Privacy-Preserving Classification on Deep Neural Network

Hervé Chabanne,Amaury de Wargny and Jonathan Milgram
and Constance Morel and Emmanuel Prouff
Safran Identity & Security
Email: {firstname.lastname}@safrangroup.com

***Abstract*—Neural Networks (NN) are today increasingly used in Machine Learning where they have become deeper and deeper to accurately model or classify high-level abstractions of data. Their development however also gives rise to important data privacy risks. This observation motives Microsoft researchers to propose a framework, called Cryptonets. The core idea is to combine simplifications of the NN with Fully Homomorphic Encryptions (FHE) techniques to get both confidentiality of the manipulated data and efficiency of the processing. While efficiency and accuracy are demonstrated when the number of non-linear layers is small (eg 2), Cryptonets unfortunately becomes ineffective for deeper NNs which let the problem of privacy preserving matching open in these contexts. This work successfully addresses this problem by combining the original ideas of Cryptonets' solution with the batch normalization principle introduced at ICML 2015 by Ioffe and Szegedy. We experimentally validate the soundness of our approach with a neural network with 6 non-linear layers. When applied to the MNIST database, it competes the accuracy of the best non-secure versions, thus significantly improving Cryptonets.**

I. Introduction

Neural networks aim to solve a so-called *classification problem* which consists in correctly assigning a label to a new observation, on the basis of a *training set* of data containing observations (or instances) whose labelling is known [31]. It may also be viewed as the problem of approximating unknown (complex) functions that can depend on a huge number of inputs and are generally unknown.[1] The main blocks of neural network models exist since the early 1980's but, for a long time, other methods (e.g. *Support Vector Machines* with Kernel [41]) have been preferred since they are not impacted by the overfitting issue (when the dimension of the inputs is high and the isze of the training database is small) and lead to unique solutions. The situation has recently changed with the increasing of both the size of the training database (*e.g.* available in the clouds) and the computing performances, and also thanks to technical improvements (e.g. the introduction of the *dropout* regularization technique [27] and the introduction of the *REctified Linear Unit* activation function [24]). They have brought out the neural networks, putting them forward in the 21st century. They, for instance, play a central role in face recognition [40], image classification [30] or speech recognition [23] and are thus widely deployed.

Machine learning algorithms are often applied on sensitive information such as medical data. The privacy protection of these sensitive data has been the subject of several articles during the last five years. They may be classified according to the learning algorithms which are involved: linear regression training or prediction [35], [47], linear classifiers [7], [8], [22], decision trees [8] or neural networks [2], [20], [36], [50], [52].

All the previously mentioned Machine learning algorithms are composed of two stages: the *training phase* from a labelled database and the *classification* of new data. The training phase essentially aims at inferring the algorithm parameters from a labelled (or classified) database by optimizing some training objective (recognize/classify digits, faces, objects, etc). The classification of a new datum then simply consists in applying the trained algorithm to it (see Appendix B for more details). Like in [48], [20], where Cryptonets' solution is introduced, our article

[1] A classical simple example (*e.g.* detailed in [34, Chapter 1]) is the recognition of handwritten digits.

deals with the privacy-preserving problem for the classification (aka *matching*) processing in the context of deep Neural Networks. More precisely, we focus on the popular *Convolutional Neural Network* (CNN) which belongs to the family of *multilayer perceptron* (MLP) networks that themselves extend the basic concept of *perceptron*[2] to address problems where the classification does not reduce to a simple linear separation. Readers not familiar with the basic concepts of MLP may refer to Appendix A where some basics are recalled. We additionally give in Appendix C some references about the study of the privacy of the training phase.

A. Problematic

The problem addressed in this work is formalized hereafter.

Problem *1 (Privacy Preserving Classification):* A client has his data X. A server has a trained (deep) neural network. The client obtains the classification (aka label) of his data X from the server, and this classification must not reveal information about the trained neural network. The server obtains nothing (no information about the client input or labelling).

To provide practical solutions to Problem 1, two main requirements have been added:

- **Efficiency:** the running time to obtain the classification needs to be low in order to be applicable for real world applications
- **Accuracy:** the classification performance needs to be close to the classification performance without privacy

B. Related Work

The problem of privacy-preserving neural networks has been widely studied since the beginning of the century. All existing methods are based on secure multi-party computation (SMC) or homomorphic encryption (HE) or a combination of those methods. The efficiency of these solutions strongly depends on the complexity of the neural network (complexity of the activation function, number of layers of each type, number of neurons per layer) seen as a classification function.

In 2006, Barni *et al* [2] proposed a privacy preserving neural network classification algorithm based on secure multi-party computation and homomorphic encryption. Their neural networks are composed of a succession of scalar products which are secured on the basis of homomorphic encryption and activation functions (threshold or sigmoid) secured with protocols based on secure multi-party computation. More precisely, to evaluate the scalar product between $\vec{x}$ and $\vec{y}$ both in $\mathbb{R}^N$ and owned respectively by the client and the server, the client encrypts each component of the vector $\vec{x}$ with Paillier's homomorphic encryption [37] and sends them to the server, then the server evaluates homomorphically the encryption of the scalar product and sends back the encrypted result to the client who finally decrypts it in the plaintext domain. Thanks to the homomorphic encryption, the server learns nothing about the client's vector $\vec{x}$ and the client learns nothing about the server's vector $\vec{y}$ except what he can infer from the product scalar result. The communication complexity of the scalar product evaluation is $O(Nn^2)$ where N is the number of components per vector and n is the RSA-modulus for the Paillier cryptosystem. Its computation complexity is $O(N)$ exponentiations modulo n^2. To securely evaluate each *threshold activation function* processing, Yao's solution to the millionaire's problem [49] is suggested. The communication complexity of this protocol is $O((\log_2 n)^2)$ where the two integers to compare belongs to $\mathbb{Z}_n$. To securely evaluate the sigmoid activation function, firstly the sigmoid is approximated by a low degree polynomial and then the polynomial is evaluated with private polynomial evaluation [33]. Finally, the values of all intermediate neurons (outputs of scalar products and activation functions) are revealed to the client. The authors noted this issue and keep it for future works. To sum up, this protocol suffers from two major drawbacks: it does not fulfill neither the privacy requirement nor the efficiency requirements of our Problem 1.

Orlandi *et al* [36] enhanced the protocol [2]: intermediate results are no longer revealed to the client. To evaluate scalar products, they

[2]A *perceptron* is a binary classifier that first applies an inner product between real-valued input vectors and a fix trained/learned vector and then associates the label 0 or 1 to the output depending on a learned threshold.

use the same protocol as Barni *et al.* with two improvements. Firstly they replace Paillier's cryptosystem by its extension by Damgård and Jurik [13]. Secondly, the scalar product results are masked before delegating the evaluation of the activation functions to the client. For instance, if the activation function is the threshold function, the server computes homomorphically $Enc(a(< \vec{x}, \vec{y} > -\delta))$ where δ is the threshold of the activation function and a is a non zero value randomly chosen by the server. Then the client decrypts it and sends $Enc(-1)$ if $a(< \vec{x}, \vec{y} > -\delta) < 0$ and $Enc(1)$ otherwise. The client cannot infer information about $< \vec{x}, \vec{y} >$ thanks to the random value a. If a is negative, the server has to multiply the encrypted result by (-1) to obtain the correct encrypted result. Unfortunately, this protocol has a similar computation/communication complexity than the Barni *et al* solution and thus it does not respect the efficiency requirement.

To overcome this drawback, Gilad-Bachrach *et al* [48], [20] proposed a new privacy preserving neural network classification without interaction between the client and the server. Their approach, which applies the principles of Fully Homomorphic Encryption (HFE), is composed of the following steps: the client encrypts his data as specified by the the chosen HFE scheme and sends it to the server, then the server processes a version of the classification algorithm which has been adapted to operate on encrypted data without decrypting them, and the encrypted result is eventually sent back to the client who can decrypt it. The privacy is obviously ensured thanks to the data encryption and the communication complexity is very low. These two assets may however come at the cost of an important increase of the processing complexity. The latter one depends on the algebraic nature of the classification algorithm, and more precisely on its multiplicative depth[3]. The main issue is therefore to adapt the classification algorithm to render it compatible with homomorphic encryption while maintaining good accuracy. To achieve this result, Gilad-Bachrach *et al* [48], [20] simply propose to replace the activation function by the squared function which has multiplicative depth 1. To maintain good accuracy, the same neural network architecture is used for the training and the classification. Thanks to the low degree polynomial activation function, the multiplicative depth of the neural network is reasonable during the secure classification, and thus the homomorphic classification with YASHE encryption scheme [6] is efficient. Unfortunately as acknowledged by the authors themselves, the fact that the squared function has an unbounded derivate induces a strange behaviour during the training phase. Therefore this protocol is only adapted to small neural networks and its accuracy is very low for neural networks with more than 2 non-linear layers (which are yet the most common in nowadays applications).

[3]We recall that the multiplicative depth of a processing is the maximum number of consecutive multiplications that must be computed during to get the output.

C. Contributions

In this paper, we design and evaluate the first privacy preserving classification for neural networks whith depth greater than 2. Like the best state-of-the-art solutions discussed in the previous section, our goal is to end up with a construction being FHE friendly while keeping accuracy at high level. This requires in particular that the multiplicative depth of the neural network is maintained reasonably low. To address this issue, the literature suggests to replace the activation function by a low degree non-linear polynomial both on the learning and classification phases. To stay as close as possible to the state of the art neural networks, and to therefore respect our accuracy requirement, we followed a different approach and we chose to keep a classical deep neural network (with the ReLU activation function) for the training phase; we afterwards only made the modifications on the network when it is used for classification. As in [20], the first modification consists in approximating the ReLU by a low degree polynomial. This modification, if applied solely, does not work because the polynomial approximation of the ReLU function is only good on a restricted distribution whereas the activation function applies on inputs with unstable distributions. To circumvent this issue, which leads to classification errors and dramatically degrades the

accuracy, our key technical innovation is to combine the latter polynomial approximation with a *batch normalization* [28]. The batch normalization concept has been introduced in 2015 by Sergey Ioffe and Christian Szegedy in order to greatly accelerate the training in deep neural network. This idea consists in adding a normalize layer before each activation layer in order to have a stable and normal distribution at the level of the activation function inputs. When such a normalization is involved, the polynomial approximation just needs to be accurate on a small and fix interval.

To validate the soundness of our approach we launched several experiments. In order to both increase our chance to achieve a good accuracy on the MNIST database [32] and to test the robustness of our methods for NNs with striclty more than 2 non-linear layers, we started by training a deep neural network with 6 activation layers (see Figure 1). We tested that, without any modification, this trained neural network achieves 99.59% accuracy. For the privacy-preserving classification step, the ReLU layers have been replaced by degree-2 polynomial approximations. To the best of our knowledge, the BGV encryption scheme [9] with the Smart-Vercauteren ciphertext packing techniques [43] and the Gentry-Halevi-Smart optimizations [18] is today the most efficient encryption scheme to evaluate polynomial functions. It is why we chose to use it for our experiments[4]. Finally, we observed that our privacy-preserving CNN achieves 99.30% accuracy, which improves that of Cryptonets (98.95%). To conclude, our solution respects the three requirements: privacy (our solution is FHE friendly), efficiency (our solution has a low multiplicative depth of 6) and accuracy (the recognition performance with and without privacy are close). Last but not least, our approach is scalable and can be applied to neural networks with an important number of non-linear layers without too much decreasing the accuracy performances (which was not the case with Cryptonets).

[4]In addition, HElib [26], a software library that implements this homomorphic encryption scheme, is available on GitHub [25]. As in [20], all real values will be mapped to fixed point representations because the BGV encryption scheme can only be applied on finite field.

D. Paper Organization

We present the required preliminaries in Section II. Our privacy preserving deep neural network classification protocols can be found in Section III. This solution requires the approximation of the ReLU function by a low degree polynomial. The section IV presents this polynomial approximation. Finally, we tested our solution firstly on a light CNN (see Section V) and then on a deeper CNN (see Section VI). Section VII concludes this paper.

II. Preliminaries

In this section, the convolutional neural network concept and the homomorphic encryption are presented.

A. Convolutional Neural Network (CNN)

Convolutional Neural Network (CNN) are particularly tailored for image recognition. For that purpose, the classical principle of MLP is completed with a new type of layer, called *convolutional layer* based on convolutional filtering. Like a perceptron, this convolutional layer is a scalar product between some input neurons and some weights which have been priorly trained/learned.

A convolutional neural network is a stack of layers that transforms an input layer, holding data to classify, into an output layer, holding the label scores. Each neuron of one layer (except the input layer) is the output of a function applied on the neurons of the previous layer. A few distinct kinds of layers are commonly used such as the fully connected layer, the convolutional layer, the activation layer and the pooling layer.

a) Fully connected layer: each neuron of this layer is connected to each neuron of the previous layer. The weight on the connection between the j-th neuron of the $(\ell-1)$-th layer and the k-th neuron of the ℓ-th layer noted is denoted by ω_{kj}^{ℓ}. The bias of the k-th neuron of the ℓ-th layer is denoted by β_k^{ℓ} and its ouptut x_k^{ℓ} equals $\beta_k^{\ell}+\sum_j \omega_{kj}^{\ell} x_j^{\ell-1}$, which is a simple inner product.

b) Convolutional layer: The issue with the fully connected layer is the number of parameters to learn. In fact, the number of weights in one fully connected layer is equal to the number of neurons in the previous layer multiplied by the number of neurons in the

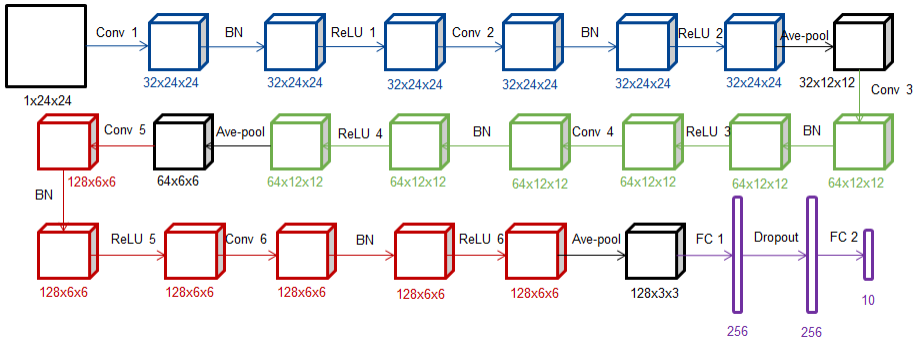

Fig. 1: Our deep neural network

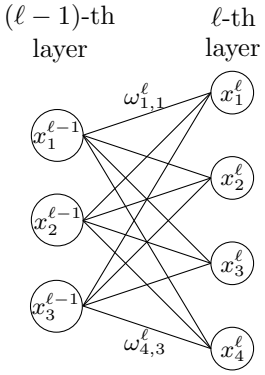

Fig. 2: Fully connected layer

current layer. For instance, if a layer and its previous layer contain respectively 100 and $10,000$ neurons, then the current layer contains 10^6 weights and 100 biases to learn. To overcome this issue, convolutional layers are used. These layers are especially tailored for image inputs because they are based on *convolutional filtering* used in image processing. In image processing, a *filter* studies successively each pixel of an image. For each pixel, a convolution is performed by multiplying the pixel and its neighbouring pixels values by the filter corresponding weights and then by adding all the results. The new pixel is set to this final result value. The same filter is used on each pixel of the image. Figure 3 shows a convolutional filtering.

Convolutional layers are based on convolutional filtering. The weights within the filter are the weights to learn during the training step. In addition, several filters are performed on the same layer in order to extract different kinds of characteristics. Hence, each layer

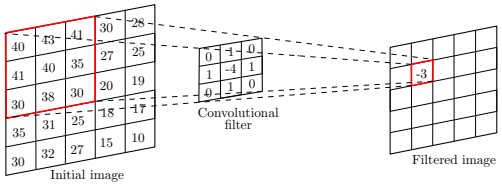

Fig. 3: Convolutional filtering

is a three-dimension layer: a two-dimension matrix is obtained from each filter and then the result of each filter is put into a stack to obtain a three-dimension result. In addition, the filter is also in three dimensions in order to take into account the three dimensions of the previous layer. For instance, if n filters are applied on a layer containing $w \times h \times d$ neurons, then each filter contains $s \times s \times d$ weights where s is the filter size and the output layer contains $w \times h \times n$ neurons.

More specifically, the convolutional layer of the ℓ-th layer contains $w^\ell \times h^\ell \times d^\ell$ neurons noted $x^\ell_{i,j,k}, \forall (i,j,k) \in [0..w^\ell - 1] \times [0..h^\ell - 1] \times [0..d^\ell - 1]$. The parameters to learn are arranged into filters. Each filter contains $s^\ell \times s^\ell \times d^\ell$ weights and one bias and is used to obtain a matrix of $w^\ell \times h^\ell$ neurons. Mathematically, the (i_0, j_0, k_0)-th neuron of the ℓ-th layer noted $x^\ell_{i_0,j_0,k_0}$ is computed with the k_0-th filter and some neurons of the previous layer with the following formula: $x^\ell_{i_0,j_0,k_0} = \beta^\ell_{k_0} + \sum_{i=0}^{s^\ell-1} \sum_{j=0}^{s^\ell-1} \sum_{k=0}^{d^\ell-1} \omega^{\ell,k_0}_{i,j,k} x^{\ell-1}_{i_0+i-\lfloor s/2 \rfloor, j_0+j-\lfloor s/2 \rfloor, k}$. Figure 4 represents the application of one filter into a convolutional layer.

c) Activation layer: A neural network containing only fully connected and convolu-

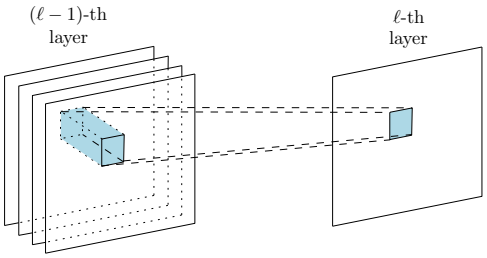

Fig. 4: Application of one filter into a convolutional layer

tional layers can only classify data linearly. To solve more complex classification problem, the activation layer has been introduced. The same non-linear function called the activation function is applied to each neuron of the previous layer to obtain one neuron of the current layer: $x^{\ell}_{i,j,k} = f(x^{\ell-1}_{i,j,k})$ where f is the activation function. Therefore, an activation layer contains the same number of neurons than its previous layer. The two most common activation functions are the ReLU function $f(x) = \max(0, x)$ and the sigmoid function $f(x) = (1 + e^{-x})^{-1}$. Activation layers are usually used immediately after convolutional or fully connected layers. Figure 5 represents an activation layer.

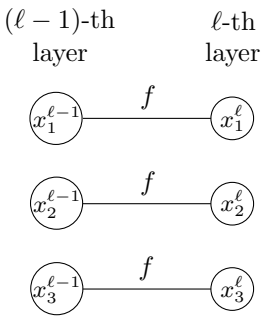

Fig. 5: Activation layer

d) Pooling layer: As activation layers, they are non-linear layers. This layer reduces the spatial size in order to reduce the amount of neurons. This layer partitions the neurons of the previous layer into a set of non overlapping rectangles and performs a function on each sub-area in order to obtain the value of one neuron of the current layer. The most common pooling functions are the max-pooling which outputs the maximum values within the sub-area and the average-pooling which outputs the average of the values of the sub-area. In addition, the pooling layer operates independently on every depth slice of the previous layer. Pooling layers are usually used immediately after activation layers. Figure 6 represents a pooling layer.

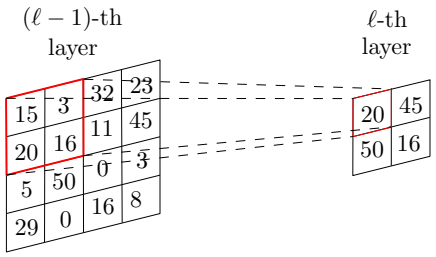

Fig. 6: Max-pooling layer

e) Common architectures: The main block of a convolutional neural network is a convolutional layer ($CONV$) directly followed by an activation layer (ACT), noted $CONV \rightarrow ACT$. The convolutional layer extracts locally information from the previous layer thanks to filters and the activation layer increases the complexity of the learned classification function thanks to its non-linearity. After some $CONV \rightarrow ACT$ blocks, a pooling layer ($POOL$) is usually added to reduce the number of neurons: $[CONV \rightarrow ACT]^p \rightarrow POOL$. This new block is repeated in the neural network until obtaining a layer of reasonable size. Then some fully connected layers (FC) are introduced in order to obtain a global result which depends of the entire input. A common convolutional network can be summarised in the following formula: $[[CONV \rightarrow ACT]^p \rightarrow POOL]^q \rightarrow [FC]^r$.

A neural network can be viewed as a function taking as inputs some parameters (weights and biases) and one data and which outputs the probability of each label for this data. For instance, if the neural network learns how to recognize handwritten digits. Its input layer will have as many neurons than pixels in image to classify and its output layer will contain ten neurons, one per possible digits. During the training step, the weights and biases will be learned in order to recognize as well as possible handwritten digits. During the training step, the ten neurons in the output layer will contains the probability for each digit. The position of the highest neuron of

the output layer indicates the digit of the input image.

B. Homomorphic Encryption

a) Partially homomorphic cryptosystem: Homomorphic encryption is an encryption which allows computations over encrypted data without knowing the encryption secret key and learning the raw data. For instance, additive homomorphic encryption allows to compute additions on encrypted data: given encryptions $Enc(m_1), Enc(m_2)$ of the messages m_1 and m_2, one can efficiently compute a ciphertext that encrypts $m_1 + m_2$ without knowing m_1, m_2 or the secret key. Homomorphic encryptions allowing only one type of operations (additions or multiplications) are called partially homomorphic encryptions and have existed for many years with, for instance, Paillier cryptosystem [37] for additive homomorphic encryption and RSA cryptosystem [38] or El Gamal cryptosystem [16] for multiplicative homomorphic encryption. These encryptions have been widely used in various applications such as electronic voting [14], [15] or simple statistics (e.g. mean, variance) in data mining.

b) Fully Homomorphic Encryption (FHE): Partially homomorphic encryption are limited to only one type of operations but in many applications (including ours), we would like to homomorphically perform additions and multiplications. The first scheme addressing this issue has been introduced by Craig Gentry in 2009 [17]. This scheme called fully homomorphic encryption (FHE) is based on ideal lattices and its construction contains two steps. It starts with a *somewhat homomorphic encryption* (SWHE) scheme which allows a fixed number of operations (additions and multiplications) on the encrypted domain. A *bootstrapping* operation is added to the SWHE. This operation refreshes ciphertexts by homomorphically evaluating the decryption circuit. The combination of a SWHE with a bootstrapping operation results to a fully homomorphic encryption: the number of operations is now unlimited. Unfortunately, this scheme is inefficient for practical applications due to its high computation and memory cost.

From the Gentry's breakthrough work, numerous FHE schemes have been introduced in order to make it practical. These schemes can be organised in three categories:

- Schemes based on ideal lattices such as the Gentry's original work [17] and its optimizations [44], [19]. These schemes have worse performances than the latter systems and so are no longer considered
- Schemes based on *learning with error* (LWE) or *ring learning with error* (RLWE) problems such as Brakerski and Vaikuntanathans schemes [10], [11] and their famous optimization Brakerski-Gentry-Vaikuntanathan (BGV) [9]
- Schemes which operate over integers instead of ideal lattices such as the van Dijk, Gentry, Halevi and Vaikuntanathans (DGHV) scheme [46]

Despite the significant progress in the FHE area, no efficient FHE scheme for all generic functions exist. But for some specific applications, very efficient solutions based on FHE can be designed. In our work, we designed an efficient solution for privacy preserving classification on deep neural network. Our solution is based on the BGV encryption scheme [9] with the Smart-Vercauteren ciphertext packing techniques [43] and the Gentry-Halevi-Smart optimizations [18]. To the best of our knowledge, this scheme is today the most efficient FHE scheme for polynomial evaluations. In addition, HElib [25] is an efficient implementation of this scheme.

c) Complexity of FHE and multiplicative depth: Like many FHE schemes, the BGV encryption scheme consists in hiding the plaintext message with noise in order to create the ciphertext message. The decryption consists in removing the noise from the ciphertext message. The noise level increases with each homomorphic operation. If the noise level exceeds a certain threshold, it is no longer possible to correctly decrypt the message. The noise growth is much more important with multiplications as opposed to additions. It is why only the multiplicative depth (maximum number of multiplications in series) is taken into account when the computational complexity is evaluated.

III. Our Privacy-Preserving Deep Neural Network Classification solution

This section presents our new proposal for a privacy preserving classification on deep neural network. This proposal respects our three requirements: privacy (achieved thanks to FHE), efficiency (reasonably low multiplicative depth) and accuracy (close to the state of the art CNN). We recall that the main issue with the current literature is that CNN architectures usually contain ReLU activation functions and max pooling functions which have a high multiplicative depth and thus are incompatible with our efficiency requirement.

Since our proposal is based on an idea originally introduced in [20] and may be viewed as an extension of this work, we start by recalling the Cryptonets solution hereafter.

A. Cryptonets solution [20]

Cryptonets solution consists in replacing the high multiplicative depth layers (ReLU and max pooling) by low multiplicative depth polynomial layers into the CNN, without too much degrading the accuracy of the classification. To achieve this goal, the authors suggest to replace the ReLU function by the square function $x \mapsto x^2$ and to replace the max-pooling by the sum-pooling which has a null multiplicative depth. The proposal is tested in a small CNN (9 layers with only 2 activation layers) which is applied on the MNIST database. They obtained an accuracy of 98.95% (the state of the art accuracy for this database is 99.77% according to http://yann.lecun.com/exdb/mnist/). The multiplicative depth of the CNN is equal to 2 and thus the classification with FHE (ie privacy preserving) is efficient. However, this design respects only two requirements among our three ones: efficiency and privacy. The non-compliance with the accuracy requirement can be explained by the low depth of the CNN (only 9 layers).

A simple and natural idea to improve the accuracy could be to increase the CNN depth. Unfortunately, as actually acknowledged by the authors themselves, the squaring activation function has a derivative which is unbounded, which leads to unstable training, and hence important accuracy loss, when the CNN is deep. This observation about the incompatibility of the approach with deep CNN seems to strongly limit the practical interest of Cryptonets. At least it shows that more investigations are needed to make it practical for deep learning.

In view of the unsuccess of the CryptoNets, we chose to consider separately the CNN used for the training phase and the CNN used for the matching:

- for the training phase the activation function has to be a non linear function with a bounded derivation in order to have a stable training (especially for deep CNN) and its multiplicative depth is not an issue since we don't want to make it privacy preserving (and hence don't need to apply FHE)
- for the classification, the activation function has to be a low degree polynomial in order to respect the efficiency requirement (FHE friendly solution)

B. Our solution

To respect the accuracy requirement, we make minor modifications in the CNN for the training step compared to state of the art CNNs. Namely, we keep the ReLU activation function which seems to be a key function in the current CNN performances and we replace the max-pooling by the average-pooling (this has only a small impact on the accuracy and has the advantage to replace a step which is not FHE-friendly by a step with null multiplicative depth). For the CNN used in the matching, we propose to start with the trained CNN and to replace the ReLU functions by low degree approximations (hence reapplying the same basic idea as in CryptoNets). To deal with the fact that the ReLU function is a high degree polynomial and thus cannot have a good low degree polynomial approximation on the entire $\mathbb{R}$, our key innovation is to combine the polynomial approximation of the ReLU with a batch normalization layer [28]. Namely, before each ReLU layer, we add a batch normalization layer in order to have a restricted stable distribution at the entry of the ReLU, thus limiting our need of an accurate polynomial approximation to a small part of $\mathbb{R}$ around the point 0. To avoid high accuracy degradation between the

training and the classification phases due to too many modifications into the CNN, the batch normalization layers will be added to the CNN for the training and the classification phases.

As a reminder a common convolutional network can be summarized in the formula $[[CONV \rightarrow ACT]^p \rightarrow POOL]^q \rightarrow [FC]^r$ (see Section II-A). In our solution, convolutional networks have the following architecture $[[CONV \rightarrow BN \rightarrow ACT]^p \rightarrow POOL]^q \rightarrow [FC]^r$ where BN represents a batch normalization layer and $POOL$ is the average-pooling. In addition, the activation layer ACT is the ReLU function for the training phase and a polynomial approximation for the classification phase.

IV. Approximation of ReLU by Polynomials

The accuracy of our solution is strongly linked to the quality of the polynomial approximation of the ReLU function on the output distribution of the batch normalization layer (see Figure 9). This latter layer is right after a convolutional layer. Thus, according to the central limit theorem, the input distribution of the batch normalization layer is normal, and hence the output distribution is the standard normal one.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Fig. 7: Batch normalization [28, Algorithm 1]

To compute the polynomial approximation of the ReLU on this standard normal distribution, we used the polynomial regression function $polyfit$ from the Python package $numpy$. This function inputs a set $X = (X_1, ..., X_N)$, a set $Y = (Y_1, ..., Y_N)$ and a polynomial degree n and outputs the coefficients of the polynomial $P(X) = c_0 + c_1X + c_2X^2 + ...c_nX^n$ such that the square error $\epsilon = \sum_{i=1}^{N}(P(X_i) - Y_i)^2$ is minimized. We applied this function on the set $\{(X_i, ReLU(X_i))\}$ where the X_i are randomly picked up from a standard normal distribution (see Table I and Figure 8).

Degree	Polynomials
2	$0.1992 + 0.5002X + 0.1997X^2$
3	$0.1995 + 0.5002X + 0.1994X^2 - 0.0164X^3$
4	$0.1500 + 0.5012X + 0.2981X^2$ $-0.0004X^3 - 0.0388X^4$
5	$0.1488 + 0.4993X + 0.3007X^2$ $+0.0003X^3 - 0.0168X^4$
6	$0.1249 + 0.5000X + 0.3729X^2$ $-0.0410X^4 + 0.0016X^6$

TABLE I: Approximation of the ReLU function by polynomials on the standard normal distribution

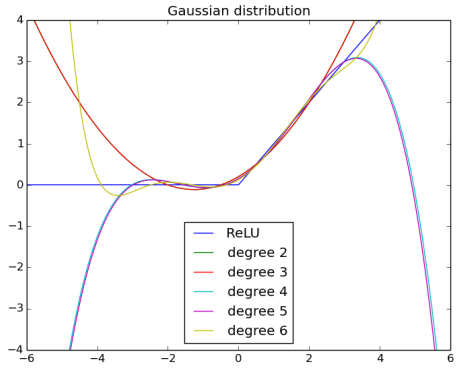

Fig. 8: Approximation of the ReLU function by polynomials on the standard normal distribution

We can observe that, for our method, the polynomials of odd degree $2n+1$ approximate similarly as the polynomial of even degree $2n$. The same trend can be mathematically observed from a Taylor series around the point 0 of a smooth approximation of the ReLU function called softplus $ln(1 + e^x)$ (see Figure 9): $ln(1+e^x) = ln(2) + \frac{x}{2} - \frac{x^4}{192} - \frac{17x^8}{645120} + \frac{31x^{10}}{14515200} + O(x^{12})$. Thus we only used the even degree polynomial approximations in our solution.

With the standard normal distribution, 99.73% of the values belong to $[-3, 3]$. When the degree of the polynomial approximation increases, our polynomial approximation

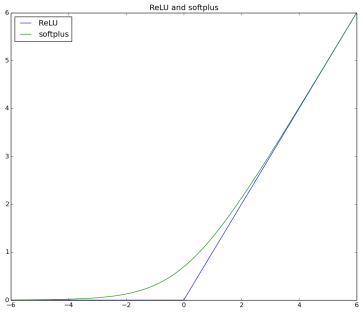

Fig. 9: ReLU and softplus

brightens on $[-3, 3]$ but deteriorates outside $[-3, 3]$.

V. Experimentation on a light CNN

To begin, we tested our solution on a light CNN (see Figure 10) with the following characteristics:

1) The convolutional layers (Conv1 and Conv2) have respectively 20 and 50 feature maps and their filter sizes are respectively $(1 \times 5 \times 5)$ and $(20 \times 5 \times 5)$.
2) Average pooling layers have (2×2) window size.
3) The fully connected layers (FC1 and FC2) have respectively 500 and 10 outputs.
4) A batch normalization layer (BN) is present just before the ReLU layer.

This light CNN contains 9 layers and only one ReLU activation layer.

Firstly, we trained this light CNN with the Caffe framework [29] on the MNIST database with the following parameters:

- base learning rate: $b\_lr = 0.01$
- learning rate policy: $b\_lr * (1 + 10^{-4} * iter)^{-0.75}$
- momentum: 0.9
- weight decay: 0.0005
- max iter: 10^4
- solver type: SGD

We obtained an accuracy of 97.95% which is far from the state of the art for the digit recognition problem (99.77%). That is due to the small size of our light CNN. The goal of this light CNN is to validate our solution by proving that our solution respects the accuracy requirement. Thus we would like to prove that the replacement of the ReLU layer by a low degree polynomial layer leads to a low accuracy degradation. To limit the accuracy degradation due to this activation layer modification, the polynomial has to approximate very well the ReLU function on the output distribution of the batch normalization layer. To do that, we firstly analyzed the output distribution of the batch normalization layer (see Figure 11).

As expected, this distribution is close to a standard normal distribution. Then we replaced the ReLU functions by our polynomial approximations and obtained the accuracy of the Table II.

Degree	Accuracy
2	97.55%
4	97.84%
6	97.91%

TABLE II: Classification accuracy on the light CNN

As expected, the performances obtained with the private classification (FHE friendly classification with polynomial activation layer) are similar to the accuracy of 97.95% obtained with the non-private classification (with the ReLU activation layer). Thus, our solution on this light CNN respects the accuracy requirements. In addition, the multiplicative depth with this light CNN is equal to $\log_2 deg$ where deg is the degree of the polynomial approximation. Thus our solution applied on this light CNN respects the three requirements: privacy (thanks to FHE), efficiency (with a multiplicative depth of 1, 2 or 3 according to the polynomial approximation degree) and accuracy (thanks to the batch normalization and the proficient polynomial approximation on a standard normal distribution).

VI. Experimentation on a deeper CNN

In this section, we tested our solution on a deeper CNN (see Figure 12) with more hidden layers and more activation layers in order to be closer to state of the art CNN. This CNN has the following characteristics:

1) In each convolutional layer, the filter size is $(n \times 3 \times 3)$ with n the number of feature maps of the input layer.
2) Conv1 and Conv2 have 32 feature maps, Conv3 and Conv4 have 64 feature maps,

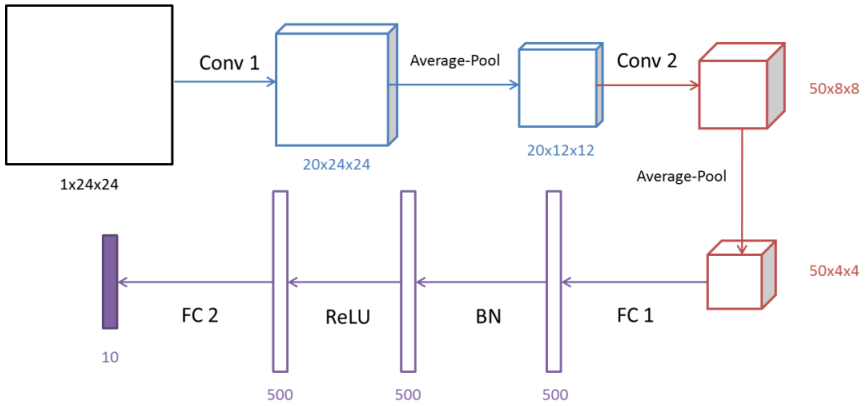

Fig. 10: Our light CNN

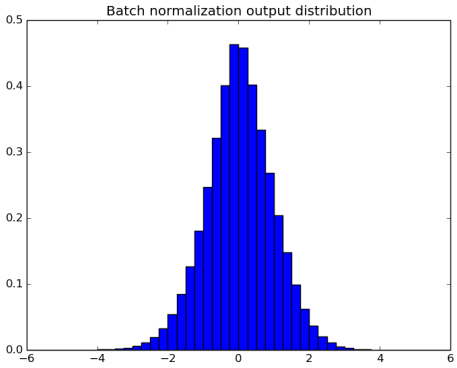

Fig. 11: Output distribution of the batch normalization layer

Conv5 and Conv6 have 128 feature maps.

3) Average pooling layers have (2×2) window size.
4) The fully connected layers (FC1 and FC2) have respectively 256 and 10 outputs.
5) A batch normalization layer (BN) is present just before each ReLU layer.
6) A dropout is used during the training phase only to select 50% of the neurons

This CNN contains 24 layers and six ReLU activation layers.

Firstly, we trained this CNN with the Caffe framework [29] on the MNIST database with the following parameters:

- base learning rate: $b\_lr = 0.02$
- learning rate policy: $b\_lr * (1 + 10^{-4} * iter)^{-0.75}$
- momentum: 0.9
- weight decay: 0.0005
- max iter: 10^5
- solver type: SGD

We obtained an accuracy of 99.59% which is similar to state of the art accuracy (99.77%). Then we replaced the ReLU functions by our polynomial approximations and evaluated the accuracy of these FHE friendly classifications (see Table III).

Degree	Accuracy
2	59.14%
4	97.91%
6	36.94%

TABLE III: Classification accuracy on our CNN

The accuracy degradation is higher than on our light CNN. This degradation can be explained by the number of activations layers in our CNN. During the private classification, some errors appear after the first activation layer due to the replacement of the ReLU functions by our polynomial approximations. These errors lead to a low distortion of the output distribution of the next batch normalization layer. Our polynomial approximations are not perfectly suited for this new distribution and thus more and bigger errors appear after the second activation layer. The errors spread and strengthen from one layer to the next layer. To visualize the distribution distortion, we plotted the output distribution of the 6-th batch normalization layer with the degree 2 or 4

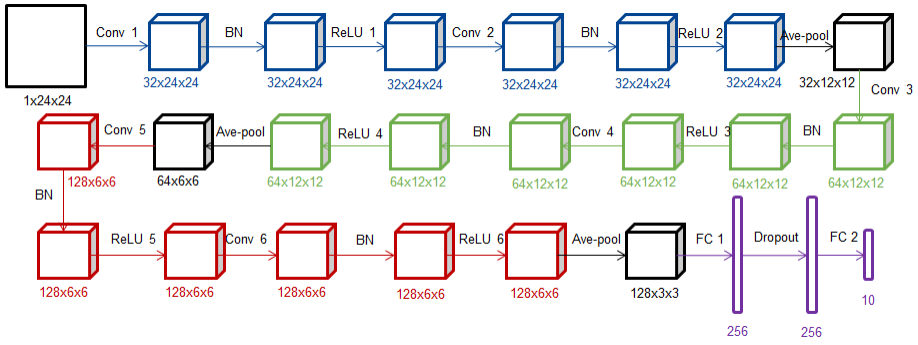

Fig. 12: Our CNN

polynomial approximations (see Figure 13).

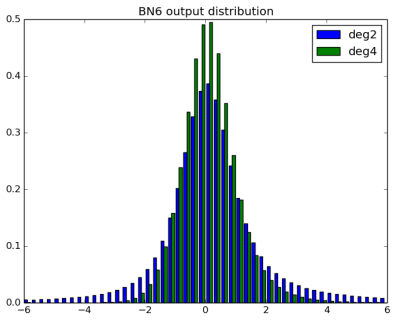

Fig. 13: Output distribution of BN6 with the degree 2 and 4 polynomial approximations

The distribution with the degree 4 polynomial approximation is closer to the standard normal distribution than the distribution with the degree 2 polynomial approximation. That explains the accuracy difference between this two approximations. These two distributions are closed to a normal distribution with a standard deviation slightly greater than 1. To improve our accuracy, we suggest to build new polynomial approximations learned from a distribution closer to the output distribution of the batch normalization (for instance on a normal distribution with a standard deviation slightly higher than 1). Table IV sums up our results with these new polynomial approximations. We limited our polynomial approximations to degree 2 and 4 to respect the efficiency requirement. Our solution has a multiplicative depth of 6 with degree 2 polynomial approximation and a multiplicative depth of 12 with degree 4 polynomial approximation. These both multiplicative depth are reasonable.

Degree	Normal distribution	Accuracy
2	$\mu = 0, \sigma = 1.1$	87.12%
2	$\mu = 0, \sigma = 1.2$	90.70%
2	$\mu = 0, \sigma = 1.3$	82.61%
4	$\mu = 0, \sigma = 1.1$	98.18%
4	$\mu = 0, \sigma = 1.2$	97.75%

TABLE IV: Classification accuracy with our new polynomial approximations

To be closer to the non secure accuracy 99.59%, we adapted the CNN weights to our polynomial activation function. To do that, we started with the CNN trained with the ReLU activation function. We replaced the ReLU functions by our degree 2 polynomial approximations learned on a normal distribution with $\mu = 0$ and $\sigma = 1.2$. Finally we continued the CNN learning with a low learning rate (10^{-5}). With these new learning weights, we obtained an accuracy of 99.28%. Unfortunately this method does not work with degree 4 polynomial approximations because the last training with degree 2 polynomial activation layer is unstable.

As the distribution after each batch normalization layer depends of the polynomial approximations used, it is difficult to build a polynomial approximation which perfectly fits this distribution. Thus we decided to consider the polynomial approximation coefficients as learning weights in the last training step. In more details, as previously, we started with the CNN trained with the ReLU activation function. Then we continued the learning after the replacement of the ReLU functions by our degree 2 polynomial approximation

(learned from a normal distribution with $\mu = 0$ and $\sigma = 1.2$) where its coefficients belong to learning weights. With this solution, we achieved an accuracy of 99.30% which is very close to the accuracy from the non private classification (99.59%).

These experiments prove that our solution respects the accuracy requirement. In addition, our last private classification employs only degree 2 polynomial approximations. Thus it respects the efficiency requirement (with a multiplicative depth of 6). Hence, our solution respects the three requirements.

VII. Conclusion

Today, convolutional neural networks (especially deep neural network) provide the best classification methods in areas such that computer vision and speech recognition. These algorithms are often applied on sensitive information and thus require a private implementation. In this area, two challenges have to be solved: privacy preserving learning and privacy preserving classification on CNN. In this article, we focus on the privacy preserving classification challenge. The main difficulty is to simultaneously respect the following three requirements: privacy, efficiency and accuracy.

To achieve the privacy requirement, our solution is based on FHE. The main difficulty is to obtain a low multiplicative depth classification (to achieve the efficiency requirement) without degrading to much the accuracy. To respect the accuracy requirement, our CNN in the training phase is similar to the state of the art CNN (with the ReLU activation layer). This training CNN can have a high multiplicative depth because we do not apply FHE on it. For the CNN used in the classification phase, we replace the ReLU functions by low degree polynomial approximations. Our private classification accuracy relies on the approximation. Unfortunately the ReLU function has a high degree, and hence it is not possible to have a good approximation of the ReLU function by a low degree polynomial on the entire $\mathbb{R}$. Our key innovation is to combine the polynomial approximation of the ReLU with a batch normalization layer. Thanks to this latter layer, we have a restricted stable distribution at the entry of each activation layer, and thus it is now possible to build a good low degree polynomial approximation of the ReLU on this distribution.

We firstly tested our solution on a slight CNN in order to prove the low accuracy degradation between the non private classification (97.95%) and the private classification (97.91%). Then we tested our solution on a deeper CNN with 24 layers. To respect the accuracy requirement, some tricks have been used: a second training to adapt the CNN weights to the polynomial layers and a polynomial approximation learned from a normal distribution slightly higher than 1. We achieved a private classification accuracy of 99.30% which is better than Cryptonets (98.95%) and close to the non private accuracy on the same CNN 99.59%. As the best of our knowledge, our solution is the first privacy preserving classification method compatible with deep CNN.

Acknowledgment

This work has been supported in part by the CRYPTOCOMP FUI17 project.

References

[1] A. Bansal, T. Chen, and S. Zhong. Privacy preserving back-propagation neural network learning over arbitrarily partitioned data. *Neural Computing and Applications*, 20(1):143–150, 2011.

[2] M. Barni, C. Orlandi, and A. Piva. A privacy-preserving protocol for neural-network-based computation. In *Proceedings of the 8th workshop on Multimedia & Security, MM&Sec 2006, Geneva, Switzerland, September 26-27, 2006*, pages 146–151, 2006.

[3] Y. Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural Networks: Tricks of the Trade - Second Edition*, pages 437–478. 2012.

[4] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

[5] D. Boneh, E. Goh, and K. Nissim. Evaluating 2-dnf formulas on ciphertexts. In *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*, pages 325–341, 2005.

[6] J. W. Bos, K. E. Lauter, J. Loftus, and M. Naehrig. Improved security for a ring-based fully homomorphic encryption scheme. In *Cryptography and Coding - 14th IMA International Conference, IMACC 2013, Oxford, UK, December 17-19, 2013. Proceedings*, pages 45–64, 2013.

[7] J. W. Bos, K. E. Lauter, and M. Naehrig. Private predictive analysis on encrypted medical data. *Journal of Biomedical Informatics*, 50:234–243, 2014.

[8] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser. Machine learning classification over encrypted data. *IACR Cryptology ePrint Archive*, 2014:331, 2014.

[9] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 309–325, 2012.

[10] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, 2011.

[11] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, 2011.

[12] T. Chen and S. Zhong. Privacy-preserving backpropagation neural network learning. *IEEE Trans. Neural Networks*, 20(10):1554–1564, 2009.

[13] I. Damgård and M. Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *Public Key Cryptography, 4th International Workshop on Practice and Theory in Public Key Cryptography, PKC 2001, Cheju Island, Korea, February 13-15, 2001, Proceedings*, pages 119–136, 2001.

[14] Ivan Damgård, Jens Groth, and Gorm Salomonsen. The theory and implementation of an electronic voting system. In *Secure Electronic Voting*. 2003.

[15] Ivan Damgård, Mads Jurik, and Jesper Buus Nielsen. A generalization of paillier's public-key system with applications to electronic voting. *Int. J. Inf. Sec.*, 9(6), 2010.

[16] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, 1984.

[17] C. Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. crypto.stanford.edu/craig.

[18] C. Gentry, S. Halevi, and N. P. Smart. Homomorphic evaluation of the AES circuit. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, pages 850–867, 2012.

[19] Craig Gentry and Shai Halevi. Implementing gentry's fully-homomorphic encryption scheme. In *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, 2011.

[20] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. E. Lauter, M. Naehrig, and J. Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 201–210, 2016.

[21] Pavel Golik, Patrick Doetsch, and Hermann Ney. Cross-entropy vs. squared error training: a theoretical and experimental comparison. In *INTERSPEECH 2013, 14th Annual Conference of the International Speech Communication Association, Lyon, France, August 25-29, 2013*, pages 1756–1760, 2013.

[22] T. Graepel, K. E. Lauter, and M. Naehrig. ML confidential: Machine learning on encrypted data. In *Information Security and Cryptology - ICISC 2012 - 15th International Conference, Seoul, Korea, November 28-30, 2012, Revised Selected Papers*, pages 1–21, 2012.

[23] A. Graves, A. Mohamed, and G. E. Hinton. Speech recognition with deep recurrent neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013*, pages 6645–6649, 2013.

[24] R Hahnloser, R. Sarpeshkar, M A Mahowald, R. J. Douglas, and H.S. Seung. Digital selection and analogue amplification coesist in a cortex-inspired silicon circuit. *Nature*, 405:947–951, 2000.

[25] S. Halevi and V. Shoup. HElib - an implementation of homomorphic encryption. https://github.com/shaih/HElib/.

[26] S. Halevi and V. Shoup. Algorithms in HElib. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, pages 554–571, 2014.

[27] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[28] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 448–456, 2015.

[29] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[30] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, pages 1106–1114, 2012.

[31] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio. Object recognition with gradient-based learning. In *Shape, Contour and Grouping in Computer Vision*, page 319, 1999.

[32] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[33] M. Naor and B. Pinkas. Oblivious transfer and polynomial evaluation. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, May 1-4, 1999, Atlanta, Georgia, USA*, pages 245–254, 1999.

[34] M. A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[35] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft. Privacy-preserving ridge regression on hundreds of millions of records. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 334–348, 2013.

[36] C. Orlandi, A. Piva, and M. Barni. Oblivious neural network computing via homomorphic encryption. *EURASIP J. Information Security*, 2007, 2007.

[37] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances*

in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding, pages 223–238, 1999.

[38] R L Rivest, L Adleman, and M L Dertouzos. On data banks and privacy homomorphisms. *Foundations of Secure Computation, Academia Press*, 1978.

[39] N. Schlitter. A protocol for privacy preserving neural network learning on horizontally partitioned data. In *Privacy in Statistical Databases - UNESCO Chair in Data Privacy, International Conference, PSD 2008, Istanbul, Turkey, September 24-26, 208. Proceedings*, 2008.

[40] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 815–823, 2015.

[41] B. Schlkopf and A. J. Smola. *Learning With Kernels: Support Vector Machines, Regularization, Optimization and Beyond*. MIT Press, 2002.

[42] R. Shokri and V. Shmatikov. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 1310–1321, 2015.

[43] N. P. Smart and F. Vercauteren. Fully homomorphic SIMD operations. *IACR Cryptology ePrint Archive*, 2011:133, 2011.

[44] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *Public Key Cryptography - PKC 2010, 13th International Conference on Practice and Theory in Public Key Cryptography, Paris, France, May 26-28, 2010. Proceedings*, 2010.

[45] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, 2012.

[46] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings*, 2010.

[47] D. Wu and J. Haven. Using homomorphic encryption for large scale statistical analysis. 2012.

[48] P. Xie, M. Bilenko, T. Finley, R. Gilad-Bachrach, K. E. Lauter, and M. Naehrig. Crypto-nets: Neural networks over encrypted data. *CoRR*, abs/1412.6181, 2014.

[49] A. C. Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*, pages 160–164, 1982.

[50] J. Yuan and S. Yu. Privacy preserving back-propagation neural network learning made practical with cloud computing. *IEEE Trans. Parallel Distrib. Syst.*, 25(1):212–221, 2014.

[51] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I*, pages 818–833, 2014.

[52] Q. Zhang, L. T. Yang, and Z. Chen. Privacy preserving deep computation model on cloud for big data feature learning. *IEEE Trans. Computers*, 65(5):1351–1362, 2016.

APPENDIX A
FROM NEURAL NETWORKS TO CONVOLUTIONAL NEURAL NETWORK

The simplest neural network is the *perceptron* (see Figure 14) which takes some binary inputs $\vec{x}$ and produces one binary output (aka *label*). To each bit coordinate of $\vec{x}$, say x_j, is attached a weight, say $\omega_j \in \mathbb{R}$, expressing the importance of the coordinate for the output value. During the training phase, the weights are *learned*, together with a bias $\beta \in \mathbb{R}$ (e.g. thanks to linear regression or *gradient descent* processing). The classification of a new input $\vec{x}$ (*i.e.* the assignment of a label $h(\vec{x})$) is afterwards done by applying the following *perceptron rule*: $h(\vec{x}) = \begin{cases} 0 & \text{if } \sum_j \omega_j x_j + \beta \leq 0 \\ 1 & \text{if } \sum_j \omega_j x_j + \beta > 0 \end{cases}$.

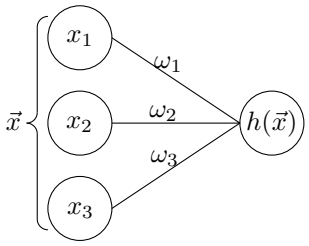

Fig. 14: Perceptron

The use of a single perceptron only serves to classify data that are linearly separable. The *multilayer perceptron* (MLP) aims to deal with this limitation thanks to the combining of several perceptrons and the adding of hidden/internal layers with non-linear *activation function* such as the *sigmoid* $f(x) = (1 + e^{-x})^{-1}$ or, more recently, the *rectified linear unit* (ReLU) $f(x) = max(0, x)$. In a MLP, the output of a perceptron is $f(\sum_j \omega_j x_j + \beta)$ where f is the non-linear activation function. To model high-level abstraction of data, the MLP is composed of several successive layers (one input layer, one output layer and one or more internal – aka hidden – layers) where each node in one layer is linked to every nodes in the next layer (see Figure 15). The output layer $h(\vec{x})$ contains as many neurons as labels and these neurons represent scores for each

label. For instance, for the digit recognition, the output layer contains 10 neurons (one for each possible digits) and each output neuron represents the score for one digit label. The index of the largest output neuron indicates the output label. In order to interpret the output layer as probabilities, a *softmax activation* layer is often added at the end of a MLP. This layer transforms a layer containing n neurons $x_1, ..., x_n$ into a layer containing the same number of neurons $y_1, ..., y_n$ with the following equations: $\forall i \in [1..n], y_i = h(x_i) \doteq \frac{e^{x_i}}{\sum_{j \in [1..n]} e^{x_j}}$.

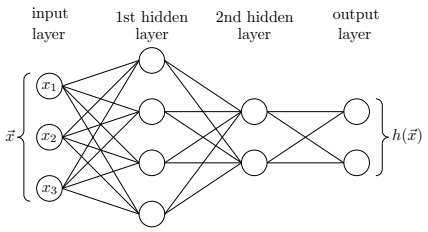

Fig. 15: Multilayer perceptron

APPENDIX B
TRAINING AND CLASSIFICATION PHASES

Like other classical neural networks, CNN is composed of two stages: the *training phase* from a labelled database and the *classification* of new data. Before the training phase, the data scientist has to choose the architecture of the neural network (number of layers, number of neurons per layer, activation function, etc) and some parameters for the learning algorithm called *hyper-parameters* (e.g. number of epochs, batch size, learning rate, etc). Then, the neural parameters (weights and biases) are inferred during the training phase from the labelled database and are adapted to the training objective (recognize/classify digits, faces, objects, etc). They are initialized with random values and are afterwards updated by applying several times the same process. For each *epoch* (one iteration into the entire training database), the training database is randomly partitioned into small sets of data called *batches*. Each batch has the same number of data called *batch size*. For each batch, its data go through the neural network to obtain their output scores with the current neural network. This step is called *feed-forward*. Then an error score is computed to measure the mismatch between the output scores and the expected ones on this batch. We will denote the expected scores of one training datum by a n-dimensional vector where n is the number of possible labels. This expected vector contains a 1 in the position corresponding to its label and zeros on the remaining positions. For instance, in the digit recognition, the expected vector for the digit 0 is $y = (1, 0, 0, 0, 0, 0, 0, 0, 0, 0)$ and the one for the digit 6 is $y = (0, 0, 0, 0, 0, 0, 1, 0, 0, 0)$. The *mean-squared error* is widely used which is equal to $MSE = \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} (h(\vec{x_i})_j - y_{i,j})^2$ where m denotes the batch size, n the number of possible labels, $y_{i,j}$ is the j-th value of the expected labels vector of the i-th training datum and $h(\vec{x_i})_j$ is the score contained in the j-th output neuron of the CNN for the i-th input vector $\vec{x_i}$. Today, the *cross-entropy cost* function, combined with the softmax activation function, is sometimes preferred to the MSE (see e.g. [21]). The cross-entropy cost is equal to $CE = -\frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} y_{i,j} \log h(\vec{x_i})_j$. Finally the error is reduced by modifying some parameters according to the *gradient descent* [34, Chapter 2]. The *learning rate* controls the degree of modifications of the parameters. This parameter updating is the well-known *back propagation* step. Then the feed-forward and back propagation steps are applied on the next batch. Once the pre-defined number of epochs is achieved, the training is finished and the last updates of the weights and of the biases are stored; the neural network is ready to assign labels to never seen data. The classification of a new datum $\vec{x}$ now simply consists in applying the feed-forward step.

The main difficulty for a data scientist is to choose, during the training phase, all the parameters discussed previously (in particular the neural network architecture and the hyper-parameters) in order to achieve a satisfying accuracy. Many heuristics exist but they are rarely sufficient alone and the data scientist know-how is often crucial. About the hyper-parameters, some heuristics [3], [4], [45] to improve them from one learning to another learning have been suggested. They are however rarely sufficient and the data scientist know-how is often crucial: as mentioned in

[34, Chapter 3], " *[the] difficulty of choosing hyper-parameters is exacerbated by the fact that the lore about how to choose hyper-parameters is widely spread, across many research papers and software programs, and often is only available inside the heads of individual practitioners*". About the neural network architecture, the authors try in [51] to understand the influence of each layer on the neural network accuracy in order to build new neural networks that outperform current ones. To do that they introduce a visualization technique that gives insight to understand the influence of intermediate layers. They apply their visualization techniques on the Krizhevsky *et al.*s architecture [30] and they deduce from this analysis some small modifications in the neural network architecture in order to improve its accuracy. These visualizations techniques may permit to improve a bit a neural network architecture but does not permit outstanding improvements and does not deliver information to build a neural network architecture from scratch.

APPENDIX C
PRIVACY-PRESERVING PROCESSING OF THE TRANING PHASE

The two main security problematics in the area of deep neural networks are the privacy preserving training and the privacy preserving classification. The training step (without security) is not easy for a data scientist due to all the factors he has to choose: the architecture of the neural network (number of layers, number of neurons for each layer, activation functions, etc), the initialization of the weights and biases and the hyper-parameters (parameters for the learning algorithm). Due to all these factors, it is almost impossible to obtain a trained neural network with sufficient accuracy at the first try (with the first trained neural network). The data scientist has to analyse, during the training phase, the progress of his neural network in order to understand why he fails, and then he has to modify the factors to obtain better accuracy in his next try. This analysis needs to be made on plaintext observations (without encryption) which makes privacy preserving training almost impossible for complex classification problems. Several articles [1], [12], [39], [50], [52] suggest some methods to secure the first try of the data scientist but they only apply to simple classification problems and become ineffective when the first try is not sufficient (which is often the case in practice).

Articles on the Privacy-reserving Learning subject can be split according to the database fragmentation (decomposition of the database into multiple smaller units called fragments). In a *relational database*, data are structured under the form of a matrix: each row corresponds to a subject (e.g. a patient) and each column to an attribute (e.g. age, race, blood type). There exist three types of database fragmentation: horizontal, vertical or arbitrary fragmentation. A database is *horizontally partitioned*, if the database is split into rows. That means that if a fragment contains one attribute of one subject, he has also all the attributes of this subject (the entire row). On the contrary, a database is *vertically partitioned*, if the database is split into columns. That means that if a fragment contains one attribute of one subject, he has also the values of this attribute for all subjects (the entire column). Finally a database is *arbitrary partitioned*, if it is a mixed of vertical and horizontal fragmentations.

In 2008, Schlitter [39] introduced a privacy preserving neural network learning protocol for horizontally partitioned database. This protocol, based on additive secret sharing, enables several parties to jointly process a neural network learning without leaking information about their respective data. In this protocol, each party performs almost all the computations (excluding weight updating) on their own data in the *plaintext domain* and they put together their results to update the weights thanks to the secret sharing properties. This protocol suffers from one major drawback. At each iteration, all the updating weights are revealed to all the participants which may leak information about the training data. To overcome this, Shokri and Shmatikov [42] enhanced the scheme by proposing a privacy preserving neural network learning protocol for horizontally partitioned database with reduced information leakage. In their protocol, each party obtains some parameters by performing computations on their own database but instead of sharing all parameters to other parties, they only share a small fraction of

them. Then the weights are updated from these shared parameters. This method has a trade-off between classification accuracy and privacy. Higher is the number of shared parameters, better is the classification accuracy but lower is the privacy.

In vertically partitioned databases, the elements of one row belongs to different parties. It is therefore no longer possible to process the feed-forward step in the plaintext domain because it inputs a full row. Chen and Zhong [12] proposed a scheme that provides a privacy preserving neural network learning protocol for vertically partitioned database. This scheme protects training data and intermediate results, and supports only two parties. This protocol is also based on additive secret sharing. All data and parameters are additively shared between the two parties and the computations are performed thanks to the secret sharing properties. Only the sigmoid activation function cannot be efficiently evaluated with the secret sharing properties due to its high non linearity. To overcome this issue, the sigmoid activation function is replaced by a piecewise linear approximation. Bansal *et al* [1] enhanced this scheme by proposing a privacy preserving neural network learning protocol for arbitrary partitioned database between two parties. Like the work of Chen and Zhong, this protocol only supports two-party scenario. Unfortunately directly extending them to the multi-party scenario drastically increases the communication and computation complexity.

To overcome this limitation, Yuan and Yu [50] proposed to securely outsource the computation to a cloud thanks to homomorphic encryption. Each party encrypts his data before sending them to the cloud. Then the cloud performs most of the learning operations over the ciphertexts and returns the encrypted results to the parties. To be efficiently evaluated over the ciphertexts, the sigmoid function is approximated by a low degree polynomial obtained through Maclaurin series. In addition, Yuan and Yu suggest to use the BGN homomorphic encryption [5] which supports one multiplication and unlimited number of additions. Thus the participants have to decrypt and re-encrypt intermediate values after each multiplication. For privacy preservation, parties decrypt only random shares of the intermediate values thanks to secret sharing method. This protocol is not really efficient for deep neural network because of the multitude of decryptions and re-encryptions. Zhang *et al* [52] enhanced this protocol by using a more efficient homomorphic encryption scheme: the BGV scheme [9] which enables unlimited number of multiplications and additions over ciphertexts. But the efficiency of the homomorphic computations depends on the *multiplicative depth*. To avoid a multiplicative depth too big, after each iteration the updated weights are sent to the parties to be decrypted and re-encrypted. Thus the communication complexity of the solution is very high.

In this article, we will not study the privacy preserving learning problem but we will focus on the privacy preserving classification problem.