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 Representational

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

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 $\mathcal{F}$ 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.

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 Θ):

- 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. 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 Θ 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 Θ 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 cyphertexts. [10] introduces the Brakerski-Gentry-Vaikuntanathan (BGV) scheme that relies on polynomial rings to define the cyphertexts 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, have then been 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* $E\left(I,\Theta,k_p\right)$ of user data I and the *Decryption* $D\left(\hat{y},\Theta,k_s\right)$ of processed data $\varphi_\Theta(\hat{I})$ are carried out on the user device given the HE parameters Θ and with the public key k_p and secret key k_s. Both $E\left(\cdot\right)$ and $D\left(\cdot\right)$ 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 *DL models* $f\left(\cdot\right)$s that are made available by the Cloud are approximated through addition and multiplication, i.e., defining the set of approximated *DL models* $\varphi\left(\cdot\right)$s. Once approximated, $\varphi\left(\cdot\right)$s have to be encoded following the rule of the HE-BFV scheme to get the *encoded deep-learning-as-a-service* $\varphi_\Theta(\cdot)$ by relying on the HE parameters Θ. This encoding phase converts plain values parameters of *DL models* in a form which can be computed by the HE-BFV scheme on encrypted inputs $\hat{I}$.

The *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 $\mathcal{P}$ 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 $E\left(I,\Theta,k_p\right)$ transforms (based on the HE-BFV scheme) a plain image I into an encrypted image $\hat{I}$ given the HE encryption parameters Θ 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 $\varphi_\Theta(\hat{I})$, 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 Θ and the secret key k_s (corresponding to k_p). The semantic of y depends on the considered working modality of *HE-DL*:

- y is the classification label of the input image I in the *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 *transfer learning* modality.

B. Approximated and encoded DL processing

We emphasize that the proposed architecture *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 $f\left(\mathcal{I}\right)$ be a CNN composed of L layers $\eta_{\theta_l}^{(l)}$ with parameters θ_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 $f\left(\mathcal{I}\right)$ 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 *HE-DL* architecture, we define

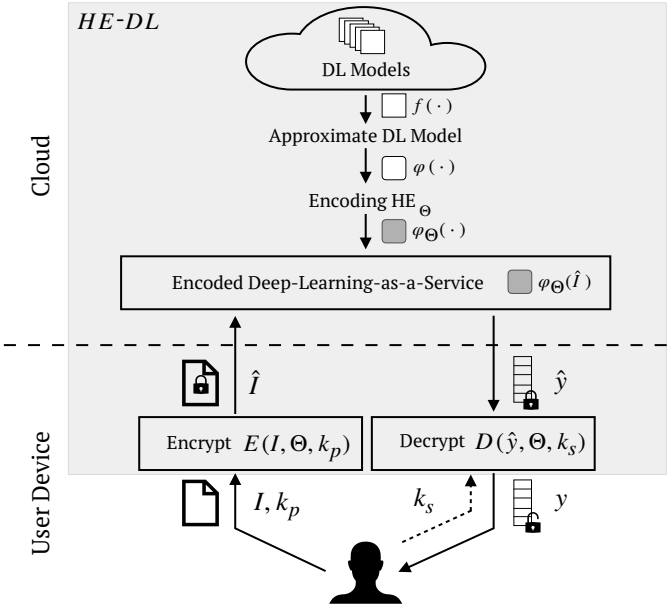


Fig. 1. The proposed privacy-preserving architecture for deep-learning-as-a-service.

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 $\frac{1}{f_s}$, where f_s 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 $\phi^{(l)}_{\hat{\theta}_l}$ can be encoded with the considered HE-BFV scheme. To simplify the notation, the parameters of each layer θ_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 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 $\hat{I}$s, the suitably approximated CNN φ must be encoded with the parameters Θ 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:

$$y = D\left(\hat{y}, \Theta, k_s\right) = D\left(\varphi_\Theta\left(E\left(I, \Theta, k_p\right)\right), \Theta, k_s\right), \quad (1)$$

where $\hat{y}$ represents the image I's encrypted classification.

C. DL models: recall and transfer learning

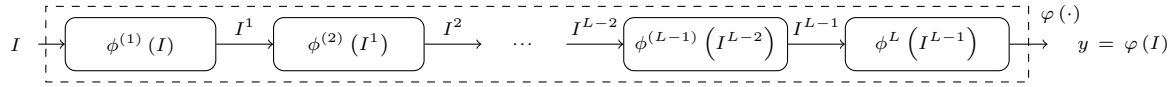
As mentioned above, the deep-learning-as-a-service computing paradigm is made available in two different modalities, *recall* and *transfer learning*. The difference between the two modalities lies in how Eq. (1) is implemented. The former operates on the decrypted output y of the CNN φ last layer L (typically a softmax on top of a classification layer), whereas the latter one works on the features $I^{\tilde{l}}$ 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 $\hat{I}$ and to be forwarded through all the layers of the encoded CNN φ_Θ, hence obtaining the final result $\hat{y}$ 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).

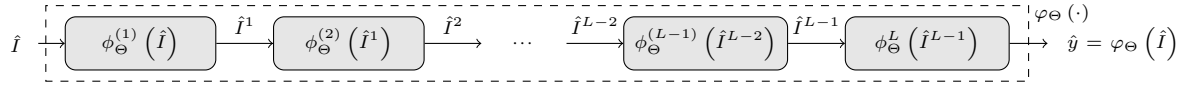
Transfer Learning: When the application problem of the user is not matched by the model $\varphi_\Theta(\cdot)$s (e.g., the user wants to distinguish between cars and bikes while available models have been trained to classify digits or faces), the transfer learning modality comes into play. In fact, 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 processing 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, $\hat{I}$s, will be forwarded through the encoded model φ_Θ up to a layer $\tilde{l}$. More specifically, φ_Θ 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., $I^{\tilde{l}} = D\left(\varphi^{\tilde{l}}_\Theta\left(E\left(I, \Theta, k_p\right)\right), \Theta, k_s\right)$, where $\varphi^{\tilde{l}}_\Theta$ represents the encoded CNN up to layer $\tilde{l}$ with parameters Θ. 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 $y = D\left(\hat{y}, \Theta, k_s\right)$, being $\hat{y}$ the output of φ_Θ. 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



(a) The plain processing of an approximated CNN $\varphi(\cdot)$ composed of L layers. Each layer ϕ^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 θ_ls, here omitted, may require to be approximated (and referred to as $\hat{\theta}_l$) in the approximated CNN $\varphi(\cdot)$.



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

Fig. 2. A comparison of the plain and approximated CNN processing with the encrypted one. The layers' parameters θ_ls 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.

D. Encryption parameters

As already mentioned, the choice of Θ 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]. In this work we considered *sec* equals to 128 *bits* which is the default value of SEAL. Hence, we relied on the SEAL function to automatically set the values of q to guarantee a 128 *bits* security level, while we selected $m \in \{1024, 2048, 4096\}$ and $p \in \{32, 712, 37780, 1.3 \cdot 10^5, 1.5 \cdot 10^5, 2.6 \cdot 10^5, 5.2 \cdot 10^5, 6.0 \cdot 10^5, 2.1 \cdot 10^6, 1.3 \cdot 10^8, 1.5 \cdot 10^8\}$, through an experimental analysis. The effects of the different choices for Θ are shown in Section VI.

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 $\varphi_{\Theta}(\cdot)$ 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.

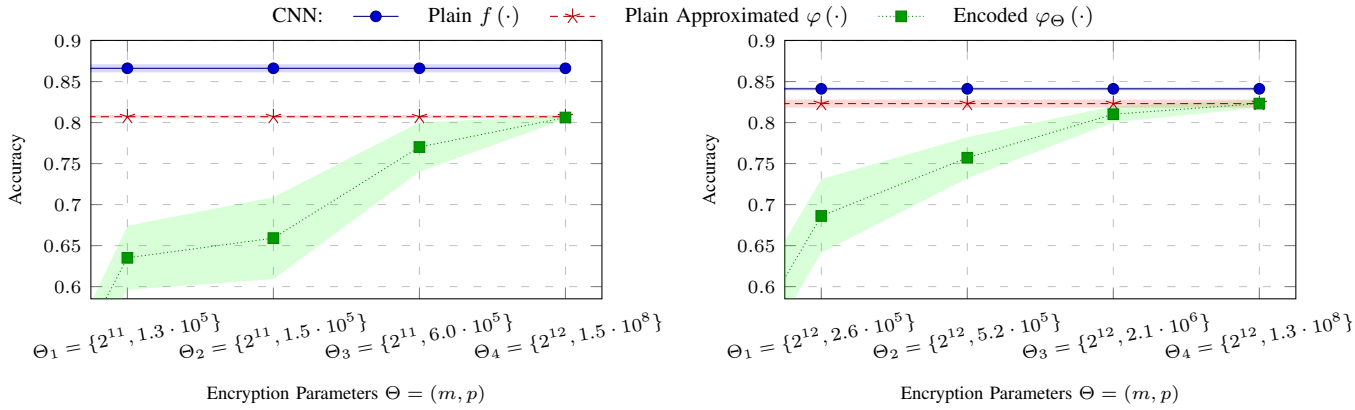
V. Implementation

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[2].

A. Client

The client-side can encrypt the input images Is 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 Θ, name and layers of the chosen model $\varphi(\cdot)$, server URL and port), the client-side of PyCrCNN exposes a function which receives I as a NumPy [34] 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 Θ (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.

[2]Pyfhel is a wrapper on the Microsoft SEAL library.



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

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

Fig. 3. The *recall* accuracy results of both the 6-layer CNN and the 5-layer CNN on the FashionMNIST dataset [28], with the standard deviation over five experiments. For each considered encryption parameters Θ_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 Θ_i, i.e., $\varphi_{\Theta_i}(\cdot)$. It is noteworthy to point out that with encryption parameters Θ_i *smaller* (i.e., smaller m and p) than those shown, the accuracy quickly drops to that of a random classifier.

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 Θ. 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 trough 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 $\varphi_\Theta(\cdot)$ 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 Θ (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 Θ_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 REQUIRED 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.

		t_c	t_t	t_s	$t = t_c + t_t + t_s$
Recall 6-layers CNN	Θ_1	2.2 ± 0.2	3.7 ± 0.0	11.8 ± 0.1	17.7 ± 0.3
	Θ_2	2.2 ± 0.1	3.7 ± 0.0	11.9 ± 0.1	17.8 ± 0.2
	Θ_3	2.1 ± 0.1	3.7 ± 0.0	11.9 ± 0.0	17.7 ± 0.1
	Θ_4	4.7 ± 0.3	14.7 ± 0.0	49.7 ± 0.5	69.1 ± 0.8
Recall 5-layers CNN	Θ_1	5.2 ± 0.0	14.7 ± 0.0	26.2 ± 0.3	46.1 ± 0.3
	Θ_2	5.2 ± 0.0	14.7 ± 0.0	26.1 ± 0.1	46.0 ± 0.1
	Θ_3	5.2 ± 0.0	14.7 ± 0.0	25.8 ± 0.1	45.7 ± 0.1
	Θ_4	5.2 ± 0.0	14.7 ± 0.0	25.8 ± 0.1	45.7 ± 0.1
Transfer Learning	Θ_1	1.2 ± 0.0	2.0 ± 0.0	5.5 ± 0.0	8.7 ± 0.0
	Θ_2	2.4 ± 0.1	3.9 ± 0.0	11.6 ± 0.1	17.9 ± 0.2
	Θ_3	2.4 ± 0.0	3.9 ± 0.0	11.5 ± 0.0	17.8 ± 0.0
	Θ_4	2.4 ± 0.0	3.9 ± 0.0	11.5 ± 0.0	17.8 ± 0.0

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 Θ 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 Θ_3 to Θ_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 $\varphi_\Theta(\cdot)$. Secondly, Θ_2 for the 6-layers CNN in the recall scenario is equal to Θ_4 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.

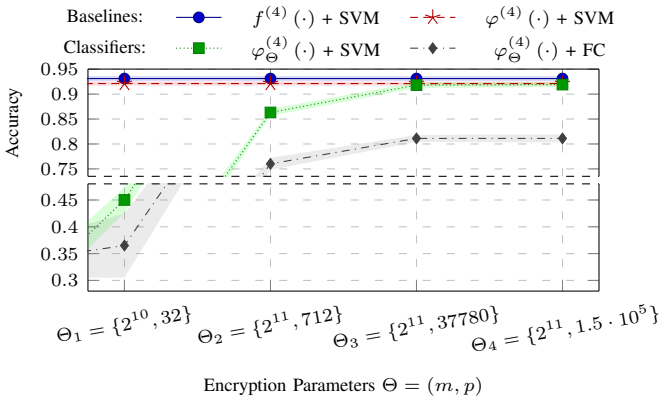


Fig. 4. The *transfer learning* accuracy results on the features extracted at layer $l = 4$ of the 6-layer CNN to the MNIST dataset [35]. For each considered encryption parameters Θ_i, four cases are compared: the plain CNN without approximations $f^{(4)}(\cdot)$ with a SVM-based classifier, the same plain CNN approximated to have only additions and multiplications $\varphi^{(4)}(\cdot)$ with the SVM, and, the encoded CNN with Θ_i, i.e., $\varphi^{(4)}_{\Theta_i}(\cdot)$ with either a SVM or a Fully-Connected classifier.

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 Θ 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 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 he data-rate $\rho = 72.2Mb/s$ [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

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 $\varphi_\Theta(\hat{I})$. 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

[1] Y. Yao, Z. Xiao, B. Wang, B. Viswanath *et al.*, "Complexity vs. performance: empirical analysis of machine learning as a service," in *Proceedings of the 2017 Internet Measurement Conference*. ACM, 2017, pp. 384–397.

[2] M. S. Hossain and G. Muhammad, "Cloud-assisted speech and face recognition framework for health monitoring," *Mobile Networks and Applications*, vol. 20, no. 3, pp. 391–399, 2015.

[3] T. Erl, R. Puttini, and Z. Mahmood, *Cloud computing: concepts, technology & architecture*. Pearson Education, 2013.

[4] E. P. Council of European Union, "Regulation (eu) no 2016/679, article 4(1)," 2016.

[5] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti, "A survey on homomorphic encryption schemes: Theory and implementation," *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, p. 79, 2018.

[6] Y. LeCun, Y. Bengio *et al.*, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[7] F. Boemer, Y. Lao, R. Cammarota, and C. Wierzynski, "ngraph-he: a graph compiler for deep learning on homomorphically encrypted data," in *Proceedings of the 16th ACM International Conference on Computing Frontiers*. ACM, 2019, pp. 3–13.

[8] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption." *IACR Cryptology ePrint Archive*, vol. 2012, p. 144, 2012.

[9] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2017, pp. 409–437.

[10] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," *ACM Transactions on Computation Theory (TOCT)*, vol. 6, no. 3, p. 13, 2014.

[11] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2010, pp. 1–23.

[12] K. Laine, "Simple encrypted arithmetic library 2.3.1," Microsoft Research, WA, USA, Tech. Rep., 2017.

[13] R. L. Rivest, L. Adleman, M. L. Dertouzos *et al.*, "On data banks and privacy homomorphisms," *Foundations of secure computation*, vol. 4, no. 11, pp. 169–180, 1978.

[14] D. Naccache and J. Stern, "A new public key cryptosystem based on higher residues," in *ACM Conference on Computer and Communications Security*. Citeseer, 1998, pp. 59–66.

[15] T. Okamoto and S. Uchiyama, "A new public-key cryptosystem as secure as factoring," in *International conference on the theory and applications of cryptographic techniques*. Springer, 1998, pp. 308–318.

[16] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1999, pp. 223–238.

[17] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE transactions on information theory*, vol. 31, no. 4, pp. 469–472, 1985.

[18] C. Gentry *et al.*, "Fully homomorphic encryption using ideal lattices." in *Stoc*, vol. 9, no. 2009, 2009, pp. 169–178.

[19] M. Van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully homomorphic encryption over the integers," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2010, pp. 24–43.

[20] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter *et al.*, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *International Conference on Machine Learning*, 2016, pp. 201–210.

[21] F. Bourse, M. Minelli, M. Minihold, and P. Paillier, "Fast homomorphic evaluation of deep discretized neural networks," in *Annual International Cryptology Conference*. Springer, 2018, pp. 483–512.

[22] F. Boemer, A. Costache, R. Cammarota, and C. Wierzynski, "ngraph-he2: A high-throughput framework for neural network inference on encrypted data," in *Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, 2019, pp. 45–56.

[23] A. C. Yao, "Protocols for secure computations," in *23rd annual symposium on foundations of computer science (sfcs 1982)*. IEEE, 1982, pp. 160–164.

[24] M. Barni, C. Orlandi, and A. Piva, "A privacy-preserving protocol for neural-network-based computation," in *Proceedings of the 8th workshop on Multimedia and security*. ACM, 2006, pp. 146–151.

[25] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 19–38.

[26] B. D. Rouhani, M. S. Riazi, and F. Koushanfar, "Deepsecure: Scalable provably-secure deep learning," in *Proceedings of the 55th Annual Design Automation Conference*. ACM, 2018, p. 2.

[27] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "{GAZELLE}: A low latency framework for secure neural network inference," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018, pp. 1651–1669.

[28] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[29] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in *Advances in neural information processing systems*, 2014, pp. 3320–3328.

[30] C. Alippi, S. Disabato, and M. Roveri, "Moving convolutional neural networks to embedded systems: the alexnet and vgg-16 case," in *Proceedings of the 17th ACM/IEEE International Conference on Information Processing in Sensor Networks*. IEEE Press, 2018, pp. 212–223.

[31] "Microsoft SEAL," https://github.com/Microsoft/SEAL, microsoft Research, Redmond, WA.

[32] A. Paszke, S. Gross, F. Massa, A. Lerer *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, 2019, pp. 8024–8035.

[33] M. O. Alberto Ibarrondo, Laurent Gomez, "Pyfhel: Python for homomorphic encryption libraries," https://github.com/ibarrond/Pyfhel, 2018.

[34] S. v. d. Walt, S. C. Colbert, and G. Varoquaux, "The numpy array: a structure for efficient numerical computation," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 22–30, 2011.

[35] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[36] Y. Xiao, "Ieee 802.11 n: enhancements for higher throughput in wireless lans," *IEEE Wireless Communications*, vol. 12, no. 6, pp. 82–91, 2005.