) \cdot Var(x_i) \tag{2.20}$$

we obtain the equation that expresses the variance of the output vector as a function of the variance of the input vector (assuming the inputs and weights to have mean equal to 0 and that all x_i and w_i are i.i.d.):

$$Var(h_j) = n_{in} \cdot Var(w_i) \cdot Var(x_i) \tag{2.21}$$

So, we would like to have $n_{in} \cdot Var(w_i) = 1$.
For this reason, the proposal by Xavier Glorot and Yoshua Bengio was to initialize weights as

$$w \sim \mathcal{N}(0, \frac{1}{n_{in}}). \tag{2.22}$$

Further research made by Glorot and Bengio discovers that the same result can be obtained if $n_{out} \cdot Var(w_i) = 1$, with n_{out} number of output vector components; to take into account both this development and the previous conclusion, the final propose by Xavier was

$$w \sim \mathcal{N}(0, \frac{2}{n_{in} + n_{out}}). \tag{2.23}$$

Xavier initialization assumes that the activation function of the neuron is linear.
Kaiming He overcomes this limitation by developing an initialization technique that takes into account the type of activation function used. It can be demonstrated that the He initialization for the ReLU activation function is

$$w \sim \mathcal{N}(0, \frac{2}{n_{in}}). \tag{2.24}$$

He initialization becomes a fundamental element for training deep neural networks due to the necessity of using ReLU activation functions in those contexts to prevent the phenomenon of vanishing gradients.

3 | State of the art

In this section, we introduce notions that are essential for understanding our work, and we describe their state-of-the-art implementation.
First, we discuss privacy attacks on deep learning models, focusing in particular on the two most famous categories of attacks: **Model inversion attacks** and **Membership inference attacks**. Then, we present for both attacks recent solutions that prove useful in overcoming the limitations of standard methods of attacks.
After that, we discuss the topic of **Differential Privacy**, starting from its theoretical basis and then describing two relaxations of its original formulation that have gained relevance in recent times, the (ε, δ)**-Differential Privacy** and the **Renyi Differential Privacy**.

3.1. Threats on privacy for learning models

Before introducing the different types of threats to privacy preservation in machine learning models, we must describe the environment in which these threats are taking place, also called **threat model**. Following the model presented by Rigaki and Garcia [34], there are four actors involved:

- **data owners**, whose data may be sensitive and represent the target of the attack;
- **model owners**, that decide how much information share about the model and, in several cases, are also the owners of the data;
- **model consumers**, that use the services offered by model owners through dedicated interfaces;
- **adversary**, that has access to the same services and model information the owner offers to its customers;

The assets possibly targeted by the adversary are either the training dataset or the model itself, in particular its parameters, hyperparameters, or architecture. Another relevant element that characterizes the attack scenario is the **adversarial knowledge**, intended

as how much information the adversary has at his disposal when performing an attack. From the model perspective, his knowledge can range from access to the exposed model interface to the complete understanding of its characteristics; regarding the data perspective, we usually assume that the adversary has no direct knowledge of the training samples but knows only the underlying data distribution.
Depending on the extent of adversarial knowledge, we differentiate the threat scenario into two types: **Black-box** and **White-box**.
In a **black-box** scenario, the adversary knows only elements of the target model that are available to the public, such as prediction vectors, but neither has any access to the model structure nor any information about the training dataset.
In a **white-box** scenario, the adversary has complete knowledge of the target model and knows the data distribution of the training samples. The black-box scenario is the more challenging for the attacker and consequently the most explored by researchers when developing new privacy solutions.
The general objective of an adversary performing a privacy attack is to obtain environmental information that owners want to keep secret.
In our study, we will focus on two types of attacks that target the training dataset of a machine learning model: **model inversion (or reconstruction) attacks** and **membership inference attacks**.

3.1.1. Model inversion attacks

Model inversion attacks try to reconstruct training samples of the attacked model starting from environmental elements known by the attacker; some attacks recover the entire sample, while others focus on finding probable values for specific sensitive attributes of the input data.
The first designs of reconstruction attacks assumed a white-box scenario in which the adversary knows the output label, the prior distribution of features for a given sample, and has complete access to the model; with these assumptions, the attacker estimates the sensitive attributes' values that maximize the probability of observing the known model parameters. These kinds of attacks are referred to as **Maximum a-posteriori (MAP)** attacks [11], but they soon have been abandoned because their effectiveness decreases as the number of features and their range of values grows.
To overcome this limitation, Frederikson et al. [12] proposed to formulate the attack as an optimization problem where the objective function depends on the model output; in the end, starting from assumptions similar to the ones considered in MAP attacks, the attacker reconstructs the input sample through gradient descent in the input space.

The drawback of this last approach is that the optimization problem is hard to solve, so Zhang et al. [42] designed a novel solution that involves the use of a generative adversarial network (GAN) [16] to learn the training data representation, exploiting the properties of this kind of network to increase the feasibility of the attack. First, the GAN learns to generate samples realistically similar to the training data, and then the attacker inverts the trained GAN to obtain the latent vector $\hat{z}$ that generates the most likely image: this can be formalized as

$$\hat{z} = \underset{z}{\operatorname{argmin}}\, L_{prior}(z) + \lambda L_{id}(z) \tag{3.1}$$

where $L_{prior}(z)$ is the loss of the generative process, while $L_{id}(z)$ ensures the likelihood of the produced images.
Finally, Yang et al. [40] proposed a black-box attack where the adversary does not know any detail about the model and its training data; instead, it knows a more "generic" distribution of the training data and the output format of the model, that is the dimension of the prediction vector. In this approach, it is usually assumed that the model reveals a m-truncated prediction vector, obtained by retaining the m largest values from the real output and setting the remaining ones to 0. This assumption is reasonable because the adversary can obtain the original dimension of the output vector by querying the model with a set of input data.
So, if we refer to the prediction vector as $F_w(\mathbf{x})$ (with $\mathbf{x}$ the targeted sample), the truncated vector as $\mathbf{f}$ and the generic data distribution as p_a, the goal of the adversary is to find the sample $\hat{\mathbf{x}}$ that satisfies the following equation:

$$\hat{\mathbf{x}} = \underset{\mathbf{x} \in X_f}{\operatorname{argmax}}\, p_a(\mathbf{x}) \tag{3.2}$$

where

$$X_f = \{\mathbf{x} \in \mathcal{X} | trunc(F_w(\mathbf{x})) = \mathbf{f}\}. \tag{3.3}$$

The training procedure resembles the autoencoders' one [20]: the inversion model G_θ (acting as the decoder) is trained and then fed with prediction vectors coming from $F_w(\mathbf{x})$ in order to produce reconstructed samples. The main difference with the traditional autoencoder's setting is that the target model, acting as the encoder here, has an unknown structure and is not trainable. The inversion model is trained to minimize this objective function

$$E_{\mathbf{x}\sim p_a}[L(G_\theta(trunc_m(F_w(\mathbf{x}))), x)] \tag{3.4}$$

with L indicating the reconstruction loss between the output of the inversion model and the real training sample.

In the following paragraphs, we describe new model inversion attacks that are capable of working in different scenarios, compared to the classical ones, and exploit new forms of adversarial knowledge.

Gradient leakage exploit for inversion attacks

This kind of inversion attack, designed by Lim and Chan [29], utilizes a privacy-breaking algorithm that exploits the information leakage of gradients, particularly in a federated learning scenario, to reconstruct the input data.
Federated learning is a distributed learning technique that allows multiple parties to share information while ensuring given privacy guarantees and optimizing the training process of the single participant.
This process involves that each participant computes its gradients during each step of its training process and then sends them to a central server. In this server, all these gradients are *"aggregated and averaged before updating the weights with synchronous stochastic gradient descent (SGD) in the resulting model"* [29]. The use of synchronous SGD and the fact that no data are shared between the parties, but only gradients, enforce a sense of privacy guarantee.
However, it can be demonstrated that, in a federated learning environment, the shared gradients can be used by an adversary to recover the input data of the parties involved; this privacy leakage is exploited to perform the model inversion attack.
The main idea behind this attack is to generate **dummy gradients** starting from a randomly initialized dummy input, then compute the loss between these dummy gradients and the true shared ones; this loss is used to update the dummy input and, at the end of this optimization process, the dummy input will be equal to the real input data.
To reach this specific goal, the authors of the study decide to use the following update formula for the dummy inputs [14]:

$$\underset{x\in(0,1)}{\operatorname{argmin}}\ 1 - \frac{\langle \nabla_\theta L_\theta(x,y), \nabla_\theta L_\theta(x^*,y)\rangle}{\parallel \nabla_\theta L_\theta(x,y) \parallel \cdot \parallel \nabla_\theta L_\theta(x^*,y) \parallel} + \alpha TV(x), \tag{3.5}$$

where $\nabla_\theta L_\theta(x^*, y)$ represents the dummy gradients, x^* is the dummy input, and $\alpha TV(x)$

the regularization parameter of the optimization process.
In the end, this study demonstrates that by manipulating some hyperparameters of this algorithm, it is possible to fool a classification model with the generated images, showing the existence of other potential threats.

Image batch recovery

Training of deep neural networks requires to average the gradients per parameter on batches of data when updating the weights of the network. So, the fact that the update of a single parameter consists in averaging over a set of data increases the confidence in privacy preservation during the process, especially in a federated learning environment.
This presumption leads to thinking that the larger the batch size used in training, the safer the learning process becomes, and previous studies have demonstrated the efficacy of reconstruction attacks only in scenarios involving small batches of input data.
The novelty introduced by Yin et al. [41] is the design of an algorithm, called **GradInversion**, that can recover with great precision single images from deep networks and large batch sizes. The first step of this approach is to convert the input reconstruction problem into an optimization task, where, starting from random noise, we synthesize images intending to minimize the distance between the gradients of these generated data and the real gradients provided by the environment. Then, the core of the algorithm lies in a batch-wise label restoration method, together with the use of auxiliary losses that aim to ensure fidelity and group consistency regularization to the final result.
This algorithm proves successful in attacking large networks (such as ResNets) with input images coming from a large batch (from 8 to 48 images).

Explainable artificial intelligence (XAI) inversion attack

Explainable artificial intelligence was born from the increasing need for responsible use of AI frameworks to support both explainability, to justify important decisions, and privacy, to protect personal information. Explainable artificial intelligence achieves this objective by providing useful information to facilitate the user's job and improve its trust in automated frameworks. For example, in image-based deep learning, XAI provides access to activation of neurons, saliency maps, feature visualization, and so on.
The issue with a similar approach is that providing this kind of information to users represents a privacy threat because this additional knowledge can be exploited to perform model inversion attacks.
This potential threat is the starting point for the work of Zhao et al. [43]; in this study, several model inversion attacks are designed where the spatial knowledge retained in im-

age explanations is exploited to achieve high fidelity reconstruction results.
In detail, the authors focus on understanding which explanations are more useful for the attacker, representing a higher privacy threat; furthermore, they demonstrate that it is also possible to achieve higher inversion performance not having at disposal the explanations of the target model, but the ones belonging to a "surrogate" model.

3.1.2. Membership inference attacks

Membership inference attacks take as input a sample and try to determine if it belongs to the training dataset of the model under attack or not.
The most common paradigm for designing such attacks is the use of **shadow models** and **meta-models or attack models** [37]. The basic idea behind this approach is to train several "shadow" models that imitate the behavior of the target one on "surrogate" (or shadow) datasets. Shadow datasets must contain samples with the same format and similar distribution to the training data of the target model; to create such datasets, three different techniques have been conceived.

- **Model-based synthesis**: the adversary explores the space of possible inputs and finds the ones that are classified by the target model with higher confidence. Then, he samples synthetic data from these records until all the shadow datasets are filled.
- **Statistics-based synthesis**: the synthetic data are generated by independently sampling the value of each feature from its marginal distribution. It requires the adversary to have some statistical knowledge about the distribution of the target model's training data.
- **Noisy real data**: the shadow datasets are filled with data sampled from a "noisy" version of the target model's data distribution.

After the training of shadow models is complete, their outputs and the known labels from the shadow datasets form the attack dataset; this dataset is used to train the meta-model, which learns to infer membership based on the shadow models' results. Membership inference is a binary classification task, and so any machine learning classifier can be used as a meta-model, such as **Multi-layer Perceptron**, **Logistic Regression**, **Random Forest** or **K-nearest Neighbors**.
The issue with shadow models is that they represent an attempt to reproduce the target model; their output's probability distributions are similar to the real one but still different. Then, the magnitude of the difference between the shadow model and the target model becomes a relevant factor in determining the effectiveness of the attack.
Moreover, the standard formulation of this approach requires strong assumptions related

to the adversarial knowledge of the target model structure and training data distribution that cannot always be satisfied in realistic scenarios. A further research [36] finds out that this kind of attack can be carried on in other three ways, depending on how much we want to relax the assumptions related to the adversarial knowledge:

- **Model independent**: the adversary has black-box access to the target model, so he must recur to other techniques to design the shadow model's structure, such as model extraction from the same Machine Learning as a Service (MLaaS) provider of the target model or through reverse-engineering; this results in a reduction of the cost of the attack due to a less limited choice concerning the shadow model's implementation.
  Apart from this, the only difference with the standard approach is that the attacker uses only one shadow model; Salem et al. [36] demonstrate that a single shadow model is sufficient for designing an effective attack.

- **Model and data independent**: this attack is similar to the first one, with the difference that the shadow dataset comes from a different distribution than the one of the target model's training data; this approach is usually referred to as **data transferring attack**.

- **Model and data-independent (without shadow models)**: this attack starts from the same relaxations of the previous case but eliminates the use of shadow models. The attack model is implemented as a simple binary classifier that takes from the prediction vector of the target model the highest posterior and compares it against a given threshold; if its value is greater than the threshold, the input sample, from which that output is obtained, is considered a member of the training dataset. It suffices to take the highest posterior because the idea behind this attack is that a machine learning algorithm is more confident in predicting data that belong to its training dataset.
  This **threshold based attack** is not only totally independent from the target model and its training data but also eliminates the overhead costs due to the design of shadow models and the creation of suitable shadow datasets; besides, it requires no training of the attack model.

In the following paragraph, we introduce a novel type of membership inference attack that overcomes the limitations of both shadow models and threshold-based attacks and proves effective in scenarios involving the use of privacy preservation methods.

Inference attack via Differential Comparisons

The effectiveness of the shadow model approach significantly depends on the transferability between the shadow model and the target one, that is the capability of the shadow model to resemble the attacked one. The difference between these two models also influences the output probability distributions, and thus the performance of the attack degrade.
On the other hand, threshold-based attacks rely on binary comparisons against a given threshold that needs to be calibrated accordingly. To learn this decision boundary, the attacker needs enough labeled output probability distributions of members and non-members, but ground truth information is not available under the black box assumption of this kind of attack.
Hui et al. [22] propose a novel type of membership inference attack, called **BlindMI**, which "probes the target model and then infers membership directly from the probing results instead of shadow models". There are two major ideas behind this algorithm: the first one is that the adversary can produce new samples that are with high probability non-members of the original training set, the second one is the concept of **differential comparison**. Differential comparison is an approach that focuses on analyzing the differential distance between two sets, one closer to members and the other to non-members. If we move a sample from the first set to the second and the distance between the two sets decreases, this sample can be considered a member; otherwise, it is considered a non-member.
The overall attack starts from the first idea to obtain a new dataset of non-members samples, and then this dataset is differentially compared with a given target dataset, removing the non-members iteratively from the target until we converge to a situation in which moving any remaining sample decreases the distance between the two sets. In the end, the resulting target dataset will contain only members of the input training set.
The authors of this study evaluate BlindMI in different settings of adversarial knowledge and show that it outperforms state-of-the-art membership inference attacks in terms of F1-score and can reasonably overcome several privacy preservation techniques, such as differential privacy.

3.2. Differential privacy

Entities that own data want to exploit them in order to obtain meaningful information and insights. The difficulty of this task depends on the dataset's characteristics, and it often results hard for a person to perform a useful analysis of the data.

Data mining is the discipline that investigates the topics of data analysis and information extraction with the objective of enhancing the results achieved by these processes. Its goal is to design algorithms capable of transforming available data into new and useful information in an automated way through the extrapolation of intrinsic patterns and relationships inside the dataset. At first, data mining had a retrospective approach to data delivery and mainly dealt with static data; the development of new technologies has allowed data mining to act prospectively, increasing the utility of extracted information and enabling the possibility to use this knowledge for trend prediction or decision making. This improvement in data mining capability is also due to the exploit of machine learning algorithms, which represent a good solution concerning the automation of data analysis. Since the early 2000s, datasets often contain massive amounts of data, whose nature is often miscellaneous (images, videos, tabular data, unstructured data, ...); these collections of data are usually referred to as **Big data**. Nowadays, data owners collect data from different sources and with various formats (emails, social networks, log files, ...), and the exponential growth of internet users has severely increased the amount of data of interest for the companies.

Companies often use deep learning models to solve data analysis-related tasks and exploit the results obtained to support the decision-making process. Deep learning models' performance is correlated strictly with the size of available data: the more data are fed to the model, the better its capability to learn representations and obtain satisfying results for the given task.

We have said that in Big data collections converge data coming from different sources, and usually most of them treat sensitive data, both in the form of crowdsourced personal information (e.g. social networks, messaging applications) or data owned by institutions (e.g. medical reports, banking accounts). The exploit of this kind of data by the companies that collect them is limited by law or regulations, such as the **General Data Protection Regulation (or GDPR)**; data owners are so obliged to feed fewer data to their deep learning models, with the consequence of having worse results for what regards extraction of information and data analysis.

Moreover, the great capability of deep learning models to encode fine details of the training data allows an adversary to exploit any knowledge regarding the model to guess sensitive information regarding the training samples.

A first intuitive approach to prevent these privacy issues can be to perturb with noise the input data or the output of the model. This method results counterproductive because, in this case, we cannot trust totally anymore the results produced by the model.

Another solution can be to add noise to the final model parameters according to a worst-case scenario, but this can significantly ruin the performance of the learned model.

A more reliable solution finds its theoretical basis on the concept of **Differential privacy** [5, 7, 8], which is devised as an effective privacy guarantee for algorithms that work with aggregated data. Before enunciating the formal definition of differential privacy, we have to explain the concept of adjacent databases. Two databases are adjacent "if they differ in a single entry, that is if one image-label pair is present in one set and absent in the other" [1].

Definition 1. A randomized mechanism $M: D \rightarrow R$ with domain D and range R satisfies ε-differential privacy if for any two adjacent inputs $d,\ d' \in D$ and for any subset of outputs $S \subseteq R$ it holds that

$$Pr[M(d) \in S\ ] \leq e^{\varepsilon} Pr[M(d') \in S\ ]. \tag{3.6}$$

The parameter ε is called privacy budget because it represents how much information leakage we can afford in our system, so a lower value indicates a stricter privacy guarantee.
Differential privacy represents a significant development in the field of privacy-preserving deep learning because it guarantees three properties that are very useful in our context of research: composability, group privacy, and robustness to auxiliary information.
Composability means that if we have a composite mechanism and each of its components is differentially private, then the overall composition is differentially private; this property allows us to design mechanisms in a modular fashion.
Group privacy assures that if the dataset contains correlated data, like the ones provided by the same individual, the privacy guarantee degrades gracefully and not abruptly.
Finally, robustness to auxiliary information guarantees that the privacy level assured by theory stands regardless of the knowledge available to the adversary.
The paradigm used to approximate a given function f with a differentially private mechanism consists in adding noise proportional to the function's sensitivity S_f, defined as

$$S_f = \max(|f(d) - f(d')|). \tag{3.7}$$

with d and d' adjacent datasets.
For example, the Gaussian noise mechanism, one of the most used in practical applications, is defined as

$$M(d) \triangleq f(d) + \mathcal{N}(0, S_f^2 \cdot \sigma^2). \tag{3.8}$$

3.2.1. (ε, δ)-Differential Privacy

In practice, DP mechanisms cannot always assure privacy guarantees, as stated in the main definition of ε-Differential Privacy, for every possible ε.
For this reason, the classical formulation of ε-Differential Privacy needs to be relaxed to allow every differentially private mechanism to be implemented with less strict privacy guarantees but at least valid for every ε. The main relaxation of differential privacy usually considered for implementing any mechanism is (ε, δ)-Differential Privacy.
The definition of (ε, δ)-Differential Privacy is almost identical to the classical DP formulation, except for the introduction of the additive factor δ [6], which represents the probability that plain ε-DP is broken.
Definition 2. A randomized mechanism *M:* $D \rightarrow R$ with domain D and range R satisfies (ε, δ)-differential privacy if for any two adjacent inputs *d, d'* $\in D$ and for any subset of outputs $S \subseteq R$ it holds that

$$Pr[\ M(d) \in S\ ] \leq e^{\varepsilon} Pr[\ M(d') \in S\ ] + \delta. \tag{3.9}$$

Once we have defined a differentially private mechanism, we can apply an a posteriori analysis giving us several (ϵ, δ) pairs that satisfy the privacy conditions of our scenario. In the case of a composite mechanism, this analysis results more complex because we need to keep track in some way of the privacy loss accumulated during the execution of each component. However, the composability property of differential privacy brought Abadi et al. [1] to design an element, called privacy accountant, that calculates the privacy cost needed for each access to the data and uses this information to define the overall privacy loss of the mechanism.
Studies on how to implement a differential privacy mechanism into a deep learning model have lead to the design of the **Differentially private stochastic gradient descent (or DP-SGD)** [1].

Algorithm 3.1 Differentially Private Stochastic Gradient Descent

Input: Samples $\{x_1, x_2, ..., x_N\}$, loss function $L(\theta) = \frac{1}{N} \cdot \sum_i L(\theta, x_i)$. Parameters: learning rate η, noise scale σ, group size L, gradient norm bound C.
Initialize θ_0 randomly.
for $for - t \in [T]$ **do**
  Take a random sample L_t with sampling probability L/N
  Compute gradient
  for $for - i \in L_t$ **do**
    $\mathbf{g}_t(x_i) \leftarrow \nabla_{\theta_t} L(\theta, x_i)$
  end for
  Clip gradient
  $\bar{\mathbf{g}}_t(x_i) \leftarrow \frac{\mathbf{g}_t(x_i)}{\max(1, \frac{||\mathbf{g}_t(x_i)||_2}{C})}$
  Add noise
  $\tilde{\mathbf{g}}_t \leftarrow \frac{1}{L} \sum_i (\bar{\mathbf{g}}_t(x_i) + \mathcal{N}(0, \sigma^2 C^2 \mathbf{I}))$
  Gradient descent
  $\theta_{t+1} \leftarrow \theta_t - \eta_t \cdot \tilde{\mathbf{g}}_t$
end for
Output: θ_T and the overall privacy cost (ε, δ) computed using a privacy accountant

The idea behind DP-SGD is to act on the model during the training process, not at the end of it, and specify the level of privacy achieved through the differential privacy guarantees. As we can see in Algorithm 3.1, two new steps are added to the standard SGD formulation: **norm clipping of the gradient** and **noise injection**.
Normally, there is no bound on the value that gradients can assume, but, in this scenario, it is preferable to keep under control the magnitudes of the gradients to better monitor the influence of differential privacy on the learning procedure. To achieve this, we set a gradient norm bound C that is used to discriminate which gradients have to be clipped and which can be preserved; the former ones are then scaled to the norm of C. Regarding the noise injection, it is done through a Gaussian additive noise mechanism perturbing each gradient before averaging.
Another improvement introduced by Abadi et al. is the use of a specific type of privacy accountant, called **Moments accountant**, that keeps track of a bound on the moments of a privacy loss random variable.
This method represents a generalization of the standard tracking of privacy loss of (ε, δ)-DP, that instead recurs to the strong composition theorem [32]. This accounting method instead results in a tighter bound on the ϵ and δ parameters, given chosen configurations

of noise scale and clipping bound, compared to the standard privacy accountant.
This result is proven by the following theorem [1]:
There exists two constants c_1 and c_2 so that given the sampling probability $q = \frac{L}{N}$ and the number of steps T, for any $\epsilon < c_1 q^2 T$, Algorithm 3.1 is (ϵ, δ)-differentially private for any $\delta > 0$ if we choose

$$\sigma \geq c_2 \frac{q\sqrt{T \log(\frac{1}{\delta})}}{\epsilon}. \tag{3.10}$$

Using the strong composition theorem in the same scenario we obtain

$$\sigma = \Omega(\frac{q\sqrt{T \log(\frac{1}{\delta}) \log(\frac{T}{\delta})}}{\epsilon}) \tag{3.11}$$

so basically, with moments accountant, we save a factor $\log(\frac{T}{\delta})$ in the noise bound, meaning that, with the same noise scale and δ, we can have a much lower ϵ compared to the strong composition theorem.
We have already referred to composability as one of the fundamental properties for a differentially private mechanism, and this property is also preserved in moments accountant; we now demonstrate how composability can be derived for this accounting method.
First off, we define the privacy loss c at an outcome o of a mechanism M as

$$c(o; M, aux, d, d') \triangleq \log \frac{Pr[M(aux, d) = o]}{Pr[M(aux, d') = o]} \tag{3.12}$$

where d and d' are two adjacent datasets and aux represents the auxiliary input.
As we have said earlier, moments accountant novelty lies in the concept of moment of the privacy loss; we define the λ^{th} moment α_M of a mechanism M as

$$\alpha_M(\lambda; aux, d, d') \triangleq \log \mathbb{E}_{o \sim M(aux,d)}[\exp(\lambda c(o; M, aux, d, d'))]. \tag{3.13}$$

The differential privacy of a mechanism is equivalent to a tail bound on its privacy loss; so, to derive it in the case of moments accountant, we bound all the possible mechanism's moments to a single value $\alpha_M(\lambda)$, defined as

$$\alpha_M(\lambda) \triangleq \max_{aux,d,d'} \alpha_M(\lambda; aux, d, d'). \tag{3.14}$$

We take as upper bound the maximum over all adjacent datasets *d*, *d'*, and all possible auxiliary inputs.
Now we can enunciate the two properties of α_M that are fundamental for using the moments accountant:

1. **(Composability)** *Assuming that a mechanism M consists of a sequence of adaptive mechanisms* $M_1, \ldots, M_k$ *where* $M_i : \prod_{j=1}^{i-1} \mathrm{R}_j \times \mathrm{D} \to \mathrm{R}_i$. *Then, for any* λ

$$\alpha_M(\lambda) \leq \sum_{i=1}^{k} \alpha_{M_i}(\lambda). \tag{3.15}$$

2. **(Tail bound)** *For any* $\varepsilon > 0$, *the mechanism M is* (ε, δ)*-differentially private for*

$$\delta = \min_{\lambda} \exp(\alpha_M(\lambda) - \lambda\varepsilon). \tag{3.16}$$

The first property states that the bound on the moment of the overall mechanism can be calculated simply by adding up the moments of each mechanism involved in the composition. Once we have computed the overall bound, we can use the tail bound property to convert it into a (ε, δ) differential privacy guarantee.
The last thing that needs to be specified to apply this method is to define a bound on the value of $\alpha_{M_i}(\lambda)$ at each step of the composition.
In particular, we focus on the case of a Gaussian mechanism: assuming to have two probability density functions μ_0 and μ_1, respectively representing a $\mathcal{N}(0, \sigma^2)$ pdf and a $\mathcal{N}(1, \sigma^2)$ pdf, and defining the mixture of the two as $\mu = (1-q)\mu_0 + q\mu_1$, we compute the moment as

$$\alpha_M(\lambda) = \log \max(E_1, E_2) \tag{3.17}$$

with

$$E_1 = \mathbb{E}_{z\sim\mu_0}\left[\left(\frac{\mu_0(z)}{\mu(z)}\right)^{\lambda}\right], \tag{3.18}$$

$$E_2 = \mathbb{E}_{z\sim\mu}\left[\left(\frac{\mu(z)}{\mu_0(z)}\right)^{\lambda}\right], \tag{3.19}$$

In practice, it can be demonstrated that

$$\alpha_M(\lambda) \leq q^2\lambda\frac{\lambda+1}{(1-q)\sigma^2} + \mathcal{O}\left(\frac{q^3}{\sigma^3}\right) \tag{3.20}$$

The two properties 3.15 and 3.16 together with the bound estimated at 3.20 are used to imply the previous fundamental Theorem 3.10.

3.2.2. Renyi Differential Privacy

The Renyi Differential Privacy is a relaxation of differential privacy based on the concept of Renyi divergence [31].

Definition 1. For two probability distributions P and Q defined over R, the Renyi divergence of order $\alpha > 1$ is

$$D_\alpha(P \parallel Q) \triangleq \frac{1}{\alpha-1}\log E_{x\sim Q}\left(\frac{P(x)}{Q(x)}\right)^\alpha. \tag{3.21}$$

It can be demonstrated that the Renyi divergence of order 1 is equal to the Kullback-Leibler divergence:

$$D_1(P \parallel Q) = E_{x\sim P}\log\frac{P(x)}{Q(x)}. \tag{3.22}$$

We can also derive the formula for $\lim_{\alpha\to\infty}$:

$$D_\infty(P \parallel Q) = \sup_{x\in supp\, Q}\log\frac{P(x)}{Q(x)}. \tag{3.23}$$

From this last formula, we can easily see a relationship between the differential privacy formulation and the Renyi divergence, which can be expressed through the following definition

Definition 2. *A randomized mechanism f: $D \to R$ is ε-differentially private if and only if its distribution over any pair of adjacent inputs $d, d' \in$ D satisfies*

$$D_\infty(f(d) \parallel f(d')) \leq \varepsilon. \tag{3.24}$$

This relationship justifies the idea of developing a relaxation of standard differential privacy based on Renyi divergence; it can be generalized through the definition of the so-called **(α, ε)-Renyi differential privacy**.

Definition 3. *A randomized mechanism f: $D \to R$ is said to have ε-Renyi differential privacy of order α if for any adjacent $d, d' \in$ D it holds that*

$$D_\alpha(f(d) \parallel f(d')) \leq \varepsilon. \tag{3.25}$$

As we have already said in Section 3.2, the popularity of differential privacy as a privacy-enforcing method lies in the several important properties that it guarantees. We can demonstrate that many of these properties are still valid for Renyi differential privacy, despite the relaxation of the original definition of DP.
In particular, we focus on showing how this relaxation satisfies five important criteria: **outcomes guarantee**, **robustness to auxiliary information**, **robustness to post-processing**, **composability** and **group privacy**.

Outcomes guarantee This property represents the most basic guarantee that data owners expect from a privacy-preserving method. It can be expressed as the assurance that the probability of observing an outcome with a certain label shifts in a limited way whether any record is present or not in the mechanism's input.
We already know from 3.6 that, in the standard formulation of differential privacy, this change in probability is guaranteed to be less than a factor e^ε. For (α, ε)-Renyi differential privacy this guarantee is relaxed according to the following formula:

$$Pr[f(d) \in S\,] \leq (e^{-\varepsilon} Pr[f(d') \in S\,])^{\frac{\alpha-1}{\alpha}} \tag{3.26}$$

Robustness to auxiliary information A fundamental element in threat models involving privacy attacks is adversarial knowledge; the quantity and type of information owned by the attacker significantly influence the effectiveness of the attack.
A valid privacy method must guarantee to make the mechanism on which it is applied resilient to threats to the privacy of its inputs; this is equivalent to assuming operating in a scenario in which we have a lack of assumptions on the adversarial knowledge. This property can be formalized through Bayesian statistics, comparing the prior and the posterior probability distributions known by the adversary.
Let $Pr(d)$ be the adversary's prior over the set of possible inputs $d \in D$ and X the corresponding outcome of the mechanism; if the mechanism is ε-differentially private, we have that for the adversary's posterior the following inequality holds

$$\frac{Pr(d|X)}{Pr(d'|X)} \leq e^\varepsilon \frac{Pr(d)}{Pr(d')}, \tag{3.27}$$

for all pairs of adjacent inputs $d, d' \in D$ and all X $\in R$.
It means that any information obtained from the mechanism does not shift the relative probabilities of adjacent inputs more than e^{ε}.
For Renyi differential privacy, this property is expressed in terms of change in the Bayes factor; let the random variable $R(d, d')$ be defined as

$$R(d, d') \sim \frac{Pr(d'|X)}{Pr(d|X)} = \frac{Pr(X|d') \cdot Pr(d')}{Pr(X|d) \cdot Pr(d)} \tag{3.28}$$

where $X \sim f(d)$.
If we consider $P = f(d')$ and $P = f(d)$, starting from the definition of Renyi divergence of order α between P and Q already enunciated in 3.21, we can obtain the following bound on the α-th moment of the change of variable R:

$$E_Q\left[\left(\frac{R_{post}(d, d')}{R_{prior}(d, d')}\right)^{\alpha}\right] = E_Q\left[P(x)^{\alpha}Q(x)^{-\alpha}\right] = \exp[(\alpha - 1)D_{\alpha}(f(d') \parallel f(d))]. \tag{3.29}$$

and applying to this result the Renyi differential privacy definition 3.25, we derive

$$E_Q\left[\left(\frac{R_{post}(d, d')}{R_{prior}(d, d')}\right)^{\alpha}\right] \leq \exp[(\alpha - 1)\varepsilon]. \tag{3.30}$$

whereas for ε-differential privacy this bound, obtained putting together the definition of R and 3.27, becomes:

$$\frac{R_{post}(d, d')}{R_{prior}(d, d')} \leq e^{\varepsilon}. \tag{3.31}$$

Robustness to post-processing A simple but fundamental property that a private mechanism must ensure is that the privacy guarantee cannot be loosened by manipulating the mechanism's output, a technique also called post-processing.
To demonstrate that Renyi differential privacy guarantee this property, we start from the following theorem on data processing inequality [10]:

For any order $\alpha \in [0, \infty]$ *and any sub-*σ*-algebra* $G \subseteq F$

$$D_{\alpha}(P_{|G} \parallel Q_{|G}) \leq D_{\alpha}(P \parallel Q). \tag{3.32}$$

If we consider a mapping $g : \tilde{R} \to \tilde{R}'$, we can reformulate Theorem 3.32 as

$$D_\alpha(g(P) \parallel g(Q)) \leq D_\alpha(P \parallel Q). \tag{3.33}$$

This means that if we have a (α, ε)-Renyi differentially private mechanism M and apply to it a post-processing function f, also $f(M(\cdot))$ will be RDP, and so the privacy guarantee of the method is preserved by post-processing.

Composability As we recall, composability indicates the property of privacy preservation under the composition of mechanisms; this property is fundamental for enabling the modular design of mechanisms and ensuring privacy guarantees on the resulting composition.
In the case of ε-differential privacy, this property is formalized in this way:

Proposition 1. If mechanism $f(\cdot)$ is ε_1-differentially private and mechanism $g(\cdot)$ is ε_2-differentially private, the composition of the two is $(\varepsilon_1 + \varepsilon_2)$-differentially private.

For Renyi differential privacy, a similar statement can be demonstrated through the following proposition:

Proposition 2. Let $f : D \to R_1$ be (α, ε_1)-RDP and $g : R_1 \times D \to R_2$ be (α, ε_2)-RDP, then the mechanism defined as (X, Y), where $X \sim f(d)$ and $Y \sim g(X, d)$, satisfies $(\alpha, \varepsilon_1 + \varepsilon_2)$-RDP.

This last proposition is fundamental because it allows us to know and compute how much privacy budget is consumed with each access to the input data.

Group privacy All the theorems and propositions associated with differential privacy enunciated until now are constrained from the assumption of adjacent inputs. In practical applications, this hypothesis cannot always be guaranteed because it often happens that some inputs share the same characteristics or that they are correlated in some way; this is usually due to the fact that there exist several relationships between the data contributors, and so the influence of the single individual's record on the output of the mechanism increases significantly.
We expect that this shift of influence of a single record on the output significantly degrades the privacy guarantee of the method; instead, ε-differential privacy assures that this degradation of the privacy level is gradual, so the privacy guarantees stated so far are still valid but in a progressively weaker form.
Another interesting effect of this property is the controlled degradation of privacy guar-

antee in case of preprocessing of the input.
To formally define the group privacy, we first introduce the concept of c-stable transformation [30]:

Proposition 3. A mapping $g : D \to D'$ is c-stable if g(A) and g(B) are adjacent in D' and there exists a sequence of length c+1 so that $D_0 = A, \ldots, D_c = B$ and all (D_i, D_{i+1}) *are adjacent in D.*

It can be demonstrated that ε-differential privacy satisfies the following proposition:

Proposition 4. If f is ε-differentially private and $g : D' \to D$ is c-stable, then $f \circ g$ is cε-differentially private.

Instead, for Renyi differential privacy a similar proposition holds:

Proposition 5. If $f : D \to R$ is (α, ε)-RDP, $g : D' \to D$ is 2^c-stable and $\alpha \geq 2^{c+1}$, then $f \circ g$ is $\left(\frac{\alpha}{2^c}, 3^c\varepsilon\right)$*-RDP.*

4 | Method

In this chapter, we first introduce the scenario in which we carry on our work, then we proceed to describe the problem that initiates our research, the goals that we want to achieve at the end of our work, and the steps we take in designing our approach to solving such problem.

In our work, we investigate the topic of privacy attacks against deep learning models; we are interested in studying the effects of privacy-preservation methods against malicious attacks targeting the models' training data, in particular when these data are images. We choose to focus our research on attacks targeting images because, in recent times, an increasing number of tasks solved by deep learning algorithms involve this kind of data; moreover, in some contexts, a great quantity of sensitive information can be extracted from images, more than from other types of data. For this reason, images are considered a valuable asset and consequently a target by malicious agents that want to disclose meaningful insights out of private data. For example, an attacker can be interested in targeting the face recognition system of a company to be able to reconstruct the appearance of an employee or recognize whether a given face belongs to an employee or not.

In our study, we want to test the effectiveness of the most widespread implementation's method of differential privacy in deep learning models, the **Differentially Private Stochastic Gradient Descent (or DP-SGD)**, in protecting a model from membership inference and model inversion attacks. We already know from previous works that the integration of differential privacy and deep learning has proven successful in preserving privacy against several attacks; however, recent studies have revealed a major flaw in this method, that is the significant impact on the model's accuracy [2], especially in cases of unfair datasets. We want to understand if this thesis holds and analyze the relationship between the model's utility and the privacy guarantee.

For this reason, we evaluate in a comparative way the performance of DP-SGD with different privacy budgets according to four metrics: resilience to both membership inference and model inversion attacks, accuracy achieved by the target model, and time duration of the training process.

In our work, we also try to find an alternative solution capable of resolving the tradeoff

between privacy guarantee and performance, both in terms of training time and utility, of the model under attack satisfactorily. The main idea behind our proposal is that there can be a connection between the generalization capability of a network and its resistance to privacy attacks. Overfitting is a usual cause of a model's lack of ability to generalize well on new samples, so our approach consists in reducing the overfitting in the network to increase the protection against the attacks. In Section 2.6, we have described different techniques used to prevent overfitting; in our solution, we choose two of them, dropout and L2 regularization. It must be noted that Salem et al. [36] first propose to use dropout as a defense mechanism against membership inference attacks due to the connection between the attacks' effectiveness and overfitting in the target model; moreover, a recent work by Galinkin [13] investigates the effect of dropout on membership inference attacks against differentially private models.
We want to prove the validity of this solution as a defense mechanism empirically, evaluating it comparatively with the same four criteria we used for differential privacy: resistance to both privacy attacks, the accuracy achieved by the target model, and the total time elapsed to train it. The two studies on differential privacy and regularizers are conducted in parallel, and only afterward the results coming from both of them are compared directly.
In the following sections, we describe in detail the target model used during the tests and explain how we have set up the two attacks' scenarios, the one with the model inversion attack and the one involving the membership inference attacks.

4.1. Target Model

We refer to the target model as the model that will be trained, evaluated, and attacked to measure its resilience to privacy threats. The model, which will remain the same in every analysis performed, is a convolutional neural network with nine layers, not including the input one; in Figure 4.1, a detailed plot of the model structure is presented (for all the convolutional layers and the fully connected one, the activation function we use is the Rectified Linear Unit).
As for the training of this network, we choose as optimizer the Stochastic Gradient Descent (SGD) algorithm, and as loss function, the categorical cross-entropy, that measures the "difference between two probability distributions for a given random variable or set of events" [4].
To introduce differential privacy in the training of the target model, it suffices to substitute its optimizer with the Differentially Private Stochastic Gradient Descent, then the structure and the number of parameters of the model remain the same. We describe

the implementation details behind this process and the configuration of the algorithm's parameters in the following chapter.
Regarding the adding of regularization inside the model, we decide to insert the Dropout layers in the middle of every convolutional block, formed by convolution+pooling, and after each Dense layer, except for the network output, and to add L2 regularization only to the output layer. Figure 4.2 shows clearly where the dropout layers are inserted in the target model. These choices are made respectively to spread the regularization effect of dropout evenly inside the model and to study the effect of L2 regularization in a black box scenario. In our experiments, we first evaluate these two regularization techniques individually and then combine them. The insertion of dropout layers changes the original structure of the network significantly; to avoid changing the aspect of the model that will be subsequently attacked, but at the same time maintain the effect brought by regularization during training, we adopt the following approach:

1. Add dropout layers to the original target model following the criteria discussed before.
2. Train the modified model with the same configuration used for the original one.
3. Pass the trained model through a function that maintains all the trained layers except the dropout ones, which are eliminated, and recreate the pending connections between layers.

At the end of this procedure, we obtain a model trained with regularization having an identical structure and the same number of parameters as the original target model. Eliminating the dropout layers at the end of the training does not influence the learned weights because dropout layers have no parameters inside them for definition.

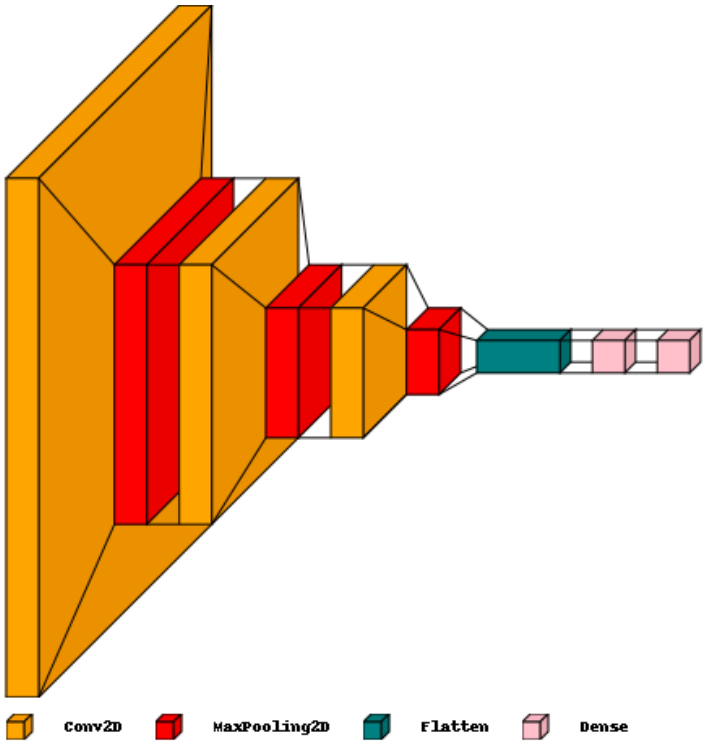


Figure 4.1: Plot of the target model's structure

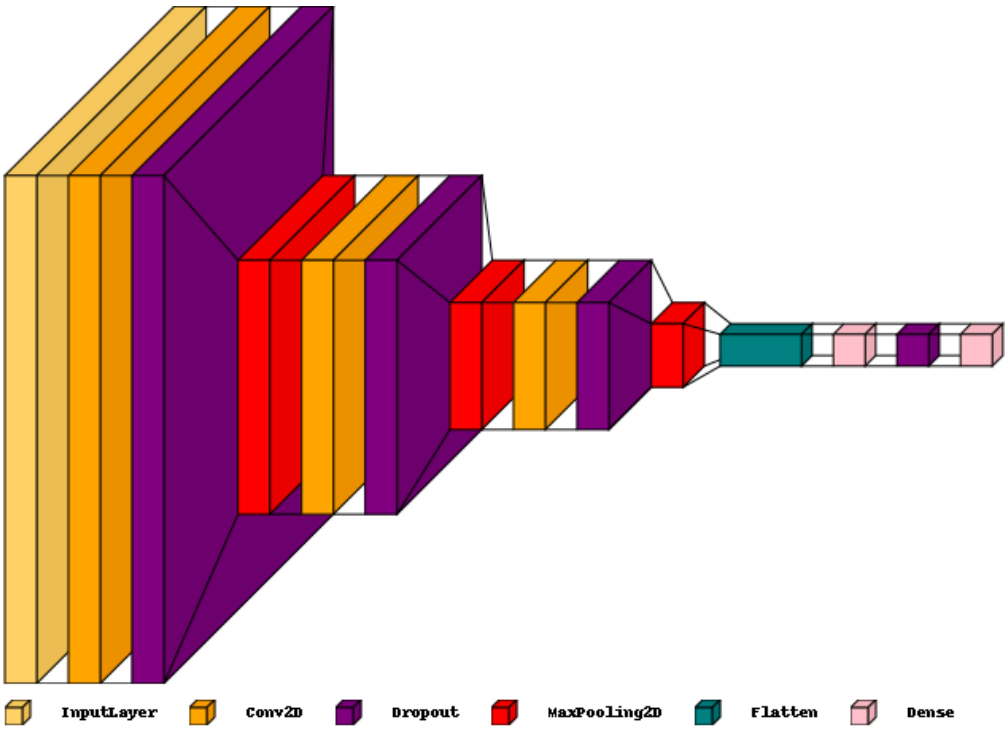


Figure 4.2: Plot of the target model's structure with regularization layers

4.2. Membership inference attack

Membership inference attacks fall under two main categories: attacks that require shadow models' training and threshold-based ones.
The first category was introduced by Shokri et al. [37]; it is based on the idea of having multiple shadow models, mimicking the target model's structure, and using their inputs-outputs pairs to train the attack model to discern if a sample is a training set's member or not. These attacks can be performed having only black-box access to the target model, assuming in most cases that we have information about the distribution of its training data when building the shadow models. Later on, Salem et al. [36] proposed three attacks strategies that work with reduced adversarial knowledge compared to the scenario considered by Shokri; the third strategy is the threshold-based attack, and it assumes no knowledge about the target model and its training data. The main idea behind this attack is to take the prediction vector produced by the target model for a given sample, extract from it the value of the highest posterior and compare it against a certain threshold. If the value is above the threshold, the sample is considered a member of the target model's training set and vice versa.
In our work, we use two black-box membership inference attacks, both inspired by Salem's paper. One is a threshold-based attack, while the other is based on an approach similar to the one proposed by Shokri but involves the training of a single shadow model and assuming no knowledge about the data distribution, called data transferring attack.
For both attacks, we provide the attacker the train and test losses, train and test logits, and train and test labels (as integer arrays) of the target model.
Moreover, we decide to differentiate the groups of data targeted by the attacks; in particular, we analyze the effectiveness of the attacks on the entire dataset, on slices containing members of each class, and on two subsets containing all the correctly classified and misclassified data points. To come up with a unique result for each attack, we show only the most successful one among all; in this way, we consider a worst-case scenario from the point of view of the target model. This result is rendered graphically through a Receiver Operating Characteristic (ROC) curve plotting the most successful membership attacks in terms of True and False Positive Rates (TPR and FPR, respectively).

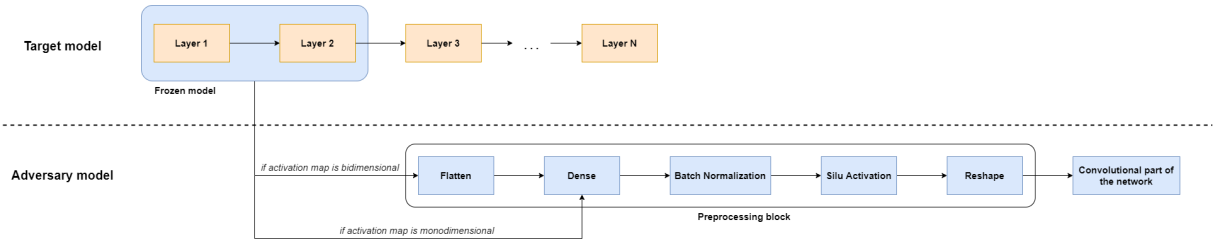


Figure 4.3: Example of how the attacking model processes the activation map of the target model

4.3. Model inversion attack

The main idea behind model inversion attacks is to exploit pieces of information concerning the target model (its architecture, its training data, or elements produced during its training process) to reconstruct the data originally fed to it.
In our approach, we have devised an inversion attack that exploits the activation maps of the target model to reconstruct its training data. An activation map is the output of a given layer of a model, and it retains meaningful information on the training samples passed into the model.
Our model inversion attack consists of two phases:

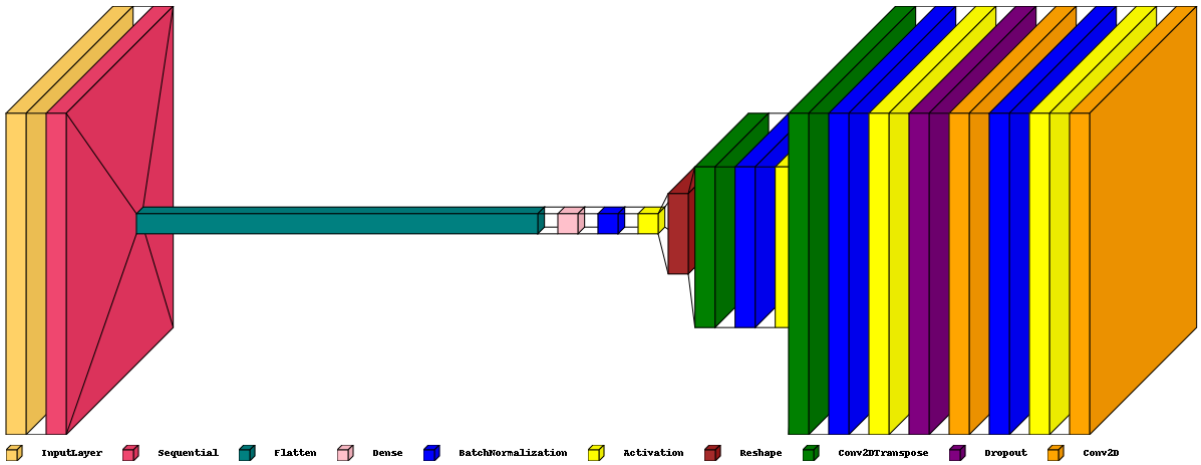
1. train the target model on a given dataset, then freeze all its layers up to the one whose activation map we want to extract;
2. attach the frozen layers at the top of the actual adversary model.

In Figure 4.3 we show more clearly this process.

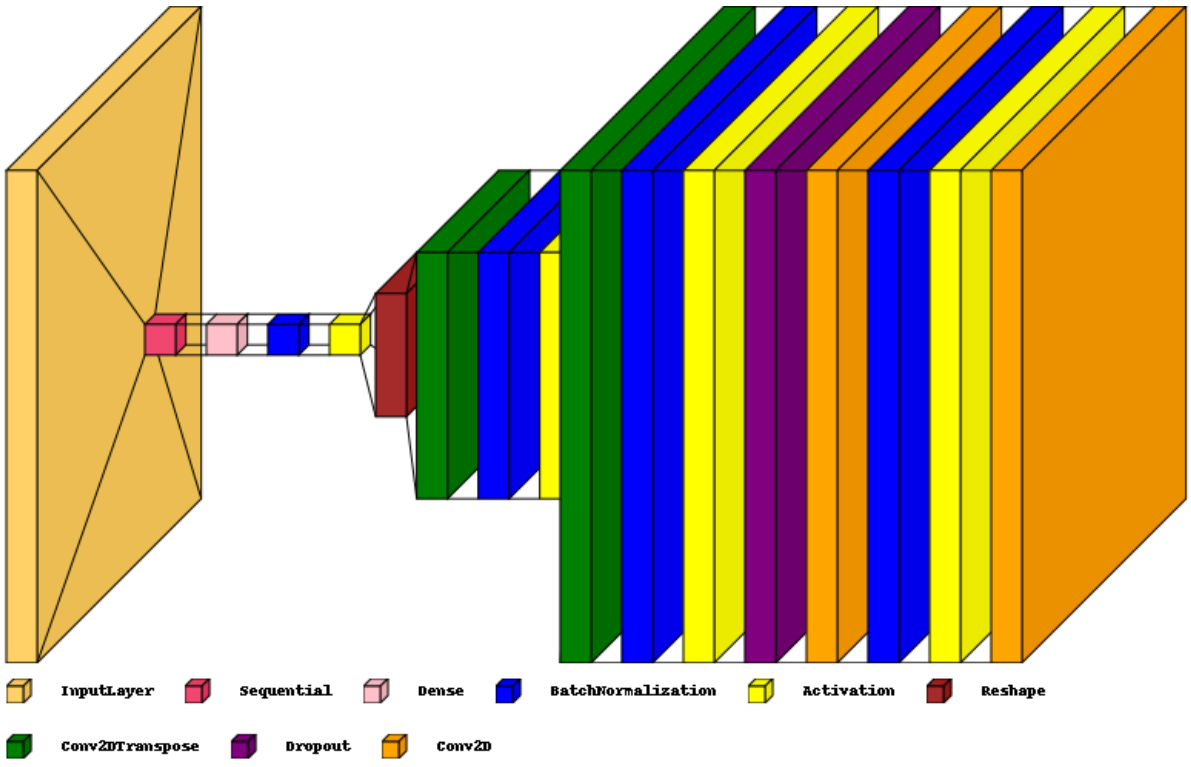
We exploit the capability of our model to process any activation map (explained in detail in the following section) to study how the depth of the activation map influences the resistance to the model inversion attack. In this way, we can evaluate the effectiveness of our attack both in a white-box and in a black-box scenario: in the former, we assume that the adversary knows the target model's structure and extracts specific intermediate maps, while, in the latter, the adversary reconstructs the input data starting from the output of the prediction layer and so assuming to have black-box access to the target model.
All our experiments concerning the evaluation of the protection against the inversion attack are performed layer per layer on the target model.

4.3.1. Attacking model

Our adversary model is structured to be capable of receiving in input any activation map, in terms of the number of dimensions and shape, among the ones produced by the target model's layers. To achieve this, we place after the frozen layers extracted from the target model a **Preprocessing block**, whose structure is presented in detail in Figure 4.3.
This block first flattens the activation map, if necessary, then feed the resulting vector to a Dense layer that maps it to a fixed dimension and, in the end, reshapes the output into a three-dimensional matrix; this last transformation is needed to feed the block output to the convolutional part of the adversary model.
This block also contains a Batch Normalization layer, which maintains the mean of the output close to 0 and its standard deviation close to 1, and the SiLU activation function. We choose this activation function in this block and the convolutional part of the network because its regularization effect significantly improves the performance of the adversary model.
The convolutional block is composed of eleven layers; a relevant role in this section is held by the Transposed Convolution, an operation that combines convolution of images and upsampling, a process opposite to pooling used to increase the spatial dimensions of the input image. The internal structure of the convolutional block is shown in the plot of the adversary model's architecture presented in Figure 4.4. The output of this network is a reconstruction of the image originally fed to the target model.
To measure the reconstruction loss between the original image and the one generated by our model, we use the Mean Squared Error (MSE). Regarding the optimizer for the training process, we decide to use the Adam algorithm [23], a stochastic optimizer that has been widely used in recent deep learning applications.
Moreover, in order to improve the performance and efficiency of our model during training, we adopt two known techniques: Early Stopping and Learning Rate Scheduling. They both act if the reconstruction loss does not decrease for a given number of epochs; when this condition triggers, the first one forcibly interrupts the training process, while the second one reduces the learning rate of a constant factor.



(a) Image activation map



(b) Flat activation map

Figure 4.4: Plots of the attacking model's structures when receiving a three dimensional activation map (a) and a flat activation map (b). The Sequential block is the stack of frozen layers coming from the target model.

5 | Experiments and results

In this chapter, we present and discuss the experiments that we have performed during our work. We will show the results obtained from the experiments and comment on them with the aim of drawing relevant conclusions for the context of our research.
We carried out two separate groups of experiments: one regarding the study of (ε, δ)-differential privacy and a second one related to the study of different regularization scenarios. For both studies, we present the results achieved in terms of accuracy of the target model, training procedure's execution time, and resistance to both membership inference attacks and our model inversion attack. Finally, we performed a final comparison among all the configurations considered in our experiments; in this case, we took into account the aforementioned performance metrics and the protection offered against privacy attacks in a black-box scenario.
First, we describe the datasets on which we performed our experiments; we have considered three image datasets, one containing RGB images
(**CIFAR-10**) and two containing grayscales images (**MNIST** and **Fashion-MNIST**).

CIFAR-10

The CIFAR-10 (Canadian Institute for Advanced Research, ten classes) dataset [24] is a collection of 60000 color images of size 32x32 labeled with one of these 10 mutually exclusive classes: airplane, automobile (but not truck or pickup truck), bird, cat, deer, dog, frog, horse, ship, and truck (but not pickup truck). There are 6000 images per class, divided into 5000 for training and 1000 for testing, so overall, we have 50000 training images and 10000 test images. In our experiments, we reserved 5000 images of the training set to validate the target model.
When we performed the model inversion attack, we trained the attacking model on a subset of data that is disjoint from the target model's training set. In particular, we chose to split the original test set into two subsets, one used to train and validate the attacking model, and the other used to evaluate the final performance of both the target and the attack model. In the case of CIFAR-10, the first subset is formed by 9000 images, 8000 for training and 1000 for validation, while the new test set consists of the remaining 1000

images.

MNIST

The MNIST dataset [28] is a subset of a larger database from NIST (National Institute of Standards and Technology) that contains 70000 grayscales images of handwritten digits. The original images from NIST are normalized to fit in a 28x28 pixel box and are labeled with 10 classes, one for each possible digit.
For the target model, we considered a training set of 54000 images, a validation set of 6000 images, and a test set of 10000 images. When we performed the model inversion attack, we trained the attacking model on 8000 images and validated it on 1000 images; for this dataset, the new test set consists of 1000 images.

Fashion-MNIST

Fashion-MNIST [39] is a dataset of 70000 28x28 grayscales images of fashion articles from Zalando, labeled with 10 classes (t-shirt/top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, ankle boot).
Fashion-MNIST is conceived as a direct replacement for the original MNIST dataset for benchmarking machine learning algorithms, and so, in our experiments, we used the same training, validation, and testing splits of the MNIST dataset.

All the experiments have been executed on an Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz with an Nvidia GeForce GTX TITAN X GPU.

5.1. (ε, δ)-Differential privacy

In the first group of experiments, our goal is to study the effects of applying differential privacy to the target model in terms of performance, total training time, and resilience to privacy attacks; in particular, we tried three different configurations, starting from a strict privacy guarantee and then progressively relaxing it.
To introduce differential privacy in our target network, we use the DP-SGD algorithm as optimizer during the training process; the specific implementation of DP-SGD used in our code is taken from **TensorFlow Privacy** [33].
TensorFlow Privacy is an open-source library by Google Research that provides the code necessary to train privacy-preserving models. In our implementation, we exploit two elements made available by this library: the class ***DPKerasSGDOptimizer***, which implements the DP-SGD using the standard Gaussian mechanism, and the function ***compute_dp_sgd_privacy***, which measures the privacy guarantee of the algorithm in terms of privacy budget ε. Several hyperparameters need to be set and tuned to make this implementation work properly.

- *DPKerasSGDOptimizer* needs:
  - norm clip, a privacy parameter that influences the gradient clipping; after some tuning, we fixed its value to 4 for all our experiments;
  - noise multiplier, a privacy parameter influencing the amount of noise injected in the learning process;
  - number of microbatches, a privacy parameter that indicates in how many microbatches each batch of data is split; this parameter is usually set equal to the batch size used in training, so we set it to 200.
- *compute_dp_sgd_privacy* needs:
  - total number of training data;
  - batch size used in training; for all our experiments, we set this value to 200;
  - noise multiplier;
  - epochs of training of the model;
  - δ, a privacy parameter indicating the probability of failure of the differentially private mechanism; by definition, its value cannot be higher than the inverse of the total number of training data, so we set it to $\min\left(\frac{1}{|Train\ set|}, 10^{-5}\right)$.

The function provided by the library computes the privacy guarantee given a fixed value of δ; this happens because DP-SGD is designed on the theoretical basis of (ε, δ)-differential privacy (see 3.9).
The unique parameter that does not have a fixed value is the noise multiplier; we modified its value each time to obtain the desired level of privacy guarantee, leaving the other parameters unchanged.
In our study of differential privacy, we considered three privacy configurations with decreasing levels of privacy guarantee; in the first configuration, we have $\varepsilon = 2$, in the second one $\varepsilon = 4$ and in the third one $\varepsilon = 8$. We recall that a minor value for the privacy budget ε corresponds to a greater privacy guarantee.
First, we show the performance results of the target model in terms of training (see Figure 5.1), validation (see Figure 5.2), and testing (see Table 5.1) accuracies both in the baseline scenario and the differentially private ones. We train these four models for the same number of epochs to have a fair comparison; this number is fixed for each dataset, and we define it by taking the average of the best epochs reached by the baseline target model on several experiments. In particular, for CIFAR-10, the number of epochs considered is equal to 30, while, for MNIST and Fashion-MNIST, it is respectively 20 and 40. For all the experiments, the batch size used for training the target model is 200, independently from the dataset under examination.

Table 5.1: Comparison between the percentage variations of the accuracy results achieved by the target model in different privacy scenarios with respect to the baseline setting on the test set.

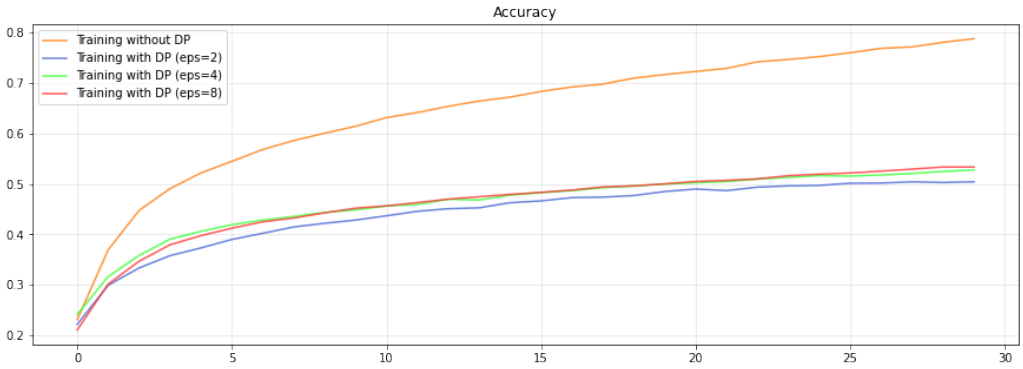
Dataset	$\varepsilon = \mathbf{2}$	$\varepsilon = \mathbf{4}$	$\varepsilon = \mathbf{8}$
CIFAR-10	-13.20	$\mathbf{-11.20}$	-11.40
MNIST	-5.00	-2.90	$\mathbf{-2.40}$
Fashion-MNIST	-7.60	-7.33	$\mathbf{-6.70}$

Table 5.2: Execution times (expressed in seconds) of the target model's training process in different privacy scenarios.

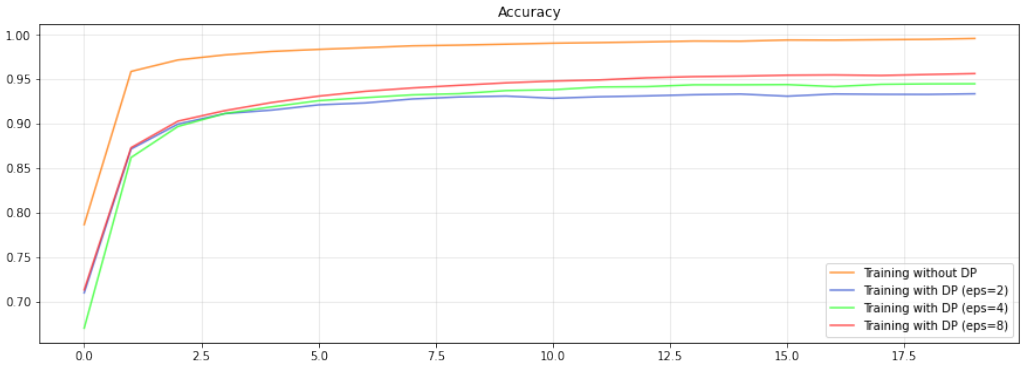
Dataset	**baseline**	$\varepsilon = \mathbf{2}$	$\varepsilon = \mathbf{4}$	$\varepsilon = \mathbf{8}$
CIFAR-10	**73.29**	3029.77	2923.09	2897.05
MNIST	**39.51**	1886.19	1870.23	1859.77
Fashion-MNIST	**77.45**	3876.66	3952.05	3967.11

We draw two conclusions from these results: the first one is that there is a significant degradation of performance with the introduction of differential privacy, confirming the thesis of Bagdasaryan et al. [2], while the second one is that the relaxation of the privacy guarantee leads to a slight improvement in the model's utility.
We have also measured the total time taken to complete the training of the target model both in the baseline scenario and after the application of differential privacy. In particular, we chose for this experiment to train the four models on MNIST for 20 epochs, on CIFAR-10 for 30 epochs, and on Fashion-MNIST for 40 epochs; these values corresponded to the best epochs found training with Early stopping the target model without differential privacy. The results are shown in Table 5.2.

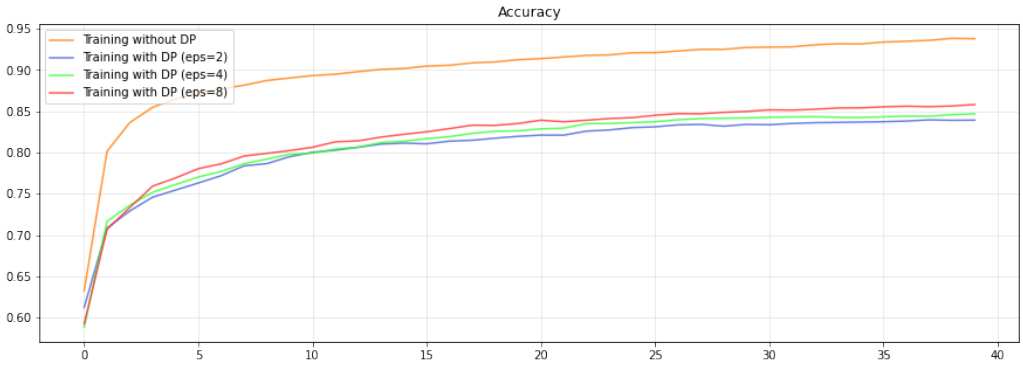
So, we conclude that differential privacy has a significant impact on the model's performance and on the time duration of the training process.
Finally, we analyzed the effects of differential privacy with regard to the protection offered against privacy attacks. First, we present the results for the membership inference attacks; the modalities with which we conducted these attacks are the ones described in the previous chapter. Our implementation of the membership inference attacks is based on a tutorial by Franziska Boenisch [3], which explains how to perform this kind of attack with Tensorflow Privacy, the library we already exploited for implementing differential privacy in our models.
Figure 5.3 shows the success rate of the membership inference attacks for each scenario and dataset. Analyzing the AUC (Area Under Curve) of the ROC curves, we conclude that differential privacy significantly reduces the probability of success of these attacks; we



(a) CIFAR-10

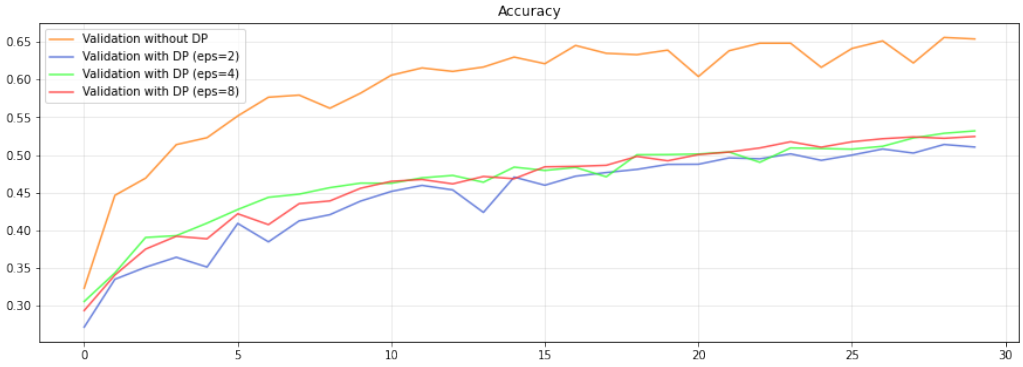


(b) MNIST

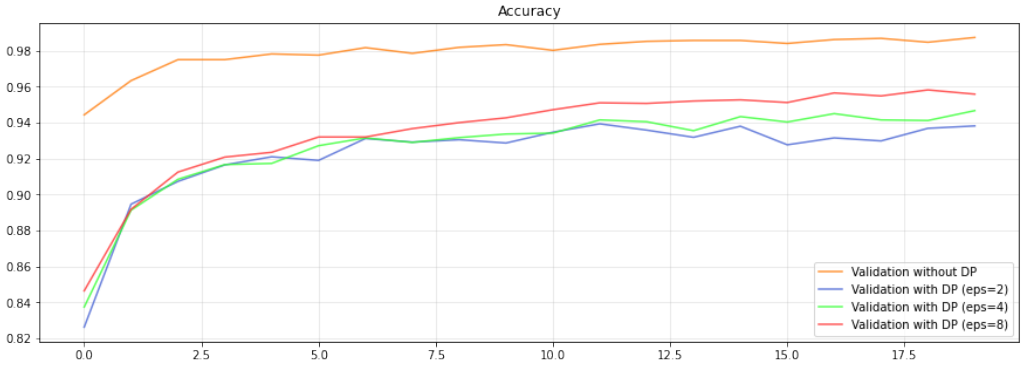


(c) Fashion-MNIST

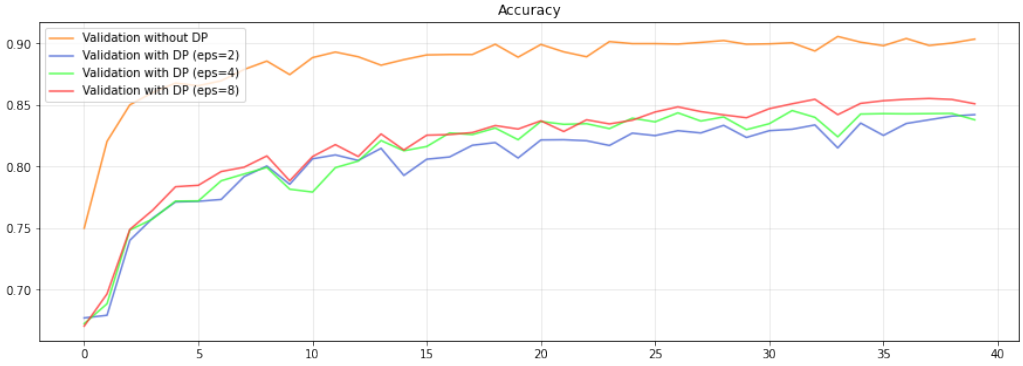
Figure 5.1: Plot with the comparison between histories of training accuracy for the baseline model and the three differentially private ones.



(a) CIFAR-10



(b) MNIST



(c) Fashion-MNIST

Figure 5.2: Plot with the comparison between histories of validation accuracy for the baseline model and the three differentially private ones.

expected this result because existing works substantiate the effectiveness of this privacy-preserving method against membership inference attacks. Moreover, it seems that modifying the level of privacy guarantee does not significantly change the resistance to the attack.

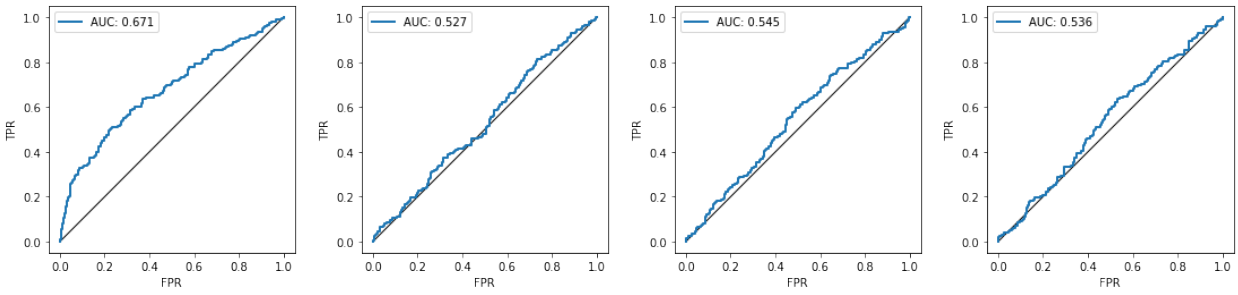
Regarding the protection against our model inversion attack, we conducted a layer per layer analysis for each scenario and evaluated the effectiveness of differential privacy from the reconstruction loss measured on the test set. From the results presented in Figure 5.4, we gather two relevant insights.

The first one is that the adversary has more difficulty in reconstructing the original image from deeper activation maps; this result is reasonable because deeper maps have lesser elements in common with the image fed in input to the model and because they correspond to a learned high-level representation of the input image.

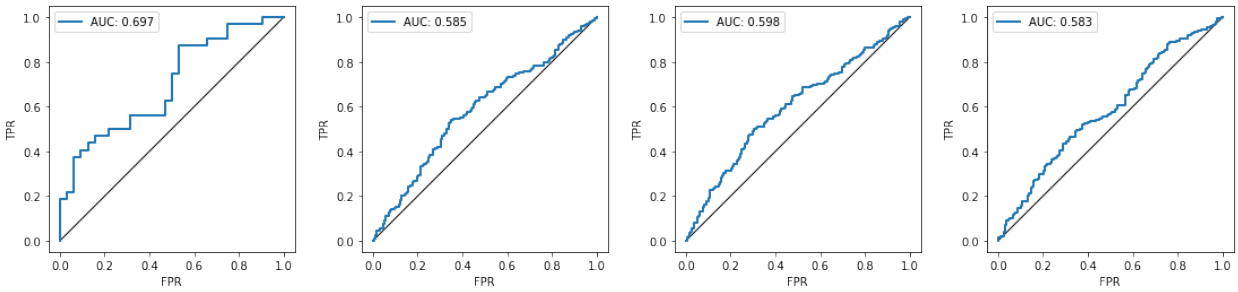
The second insight is that differential privacy does not significantly improve the resilience of the model to the attack; indeed, for some layers it worsens it, especially for the last three. This result is presented in numerical form in Table 5.3. Moreover, the relaxations of the privacy guarantee further worsen the level of protection provided by the method, as expected.

To provide a perception of how these results are reflected in the quality of image reconstruction, in Figure 5.5 we present examples of images recovered by the adversary for each layer and in all the privacy scenarios; from this comparative, it is more clear how, especially in the last three layers, the application of DP increases the success of the attack.

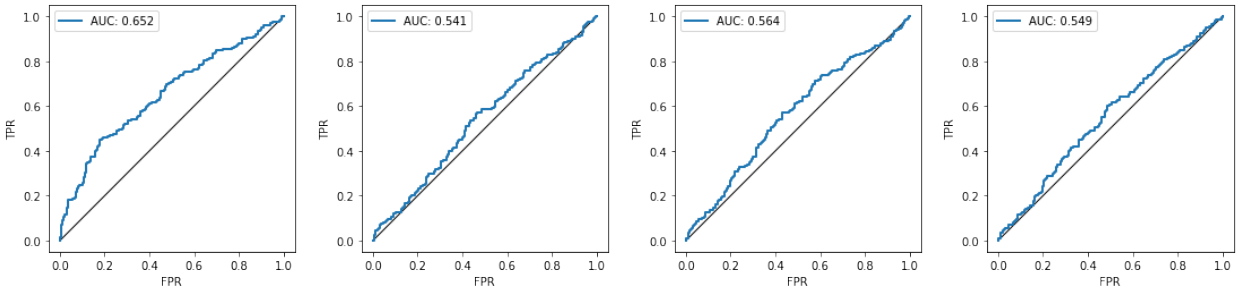
To obtain an overall view of the amount of resistance gained or lost with respect to the baseline scenario, we take the percentage average of the loss variation across all layers; the results are shown in Table 5.4, and from them, we can conclude that $\varepsilon = 2$ remains the best choice among the differentially private configurations in terms of overall protection against the attack.



(a) CIFAR-10



(b) MNIST



(c) Fashion-MNIST

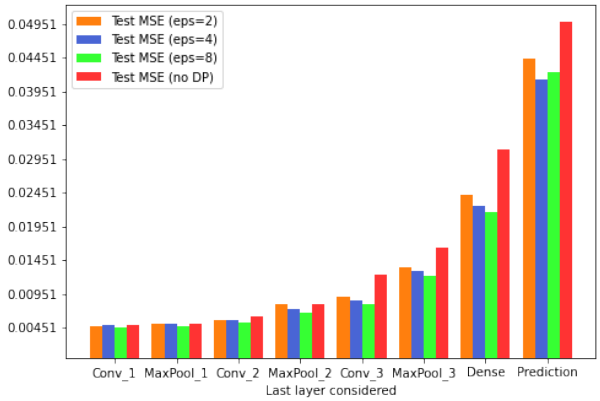
Figure 5.3: ROC curves representing the most successful membership inference attack against each differentially private model. From left to right the curves show respectively attacks against the model without differential privacy, with $\varepsilon = 2$, with $\varepsilon = 4$ and with $\varepsilon = 8$. In this comparison, lower results of AUC are the better ones because they indicate a lower probability of success of the attack.

Table 5.3: Average variation on all datasets of the reconstruction MSE of differentially private models for each layer with respect to the baseline scenario (expressed in %).

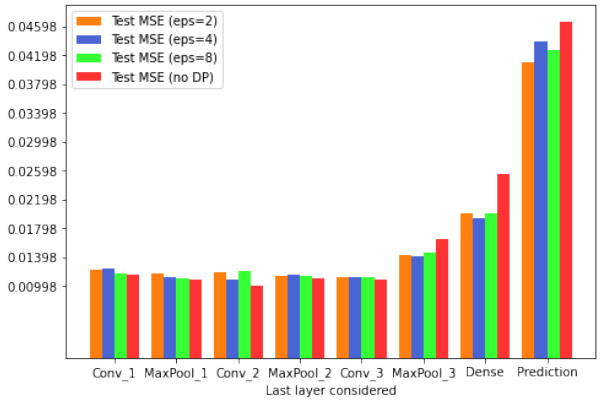
Layer	$\varepsilon = \mathbf{2}$	$\varepsilon = \mathbf{4}$	$\varepsilon = \mathbf{8}$
Conv1	2.1	**2.5**	-1.4
MaxPool1	**3.1**	0.6	1.7
Conv2	**4.6**	0.7	3.2
MaxPool2	**3.4**	-0.5	-1.3
Conv3	**4.2**	-9.3	-10.5
MaxPool3	**−14.6**	-15.6	-17.2
Dense	**−21.7**	-25.8	-25.2
Prediction	-17.7	-18.2	**−15.8**

Table 5.4: Average variation on all layers of the reconstruction MSE of differentially private models for each dataset with respect to the baseline scenario (expressed in %).

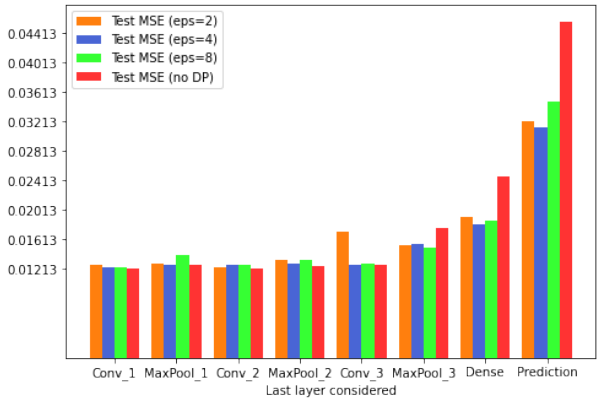
Dataset	$\varepsilon = \mathbf{2}$	$\varepsilon = \mathbf{4}$	$\varepsilon = \mathbf{8}$
CIFAR-10	**−10.7**	-14.5	-18.9
MNIST	**−1.1**	-2.4	-1.3
Fashion-MNIST	**−1.9**	-7.7	-4.7



(a) CIFAR-10

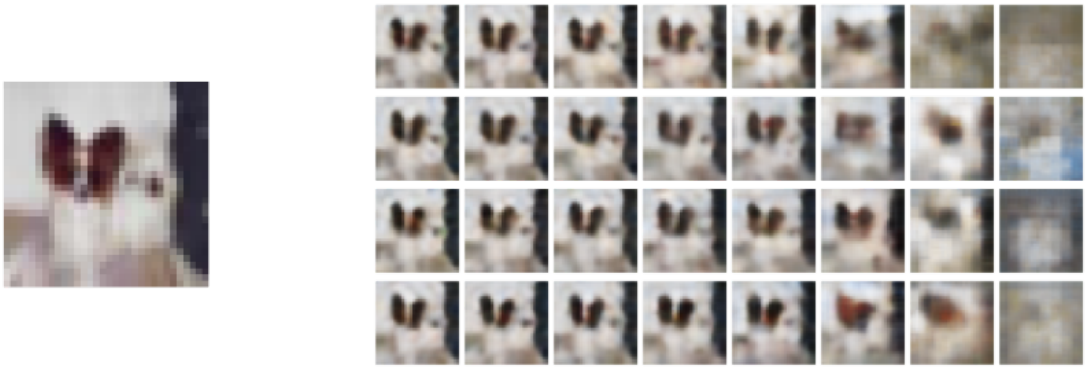


(b) MNIST

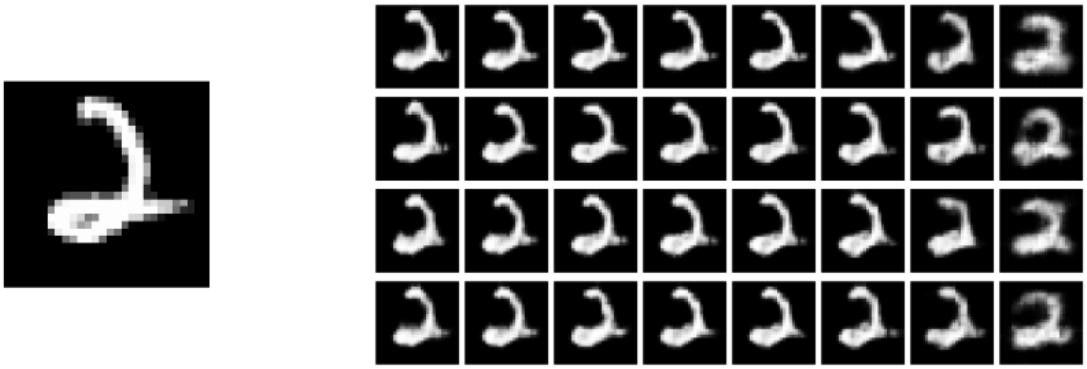


(c) Fashion-MNIST

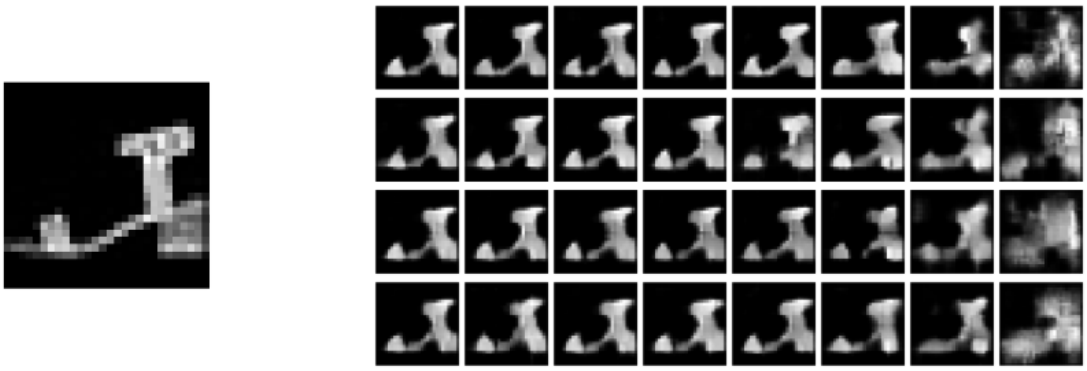
Figure 5.4: Comparisons between reconstruction mean squared errors obtained attacking the four models (baseline and differentially private ones) layer per layer. We see how the gap between the effectiveness of the attack with and without DP increases noticeably in the last three layers, and DP becomes less and less useful as a defense mechanism as we consider deeper maps.



(a) CIFAR-10



(b) MNIST



(c) Fashion-MNIST

Figure 5.5: The original image (on the left) and comparisons between images reconstructed by the adversary at each layer in each privacy scenario (on the right). In each row from top to bottom, we have reconstruction respectively without DP, with $\varepsilon = 2$, with $\varepsilon = 4$ and with $\varepsilon = 8$. The leftmost column corresponds to the layer closer to the input and, as we proceed to the right, we consider the other layers, progressively getting closer to the output of the network, to which corresponds the rightmost column. We see how, in the last three columns, the perceptual differences between the images reconstructed by each model are more remarkable compared to the other layers.

5.2. Regularizers

In the previous section, we have demonstrated that differential privacy does not offer any significant protection against our model inversion attack; moreover, its application has a relevant impact on both the accuracy of the target model and the time duration of its training process. However, we have also seen that differential privacy still represents a valid defense mechanism against membership inference attacks.
For this reason, we start searching for an alternative solution that can overcome the drawbacks of differential privacy and, at the same time, behave equally well against membership inference attacks.
Our proposal is to introduce regularization into the target model using two different techniques, i.e., Dropout and L2 (or Ridge) regularization. To implement them inside our code, we exploit two classes provided by **Keras**, the deep learning API of TensorFlow: ***layers.Dropout*** and ***regularizers.L2***.
The first one provides a layer that randomly sets input units to 0 with a probability that can be set through the parameter *rate*; after some tuning, we set this value to 0.2 because it guarantees the best result in terms of both accuracy and protection.
The second class allows to penalize one layer's parameters (weights and bias) or one layer's output; in our approach, we decided to apply penalties only on the matrix of weights (also called kernel) of the layer. Keras layers expose a keyword argument for this kind of penalization, called ***kernel_regularizer***; so, in our approach, it sufficed to set this argument equal to ***regularizers.L2*** for the layer we wanted to regularize (the output one). This element also has an attribute that needs to be tuned, the regularization factor *l2*; following the same criterion used for the dropout rate, in the end, we set its value to 0.02.
In this group of experiments, we tried three scenarios of regularization: use of L2 regularization, use of dropout, and use of both techniques together. As we have done for the study of differential privacy, we first present the accuracy performance achieved by the target model (Figures 5.6 and 5.7 and Table 5.5). The number of epochs of training considered for each dataset is the same used for the study of differential privacy.

Table 5.5: Comparison between the percentage variations of the accuracy results achieved by the target model in different regularization scenarios with respect to the baseline setting on the test set.

Dataset	L2	Dropout	L2+Dropout
CIFAR-10	-1.50	**2.80**	-1.60
MNIST	-0.10	**0.40**	0.20
Fashion-MNIST	**−0.4**	-1.00	-2.1

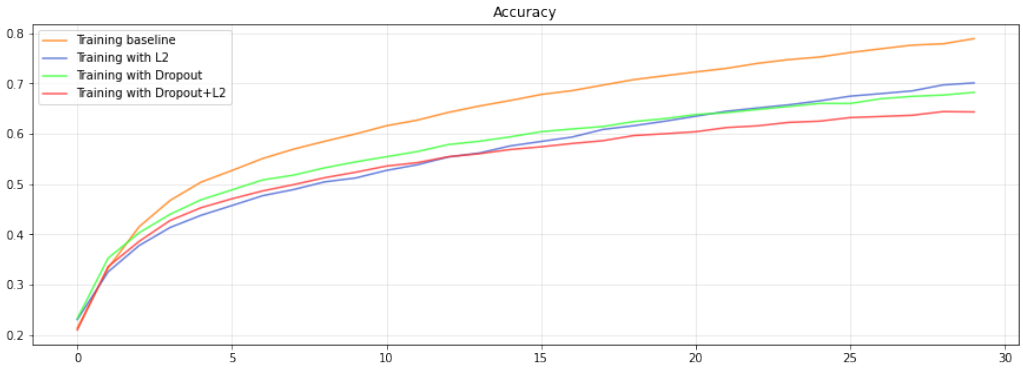
Table 5.6: Execution times (expressed in seconds) of the target model's training process in different regularization scenarios.

Dataset	baseline	L2	Dropout	L2+Dropout
CIFAR-10	73.29	69.11	**67.35**	70.38
MNIST	39.51	31.19	**29.46**	29.55
Fashion-MNIST	**77.45**	78.22	82.99	78.50

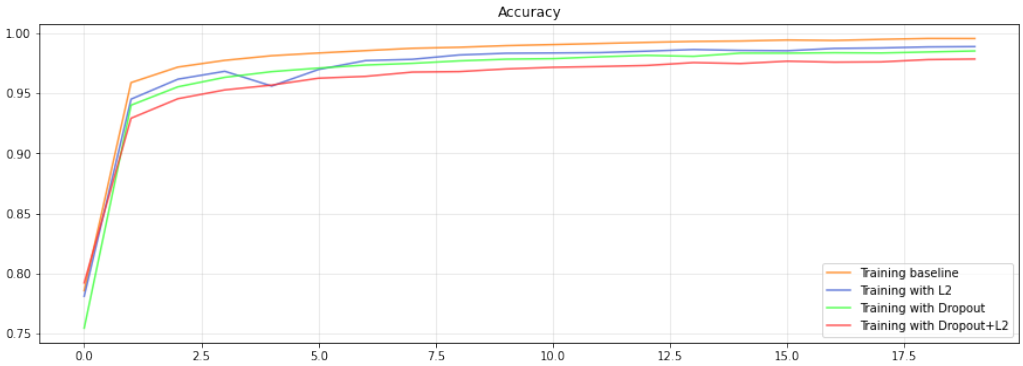
As we expected, the application of regularization does not impact noticeably on the model's utility and, in some cases, even improves the accuracy level reached by it.
Concerning the time performance, there is no significant change in the total time required to complete the training process, as can be seen in Table 5.6.
Then, we proceed to show the results regarding the protection against privacy attacks. Both dropout alone and the combination of L2 regularization and dropout offer reasonable protection against membership inference attacks, as it is shown in Figure 5.8.

Regarding the inversion attack, we gather several interesting results (see Figure 5.9 and Tables 5.7 and 5.8):

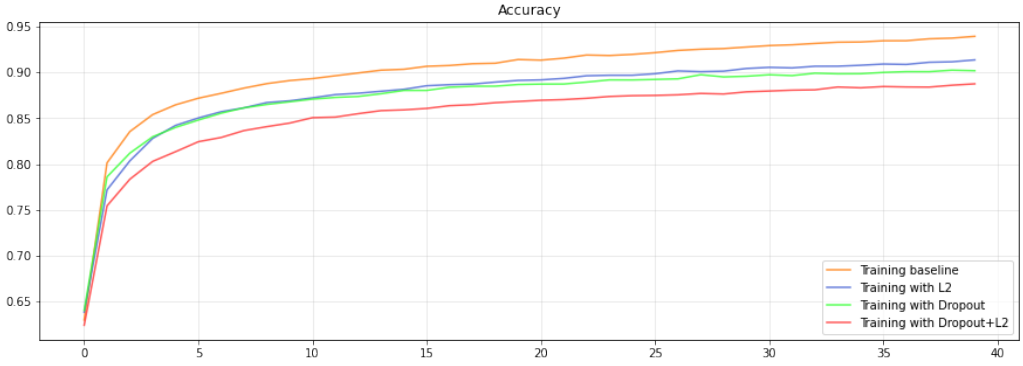
- L2 regularization significantly increases the resistance to the attack when the extracted activation map belongs to the regularized layer. In our case, for the attacker is more difficult to reconstruct the input from the prediction vector, representing the activation map associated with the output layer, with respect to the baseline scenario.
- Dropout increases the overall resistance to the attack against intermediate layers, but it does not improve the protection of the output layer.
- Combining L2 regularization and dropout, we obtain a compromise that guarantees a slight improvement in the level of protection of the intermediate layers and significant growth in the resilience to the attack of the output layer.



(a) CIFAR-10

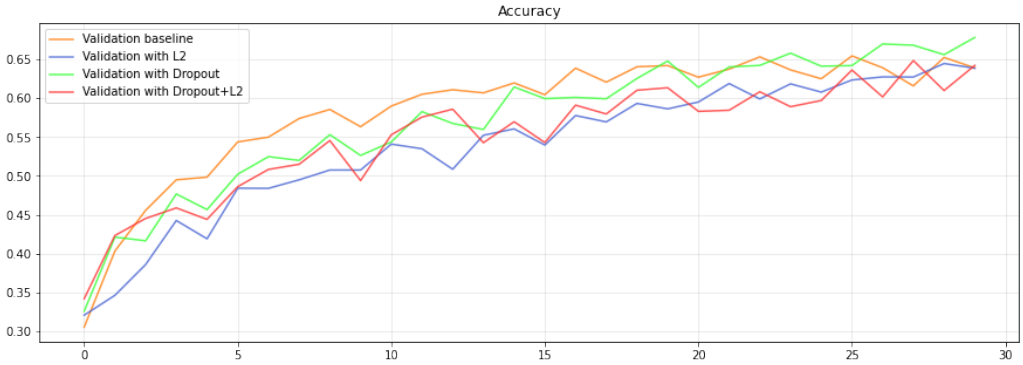


(b) MNIST

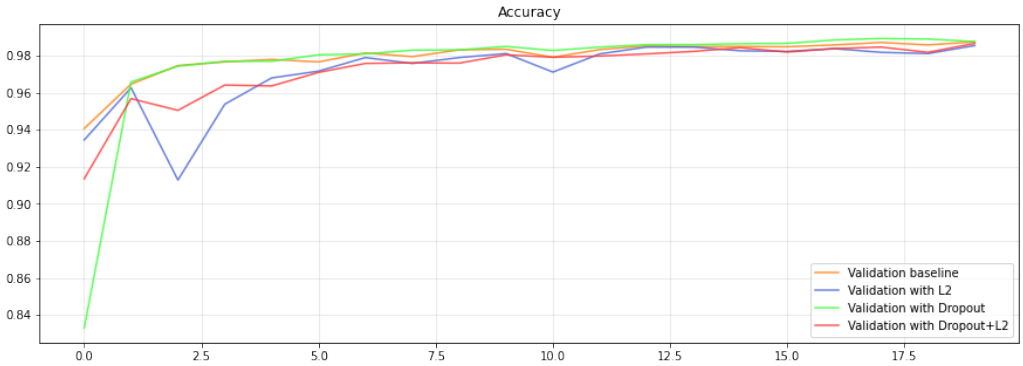


(c) Fashion-MNIST

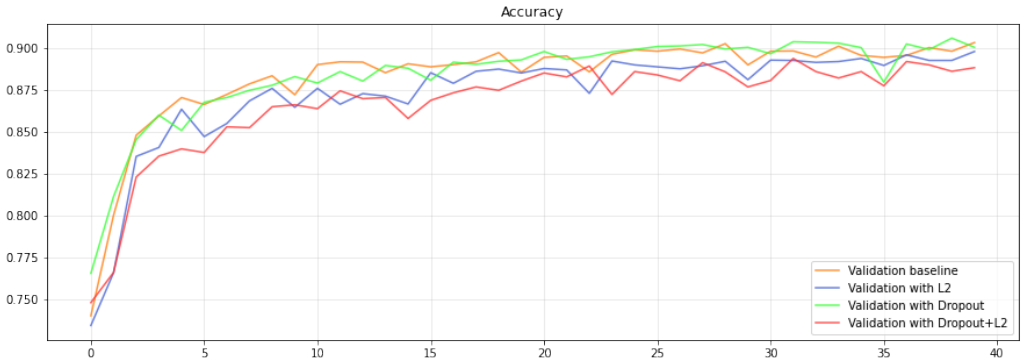
Figure 5.6: Plot with the comparison between histories of training accuracy for the baseline model and the three regularized ones.



(a) CIFAR-10

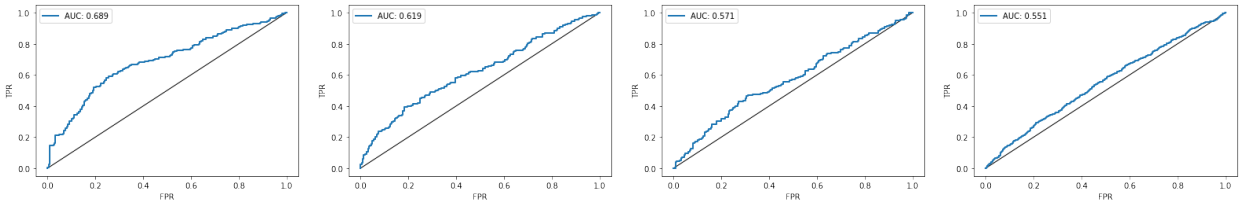


(b) MNIST

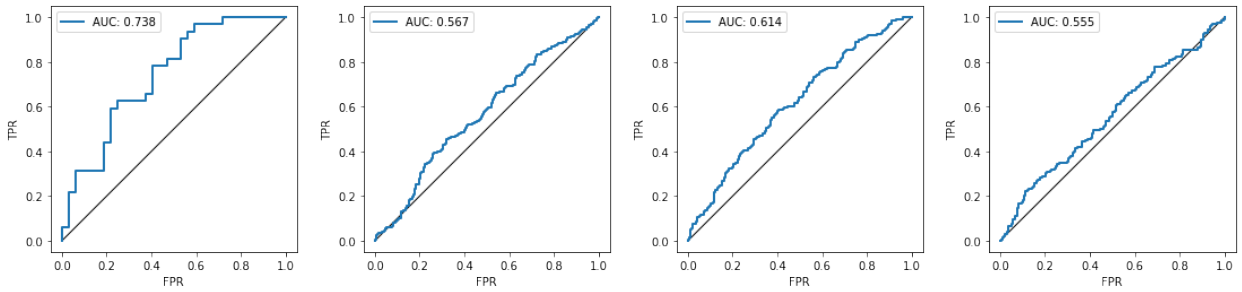


(c) Fashion-MNIST

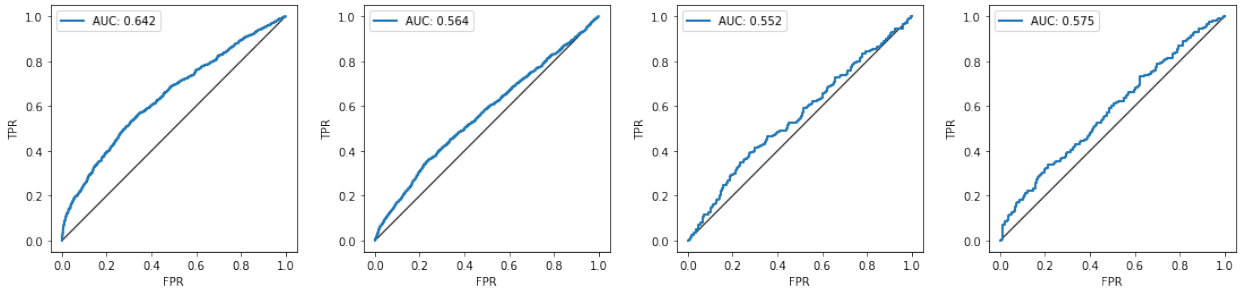
Figure 5.7: Plot with the comparison between histories of validation accuracy for the baseline model and the three regularized ones.



(a) CIFAR-10



(b) MNIST



(c) Fashion-MNIST

Figure 5.8: ROC curves representing the most successful membership inference attack against each regularized model. From left to right, the curves show respectively attacks against the baseline model, model with L2 regularization, model with dropout, and model with both L2 regularization and dropout. In this comparison, lower results of AUC are the better ones because they indicate a lower probability of success of the attack.

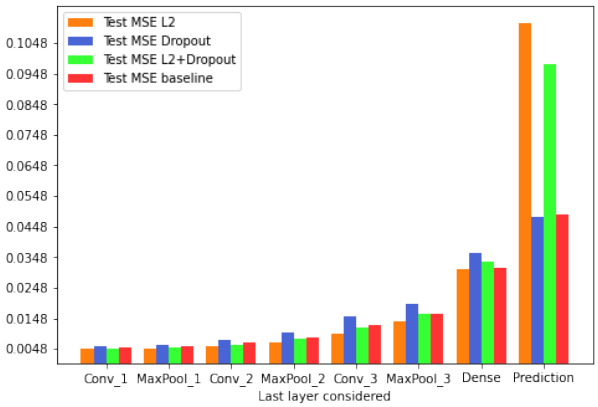
Table 5.7: Average variation on all datasets of the reconstruction MSE of regularized models for each layer with respect to the baseline scenario (expressed in %).

Layer	L2	Dropout	L2+Dropout
Conv1	-2.7	**19.3**	8.5
MaxPool1	-4.5	**13.9**	-4.5
Conv2	-5.9	**14.1**	-1.7
MaxPool2	-5.5	**25.2**	1.7
Conv3	-7.6	**10.7**	2.1
MaxPool3	-10.3	**9.9**	-2.4
Dense	14.5	14.7	**18.6**
Prediction	**78.4**	0.7	68.5

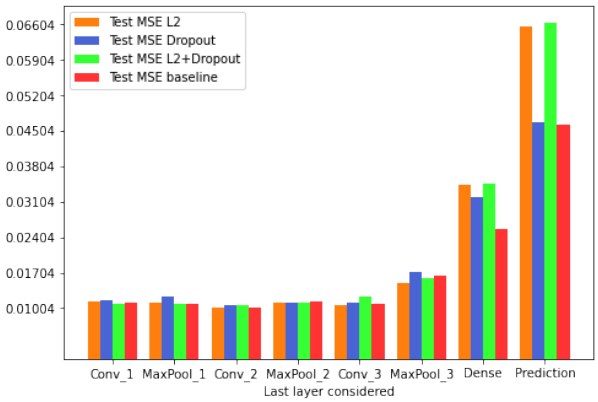
Table 5.8: Average variation on all layers of the reconstruction MSE of regularized models for each dataset with respect to the baseline scenario (expressed in %).

Dataset	L2	Dropout	L2+Dropout
CIFAR-10	3.7	**11.9**	8.3
MNIST	8.4	6.8	**11.4**
Fashion-MNIST	9.0	**22.0**	14.4

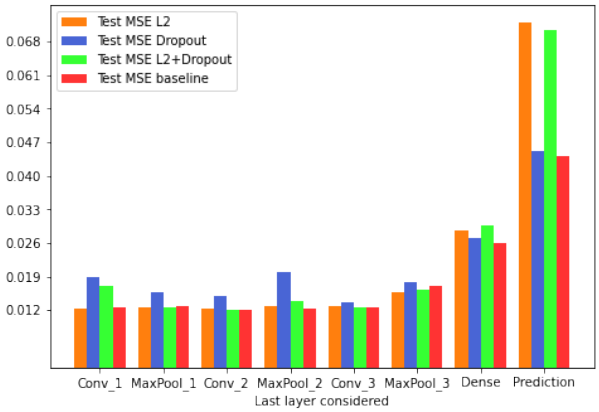
To sum up these results, we can conclude that, if we evaluate these solutions in terms of overall protection against both attacks, the combination of L2 regularization and dropout is the best solution; otherwise, if we consider only the model inversion attack, applying L2 regularization to the output layer provides the best protection in a black-box scenario. For completeness, in Figure 5.10 we also report the comparison between images reconstructed by the adversary in each scenario.



(a) CIFAR-10

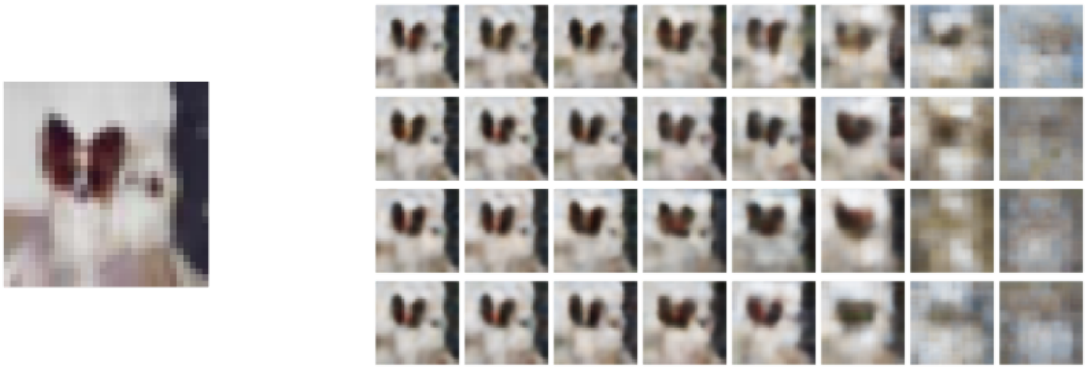


(b) MNIST

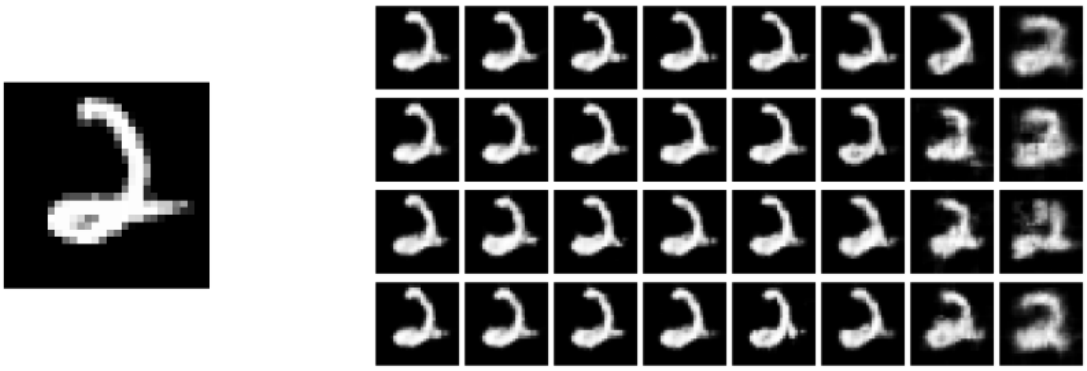


(c) Fashion-MNIST

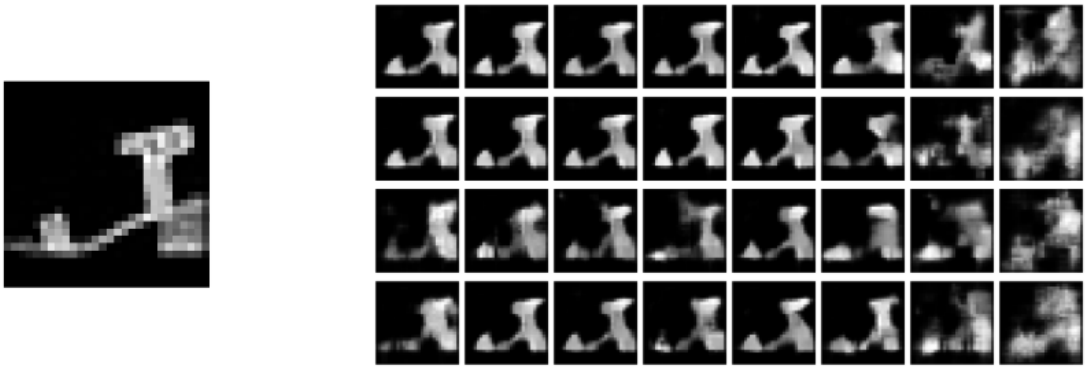
Figure 5.9: Comparisons between reconstruction mean squared errors obtained attacking the four models (baseline and regularized ones) layer per layer. We see that dropout provides the best protection for the intermediate layers of the network and that the resistance to the attack of the output layer is noticeably increased by the use of L2 regularization.



(a) CIFAR-10



(b) MNIST



(c) Fashion-MNIST

Figure 5.10: The original image (on the left) and comparisons between images reconstructed by the adversary at each layer in each regularization scenario (on the right). In each row from top to bottom, we have reconstruction respectively from the baseline model, the model with L2, the model with dropout, and the model with both L2 regularization and dropout. We can see that the reconstructions in the third row, corresponding to the dropout model, are of inferior quality compared to the other models when we consider the intermediate layers. Instead, when we analyze the images of the last column, corresponding to the output layer, we see that the worst reconstructions are the ones provided by the L2 and L2+dropout models (second and fourth rows, respectively).

Table 5.9: Comparison between the percentage variations of the accuracy results achieved by the target model in different best cases scenarios with respect to the baseline setting on the test set.

Dataset	DP	L2+Dropout	DP+Dropout
CIFAR-10	-17.00	**0.30**	-16.60
MNIST	-4.90	**0.20**	-2.10
Fashion-MNIST	-7.40	**−1.30**	-8.40

Table 5.10: Execution times (expressed in seconds) of the target model's training process in different best cases scenarios.

Dataset	baseline	DP	L2+Dropout	DP+Dropout
CIFAR-10	73.29	3029.77	**70.38**	3198.98
MNIST	39.51	1886.19	**29.55**	2043.52
Fashion-MNIST	**77.45**	3876.66	78.50	4325.32

5.3. Final results

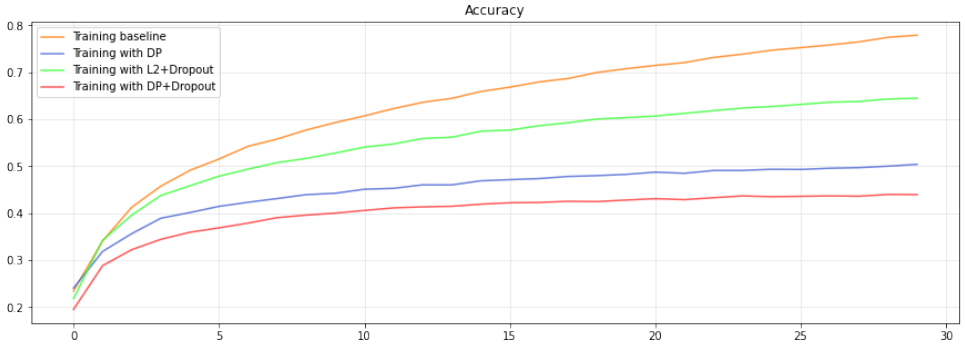
At this point in our work, we have discovered the lack of resilience of differential privacy to the model inversion attack. We have also found an alternative solution capable of achieving a satisfying compromise between protection to privacy attacks, both model inversion and membership inference attacks, utility of the target model, and time duration of the training process.

In this section, we first compare directly the best solutions obtained by the two previous studies, which are differential privacy with $\varepsilon = 2$ and the combination of L2 regularization and dropout; moreover, we added to this comparison a new case that combines differential privacy with $\varepsilon = 2$ and the dropout.

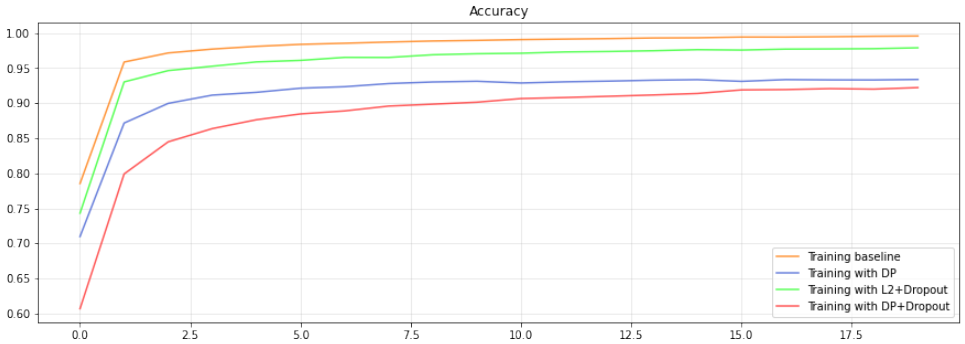
We focused on the results of this last case because we were interested in finding out if dropout can mitigate the drawbacks of differential privacy in terms of resistance to the inversion attack and performance.

From Figures 5.11 and 5.12 and Table 5.9, we can see that the level of accuracy achieved by the model does not deviate much from the one reached by differential privacy alone. Regarding the execution time, it is even greater than the one required for training the model with differential privacy only, as it is shown in Table 5.10.

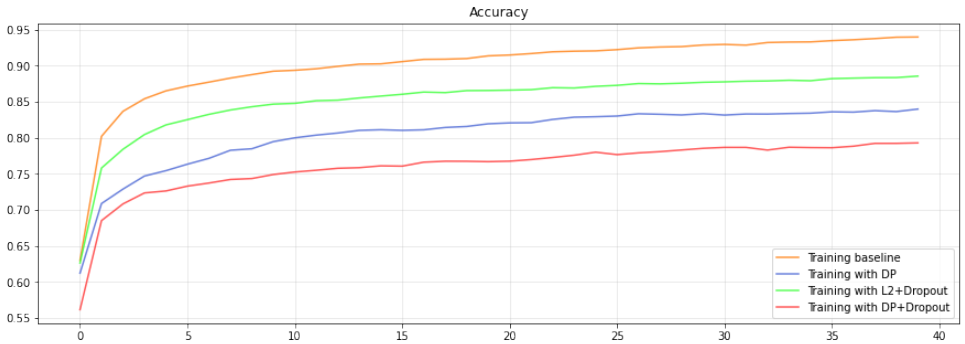
The resistance provided by the combination of differential privacy and dropout to the membership inference attacks is almost at the same level as the one of the solution with L2 and dropout together. The results of these attacks are shown in Figure 5.13.



(a) CIFAR-10

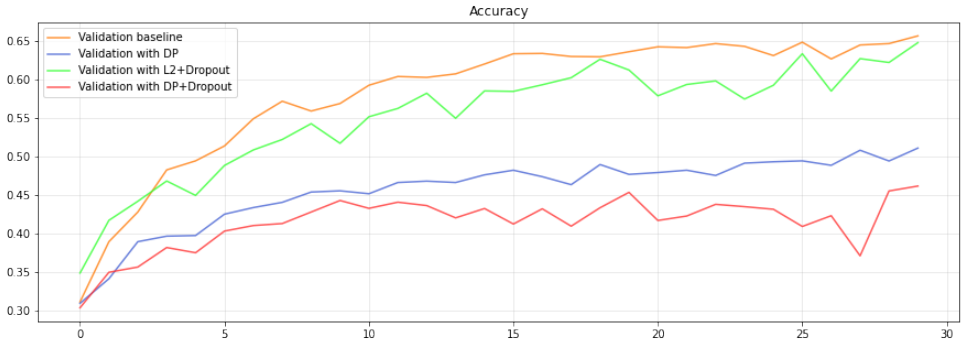


(b) MNIST

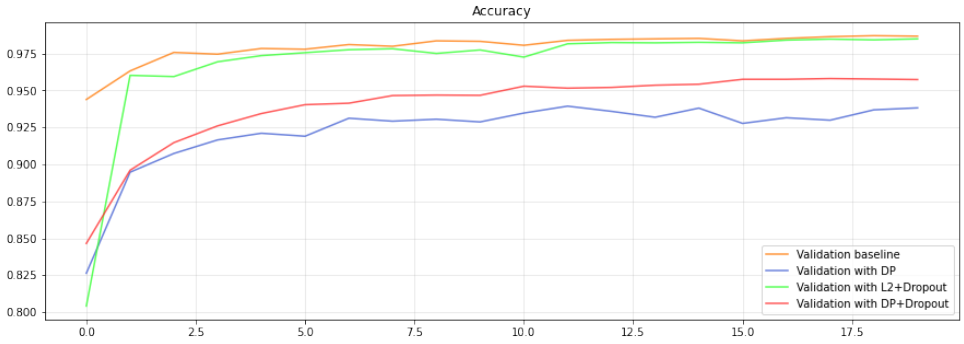


(c) Fashion-MNIST

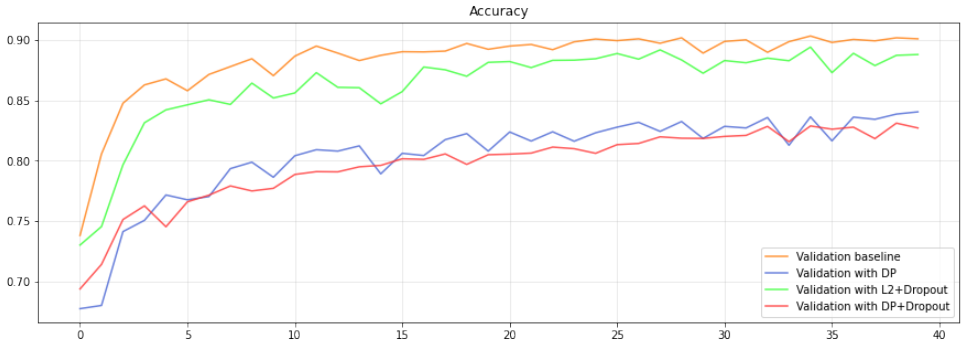
Figure 5.11: Plot with the comparison between histories of training accuracy for the baseline model and the best solutions of previous studies.



(a) CIFAR-10



(b) MNIST



(c) Fashion-MNIST

Figure 5.12: Plot with the comparison between histories of validation accuracy for the baseline model and and the best solutions of previous studies.

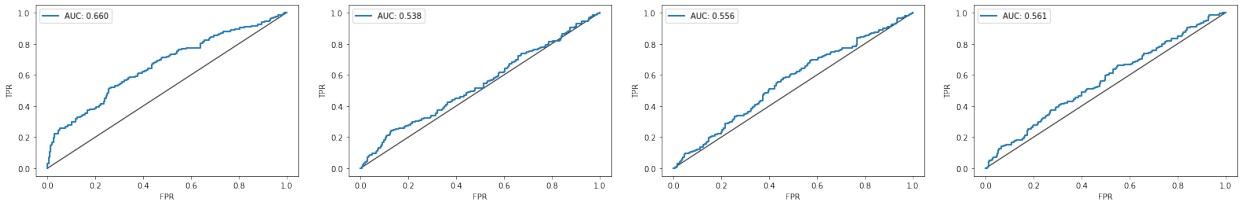
Table 5.11: Average variation on all datasets of the reconstruction MSE of the best cases models for each layer with respect to the baseline scenario (expressed in %).

Layer	DP	L2+Dropout	DP+Dropout
Conv1	2.9	-1.3	**17.3**
MaxPool1	**4.1**	2.2	1.9
Conv2	5.2	6.4	**10.6**
MaxPool2	4.1	9.6	**14.4**
Conv3	1.1	1.8	**5.2**
MaxPool3	-8.1	0.3	**8.0**
Dense	-22.1	**15.2**	0.5
Prediction	-17.8	**65.5**	-11.1

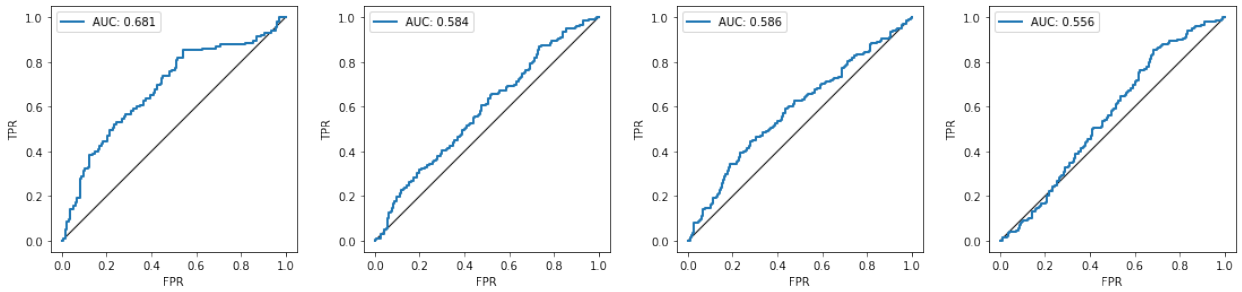
Table 5.12: Average variation on all layers of the reconstruction MSE of the best cases models for each dataset with respect to the baseline scenario (expressed in %).

Dataset	DP	L2+Dropout	DP+Dropout
CIFAR-10	-5.0	**11.7**	8.6
MNIST	-1.3	**7.8**	1.8
Fashion-MNIST	-5.3	**17.8**	7.1

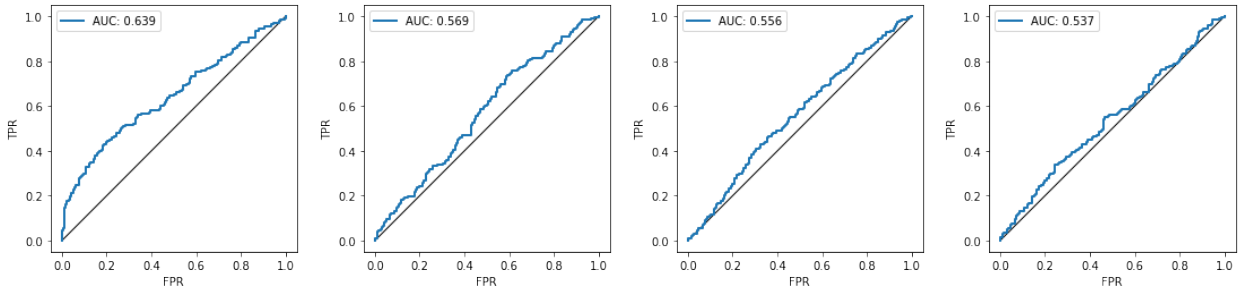
Regarding the model inversion attack, this solution provides higher protection with respect to the original differentially private scenario but still lower than the one provided by the case with both L2 regularization and dropout.



(a) CIFAR-10

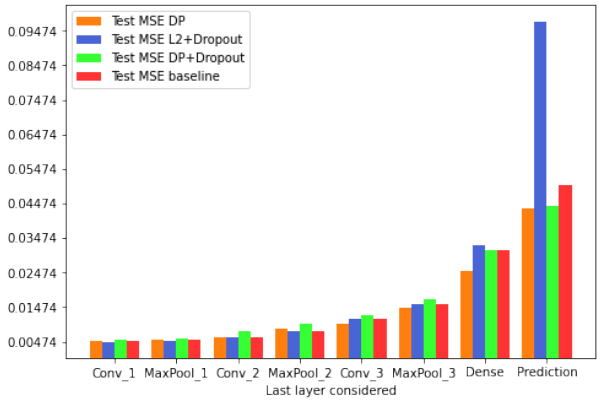


(b) MNIST

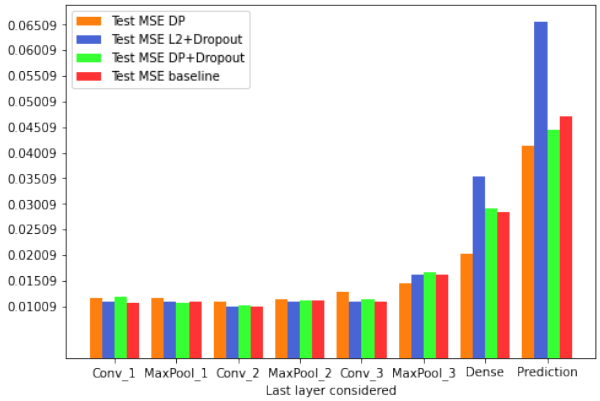


(c) Fashion-MNIST

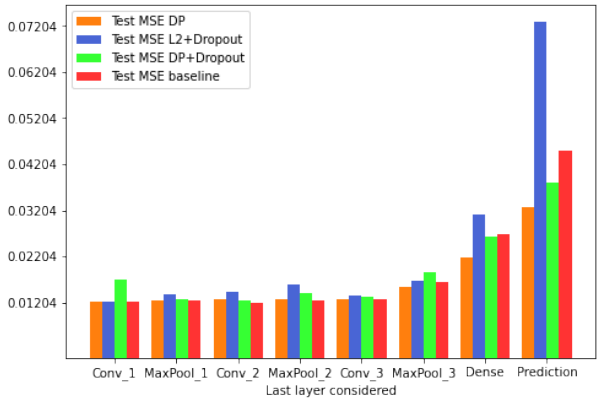
Figure 5.13: ROC curves representing the most successful membership inference attack against each best case model. From left to right, the curves show respectively attacks against the baseline model, model with DP ($\varepsilon = 2$), model with L2+dropout, and model with both DP ($\varepsilon = 2$) and dropout. In this comparison, lower results of AUC are the better ones because they indicate a lower probability of success of the attack.



(a) CIFAR-10

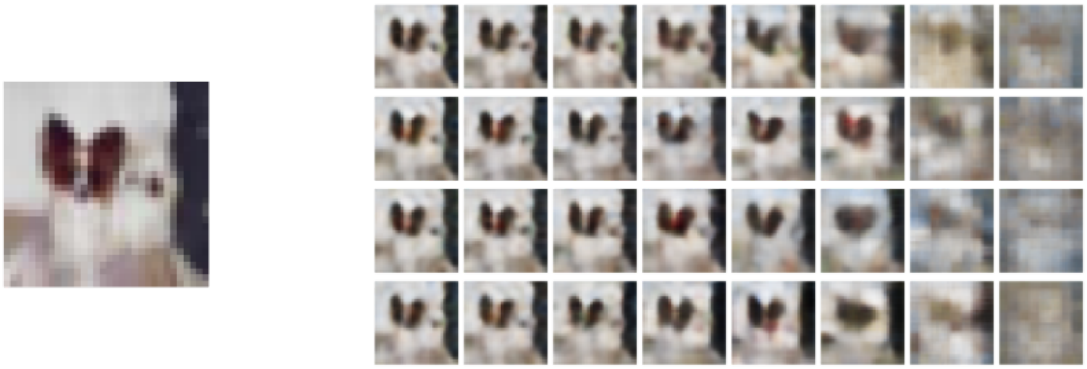


(b) MNIST

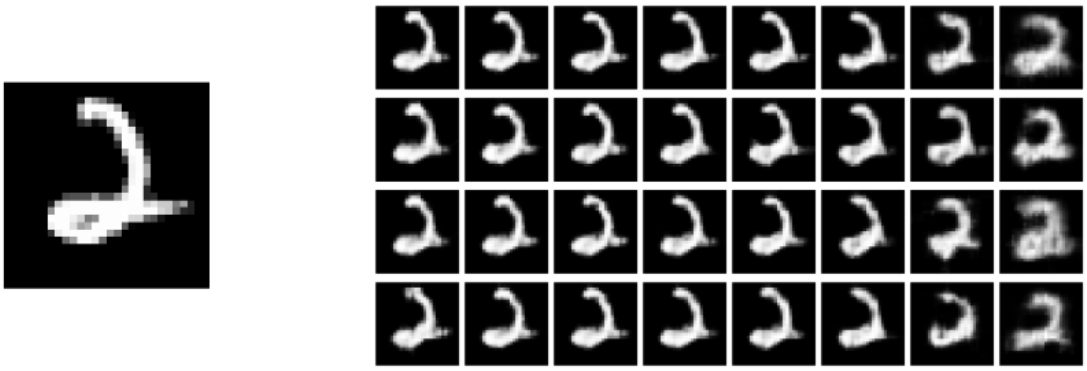


(c) Fashion-MNIST

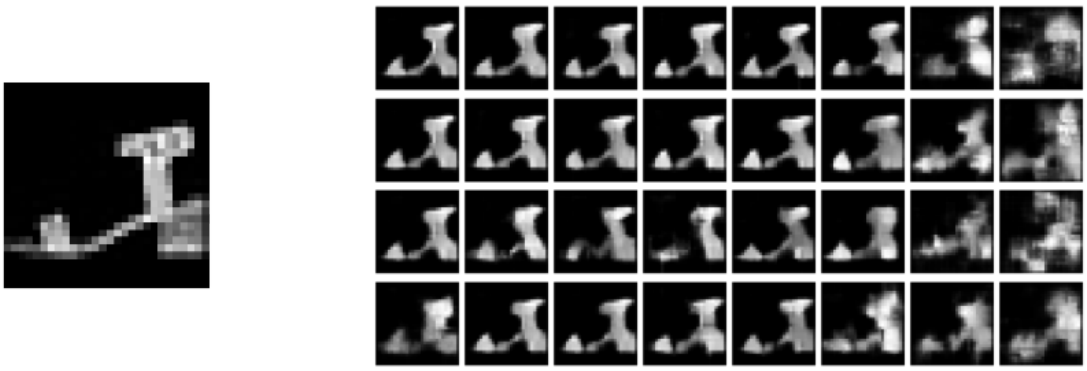
Figure 5.14: Comparisons between reconstruction mean squared errors obtained attacking the four models (baseline and best cases) layer per layer. We see that DP ($\varepsilon = 2$)+dropout represents the best solution in terms of resistance to the attack provided to the intermediate layers, while L2+dropout guarantees a reasonable level of protection for the intermediate layers and is the best defense mechanism for the last two layers.



(a) CIFAR-10



(b) MNIST



(c) Fashion-MNIST

Figure 5.15: The original image (on the left) and comparisons between images reconstructed by the adversary at each layer in each best case scenario (on the right). In each row from top to bottom, we have reconstruction respectively from the baseline model, the model with DP ($\varepsilon = 2$), the model with both L2 regularization and dropout, and the model with DP ($\varepsilon = 2$) and dropout. We see that the reconstructions in the third and fourth rows, respectively corresponding to the models with L2+dropout and DP ($\varepsilon = 2$)+dropout, are worse than the ones produced by the other two models in the first six columns. Instead, for the last two layers, corresponding to the last two columns, the worst reconstruction belong to the L2+dropout model (third row).

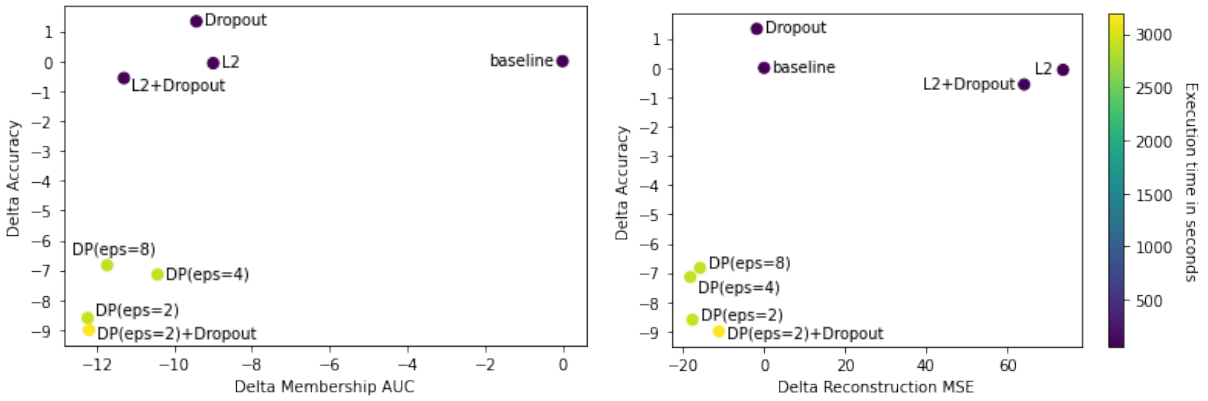


Figure 5.16: Final comparison showing the performances of all the scenarios considered during our studies evaluated with all four metrics used in our experiments. We express the three metrics put on the axes of the two plots in terms of average percentage variation with respect to the baseline scenario (whose values are equal to 84.6% for the accuracy, 67% for the membership AUC and 0.047 for the reconstruction MSE); the average execution time of the training process is represented in the plot through color gradients of the points. In the left chart, the scenarios we would like to consider are the ones in the top left corner of the plot because, in that area, we will have both high accuracy of the target model and more resistance to the membership inference attack. In the right chart, the area of interest is the top right corner, where we have scenarios with higher accuracy and more resilience to the black-box model inversion attack.

At the end of our work, we provided a final overview containing all the results obtained from our experiments; in this way, we can sum up the conclusions drawn from the studies of all the considered scenarios. In this final comparison, shown in Figure 5.16, we took into account together all the four metrics used throughout our experiments: reconstruction MSE for the model inversion attack, the AUC for the membership inference ones, the accuracy of the target model on the test set, and the time duration of the training process. In particular, we assumed that both privacy attacks occur in a black-box scenario to consider the most interesting threat model from a research perspective.
This final collection of results confirmed the conclusion drawn in the previous studies: the solution with both dropout and L2 regularization is the best overall defense mechanism against privacy attacks that also provides good performances in terms of utility and execution time. However, applying L2 regularization alone at the output layer is a valid alternative solution that performs better in protecting from the inversion attack but slightly worse against membership inference ones, with similar performance regarding the accuracy reached and the execution time of the training process.

6 | Conclusions and Future Works

In this thesis, we demonstrated the drawbacks of using differential privacy as a privacy-preservation method for deep learning models; in particular, we showed its significant impact on the performance of the model under attack, both in terms of the level of accuracy achieved and time duration of the training process, and its lack of effectiveness in protecting against a model inversion attack designed by us.

Moreover, we found out that introducing regularization in the target model is a solution that provided a good compromise between the preservation of the model's performance and resistance to both our inversion attack and membership inference ones. We carried out experiments using two regularization techniques, dropout, and the L2 regularizer, with the same criteria of evaluation used for studying differential privacy. At the end of our work, we compared all the results obtained, both with differential privacy and the regularizers, and we concluded that, against both black-box and white-box privacy attacks, applying dropout and L2 regularization to the output layer of the target model is the best defense mechanism.

However, we noticed that L2 regularization alone is the best solution in the case of a black-box model inversion attack but lacks resistance to membership inference ones.

These conclusions were drawn relying on empirical results obtained from our experiments; we have not formalized a theoretical basis for explaining the effectiveness of this solution, neither we have a method to calculate the privacy level reached in regularization scenarios comparable to the privacy accountant used for differential privacy.

For this reason, one future development of this study can consist in designing an algorithm capable of measuring privacy leakage in such contexts and mathematically formalizing the relationship between overfitting and protection against privacy attacks.

From our experiments, we have come to think of L2 regularization as a targeted defense mechanism against the model inversion attack due to the significant protection provided to the regularized layer.

In future works, this hypothesis can be proven by trying to apply L2 regularization at different layers of the network and evaluating the resistance to the inversion attack of those same layers.

Another interesting starting point for further development of our work can be to prove other regularizers, such as the L1 regularizer or Gaussian dropout, to see if any of them overcomes the results obtained with L2 regularization and dropout.

Bibliography

[1] M. Abadi, A. Chu, I. Goodfellow, H. McMahan, I. Mironov, K. Talwar, and L. Zhang. Deep learning with differential privacy. pages 308–318, July 2016. doi: 10.1145/2976749.2978318.

[2] E. Bagdasaryan, O. Poursaeed, and V. Shmatikov. *Differential Privacy Has Disparate Impact on Model Accuracy*. Curran Associates Inc., Red Hook, NY, USA, 2019. doi: 10.5555/3454287.3455674.

[3] F. Boenisch. Attacks against machine learning privacy (part 2): Membership inference attacks with tensorflow privacy, 2021. URL https://franziska-boenisch.de/posts/2021/01/membership-inference/.

[4] J. Brownlee. A gentle introduction to cross-entropy for machine learning, 2019. URL https://machinelearningmastery.com/cross-entropy-for-machine-learning/.

[5] C. Dwork. A firm foundation for private data analysis. *Communications of the ACM*, 54(1):86–95, 2011.

[6] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor. Our data, ourselves: Privacy via distributed noise generation. In S. Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006*, pages 486–503, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-34547-3.

[7] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*, pages 265–284. Springer, 2006.

[8] C. Dwork, A. Roth, et al. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3-4):211–407, 2014.

[9] S. Elfwing, E. Uchibe, and K. Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107, January 2018. doi: 10.1016/j.neunet.2017.12.012.

[10] T. Erven and P. Harremoës. Rényi divergence and kullback-leibler divergence. *In-*

formation Theory, IEEE Transactions on, 60:3797–3820, July 2014. doi: 10.1109/TIT.2014.2320500.

[11] 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 *23rd USENIX Security Symposium (USENIX Security 14)*, pages 17–32, 2014.

[12] M. Fredrikson, S. Jha, and T. Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. pages 1322–1333, October 2015. doi: 10.1145/2810103.2813677.

[13] E. Galinkin. The influence of dropout on membership inference in differentially private models. *arXiv preprint arXiv:2103.09008*, 2021.

[14] J. Geiping, H. Bauermeister, H. Dröge, and M. Moeller. Inverting gradients – how easy is it to break privacy in federated learning?, March 2020.

[15] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[16] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[17] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[18] D. O. Hebb. *The organization of behavior: A neuropsychological theory*. Wiley, New York, 1949. ISBN 0-8058-4300-0.

[19] S. Himanshu. Activation functions: Sigmoid, tanh, relu, leaky relu, prelu, elu, threshold relu and softmax basics for neural networks and deep learning, 2019. URL https://himanshuxd.medium.com/activation-functions-sigmoid-relu-leaky-relu-and-softmax-basics-for-neural-networks-and-deep-8d9c70eed91e.

[20] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[21] K. Hornik. Approximation capabilities of multilayer feedforward network. *Neural Networks*, 4:251–257, January 1991. doi: 10.1016/0893-6080(91)90009-T.

[22] B. Hui, Y. Yang, H. Yuan, P. Burlina, N. Gong, and Y. Cao. Practical blind membership inference attack via differential comparisons, January 2021.

[23] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, December 2014.

[24] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[26] S. Kullback and R. A. Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[27] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989. doi: 10.1162/neco.1989.1.4.541.

[28] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[29] J. Lim and C. S. Chan. From gradient leakage to adversarial attacks in federated learning. pages 3602–3606, September 2021. doi: 10.1109/ICIP42928.2021.9506589.

[30] F. McSherry. Privacy integrated queries: An extensible platform for privacy-preserving data analysis. *Commun. ACM*, 53:89–97, September 2010. doi: 10.1145/1559845.1559850.

[31] I. Mironov. Rényi differential privacy. pages 263–275, August 2017. doi: 10.1109/CSF.2017.11.

[32] S. Oh and P. Viswanath. The composition theorem for differential privacy. *IEEE Transactions on Information Theory*, PP, November 2013. doi: 10.1109/TIT.2017.2685505.

[33] C. Radebaugh and U. Erlingsson. Introducing tensorflow privacy: Learning with differential privacy for training data, Mar 2019. URL https://blog.tensorflow.org/2019/03/introducing-tensorflow-privacy-learning.html.

[34] M. Rigaki and S. García. A survey of privacy attacks in machine learning, July 2020.

[35] F. Rosenblatt. *Principles of neurodynamics; perceptrons and the theory of brain mechanisms*. Spartan Books, 1962.

[36] A. Salem, Y. Zhang, M. Humbert, P. Berrang, M. Fritz, and M. Backes. Ml-leaks: Model and data independent membership inference attacks and defenses on machine learning models. January 2019. doi: 10.14722/ndss.2019.23119.

[37] R. Shokri, M. Stronati, C. Song, and V. Shmatikov. Membership inference attacks against machine learning models. May 2017. doi: 10.1109/SP.2017.41.

[38] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, June 2014.

[39] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[40] Z. Yang, J. Zhang, E.-C. Chang, and Z. Liang. Neural network inversion in adversarial setting via background knowledge alignment. CCS '19, page 225–240, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450367479. doi: 10.1145/3319535.3354261.

[41] H. Yin, A. Mallya, A. Vahdat, J. Alvarez, J. Kautz, and P. Molchanov. See through gradients: Image batch recovery via gradinversion. pages 16332–16341, June 2021. doi: 10.1109/CVPR46437.2021.01607.

[42] Y. Zhang, R. Jia, H. Pei, W. Wang, B. Li, and D. Song. The secret revealer: Generative model-inversion attacks against deep neural networks. pages 250–258, June 2020. doi: 10.1109/CVPR42600.2020.00033.

[43] X. Zhao, W. Zhang, X. Xiao, and B. Lim. Exploiting explanations for model inversion attacks, April 2021.

A | Appendix A

A.1. Evaluation of model inversion attack with Kullback–Leibler divergence

The Kullback–Leibler divergence (or KLD in short) [26] is a statistical measure that indicates how much one probability distribution differs from a second reference probability distribution.

Given two discrete probability distributions P and Q defined on the same probability space X, the KLD from Q to P is calculated as

$$D_{KL}(P \parallel Q) = -\sum_{x \in X} P(x) \log \left(\frac{Q(x)}{P(x)} \right). \tag{A.1}$$

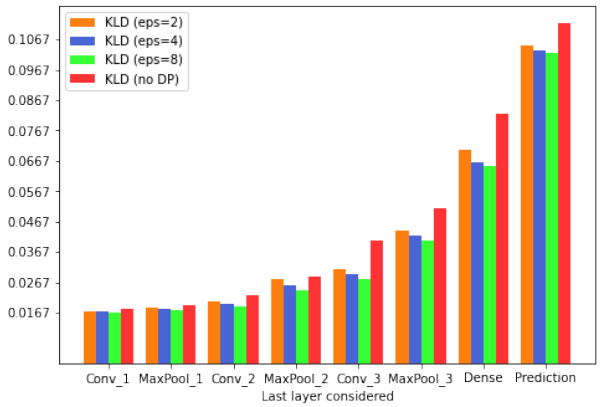
The Kullback–Leibler divergence is also used in machine learning as a function that measures the distance between the probability distribution predicted by a model and the true distribution of the input data. It can be used standalone or as a loss function during the model's training.

In our work, we made a standalone usage of KLD as an accessory metric to evaluate the reconstruction loss of our model inversion attack. In particular, we measured the KLD between the image reconstructed by the adversary and the original ones. To implement this calculation in our code, we used the ***losses.KLDivergence()*** function provided by Keras library.

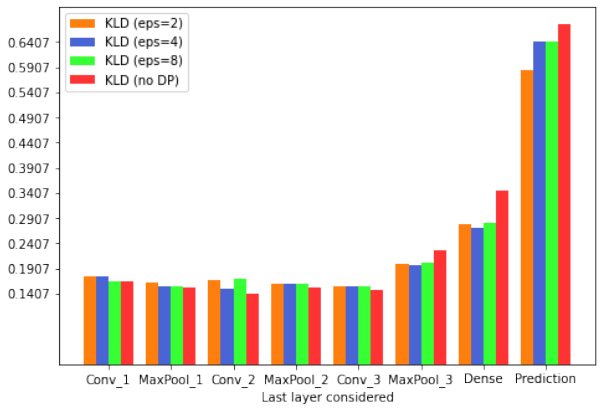
Before performing the computation of the KLD, we needed to flatten both the reconstructed image and the original one and then transform the resulting vectors of values into a probability distribution. Then, we passed the two distributions to the aforementioned function and obtained the corresponding KLD score. In the case of RGB images, we performed this operation for each channel, and the final score was the average of the three resulting values.

For showing the results of the inversion attack in terms of KLD, we performed the same comparative study made for the reconstruction mean squared error. The mean squared

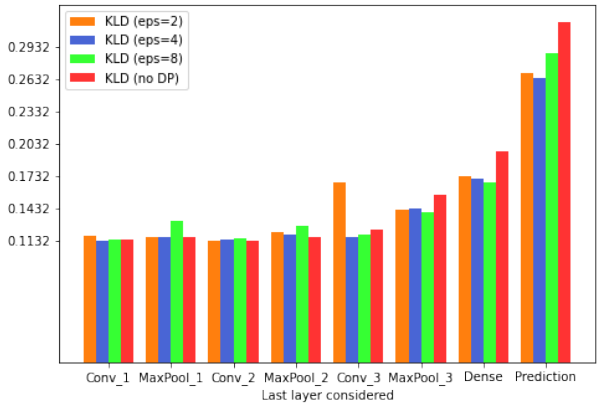
error was calculated by evaluating the model on the entire test dataset, so to have a unique KLD score for each attack, we made an average of all the KLD scores of images in the test dataset.
We performed the experiments with KLD as the metric for the reconstruction loss in three different groups, in the same way as it has been shown in Chapter 5. We will show the results obtained in the study of differential privacy (see Figure A.1), in the study of regularizers (see Figure A.2), and in the final scenario (see Figure A.3).
As expected, the results obtained show a similar trend to the ones of the mean squared error seen in previous experiments; the behavior of the inversion attack in different scenarios is so confirmed even when we use another metric for the reconstruction loss.



(a) CIFAR-10

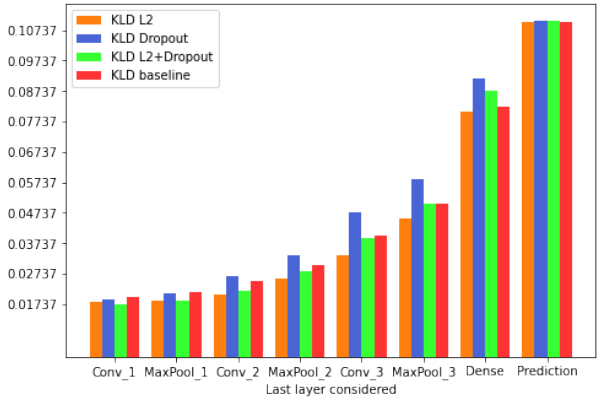


(b) MNIST

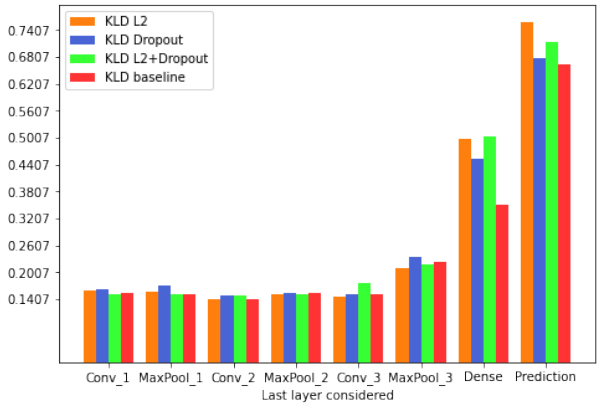


(c) Fashion-MNIST

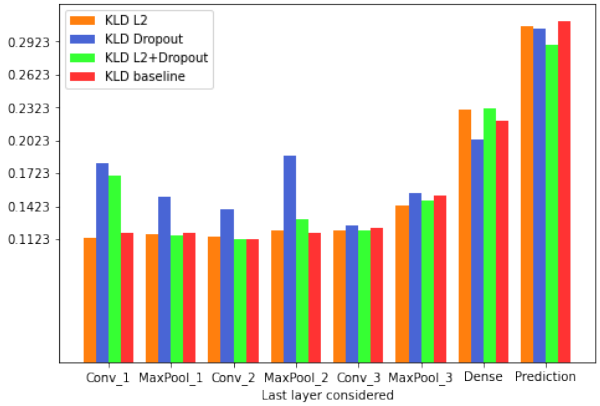
Figure A.1: Comparisons between reconstruction KLDs obtained attacking the four models (baseline and differentially private ones) layer per layer.



(a) CIFAR-10

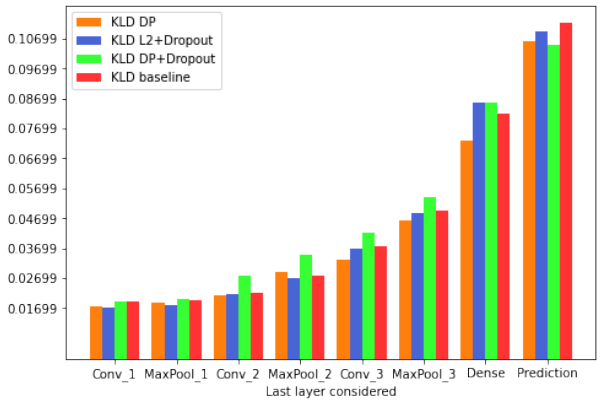


(b) MNIST

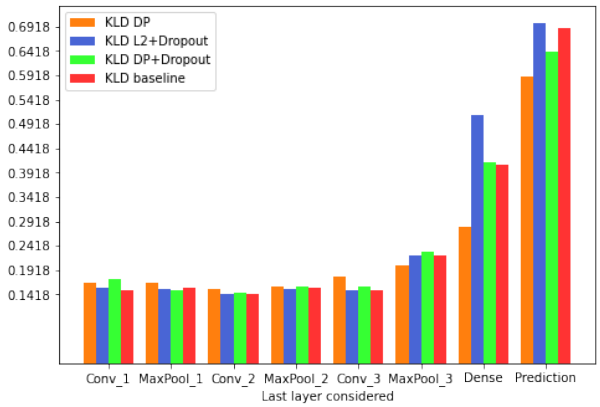


(c) Fashion-MNIST

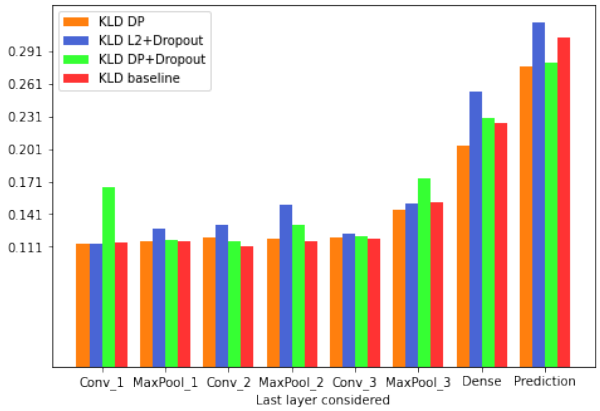
Figure A.2: Comparisons between reconstruction KLDs obtained attacking the four models (baseline and regularized ones) layer per layer.



(a) CIFAR-10



(b) MNIST



(c) Fashion-MNIST

Figure A.3: Comparisons between reconstruction KLDs obtained attacking the four models (baseline and best cases ones) layer per layer.

Acknowledgements

Desidero ringraziare inanzitutto il professor Matteucci per avermi dato l'occasione di potermi cimentare in un ambito di ricerca che ha sempre suscitato in me interesse durante il mio percorso universitario, e per essersi dimostrato una splendida persona durante questo periodo di lavoro insieme.
Ringrazio particolarmente Eugenio, che mi ha accompagnato con costanza e dedizione durante questo percorso e mi ha fornito tutta la sua conoscenza ed esperienza per completare al meglio il mio lavoro.
Ci tengo a ringraziare tutti gli amici di Palermo, in particolare Giovanni, Alberto, Federica, Andrea, Irene, Walter, Antonio e Giuseppe, che mi hanno sempre sostenuto nei momenti di bisogno con allegria e consigli sinceri, nonostante la distanza.
Ringrazio Tiziano, Stefano, Simone, Benedetta, Alessandro, Lorenzo e tutti coloro che hanno rallegrato e reso ancora più godibile la mia esperienza universitaria.
Desidero ringraziare particolarmente Francesco G., Matilde e Francesco D., persone fantastiche che mi hanno mostrato una grande vicinanza in questi anni e con cui ho vissuto bellissime esperienze.
Un grande ringraziamento anche ai miei coinquilini Chiara, Cristina, Michela, Sara e Riccardo per il supporto offertomi durante la mia permanenza a Milano e per avere reso piacevole la complessa esperienza da fuorisede.
I miei ringraziamenti vanno inoltre ad Antonella, Ugo, Gabriella, Fabrizio, Alice e Rebecca, per essere stati sempre partecipi dei miei progressi in questo percorso iniziato cinque anni fa.
Infine, desidero dare i miei più sentiti ringraziamenti a tutta la mia famiglia, che tra Palermo e Milano non mi ha mai fatto mancare il proprio appoggio, accompagnandomi costantemente in una esperienza nuova e difficile e permettendomi di compiere il mio percorso in maniera autonoma, supportandomi nelle decisioni importanti, spronandomi ed offrendomi aiuto nei momenti di necessità.