Crypt-CNN(II): Crytographically Evaluate Non-linear Convolutional Neural Network

Wen-jie Lu[1] Jun Sakuma[1,2,3]

Abstract: Deep learning methods such as convolutional neural networks have brought breakthroughs in many machine learning tasks. In this paper, we present a protocol for privately evaluating non-linear convolutional neural networks. The protocol operates under the secure two-party setting with semi-honest adversaries, where the service provider holds a convolutional neural network model, and the client provides a query. At the end of the protocol execution, the client learns only the prediction output on his query along with few descriptive parameters of the model; the service provider learns nothing. We implement the protocol and show that it can process a non-linear convolutional network with eight hidden layers in one minute and a modest amount of bandwidth. Compared to the previous private CNN evaluation protocol, we demonstrate a ten-fold improvement in computation time, with about 32% higher prediction accuracy.

Keywords: PWS, Machine Learning, Secure Two-party Computation, Garbled Circuit, Hybrid Solution.

1. Introduction

The recent explosive evolution of deep learning research has led to breakthroughs in many machine learning tasks. The application area covers not only image and speech recognition but also diverse types of predictive and cognitive modeling. The unprecedented accuracy of deep learning models enables various novel services that might have marked effects on our society, for example, human virus detection [7], and drug discovery [3]. Such highly influential services often involve treatment of private or confidential information. For a research institute (service provider, SP for short) hosting a deep learning-based prediction service that forecasts pharmacological activities of chemical compounds upon requests from clients, and presuming that a pharmaceutical company (client) wishes to issue queries to investigate pharmacological activities of chemical compounds under development, then in view of the client, the query compounds are regarded as extremely confidential information. Consequently, clients might not wish to reveal the contents of their queries for some application domains.

Considered from the opposite point of view, the SP might require privacy for deep learning models. Typically, tremendous amounts of resources must be used to build the predictive model. In addition, the model details often include commercial secrets of the SP. Consequently, the SP must take pains to ensure that the model details are not leaked through the entire process of the forecasting service. Additionally, the deep learning model might have been trained from sensitive data or confidential data that should not be released publicly. Such sensitive or confidential data might be recovered from the models through reverse engineering [10]. Consequently, revealing the particular model parameters might increase risks of violating privacy and confidentiality.

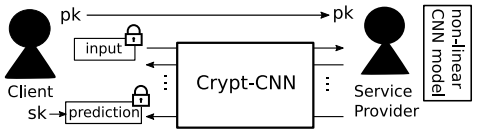

Fig. 1 Crypt-CNN.

In this work, we specifically examine one popularly used deep learning-based classifier: convolutional neural networks (CNNs). The initial concept of CNNs dates back several decades [21]. Their prediction performance has improved rapidly in recent years by virtue of highly efficient computation provided by GPGPU, particularly in areas such as human face detection and object recognition. Recent literature [29], [31] has included reports that the precision of image recognition by the state-of-the-art CNN exceeds the precision of image recognition by humans.

We present Crypt-CNN, an interactive and practical protocol that privately evaluates nonlinear CNNs (Fig. 1). In our setting, a CNN model is privately held by a SP. The client provides private input to the model for prediction. Our desired security property is that, in the phase of prediction, after the execution of Crypt-CNN, the SP should not learn anything about the client's input (input privacy). The client should not learn anything about the SP's model except what can be inferred directly from the prediction result (model privacy).

Related Work. The problem of private evaluation of

[1] University of Tsukuba
[2] JST CREST
[3] RIKEN center of AIP

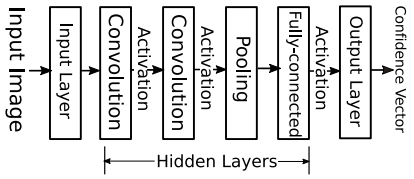

Fig. 2 Six-layered neural network with four hidden layers.

CNNs falls under the general framework of private evaluation of machine learning classifiers. Many efforts have been made to develop secure two-party computation (2PC) protocols for the private evaluation of the classic machine learning classifiers, which include decision tree [34], k-nearest neighbor classifier [33], support vector machine [5], Fisher linear discriminant classifier, logistic regression [15], and perceptron [2]. Because of the rapid and continuous improvement of deep learning methods, the classification accuracy of deep learning-based classifiers, such as CNN, markedly outperforms that of the classic classifiers. The unprecedented accuracy of the deep learning methods and concerns for data and model privacy motivate researchers to develop private evaluation protocols of deep learning methods.

Gilad-Bachrach et al. [12] presented the first private evaluation protocol for CNNs using fully homomorphic encryption (FHE). Because of the increased complexity in computing nonlinear operations from FHE, Gilad-Bachrach et al. reported replacement and approximation of the nonlinear operations with linear operations. As a result, their approach enables the client to send encrypted input to the SP using FHE, and allows the SP perform all the computation needed in the linear CNN evaluation with no interaction with the client. However, linear CNN is fundamentally a linear classifier. Therefore, the classification performance of the linear CNN is only slightly better than the classic linear classifiers such as support vector machine and logistic regression. This report is the first of the relevant literature to describe a study examining practical protocols for the private evaluation of a large and *nonlinear* CNN.

Another possible approach would be the use of differential privacy techniques [1], [9], [26] which allow the SP to protect the model privacy by introducing random noises. In fact, these techniques are orthogonal to our 2PC approach. Specifically, we can leverage these techniques to enforce protection of the classifier model of the SP.

Our Contributions. We begin by showing that matrix–vector multiplication of two types is necessary to evaluate a CNN. Specifically, for the first type, the client input is a long vector, and the input of the SP is a large-scale matrix. In this case, only a single matrix–vector multiplication is performed. For the second type, the input of the client is a few matrices, and input of the SP is the same number of vectors. In this case, batch evaluation of matrix–vector multiplications is needed.

Then, we present the first private *nonlinear* CNN evaluation protocol by combining our FHE-based building blocks (presented in our first paper) with the garbled circuit in a delicate way. Our hybrid protocol runs in a multi-round manner and provides provable security against semi-honest adversaries. To demonstrate the practicality of our protocol, we conduct experiments with a ten-layer nonlinear CNN model (Table 2). We compare the performance of our protocol against the protocol of Gilad-Bachrach et al. [12]. We demonstrate greater than ten-fold reduction in computation time while providing higher classification accuracy on the same image recognition task, that is, 82.8% accuracy by ours as opposed to 50.2% accuracy of the model of Gilad-Bachrach et al.

2. Preliminaries

2.1 Notation

Notation follows the same notations in our first paper (i.e., Crypt-CNN(I): Secure Two-party Computation of Large-scale Matrix-vector Multiplication)

2.2 Cryptographic Primitives

This section presents details about the cryptographic primitives used in our construction.

Fully Homomorphic Encryption. In this paper, we prefer the Ring-LWE [23] variant of BGV's scheme [6] whose properties enable us to build practical private CNN evaluation protocols. The setup parameters of BGV's scheme consist of three positive integers m, t, and L where t is an exponent of prime values. In our construction, we specify m as an exponent of 2 for efficiency concerns. The message space of this scheme is given as a ring $\mathbb{A}_t := \mathbb{Z}_t[X]/(X^m+1)$.

We give brief descriptions related to the scheme. Let $(\mathsf{sk}, \mathsf{pk})$ be a key pair generated with parameters m, t, and L. For any element $A, B \in \mathbb{A}_t$, we leverage the following properties of BGV's scheme in our construction.

- Asymmetric scheme:
  $\mathsf{Dec}_{\mathsf{sk}}(\mathsf{Enc}_{\mathsf{pk}}(A)) = A \mod (X^m + 1, t)$
- Additive homomorphism:

$$\mathsf{Dec}_{\mathsf{sk}}(\mathsf{Enc}_{\mathsf{pk}}(A) \oplus \mathsf{Enc}_{\mathsf{pk}}(B)) = A + B \mod (X^m + 1, t)$$
$$\mathsf{Dec}_{\mathsf{sk}}(\mathsf{Enc}_{\mathsf{pk}}(A) \oplus B) = A + B \mod (X^m + 1, t)$$

- Multiplication with scalars:

$$\mathsf{Dec}_{\mathsf{sk}}(\mathsf{Enc}_{\mathsf{pk}}(A) \otimes B) = A \times B \mod (X^m + 1, t)$$

The operators $\oplus$ and $\otimes$ respectively indicate homomorphic addition and homomorphic multiplication. Also, we write $\ominus$ to denote the homomorphic subtraction.

Yao's Garbled Circuit. Yao's garbled circuits (GC) were developed for secure two-party computation [16], [35]. The main idea of Yao's garbled circuits is to transform a Boolean circuit $\mathcal{C}, \{0,1\}^{\mathsf{ib}} \to \{0,1\}^{\mathsf{ob}}$ to a garbled circuit $\tilde{\mathcal{C}}$ along with ib pairs of encodings $\{w_i^0, w_i^1\}_{i=0}^{\mathsf{ib}-1}$. Then for any input $\boldsymbol{x} \in \{0,1\}^{\mathsf{ib}}$, the combination of the garbled circuit $\tilde{\mathcal{C}}$ and the encodings $\mathcal{S}_{\boldsymbol{x}} = \{w_i^{(\boldsymbol{x})_i}\}_{i=0}^{\mathsf{ib}-1}$ enable a person to evaluate $\mathcal{C}(\boldsymbol{x})$, and yet reveal nothing else about $\boldsymbol{x}$.

Handcrafting the garbled circuits is an error-prone and time-consuming task. We use automation tools [22], [30]

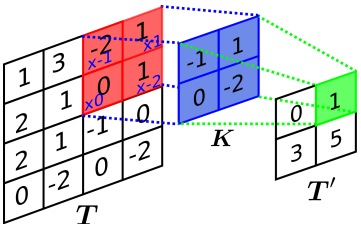

Fig. 3 Convolution on a 2D-matrix with $s = 2$.

for this task. For our experiments, we use the ObliVM library [22] to transform a Java program to the equivalent garbled circuit counterpart.

3. Convolutional Neural Networks

CNNs have been the most influential innovations in the area of pattern recognition and image recognition. Let us take image recognition as an example. The image recognition is a task that classifies an image of an object according to its visual content. The classification performance of CNNs has improved rapidly in recent years. The CNN of Krizhevsky et al. [20] won the 2012 ImageNet competition [27] by dramatically dropping the classification error record from 26% to 15%. Two years later, the CNN model of Simonyan et al. [29] has improved the classification error to 7.32%.

The input image of the CNN can be an RBG picture (i.e., a 3D tensor) or a gray scale picture (i.e., a 2D matrix). Usually, we already know how many categories to classify. For each category, the CNN outputs a real value of 0.0 – 1.0. This value is interpreted as the confidence of classifying the input image to that category. Thereby, we choose the category with maximum confidence as the classification result.

The common architecture of CNNs such as [20], [29] forms a multi-layer neural network structure. Fig. 2 portrays a typical six-layer CNN. Aside from the input layer and the output layer, we can have layers of three kinds: the convolution layer, the pooling layer, and the fully-connected layer. The pooling layer usually comes after the convolution layer. The fully-connected layer is usually placed in the last few layers of the network. Layers that are neither the input layer nor the output layer are called hidden layers. Between two hidden layers, there might be an activation operation. We now give descriptions of the three hidden layers. We also introduce two activation functions that are commonly used for CNNs: rectified linear unit and softmax.

3.1 Convolution Layer

The term CNN indicates that it involves a mathematical operation called convolution. The convolution operation preserves the spatial relations among pixels by learning features of images using small areas of the input images.

We have three arguments for a convolution operation: two tensors $\boldsymbol{T} \in \mathbb{R}^{n\times n\times c}$, $\boldsymbol{K} \in \mathbb{R}^{h\times h\times c}$, and a positive integer s. $\boldsymbol{T}$ is designated as the *input tensor*. $\boldsymbol{K}$ is designated as the *kernel*. s is designated as the *stride*. We consider the following convolution $\boldsymbol{T}' = \mathsf{conv}(\boldsymbol{T}, \boldsymbol{K}, s)$.

$$(\boldsymbol{T}')_{i',j'} = \sum_{k=0}^{c-1}\sum_{i=0}^{h-1}\sum_{j=0}^{h-1}(\boldsymbol{K})_{i,j,k}\cdot(\boldsymbol{T})_{si'+i,sj'+j,k}$$
$$\text{for } i', j' \in [n'] \text{ and } n' = \lfloor (n-h)/s \rfloor + 1.$$

Here $\lfloor\cdot\rfloor$ returns the greatest preceding integer. The output tensor $\boldsymbol{T}'$ is designated as the *feature map*. Fig. 3 depicts a convolution evaluated to a 4×4 matrix.

In the equation above, we use a single kernel. The output forms a matrix. In practice, we prefer to use multiple $\mathcal{K} = \{\boldsymbol{K}_w\}_{w=0}^{c'-1}$ kernels to learn multiple spatial features such as edges and curves. In this case, the convolution layer computes conv *independently* for each $\boldsymbol{K}_w$ with the $\boldsymbol{T}$:

$$\mathsf{Conv}(\boldsymbol{T}, \mathcal{K}, s) = \mathsf{conv}(\boldsymbol{T}, \boldsymbol{K}_0, s), \ldots, \mathsf{conv}(\boldsymbol{T}, \boldsymbol{K}_{w-1}, s) \tag{1}$$

The feature map $\boldsymbol{T}'$ then becomes an $n' \times n' \times c'$ tensor.

3.1.1 Convolution Evaluation Revisit

Our practical building block for the convolution evaluation uses a transformation $\mathcal{T}$, by which the convolution (Eq. 1) is reduced to matrix–vector multiplication. In addition, $\mathcal{T}$ is commonly used in deep learning libraries, as described by Chainer [32].

In our private CNN evaluation setting, the client uses $\mathcal{T}$ as a local pre-processing (with public parameters s and h informed by SP). Given the $n\times n\times c$ tensor $\boldsymbol{T}$ and parameters s and h, the client performs $\mathcal{T}(\boldsymbol{T}, s, h) = \{\boldsymbol{U}_k\}_{k=0}^{c-1}$, where the matrix size is $n'^2 \times h^2$ ($n' = \lfloor (n-h)/s \rfloor$) for each $\boldsymbol{U}_k$. The SP reshapes each channel of the kernel $\boldsymbol{K}_w$ into vectors for $w \in [c]$. Particularly, the SP reshapes the $h\times h\times c$ kernel $\boldsymbol{K}_w$ into c vectors $\{\boldsymbol{v}_{kw}\}_{k=0}^{c-1}$, where the length $|\boldsymbol{v}_{kw}| = h^2$. In fact, Eq. 1 is equivalent to the following computation

$$\sum_{k=0}^{c-1}\boldsymbol{U}_k\boldsymbol{v}_{k0}, \sum_{k=0}^{c-1}\boldsymbol{U}_k\boldsymbol{v}_{k1}, \ldots, \sum_{k=0}^{c-1}\boldsymbol{U}_k\boldsymbol{v}_{kc'-1},$$

which is done by $c \cdot c'$ matrix–vector multiplications. Evaluation of a fully-connected layer is also performing matrix–vector multiplication. Therefore, we can state that matrix–vector multiplication is a fundamental operation for evaluating CNNs.

3.1.2 Pooling Layer

Pooling is an important concept of CNNs. Pooling layers serve to reduce the feature map size, and therefore control overfitting. A common pooling function is *max pooling* [28]. A max pooling layer uses a sliding window to partition each channel of the feature map into many non-overlapping sub-regions, and outputs the maximum values of these sub-regions. Let ρ be the sliding window size. We consider the following max pooling $\hat{\boldsymbol{T}} = \mathsf{Pool}(\boldsymbol{T}', \rho)$:

$$(\hat{\boldsymbol{T}})_{\hat{i},\hat{j},k} = \max\left(\{(\boldsymbol{T}')_{\hat{i}\rho+i,\hat{j}\rho+j,k}\}_{i,j\in[\rho]}\right), \tag{2}$$

where $\hat{i}, \hat{j} \in [\lfloor n'/\rho \rfloor]$, and $k \in [c']$. In other words, the (max) pooling operation sub-samples the $\frac{n'}{\rho}\times\frac{n'}{\rho}\times c'$ tensor $\hat{\boldsymbol{T}}$ from the feature map $\boldsymbol{T}'$. Figure 4 presents an example of a max

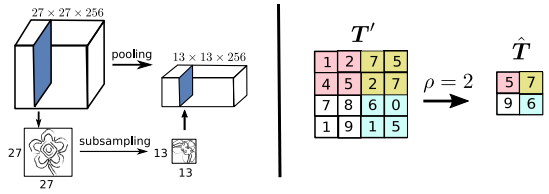

Fig. 4 (Left) Pooling operation sub-samples the feature map but keeps the number of channels unchanged. (Right) Max pooling on a 2D-matrix with $\rho = 2$.

pooling operation. It is noteworthy that the only parameter of a pooling layer is ρ. The pooling does not change the number of channels c'.

3.2 Fully-connected Layer

The input of a fully-connected layer is a vector. We write this vector as $\boldsymbol{x} \in \mathbb{R}^{\hat{n}_1}$, and denote the weight matrix of the layer as $\boldsymbol{W} \in \mathbb{R}^{\hat{n}_2 \times \hat{n}_1}$. To evaluate a fully-connected layer we simply do matrix–vector multiplication

$$\mathsf{FC}(\boldsymbol{W}, \boldsymbol{x}) = \boldsymbol{W}\boldsymbol{x}, \tag{3}$$

which requires $\mathcal{O}(\hat{n}_1 \hat{n}_2)$ additions and multiplications.

Flatten Operation. The output of the convolution layer and pooling layer are tensors, whereas the input to the fully-connected layer is vectors. To connect a convolution layer (or a pooling layer) with a fully-connected layer, we need to "flatten" the output tensor from the convolution layer (or pooling layer). For one of the channels of the tensor (i.e., a 2D-matrix), we concatenate the rows one by one and output a column vector. We process all the channels and output one long vector by concatenating all the column vectors.

3.3 Activation Functions

As described in this paper, we consider two popularly used *nonlinear* activation functions: rectified linear unit (ReLU) [20] and softmax.

ReLU is the most popular activation function used in many modern CNNs. For a real value x, it outputs

$$\mathsf{ReLU}(x) = \max(0, x). \tag{4}$$

For the case of a real vector $\boldsymbol{x}$ or a real tensor $\boldsymbol{T}$, ReLU is separately evaluated on each element of $\boldsymbol{x}$ and $\boldsymbol{T}$. We simply write $\mathsf{ReLU}(\boldsymbol{x})$ and $\mathsf{ReLU}(\boldsymbol{T})$ for this case.

The softmax function is often used in the output layer. Assuming that we have τ categories to classify, the softmax takes as input a real vector $\boldsymbol{y} \in \mathbb{R}^{\tau}$ and outputs a real vector $\boldsymbol{y}'$ such that $(\boldsymbol{y}')_i = e^{(\boldsymbol{y})_i} / \sum_j e^{(\boldsymbol{y})_j}$ for $i \in [\tau]$. The value of $(\boldsymbol{y}')_i$ is interpreted as the probability that the input belongs to the i-th category. We argue that we need not implement the softmax in the domain of secure computation because 1) the softmax is commonly applied as the last operation of a CNN, and 2) what can be inferred from $\boldsymbol{y}'$ is exactly the same as what can be inferred from $\boldsymbol{y}$. Thereby, in our construction, the softmax function is evaluated as the post-process with the plaintext of $\boldsymbol{y}$.

Table 1 Matrix-vector multiplication of two kinds.

	Client		SP	
	Input	Output	Input	Output
MvM-v	$\boldsymbol{v} + \boldsymbol{r}$	$\boldsymbol{M}\boldsymbol{v} + \boldsymbol{r}'$	$\{\boldsymbol{M}, \boldsymbol{r}\}$	$\boldsymbol{r}'$
MvM-m	$\{\boldsymbol{M}_i + \boldsymbol{R}_i\}$	$\sum_i \boldsymbol{M}_i \boldsymbol{v}_i + \boldsymbol{r}'$	$\{\boldsymbol{v}_i, \boldsymbol{R}_i\}$	$\boldsymbol{r}'$

4. Problem Formulation

4.1 Matrix–vector Multiplication of Two Kinds

Even the evaluation of the convolution layer and the evaluation of the fully-connected layer can be accomplished through the matrix–vector multiplication. It is noteworthy to note that the matrix–vector multiplication used in the fully connected layer differs from the matrix–vector multiplication used in the convolution layer. Specifically, for the fully-connected layer, the client's input is a long vector. Only one large scale matrix-vector multiplication is needed for the evaluation. However, the client's input consists of small matrices in the case of the convolution layer. A batch of matrix-vector multiplications must be used. Based on this subtle but significant difference, we design a building block separately for these two layers (see our first paper).

Put abstractly, our building blocks are reduced to the secure 2PCs presented in Table 1. MvM-v signifies the matrix–vector multiplication with a vector as the client's input. We suppose MvM-v is used for the large-scale matrix–vector multiplication in the fully-connected layer. MvM-m denotes the matrix–vector multiplication with matrices as the client's input. We suppose MvM-m is used for the batch matrix–vector multiplication in the convolution layer. In both settings, we assume the client generates the key pair $(\mathsf{sk}, \mathsf{pk})$ while the SP can only access to the public key pk.

4.2 Private CNN Evaluation

We introduce a new notation. For a protocol P, we write $\mathsf{P}(\mathcal{U}, \mathcal{V}) \to (\mathcal{X}, \mathcal{Y})$ to denote an execution of P. We make an arrangement that the first argument (i.e., $\mathcal{U}$) of P denotes the input from the client, and the second argument denotes the input from the SP. $\mathcal{X}$ and $\mathcal{Y}$ denote the output of the client and the output of the SP at the end of the protocol execution, respectively.

We write $\mathbb{W}$ to denote the set of weight matrices used in the CNN, that is, $\mathbb{W} = \{\boldsymbol{W}^j\}$ where $\boldsymbol{W}^j$ is the weight matrix used in the j-th layer, which is a fully-connected layer. Similarly, we write $\mathbb{K} = \{\mathcal{K}^i\}$ to denote the set of kernels used in the CNN, where $\mathcal{K}^i$ is the set of kernels used in the i-th layer, which is a convolution layer. Given $\boldsymbol{T}$ as the private input of the client, our goal is to develop a practical protocol that correctly and privately computes

$$\mathsf{CNN}(\boldsymbol{T}, \{\mathbb{K}, \mathbb{W}\}) \to (\boldsymbol{y}, \emptyset). \tag{5}$$

The client can learn the classification result by applying $\boldsymbol{y}$ to the softmax as a post-processing.

Security Model. We assume that the SP and the client behave semi-honestly [35]. Our protocol reveals descriptive information about the CNN. Precisely, the client knows the CNN structure: the number of layers, the order of layers,

Table 2 Description of the CNN we used.

Layer	Output Size	Activation	Note
Input	$32 \times 32 \times 3$	-	RGB image
Conv-1	$30 \times 30 \times 32$	ReLU	$c' = 32, h = 3, s = 1$
Conv-2	$28 \times 28 \times 32$	ReLU	$c' = 32, h = 3, s = 1$
MaxPool-1	$14 \times 14 \times 32$	-	$\rho = 2$
Conv-3	$13 \times 13 \times 64$	ReLU	$c' = 64, h = 2, s = 1$
Conv-4	$12 \times 12 \times 64$	ReLU	$c' = 64, h = 2, s = 1$
MaxPool-2	$6 \times 6 \times 64$	-	$\rho = 2$
FC-1	512×1	ReLU	2304×512
FC-2	10×1	Softmax	512×10

Algorithm 1 PrivateReLU.

Input of the client: random share of private tensor $\boldsymbol{T}' + \boldsymbol{R}$.
Input of the SP: random share of private tensor $\boldsymbol{R}$.
Output of the client: random share of the output tensor $\mathsf{ReLU}(\boldsymbol{T}') + \boldsymbol{R}'$.
Output of the SP: random share of the output tensor $\boldsymbol{R}'$.
Note: The size of $\boldsymbol{T}', \boldsymbol{R} \in \mathbb{Z}_t^{n' \times n' \times c'}$ is known in advance.

1: The client generates a garbled circuit counterpart $\tilde{\mathcal{C}}$ of Alg. 4, and sends $\tilde{\mathcal{C}}$ (along with the needed information for the circuit evaluation) to the SP.
2: The SP samples the new private share $\boldsymbol{R}' \xleftarrow{\$} \mathbb{Z}_t^{n' \times n' \times c'}$.
3: The client and the SP cooperate to evaluate the garbled circuit $\tilde{\mathcal{C}}$: $\tilde{\mathcal{C}}(\boldsymbol{T}' + \boldsymbol{R}, \{\boldsymbol{R}, \boldsymbol{R}'\}) \to (\mathsf{ReLU}(\boldsymbol{T}') + \boldsymbol{R}', \emptyset)$.

Algorithm 2 PrivateReLUPool.

Input of the client: random share of private tensor $\boldsymbol{T}' + \boldsymbol{R}$.
Input of the SP: random share of private tensor $\boldsymbol{R}$.
Output of the client: random share of the output tensor $\mathsf{ReLU}(\mathsf{Pool}(\boldsymbol{T}', \rho)) + \boldsymbol{R}'$.
Output of the SP: random share of the output tensor $\boldsymbol{R}'$.
Note: The size of $\boldsymbol{T}', \boldsymbol{R} \in \mathbb{Z}_t^{n' \times n' \times c'}$ and ρ are already known.

1: The client generates a garbled circuit counterpart $\tilde{\mathcal{C}}$ of Alg. 5, and sends $\tilde{\mathcal{C}}$ (along with the needed information for the circuit evaluation) to the SP.
2: The SP samples the new share $\boldsymbol{R}' \xleftarrow{\$} \mathbb{Z}_t^{(n'/\rho) \times (n'/\rho) \times c'}$.
3: The client and the SP cooperate to evaluate $\tilde{\mathcal{C}}$:

$$\tilde{\mathcal{C}}(\boldsymbol{T}' + \boldsymbol{R}, \{\boldsymbol{R}, \boldsymbol{R}'\}) \to (\mathsf{ReLU}(\mathsf{Pool}(\boldsymbol{T}', \rho)) + \boldsymbol{R}', \emptyset).$$

and the number of kernels used in convolution layers, the size of pooling window, and the size of fully-connected layers. For our security analysis, we assume that the SP informs these descriptive information related to the CNN to the client in advance. Our security analysis follows the simulation paradigm of [8], [13].

5. ReLU and Max Pooling

We suppose to use GC for the private evaluation of the ReLU and the max pooling. To do so, we need data oblivious [24] algorithms of the ReLU and the max pooling.

We can directly construct data oblivious ReLU and data oblivious max pooling using the multiplexer function. In Appendix A.1, we detail the data oblivious ReLU (Alg. 4) and the data oblivious max pooling (Alg. 5).

5.1 Private ReLU

The private ReLU used in our construction is presented in Alg. 1. In Step 2, the SP samples his new private share $\boldsymbol{R}'$, and applies it to the GC execution in Step 3 so that after the GC execution the ReLU result (e.g., $\mathsf{ReLU}(\boldsymbol{T}')$) is distributed in the form of secret shares.

Algorithm 3 Crypt-CNN.

Input of the client: (public) encryption key pk; (private) the input image $\boldsymbol{T}$ and the decryption key sk.
Input of the SP: (public) descriptive information of its CNN model; (private) the kernels $\{\mathcal{K}^i\}$ and the weight matrices $\{\boldsymbol{W}^j\}$.
Output of the client: $\boldsymbol{y}$.
Output of the SP: $\emptyset$.

1: The client sends the encryption key pk to the SP. The SP sends the descriptive information of the CNN to the client. The client then initializes $\mathsf{T} = \boldsymbol{T}$. The SP initializes $\mathsf{R} = \boldsymbol{0}$.
2: **for** $\eta = 1; \eta < l; \eta = \eta + 1$ **do**
  if the η-th layer is a convolution layer **then**
3: The client and the SP cooperate to run Alg. **??**

$$\mathsf{PrivateMvM\text{-}m}(\{\mathsf{T}, \mathsf{sk}\}, \{\mathsf{R}, \mathcal{K}^\eta\}) \to (\boldsymbol{T}' + \boldsymbol{R}', \boldsymbol{R}'),$$

  else if the η-th layer is a fully-connected layer **then**
4: The client and the SP cooperate to run Alg. **??**

$$\mathsf{PrivateMvM\text{-}v}(\{\mathsf{T}, \mathsf{sk}\}, \{\mathsf{R}, \boldsymbol{W}^\eta\}) \to (\boldsymbol{W}^\eta \boldsymbol{x} + \boldsymbol{r}', \boldsymbol{r}').$$

  if the ReLU is used in the η-th layer **then**
    if the next layer is a max pooling layer **then**
5: The client and the SP cooperate to run Alg. 2

$$\mathsf{PrivateReLUPool}(\mathsf{T}, \mathsf{R}) \to (\mathsf{ReLU}(\mathsf{Pool}(\boldsymbol{T}', \rho_{\eta+1})) + \boldsymbol{R}', \boldsymbol{R}'),$$

where $\rho_{\eta+1}$ is the pooling size used in the η+1-th layer.
    else (that is the next layer is NOT a pooling layer)
6: The client and the SP cooperate to run Alg. 1

$$\mathsf{PrivateReLU}(\mathsf{T}, \mathsf{R}) \to (\mathsf{ReLU}(\boldsymbol{T}') + \boldsymbol{R}', \boldsymbol{R}').$$

7: **end for**
8: To this end, the SP gives the share R to the client, so that the client can obtain $\boldsymbol{y}$ from T and R.

5.2 Private Max Pooling with ReLU

In the typical CNN architecture, max pooling follows after an activation, that is, ReLU in our CNN. We note that the computation result is unchanged no matter we do the ReLU first or do the max pooling first, that is, $\mathsf{Pool}(\mathsf{ReLU}(\boldsymbol{T}), \rho) = \mathsf{ReLU}(\mathsf{Pool}(\boldsymbol{T}, \rho))$ holds. However, the computation complexity of the first one is larger than that of the second one. From a simple calculation, we can know the computation complexity is $2n'^2$ and $n'^2 + n'^2/\rho^2$, respectively. Moreover, both ReLU and max pooling require the max operation only. This motivates us to evaluate $\mathsf{ReLU}(\mathsf{Pool}(\cdot))$ by combining the ReLU operation with the following max pooling operation (if exist).

To be precise, Alg. 5 is a data oblivious algorithm for the $\mathsf{ReLU}(\mathsf{Pool}(\cdot))$. The private evaluation of this function is presented in Alg. 2, in which the pool size ρ is known by assumption. Also, the SP samples the new private share $\boldsymbol{R}'$ and applies it to the GC execution in Step 2.

6. Crypt-CNN

Finally, we present Crypt-CNN, the algorithm of private evaluation of non-linear CNN in Alg. 3. The algorithm is designed by a straightforward combination of the primitives presented in the last section layer by layer.

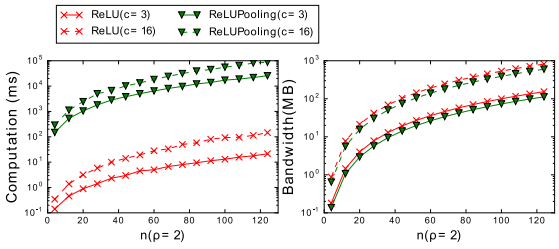

Fig. 5 The end-to-end processing time for ReLU and ReLU(Pool(·)) evaluations, and the bandwidth consummation. The the size of the input tensor was $n \times n \times c$.

In order to privately convey information from layer to layer, we use a local stateful variable R on the SP's side and another stateful variable T on the client's side. T and R store the input of the client and the input of the SP, respectively. Also, these variables store output of the sub-protocols used in the algorithm. For instance, in Step 4, PrivateMvM-v is used to evaluate the fully-connected layer. After the execution of PrivateMvM-v, the client will overwrite its local variable T with $\boldsymbol{W}^{\eta}\boldsymbol{x}+\boldsymbol{r}'$, and the SP will overwrite R with $\boldsymbol{r}'$. These overwritings are implicitly done in Alg. 3.

We remark that the stateful variables T and R can be a tensor or a vector. The client and the SP perform the necessary pre-processing on these variables, according to the type of the current layer. For instance, in Step 4, the current layer is a fully-connected layer, and thus the client flatten T into a vector before calling MvM-v. We omit the descriptions of these pre-processing in Alg. 3.

Theorem 1. *Assume the client and the SP behave semi-honestly. Then Alg. 3 correctly and privately computes Eq. 5. The client learns the confidence vector* $\boldsymbol{y}$ *but nothing else. The SP learns nothing.*

7. Experiments

Settings. We implemented Alg. 1 and Alg. 2 using the Java-based GC library, i.e., ObliVM [22]. For the GC, we used 16-bit fixed point values and used the optimizations such as [17], [18], which are all already provided by ObliVM. All experiment codes were run by machines with a 2.60GHz Xeon E5-2640 v3 processor and 32GB of RAM. The network speed in our experiments was about 940 Mbps. Multi-threads programming was not employed.

We used the parameters $m = 2^{13}, t = 257^4$, and $L = 5$ of BGV's scheme, which provides at least 128-bit security level according to the security analysis of [11]. The combination of these parameters provides $\ell = 128$ plaintext slots and about 32-bit plaintext space. This combination satisfies the requirement of the double-packing. Under this parameters setting, the size of pk was around 5.4 MB and the size of one FHE ciphertext was about 1.2 MB.

Measurements. We measured the computation time and the bandwidth cost for one call of the FHE-based building blocks. The computation time consists of three: time for encryption on the client's side, time for homomorphic operations on SP', and the decryption time on the client's side. The packing on the SP's side can be considered as a pre-processing, and thus was not included in our experiments.

For the GC implementations of the ReLU and the max pooling, we measured the end-to-end running time and the bandwidth cost.

The bandwidth cost consists of the upstream and downstream cost. The upstream cost means the total amount of data that sent by the client, and the downstream cost means the total amount of data sent by the SP.

Datasets. Also, to measure the prediction performance of our CNN (described in Table 2), we trained the CNN on three real datasets, i.e., MNIST [21], SVHN [25] and CIFAR-10 dataset [19]. These training were performed in plaintexts. The prediction accuracy of our CNN on these datasets is about 99.1%, 92.7% and 82.8%, respectively.

7.1 Scalability of the Core Components

The process of our private CNN evaluation protocol is basically a sequential independent calls of four core components, that is, the private evaluation of MvM-v, the private evaluation of MvM-m, and the private evaluation of ReLU and max pooling. Thus, to see the scalability of these components, we separately measured the performance of them with various input sizes.

MvM-v and MvM-m. The performances of our matrix-vector multiplication approaches refer to our first paper.

ReLU and ReLUPool. We present the performance of the private ReLU and ReLU(Pool(·))) evaluations in Fig. 5. The input for these experiments was an $n \times n \times c$ tensor where $c \in \{3, 16\}$ and n was changed from 4 to 124. Also, we fixed the pooling window $\rho = 2$ for the ReLU(Pool(·))) evaluations.

7.2 Private CNN Evaluation

We benchmarked our private CNN evaluation protocol on the pre-trained CNN described in Table 2. We measured the evaluation time and bandwidth for each layer, separately. The results are shown in Table 3, in which the third column and the fourth column denote the amount of data sent by the client and sent by the SP, respectively.

7.3 Discussion and Conclusion

The benchmarks on Fig. 5 show that our GC implementations for the ReLU and ReLUPool were not practical enough when the input tensor is large, e.g., a $124 \times 124 \times 16$ tensor. It cost about 100 seconds and consumed about 1000 MB bandwidth. We remark that our GC implementations are far from optimal and we can use tools such as [4], [30], to generate more compact circuits.

According to the experimental results in Table 3, our private CNN evaluation protocol (Alg. 3) costs less than one minute to privately evaluate a ten-layer non-linear CNN, which is ten times faster than the past work of [12] in which more than 570 seconds were needed to evaluate a six-layer linear CNN. Moreover, we can easily accelerate our protocol by employing the parallel computation in our building blocks. Our protocol consumed slightly more bandwidth

Table 3 Experimental results of our private CNN evaluation protocol. The evaluation time includes the computation time and the communication time.

	Evaluation (ms)	Bandwidth (MB)	
		client → SP	SP → client
Conv-1	339 ± 2.88	15.08	2.23
ReLU	13678 ± 335.64	23.52	78.96
Conv-2	3464 ± 38.34	160.48	2.23
ReLUPool	9218 ± 32.29	12.12	58.07
Conv-3	2355 ± 14.75	71.47	1.12
ReLU	6015 ± 175.55	9.60	32.22
Conv-4	4672 ± 34.05	142.95	1.12
ReLUPool	3395 ± 100.86	3.88	18.59
FC-1	4152 ± 2.77	4.47	10.05
ReLU	524 ± 13.12	0.41	1.37
FC-2	763 ± 0.53	0.56	2.23
Total	48575 ± 750.78	444.54	208.19

than the past work of [12], that was, 65 MB per layer of ours as opposed to 62 MB per layer of [12]. This gap can be shrunk by optimizing the GC of ReLU and ReLUPool.

Comparison with the CryptNets. Gilad-Bachrach et al. [12] presented the first private CNN evaluation protocol, i.e., CryptNets. Our private CNN evaluation protocol is more practical than theirs. Gilad-Bachrach et al.'s approach cost more than 570 seconds to evaluate a six-layer linear CNN. Our protocol cost less than one minute to privately evaluate a ten-layer non-linear CNN.

We also compared the prediction performance of our CNN with the CNN employed by CryptNets using the MNIST, SVHN and CIFAR-10 datasets. The MNIST dataset consists of gray scale images of handwritten digits. The SVHN dataset also consists of color images of digits while these images were sampled from natural street images. CIFAR-10 dataset consists of color images in 10 classes, such as airplane, cat, and horse etc. The task of recognizing natural objects is much harder than the task of recognizing digits.

The experimental results are shown in Table 4. From the results, we can see that our non-linear CNN significantly outperforms the linear CNN of [12], especially on the complicated image recognition tasks, such as CIFAR-10. We remark that the prediction perform ace of our CNN is inferior to the state-of-the-art CNNs. However, our protocol can evaluate a non-linear CNN that provides more than 82% prediction accuracy without compromising the input privacy and the model privacy, at the cost of one minute computation time and 600 MB bandwidth. We thus consider our approach is useful for the CNN-based applications that require both model privacy and input privacy.

Conclusion. We conclude that our FHE-based approaches are practical enough for large-scale matrix-vector multiplication under the secure two-party computation setting. We consider our approaches are useful for the development of secure protocol of evaluating deep learning algorithm such as convolutional neural networks.

Acknowledgment. This work is supported by JST CREST program "Advanced Core Technologies for Big Data Integration".

Table 4 Comparison with the model of CryptNets [12].

	#hidden layers	linearity	prediction accuracy		
			MNIST	SVHN	CIFAR-10
CryptNets	4	linear	99.0%	57.6%	50.2%
ours	8	non-linear	99.1%	92.7%	82.8%
[29] [14] (non-private)	≥ 9	non-linear	99.3%	97.8%	92.4%

References

[1] Martín Abadi, Andy Chu, Ian J. Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 308–318, Vienna, Austria, October 24-28, 2016.

[2] Mauro Barni, Claudio Orlandi, and Alessandro Piva. A privacy-preserving protocol for neural-network-based computation. In *Proceedings of the 8th workshop on Multimedia and security*, pages 146–151, Geneva, Switzerland, September 26 - 27, 2006.

[3] Igor I Baskin, David Winkler, and Igor V Tetko. A renaissance of neural networks in drug discovery. *Expert opinion on drug discovery*, 11(8):785–795, 2016.

[4] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. Efficient garbling from a fixed-key blockcipher. In *2013 IEEE Symposium on Security and Privacy, SP 2013*, pages 478–492, Berkeley, CA, USA, May 19-22, 2013.

[5] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. Machine learning classification over encrypted data. In *22nd Annual Network and Distributed System Security Symposium NDSS*, San Diego, CA, USA, February 8-11, 2015.

[6] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In *Innovations in Theoretical Computer Science 2012*, pages 309–325, Cambridge, MA, USA, January 8-10, 2012.

[7] Gail Brion, Chandramouli Viswanathan, TR Neelakantan, Srinivasa Lingireddy, Rosina Girones, David Lees, Annika Allard, and Apostolos Vantarakis. Artificial neural network prediction of viruses in shellfish. *Applied and environmental microbiology*, 71(9):5244–5253, 2005.

[8] Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.

[9] Cynthia Dwork. Differential privacy. In *33rd International Colloquium on Automata, Languages and Programming, part II (ICALP 2006)*, volume 4052, pages 1–12, Venice, Italy, July 10-14, 2006.

[10] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1322–1333, Denver, CO, USA, October 12-6, 2015.

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

[12] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin E. Lauter, Michael Naehrig, and John 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*, pages 201–210, NY, USA, June 19-24, 2016.

[13] Oded Goldreich. *Foundations of cryptography: volume 2, basic applications*. Cambridge University Press, 2009.

[14] Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, and Vinay D. Shet. Multi-digit number recognition from street view imagery using deep convolutional neural networks. *CoRR*, abs/1312.6082, 2013.

[15] Thore Graepel, Kristin Lauter, and Michael Naehrig. Ml confidential: Machine learning on encrypted data. In *Information Security and Cryptology - ICISC 2012 - 15th International Conference*, pages 1–21, Seoul, Korea, November 28-30, 2012.

[16] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. Faster secure two-party computation using garbled circuits. In *20th USENIX Security Symposium*, San Francisco, CA, USA, August 8-12, 2011.

[17] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference*, pages 145–161, Santa Barbara, CA, USA, August 17-21, 2003.

[18] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Proceedings, Part II*, pages 486–498, Reykjavik, Iceland, July 7-11, 2008.

[19] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey 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.*, pages 1106–1114, Lake Tahoe, Nevada, USA, December 3-6, 2012.

[21] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[22] Chang Liu, Xiao Shaun Wang, Kartik Nayak, Yan Huang, and Elaine Shi. Oblivm: A programming framework for secure computation. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA*, pages 359–376, May 17-21, 2015.

[23] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 1–23, French Riviera, May 30 - June 3, 2010.

[24] Daniele Micciancio. Oblivious data structures: applications to cryptography. In *Proceedings of the 29th Annual ACM Symposium on the Theory of Computing*, pages 456–464, El Paso, Texas, USA, May 4-6, 1997.

[25] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning, 2011*, 2011.

[26] NhatHai Phan, Yue Wang, Xintao Wu, and Dejing Dou. Differential privacy preservation for deep auto-encoders: an application of human behavior prediction. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, pages 1309–1316, Phoenix, Arizona, USA, February 12-17, 2016.

[27] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[28] Dominik Scherer, Andreas C. Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In *Artificial Neural Networks - ICANN 2010 - 20th International Conference. Proceedings, Part III*, pages 92–101, Thessaloniki, Greece, September 15-18, 2010.

[29] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[30] Ebrahim M Songhori, Siam U Hussain, Ahmad-Reza Sadeghi, Thomas Schneider, and Farinaz Koushanfar. Tinygarble: Highly compressed and scalable sequential garbled circuits. In *2015 IEEE Symposium on Security and Privacy, SP 2015*, pages 411–428, San Jose, CA, USA, May 17-21, 2015.

[31] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014*, pages 1701–1708, Columbus, OH, USA, June 23-28, 2014.

[32] Seiya Tokui, Kenta Oono, Shohei Hido, and Justin Clayton. Chainer: a next-generation open source framework for deep learning. In *Proceedings of Workshop on Machine Learning Systems (LearningSys) in the 29th Annual Conference on Neural Information Processing Systems (NIPS)*, December 7-12,2015.

[33] Wai Kit Wong, David Wai-lok Cheung, Ben Kao, and Nikos Mamoulis. Secure knn computation on encrypted databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2009*, pages 139–152, Providence, Rhode Island, USA, June 29 - July 2, 2009.

[34] David J. Wu, Tony Feng, Michael Naehrig, and Kristin E. Lauter. Privately evaluating decision trees and random forests. *PoPETs*, 2016(4):335–355, 2016.

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

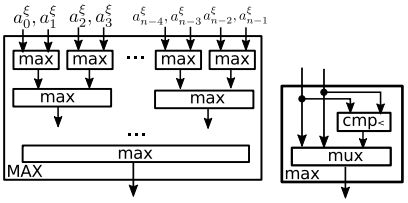

Fig. A·1 Maximum circuit on ξ-bit array $\{a_i^\xi\}_{i=0}^{n-1}$.

Algorithm 4 Data Oblivious ReLU.

Input of the client: private shares $a + r$ and r'.
Input of the SP: private share r.
Output of the client: $\mathsf{ReLU}(a) + r'$.
Output of the SP: $\emptyset$.

1: Compute the subtraction $g = a + r - r$.
2: Compute the comparison $b = \mathbf{1}\{g < 0\}$.
3: Compute the addition $\tilde{a} = g + r'$.
4: Compute $a^* = \mathsf{MUX}(\tilde{a}, r', b)$.
5: Output a^* to the client.

Algorithm 5 Data Oblivious ReLUPool.

Input of the client: private shares $\boldsymbol{T}' + \boldsymbol{R} \in \mathbb{Z}_t^{\rho \times \rho}$.
Input of the SP: private share $\boldsymbol{R} \in \mathbb{Z}_t^{\rho \times \rho}$ and r'.
Output of the client: $\mathsf{ReLU}(\mathsf{Pool}(\boldsymbol{T}', \rho)) + r'$.
Output of the SP: $\emptyset$.

1: For $0 \le i < \rho$
  For $0 \le j < \rho$
    Compute the subtraction $g_{ij} = (\boldsymbol{T}' + \boldsymbol{R})_{i,j} - \boldsymbol{R}_{i,j}$.
2: Compute the maximum value $g = \mathrm{MAX}(\{g_{ij}\})$.
3: Compute the comparison $b = \mathbf{1}\{g < 0\}$.
4: Compute the addition $\tilde{a} = g + r'$.
5: Compute $a^* = \mathsf{MUX}(\tilde{a}, r', b)$.
6: Output a^* to the client.

Appendix

A.1 Data Oblivious Algorithms

We firstly introduce notations. The indicator function $\mathbf{1}\{\mathcal{P}\}$ returns 1 if $\mathcal{P}$ is true. Otherwise, it returns 0. The $\mathrm{MUX}(a, b, c)$ function returns a if bit $c = 0$. Otherwise, it returns b. Here a and b are strings of bits. For two length-ξ bit strings, we can implement this MUX function with ξ non-XOR gates [18].

We now present the data oblivious ReLU in Alg. 4. When $a < 0$, we have $b = 1$, and thus $a^* = r' = \mathsf{ReLU}(a) + r'$. On the other hand, when $a \ge 0$, the MUX function returns $a^* = a + r'$ which is also equivalent to $\mathsf{ReLU}(a) + r'$.

The data oblivious ReLUPool is presented in Alg. 5. From Step 1 to Step 2, we firstly do the max pooling using a MAX circuit (Figure A·1) to compute the maximum value from an array of input. From Step 3 to Step 6, the oblivious ReLU operation is performed. The correctness of this algorithm can be reduced to the correctness of Alg. 4.