applied sciences

MDPI

Article

Secure Convolution Neural Network Inference Based on Homomorphic Encryption

Chen Song and Ruwei Huang *

School of Computer and Electronic Information, Guangxi University, Nanning 530004, China
* Correspondence: ruweih@gxu.edu.cn; Tel.: +86-136-0313-7282

Abstract: Today, the rapid development of deep learning has spread across all walks of life, and it can be seen in various fields such as image classification, automatic driving, and medical imaging diagnosis. Convolution Neural Networks (CNNs) are also widely used by the public as tools for deep learning. In real life, if local customers implement large-scale model inference first, they need to upload local data to the cloud, which will cause problems such as data leakage and privacy disclosure. To solve this problem, we propose a framework using homomorphic encryption technology. Our framework has made improvements to the batch operation and reduced the complexity of layer connection. In addition, we provide a new perspective to deal with the impact of the noise caused by the homomorphic encryption scheme on the accuracy during the inference. In our scheme, users preprocess the images locally and then send them to the cloud for encrypted inference without worrying about privacy leakage during the inference process. Experiments show that our proposed scheme is safe and efficient, which provides a safe solution for users who cannot process data locally.

Keywords: convolution neural network; cloud computing; homomorphic encryption; privacy preserving machine learning; CKKS FHE scheme

Citation: Song, C.; Huang, R. Secure Convolution Neural Network Inference Based on Homomorphic Encryption. *Appl. Sci.* **2023**, *13*, 6117. https://doi.org/10.3390/app13106117

Academic Editor: Chihhsuan Wang

Received: 5 April 2023
Revised: 6 May 2023
Accepted: 11 May 2023
Published: 16 May 2023

1. Introduction

Due to its high efficiency, automation, and wide coverage, deep learning technology is widely used in many fields such as computer vision, natural language processing, and automatic driving. In real life, computing power is mainly concentrated in some cloud service providers. As a result, the combination of cloud computing and deep learning is also developing rapidly. Local users and enterprises can upload their personal data to the cloud, and use the computing power of the cloud service provider for model training and inference. For example, a furniture company can upload the type and price distribution of furniture purchased by different users to the cloud in the past year, and customize a model on the cloud to predict the possibility and quantity of residents in the area to buy different types of furniture in the next period of time. This forecasting model can obtain the minimum inventory quantity that the enterprise should keep at a relatively low cost, which is conducive to the turnover of the capital chain and also forecasts the future development direction of the enterprise. At the same time, the user's private data will also be leaked to the cloud. If the cloud uses these private data maliciously, such as using home address, phone number, and purchase tendency, the interests of these data providers are likely to be violated. In September 2022, the security company SOCRadar found that an improperly configured Azure Blob Storage server of Microsoft had a great possibility of information leakage. This sensitive information was associated with more than 65,000 corporate customers in 111 countries (regions). If information is leaked, users could face illegal activities such as extortion.

In order to protect the privacy data of these users, researchers focus their research on statistical techniques and cryptography techniques. In recent years, with the introduction of Differential Privacy (DP) [1], some researchers combined it with various machine learning

models and proposed various privacy protection schemes suitable for machine learning. The main protection point of this technology focuses on the private data used to train the model. Cryptography technologies such as Homomorphic Encryption (HE) and some Secure Multi-Party Computation (SMC) protocols are combined with the model reasoning stage, and some application solutions have emerged. However, these schemes require adding noise to the original messages, which has a significant impact on the final inference accuracy. Homomorphic encryption schemes can be traced back to 1978 when Rivest et al. [2] proposed the idea of processing data without revealing the information by using large prime factorization to ensure the security of the information. In 1982, Andrew Yao proposed garbled circuits [3], and many homomorphic encryption schemes were based on garbled circuits construction. However, these schemes had poor usability and large communication overhead, making them difficult to use in practice. It was not until 2009 when Gentry proposed the bootstrapping process that the first fully homomorphic encryption scheme was constructed [4], ushering in a new stage of development and expanding the usage scenarios of homomorphic encryption. In recent years, according to the mathematical problems on which the schemes are based and the differences in construction processes, homomorphic encryption schemes have been divided into four generations, which will be explained in Section 3.3.

1.1. Our Contribution

In this paper, we will use homomorphic encryption technology to protect user privacy in the inference stage, which is also the key stage of protecting user privacy information, in order to solve the privacy leakage problem of users using cloud providers. Users can use the technology in this paper to encrypt and upload personal information to the cloud, and the cloud uses a pre-trained model to infer the ciphertext. During the inference process, the cloud cannot obtain any information about the user. Through this scheme, user privacy is guaranteed. There are some schemes that use homomorphic encryption combined with a neural network reasoning stage, but we found that there is still room for improvement in combining the network structure with the ciphertext structure in these schemes, which will effectively reduce the complexity of ciphertext transmission in different layers of the model. To solve this problem, we adopt a more reasonable ciphertext allocation for the weight matrix. In the batch processing part, we make full use of the ciphertext slots, which reduces memory usage and computational complexity, thereby improving inference efficiency. In addition, we found that the parameter selection of the homomorphic encryption scheme will change the original image to a certain extent, and at the end of the inference stage, it will have a certain impact on the accuracy rate. In order to deal with this problem, we provide a new perspective , to enhance the robustness of the model by adding corresponding noise during the training phase to adapt the model to slightly "blurred" images. In this way, the influence of encryption scheme noise is reduced in the inference stage. At last, we use the CKKS fully homomorphic encryption scheme that supports floating-point operations, which can avoid the process of encoding floating-point numbers into integers in other schemes. Our system architecture is illustrated in Figure 1.

1.2. Organization

In Section 2, we summarize some related work. Then we provide the relevant definition of the homomorphic encryption scheme and the design goal of the scheme in Section 3. In Section 4, we provide the relevant algorithms and explanations of the scheme in this paper, as well as the construction process of the overall scheme. Then in Section 5, we provide the security analysis. In Section 6, we report the performance analysis and the comparison with other schemes. And in Section 7, We discussed the versatility of the technology in the article. The conclusion is shared in Section 8.

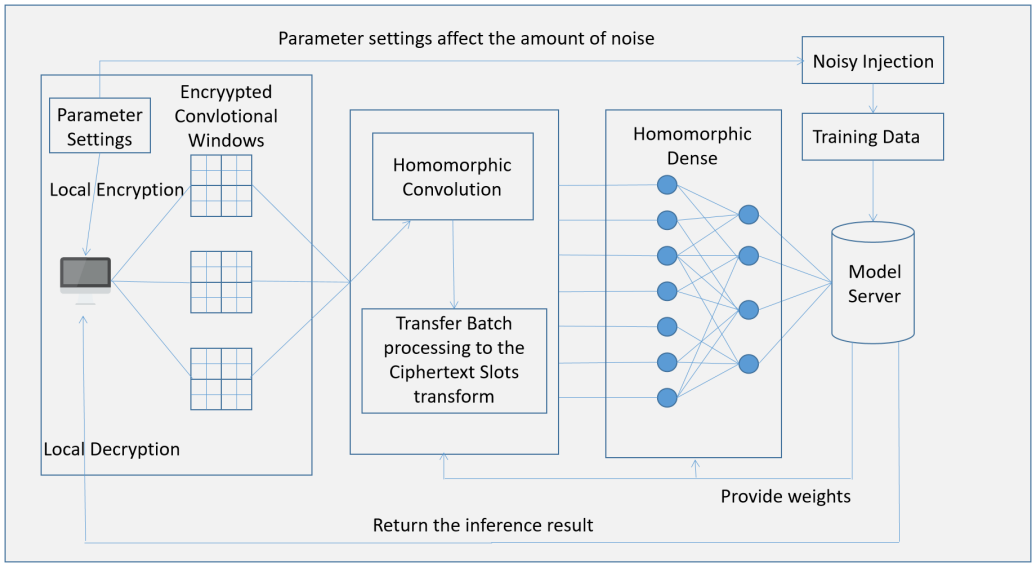

Figure 1. System architecture.

2. Related Works

Some schemes combining CNN with homomorphic encryption have been proposed. Gilad-Bachrach et al. [5] proposed one of the first works to combine homomorphic encryption with the inference stage of neural networks. The scheme uses the square function $f(x) = x^2$ to replace the activation function $ReLU : f(x) = max(0, x)$, use the scaling and summing function $f(x) = \sum(x_i)$ to replace the maximum pooling layer to deal with the inability of homomorphic encryption to handle non-polynomial operations and comparison operations. Due to the use of a homomorphic encryption scheme that only supports integers, polynomial encoding is used to approximate floating-point numbers. CryptoNets can achieve a classification accuracy of 98.95% on the MNIST data set, and can make more than 51,000 predictions per hour, but the delay of a single prediction is 570 s and long inference time per single run will greatly affect user experience. Chabanne et al. [6] combined the BatchNorm layer in the neural network with the encryption scheme, constrained the input of the nonlinear activation layer to a distribution, deepened the depth of the network, and enabled the ciphertext inference scheme to be applied to deeper layers and some complex data sets and practical applications. Chou et al. [7] proposed FasterCryptoNets, which intelligently prunes the network parameters, reduces the calculation of ciphertext multiplication, and uses the maximum sparse coding to derive the optimal quantization polynomial approximation of the activation function, using $f(x) = 2^{-3} * x^2 + 2^{-1}x + 2^{-2}$ which replaces the ReLU activation function. The optimized scheme is an order of magnitude faster than the CryptoNets scheme. Bourse et al. [8] proposed a homomorphic encryption inference framework FHE-DiNN specifically for parameter discretized neural networks. This scheme reduces the size of the input ciphertext and realizes homomorphic symbolic function operations. In the case of the same security level, the inference speed is two orders of magnitude higher than that of CryptoNets, but there is a certain loss in accuracy. SANYAL et al. [9] further refined the research focus to the binary parameter network, and proposed the Reduce Tree Adder and sorting network technology, which effectively improved the convolutional layer and dense layer. Ciphertext inference efficiency of dot product operation is also correspondingly improved. They transformed the network parameters from $\{-1, 1\}$ to $\{0, 2\}$ to improve the sparsity of the model. Hesamifard et al. [10] proposed the CryptoDL model in 2019 to optimize the approximation of non-polynomial functions, and provided polynomial approximation and error theory guarantees for activation functions such as ReLU, Sigmoid, and Tanh, which can be realized on the MNIST dataset reaching 99.25% classification accuracy. The Lola scheme proposed by Brutzkus et al. [11] designs an optimized data encoding method based on the idea of vectorization, and introduces transfer learning technology in ciphertext inference, and filters image sensitive information

before inputting it into the network. Ishiyama et al. [12] conducted a detailed study on the activation layer in homomorphic encryption neural network inference, using batch normalization to constrain the input of the approximate activation function, and tested the effects of accuracy on different approximate polynomials of Google Swish and ReLU activation functions in Ciphertext Inference. They improve the classification accuracy of MNIST to 99.29%. YuXiao LU et al. [13] aimed at the time-consuming problem of rotation operation in ciphertext packaging technology, and proposed a low-rank matrix decomposition scheme, which unifies the packaging operations of the convolutional layer and dense layer, and decomposes the convolutional layer into a volume A convolutional layer with a smaller kernel size; a convolutional layer with a kernel size of 1 improves the speed of inference, but there is a certain loss in classification accuracy. Joon-Woo Lee et al. [14] implemented the standard ResNet-20 model using the RNS-CKKS scheme with a bootstrap function, and used the minimax synthesis method to make an approximate replacement for the nonlinear activation function. In terms of ciphertext reasoning, it is very close to the classification accuracy of the ResNet-20 model using unencrypted data. It opens the possibility for FHE to be applied to deep PPML models. Using the CKKS homomorphic encryption scheme, it supports receiving input from the real number field at the input end, achieving higher-precision calculations, which better adapts to the data structure of neural networks. The BFV scheme only supports integer field encryption, and to achieve real number input adaptation, additional encoding processes are required, which adds additional time and space overhead.

In addition to combining homomorphic encryption with model inference, some researchers have adopted a hybrid protocol approach to further improve the speed of model inference. Liu et al. [15] constructed an Oblivious Neural Network (ONN) scheme using secret sharing technology. The security protocol in the scheme can satisfy data privacy when the server is semi-honest. Juvekar et al. [16] proposed a secure neural network reasoning framework GAZELLE using homomorphic encryption and obfuscated circuit technology. The framework includes three parts: a homomorphic layer, linear algebra core and network reasoning. In the network inference stage, a protocol that can convert between homomorphic and confusing circuit coding is designed. This scheme can hide more information of the neural network, provide higher security, and greatly shorten the inference time. Shaohua Li et al. [17] give a CNN prediction framework based on the fast Fourier transform, and for the first time give the privacy preserving protocol of the softmax function. Lucien K.L.Ng et al. and Kumar et al. [18] proposed some protocals for multiparty secure inference and gave a compiler for transforming normal models to secure inference models. Mishra et al. [19] proposed a secure prediction system that allows two parties to execute neural network inference without revealing either party's data called Delphi. They also used garbled circuits and linearly homomorphic public-key encryption but achieved high performance in ResNet-32. Lucien K.L.Ng et al. [20] formulate stochastic rounding and truncation (SRT) layers, making a quantization-aware training scheme SWALP more compatible with their cryptographic tools, and they propose a suite of GPU-friendly protocols for both linear layers and common non-linear layers for GPU parallelism. They use additive homomorphic encryption and additive secret sharing to avoids the step of using polynomials to approximate nonlinear layers. However, privacy-preserving inference schemes based on hybrid protocols have poor portability and scalability, making it difficult to adapt to models of different types.

3. System Model, Encryption Scheme and Design Goals

In this section, we briefly describe the system model, formal definitions, and design goals.

3.1. System Model

We assume that in a common scenario, the data owner encrypts the data locally and then uploads it to a third party for calculation. The third party has a pre-trained model. The ciphertext is returned to the data owner, and then decrypted to restore the plaintext of

the reasoning result. In the case where a third party is involved in the reasoning process, the data may encounter the risk of data theft such as interception and side channel attacks. This risk can be avoided through a multi-party secure computing protocol. However, what is more worrying is that the cloud service provider could exploit the data provided by data owners. This data may be sold, or used to carry out illegal activities. Homomorphic encryption, a cryptographic method, can effectively deal with this problem. The data owner uses the public key to encrypt the data, while the cloud can only obtain information in the form of ciphertext, and use a dedicated evaluation key (evaluation key) to perform limited operations on the ciphertext. This mode can guarantee the privacy of the data provider. Our reasoning system process is illustrated in Figure 2.

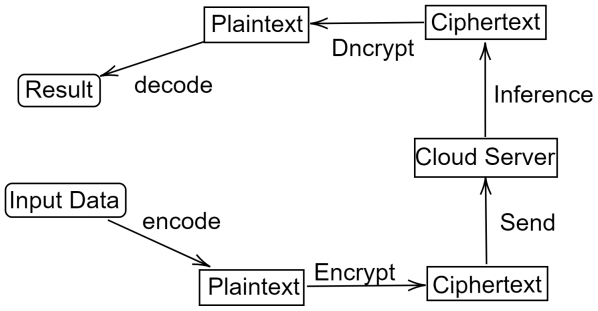

Figure 2. Inference procedure.

3.2. Security Model

According to the intention of the participants in the protocol, we divide the security model into two models: semi-honest model and malicious adversary model. The semi-honest model means that during the execution of the agreement, the participants follow the procedures stipulated in the agreement, but malicious attackers may monitor and obtain their own input and output during the execution of the agreement and during the operation of the agreement. The malicious adversary model (the malicious model) means that during the execution of the protocol, the attacker can analyze the private information of honest participants through illegal input or malicious tampering of input by the participants under their control. A termination of the agreement can also be caused by early termination and refusal to participate.

3.3. Homomorphic Encryption Scheme

The concept of homomorphism was proposed by Rivest et al. [2], and it means that certain operations can be performed on data when the data is in ciphertext. Homomorphic encryption refers to allowing a third party to perform certain types of operations on ciphertext and to ensure that the ciphertext after the operation is decrypted, as well as making the obtained plaintext equivalent to performing the same operation on the unencrypted plaintext; in this process, no third party can obtain any information in plain text.

Homomorphic encryption schemes can be classified into three categories based on the depth of computation: (1) partially homomorphic encryption: this scheme can only perform one type of operation (addition or multiplication) on ciphertexts [21]. (2) Somewhat homomorphic encryption: this scheme allows for multiple operations on ciphertexts, but limits the depth of computation [22]. (3) Fully homomorphic encryption: this scheme enables arbitrary operations to be performed on ciphertexts an unlimited number of times [4]. As homomorphic encryption has evolved over time, these schemes can be divided into four generations based on their time of development. In Table 1, some characteristics of these generations of homomorphic encryption schemes are presented.

Table 1. The classification of homomorphic encryption schemes.

Generation	Input Data Types	Fast Packing or Batching	Fast Bootstrapping	Applications	Deficiency
First (Gentry09 [4])	Binary data	Supported	Unsupported	SIMD	Large ciphertext size
Second (BGV [23], BFV [22])	Integer	Supported	Unsupported	Fast Escalar Multiplication	Slow Bootstrapping
Third (FHEW [24], TFHE [25])	Bitwise	Unsupported	Supported	Efficient Boolean Circuits	No support for Batching
Fourth (CKKS [26])	Real Number	Supported	Unsupported	Fast polynomial approx;SIMD	Slow Bootstrapping

Definition 1. *Given input data x and any operation f, if there exists an encryption scheme ε that satisfies the following equation, where Enc is the encryption operation, Dec is the decryption operation, f' is the corresponding ciphertext operation of f, then the scheme ε is a homomorphic encryption scheme if it meets the following equation:*

$$f(x) = Dec(f'(Enc(x))) \tag{1}$$

In the initial stage of the combination of homomorphic encryption technology and neural network reasoning, many schemes used homomorphic encryption schemes that only supported integers. The unified image representation is also a floating point number. This requires adding a floating-point-to-integer mapping process.

3.4. Formal Definition

In our scheme, the homomorphic encryption scheme supporting floating point numbers proposed by Cheon et al. [26] is used. Some formal arithmetic definitions of the scheme are given below:

Definition 2. *Fix a base $p > 0$ and a modulus q_0, and let $q_l = p^l \cdot q_0$, for $0 < l < L$, choose λ as a security parameter, so we can choose a cyclotomic polynomial parameter $M = M(\lambda, q_L)$. For a level $0 < l < L$ and a fix k, we definite the ℓ-th level ciphertext is a vector in $\mathcal{R}_{q_l}^k$. A leveled homomorphic encryption scheme is described over the polynomial ring $\mathcal{R} = \mathbb{Z}(X)/(\phi_M(X))$ which contains five basic algorithms $(\mathbf{KeyGen}, \mathbf{Enc}_{pk}, \mathbf{Dec}_{sk}, \mathbf{Add}, \mathbf{Mult}_{evk})$ and a rescaling procedure:*

- $\mathbf{KeyGen}(1^\lambda)$*: choose a security parameter λ, generate a secret key sk, a public key pk and a key evk used for evaluation process;*
- $\mathbf{Enc}_{pk}(m)$*: for a given polynomial $m \in \mathcal{R}$, output a ciphertext $c \in \mathbf{R}_{q_L}^k$. For some small noisy e,the ciphertext c for message m satisfy $\langle c, sk \rangle = m + e (mod q_L)$. A constant B_{clean} denotes an encryption bound, i.e., error polynomial of a fresh ciphertext satisfies $||e||_\infty^{can} \leq B_{clean}$ with an overwhelming probability;*
- $\mathbf{Dec}_{sk}(\mathbf{c})$*: for a ciphertext c at a level ℓ, output a polynomial $m\prime \leftarrow \langle c, sk \rangle (mod q_l)$;*
- $\mathbf{Add}(\mathbf{c}_1, \mathbf{c}_2)$*: for the given encrypts of m_1, m_2, output an encryption of $m_1 + m_2$. An error of output ciphertext is bounded by sum of two errors in input ciphertexts;*
- $\mathbf{Mul}t_{evk}(\mathbf{c}_1, \mathbf{c}_2)$*: for a pair of ciphertexts (c_1, c_2), output a ciphertext $c_{mult} \in \mathcal{R}_{q_\ell}^k$ satisfies $\langle c_{mult}, sk \rangle = \langle c_1, sk \rangle \langle c_2, sk \rangle + e_{mult} (mod\ q_l)$ for some polynomial $e_{mult} \in \mathcal{R}$ with $||e_{mult}||_\infty^{can} \leq B_{mult}(\ell)$;*
- $\mathbf{RS}_{\ell \to \ell\prime}(c)$*: for a ciphertext $c \in \mathbf{R}_{q_\ell}^k$ at level ℓ and a lower level $\ell' < \ell$, output the ciphertext $c' \leftarrow \left\lfloor \frac{q_{\ell\prime}}{q_\ell} c \right\rceil$ in $c_{mult} \in \mathcal{R}_{q_\ell}^k$, i.e., c' is obtained by scaling $\frac{q_{\ell\prime}}{q_\ell}$ to the entries of $\mathbf{c}$ and rounding the coefficients to the closest integers. We will omit the subscript $\ell' \leftarrow \ell$, when $\ell' = \ell - 1$.*

3.5. Design Goals

A privacy-preserving neural network inference scheme built using the encryption techniques described in Section 3.4 should obtain the following security and performance guarantees:

- Correctness: after the user correctly implements our proposed scheme, the prediction result should be correct. The correctness of homomorphic encryption schemes can be understood as follows: encrypting a plaintext, evaluating the ciphertext, and then decrypting the result is equivalent to evaluating the plaintext with the same operation to obtain the same result [4];
- Privacy: after the cloud service provider obtains the ciphertext provided by the user, it cannot obtain any valid information from the ciphertext, and can only perform certain operations on the ciphertext to achieve the user's expected goal;
- Verifiability: after the user obtains the ciphertext returned by the cloud service provider, there is a way to verify the correctness of the result.
- Efficient: including channel transmission time and inference time, the total time should be much shorter than the user's own local inference time.

4. Our Scheme

In this section, we systematically explain how our solution works and introduce the overall architecture of the solution. In the second summary, we explain in detail the improvements we have made in batch processing, and then introduce our verification process and solutions for the noise of the encryption scheme, and finally give the correctness analysis.

4.1. CNN Private Inference Scheme

We adopted a network architecture similar to that of Jiang et al. [27], and adjusted the size of the convolution kernel, the number of channels, and the step size. The Swish function mentioned in the article by Ishiyama et al. [12] is used in both activation layers to improve the accuracy of inference. The comparison of our model architecture with that of Jiang et al. [27] is shown in Table 2.

Table 2. Model comparison with another scheme which Jiang et al. [27] used.

Layer	Basic Model	Our Model
CONV	64 input images of size 28×28, 4 kernels of size 7×7 (4 channels), stride size of (3, 3)	64 input images of size 28×28, 5 kernels of size 4×4 (5 channels), stride size of (2, 2)
BN	None	$13 \times 13 \times 5 = 845$ outputs
ACT-1	Squaring 256 input valus $x \leftarrow x^2$	$x \leftarrow 0.03347 + 0.5 \cdot x + 0.19566 \cdot x^2 - 0.005075x^4$
FC-1	Fully connecting with $8 \times 8 \times 4 = 256$ inputs and 64 (neural nodes) outputs	Fully connecting with $13 \times 13 \times 5 = 845$ inputs and 100 outputs
BN	None	100 outputs
ACT-2	Squaring 64 input valus $x \leftarrow x^2$	$x \leftarrow 0.03347 + 0.5 \cdot x + 0.19566 \cdot x^2 - 0.005075x^4$
FC-2	Fully connecting with 64 inputs and 10 outputs	Fully connecting with 100 inputs and 10 outputs

The dataset we used to validate the experiment is the MNIST dataset, which contains 70,000 images, of which 60,000 were used for training and 10,000 for testing, containing ten types of handwritten digits from 0 to 9. The MNIST dataset, although old, is still a benchmark for testing in the field of privacy-preserving neural network inference. Figure 3 is an example image from the MNIST dataset. CIFAR-10 is a 10-class image classification dataset consisting of RGB color images of ten different categories. This dataset contains 50,000 training images and 10,000 testing images, with an image size of 32×32 pixels. The ten categories in CIFAR-10 are airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Unlike the MNIST dataset, CIFAR-10 [28] is a relatively larger dataset and more challenging as the images have more details and complexity. CIFAR-10 is also widely used in

machine learning and deep learning image classification tasks and is one of the benchmark datasets for many computer vision algorithms. Both of these datasets are relatively small in scale but suitable for model privacy protection in the inference field. When images are encrypted, the size of the information will rapidly expand, which is also a challenge for system performance. Therefore, we have chosen these two benchmark datasets.

Figure 3. Examples in MNIST dataset.

The training phase of the network and the add noise phase mentioned in Section 3.3 are implemented using Python and the Keras framework [29]. Keras is a Python-based deep learning framework that provides many convenient interfaces for researchers to use, and the inference stage is implemented using the SEAL library [30]. The following Table 3 provides an introduction to the characteristics of some open-source libraries for homomorphic encryption schemes.

Table 3. Available open-source libraries for homomorphic encryption.

Library	BGV [23]	TFHE [25]	FHEW [24]	BFV [22]	CKKS [26]	Language
SEAL [30]	✓	✗	✗	✓	✓	C++/C
HElib [31]	✓	✗	✗	✗	✓	C++
FHEW [24]	✗	✗	✓	✗	✗	C++
TFHE [25]	✗	✓	✗	✗	✗	C++/C
HEAAN [32]	✗	✗	✗	✗	✓	C++

4.2. Batch Processing in Inference Stage

We propose a method to fully utilize the ciphertext slot for batch image encryption in the inference process, addressing the issue of incomplete utilization of the ciphertext slot. To implement this method, we first divide each image into corresponding convolution windows, and then stitch together the images that need to be batched in a manner that does not interfere with subsequent rotation operations. Specifically, we employ an image segmentation algorithm to extract the convolutional windows used by the convolutional layer, where p represents the resulting window matrix, with each row representing a convolutional window. The implementation process of the image segmentation algorithm is described in Algorithm 1.

Algorithm 1 Image Divider

Input: A image I of the size $h \times w$(matrix), the size $k \times k$ of a kernel, and a stride s of 2 (vertical and horizontal), index of the position of image
Output: conv partial of the size $f^2 \times k^2$

```
f ← (h − k)/strides +1;
for i := 0 to f do
    for k := 0 to f do
        for t := 0 to k do
            for j := 0 to k do
                p[i × f + k] ← I[s × i + t][s × k + j]
            end for
        end for
    end for
end for
return conv partial
```

Subsequently, we need to stitch the batched images together, and the necessary operations required for the subsequent inference steps need to be considered during the stitching process, so we assign the number of plaintext slots of the convolution kernel area plus the sum of the output dimensions of the first dense layer at the same window position for each image. In the preprocessing picture stage, each position is filled with the value of the window, then filled with the number of slots minus 0 of the window size, and then the window in the same position as the next picture, and so on. In subsequent convolution operations, the accumulation step of the vector after the dot product requires a rotation operation. Assuming that there are values at positions 1–8 of the plaintext slot, then we want to add them up, we need to shift them left 1 time and then add them to the original vector, in this case the values in the slot are $1+2, 2+3, 3+4, \ldots, 8+0$. Next we need to add the first slot to the third slot, so that we can obtain a value of 1–4 in the first slot, and so on, adding up the eight slots needs to be rotated log_2^s times, and s is the number of slots that need to be accumulated. In the encryption scheme, the plaintext slot is similar to the structure of the loop queue, the left shifted slot will move to the end of the entire plaintext, in order not to let the value moved to the back end affect the rotation accumulation process, we need to reserve enough slots for each position as Algorithm 2.

Algorithm 2 Image Padding

Input: Image matrix of the size $h \times w$, kernel size 4 (height and width), batch size bs, strides 2 (height and width), each slots of every window to batch operation
Output: The k cipertexts $ct.S_{[i]}$ for $1 \leq i \leq k$, each cipher stores the same position of convlution window of batch images

```
each slots ← 4 × 4 + the number of dense outputs
W ← Imagedivider (images, kernel, 0)
for i := 1 to batch size do
    for k := 0 to W.rows do
        for j := 0 to each slots-4^2 do
            W_i ← 0
        end for
        for p := 0 to 4^2 do
            W_i ← Imagedivider(images, kernel, i)
        end for
    end for
end for
for i := 0 to W. rows do
ct.C ← Enc_pk(W_i)
S_i ← ct.C
end for
return S
```

After the image window is filled, each row of the window matrix contains a batch of windows in the same position of the picture, and 0 in between. Then we need to perform the same filling operation for the convolution kernel and the convolution deviation, so that the convolution kernel is aligned with the image window. After the convolution kernel and window dot product have passed, the convolution operation of the ciphertext is completed by rotating the accumulation, and finally the bias is added, and the ciphertext feature map matrix is output, and each column represents the characteristics of different channels. Algorithm 3 shows the procedure.

Algorithm 3 Homomorphic Convlution

Input: A encrypted matrix($c \times f$) of convolution windows S, padding convlution bias vector(plaintext) B, channel c, padding convlution weights vector(plaintext) P, each row represents a channel, kernel size k^2

Output: Encrypted feature map of convlution layer output M

for k := 0 to c **do**
  for i := 0 to f **do**
    $ct.temp_i \leftarrow MultP(ct.S_i, p.P_k)$
    $ct.temp_i \leftarrow$ Rescale()
  end for
  $T_k \leftarrow$ ct.temp
end for
for t := 0 to c **do**
  for i := 0 to f **do**
    for j := 0 to $\lceil \texttt{log2}(k^2) \rceil$ **do**
      temp $\leftarrow$ Rot($T_{[t][i]}$, exp2(j))
      $T_{[t][i]} \leftarrow$ Add(ct.$T_{[t][i]}$, temp)
    end for
    $T_{[t][i]} \leftarrow$ AddPlain(ct.$T_{[t][i]}$, ct.B_t)
  end for
end for
for i := 0 to f **do**
  for j := 0 to c **do**
    $M_{[i][j]} \leftarrow T_{[j][i]}$
  end for
end for
return M

Next, we will pass the eigenvalue ciphertext output by the convolutional layer through the activation function, here we use the linear polynomial function $poly_swish(M_k) = 0.03347 + 0.5x = 0.19566x^2 - 0.005075x^4$ proposed by Ishiyama et al. [12] to approximate the Swish activation function. This step requires the convolutional layer to output the values of each neuron through the activation function: $Ct.A \leftarrow poly_swish(M_k), 0 \leq k \leq channel \times featuremapsize$.

Since the highest order of the approximate polynomial is order 4, the highest-order input is squared twice and then multiplied by the corresponding coefficient, thus consuming three levels of the homomorphic encryption scheme.

The activated neuron will be fully connected through two dense layers, we expand the activation layer output into a ciphertext vector with the size channel * feature map size, and then the $outputsize * Inputsize$ of the first dense layer is batch expanded similarly to a convolution window, aligned with the activation layer output. After encryption, there is an input size ciphertext. In the ciphertext output of the previous layer, only the first slot per position of each image is the correct result of accumulation, in order to perform the next matrix multiplication, we will activate the output tensor of the layer to expand horizontally, replacing the slots of the output dimension size of the subsequent dense layer with the value of the first slot. To achieve this, we choose to consume one more level, multiply each line of

the output ciphertext with the $[1, 0, 0, \ldots, 1, 0, 0]$ vector, which corresponds to the correct slot and the 0 value empties the other slots. Then, we need to rotate log_2^{do} times to expand the value of the first slot to the same scale as the weight matrix, so it is the output dimension of dense layer 1. The same area of the feature map of different images in each ciphertext is spaced at the same interval, so the rotation operation can be synchronized to all images in the batch. Figure 4 shows the simplified process of ciphertext slot transformation. When the modulus is $N = 32{,}678$, the number of plaintext slots is N/2 = 16,384, 141 pictures can be batched at the same time, if the N value is larger, more plaintext slots can be generated for batch processing, but the SEAL library limits each plaintext to 2^{60} bits. Algorithm 4 provides a detailed description of the specific implementation process. Figure 5 shows the flowchart of Algorithm 4.

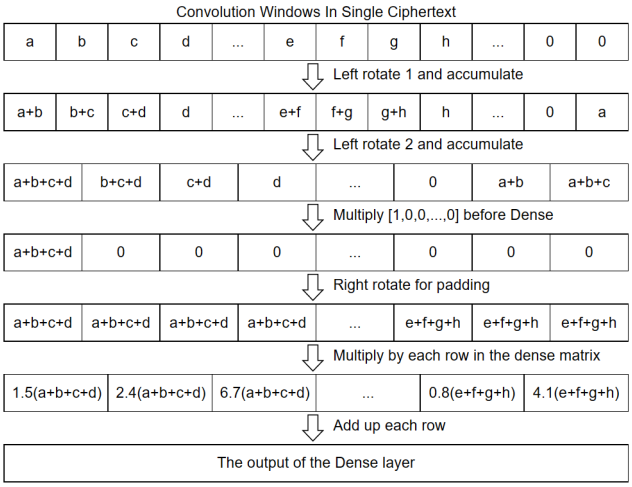

Figure 4. Simplified batch processes.

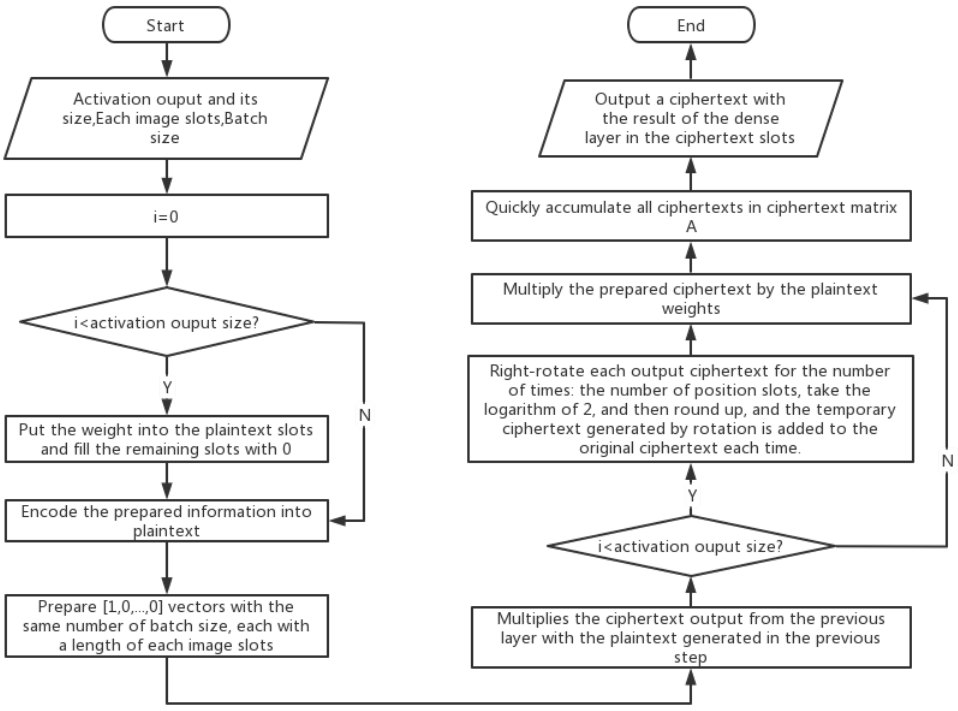

Figure 5. The flowchart of Algorithm 4.

Algorithm 4 Homomorphic Dense

Input: An activation output ciphertext vector A of the size f, dense weights matrix W of the size $d \times f$, d is the dense layer output size, dense bias vector b of the size $d \times 1$, each image slots s, batch size bs
Output: Ciphertexts ct.D of dense output vector

dense weights plain $dp \leftarrow 0$ ▷ $dp \in \mathbb{R}^{f \times (bs \times s)}$
for i := 0 to f **do**
  for j := 0 to bs **do**
    for q := 0 to d **do**
      dp_i.append($W_{[i][p]}$)
    end for
    for k := 0 to s-d **do**
      dp_i.append(0)
    end for
  end for
  $dp_i \leftarrow$ Encode(dp_i)
end for
for i := 0 to bs **do**
  V.append([1, 0, 0, . . . , 0, 0, 0]) ▷ append a vector that size is bs to V evert round
end for
$V \leftarrow$ Encode(V)
ct.$A \leftarrow$ *MultP*(ct.A,V) ▷ Only reserve the first slot of each window
for i := 0 to f **do**
  for j := 0 to $\lceil \mathtt{log2}(s) \rceil$ **do**
    temp $\leftarrow$ Rot($ct.A_i$, $-\mathtt{exp(j)}$) ▷ Move the first slot value to other slots
    $ct.A_i \leftarrow$ AddInplace(temp)
  end for
end for
for i := 0 to f **do**
  $ct.A_i \leftarrow$ MultPInplace(dp_i)
  $ct.A_i$.RescaletoNext()
end for
ct.$D \leftarrow$ AddMany(A) ▷ Add f ciphertexts from A together and store into ct.D

SIMD [33] technology has long been used by [5,15]. Cryptonets and MiniONN use CRT technology to split a single large number into multiple small numbers for parallel operations. We have expanded the use of SIMD technology to better integrate the input and output structures of neural networks. It should be noted that the method proposed by us can be applied to all matrix operations necessary for the output of the convolutional layer in all CNNs to enter the dense layer. In the process of connecting neurons, unlike the previous matrix multiplication, we expand each value of the output of the convolutional layer to the same size as the output dimension of the dense layer, and then perform a dot product operation with each row of the dense layer weight matrix, and finally add all the rows at once to compress the output into a ciphertext, and the change of the matrix to the output dimension is also indirectly transferred to the expansion process of the ciphertext slot.

4.3. Dealing with Noise Effects

In Section 3.4, the ciphertext encrypted using the CKKS scheme contains hierarchies, messages, and noise boundaries in addition to the ciphertext itself to dynamically manage the size of the ciphertext. A complete ciphertext can be seen as a quadruple form: $(c, l, v, B), c \in \mathcal{R}_{q_l}^k$. Compared with the general homomorphic encryption scheme, the CKKS scheme does not prepare an independent plaintext space for the embedded noise, and the message $m' = m + e$ output by the decryption algorithm is slightly different from the original message m, but when the maximum value $||e||_\infty^{can}$ can of the noise vector

is much smaller than the maximum value $||m||_{\infty}^{can}$ can of the message vector, it can be regarded as an approximation of the approximate calculation.

For two ciphertext (c_1, l, v_1, B_1) and (c_2, l, v_2, B_2), all parts of noisy if defined by $\beta_i = B_i/v_i$, $B_{scale} = \sqrt{N/3} \cdot (3 + 8\sqrt{h})$, $B_{ks} = 8\sigma N/\sqrt{3}$, $B_{mult}(l) = P^{-1} \cdot q_l \cdot B_{ks} + B_{scale}$. We multiply it once and scale the result to the next level, resulting in a relative error of:

$$\beta' = \beta_1 + \beta_2 + \beta_1\beta_2 + \frac{B_{mult}(l) + p^{l'-l}B_{scale}}{v_1 v_2} \tag{2}$$

We can approximate it as consuming one level for homomorphic operations, if it is directly decrypted, it will bring β'/q_l error to the message. Since our experimental process is trained with ciphertext, ciphertext reasoning, and the training process is adjusted in plaintext, this means that for the image of the ciphertext state, the weights originally trained may be somewhat unmatched.

The decryption result of the CKKS scheme decryption algorithm is actually an approximation of the plaintext, so the noise growth caused by homomorphism may bring some significant accuracy loss. Experimental tests are also carried out in the original text of the encryption scheme, and the output of the x^{1024} polynomial is calculated under certain parameters, and the accuracy of 10 bits is lost [26]. In our specific image classification privacy protection scheme, we can understand that after the picture is encrypted, even if it is immediately decrypted, the value of each pixel is not the original value, but this error is very small, and the error of the entire picture will make the image "blurry", which will have a certain impact on the inference result at the macroscopic level. For example, if we first scale a pixel with a grayscale value of 18 to the [0, 1] interval, the value is about 0.0705882353, and when the plaintext scale scale is 2^{40}, the noise carried by the encryption scheme will cause an error to the value of the eighth decimal place after decryption. After ciphertext inference, the error will rise to 5–6 decimal places, and the error of this size will have a certain impact on the accuracy of network reasoning.

To solve this problem, we believe that if similar noise is added during the training stage, the model can extract the information of the "blurred" state picture, so that the model's learning ability can resist the influence of noise after trying to add standard $\sigma = 5.0$ Gaussian distribution random sampling to the image of the training process as an approximate error, so as to achieve the accuracy effect generated by the above error analysis.

In the following Section 6.2, we show the validation experiment and the experimental results, and it turns out that our method has a certain effect.

5. Security Analysis

In this Section, we provide the security analysis of the homomorphic encrypted neural network inference scheme; for our solution, it needs to be proved that the ciphertext vector packaged by the local client (each ciphertext contains an extended convolution window) cannot obtain any valid information when the cloud service provider performs operations.

Definition 3. *Let $\mathcal{R}^{\vee}$ be the dual fractional ideal of $\mathcal{R}$ and write $\mathcal{R}_q^{\vee} = \mathcal{R}^{\vee}/q\mathcal{R}^{\vee}$. For a positive integer modulus $q \geq 2, s \in \mathcal{R}_q^{\vee}, r \in (\mathbb{R}^+)^N$, and an error distribution $\mathcal{X} := \lfloor \Psi_{\mathbf{r}} \rceil_{\mathcal{R}^{\vee}}$, we define $A_{N,q,\chi}(s)$ as the RLWE distribution obtained by sampling $a \leftarrow \mathcal{R}_q$ uniformly at random, $e \leftarrow \mathcal{X}$ and returning $(a, a \cdot s + e) \in \mathcal{R}_q \times \mathcal{R}_q^{\vee}$.*

Definition 4. *We define a problem to distinguish arbitrarily many independent samples chosen according to $A_{N,q,\chi}(s)$ for a random choice of s sampled from the distribution $\mathcal{D}$ over $\mathcal{R}^{\vee}$ from the same number of uniformly random and independent samples from $\mathcal{R}_q \times \mathcal{R}_q^{\vee}$ to be the (decision) ring learning with errors problem, denoted by $RLWE_{N,q,\chi}(\mathcal{D})$.*

Definition 5. *We define encoding procedure $Ecd(z; \Delta)$ as: For a $(N/2)-$dimensional vector $z = (z_i)_{i \in T}$ of complex numbers, the encoding procedure first expands it into the vector $\pi^{-1}(z) \in$*

$\mathbb{H}, \mathbb{H} = \left\{ z = (z_j)_{j \in \mathbb{Z}_M^*} \in \mathbb{C}^N : z_j = \overline{z_{-j}}, \forall j \in \mathbb{Z}_M^* \right\}$ *and computes its discretization to* $\sigma(\mathcal{R})$ *after multiplying a scaling factor* Δ. *Return the corresponding integral polynomial* $m(X) = \sigma^{-1}(\lfloor \Delta \cdot \pi^{-}1(z)\sigma R \rceil) \in \mathcal{R}$

Definition 6. *For a real* $0 \leq \rho \leq 1$, *we define* $\mathcal{ZO}(\rho)$ *as a distribution that draws each entry in the vector from* $v \in \{0, \pm 1\}^N$, *with probability* $\rho/2$ *for each of* -1 *and* $+1$, *and probability being zero* $1 - \rho$.

Definition 7. *We define* $\mathcal{DG}(\sigma^2)$ *as a sample procedure that samples a vector in* F^N *by drawing its coefficient independently from the discrete Gaussian distribution of variance* σ^2

Theorem 1 ([34]). *For and d that is a power of 2, ring* $\mathcal{R} = \mathbb{Z}[x]/\left(x^d + 1\right)$, *prime integer* $q = q(d) = 1 mod 2d$, *and* $B = \omega\left(d\sqrt{logd}\right)$, *there is an efficiently samplable distribution* χ *that outputs elements of* $\mathcal{R}$ *of length at most B with overwhelming probability, such that if there exists an efficient algorithm that solves* $RLWE_{d,q,\chi}$ *then there is an efficient quantum algorithm for solving* $f(d) \cdot (q/B)$ *-approximate worst-case SVP for ideal lattices over* $\mathcal{R}$, *for every super-polynomial factor* $f(d) = d^{\omega(1)}$.

Theorem 2 ([34]). *Let K denote an arbitrary number field of degree n. Let* $\alpha = \alpha(n) \in (0,1)$ *be arbitrary, and let the (rational) integer modulus* $q = q(n) \geq 2$ *be such that* $\alpha - q \geq \omega(\sqrt{logn})$. *There is a probabilistic polynomial-time quantum reduction from* $K - DGS\gamma$ *to* $O_K - LWE_{q,\Psi \leq \alpha}$, *where* $\gamma = \eta_\varepsilon I - \omega logn/\alpha$.

Our local client fills the convolution window of multiple images in the same position into a vector s, and enters the vector s_i each into the encoding function $Ecd(z; \Delta) \rightarrow p_i$, returning the plaintext polynomial p_i. Then specify the security level and call the homomorphic encryption scheme $KeyGen(1^\lambda) \rightarrow sk, pk, evk$, described in Section 3.2 to obtain the secret key sk which users keep themselves, the evaluation key evk is sent to the cloud sever directly. Then call $Enc_{pk}(p_i)$ with the returned public key pk, sampling $v_i \leftarrow \mathcal{ZO}(0.5)$ and $e_{i_1}, e_{i_2} \leftarrow \mathcal{DG}(\sigma^2)$, outputs $v_i \cdot pk + (p_i + e_{i_1}, e_{i_2})(mod\ q_L)$, encrypts the plaintext polynomial p_i to ciphertext c_i, ciphertext c_i satisfy $\langle ci, sk \rangle = p_i + e_i(mod q_L)$, $\langle \cdot \rangle$ is the inner product operation. According to Definition 3, the ciphertext form satisfies the uniform random sampling $A_{N,q,\chi}(c_i)$ on the RLWE distribution, thereby specifying the $RLWE_{N,q,\chi}(\mathcal{D})$ problem in Definition 4. From Theory 1, the approximate SVP problem on the ideal lattice can be reduced to $RLWE_{d,q,\chi}$. Then, the difficulty of restoring the p_i from $v_i \cdot pk + (p_i + e_{i_1}, e_{i_2})(mod\ q_L)$ can be based on the difficulty of the approximate SVP problem on the ideal grid. Vadim Lyubashevsky et al. [34] also provide approximate SVP (worst-case) to the search version of R-LWE on an ideal lattice (quantum reduction). When specifying the encryption scheme parameters, we need to consider that the security level of λ-bits [35,36] is met when $N \geq \frac{\lambda + 110}{7.2} log(P \cdot q_L)$ (P is the plaintext module), and the security level of 192 bits can be achieved when N is 2^{15}, and the maximum bit length of the coefficient modulo q_L is 611 bits. The total length of the coefficient modulus used in our scheme is 540 bits, so this scheme can meet the security level of λ = 192 bits. Therefore, the security of the information uploaded by users on the cloud service side can be guaranteed.

6. Performance Evaluation

In this Section, we evaluate the performance of our proposed privacy-preserving neural network security inference scheme. First, we test the time of atomic operations required in various inference processes of homomorphic encryption schemes under specific parameters, and calculate the total number of various operations in the entire inference process, and the test results are shown in the table of Section 6.1. In Section 6.2, we monitored the time spent by different network layers and memory usage during inference, and the test results are presented in tabular form.

Our experiment is outlined as follows: we propose a methodology for training neural networks on encrypted data using homomorphic encryption. Our approach involves evaluating the computation depth required for the model inference and choosing suitable encryption parameters, followed by evaluating the noise error generated by the ciphertext produced by this parameter after undergoing homomorphic ciphertext evaluation at this depth. We modify the model to adapt to the arithmetic range (linear, polynomial operations) of homomorphic encryption and embed the noise into the original data to generate adversarial samples for training the model. Our batched ciphertext inference algorithm is embedded into the inference process, and we use both the model trained with and without adversarial training to perform ciphertext inference simultaneously. We observe the similarity between their output probabilities and the output probabilities of plaintext inference to determine the impact of the noise and also observe the efficiency improvement of our batched ciphertext inference algorithm.

Our experimental environment is a CPU with Intel-i7 3.6 Ghz, as well as 128 GB of physical memory. The entire client-side encryption process and the inference process on the cloud service side are completed by the experimental environment.

6.1. Numerical Analysis

In this Section, we evaluate the performance of our proposed privacy-preserving neural network security inference scheme, and first we analyze the computational complexity of the entire cryptographic network at the arithmetic level.

We write the length and width of the image to be uploaded as $h \times w$, the length and width of the convolution kernel as $k \times k$, the step size as st, the number of channels as c, the length and width of dense layer 1 as $wh \times ww$, wh as the width of the input end, the length and width of dense layer 1 as $di \times do$, the number of plaintext slots allocated for each picture as s, and the time consumption of various homomorphic atomic operations is shown in Table 4.

Table 4. Every evaluation operation time costs and evaluation times.

Operation	Time Consumption(s)	Evaluation Times
Encryption	0.43155	$((h-k)/s+1)^2$
Multiplication(Ciphertext)	0.00714	$4 \times ((h-k)/s+1)^2$
Multiplication(Plaintext)	0.00198	$(c+4) \times ((h-k)/s+1)^2 + (wh+di+3)$
Relinearization	0.07435	$2 \times ((h-k)/s+1)^2 + 2$
Rescale	0.01009	$(c+8) \times ((h-k)/s+1)^2 + (wh+di+3)$
Rotation	0.01009	$c \times (\lceil \mathtt{log2}(k^2) \rceil + \lceil \mathtt{log2}(s) \rceil) \times ((h-k)/s+1)^2$
Add	0.00080	$(c+5)(\lceil \mathtt{log2}(k^2) \rceil + \lceil \mathtt{log2}(s) \rceil) \times ((h-k)/s+1)^2 + (3+do)$

6.2. Evaluation of Results

Previously, in Section 4.2 we provided improvements to the batch algorithm, in Section 4.3 we stated the effect of noise on the inference results in homomorphic schemes, and in this chapter, our experimental results are shared.

In order to verify whether random noise will have an effect at the end of the inference stage, and whether the above method has a certain effect, we verify it by the following method: first randomly select 64 pictures from the test set, and in the case of adding noise in the training phase, the test pictures with the same noise will be (so the test images all contain noise, only show the extraction results) and the output (without classifier) vector in the last layer of the network and the corresponding label vector (the correct result index position is 1, the rest of the index position is 0) to enable cosine similarity with the image without adding noise under the same model, and then compare the effect of the two training modes (the image position is the same), and the comparison results are shown in the Table 5. Under the parameters of ten epochs, we can see that there is a relatively large

improvement ratio observed. We speculate that the larger numerical deviation observed under this parameter is likely due to the use of a larger learning rate, which leads to insufficient stability of the model when facing perturbed tests. However, it can be seen that our method still maintains a stable similarity of about 10 between the model input and the original output under this parameter.

Table 5. The cosine similarity result of two training methods.

Epochs	Batch Size	Learning Rate	Cosine Similarity (Noisy)	Cosine Similarity (Original)	Promotion Ratio
10	64	1.0×10^{-3}	10.0167	7.0016	43.06%
10	64	2.0×10^{-3}	10.2742	4.9816	106.24%
20	64	1.0×10^{-4}	23.3822	21.3689	9.42%
10	128	1.0×10^{-3}	21.5057	21.3939	0.52%
15	128	8.0×10^{-4}	17.4479	17.0862	2.12%
12	64	2.0×10^{-3}	10.0247	6.0178	66.58%

In the model containing three convolutional layers and two fully connected layers for inference of the CIFAR-10 dataset using the same activation function, the cosine similarity of the output result and the target result increased by about 20% on average (under different training parameters). After experimental analysis, the cosine similarity of adding matching noise in the training stage is significantly improved compared with adding noise in the training process. It can be said that this method should deal with the noise generated by homomorphic operations, which can bring the prediction result closer to the target result. Using the last row of parameters of the table, plus 0.1 momentum, training with added noise can make the ciphertext inference containing noise reach 98.55% accuracy, and under the same parameters, the training process can only achieve 98.12% accuracy without adding noise, which can show that the noise of the homomorphic scheme has an impact on the accuracy of the ciphertext inference results, and our method can indeed offset this effect to a certain extent.

Next, we show the effect of the algorithm mentioned in Section 4.2 in a concrete experiment. In the case of N = 32,768, there are 16,384 slots available for plaintext, and our batch scheme prepares the convolution kernel area k^2 plus the number of columns of the dense layer 1 column at the input end for the convolution window of each picture, that is, the output dimension wh, a total of left $\left(k^2 + wh\right)$ plaintext slots, in the specific implementation, occupies left$(4 \times 4 + 100 = 116)$ plaintext slots. Theoretically, 16,384/116 = 141 pictures can be processed in parallel in the case of a single thread, and if you consider multi-threaded parallel processing, the pictures that can be processed synchronously depend on the performance of the actual environment, that is, the concurrency capacity of the CPU and the total amount of memory. We use the C++ standard thread library to perform 16-thread concurrent encrypted inference computations, and the entire process takes 27.15 s, (the time it took to contain the local client-side encrypted data), and this was obtained with the addition of some noise effect in the test set. Table 6 shows the time consumed by each stage of the ciphertext network inference process and the processing memory consumed by the runtime.

Compared with the inference scheme containing the fourth-order approximation Swish activation function mentioned in Takumi et al. [11], we consume about 4 s more time, but we choose polynomial modulos twice as large as it to provide higher security and deeper computational depth. We denote the output dimension of the last convolutional layer as c, and the size of the first dense layer is h $\times$ w(c = h), Table 7 displays the inference time, accuracy, and conversion complexity between different layers of the scheme.

Table 6. Inference stage time and memory consumption.

Layer	Evaluation Time (s)	Memory Consumption (mb)
Local Pretreatment	0.0875	1539
Convlutional Layer	13.843	6018
Activation Layer-1	10.65	27,922
FC-1	2.425	2676
Activation Layer-2	0.07	17
FC-2	0.075	239
Total	27.15	38,411

Table 7. Comparison of scheme parameters.

Framework	HE Inference Time(s)	Accuracy	Conv-Dense Multiplication
Cryptonets	249.6	98.95%	$h \times w$
CryptoDL	148.9	98.1%	$h \times w$
FCryptonets	39.1	98.71%	$h \times w$
Ours	27.15	99.05%	h

Our scheme can compress the number of ciphertexts in the process of connecting the convolutional layer and the dense layer, transfer the transformation of the network structure to the movement of the ciphertext slot, effectively reduce the computational complexity of the convolutional layer or pooling layer when connecting the dense layer in the inference process, and reduce a certain amount of data traffic during use.

7. Discussion

The experiments have demonstrated the effectiveness of our batch processing algorithm, and our noise handling method has also achieved some results. Therefore, during the adversarial training process, the author suggests using different homomorphic encryption parameters to encrypt numbers within the input range of the model, observing the magnitude of the error introduced by the encryption process, and then using adversarial sample generation algorithms (such as FGSM) to generate adversarial samples with errors similar to those introduced by encryption. The algorithm parameters can be adjusted until the similarity between the two is high, after which adversarial and original data can be jointly input into the model for training. This method can also be used for differential privacy-based model protection schemes to stabilize model accuracy. In addition, for privacy-preserving inference in language models, techniques such as secure multiparty computation (MPC) and garbled circuits (GC) can be used to avoid the huge computation and storage costs brought by homomorphic encryption schemes. It is worth further exploring whether there are more post-quantum cryptography methods that can be applied to the complex computing problems in deep learning.

8. Conclusions

In this paper, a batch method to optimize the inference of homomorphic encryption neural network is proposed, which can make full use of the ciphertext slot provided by the homomorphic encryption scheme, and the inference time when batching 141 pictures is 27.15 s, and the amortization time of a single image is 0.19 s, with an accuracy of 99.05%. At the same time, a new perspective is provided to study how random noise added by encryption schemes can add noise to the training process to counteract its effects. In our scenario, users can encrypt private images locally, and then securely perform inference and prediction on the cloud service side, for which we provide proof of security. One limitation is that the application scenarios of homomorphic encryption are still not extensive enough, and need to be further studied and expanded.

Author Contributions: Conceptualization: C.S.; methodology, C.S.; validation, C.S., R.H.; formal analysis, C.S.; writing—original draft preparation, C.S.; writing—review and editing, C.S.; funding acquisition, R.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation Project of China under Grant No. 62062009 and the Guangxi Innovation-driven Development Project under Grant Nos. AA17204058-17 and AA18118047-7.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: I would like to thank my senior Kunhong Li for helping me with experimental ideas, and my college classmate Guodong Zheng for answering my doubts in programming.

Conflicts of Interest: The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

1. Dwork, C. Differential privacy: A survey of results. In Proceedings of the International Conference on Theory and Applications of Models of Computation, Xi'an, China, 25–29 April 2008; Springer: Berlin/Heidelberg, Germany, 2008; pp. 1–19.
2. Rivest, R.L.; Adleman, L.; Dertouzos, M.L. On data banks and privacy homomorphisms. *Found. Secur. Comput.* **1978**, *4*, 169–180.
3. Yao, A.C. Protocols for secure computations. In Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (SFCS 1982), Chicago, IL, USA, 3–5 November 1982; pp. 160–164.
4. Gentry, C. Fully homomorphic encryption using ideal lattices. In Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing, Bethesda, MD, USA, 31 May–2 June 2009; pp. 169–178.
5. Gilad-Bachrach, R.; Dowlin, N.; Laine, K.; Lauter, K.; Naehrig, M.; Wernsing, J. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In Proceedings of the International Conference on Machine Learning, PMLR, New York, NY, USA, 20–22 June 2016; pp. 201–210.
6. Chabanne, H.; De Wargny, A.; Milgram, J.; Morel, C.; Prouff, E. Privacy-Preserving Classification on Deep Neural Network. Cryptology ePrint Archive. 2017. Available online: https://eprint.iacr.org/2017/035 (accessed on 15 November 2022).
7. Chou, E.; Beal, J.; Levy, D.; Yeung, S.; Haque, A.; Li, F.F. Faster cryptonets: Leveraging sparsity for real-world encrypted inference. *arXiv* **2018**, arXiv:1811.09953.
8. Bourse, F.; Minelli, M.; Minihold, M.; Paillier, P. Fast homomorphic evaluation of deep discretized neural networks. In Proceedings of the Annual International Cryptology Conference, Santa Barbara, CA, USA, 19–23 August 2018; Springer: Berlin/Heidelberg, Germany, 2018; pp. 483–512.
9. Sanyal, A.; Kusner, M.; Gascon, A.; Kanade, V. TAPAS: Tricks to accelerate (encrypted) prediction as a service. In Proceedings of the International Conference on Machine Learning, PMLR, Stockholm, Sweden, 10–15 July 2018; pp. 4490–4499.
10. Hesamifard, E.; Takabi, H.; Ghasemi, M. Cryptodl: Deep neural networks over encrypted data. *arXiv* **2017**, arXiv:1711.05189.
11. Brutzkus, A.; Gilad-Bachrach, R.; Elisha, O. Low latency privacy preserving inference. In Proceedings of the International Conference on Machine Learning, PMLR, Long Beach, CA, USA, 9–15 June 2019; pp. 812–821.
12. Ishiyama, T.; Suzuki, T.; Yamana, H. Highly accurate CNN inference using approximate activation functions over homomorphic encryption. In Proceedings of the 2020 IEEE International Conference on Big Data (Big Data), Atlanta, GA, USA, 10–13 December 2020; pp. 3989–3995.
13. Lu, Y.; Lin, J.; Jin, C.; Wang, Z.; Aung, K.M.M.; Li, X. FFConv: Fast factorized neural network inference on encrypted data. *arXiv* **2021**, arXiv:2102.03494.
14. Lee, J.W.; Kang, H.; Lee, Y.; Choi, W.; Eom, J.; Deryabin, M.; Lee, E.; Lee, J.; Yoo, D.; Kim, Y.S.; et al. Privacy-preserving machine learning with fully homomorphic encryption for deep neural network. *IEEE Access* **2022**, *10*, 30039–30054. [CrossRef]
15. Liu, J.; Juuti, M.; Lu, Y.; Asokan, N. Oblivious neural network predictions via minionn transformations. In Proceedings of the 2017 ACM Sigsac Conference on Computer and Communications Security, Dallas, TX, USA, 30 October–3 November 2017; pp. 619–631.
16. Juvekar, C.; Vaikuntanathan, V.; Chandrakasan, A. {GAZELLE}: A low latency framework for secure neural network inference. In Proceedings of the 27th USENIX Security Symposium (USENIX Security 18), Baltimore, MD, USA, 15–17 August 2018; pp. 1651–1669.
17. Li, S.; Xue, K.; Zhu, B.; Ding, C.; Gao, X.; Wei, D.; Wan, T. Falcon: A fourier transform based approach for fast and secure convolutional neural network predictions. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA, 13–19 June 2020; pp. 8705–8714.
18. Kumar, N.; Rathee, M.; Chandran, N.; Gupta, D.; Rastogi, A.; Sharma, R. Cryptflow: Secure tensorflow inference. In Proceedings of the 2020 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 18–21 May 2020; pp. 336–353.

19. Mishra, P.; Lehmkuhl, R.; Srinivasan, A.; Zheng, W.; Popa, R.A. DELPHI: A Cryptographic Inference Service for Neural Networks. In Proceedings of the 29th USENIX Conference on Security Symposium (SEC'20), Vancouver, BC, Canada, 16–18 August 2017.
20. Ng, L.K.; Chow, S.S. GForce: GPU-Friendly Oblivious and Rapid Neural Network Inference. In Proceedings of the 30th USENIX Security Symposium (USENIX Security 21), Online, 11–13 August 2021; pp. 2147–2164.
21. Paillier, P. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology, Proceedings of the EUROCRYPT'99: International Conference on the Theory and Application of Cryptographic Techniques Prague, Czech Republic, 2–6 May 1999*; Springer: Berlin/Heidelberg, Germany, 1999; Proceedings 18; pp. 223–238.
22. Fan, J.; Vercauteren, F. Somewhat Practical Fully Homomorphic Encryption. Cryptology ePrint Archive. 2012. Available online: https://eprint.iacr.org/2012/144 (accessed on 2 November 2022).
23. Brakerski, Z.; Gentry, C.; Vaikuntanathan, V. (Leveled) fully homomorphic encryption without bootstrapping. *ACM Trans. Comput. Theory (TOCT)* **2014**, *6*, 1–36. [CrossRef]
24. Ducas, L.; Micciancio, D. FHEW: Bootstrapping homomorphic encryption in less than a second. In *Advances in Cryptology, Proceedings of the EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, 26–30 April 2015*; Proceedings, Part I 34; Springer: Berlin/Heidelberg, Germany, 2015; pp. 617–640.
25. Chillotti, I.; Gama, N.; Georgieva, M.; Izabachène, M. TFHE: Fast fully homomorphic encryption over the torus. *J. Cryptol.* **2020**, *33*, 34–91. [CrossRef]
26. Cheon, J.H.; Kim, A.; Kim, M.; Song, Y. Homomorphic encryption for arithmetic of approximate numbers. In Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security, Hong Kong, China, 3–7 December 2017; Springer: Berlin/Heidelberg, Germany, 2017; pp. 409–437.
27. Jiang, X.; Kim, M.; Lauter, K.; Song, Y. Secure outsourced matrix computation and application to neural networks. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, Toronto, ON, Canada, 15–19 October 2018; pp. 1209–1222.
28. CIFAR-10—Canadian Institute For Advanced Research. Available online: https://www.cs.toronto.edu/~kriz/cifar.html (accessed on 1 May 2023).
29. Gulli, A.; Pal, S. *Deep Learning with Keras*; Packt Publishing Ltd.: Birmingham, UK, 2017.
30. Microsoft SEAL (Release 4.0); Microsoft Research, Redmond, WA, USA, 2022. Available online: https://github.com/Microsoft/SEAL (accessed on 15 July 2022).
31. Halevi, S.; Shoup, V. Algorithms in helib. In *Advances in Cryptology, Proceedings of the CRYPTO 2014: 34th Annual Cryptology Conference, Santa Barbara, CA, USA, 17–21 August 2014*; Proceedings, Part I 34; Springer: Berlin/Heidelberg, Germany, 2014; pp. 554–571.
32. Cheon, J.H.; Kim, A.; Yhee, D. Multi-Dimensional Packing for Heaan for Approximate Matrix Arithmetics. Cryptology ePrint Archive. 2018. Available online: https://eprint.iacr.org/2018/1245 (accessed on 15 November 2022).
33. Smart, N.P.; Vercauteren, F. Fully homomorphic SIMD operations. *Des. Codes Cryptogr.* **2014**, *71*, 57–81. [CrossRef]
34. Lyubashevsky, V.; Peikert, C.; Regev, O. On ideal lattices and learning with errors over rings. In Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques, Riviera, France, 30 May–3 June 2010; Springer: Berlin/Heidelberg, Germany, 2010; pp. 1–23.
35. Gentry, C.; Halevi, S.; Smart, N.P. Homomorphic evaluation of the AES circuit. In Proceedings of the Annual Cryptology Conference, Santa Barbara, CA, USA, 19–23 August 2012; Springer: Berlin/Heidelberg, Germany, 2012; pp. 850–867.
36. Lindner, R.; Peikert, C. Better key sizes (and attacks) for LWE-based encryption. In Proceedings of the Cryptographers' Track at the RSA Conference, San Francisco, CA, USA, 14–18 February 2011; Springer: Berlin/Heidelberg, Germany, 2011; pp. 319–339.