This CVPR 2020 paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the accepted version;
the final published version of the proceedings is available on IEEE Xplore.

FALCON: A Fourier Transform Based Approach for Fast and Secure Convolutional Neural Network Predictions

Shaohua Li[1] Kaiping Xue[1] Bin Zhu[1] Chenkai Ding[1] Xindi Gao[1] David Wei[2] Tao Wan[3]
[1]University of Science and Technology of China [2]Fordham University [3]CableLabs
lshhsl@mail.ustc.edu.cn kpxue@ustc.edu.cn {cnzb01, chimkie, khronos}@mail.ustc.edu.cn
dsl.wei01@gmail.com t.wan@cablelabs.com

Abstract

Deep learning as a service has been widely deployed to utilize deep neural network models to provide prediction services. However, this raises privacy concerns since clients need to send sensitive information to servers. In this paper, we focus on the scenario where clients want to classify private images with a convolutional neural network model hosted in the server, while both parties keep their data private. We present FALCON, a fast and secure approach for CNN predictions based on fast Fourier Transform. Our solution enables linear layers of a CNN model to be evaluated simply and efficiently with fully homomorphic encryption. We also introduce the first efficient and privacy-preserving protocol for softmax function, which is an indispensable component in CNNs and has not yet been evaluated in previous work due to its high complexity.

1. Introduction

Deep learning has been applied to quite a few fields to overcome the limitations of traditional data processing methods, such as image classification [19, 17], speech recognition [3, 4], medical diagnosis [28, 11], etc. Some companies and institutions have also invested in deep learning technologies, and trained their own deep neural network models to provide users with paid or free services. For example, Google Vision [9] provides an API for image classification for developers and general users, and one can upload an image to the cloud to obtain the classification result and its corresponding probability. Although these services provide rich experiences to users, they also cause serious privacy concerns because uploaded user data may contain private information [29], such as face pictures and X-ray images. Although many companies claim that they will never leak or use users' data for commercial purposes, the increasing number of data leaks alert us that there is no guarantee on what they promised [2].

Certainly, the clients' input data is not the only sensitive information, because for servers, their own models also need to be protected from adversarial clients. First, models may be trained with large amount of private data, e.g., medical records to obtain a model for disease prediction. Thus, sensitive information could be extracted from a trained model if disclosed to a malicious client [30, 31]. Second, model parameters and detailed prediction results, i.e., accurate probabilities over all classes, can be used to generate adversarial examples to deceive deep learning models [27, 8], to result in incorrect classification results. Third, many prediction models themselves, even without considering sensitive training data, require intellectual property protection and cannot be disclosed to third parties including their clients [21, 18].

To tackle this problem, researchers have put forward a secure deep learning scenario, where the server has a model, the client has data, and these two interact in such a way that the client can obtain the prediction result without leaking anything to the server, while learning nothing about the model. They usually use homomorphic encryption (e.g., CryptoNets [15]), secure multi-party computation [34] (e.g., MiniONN [21]), or their combination (e.g., GAZELLE [18]), for secure evaluation.

In this paper, we focus on the fast and secure solution for Convolutional Neural Networks (CNNs), one of the most important neural networks in deep learning. CNNs are characterized by the spatial input data, such as images and speeches. Typically, a CNN model consists of convolutional, activation, pooling, and fully-connected layers, and often follows by a softmax layer. Convolutional and fully-connected layers have linear property, while activation and pooling are nonlinear layers. The softmax layer is used to normalize the output into probabilities, usually used as the last layer of a CNN. The softmax layer is indispensable in many use cases. For example, a CNN model classifies an X-Ray image into "Pneumonia" with probability 5%. Although "Pneumonia" is the top label, the real result indicates that the patient probably has no such disease, and this

cannot be known without the probability output. Because the softmax function involves division and exponentiation that would introduce incredibly high overhead when evaluating with privacy tools, the existing work, e.g., CryptoNets, MiniONN and GAZELLE, used argmax function instead of softmax to obtain only the top one label. We, however, propose a novel efficient protocol for the softmax layer.

In this paper, we propose FALCON for fast and secure convolutional neural network predictions. Our contributions can be summarized as follows:

- For convolutional and fully-connected layers, we secure them with fully homomorphic encryption and achieve high efficiency by fast Fourier Transform (FFT) based ciphertext calculation.
- For ReLU and pooling layers, we propose a secure two-party computation protocol to evaluate them. Their evaluation efficiency is further improved by our optimized processing pipeline.
- For Softmax function, its secure evaluation has not yet been addressed by any previous work. We propose an *Approximate Theorem* to simplify calculation for softmax function, based on which we implement the first secure and efficient two-party softmax computation protocol.

2. Related Work

CryptoNets [15] inspired us to process neural network models securely with leveled homomorphic encryption (LHE). Since only LHE is used, CryptoNets needs to replace nonlinear activation and pooling functions with linear functions and re-train the model. SecureML [23] proposed protocols based on secure two-party computation for training several kinds of machine learning models between two non-colluding servers. DeepSecure [25] used Yao's Garbled Circuits only to enable scalable execution of neural network models between semi-trusted client and server. Chameleon [24] used additively secret sharing [6], Yao's Garbled Circuits [34], and GMW protocol [16] to implement secure CNN evaluation. But it requires an extra semi-honest third party. MiniONN [21] transformed a neural network model into an oblivious version, and used additively homomorphic encryption to generate multiplication triplets first, and then evaluated the model using secure two-party computation efficiently. GAZELLE [18] utilized the fully homomorphic encryption and designed efficient schemes for privacy-preserving convolution and matrix-vector multiplication operations. GAZELLE used homomorphic encryption in convolutional and fully-connected layers, and secure two-party computation in ReLU and pooling layers. Since MiniONN and GAZELLE outperform all previous work, we compare FALCON with them to show our performance superiority.

2.1. Building Blocks

2.2. Fast Fourier Transform

In image processing, the well-known fast Fourier Transform (FFT) is an algorithm that can convert an image from its space domain to a representation in the frequency domain and vice versa [33]. Letting $f(x, y)$ denote the pixel value of an image at point (x, y), after FFT, $f(x, y)$ will turn to $\mathcal{F}_f(u, v)$, which is a complex number. For simplicity, we denote the FFT of input $\mathbf{x}$ as $\mathcal{F}(\mathbf{x})$. An important property of the FFT used in this paper is *linearity*, i.e. for two inputs $\mathbf{x}$ and $\mathbf{y}$, we have:

$$\mathcal{F}(\mathbf{x}) + \mathcal{F}(\mathbf{y}) = \mathcal{F}(\mathbf{x} + \mathbf{y}). \tag{1}$$

FFT also has an important *Convolution Theorem*: the convolutions in the space domain are equivalent to pointwise products in the frequency domain. Denoting $\mathcal{F}^{-1}$ as the inverse FFT, the convolutions between x and y can be computed by:

$$\mathbf{x} * \mathbf{y} = \mathcal{F}^{-1}(\mathcal{F}(\mathbf{x}) \cdot \mathcal{F}(\mathbf{y})). \tag{2}$$

2.2.1 Lattice-based Homomorphic Encryption.

To implement privacy-preserving convolutions in FALCON, we require two kinds of homomorphic operations: SIMDAdd and SIMDMul. The SIMD here means we can pack a vector of plaintext elements into a ciphertext, and perform calculations on ciphertexts corresponding to each plaintext element, which reduces required ciphertext size and evaluation time. The SIMDAdd represents homomorphic addition between two ciphertexts, while the SIMDMul represents homomorphic multiplication between a ciphertext and a plaintext. All these requirements can be satisfied by modern lattice-based homomorphic encryption systems [14, 12, 7]. There are three required parameters in these schemes, namely number of plaintext slots n, plaintext module p, and ciphertext module q. Parameter n is the maximum number of data that can be processed in SIMD style. Parameter p limits the range of plaintext data. Parameter q can be calculated from given n and p. In this paper, we denote the ciphertext of $\mathbf{x}$ as $[\mathbf{x}]$.

2.2.2 Secure Two-Party Computation.

Secure two-party computation protocols allow two parties to jointly evaluate functions on each other's private data while preserving their privacy. Functions are represented as *boolean circuits* and then computed by these protocols. Yao's Garbled Circuits is a representative implementation of such protocols and will be used in this paper for the secure computation between a client and a server.

The ABY framework [10] is an open source library that supports secure two-party computation, and we use this library to implement secure ReLU, Max Pooling and softmax layer. This library has encapsulated several basic operations

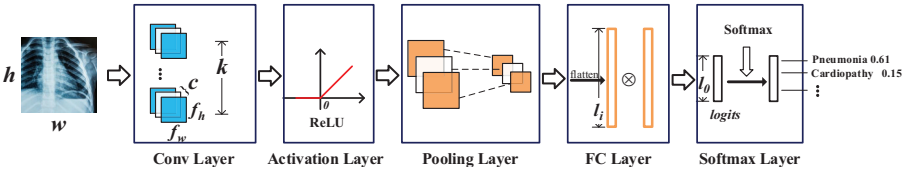

Figure 1. An example of convolutional neural networks.

for secure computation, and we here introduce the operations used in this paper (Note that, the term "share' used in what follows means Yao sharing, which is a type of sharing used by Yao's Garbled Circuits.):

– *ADDGate(a, b)* performs an arithmetic addition on input shares a and b, and returns the result as a share.

– *SUBGate(a, b)* performs an arithmetic subtraction on input shares a and b, and returns the result as a share.

– *GTGate(a, b)* performs a ternary operation "$a > b$? 1 : 0", and returns 1 if $a > b$, 0 otherwise.

–*MUXGate(a, b, s)* performs as a multiplexer, and returns a if s is 1, returns b if s is 0.

3. FALCON Execution Flow

Consider such a scenario that a doctor wants to learn the potential disease a patient might have from an X-Ray image, only knowing the top label without the corresponding probability may lead to unreliable diagnostic result. For example, the output top label "Pneumonia" with probability "0.9" and "0.1" definitely have different meanings for treatment. In this section, we will outline the execution flow using the convolutional neural network shown in Fig. 1.

System model. We consider a client *C* who wants to predict an input (e.g. an X-Ray image) with a convolutional neural network model held by a server *S*. For client *C*, the input is private. For server *S*, the parameters of convolutional and fully connected layers are also private. Our design goal is to preserve privacy for both parties when evaluating CNN models. We assume that both *C* and *S* are $semi-honest$. That is, they adhere to the execution flow defined by FALCON protocols, while trying to learn the other party's private information as much as possible.

Privacy guarantees. For server *S*, FALCON protects the following information about the model: all the weight parameters of convolutional and fully-connected layers, and the filter size of convolutional layers. FALCON does not hide the model architecture, i.e., the type of layer, layer size (the number of neurons in a layer), and the number of layers. For client *C*, FALCON leaks no information about the input content but do not protect the input size.

Execution flow at a high leverl.
At the beginning, *C* holds an input vector $\mathbf{x}$ and the private key, and *S* holds the neural network model. To evaluate the first layer, which is mostly a convolutional layer, *C* encrypts the FFT of $\mathbf{x}$, denoted by $[\mathcal{F}(\mathbf{x})]$, and transfers it to *S*. Then, *S* and *C* together do the following:

1. **(Evaluate the Conv layer)** *S* feeds the convolutional layer with $[\mathcal{F}(\mathbf{x})]$ and obtains the output $[\mathcal{F}(\mathbf{y})]$, where $\mathbf{y}$ is the plaintext output. In order to compute the next activation layer, *S* and *C* will each hold an additive share of $\mathbf{y}$, i.e., $\mathbf{x}^S + \mathbf{x}^C = \mathbf{y}$. This can be done by the proposed translation method. (See details in Section 4.2.)
2. **(Evaluate the ReLU layer)** For the ReLU layer, *S* and *C* run the designed boolean circuits for the ReLU function. Note that, we still require that the output value is additively shared by two parties. (See details in Section 4.3.2.)
3. **(Evaluate the pooling layer)** Evaluating the mean pooling function on two additive shares is simple. We can have these two parties perform mean pooling on their shares respectively. For Max Pooling, we also design boolean circuits to realize it. Note that, the same as ReLU, we need to ensure that *S* and *C* additively share the output value. (See details in Section 4.3.3.)
4. **(Evaluate the FC layer)** Typically, a FC layer is treated as matrix multiplication. In our design, we first convert this layer into an equivalent Conv layer and then use the same method in the convolutional layer to evaluate. Note that the input to this layer is additively shared by *S* and *C*, so we need to translate from shares to ciphertexts as described in Section 4.1. (See details in Section 4.4.)
5. **(Evaluate the softmax layer)** The input to softmax function is generally the output of a FC layer. In our design, we first have *S* and *C* additively share the input. Then we disassemble the softmax function into a max and an inner product function to enable the client *C* to efficiently obtain the target class with probability. (See details in Section 4.5.)

4. FALCON Design

4.1. Setup

Before moving on to the implementation details of each layer, we first introduce the encryption method and the translation between a ciphertext and additive shares. At the beginning, the client *C* holds the input $\mathbf{x}$ and needs to transfer its ciphertext to the server *S*. In our design, all ciphertexts correspond to plaintext data in the frequency domain.

That is, for input $\mathbf{x}$ of size $w \times h$, the client *C* first performs FFT to obtain $\mathcal{F}(\mathbf{x})$, and then encrypts it to $[\mathcal{F}(\mathbf{x})]$. The $\mathcal{F}(\mathbf{x})$ inherits the size of $\mathbf{x}$, but every element of it is a complex number, e.g. $\mathcal{F}(\mathbf{x})_{0,0} = a + bj$ where a and b are real numbers. Note that we cannot apply homomorphic encryption directly to complex numbers. Thus, we let the client *C* encrypt the real parts (e.g., a) and the imaginary parts (e.g., b) into two ciphertexts respectively. That is, for every element in $\mathcal{F}(\mathbf{x})$, *C* packs all the real parts into a plaintext vector and encrypts this vector, which is denoted as $[\mathcal{F}(\mathbf{x})_R]$. Accordingly, the ciphertext of all the imaginary parts is denoted as $[\mathcal{F}(\mathbf{x})_I]$. All the ciphertexts involved in the FALCON have this form.

The output of a linear layer, i.e., convolutional and fully connected layer, is a ciphertext, while the input to a non-linear layer is additive shares. Therefore, before feeding the output of a linear layer into a non-linear layer, we need to translate from a ciphertext to additive shares. Assume that the output of a linear layer is $[\mathcal{F}(\mathbf{y})]$, which is actually $[\mathcal{F}(\mathbf{y})_R]$ and $[\mathcal{F}(\mathbf{y})_I]$, the goal of server *S* and client *C* is to respectively obtain $\mathbf{x}^S$ and $\mathbf{x}^C$, satisfying $\mathbf{x}^S + \mathbf{x}^C = \mathbf{y}$ and guaranteeing no information about $\mathbf{y}$ will be exposed to either *S* or *C*. In order to achieve this goal, *S* generates a random vector $\mathbf{r}$ of the same size to $\mathbf{y}$, and performs the FFT to obtain $\mathcal{F}(\mathbf{r})_R$ and $\mathcal{F}(\mathbf{r})_I$. Using the SIMDAdd, *S* adds these values to the ciphertext homomorphically to obtain $[\mathcal{F}(\mathbf{y})_R - \mathcal{F}(\mathbf{r})_R]$ and $[\mathcal{F}(\mathbf{y})_I - \mathcal{F}(\mathbf{r})_I]$. Recall the linearity of the FFT shown in Eq. 1, we have $[\mathcal{F}(\mathbf{y})_R - \mathcal{F}(\mathbf{r})_R] = [\mathcal{F}(\mathbf{y} - \mathbf{r})_R]$ and $[\mathcal{F}(\mathbf{y})_I - \mathcal{F}(\mathbf{r})_I] = [\mathcal{F}(\mathbf{y} - \mathbf{r})_I]$.

The client *C* decrypts them and combines the imaginary parts with the real parts to obtain $\mathcal{F}(\mathbf{y} - \mathbf{r})$, and then performs the inverse FFT to get $(\mathbf{y} - \mathbf{r})$. Letting $\mathbf{x}^S = \mathbf{r}$ and $\mathbf{x}^C = (\mathbf{y} - \mathbf{r})$, we have the $\mathbf{y}$ be additively shared.

To translate from additive shares to ciphertexts, namely feed the output of non-linear layer to linear layer, we can run the reverse of above process. Note that FALCON works in $\mathbb{Z}_p$, where p is the selected plaintext module for homomorphic encryption. For any intermediate value x_m, $x_m < \lfloor p/2 \rfloor$ implies x_m is positive, otherwise negative.

4.2. Secure Convolutional Layer

The input image (or feature map) to a Conv layer is $\mathbf{x}$ of size $w \times h \times c$, where w and h are respectively width and height and c is the number of channels. We assume that there are a total of k filters (or kernels) in a Conv layer, each of which has a size of $f_w \times f_h \times c$. In this part, we first introduce a simple case where the input has single channel ($c = 1$) and the layer has only one filter ($k = 1$) to present our key idea. Then we describe a more general case where $c > 1$ and $k > 1$.

Simple Case ($c = 1, k = 1$). Firstly, client *C* performs FFT on input $\mathbf{x}$ of size $w \times h$ to obtain $\mathcal{F}(\mathbf{x})_R$ and $\mathcal{F}(\mathbf{x})_I$; server *S* performs FFT on filter $\mathbf{f}_i$ of size $f_w \times f_h$ to obtain $\mathcal{F}(\mathbf{f}_i)_R$ and $\mathcal{F}(\mathbf{f}_i)_I$. Secondly, client *C* encrypts FFT results as $[\mathcal{F}(\mathbf{x})_R]$ and $[\mathcal{F}(\mathbf{x})_I]$, and sends them to server. Assuming the convolution result of $\mathbf{x}$ and $\mathbf{f}_i$ is $\mathbf{y}$, server *S* does the following calculations:

$$[\mathcal{F}(\mathbf{y})_R] = [\mathcal{F}(\mathbf{x})_R] \otimes \mathcal{F}(\mathbf{f}_i)_R \ \oplus \ [\mathcal{F}(\mathbf{x})_I] \otimes (-\mathcal{F}(\mathbf{f}_i)_I),$$
$$[\mathcal{F}(\mathbf{y})_I] = [\mathcal{F}(\mathbf{x})_R] \otimes \mathcal{F}(\mathbf{f}_i)_I \ \oplus \ [\mathcal{F}(\mathbf{x})_I] \otimes \mathcal{F}(\mathbf{f}_i)_R,$$

where "$\otimes$" represents SIMDMul, and "$\oplus$" represents SIMDAdd. Then *S* generates a random vector $\mathbf{r}$ of size $w \times h$, and encrypts its FFT values as $[-\mathcal{F}(\mathbf{r})_R]$ and $[-\mathcal{F}(\mathbf{r})_I]$. Finally, *S* sends the following two ciphertexts to *C*:

$$[\mathcal{F}(\mathbf{y} - \mathbf{r})_R] = [\mathcal{F}(\mathbf{y})_R] \oplus [-\mathcal{F}(\mathbf{r})_R],$$
$$[\mathcal{F}(\mathbf{y} - \mathbf{r})_I] = [\mathcal{F}(\mathbf{y})_I] \oplus [-\mathcal{F}(\mathbf{r})_I].$$

Client *C* decrypts the ciphertexts, combines the real parts with the imaginary parts, and performs the inverse FFT to obtain $(\mathbf{y} - \mathbf{r})$, which is set to *C*'s share $\mathbf{x}^C$. Then, server *S* sets $\mathbf{r}$ to its share $\mathbf{x}^S$. At this point, the convolutional layer has been evaluated, and the result $\mathbf{y}$ is additively shared by *S* and *C*.

General Case. In order to present our idea clearly, we first explain how to calculate the convolution in the plaintext domain and then the ciphertext domain.

(Plaintext domain) For the filter $\mathbf{f}_i$ ($i \in [1, k]$) that contains c channels $\mathbf{f}_{i1}, \mathbf{f}_{i2}, \cdots, \mathbf{f}_{ic}$, and the input $\mathbf{x}$ that contains c channels $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_c$, firstly, $c \times$ 2-D filters and $c \times$ 2-D inputs are transformed using the FFT, then the corresponding channels are multiplied to get c intermediate results. Finally, these intermediate results are added to obtain the final output in the frequency domain.

(Ciphertext domain) Consider these 2-D filters and inputs as c independent groups, server *S* applies the calculation process shown in the simple case to these groups to get c intermediate ciphertexts, and then returns them to client *C* after masking with random vector $\mathbf{r}$. *C* decrypts these ciphertexts and adds them up in the plaintext domain. Let $\mathbf{y}$ denote real output values, then at this point, shares of *S* and *C* are respectively $\mathbf{r}$ and $\mathbf{y}$-$\mathbf{r}$.

Security analysis. The input data of client *C*, weight parameters, and the size of filters require protection. Since the input data remain encrypted during the evaluation of server *S*, the data are protected. Client *C* only obtains the masked convolutional result, and thus learns nothing about the weight parameters. Because filters are padded into the same size with the input, their size is also preserved.

4.3. Secure Activation Layer & Pooling Layer

In what follows, we first introduce the data preprocessing, which translates additive shares to Yao sharing. It also guarantees that Yao sharing lies in $[0, p)$. Then we present implementations for ReLU and Max Pooling.

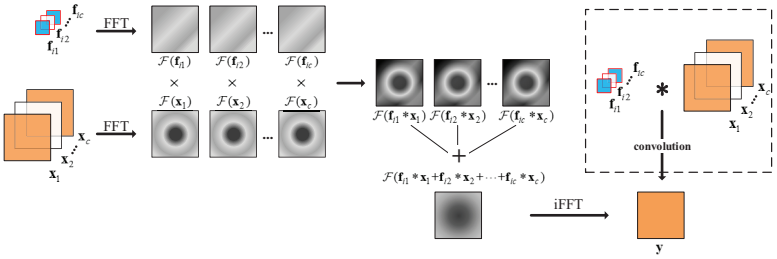

Figure 2. The convolution operations for multiple channels in plaintext.

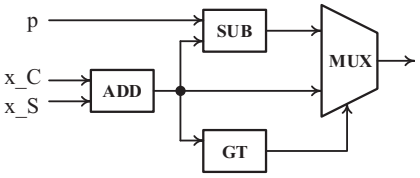

Figure 3. Boolean circuits for data preprocessing.

```
1 /*data preprocessing, returns (x_C+x_S)%p in
      Yao Sharing*/
2 yshr DataPreprocessing(yshr x_C, yshr x_S,
      yshr p){
3     yshr x, gt, dif, res;
4     x = ADDGate(x_C, x_S);
5     gt = GTGate(x, p);
6     dif = SUBGate(x, p);
7     res = MUXGate(x, dif, gt);
8     return res;
9 }
```

Listing 1. Function description of data preprocessing.

```
1 /* ReLU, returns max(x, 0);
2 x is the output of DataPreprocessing();
3 p_2 is p/2. */
4 yshr ReLU(yshr x, yshr p_2){
5     yshr gt, res;
6     gt = GTGate(x, p_2);
7     res = MUXGate(x, 0, gt);
8     return res;
9 }
```

Listing 2. Function description of ReLU.

4.3.1 Data preprocessing.

Assume that $\mathbf{x}^C = \{x_1^C, x_2^C, \cdots, x_N^C\}$ and $\mathbf{x}^S = \{x_1^S, x_2^S, \cdots, x_N^S\}$ are the additive shares held by client C and server S, respectively. Since both x_i^C and x_i^S belong to $[0, p)$, we have $x_i^C + x_i^S$ belongs to $[0, 2p)$. Since FALCON works in $\mathbb{Z}_p$, we need to limit the sum of two input shares to $[0, p)$. The illustration of boolean circuits for data preprocessing is shown in Fig. 3 and the pseudocode is shown in Listing 1. We first use *ADDGate* to recover true value (line 4) and judge if it exceeds p or not (line 5). A *MUXGate* is then used to select x or $p - x$ (line 7). All calculations are performed on Yao sharings and leak no information.

4.3.2 Secure ReLU layer.

Typically, Conv layers and non-last FC layers are followed by ReLU layer, which is $f(\mathbf{x}) = \max(\mathbf{x}, 0)$. In our setting, the input $\mathbf{x}$ is additively shared by client C and server S, i.e. $\mathbf{x}^C + \mathbf{x}^S = \mathbf{x}$. Our aim is to enable that C and S additively share $\max(\mathbf{x}, 0)$. That is, C holds $\max(\mathbf{x}, 0) - \mathbf{r}$ while S holds $\mathbf{r}$, where $\mathbf{r}$ is randomly generated by S. The pseudocode is shown in Listing 2. The first *GTGate* performs a great-than operation ($>$) (line 6), and the output is passed to *MUXGate* to select the positive x or 0 as the result (line 7).

4.3.3 Secure pooling layer.

Pooling layer performs down-sampling by dividing the input into rectangular pooling regions and computing the mean or maximum of each region. To evaluate mean pooling, we can simply let client C and server S compute the mean value of their respective shares. Our focus is the evaluation of Max Pooling. Letting $\mathbf{x}_{\text{region}} = \{x_1, x_2, \cdots, x_k\}$ be one of the rectangular pooling regions, our aim is to calculate $\max(x_1, x_2, \cdots, x_k)$. The pseudocode of designed boolean circuits is shown in Listing 3.

Since the input has been limited from 0 to $p/2$ by ReLU, we can iteratively compare two elements to obtain the max element with *GT* and *MUX* circuits without considering the existence of negative elements (line 11-14). Because comparisons are performed inside each region, we pack N elements into k vectors of size N/k via *SubsetGate* (line 9).

4.3.4 The Optimized ReLU and Max Pooling layers.

In a typical processing pipeline, a ReLU layer is followed by a Max Pooling layer, and the basic operation of both is $\max()$. Assume that $\mathbf{x}_{\text{region}} = \{x_1, x_2, \cdots, x_k\}$ is one of the rectangle pooling regions but has not applied to ReLU. Then, the final output of the Max Pooling layer should be

$$\max\left(\max\left(x_1, 0\right), \max\left(x_2, 0\right), \cdots, \max\left(x_k, 0\right)\right),$$

where the inside $\max()$ corresponds to the ReLU function while the outside is the Max Pooling function.We can find that this process is equivalent to the following one:

$$\max\left(\max\left(x_1, x_2, \cdots, x_k\right), 0\right),$$

where the inside $\max()$ can be considered as the Max Pooling function while the outside as the ReLU. Based on this observation, reversing the position of ReLU layer and Max Pooling layer in the processing pipeline will reduce the number of $\max()$ operations. An example is shown in Fig. 4. In fact, this trick has been proposed in the study of deep learning [1]. Nevertheless, due to the fact that ReLU and Max Pooling functions are relatively much cheaper than the heavy Conv and FC layers in the plaintext domain, this optimization has been discarded. However, in the ciphertext domain, all these functions have great impacts on the overall performance. We report and utilize this optimization here to further improve the FALCON performance. With this ap-

```
1  /* MaxPooling, returns max(x1,x2,...,xk);
2  x is the output of ReLU();
3  p_2 is p/2; k is the region size. */
4  yshr MaxPooling(yshr x, yshr p_2, int k){
5     yshr x_reg[k], gt, res;
6     // split the x into k pieces.
7     x_reg = SubsetGate(x);
8     res = x_reg[0];
9     for (int i = 1; i < k; i++) {
10       gt = GTGate(res, x_reg[i]);
11       res = MUXGate(res, reg[i], gt);
12    }
13    return res;
14 }
```

Listing 3. Function description of Max Pooling.

```
1  /* Optimized MaxPooling and ReLU;
2  x is the output of DataPreprocessing();
3  p_2 is p/2; k is the region size. */
4  yshr OptMaxPoolingAndReLU(yshr x, yshr p,
       yshr p_2, int k){
5     yshr x_reg[k], gt, res;
6     // Firstly, perform Max Pooling.
7     x_reg = SubsetGate(x);
8     res = x_reg[0];
9     for (int i = 1; i < k; i++) {
10       gt = GTGate(res, x_reg[i]);
11       res = MUXGate(res, reg[i], gt);
12    }
13    //Secondly, perform ReLU.
14    gt = GTGate(res, p_2);
15    res = MUXGate(res, 0, gt);
16    return res;
17 }
```

Listing 4. Function description of our Max Pooling and ReLU.

proach, we can see that the number of $\max()$ operations in the Max Pooling layer does not change, but the ReLU layer is reduced greatly. For a Max Pooling layer with (2×2) region with a stride of 2, our method can save 75% of ReLU operations. The pseudocode is shown in Listing 4. We first apply Max Pooling operations to obtain max values of each region (line 9-14), and ReLU operations follow to filter out all negative values (line 16-17).

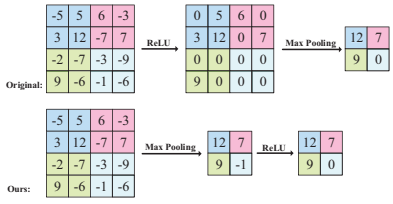

Figure 4. Original ReLU and Max Pooling v.s. Ours.

4.3.5 Output circuits for ReLU and Max Pooling

The original outputs of ReLU or Max Pooling circuits are in the form of Yao sharing. Our aim is to additively share the result between *C* and *S*. This can be achieved by taking the output $\mathbf{y}$ and a random vector $(-\mathbf{r} \bmod p)$ generated by *S* as two inputs of data preprocessing circuits, the result of which is $(\mathbf{y} - \mathbf{r} \bmod p)$ and is sent to *C*.

Security analysis. Since ReLU and Max Pooling layers do not have private model parameters, we only focus on the confidentiality of the input. Due to the security of Yao's Garbled Circuits, the input data are hidden.

4.4. Secure Fully Connected Layer

Normally, a FC layer can be treated as multiplication of a weight matrix and an input vector, and this can be executed very fast in the plaintext domain. However, in the ciphertext domain, this kind of multiplication is expensive. Inspired by the observation that FC layers can be viewed as convolutional layers with filters that cover the entire input regions [22], we propose an efficient solution by transforming the FC layer to the convolutional layer first, then utilizing the acceleration method in Section 4.2 to evaluate the FC layer.

4.5. Secure Softmax Layer

In classification CNNs, the last FC layer is always followed by a softmax layer to generate probability distribution over K different possible classes. However, to our best knowledge, in all previous work, researchers presented that the server can return *logits* to the client, who could obtain probabilities by performing softmax function locally, e.g. GAZELLE, or the client runs *argmax* using secure two-party computation to only obtain the classification result without knowing *logits* and probabilities, e.g. MiniONN. The main reason why these schemes bypass the encrypted computation is that implementing softmax function would introduce high computation complexity, no matter using homomorphic encryption or secure two-party protocols.

This high computation overhead is due to the division and exponentiation operations in the softmax function and we thus propose a division and exponentiation free protocol in FALCON. We notice that in a client-server scenario, by only accessing prediction results, *C* is able to extract model information [32, 30, 26]. To tackle this issue, *S* can only return necessary results, i.e., the class to which the input belongs and its corresponding probability, to *C*. Softmax function is given by

$$f(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{k=1}^{K} e^{x_k}}, \quad \text{for } i = 1, 2, \cdots, K,$$

where $f(\mathbf{x})_i$ is the probability that the input belongs to the class i. Letting the target class be t, our aim is to calculate $p_t = \frac{e^{x_t}}{\sum_{k=1}^{K} e^{x_k}}$. Before moving to the detailed protocols, we first give the following theorem:

Theorem 1. *(Approximation Theorem) For* $p_t = \frac{e^{x_t}}{\sum_{k=1}^{K} e^{x_k}}$, *where* $x_t = \max(x_1, \cdots, x_K)$, *and* $p'_t = \frac{e^{x_t}}{\sum\limits_{x_k \geq x_t - m} e^{x_k}}$, *where* $m \geq ln\left[(10^l - 1)(K - 1)\right]$ *and* $l \geq 1$, *we have* $|p_t - p'_t| \leq 10^{-l}$.

(*The proof can be found in the full version [20].*)

This theorem shows that in the case of a precision requirement of 10^{-l}, we can replace p_t with p'_t. In another word, we can set a threshold $x_t - m$ to filter all values less than $x_t - m$. Also, p'_t can be written as

$$p'_t = \frac{e^{x_t}}{\sum\limits_{x_k \geq x_t - m} e^{x_k}} = \frac{e^{x_t}/e^{x_t-m}}{\sum\limits_{x_k \geq x_t - m} e^{x_k}/e^{x_t-m}} = \frac{e^m}{\sum\limits_{x_k \geq x_t - m} e^{x_k-(x_t-m)}},$$

where all the intermediate values are limited to $\left[e^0, e^m\right]$, which enables us to use a small bit length to evaluate the secure softmax with Yao's Garbled Circuits. For example, for $l = 10^{-3}$ and $K = 100$, we have $m \geq ln(10^{-3} * 100 - 100) \approx 11.52$, and e^{12} takes only 18 bits, while the original x_t may reach up to > 100 [8] and e^{100} takes 145 bits. Based on the above analysis, the outline protocol of our proposed secure softmax is as follows:

1. Let $[\mathcal{F}(\mathbf{x})]$, where $\mathbf{x} = \{x_1, x_2, \cdots, x_K\}$ be the input to the softmax layer. Server *S* masks it with a random vector $\mathbf{r}$, and sends $[\mathcal{F}(\mathbf{x} - \mathbf{r})]$ to client *C*. Then *S* sets its share to $\mathbf{x}^S = \mathbf{r} = \{r_1, r_2, \cdots, r_K\} \bmod p$.

2. The client *C* decrypts $[\mathcal{F}(\mathbf{x} - \mathbf{r})]$ and performs the inverse FFT to obtain $(\mathbf{x} - \mathbf{r})$. Then *C* sets its share to $\mathbf{x}^C = \mathbf{x} - \mathbf{r} = \{x_1 - r_1, x_2 - r_2, \cdots, x_K - r_K\} \bmod p$.

3. Now *C* and *S* interact with each other to find the maximum value x_t and decide which x_i can be ignored according to the selected integer m, and set the ignored one and the left x_i to 0 and $m - (x_t - x_i)$, respectively. To be noted, the plaintext modulo is converted to $(m + 1)$, and there no longer exist negative values. At the end of this procedure, *C* and *S* hold newly generated shares, $\mathbf{x}^S = \{r'_1, r'_2, \cdots, r'_K\} \bmod (m+1)$ and $\mathbf{x}^C = \{x'_1 - r'_1, x'_2 - r'_2, \cdots, x'_K - r'_K\} \bmod m + 1$, where x'_i is 0 or $m - (x_t - x_i)$ and r'_i is randomly generated by *S*.

4. Next, to calculate the denominator of p'_t, client *C* and server *S* first calculate
$$e^{\mathbf{x}^C} = \{e^{x'_1 - r'_1}, e^{x'_2 - r'_2}, \cdots, e^{x'_K - r'_K}\} \quad \text{and}$$
$$e^{\mathbf{x}^S} = \{e^{r'_1}, e^{r'_2}, \cdots, e^{r'_K}\}.$$
Then, they use Yao's Garbled Circuits to calculate the denominator of p'_t. The boolean circuits used here for Yao's Garbled Circuits can be simply implemented with *ADDGate*, *MULGate* and *MUXGate*, and we ignore the details here. The $\mathbf{x}^C$ and $\mathbf{x}^S$ are used to guarantee that every $e^{(x'_i - r'_i \bmod m+1) + (r'_i \bmod m+1)}$ does not exceed e^{m+1} and decide whether to drop it. The final calculation result, i.e., the denominator of p'_t, will be obtained by *C*.

At this point, since the numerator e^m is public and *C* has the denominator of p_t, *C* is able to calculate p'_t. Note that, the potential leakage resulting from p'_t is out of scope of this paper. Actually, it is a general problem of neural networks [27]. Since the numerator of p'_t, i.e. e^m, is public and contains no information about p'_t, the possible information that could infer from the denominator of p'_t is equal to p'_t.

Security analysis. The possible privacy leakages are from $x_1, x_2, \cdots, x_K$ as they are all that used by the client to calculate the final result. In our design, $x_i, i \in [1, K]$ remains shared throughout the calculation, which means the client can only obtain $x_i - r_i$. Since r_i is a random number generated and kept by the server, x_i is protected from the client.

5. Performance Evaluation

We implemented FALCON in C++. For fast Fourier Transform (FFT), we used FFTW library [13]. For additively homomorphic encryption, we used the Fan-Vercauteren (FV) scheme [5]. For secure two-party computing, we used Yao's Garbled Circuits implemented by the ABY framework [10]. Specifically, the number of slots n, the plaintext module p, and the ciphertext module q are needed for the initialization of the FV scheme. We chose $n = 2048$, which means we can process up to 2048 elements in parallel, and the ciphertext module q was set to 1152921504382476289. The plaintext module p was set to 1316638721, which has 30-bit length and is enough for all the intermediate values.

We tested FALCON on two computers, both of which are equipped with Intel i5-7500 CPU with 4 3.40 GHz cores and 8GB memory, and have Ubuntu 16.04 installed. We let one be the client *C* and the other play as the server *S*. We took experiments in the LAN setting similar to previous work [21, 18]. Each experiment was repeated for 100 times and we report the mean in this paper.

5.1. Benchmarks for Layers

Here we introduce the performance of FALCON on individual layers. Since GAZELLE is the best known related work, we compare FALCON with it in all layers except the softmax layer, which is not implemented by GAZELLE. For benchmarking, all input data to each layers are randomly sampled from $[0, p)$. Parameters of Conv and FC layers are chosen from the CIFAR-10 model stated in Section 5.2.

We present the benchmarks for Conv and FC layers in different input sizes. As shown in Table 1, for Conv layers, we show the online running time with input $(w \times h \times c)$ and filter $(f_w \times f_h \times c, k)$ using different frameworks. For fully-connected layers, we report the running time with the input vector of length l_i and the output vector of length l_o. Note that, the setup phases involve performing FFT on filters and encrypting random values for masking, and the online phases take only the server's computation into account. As one can see from Table 1, FALCON outperforms GAZELLE in both setup and online phases. Especially for

online phases, our efficient Conv and FC implementations offer us over $10\times$ less runtime.

Table 1. Benchmarks and Comparisons for Conv and FC.

Layer	Input	Filter/Output	Framework	**Time** (ms)	
				setup	online
Conv Layer	$(28 \times 28 \times 1)$	$(5 \times 5 \times 1, 5)$	GAZELLE	11.4	9.2
			FALCON	3.1	0.25
	$(16 \times 16 \times 128)$	$(3 \times 3 \times 128, 128)$	GAZELLE	3312	704
			FALCON	615	51.2
FC Layer	2048	1	GAZELLE	16.2	8.0
			FALCON	1.2	0.1
	1024	16	GAZELLE	21.8	7.8
			FALCON	9.6	0.8

In Table 2, we report the running time and communication overhead of setup and online phases for ReLU and Max Pooling layers. Comparing to data preprocessing, the online communication overhead of ReLU and Max Pooling operations is almost negligible. We can also see that the optimized Max Pooling and ReLU operations have reduced the computation and communication overhead in all phases. Therefore, in ReLU and MaxPooling layers, FALCON which uses the optimized version outperforms GAZELLE which uses the original version.

Table 2. Benchmarks for ReLU and Max Pooling.

Operation	**Number of Inputs**	**Time** (ms)		**Comm** (MB)	
		setup	online	setup	online
Data Preprocessing	1000	32.3	14.5	4.82	1.45
	10000	265.5	136.4	48.2	14.9
ReLU	1000	9.82	4.20	1.95	0.01
	10000	96.2	43.2	19.2	0.11
MaxPooling	1000	12.1	5.6	1.94	0.01
	10000	100	45.5	20.0	0.12
ReLU+MaxPooling	1000	21.9	9.8	3.89	0.02
	10000	196.3	88.7	39.2	0.23
Optimized MaxPooling+ReLU	1000	12.5	5.2	2.44	0.02
	10000	134.3	54.4	24.4	0.14

We tested the performance of our proposed protocol for softmax function in different settings. As shown in Table 3, both runtime and communication overhead of setup and online phases grow with the precision l and the number of classes K. These overhead is relatively small compared with other layers in FALCON.

Table 3. Benchmarks for the Softmax.

Precision	**Classes**	**Time** (ms)		**Comm** (MB)	
		setup	online	setup	online
10^{-2}	10	8.56	3.89	0.996	0.0294
	100	58.7	24.5	9.96	0.294
	1000	574.8	254.6	99.6	29.4
10^{-4}	10	8.66	4.02	0.996	0.0294
	100	60.3	26.7	9.96	0.294
	1000	588.0	257.6	99.6	29.4

5.2. Evaluations on Real Models

We evaluated the performance of FALCON on two datasets, MNIST and CIFAR-10. CNN models for them are both from [21]. The CNN model for MNIST takes a gray scale image with size 28×28 as input and has 2 Conv, 2 FC, 3 ReLU and 2 Max Pooling layers; The CNN model for CIFAR-10 takes a three channel image of size $32\times32\times3$ as input and has 7 Conv, 1 FC, 7 ReLU and 2 Mean Pooling layers. We show runtime cost and communication overhead in both setup and online phases. To be noted, for the fairness of comparison, softmax layer is excluded in both models.

Table 4. Performance Comparison on MNIST and CIFAR-10.

CNN	**Framework**	**Time** (s)			**Comm** (MB)		
		setup	online	total	setup	online	total
MNIST	MiniONN	3.58	5.74	9.32	20.9	636.6	657.5
	GAZELLE	1.09	0.28	1.37	40.5	21.6	62.1
	FALCON	0.64	0.18	0.82	40.5	21.6	62.1
CIFAR-10	MiniONN	472	72	544	3046	6226	9272
	GAZELLE	15.5	4.25	19.8	906	372	1278
	FALCON	10.5	3.31	13.8	906	372	1278

Since efficient and secure implementation for Conv and FC layers are the main advantage in FALCON, in order to highlight them, we replace implementations for ReLU and Max Pooling in GAZELLE with the optimized version. The results are shown in Table 4. When evaluating the online overhead on both models, FALCON is running over $30\times$ faster than MiniONN while reducing communication overhead by over 97%. The significant improvement in running time is due to the repeatedly use of FFT and lattice-based homomorphic encryption, which saves many multiplications over ciphertexts.

5.3. Prediction Accuracy on Real Models

Since we treat decimal numbers as integers by proper scaling, there might have accuracy concerns on the encrypted models. However, experimental results in Fig. 5 show that the loss of accuracy in FALCON is negligible, and different schemes achieve nearly the same results.

Table 5. Prediction Accuracy on MNIST and CIFAR-10.

	Plaintext	MiniONN	GAZELLE	FALCON
MNIST	99.31%	99.0%	99.0	99.26%
CIFAR-10	81.61%	81.61 %	81.60	81.61%

6. Conclusion

In this paper, we presented a fast and secure evaluation approach for CNN predictions. For linear layers including Conv and FC, our FFt-based scheme achieves a low latency performance. For non-linear layers including ReLU and Max Pooling, we provided a detailed implementation for our optimized processing pipeline. For the softmax layer that has not been studied in previous work, we introduced the first efficient and privacy-preserving protocol.

Acknowledgements This work is supported in part by the National Natural Science Foundation of China under Grant No. 61972371 and Youth Innovation Promotion Association of the Chinese Academy of Sciences (CAS) under Grant No. 2016394. K. Xue is the corresponding author of this paper.

References

[1] Execution order of relu and max-pooling. https://github.com/tensorflow/tensorflow/issues/3180. Accessed Nov. 11, 2019.

[2] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 23th ACM Conference on Computer and Communications Security (CCS'16)*, pages 308–318. ACM, 2016.

[3] Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, Li Deng, Gerald Penn, and Dong Yu. Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 22(10):1533–1545, 2014.

[4] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In *Proceedings of the 33rd International Conference on Machine Learning (ICML'16)*, pages 173–182, 2016.

[5] Jean-Claude Bajard, Julien Eynard, M Anwar Hasan, and Vincent Zucca. A full rns variant of fv like somewhat homomorphic encryption schemes. In *Proceedings of the 23rd International Conference on Selected Areas in Cryptography (SAC'16)*, pages 423–442. Springer, 2016.

[6] Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In *Proceedings of the 13th European Symposium on Research in Computer Security (ESORICS'08)*, pages 192–206. Springer, 2008.

[7] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory*, 6(3):13:1–13:36, 2014.

[8] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *Proceedings of the 38th IEEE Symposium on Security and Privacy (SP'17)*, pages 39–57, 2017.

[9] Google Cloud. Vision api - image content analysis. https://cloud.google.com/vision/, 2018. Accessed Nov. 11, 2019.

[10] Daniel Demmler, Thomas Schneider, and Michael Zohner. Aby-a framework for efficient mixed-protocol secure two-party computation. In *Proceedings of the 22nd Annual Network and Distributed System Security Symposium(NDSS'15)*, 2015.

[11] Andre Esteva, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115–118, 2017.

[12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.

[13] FFTW. Fast fourier transform. http://www.fftw.org, 2018. Accessed Nov. 11, 2019.

[14] Craig Gentry, Shai Halevi, and Nigel P Smart. Fully homomorphic encryption with polylog overhead. In *Proceedings of the 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT'12)*, pages 465–482. Springer, 2012.

[15] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *Proceedings of the 33rd International Conference on Machine Learning (ICML'16)*, pages 201–210, 2016.

[16] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC'87)*, pages 218–229. ACM, 1987.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the 29th IEEE Conference on Computer Vision and Pattern Recognition (CVPR'16)*, pages 770–778, 2016.

[18] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In *Proceedings of the 27th USENIX Security Symposium (USENIX Security'18)*, 2018.

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS'12)*, pages 1097–1105, 2012.

[20] Shaohua Li, Kaiping Xue, Chenkai Ding, Xindi Gao, David SL Wei, Tao Wan, and Feng Wu. FALCON: A fourier transform based approach for fast and secure convolutional neural network predictions. *arXiv preprint arXiv:1811.08257*, 2018.

[21] Jian Liu, Mika Juuti, Yao Lu, and N Asokan. Oblivious neural network predictions via MiniONN transformations. In *Proceedings of the 24th ACM Conference on Computer and Communications Security (CCS'17)*, pages 619–631. ACM, 2017.

[22] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition(CVPR'15)*, pages 3431–3440, 2015.

[23] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *Proceedings of the 38th IEEE Symposium on Security and Privacy (SP'17)*, pages 19–38. IEEE, 2017.

[24] M Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M Songhori, Thomas Schneider, and Farinaz Koushanfar. Chameleon: A hybrid secure computation framework for machine learning applications. *arXiv preprint arXiv:1801.03239*, 2018.

[25] Bita Darvish Rouhani, M Sadegh Riazi, and Farinaz Koushanfar. Deepsecure: Scalable provably-secure deep learning. *arXiv preprint arXiv:1705.08963*, 2017.

[26] Ahmed Salem, Yang Zhang, Mathias Humbert, Mario Fritz, and Michael Backes. Ml-leaks: Model and data independent membership inference attacks and defenses on machine learning models. *arXiv preprint arXiv:1806.01246*, 2018.

[27] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K Reiter. Accessorize to a crime: Real and stealthy

attacks on state-of-the-art face recognition. In *Proceedings of the 23rd ACM Conference on Computer and Communications Security (CCS'16)*, pages 1528–1540. ACM, 2016.

[28] Dinggang Shen, Guorong Wu, and Heung-Il Suk. Deep learning in medical image analysis. *Annual Review of Biomedical Engineering*, 19:221–248, 2017.

[29] Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM Conference on Computer and Communications Security (CCS'15)*, pages 1310–1321. ACM, 2015.

[30] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *Proceedings of the 38th IEEE Symposium on Security and Privacy (SP'17)*, pages 3–18, 2017.

[31] Congzheng Song, Thomas Ristenpart, and Vitaly Shmatikov. Machine learning models that remember too much. In *Proceedings of the 24th ACM Conference on Computer and Communications Security (CCS'17)*, pages 587–601. ACM, 2017.

[32] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction apis. In *Proceedings of the 25th USENIX Security Symposium (USENIX Security'16)*, pages 601–618, 2016.

[33] Shmuel Winograd. On computing the discrete fourier transform. *Mathematics of computation*, 32(141):175–199, 1978.

[34] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science (FOCS'86)*, pages 162–167. IEEE, 1986.