2020 7th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2020 6th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)

Deep Binarized Convolutional Neural Network Inferences over Encrypted Data

Junwei Zhou
School of Computer Science and Technology,Wuhan University of Technology, P.R. China
Email: junweizhou@msn.com

Junjiong Li
School of Computer Science and Technology,Wuhan University of Technology, P.R. China
Email: junjiongli@gmail.com

Emmanouil Panaousis
Department of Computing and Information Systems ,University of Greenwich, UK
Email: e.panaousis@gre.ac.uk

Kaitai Liang
Department of Computer Science , University of Surrey, UK
Email: k.liang@surrey.ac.uk

***Abstract*—Homomorphic encryption provides a way to perform deep learning over encrypted data and permits the user to encrypt the data before uploading, leaving the control of data on the user side. However, operations on encrypted data based on homomorphic encryption are time-consuming, especially in a deep convolutional neural network (CNN), which incorporates a large number of layers and operations. To speed up deep learning on encrypted data, we binarized the input data and weights of CNN model, while operations including the addition and multiplication in CNN become bit-wise operations. Therefore, the homomorphic evaluation of CNN can be performed in the binary field in a highly efficient way. We also construct an efficient pooling layer by designing circuits to perform comparison operations on the ciphertext. Simulation results clearly show that the convolution operation of the proposed model is at least 6.3 times faster than that of existing schemes. Last, our model exhibits no privacy leakage associated with the data being processed.**

***Index Terms*—Privacy-preserving, Deep learning, Fully homomorphic encryption, Convolutional neural network,Privacy computing**

I. Introduction

With the development of information technology, the data produced by users is proliferating. While the requirements for data processing increase, the rapid development of machine learning has empowered the large-scale processing and analysis of big data. Benefited from cloud computing and network techniques, users tend to upload their personal data to servers accessing machine learning services, such as face recognition [1], [2], gene diagnosis [3], and financial review [4]. At the same time, a series of privacy concerns arise when the personal data is related to Personal Identifiable Information (PII) introducing the challenge of having to use the data but without losing control of it. Homomorphic encryption [5], [6] allows operations on encrypted data providing a solution to this challenge. However, the high time consumption of homomorphic evaluation on encrypted data remains a challenge, especially for deep learning [7]. The convolutional neural network (CNN) is widely used in image classification [8], [9], natural language processing [10], [11] and so on. Meehan [12] proposed a novel and privacy-preserving CNN model with hybrid fully homomorphic encryption (HFHE) allowing servers operating on encrypted data. Although Meehan's model improves the speed of homomorphic calculations, it still needs 55 hours to classify one image. Hesamifard *et al.* [13] proposed a deep neural network on encrypted data named CryptoDL. They use low degree polynomials as the activation functions in CNN inference with a little precision sacrificing.

Furthermore, the employed Leveled Homomorphic Encryption [14] (LHE) limited the layers and capacity of CNN, since the supported operations in leveled homomorphic encryption are bound. By redefining the TFHE library [15], Bourse *et al.* [16] proposed a deep discretized neural network that significantly improves the classification speed over encrypted data. In this model, only neuron calculation on discretized value is performed on the server-side, leaving some operations on the user side. Moreover, the dimension reduction function, like the pooling layer, is not implemented, which limits the capacity of the CNN.

Since real and discretized values are used in neural networks, all above methods need to binarize the data and weights or employ real numbers supported by homomorphic encryption, which is usually time-consuming. In this paper, instead of using real-valued CNN, we employ a binarized CNN network [17] evaluated directly by bit-wise supported homomorphic encryption, implementing secure CNN inferences to protect the security of users' data.

We leave the network weights as plaintext, while the data is encrypted to protect users' privacy. Since the weights are known to the service provider, we construct a hybrid homomorphic evaluation that can finish bit-wise operations between plaintext weights and ciphertext that do not need bootstrapping operations. Thus, the number of time-consuming bootstrapping operations can be dramatically reduced. Benefited from the binarized implementation, the speed of convolution operation is greatly improved, and storage cost for the model decreases. Compared to the hybrid convolution in [12] under fully homomorphic encryption (FHE), the convolution operation is 6.3 times faster.

978-1-7281-6550-9/20/$31.00 ©2020 IEEE
DOI 10.1109/CSCloud-EdgeCom49738.2020.00035

Fig. 1. The overview of our privacy-preserving deep learning system.

Moreover, we use the homomorphic supported bit-wise operation to realize the comparison operations, which can allow the hybrid homomorphic operations. Based on the comparison operation on ciphertext, we construct a novel max-pooling layer to reduce the data dimension. Especially, we use hybrid homomorphic bit-wise operations to construct the ReLU function [18] on bit-level calculation.

To validate the efficiency of our approach, we construct a binarized perceptron model whose weights and input are binary value under the hybrid homomorphic encryption. We run the binarized perceptron model on a breast cancer dataset. Simulation results show that our model is 3.6× faster than that of Meehan's perception [12] with the same accuracy.

As shown in Fig. 1, our model can be applied to a privacy-preserving deep learning system consisting of two parts: (1) **Service Provider (SP)**: The binarized CNN model is trained and stored on SP which can support homomorphic operations. (2) **Client**: Users perform encryption and decryption. The encrypted data is being uploaded to SP and after it is processed by the CNN inference in the cloud, the SP will send the encrypted result back to the client. In this system, the only client who has the secret key can decrypt the data, and the data is always kept encrypted preventing to large extend any data leakage.

The rest of this paper is organized as follows. In section II, the related work about homomorphic encryption is summarized. In section III, we give the CNN inferences on encrypted data. The experimental results and analysis will be given in section IV, respectively. We conclude our paper in section V.

II. Related Work

Traditional encryption methods do not allow data to be manipulated in the form of ciphertext. In 1987, Rivest proposed the concept of privacy homomorphism in [19]. Despite this, homomorphic encryption had not been widely used because of the noise growth in homomorphic computing and high complexity. Gentry proposed the concept of FHE and provided the bootstrapping operation to reduce noise from ciphertext [5], which is a milestone in the field of homomorphic encryption.

Although FHE permits unlimited operations on encrypted data, the bootstrapping operation has high complexity. The time cost of bootstrapping is still a challenge for FHE used in practical applications. Many scholars try to solve this problem. Recently, benefited from the contribution of scholars in homomorphic encryption, a lot of friendly used implementations have been open-sourced.

The popular open-source named HElib [20] is an implementation of the Brakerski-Gentry-Vaikuntanathan (BGV) scheme [14] which was based on learning with error (LWE) and ring-LWE problems. HElib offers the addition and multiplication operations over encrypted data. To speed up the calculation in HElib, Halevi [21] proposed to use faster linear transformations to reduce the time of bootstrapping operation. The algorithm for linear transformations results in a 6× speedup bootstrapping.

Ducas provided a faster homomorphic encryption scheme named FHEW library [22], which homomorphically performs simple bit operations and bootstrapping. Compare with HElib, FHEW can reduce bootstrapping time cost to less than 0.5 seconds. Besides, FHEW also offers NAND operation for binary numbers. In 2017, Roy [23] realized homomorphic evaluations of arbitrary depth with a re-encryption box, which takes 0.43ms to refresh one ciphertext for reducing noise under the FV homomorphic encryption scheme [24].

Based on FHEW, Chillotti *et al.* proposed a formalization of LWE [25] over the real torus $\mathbb{T}$ and TFHE [15] with a low latency bootstrapping based on the LWE and GSW [26] schemes. TFHE generates the secret key $\boldsymbol{s} \in \mathbb{Z}^n$. The message $m \in \{0,1\}$ is the plaintext. It chooses a random vector $\boldsymbol{a}$ from $\mathbb{T}^n$ and a random noise e from Gaussian distribution ϕ, then calculates $b = [\boldsymbol{a}\cdot\boldsymbol{s} + m + e]_1$. Then message m is converted into a ciphertext vector $\boldsymbol{c} = (\boldsymbol{a}, b)$. In TFHE, the bit-wise operations (such as AND, XNOR) on encrypted bits are evaluated by using a linear combination in the bootstrapping of TFHE. The linear combination outputs a ciphertext $\boldsymbol{c}' = (\boldsymbol{a}', b')$ which ensures that the ciphertext noise is within the allowable range. Based on TFHE, Meehan *et al.* [12] proposed a machine learning model to permit classification on ciphertext with plaintext weights. They used binary NOT gate which doesn't require bootstrapping to construct hybrid bit-wise operation and proposed a hybrid FHE scheme (HFHE). By implementing hybrid bit-wise operations, they can realize addition and multiplication between ciphertext and plaintext at the bit level. Besides, they created a privacy-preserving CNN model whose weights are plaintext and input data is ciphertext. Since HFHE reduced numerous unnecessary bootstrapping operations, the computation speed of addition and multiplication are improved significantly.

Based on these homomorphic encryption schemes, scholars try to develop a security deep learning model. Based on HElib, Hesamifard, *et al.* [13] proposed a technique, named CryptoDL, to construct privacy-preserving deep neural networks. They used low degree polynomials to replace the activation function in CNN, which is trained with approximation polynomials on the plaintext. The evaluation accuracy on the MNIST dataset can reach 99.52%.

By redefining TFHE library [27], Bourse [16] proposed deep discretized neural networks which greatly improved the classification speed over encrypted data. They increased the message space and sign computation in homomorphic evaluation. Unlike CryptoDL, their model is constructed by a neuron calculation. The bootstrapping is used in neurons

to reduce ciphertext noise. After input data is processed by homomorphic neuron calculation, encrypted output data is converted into $\{1, -1\}$. By using such discretized neural networks, they classified encrypted images from the MNIST dataset with an accuracy 96%, and it only costs 1.7s per image.

Since homomorphic operations over ciphertext and bootstrapping are time-consuming, it is a challenge for the practical application of the privacy protection models. Moreover, when the deep learning model becomes more complicated, the time cost of privacy-preserving deep learning model is extremely high, and the storage cost of the model is also enormous. In this paper, we propose a binarized CNN inference over encrypted data with sacrificing a minor degree of accuracy to accelerate the deep CNN and reduce complexity. Since the input and weights of our model are binarized, it is more friendly with TFHE, which is designed for bit-wise operations. Since we leave the model weights as plaintext, we can also construct the hybrid bit-wise operations to avoid time-consuming bootstrapping. We also constructed an efficient pooling layer by designing circuits to perform comparison operations on the ciphertext. Therefore, the speed of the privacy-preserving CNN model can be accelerated and the storage cost can be reduced.

III. The Proposed Model

In this paper, we propose a binarized CNN inference model on encrypted data that binarizes input and model weights. Besides, we use bit-wise operations to construct an efficient ciphertext comparison calculation. The proposed model is constructed by the Binarized Convolutional Layer, Max Pooling Layer, and Fully Connection Layer. We also implement hybrid homomorphic bit-wise operations to construct lossless ReLU function as the activation function.

The structure of the deep binarized CNN model is shown in Fig. 2. Two preprocessings including **Batchnorm** and **Binactive** with **Encrypt** are performed in the client-side. **Batchnorm** is a regularization operation in the binarized CNN, which can help improve model accuracy. **Binactive** operation is used to binarize the original data to improve the speed of arithmetic operations in CNN model. We implement **Encrypt** by fast FHE over the Torus to encrypt user's privacy data. After that, the user uploads the encrypted data to SP. When the encrypted data is performed in binarized CNN inference, SP returns the encrypted result to the client without privacy leakage.

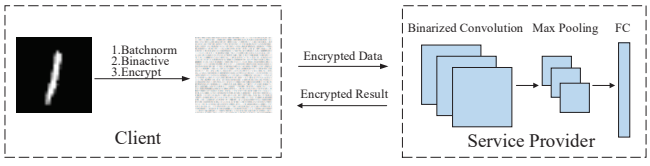

Fig. 2. The structure of our binarized CNN inference.

A. Client

Suppose the input data is an image matrix, and the pixel is a gray value in the range of 0 and 255. Two preprocessings including **Batchnorm** and **Binactive** with **Encrypt** are performed on the input image in the client.

Batchnorm: CNN usually needs a regularization on the input image to improve the performance [28]. This low complexity step can be performed on the client, which can reduce ciphertext calculation in the cloud. For each pixel value m in the image, its normalized value n=$\gamma \times \Phi(m)$+β, where β and γ are two trained weights in batch normalization layer. Here we define $\Phi(x)$ is a normal distribution. By performing **Batchnorm** on the original image, we can get float matrix $\boldsymbol{I} \in \mathbb{R}^{w_{in} \times h_{in}}$, where (w_{in}, h_{in}) represent the input width and height. This matrix $\boldsymbol{I}$ is the input of the **Binactive**.

Binactive: In this step, we convert $\boldsymbol{I}$ into float matrix $\boldsymbol{K}$ and sign matrix *sign*($\boldsymbol{I}$). According to the sign of input $\boldsymbol{I}$, we can get the binary sign matrix *sign*($\boldsymbol{I}$), where binary values 0 and 1 represent negative sign and positive sign, respectively. For $0 \leq i \leq w_{in}$ and $0 \leq j \leq h_{in}$, we compute a matrix $\boldsymbol{A}$, where $\boldsymbol{A}_{ij}$=$|\boldsymbol{I}_{ij}|$. Then we convolve $\boldsymbol{A}$ with a 2D filter $\boldsymbol{k} \in \mathbb{R}^{w_k \times h_k}$, where w_k, h_k represents filter's weight and height, each value in $\boldsymbol{k}$ is $\frac{1}{w_k \times h_k}$. Float matrix $\boldsymbol{K} = \boldsymbol{A} * \boldsymbol{k}$, where $*$ is convolution operation.

Encrypt: we define **Encrypt** operation $Enc(\cdot)$ to encrypt message m as a TLWE ciphertext(TLWE is a generalization of LWE and Ring-LWE [15]): generate the secret key $\boldsymbol{s}$, random vector $\boldsymbol{a}$ and noise e, calculate b=$[\boldsymbol{a} \cdot \boldsymbol{s} + m + e]_1$ and ciphertext $\boldsymbol{c}$=($\boldsymbol{a}$,b). Every element $m \in$ *sign*($\boldsymbol{I}$) belongs to $\{0, 1\}$. For $0 \leq i \leq w_{in}$ and $0 \leq j \leq h_{in}$, we perform $Enc(\boldsymbol{s}, m_{ij}) \rightarrow \boldsymbol{c}$. After **Encrypt** operation we get encrypted matrix [*sign*($\boldsymbol{I}$)] for subsequent calculations.

Since the element in matrix $\boldsymbol{K}$ is float, we can't encrypt the matrix directly. We need to quantize all the values of matrix $\boldsymbol{K}$ and get the ten most important bits of the values. After this step, values in float matrix $\boldsymbol{K}$ are converted into 10-bit precision. Then we use $Enc(\cdot)$ to encrypt matrix $\boldsymbol{K}$ and get encrypted matrix [$\boldsymbol{K}$].

After performing **Batchnorm**, **Binactive**, and **Encrypt** operations on the input data, the user can get ciphertext [$\boldsymbol{K}$] and [*sign*($\boldsymbol{I}$)] and uploads it to SP as the input of binarized CNN model.

B. Service Provider

SP trains its model on unencrypted data. After the model is trained, the SP uses the users' encrypted data to perform binarized CNN inference, which is constructed by Binarized Convolutional Layer, Max Pooling Layer, and Fully Connection Layer.

The input data $\boldsymbol{I}$ and model's weights $\boldsymbol{W}$ are converted into binary values, respectively. We implement **BinaryForward**$(\cdot)$ from [17] for forward propagation, whose weights and input are binary values. Then, we get result $\hat{\boldsymbol{Y}}$=**BinaryForward**$(\boldsymbol{I}, \boldsymbol{W})$. In back propagation, with cost function $\boldsymbol{C}(\hat{\boldsymbol{Y}}, \boldsymbol{Y})$ and the gradient descent method, we can

optimize the binarized CNN model. Then the optimized weight $\boldsymbol{W}$ will be used in the next forward propagation.

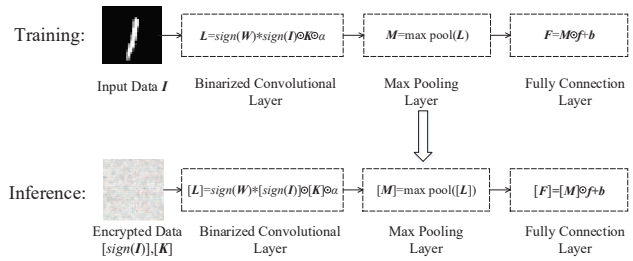

Fig. 3. Training and inference on SP side.

As shown in Fig. 3, in the training phase, SP trains the binarized CNN model on plaintext text with a three-layer structure. After data is encrypted in the client-side, SP accepts the users' encrypted data to perform binarized CNN inference, which can support homomorphic operations.

1) Binarized CNN Model in Training: Our binarized CNN model includes a Binarized Convolutional Layer, Max Pooling Layer, Fully Connection Layer. We train our model with mini-batch algorithm [29] on plaintext. Here our loss function is cross entropy-loss $\boldsymbol{C}$.

Binarized Convolutional Layer: In this layer, inputs and weights are binarized. Comparing with the traditional convolutional layer [30] which uses float number as weights, it uses efficient bit calculation instead of multiplication. Therefore, Binarized Convolutional Layer is faster. With initialized weights $\boldsymbol{W}$, we approximate $\boldsymbol{W}$ with a sign matrix *sign*($\boldsymbol{W}$) and a float parameter α as follows: (1) We construct the binary weight matrix *sign*($\boldsymbol{W}$) by extracting the sign of weight $\boldsymbol{W}$, where binary values 0 and 1 represent negative sign and positive sign values in $\boldsymbol{W}$, respectively. (2) We get the absolute value of each number in $\boldsymbol{W}$ and calculate the average: for $0\leq i \leq w$, $0\leq j \leq h$ and $0\leq k \leq c$, we define $\alpha = \frac{\sum |\boldsymbol{W}_{ijk}|}{c\times w\times h}$. With matrix sign($\boldsymbol{I}$) and $\boldsymbol{K}$ from client-side, float point convolutional operation can be approximated as followings:

$$\boldsymbol{I} * \boldsymbol{W} \approx sign(\boldsymbol{I}) * sign(\boldsymbol{W}) \odot \boldsymbol{K} \odot \alpha \tag{1}$$

In this formula, we define $*$ and $\odot$ are convolution and dot product operations, respectively. And we define output matrix $\boldsymbol{L}$=*sign*($\boldsymbol{I}$)$*$*sign*($\boldsymbol{W}$)$\odot\boldsymbol{K}\odot\alpha$.

Max Pooling Layer: This layer is a sample-based discretization process. It provides an abstracted form of representation to solve the over-fitting problem. Besides, it reduces the computational cost by reducing the number of parameters to learn. In this layer, we apply a 2×2 max filter to non-overlapping subregions of the input $\boldsymbol{L}$. In the regions represented by the filter, we can calculate the max of the down-sampling area and get the maximum value. Then, we get the matrix $\boldsymbol{M}$ as the output of this layer, every element of $\boldsymbol{M}$ is the max of a region from the input matrix $\boldsymbol{L}$.

Fully Connection Layer: The role of this layer is to reorganize the local features extracted from the previous layers through the weights $\boldsymbol{f}$ and $\boldsymbol{b}$. The input of this layer is the matrix $\boldsymbol{M}$ from Max Pooling Layer. And we calculate the output matrix $\boldsymbol{F}$=$\boldsymbol{M}\odot\boldsymbol{f}+\boldsymbol{b}$, where we use $\odot$ to represent dot product operation.

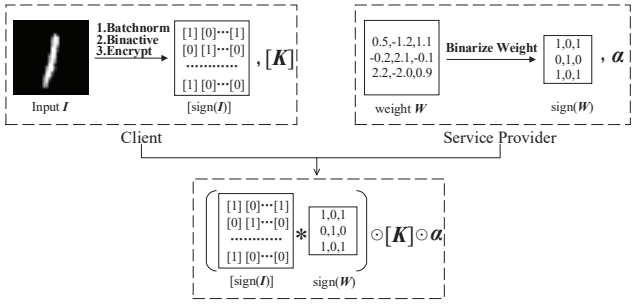

Fig. 4. The structure of binarized convolution.

2) Binarized Convolutional Layer in Inference: Different from processing on the plaintext, our binarized CNN inference model accepts the encrypted data from the user side, which is implemented under bit-wise operations on plaintext and ciphertext. From client-side, SP receives [*sign*($\boldsymbol{I}$)] and [$\boldsymbol{K}$] as this layer's input. From the training phase, we can get binarized *sign*($\boldsymbol{W}$) and α as weights. Since the homomorphic calculation requires support bit-wise operations, we convert float number α into a 10-bit vector $\boldsymbol{\alpha}$ keeping the 10 most important bits of α.

Then, the process of this layer can be represented as the following formula:

$$[\boldsymbol{L}] = [sign(\boldsymbol{I})] * sign(\boldsymbol{W}) \odot [\boldsymbol{K}] \odot \boldsymbol{\alpha} \tag{2}$$

Similarly, we define $*$ and $\odot$ are convolution and dot product operations, respectively.

As shown in Fig. 4, on the SP side, two key steps in Binarized Convolutional Layer are **Dot Product Operation**$(\cdot)$ between [$\boldsymbol{K}$] and $\boldsymbol{\alpha}$, as well as **Binarized Convolution Operation**$(\cdot)$ between *sign*($\boldsymbol{W}$), [*sign*($\boldsymbol{I}$)].

Binarized Convolution Operation$(\cdot)$: In this operation, input are $n\times n$ [*sign*($\boldsymbol{I}$)], $c\times m\times m$ *sign*($\boldsymbol{W}$), and output is an encrypted matrix $c\times p\times p$ [$\boldsymbol{S}$], where $p=n-m+1$. For the ith channel in *sign*($\boldsymbol{W}$), we can calculate [$\boldsymbol{S}_{i,j,k}$] as followings(i,j,k represent the subscripts of [$\boldsymbol{S}$]):

$$[\boldsymbol{S}_{i,j,k}] = \sum_{a=0}^{m}\sum_{b=0}^{m} \mathbf{ep}(\mathbf{h\text{-}xnor}([sign(\boldsymbol{I}_{i+a,j+b})], sign(\boldsymbol{W}_{i,a,b}))) \tag{3}$$

We use **h-xnor**$(\cdot)$ to perform homomorphic calculation between [*sign*($\boldsymbol{I}$)] and *sign*($\boldsymbol{W}$), and we get 1-TLWE $m\times m$ encrypted matrix [$\boldsymbol{E}$]. Then we use **ep**$(\cdot)$ to convert every element in matrix [$\boldsymbol{E}$] into 10-TLWE ciphertext. Finally, we use **Adder**$(\cdot)$ to complete $\sum$ function in the above formula, and sum all ciphertext in [$\boldsymbol{E}$] and get the ciphertext [$\boldsymbol{S}_{i,j,k}$].

Since our model's weights are plaintext and input data is encrypted, our bit-wise operations should support homomorphic calculation between plaintext and ciphertext. Here, we design

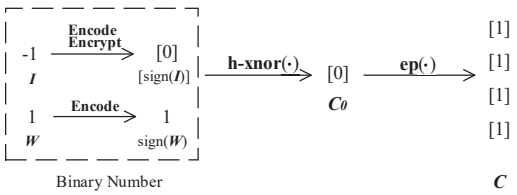

Fig. 5. Operations on binary number.

hybrid xnor operation(**h-xnor**$(\cdot)$), hybrid and operation(**h-and**$(\cdot)$), hybrid xor operation(**h-xor**$(\cdot)$), not operation(**not**$(\cdot)$) and constant operation(**constant**$(\cdot)$). In these operations, inputs are ciphertext $\boldsymbol{C}$ and plaintext P. We define $P \in \{0,1\}$ and ciphertext $\boldsymbol{C}$ can be defined from $Enc(m, \boldsymbol{s})$, where m is the message $\in \{0,1\}$ and $\boldsymbol{s}$ represents secret key. Here, $\boldsymbol{C}$=$Enc(m, \boldsymbol{s})$=$(\boldsymbol{a}, b)$, where $\boldsymbol{a}$ is a vector and b is an integer. We define **h-xnor**$(\cdot)$, **h-and**$(\cdot)$, **h-xor**$(\cdot)$, **not**$(\cdot)$, and **constant**$(\cdot)$ as following formulas:

$$\textbf{h-xnor}(\boldsymbol{C}, P) = \begin{cases} (-\boldsymbol{a}, -b) & \text{If } P = 0 \\ (\boldsymbol{a}, b) & else \end{cases} \tag{4}$$

$$\textbf{h-xor}(\boldsymbol{C}, P) = \begin{cases} (\boldsymbol{a}, b) & \text{If } P = 0 \\ (-\boldsymbol{a}, -b) & else \end{cases} \tag{5}$$

$$\textbf{h-and}(\boldsymbol{C}, P) = \begin{cases} (\boldsymbol{0}, 0) & \text{If } P = 0 \\ (\boldsymbol{a}, b) & else \end{cases} \tag{6}$$

$$\textbf{not}(\boldsymbol{C}) = (-\boldsymbol{a}, -b) \tag{7}$$

$$\textbf{constant}(\boldsymbol{C}) = (\boldsymbol{0}, 0) \tag{8}$$

In the above formula we use $(-\boldsymbol{a}, -b)$ to represent the result of negating vector $\boldsymbol{a}$ and the integer b. And we use $(\boldsymbol{0}, 0)$ to represent that $\boldsymbol{a}$ is converted into vector $\boldsymbol{0}$, integer b is converted into integer 0.

ep$(\cdot)$: After **h-xnor**$(\cdot)$ between $sign(\boldsymbol{W})$ and $[sign(\boldsymbol{I})]$, we get 1-TLWE ciphertext that is not compatible with subsequent addition calculations. This operation is responsible for converting 1-TLWE ciphertext into N-TLWE ciphertext. Here we define the input of **ep**$(\cdot)$ is 1-TLWE ciphertext $[a]$ and the output is N-TLWE ciphertext $[b]$. The operation of **ep**$(\cdot)$ is described in **Algorithm 1**, where m is the loop variable and $[temp]$ is 1-TLWE ciphertext. For example, as shown in Fig. 5, when we need to calculate binary number -1 (users' data) and 1(weights' data) and expend the result to 4-TLWE ciphertext $\boldsymbol{C}$. (1) We encode the binarized number. (2) Encrypt users' data, $[0] = Enc(0, \boldsymbol{s})$. (3) We get $\boldsymbol{C}_0$=**h-xnor**([0],1)=[0]. (4) We have **ep**$(\cdot)$ on $\boldsymbol{C}_0$ and get $\boldsymbol{C}$ as final result: first, perform **not**$(\boldsymbol{C}_0)$ to get intermediate variables $[temp]$, which is equal to [1]. Perform **copy**$(\cdot)$ and make every TLWE ciphertext in $\boldsymbol{C}$ is equal to $\boldsymbol{C}_0$. Then we perform **constant**$(\cdot)$ and **not**$(\cdot)$ on $\boldsymbol{C}_0$. Finally, we get the result [1][1][1][1] as $\boldsymbol{C}$.

Algorithm 1: ep$(\cdot)$

Input: 1-TLWE ciphertext $[a]$
Output: N-TLWE ciphertext $[b]$
$[temp]$=NOT($[a]$);
$m = 0$;
while $m < N$ **do**
  $[b_m] = [temp]$;
  $m = m + 1$;
end
$[b_0]$=**constant**$[b_0]$;
($[b_0]$)=**not**($[b_0]$);

Adder$(\cdot)$: Homomorphic add is the basic operation for our binarized CNN model. In this operation, we input n-bit X and Y, get n-bit Sum as the output. Here, we define n-bit $Carry$ as intermediate variable. We can build the adder according to the following formula:

$$Carry_{i+1} = (X_i \times Y_i) + (X_i + Y_i) \times Carry_i \tag{9}$$

$$Sum_i = X_i + Y_i + Carry_i \tag{10}$$

In our model, we use **and**$(\cdot)$ and **xor**$(\cdot)$ to represent $\times$ and $+$ in formula. Before perform bit-wise operations, the adder will determine if the input is plain or ciphertext. If X_i or Y_i is plaintext, we use **h-and**$(\cdot)$ and **h-xor**$(\cdot)$ to perform homomorphic calculation. Otherwise, we use bit-wise operations in [15] to finish add. Therefore, **Adder**$(\cdot)$ not only performs add calculation between ciphertext and plaintext, but also between ciphertext and ciphertext.

After finishing **Binarized Convolution Operation**$(\cdot)$, SP implements **Dot Product Operation**$(\cdot)$ on matrix $[\boldsymbol{K}]$, the weight $\boldsymbol{\alpha}$ and encrypted matrix $[\boldsymbol{S}]$ by using homomorphic **Multiplier**$(\cdot)$.

Dot Product Operation$(\cdot)$: This operation's inputs are $p \times p$ encrypted matrix $[\boldsymbol{K}]$, $c \times p \times p$ matrix $[\boldsymbol{S}]$ and the parameter $\boldsymbol{\alpha}$. And its output is a $c \times p \times p$ encrypted matrix $[\boldsymbol{L}]$. For $0 \leq i, j \leq p$ and $0 \leq u \leq c$, where i, j, u represent the index of the matrix:

$$[\boldsymbol{L}_{l,i,j}] = \textbf{Multiplier}([\boldsymbol{S}_{u,i,j}], [\boldsymbol{K}_{i,j}], \boldsymbol{\alpha}) \tag{11}$$

Multiplier$(\cdot)$: We build this homomorphic multiplier by using **and**$(\cdot)$ and **Adder**$(\cdot)$. In this multiplier, we input two n-bit X and Y, and we define n-bit C^0,C^1,$\ldots$,C^{n-1} are partial products. We can calculation the ith partial product by using **and**$(\cdot)$, where j represents the matrix index and $j \in \{0, 1, \cdots, n-1\}$:

$$C_j^i = \textbf{and}(X_i, Y_j) \tag{12}$$

Then, we perform **not**$(\cdot)$ on partial products' highest bit and create new C^0,$\ldots$,C^{n-1}. With our **Adder**$(\cdot)$, we get result Sum=**Adder**(C^0,C^1,$\ldots$ C^{n-1}) according to dislocation addition.

By finishing **Dot Product Operation**$(\cdot)$ under FHE, we take the encrypted matrix $[\boldsymbol{L}]$ as the input of the pooling layer.

3) Max Pooling Layer: In the binarized CNN model, we use Max Pooling Layer [30] to reduce the size of the model and improve the robustness of the extracted features, as well as increase the calculation speed. In this layer, the input is a $l \times p \times p$ encrypted matrix $[\boldsymbol{L}]$ and output is the $l \times q \times q$ encrypted matrix $[\boldsymbol{M}]$, where q=$p/2$. We use 2×2 max filter to non-overlapping subregions of input matrix $[\boldsymbol{L}]$, and then we use **Compare**$(\cdot)$ to calculate the max of down-sampling area. With the max of the region from input matrix, we create the encrypted matrix as the output.

Compare$(\cdot)$: This operation can compare two ciphertext and get the maximum value. Suppose $[x]$ and $[y]$ are two m-TLWE ciphertext, $x = \sum_{i=0}^{m-1} x_i 2^i$ and $y = \sum_{i=0}^{m-1} y_i 2^i$ (m is the bit-depth of x and y, x and y are decomposed and encrypted bit by bit as $[x_0], [x_1], \cdots, [x_{m-1}]$ and $[y_0], [y_1], \cdots, [y_{m-1}]$). We construct the circuit by following calculation equation:

$$t_0 = [0], t_{i+1} = ([1] - ([x_i] - [y_i])^2) \times t_i + [x_i] \times ([1] - [y_i]) \tag{13}$$

After m times rounds of calculations, we will get 1-TLWE ciphertext t_m (t_m is the result for $[x > y]$: if $[x] > [y]$, we will get $t_m = [1]$, else $t_m = [0]$). Obviously, we need to adjust this algorithm to make it compatible with our model. We use **h-and**$(\cdot)$, **h-xnor**$(\cdot)$, **h-xor**$(\cdot)$ represent the operators $+, -, \times$ in the above formula:

$$t_0 = 0, v_i = [x_i] \textbf{ h-xor } [y_i] \tag{14}$$

$$t_{i+1} = \{(1 \textbf{ h-xnor } v_i) \textbf{ h-and } t_i\} \textbf{ h-xor } \{[x_i] \textbf{ h-and } (1 \textbf{ h-xor } [y_i])\} \tag{15}$$

Consider the sign bit, we define ciphertext $[Sign]$=**h-xor**($[x_{m-1}]$,$[y_{m-1}]$). Then we update t_m=**h-xor**(t_m,$[Sign]$). We define m-TLWE $[R]$ as the output of **Compare**$([x], [y])$. To get $[R]$, we use t_m to calculate with $[x]$ and $[y]$ on bit level as following rule:

$$[R_i] = \{t_m \textbf{ h-and } [x_i]\} \textbf{ h-and } \{\textbf{not}(t_m) \textbf{ h-and } [y_i]\} \tag{16}$$

After m times rounds of calculations, we will get final ciphertext $[R]$ as the result to represent the maximum between $[x]$ and $[y]$.

4) Fully Connection Layer: The fully connected layer combines the features extracted from the convolutional layer and the pooling layer. This layer plays the role of mapping the learned feature representation to the sample label space. In this layer, input data is a $l \times q \times q$ encrypted matrix $[\boldsymbol{M}]$ and output is a $1 \times N$ $[\boldsymbol{F}]$, where N represents number of categories. With weights $\boldsymbol{f}$ and $\boldsymbol{b}$, we define the output $[\boldsymbol{F}]$=$[\boldsymbol{M}] \odot \boldsymbol{f} + \boldsymbol{b}$, where the size of $\boldsymbol{f}$ is $N \times l \times q \times q$ and the size of $\boldsymbol{b}$ is $N \times 1$. We use **Dot Product Operation**$(\cdot)$ and **Adder**$(\cdot)$ to construct the operations in Fully Connection Layer as follows:

(1)We define $1 \times N$ encrypted vector $[\boldsymbol{H}]$=**Dot Product Operation**($[\boldsymbol{M}]$,$\boldsymbol{f}$) (2) Output encrypted data $[\boldsymbol{F}]$=**Adder**($[\boldsymbol{H}]$,$\boldsymbol{b}$).

Here, we describe the activation function in the binarized CNN inference. In our paper, we use **ReLU** function as the implementation of activation function.

For input data x, **ReLU**(x)=Max(0,x). Since we have realized ciphertext comparison, we can easily construct ReLU($[x]$)=**Compare**(0,$[x]$). However, since the ciphertext is expressed as bits, it is time-consuming. Here, we provide a more efficient way to construct **ReLU** function as shown in **Algorithm 2**, where the input is a *n*-TLWE ciphertext $[a]$ and output is a *n*-TLWE ciphertext $[b]$. We also define $[temp]$=**not**($[a_{n-1}]$), for 0≤ i ≤n-1: $[b_i]$=**and**($[a_i]$,$[temp]$)

Algorithm 2: ReLU

Input: n-TLWE ciphertext $[a]$
Output: n-TLWE ciphertext $[b]$
$[temp]$=**not**($[a_{n-1}]$)
while *0≤ i ≤n-1* **do**
  $[b_i]$ = **and**($[temp]$,$[a_{n-1}]$)
end

In this layer, we don't use softmax layer. Since softmax layer is calculating the percentage of each category, and it won't change the result itself. Consequently, when users get and decrypt the result, they can still know which category the data belongs to without the softmax layer.

IV. Results and Analysis

Our model is implemented in C++. The model and operations are executed using a PC with i7-6700 CPU and 8G RAM. To describe our framework, we construct a three-layer binarized CNN and single-layered binarized perceptron in Pytorch. And all of our models are carried out using single thread.

Binarized CNN: We build and test the CNN model on the MNIST dataset, and the input and convolutional layers' weights are binarized. In this model, we use 60,000 images for training and 10,000 images for testing.

Binarized perceptron: We build and test the perceptron on the Breast Cancer dataset, and the input and weights are binarized in this model. In this model, we use 460 cancer data for training and 115 cancer data for testing. The time-consuming of the model in experiments is the average time overhead on the test dataset.

The code of these models can be found in *https://github.com/Karry11/Package*.

A. Results

As shown in Table I, we give models' time-consuming and accuracy under homomorphic encryption, including our binarized CNN inference, binarized perceptron, hybrid CNN [12], perceptron [12] CryptoDL [13], FHE-DiNN [16] and MiniONN [3]. Our model is constructed under the boolean circuit scheme, which performs bootstrapping in gate circuit calculation. Compared with the perceptron in [12] under the boolean circuit, our binarized perceptron is 3.6 × faster than it with almost the same accuracy.

TABLE I
TIME-CONSUMING OF INFERENCES OVER ENCRYPTED DATA

Encryption	Operation	ReLU	Models	Time	Accuracy
FHE	Boolean	Supported	Perceptron [12]	4m 13s	97.7%
			Binarized Perceptron	1m 11s	**97.6**%
			Hybrid CNN [12]	58h 11m 21s	99%
			Binarized CNN	41h 36m 29s	**97.11**%
LHE	Non-Boolean	Unsupported	CryptoDL [13]	2m 29s	99.21%
			FHE-DiNN [16]	1.7s	96%
			MiniONN [3]	9.32s	99%

TABLE II
TIME-CONSUMING OF HOMOMORPHIC OPERATIONS

Operations	Time	Bit-Depth
Binarized Ciphertext Comparision	0.27s	8
Ciphertext Comparision [15]	0.71s	8
Binarized Ciphertext Comparision	0.64s	16
Ciphertext Comparision [15]	1.44s	16
Binarized Convolution	6h 24m 14s	20
Hybrid Convolution	40h 36m 48s	20
Encrypted Convolution	102h 9m 58s	20
ReLU	0.09s	8
ReLU	0.21s	16

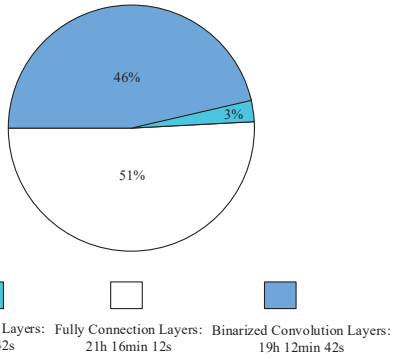

Fig. 6. time-consuming distribution in Binarized CNN inference

TABLE III
ACCURACY ON PLAINTEXT AND CIPHERTEXT

Model	MNIST	Accuracy
Binarized CNN	Plaintext Dataset	97.34%
Binarized CNN inference	Ciphertext Dataset	97.11%

The binarized CNN inference we have designed is similar to hybrid CNN in [12], but we construct a Max Pooling Layer that can help reducing data dimensions. Experimental results show that our binarized CNN inference is almost 1.4 $\times$ faster than theirs. It demonstrates that binarized operations have a positive effect on accelerating ciphertext calculation. Different from our model being implemented using FHE, these three models including FHE-DiNN [16], MiniONN [3], and CryptoDL [13] are built under non-boolean circuit and level homomorphic encryption (LHE). Benefit from LHE, the above three models perform better than ours in computational speed. However, the accuracy of the activation function still needs to be improved in their models. For example, in CryptoDL [13], they used a polynomial to replace a nonlinear function. To a certain extent, this will lead to a decrease in the accuracy of the model. Benefit from bit-wise operations under FHE, our models construct **ReLU** activation function without loss of precision. Besides, our model constructs ciphertext comparison and max pooling layer [30], which has the effect of dimensionality reduction on data.

To illustrate the efficiency of our binarized convolution, we reconstruct the hybrid convolution in [12] with the same 20 bit-depths, 5 $\times$ 5 convolution kernel and 28 $\times$ 28 encrypted image. At the same time, we construct encrypted convolution whose weights and data are encrypted. As shown in Table II, our binarized convolution is 6.3 $\times$ faster than hybrid convolution in [12] and 16 $\times$ faster than encrypted convolution. When models use larger bit-depths, our method can improve the computational efficiency more.

For ciphertext comparison under the boolean circuit, in 8-TLWE ciphertext level, our algorithm is 2.6 $\times$ faster than ciphertext comparison in [15]. In 16-TLWE ciphertext level [15], ours is 2.3 $\times$ faster than theirs. It shows our method works in bit-level ciphertext calculation.

Here we give the time spent to the specific operations. As shown in Fig. 6, the max pooling layer and Fully Connection Layer of our binarized CNN inference take 39 minutes 42 seconds and 21 hours 16 minutes 12 seconds, respectively. In these two layers, we use 25 bit-depths to represent the weights and encrypted data to get more precise results.

As shown in Table III, we compare the accuracy of binarized CNN inference and CNN on the test dataset. It indicates that there is little loss of accuracy when encrypted data is processed in the inference models. In this area, we can also try to use fewer bits to encode inputs and weights for reducing the time-consuming of the model.

B. Analysis

Due to the complexity of homomorphic calculations and the enormous computational complexity of neural networks, the computational process is time-consuming under the boolean circuit. Compared to the traditional secure CNN inference in [12], our model binarizes the input data and weights in binarized convolution to accelerate the calculation speed of CNN inference. In our model, we implement bit-wise operations into convolutional layers instead of ciphertext multiplication. Since the weights and input are binarized in our model, we use less ciphertext space to encrypt original data and store. This makes our model performs better in speed and storage than their work.

In section 4.1, simulation results indicate that our binarized convolution operation costs 6 hours 24 minutes 14 seconds, and the hybrid convolution [12] costs 40 hours 36 minutes 48 seconds in the same condition. Ours is 6.3 $\times$ faster than theirs. By comparing with the secure perceptron in [12], our secure binarized perceptron costs 1 minute 11 seconds and performs 3.6 $\times$ faster than theirs. Besides, we achieve a

more efficient ciphertext comparison algorithm. Our binarized ciphertext comparison is 2.6 $\times$ and 2.3 $\times$ faster than traditional comparison algorithm [15] in 8-TLWE and 16-TLWE, respectively.

We compare our model with the non-boolean circuit and LHE, like CryptoDL, our binarized inference model uses the bite-wise operation to construct ReLU function without fitting a polynomial as an activation function. This will help us improve the recognition rate of the model during the training and testing phases. Similarly, compare to Fast-DiNN and MiniONN, our model supports efficient ciphertext comparison, which is the key to Max Pooling Layers. It will reduce the complexity of the model and the dimensions of the data.

V. Conclusion

In this paper, we proposed the secure binarized CNN inference and perceptron, where the models are implemented by bit-wise operations to optimize ciphertext inference. Furthermore, we designed a ciphertext comparison algorithm to reduce the dimension of ciphertext data. To increase the nonlinear expression of the model, we used the ReLU function as an activation function, which is implemented by bit-wise operations including $\mathbf{not}(\cdot)$ and $\mathbf{and}(\cdot)$. Since Fully Connection Layer's weights aren't binarized, we found that fully connected layers occupy 51.2% of the time in the entire model. Despite this, there is still great potential to increase the speed of computing in the neural network inference model. In the future, we will consider a training model with binary value and trying to create a method to replace the batch normalization in deep learning under fully homomorphic encryption.

VI. Acknowledgments

This work is supported by the National Natural Science Foundation of China (61601337).

References

[1] Z. Zhao, A. Kumar, IEEE Transactions on Information Forensics & Security **12**(5), 1017 (2017)
[2] M.A. Turk, A.P. Pentland, in *Proceedings. 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (IEEE, 1991), pp. 586–591
[3] F. Celesti, A. Celesti, L. Carnevale, A. Galletta, S. Campo, A. Romano, P. Bramanti, M. Villari, in *Computers & Communications* (2017), pp. 1–9
[4] J. Heaton, N. Polson, J.H. Witte, Applied Stochastic Models in Business and Industry **33**(1), 3 (2017)
[5] C. Gentry, Symposium on Theory of Computing **9** (2009)
[6] M. Van Dijk, C. Gentry, S. Halevi, V. Vaikuntanathan, in *Annual International Conference on the Theory and Applications of Cryptographic Techniques* (Springer, 2010), pp. 24–43
[7] C. Gentry, S. Halevi, International Association for Cryptologic Research pp. 129–148 (2011)
[8] A. Krizhevsky, I. Sutskever, G.E. Hinton, in *Advances in neural information processing systems* (2012), pp. 1097–1105
[9] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, in *Proceedings of the IEEE conference on computer vision and pattern recognition* (2015), pp. 1–9
[10] J. Berant, V. Srikumar, P.C. Chen, A. Vander Linden, B. Harding, B. Huang, P. Clark, C.D. Manning, in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (2014), pp. 1499–1510
[11] T. Shen, T. Zhou, G. Long, J. Jiang, S. Pan, C. Zhang, in *Thirty-Second AAAI Conference on Artificial Intelligence* (2018), pp. 1–11
[12] A. Meehan, R.K. Ko, G. Holmes, arXiv preprint arXiv:1412.7522v4 (2018)
[13] E. Hesamifard, H. Takabi, M. Ghasemi, arXiv preprint arXiv:1711.05189 (2017)
[14] Z. Brakerski, C. Gentry, V. Vaikuntanathan, Transactions on Computation Theory **6**(3), 13 (2014)
[15] I. Chillotti, N. Gama, M. Georgieva, M. Izabachene, International Conference on the Theory and Application of Cryptology and Information Security pp. 3–33 (2016)
[16] F. Bourse, M. Minelli, M. Minihold, P. Paillier, in *Annual International Cryptology Conference* (2018), pp. 483–512
[17] M. Rastegari, V. Ordonez, J. Redmon, A. Farhadi, European Conference on Computer Vision pp. 525–542 (2016)
[18] A.L. Maas, A.Y. Hannun, A.Y. Ng, International Conference on Machine Learning **30**(1), 3 (2013)
[19] R.L. Rivest, L. Adleman, M.L. Dertouzos, Foundations of Secure Computation **4**(11), 169 (1978)
[20] S. Halevi, V. Shoup, IBM Research (Manuscript) **6**, 12 (2013)
[21] S. Halevi, V. Shoup, in *Annual International Cryptology Conference* (2018), pp. 93–120
[22] L. Ducas, D. Micciancio, Theory and Applications of Cryptographic Techniques pp. 617–640 (2015)
[23] S.S. Roy, F. Vercauteren, J. Vliegen, I. Verbauwhede, IEEE Transactions on Computers **66**(9), 1562 (2017)
[24] J. Fan, F. Vercauteren, IACR Cryptology ePrint Archive **2012**, 144 (2012)
[25] O. Regev, Journal of the ACM **56**(6), 34 (2009)
[26] C. Gentry, A. Sahai, B. Waters, in *Annual Cryptology Conference* (2013), pp. 75–92
[27] I. Chillotti, N. Gama, M. Georgieva, M. Izabachène, in *International Conference on the Theory and Application of Cryptology and Information Security* (Springer, 2017), pp. 377–408
[28] S. Ioffe, C. Szegedy, International Conference on Machine Learning (2015)
[29] M. Li, T. Zhang, Y. Chen, A.J. Smola, in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining* (ACM, 2014), pp. 661–670
[30] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, et al., Proceedings of the IEEE **86**(11), 2278 (1998)