Efficient Sum-Check Protocol for Convolution

CHANYANG JU1, HYEONBUM LEE1, HEEWON CHUNG1, JAE HONG SEO1, AND SUNGWOOK KIM2

1Department of Mathematics & Research Institute for Natural Sciences, Hanyang University, Seoul 04763, Republic of Korea (e-mail: {chanyangju, leehb3706, heewonchung, jaehongseo}@hanyang.ac.kr)
2Department of Information Security, Seoul Women’s University, Seoul 01797, Republic of Korea (e-mail: kim.sungwook@swu.ac.kr)

Corresponding author: Sungwook Kim (e-mail: kim.sungwook@swu.ac.kr).

This work was partly supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.20210007270012002, A Study on Cryptographic Primitives for SNARK, 50%) and the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. 2020R1C1C1A0100696812, 50%). The work of S. Kim was partly supported by a research grant funded by Seoul Women’s University (2020-0454).

ABSTRACT Many applications have recently adopted machine learning and deep learning techniques. Convolutional neural networks (CNNs) are made up of sequential operations including activation, pooling, convolution, and fully connected layer, and their computation cost is enormous, with convolution and fully connected layer dominating. In general, a user with insufficient computer capacity delegated certain tasks to a server with sufficient computing power, and the user may want to verify that the outputs are truly machine learning model predictions. In this paper, we are interested in verifying that the delegation of CNNs, one of the deep learning models for image recognition and classification, is correct. Specifically, we focus on the verifiable computation of matrix multiplications in a CNN convolutional layer. We use Thaler’s idea (CRYPTO 2013) for validating matrix multiplication operations and present a predicate function based on the insight that the sequence of operations can be viewed as sequential matrix multiplication. Furthermore, we lower the cost of proving by splitting a convolution operation into two halves. As a result, we can provide an efficient sum-check protocol for a convolution operation that, like the state-of-the-art zkCNN (ePrint 2021) approach, achieves asymptotically optimal proving cost. The suggested protocol is about 2× cheaper than zkCNN in terms of communication costs. We also propose a verified inference system based on our method as the fundamental building component.

INDEX TERMS verifiable computation, matrix multiplication, convolutional neural networks, interactive proofs, sum-check protocol

I. INTRODUCTION

Machine learning enables computers to improve themselves through experience with data. A convolutional neural network (CNN) is a machine learning technique that is particularly useful for recognizing and classifying the image. In recent years, there has been massive progress in efficiency, and this has led to many practical applications of machine learning [1]–[3]. However, one may concern that the malicious service provider manipulates the outputs; hence, the service clients want to ensure that some models’ results are true predictions. Thus, we can raise the following natural and meaningful question:

How can a user ensure the results which are indeed predictions models efficiently?

The naïve approach is that a user computes a model directly and compares the outputs. However, the machine learning model could be too complicated and a burden for the user with limited computational resources.

The verifiable computation, which is formalized by Gennaro, Gentry, and Parno [4] is a powerful building block to resolve this issue. It enables outsourcing some computations to an untrusted server while maintaining computation results. After the server computes some public function, it sends back the output together with a proof; a user can verify the result efficiently with consuming less computational cost for verification than the direct computation. Due to its strength, this concept as known as a verifiable delegation of computation is widely employed in various applications, which want to be guaranteed the results [4]–[6].
SafetyNet [7] and zkCNN [8] employ a verifiable computation technique to give guarantee the results of the deep learning inference models, especially verifying CNN. As is well-known, CNN has a several layers, and each layer consists of multiple operations such as convolution operation, activation, and pooling; however, among these, a convolution operation dominates all other operations in terms of verifiable computation. Since a convolution operation can be expressed into matrix multiplication, they focus on reducing the computational cost for matrix multiplication. To achieve it, SafetyNet uses a variant of GKR protocol [9] that is specialized for matrix multiplication, and they can reduce the computational cost from $O(n^3)$ to $O(n^2)$ where $n$ is the dimension of the square matrix; however, unnecessary large $n$ is required because 3-dimensional data is appropriate for the CNN, and thus $n$ becomes more significant than the original dimension of the convolution operation. zkCNN proposes a new sum-check protocol for the convolution operation using Fast Fourier Transform (FFT). Even though zkCNN accomplishes an asymptotically optimal proving cost, it requires three sum-check protocols proportional to the communication cost.

A. OUR CONTRIBUTION

In this paper, we propose a new efficient sum-check protocol for a CNN convolution operation. Inspired by [9], we introduce a function for a matrix that takes binary vectors representing the index and outputs a corresponding element. When a convolution process is translated into 2-dimensional matrices, CNN is impracticable since it is specified as a form of matrix multiplication that takes a cubic time in the matrix dimension naively. Thus, we interpret the input and intermediate values as 3-dimensional matrices, which is a more intuitive approach. However, a full binary adder should be used to compute a convolution operation in this setting and it does not support linearity. To overcome this barrier, we introduce a predicate function that allows a convolution operation to be linearized in each binary variable, which leads to saving the proving cost than naïve approach. Moreover, we can reduce further by well understanding our linearized convolution operation. As a result, we can achieve an asymptotically optimal proving cost for a convolution operation. Moreover, our scheme employs the sum-check protocol [10] and GKR protocol [6], thus does not require any trusted setup and cryptographic assumption. Additionally, we present a new verifiable CNN by applying our construction.

We rigorously analyze the efficiency of the proposed sum-check protocol for a convolution layer in CNN. Our analysis shows that our approach also reaches the optimal computation cost for the prover as in that of zkCNN, which is substantially lower than that of SafetyNet. The computation cost for the verifier of our approach is much smaller than that of SafetyNet and larger than that of zkCNN. However, because zkCNN does not consider batch operations, we expect that our approach can be more efficient than that of zkCNN for large batch sizes. Finally, the proof size of our approach is slightly larger than that of SafetyNet and approximately $2 \times$ smaller than that of zkCNN.

B. RELATED WORK

Verifiable Computation. In the verifiable delegation of computation, there are two participants a prover and a verifier. The prover provides a computation result and a proof claiming the result is correct, and the verifier wants to check the proof efficiently so that convince if the result is correct or not [4]–[6]. Goldwasser et al. [6] proposed an interactive proof protocol for a layered arithmetic circuit, called GKR protocol, that makes the verifier can be sure of the computational integrity with much less computational cost than the cost of performing the computation by itself. Several optimization works of GKR protocol are provided, including, Cormode et al. [11] and Vu et al. [12] with practical implementations, and assuming structured circuits [9], [13]. In [9], Thaler proposed a highly efficient matrix multiplication sum-check protocol with linear prover time, which enables the matrix multiplication task to be delegated.

zkSNARK. Gennaro et al. [14] proposed a novel zero-knowledge succinct non-interactive argument system (zk-SNARK) by introducing an efficient encoding method called quadratic arithmetic programming (QAP). The following works, [15]–[18] are proposed as improvements of QAP based zkSNARK. Though these methods yield efficient verification and proof sizes that are suitable for practical applications, they inherently require a trusted setup to generate the structured reference string. The trusted setup issue is addressed in [19]–[21]. There are alternatives techniques [20], [22]–[24] to construct zk-SNARK using polynomial commitment schemes [25], [26]. A prover provides the commitment of a polynomial and then proves the value of an evaluation of the polynomial at an arbitrary point chosen by the verifier in a polynomial commitment scheme. And its security and assumptions rely on the underlying polynomial commitment schemes.

Verifiable Matrix Multiplication for CNN. SafetyNet [7] is a verifiable computation of neural network inference using an interactive proof protocol. They express each operation in the deep neural network as the form of matrix multiplication and use a variant of GKR protocol that is specialized for matrix multiplication [9]. This approach reduces the verifier’s computational costs for matrix multiplication from $O(n^3)$ to $O(n^2)$ assuming the two square matrices. However, the 3-dimensional data must be modified to fit with the 2-dimensional matrix representation of operation when convolution operation, in particular, exacerbates performance loss since one side of the matrix grows greater than the original. We will describe more about this in Section III-A. It is worth noting that the gap introduces a slew of redundant procedures and increases the protocol’s overall complexity.

ZEN [27] is a compiler that provides a verifiable inference model for neural network models while preserving the privacy of input data. The authors devise the compiler schemes
to efficiently apply RICS-based zk-SNARKs to the neural network model. However, they must presume that before execution, the prover and the verifier agree on a neural network model, including the network architecture, hyperparameters, and the type of zk-SNARKs they will employ (since the structure of CRS is dependent on the type of zk-SNARKs). This means that every change in the network necessitates re-starting the entire setup process. Note that the number of multiplication operations makes not only the setup time and CRS size bigger but also the prover computational cost worse in most RICS-based zk-SNARKs. Because the neural network has a large number of multiplication operations, this poses issues depending on the use case.

Mystique [28] is a ZK system for large-scale neural-network inference. They construct an improved ZK protocol for matrix multiplication using Quicksilver [29] that is a ZK proofs based subfield vector oblivious linear evaluation (s-OLE). It requires less memory size than other zk-SNARKs, but it requires larger proof size than other zk-SNARKs. More concretely, for matrix multiplication, Quicksilver demands about \(100\) proof size than Virgo [30] which is a zk-SNARK system for large-scale neural networks. We denote the feature map as an input and outputs a corresponding element. For example, \(A \times x\) is a matrix multiplication of \(x\) and \(A\), which can achieve an asymptotically optimal proving cost \(O(n \log n \cdot m)\) for a vector of size \(n\) and \(m\) kernel, which can achieve an asymptotically optimal proving cost \(O(n^2 + w^2)\) using it as a building block, the total proving cost for the CNNs becomes \(O((c_1 + c_2)n^2 + c_1 c_2 w^2)\), where \(c_1\) and \(c_2\) denotes the number of input and output channels, respectively. However, they require three sum-check protocols for FFT, Hadamard product, and inverse FFT. Since the proof size is linear in the number of sum-check protocols and our scheme only needs two sum-check protocols, our proof size is less than zkCNN.

C. ORGANIZATION

The remainder of this paper is organized as follows. Section II provides some necessary background for the rest of the paper. In Section III, we present a main building block protocol for verifiable convolution operation. We then present how our technique for the convolution operation can be applied to a verifiable CNN in Section IV. Finally, we provide some concluding remarks in Section V.

II. BACKGROUNDS

Notations. We use a notation \( \mathbb{R} \) to denote a set of real numbers with fixed precision. For a security parameter, \( \lambda \) and a prime \( p \) of length \( \lambda \), \( \mathbb{Z}_p \) denotes a field of integers modulo \( p \), and \( \mathbb{Z}_p^* \) denotes its multiplicative group. Let \( \mathbb{Z}_p[X_1, \ldots, X_n] \) be a set of \( n \)-variate polynomials on \( \mathbb{Z}_p \). We use two constant vectors \((1, \ldots, 1), (2^{i-1}, \ldots, 2, 1) \in \mathbb{Z}_p^i\), denoted by \(1^i\) and \(2^i\), respectively. For two integers \( j, k \) with \( j < k \), we use a subscript \([j : k]\) to represent a vector whose components are consecutive integers, i.e., \( x[j : k] := (x_j, x_{j+1}, \ldots, x_k) \). \( F_i \in \mathbb{R}^{n_i \times n_i \times c_i \times b} \) represents a feature map, which is the input of \( i \)-th operation in CNNs, where \( n_i \) is the number of components in width and height, \( c_i \) is the number of channels at \( F_i \), and \( b \) is the batch size. For the simplicity, we let \( N_i = n_i \cdot c_i \cdot c_i \). For a matrix \( A \in \mathbb{Z}_p^{n \times m} \), we denote \( A_{x,y} \) to represent \((x, y)\)-th element in \( A \) and interchangeably use it as a function mapping \( A : \{0, 1\}^{\log n} \times \{0, 1\}^{\log m} \rightarrow \mathbb{Z}_p \) that takes binary vectors representing indices of \( A \) as inputs and returns a corresponding element. For example, \( A(x, y) = A_{x,y} \) where \( x \in \{0, 1\}^{\log n} \) and \( y \in \{0, 1\}^{\log m} \) are binary vectors of \( x \) and \( y \) with the order of the higher bit first. We remark that function mapping always exists for any matrix. Thus, we overwrite the function notion for any given matrix without redefining it. This can be extended to any high dimensional matrices. We denote the feature map space and weight space by \( F \) and \( W \), respectively. We define convolution operation as \( C : F \times W \rightarrow F \), activation as \( A : F \rightarrow F \), pooling as \( P : F \rightarrow F \), and the fully connected operation as \( FC : F \rightarrow F \).
Suppose that the ℓ-th layer is a convolution layer having \( n_ℓ \times n_ℓ \times c_ℓ \) output neurons storing a weight matrix 
\( W_{ℓ-1} \in \mathbb{R}^{d_{ℓ-1} \times d_{ℓ-1} \times c_{ℓ-1} \times c_ℓ} \) and a bias \( B_{ℓ-1} \in \mathbb{R}^{n_ℓ \times n_ℓ \times c_ℓ} \).
It takes \( X_{ℓ-1} \in \mathbb{R}^{n_ℓ \times n_ℓ \times c_{ℓ-1} \times b} \) as an input and outputs
\[
σ_ℓ(X_{ℓ-1} \ast W_{ℓ-1} + B_{ℓ-1} \cdot 1^{bT}) \in \mathbb{R}^{n_ℓ \times n_ℓ \times c_ℓ \times b},
\]
where the (2-dimensional) convolution \( U = W \ast X \in \mathbb{R}^{n_ℓ \times n_ℓ \times c_ℓ \times b} \) is defined as
\[
U_{α,β,γ,τ} = \sum_{r=0, s=0, t=0}^{d_{ℓ-1}} X_{α+r, β+s, γ+t, τ} \cdot W_{r, s, t, γ}.
\]
Note that we consider only the case of stride 1 with no padding for the sake of simplicity, and thus \( n_ℓ = n_{ℓ-1} - d_{ℓ-1} + 1 \).

3) Embedding Real Numbers into a Finite Field
Although the fact that neural networks can conduct all real-number operations, proving the integrity of real-number operations, even with state-of-the-art approaches, is difficult. Instead, we quantize each real number and embed it into a finite field with a large characteristic.

**B. INTERACTIVE ORACLE PROOFS**

1) Sum-Check Protocol
The sum-check protocol enables solving the sum-check problem, proving that evaluations of some \( n \)-variate polynomial over the Boolean hypercube agrees with a claimed value. More specifically, for \( P(X_1, \cdots, X_n) \in \mathbb{F}[X_1, \cdots, X_n] \) and \( y \in \mathbb{F} \), the sum-check problem is proving
\[
y = \sum_{x \in \{0,1\}^n} P(x) = \sum_{x_1 \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} P(x_1, x_2, \cdots, x_n).
\]
In general, one can verify it by evaluating all \( 2^n \) points, however, it causes inefficiency problem as \( n \) increases. The sum-check protocol enables to verify only \( O(n) \) polynomial evaluations.

In the sum-check protocol, both parties \( P \) and \( V \) share a multivariate polynomial \( P(X_1, \ldots, X_n) \in \mathbb{F}[X_1, \ldots, X_n] \) and an output \( y \) claimed by \( P \). At first, \( P \) constructs a univariate polynomial
\[
P_1(X) = \sum_{x_1 \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} P(x_1, x_2, \cdots, x_n)
\]
and sends \( P_1(X) \) to \( V \). After receiving \( P_1(X) \), \( V \) checks whether \( y = P_1(0) + P_1(1) \) and if it holds, sends a random element \( r_1 \in \mathbb{F} \). Similarly, \( P \) constructs a univariate polynomial
\[
P_2(X) = \sum_{x_1 \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} P(r_1, X, x_2, \cdots, x_n)
\]
and sends \( P_2(X) \) to \( V \). After receiving \( P_2(X) \), \( V \) checks whether \( P_1(r_1) = P_2(0) + P_2(1) \) and sends a random number \( r_2 \in \mathbb{F} \) to \( P \) and repeat this procedure until the end of the protocol. More specifically, for each round \( 2 \leq i \leq n \), \( P \) sends a univariate polynomial
\[
P_i(X) = \sum_{x_i+1 \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} P(r_1, \cdots, r_{i-1}, X, x_{i+1}, \cdots, x_n)
\]
and \( V \) checks whether \( P_i(r_i) = P_{i+1}(0) + P_{i+1}(1) \). In the final round, \( V \) additionally checks whether
\[
P(r_1, r_2, \cdots, r_n) = P_n(r_n).
\]
If all tests are passed, \( V \) accepts, otherwise \( V \) rejects.

The sum-check protocol satisfies perfect correctness and soundness with error \( nd/|\mathbb{F}| \) where \( \deg(P) \) is at most \( d \). If \( P \) follows the whole protocol honestly, \( V \) accepts the protocol with probability \( 1 \). However, if \( P \) does not follow the protocol, \( V \) accepts the protocol with probability at most \( nd/|\mathbb{F}| \). The sum-check protocol consists of \( n \) rounds. Let \( \deg(P_i) \) be the degree of variable \( X_i \) in \( P \). At each round, \( P \) sends \( \deg(P_i) + 1 \) elements in \( \mathbb{F} \) and \( V \) sends one element in \( \mathbb{F} \) except the last round. Thus, the total communication cost comprises \( n + \sum_{i=1}^{n} \deg(P_i) \) field elements. At each round, \( P \) should calculate \( O(\deg(P_i) \cdot 2^{n-i}) \) evaluations to compute \( P_i(X) \), hence, \( O(\sum_{i=1}^{n} \deg(P_i) \cdot 2^{n-i}) \) evaluations required in total. At each round, \( V \) computes \( P_1(0), P_1(1), P_i(r_i), \) and additionally \( P(r_1, \ldots, r_n) \) for the last round, thus, the overall computation cost for \( V \) is \( O(n + \sum_{i=1}^{n} \deg(P_i)) \) evaluations of univariate polynomials \( P_i \)'s. The sum-check protocol is a public coin interactive protocol, and it can be converted non-interactive protocol by Fiat-Shamir transformation under random oracle model [31].

2) Multilinear Extensions
For an \( n \)-dimensional vector \( x \in \mathbb{F}^n \), we can construct a function \( f : \{0,1\}^{\log n} \to \mathbb{F} \) that takes a binary vector representing an index of \( x \) as an input and outputs corresponding component of \( x \). Multilinear extensions (MLE) allow us to transform from an index-component function \( f : \{0,1\}^{\log n} \to \mathbb{F} \) to multilinear polynomial \( \tilde{f} : \mathbb{F}^{\log n} \to \mathbb{F} \).

Define a function \( f : \{0,1\}^{\log n} \to \mathbb{F} \) from log \( n \)-dimensional hypercube to \( \mathbb{F} \). Then there is a unique MLE \( \tilde{f} : \mathbb{F}^{\log n} \to \mathbb{F} \) of \( f \) [11]. By uniqueness property, \( n \) distinct field elements can be viewed as the MLE polynomial \( \tilde{f} \) of \( f \) and vice versa. Using Lagrange interpolation, the MLE of \( f \) can be represented as
\[
\forall x \in \mathbb{F}^{\log n},
\tilde{f}(x) = \sum_{w \in \{0,1\}^{\log n}} f(w) \prod_{i=1}^{\log n} (x_i - w_i)(1 - x_i)(1 - w_i).
\]

3) Sum-Check Protocol for Matrix Multiplications
Thaler proposes a new efficient sum-check protocol for the matrix multiplications [9]. Let \( W, X \in \mathbb{F}^{n \times n} \). Even though

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3133442, IEEE Access
it costs $O(n^3)$ field operations to multiply $X \cdot W$, one can verify only $O(n^2)$ operation using the sum-check protocol due to Thaler [9]. To apply the sum-check protocol, an arithmetic circuit of a matrix multiplication must be transformed according to the sum-check protocol.

Without loss of generality, assume that $n$ is the power of 2. Let $W, X \in \mathbb{F}^{n \times n}$ be matrices. Then $Y = W \cdot X$ implies

$$Y(i, j) = \sum_{k \in \{0, 1\}^{\log n}} W(i, k) \cdot X(k, j),$$  

(3)

for all $(i, j) \in \{0, 1\}^{\log n} \times \{0, 1\}^{\log n}$. Let $\bar{W} : \mathbb{F}^{\log n} \times \mathbb{F}^{\log n}, \bar{X} : \mathbb{F}^{\log n} \times \mathbb{F}^{\log n}$ and $\bar{Y} : \mathbb{F}^{\log n} \times \mathbb{F}^{\log n}$ be a MLE of $W, X$ and $Y$, respectively. Since a MLE for a function is unique, we conclude that for $(i, j) \in \{0, 1\}^{\log n} \times \{0, 1\}^{\log n}$

$$\bar{Y}(i, j) = \sum_{k \in \{0, 1\}^{\log n}} \bar{W}(i, k) \cdot \bar{X}(k, j)$$  

(4)

is the MLE of (3). Instead of checking polynomial equivalence, it is enough to check the evaluation on random point $(r_L, r_R) \in \{0, 1\}^{\log n} \times \{0, 1\}^{\log n}$. Here, soundness error is bound on $\log n^2/|\mathbb{F}|$ by Schwartz-Zippel lemma [32], [33]. Put a random point $(r_L, r_R)$ on (4). Then

$$\bar{Y}(r_L, r_R) = \sum_{k \in \{0, 1\}^{\log n}} \bar{W}(r_L, k) \cdot \bar{X}(k, r_R).$$  

(5)

This is a form of the sum-check problem. The last step is applying sum-check protocol on (5).

III. AN EFFICIENT MATRIX REPRESENTATION FOR CNN

We remark that our goal is to construct an efficient proof system that proves correct evaluation for a given CNN. To achieve this goal, we propose a new sum-check protocol specializing in image data. In this section, we explain how our approach achieves the efficiency for each operation.

A. CONVOLUTION OPERATION

We consider zero padding with stride 1 for the purpose of simplicity. As mentioned in Section II-A, the convolution operation differs from the standard matrix multiplication. The convolution operation can be transformed into a matrix multiplication by reformulating the weights to a sparse matrix because it performs dot products between the weights and input feature maps. After looking into how it can be expressed as 2-dimensional matrix multiplication, we describe our approach and compare the efficiency of two different approaches. In Fig. 1, we illustrate the existing approach and our approach to a matrix multiplication for a convolution operation.

1) Naïve Approach

Consider a direct approach from SafetyNet [7] to fit in the proof system for 2-dimensional matrix multiplication. We write $N_i = n_i \cdot n_i \cdot c_i$. In fact, the operations in (1) and (2) are conducting a same operations called inner-product operation. Hence it can be reduced as the form of matrix multiplication. In the convolution operation in (2), a feature map $F \in \mathbb{F}^{p_i \times n_i \times c_i \times b}$ and a weight map $W \in \mathbb{F}^{d_i \times d_i \times c_i \times c_i+1}$ in $i$-th layer can be reconstructed to $2 \times 2$ matrix $F' \in \mathbb{F}^{N_i \times N_i}$ and $W' \in \mathbb{F}^{N_i \times N_i}$ as follows. For $0 \leq u, v < n_i, 0 < k < c_i, 0 \leq \ell < b, 0 \leq s, t < n_{i+1}, 0 < m < c_{i+1}$,

$$F'_{u, v + n_i \cdot k + n_i^2} := F_{u, v, k, \ell},$$

$$W'_{u + v + n_i \cdot k + n_i^2 + s + t \cdot n_{i+1} + m \cdot n_{i+1}^2} := \begin{cases} W_{u, v, k, m} & \text{if } 0 \leq u, v < d_i, 0 \leq k < c_i, \\ 0 & \text{otherwise.} \end{cases}$$

Let $U := C(F, W) \in \mathbb{F}^{N_i \times N_i \times c_i \times c_i+1 \times b}$ be the resulted feature map of convolution operation. Then we can express the convolution to matrix multiplication of $F'$ and $W'$ as

$$U'(\alpha, \beta) = \sum_{x \in \{0, 1\}^{\log N_i}} F'(\alpha, x) \cdot W'(x, \beta)$$  

(6)

for all $(\alpha, \beta) \in \{0, 1\}^{\log b} \times \{0, 1\}^{\log N_i}$. Note that, $U'$ is the reorganization of $U$. After that, we can directly apply the sum-check protocol for 2-dimensional matrix multiplication on

$$\tilde{U}'(\alpha, \beta) = \sum_{x \in \{0, 1\}^{\log N_i}} \tilde{F}'(\alpha, x) \cdot \tilde{W}'(x, \beta).$$  

(7)

(7) holds by the fact that (6) is multilinear in each binary variable. However, this transformation causes redundant multiplications because it increases the size of two side in $W$ from $d_i$ to $n_i$ and in the practical usages, the parameters tend to be $d_i \ll n_i$. Thus, the transformation results in significant performance degradation.

2) Our Approach

We use a method to eliminate the transformation in order to avoid the weight matrix blowing up in size. This is a straightforward technique, but it is not simple. The first issue is that the convolution operation cannot be represented in terms of each index as a linear polynomial. As a result, using the MLE-based sum-check protocol is difficult. The second point is that, while we can describe the convolution process linearly, this does not ensure the efficiency of the sum-check protocol, whose performance is largely determined by how efficiently the MLE is calculated. Finally, because we are working with a deep neural network, the formula should be recursively induced. This allows the sum-check procedure to be applied repeatedly to backward operations.

Our Linearized Convolution Operation. Let $F \in \mathbb{F}^{p_i \times n_i \times c_i \times b}$ and $W \in \mathbb{F}^{d_i \times d_i \times c_i \times c_i+1}$ be the feature map and weight map in $i$-th layer, respectively, and $U := C(F, W) \in \mathbb{F}^{N_i \times N_i \times c_i \times c_i+1 \times b}$ be the result of convolution operation in $i$-th layer, then we express a convolution operation as

$$U(\alpha, \beta, \gamma, \tau) = \sum_{a, b, c, \tau \in \mathbb{F}^{p_i \times n_i \times c_i \times b}} F(\alpha \oplus a, \beta \oplus b, c, \tau) \cdot W(a, b, c, \gamma)$$  

(8)
for \( \alpha, \beta \in \{0,1\}^{\log N_{i+1}} \), \( \tau \in \{0,1\}^{\log b} \), and \( \gamma \in \{0,1\}^{\log c_{i+1}} \), where \( \oplus \) denotes a binary full adder. However, a binary full adder does not support linearity. To express the convolution operation as a linear form in each binary variable that representing index, we introduce two additional functions \( B_d \) and \( J_d \) for some \( d \in \mathbb{Z} \) as follows:

- \( B_d : \{0,1\}^{m_1} \times \{0,1\}^{m_2} \rightarrow \mathbb{Z} \) is given by
  \[
  B_d(a, b) = (a, 2^{m_1}) \oplus (b, 2^{m_2}),
  \]
- \( J_d : \{0,1\}^{m_1} \times \{0,1\}^{m_2} \times \{0,1\}^{m_3} \rightarrow \{0,1\} \) is given by
  \[
  J_d(a, b, c) = \begin{cases} 
  1 & \text{if } B_d(a, b) = B_d(0, c) \\
  0 & \text{otherwise}
  \end{cases}
  \]

where \( m_1, m_2, m_3 \in \mathbb{Z} \), \( 2^i = (2^{i-1}, \ldots, 2, 1) \) for some \( i \in \mathbb{Z} \) and \( (,) \) denotes an inner-product operation. Thus, we get the following linear expression for the convolution operation for (8):

\[
U(\alpha, \beta, \gamma, \tau) = \sum_{a,b \in \{0,1\}^{\log d_{i+1}}} \sum_{p,q \in \{0,1\}^{\log n_i}} \{ F(p, q, c, \tau) \cdot J_0(\alpha, a, p) \cdot J_0(\beta, b, q) \cdot W(a, b, c, \gamma) \}
\]

for \( \tau \in \{0,1\}^{\log b} \) and \( \gamma \in \{0,1\}^{\log c_{i+1}} \).

**Reducing Cost for \( J_0 \).** We linearize a convolution operation to execute the sum-check protocol by introducing a new function \( J_0 \) and the prover must show that \( F, J_0, \) and \( W \) satisfies (9). Due to [9], the computational complexity of the sum-check protocol totally depends on the degree of the polynomial and then it takes approximately \( O(n_i^2 c_i b + n_i n_{i+1} d_i + c_i c_{i+1} d_i^2 + n_i^2 c_i d_i^2) \). Note that it takes roughly \( O(n_i^2 c_i b), O(n_i n_{i+1} d_i), \) and \( O(c_i c_{i+1} d_i^2) \) to construct multilinear extension of \( F, J_0, \) and \( W \), respectively, and it takes about \( O(n_i^2 c_i d_i^2) \) to execute the sum-check protocol with (9). Due to \( J_0 \), it is apart from asymptotically optimal proving cost, thus we need to present \( J_0 \) with less degree polynomial.

Let \( k = \log d_i \), \( \alpha = (\alpha_1, \alpha_2) \in \{0,1\}^{\log n_{i+1} - k} \times \{0,1\}^k \), and \( \beta = (\beta_1, \beta_2) \in \{0,1\}^{\log n_{i+1} - k} \times \{0,1\}^k \). By the construction, \( J_0(\alpha, a, p) \) and \( J_0(\beta, b, q) \) only survive if the following equations hold:

\[
\begin{align*}
  p &= \alpha \oplus a = (\alpha_1 \parallel 0^{k}) \oplus \alpha_2 \oplus a, \\
  q &= \beta \oplus b = (\beta_1 \parallel 0^{k}) \oplus \beta_2 \oplus b,
\end{align*}
\]

where \( 0^{k} = (0, \ldots, 0) \in \mathbb{Z}^k \) and \( \parallel \) denotes a concatenation. We then have

\[
\begin{align*}
  J_0(x, a, p) &= J_0(x_2, a, \eta) \cdot J_k(x_1, \eta, p), \\
  J_0(y, b, q) &= J_0(y_2, b, \zeta) \cdot J_k(y_1, \zeta, q)
\end{align*}
\]

for \( \eta, \zeta \in \{0,1\}^{k+1} \). Thus, (9) becomes

\[
\begin{align*}
U(\alpha, \beta, \gamma, \tau) &= \sum_{a,b \in \{0,1\}^{k}} \sum_{p,q \in \{0,1\}^{\log n_i}} \sum_{\eta,\zeta \in \{0,1\}^{k+1}} \{ F(p, q, c, \tau) \cdot J_0(\alpha, a, \eta) \cdot J_k(\alpha_1, \eta, p) \cdot J_0(\beta, b, \zeta) \cdot J_k(\beta_1, \zeta, q) \cdot W(a, b, c, \gamma) \}
\end{align*}
\]

for \( \tau \in \{0,1\}^{\log b} \) and \( \gamma \in \{0,1\}^{\log c_{i+1}} \).

**Reducing Cost for Sum-Check Protocol.** In the above paragraph, we reduce computational complexity for multilinear extension from \( O(n_i n_{i+1} d_i) \) to \( O(2n_i n_{i+1} + O(2d_i^2) \)

\[
\begin{align*}
\end{align*}
\]

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/
However, in (10), it takes about $O(4n^2c_i d_i^2)$ to execute the sum-check protocol and it is still far from the asymptotically optimal complexity.

We run the sum-check protocol in two steps to reduce the cost during the sum-check protocol. More specifically, we divide the Eq. (10) and apply a linear extension. Define

$$
\tilde{F}(\alpha_1, \eta, (\beta_1, \zeta, c, \tau) = \sum_{p, q, c \in \{0, 1\}^k} \tilde{F}(p, q, c, \tau) \cdot \tilde{J}_k(\alpha_1, \eta, p) \cdot \tilde{J}_k(\beta_1, \zeta, q).
$$

(11)

Then, (10) is translated to

$$
\tilde{U}(\alpha, \beta, \gamma, \tau) = \sum_{a, b, c \in \{0, 1\}^k} \left\{ \tilde{F}(\alpha_2, a, \eta) \cdot \tilde{J}_0(\beta_2, b, \zeta) \cdot \tilde{W}(\alpha, b, c, \gamma) \right\},
$$

(12)

The equality in the above two equations holds by the uniqueness of MLE. The prover claims that the matrix $U$ is the result of a valid convolution operation. To check whether $U$ is computed correctly, the verifier use (12) and (11) instead of (10). As we noted about MLE in Section II-B, checking whether the MLE $U$ has the right value at a random point that the verifier selects is enough to check the validity of the computation. Hence the prover and the verifier first run the sum-check protocol for (12). At the final round, the verifier must know the values $\tilde{F}(\alpha_1, p_0), (\beta_1, q_0), (\zeta, c_0), (\tau, J_0(\alpha_2, a_0, \eta), J_0(\beta_2, b_0, \zeta) and \tilde{W}(\gamma, a_0, b_0, c_0)$ where $a_0, b_0, c_0, p_0, q_0$ are randomly chosen during the sum-check protocol. The values of $J_0$ and $\tilde{W}$ are easily checked by verifier and the value of $\tilde{F}(\alpha_1, p_0), (\beta_1, q_0), (\zeta, c_0), (\tau)$ is again the claim by prover that must be proved. They continue to run the sum-check protocol for (11) to make the verifier convince that the $\tilde{F}$ is derived from $F$. Assuming that the $\tilde{F}$ is input, the verifier can finalize the continued sum-check protocol by computing $\tilde{F}(p, q, c, \tau)$ using the input. We present our approach for a sum-check protocol for a CNN convolution operation in Protocol 1.

**Theorem 1:** The interactive protocol presented above has soundness error $2^{log(n^2c_i d_i^2(d_i+1)^2)}$.

**Proof:** First, we consider the sum-check protocol of (12). In each round, the prover sends univariate polynomial degree at most 2. Hence, if there is only one round, the provability that a dishonest prover’s claim satisfies the verification is less than $2^{-\frac{\mu}{2}}$. Denote $\nu := log(c_i d_i^2(d_i+1)^2)$, $\mu := log n^2 c_i$ and let $f'_i$ be a dishonest prover’s polynomial that is sent to verifier in $i$-th round and $f_i$ be an honest one. And let E be the event that verifier accepts invalid statement. Then we have

$$
Pr[E] = Pr[E | f'_i = f'_i] + Pr[E | f'_i \neq f'_i] \\
\leq Pr[E | f'_i = f'_i] + \frac{2}{p}
$$

Therefore, the soundness error for (12) is $\frac{2p}{p}$. Similarly the soundness error for (11) is $\frac{2p}{p}$. By adding the two soundness errors, we get the upper bound of soundness error for Protocol 1.

3) Efficiency Analysis

In the below analysis, the unit operation is the field operation.

**Naïve Approach.** We analyze the efficiency of naïve approach first. We recall that during the protocol, the prover should substitute a random number from the verifier for the previous round’s variable and compute a univariate polynomial for the following round of sum-check. As a result, the proving cost in the sum-check protocol is proportional to the number and degree of variables. Therefore, the total proving cost for evaluating the sum-check protocol (7) is $O(n^2c_i n^2 i + n^2 c_i b + n^2 c_i + n^2 c_i b + n^2 c_i + n^2 c_i b)$. More concretely, $O(Nb = n^2 c_i b), O(Ni+1Ni = n^2 c_i n^2 i + 1 c_i + 1), and O(Ni+1 = n^2 c_i n^2 i + 1 c_i + 1)$ operations are required for evaluating MLE of a reconstructed feature map matrix $F'$ in $\mathbb{Z}_p^{Nxb}$, a reconstructed weight matrix $W' \in \mathbb{Z}_p^{N+1 \times N}$, and resulted feature map $U' \in \mathbb{Z}_p^{N+1 \times N}$, respectively. The prover then computes $O(n^2 c_i)$ field operations for constructing univariate polynomials in the sum-check protocol. Now we consider the computing cost of the verifier. As well as prover evaluates MLE of $W'$, the verifier should evaluate it and this evaluation requires $O(n^2 c_i n^2 i + 1 c_i + 1)$ operations. At the first layer, the verifier additionally computes $F'(\alpha, x)$, which costs $O(n^2 c_i b)$ operations.

In the case of the last layer, the verifier additionally computes $U'(\alpha, \beta)$ with $O(2 n^2 c_i b + 1)$ operations. And then verifier needs $O(log(n^2 c_i))$ for checking consistency of polynomial computed by the prover. Since this term is dominated by above terms $O(n^2 c_i n^2 i + 1 c_i + 1)$, we conclude that total verifier cost is $O(n^2 c_i n^2 i + 1 c_i + 1)$ for $i$-th intermediate layer, $O(n^2 c_i n^2 c_i + 1 c_i + 1)$ for the first layer, $O(n^2 c_i n^2 i + 1 c_i + 1 + n^2 i + 1 c_i + 1)$ for the last layer. The prover and the verifier run Sum-check protocol for equation (7) and its target polynomial has degree 2 for all log $n^2 c_i$ variables. Then the prover sends $3 log n^2 c_i$ field elements to the verifier and verifier sends log $n^2 c_i - 1$ random values to the prover.

**Our Approach.** Now we consider our approach. Our approach for convolution layer consists of two sum-check protocols to prove (11) and (12). First, to prove (12),
Protocol 1 \( \text{Conv}(pp, F \in \mathbb{Z}_p^{N_0 \times b}, W \in \mathbb{Z}_p^{d \times d \times c_0 \times c_1}, y \in \mathbb{Z}_p, \alpha, \beta \in \mathbb{Z}_p^{\log n_1}, \gamma \in \mathbb{Z}_p^{\log c_1}, \tau \in \mathbb{Z}_p^{\log b}) \)

Relation:

\[
y = \sum_{\alpha, \beta \in \{0,1\}^{k}} \sum_{c \in \{0,1\}^{c_0}} \left\{ \tilde{F}((\alpha_1, \eta_1), (\beta_1, \varepsilon_1), c, \tau) \cdot \tilde{J}_0(\alpha_2, a, \eta_2) \cdot \tilde{J}_0(\beta_2, b, \varepsilon_2) \cdot \tilde{W}(a, b, c, \gamma) \right\}
\]

\[
\tilde{F}((\alpha_1, \eta_1), (\beta_1, \varepsilon_1), c, \tau) = \sum_{p, q \in \{0,1\}^{\log n_0}} \tilde{F}(p, q, c, \tau) \cdot \tilde{J}_k(\alpha_1, p) \cdot \tilde{J}_k(\beta_1, q)
\]

where \( \log d = k, \alpha = (\alpha_1 \parallel \alpha_2) \) and \( \beta = (\beta_1 \parallel \beta_2) \) with \( \alpha_1, \beta_1 \in \mathbb{Z}_p^{\log n_1 - k}, \alpha_2, \beta_2 \in \mathbb{Z}_p^k \)

Common inputs: public parameter \( pp = (n_0, c_0, n_1, c_1, d, b) \), input feature maps \( F \), Weight matrix \( W \), evaluation \( y \) of MLE of output feature maps at random points \( (\alpha, \beta, \gamma, \tau) \)

Rounds indices for Sum-check protocols \( j := 4k + \log c_0 + 2 \) and \( l := 2 \log n_0 \)

Step 1 \( j \)-rounds Sum-check protocol for following equation

\[
y = \sum_{x \in \{0,1\}^j} f(x) := \sum_{x \in \{0,1\}^j} \tilde{F}(\alpha_1, x_1, x_2, (\beta_1, x_3), \tau) \cdot \tilde{J}_0(\alpha_2, x_4, x_5) \cdot \tilde{J}_0(\beta_2, x_6, x_7) \cdot \tilde{W}(x_1, x_2, x_3, \rho_3)
\]

1: \( P \) computes and sends \( f_0(1), f_1(1), f_i(2) \) to \( V \) where \( f_i(X) := \sum_{i=1}^{j-1} f_i(X, t_{[i,j]}) \)

2: \( V \) checks \( y = f_0(1) + f_1(1) \) and if it does not hold, abort it. And then \( V \) sends a random \( \delta_i \in \mathbb{Z}_p \) to \( P \)

3: for \( i = 2 \) to \( j \) do

4: \( P \) computes and sends \( f_0(0), f_1(1), f_i(2) \) to \( V \) where \( f_i(X) := \sum_{i=1}^{j-1} f_i(X, t_{[i,j]}) \)

5: \( V \) checks \( f_i(\delta_i) = f_0(0) + f_1(1) \) and it does not hold, abort it. And then \( V \) sends a random \( \delta_i \in \mathbb{Z}_p \) to \( P \)

6: \( P \) and \( V \) parse \( \delta_{[j]} = (\delta_1 \parallel \delta_2 \parallel \delta_3 \parallel \delta_4 \parallel \delta_5) \) where \( \delta_1, \delta_2, \delta_3, \delta_4, \delta_5 \in \mathbb{Z}_p^{\log c_0}, \delta_4, \delta_5 \in \mathbb{Z}_p^{\log b} \) and then \( V \) computes

\[
y_i := f_i(\delta_i) \cdot \tilde{J}_0(\alpha_2, x_4, x_5) \cdot \tilde{J}_0(\beta_2, x_6, x_7) \cdot \tilde{W}(\delta_1, \delta_2, \delta_3, \gamma)^{-1}
\]

Step 2 \( l \)-rounds Sum-check protocol for following equation

\[
y_l = \sum_{x \in \{0,1\}^l} g(x) := \sum_{x \in \{0,1\}^l} \tilde{F}(x_1, x_2, x_3, \delta_3, \tau) \cdot \tilde{J}_k(x_1, x_2, x_4, x_5) \cdot \tilde{J}_k(\beta_1, \delta_5, x_2)
\]

7: \( P \) computes and sends \( g_0(0), g_1(1), g_i(2) \) to \( V \) where \( g_i(X) := \sum_{i=1}^{j-1} g_i(X, t_{[i,j]}) \)

8: \( V \) checks \( y_l = g_0(0) + g_1(1) \) and if it does not hold, abort it. And then \( V \) sends a random \( \epsilon_i \in \mathbb{Z}_p \) to \( P \)

9: for \( i = 2 \) to \( l \) do

10: \( P \) computes and sends \( g_0(0), g_1(1), g_i(2) \) to \( V \) where \( g_i(X) := \sum_{i=1}^{j-1} g_i(X, t_{[i,j]}) \)

11: \( V \) checks \( g_i(\epsilon_i) = g_0(0) + g_1(1) \) and it does not hold, abort it. And then \( V \) sends a random \( \epsilon_i \in \mathbb{Z}_p \) to \( P \)

12: \( V \) checks \( g_i(\epsilon_i) = g(\epsilon_i) \) if all checks hold, \( V \) accept the protocol
TABLE 1. Comparison of Computation Complexity for Each Layer of CNN
For our approach and SafetyNet, both non-batch and batch cases are presented while zkCNN does not consider the batch case. In zkCNN, we do not include the prover and the verifier cost for polynomial commitment which is required for zero knowledge.

| Protocols  | Computation | Verifier |
|------------|-------------|----------|
| SafetyNet  |             |          |
| non-batch  | $O(n_i^2 c_i n_{i+1}^2 c_{i+1})$ | $O(n_i^2 c_i n_{i+1}^2 c_{i+1})$ |
| batch      | $O(n_i^2 c_i + n_{i+1}^2 c_{i+1} + n_i n_{i+1} + d_i^2 c_i + d_i^2)$ | $O(c_i n_{i+1}^2 c_{i+1} d_i^2)$ |
| zkCNN      |             |          |
| non-batch  | $O(n_i^2 c_i n_{i+1}^2 c_{i+1})$ | $O(n_i^2 c_i n_{i+1}^2 c_{i+1})$ |
| batch      | $O(n_i^2 c_i + n_{i+1}^2 c_{i+1} + n_i n_{i+1} + d_i^2 c_i + d_i^2)$ | $O(c_i n_{i+1}^2 c_{i+1} d_i^2)$ |

This work
| non-batch  | $O(n_i^2 c_i n_{i+1}^2 c_{i+1})$ | $O(n_i^2 c_i n_{i+1}^2 c_{i+1})$ |
| batch      | $O(n_i^2 c_i + n_{i+1}^2 c_{i+1} + n_i n_{i+1} + d_i^2 c_i + d_i^2)$ | $O(c_i n_{i+1}^2 c_{i+1} d_i^2)$ |

TABLE 2. Comparison of Proof Size for Each Layer of CNN
For our approach and SafetyNet, both non-batch and batch cases are presented while zkCNN does not consider the batch case. In zkCNN, we do not include communication cost for polynomial commitment which is required for zero knowledge.

| Protocols  | Proof Size |
|------------|------------|
| SafetyNet  |            |
| non-batch  | $6 \log n_i + 3 \log c_i$ |
| batch      |            |
| zkCNN      |            |
| non-batch  | $6(2 \log n_i + \log c_i + \log c_{i+1} + \log d_i)$ |
| batch      |            |
| This work  |            |
| non-batch  | $6 \log n_i + 3 \log c_i + 12 \log d_i + 6$ |

TABLE 3. Estimate of Proof Size for the First Convolution Operation of CNNs
We estimate the proof size in bytes. The base field is $\mathbb{F} = \mathbb{Z}_p$ for $p = 2^{64} - 1$ as considered in SafetyNet. Original convolutional layers of ResNet have a stride of 2, however, we consider a stride of 1 for convenience sake. For all networks, we only consider the non-batch case.

| CNNS | $n_0$ | $d_0$ | $c_0$ | $c_1$ | SafetyNet | zkCNN | This work |
|------|-------|-------|-------|------|-----------|-------|-----------|
| LeNet-5 | 32    | 5     | 1     | 6    | 229       | 732   | 549       |
| VGG  | 224   | 3     | 3     | 64   | 412       | 1,190 | 641       |
| ResNet | 224   | 7     | 3     | 64   | 412       | 1,236 | 732       |

Totally, the prover sends $3 \log(n_i^2 c_i d_i^4) + 6$ field elements and the verifier sends $\log n_i^2 d_i^4 c_i + 1$ random elements for the convolution sum-check protocol.

Efficiency Comparison. We summarize our analysis on efficiency compared with SafetyNet and zkCNN in Table 1 and Table 2. Table 1 shows that the computation cost for the prover in our approach is comparable with that of zkCNN, which is quite smaller than that of SafetyNet. The computation cost for the verifier in our approach is smaller than that of SafetyNet and larger than that of zkCNN. However, we should note that zkCNN does not explain how to deal with batch operations; thus, the computation cost for the verifier in zkCNN may increase in proportion to the batch size $b$.

Table 2 shows that the proof size of our approach is highly efficient. The proof size in our approach is comparable with that of SafetyNet and almost $2 \times$ smaller than that of zkCNN. We also note that the proof size of zkCNN may increase in proportion to the batch size $b$. We present the actual proof size for the existing CNNs, i.e., LeNet-5 [11], VGG [34], and ResNet [35] in Table 3. In Table 3, we estimate the proof sizes of SafetyNet, zkCNN, and our approach from Table 2 for the first convolution operation of each network.

We should note that $n_i$ is greater than $d_i$ in the real application. As a result, our methodology implies that the computational complexity and communication complexity of the prover and the verifier will be significantly reduced. The reason for this is that while the naïve approach is effective with one-dimensional data, it does not suit well with the convolution operation, which deals with three-dimensional data. We point out that our method is the first sum-check protocol optimized for three-dimensional convolution operations, which are employed in image recognition, image classification, and other image-processing neural networks.

Finally, we remark that the computation cost of the verifier is comparable with that of zkCNN for a CNN with deep layers such as VGG and ResNet. This is because $c_i$ becomes larger than $n_i$ for deep layers while $n_i$ is larger than $c_i$ for early layers. For example, the last layer $\ell$ of VGG has parameters $n_{\ell-1} = 14$, $n_\ell = 7$, and $c_\ell-1 = c_\ell = 512$.

B. ACTIVATION AND POOLING
We adopt quadratic activation and sum pooling to make all of the neural network processes can appear only with arithmetic operations following the ideas used in SafetyNet [7]. Here, we address how to conduct the activation and sum pooling operations in our network settings.
1) Quadratic Activation

The quadratic activation performs an element-wise operations. For our purpose, we should express the quadratic function as a linear function in each binary variable. We consider $U \in \mathbb{Z}_p^{n_1+1 \times n_1+1 \times c_1+1 \times b}$. To show the correct execution of the quadratic operations, we observe that

$$T(\alpha, \beta, \gamma, \tau) = U(\alpha, \beta, \gamma, \tau)^2 = \sum_{a, b \in \{0,1\}^\log n_1+1} \sum_{c \in \{0,1\}^\log c_1+1} I(\alpha, a) \cdot I(\beta, b) \cdot I(\gamma, c) \cdot (U(a, b, c, \tau))^2,$$

(13)

where $I$ is the identity matrix. We emphasize that it allows us to take a linear extension. The prover and the verifier run the sum-check protocol on the MLE of above (13). At the end of the sum-check protocol, the verifier yields an assertion about $U$. And to prove/verify the assertion is correct, both keep going on the proof procedure until the verifier gets the assertion from the input layer. The prover computational complexity is $O(n_1+2^2c_1+1b)$, the verifier computational complexity is $O(\log (n_1+2c_1+1))$ and to check in the last round. They communicate with $O(\log (n_1+2c_1+1))$ field elements.

2) Sum Pooling

The purpose of using pooling layer is to reduce the dimensions of the feature maps to prevent overfitting. For the sake of simplicity, we consider 2 by 2 pooling layer but it can be easily generalized by $n$ by $n$ pooling layer. Let $T \in \mathbb{Z}_p^{n_1 \times n_1 \times c_1 \times b}$ be an input matrix of pooling layer, $S \in \mathbb{Z}_p^{n_1+1 \times n_1+1 \times c_1+1 \times b}$ be an output matrix of pooling layer. We can represent sum pooling as multiplication of three matrices by defining a sparse matrix $P \in \mathbb{Z}_p^{n_1+1 \times 2^2}$ such that

$$P_{i,j} = \begin{cases} 1 & \text{if } j = 2i - 1 \text{ or } 2i \\ 0 & \text{otherwise} \end{cases}$$

If $S$ is valid output of pooling layer from the input matrix $T$, $S_{\gamma,r} = P^T \cdot T_{\gamma,r} \cdot P$ holds for any channel $\gamma \in [c_1+1]$ and batch $r \in [b]$, where $P^T$ denotes a transpose of $P$. Using the matrix equality and property of $P^T(x, y) = P(y, x)$, we have

$$S(\alpha, \beta, \gamma, \tau) = \sum_{a, b \in \{0,1\}^\log n_1-1} P(\beta, b) \cdot T(a, b, \gamma, \tau) \cdot P(\alpha, a).$$

(14)

From above equation, we consider a linear extension and use the sum-check protocol. Note that at the end of the protocol, the verifier checks the correctness of the value $P$ locally and keeps going on another sum-check protocol to check $T$. The prover computational cost is $O(n_1+2^2c_1+1b)$ and the verifier computational complexity is $O(\log (n_1+2))$. They communicate with $O(\log (n_1+2))$ field elements.

IV. EXTENSION TO CNN

Now we describe our protocol for simplified version of CNN that consists of convolution-activation-pooling-fully connected layers. From this simple neural network, it is straightforward to extend to other common CNNs. We assume that the prover and the verifier agree on the computational model and parameters. In other words, model parameters such as weights, biases, kernel size, and so on are publicly available. As we described in Section III, we assume that all of the operations in the CNN are the matrix multiplications.

A. HIGH LEVEL DESCRIPTION

Softmax is frequently used as the activation after the fully connected layer in a classification challenge to make the result interpretable as a probability. However, the proof generation for softmax is computationally demanding works. Thus, we assume that the verifier computes the weights of the fully connected layers first, rather than the probability result of network. The verifier then proves the claim for a fully connected layer using the sum-check protocol. For $1 \leq i \leq N_3$, let $f_i \in \mathbb{Z}_p[X_1, \ldots, X_{N_3}]$ be the polynomial that the prover sends to the verifier in $i$-th round of the sum-check protocol for fully connected layer. To finalize the claim about fully connected layer, the verifier checks whether

$$f_{N_3}(r_{N_3}) = \tilde{W}(\gamma) \cdot \tilde{F}(r)$$

(15)

for random point $r = (r_1, \ldots, r_{N_3}) \in \mathbb{Z}_p^{N_3}$. Here, $\tilde{W}$ and $\tilde{F}$ are MLE of weight and inputs of fully connected layer, respectively.

To check the above equation, the verifier should compute MLE of $F$, which is the output of pooling layer. Instead of evaluating directly, the verifier checks that the $\tilde{F}$ is from the previous pooling layer using (15). Since $\tilde{F}$ is the output of the previous pooling layer, $F$ has the form of (14). Both parties run the sum-check protocol on $\tilde{F}(r)$. This protocol guarantees that $F$ is a correct evaluation from pooling layer.

After that, the verifier needs to compute the MLE of inputs of pooling layer, which should be equivalent to outputs of activation layer. In this step, both parties run the sum-check protocol for the activation layer. As the last step, they run the convolution sum-check protocol. Since the verifier already knows the inputs of the network, the verifier can complete the convolution sum-check. It is crucial to improve the efficiency of the convolution part because the evaluation of MLE for weight matrix in each convolutional layer is the most computationally demanding works for the verifier. Thus, we remind that our approach of convolution is a key idea for efficient verifiable CNN.

B. PROTOCOL DESCRIPTION

Our goal is to build an efficient protocol for verifying the inference of CNN. Given a trained model for classification tasks, we first quantize the model so that it works over finite fields. And we represent all of the operations in the CNN
Protocol 2 CNN ($pp, W_{FC} \in \mathbb{Z}_p^{N_{x} \times s}$, $W_{c} \in \mathbb{Z}_p^{N_{c} \times s}$, $X \in \mathbb{Z}_p^{N_{x} \times b}$, $Y \in \mathbb{Z}_p^{s \times b}$, $F_1 \in \mathbb{Z}_p^{N_{x} \times b}$, $F_2 \in \mathbb{Z}_p^{N_{x} \times b}$, $F_3 \in \mathbb{Z}_p^{N_{s} \times b}$)

Common input: public parameter $pp = (n_i, c_i, d, s, b)$ for $i = 0, 1, 2, 3$

Prover’s witness: Intermediate layer $F_1, F_2, F_3$

Rounds indices for Sum-check protocols $j := 2 \log n_3 + \log c_3$ and $k := 2 \log n_2$ and $l := 2 \log n_1 + \log c_1$

1. V sends randoms $(\delta, \tau) \in \mathbb{Z}_p^{\log ab}$ to P. And then both parties compute $\tilde{Y}(\delta, \tau)$

Sum-check for fully connected layer $j$-rounds Sum-check protocol for following equation

$$\tilde{Y}(\delta, \tau) = \sum_{x \in \{0,1\}^j} f(x) := \sum_{x \in \{0,1\}^j} \tilde{W}_{FC}(x, \delta, \tilde{F}_3(x, \tau))$$

2. P computes and sends $f_1(0), f_1(1), f_2(2)$ to V where $f_1(X) := \sum_{t[2:j] \in \{0,1\}^{j-1}} f(X, t[2:j])$ and

3. V checks $\tilde{Y}(\delta, \tau) = f_1(0) + f_1(1)$ and if it does not hold, abort it. And then V sends a random $\xi_1 \in \mathbb{Z}_p$ to P

4. for $i = 2$ to $j$ do

5. P computes and sends $f_i(0), f_i(1), f_i(2)$ to V where $f_i(X) := \sum_{t[i+1:j] \in \{0,1\}^{j-1}} f(\xi_{i+1}, X, t[i+1:j])$

6. V checks $f_i-1(\xi_{i-1}) = f_i(0) + f_i(1)$ and if it does not hold, abort it. And then V sends a random $\xi_i \in \mathbb{Z}_p$ to P

7. P and V sparse $\xi_{[1:k]} = (\alpha \| \beta \| \gamma)$ with $\alpha, \beta, \gamma \in \mathbb{Z}_p^{\log n_3}$, $\gamma \in \mathbb{Z}_p^{\log c_3}$ and then V computes

$$y_1 := f_j(\xi_j) \cdot (\tilde{W}_{FC}((\alpha, \beta, \gamma), \delta))^{-1}$$

Sum-check for fully sum pooling layer $k$-rounds Sum-check protocol for following equation

$$y_1 = \sum_{x \in \{0,1\}^k} g(x) := \sum_{x \in \{0,1\}^k} \tilde{P}_{FC}(\alpha, x_1) \cdot \tilde{F}_2(x_1, x_2, \gamma, \tau) \cdot \tilde{P}_{FC}(\beta, x_2)$$

where $x := (x_1 \| x_2)$ with $x_1, x_2 \in \{0,1\}^{\log n_2}$

8. P computes and sends $g_1(0), g_1(1), g_1(2)$ to V where $g_1(X) := \sum_{t[2:k] \in \{0,1\}^{k-1}} g(X, t_2, \cdots, t_k)$

9. V checks $y_1 = g_1(0) + g_1(1)$ and if it does not hold, abort it. And then V sends a random $\epsilon_1 \in \mathbb{Z}_p$ to P

10. for $i = 2$ to $k$ do

11. P computes and sends $g_i(0), g_i(1), g_i(2)$ to V where $g_i(X) := \sum_{t[i+1:k] \in \{0,1\}^{k-1}} g(\epsilon_{i+1}, X, t[i+1:k])$

12. V checks $g_{i-1}(\epsilon_{i-1}) = g_i(0) + g_i(1)$ and if it does not hold, abort it. And then V sends a random $\epsilon_i \in \mathbb{Z}_p$ to P

13. P and V sparse $\epsilon_{[1:k]} = (\epsilon_1, \epsilon_2)$ where $\epsilon_1, \epsilon_2 \in \mathbb{Z}_p^{\log n_2}$ V computes

$$y_2 := g_k(\epsilon_k) \cdot (\tilde{P}(\alpha, \epsilon_1) \cdot \tilde{P}(\beta, \epsilon_2))^{-1}$$

Sum-check for quadratic activation $l$-rounds Sum-check protocol for following equation

$$y_2 = \sum_{x \in \{0,1\}^l} h(x) := \sum_{x \in \{0,1\}^l} \tilde{F}_1(x_1, x_2, \tau)^2 \cdot \tilde{I}(\epsilon_1, x_1) \cdot \tilde{I}(\epsilon_2, x_2) \cdot \tilde{I}(\gamma, x_3)$$

where $x := (x_1 \| x_2 \| x_3)$ with $x_1, x_2 \in \{0,1\}^{\log n_1}$, $x_3 \in \{0,1\}^{\log c_1}$

14. P computes and sends $h_1(0), h_1(1), h_1(2)$ to V where $h_1(X) := \sum_{t[2:l] \in \{0,1\}^{l-1}} g(X, t_2, \cdots, t_k)$

15. V checks $y_2 = h_1(0) + h_1(1)$ and if it does not hold, abort it. And then V sends a random $\rho_1 \in \mathbb{Z}_p$ to P

16. for $i = 2$ to $l$ do

17. P computes and sends $h_i(0), h_i(1), h_i(2)$ to V where $h_i(X) := \sum_{t[i+1:l] \in \{0,1\}^{l-1}} g(\rho_{i+1}, X, t[i+1:l])$

18. V checks $h_{i-1}(\rho_{i-1}) = h_i(0) + h_i(1)$ and if it does not hold, abort it. And then V sends a random $\rho_i \in \mathbb{Z}_p$ to P

19. P and V sparse $\rho_{[1:l]} = (\rho_1 \| \rho_2 \| \rho_3)$ where $\rho_1, \rho_2 \in \mathbb{Z}_p^{\log n_1}$, $\rho_3 \in \mathbb{Z}_p^{\log c_1}$. And then V computes

$$y_3 := h_1(\rho_1) \cdot (\tilde{I}(\epsilon_1, \rho_1) \cdot \tilde{I}(\epsilon_2, \rho_2) \cdot \tilde{I}(\gamma, \rho_3))^{-1}$$

20. P and V run Conv($(n_0, c_0, n_1, c_1, d), X, W_{c}, y_3, \rho_1, \rho_2, \rho_3, \tau)$

model as sequential matrix multiplications. Therefore, it is compatible with the GKR protocol variant for efficient matrix multiplication [9]. Note that the goal of the protocol is to convince the client that the claimed result was computed correctly using the predetermined neural network. It suffices to deal with the simple version of CNN. We denote $F_i \in \mathbb{Z}_p^{N_i \times b}$ where $N_i = n_i \times n_i \times c_i$ for $0 \leq i \leq 3$. Thus, we aim to show the following relation

$$R_{CNN} = \left\{ \begin{array} { l } { (pp, W_{FC} \in \mathbb{Z}_p^{N_{x} \times s}, W_{C} \in \mathbb{Z}_p^{N_{c} \times d \times d \times c_0}, X \in \mathbb{Z}_p^{n_0 \times n_0 \times c_0 \times b}, Y \in \mathbb{Z}_p^{s \times b}, F_1 \in \mathbb{Z}_p^{N_{x} \times N_3 \times c_3}, \text{ for } j = 1, 2, 3 : F_1 = C(X, W_{c}) \land F_2 = A(F_1) \land F_3 = P(F_2) \land Y = FC(F_3, W_{FC})) } \end{array} \right\}$$
where \( pp = (n_1, c_1, d, s, b) \), \( d \) is the kernel size of the convolution operation, \( s \) is the number of classes, and \( b \) is the batch size.

Our protocol consists of \( FC, P, A \) and \( C \) sequentially. We describe our protocol on Protocol 2. In \( FC \) part, \( V \) sends random vectors \( \delta \in \mathbb{Z}_{log^s}^p \) and \( \tau \in \mathbb{Z}_{log^b}^p \) to \( P \) and then both parties computes \( \tilde{Y}(\delta, \tau) \). And they run sum-check protocol for
\[
\tilde{Y}(\delta, \tau) = \sum_{(a, b, c) \in \{0,1\} \log N_3} \tilde{W}_{FC}((a, b, c), \delta) \cdot \tilde{F}_3(a, b, c, \tau).
\]
(16)

We denote \( (\alpha, \beta, \gamma) \in \mathbb{Z}_{p}^{log N_3} \) by \( V \)'s randoms for sum-check protocol. To check last verification equation for (16), \( V \) needs an evaluation of \( W_1(\alpha, \beta, \gamma, \delta) \cdot F_3(\alpha, \beta, \gamma, \tau) \). Though the verifier can compute \( W_1(\alpha, \beta, \gamma, \delta) \) cannot \( F_3(\alpha, \beta, \gamma, \tau) \) since he does not know \( F_3 \). Instead, prover sends the evaluation value and proves the correctness of the value, i.e., \( P \) and \( V \) run sum-check protocol for \( 2 \times 2 \) sum pooling of following equation
\[
\tilde{F}_3(\alpha, \beta, \gamma, \tau) = \sum_{a, b \in \{0,1\} \log n_2} \tilde{P}(a) \cdot \tilde{F}_2(a, b, \gamma, \tau) \cdot \tilde{F}(\beta, b),
\]
where \( n_2 = \frac{nn_1}{2} \). Now we denote \( \epsilon_1, \epsilon_2 \in \mathbb{Z}_{n_2}^p \) by \( V \)'s randoms from sum-check protocol of \( P \) part. The matrix \( P \) is sparse so that the verifier can check \( P \) by himself but cannot check \( F_2 \). Thus, the prover provides the evaluation of \( \tilde{F}_2(\epsilon_1, \epsilon_2, \gamma, \tau) \) and proves the assertion about the pooling layer according to the following equation from quadratic activation layer
\[
\tilde{F}_2(\epsilon_1, \epsilon_2, \gamma, \tau) = \sum_{a, b, c \in \{0,1\} \log n_1} I(\epsilon_1, a) \cdot I(\epsilon_2, b) \cdot I(\gamma, c),
\]
where \( n_1 = n_2 \) and \( c_1 = c_2 \). Let \( (\rho_1, \rho_2, \rho_3) \in \mathbb{Z}_{N_3}^p \) that are the random values chosen by verifier. The value of the MLE for the identity matrix is easy to calculate. Hence the only value that the verifier needs help from the prover is \( \tilde{F}_1(\rho_1, \rho_2, \rho_3, \tau) \). As we noted in (12), this is reduced to the claim for convolution operation on the input data. Finally, we obtain
\[
\tilde{F}_1(\rho_1, \rho_2, \rho_3, \tau) = \sum_{a, b, c \in \{0,1\} \log d} \sum_{p, q \in \{0,1\} \log n_0} \sum_{e \in \{0,1\} \log n_0} \tilde{X}(p, q, c) \cdot \tilde{J}_0(\rho_1, a, p) \cdot \tilde{J}_0(\rho_2, b, q) \cdot \tilde{W}_C(\rho_3, a, b, c)
\]
To reduce prover cost, we adapt the two-step sum-check idea. At the end of the convolution sum-check parts, \( V \) computes \( \tilde{X} \) using the input value of the CNN and then finalizes the protocol by outputting accept or reject.

\[\text{Theorem 2:} \] The interactive protocol presented in Section IV-B has soundness error
\[
O \left( \frac{\sum_{i=0}^{3} \log N_i + \log b + \log d + \log s}{p} \right).
\]

Proof: The protocol starts from reducing the prover’s claim about output matrix to a single point of the multilinear extension. This introduces the soundness error \( \frac{\log s}{p} \). The sum-check is carried over \( \log N_3 \) rounds for the fully connected layer and \( 2 \log n_2 \) rounds for the pooling layer and \( \log N_1 \) rounds for quadratic activation layer. Finally, the two sum-check protocols for convolutional layer take \( \log (d^2(d + 1)^2c_0) + 2 \log n_0 \) rounds. And the univariate polynomials that transmitted by prover during the sum-check protocol for pooling layer have degree at most 3 and all the rest of univariate polynomials have degree at most 2. This causes the factor of 3 and 2 in the soundness error. Hence, as the observation of Theorem 1, by adding up all the soundness error from all layers based on the protocols, we obtain the soundness error
\[
\frac{1}{p} \cdot (\log sb + 2 \log N_3 + 6 \log n_2 + 2 \log N_1 + 2 \log (d^2(d + 1)^2c_0) + 4 \log n_0).
\]

Since the sum-check round for pooling layer is only \( 2 \log n_2 \) rounds, \( \log c_2 \) does not appear in this step. However, for convenience sake, we write the term \( \frac{\log n_2}{p} \) as \( O \left( \frac{\log N_2}{p} \right) \).

V. CONCLUSION

In this paper, we construct a new efficient sum-check protocol for a convolution operation of CNN. The existing protocol such as SafetyNet and zkCNN transform a convolution operation of CNN to 2-dimensional matrix multiplication. However, CNN is defined as a form of matrix multiplication that takes a cubic time in the matrix dimension, and those approaches should apply a sum-check protocol for large weight matrices. The idea behind the construction is to interpret the input and intermediate values as 3-dimensional matrices, which is a natural approach for convolution operations. Our approach is non-trivial because convolution operations in this setting do not satisfy linearity. We resolve the challenge by introducing predicate functions that make a convolution operation linearized in the each binary variable. Our construction provides asymptotically optimal proving cost as zkCNN, which is much more efficient than SafetyNet. The proof size of our construction is approximately \( 2 \times \) smaller than that of zkCNN. In addition, by applying our construction, we present a new verifiable CNN which employs quadratic activation and sum pooling. We will continue to research applying our approach for CNN with other activation functions and pooling operations widely used in future work.

REFERENCES

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, 1998.
CHANYANG JU was born in Incheon, Republic of Korea in 1994. She received the B.S. in mathematics from Myongji University, Yongin, Republic of Korea in 2013. She is currently pursuing the Ph.D. degree in applied mathematics from Hanyang University, Seoul, Republic of Korea. Her research interest includes zero-knowledge proof (ZKP) and verifiable computation (VC).

HYEONBUM LEE received the B.S. in mathematics from Hanyang University, Seoul, Republic of Korea in 2018. He is currently pursuing the Ph.D. degree in applied mathematics from Hanyang University, Seoul, Republic of Korea. His research interests include zero-knowledge proof (ZKP) and SNARKs.

HEEWON CHUNG received the B.S. in mathematics from the Korea Advance Institute Science and Technology (KAIST), Daejon, Republic of Korea, in 2009 and M.S. degrees and the Ph.D. degree in mathematics from Seoul National University, Seoul, Republic of Korea, in 2017. From 2016 to 2017, he was a Research Assistant with the Agency for Science, Technology, and Research (A*STAR) in Singapore. From 2018 to 2019, he was a Manager with Korea Telecom in Republic of Korea. Since 2020, he has been a Postdoctoral researcher in Hanyang University, Seoul, Republic of Korea. His research interest includes resolving the scalability problem and confidential transactions in blockchain using zero-knowledge proofs, especially, non-interactive zero-knowledge (NIZK) proofs and SNARKs.

JAE HONG SEO received the B.S. degree in mathematics from Korea University in 2004 and the Ph.D. degree from the Department of Mathematical Science, Seoul National University in 2011. From 2011 to 2013, he was with National Institute of Communications and Technology, Japan. From 2013 to 2018, he was with Myongji University, Korea. Since 2018, he has been an Associate Professor with the Department of Mathematics, Hanyang University, Korea. His research interests include computational algorithms, cryptology, and their applications to deep learning and blockchain.

Dr. Seo has served as a Program Committee Member for IACR conferences and workshops. He was a recipient of Sangsan Young Mathematician Award, Korean Mathematical Society, in 2012 and Young Scientist Award for Excellence, Elsevier & NRF in 2018. He was the Program Committee (Co-)Chair of APKC 2018 and ICISC 2019.

SUNGWOOK KIM received the B.S and joint M.S/Ph.D. degree in mathematics from Seoul National University (SNU), South Korea, in 2005 and 2012, respectively. He is currently an assistant professor with the Department of Information Security, Seoul Women’s University (SWU). Before joining SWU in 2020, he was a principal/senior engineer at Samsung Research, Samsung Electronics, Seoul from 2014 to 2020. His research interests include cryptography, information security, and privacy-preserving technology.