Precise Approximation of Convolutional Neural Networks for Homomorphically Encrypted Data

Junghyun Lee\textsuperscript{1}, Eunsang Lee\textsuperscript{1}, Joon-Woo Lee\textsuperscript{1}, Yongjune Kim\textsuperscript{2}, Young-Sik Kim\textsuperscript{3}, Jong-Seon No\textsuperscript{1}

\textsuperscript{1}Department of Electrical and Computer Engineering, INMC, Seoul National University
\textsuperscript{2}Department of Information and Communication Engineering, DGIST
\textsuperscript{3}Department of Information and Communication Engineering, Chosun University
ljhfree530@snu.ac.kr, shaeunsang@snu.ac.kr, joonwoo42@snu.ac.kr, yjk@dgist.ac.kr, iamyskim@chosun.ac.kr, jsno@snu.ac.kr

Abstract

Homomorphic encryption is one of the representative solutions to privacy-preserving machine learning (PPML) classification enabling the server to classify private data of clients while guaranteeing privacy. This work focuses on PPML using word-wise fully homomorphic encryption (FHE). In order to implement deep learning on word-wise homomorphic encryption (HE), the ReLU and max-pooling functions should be approximated by some polynomials for homomorphic operations. Most of the previous studies focus on HE-friendly networks, where the ReLU and max-pooling functions are approximated using low-degree polynomials. However, for the classification of the CIFAR-10 dataset, using a low-degree polynomial requires designing a new deep learning model and training. In addition, this approximation by low-degree polynomials cannot support deeper neural networks due to large approximation errors. Thus, we propose a precise polynomial approximation technique for the ReLU and max-pooling functions. Precise approximation using a single polynomial requires an exponentially high-degree polynomial, which results in a significant number of non-scalar multiplications. Thus, we propose a method to approximate the ReLU and max-pooling functions accurately using a composition of minimax approximate polynomials of small degrees. If we replace the ReLU and max-pooling functions with the proposed approximate polynomials, well-studied deep learning models such as ResNet and VGGNet can still be used without further modification for PPML on FHE. Even pre-trained parameters can be used without retraining. We approximate the ReLU and max-pooling functions in the ResNet-152 using the composition of minimax approximate polynomials of degrees 15, 27, and 29. Then, we succeed in classifying the plaintext ImageNet dataset with 77.52\% accuracy, which is very close to the original model accuracy of 78.31\%.

1 Introduction

One of the most promising areas in privacy-preserving machine learning (PPML) is deep learning using homomorphic encryption (HE). HE is a cryptographic system that allows algebraic operations such as addition and multiplication on encrypted data. Using HE, the server can perform deep learning on the encrypted data from the client while fundamentally preventing the privacy leakage of private

\textsuperscript{*}Equal contribution.
\textsuperscript{†}Corresponding authors.
data. Recently, deep learning using HE has been studied extensively [1, 2, 3, 4, 5, 6, 7]. These studies can be classified into deep learning studies using only HE and those using both HE and multi-party computation (MPC) techniques together. Our work focuses on the former, which has advantages of low communication cost and no risk of model exposure compared to hybrid methods using both HE and MPC techniques. In particular, we focus on deep learning using the word-wise fully homomorphic encryption (FHE), which allows the addition and multiplication of an encrypted array over the set of complex numbers $\mathbb{C}$ or $\mathbb{Z}_p$ for an integer $p > 2$ without restriction on the number of operations.

In word-wise FHE, the ReLU and max-pooling functions cannot be performed on the encrypted data because they are non-arithmetic operations. Thus, after the approximate polynomials replace the ReLU and max-pooling functions, deep learning on the encrypted data can be performed. Most of the previous studies of deep learning on HE focus on HE-friendly networks, where the ReLU and max-pooling functions are approximated using low-degree polynomials. These studies on HE-friendly networks have some limitations. For instance, designing a new deep learning model and training are required in [5, 7, 8], which classified the CIFAR-10 [9]. However, in many real-world applications, training data or training frameworks are not accessible, and thus, the previously studied low-degree approximate polynomials cannot be used in these applications. Furthermore, HE-friendly networks cannot support very deep neural networks due to large approximation errors. On the other hand, the computational time and accuracy of bootstrapping in word-wise FHE have recently been improved significantly [10, 26, 12, 13], allowing the evaluation of higher-degree polynomials for many times. Thus, it is essential to study precise approximate polynomials.

### 1.1 Our contributions

Precise approximation requires a polynomial of degree at least about $2^{1.013\alpha-2.8483}$ for the precision parameter $\alpha$, which results in $2^{6\alpha}$ number of non-scalar multiplications and several numerical issues (see Section 3.3). Thus, we propose a precise polynomial approximation technique for the ReLU and max-pooling functions that uses a composition of minimax approximate polynomials of small degrees. Hence, we can replace the ReLU and max-pooling functions with the proposed approximate polynomials in well-studied deep learning models such as ResNet [4], VGGNet [5], and GoogLeNet [6], and the pre-trained parameters can be used without retraining. Figure 1 shows the comparison between the original deep learning model $F$ and the proposed approximate deep learning model $F^\alpha$ for the precision parameter $\alpha$, where $F$ is the function that outputs the inference result for an input $x$.

We prove that the two inference results, $F^\alpha(x)$ and $F(x)$, satisfy $\|F^\alpha(x) - F(x)\|_\infty \leq C2^{-\alpha}$ for a constant $C$, which can be determined independently of the precision parameter $\alpha$ (see Theorem 4). Theoretically, if we increase the precision parameter $\alpha$ enough, the two inference results would be identical. The numerical evaluation also supports that the two inference results are very close for an appropriate $\alpha$.

We classify the CIFAR-10 without retraining using well-studied deep learning models such as ResNet and VGGNet, and pre-trained parameters. We can also further reduce the required precision parameter $\alpha$ by allowing retraining for only a few epochs. For classifying the ImageNet [17] in ResNet-152, we...
approximate the ReLU and max-pooling functions using the composition of minimax approximate polynomials of degrees 15, 27, and 29. Then, for the first time for PPML using word-wise HE, we successfully classify the ImageNet with 77.52% accuracy, which is very close to the original model accuracy of 78.31%. The proposed approximate polynomials are generally applicable to various deep learning models such as ResNet, VGGNet, and GoogLeNet for PPML on FHE.

1.2 Related works

Deep learning on the homomorphically encrypted data, i.e., PPML using HE, can be classified into two categories: One uses HE only, and the other uses HE and MPC techniques together. Gazelle [2], Cheetah [6], and nGraph-HE [4] are representative studies that perform deep learning using HE and MPC techniques together. They used garbled circuits to perform activation functions through communication with clients and servers. However, the use of MPC techniques can lead to high communication costs or disclose information on the configuration of the deep learning models.

PPML using only HE can be classified into PPML using bit-wise HE and PPML using word-wise HE. In [18], PPML using the fastest fully homomorphic encryption over the torus (TFHE) [19], which is the most promising bit-wise FHE, was studied. This PPML can evaluate the ReLU and max-pooling functions exactly on the encrypted data. However, leveled HE was used instead of FHE because of the large computational time of FHE operations, which limits performing a very deep neural network on HE. Our work focuses on deep learning using word-wise HE such as the Brakerski/Fan-Vercouteren [20] and Cheon-Kim-Kim-Song (CKKS) scheme [21]. CryptoNets [1], Faster CryptoNets [5], SEALion [3], and studies in [7, 8] also performed deep learning using word-wise HE. By designing a new deep learning model and training, the authors in [5], [7], and [8] classified the CIFAR-10 with an accuracy of about 75%, 77%, and 91.5%, respectively. CryptoNets and SEALion replaced the ReLU function with square function $x^2$, and Faster CryptoNets replaced the ReLU function with quantized minimax approximate polynomial $2^{-3}x^2 + 2^{-1}x + 2^{-2}$.

2 Preliminaries

2.1 Minimax composite polynomial

Unfortunately, FHE schemes do not support non-arithmetic operations such as sign function, where $\text{sgn}(x) = x / |x|$ for $x \neq 0$, and 0, otherwise. Hence, instead of the actual sign function, a polynomial approximating the sign function should be evaluated on the encrypted data. Recently, authors in [22] proposed a composite polynomial named minimax composite polynomial that optimally approximates the sign function with respect to the depth consumption and the number of non-scalar multiplications. The definition of minimax composite polynomial is given as follows:

**Definition 1.** Let $D$ be $[-b, -a] \cup [a, b]$. A composite polynomial $p_k \circ p_{k-1} \circ \cdots \circ p_1$ is called a minimax composite polynomial on $D$ for $\{d_i\}_{1 \leq i \leq k}$ if the followings are satisfied:

- $p_1$ is the minimax approximate polynomial of degree at most $d_1$ on $D$ for $\text{sgn}(x)$.
- For $2 \leq i \leq k$, $p_i$ is the minimax approximate polynomial of degree at most $d_i$ on $p_{i-1} \circ p_{i-2} \circ \cdots \circ p_1(D)$ for $\text{sgn}(x)$.

For $0 < \epsilon < 1$ and $\beta > 0$, a polynomial $p(x)$ that approximates $\text{sgn}(x)$ is called $(\beta, \epsilon)$-close if it satisfies $|p(x) - \text{sgn}(x)| \leq 2^{-\beta}$ for $x \in [-1, -\epsilon] \cup [\epsilon, 1]$ [23]. Then, the optimal $(\beta, \epsilon)$-close composite polynomial $p(x)$ is the minimax composite polynomial on $[-1, -\epsilon] \cup [\epsilon, 1]$ for $\{d_i\}_{1 \leq i \leq k}$, which obtained from Algorithm 5 for inputs $\beta$ and $\epsilon$ in [22].

2.2 Homomorphic max function

To perform the max function on the encrypted data, called the homomorphic max function, a polynomial that approximates the max function should be evaluated on the encrypted data instead of the max function. The approximate polynomial of the max function, $m(a, b)$, should satisfy the following inequality for the precision parameter $\alpha > 0$:

$$|m(a, b) - \max(a, b)| \leq 2^{-\alpha} \quad \text{for} \quad a, b \in [0, 1].$$  (1)
In \cite{23}, considering $\max(a, b) = \frac{(a+b)+ (a-b) \sgn(a-b)}{2}$, they obtained a polynomial $p(x)$ that approximates $\sgn(x)$ and then used it to determine a polynomial $m(a, b)$ that approximates $\max(a, b)$, that is,

$$m(a, b) = \frac{(a + b) + (a - b)p(a - b)}{2}. \quad (2)$$

In \cite{24}, the authors proposed a minmax composite polynomial $p(x)$ approximating the sign function. Then, they proposed a polynomial $m(a, b)$ approximating the max function using $p(x)$ and equation (2). Specifically, for a precision parameter $\alpha$ and the optimized max function factor $\zeta_\alpha > 0$, an $(\alpha - 1, \zeta_\alpha \cdot 2^{-\alpha})$-close minimax composite polynomial $p_\alpha(x) = p_{\alpha,k} \circ \cdots \circ p_{\alpha,1}(x)$ was obtained by using the optimal degrees obtained from Algorithm 5 in \cite{22}, where the depth consumption is minimized. Then, it was shown that $m_\alpha(a, b) = \frac{(a+b)+ (a-b)p_\alpha(a-b)}{2}$ from $p_\alpha(x)$ satisfies the error condition in equation (1). Table 1 lists the values of $\zeta_\alpha$, corresponding depth consumption for evaluating $m_\alpha(a, b)$, and degrees of the component polynomials $p_{\alpha,1}, \cdots, p_{\alpha,k}$ according to $\alpha$. This approximate polynomial $m_\alpha(a, b)$ has the best performance up to now.

### 3 Precise polynomial approximation of the ReLU and max-pooling functions

It is possible to approximate the ReLU and max-pooling functions by Taylor polynomials or minimax approximate polynomials \cite{24}. However, these approximate polynomials with minimal approximation errors require many non-scalar multiplications. For this reason, we approximate the ReLU and max-pooling functions using $p_\alpha(x)$ and $m_\alpha(a, b)$, which are defined in Section 2.2. Although we only deal with the ReLU function among many activation functions, the proposed approximation method can also be applied to other activation functions such as the leaky ReLU function.

#### 3.1 Precise approximate polynomial of the ReLU function

In this subsection, we propose a polynomial that precisely approximates the ReLU function, a key component of deep neural networks. For a given precision parameter $\alpha$, it is required that the approximate polynomial of the ReLU function, $r_\alpha(x)$, satisfies the following error condition:

$$|r_\alpha(x) - \text{ReLU}(x)| \leq 2^{-\alpha} \quad \text{for } x \in [-1, 1]. \quad (3)$$

![Figure 2: The component polynomials of $p_\alpha(x)$ and the proposed $r_\alpha(x)$ with precision parameter $\alpha = 11$.](image)

(a) $p_{\alpha,1}(x)$, (b) $p_{\alpha,2}(x)$, (c) $p_{\alpha,3}(x)$, (d) $p_\alpha(x)$, and (e) $r_\alpha(x)$ are the approximate polynomial of the ReLU function. The degrees of $p_{\alpha,1}(x)$, $p_{\alpha,2}(x)$, and $p_{\alpha,3}(x)$ are 7, 7, and 27, respectively.

### Table 1: The optimized max function factor, degrees of the component polynomials, and depth consumption for evaluating $m_\alpha(a, b)$ according to the precision parameter $\alpha$.

| $\alpha$ | $\zeta_\alpha$ | degrees $p_{\alpha,1}$ | degrees $p_{\alpha,2}$ | degrees $p_{\alpha,3}$ | depth consumption |
|----------|----------------|------------------------|------------------------|------------------------|-------------------|
| 4        | 5              | [5]                    | 4                      |                        | 4                 |
| 5        | 5              | [13]                   | 5                      |                        | 5                 |
| 6        | 10             | {3.7}                  | 6                      |                        | 6                 |
| 7        | 11             | {7.7}                  | 7                      |                        | 7                 |
| 8        | 12             | {7.15}                 | 8                      |                        | 8                 |
| 9        | 13             | {15,15}                | 9                      |                        | 9                 |
| 10       | 13             | {7.7,13}               | 11                     |                        | 11               |
| 11       | 15             | {7,7,27}               | 12                     |                        | 12               |
| 12       | 15             | {7,15,27}              | 13                     |                        | 13               |
| 13       | 16             | {15,15,27}             | 14                     |                        | 14               |
| 14       | 17             | {15,27,29}             | 15                     |                        | 15               |
It is well-known that \( \text{ReLU}(x) = \frac{x + x \text{sgn}(x)}{2} \). Then, we propose the following approximate polynomial of the ReLU function, \( r_\alpha(x) = \frac{x + x_p(x)}{2} \). The specific coefficient values of \( r_\alpha(x) \) for several \( \alpha \) can be seen in Appendix A. Figure 2 shows the component polynomials of \( p_\alpha(x) \) and the proposed \( r_\alpha(x) \) with the precision parameter \( \alpha = 11 \). The following theorem shows that the proposed \( r_\alpha(x) \) satisfies the error condition of the ReLU function in equation (5).

**Theorem 1.** For any \( \alpha > 0 \), \( r_\alpha(x) \) satisfies the following inequality:

\[
| r_\alpha(x) - \text{ReLU}(x) | \leq 2^{-\alpha} \quad \text{for} \quad x \in [-1, 1].
\]

The proofs of all theorems and a lemma are given in Appendix B.

**Approximation range** The value of polynomial \( r_\alpha(x) \) is close to the ReLU function value only for \( x \in [-1, 1] \). However, since some input values of the ReLU function during deep learning are not in \([-1, 1]\), the approximation range should be extended from \([-1, 1]\) to a larger range \([-B, B]\) for some \( B > 1 \). Thus, we propose a polynomial \( \tilde{r}_{\alpha,B}(x) := Br_\alpha(x/B) \) approximating the ReLU function in the range \([-B, B]\). Then, we have

\[
| \tilde{r}_{\alpha,B}(x) - \text{ReLU}(x) | = | Br_\alpha(x/B) - \text{ReLU}(x) | = | Br_\alpha(x/B) | - B \text{ReLU}(x/B) = | B(\tilde{r}_{\alpha,B}(x/B) - \text{ReLU}(x/B)) | \leq B 2^{-\alpha} \quad \text{for} \quad x \in [-B, B].
\]

Large \( B \) has the advantage of enlarging input ranges of the ReLU function at the cost of causing a larger approximation error with the ReLU function.

### 3.2 Precise approximate polynomial of the max-pooling function

The max-pooling function with kernel size \( k \times k \) outputs the maximum value among \( k^2 \) input values. To implement the max-pooling function for the homomorphically encrypted data, we should find a polynomial that approximates the max-pooling function.

We define a polynomial \( M_{\alpha,n}(x_1, \ldots, x_n) \) that approximates the maximum value function \( \max(x_1, \ldots, x_n) \) using \( m_\alpha(a, b) \). To reduce the depth consumption of the approximate polynomial of the max-pooling function, we construct \( M_{\alpha,n} \) by compositing polynomial \( m_\alpha(a, b) \) with a minimal number of compositions. We use the following recursion equation to obtain the approximated maximum value among \( x_1, \ldots, x_n \):

\[
M_{\alpha,n}(x_1, \ldots, x_n) = \begin{cases} 
  x_1, & n = 1 \\
  m_\alpha(M_{\alpha,k}(x_1, \ldots, x_k), M_{\alpha,k}(x_{k+1}, \ldots, x_{2k})), & n = 2k \\
  m_\alpha(M_{\alpha,k}(x_1, \ldots, x_k), M_{\alpha,k+1}(x_{k+1}, \ldots, x_{2k+1})), & n = 2k + 1.
\end{cases}
\]

The following theorem shows the upper bound of approximation error of \( M_{\alpha,n}(x_1, \ldots, x_n) \).

**Theorem 2.** For given \( \alpha > 0 \) and \( n \in \mathbb{N} \), the polynomial \( M_{\alpha,n}(x_1, \ldots, x_n) \) obtained from the recursion equation in (4) satisfies

\[
| M_{\alpha,n}(x_1, \ldots, x_n) - \max(x_1, \ldots, x_n) | \leq 2^{-\alpha} \lceil \log_2 n \rceil
\]

for \( x_1, \ldots, x_n \in \left[ \lceil \log_2 n \rceil - 1 \right) 2^{-\alpha}, 1 - \left( \lceil \log_2 n \rceil - 1 \right) 2^{-\alpha} \rceil \).

**Approximation range** To extend the approximation range \( \left( \lceil \log_2 n \rceil - 1 \right) 2^{-\alpha}, 1 - \left( \lceil \log_2 n \rceil - 1 \right) 2^{-\alpha} \) to \([-B, B]\) for \( B > 1 \), we use the following polynomial as an approximation polynomial:

\[
\tilde{M}_{\alpha,n,B}(x_1, \ldots, x_n) = B' \cdot M_{\alpha,n}(\frac{x_1}{B'}, \ldots, \frac{x_n}{B'}, 0.5) - 0.5,
\]

where \( B' = B/(0.5 - \left( \lceil \log_2 n \rceil - 1 \right) 2^{-\alpha}) \). Then, we have

\[
| \tilde{M}_{\alpha,n,B}(x_1, \ldots, x_n) - \max(x_1, \ldots, x_n) | \leq B' \cdot 2^{-\alpha} \lceil \log_2 n \rceil,
\]

for \( x_1, \ldots, x_n \in [-B, B] \), which directly follows from Theorem 2.

### 3.3 Comparison between the minimax approximate polynomial and the proposed approximate polynomial for the ReLU and max-pooling functions
One can think that precise approximation of the ReLU function can be achieved simply by using a high-degree polynomial such as a Taylor polynomial and minimax approximate polynomial [24]. In particular, the minimax approximate polynomials have the minimum minimax error for a given degree. Thus, the minimax approximate polynomials are good candidates for the approximate polynomials. In this subsection, we compare the proposed approximation method using \( r_\alpha(x) \) and the method using the minimax approximate polynomial. For simplicity, let the approximation range of the ReLU function be \([-1, 1]\). Then, for a given degree \( d \), the minimax approximate polynomial of degree at most \( d \) on \([-1, 1]\) for \( \text{ReLU}(x) \) can be obtained from the Remez algorithm [24]. The precision parameter \( \alpha \) corresponds to the logarithm of the minimax error using base two. Figure 3 presents the corresponding precision parameter \( \alpha \) according to the logarithm of the polynomial degree (base two). Because performing the Remez algorithm for a high-degree polynomial has several difficulties, such as using high real number precision and finding extreme points with high precision, we obtain the polynomials for only up to degree 200. It can be seen that \( \log_2(d) \) and \( \alpha \) have an almost exact linear relationship, and we obtain the following regression equation:

\[
\alpha = 0.9987 \log_2(d) + 2.8446.
\]  

Table 2 compares the approximation method using the minimax approximate polynomial and that using the proposed approximate polynomial \( r_\alpha(x) \). The required number of non-scalar multiplications for evaluating a polynomial of degree \( d \) is \( O(\sqrt{2d}) \) [25], and thus, \( 2^{\Theta(\alpha)} \) non-scalar multiplications are required for the precision parameter \( \alpha \). The exact number of non-scalar multiplications is obtained using the odd baby-step giant-step algorithm for polynomials with only odd-degree terms and using the optimized baby-step giant-step algorithm for other polynomials [11]. From Table 2, it can be seen that the minimax approximate polynomial of exponentially high degree is required for a large \( \alpha \). This high degree requires many non-scalar multiplications, i.e., large computational time. In addition, using a high-degree polynomial has some numerical issues, such as the requirement of high precision of the real number. Thus, precise approximation using the minimax approximate polynomial is inefficient to use in practice.

### Homomorphic max function

Considering \( \max(a, b) = \frac{a + b + |a - b|}{2} \), we obtain the minimax approximate polynomial \( p(x) \) on \([-1, 1]\) for \( |x| \). Then, \( m(a, b) = \frac{a + b + p(a - b)}{2} \) can be used as an approximate polynomial of the max function \( \max(a, b) \). Since \( \text{ReLU}(x) = \frac{x + |x|}{2} \), the approximation

| \( \alpha \) | minimax approx. polynomial | proposed approx. polynomial |
|-----|--------------------------|---------------------------|
|     | degree | #mults | degrees | #mults |
| 6   | 10     | 5      | [3,7]   | 9      |
| 7   | 20     | 8      | [7,7]   | 12     |
| 8   | 40     | 12     | [7,15]  | 15     |
| 9   | 80     | 18     | [15,15] | 15     |
| 10  | 150    | 25     | [7,7,13]| 16     |
| 11  | 287*   | 35     | [7,7,27]| 19     |
| 12  | 575*   | 49     | [7,15,27]| 22 |
| 13  | 1151*  | 70     | [15,15,27]| 25 |
| 14  | 2304*  | 98     | [15,27,29]| 28 |
| 15  | 4612*  | 140    | [29,29,29]| 30 |
of $|x|/2$ and ReLU($x$) are equivalent. Thus, when the max function is approximated with the minimax approximate polynomial, an exponentially large degree is also required to achieve large $\alpha$.

4 Theoretical performance analysis of approximate deep learning model

In this section, we propose an approximate deep learning model for homomorphically encrypted input data, replacing the original ReLU and max-pooling functions with the proposed approximate polynomials $\tilde{r}_{\alpha,B}(x)$ and $\tilde{M}_{\alpha,B}(x_1, \ldots, x_n)$ in the original deep learning model. In addition, we analytically derive that if the pre-trained parameters of the original deep learning model are available, classification would be accurate if we adopt a proper precision parameter $\alpha$.

In more detail, we estimate the L-infinity norm of the difference between the inference results $F(x)$ and $F^\alpha(x)$, $\|F^\alpha(x) - F(x)\|_\infty$. Here, $F(x)$ and $F^\alpha(x)$ denote the inference results of the original deep learning model $F$ and the proposed approximate deep learning model $F^\alpha$ with the precision parameter $\alpha$, respectively. Here, $F$ and $F^\alpha$ share the same pre-trained parameters. The main conclusion of this section is that we can reduce the difference $\|F^\alpha(x) - F(x)\|_\infty$ by increasing $\alpha$, which implies that the proposed model shows almost the same performance as the original pre-trained model.

To estimate $\|F^\alpha(x) - F(x)\|_\infty$, we attempt to decompose the original deep learning model, analyze each component, and finally re-combine them. First, we decompose given deep learning model $F$ as $A_0 \circ A_{n-1} \circ \cdots \circ A_0$, where $A_i$ is called a block. Each block can be a simple function: convolution, batch normalization, activation function, and pooling function. It can also be a mixed operation that cannot be decomposed into simple operations. Here, all of the inputs and outputs of block $A_i$’s are considered as a one-dimensional vector. Also, we denote $A^\alpha_i$ as the corresponding approximate polynomial operation of each block $A_i$ with precision parameter $\alpha$. If block $A_i$ contains only a polynomial operation, $A^\alpha_i$ is the same as $A_i$ since there is nothing to approximate. Then, the approximate model $F^\alpha$ can be considered as a composition of approximate blocks, $A^\alpha_n \circ A^\alpha_{n-1} \circ \cdots \circ A^\alpha_0$.

If there is only a single approximate block in the approximate deep learning model, the difference in inference results can be easily determined. However, the model $F^\alpha$ composed of more than two approximate blocks makes the situation more complicated. Consider an input vector $x$, two consecutive blocks $A_1$ and $A_2$, and their approximation $A^\alpha_1$ and $A^\alpha_2$. The block $A^\alpha_1$ makes an approximation error which can be measured as $e := \|A^\alpha_1(x) - A_1(x)\|_\infty$. For a given input $x$, the magnitude of the error $e$ is only determined by the intrinsic feature of block $A_1$. However, the output error of the second block is quite different. Note that we can write the second block error as $\|A^\alpha_2(y + e) - A_2(y)\|_\infty$, where $y = A_1(x)$ and $e = A^\alpha_1(x) - A_1(x)$. Considering $\|e\|_\infty = e$, both block $A_2$ and the magnitude of $e$ affect the second block error. Then, we define a new measure for each block, error propagation function.

In this study, all inputs $x$ in the deep learning models belong to a bounded set for a given dataset. We should set the approximation range $[-B, B]$ for the proposed approximate polynomials for the ReLU and max-pooling functions. We assume that we can choose a sufficiently large $B$ such that the inputs of all blocks fall in $[-B, B]$. That is, for a given deep learning model $F = A_n \circ A_{n-1} \circ \cdots \circ A_0$ and the corresponding approximate model $F^\alpha = A^\alpha_n \circ A^\alpha_{n-1} \circ \cdots \circ A^\alpha_0$, we have $\|A_i \circ \cdots \circ A_0(x)\|_\infty \leq B$ and $\|A^\alpha_i \circ \cdots \circ A^\alpha_0(x)\|_\infty \leq B$ for $0 \leq i \leq n - 1$ and every input $x$. In fact, we confirm through numerical analysis that these are satisfied for an appropriately large $B$ in Section 5. We define error propagation function as follows:

\textbf{Definition 2.} For a block $A$, an error propagation function of $A$, $E^\alpha_A(\cdot)$, is defined as $E^\alpha_A(e) := \sup_{\|x\|_\infty \leq B, \|e\|_\infty \leq e} \|A^\alpha(x + e) - A(x)\|_\infty$, where $e$ denotes the magnitude of output error of the previous block of $A$.

Roughly speaking, block $A^\alpha$ propagates the input error $e$ (or the output error of the previous block) to output error $E^\alpha_A(e)$. With this error propagation function, we can estimate the difference between final inference results of $F$ and $F^\alpha$ straightforwardly as in the following theorem.

\textbf{Theorem 3.} If the original model $F$ can be decomposed as $A_n \circ A_{n-1} \circ \cdots \circ A_0$, then we have $\|F^\alpha(x) - F(x)\|_\infty \leq (E^\alpha_{A_n} \circ E^\alpha_{A_{n-1}} \circ \cdots \circ E^\alpha_{A_0})(0)$ for every input $x$. 

7
This theorem asserts that we can achieve an upper bound of inference error of the whole model by estimating an error propagation function for each block.

We analyze the error propagation function for four types of single blocks: i) linear block, ii) ReLU block, iii) max-pooling block, and iv) softmax block, commonly used in deep neural networks for classification tasks. From now on, we call these four types of blocks basic blocks for convenience. Note that convolutional layers, batch normalization layers (for inference), average pooling layers, and fully connected layers are just linear combinations of input vectors and model parameters. Therefore, it is reasonable to include all these layers in a linear block. We can express linear blocks as \( A(x) = Ax + b \) for some matrix \( A \) and vector \( b \). The following lemma shows practical upper bounds of error propagation function for basic blocks. Because the kernel size of the max-pooling function is not greater than ten in most applications, we constrain the kernel size not greater than ten in the following lemma.

**Lemma 1.** For the given error \( e \), the error propagation functions of four basic blocks are upper bounded as follows: (a) If \( A \) is a linear block with \( A(x) = Ax + b \), then \( E_A^\alpha(e) \leq \|A\|_\infty e \), where \( \| \cdot \|_\infty \) denotes the infinity norm of matrices. (b) If \( A \) is a ReLU block, then \( E_A^\alpha(e) \leq B2^{-\alpha} + e \). (c) If \( A \) is a max-pooling block with kernel size \( k_0 \leq 10 \) and \( \alpha \geq 4 \), then \( E_A^\alpha(e) \leq 10B\lceil \log_2 k_0^2 \rceil 2^{-\alpha} + e \). (d) If \( A \) is a softmax block, then \( E_A^\alpha(e) \leq e/2 \).

Since the upper bounds of error propagation functions of all basic blocks have been determined, we can obtain an upper bound of the inference error theoretically as in the following theorem.

**Theorem 4.** If \( \alpha \geq 4 \) and a deep learning model \( F \) contains only basic blocks in Lemma 1, then there exists a constant \( C \) such that \( \|F^\alpha(x) - F(x)\|_\infty \leq C2^{-\alpha} \) for every input \( x \), where the constant \( C \) can be determined independently of the precision parameter \( \alpha \).

Therefore, the performance of the proposed approximate deep learning model containing only basic blocks can be guaranteed theoretically by Theorem 4 if the original model \( F \) is well trained.

**Practical application of Theorem 4** Theorem 4 is valid for a deep learning model composed of only basic blocks. VGGNet [15] is a representative example. However, there are also various deep learning models with non-basic blocks. For example, ResNet [14] contains residual learning building block, which is hard to decompose into several basic blocks. Thus, the generalized version of Theorem 4 is presented in Appendix C, which is also valid for ResNet. Numerical results in Section 5 also support that the inference result of the original deep learning model and that of the proposed approximate deep learning model are close enough for a practical precision parameter \( \alpha \).

## 5 Results of numerical analysis

In this section, the simulation results are presented for the original and the proposed approximate deep learning models for the plaintext input data. Overall, our results show that using the large precision parameter \( \alpha \) obtains high performance in the proposed approximate deep learning model, which confirms Theorem 4. All simulations are proceeded by NVIDIA GeForce RTX 3090 GPU along with AMD Ryzen 9 5950X 16-Core Processor CPU.

### 5.1 Numerical analysis on the CIFAR-10

**Pre-trained backbone models** We use the CIFAR-10 [9] as the input data, which has 50k training images and 10k test images with 10 classes. We use ResNet and VGGNet with batch normalization as backbone models of the proposed approximate deep learning model. To verify that the proposed approximate deep learning models work well for a various number of layers, we use ResNet with 20, 32, 44, 56, and 110 layers and VGGNet with 11, 13, 16, and 19 layers proposed in [14] and [15], respectively. In order to obtain pre-trained parameters for each model, we apply the optimizer suggested in [14] on both ResNet and VGGNet for the CIFAR-10.

**Approximation range** The approximation range \([-B, B]\) should be determined in the proposed approximate deep learning model. We examine the input values of the approximate polynomials of the ReLU and max-pooling functions and determine the value of \( B \) by adding some margin to the maximum of the input values. For the CIFAR-10, we choose \( B = 50 \) for both the approximate
We also analyze the classification accuracy of the proposed approximate deep learning model for the plaintext CIFAR-10 dataset. Baseline means the accuracy of the original deep learning model. The accuracies of the approximate deep learning models with a difference less than 5% from the baseline accuracy are underlined, and less than 1% are underlined with boldface.

| Backbone | Baseline (%) | α = 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|----------|--------------|-------|---|---|----|----|----|----|----|
| ResNet-20 | 88.36 | 10.00 | 10.00 | 10.00 | 19.96 | 80.91 | 86.44 | 88.01 | 88.41 |
| ResNet-32 | 89.38 | 10.00 | 10.00 | 10.00 | 36.16 | 84.92 | 88.08 | 88.99 | 89.23 |
| ResNet-44 | 89.41 | 10.00 | 10.00 | 10.00 | 38.35 | 85.75 | 88.31 | 89.31 | 89.37 |
| ResNet-56 | 89.72 | 10.00 | 10.00 | 9.94 | 39.48 | 86.54 | 89.18 | 89.53 | 89.67 |
| ResNet-110 | 89.87 | 10.00 | 10.00 | 10.00 | 10.04 | 67.02 | 86.75 | 89.17 | 89.62 | 89.84 |
| VGG-11 | 90.17 | 10.77 | 11.74 | 15.61 | 29.39 | 66.60 | 85.22 | 88.95 | 89.91 |
| VGG-13 | 91.90 | 11.32 | 11.57 | 14.37 | 36.84 | 77.55 | 88.72 | 90.66 | 91.37 |
| VGG-16 | 91.99 | 10.02 | 10.06 | 10.76 | 32.93 | 78.27 | 89.69 | 91.13 | 91.87 |
| VGG-19 | 91.75 | 10.18 | 13.84 | 18.18 | 43.56 | 80.40 | 89.42 | 91.10 | 91.61 |

Table 3: The top-1 accuracy of each approximate deep learning model with respect to α for the plaintext CIFAR-10 dataset. The inference results of the proposed approximate deep learning models are given in Table 4. Table 4: Obtained the top-1 accuracy of each deep learning model using the same optimizer for pre-training but only for ten epochs and drop learning rate with multiplicative factor 0.1 at the 5th epoch. The results of retraining with precision parameter α = 14 are summarized in Table 5.

| Backbone | Baseline (%) | α = 7 | 8 | 9 | 10 |
|----------|--------------|-------|---|---|----|
| ResNet-20 | 88.36 | 80.28 | 84.83 | 86.76 | 86.61 |
| ResNet-32 | 89.38 | 80.09 | 86.08 | 87.76 | 87.85 |
| ResNet-44 | 89.41 | 76.13 | 82.93 | 85.61 | 88.06 |
| ResNet-56 | 89.72 | 70.90 | 77.59 | 85.06 | 87.96 |
| ResNet-110 | 89.87 | 50.47 | 31.24 | 79.93 | 88.28 |
| VGG-11 | 90.17 | 66.54 | 78.57 | 86.56 | 88.04 |
| VGG-13 | 91.90 | 73.09 | 83.67 | 88.97 | 90.74 |
| VGG-16 | 91.99 | 73.75 | 84.12 | 89.37 | 90.65 |
| VGG-19 | 91.75 | 75.54 | 83.77 | 88.90 | 90.21 |

Table 4: Obtained the top-1 accuracy of each deep learning model with respect to α for the plaintext CIFAR-10 dataset by retraining for ten epochs. The accuracies with difference less than 5% from the baseline accuracy are underlined.

Table 5: The top-1 accuracy of each approximate deep learning model for the plaintext ImageNet dataset with α = 14. The accuracies are underlined and bold-faced in the same manner as in Table 3.

| Backbone | Baseline (%) | Proposed |
|----------|--------------|-----------|
| Inception-v3 | 69.54 | 69.23 |
| GoogLeNet | 68.13 | 69.76 |
| VGG-19 | 74.22 | 73.46 |
| ResNet-152 | 78.31 | 77.52 |

Numerical results First of all, neither the activation functions x^2, suggested in [11] and [3], nor 2^{-3}x^2 + 2^{-1}x + 2^{-2}, suggested in [5], shows good performance in our backbone models. These activation functions give the inference results around 10.0% with our pre-trained parameters. Also, ResNet and VGGNet cannot be trained by the activation functions x^2 and 2^{-3}x^2 + 2^{-1}x + 2^{-2}, since these activation functions cause a more significant gradient exploding problem as the original model gets deeper. Therefore, a polynomial with a small degree is inappropriate to utilize the pre-trained model, which implies the importance of the proposed precise approximate polynomial for PPML in ResNet and VGGNet.

The inference results of the proposed approximate deep learning models are given in Table 3. Table 3 shows that if we adopt a large precision parameter α, the proposed model becomes more accurate, confirming Theorem 4. We achieve the polynomial-operation-only VGG-16 model with 91.87% accuracy for the CIFAR-10 using α = 14. In addition, the proposed approximate deep learning model performs almost similarly (1% difference) to the original deep learning model when α = 12–14.

Moreover, if retraining is allowed for a few epochs, we also obtain high performance in the proposed approximate deep learning model with a smaller precision parameter α. We retrain the proposed model using the same optimizer for pre-training but only for ten epochs and drop learning rate with multiplicative factor 0.1 at the 5th epoch. The results of retraining with precision parameter α = 7–10 are summarized in Table 4. Comparing Tables 3 and 4 shows that the required precision parameter α for high classification accuracy is reduced if we allow retraining for ten epochs.

5.2 Numerical analysis on the ImageNet

We also analyze the classification accuracy of the proposed approximate deep learning model for the ImageNet dataset [17], which has 1.28M training images and 50k validation images with 1000 classes. We simulate various models whose pre-trained parameters are available for inference of...
the ImageNet. For the approximation range $[-B, B]$, we choose $B = 400$ for the ReLU function and $B = 100$ for the max-pooling function. The models that we use and their inference results are summarized in Table 5. Without any training, we easily achieve a polynomial-operation-only model with an accuracy of 77.52% for the ImageNet. From this result, it can be seen that even if the dataset is large, the proposed model shows high performance, similar to the original model.

5.3 Discussion of classification accuracy performance on FHE

In this study, deep learning was performed for plaintext input data, not for the homomorphically encrypted input data. However, since polynomials replaced all non-arithmetic operations, the proposed approximate deep learning model can also efficiently be performed in FHE, and the classification accuracy would be identical. In particular, the proposed approximate deep learning model will work well in the RNS-CKKS scheme [28] because the computational time and precision of RNS-CKKS scheme operations have been improved significantly in recent years [10, 26, 12, 13].

6 Conclusion

We proposed the polynomials that precisely approximate the ReLU and max-pooling functions using a composition of minimax approximate polynomials of small degrees for PPML using FHE. We theoretically showed that if the precision parameter $\alpha$ is large enough, the difference between the inference result of the original and the proposed approximate models becomes small enough. For the CIFAR-10 dataset, the proposed model approximating ResNet or VGGNet had classification accuracy with only 1% error from the original deep learning model when we used the precision parameter $\alpha$ from 12 to 14. Furthermore, for the first time for PPML using word-wise HE, we achieved 77.52% of test accuracy for ImageNet classification, which was close to the original accuracy of 78.31% for the precision parameter $\alpha = 14$. 

10
Appendix

A Coefficients of the approximate polynomial of the sign function, \( p_\alpha(x) \)

In this section, for the approximate polynomial of the sign function, \( p_\alpha(x) \), the coefficients of the component polynomials \( p_{\alpha,1}(x), \cdots, p_{\alpha,k}(x) \) using the Remez algorithm \[24\] are shown for the precision parameter \( \alpha = 7, 8, \cdots, 14 \). Using the coefficients in the table below, we can obtain the proposed approximate polynomial of the ReLU function using the following equation:

\[
r_\alpha(x) = \frac{1}{2}(x + x p_\alpha(x)) = \frac{1}{2}(x + x (p_{\alpha,k} \circ \cdots \circ p_{\alpha,1})(x)).
\]

Note that \( p_{\alpha,i}(x) \) is an odd function theoretically, and the obtained coefficients of even-degree terms of \( p_{\alpha,i}(x) \) by using the Remez algorithm \[24\] are almost zero. Even if we delete the even-degree terms of \( p_{\alpha,i}(x) \) and obtain \( r_\alpha(x) \) using the modified \( p_{\alpha,i}(x) \), the approximate deep learning model using the modified \( r_\alpha(x) \) has almost the same inference results.

| \( \alpha \) | \( i \) | \( m \) | Coefficients of \( x^m \) of \( p_{\alpha,i}(x) \) |
|---|---|---|---|
| 7 | 1 | 0 | 3.60471572275560 \times 10^{-36} |
| | | 1 | 7.30445164958251 |
| | | 2 | -5.05471704202722 \times 10^{-35} |
| | | 3 | -3.46825871108659 \times 10^1 |
| | | 4 | 1.16564665409095 \times 10^{-34} |
| | | 5 | 5.98596518298826 \times 10^1 |
| | | 6 | -6.54298492839531 \times 10^{-35} |
| | | 7 | -3.18755225906466 \times 10^1 |
| 2 | 0 | 3.60471572275560 \times 10^{-36} |
| | | 1 | 7.30445164958251 |
| | | 2 | 6.41744632725342 \times 10^{-49} |
| | | 3 | -2.63125454261783 |
| | | 4 | 7.25383864676814 \times 10^{-48} |
| | | 5 | 1.54912674773593 |
| | | 6 | 2.06916466421812 \times 10^{-48} |
| | | 7 | -3.31172956504304 \times 10^{-1} |
| 8 | 1 | 0 | -5.30489756589578 \times 10^{-48} |
| | | 1 | 8.83133072022416 |
| | | 2 | 2.15841891006552 \times 10^{-46} |
| | | 3 | -4.64575039895512 \times 10^1 |
| | | 4 | -6.58937888826136 \times 10^{-46} |
| | | 5 | 8.30282234720408 \times 10^1 |
| | | 6 | 4.43205802152239 \times 10^{-46} |
| | | 7 | -4.49928477828070 \times 10^1 |
| 2 | 0 | -3.3945726447112 \times 10^{-32} |
| | | 1 | 3.94881885083263 |
| | | 2 | 8.77744308285903 \times 10^{-31} |
| | | 3 | -1.29103010992282 \times 10^1 |
| | | 4 | -3.73356852706615 \times 10^{-30} |
| | | 5 | 2.80865362174658 \times 10^1 |
| | | 6 | 5.59273808588447 \times 10^{-30} |
| | | 7 | -3.55969148965137 \times 10^1 |
| | | 8 | -3.36963375307073 \times 10^{-30} |
| | | 9 | 2.65159370881337 \times 10^1 |
| | | 10 | 5.36813679148778 \times 10^{-31} |
| | | 11 | -1.14184889368449 \times 10^1 |
| | | 12 | 1.91081017684427 \times 10^{-31} |
| | | 13 | 2.62558443881334 |
| | | 14 | -5.50686982942230 \times 10^{-32} |
| | | 15 | -2.49172299998642 \times 10^{-1} |
| 9 | 1 | 0 | 3.851 697 412 341 83 × 10^{-44} |
| 1 | 1.809 662 857 188 07 × 10^{1} |
| 2 | -4.597 304 169 163 77 × 10^{-42} |
| 3 | -4.340 387 032 748 86 × 10^{2} |
| 4 | 7.962 991 603 756 90 × 10^{-41} |
| 5 | 4.154 971 035 456 96 × 10^{3} |
| 6 | -5.289 771 103 963 16 × 10^{-40} |
| 7 | -1.868 469 436 131 49 × 10^{4} |
| 8 | 1.672 195 511 489 17 × 10^{-39} |
| 9 | 4.416 571 778 893 29 × 10^{4} |
| 10 | -2.697 774 247 985 06 × 10^{-39} |
| 11 | -5.655 279 289 834 01 × 10^{4} |
| 12 | 2.141 245 913 835 69 × 10^{-39} |
| 13 | 3.711 561 227 257 81 × 10^{4} |
| 14 | -6.617 224 559 271 98 × 10^{-40} |
| 15 | -9.782 419 338 927 81 × 10^{3} |
| 2 | 0 | -1.045 010 740 638 54 × 10^{-46} |
| 1 | 3.797 533 233 608 56 |
| 2 | 4.228 422 098 180 16 × 10^{-45} |
| 3 | -1.177 181 577 711 92 × 10^{1} |
| 4 | -2.255 711 139 366 39 × 10^{-44} |
| 5 | 2.497 710 866 783 46 × 10^{1} |
| 6 | 4.424 628 751 068 62 × 10^{-44} |
| 7 | -3.152 388 416 039 93 × 10^{1} |
| 8 | -4.135 541 944 116 45 × 10^{-44} |
| 9 | 2.372 948 631 267 22 × 10^{1} |
| 10 | 2.000 601 587 830 94 × 10^{-44} |
| 11 | -1.043 318 001 959 23 × 10^{1} |
| 12 | -4.860 411 327 127 96 × 10^{-45} |
| 13 | 2.467 439 762 608 38 |
| 14 | 4.712 562 140 520 49 × 10^{-46} |
| 15 | -2.421 301 002 476 17 × 10^{-1} |

| 10 | 1 | 0 | -1.080 488 122 385 97 × 10^{-44} |
| 1 | 1.085 418 425 774 42 × 10^{1} |
| 2 | 5.192 134 056 042 61 × 10^{-46} |
| 3 | -6.228 339 252 110 98 × 10^{1} |
| 4 | -1.673 587 150 074 38 × 10^{-45} |
| 5 | 1.143 692 278 204 43 × 10^{2} |
| 6 | 1.154 370 766 923 63 × 10^{-45} |
| 7 | -6.280 234 969 730 74 × 10^{1} |
| 2 | 0 | 7.862 535 624 839 70 × 10^{-39} |
| 1 | 4.139 761 709 851 11 |
| 2 | -7.182 417 416 499 40 × 10^{-38} |
| 3 | -5.849 976 402 116 79 |
| 4 | 5.178 786 344 427 82 × 10^{-38} |
| 5 | 2.943 762 556 592 80 |
| 6 | -9.330 597 439 600 49 × 10^{-39} |
| 7 | -4.545 304 374 601 52 × 10^{-1} |
| 3 | 0 | 3.753 741 535 832 92 × 10^{-39} |
| 1 | 3.299 567 390 437 33 |
| 2 | -1.045 371 340 020 89 × 10^{-37} |
| 3 | -7.842 272 602 913 55 |
| 4 | 4.186 476 959 842 31 × 10^{-37} |
| 5 | 1.289 077 641 155 64 × 10^{1} |
| 6 | -6.095 101 595 408 55 × 10^{-37} |
| 7 | -1.249 171 125 844 86 × 10^{1} |
|   | 8      | 9      | 10                     | 11                     | 12                     | 13                     |
|---|--------|--------|------------------------|------------------------|------------------------|------------------------|
|   | 4.054 754 412 471 24 × 10^{-37} | 6.941 679 914 280 74 | -1.267 700 878 158 48 × 10^{-37} | -2.042 980 673 999 42 | 1.524 521 974 006 36 × 10^{-38} | 2.464 071 389 260 31 × 10^{-1} |
| 11| 7.577 397 399 774 06 × 10^{-34} | 1.125 906 674 029 54 × 10^1 | 3.502 987 364 391 09 × 10^{-29} | -6.546 929 333 299 73 × 10^1 | -1.109 395 299 769 21 × 10^{-28} | 1.206 946 342 777 57 × 10^2 |
| 2  | 6.707 459 348 521 79 × 10^{-49} | 4.704 776 242 108 83 | -3.560 895 946 155 42 × 10^{-48} | -6.798 848 515 966 81 | 1.435 956 460 699 24 × 10^{-48} | 3.315 251 043 828 73 |
| 3  | -3.724 666 663 816 43 × 10^{-46} | 5.363 342 576 549 53 | 4.387 325 688 537 77 × 10^{-44} | -3.551 695 554 419 62 × 10^1 | -7.275 586 270 951 35 × 10^{-43} | 1.778 073 041 156 44 × 10^2 |
|   | 4.675 633 531 474 43 × 10^{-42} | -5.922 973 954 150 24 × 10^2 | -1.559 191 149 483 18 × 10^{-41} | 1.348 916 918 893 62 × 10^3 | 3.101 987 517 898 98 × 10^{-41} | -2.158 764 450 849 38 × 10^3 |
|   | 2.473 656 855 869 18 × 10^3 | 3.442 583 671 739 57 × 10^{-41} | -2.049 135 425 362 48 × 10^3 | -2.054 473 617 911 60 × 10^{-41} | 1.227 393 170 905 59 × 10^3 | 8.496 944 474 386 26 × 10^{-42} |
|   | -5.258 261 751 340 02 × 10^2 | -2.398 563 650 034 46 × 10^{-42} | 1.569 305 587 128 40 × 10^2 | 4.422 234 685 976 95 × 10^{-43} | -3.096 585 956 459 12 × 10^3 | -4.813 314 298 193 61 × 10^{-44} |
| 12| 3.286 712 537 981 58 × 10^{-43} | 1.155 230 423 572 23 × 10^3 | -2.923 425 522 868 17 × 10^{-43} | -6.777 945 134 409 68 × 10^1 | 9.336 595 532 436 19 × 10^{-43} | 1.252 837 404 045 62 × 10^2 |
| 2  | -6.413 195 120 761 87 × 10^{-43} | 1.511 602 837 499 39 × 10^{-4} | 1.511 602 837 499 39 × 10^{-4} | 1.511 602 837 499 39 × 10^{-4} | 1.511 602 837 499 39 × 10^{-4} | 1.511 602 837 499 39 × 10^{-4} |
|   |                                                           |
|---|-----------------------------------------------------------|
| 7 | -6.901 429 082 329 34 × 10^4                               |
| 2 0 | 6.410 873 889 486 33 × 10^{-46}                             |
| 1 | 9.651 676 361 816 26                                       |
| 2 | -1.228 223 295 060 37 × 10^{-43}                            |
| 3 | -6.169 391 745 384 69 × 10^1                               |
| 4 | 6.206 245 663 408 35 × 10^{-43}                             |
| 5 | 1.551 703 516 522 98 × 10^2                                |
| 6 | -9.962 184 919 193 33 × 10^{-43}                            |
| 7 | -1.826 975 823 832 14 × 10^2                               |
| 8 | 7.277 489 682 706 10 × 10^{-43}                             |
| 9 | 1.129 107 265 254 06 × 10^2                                |
| 10 | -2.691 349 248 456 14 × 10^{-43}                            |
| 11 | -3.777 524 117 702 63 × 10^1                               |
| 12 | 4.932 350 887 428 35 × 10^{-44}                             |
| 13 | 6.475 039 097 323 44                                       |
| 14 | -3.568 768 264 589 06 × 10^{-45}                            |
| 15 | -4.456 133 657 233 61 × 10^{-1}                             |
| 3 0 | 4.777 105 763 127 91 × 10^{-47}                             |
| 1 | 5.258 883 555 717 45                                       |
| 2 | -2.945 719 214 383 75 × 10^{-45}                            |
| 3 | -3.372 335 937 942 84 × 10^1                               |
| 4 | 4.432 794 011 328 79 × 10^{-44}                             |
| 5 | 1.649 830 850 134 57 × 10^2                                |
| 6 | -2.768 639 855 195 52 × 10^{-43}                            |
| 7 | -5.414 088 914 069 92 × 10^2                               |
| 8 | 9.151 810 029 942 63 × 10^{-43}                             |
| 9 | 1.222 962 079 979 63 × 10^3                                |
| 10 | -1.820 711 280 479 40 × 10^{-42}                            |
| 11 | -1.952 019 105 664 79 × 10^3                               |
| 12 | 2.348 527 584 557 81 × 10^{-42}                             |
| 13 | 2.240 840 213 783 00 × 10^3                                |
| 14 | -2.051 693 005 032 05 × 10^{-42}                            |
| 15 | -1.866 349 169 831 70 × 10^3                               |
| 16 | 1.241 575 282 208 00 × 10^{-42}                             |
| 17 | 1.127 221 178 431 21 × 10^3                                |
| 18 | -5.228 507 797 773 04 × 10^{-43}                            |
| 19 | -4.880 704 476 383 80 × 10^2                               |
| 20 | 1.509 232 497 138 14 × 10^{-43}                             |
| 21 | 1.474 978 463 089 20 × 10^2                                |
| 22 | -2.857 630 453 646 43 × 10^{-44}                            |
| 23 | -2.951 710 488 795 26 × 10^1                               |
| 24 | 3.207 685 496 547 60 × 10^{-45}                             |
| 25 | 3.512 695 209 309 94                                       |
| 26 | -1.622 639 854 933 95 × 10^{-46}                            |
| 27 | -1.881 018 365 578 79 × 10^{-1}                             |

| 13 |                                                           |
| 1 | 1.345 957 692 399 10 × 10^{-34}                             |
| 2 | 2.455 894 154 250 04 × 10^1                                 |
| 3 | 4.850 956 672 382 42 × 10^{-32}                             |
| 4 | -6.696 604 497 168 94 × 10^2                               |
| 5 | -2.445 412 358 538 40 × 10^{-30}                            |
| 6 | 6.672 998 483 013 39 × 10^3                                |
| 7 | 1.868 748 119 446 40 × 10^{-29}                             |
| 8 | -3.060 366 561 638 98 × 10^4                               |
| 9 | -5.762 278 175 772 42 × 10^{-29}                            |
| 10 | 7.318 840 329 877 87 × 10^4                                |
| 11 | 8.536 806 730 092 59 × 10^{-29}                             |
| 12 | -9.444 332 170 500 84 × 10^4                               |
| 12 | $-6.027\,014\,746\,946\,67 \times 10^{-29}$ |
| 13 | $6.232\,540\,942\,125\,46 \times 10^4$ |
| 14 | $1.623\,428\,436\,619\,40 \times 10^{-29}$ |
| 15 | $-1.649\,467\,441\,178\,05 \times 10^4$ |
| 2  | 0   | $1.532\,615\,885\,856\,30 \times 10^{-47}$ |
| 1  | 9.356\,256\,360\,354\,39 |
| 2  | $-3.689\,721\,230\,482\,49 \times 10^{-46}$ |
| 3  | $-5.916\,389\,639\,336\,26 \times 10^1$ |
| 4  | $1.742\,543\,997\,033\,03 \times 10^{-45}$ |
| 5  | $1.488\,609\,306\,264\,48 \times 10^2$ |
| 6  | $-3.206\,721\,100\,022\,13 \times 10^{-45}$ |
| 7  | $-1.758\,128\,748\,785\,82 \times 10^2$ |
| 8  | $2.791\,157\,389\,486\,45 \times 10^{-45}$ |
| 9  | $1.091\,112\,996\,859\,55 \times 10^2$ |
| 10 | $-1.225\,903\,093\,061\,00 \times 10^{-45}$ |
| 11 | $-3.667\,688\,399\,787\,55 \times 10^3$ |
| 12 | $2.621\,891\,425\,579\,62 \times 10^{-46}$ |
| 13 | $6.318\,462\,903\,112\,94$ |
| 14 | $-2.166\,623\,264\,212\,75 \times 10^{-47}$ |
| 15 | $-4.371\,134\,150\,821\,77 \times 10^{-1}$ |
| 3  | 0   | $6.435\,519\,383\,199\,83 \times 10^{-48}$ |
| 1  | 5.078\,135\,697\,588\,61 |
| 2  | $8.126\,010\,388\,557\,62 \times 10^{-46}$ |
| 3  | $-3.073\,299\,181\,371\,86 \times 10^1$ |
| 4  | $-1.601\,984\,746\,784\,27 \times 10^{-44}$ |
| 5  | $1.441\,097\,468\,128\,09 \times 10^2$ |
| 6  | $1.074\,631\,544\,605\,11 \times 10^{-43}$ |
| 7  | $-4.596\,616\,888\,261\,42 \times 10^2$ |
| 8  | $-3.634\,487\,230\,445\,12 \times 10^{-43}$ |
| 9  | $1.021\,520\,644\,704\,59 \times 10^3$ |
| 10 | $7.252\,071\,253\,697\,84 \times 10^{-43}$ |
| 11 | $-1.620\,562\,567\,088\,77 \times 10^3$ |
| 12 | $-9.273\,063\,978\,536\,55 \times 10^{-43}$ |
| 13 | $1.864\,676\,464\,165\,70 \times 10^3$ |
| 14 | $7.958\,430\,973\,540\,65 \times 10^{-43}$ |
| 15 | $-1.567\,493\,008\,771\,43 \times 10^3$ |
| 16 | $-4.691\,901\,031\,475\,27 \times 10^{-43}$ |
| 17 | $9.609\,703\,090\,934\,22 \times 10^2$ |
| 18 | $1.908\,633\,496\,540\,16 \times 10^{-43}$ |
| 19 | $-4.243\,261\,618\,716\,46 \times 10^2$ |
| 20 | $-5.274\,396\,780\,206\,96 \times 10^{-44}$ |
| 21 | $1.312\,785\,092\,560\,03 \times 10^2$ |
| 22 | $9.470\,449\,379\,747\,86 \times 10^{-45}$ |
| 23 | $-2.698\,125\,766\,261\,15 \times 10^1$ |
| 24 | $-9.981\,815\,617\,637\,50 \times 10^{-46}$ |
| 25 | $3.306\,513\,873\,155\,65$ |
| 26 | $4.693\,904\,661\,921\,99 \times 10^{-47}$ |
| 27 | $-1.827\,429\,446\,275\,33 \times 10^{-1}$ |
| 14 | 1   | $-3.385\,722\,834\,334\,92 \times 10^{-47}$ |
| 1  | $2.490\,521\,431\,937\,54 \times 10^1$ |
| 2  | $7.670\,642\,967\,078\,65 \times 10^{-45}$ |
| 3  | $-6.823\,830\,575\,824\,30 \times 10^2$ |
| 4  | $-1.333\,185\,272\,588\,59 \times 10^{-43}$ |
| 5  | $6.809\,428\,453\,905\,99 \times 10^3$ |
| 6  | $9.194\,645\,680\,020\,43 \times 10^{-43}$ |
| 7  | $-3.125\,071\,000\,171\,05 \times 10^4$ |
| 8  | $-3.025\,478\,830\,899\,49 \times 10^{-42}$ |
|   | 7.476 593 883 637 57 \times 10^4 |
|---|-------------------------------|
| 10 | 5.024 260 275 717 70 \times 10^{-42} |
| 11 | -9.650 468 384 758 39 \times 10^4 |
| 12 | -4.059 312 403 214 43 \times 10^{-42} |
| 13 | 6.369 779 237 782 46 \times 10^4 |
| 14 | 1.266 714 278 278 97 \times 10^{-42} |
| 15 | -1.686 026 213 471 90 \times 10^4 |
|    | -9.279 917 569 679 91 \times 10^{-46} |
| 1   | 1.682 855 119 260 11 \times 10^1 |
| 2   | 8.324 081 146 866 71 \times 10^{-44} |
| 3   | -3.398 117 504 956 59 \times 10^2 |
| 4   | -1.277 565 666 258 11 \times 10^{-42} |
| 5   | 2.790 699 987 938 47 \times 10^3 |
| 6   | 7.701 528 367 291 31 \times 10^{-42} |
| 7   | -1.135 141 515 737 90 \times 10^4 |
| 8   | -2.411 599 188 059 90 \times 10^{-41} |
| 9   | 2.662 300 102 837 45 \times 10^4 |
| 10  | 4.488 070 562 138 74 \times 10^{-41} |
| 11  | -3.938 403 286 619 75 \times 10^4 |
| 12  | -5.348 216 229 722 02 \times 10^{-41} |
| 13  | 3.878 842 303 480 60 \times 10^4 |
| 14  | 4.257 225 027 985 59 \times 10^{-41} |
| 15  | -2.623 953 038 449 88 \times 10^4 |
| 16  | -2.311 466 242 633 47 \times 10^{-41} |
| 17  | 1.236 562 070 165 32 \times 10^4 |
| 18  | 8.585 714 635 337 18 \times 10^{-42} |
| 19  | -4.053 364 600 899 99 \times 10^3 |
| 20  | -2.145 649 403 012 55 \times 10^{-42} |
| 21  | 9.060 428 809 510 87 \times 10^2 |
| 22  | 3.448 033 678 999 92 \times 10^{-43} |
| 23  | -1.316 876 492 082 88 \times 10^2 |
| 24  | -3.217 170 593 366 02 \times 10^{-44} |
| 25  | 1.121 760 790 336 23 \times 10^4 |
| 26  | 1.324 256 004 034 43 \times 10^{-45} |
| 27  | -4.249 380 204 674 71 \times 10^{-1} |
| 3   | 6.728 749 687 165 30 \times 10^{-48} |
| 1   | 5.317 554 976 893 91 |
| 2   | 5.681 992 758 010 86 \times 10^{-46} |
| 3   | -3.543 715 315 315 77 \times 10^1 |
| 4   | -1.351 878 131 554 54 \times 10^{-44} |
| 5   | 1.841 224 413 291 40 \times 10^2 |
| 6   | 1.055 317 662 895 89 \times 10^{-43} |
| 7   | -6.553 868 301 462 53 \times 10^2 |
| 8   | -4.142 665 188 717 60 \times 10^{-43} |
| 9   | 1.638 783 354 280 60 \times 10^3 |
| 10  | 9.630 973 611 663 16 \times 10^{-43} |
| 11  | -2.953 862 370 482 26 \times 10^3 |
| 12  | -1.445 566 884 093 60 \times 10^{-42} |
| 13  | 3.908 064 233 624 18 \times 10^3 |
| 14  | 1.472 650 138 644 85 \times 10^{-42} |
| 15  | -3.834 967 391 651 31 \times 10^3 |
| 16  | -1.047 282 511 696 15 \times 10^{-42} |
| 17  | 2.799 606 547 665 17 \times 10^3 |
| 18  | 5.261 087 287 862 76 \times 10^{-43} |
| 19  | -1.512 862 318 866 92 \times 10^3 |
| 20  | -1.860 839 022 225 46 \times 10^{-43} |
| 21  | 5.961 601 393 400 09 \times 10^2 |
B Proofs of theorems and a lemma

Proof of Theorem[1]

Proof. From the definition, \( p_\alpha(x) \) satisfies the following inequality:

\[
|m_\alpha(a, b) - \max(a, b)| = \frac{|(a + b) + (a - b)p_\alpha(a - b) - \max(a, b)|}{2} \leq 2^{-\alpha},
\]

for \( a, b \in [0, 1] \). Then, for \( x \in [0, 1] \), we have \( |r_\alpha(x) - \text{ReLU}(x)| = |\frac{x + xp_\alpha(x)}{2} - x| = |m_\alpha(x, 0) - \max(x, 0)| \leq 2^{-\alpha} \). In addition, for \( x \in [-1, 0] \), we have \( |r_\alpha(x) - \text{ReLU}(x)| = |\frac{x + xp_\alpha(x)}{2}| = |\frac{-x + xp_\alpha(x)}{2} + x| = |m_\alpha(0, -x) - \max(0, -x)| \leq 2^{-\alpha} \). Thus, we have \( |r_\alpha(x) - \text{ReLU}(x)| \leq 2^{-\alpha} \) for \( x \in [-1, 1] \).

Proof of Theorem[2]

Proof. To prove this theorem, we require two lemmas.

(Lemma (a)) For \( n \in \mathbb{N} \), let \( A \) and \( B \) satisfy

\[
([\log_2 n] - 1)2^{-\alpha} \leq A < B \leq 1 - ([\log_2 n] - 1)2^{-\alpha}.
\]

Then, for \( x_1, x_2, \ldots, x_n \in [A, B] \), the following is satisfied:

\[
M_{\alpha,n}(x_1, \ldots, x_n) \in [A - [\log_2 n]2^{-\alpha}, B + [\log_2 n]2^{-\alpha}].
\]

(Proof) We will use mathematical induction to show that Lemma (a) holds for all \( n \in \mathbb{N} \). For \( n = 1 \), it is trivial because \( M_{\alpha,n}(x_1, \ldots, x_n) \in [A - [\log_2 n]2^{-\alpha}, B + [\log_2 n]2^{-\alpha}] \) if and only if \( x_1 \in [A, B] \).

For \( n = 2 \), we have \( 0 \leq A < B \leq 1 \). We have to show that \( M_{\alpha,2}(x_1, x_2) = m_\alpha(x_1, x_2) \in [A - 2^{-\alpha}, B + 2^{-\alpha}] \) for \( x_1, x_2 \in [A, B] \). We note that

\[
|m_\alpha(a, b) - \max(a, b)| \leq 2^{-\alpha} \quad \text{for } a, b \in [0, 1].
\]

Because \( |m_\alpha(x_1, x_2) - \max(x_1, x_2)| \leq 2^{-\alpha} \), we have

\[
-2^{-\alpha} + A \leq m_\alpha(x_1, x_2) - \max(x_1, x_2) + A \\
\leq m_\alpha(x_1, x_2) - \max(x_1, x_2) + \max(x_1, x_2) \\
= m_\alpha(x_1, x_2).
\]

Also, we have

\[
m_\alpha(x_1, x_2) = m_\alpha(x_1, x_2) - \max(x_1, x_2) + \max(x_1, x_2) \\
\leq 2^{-\alpha} + \max(x_1, x_2) \\
\leq A - 2^{-\alpha} + B.
\]

Thus, Lemma (a) holds for \( n = 2 \). Now, we assume that Lemma (a) holds for \( n, 1 \leq n \leq m - 1 \) for some \( m \geq 3 \). It is enough to show that Lemma (a) also holds for \( n = m \).

(i) \( m = 2k \)

We have

\[
([\log_2 2k] - 1)2^{-\alpha} \leq A < B \leq 1 - ([\log_2 2k] - 1)2^{-\alpha},
\]

\[
\begin{array}{|c|c|c|c|c|c|c|}
\hline
22 & 4.536 441 101 994 68 \times 10^{-44} \\
23 & -1.663 217 393 029 58 \times 10^{2} \\
24 & -7.257 822 876 553 13 \times 10^{-45} \\
25 & 3.109 883 697 398 84 \times 10^{1} \\
26 & 6.858 005 206 344 85 \times 10^{-46} \\
27 & -3.493 493 745 061 90 \\
28 & -2.898 498 112 066 37 \times 10^{-47} \\
29 & 1.781 421 569 564 95 \times 10^{-1} \\
\hline
\end{array}
\]
which is equivalent to \((\lceil \log_2 k \rceil)2^{-\alpha} \leq A < B \leq 1 - (\lceil \log_2 k \rceil)2^{-\alpha}\). Then, we have to show that

\[
M_{\alpha,2k}(x_1, \ldots, x_{2k}) \in [A - \lceil \log_2 2k \rceil 2^{-\alpha}, B + \lceil \log_2 2k \rceil 2^{-\alpha}],
\]

for \(x_1, \ldots, x_{2k} \in [A, B]\). Because Lemma (a) holds for \(n = k\) by the intermediate induction assumption, we have

\[
M_{\alpha,k}(x_1, \ldots, x_k) \in [A - \lceil \log_2 k \rceil 2^{-\alpha}, B + \lceil \log_2 k \rceil 2^{-\alpha}]
\]

and

\[
M_{\alpha,k}(x_{k+1}, \ldots, x_{2k}) \in [A - \lceil \log_2 k \rceil 2^{-\alpha}, B + \lceil \log_2 k \rceil 2^{-\alpha}].
\]

From the inequality in (8), we have \(0 \leq A - \lceil \log_2 k \rceil 2^{-\alpha} \leq 1\). Thus, we have

\[
M_{\alpha,k}(x_1, \ldots, x_k), M_{\alpha, k}(x_{k+1}, \ldots, x_{2k}) \in [A - \lceil \log_2 k \rceil 2^{-\alpha}, B + \lceil \log_2 k \rceil 2^{-\alpha}] \subseteq [0, 1].
\]

Then, from Lemma (a) for \(n = 2\), we have

\[
M_{\alpha,2k}(x_1, \ldots, x_{2k}) = m_\alpha(M_{\alpha,k}(x_1, \ldots, x_k), M_{\alpha,k}(x_{k+1}, \ldots, x_{2k}))
\]

\[
\in [A - \lceil \log_2 k \rceil 2^{-\alpha} - 2^{-\alpha}, B + \lceil \log_2 k \rceil 2^{-\alpha} + 2^{-\alpha}]
\]

\[
= [A - \lceil \log_2 2k \rceil 2^{-\alpha}, B + \lceil \log_2 2k \rceil 2^{-\alpha}]
\]

Thus, Lemma (a) holds for \(n = m = 2k\).

(ii) \(m = 2k + 1\)

We have

\[
(\lceil \log_2 (2k + 1) \rceil - 1)2^{-\alpha} \leq A < B \leq 1 - (\lceil \log_2 (2k + 1) \rceil - 1)2^{-\alpha}.
\]

This is equivalent to

\[
(\lceil \log_2 (k + 1) \rceil)2^{-\alpha} \leq A < B \leq 1 - (\lceil \log_2 (k + 1) \rceil)2^{-\alpha}
\]  

(9)

because \(\lceil \log_2 (2k + 1) \rceil = \lceil \log_2 (2k + 2) \rceil\) for every integer \(k \geq 1\). Then, we have to show that

\[
M_{\alpha,2k+1}(x_1, \ldots, x_{2k+1}) \in [A - \lceil \log_2 (2k + 1) \rceil 2^{-\alpha}, B + \lceil \log_2 (2k + 1) \rceil 2^{-\alpha}],
\]

for \(x_1, \ldots, x_{2k+1} \in [A, B]\). Because Lemma (a) holds for \(n = k\) and \(n = k + 1\) by the intermediate induction assumption, we have

\[
M_{\alpha,k}(x_1, \ldots, x_k) \in [A - \lceil \log_2 k \rceil 2^{-\alpha}, B + \lceil \log_2 k \rceil 2^{-\alpha}]
\]

and

\[
M_{\alpha,k+1}(x_{k+1}, \ldots, x_{2k+1}) \in [A - \lceil \log_2 (k + 1) \rceil 2^{-\alpha}, B + \lceil \log_2 (k + 1) \rceil 2^{-\alpha}]
\]

From the inequality in (9) by the intermediate induction assumption, we have \(0 \leq A - \lceil \log_2 (k + 1) \rceil 2^{-\alpha} \leq 1\). Thus, we have

\[
M_{\alpha,k}(x_1, \ldots, x_k), M_{\alpha,k+1}(x_{k+1}, \ldots, x_{2k+1})
\]

\[
\in [A - \lceil \log_2 (k + 1) \rceil 2^{-\alpha}, B + \lceil \log_2 (k + 1) \rceil 2^{-\alpha}]
\]

\[
\subseteq [0, 1].
\]

Then, from Lemma (a) for \(n = 2\), we have

\[
M_{\alpha,2k+1}(x_1, \ldots, x_{2k+1}) = m_\alpha(M_{\alpha,k}(x_1, \ldots, x_k), M_{\alpha,k+1}(x_{k+1}, \ldots, x_{2k+1}))
\]

\[
\in [A - \lceil \log_2 (k + 1) \rceil 2^{-\alpha} - 2^{-\alpha}, B + \lceil \log_2 (k + 1) \rceil 2^{-\alpha} + 2^{-\alpha}]
\]

\[
= [A - \lceil \log_2 (2k + 2) \rceil 2^{-\alpha}, B + \lceil \log_2 (2k + 2) \rceil 2^{-\alpha}]
\]

\[
= [A - \lceil \log_2 (2k + 1) \rceil 2^{-\alpha}, B + \lceil \log_2 (2k + 1) \rceil 2^{-\alpha}]
\]

Thus, Lemma (a) holds for \(n = m = 2k + 1\). Then, Lemma (a) holds for all \(n \geq 1\) by mathematical induction.

(Lemma (b)) For \(a, b, c, d \in \mathbb{R}\), we have

\[
|\max(a, b) - \max(c, d)| \leq \max(|a - c|, |b - d|)
\]

(10)

(Proof) Let \(\max(a, b) = a\) without loss of generality. We denote the left-hand side and right-hand side of the inequality in (10) by LHS and RHS, respectively.
1. \( \max(c, d) = c \)
   We have \( \text{LHS} = |a - c| \). Thus, the lemma holds in this case.

2. \( \max(c, d) = d \)
   We have \( \text{LHS} = |a - d| \).
   
   (a) \( a \geq d \)
   We have \( a \geq d \geq c \). Thus, we have \( \text{LHS} = |a - d| \leq |a - c| \leq \text{RHS} \).

   (b) \( a < d \)
   We have \( b \leq a < d \). Thus, we have \( \text{LHS} = |a - d| \leq |b - d| \leq \text{RHS} \).

Thus, the lemma (b) is proved. \( \square \)

Now, we prove the theorem using Lemma (a) and Lemma (b). We use mathematical induction to show that the inequality in (5) holds for all \( n \in \mathbb{N} \). First, for \( n = 1 \), we have
\[
|M_{\alpha,1}(x_1) - \max(x_1)| = |x_1 - x_1| = 0 = 2^{-\alpha\lceil \log_2 1 \rceil}.
\]
Therefore, the inequality in (5) holds for \( n = 1 \). We assume that the inequality holds for all \( n \), \( 1 \leq n \leq m - 1 \). Then, it is enough to show that the inequality in (5) also holds for \( n = m \). Suppose that \( x_1, \ldots, x_m \in \{\lfloor \log_2 m \rfloor - 1\}2^{-\alpha}, 1 - (\lfloor \log_2 m \rfloor - 1)2^{-\alpha} \).

(i) \( m = 2k \)
We have to show that
\[
|M_{\alpha,2k}(x_1, \ldots, x_{2k}) - \max(x_1, \ldots, x_{2k})| \leq 2^{-\alpha\lceil \log_2 2k \rceil}
\]
for \( x_1, \ldots, x_{2k} \in \{\lfloor \log_2 2k \rfloor - 1\}2^{-\alpha}, 1 - (\lfloor \log_2 2k \rfloor - 1)2^{-\alpha} \).

We have \( M_{\alpha,2k}(x_1, \ldots, x_{2k}) = m_{\alpha}(M_{\alpha,k}(x_1, \ldots, x_k), M_{\alpha,k}(x_{k+1}, \ldots, x_{2k})) \).

Let \( P = \max(x_1, \ldots, x_k), Q = \max(x_{k+1}, \ldots, x_{2k}), \tilde{P} = M_{\alpha,k}(x_1, \ldots, x_k), \) and \( \tilde{Q} = M_{\alpha,k}(x_{k+1}, \ldots, x_{2k}) \). Since
\[
x_1, \ldots, x_{2k} \in \{\lfloor \log_2 2k \rfloor - 1\}2^{-\alpha}, 1 - (\lfloor \log_2 2k \rfloor - 1)2^{-\alpha} \]
\[
\subseteq \{\lfloor \log_2 k \rfloor - 1\}2^{-\alpha}, 1 - (\lfloor \log_2 k \rfloor - 1)2^{-\alpha},
\]
we can apply the intermediate induction assumption for \( n = k \) as
\[
|\tilde{P} - P| = |M_{\alpha,k}(x_1, \ldots, x_k) - \max(x_1, \ldots, x_k)| \leq 2^{-\alpha\lceil \log_2 k \rceil},
\]
\[
|\tilde{Q} - Q| = |M_{\alpha,k}(x_{k+1}, \ldots, x_{2k}) - \max(x_{k+1}, \ldots, x_{2k})| \leq 2^{-\alpha\lceil \log_2 k \rceil}.
\]
The left-hand side of the inequality in (5) for \( n = m \) becomes \( |m_{\alpha}(\tilde{P}, \tilde{Q}) - \max(P, Q)| \). From Lemma (a) for \( n = k \), we have \( \tilde{P}, \tilde{Q} \in [0, 1] \). Then, we have
\[
|m_{\alpha}(\tilde{P}, \tilde{Q}) - \max(P, Q)| \leq |m_{\alpha}(\tilde{P}, \tilde{Q}) - \max(\tilde{P}, \tilde{Q})| + |\max(\tilde{P}, \tilde{Q}) - \max(P, Q)|
\]
\[
\leq |m_{\alpha}(\tilde{P}, \tilde{Q}) - \max(\tilde{P}, \tilde{Q})| + \max(|\tilde{P} - P|, |\tilde{Q} - Q|) \quad \text{(from Lemma (b))}
\]
\[
\leq 2^{-\alpha} + \max(|\tilde{P} - P|, |\tilde{Q} - Q|) \quad \text{(from (7))}
\]
\[
\leq 2^{-\alpha} + 2^{-\alpha\lceil \log_2 k \rceil}
\]
\[
= 2^{-\alpha}[\log_2 2k] = 2^{-\alpha}[\log_2 m].
\]

(ii) \( m = 2k + 1 \)
We have to show that
\[
|M_{\alpha,2k+1}(x_1, \ldots, x_{2k+1}) - \max(x_1, \ldots, x_{2k+1})| \leq 2^{-\alpha\lceil \log_2 (2k + 1) \rceil}
\]
for \( x_1, \ldots, x_{2k+1} \in \{\lfloor \log_2 (2k + 1) \rfloor - 1\}2^{-\alpha}, 1 - (\lfloor \log_2 (2k + 1) \rfloor - 1)2^{-\alpha} \).

We have \( M_{\alpha,2k+1}(x_1, \ldots, x_{2k+1}) = m_{\alpha}(M_{\alpha,k}(x_1, \ldots, x_k), M_{\alpha,k+1}(x_{k+1}, \ldots, x_{2k+1})) \).
Let $P = \max(x_1, \ldots, x_k)$, $Q = \max(x_{k+1}, \ldots, x_{2k+1})$, $\hat{P} = M_{a,k}(x_1, \ldots, x_k)$, and $\hat{Q} = M_{a,k+1}(x_{k+1}, \ldots, x_{2k+1})$. Since
\[
x_1, \ldots, x_k \in [(\lceil \log_2(k + 1) \rceil - 1)2^{-a}, 1 - (\lceil \log_2(k + 1) \rceil - 1)2^{-a}]
\subseteq [(\lceil \log_2 k \rceil - 1)2^{-a}, 1 - (\lceil \log_2 k \rceil - 1)2^{-a}]
\quad \text{and}
\]
x_{k+1}, \ldots, x_{2k+1} \in [(\lceil \log_2(k + 1) \rceil - 1)2^{-a}, 1 - (\lceil \log_2(k + 1) \rceil - 1)2^{-a}]
\subseteq [(\lceil \log_2(k + 1) \rceil - 1)2^{-a}, 1 - (\lceil \log_2(k + 1) \rceil - 1)2^{-a}]
\]
we can apply the induction assumption for $n = k$ and $n = k + 1$ as
\[
|\hat{P} - P| = |M_{a,k}(x_1, \ldots, x_k) - \max(x_1, \ldots, x_k)| \leq 2^{-a} \lfloor \log_2 k \rfloor,
\]
\[
|\hat{Q} - Q| = |M_{a,k+1}(x_{k+1}, \ldots, x_{2k+1}) - \max(x_{k+1}, \ldots, x_{2k+1})| \leq 2^{-a} \lfloor \log_2(k + 1) \rfloor.
\]
The left-hand side of the inequality in (5) for $n = m$ becomes $|m_n(\hat{P}, \hat{Q}) - \max(P, Q)|$. From Lemma (a) for $n = k$ and $n = k + 1$, we have $\hat{P}, \hat{Q} \in [0, 1]$. Then, we have
\[
|m_n(\hat{P}, \hat{Q}) - \max(P, Q)| \leq |m_n(\hat{P}, \hat{Q}) - \max(\hat{P}, \hat{Q})| + |\max(\hat{P}, \hat{Q}) - \max(P, Q)|
\]
\[
\leq |m_n(\hat{P}, \hat{Q}) - \max(\hat{P}, \hat{Q})| + \max(|\hat{P} - P|, |\hat{Q} - Q|) \quad \text{(from Lemma (b))}
\]
\[
\leq 2^{-a} + \max(|\hat{P} - P|, |\hat{Q} - Q|) \quad \text{(from (7))}
\]
\[
\leq 2^{-a} + 2^{-a} \lfloor \log_2(k + 1) \rfloor = 2^{-a} \lfloor \log_2(2k + 2) \rfloor = 2^{-a} \lfloor \log_2(2k + 1) \rfloor = 2^{-a} \lfloor \log_2 m \rfloor
\]
since $\lfloor \log_2(2k + 2) \rfloor = \lfloor \log_2(2k + 1) \rfloor$ for every integer $k \geq 1$. Thus, the inequality in (5) holds for $n = m$, and the theorem is proved by mathematical induction. 

**Proof of Theorem (3)**

**Proof.** Equivalently, we have to show that
\[
\|A^0_0 \circ \cdots \circ A^0_n(x) - A_0 \circ \cdots \circ A_0(x)\|_\infty \leq (E^a_{A_n} \circ E^a_{A_{n-1}} \circ \cdots \circ E^a_{A_0})(0).
\]
(11)
We will prove it by mathematical induction. First, we will prove for $n = 0$.
\[
\|A^0_0(x) - A_0(x)\|_\infty \leq \sup_{\|x\|_\infty \leq B} \|A^0_0(x) - A_0(x)\|_\infty
\]
\[
= \sup_{\|x + e\|_\infty \leq B, \|e\|_\infty \leq 0} \|A^0_0(x + e) - A_0(x)\|_\infty = E^a_{A_0}(0).
\]
Thus, the inequality holds for $n = 0$. Next, we assume that the inequality in (11) holds for $n = k$, that is,
\[
\|A^0_k \circ \cdots \circ A^0_0(x) - A_k \circ \cdots \circ A_0(x)\|_\infty \leq (E^a_{A_k} \circ E^a_{A_{k-1}} \circ \cdots \circ E^a_{A_0})(0)
\]
for some $k \geq 0$. It is enough to show that the inequality in (11) also holds for $n = k + 1$, that is,
\[
\|A^0_{k+1} \circ \cdots \circ A^0_0(x) - A_{k+1} \circ \cdots \circ A_0(x)\|_\infty \leq (E^a_{A_{k+1}} \circ E^a_{A_k} \circ \cdots \circ E^a_{A_0})(0).
\]
We have
\[
\|A^0_{k+1} \circ \cdots \circ A^0_0(x) - A_{k+1} \circ \cdots \circ A_0(x)\|_\infty
\]
\[
= \|A^0_{k+1}(A_k \circ \cdots \circ A_0(x)) + (A_k \circ \cdots \circ A_0(x) - A_k \circ \cdots \circ A_0(x))\|_\infty - A_{k+1} \circ \cdots \circ A_0(x)\|_\infty.\]
Let $x' = A_k \circ \cdots \circ A_0(x)$ and $e' = A^0_k \circ \cdots \circ A^0_0(x) - A_k \circ \cdots \circ A_0(x)$. Because $\|x' + e'\|_\infty = \|A^0_k \circ \cdots \circ A^0_0(x)\|_\infty \leq B$, we have
\[
\|A^0_{k+1} \circ \cdots \circ A^0_0(x) - A_{k+1} \circ \cdots \circ A_0(x)\|_\infty = \|A^0_{k+1}(x' + e') - A_{k+1}(x')\|_\infty
\]
\[
\leq \sup_{\|x + e\|_\infty \leq B, \|e\|_\infty \leq \|e'\|_\infty} \|A^0_{k+1}(x + e) - A_{k+1}(x)\|_\infty
\]
\[
= E^a_{A_{k+1}}(\|e'\|_\infty)
\]
\[
= E^a_{A_{k+1}}(\|A^0_k \circ \cdots \circ A^0_0(x) - A_k \circ \cdots \circ A_1(x)\|_\infty)
\]
\[
\leq E^a_{A_{k+1}}(E^a_{A_k} \circ E^a_{A_{k-1}} \circ \cdots \circ E^a_{A_0})(0) \quad (\because E^a_{A_{k+1}}(\cdot) : \text{increasing})
\]
\[
= E^a_{A_{k+1}} \circ E^a_{A_k} \circ \cdots \circ E^a_{A_0}(0).
\]

20
Thus, the inequality in \([\textbf{11}]\) holds for \(n = k + 1\), and the theorem is proved by mathematical induction.

\[\square\]

**Proof of Lemma \([\textbf{11}]\)**

**Proof.** (a) For \(A(x) = Ax + b\), \(A^\alpha = A\). Then \(A^\alpha(x + e) - A(x) = Ae\). Therefore,

\[
E^\alpha_A(e) = \sup_{\|e\|_\infty \leq e} \|A\|_\infty \leq \sup_{\|e\|_\infty \leq e} \|A\|_\infty \|e\|_\infty = \|A\|_\infty e.
\]

(b) For \(A(x) = \text{ReLU}(x)\), \(A^\alpha(x) = \tilde{r}_{\alpha,B}(x)\). Therefore,

\[
E^\alpha_A(e) = \sup_{\|x + e\|_\infty \leq B, \|e\|_\infty \leq e} \|\tilde{r}_{\alpha,B}(x + e) - \text{ReLU}(x)\|_\infty \leq \sup_{\|x + e\|_\infty \leq B, \|e\|_\infty \leq e} (\|\tilde{r}_{\alpha,B}(x + e) - \text{ReLU}(x + e)\|_\infty + \|\text{ReLU}(x + e) - \text{ReLU}(x)\|_\infty) \leq \sup_{\|x + e\|_\infty \leq B, \|e\|_\infty \leq e} (B \cdot 2^{-\alpha} + \|e\|_\infty) = B \cdot 2^{-\alpha} + e.
\]

(c) If \(A\) is a max-pooling block with kernel size \(k_0\), \(\sup_{\|x + e\|_\infty \leq B, \|e\|_\infty \leq e} \|A^\alpha(x + e) - A(x)\|_\infty\) becomes

\[
\sup_{\|x + e\|_\infty \leq B, \|e\|_\infty \leq e} \|\tilde{M}_{\alpha,k_0,B}(x_1 + e_1, \cdots, x_{k_0} + e_{k_0}) - \max(x_1, \cdots, x_{k_0})\|
\]

where \(x = (x_1, \cdots, x_{k_0})\) and \(e = (e_1, \cdots, e_{k_0})\). Similar technique of proof for (b) and from Theorem \([\textbf{2}]\) we have \(E^\alpha_A(e) \leq B' \frac{\log k_0^2}{2} 2^{-\alpha} + e\), where \(B' = B/(0.5 - (\log k_0^2) - 1)2^{-\alpha}\). Since \(k_0 \leq 10\) and \(\alpha \geq 4\), \(B' < 10B\), which leads the conclusion.

(d) Let us denote softmax block \(A\) as an \(\mathbb{R}^N \rightarrow \mathbb{R}^N\) function with \(A(x) = (\frac{\exp(x_i)}{\sum_j \exp(x_j)})_{1 \leq i \leq N}\) for \(x = (x_1, \cdots, x_N)\). Then mean value theorem for multivariate variable functions gives

\[
\|A(x') - A(x)\|_\infty \leq \sup_{z \in [x', x]} \|J(z)\|_\infty \cdot \|x' - x\|_\infty,
\]

where \(J(z)\) denotes the Jacobian matrix of softmax block \(A\). For a given vector \(z = (z_1, \cdots, z_N) \in \mathbb{R}^N\), the infinity norm of \(J(z)\) is given as \(\|J(z)\|_\infty = \max_i \sum_k \left| \frac{\partial}{\partial z_k} \sum_j \exp(z_j) \right| \). Note that

\[
\left| \frac{\partial}{\partial z_k} \sum_j \exp(z_j) \right| = \begin{cases} \frac{\exp(z_i) \exp(z_k)}{\left(\sum_j \exp(z_j)\right)^2}, & k \neq i \\ \frac{\exp(z_i) \sum_{j \neq i} \exp(z_j)}{\left(\sum_j \exp(z_j)\right)^2}, & k = i \end{cases}
\]

and thus

\[
\sum_k \left| \frac{\partial}{\partial z_k} \sum_j \exp(z_j) \right| = \frac{\exp(z_i) \sum_{k \neq i} \exp(z_k)}{\left(\sum_j \exp(z_j)\right)^2} + \frac{\exp(z_i) \sum_{j \neq i} \exp(z_j)}{\left(\sum_j \exp(z_j)\right)^2} = 2p_i(1 - p_i),
\]

where \(p_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}\). Therefore, \(\|J(z)\|_\infty \leq 2p_i(1 - p_i) \leq 1/2\) for any vector \(z \in \mathbb{R}^N\), and this gives

\[
\|A^\alpha(x + e) - A(x)\|_\infty = \|A(x + e) - A(x)\|_\infty \leq \|e\|_\infty/2,
\]

which leads the conclusion. \(\square\)
Proof of Theorem 4

Proof. From Theorem 3 it is enough to show that
\[
(E^\alpha_{A_n} \circ \cdots \circ E^\alpha_{A_0})(0) \leq C 2^{-\alpha}
\]  
for some constant $C$ and $n \geq 0$. To show that this inequality holds for all $n \geq 0$, we use mathematical induction. First, for $n = 0$, assume that $F$ has one block $A$. The upper bounds of $E^\alpha_A(0)$ for the four basic blocks suggested in Lemma 1 have forms of $C_0 \cdot 2^{-\alpha}$, where $C_0$ can be zero. Thus, the inequality in (12) holds for some constant $C$ and $n = 0$.

Then, we assume that the inequality in (12) holds for $n = k$ and some constant $C_k$, that is, $(E^\alpha_{A_k} \circ \cdots \circ E^\alpha_{A_0})(0) \leq C_k 2^{-\alpha}$ for some $C_k$. Then, it is enough to show that the inequality in (12) holds for $n = k + 1$ and some constant $C_{k+1}$, $E^\alpha_{A_{k+1}}(C_k 2^{-\alpha})$ is not greater than

(i) $\|A\|_{\infty} C_k 2^{-\alpha}$ when $A_{k+1}$ is a linear block with $A(x) = Ax + b$,

(ii) $(B + C_k) 2^{-\alpha}$ when $A_{k+1}$ is a ReLU block,

(iii) $(10B[\log_2 k_0^2] + C_k) 2^{-\alpha}$ when $A_{k+1}$ is a max-pooling block with kernel size $k_0$,

(iv) $\frac{1}{2} C_k 2^{-\alpha}$ when $A_{k+1}$ is a softmax block

by Lemma 1. For each case, we can determine a constant $C_{k+1}$ that satisfies $(E^\alpha_{A_{k+1}} \circ \cdots \circ E^\alpha_{A_0})(0) \leq E^\alpha_{A_{k+1}}(C_k 2^{-\alpha}) \leq C_{k+1} 2^{-\alpha}$. Thus, the theorem is proved.

C Generalization of Theorem 4 for ResNet model

ResNet model cannot be decomposed of basic blocks, since it contains a “residual block”. The authors of [14] suggest an operation $R$ that satisfies
\[
R(x) = \mathcal{G}(x) + Px,
\]
where $\mathcal{G}(x)$ denotes the residual mapping which will be learned, and $P$ is a linear projection matrix to match the dimension of $\mathcal{G}(x)$. In this section, we call such operation $R$ as a residual block. In the residual block designed in ResNet, all residual mapping $\mathcal{G}(x)$ is a composition of basic blocks [14]. Therefore, by Theorem 4 we can determine a constant $C_\mathcal{G}$ that satisfies $\|\mathcal{G}^\alpha(x) - \mathcal{G}(x)\|_{\infty} \leq C_{\mathcal{G}} 2^{-\alpha}$.

In case of $P$, there are two methods of constructing projection in ResNet: the first one pads extra zeros, and the second one uses $1 \times 1$ convolution [14]. We note that both methods can be considered as linear blocks, and thus the approximate block $R^\phi(x)$ can be represented by $\mathcal{G}^\alpha(x) + Px$. Considering this situation, we generalize the original Theorem 4 so that it is also valid for ResNet model.

Theorem 5 (generalized version of Theorem 4). If $\alpha \geq 4$ and a deep learning model $F$ contains only basic blocks and residual blocks, then there exists a constant $C$ such that $\|F^\alpha(x) - F(x)\|_{\infty} \leq C 2^{-\alpha}$ for every input $x$, where the constant $C$ can be determined only by model parameters.

Proof. We prove this statement by obtaining error propagation function $E^\alpha_{R}(e)$ of residual block $R(x) = \mathcal{G}(x) + Px$. From the definition of error propagation function, we have
\[
E^\alpha_{R}(e) = \sup_{\|x+e\|_{\infty} \leq B, \|e\|_{\infty} \leq e} \|R^\alpha(x + e) - R(x)\|_{\infty}
\]
\[
\leq \sup_{\|x+e\|_{\infty} \leq B, \|e\|_{\infty} \leq e} (\|R^\alpha(x + e) - R(x + e)\|_{\infty} + \|R(x + e) - R(x)\|_{\infty})
\]
\[
\leq \sup_{\|x+e\|_{\infty} \leq B, \|e\|_{\infty} \leq e} (\|\mathcal{G}^\alpha(x + e) - \mathcal{G}(x + e)\|_{\infty} + \|\mathcal{G}(x + e) - \mathcal{G}(x)\|_{\infty} + \|P\|_{\infty} e)
\]
\[
\leq C_{\mathcal{G}} 2^{-\alpha} + \|P\|_{\infty} e + \sup_{\|x+e\|_{\infty} \leq B, \|e\|_{\infty} \leq e} \|\mathcal{G}(x + e) - \mathcal{G}(x)\|_{\infty},
\]
where $C_{\mathcal{G}}$ is the constant that satisfies $\|\mathcal{G}^\alpha(x) - \mathcal{G}(x)\|_{\infty} \leq C_{\mathcal{G}} 2^{-\alpha}$. To estimate $\|\mathcal{G}(x+e) - \mathcal{G}(x)\|_{\infty}$, we decompose residual mapping $\mathcal{G}$ into $G_k \circ \cdots \circ G_0$, where $G_i$’s are basic blocks. Then, we define
\[
\Delta_A(e) = \sup_{\|x+e\|_{\infty} \leq B, \|e\|_{\infty} \leq e} \|A(x + e) - A(x)\|_{\infty}
\]
for a block $A$ and magnitude of the error, $e$. Then $\Delta_A(e)$ is a non-decreasing function of $e$ for every block $A$. Thus, similar argument in the proof of Theorem 3 shows that

$$\|G(x + e) - G(x)\|_\infty \leq (\Delta_{G_k} \circ \cdots \circ \Delta_{G_0})(\|e\|_\infty)$$

(13)

for every error vector $e$. Also, similar argument in the proof of Lemma 1 shows that

$$\Delta_A(e) \leq \begin{cases} \|A\|_\infty e, & A: \text{linear block, } A(x) = Ax + b \\ e, & A: \text{ReLU} \\ e, & A: \text{max-pooling} \\ e/2 & A: \text{softmax} \end{cases}$$

(14)

for every basic block $A$ and $e \geq 0$. These inequalities for $\Delta_A(e)$ corresponds to the inequalities for $E_A^\alpha(0)$ in Lemma 1 when the precision parameter $\alpha$ goes to infinity. From two inequalities (13) and (14), there exists a constant $C'_G$ that satisfies $\|G(x + e) - G(x)\|_\infty \leq C'_G\|e\|_\infty$ since $G_i$'s are all basic blocks. Therefore, we have

$$E_R^\alpha(e) \leq C'_G 2^{-\alpha} + (\|P\|_\infty + C'_G)e$$

for every $e \geq 0$.

Now, we prove the theorem using mathematical induction. Let $F$ be a deep learning model which can be decomposed into $A_n \circ \cdots \circ A_0$ where $A_k$'s are all basic blocks or residual blocks. First, for $n = 0$, assume that $F$ has one block $A$. The upper bounds of $E_A^\alpha(0)$ for the four basic blocks and residual blocks have forms of $C_0 2^{-\alpha}$, where $C_0$ can be zero. Inductively, for $n = k$, $(E_{A_k}^\alpha \circ \cdots \circ E_{A_0}^\alpha)(0) \leq C_k 2^{-\alpha}$ for some $C_k$. For $n = k + 1$, we can determine a constant $C_{k+1}$ that satisfies $(E_{A_{k+1}}^\alpha \circ \cdots \circ E_{A_0}^\alpha)(0) \leq C_{k+1} 2^{-\alpha}$ when $A_{k+1}$ is a basic block (see the proof of Theorem 4). If $A_{k+1}$ is a residual block with $A_{k+1}(x) = G(x) + Px$ for some residual mapping $G(x)$ and linear projection matrix $P$, then

$$E_{A_{k+1}}^\alpha(C_k 2^{\alpha}) \leq C'_G 2^{-\alpha} + (\|P\|_\infty + C'_G)(C_k 2^{\alpha}) = C_{k+1} 2^{-\alpha}$$

where $C_{k+1} = C'_G + (\|P\|_\infty + C'_G)C_k$ which is independent of $\alpha$. Therefore,

$$(E_{A_{k+1}}^\alpha \circ \cdots \circ E_{A_0}^\alpha)(0) \leq E_{A_{k+1}}^\alpha(C_k 2^{\alpha}) \leq C_{k+1} 2^{-\alpha},$$

which completes the proof.

\[\Box\]

References

[1] 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 International Conference on Machine Learning, pages 201–210. PMLR, 2016.

[2] Chirag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In 27th USENIX Security Symposium, pages 1651–1669. 2018.

[3] Tim van Eeloo, Giorgio Patrini, and Hamish Ivey-Law. Sealion: A framework for neural network inference on encrypted data. arXiv preprint [arXiv:1904.12840] 2019.

[4] Fabian Boemer, Yixing Lao, Rosario Cammarota, and Casimir Wierzynski. nGraph-HE: A graph compiler for deep learning on homomorphically encrypted data. In Proceedings of the 16th ACM International Conference on Computing Frontiers, pages 3–13, 2019.

[5] Edward Chou, Josh Beal, Daniel Levy, Serena Yeung, Albert Haque, and Li Fei-Fei. Faster Cryptonets: Leveraging sparsity for real-world encrypted inference. arXiv preprint [arXiv:1811.09953] 2018.

[6] Brandon Reagen, Woo-Seok Choi, Yeongil Ko, Vincent T Lee, Hsien-Hsin S Lee, Gu-Yeon Wei, and David Brooks. Cheetah: Optimizing and accelerating homomorphic encryption for private inference. In 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA), pages 26–39. IEEE, 2020.

[7] Ahmad Al Badawi, Jin Chao, Jie Lin, Chan Fook Mun, Jun Jie Sim, Benjamin Hong Meng Tan, Xiao Nan, Khin Mi Mi Aung, and Vijay Ramaseshan Chandrasekhar. Towards the AlexNet moment for homomorphic encryption: HCNN, the first homomorphic CNN on encrypted data with GPUs. IEEE Transactions on Emerging Topics in Computing, 2020.
[8] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. Deep neural networks classification over encrypted data. In *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy*, pages 97–108, 2019.

[9] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. *CiteseerX Technical Report, University of Toronto*, 2009.

[10] Wonkyung Jung, Sanggyo Kim, Jung Ho Ahn, Jung Hee Cheon, and Younho Lee. Over 100x faster bootstrapping in fully homomorphic encryption through memory-centric optimization with GPUs. *Cryptol. ePrint Arch., Tech. Rep. 2021/508*, 2021.

[11] Joon-Woo Lee, Eunsang Lee, Yongwoo Lee, Young-Sik Kim, and Jong-Seon No. High-Precision Bootstrapping of RNS-CKKS Homomorphic Encryption Using Optimal Minimax Polynomial Approximation and Inverse Sine Evaluation. In *International Conference on the Theory and Applications of Cryptographic Techniques*, accepted for publication, 2021.

[12] Yongwoo Lee, Joon-Woo Lee, Young-Sik Kim, HyungChul Kang, and Jong-Seon No. High-precision approximate homomorphic encryption by error variance minimization. *Cryptol. ePrint Arch., Tech. Rep. 2020/834*, 2021.

[13] Andrey Kim, Antonis Papadimitriou, and Yuriy Polyakov. Approximate homomorphic encryption with reduced approximation error. *IACR Cryptol. ePrint Arch., 2020*, 2020.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

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

[16] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

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

[18] Qian Lou, and Lei Jiang. She: A fast and accurate deep neural network for encrypted data. In *Proceedings of the Neural Information Processing Systems*, 2019.

[19] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, 2020.

[20] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, 2012:144, 2012.

[21] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 409–437. Springer, 2017.

[22] Eunsang Lee, Joon-Woo Lee, Jong-Seon No, and Young-Sik Kim. Minimax approximation of sign function by composite polynomial for homomorphic comparison. *Cryptol. ePrint Arch., Tech. Rep. 2020/834*, 2020.

[23] Jung Hee Cheon, Dongwoo Kim, and Duhyeong Kim. Efficient homomorphic comparison methods with optimal complexity. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 221–256. Springer, 2020.

[24] Eugene Y Remez. Sur la détermination des polynômes d’approximation de degré donnée. *Comm. Soc. Math. Kharkov*, 10(196):41–63, 1934.

[25] Michael S Paterson and Larry J Stockmeyer. On the number of nonscalar multiplications necessary to evaluate polynomials. *SIAM Journal on Computing*, 2(1):60–66, 1973.

[26] Joon-Woo Lee, Eunsang Lee, Yong-Woo Lee, and Jong-Seon No. Optimal minimax polynomial approximation of modular reduction for bootstrapping of approximate homomorphic encryption. *Cryptol. ePrint Arch., Tech. Rep. 2020/552*, 2020.

[27] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[28] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. A full RNS variant of approximate homomorphic encryption. In *International Conference on Selected Areas in Cryptography*, pages 347–368. Springer, 2018.