A privacy-preserving distributed architecture for deep-learning-as-a-service

Advisor: Prof. Manuel Roveri
Co-Advisor: Simone Disabato
Alessio Mongelluzzo

Thesis by:
Alessandro Falcetta Matr. 919836

Academic Year 2019–2020
Abstract

Machine learning is one of the most interesting and promising techniques of recent years, thanks to its ability to solve problems in many different domains. In particular, Machine-learning-as-a-service is a novel and promising computing paradigm which can provide various kinds of machine learning solutions leveraging a Cloud-based computing infrastructure. Thanks to the main characteristics of such infrastructures (high performance, scalability, availability) complex machine learning solutions can be offered directly to customers through Internet. Examples of them include but are not restricted to image recognition and sentiment analysis. Often these solutions rely on the final user sending data (e.g., signals, images, positions, sounds, videos) to the Cloud, hence posing a series of ethical and legal problems on the management of the data. The purpose of the thesis is to present a privacy-by-design architecture able to perform machine learning algorithms on user-provided data. Such architecture will be able to pursue the goals of a classic Machine-learning-as-a-service paradigm, meanwhile preventing the service provider from exploiting the provided data; this will be achieved using Homomorphic Encryption (HE). The architecture has been implemented with a client-server REST-based application where the server is able to perform machine learning algorithms on data sent by the client, represented by the user’s machine. The experimental results are conducted using Convolutional Neural Networks (CNNs) to demonstrate the effectiveness of the proposed architecture. All in all, the results show that machine learning based services can be provided to users in a privacy respectful manner. The solution proposed in this thesis has been published at “2020 International Joint Conference on Neural Networks (IJCNN), Glasgow, 2020”.
La tecnica del Machine Learning è una delle più interessanti e promettenti degli ultimi anni, grazie alla sua abilità di risolvere problemi appartenenti a molti domini diversi. In particolare, il Machine-learning-as-a-service è un nuovo e promettente paradigma il quale può fornire soluzioni di machine learning di vario tipo sfruttando una architettura di tipo Cloud. Grazie alle principali caratteristiche di tali infrastrutture (alte performance, scalabilità, disponibilità), soluzioni di machine learning complesse possono essere fornite direttamente agli utenti finali tramite Internet. Tali soluzioni includono il riconoscimento di immagini e il sentiment analysis, insieme a molte altre. Tuttavia, spesso, queste soluzioni utilizzano dati inviati dagli utenti (es. immagini, coordinate geografiche, video) al Cloud, dando origine a problematiche etiche e legali sulla gestione di tali dati. Lo scopo di questa tesi è presentare una architettura privacy-by-design abile ad eseguire algoritmi di machine learning su dati inviati dagli utenti. Tale architettura ricalcherà quella di un paradigma machine-learning-as-a-service classico, impedendo al contempo al provider del servizio di sfruttare i dati forniti dall’utente; questo sarà ottenuto grazie alla Crittografia Omomorfica. L’architettura è stata implementata con una applicazione client-server REST, dove il server può eseguire algoritmi di machine learning su dati inviati dal client, collocato sulla macchina dell’utente. I risultati sperimentali sono ottenuti usando Reti Neurali Convoluzionali per dimostrare l’efficacia della soluzione. Questi mostrano che è possibile fornire servizi usando algoritmi di machine learning mantenendo al contempo la privacy degli utenti. La soluzione proposta in questa tesi è stata pubblicata nell’ambito della conferenza ”2020 International Joint Conference on Neural Networks (IJCNN), Glasgow, 2020”.

Sommaio
List of Figures

2.1 Simple comparison between homomorphic encryption and common encryption schemes. 8

3.1 Secure Multi-Party computation [1]. 35

4.1 The proposed privacy-preserving architecture for deep-learning-as-a-service. 40

4.2 Example of a plain mode with the corresponding approximated model. 45

4.3 A comparison of the plain and approximated CNN processing with the encrypted one. The layers parameters $\theta_l$ are omitted to simplify the notation. 47

4.4 Illustration of the recall modality. The user transmits an encrypted image to the server which returns the encrypted final classification of the image. It is then decrypted on the user machine. 48

4.5 Illustration of the transfer learning modality. The user transmits an encrypted image to the server which returns the encrypted extracted features. They are then used on the client machine to obtain the final classification, using a local classifier. 49

6.1 The recall accuracy results of both the 6-layer CNN and the 5-layer CNN on the FashionMNIST dataset [2], with the standard deviation over five experiments. 65

6.2 The transfer learning accuracy results on the features extracted at layer $l = 4$ of the 6-layer CNN to the MNIST dataset [3]. 69
VI

LIST OF FIGURES


## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Code coverage obtained using the library coverage.py.</td>
<td>59</td>
</tr>
<tr>
<td>6.1</td>
<td>The results, in terms of accuracy, of the 6-layers CNN model in recall modality</td>
<td>67</td>
</tr>
<tr>
<td>6.2</td>
<td>The results, in terms of accuracy, of the 5-layers CNN model in recall modality</td>
<td>68</td>
</tr>
<tr>
<td>6.3</td>
<td>The results, in terms of accuracy, of the transfer learning modality using a local SVM-based classifier</td>
<td>70</td>
</tr>
<tr>
<td>6.4</td>
<td>The results, in terms of accuracy, of the transfer learning modality using a local fully-connected based classifier</td>
<td>71</td>
</tr>
<tr>
<td>6.5</td>
<td>The three described configurations results, with a common PC as client and an Amazon EC2 instance as server. The main result, ( t ), is the time requested to process an image for each scenario. The three components of ( t ) are ( t_c ), the time requested for the local encryption/decryption, ( t_t ), the time for the data transfer and ( t_a ), the time requested for the processing on the Cloud. The proposed values are expressed in seconds.</td>
<td>73</td>
</tr>
<tr>
<td>6.6</td>
<td>The ranking of the best combination of models/encryption parameters on the FashionMNIST dataset, including also the timing (expressed in seconds).</td>
<td>74</td>
</tr>
</tbody>
</table>
# Contents

1 Introduction .............................................................. 1
   1.1 Problem and Motivation ........................................ 1
      1.1.1 Privacy .................................................. 2
   1.2 Goal and Results ............................................... 3
   1.3 Thesis Structure ............................................... 4

2 Background ................................................................... 7
   2.1 Homomorphic Encryption ........................................ 7
      2.1.1 Classification of HE schemes ........................... 8
      2.1.2 Brakersi/Fan-Vercauteren scheme (BFV) ............. 9
      2.1.3 Practical considerations on the use of HE .......... 19
   2.2 Machine Learning ............................................... 21
      2.2.1 Deep Learning ............................................ 22
      2.2.2 Convolutional Neural Networks ......................... 22
   2.3 Cloud ............................................................. 25
      2.3.1 Cloud provider ........................................... 26
      2.3.2 Advantages of Cloud solutions ......................... 28
      2.3.3 Drawbacks of Cloud solutions ......................... 29

3 Related Literature ....................................................... 31
   3.1 First HE schemes .............................................. 32
   3.2 Use of HE ...................................................... 33
   3.3 Secure Multi-Party Computation .............................. 35

4 Architecture of the proposed solution ............................. 39
   4.1 Encryption and Decryption ..................................... 41
   4.2 Approximated and encoded DL processing .................. 42
   4.3 Offered DL modalities ......................................... 46
## CONTENTS

4.3.1 Recall ................................................. 48
4.3.2 Transfer Learning ................................. 49
4.4 Encryption parameters ......................... 50
4.5 Communication between User Device and Cloud . . . 52

5 Implementation of the proposed solution 53
  5.1 External libraries used ............................... 54
  5.2 Structure ...................................... 54
  5.3 Compatible models ................................. 56
  5.4 Client .......................................... 56
  5.5 Server .......................................... 58
  5.6 Testing .......................................... 58

6 Experimental results 61
  6.1 Description of the CNNs ......................... 61
  6.2 Datasets ........................................ 63
    6.2.1 MNIST dataset .............................. 63
    6.2.2 FashionMNIST dataset ..................... 63
  6.3 Recall ........................................ 64
  6.4 Transfer learning ................................. 67
  6.5 Timing ........................................ 71

7 Conclusions 77
  7.1 Conclusions ................................... 77
  7.2 Future Works .................................. 78

A IJCNN Paper 89
Chapter 1

Introduction

1.1 Problem and Motivation

Machine Learning (ML) is a field of artificial intelligence involved in the study of algorithms able to give computers the ability to “learn”; in other words, able to improve their capability on a specific task without explicit instructions [4].

The peculiarity of those algorithms is to be agnostic from any specific domain in which they are used. In fact, machine learning can be applied to many different kinds of problems. This made the interest about this paradigm arose. After being a pure mathematical tool, ML has been introduced in many industries becoming a practical tool to solve issues that, to that moment, were impossible or demanded an high effort from human operators.

The development of ML is related to the one of Cloud computing. While there exist many definitions for this term, Cloud computing can be considered as the possibility for a user to use computational, network and storage resources offered by a coherent, large-scale network of machines. These machines are managed by a provider, which offers the service on a pay-per-use basis. Often the resources are offered through a Web service, with which the user can start computations, use a service to its own users etc. The nature of Cloud computing environments favours features like scalability, availability, maintainability [5].

The point of encounter of these two technologies is *machine-learning-as-a-service* [6] (MLaaS). According to this paradigm, Cloud providers
offer ready-to-use remotely-executable machine learning services. There are at least two explanations for the success of this modality.

The first is that ML solutions are often complex and require intensive resources. Moreover, the setup of ML environments can severely affect the performance in terms of time and space required by the algorithms. The possibility to directly access state-of-art ML architectures is captivating for industries which see ML as a convenient technology to solve different problems.

The second one is that there is a range of scenarios for which already perfected ML solutions exist. These are the ready-to-use solutions discussed above: ML models which do not need to be trained (being pre-trained by the Cloud provider). This is an attractive point for users.

Examples of MLaaS are the identification of faces/objects in images or video, or the conversion of text-to-speech/speech-to-text [7].

1.1.1 Privacy

One of the main drawbacks of Cloud solutions (including MLaaS) is the privacy of the transmitted data. MLaaS, to be effective, usually require users to upload some sensible data to retrieve information. To identify faces in an image, the image has to be uploaded in the first place. Often, the attractiveness of this kind of services causes users to underestimate the consequences of an unfair use of their data. Indeed, many MLaaS need personal data to work: personal pictures or video, medical diagnoses, as well as data that might reveal ethnic origin, political opinions, but also genetic, biometric and health data [8]. One of the many definitions of privacy is the one proposed by Lane et al. [9], which states that privacy is the requirement that information flows in the appropriate way, according to some informational norms. These norms state which are the key actors of every flow: recipients, senders and information subjects; which are types of information involved, and the constraints under which such flows occur. In other words, privacy is respected when information is treated according to some definite and precise bounds. When these bounds are violated, the information subject may undergo consequences whose gravity depends
on the magnitude of the violation. Resuming the previous example, if the user-provided pictures used by a MLaaS to provide a service fall into the wrong hands, a malicious actor may use them to perform various kinds of social engineering attacks.

1.2 Goal and Results

This work proposes an approach to the problem of privacy in a Cloud scenario, with the focus on MLaaS. In particular, a privacy-preserving architecture which relies on Homomorphic Encryption (HE) to guarantee the confidentiality of the elaborated data. While the architecture can be tailored to multiple uses, the study is centered on Convolutional Neural Networks, a class of ML algorithms often used to classify images. In the proposed architecture, by exploiting the properties of HE, the data is encrypted on the user machine (e.g., a personal computer or a mobile device) through a public key. The encrypted data is sent to a Cloud-based special MLaaS, amended to work on encrypted data. After the required computation is finished, the result is sent back to the user machine, where it will be decrypted and made available. This involves that plain data never leaves the device of the user; the MLaaS provider will only be able to work on encrypted data, whose content can not be accessed. The benefits of the Cloud are still guaranteed.

The ability to process encrypted data of HE comes at two main drawbacks. The first involves the load and memory demand of HE-encoded operations. With respect to plain computations, HE introduces an important overhead which explains why a Cloud environment is needed to carry out the most heavy computations planned by the architecture. The second one concerns the limitations of HE. The majority of HE schemes (including the one used for this work) allows only a limited set of operations to be carried out on encrypted data, i.e. additions and multiplications. This unavoidably restrict the possible uses of HE; however, for the case of CNNs, additions and multiplications are sufficient to get a good result. The models will have to be redesigned and retrained, taking into account this fact. Moreover, the use of HE requires a special attention because the correctness, com-
putational load and precision of HE operations depends on a set of parameters that have to be tuned to obtain a trade-off between all these characteristics. Nonetheless, the proposed architecture provides ways to hide this complexity, letting the service provider take care of such parameters.

This work includes an implementation of the proposed architecture. It includes a Python library which consists of a client (locally executed on the user device) and a server, in the form of a MLaaS container implemented on Amazon AWS (a standalone Python application, usable on common systems, is also provided). The developed implementation relies on a Representational state transfer (REST) paradigm which makes it possible to exchange encrypted data and results between client and server. Messages are encoded using JSON format. In particular, the library is focused on the use of CNNs for image recognition. The code is made available to the scientific community\[1\].

Lastly, a wide experimental campaign shows the feasibility of the architecture and gives exhaustive statistics about the performances of the encrypted MLaaS, both in terms of accuracy and in terms of computational resources.

The solution proposed in this thesis has been used for a paper which has been published at “2020 International Joint Conference on Neural Networks (IJCNN), Glasgow, 2020”. The full text of the paper is attached in Appendix A.

1.3 Thesis Structure

Chapter 2 introduces the fundamental notions about HE, its limitations and possibilities, the advantages of its adoption and the necessary steps to use it. In particular, the peculiarity of the chosen HE scheme are presented, with guidelines to select the optimal parameters. Moreover, a basic background on ML and CNNs is exposed, along with an introduction the concept of Cloud. These are concepts needed to fully understand the characteristics of the architecture. Chapter 3

\[1\]Code is available for download as a public repository at https://github.com/AlexMV12/PyCrCNN.git
describes the current state of art of privacy-preserving solutions for MLaaS. Chapter 4 is the core of the work, in which the proposed architecture is shown in detail. Chapter 5 explores PyCrCNN, the Python Library which implements the architecture. Chapter 6 includes the experiments conducted on some typical use-cases in the image recognition field. Conclusions are finally drawn in Chapter 7.
Chapter 2

Background

This Chapter collects the main notions required to fully understand the various aspects of this work. More specifically, Section 2.1 presents the encryption scheme used in this work, along with considerations on advantages and drawbacks of the proposed solutions. This Section includes considerations on the security assumptions, the permitted operations and the computational overhead. In Section 2.2 a brief summary on Machine Learning will be presented, along with more specific notions regarding Convolutional Neural Networks in Subsection 2.2.2. Lastly, Section 2.3 gives an overview on the current state-of-art Cloud solutions and the considerations related to the use of Cloud in this specific work.

2.1 Homomorphic Encryption

Homomorphic encryption is a special type of encryption that allows (a set of) operations to be performed on encrypted data, i.e., directly on the ciphertexts. More specifically, as detailed in [10], an encryption function $E$ and its decryption function $D$ are homomorphic with respect to a class of functions $\mathcal{F}$ if, for any function $f \in \mathcal{F}$, we can construct a function $g$ such that $f(x) = D(g(E(x)))$ for a set of input $x$. This brief definition contains the main important aspect of this encryption scheme. Common encryption schemes, both symmetrical and asymmetrical, are not homomorphic in the sense that, after encryption, the algebraic structure of the plaintexts is not maintained. Basically,
ciphertexts without the secret key, needed to decrypt them, can not be used in meaningful operations. Adding or multiplying ciphertexts produces an output which, when decrypted, does not correspond to the correct result of the operation, executed in the plaintext space. With homomorphic encryption, it is possible to work on ciphertexts in a meaningful manner (a practical example is shown in Figure 2.1).

![Figure 2.1: Simple comparison between homomorphic encryption and common encryption schemes.](image)

### 2.1.1 Classification of HE schemes

Before going into the details of the scheme used in this work, it is useful to distinguish between the three main forms of HE schemes available.

**Somewhat homomorphic schemes**

These schemes support mathematical operations restricted to the use of additions and multiplications. This means that ciphertexts can only be added or multiplied; other operations have to be approximated to use only these two operations. Moreover, it is not possible to use the ciphertexts in an unbounded number of consecutive operations. After every operation a certain amount of noise is added to the ciphertexts: after a certain limit, it is no longer possible to decrypt them without corrupting the data.
Partially homomorphic schemes

Partially homomorphic schemes are designed to support an unbounded number of consecutive operations on the ciphertexts; however, only one type of operation is allowed. An example of a partially homomorphic scheme is RSA [11]: in particular, RSA is multiplicatively homomorphic. Given two ciphertexts in the RSA scheme, $c_1$ and $c_2$, obtained from the plaintexts $m_1$ and $m_2$, the decryption of $c_3 = c_1 \cdot c_2$ is equivalent to the product of $m_1$ and $m_2$.

Fully homomorphic schemes

An encryption scheme is fully homomorphic if it is homomorphic with respect to additions and multiplications, applied for an unbounded number of times to the ciphertexts.

2.1.2 Brakersi/Fan-Vercauteren scheme (BFV)

The HE scheme considered in this paper is the Brakerski/Fan-Vercauteren (BFV) scheme [12] that, similarly to other works [13], [14], is based on the Ring-Learning With Errors (RLWE) problem. The BFV encryption scheme has been introduced by Junfeng Fan and Frederik Vercauteren in 2012 [12]. Their idea was to modify the scheme proposed by Brakerski [15], porting it from a setting in which the Learning With Error (LWE) problem was used to RLWE.

Polynomial rings

The main object used in the BFV scheme is the polynomial ring. Some preliminary definitions are needed before exploring the details of the scheme.

**Definition 2.1.1.** A ring is a set $R$ with the operations of addition and multiplication. Addition is defined as $+: R \times R \rightarrow R, (a, b) \mapsto a + b$, while multiplication is defined as $\ast: R \times R \rightarrow R, (a, b) \mapsto a \ast b$ [16].

Moreover, a ring satisfies the following conditions:
1. \((R, +)\) is a commutative group, i.e. \(a + b = b + a\). The element \(0 \in R\) is the neutral element in this group. The inverse element of \(a \in R\) is \(-a\).

2. The multiplication is associative, i.e. \((a * b) * c = a * (b * c)\) for all \(a, b, c \in R\).

3. The distributive laws hold, i.e. for all \(a, b, c \in R\):
   \[
a * (b + c) = a * b + a * c, \quad (a + b) * c = a * c + b * c.
   \]

The element \(1 \in R\) is called unit and \(1 * a = a * 1 = a\) for all \(a \in R\). If \(R\) is a ring, the polynomial ring \(R[x]\) is the ring of all polynomials with coefficients in \(R\):

\[
R[x] = \{a_0 + a_1 x + \cdots + a_n x^n : a_i \in R \forall i\}
\]

The addition and multiplications operations are defined as:

\[
\sum_{i=0}^{n} a_i x^i + \sum_{i=0}^{n} b_i x^i = \sum_{i=0}^{n} (a_i + b_i) x^i
\]

\[
\sum_{i=0}^{n} a_i x^i \ast \sum_{j=0}^{m} b_j x^j = \sum_{i=0}^{n} \sum_{j=0}^{m} a_i b_j x^{i+j}
\]

The ring used in the BFV scheme is:

\[
R = \mathbb{Z}[x]/(f(x))
\]

\(f(x) \in \mathbb{Z}[x]\) is a monic irreducible polynomial of degree \(m\). In particular, a cyclotomic polynomial \(\Phi_m(x)\) of degree \(m\) is used; \(m\) is a positive power of 2.

**Notations**

The explanation of the RLWE problem requires the use of a series of notations. Used in [12], they are here reported with the same order used by the authors of the scheme’s introductory paper, for easier reading.
2.1 Homomorphic Encryption

- \( R = \mathbb{Z}[x]/(f(x)) \) is the polynomial ring where \( f(x) \in \mathbb{Z}[x] \) is a monic irreducible polynomial of degree \( m \); in practice a cyclotomic polynomial \( \Phi_m(x) \) is used. For expository purposes, it is convenient to take \( f(x) = x^m + 1 \), with \( m = 2^n \).

- Elements of the ring \( R \) are denoted in lowercase bold, e.g. \( a \in R \). The coefficient of \( a \) are denoted by \( a_i \), i.e. \( a = \sum_{i=0}^{d-1} a_i \cdot x^i \).

- The infinity norm \( \|a\|_\infty \) is defined as \( \max_i |a_i| \): it is simply the largest coefficient of \( a \).

- Let \( q > 1 \) be an integer, then by \( \mathbb{Z}_q \) we denote the set of integers \((-q/2,q/2]\).

- \( R_q \) is the set of polynomials in \( R \) with coefficients in \( \mathbb{Z}_q \).

- For \( a \in \mathbb{Z} \), \( [a]_q \) denotes the unique integer in \( \mathbb{Z}_q \) with \( [a]_q = a \mod q \).

- The reduction of the interval \([0,q)\) will be denoted as \( r_q(a) \), and called remainder modulo \( q \).

- Given a probability distribution \( D \), \( x \leftarrow D \) denotes that \( x \) is sampled from \( D \).

- For a set \( S \), \( x \leftarrow S \) denotes that \( x \) is sampled uniformly from \( S \).

- The discrete Gaussian distribution \( D_{\mathbb{Z},\sigma} \) over the integers is the probability distribution which assigns a probability proportional to \( \exp(-\pi|x|^2/\sigma^2) \) to each \( x \in \mathbb{Z} \). It is then used to define a distribution \( \chi \) on \( R \).

**Ring Learning With Errors problem**

Ring Learning With Errors is a computational problem used in many field of cryptography, including the development of encryption schemes resistant to attacks conducted with quantum computers.

There exist two different versions of this problem: “search” and “decision”. BFV uses the “decision” version.

**Definition 2.1.2. RLWE-Decision** Given:
• $a_i$ is a set of random but known polynomials from $R_q$, which is the
  ring $R$ with coefficients in $\mathbb{Z}_q$;

• $e_i$ is a set of small random and unknown polynomials relative to
  a bound $b$ in the ring $\mathbb{Z}_q$;

• $s$ be a small unknown polynomial relative to a bound $b$ in the
  ring $\mathbb{Z}_q$.

• $b_i = (a_i \cdot s) + e_i$

And a list of polynomial pairs $(a_i, b_i)$, the RLWE-Decision problem
consists in determining if the $b_i$ polynomials were constructed as $b_i =
(a_i \cdot s) + e_i$ or were generated randomly from $R_q$.

The security of a scheme built on the RLWE problem is explored
in the work by Lyubashevsky et al. [17]. The main idea is that solving
the RLWE problem is equivalent to solve the Approximate Shortest
Vector Problem ($\alpha$-SVP). This problem is known to be NP-hard [18].

**Encryption Scheme**

[17] describes the encryption scheme which has been used for the base
of the BFV scheme [12]. $R_t$ is the plaintext space, for some $t > 1$.
Consider $\Delta = \lfloor q/t \rfloor$, and $r_t(q) = q \mod t$. Thus, it holds:

$$q = \Delta \cdot t + r_t(q)$$

As reported by the authors, $q$ and $t$ have not to be prime nor co-prime.
The basic operations of the scheme are:

• **SecretKeyGen** ($1^\lambda$): sample $s \leftarrow \chi$ and output $sk = s$.

• **PublicKeyGen** (sk): set $s = sk$, sample $a \leftarrow R_q$, $e \leftarrow \chi$ and output
  $$pk = \left[ -(a \cdot s + e) \right]_q, a$$

• **EvaluateKeyGen** (sk, $p$): sample $a \leftarrow R_{p \cdot q}$, $e \leftarrow \chi'$ and return
  $$rlk = \left[ -(a \cdot s + e) + p \cdot s^2 \right]_{p \cdot q}, a$$

Evaluation keys are necessary to perform an operation called
relinearization, described more in detail in Sub-Section [2.1.2]
2.1 Homomorphic Encryption

• Encrypt \((pk, m)\): to encrypt a message \(m \in R_t\), let:

\[
p_0 = pk[0] \\
p_1 = pk[1]
\]

Then, sample \(u, e_1, e_2 \leftarrow \chi\) and return:

\[
ct = ([p_0 \cdot u + e_1 + \Delta \cdot m]_q, [p_1 \cdot u + e_2]_q)
\]

• Decrypt \((sk, ct)\): set \(s = sk, c_0 = ct[0], c_1 = ct[1]\). The decrypted value is:

\[
\left\lceil \frac{t \cdot [c_0 + c_1 \cdot s]_q}{q} \right\rceil_t
\]

• Add \((ct_0, ct_1)\): Output \((ct_0[0] + ct_1[0], ct_0[1] + ct_1[1])\)

• Multiply \((ct_0, ct_1, rlk)\): compute

\[
c_0 = \left\lceil \frac{t \cdot (ct_1[0] \cdot ct_2[0])}{q} \right\rceil_q \\
c_1 = \left\lceil \frac{t \cdot (ct_1[0] \cdot ct_2[1] + ct_1[1] \cdot ct_2[0])}{q} \right\rceil_q \\
c_2 = \left\lceil \frac{t \cdot (ct_1[1] \cdot ct_2[1])}{q} \right\rceil_q
\]

These are the basic operations employed in the BFV scheme. If the hardness of the RLWE problem is assumed, this scheme can be shown to be semantically secure \[17\].

The aforementioned operation of relinearization is related to noise and noise growth in the ciphertexts. These concepts are fundamental from a practical point of view for the use of the BFV scheme.

Noise

**Definition 2.1.3. Invariant noise** Let \(ct = (c_0, c_1, \ldots, c_k)\) be a ciphertext encrypting the message \(m \in R_t\). Its invariant noise \(v\) is the polynomial with the smallest infinity norm such that:

\[
\frac{t}{q} \text{ct}(s) = \frac{t}{q} (c_0 + c_1s + \cdots + c ks^k) = m + v + at
\]

for some polynomial \(a\) with integer coefficients \[19\].
During the encryption phase, noise is added to the ciphertexts to guarantee that, being \( p_1 = p_2 \) two plain values to be encrypted with the same public key, the corresponding ciphertexts \( c_1 \) and \( c_2 \) are different (i.e., \( c_1 \neq c_2 \)). However, performing operations on ciphertexts increases their noise. When the noise reaches a certain threshold, it becomes impossible to decrypt the data without corrupting it.

The Noise budget is a practical way to measure the noise of a ciphertext.

**Definition 2.1.4. Noise budget** Let \( v \) be the invariant noise of a ciphertext \( ct \) encrypting the message \( m \in R_t \). Then the noise budget (NB) of \( ct \) is \(-\log_2(2||v||)\).

The NB is an integer positive number. It if is greater than 0, it still possible to decrypt a ciphertext and obtain the correct plaintext value. Hence, for NB, greater is better. As stated in [19]:

**Lemma 2.1.1.** The function Decrypt correctly decrypts a ciphertext \( ct \) encrypting a message \( m \), as long as the NB of \( ct \) is positive.

It is useful to understand how noise grows when an operation is performed on a ciphertext. In general, the important aspect is that additions and subtractions have a small impact on the noise growth, while multiplications are the most NB-consuming operations. The precise growth of the noise, for each operations, is here reported for completeness.

First, a fresh encrypted value has a positive NB value, as stated in [19]:

**Lemma 2.1.2.** Let \( ct = (c_0, c_1) \) be a fresh encryption of a message \( m \in R_t \). The noise \( v \) in \( ct \) satisfies:

\[
||v|| \leq \frac{r_t(q)}{q} ||m|| + \frac{tB}{q}(2n + 1)
\]

Additions between two ciphertexts cause the noise to be the sum of the noise value of the two ciphertexts [19].

**Lemma 2.1.3.** Let \( ct_1 = (c_0, c_1, \ldots, c_j) \) and \( ct_2 = (d_0, d_1, \ldots, d_k) \) be two ciphertexts encrypting \( m_1, m_2 \in R_t \) and having noises \( v_1, v_2 \),
respectively. The noise \( v_{\text{add}} \) in their sum \( ct_{\text{add}} \) is \( v_{\text{add}} = v_1 + v_2 \) and satisfies:

\[
||v_{\text{add}}|| \leq ||v_1|| + ||v_2||
\]

Multiplications not only make the noise grow with a greater factor, but also depends on the size of the two ciphertexts [19]:

**Lemma 2.1.4.** Let \( ct_1 = (x_0, \cdots, x_{j_1}) \) be a ciphertext of size \( j_1 + 1 \) encrypting a \( m_1 \) with noise \( v_1 \), and let \( ct_2 = (y_0, \cdots, y_{j_2}) \) be a ciphertext of size \( j_2 + 1 \) encrypting \( m_2 \) with noise \( v_2 \). \( N_{m_1} \) and \( N_{m_2} \) are upper bounds on the number of non-zero terms in the polynomials \( m_1 \) and \( m_2 \), respectively. The noise \( v_{\text{mult}} \) in the product \( ct_{\text{mult}} \) satisfies:

\[
||v_{\text{mult}}|| \leq \left[ (N_{m_1} + n)||m_1|| + \frac{nt}{2} \cdot \frac{n^{j_1+1} - 1}{n-1} \right] ||v_2||
+ \left[ (N_{m_2} + n)||m_2|| + \frac{nt}{2} \cdot \frac{n^{j_2+1} - 1}{n-1} \right] ||v_1||
+ 3n||v_1|| ||v_2||
+ \frac{t}{2q} \left( \frac{n^{j_1+j_2+1} - 1}{n-1} \right)
\]

It is very important to reduce the size of the ciphertexts in order to limit the noise growth during multiplications. To achieve this goal, the operation of relinearization is needed. As anticipated in Section 2.1.2, evaluation keys are needed to perform this operation.

With reference to the Multiply operation (Sub-Section 2.1.2), relinearization [12] consists in writing \( c_2 \) in base \( T \), i.e. write \( c_2 = \sum_{i=0}^l c_2^{(i)} T^i \) with \( c_2^{(i)} \in \mathbb{R}_T \) and set:

\[
c'_0 = \left[ c_0 + \sum_{i=0}^l \text{rlk}[i][0] \cdot c_2^{(i)} \right]_q
\]

\[
c'_1 = \left[ c_1 + \sum_{i=0}^l \text{rlk}[i][1] \cdot c_2^{(i)} \right]_q
\]

Then, return \((c'_0, c'_1)\).

The relinearization operation consumes noise [19].
Lemma 2.1.5. Let $ct$ be a ciphertext of size $M+1$ encrypting $m$, and having noise $v$. Let $ct_{\text{relin}}$ of size $N+1$ be the ciphertext encrypting $m$, obtained by the relinearization of $ct$, where $2 \leq N + 1 \leq M + 1$. The noise $v_{\text{relin}}$ in $ct_{\text{relin}}$ is given by

$$v_{\text{relin}} = v - \frac{t}{q} \sum_{j=0}^{M-N-1} M - 1 \sum_{i=0}^{l} e_{M-j}, C_{i}^{(i)}.$$ 

\textbf{Plain operations}

The term plain operation refers to an operation performed in the ciphertext space between a ciphertext and a plaintext. This type of operations is frequent, because many times a ciphertext has to be used in a function with other values which are not encrypted. For example, in this work the majority of the performed operations involve an encrypted value and a plain value, already available and known. Plain operations are less computational consuming with respect to operations between ciphertexts; moreover, they consume much less noise. As stated in [10], the noise of the result of a plain addition is:

Lemma 2.1.6. Let $ct = (x_0, \ldots, x_j)$ be a ciphertext encrypting $m_1$ with noise $v$ and let $m_2$ be a plaintext polynomial. Let $ct_{\text{padd}}$ denote the ciphertext obtained by plain addition of $ct$ with $m_2$. Then the noise of $ct_{\text{padd}}$ is $v_{\text{padd}} = v - \frac{r_1(q)}{q} m_2$, with the bound:

$$||v_{\text{padd}}|| \leq ||v|| + \frac{r_1(q)}{q} ||m_2||$$

While for plain multiplication:

Lemma 2.1.7. Let $ct = (x_0, \ldots, x_j)$ be a ciphertext encrypting $m_1$ with noise $v$ and let $m_2$ be a plaintext polynomial. Let $N_{m_2}$ be an upper bound on the number of non-zero terms in the polynomial $m_2$. Let $ct_{\text{pmult}}$ denote the ciphertext obtained by plain multiplication of $ct$ with $m_2$. Then the noise of $ct_{\text{pmult}}$ is $v_{\text{pmult}} = m_2v$, with the bound:

$$||v_{\text{pmult}}|| \leq N_{m_2} ||m_2|| ||v||$$

However, plain values have to be encoded before they can be used in operations with other ciphertexts.
Encoding

In the most cases, data processing involves numbers, be them integer or real. In order to be able to fully leverage the potential of HE, a way to encode numbers into polynomial is necessary. BFV provides operations to carry out this encoding. It should be considered that every number must be first encoded into a plaintext polynomial in \( R_t \), and can only be encrypted after that \([19]\). Obviously, an operation of decoding is needed in order to obtain the number corresponding to a (possibly previously decrypted) plain polynomial. The problem is not trivial. It is clear that the rings \( \mathbb{Z} \) and \( R_t \) are very different: for example the set of integers is infinite, whereas \( R_t \) is not. This is the main reason for which HE applications must be carefully designed: in particular, the evaluating party (the party which will have to perform the computation on encrypted data) should find appropriate parameters to guarantee a correct encoding of the involved values.

Integer encoder

The exist different integer encoders, one for each base. The base is denoted by \( B \geq 2 \). Consider the case with \( B = 2 \). A possible way \([19]\) to encode an integer in the range \(-2^n - 1 \leq a \leq 2^n - 1\) is to form the n-bit binary expansion of \(|a|\), say \( a_n, \ldots, a_1 a_0 \). The binary encoding of \( a \), then, is:

\[
\text{IntegerEncode}(a, B = 2) = \text{sign}(a) \cdot (a_{n-1}x^{n-1} + \cdots + a_1 x + a_0)
\]

If \( B > 2 \), instead of a binary expansion, a base-\( B \) expansion is used. The coefficients are chosen from the symmetric set \([-(B-1)/2, \ldots, (B-1)/2]\), given that there is a unique representation with at most \( n \) coefficients for each integer in \([-(B^n-1)/2, (B^n-1)/2]\). Usually, it is used \( B = 2 \) or \( B = 3 \).

Decoding is simple as evaluating the plaintext polynomial at \( x = B \).

It is important to highlight that the modulo operations performed during the computations may make it impossible to decode the polynomials. The first case is the modulo \( x^n + 1 \). If there is at least one degree in the polynomial which is greater than \( n \), the decoding will silently fail. The second case is related to the coefficients of the polynomials. During computation the modulo \( t \) operation is performed. If
there is at least one coefficient greater than \( t \), the decoding will not be possible. Again, no indications of a problem will be given, making it crucial to avoid such situations during the running of a service.

**Fractional encoder**

It is possible to encode real numbers using the fractional encoder. A slightly different procedure is needed to carry out the encoding. As for the integer encoder, fractional encoders are parametrized by an integer base \( B \geq 2 \) \[20\], with the same exact role it had in the previous case. The procedure consists in encoding the integer part of the value, using the usual integer encoder. \( n \) is added to each exponent of the fractional part of the binary expansion of the value. Then, the base \( B \) is changed into the variable \( x \); lastly, the signs of each term are flipped. \[19\] presents a practical example. Consider \( B = 2 \) and the rational number 5.8125. It has a finite binary expansion:

\[
5.8125 = 2^2 + 2^0 + 2^{-1} + 2^{-2} + 2^{-4}
\]

The integer part is encoded as usual, obtaining the polynomial \( \text{IntegerEncode}(5, B = 2) = x^2 + 1 \). Then, \( n \) is added to each exponent of the fractional part \((2^{-1} + 2^{-2} + 2^{-4})\) and the base 2 is changed into \( x \): the result is \( x^{n-1} + x^{n-2} + x^{n-4} \). Lastly, each sign of the terms is switched:

\[-x^{n-1} - x^{n-2} - x^{n-4}\]

More formally, for any rational number \( r \) with finite binary expansion it holds:

\[
\text{FracEncode}(r, B = 2) = \text{sign}(r) \cdot [\text{IntegerEncode}(|r|, B = 2) + \text{FracEncode}(|r|, B = 2)]
\]

The decoding consists essentially in the application of the steps described above, in reverse order. It should be noted that not every rational number has a finite binary expansion. In this case, the expansion of the fractional part has to be truncated to some precision (which can be expressed as \( n_f \) bits). The simplest solution is to fix a number \( n_i \) to denote the number of coefficients reserved for the integer part,
while the remaining \( n - n_i \) coefficients will be used for the fractional part. The condition \( n_f + n_i \leq n \) should be respected.

Similarly to the Integer encoder, also the decoding with the Fractional encoder can fail. The first reason is the same of the previous case: if any of the coefficients of the plaintext polynomials is greater than the plaintext modulus \( t \), the decoding will probably fail. The second reason is that an homomorphic multiplication may cause the fractional parts of the plaintext polynomials to expand down towards the integer part, with the result that the two parts get mixed up.

### 2.1.3 Practical considerations on the use of HE

To summarize the aspects exposed in the previous Section, the fundamental notions needed for a practical use of HE are here exposed. The BFV scheme relies on the following set of encryption parameters (from now on denoted with \( \Theta \)):

- \( m \): Polynomial modulus degree,
- \( p \): Plaintext modulus, and
- \( q \): Ciphertext coefficient modulus.

The parameter \( m \) must be a positive power of 2 and represents the degree of the cyclotomic polynomial \( \Phi_m(x) \). The plaintext modulus \( p \) is a positive integer that represents the module of the coefficients of the polynomial ring \( R_p = \mathbb{Z}_p[x]/\Phi_m(x) \) (onto which the RLWE problem is based). Finally, the parameter \( q \) is a large positive integer resulting from the product of distinct prime numbers and represents the modulo of the coefficients of the polynomial ring in the ciphertext space.

Using different values for this set of parameters affects the performance obtained in the HE setting. This is reflected in the number of operations which can be done on a ciphertext without corrupting it, on the computational load requested to perform each operation, on the memory requested and on the security against ciphertexts attacks. Moreover, parameters affects also the precision of the operations’ result, in particular when real numbers are considered.
The Noise Budget (NB), introduced in Sub-Section 2.1.2, is a practical indicator for the number of operations which can be done on the selected ciphertext while guaranteeing the correctness of the decryption operation. Operations like additions and multiplications consume NB, in particular multiplications between ciphertexts are the most expensive ones in terms of NB consumption. While plaintext values do not have a NB, operations which involve a ciphertext and a plaintext still consume a certain amount of NB (consider that the result of such operation will be an encrypted value). If the NB of a ciphertext reaches 0, it becomes impossible to correctly decrypt it.

Hence, it is necessary to carefully tune the encryption parameters to guarantee a sufficient initial NB to carry out all the planned operations.

Choice of the parameters

In this part, the effects of every parameter are explained more in detail.

- Increasing the value of \( m \) drastically increments the initial NB of fresh encrypted ciphertexts. However, it increases the memory occupations of ciphertexts and the computational load i.e. the time requested to perform an operation involving ciphertexts;

- The value of \( p \) is directly related to the accuracy of the operations’ results. Increasing \( p \) will increment the precision of the operations: the decrypted result of an operation will be more closer to the operation’s result performed in the plaintext space. However, increasing \( p \) also increases NB consumption during operations: this can make the NB drop to 0.

- \( q \) affects the security of the scheme, and represents a very difficult parameter to set. For this work the choice of \( q \) was delegated to the SEAL library [21] which provides a way to automatically set \( q \) given \( m \) and the desired AES-equivalent security level [19]. This is the suggested method.

To sum up, the experiments presented in Chapter 6 use different values of \( m \) and \( p \), while \( q \) is always automatically set using an AES security level of 128-bits. More considerations on the encryption parameters setting are presented in Section 4.4.
2.2 Machine Learning

The term Machine Learning (ML) indicates a subset of artificial intelligence. In particular, ML is the study of computer algorithms which are able to improve automatically in solving a specific task: they reflect the behaviour of human beings who are able to improve their performance with experience [4]. A more precise definition for a learning algorithm is:

**Definition 2.2.1.** A program learns from a certain experience \( E \), with respect to a class of tasks \( T \), obtaining a performance \( P \), if its performance in solving tasks of type \( T \), measured by performance \( P \), improves with the experience \( E \).

ML algorithms are often used in situations in which it is needed to classify some input data: they are often used for image recognition, spam filtering, text-to-speech or speech-to-text applications. Such algorithms use a mathematical model which is optimized using sample data (training data). There are at least three different paradigms used to build the model, described in the next parts of this Section.

**Supervised learning**

The program is given some inputs \( (X) \) and the desired outputs \( (Y) \). Through a series of attempts, the program constantly improves with the goal of finding the best possible mapping between the inputs and the outputs \( (Y = f(x)) \). If the examples are enough representative of the problem’s domain, the final mapping will have a good performance even on new, unseen data. Supervised learning is generally used in classification task, where every input data should be labeled in the correct way.

**Unsupervised learning**

In this modality the inputs are not labeled beforehand. The algorithm has to solve a clustering problem: common patterns will have to be found in inputs, with the goal of grouping together data which present some common traits.
Reinforcement learning

Used in dynamic environments, reinforcement learning uses a reward system in order to make the program optimal in doing a complex task, i.e. playing a game with an opponent. Advantageous moves receive a positive feedback from an external actor: given that the goal for the program is to maximize the reward - thus the positive feedbacks - a good performance in the task will be achieved.

In general, ML algorithms outperform human ones in tasks which encompass an high amount of possible input data with a corresponding amount of hidden patterns in the mapping between inputs and outputs. Building ad-hoc algorithms in such domains can be time consuming and have bad performances, while the capacity of ML algorithms of improving themselves simplifies the work of programmers. The work will shift to the identification of representative couples of input and output data and to the manual labeling of such training set.

2.2.1 Deep Learning

One of the most important types of ML algorithms is Deep Learning (DL). The main characteristic of DL models is to be composed of various layers, positioned in a series. Input data flows through every layer, being modified by each of them. This lets the model recognize patterns in input data in a progressive way, making it possible to obtain much more precise mappings.

While there exists a number of DL models, this work will focus in particular on Convolutional Neural Networks.

2.2.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a class of DL models usually applied in computer vision for image recognition, introduced Le Cun et al. in 1990 [22]. There is a number of characteristics which accumulate CNNs and human brains. First of all, CNNs are Artificial Neural Networks (ANNs). ANNs are computing systems which are inspired by the structure of human (more general, animal) brains. In
particular, ANNs are constituted of a series of connected nodes called artificial neurons. Each node can transmit signals to the next nodes. When a node receives an input signal it processes it, according to a function, and transmits the output to connected nodes. If the nodes are organized in layers, where every layer propagate signals only to the next one, the network is a feed-forward neural network.

This structure lets CNNs decompose and analyze images in a progressive way; every layer, with training, becomes able to recognize one or more patterns (features), making the model moldable. Another point of strength is that there is no need to manually extract such features from the training set: CNNs, during training, automatically recognize patterns useful to distinguish an image of a bike from the one of a cat, as long as the training set is large enough and the learning parameters are correctly set.

The typical layers of a CNN are here explained with more details.

**Convolutional layers**

Being the core block of a CNN, Convolutional layers are composed of a set of filters (also called kernels). Filters are matrices of real numbers; these values are often called the “weights” of the CNN. Every input image, modeled as a 3D matrix (in case of a colour image), passes through the convolutional layers where filters are applied. Every input image, divided into portions of size equal to the size of the filters, is multiplied through a dot product with the value of the filters. The result is a 2D activation map of that filter.

The idea is that after every convolutional layer some patterns of the input image are highlighted, until the input data is transformed into a prediction, i.e. an association of the input image with a label. Example of features are edges and points, at the simplest level, but near the end of the model even more complex figures can be recognized.

**Activation functions**

Between the various layers, activation functions can be applied. These functions modify the output value of a node before passing it to the next nodes, in a way similar to the one applied by animal neurons
in transmitting messages. Only non-linear functions are useful in the context of neural networks. Once a function is assigned, every input value will be transformed according to such function. While a number of different activation functions are used in CNNs, Rectifiers (which are employed in Rectified Linear Units - ReLU) are the most used one. A ReLU computes the function:

\[ f(x) = \max(0, x) \]

### Pooling layers

When dealing with large images, it can be useful to know only the relative position of different characteristics, instead of the absolute one. With this assumption, input images can be simplified (lowering the computation load of the CNN) using pooling layers. Usually placed between consecutive convolutional layers, pooling layers can use different functions to implement such simplification: the most common ones are average poolings and max poolings. In the process, portions of the input images are analyzed, for example a small window of 2×2 pixels. The 4 values contained in such window are averaged or the maximum one is taken. Then, the window is moved at a certain distance, called stride. The process is repeated until the whole image is consumed. The result will be an image of lower dimensions; however, the important features are still correctly placed with respect to each other. Often, this is sufficient to get a correct final prediction.

### Batch normalization layers

In a CNN, during the training phase, layers’ weights are updated layer-by-layer backward, from the output to the input, with the goal of making the model more accurate. However, for optimal results, this process should update only one layer each time, keeping all the other ones fixed. For practical reasons, though, usually all the layers are updated at the same time, causing a loss in the learning performance. Batch normalization is a possible solution to this problem; after an activation function, a batch normalization layer subtracts the batch mean from the output and divide it by the batch standard deviation.
The result is that the time needed to train the model is drastically reduced. In this work, batch normalization layers have not been used.

**Fully connected layers**

Together with the convolutional layers, fully connected layers (also called dense layers) are the most common ones in a CNN. Usually placed at the end of the model, fully connected layers receive as input the flattened values of the previous layers and multiplies it for a matrix of weights. Often, a bias offset is added to the output. These layers are the ones responsible of converting the output of the convolutional layers (the application of the filters), which are still 2D images, to a 1D vector. This is the key to obtain predictions.

**Softmax layer**

Usually the last layer of a CNN, it has a number of neurons equal to the number of classes which should be used for the image classification. Moreover, the values corresponding to each class are squeezed in the range from 0 to 1: in practise, the output vector will contain the probability of the input being in each one of the classes.

### 2.3 Cloud

Cloud computing is of interest for this work because the architecture proposed in Chapter 4 includes a server part, which is meant to be hosted on a Cloud environment. While the concept is still relatively new, one of the accepted definition of Cloud Computing is the one proposed by the National Institute of Standards and Technology (NIST) [23].

**Definition 2.3.1. Cloud computing** Cloud Computing is a model for enabling ubiquitous, convenient, on demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.
Another definition, proposed by Wang et al. in 2008 \cite{24} is more centered on the idea that users have of the Cloud paradigm:

**Definition 2.3.2. Cloud computing** A computing Cloud is a set of network enabled services, providing scalable, QoS guaranteed, normally personalized, inexpensive computing platforms on demand, which could be accessed in a simple and pervasive way.

At the basis, the Cloud is composed of a network of machines, usually with an important total amount of network bandwidth, computational power and storage. However, the peculiarity of the Cloud paradigm is in the way these resources can be accessed. Clients can see the cloud as a potentially infinite resources pool, and use only the amount they need, usually with a “pay-as-you-go” payment model.

### 2.3.1 Cloud provider

A Cloud provider is a person (more often an organization) responsible for acquiring and managing a computing infrastructure and for making a service available to interested parties \cite{25}. The nature of the service may vary. In the simplest case, direct access is given to some machine - often virtual ones - while in other case more complete and “ready” solutions are provider. While the list is constantly growing, the NIST indicates three fundamentals services paradigms offered by a cloud:

**Software as a service**

The Software as a service (SaaS) solution makes available to final consumers the access to software applications which are installed on the Cloud provider’s machines. It is responsibility of the provider to install the software, keep it updated and allocate the adequate amount of resources to guarantee its correct functioning. Consumers can have access to the software through a web browser, or a dedicate thin client. This is a very convenient paradigm for companies or single users who do not want or can not manage the environment needed to keep the application running, for lack of resources or time. Examples of SaaS are online office suites, accessed through browsers by consumers, or online games. Also file hosting services can be seen as SaaS, as long as they
provide services through a ready-to-use interface, directly available to consumers.

**Platform as a service**

Platform as a service (PaaS) differs from the SaaS paradigm in the fact that the running application is not managed by the provider; the consumer has the possibility to deploy its software application on the cloud infrastructure using services like execution stacks, compilers, database or, in general, middleware. Direct access to machines hardware is not given to consumers. The platform is set to give specific boundaries on permitted operations. For example consumers can set some hosting environment settings (e.g. chose a specific version of a middleware), but can not access the operating system or the hardware configuration. Examples of PaaS are web-applications hosting services. Consumers can deploy their own applications and make it available to end users. The responsibility of the provider is to guarantee the correct functioning of the underlying stack and the allocation of the adequate amount of resources.

**Infrastructure as a service**

The infrastructure as a service (SaaS) is directly above the direct machines-renting. Consumers have access to every function of their systems, which includes the possibility to change OS, access network settings and plain storage space. This way the consumer can tailor its experience in the lowest detail. The Cloud provider has to run the minimal software needed to make the computing resources available to consumers, including abstractions like virtual machines and virtual network interfaces. The physical access to hardware remains responsibility of the Cloud provider.

An additional paradigm, on which part of this work is based, is Machine Learning as a service (MLaaS). While a precise definition of MLaaS does not exists, the paradigm is heavily driven by some of the
dominant Cloud companies in the world: Google\footnote{https://cloud.google.com/ai-platform}, Microsoft\footnote{https://azure.microsoft.com/en-us/services/machine-learning/} and Amazon\footnote{https://aws.amazon.com/it/machine-learning/} have made available their own idea of MLaaS. MLaaS can be seen as a particular case of PaaS. Managing a ML infrastructure is not easy and, considering the huge amount of resources needed by ML solutions, is it also not convenient in the most cases. In this case, Cloud providers can offer platforms already optimized for the deploy of ML algorithms, which are managed by the customers.

### 2.3.2 Advantages of Cloud solutions

\cite{26} summarizes the main advantages of Cloud computing:

1. Smaller firms, often, can not afford to build an entire infrastructure to carry out computational exercises which typically involve large amounts of computing power for short amounts of time. The payment models of Cloud computing allow such dynamic provisioning of resources.

2. The access to computational resources is basically instant and do not request an upfront capital investment for consumers. This drastically reduces the entry barrier for new startups, especially in the Internet business area.

3. Startups, in particular, can benefit from a lower IT entry barrier, making them able to focus on their core business.

4. Cloud computing makes it easier to scale resources on the basis of services’ needs. Software can scale resources up and down dynamically, leading to lower costs and higher efficiency.

5. In some cases, Cloud computing can be the only options to deploy some particular applications or services, in particular when such services require worldwide interactions or huge amount of computational resources.

Moreover, Cloud computing solutions are often characterized by an high level of dependability.
Definition 2.3.3. Dependability The dependability of a computing system is the ability to deliver service that can justifiably be trusted [27].

This includes a variety of characteristics like resistance to failures, guarantees on the correctness of provided services and so on.

2.3.3 Drawbacks of Cloud solutions

There are at least two factors which should be taken into consideration when evaluating the possibility of using the Cloud. The first one is related to vendor lock-in. The vendor lock-in problem in cloud computing is the situation where customers are dependent (i.e. locked-in) on a single cloud provider technology implementation and cannot easily move in the future to a different vendor without substantial costs, legal constraints, or technical incompatibilities [28]. Also, the Cloud has a cost in terms of customization of the solutions, because the customer can not choose or modify every single aspects of its IT architecture.

The second drawback of Cloud solutions is related to privacy. This aspect has been discussed in Chapter [1].
Chapter 3

Related Literature

This Chapter collects studies and works with the goal of offering privacy-preserving solutions for Machine Learning and other examples of architectures similar to the one presented in this thesis.

The first work which proposed the use of HE to maintain the privacy of data during the computation has been introduced by Rivest et al., in their 1978 work “On data banks and privacy homomorphisms” [29]. Interestingly, this idea was inspired by a need similar to the one discussed in this work. Their example was the one of a small loan company which uses the data banks of a time-sharing service. The problem is that if the company decides to encrypt their data before submitting it to such data-bank, in order to maintain the information contained private, it becomes impossible to use other computational resources offered by the time-sharing service to answer some questions about the stored data. Consider the case in which the company wants to know how much income from loan payments is expected the next month, remembering that the data is stored, encrypted, in the remote data-bank. The company, according to the authors, has four options:

- Stop using the time-sharing service and rely on an internal computing system;
- Use the data-bank only to store the encrypted data, and use an internal computer system to gather the needed data, decrypt it, and compute the requested functions;
- Request special modifications to the time-sharing service’s com-
puters which would allow them to decrypt data only for the time needed to compute a specific function, with no possibility of external accesses;

• Use a special “privacy homomorphism” to encrypt the data so that the time-shared service’s computers can still operate on the data, without decrypting it.

The fourth option is basically the one pursued in this work; the concept of time-shared service has been replaced by the concept of Cloud, but the needs of consumers and final users to maintain the privacy of data while being still able to use external services remain the same.

3.1 First HE schemes

In the following years a number of HE schemes were introduced. However, there were important limitations which slowed down the adoption of HE. Three important works introduced new HE schemes which would allow only additions. The first [30] proposed a public-key cryptosystem which used RSA integers, while offering some advantages over that scheme: one of them being the possibility to perform additions and subtractions between ciphertexts. According to the authors, this could be sufficient to obtain the tally of an election without the need of decrypting every vote. This is an interesting idea which, however, is still far to be implemented worldwide. [31] and [32] schemes were similar under the aspect of being homomorphic only with respect to additions.

It took time until 2009 for an homomorphic encryption scheme allowing both additions and multiplications to be proposed. Gentry [33] introduced a scheme based on ideal lattice-based cryptography, with the goal of supporting arbitrary circuits. Moreover, in the same work Gentry introduced the idea of bootstrapping:

**Definition 3.1.1. Bootstrappable** A C-evaluation scheme is called bootstrappable if it is able to homomorphically evaluate its own decryption circuit plus one additional NAND gate [34].
This was the starting point for the idea of a Fully Homomorphic Encryption scheme. The basic idea is to re-encrypt a ciphertext when its value of noise become to high: the fresh ciphertext would still be a ciphertext corresponding to the initial plain value, but with a lower value of noise.

Gentry’s work has been improved in various ways. In 2010 a new work [35], which had Gentry as one of its author, relaxed the ideal lattice assumption of the previous scheme, and introduced the idea of using integer polynomials rings to define the ciphertexts. The interest in HE grew, and with that new more efficient schemes became available. These are the schemes on which the majority of HE’s based applications are based. In 2014 the Brakerski-Gentry-Vaikuntanathan (BGV) was introduced in [14]. This scheme relied on polynomial rings to define ciphertexts. The theoretical guarantees of its security were now based on the learning with error problem (LWE), with extensions for the ring learning with errors problem (RLWE). A further evolution is represented by the Brakerski/Fan-Vercauteren (BFV) scheme, detailed in Section 2.1.2. Possibly, the most recent HE scheme is the Cheon-Kim-Kim-Song (CKKS) scheme [13], which extends the polynomial rings to complex numbers and isometric rings.

### 3.2 Use of HE

Once HE schemes became sufficiently efficient to be used in real applications, also researchers in the field of machine learning started to consider its use in the related processing chains. One of the most important implementation is CryptoNets [36]. The authors motivated their work with the need of ML algorithms to work on sensitive data, like medical one. Using the HE BFV scheme, they presented a method to convert plain learned neural networks into CryptoNets, neural networks which can be applied on encrypted data. Their contribution includes several possible ways of approximating the non-linear computations requested in many layers of a ML model. Experimental results shows the effectiveness of such approximations, with a prediction accuracy of 99% on the MNIST [3] dataset. One difference with respect
to this work is that CryptoNets focused only on CNNs, while the architecture presented here is meant to be more general.

Bourse et al. [37] presented a new framework (called FHE-DiNN) which allows a scale-invariant approach to neural networks evaluation on encrypted data. In their approach, each neuron’s output is refreshed through bootstrapping. The consequence is that, theoretically, arbitrary deep networks can be homomorphically evaluated. The computational load will be proportional to the depth of the network or, if neurons are grouped in layers, to the number of the layers. The models used in that work are called “Discretized Neural Networks”:

**Definition 3.2.1. Discretized Neural Network** A Discretized Neural Network (DiNN) is a feed-forward artificial neural network whose inputs are integer values in \([-I, \ldots, I]\) and whose weights are integer values in \([-W, \ldots, W]\) for some \(I, W \in \mathbb{N}\). For every neuron of the network, the activation function maps the inner product between the incoming inputs vector and the corresponding weights to integer values in \([-I, \ldots, I]\).

This idea allowed them to use a restricted input space (\([-1, 1]\)) and \(\text{sign}(\cdot)\) as the activation functions of hidden layers. The advantage is that if the message space is smaller, the efficiency of the overall evaluation can be incremented.

The very recent nGraph-HE2 framework [38] is an extension to the Intel nGraph deep learning (DL) compiler [1], which allows users to train CNNs in plaintext on a given hardware and deploy trained models to HE cryptosystems, with the goal to hide as much as possible the underlying complexity of HE. Using the CKKS scheme (with ad-hoc optimizations), nGraph-HE2’s goal is to let data scientists deploy pre-trained model using their native activation functions and number fields, with minimal code changes. In particular, for non-polynomials activation functions (like ReLU) the framework uses a client-aided computation approach: the intermediate results of the network evaluation are sent to the client which decrypts the data, computes the activation

---

function in plaintext space, then re-encrypts the data and send it again to the server, where it will be used in the remaining part of the model.

3.3 Secure Multi-Party Computation

The Secure Multi-Party Computation (SMC) approach provides methods for different actors (namely, different parties) to collaborate in computing a function over their inputs, while keeping those inputs private. The idea is that not only the plain data should not be intercepted from malicious actors not directly involved in the communication, but also the privacy should be maintained between each participant. SMS consists of the following three phases: sharing, computation and reconstruction [1], as shown in Figure 3.1. In the sharing phase, each player acts as a dealer to distribute shares of his secret among all the other parties. Subsequently, computation is done using two secure operations: addition and multiplication. They then collaborate and perform some further computations. Finally, in the reconstruction phase, players reveal the shares of the function value so that all the players learn it.

Consider a simple example. Alice, Bob, and Charlie would like to know the age of the older one between them, without revealing their age to each other. If $x, y, z$ denote the age of, respectively, Alice, Bob
and Charlie, then the function they want the compute is:

\[ F(x, y, z) = \max(x, y, z) \]

The only information that each party can learn from the result of the function is related to their private input. If the result was 42 and Alice is 40 years old, she can know that she is not the oldest one of the group; however, she can not know if the oldest is Bob and Charlie. SMC makes it possible to implement such example.

The SMC approach was formally introduced by [39] in 1998, even though the sub-problem of Secure Two-Party Computation (2PC) was anticipated by Yao [40]. Yao proposed 2PC as one possible solution for the Yao’s Millionaires’ problem, in which two parties want to know who between them is richer, without revealing their actual wealth.

In the literature some works involving ML and SMC have been proposed. One of them is [41] which describes a protocol which uses the Pailler HE scheme [32] and SMC algorithms to allows Neural Networks owner to process encrypted data, coming from another party. One peculiarity of this work is that also the knowledge contained in the network is protected; at the simplest level only the weights are not revealed, while extensions can be made to hide also the internal structure of the model.

A slightly different approach is the one of Mohassel et al. [42]. SecureML is a protocol which falls in the two-server model. Two servers wants to train a ML model on the joint data coming from data owners, using 2PC. Data owners distribute their private data to the two servers (which are non-colluding), in order to let each server have only a partial knowledge on the incoming data. The resulting model will have an higher accuracy with respect to single models trained by each server on their own, however no exchange of users-provided data will be made.

Finally, the Gazelle framework [43] applies SMC and HE directly on CNNs. This is an important work, which includes an high number of optimization and ad-hoc solutions for improving the performance of CNNs evaluation on encrypted data. In particular, it includes custom implementations for additions and multiplications, which make operations on ciphertexts only 10-20 times slower than the corresponding operations on plaintexts. Moreover, custom algorithms were developed
for matrix operations and convolutions, which are obviously some of the main ingredients of CNNs.
Chapter 4

Architecture of the proposed solution

The goal of this work is to propose an architecture able to fulfill the typical requirements of a deep-learning-as-a-service paradigm, while maintaining the privacy of the data sent by the users. The architecture, called HE-DL, is shown in Figure 4.1. HE-DL relies on a distributed approach and involves two actors, namely the user and the Cloud. The goal of the user is to use a DLaaS, published by a service provider. This service requires that the user sends some data to the cloud, where it will be processed. Usually the result of this process is the actual service needed by the user, who will receive it on their machine.

Note that this goal is let broad on purpose to highlight the fact that the architecture is not specifically focused on a particular use case; however, it may useful to keep in mind some examples of similar services, like Amazon AWS Rekognition\footnote{https://aws.amazon.com/rekognition}.

Differently from traditional situations, in HE-DL the user is requested to encrypt the data which will be elaborated by the Cloud. This has to be done according to specific rules, in order to guarantee the correctness and the security of the expected results; the aspects related to this phase will be described in Section 4.1. Once the data is sent to the Cloud, the DLaaS provider will have to process the encrypted data, without being able to decrypted it. In Section 4.2 the advantages and limits of this approach will be presented. Consider-
Figure 4.1: The proposed privacy-preserving architecture for deep-learning-as-a-service.
ations about the encryption parameters and the transmission of data between the user device and the Cloud are made in Sections 4.4 and 4.5.

In order to provide a more practical meaning to the work, the general architecture is tailored to a specific use case, which consists in the classification of input images $I$ into a class $y \in Y$, using CNNs as DL models. The provider offers the service in two different modalities, which will be detailed in depth in Section 4.3.

### 4.1 Encryption and Decryption

The phases of encryption and decryption are fundamental for the architecture. The possibility for the Cloud party to have access to the plain data sent by the user relies on the weakness of the encryption phase: weak parameters or incorrect processes can compromise the security of the user, exposing him or her to all the risks exposed in Chapter 1. However, errors in the encryption or decryption phases can also lead to data corruption. If these phases are not carried out correctly, it may be possible that the Cloud party can not perform the requested computation.

As commented in Chapter 2, the encryption, processing and decryption phases request a set of encryption parameters $\Theta = \{m, p, q\}$ whose choice influences the correctness of the result and the security of the encrypted data; being it an asymmetric encryption scheme, a key pair is generated on the user device according to the values of the encryption parameters. Formally, the user is required to encrypt the data $I$ to be processed, through a process called Encryption:

$$\hat{I} = E(I, \Theta, k_p)$$

The required parameters of $E(\cdot)$ are:

- $I$: user data
- $\Theta$: encryption parameters
- $k_p$: public key

The result $\hat{I}$ represents the encrypted user data. This data is ready to be processed by a third party, namely a server on the DLaaS
provider. The data can be of any kind, as long as it is representable: images, audio, biometric data. After the transmission of $\hat{I}$ from the user device to the Cloud, the DLaaS can process it, obtaining the result denoted by $\hat{y}$. The result is still encrypted, and can only be decrypted by the user. After the transmission of $\hat{y}$ from the Cloud to the user device, the user has to decrypt it in order to access it, through a process called *Decryption*:

$$y = D(\hat{y}, \Theta, k_s)$$

The required parameters of $D(\cdot)$ are:

- $\hat{y}$: the output of $\varphi_\Theta(\hat{I})$
- $\Theta$: encryption parameters
- $k_s$: secret key

$y$ denote the plain output of the requested service. The meaning of $\varphi_\Theta(\hat{I})$ will be explained in the next Section; at this point it is necessary to know that it represents the process that the service provider performs on the encrypted data.

In the following, the exchange of encrypted images along with the encrypted results will be considered. More precisely, an image will be denoted as $I \in \mathbb{R}^{w \times h \times c}$ with height $h$, width $w$ and channels $c$. $c$ will be equal to 3 if RGB images are considered, 1 for grey-scale images. For the sake of this work, $I$ can be considered as a tensor of dimension $c \times h \times w$ whose values are float numbers ranging from 0 to 1. No attempt is made to hide the dimension of the images; hence, an encrypted image $\hat{I}$ can be considered as a tensor of dimension $c \times h \times w$ whose values are the ciphertexts corresponding to the plain values of $I$, encrypted according to the HE-BFV.

### 4.2 Approximated and encoded DL processing

At the other end, the deep learning processing is carried out in the Cloud. This is reasonable considering that the processing of encrypted data is more demanding in terms of computational load and memory
4.2 Approximated and encoded DL processing

occupation with respect to the same processing applied on plain data. In this context, the computational resources offered from the Cloud paradigm are fundamental to the feasibility of the architecture. Consider a set of deep-learning models $DL \ models \ f(\cdot)$s. These are the models the DLaaS Provider offer to the users. As stated in Chapter 2, one of the main limits of HE-BFV is that only additions and multiplications can be carried out on encrypted data. This means that the $f(\cdot)$s cannot be used as they are, but they have to be modified in order to not use operations unavailable in HE-BFV.

This process is called Approximation. After being approximated the set of approximated $DL \ models \ \varphi(\cdot)$s is obtained. To make it possible to process the encrypted data $\hat{I}$ sent by the user, the approximated DL models have to be encoded. The Encoding process is the one which converts a plain number in a form which can be added or multiplied with a ciphertext, in the scheme of HE-BFV. Once the encoding is applied on all the weights of every model, the set of encoded deep-learning-as-a-service $\varphi_\Theta(\cdot)$ is obtained. This process requires the encryption parameters $\Theta$. As anticipated, the Cloud, after the approximation and encoding of the DL models, is ready to process the encrypted data $\hat{I}$ sent by the user, obtaining:

$$\hat{y} = \varphi_\Theta(\hat{I})$$

The use case outlined in the first part of the Chapter consists in the classification of input images $I$ into a class $y \in Y$. To perform this task CNNs will be used as $DL \ models$. The reason for this is that CNNs are widely used in the field of image classifications. Moreover, CNNs are composed by mainly additions and multiplications, making them suitable candidates to be considered within a HE scheme.

Let $f(\mathcal{I})$ be a CNN composed of $L$ layers $\eta_l^{(i)}$ with parameters $\theta_l$ and $l = 1, \ldots, L$, aimed at extracting features and providing the classification output $y$ of an input image $\mathcal{I}$. The general architecture of $f(\mathcal{I})$ is shown in Figure 4.3a.

In this case, the approximation phase will be focused on activation functions. That is because convolutional layers and linear (full-dense) layers are composed only by additions and multiplications, so no approximation is needed for them. In other words, these layers involve
only polynomial functions, which are the only ones which can be computed directly in the HE-BFV scheme. The function of the ReLU activation function, for instance, is:

$$f(x) = \max(0, x)$$

Being the max operation non-polynomial, it can not be executed directly in a HE-BFV scheme. There are some options to replace it with another one. In this case the Square activation function will be used, whose function is:

$$f(x) = x^2$$

Another example of layer which cannot be executed directly in a HE-BFV scenario is the Max-pooling operator. The pooling operation transforms a sub-matrix coming from the image into a smaller one whose values depend on the type of pooling applied. In the Max-pooling the maximum value will be taken from a sub-matrix, and then used for the output. Given that the max operation is not polynomial, instead the Average-pooling operator will be used. If $$f_s$$ is the pooling size, fixed and a-priori known, the output can be obtained by summing all the values in the submatrix and multiplying the sum for $$\frac{1}{f_s}$$. The result of this approximation is a CNN $$\varphi(\cdot)$$ whose processing layers $$\phi_{\hat{\theta}_l}^{(l)}$$ can be encoded with the considered HE-BFV scheme. To simplify the notation the parameters of each layer $$\theta_l$$ or $$\hat{\theta}_l$$ are omitted from now on.

It is important to note that, after performing the replacement of the non-polynomial layers, the model has to be trained again. This is necessary because the weights of the plain model are not valid anymore if the activation functions or other layers have been replaced by different ones. Hence, to provide a deep-learning-as-a-service, $$\varphi(\cdot)$$ must be retrained with the same settings in which the plain one was trained (e.g., same dataset, same learning algorithm, etc..). Obviously, if the original model $$f(\cdot)$$ already contains only HE-compatible processing layers, this procedure is not necessary.

Moreover, it’s noteworthy that this approximation process can introduce a variation in the accuracy between $$f(\cdot)$$ and $$\varphi(\cdot)$$. This aspect will be explored in the experimental part described in Chapter 6.
data is to *encode* the values of their weights. As already mentioned, it is necessary to transform the plain values into an encoded form, compatible with the encrypted values. It’s important to note that the model is not encrypted: the encoded values are simply in a different form with respect to the plain ones, but they are still “plain”. No attempt is made to obfuscate the weights of the model. In order to encode the model, the values of $\Theta$ are needed; they have to be transmitted by the Client, and they have to be the same used to generate the encryption keys and to encrypt the values. The approximated and encoded CNN is denoted by $\varphi_\Theta$. We are ready to define the HE-based encrypted processing, as shown in Figure 4.3b which can be formalized as follows:

$$y = D(\varphi_\Theta(E(I, \Theta, k_p)), \Theta, k_s)$$  \hspace{1cm} (4.1)

This result encloses all the fundamental aspects of the architecture: the encoded and approximated DL model $\varphi_\Theta$ transforms the encrypted incoming data denoted by $E(I, \Theta, k_p)$. The encrypted result $\hat{y}$ is then decrypted by $D(\cdot)$, using the secret key $k_s$, in $y$. This is the plain result ready to be used by the user.

A word should be spent on the security and privacy of the employed
model. The term *exploratory attacks* indicates an attempt to uncover information about the inner workings of a ML model in order to identify weaknesses of the algorithm [44, 45, 46] or to steal its structure and/or the its weights. This attack may attempt to extract:

- The decision boundary used by the algorithm,
- a general set of rules that the algorithm follows,
- a set of logical or probabilistic properties about the algorithm,
- information about the data that was used (or not used) to train the algorithm.

In this work, the privacy of the trained models has not been considered. The use of HE, at least in the modalities here presented, does not prevent an attacker to conduct systematic attacks in order to steal information about the models. Different countermeasures should be employed, even in combination with HE, to protect the models from malicious attackers, meanwhile guaranteeing the privacy of the users.

### 4.3 Offered DL modalities

As anticipated ad the beginning of the Chapter, the provider can leverage the flexibility of both DL and HE in order to differentiate the offered service. Taking into consideration the case of CNNs, the provider offers the service in two different modalities:

- **recall**: the processing $\varphi_\Theta(\cdot)$ provides the encrypted version $\hat{y}$ of the final classification $y$ of $I$ (Figure 4.4);

- **transfer learning**: the processing $\varphi_\Theta(\cdot)$ provides the encrypted version of only a processing stage of the considered CNN, applied to the input image $I$ (Figure 4.5).

The difference between the two modalities lies in the output of Eq. 4.1. In the first case the decrypted output $y$ of the CNN $\varphi$ last layer $L$ is already the final result of the entire CNN, applied to image $I$. Usually it is a softmax operation on top of a classification one. In the
4.3 Offered DL modalities

(a) The plain processing of an approximated CNN $\varphi(\cdot)$ composed of $L$ layers. Each layer $\phi^l$, with $1 \leq l \leq L$ is here composed only of multiplications and/or additions. The difference w.r.t. the usual CNN-based classifier $f(\cdot)$ relies only in these approximations. Note that also the layers parameters $\theta_l$, here omitted, may require to be approximated (and referred to as $\hat{\theta}_l$) in the approximated CNN $\varphi(\cdot)$.

(b) The encrypted CNN processing of the CNN $\varphi_\Theta(\cdot)$. The CNN is encoded with HE parameters $\Theta$, operates on images $\hat{I}$s with the same parameters $\Theta$ and returns the encrypted classification output $\hat{y}$.

Figure 4.3: A comparison of the plain and approximated CNN processing with the encrypted one. The layers parameters $\theta_l$ are omitted to simplify the notation.
Figure 4.4: Illustration of the recall modality. The user transmits an encrypted image to the server which returns the encrypted final classification of the image. It is then decrypted on the user machine.

second case the decrypted output $y$ is instead the result of only a part of CNN, namely the features $I^l$ extracted at a given CNN level $\hat{l}$, with $1 \leq \hat{l} \leq L$, typically a convolutional or pooling one. This modality opens up interesting scenarios.

4.3.1 Recall

In this modality the goal of the user is to classify an image $I$. A CNN $\varphi(\cdot)$ in the set of the DL models offered by the DLaaS provider is already trained to classify images belonging to the same domain of $I$. The user can utilize it, without further modifications. More precisely, the image $I$ will be encrypted into $\hat{I}$ and forwarded through all the layers of the model, after it has being approximated, obtaining $\hat{y} = \varphi(\hat{I})$. $\hat{y}$ is the final result the classification task. Then, $\hat{y}$ will be decrypted into $y$ on the user device, making the classification available to the user.

The limitation of this modality is that the chosen $\varphi(\cdot)$ has to be trained to classify images in the same domain of the input image $I$. 

An entire DL model is used.
4.3 Offered DL modalities

4.3.2 Transfer Learning

In this scenario the situation is different: the user would like to use a CNN to classify an image belonging to a certain domain, but no published ready-to-use CNN \( \varphi(\cdot) \) is trained to distinguish images in that domain. For example, the user would like to distinguish cats from dogs, but only CNNs able to distinguish cars from bikes are available. To solve this, the transfer learning modality can be used. Accordingly to the transfer learning paradigm \([47, 48]\), a pre-trained CNN can be split into two parts: feature extraction and classification.

In this modality the first part of the published CNN can be used as a pre-trained feature extractor whose output can be, then, used by an ad-hoc classifier to classify images in a different domain. The benefit for the user is that the feature extraction part is usually the

---

Figure 4.5: Illustration of the transfer learning modality. The user transmits an encrypted image to the server which returns the encrypted extracted features. They are then used on the client machine to obtain the final classification, using a local classifier.

For example, if \( \varphi(\cdot) \) is trained to recognize the digits, \( I \) should be an image of a digit. If \( I \) represents a dog, the classification \( y \) would be meaningless.
more resources consuming one in a CNN; having the possibility to shift this work on the cloud and locally train only the final classifier makes it possible to execute CNNs that, otherwise, would be too heavy. Moreover, it reduces the amount of images requested for the training. In this scenario, the encrypted images \( \hat{I} \) will be forwarded through the published encoded model \( \varphi_{\Theta} \) up to a layer \( \hat{l} \). The layer \( \hat{l} \) should be the last layer of the convolutional part. The result of this processing is:

\[
\hat{I}^l = D \left( \varphi_{\Theta}^l \left( E \left( I, \Theta, k_p \right) \right), \Theta, k_s \right)
\]

where \( \varphi_{\Theta}^l \) represents the encoded CNN up to layer \( \hat{l} \) with parameters \( \Theta \). \( \hat{I}^l \) represents the extracted features from every image \( I \). The user, having now the features extracted from every training image, can use them to train a local classifier (e.g., a Support Vector Machine) for the requested task. In particular, a set of \( K \) images \( I_1, \ldots, I_K \) will be sent to the HE – DL in the encrypted form \( \hat{I}_1, \ldots, \hat{I}_K \). The corresponding encrypted extracted features \( \hat{y}_1, \ldots, \hat{y}_K \) are then sent to the user who will decrypt them in \( y_1, \ldots, y_K \). This vector set is used together with the corresponding labels (which are available to the user) to locally train a classifier. Once trained the user can start using the entire distributed CNN to classify the images, always sending them to the HE – DL for the feature extraction part and then locally classifying them.

### 4.4 Encryption parameters

As already mentioned, the choice of the encryption parameters \( \Theta \) is critical to guarantee the correctness, feasibility and security of the encrypted processing. In the BFV schema, as described in Chapter 2, three parameters are fundamental: the Polynomial modulus degree \( m \), the Plaintext modulus \( p \) and the Ciphertext coefficient modulus \( q \). In particular the choice of \( q \) is difficult and it is the one which can severely affect the security of the encrypted data. For this purpose, \( q \) will established through the SEAL library [21] which provides a specific function that, given \( m \) and the desired AES-equivalent security level \( sec \), returns a suggested value for \( q \) [19]. For this reason, in the architecture the user will be able to set the two real encryption parameters,
4.4 Encryption parameters

$p$ and $m$, and a meta-parameter, $sec$. $q$ will be automatically chosen to guarantee the security level $sec$.

In the experimental part of this work a security level of 128 bits is always used. This is an arbitrary choice, motivated by the fact that the higher $sec$ is, the higher $q$ will be, resulting in an important computational overhead.

Set apart the choice of $q$, the problem here is to determine the best values for $p$ and $m$. First of all, it should be noted that such values are not secret: instead, they are needed from the Cloud in order to correctly encode the CNN. So, a set of pre-determined optimal $\Theta$ could be suggested from the provider directly to the users, for the most common use cases. This way, the user could simply delegate the choice to the provider.

To understand the effects of $m$ and $p$ it should be considered that, in general, incrementing $m$ will result in a higher computational and memory load, but also increment the available initial NB. $p$ directly affects the precision of the results, but also the NB consumption along operations. For every possible value for $m$ and for every possible set of operations, there is a maximum value of $p$ which can be used without making the NB going to 0, resulting in a corrupted output. Considering this, a possible strategy to find the best values for a given task is to set a value for $m$ (ideally starting from the lowest possible one permitted by SEAL, which is 1024) and to try different values for $p$, starting from the very low values (for example 32). If the output has a NB $> 0$, this means that $p$ can be incremented. Otherwise, it means that $p$ is too high and should be decremented. If the highest $p$ which doesn’t make the NB go to 0 produces an output with the requested precision, then parameters are obtained. Otherwise, increment $m$ and try with higher values for $p$. The effects of different choices for $\Theta$ on the precision and the computational load are shown in Chapter 6.
4.5 Communication between User Device and Cloud

The communication between the User Device and the Cloud leverages JSON-format messages. According to the REST paradigm, a request is done by the User Device to the on-line deep-learning-as-a-service provider when a computation is needed. This request will contain:

- A set of parameters, including: \( m \), \( p \) and \( sec \), the identifier of the chosen DL model \( \varphi(\cdot) \) and the numbers of layers to use (which will determine the modality, i.e, recall or transfer learning);

- The encrypted image \( \hat{I} \), encrypted using a public key generated by the User Device according to the encryption parameters.

For what concerns the parameters, \( m \), \( p \) and \( sec \) have to be sent to the Cloud in order to correctly encode the DL model. Moreover, the DLaaS provider should publish a list of offered DL models, from which the user can choose. The images \( \hat{I} \), once encrypted, are transmitted in a vector in which the ciphertexts are encoded as base64 strings. This makes it possible to embed them into JSON files; also the output of the computation, which is the answer provided by the Cloud, are encoded in the same modality. To give a practical example, let’s imagine that the user wants to classify a batch of 20 images from the FashionMNIST using a CNN whose identifier is Model1. Given that the user wants to use the CNN in its entirety, all the layers, from 0 to 6 will be requested. The JSON sent by the user will contain:

- \( m = 2048 \), \( p = 600201 \), \( sec = 128 \)

- The details of the model: (“model”=“Model1”, “layers”=[0, 1, 2, 3, 4, 5, 6])

- The encrypted image which will be a vector with dimensions [20, 1, 28, 28]

The answer JSON will contain the encrypted classification, so a vector of dimension [20,10] of ciphertexts.
Chapter 5

Implementation of the proposed solution

In this Chapter the PyCrCNN library will be introduced. This library is one of the possible implementation of the architecture proposed in this work. PyCrCNN is the Python implementation of a Crypto Convolutional Neural Network library. The difference with respect to other proposed library with similar goals (See Chapter 3) is that PyCrCNN aims to make it possible to hide the complexity of HE, both on the Cloud side and the client side. In particular, it can handle the parts of encryption/decryption (see Section 4.1), the part of encoding (see Section 4.2), and the encrypted processing (Section 4.3). Moreover, the network part of the library has been optimized to work in an Amazon AWS environment, showing the feasibility of the proposed work in a real-case scenario, with standardized and state-of-art solutions. The library which is introduced in the following Sections is divided in two parts: client and server. The server part satisfies these main requirements:

- **Requirement 1** Import the pre-trained model of a CNN and make it available to computation for incoming requests;

- **Requirement 2** Accept incoming computation requests and perform them.

while the client part satisfies these main requirements:
• **Requirement 3** Provide a callable API to send encrypted computation requests on a remote server.

## 5.1 External libraries used

In PyCrCNN these external libraries are used:

- Pyfhel [49]: Homomorphic encryption library
- numpy [50]: Scientific computation
-jsonpickle\(^1\): Serialization and deserialization of complex Python objects to and from JSON
- flask\(^2\): Micro web framework

## 5.2 Structure

PyCrCNN\(^3\) is a library written in Python language. It relies on the Pyfhel library v2.0.1 [49], Laurent (SAP) and Onen (EUROCOM) licensed under the GNU GPL v3 license for the tasks of HE; Pyfhel is a wrapper of the Microsoft SEAL library [21]. The parts of CNN like convolutional layers, fully connected layers, square activation functions have been built from scratch, with an eye on the PyTorch [51] library. The library can be divided into 8 sub-packages:

1. **convolutional**: Code for convolutional layers;
2. **crypto**: Code for cryptographic operations, like encrypting matrices, encoding matrices, etc;
3. **functional**: Code for functional operations like Square Layers, AvgPool layers, etc;
4. **linear**: Code for linear layers;

\(^1\)https://jsonpickle.github.io/index.html
\(^2\)https://flask.palletsprojects.com/en/1.1.x/
\(^3\)Code is available for download as a public repository at https://github.com/AlexMV12/PyCrCNN.git
5.2 Structure

5. **net_builder**: Code for building encoded model, starting from a PyTorch model;

6. **network**: Client and server code;

7. **parameters_tester**: Code to let user test the encryption parameters, given a model and such parameters. Statistics will be put in output;

8. **tests**.

The library is able to handle various types of layers:

- **convolutional_layer**
- **average_pool**
- **flatten_layer**
- **square_layer**
- **linear_layer**

Every layer has been built taking into consideration the paradigm used by PyTorch. Thus, every layer supports the following two methods:

- `__init__`: this function builds a `Layer` object using the same parameters that the corresponding function in PyTorch would accept for that `Layer`

- `__call__`: this function receives as parameter a tensor and process that tensor in the called layer. Returns the output.

This is intended to make the code easier to read and maintain. For the network part, two different server implementation are proposed. The first one is the one used for tests in Chapter 6 and it is designed to run on Amazon AWS. The second one is a simple Python application which exposes a REST server. It can be used on every system. This is detailed in Section 5.5.
5.3 Compatible models

The code provided in the net_builder sub-package can retrieve a PyTorch model with some restrictions and build an encoded model with the same layers/weights: that is, an equivalent model able to work on encrypted data.

To be used in PyCrCNN, a PyTorch model has to respect these constraints:

- It must be a Sequential model\(^4\);
- It must have extension .pt / .pth;
- It must have been saved in PyTorch with the \texttt{save()} function.

Models are saved pre-trained on the server location. Their id is communicated beforehand to potential users, which will be able to choose the model they want to use.

5.4 Client

The client part is executed on the user device; the request can start with a manual invocation providing the input data and the parameters by JSON or by a convenient API, callable in a PyTorch environment:

\[
\text{remote\_execution(data, parameters)}. \tag{5.4}
\]

This function accepts as input a NumPy vector and a set of parameters which includes \(m, p, sec\), the address of the endpoint, the ID of the chosen model \(\varphi(\cdot)\) and other secondary ones.

It automatically encrypts the input image \(I_s\) and decrypts the resulting answer \(\hat{y}\) in a fully transparent way with respect to the user. The return data is still a NumPy array.

Before starting the computation, a public and secret key pair \((k_p, k_s)\) is generated. The input batch is encrypted and the ciphertexts are encoded in base64 strings. Both the parameters needed by the server and the ciphertexts are included in the JSON payload.

\(^4\)\url{https://pytorch.org/docs/stable/generated/torch.nn.Sequential.html}
The JSON payload is either uploaded automatically to an Amazon S3 bucket or sent directly to the server if the standalone implementation is used. In the first case, using an S3 bucket is useful because the payload can have a important size, and it is possible for the user to specify a bucket of their property to be used. When the upload is finished, a POST request containing the address of the uploaded data is made to the URL specified in the parameters. This is the URL of the server endpoint which can now download $\hat{I}$ and proceed with the encrypted processing. Once the reply $\hat{y}$ is ready, the endpoint will upload it on the same S3 bucket used before, an ACK will sent to the client which will download it. It will be then decrypted in $y$ and returned to the caller as a NumPy array.

In the second case the payload is directly sent to the REST server and the client will wait for the response. After the request has been satisfied, the reply $\hat{y}$ is received, decrypted in $y$ and returned to the caller as a NumPy array.

An example of the parameters part of the JSON payload is:

```json
{
    "address" : "https://endpoint.com/compute",
    "encryption_parameters" : [
        {
            "m" : 2048,
            "p" : 13513511,
            "sec" : 128,
            "base" : 2
        }
    ],
    "net" : "1",
    "layers" : [0, 1, 2, 3]
}
```

In this example, the client is asking to forward the image on the net whose id is “1”, only on layers from 0 to 3. The client has encrypted the image with such parameters: the server will encode the model with them as well.
5.5 Server

The server side of the deep-learning-as-a-service is invoked via web API. Two different implementation are proposed:

Amazon AWS

To manage the request and transmit it to the processing part of the architecture the set of Amazon Web Services (AWS) tools is used. They comprise Sagemaker, Elastic Container Registry (ECR), AWS Lambda, API Gateway, and S3. In particular, the PyCrCNN algorithm extends the built-in models offered by Sagemaker: a Docker container compliant with Sagemaker Docker specifications is uploaded to ECR and the model is deployed on Sagemaker. The NginX is used as a webserver, Gunicorn as a WSGI and Flask as a web framework to expose the APIs required by Sagemaker.

The client has to invoke the endpoint URL with a POST request containing the S3 path to the image $\hat{I}$ and the aforementioned parameters. At the beginning, the plain model $f(\cdot)$ requested by the client is loaded into memory. Then, according to the $m$, $p$ and $sec$ specified by the client the model is encoded into $\varphi(\cdot)$. Then, the computation starts. Once finished, the results are uploaded on the same S3 bucket where the input data were uploaded, and an ACK message is sent to the client which can now download the results.

Standalone application

Built with Flask, this version runs a simple REST server able to receive POST requests from the client and to execute them using saved PyTorch models. It can be personalized to choose the amount of threads to be used (to speed up the computation) and can be run on every system where Python can be used. Docker images are also provided.

5.6 Testing

PyCrCNN implements many operations typical of CNNs from scratch. This was necessary because working with encrypted values has the
disadvantage that many operations commonly available in Python, in particular vector-wise ones, can not directly used. This requested a precise approach to the problem: analyzing the behaviour of CNNs on ciphertexts is not possible if the layers’ implementation is not correct because, for example, convolutional operations are not carried out correctly.

Thus, PyCrCNN has been designed with testing as first-class citizen. The goal was to obtain a 100% code coverage on all the fundamental layers used in a CNN (Table 5.1). Even if the models used in Chapter 6 are quite simple, the tests have been designed to take into account even corner cases like not-square filters/windows, not-square padding or not-square strides. For each of these cases a model has been designed first in PyTorch and tested against some arbitrary input data. Then, these models were ported into PyCrCNN and the same operations were performed on encrypted data. The results have been decrypted and checked for correctness. In this last case the values have been encrypted with encryption parameters known to be working, in order to make it possible to focus on the correctness of the circuits,

<table>
<thead>
<tr>
<th>Package</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>convolutional_layer.py</td>
<td>100%</td>
</tr>
<tr>
<td>linear_layer.py</td>
<td>100%</td>
</tr>
<tr>
<td>average_pool.py</td>
<td>100%</td>
</tr>
<tr>
<td>flatten_layer.py</td>
<td>100%</td>
</tr>
<tr>
<td>square_layer.py</td>
<td>100%</td>
</tr>
<tr>
<td>crypto.py</td>
<td>100%</td>
</tr>
<tr>
<td>encoded_net_builder.py</td>
<td>100%</td>
</tr>
<tr>
<td>padding.py</td>
<td>100%</td>
</tr>
</tbody>
</table>

Table 5.1: Code coverage obtained using the library coverage.py.
without taking into consideration problems caused by HE.

The tests have been done using the unittest library \footnote{https://docs.python.org/3/library/unittest.html}, while the code coverage has been obtained with coverage.py \footnote{https://coverage.readthedocs.io}.
Chapter 6

Experimental results

In this Chapter the implementation of the architecture is tested in order to evaluate the accuracy and the computation load of the deep-learning-as-service provided through PyCrCNN both in recall and transfer learning modality. The goal is to demonstrate the feasibility of the work in a real and concrete scenario: both the recall and transfer learning are tested with a common PC as a client and an Amazon EC2 machine as server. Two different CNNs models are used. For each one the approximation phase is described, with all the changes needed to make them compatible with a HE environment. Last, in every scenario different values for $m$ and $p$ are tested, to give a practical experience of what are the consequences of a bad setup. Section 6.1 describes the CNNs provided by the deep-learning-as-service, while Section 6.2 details the considered datasets. Accuracy and computational load on both recall and transfer learning modality are shown in Sections 6.3, 6.4 and 6.5.

6.1 Description of the CNNs

6-layers CNN

The first deep learning model is a 6-layers CNN. Its structure before the approximation phase is:

1. Convolutional layer: The input image has dimension $28 \times 28 \times 1$. The layer has $8 \times 3 \times 3$ filters with stride $1 \times 1$. The output of
this layer has then dimension $26 \times 26 \times 8$.

2. **Maximum pooling layer**: This layer has a window of $2 \times 2$, with stride $3 \times 3$. The output is then $9 \times 9 \times 8$.

3. **Convolutional layer**: The convolution has $16$ $3 \times 3$ filters, and stride $2 \times 2$. The output has dimension $4 \times 4 \times 16$.

4. **Maximum pooling layer**: This layer has a window of $2 \times 2$, with stride $1 \times 1$. The output is $2 \times 2 \times 16$.

5. **Fully connected layer**: This layer connects the incoming $2 \cdot 2 \cdot 16 = 64$ nodes to $16$ exit nodes.

6. **Fully connected layer**: This layer connects the incoming $16$ nodes to the final $10$ nodes.

This model presents two Maximum pooling layers. As explained in Chapter [4] it is impossible to perform this operation on encrypted data. In the approximated version, two Average pooling layers will be used instead of the Maximum pooling layers. The model, after this change, is trained again.

**5-layers CNN**

The second deep learning model is a **5-layers CNN**. Before the approximation phase, its structure is:

1. **Convolutional layer**: The input image has dimension $28 \times 28 \times 1$. The layer has $16$ $3 \times 3$ filters with stride $3 \times 3$. Thus, the output is $9 \times 9 \times 16$.

2. **ReLU activation layer**: This layer applies the ReLU function to each input node. The dimension of the batch is unchanged.

3. **Maximum pooling layer**: This layer has window $3 \times 3$, with stride $3 \times 3$. The output has dimension $3 \times 3 \times 16$.

4. **Fully connected layer**: This layer connects the incoming $16 \cdot 3 \cdot 3 = 144$ nodes to $72$ exit nodes.
5. **Fully connected layer**: This layer connects the incoming 72 nodes to the final 10 nodes.

This model presents a Maximum pooling layer and a ReLU activation layer. Similarly to the previous model, the Maximum pooling layer will be replaced with an Average pooling layer. For what concerns the ReLU activation layer, it will be replaced by a Square activation layer which simply squares every input node. After the substitution, the model is trained again.

### 6.2 Datasets

Two datasets have been considered in the analysis: the MNIST dataset and the FashionMNIST dataset. In particular, MNIST has been used in the transfer learning modality, while the FashionMNIST has been used on the recall modality.

#### 6.2.1 MNIST dataset

The MNIST database [3] (Modified National Institute of Standards and Technology database) is a database of handwritten digits, commonly used in the ML field to train and test models able to recognize images. It was created starting from the NIST database [53], and can be seen as an improvement of it. The problem with the original NIST database was that the training set was taken from American Census Bureau employees, while the testing set was taken from American high school students. This combination, as explained in [3], made it not well-suited for machine learning experiments. The contained images have been normalized to have dimension $28 \times 28$, with only the grayscale level. The dataset is composed of 70000 images: 60000 are training images, while the remaining 10000 are testing images. Images belong to 10 classes, each one representing a digit.

#### 6.2.2 FashionMNIST dataset

The FashionMNIST dataset [2] is a dataset of Zalando’s articles images. It has been designed to work as a direct drop-in replacement for
6.3 Recall

In this modality a user wants to use a CNN \( \varphi_{\Theta}(\cdot) \) published by a Cloud service provider, obtaining the classification \( y \) of an input image \( I \). To accomplish this, the two models presented in Section 6.1 have been trained on the FashionMNIST training dataset for 20 epochs, with a learning rate of 0.001. This has been done both on the plain version and on the approximated version of each model. Being the recall modality the one in which a user wants to recognize images in the domain of the images used for training the models, the testing part of the FashionMNIST will be used to test the accuracy of the models.

Figure 6.1a and 6.1b show the accuracy of the 6-layers CNN and the 5-layers CNN on the FashionMNIST dataset, respectively, using different values of \( \Theta \) (the parameter \( q \) has been omitted since automatically set). Table 6.1 and 6.2 collect the obtained experimental results.

For each considered encryption parameters \( \Theta_i \), three cases are compared: the plain CNN without approximations \( f(\cdot) \), the same plain CNN approximated to have only additions and multiplications \( \varphi(\cdot) \), and, finally, the encoded CNN with \( \Theta_i \), i.e., \( \varphi_{\Theta_i}(\cdot) \).

Taking into consideration the 6-layers CNN, the plain version \( f(\cdot) \)
6.3 Recall

CNN:  
- Plain $f(\cdot)$
- Plain Approximated $\varphi(\cdot)$
- Encoded $\varphi_\Theta(\cdot)$

![Graph](image-url)

(a) The results of the 6-layer CNN.

(b) The results of the 5-layer CNN.

Figure 6.1: The recall accuracy results of both the 6-layer CNN and the 5-layer CNN on the FashionMNIST dataset [2], with the standard deviation over five experiments.

reached an accuracy of 87.3%. After the approximation, needed to perform encrypted operations, the model $\varphi(\cdot)$ has its accuracy dropped to 80.7%. Different values for $m$ and $p$ are tested on the encoded model $\varphi_\Theta(\cdot)$. Going from $\Theta_1$ to $\Theta_4$, the values for $m$ and $p$ are gradually incremented.

The values which produce the best accuracy for this scenario are
the ones used in Θ₁. This is the case which can be understood more easily. Simply, the combination of \( m \) (4096) and \( p \) used makes it possible to have high values of NB on the fresh encrypted images, while maintaining an high accuracy on the executed operations. The results of every single operation has a very little error with respect to the same operation applied on plain data. Indeed, the testing using Θ₄ resulted in an accuracy of 80.7\%, making it as performant as the approximated model, used on plain data. The encryption has not introduced errors, only a computational overhead.

Θ₃ uses lower values for \( m \) (2048) and \( p \). Thus, the accuracy drops to a value of 77\%. This is because:

- Decreasing \( m \) reduces the initial amount of NB. Hence, smaller values of \( p \) have to be used or the NB will reach 0, making the results of the operations corrupted;

- Decreasing \( p \) in order to not exhaust all the NB lowers the precision of the computations.

Either way, the result is a lower accuracy. However, as it will be shown in Section 6.5, Θ₃ is way less demanding in terms of computational load with respect to Θ₄. This is due to the lower value of \( m \).

If the accuracy is sufficient for the goal the user has to accomplish, a trade-off can be made between accuracy and computational load; it can be said that Θ₄ produces results with higher accuracy, but the cost for the accuracy gain may be too high, and Θ₃ may be preferred.

For what concerns Θ₂ and Θ₁, it is fair to say that they are bad choices. \( m \) remains the same of Θ₃, while \( p \) is decreased. This means that a lot of NB is still present in the final results’ ciphertexts, and the accuracy of the computations is lower. It is noteworthy to point out that when using a very high value of \( p \) with a low value of \( m \), all the computations will run out the NB in the ciphertexts. The results will be completely random, and the accuracy will drop to that of a random classifier. Similar considerations can be made for the 5-layers CNN.

\footnote{Remember that higher values of \( p \) increment the accuracy of the result, but also the NB consumption.}
However, it is very important to note that the 6-layers CNN (Figure 6.1a), when used plain, is better than the 5-layers CNN (Figure 6.1b): in fact, the former has higher accuracy than the latter. However, after the approximation, the 5-layers CNN outperforms the 6-layers CNN (as can be seen comparing the first two rows of Table 6.1 and 6.2).

This suggests that if a CNN has to be used on encrypted data, the limitations introduced by HE should be taken into consideration at the very early stage of design, resulting in a $\varphi(\cdot)$ designed from scratch. Simply finding the best possible model for a domain on plain data and approximating it may lead to worse results.

### Accuracy 6-layers CNN

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plain model $f(\cdot)$</td>
<td>86.6%</td>
</tr>
<tr>
<td>Approximated model $\varphi(\cdot)$</td>
<td>80.7%</td>
</tr>
<tr>
<td>Encoded model $\varphi_{\theta_4}(\cdot)$</td>
<td>80.6%</td>
</tr>
<tr>
<td>Encoded model $\varphi_{\theta_3}(\cdot)$</td>
<td>77%</td>
</tr>
<tr>
<td>Encoded model $\varphi_{\theta_2}(\cdot)$</td>
<td>65.9%</td>
</tr>
<tr>
<td>Encoded model $\varphi_{\theta_1}(\cdot)$</td>
<td>63.5%</td>
</tr>
</tbody>
</table>

Table 6.1: The results, in terms of accuracy, of the 6-layers CNN model in recall modality.

### 6.4 Transfer learning

In this modality, the user relies on deep-learning-as-a-service $\varphi_{\theta}(\cdot)$ as a feature extractor to train a local classifier, as described in Section 4.3.2.

For this purpose, the 6-layers CNN pre-trained on the FashionMNIST dataset is used. Images coming from the MNIST dataset will be forwarded through the first 4 layers of such CNN, in order to extract...
Accuracy 5-layers CNN

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plain model $f(\cdot)$</td>
<td>84.1%</td>
</tr>
<tr>
<td>Approximated model $\varphi(\cdot)$</td>
<td>82.3%</td>
</tr>
<tr>
<td>Encoded model $\varphi_{\Theta_4}(\cdot)$</td>
<td>82.3%</td>
</tr>
<tr>
<td>Encoded model $\varphi_{\Theta_3}(\cdot)$</td>
<td>81%</td>
</tr>
<tr>
<td>Encoded model $\varphi_{\Theta_2}(\cdot)$</td>
<td>75.7%</td>
</tr>
<tr>
<td>Encoded model $\varphi_{\Theta_1}(\cdot)$</td>
<td>68.6%</td>
</tr>
</tbody>
</table>

Table 6.2: The results, in terms of accuracy, of the 5-layers CNN model in recall modality.

the features from them. Such features will be used to train two different classifiers: a SVM-based classifier and a Fully-Connected based classifier. In particular, 5000 images from the MNIST will be used to train the classifiers, while others 5000 will be used for testing.

Remember that, in this scenario, the fact that the model is trained on the FashionMNIST dataset while the classifier will be trained on the MNIST dataset is expected: even if trained on a different model, the convolutional layers of the CNN will be able to extract relevant features from the coming images, and such features will be used to train the classifiers.

Figure 6.2 shows the accuracy of the SVM-based classifier and Fully-Connected classifier. The full results are shown in Table 6.3 and 6.4 For each considered encryption parameters $\Theta_i$, four cases are compared: the plain CNN without approximations $f^{(4)}(\cdot)$ with a SVM-based classifier, the same plain CNN approximated to have only additions and multiplications $\varphi^{(4)}(\cdot)$ with the SVM, and, the encoded CNN with $\Theta_i$, i.e., $\varphi^{(4)}_{\Theta_i}(\cdot)$ with either a SVM or a Fully-Connected classifier.

The best result is obtained using the plain CNN $f^{(4)}(\cdot)$ with the SVM-based classifier. However, using the approximated version of the
6.4 Transfer learning

Figure 6.2: The transfer learning accuracy results on the features extracted at layer $l = 4$ of the 6-layer CNN to the MNIST dataset [3].

CNN $\varphi^{(4)}(\cdot)$ on plain data decreases the accuracy only of a small fraction: in particular, the accuracy drops from 93.1\% to 92.1\%. Then, different values for $\Theta$ show the impact on the precision of the extracted features, hence the accuracy of the trained classifiers. Similarly to the recall case, there are some values for $m$ and $p$ which make the error of the encrypted computation very low, thus producing results practically identical to the ones performed on plain data: in this case they are both $\Theta_3$ and $\Theta_4$, which produce a total final accuracy of 91.9\%.
However, moving from $\Theta_3$ to $\Theta_4$ (with a relevant increase in the parameter $m$) does not induce a significant improvement in the accuracy. This means that the value $p = 37780$ well characterizes the processing chain of $\varphi_\Theta(\cdot)$. Incrementing $m$ and $p$ to the ones used in $\Theta_4$ causes an increment on the computational load, but does not increment the accuracy of the results. It is fair to say that $\Theta_3$ is a better choice with respect to $\Theta_4$.

Another interesting point is that the values for $m$ and $p$ used in $\Theta_2$ for the 6-layers CNN in the recall scenario (Figure 6.1a) are equal to the ones used in $\Theta_4$ in this scenario. However, in the latter case, this set of parameters provides enough NB and precision to carry out the computations correctly, while in the former case it does not (there were values which produced better results). This can be explained by the fact that in this transfer learning scenario the number of encoded layers is lower than in the recall one. A smaller number of operations means that the error produced by each operation is propagated less times, producing final results which are overall more correct.

### Accuracy 6-layers CNN with transfer learning (SVM)

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plain model $f(\cdot) + SVM$</td>
<td>93.1%</td>
</tr>
<tr>
<td>Approximated model $\varphi(\cdot) + SVM$</td>
<td>92.1%</td>
</tr>
<tr>
<td>Encoded model $\varphi_{\Theta_4}(\cdot) + SVM$</td>
<td>91.9%</td>
</tr>
<tr>
<td>Encoded model $\varphi_{\Theta_3}(\cdot) + SVM$</td>
<td>91.8%</td>
</tr>
<tr>
<td>Encoded model $\varphi_{\Theta_2}(\cdot) + SVM$</td>
<td>86.3%</td>
</tr>
<tr>
<td>Encoded model $\varphi_{\Theta_1}(\cdot) + SVM$</td>
<td>45%</td>
</tr>
</tbody>
</table>

Table 6.3: The results, in terms of accuracy, of the transfer learning modality using a local SVM-based classifier.
6.5 Timing

In addition to the accuracy, the performance of the proposed PyCrCNN implementation has been tested by measuring the computational times on the client and the server-side and by estimating the transmission times of the data exchange. For this purpose, two tests have been made. In the first, a single image from the FashionMNIST dataset has been used in the recall modality. The two models, with the same $\Theta$s used in the previous tests, classified the image. In the second test a single image from the MNIST dataset has been classified using the transfer learning modality. Also in this case, the same $\Theta$s used in the previous tests have been used also for this one.

The tests have been made using a single-threaded computation. PyCrCNN supports also a multi-threading computation; if more cores and RAM are available, more images can be processed simultaneously.

The steps needed by the client to perform this operation are, in order:

1. Generate the encryption keys

2. Encrypt the tensor to be processed

### Table 6.4: The results, in terms of accuracy, of the transfer learning modality using a local fully-connected based classifier.

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plain model $f(\cdot) + \text{SVM}$</td>
<td>93.1%</td>
</tr>
<tr>
<td>Approximated model $\varphi(\cdot) + \text{SVM}$</td>
<td>92.1%</td>
</tr>
<tr>
<td>Encoded model $\varphi_{\Theta_4}(\cdot) + \text{FC}$</td>
<td>81.1%</td>
</tr>
<tr>
<td>Encoded model $\varphi_{\Theta_3}(\cdot) + \text{FC}$</td>
<td>81.1%</td>
</tr>
<tr>
<td>Encoded model $\varphi_{\Theta_2}(\cdot) + \text{FC}$</td>
<td>76%</td>
</tr>
<tr>
<td>Encoded model $\varphi_{\Theta_1}(\cdot) + \text{FC}$</td>
<td>36.5%</td>
</tr>
</tbody>
</table>
3. Correctly process the request payload and transmit it
4. Wait for the response
5. Decrypt the batch when the response is received

While the server will have to:

1. Process the request payload into encrypted tensors
2. Encode the model with the requested encryption parameters
3. Pass the data through the model
4. Process the answer payload and transmit it

The transfer time is not negligible. Unfortunately, the ciphertexts are way bigger than the plaintexts, at least with the current HE schemes.

The experimental results about the computational time are shown in Table 6.5 where

- $t_c$ is the time spent on the client to generate the keys couple $(k_p, k_s)$, to execute the encryption function $E(I, \Theta, k_p)$ and the decryption function $D(\hat{y}, \Theta, k_s)$. The machine used for the client is equipped with a 2.30GHz 64-bit dual-core processor and 8192 MB of RAM.

- $t_s$ is the time spent by the server to encode the model $\varphi(\cdot)$ and process the encrypted image, $\varphi_{\Theta}(\hat{I})$. The machine used as a server is an Amazon EC2 instance with 72 64-bit cores at 3.6GHz and 144 GIB of RAM.

- $t_t$ estimates the transmission times of sending the encrypted image $\hat{I}$ and receiving back the encrypted result $\hat{y}$. For the transmission part we modeled an high-bandwidth scenario, where we employ the transmission technology Wi-Fi 4 (standard IEEE 802.11n) using a single-antenna with 64-QAM modulation on the 20 MHz channel with data-rate $\rho = 72.2 Mb/s$.

Two main comments arise. First, as expected, all the three component of the computational times increase with $m$. More specifically, $t_c$ and
### 6.5 Timing

![Image of a table with data]

<table>
<thead>
<tr>
<th></th>
<th>$t_c$</th>
<th>$t_t$</th>
<th>$t_s$</th>
<th>$t = t_c + t_t + t_s$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Recall 6-layers CNN</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\Theta_1$</td>
<td>2.2 ± 0.2</td>
<td>3.7 ± 0.0</td>
<td>11.8 ± 0.1</td>
<td>17.7 ± 0.3</td>
</tr>
<tr>
<td>$\Theta_2$</td>
<td>2.2 ± 0.1</td>
<td>3.7 ± 0.0</td>
<td>11.9 ± 0.1</td>
<td>17.8 ± 0.2</td>
</tr>
<tr>
<td>$\Theta_3$</td>
<td>2.1 ± 0.1</td>
<td>3.7 ± 0.0</td>
<td>11.9 ± 0.0</td>
<td>17.7 ± 0.1</td>
</tr>
<tr>
<td>$\Theta_4$</td>
<td>4.7 ± 0.3</td>
<td>14.7 ± 0.0</td>
<td>49.7 ± 0.5</td>
<td>69.1 ± 0.8</td>
</tr>
<tr>
<td><strong>Recall 5-layers CNN</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\Theta_1$</td>
<td>5.2 ± 0.0</td>
<td>14.7 ± 0.0</td>
<td>26.2 ± 0.3</td>
<td>46.1 ± 0.3</td>
</tr>
<tr>
<td>$\Theta_2$</td>
<td>5.2 ± 0.0</td>
<td>14.7 ± 0.0</td>
<td>26.1 ± 0.1</td>
<td>46.0 ± 0.1</td>
</tr>
<tr>
<td>$\Theta_3$</td>
<td>5.2 ± 0.0</td>
<td>14.7 ± 0.0</td>
<td>25.8 ± 0.1</td>
<td>45.7 ± 0.1</td>
</tr>
<tr>
<td>$\Theta_4$</td>
<td>5.2 ± 0.0</td>
<td>14.7 ± 0.0</td>
<td>25.8 ± 0.1</td>
<td>45.7 ± 0.1</td>
</tr>
<tr>
<td><strong>Transfer Learning</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\Theta_1$</td>
<td>1.2 ± 0.0</td>
<td>2.0 ± 0.0</td>
<td>5.5 ± 0.0</td>
<td>8.7 ± 0.0</td>
</tr>
<tr>
<td>$\Theta_2$</td>
<td>2.4 ± 0.1</td>
<td>3.9 ± 0.0</td>
<td>11.6 ± 0.1</td>
<td>17.9 ± 0.2</td>
</tr>
<tr>
<td>$\Theta_3$</td>
<td>2.4 ± 0.0</td>
<td>3.9 ± 0.0</td>
<td>11.5 ± 0.0</td>
<td>17.8 ± 0.0</td>
</tr>
<tr>
<td>$\Theta_4$</td>
<td>2.4 ± 0.0</td>
<td>3.9 ± 0.0</td>
<td>11.5 ± 0.0</td>
<td>17.8 ± 0.0</td>
</tr>
</tbody>
</table>

Table 6.5: The three described configurations results, with a common PC as client and an Amazon EC2 instance as server. The main result, $t$, is the time requested to process an image for each scenario. The three components of $t$ are $t_c$, the time requested for the local encryption/decryption, $t_t$, the time for the data transfer and $t_s$, the time requested for the processing on the Cloud. The proposed values are expressed in seconds.
Experimental results

$t_s$ increase due to the larger computational load required to process encrypted data with larger $m$, while $t_t$ increases due to the increase of the size of the ciphertexts. In addition, $t_c$ is always lower than $t_s$ since $E(I, \Theta, k_p)$ and $D(\hat{y}, \Theta, k_s)$ are less computational demanding than $\varphi_\Theta(I)$. This is important when it is necessary to choose between two values for $m$. As seen in Section 6.3, not always incrementing $m$ leads to a much better result in terms of accuracy. However, the computational overhead can be important: in this case $t$ for $\Theta_4$ is 4 times higher than the $t$ for $\Theta_3$.

Second, an increase in $p$ does not result in a variation of the computational times $t_s$. All in all, $p$ should be tuned focusing on the accuracy of the results with an eye on the NB consumption, while $m$ must be tuned by trading-off the starting NB amount and computational load.

It is also interesting to compare the two models used in the recall modality. First of all, in general the 6-layers CNN can work with lower values of $m$ and $p$, thus requiring a lower computational load if 2048 is used as $m$. This does not apply to the 5-layers CNN, which can not obtain relevant result if values lower than 4096 are used for $m$. This can be explained by the fact that this model contains a Square activation layer, which consumes a large amount of NB. As discussed in Section 6.3, the 5-layers CNN obtained better results in terms of accuracy on the testing set (on encrypted data). However, the models can also be compared from the point of view of the computational load. As showed in Table 6.6, the best overall combination is the 5-

<table>
<thead>
<tr>
<th>Combined ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
</tr>
<tr>
<td>5-layers CNN, $\Theta_4$</td>
</tr>
<tr>
<td>6-layers CNN, $\Theta_4$</td>
</tr>
<tr>
<td>6-layers CNN, $\Theta_3$</td>
</tr>
</tbody>
</table>

Table 6.6: The ranking of the best combination of models/encryption parameters on the FashionMNIST dataset, including also the timing (expressed in seconds).
layers CNN using $\Theta_4$. Not only it achieves the higher accuracy, but has also a lower computational load with respect to the 6-layers CNN in combination with $\Theta_4$. It should be taken into consideration, though, that the 6-layers CNN in combination with $\Theta_3$, while not achieving the higher accuracy, has a considerable lower computational load. The use of this combination may be preferred if a loss in accuracy is tolerable, considering the computational load decrease.
Chapter 7

Conclusions

In this Chapter conclusions regarding the proposed architecture are drawn (Section 7.1), and some ideas for future works are presented (Section 7.2).

7.1 Conclusions

The aim of this work was to propose a novel privacy-preserving distributed architecture capable of executing algorithms of Deep Learning. In particular, the architecture can be used as a deep-learning-as-a-service. The peculiarity with respect to a classic DLaaS solution is the use of Homomorphic Encryption, which makes it possible to make strong guarantees on the confidentiality of processed data.

Convolutional Neural Networks applied to image recognition are the reference ML models used in this work. The process of adapting plain models to work with HE involves the modification of some layers, the use of different activation functions and a possible loss in accuracy. While in this work this process has been done for CNNs, it may be of inspiration for those who may want to modify other types of ML models to work with HE.

Along with the formulation of the architecture, a Python implementation, PyCrCNN, is provided to the scientific community as an open-source project. The library, composed of a client and a server, is a functioning example of the proposed architecture; it has been tested using Amazon AWS, one of the leaders in the field of Cloud Computing.
A series of experiments has been conducted in order to provide the reader some real-worlds metrics about the computational overhead of a privacy-preserving solution for DLaaS, unavoidable given the current state-of-art of HE. In particular, two different CNNs with different structure have been trained specifically for this work: the experiments show the loss in accuracy and the computational overhead for a wide set of different encryption parameters.

All in all, the goal of this thesis is to reaffirm that privacy problems related to technology are a serious issue in nowadays society. While paradigms like Cloud Computing and MLaaS are a powerful resource to deal with difficult problems, capable of offering excellent services even to end-users, a particular attention must be given to data and to the risks related to an abuse of that.

### 7.2 Future Works

The use of Homomorphic Encryption to mitigate privacy problems may gain a huge traction in the following years. The first possible extension of this work is related to encryption parameters choice. As seen in Chapter 4 and 6, a bad choice of encryption parameters can lead to poor results in terms of accuracy. Computational times are also affected. An algorithm able to automatically set the best possible parameters in a reasonable amount of time may mitigate this issue, and lower the difficult for users and providers in the adoption of HE. This solution may take as input a series of measures about the required task: for example the requested security level, the maximum tolerance in accuracy loss, etc. The result would be a set of parameters optimal to carry out that task in the minimum amount of time, while respecting the constraints set by the user and/or the provider.

The second possible extension is related to the use of different DL models. In particular, Recurrent Neural Networks may be a good candidate for the test with HE solutions. This class of DL model offers excellent performance in the field of handwriting recognition, speech-to-text recognition but also speech synthesis. The nature of some activation functions used in such models makes the use of HE
difficult: the logistic function can not be executed directly in the HE space, and will have to be approximated.

Lastly, a third extension may involve an optimized client implementation for Internet-of-Things devices, which are characterized by constraints on computation power and memory.
Bibliography


[21] Microsoft SEAL. [https://github.com/Microsoft/SEAL](https://github.com/Microsoft/SEAL) Microsoft Research, Redmond, WA.


Appendix A

IJCNN Paper

This appendix includes the complete text of the paper published at “2020 International Joint Conference on Neural Networks (IJCNN), Glasgow, 2020”, as attachment. It is based on the same solution proposed in this thesis.
A Privacy-Preserving Distributed Architecture for Deep-Learning-as-a-Service

Simone Disabato, Alessandro Falcetta, Alessio Mongelluzzo, and Manuel Roveri
Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Milano, Italy
Email: alessandro.falcetta@mail.polimi.it, {simone.disabato,alessio.mongelluzzo,manuel.roveri}@polimi.it

Abstract—Deep-learning-as-a-service is a novel and promising computing paradigm aiming at providing machine/deep learning solutions and mechanisms through Cloud-based computing infrastructures. Thanks to its ability to remotely execute and train deep learning models (that typically require high computational loads and memory occupation), such an approach guarantees high performance, scalability, and availability. Unfortunately, such an approach requires to send information to be processed (e.g., signals, images, positions, sounds, videos) to the Cloud, hence having potentially catastrophic-impacts on the privacy of users. This paper introduces a novel distributed architecture for deep-learning-as-a-service that is able to preserve the user sensitive data while providing Cloud-based machine and deep learning services. The proposed architecture, which relies on Homomorphic Encryption that is able to perform operations on encrypted data, has been tailored for Convolutional Neural Networks (CNNs) in the domain of image analysis and implemented through a client-server REST-based approach. Experimental results show the effectiveness of the proposed architecture.

I. INTRODUCTION

In recent years, the technological evolution of Cloud-based computing infrastructures intercepted the ever-growing demand of machine and deep-learning solutions leading to the novel paradigms of machine and deep-learning-as-a-service [1]. The core of such computing paradigms is that Cloud providers provide ready-to-use remotely-executable machine/deep learning services in addition to virtual computing environments (as in infrastructure-as-a-service) or platform-based solutions (as in platforms-as-a-service). Examples of such services are the identification of faces in images or videos or the conversion of text-to-speech or speech-to-text [2]. From the perspective of the user, being ready-to-use, these services do not require the training of the models (that are pre-trained by the Cloud provider) nor the local recall of such models (that are executed on the Cloud). Moreover, the Cloud-based computing infrastructure providing such machine/deep learning solutions as-a-service allows to support scalability, availability, maintainability, and pay-per-use billing mechanisms [3].

Unfortunately, to be effective, such an approach involves the processing of data that might be sensitive, e.g., personal pictures or videos, medical diagnoses, as well as data that might reveal ethnic origin, political opinions, but also genetic, biometric and health data [4].

The aim of this paper is to introduce a novel distributed architecture meant to preserve the privacy of user data in the deep-learning-as-a-service computing scenario. To achieve this goal, the proposed architecture relies on Homomorphic Encryption (HE) that is an encryption scheme allowing the process of encrypted data [5]. In the proposed architecture, by exploiting the properties of HE, users can locally encrypt their data through a public key, send them to a suitably-encoded Cloud-based deep-learning service (provided through the deep-learning-as-a-service approach), and receive back the encrypted results of the computation that are locally decrypted through the private key. More specifically, such architecture allows to decouple the encryption/decryption phases, which are carried out on the device of the user (e.g., a personal computer or a mobile device), from the deep-learning processing, which is carried out on the Cloud-based computing infrastructure. Such a HE-based distributed architecture allows to preserve the privacy of data (plain data are never sent to the Cloud provider) while guaranteeing scalability, availability, and high performance provided by Cloud-based solution.

The ability to process encrypted data of HE comes at two main drawbacks. First, the computational load and the memory demand of HE-encoded operations is much higher than regular ones, hence making the HE-encoded deep-learning processing highly demanding in terms of computation and memory. This is the reason why we focused on a deep-learning-as-a-service approach where the computation is carried out on high performing units on the Cloud. Second, HE supports only a limited set of operations (typically sums and multiplications). For this reason, prior to the encoding provided by the HE scheme, the deep-learning models have to be redesigned and retrained taking into account the constraints on the set of available operations. In addition, HE schemes have to be configured through some parameters that trade-off the accuracy in the computation with the computational loads and memory occupation. Such a configuration, that depends on the processing chain and the data to be processed, is managed at the Cloud-level by providing different settings of parameters that can be explored by the user.

The proposed architecture is intended to work with any machine and/or deep learning solution. However, in this work, it has been tailored to image analysis solutions leveraging Convolutional Neural Networks (CNNs) [6], and implemented through a client (locally executed on the user device) developed as a Python library and a server developed as a deep-learning-as-a-service container implemented on Amazon AWS. The developed architecture relies on a Representative
state transfer (REST) paradigm for exchanging encrypted data and results between client and server, while messages rely on JSON format.

A wide experimental campaign shows the feasibility and evaluates the performance of the proposed architecture. The Python Library for the client and the Amazon AWS Container are made available to the scientific community.1

The paper is organized as follows. Section II introduces a background on HE, while Section III describes the related literature. The proposed architecture is detailed in Section IV, while the technological implementation is described in Section V. Experimental results are described in Section VI and conclusions are finally drawn in Section VII.

II. BACKGROUND

The homomorphic scheme encryption is a special type of encryption that allows (a set of) operations to be performed on encrypted data, i.e., directly on the ciphertexts. More specifically, as detailed in [7], an encryption function $E$ and its decryption function $D$ are homomorphic w.r.t. a class of functions $F$ if, for any function $f \in F$, we can construct a function $g$ such that $f(x) = D(g(E(x)))$ for a set of input $x$.

The HE scheme considered in this paper is the Brakerski/Fan-Vercauteren (BFV) scheme [8] that, similarly to other works [9], [10], is based on the Ring-Learning With Errors (RLWE) problem. While a detailed description of such a problem and its security/implementation aspects can be found in [11], we here provide a brief introduction to the main concepts. The BFV scheme relies on the following set of encryption parameters (from now on denoted with $\Theta$):

- $m$: Polynomial modulus degree,
- $p$: Plaintext modulus, and
- $q$: Ciphertext coefficient modulus.

The parameter $m$ must be a positive power of 2 and represents the degree of the cyclotomic polynomial $\Phi_m(x)$. The plaintext modulus $p$ is a positive integer that represents the module of the coefficients of the polynomial ring $\mathbb{Z}_p = \mathbb{Z}_p[x]/\Phi_m(x)$ (onto which the RLWE problem is based). Finally, the parameter $q$ is a large positive integer resulting from the product of distinct prime numbers and represents the modulo of the coefficients of the polynomial ring in the ciphertext space. A crucial concept of a HE scheme is the Noise Budget (NB) that is an indicator related to the number of operations that can be done on a ciphertext while guaranteeing the correctness of the result. This problem (i.e., the maximum number of operations on the ciphertext) comes from the fact that, during the encryption phase, noise is added to the ciphertexts to guarantee that, being $p_1 = p_2$ two plain values to be encrypted with the same public key, the corresponding ciphertexts $c_1$ and $c_2$ are different (i.e., $c_1 \neq c_2$). All the operations performed on the ciphertext consume a certain amount of NB (depending on the type of operation and the input): operations like additions and multiplications between ciphertext and plaintext consume a small amount of NB, while multiplications between ciphertexts are particularly demanding in terms of NB. When the NB decreases to 0, decrypting that ciphertext will produce an incorrect result.

From a practical point of view, the choice of the encryption parameters $\Theta$ determines several aspects: the initial value of the NB, its consumption during computations (hence the number of operations to be performed on a ciphertext), the level of security against ciphertext attacks, the computational load and memory occupation of the HE processing and the accuracy of the results (i.e., measuring the correctness of the decrypted values). For example, the initial NB increases with $m$ at the expense of larger memory occupation and computational loads. The plaintext modulus $p$ is directly related to the accuracy of the HE processing. Despite being a very difficult parameter to be tuned, the theory states that larger values of $p$ will produce more accurate results at the expense of larger reductions of the NB. Finally, the parameter $q$ influences both the initial NB and the level of security of the encryption. A detailed description of the parameters and their effect on the HE scheme can be found in [12].

We emphasize that choosing the best parameter configuration is a trade-off between accuracy and performance and depends on the type and complexity of the processing, the set of feasible operations and the available computational resources. Practical guidelines to choose $\Theta$ will be given in Section IV-D.

III. RELATED LITERATURE

The idea of using HE to preserve the privacy of data during the computation has been introduced in [13]. In this work, privacy homomorphisms are defined as encryption functions that allow one to operate on encrypted data without preliminarily decrypting the operands [13]. The first HE schemes allow only additions [14], [15], [16], or multiplications [17].

The first homomorphic encryption scheme allowing both multiplication and additions has been proposed in [18]. There, the idea was to rely on ideal lattice-based cryptography to provide a scheme supporting additions and multiplications with theoretically-grounded security guarantees. After that, [19] extended this work by relaxing the ideal lattice assumption (and its security), but allowing the usage of integer polynomial rings to define the ciphertexts. [10] introduces the Brakerski-Gentry-Vaikuntanathan (BGV) scheme that relies on polynomial rings to define the ciphertexts and on the learning with error (LWE) and ring learning with errors (RLWE) problems to provide theoretically-grounded security guarantees. The RLWE problem is also the basis of the Brakerski/Fan-Vercauteren (BFV) scheme [8], detailed in Section II, and the Cheon-Kim-Kim-Song (CKKS) scheme [9], that extends the polynomial rings to the complex numbers and isometric rings.

The HE schemes mentioned above are theoretical and, to be applied, are then implemented to specific processing chains. As regards deep learning solutions, CryptoNets [20] relies on the HE BFV scheme to execute CNNs on encrypted

1 Code is available for download as a public repository at https://github.com/AlexMV12/PyCrCNN.git
inputs by introducing several possible ways of approximating the non-linear computation characterizing many layers of a CNN. Similarly, [21] provides a fast HE scheme for the (discretized) CNN inference. Recently, the nGraph-HE framework [22] has been proposed. This framework allows to train CNNs in plaintext on a given hardware and deploy trained models to HE cryptosystems operating on encrypted data. Unfortunately, these works are specific of a given DL solution (e.g., CNNs in [20]), whereas our architecture is meant to be general-purpose and able to hide the complexity of adopting HE solutions, similarly to what proposed in [22], still maintaining the as-a-service paradigm.

The literature presents also works aiming at offering encrypted computation. For example, [23] proposed the Secure Multi-Party Computation (SMC) approach, where more than one actor (namely, a party) collaborate in computing a function and having only partial knowledge of the data they are working on. These solutions do not encompass HE. [24] applied SMC with the Pailler HE [16] to CNNs, where a party owns the data and another owns the CNN. Hence, both the data and CNN are kept secret during the computation. Other examples can be found in [25], [26]. Finally, the Gazelle framework [27] relies on SMC and HE, to provide low-latency inference for CNN.

IV. THE PROPOSED ARCHITECTURE

The proposed privacy-preserving distributed architecture for deep-learning-as-a-service, called HE-DL, is shown in Figure 1. More specifically, HE-DL relies on a distributed approach where the Encryption $E(I, \Theta, k_p)$ of user data $I$ and the Decryption $D(\hat{y}, \Theta, k_s)$ of processed data $\varphi_\Theta(I)$ are carried out on the user device given the HE parameters $\Theta$ and with the public key $k_p$ and secret key $k_s$. Both $E(\cdot)$ and $D(\cdot)$ are based on HE-BFV scheme described in Section II.

Conversely, the deep learning processing $\varphi_\Theta(\cdot)$ is carried out in the Cloud. This is a crucial step since deep learning processing is typically highly demanding in terms of computational load and memory occupation. We emphasize that, as commented in Section II, the considered deep-learning-as-a-service computation has to be approximated by using only addition and multiplication in order to process the ciphertext $\hat{I}$. For this reason, the set of deep-learning models \textit{DL models} $f(\cdot)$s that are made available by the Cloud are approximated through addition and multiplication, i.e., defining the set of approximated \textit{DL models} $\varphi(\cdot)$s. Once approximated, $\varphi(\cdot)$s have to be encoded following the rule of the HE-BFV scheme to get the \textit{encoded deep-learning-as-a-service} $\varphi_\Theta(\cdot)$ by relying on the HE parameters $\Theta$. This encoding phase converts plain values parameters of \textit{DL models} in a form which can be computed by the HE-BFV scheme on encrypted inputs $\hat{I}$.

The \textit{DL models} considered in this work are the CNNs aiming at classifying the input images $I$ into a class $y \in Y$. In such a scenario the proposed HE-DL makes available the deep-learning-as-a-service computing paradigm into two different modalities:

- **recall**: the processing $\varphi_\Theta(\cdot)$ provides the encrypted version $\hat{y}$ of the final classification $y$ of $I$;
- **transfer learning**: the processing $\varphi_\Theta(\cdot)$ provides the encrypted version of a processing stage of the considered CNN applied to the input image $I$. The final classification $y$ is carried out on the User Device thanks to a suitably-trained classifier (e.g., a Support Vector Machine or a neural based classifier).

These two modalities will be detailed in the rest of the section, together with the description of the encryption/decryption phases, the approximation and encoding of CNNs, the configuration of the encryption parameters and the communication between user device and Cloud.

A. Encryption and Decryption

Let $P$ be a process generating images $I \in \mathbb{R}^{w \times h \times c}$ of height $h$, width $w$ and channels $c$ and let $\Theta = \{m, p, q\}$ be the array of encryptions parameters, as defined in Section II.

The encryption function $E(I, \Theta, k_p)$ transforms (based on the HE-BFV scheme) a plain image $I$ into an encrypted image $\hat{I}$ given the HE encryption parameters $\Theta$ with the support of a public key $k_p$. The decryption function $D(\hat{y}, \Theta, k_s)$ operates on the encrypted output $\hat{y}$ of the computation $\varphi_\Theta(I)$, being $\hat{I}$ the encrypted image. More specifically, $D(\hat{y}, \Theta, k_s)$ computes the plain output $y$ given the same set of parameters $\Theta$ and the secret key $k_s$ (corresponding to $k_p$). The semantic of $y$ depends on the considered working modality of \textit{HE-DL}:

- $y$ is the classification label of the input image $I$ in the \textit{recall} modality;
- $y$ is an array of extracted features representing the values of the activation function of a given layer of the CNN in the \textit{transfer learning} modality.

B. Approximated and encoded DL processing

We emphasize that the proposed architecture \textit{HE-DL} is general enough to employ a wide range of machine/deep learning models. In this paper, we decided to focus on CNNs for two main reasons. First, CNNs are widely-used and very-effective solutions for image classifications. Second, for most of their processing, CNNs are composed of addition and multiplication operations making them suitable candidates to be considered within a HE scheme.

Let $f(I)$ be a CNN composed of $L$ layers $\eta^{(l)}$ with parameters $\theta_l$ and $l = 1, \ldots, L$, aimed at extracting features and providing the classification output $y$ of an input image $I$. The general architecture of $f(I)$ is shown in Figure 2a.

As mentioned above, in order to be used with HE, CNNs have to be approximated by considering only computing layers and activation functions that are suitable for the considered HE-BFV scheme. Given that only addition and multiplication are permitted, only polynomials functions can be computed directly, while non-polynomials operations must be either approximated with a polynomial form or replaced with other (and permitted) types of operations. For instance, the ReLU activation function is a non-polynomial operation, hence it cannot be considered in the HE scenario. Similarly to what done in [20], in the proposed \textit{HE-DL} architecture, we define
the approximated CNN model $\varphi(\cdot)$ of the original CNN $f(\cdot)$ by considering the following rules:

- the ReLU activation function is replaced with a Square activation function that simply squares the input value;
- the max-Pooling operator is replaced with the average one, with the division converted to a multiplication by $\frac{1}{F}$, where $F$ is the pooling size (fixed and a-priori known);
- Approximate the other non-polynomial layers as in [20].

The result of this approximation is a CNN $\varphi(\cdot)$ whose processing layers $\varphi_l$ can be encoded with the considered HE-BFV scheme. To simplify the notation, the parameters of each layer $\theta_l$ or $\varphi_l$ are omitted from now on. It is important to note that, after performing the replacement of the non-polynomial layers, the model has to be trained again. This is necessary because the weights of the plain model are not valid anymore if the activation functions or other layers have been replaced by different ones. Hence, to provide a deep-learning-as-a-service, $\varphi(\cdot)$ must be retrained with the same settings in which the plain one was trained (e.g., same dataset, same learning algorithm, etc.). Obviously, if the original model $f(\cdot)$ already contains HE-compatible processing layers, this procedure is not necessary. Moreover, it’s noteworthy that this approximation process can introduce a variation in the accuracy between $f(\cdot)$ and $\varphi(\cdot)$. This aspect will be explored in the experimental section described in Section VI.

We emphasize that we considered (and made available to the scientific community) two already approximated and trained models, i.e., a 5-layers CNN and a 6-layers CNN trained on the FashionMNIST data-set [28]; these models will be used in the experimental section.

To work with the encrypted images $\tilde{I}s$, the suitably approximated CNN $\varphi(\cdot)$ must be encoded with the parameters $\Theta$ as defined by the HE-BFV scheme leading to the encoded CNN $\varphi_\Theta(\cdot)$. As shown in Figure 2b, the HE-based encrypted processing can be formalized as follows:

$$y = D(\tilde{y}, \Theta, k_s) = D(\varphi_\Theta(E(I, \Theta, k_p)), \Theta, k_s),$$

where $\tilde{y}$ represents the image $I$’s encrypted classification.

C. DL models: recall and transfer learning

As mentioned above, the deep-learning-as-a-service computing paradigm is made available in two different modalities, recall and transfer learning. The difference between the two modalities lies in how Eq. (1) is implemented. The former operates on the decrypted output $y$ of the CNN $\varphi$ last layer $L$ (typically a softmax on top of a classification layer), whereas the latter operates on the features $\tilde{l}_I$ extracted at a given CNN level $\tilde{l}$, with $1 \leq \tilde{l} < L$ (typically a convolutional or pooling one). The two modalities are detailed in what follows.

Recall: This is the modality where the user relies on one of the ready-to-use encoded CNN $\varphi_\Theta(\cdot)$s to classify the image $I$. More precisely, the user wants the image $I$ to be encrypted into $\tilde{I}$ and to be forwarded through all the layers of the encoded CNN $\Theta$, hence obtaining the final result $\tilde{y}$ of the classification task, without transmitting the image $I$ to the service provider. The assumption underlying this modality is that the chosen model $\varphi_\Theta(\cdot)$ is trained to classify images of the same domain of the input image $I$ (e.g., the model $\varphi_\Theta(\cdot)$ is trained to recognize the digits and $I$ is an image of a digit).

Transfer Learning: When the application problem of the user is not matched by the model $\varphi_\Theta(\cdot)$s (e.g., the user wants to distinguish between cars and bikes while available models have been trained to classify digits or faces), the transfer learning modality comes into play. In fact, following the transfer learning paradigm [29], [30], the processing of a pre-trained CNN can be split into two parts: feature extraction and classification. The feature extraction process represents a pre-trained feature extractor able to feed an ad-hoc classifier trained on the specific image classification problem (that can be different from the one originally used to train the CNN). This allows to use part of a pre-trained CNN and train only a final classifier (hence reducing the complexity for the training and the number of images required for the training).

In our scenario, the encrypted input images, $\tilde{I}s$, will be forwarded through the encoded model $\varphi_\Theta$ up to a layer $\tilde{L}$. More specifically, $\varphi_\Theta$ comprises layers from 1 to $\tilde{L}$, with $1 \leq \tilde{L} \leq L$, whereas all the (eventually) remaining layers, from $\tilde{L} + 1$ to the final one $L$ remain plain and operate on the decrypted output of layer $\tilde{l}$, i.e., $\tilde{l}_I = D(\varphi_\Theta(E(I, \Theta, k_p)), \Theta, k_s)$, where $\varphi_\Theta$ represents the encoded CNN up to layer $\tilde{l}$ with parameters $\Theta$. The output of the model will be, in this case, the features extracted from every image $I$. The user may use these features to train a local classifier (e.g., a Support Vector Machine); an example will be shown in section VI.

We emphasize that, following such an approach, the user is able to locally train a classifier on the decrypted vectors $y = D(\tilde{y}, \Theta, k_s)$, being $\tilde{y}$ the output of $\varphi_\Theta$. A set of $K$ images $\{I_1, \ldots, I_K\}$ is sent to $HE-DL$ providing the corresponding output $\{\hat{y}_1, \ldots, \hat{y}_K\}$ that are locally decrypted into...
\begin{enumerate}
\item The plain processing of an approximated CNN $\phi(\cdot)$ composed of $L$ layers. Each layer $\phi_l$, with $1 \leq l \leq L$ is here composed only of multiplications and/or additions. The difference w.r.t. the usual CNN-based classifier $f(\cdot)$ relies only in these approximations. Note that also the layers parameters $\theta_l$s, here omitted, may require to be approximated (and referred to as $\hat{\theta}_l$) in the approximated CNN $\phi(\cdot)$.
\item The encrypted processing of the CNN $\phi_{\Theta}(\cdot)$. The CNN is encoded with HE parameters $\Theta$, operates on images $\hat{I}$s with the same parameters $\Theta$ and returns the encrypted classification output $\hat{y}$.
\end{enumerate}

Fig. 2. A comparison of the plain and approximated CNN processing with the encrypted one. The layers’ parameters $\theta_l$s are omitted to simplify the notation.

\{y_1, \ldots, y_K\}. The vector set \{y_1, \ldots, y_K\} is used together with the corresponding labels (that are available to the user) to locally train a classifier. Once trained, the system is ready-to-use: the user can send an encrypted image $\hat{I}$ to the Cloud, receive the CNN output $\hat{y}$, decrypt it to $y$ and apply the classifier on $y$.

\subsection*{D. Encryption parameters}

As already mentioned, the choice of $\Theta$ is critical to get correct processing of the encrypted image $\hat{I}$. The choice for $q$ is particularly difficult and influences the security of the scheme. For this purpose SEAL library [31] provides a specific function that, given the polynomial modulus degree $m$ and the desired AES-equivalent security level (sec), returns a suggested value for $q$ [12]. Once trained, the system is ready-to-use: the user can send an encrypted image $\hat{I}$ to the Cloud, receive the CNN output $\hat{y}$, decrypt it to $y$ and apply the classifier on $y$.

\subsection*{E. Communication between User Device and Cloud}

The communication between the User Device and the Cloud is carried out through a JSON-format message. More specifically, being an on-demand computation, clients have to perform a request to the on-line deep-learning-as-service provider including:

- a set of parameters, including the encryption parameters $m$ and $p$, the security level sec, the identifier of the chosen DL model $\phi_{\Theta}(\cdot)$ to use in the computation, and the specific layers to use (which will determine the modality, i.e., recall or transfer learning);
- the encrypted image $\hat{I}$ on which the computation is performed, which has to be encrypted using a public key generated according to the encryption parameters.

Information about the available models will be published by the provider. $\hat{I}$ is transmitted as a vector in which the ciphertexts are encoded as base64 strings making it possible to embed them into JSON files. Once the computation has been carried out, the Cloud responds with a JSON message containing the encrypted result vector.

As an example, if the user wants to classify a batch of 20 images from the FashionMNIST using Model1, the JSON will contain $[m = 2048, p = 600201, sec = 128]$, the details of the models ("model=""Model1", "layers"=[0, 1, 2, 3, 4, 5, 6]) and the encrypted image (a vector with dimensions [20, 1, 28, 28]). The answer JSON message will contain the encrypted classification, so a vector of dimension [20, 10] of ciphertexts.

\section{V. Implementation}

The architecture introduced in the previous section has been implemented through a Python library, named PyCrCNN, comprising a client-side and a server application. PyCrCNN supports the encryption and decryption of batches of integer or float values and the application of the common layers used in CNNs like convolutional layers, average pool layers, and fully connected layers, relying on PyTorch library [32]. For the HE operations, PyCrCNN relies on the Pyfhel library v2.0.1 [33], Laurent (SAP) and Onen (EURECOM), licensed under the GNU GPL v3 license.

\subsection*{A. Client}

The client-side can encrypt the input images $I$s and decrypt the resulting answer $\hat{y}$ in a transparent way with respect to the user. Once the parameters are set (which include encryption parameters $\Theta$, name and layers of the chosen model $\phi_{\Theta}(\cdot)$, server URL and port), the client-side of PyCrCNN exposes a function which receives $I$ as a NumPy vector and returns $y$ as a NumPy vector; this makes it compliant with many machine learning frameworks for Python. Before starting the computation, a public and secret key pair $(k_p, k_s)$ is generated. The input batch is encrypted and encoded in base64 strings that will be included in the JSON payload along with the parameters $\Theta$ (as described in the previous section). To perform the request, the JSON payload is uploaded to an Amazon S3 bucket. Then, a POST request containing the address to the uploaded data is made to the deep-learning-as-a-service URL and, once the reply $\hat{y}$ is received, the resulting batch is downloaded from the bucket and decrypted using the key $k_s$ generated before. Finally, the user receives back the decrypted value $y$ as a NumPy array.

\textsuperscript{2}Pyfhel is a wrapper on the Microsoft SEAL library.
**B. Server**

The server side of the deep-learning-as-a-service must be invoked via web API. For this purpose, we relied on a set of Amazon Web Services (AWS) tools comprising Sagemaker, Elastic Container Registry (ECR), AWS Lambda, API Gateway, and S3. More precisely, we extended the built-in models offered by Sagemaker with our own custom algorithm, i.e., PyCrCNN, by creating a Docker container compliant with Sagemaker Docker Images specifications, uploading it to ECR and deploying the model on Sagemaker. The Docker container uses NginX as a web server, Gunicorn as a WSGI and Flask, a Python library, as a web framework to expose the APIs required by Sagemaker. With a mock fit method we load and store the model to S3; with the actual predict method the model performs the feature extraction task. Hence, the proposed deep-learning-as-a-service is made available through a REST API: the client invokes the endpoint URL with a POST request whose JSON payload contains the S3 path to the image \( \hat{I} \) encrypted by the client and the aforementioned encryption parameters \( \Theta \). In order to obtain a JSON-serializable payload, we encode the encrypted image \( \hat{I} \) as a base64 string. The client receives back the encrypted server response \( \hat{y} \) as a base64 string containing the extracted features.

**VI. EXPERIMENTAL RESULTS**

The aim of this section is to evaluate the accuracy and the computation load of the deep-learning-as-service provided through PyCrCNN both in recall and transfer learning modality. Section VI-A describes the CNNs provided by the deep-learning-as-service, while Section VI-B details the considered datasets. Accuracy and computational load on both recall and transfer learning modality are shown in Sections VI-C, VI-D and VI-E.

**A. Description of the CNNs**

The first deep learning model is a 6-layer CNN composed by the following processing layers: a convolutional layer with 8 3x3 filters, a 2x2 maximum pooling layer with stride 3, a convolutional layer with 16 3x3 filters and stride 2, a 2x2 maximum pooling layer and two fully-connected layers with 16 and 10 neurons respectively. The second deep learning model is a 5-layer CNN composed by a convolutional layer with 16 3x3 filters with stride 3 and a ReLU activation function, a 3x3 maximum pooling layer with stride 3 and two fully connected layers with 72 and 10 neurons respectively.

**B. Datasets**

Two datasets have been considered in the analysis:

- **MNIST** [35] is a datasets of handwritten digits composed of 70000 grey-scale 28x28 images, belonging to 10 classes. From the datasets, 5000 images were used for training and 5000 for validation.
- **FashionMNIST** [28] is a datasets of fashion products composed of 70000 grey-scale 28x28 images, belonging to 10 classes. From the datasets, 60000 images were used for training and 10000 for validation.

In particular, the FashionMNIST dataset has been considered in the recall modality, while MNIST has been used in the transfer learning one.

**C. Recall**

In this modality a user wants to use a deep-learning-as-a-service model \( \phi_\Theta(\cdot) \) published by a Cloud service provider, obtaining the classification \( y \) of an input image \( I \). Figure 3a and 3b show the accuracy of the 6-layers CNN and the 5-layers CNN on the FashionMNIST dataset, respectively, with respect to different values of \( \Theta \) (the parameter \( q \) has been omitted since automatically set). The two CNNs in both the configurations, plain and approximated, have been trained on the FashionMNIST training dataset for 20 epochs, with a learning rate of 0.001. As expected, the accuracy of the encoded model \( \varphi_\Theta(\cdot) \) increases with \( m \) and \( p \). In particular, the configuration of parameters \( \Theta_4 \) (characterized by the largest values of \( m \) and \( p \)) provides the same performance of the approximated...
TABLE I

THE THREE DESCRIBED CONFIGURATIONS RESULTS, WITH A COMMON PC AS A CLIENT AND AN AMAZON EC2 INSTANCE AS A SERVER. THE MAIN RESULT, \( t \), IS THE TIME REQUIRED TO PROCESS AN IMAGE FOR EACH SCENARIO. THE THREE COMPONENTS OF \( t \) ARE \( t_c \), THE TIME FOR THE LOCAL ENCRYPTION/DECRYPTION, \( t_t \), THE TIME FOR THE DATA TRANSFER AND \( t_s \), THE TIME REQUIRED FOR THE PROCESSING ON THE CLOUD. THE PROPOSED VALUES ARE EXPRESSED IN SECONDS.

<table>
<thead>
<tr>
<th>Encryption</th>
<th>( \Theta_1 )</th>
<th>( \Theta_2 )</th>
<th>( \Theta_3 )</th>
<th>( \Theta_4 )</th>
<th>( t = t_c + t_t + t_s )</th>
</tr>
</thead>
<tbody>
<tr>
<td>6-layers</td>
<td>2.2 ± 0.2</td>
<td>2.2 ± 0.1</td>
<td>2.1 ± 0.1</td>
<td>4.7 ± 0.3</td>
<td>11.8 ± 0.1</td>
</tr>
<tr>
<td>Recall CN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5-layers</td>
<td>5.2 ± 0.0</td>
<td>5.2 ± 0.0</td>
<td>5.2 ± 0.0</td>
<td>5.2 ± 0.0</td>
<td>26.2 ± 0.3</td>
</tr>
<tr>
<td>Recall CNN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transfer</td>
<td>1.2 ± 0.0</td>
<td>2.4 ± 0.1</td>
<td>2.4 ± 0.0</td>
<td>2.4 ± 0.0</td>
<td>5.5 ± 0.0</td>
</tr>
<tr>
<td>Learning</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

model \( \varphi(\cdot) \) operating on plain data. It is noteworthy to point out that the 6-layers CNN (Figure 3a), when used plain, is indeed better than the 5-layers CNN (Figure 3b): in fact, the former has higher accuracy than the latter. However, after the approximation, the 5-layers CNN outperforms the 6-layers CNN. This suggests that the approximated CNN \( \varphi(\cdot) \) could be designed from scratch.

D. Transfer learning

In this modality, the user relies on deep-learning-as-a-service \( \varphi_{\Theta}(\cdot) \) as a feature extractor to train a local classifier, as described in Section IV-C. Two types of classifiers have been used, i.e., an SVM-based classifier and a Fully-Connected based classifier. Both classifiers have been trained using the features extracted from images coming from the MNIST [35] dataset, using the first 4 layers of the pre-trained 6-layers CNN. In particular, 5000 images were used for the training of the classifiers and 5000 for the testing.

Figure 4 shows the accuracy of the SVM-based classifier and Fully-Connected classifier. Different values for \( \Theta \) show the impact on the precision of the extracted features, hence the accuracy of the trained classifiers. Here, two main comments arise. First, moving from \( \Theta_3 \) to \( \Theta_4 \) (with a relevant increase in the parameter \( p \)) does not induce a significant improvement in the accuracy. This means that the value \( p = 37780 \) well characterizes the processing chain of \( \varphi_{\Theta_3}(\cdot) \). Secondly, \( \Theta_3 \) for the 6-layers CNN in the recall scenario is equal to \( \Theta_1 \) in the transfer learning scenario. However, in the latter case, this set of parameters provides enough NB and precision to carry out the computations correctly, while in the former case it does not. This can be explained by the fact that in this transfer learning scenario the number of encoded layers is lower than in the recall one.

E. Timing

In addition to the accuracy, we evaluated the performance of the proposed PyCrCNN implementation by measuring the computational times on the client and the server-side and by estimating the transmission times of exchange information. For this purpose, we considered a single image taken from the FashionMNIST dataset for the recall modality and from the MNIST dataset for the transfer learning modality, in a single-threaded scenario. The models \( \varphi_{\Theta}(\cdot) \) have been encoded with the same \( \Theta \) used for the analysis of the accuracy described above.

The experimental results about the computational time are shown in Table I where

- \( t_c \) is the time spent by the server to encode the model \( \varphi(\cdot) \) and process the encrypted image, \( \hat{I} \). The machine used as a server is an Amazon EC2 instance with 72 64-bit cores at 3.6GHz and 144 GB of RAM.
- \( t_s \) estimates the transmission times of sending the encrypted image \( \hat{I} \) and receiving back the encrypted result \( \hat{y} \). For the transmission part we modeled an high-bandwidth scenario, where we employ the transmission technology Wi-Fi 4 (standard IEEE 802.11n) using a single-antenna with 64-QAM modulation on the 20 MHz channel with data-rate \( p = 72.2 \text{Mbps} \) [36].

Two main comments arise. First, as expected, all the three component of the computational times increase with \( m \). More specifically, \( t_c \) and \( t_s \) increase due to the larger computational load required to process encrypted data with larger \( m \), while
t₁ increases due to the increase of the size of the ciphertexts. In addition, tₑ is always lower than tₛ since E(I, Θ, kₚ) and D(γ, Θ, kₛ) are less computational demanding than φₑ(I).

Second, an increase in p does not result in a variation of the computational times tₛ. All in all, p should be tuned focusing on the accuracy of the results, while m must be tuned by trading-off accuracy and computational load.

VII. CONCLUSIONS

The aim of this paper was to introduce a novel privacy-preserving distributed architecture for deep-learning-as-service. The proposed architecture, which relies on Homomorphic Encryption, supports the Cloud-based processing of encrypted data to preserve the privacy of user data. The proposed architecture has been tailored to Convolutional Neural Networks and an implementation based on Python and Amazon AWS is made available. Experimental results show the effectiveness of what proposed.

Future work will consider the automatic configuration of the Homomorphic Encryption parameters, the extension of the deep learning models to deep recurrent neural networks and optimized client implementation for Internet-of-Things devices (characterized by constraints on computation and memory).

ACKNOWLEDGEMENT

This work has been partially supported by the project “GAUCHO” Project funded by MIUR under PRIN 2015.

REFERENCES
