IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received July 15, 2021, accepted August 6, 2021, date of publication August 24, 2021, date of current version August 31, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3106888

Secure Image Inference Using Pairwise Activation Functions

JONAS T. AGYEPONG[1], MOSTAFA SOLIMAN[1], YASUTAKA WADA[2], (Member, IEEE), KEIJI KIMURA[3], (Member, IEEE), AND AHMED EL-MAHDY[1], (Senior Member, IEEE)

[1]Computer Science and Engineering Department, Egypt-Japan University of Science and Technology, Alexandria 21934, Egypt
[2]Department of Information Science, Meisei University, Tokyo 191-8506, Japan
[3]Department of Computer Science and Engineering, Waseda University, Tokyo 169-8555, Japan

Corresponding author: Jonas T. Agyepong (jonas.tawiah@ejust.edu.eg)

This work was supported by the JSPS KAKENHI under Grant 18K19786.

ABSTRACT Polynomial approximation has for the past few years been used to derive polynomials as an approximation to activation functions for use in image prediction or inference employing homomorphic encryption technique to induce data privacy and security. Most proposed works thus far have only been limited to deriving very few polynomials to use for these tasks. While the literature has considered forming new activation functions as pairwise multiplication of well-known activation functions, the design space is mostly unexplored. In some practical applications, there is usually a mix of activation functions used, so looking ahead, there is the need to explore into using other potential functions that can also improve performance whiles not relying on a few ones proposed such as ReLU and Swish. This paper explores the design space of such pairwise, multiplied activation functions and their application in homomorphic image inference or prediction using the widely popular MNIST and CIFAR-10 benchmark datasets. Moreover, we analyzed corresponding curve fitting parameters (range and degree), homomorphic-friendly pooling methods, and optimization methods in the ciphertext domain to avoid incurring huge computation costs but not compromising accuracy. Results show new activation function combinations yielding similar or better results in ciphertext as compared to the ones in plaintext.

INDEX TERMS Exploratory analysis, homomorphic encryption scheme, homomorphic image inference, pairwise functions, polynomial approximation, privacy-preserving machine learning.

I. INTRODUCTION

The area of deep learning is continually growing and has thus become a promising research field. The accuracy and throughput of deep neural networks are improving over the years, making them more useful for applications in diverse areas. However, as machine learning algorithms still rely heavily on raw data, it poses a considerable risk to users' security and privacy. As most processes are delegated to very high-performance computing equipment far from the user, it makes the data involved less secure. An alternative solution to this is to encrypt the data to be processed by the deep neural network, where the encrypted data and public key are sent to the server. The server performs the required computations (or say training, classification or prediction) on the encrypted data and yields encrypted results which are sent back to the user. The user then decrypts the results to plaintext. Performing image processing using CNNs is a very complex and slow process, mainly when using more complex models and massive datasets, so mostly Graphical Processing Units (GPUs) are employed to make processing faster. Though there continues to be an improvement in accuracy and run times, performance is still far from satisfactory [7]. Incorporating Fully-Homomorphic Encryption (FHE), which is also a slow process, will further augment the models' complexity, making them even slower in their computation. That means we would have to tune these models when employing FHE to realize them practically, using few parameters or less-complex models, but this can affect accuracy adversely.

Most successfully implemented privacy-preserving deep learning methods use generally less complex datasets (MNIST [14], [15] and CIFAR-10 [3], [4] especially) and employ spatial convolutions in the neural network for image classification. Although they produce good results compared to non-encrypted classification, their high latency, amount of parameters, and computation restrict their use for other more complex computer vision areas employing other forms of datasets. That might not make this area favourable for

The associate editor coordinating the review of this manuscript and approving it for publication was Massimo Cafaro.

researchers who might still opt for machine learning in plaintext space.

Privacy-preserving machine learning research has gained much significant attention over the past decade [31], and as machine learning continues to grow, so does the relevance of the former. The two notable techniques [1], [5]–[9], [31] currently being used in implementation are homomorphic encryption (HE) and multi-party computation (MPC) or sometimes the mix of the two and also using secret sharing techniques. This form of cryptographic technique proposed in literature known as homomorphic encryption allows for performing computation on encrypted data as stated earlier, yielding results equivalent to one where the same type of processing is executed but on raw unsecured data. A newer form of this type of encryption called Fully Homomorphic Encryption (FHE) [29], [30] allows to compute arbitrary functions on encrypted data [2] meaning the broadening of the effectiveness of this encryption scheme to perform several privacy-preserving tasks across many disciplines no matter the level of complexity and diversity involved in such applications.

Several proposed works in the past few years have incorporated homomorphic encryption in machine learning tasks such as image classification, character recognition, and so on. These machine learning processes integrate non-linearities via transcendental functions such as exponential function and logarithmic functions. Nevertheless, the homomorphic encryption scheme does not directly compute or support such functions. There thus would be the need to use instead polynomial functions, mainly involving addition and multiplication operations, which allows this privacy-preserving task to become feasible to execute. Several methods have been proposed and utilized in the literature.

We realised from most existing works that the design space for implementing feasible HE solutions that would mostly involve using calculable polynomials (to replace most non-linear transcendental functions in the neural network) is not explored much. The motive is to be able to compute functions in the encrypted domain thus needing to convert tasks into additive and multiplicative operations. Most proposed works just derive one or two functions which yielded the best result in the end but in most practical machine learning tasks, unpopular types of activation functions used might produce a better performance so the same should be done for privacy-preserving tasks if we are looking to implement such systems practically.

In this paper, we looked at how to derive alternative, but effective polynomials for implementation in convolutional neural networks for applications in privacy-preserving image classification using the MNIST [14], [15] and CIFAR-10 [3], [4] datasets by exploring the design space much thoroughly. We were able to deduce that our approximation using polynomials yielded results as accurate as those utilizing non-polynomials and can thus be further utilized in homomorphic encryption to yield competitive results.

A. CONTRIBUTIONS

This paper has the following contributions:

- We derived several low-degree polynomials to use in implementing the HE machine learning tasks, thus not being limited to just a few as other works proposed. These polynomials yield good accuracy in both plaintext and ciphertext domains.
- We showed that some activation functions, which usually have a poor performance and thus rarely used (e.g. Sigmoid, Softplus), can be combined to yield alternatives that provide good performance and thus can be utilized for encrypted and non-encrypted computations.
- In our proposed work, we showed that through effective HE scheme parameter selection and machine learning optimizations, we can reduce the number of multiplication operations (multiplicative depth), memory load, and improve execution speed.
- We also showed that though activation functions' products (might) increase computational cost, the cost can be amortized. In particular, the subsequent polynomial derivation and use in homomorphic inference allows us to weigh the polynomial (activation) functions in terms of their degrees thus giving many choices in selection to yield the best accuracy, speed and throughput.

B. THREAT MODEL

This being a two-party or two way client-server model means there is some level of trust to be maintained between both parties. The client encrypts the data and sends to the server for prediction but the server would not be able to decrypt this data likewise the predicted output without the client's secret key. The server sends the encrypted result to the client who can decrypt with the secret key. A third party though can have access to the server's model and communication channel's internal state. The former means that as training was done in plaintext, model weights say can be accessed prior to inference and the latter means one can perceive the data transmission status and the client's encrypted message can be accessed in transit though not deciphered without the secret key.

C. ORGANIZATION OF PAPER

The rest of the paper is organized as follows. We first briefly give a review of related work in Section II where we describe some strengths and areas that can potentially be improved in relation to our work. In Section III, we highlight the cryptographic technique of homomorphic encryption and touch on convolutional neural networks with emphasis on activation layer functions as used to implement homomorphic image inference using some benchmark datasets from literature. The set of methods used to accomplish our work are discussed in Section IV and this is followed by Section V where we give the details of the network architecture and hardware configuration used to implement the work. We also discuss the results we attained first in plaintext domain by using conventional activation functions and polynomials derived

from the former for both datasets. Section VI describes the graph compiler used in implementing the encrypted image inference process and we also report the results of homomorphic inference on both datasets. Finally, we give the general conclusions of our work and some recommendations for future work in Section VII.

II. RELATED WORK

As was mentioned in Section I, HE and MPC form the two major categories of privacy-preserving machine learning implemented in literature and in this section, we discuss proposed works and their limitations that predominantly used the former technique as this relates to what our proposed solution also utilized.

Gilad-Bachrach *et al.* [1] proposed the first neural network over encrypted data, called *CryptoNets* [7] which provided a method for executing the inference phase of privacy-preserving deep learning. They showed that neural networks can be applied to encrypted data to yield encrypted results or predictions, where the latter can then be decrypted with a secret key. They demonstrated CryptoNets on MNIST optical character recognition tasks and achieved approximately 99% accuracy using a *square* function. Using the *square* function has the tendency to blow up activation as it squares incoming values, unlike other activation functions. ReLU has a similar characteristic and the square unit will just compound it more including that of negative values.

Hesamifard *et al.* [6] developed a technique to approximate the activation functions used in CNNs with low degree polynomials. The proposed work here outperformed other solutions [1], [31] owing to how efficient and accurate it is. They trained CNNs using different approximation methods including *Chebyshev* polynomials (also known as min-max approximation) but were not able to achieve satisfactory results on MNIST. They instead used an approach based on the derivative of the ReLU unit to achieve a higher accuracy but the approximation for sigmoid and tanh units did not achieve good results. This work was implemented on a CPU and the focus was on privacy-preserving classification. They used the MNIST and CIFAR-10 datasets.

Chabanne *et al.* [31] proposed the implementation of privacy-preserving machine learning on deeper neural networks. They incorporated several non-linear layers and used the batch normalization principle proposed by [21], and were able to achieve satisfactory result on the MNIST dataset. They however used a much deep neural network to achieve an accuracy of 99.30% on MNIST (with average pooling) and attained a much lower accuracy with a shallower (network) variant which shows how costly their method or task can be to attain a good accuracy.

Ishiyama *et al.* [16] employed various polynomial approximations of ReLU and Google's Swish [33], [34] function to implement homomorphic inference. They achieved favourable results better than some of the works described in this section. Although they presented a lower value degree polynomial, their proposed solution does not provide many options for approximation functions and other ranges to consider. The accuracy gain margin can thus be improved. Their best result on MNIST was 99.29% using a degree 2 Swish polynomial and average pooling.

Chen *et al.* [10] proposed using logistic regression on encrypted medical (genomic) data. They used the min-max approximation method to approximate the *sigmoid* function, which they used as an activation function. Specifically, the Remez algorithm [13], being an iterative minimax approximation algorithm, was used to generate a more accurate and low degree polynomial for use in the encrypted logistic regression model.

Bos *et al.* [17] also used homomorphic encryption to implement predictive analysis on encrypted health data in the cloud. They used logistic regression and *Cox proportional hazard* regression models to effectively implement a real-life privacy-preserving predictive analysis using the Taylor series method to approximate their analytic function. To attain a good accuracy, they used a very high degree of the polynomial (up to the 7^{th} degree), which increased the computation cost of the polynomial in ciphertext form.

Cheon *et al.* [18] also proposed a similar work as the previous one. However, they used the mini-max approximation and Non-Adjacent Form (NAF) encoding. That allowed them to remove the limitation induced by an input range, making their analyses faster and more accurate than [17]. Nevertheless, their regression models still induced a higher polynomial degree.

Table 1 illustrates how our proposed solution compares to that of existing ones using several common parameters in the design space.

There is the need for further space exploration of activation function, considering the formation of activation functions and their polynomial fittings, especially using low-order polynomials. We realised from these works that much emphasis was not placed on yielding an optimal activation function to use but rather the resultant accuracy to achieve, though it is arguable. Methods as the type of pooling used were based on the author's preference without any comparison to others, though other combinations to other functions could have proved better. It was though stated in [7] that pooling can be avoided without adversely affecting accuracy in MNIST prediction, but looking into much complex applications, it will be noteworthy to not overlook such. It can be that the area of searching for much optimized polynomials being derived from activation functions can prove to be a tedious search process, as other even better types of activation functions can be derived in future for use in machine learning algorithms. We considered all these various aspects in our work thus not being bounded by few arguments to use in our search space.

III. BACKGROUND

In this section we describe key areas relevant to the field and in this work. As this paper sought to implement homomorphic image inference (as being part of the much broader privacy-preserving machine learning), homomorphic

TABLE 1. Comparison between our proposed solution and existing ones.

Comparison	[1]	[6]	[31]	[16]	[10]	[17]	[18]	Ours Proposed
Privacy-preserving technique	HE	HE	HE	HE	HE	HE	HE	HE
HE scheme	BFV	BGV	BGV	CKKS	BFV	BFV	BGV	CKKS
Dataset(s) used	MNIST	MNIST, CIFAR-10	MNIST	MNIST, CIFAR-10	MNIST	–	–	MNIST, CIFAR-10
Activation/ Polynomial Function used	Square	ReLU	ReLU	ReLU/Swish	Sigmoid	Sigmoid	Sigmoid	Various
Batchnormalization	×	×	✓	✓	×	×	×	✓
Range of approximation	N/A	Arbitrary	[-3, 3]	Various	[-5, 5]	(-1, 1)	[-3.6, 5.7]	Various
Pooling Method(s)	Scaled mean	Scaled mean	Average	Average	Average	N/A	N/A	Scaled mean, Average
Polynomial degree(s) considered	2	2, 3	2, 4	2, 4	3	7	7	2, 3, 4

BFV – (Brakerski)-Fan-Vercauteren – as originally proposed by [20]
BGV – Brakerski-Gentry-Vaikuntanathan – scheme proposed in [19]
CKKS – Cheon-Kim-Kim-Song – presented by [28]
Datasets have been limited to just the two used in this paper.
✓ – applied × – not applied

encryption, convolutional neural networks and datasets as utilized in this work are clearly distinguished.

A. FULLY HOMOMORPHIC ENCRYPTION

Fully homomorphic encryption (or FHE for short) allows for computing arbitrary number of operations on encrypted data which is ideal considering it to be very useful for use in several practical tasks. It though tends to be computationally expensive and induces a very huge overhead making them almost infeasible for use practically. We can use the following algorithms to define any public key homomorphic encryption scheme [7], [11]:

Let $k, q, t > 1$ with $N = 2^k$, t prime and $\mathcal{R} = \mathbb{Z}[X]/\langle X^N+1\rangle$. We can refer to some message space as $\mathcal{R}_t = \mathcal{R}/t\mathcal{R}$ and a ciphertext space as $\mathcal{R}_q = \mathcal{R}/q\mathcal{R}$.

- $KeyGen_\epsilon(\lambda, L)$: Being a probabilistic algorithm which takes a security parameter λ and produces a public key pk and a secret key sk.
  Given λ and L as inputs, k, q is chosen so as to attain λ. A secret key $s_k \in \mathcal{R}_2$, random element $a \in \mathcal{R}_q$, some noise $e \in \mathcal{R}_q$ is selected, we have a public key, pk being $pk = (b = e - ask, a)$.
- $Encrypt_\epsilon(m, pk)$: Being a probabilistic algorithm which takes messages ($m \in \mathcal{R}_t\{0, 1\}$), and public key pk and outputs a ciphertext $c = Encrypt_\epsilon(m; pk) = (br' + e' + \lfloor q/t \rfloor m, ar')$, for some random noise $e', r' \in \mathcal{R}_q$.
- $Decrypt_\epsilon(c, sk)$: Being a deterministic algorithm which takes a ciphertext $c = (c_0, c_1) \in \mathcal{R}_q^2$, secret key sk, and outputs message $m = Decrypt_\epsilon(c; sk) = \lceil (t/q)(c_0 + c_1 s \bmod q) \rfloor \bmod t$.
- $Evaluate_\epsilon(f, c_1, \cdots, c_t; pk)$: This algorithm takes pk, n ciphertexts $c_1, c_2, \cdots, c_n$, and a permitted circuit C^n and outputs $C^n(c_1, c_2, \cdots, c_n)$. Let us assume we take the set of ciphertexts c_i with respective messages m_i, and the circuit C, this algorithm stated under this point will result in a new ciphertext c. This holds for:

$$Decrypt_\epsilon(Evaluate_\epsilon(C, c_i, pk), sk) = C(m_1, \cdots, m_t). \quad (1)$$

We can now formally define what fully homomorphic encryption is based on the algorithms stated. The scheme $\zeta = (KeyGen, Encrypt, Decrypt, Evaluate)$ would thus be considered as homomorphic for some circuits C if it holds for (1) for all circuits $C' \subset C$ and ζ is fully homomorphic if it holds for all Boolean circuits. We can go ahead to also state that if for any circuit $C' \subset C$ having a number of inputs which is polynomial in λ, the size of ciphertexts output by $Evaluate$ is bounded by a fixed value which is polynomial in λ, and thus ζ is compact.

The two major operations underlying FHE are addition and multiplication and they can be defined as below using say two hypothetical ciphertexts as inputs. Given the two ciphertexts, thus $c_1 = (c_{0,1}, c_{1,1})$ and $c_2 = (c_{0,2}, c_{1,2})$;

- $\mathsf{HAdd}(c_1, c_2)$: We get the resultant ciphertext, $c_1 + c_2 = c' = (c_{0,1} + c_{0,2}, c_{1,1} + c_{1,2})$ which is a component-wise addition operation.
- $\mathsf{HMult}(c_1, c_2)$: We evaluate as $c_1 \times c_2$;
  - Compute tensor, c^*
    $c^* = (c_0 = c_{0,1}c_{0,2}, c_1 = c_{0,1}c_{1,2} + c_{1,1}c_{0,2}, c_2 = c_{1,1}c_{1,2})$;
  - Scale and Relinearize the output, c'
    $c' = \lceil \mathsf{Relinearize}(\lceil (t/q)c^* \rfloor) \rfloor \bmod q$.

Where $\mathsf{Relinearize}(c^*)$, is a technique used to reduce the growth in the product or size of the ciphertext result c^* brought about by the multiplication operation. In relinearization, an evaluation key (evk) is generated to control this growth. Using an integer w to control the decomposition rate and number of components $l + 1$ in evk, where $l = \lfloor log_w q \rfloor$. For $0 < i \leq l$, sample $a_i \in \mathcal{R}_q$ and $e_i \in \mathcal{R}_q$ and calculate $evk[i] = ([w^i sk^2 - (a_i s + e_i)]_q, a_i)$. Thus we break down c_2 in base w which implies $c_2 = \sum_{i=0}^{l} c_2^{(i)} w^i$. We have $c' = c_j + \sum_{i=0}^{l} evk[i][j] c_2^{(}i)$, where $j \in 0, 1$.

Gentry in his work [29], [30] was the first to propose the construction of a FHE. This has further opened the doors for researchers from diverse backgrounds to adopt HE in their work as FHE allows for several types of operations to be performed on ciphertexts and most especially to compute multiple addition and multiplication operations. It is seen as the go-to method when considering privacy-preserving

machine learning application though other works in this area choose to go for other types of techniques and even apply a blend of some of them in their works. In our paper, we adopted FHE scheme in our evaluation in order to benefit from the numerous features it renders and for details about this cryptographic technique, one can refer to [20], [28] and Gentry's work as it falls outside the scope of this paper.

B. CONVOLUTIONAL NEURAL NETWORKS

Convolutional Neural Networks (CNNs), being a type of deep learning architecture used mostly in computer vision tasks and in this paper nonetheless, consist of several layers running from the input layer to the output layer, where each layer has a particular task to perform on data that is traversing the network. The units present in these layers perform computation on data using operations like addition and multiplication or using some function(s) to transform data. Some of these layers are the convolutional layer(s), activation layer(s), pooling layer(s), fully connected layer(s), dropout layer, softmax (classifier) layer, and so on. In this section, we discuss the activation layer as it is central to this paper.

1) ACTIVATION LAYER

The inducing of non-linearity within the network arises from this layer, and HE does not support the corresponding activation (or transfer) functions used. There have been alternative methods such as using square activation [1], and the derivative of ReLU [6]. These chosen methods performed similarly to an activation function, and HE supports them. The more notable activation functions as *ReLU*, *sigmoid*, *tanh*, and *softplus* as used in literature are considered for use in our experiments.

- **ReLU:** The *ReLU* activation (or transfer) function uses the maximum function, $ReLU(x) = max(x, 0)$, to induce non-linearity after the convolution operation is performed by clipping off all negative numbers, as in a filter, and passing only positive numbers to the pooling layer. This function has proven to be very effective in most CNN applications, thus adopted mostly as compared to others such as *sigmoid* and *tanh*. A merit of using this function is that it does not cause gradients to vanish during training thus weights can be easily adapted to enhance learning and having a better convergence during training helps to improve performance. A downside to using it is that as it is linearly increasing, it blows up activation function, which causes weights to have large values and this affects learning adversely.
- **Sigmoid:** The *Sigmoid* function, also known as the *log-sig* function, uses the *logistic-sigmoid* function, $\sigma(x) = \frac{1}{1+e^{-x}}$ to induce non-linearity just like other activation functions. It normalizes all positive and negative numbers to values between 0 and 1; thus, it does not blow its activation, ensuring smooth learning. The downside is that it tends to cause vanishing gradients, especially in deep networks, which causes problems in the computation of the loss function and weight adaption. The *Sigmoid* function can be viewed as a smoothened form of a *Heaviside* (step) function where the latter collapses all negative and positive numbers to 0 and 1 respectively instead of having intermediate values between these two extremes.
- **Tanh:** The *Tanh* or hyperbolic tangent function in a similar fashion as the *Sigmoid* also squashes positive and negative numbers but into the interval -1 and 1. It uses the *Tanh* function, $\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$ to induce non-linearity into the CNN. In some applications, it tends to perform better than *Sigmoid* in terms of speed and accuracy so it can be employed based upon output performance.
- **Softplus:** The *Softplus* function has the relation $\ln(1 + e^{x})$; thus, it computes both an exponent and a natural logarithm of input fed to it. It has a very close resemblance to the *ReLU* function, but unlike the latter, it is not used very often as *ReLU* and *Sigmoid* tends to have a much better performance effect. Also, we can avoid computing both exponent and natural logarithm in one instance as this will be costly. We adopted this function in our paper because it is still very favourable owing to its likeness to *ReLU* and as will be later seen, induces good performance in tasks when it was combined with other functions.

C. DATASETS

Data forms a core of machine learning. Though mathematical formulations and algorithms are also vital here, datasets always play the central role in almost every problem we want to solve with machine learning.

The datasets usually considered for implementing privacy-preserving deep (or machine) learning currently consider the accessibility and ease-of-use of the database, relative size (or the resolution of images in the latter's case) and storage requirements of datasets, and so on as tasks are intuitively very complex and computationally costly. Given these reasons, notable ones used are thus the MNIST and CIFAR-10 datasets and have become a benchmark for implementing tasks like classification in general. Thus, it seemed fit to use such datasets for homomorphic neural network inference in this paper.

1) MNIST

The Modified National Institute of Standards and Technology database, known as MNIST, is an extensive database of handwritten digits usually used for image processing and character recognition tasks. It is made up of 10 categories or classes of integers from 0 to 9, with each digit having a 28×28 pixel image format. The dataset has a training set of 60,000 examples or images and a test set of 10,000 examples of gray-level images [14].

This dataset's simplicity and lightweight nature makes it very resourceful for performing homomorphic (encrypted) deep learning.

2) CIFAR-10

One of the most popular datasets for machine learning and computer vision is the CIFAR-10 dataset (Canadian Institute For Advanced Research). There exists the CIFAR-10 and CIFAR-100 variants and both are labeled subsets of the *80 million tiny images* [12] dataset. CIFAR-10 dataset has 10 classes of 32×32 pixel colour images, having 6000 images per class. Thus, there is 60,000 in total, with 50,000 being training images and 10,000 test images. The classes do not overlap and are very distinct from each other, making them completely mutually exclusive. These 10 classes are aeroplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.

The description given makes it an ideal dataset for privacy-preserving deep learning inference.

IV. PROPOSED METHOD

Obviously, as the HE scheme does not support most operations (or operators such as division) and especially non-polynomial functions, the drive is to convert most computations into additive and multiplicative tasks within the neural network. The mathematical technique used for such is known as *function approximation*. Table 3 gives a sample of the outcomes of approximating functions using *least squares* method [22], [26] (from numerical analysis) along with their respective accuracies we got from our experiments. We describe in the subsections here the various methods we utilized in our paper to perform the homomorphic prediction of the MNIST and CIFAR-10 datasets.

For the CNN pooling layers, we considered two methods that can be used in the encrypted domain. Thus the *Average Pool* and *Scaled mean* or *Sum Pool* and either can be selected based upon its performance in the plaintext domain (as can be seen in comparison to the *MaxPool* in Section V).

Table 3 bluntly illustrates the matching of parameters as the accuracy and the least square polynomials we generated from the conventional activation functions (whether in their raw form or after been paired through multiplication). For the ranges used for approximating the functions in least squares, we considered $[-1, 1]$, $[-2, 2]$, $[-3, 3]$, $[-4, 4]$, $[-5, 5]$, $[-6, 6]$, and $[-10, 10]$ on the x-axis of the $x - y$ plane and used an interval of 0.5. For ease of illustration in this paper, we present the plots of the approximations of the activation functions in one range with their low-degree polynomials considered in the Figs. 4, 5, and 6.

A. ACTIVATION FUNCTION COMBINATIONS

We sought to combine notable activation functions in literature to yield activation function products in our work. We then used them for neural network training and prediction in the plaintext domain to assess which ones would have good performance favourable for deriving polynomials for the homomorphic inference process. These activation function combinations are as seen in Table 2 with repeated products omitted. As noticed, the Swish [33], [34] and Square are also derived and are widely popular in literature. We grouped these resultant functions under three (3) categories for easy description and they are namely;

1) WEIGHTED LINEAR UNITS

These units are as illustrated in Fig. 1 and are briefly defined as;

- **Swish:** This linear unit and Sigmoid product is referred to as the Sigmoid-weighted Linear Unit (SIL). The Swish, as proposed by [33] uses a trainable parameter, β whereas SIL, as proposed by [34], is just the product of the two functions and the latter is as used in this paper.
- **XTanh:** This product of a linear unit and hyperbolic tangent is referred to as the Tangent-weighted linear unit.
- **XSoftplus:** This product of a linear unit and Softplus is referred to as the Softplus-weighted linear unit.

2) EXPONENTIALLY-WEIGHTED UNITS

These units as illustrated in Fig. 2 are briefly defined as;

- **Square:** Being one of the widely used functions in earlier implementations of privacy-preserving machine learning or image classification [1], [7], it is mostly used to approximate the ReLU function.
- σ^2 : Square of the functions Sigmoid: $\frac{1}{1+e^{-x}}$, Tanh: $\frac{e^x - e^{-x}}{e^x + e^{-x}}$, and Softplus: $\ln(1 + e^x)$.

3) ACTIVATION PRODUCTS

These units as illustrated in Fig. 3 are briefly defined as;

- **SigTanh:** The product of Sigmoid and hyperbolic tangent units or vice versa due to the commutative property of multiplication.
- **SigSoftplus:** The product of Sigmoid and Softplus units or vice versa.
- **TanhSoftplus:** The product of hyperbolic tangent and Softplus units or vice versa.

B. DEEP LEARNING NORMALIZATION METHODS

Using a conventional function as Tanh allows the computation of values in the range $(-\infty, +\infty)$ during training; thus, any number on the x-axis (in the $x - y$ plane) can be easily computed without any difficulty. In the case of polynomials, there is the limitation of only being able to approximate a function as Tanh in some range $[-x, +x]$ (where say $x = 1, 2, 3, 4 \ldots$) and during computation, values yielded outside such a range can pose a problem thus needing to use a wider range at all cost. We first addressed this by normalizing the pixel values of the MNIST and CIFAR-10 datasets, which is in the range [0, 255] into [0.0, 1.0] to yield smaller values of input data so as not to scale up weight values during training. In addition to this, we used the Batchnormalization technique proposed by [21]. Both of these techniques are usually promoted and thus adopted in machine learning as they tend to improve performance, so we found it very vital to utilize in training. We applied batch normalization before an activation (function) process, and this has benefits in many

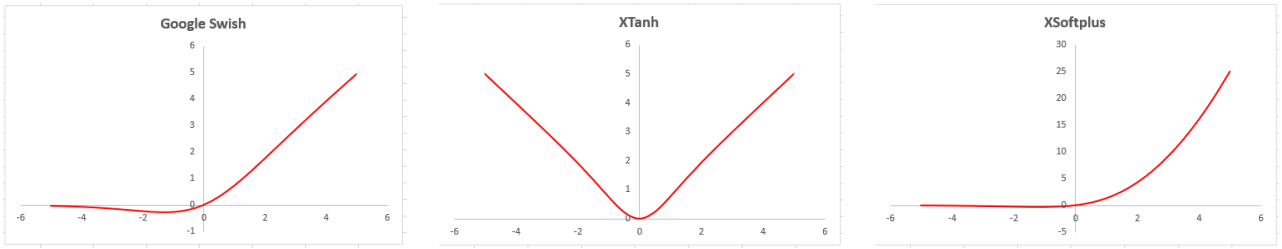


FIGURE 1. The linear and Sigmoid functions combination gives the Sigmoid-linear unit or the Swish (a). The product of linear and hyperbolic tangent functions gives the tangent-linear unit shown in (b) and the Softplus-linear unit is as seen in (c).

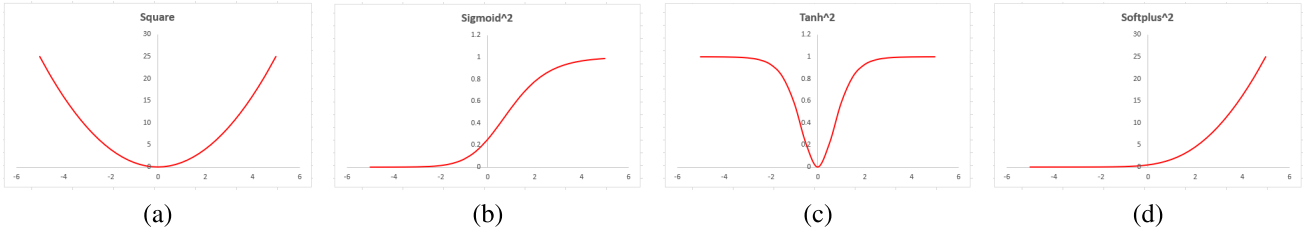


FIGURE 2. Squaring the linear, Sigmoid, hyperbolic tangent, and Softplus functions gives the squared units (a), (b), (c), and (d) respectively.

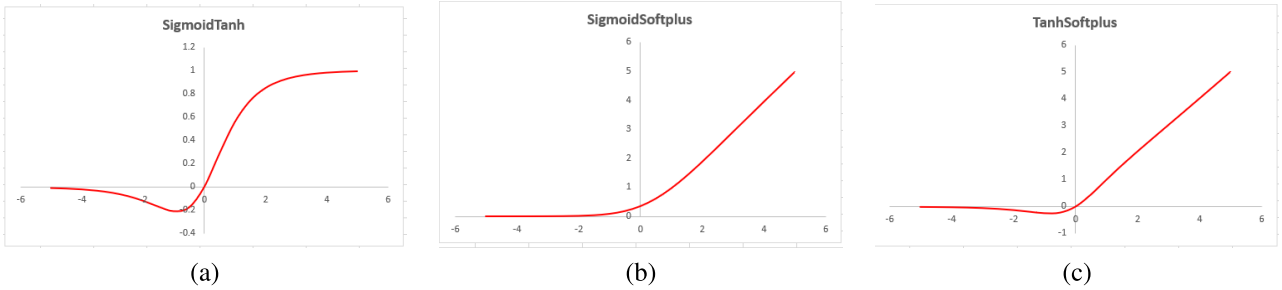


FIGURE 3. The combination of the hyperbolic tangent and Sigmoid functions gives the Sigmoid-Tangent unit (a), the product of Sigmoid and Softplus functions gives the Sigmoid-Softplus unit shown in (b) and the Softplus and hyperbolic tangent product is as seen in (c).

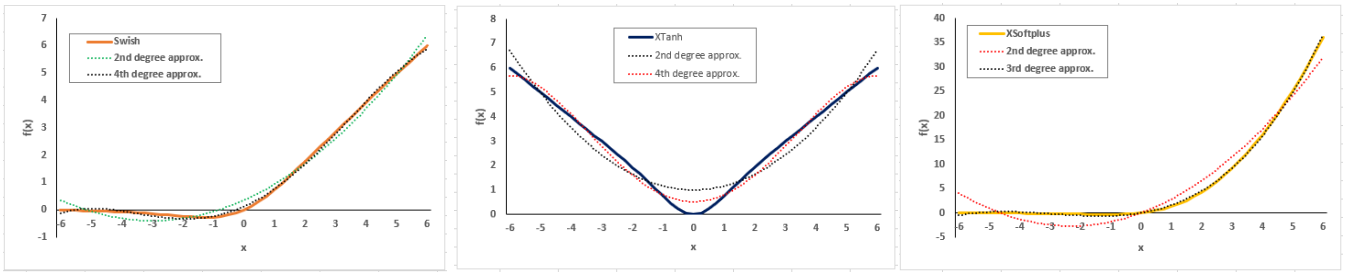
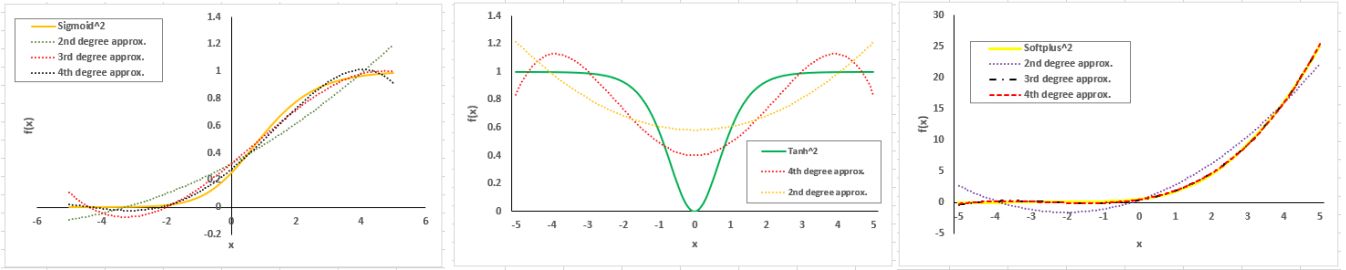


FIGURE 4. The approximations of the Sigmoid-linear unit (a) and the tangent-linear unit shown in (b) to derive degrees 2 and 4 polynomials and approximation of the Softplus-linear unit to derive degrees 2 and 3 polynomials is as seen in (c).

ways as: (1) normalizing input features fed to the activation to attain a Gaussian distribution so as not to let variables shift that much, (2) ensuring numerical stability in computation due to the use of a much-constricted range of the least square activation polynomial, (3) inducing some bit of regularization which helps to reduce overfitting, and overall (4) speeding up training and improving performance. The *Algorithm 1* listing shows how batch normalization is implemented;

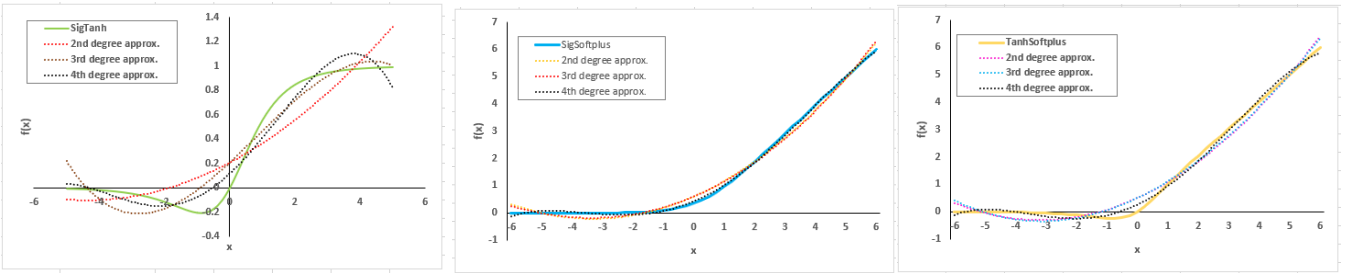
C. CKKS HOMOMORPHIC ENCRYPTION SCHEME

This recently popular HE scheme is as proposed by [28]. Using the products of functions we derived in Table 2,



(a) (b) (c)

FIGURE 5. The approximations of the squares of Sigmoid, hyperbolic tangent, and Softplus to derive the low-degree polynomials are as respectively illustrated in (a), (b), and (c).



(a) (b) (c)

FIGURE 6. The approximations of the Sigmoid-Hyperbolic Tangent, Sigmoid-Softplus, and Hyperbolic Tangent-Softplus units to derive degrees 2, 3, and 4 polynomials in all are as seen in (a), (b), and (c) respectively.

TABLE 2. Activation function combinations.

×	Linear (X)	Sigmoid	Tanh	Softplus
Linear (X)	Square	Swish	X·Tanh	X·Softplus
Sigmoid	–	Sigmoid^2	Sig· Tanh	Sig· Softplus
Tanh	–	–	Tanh^2	Tanh· Softplus
Softplus	–	–	–	Softplus^2

TABLE 3. Least squares polynomial approximation.

Activation	Range	Polynomial	Accuracy (%)
ReLU	[-2,2]	$0.1948 + 0.5x + 0.2165x^2$	99.16
Softplus	[-5,5]	$0.7975 + 0.5x + 0.0731x^2$	98.94
Tanh	[-5,5]	$0.4899 - 0.0129x^3$	99.34
Sigmoid	[-5,5]	$0.5 + 0.1945x - 0.0041x^3$	99.10

we generated the corresponding least square polynomials for the ranges stated in Section IV of which some can be seen in Table 3. Using the CNNs in Tables 4 and 5, we performed training and prediction (in plaintext) using all the least square polynomials we generated to assess their efficacy and feasibility for use in the homomorphic prediction. As we sought to perform homomorphic inference, we first trained the CNN with the polynomial in the plaintext domain to generate our layer weights. We later performed prediction on a set of encrypted images using this model. We thus avoid the huge overhead, complexity, and costs of performing encrypted training, which is not the goal of this paper, and we can nonetheless use the trained weights of the network in the end. During homomorphic inference, we encoded our input data using the *polymodulus* **(t)** parameter and then encrypted this with the *coefficient modulus* **(q)**. We utilized degrees 2, 3, and 4 polynomials and the values of parameters used for homomorphic inference is as discussed in Section VI.

Algorithm 1 BatchNormalization Technique

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \ldots_m\}$;
Parameters to be learned: γ, β
Output: $\{y_i = BN_{\gamma,\beta}(x_i)\}$

1. $\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$ // mini-batch mean
2. $\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$ // mini-batch variance
4. $\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}}$ // normalize
5. $y_i \leftarrow \gamma\hat{x}_i + \beta \equiv BN_{\gamma,\beta}(x_i)$ // scale and shift

D. OPTIMIZATIONS IN HOMOMORPHIC INFERENCE

An important factor in HE is that we can adopt some optimization techniques to reduce complexity and improve runtime considerably, and some of these were proposed by [35], [36], and also utilized by [16]. These methods are employed in machine learning and can be used to improve performance in training considerably.

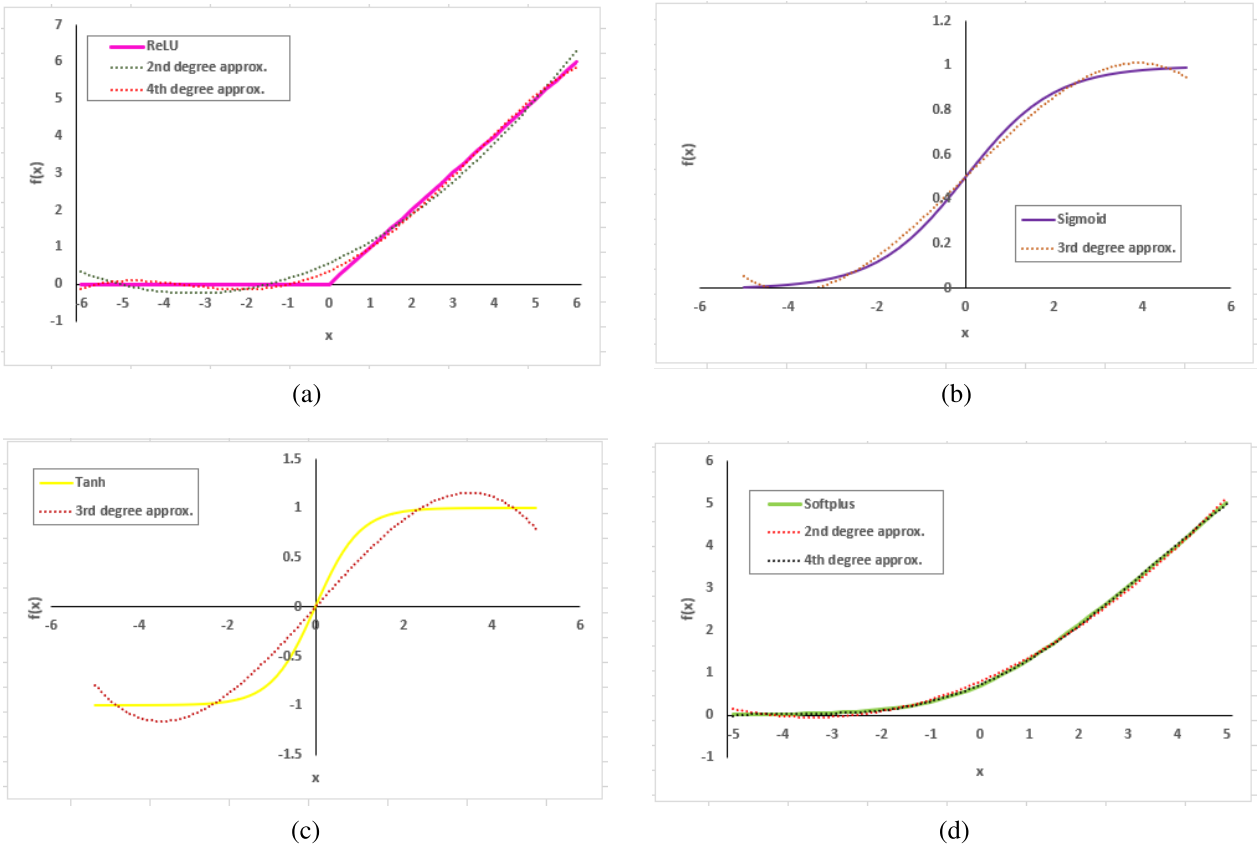


FIGURE 7. The approximations of the conventional activation functions as also considered are as illustrated in (a), (b), (c), and (d).

These graph-level optimization methods are termed as *folding* and we utilized them as explained briefly below;

- **BatchNorm (BN):** Due to the availability of the weights at test time, we can reuse parameters without incurring extra costs from retraining thus not consuming more levels. From Section IV-B, the variables $\gamma, \beta, \mu, \sigma$ are all fixed during the inference phase. The use of the BN means that when we compute some z where $z = W * x$ and W represents the weight in a convolution or fully connected layer with x being the input, we get the output pre-activation as $z_{BN} = \gamma\hat{z} + \beta$ where $\hat{z} = \frac{z-\mu_z}{\sqrt{\sigma_z^2+\epsilon}}$
  At inference thus, we compute (using BN) $z_{BN} = \gamma_T z + \beta_T$ where $\gamma_T = \frac{\gamma}{\sqrt{\sigma_z^2+\epsilon}}$ and $\beta_T = \beta - \frac{\gamma\mu_z}{\sqrt{\sigma_z^2+\epsilon}}$
  Computing z and z_{BN} entails a multiplicative depth of 2 but it can be reduced to 1 as the variables W, γ_T in $z_{BN} = \gamma_T Wx + \beta$ are constants.
- **Activation:** We manipulated the polynomials in such a way as to reduce the multiplicative depth. Given the 4^{th} degree polynomial, say $ax^4+bx^3+cx^2+dx+e$, 3^{rd} with ax^3+bx^2+cx+d, and 2^{nd} with ax^2+bx+c having multiplicative depths of 3, 3, and 2 respectively, we reduced them to $x^4 + (b/a)x^3 + (c/a)x^2 + (d/a)x + (e/a)$, $x^3 + (b/a)x^2 + (c/a)x + (d/a)$, and $x^2 + (b/a)x + (c/a)$ in that order to give reduced multiplicative depths of 2, 2, and 1. Thus before computing an activation, we scale the weights in the convolution or fully connected layer with a (i.e. aW).
- **Pooling:** Though we used both scaled-mean and average pooling in our paper, the two were nonetheless treated differently. Given that our scaled-mean operation is some X whereby we utilized the window size $a_1 \times a_2$, our multiplicative depth will be limited to X but for the average pooling case, the division by $a_1 \times a_2$ giving $(a_1 \times a_2)^{-1} \cdot X$ increases the multiplicative depth by 1. This can be addressed by multiplying the subsequent convolution layer's weights W instead (if any) with the window size to yield $(a_1 \times a_2)^{-1} \cdot W$ so that the average pooling layer uses the X operation to maintain a multiplicative depth of 1.

From these procedures, we yield the levels as discussed in Section VI for the various polynomials considered.

V. PRELIMINARY PLAINTEXT RESULTS

As an obvious key requirement, we first performed training and testing using the activation function product-forms or pairs and their polynomials generated before carrying out experiments in the ciphertext domain. This is presented in order for both datasets and owing to the numerosity of resultant data generated, we illustrate using plots and tables given under this section.

TABLE 4. ConvNet architecture for training and testing using MNIST.

Layer	Parameters	Output Size
1^{st} Convolution	Input image of size 28×28 is fed here. 32 filters each of size (5×5), stride (2,2), and zero padding	$28 \times 28 \times 32$
Activation 1	**ReLU*** function is applied to previous layer inputs	$28 \times 28 \times 32$
Pooling 1	**Max**† pooling applied using stride 2 and kernel (2,2)	$13 \times 13 \times 32$
2^{nd} Convolution	64 filters each of size (5×5), stride (2,2), and zero padding	$13 \times 13 \times 64$
Activation 2	**ReLU*** function is applied to previous layer inputs	$13 \times 13 \times 64$
Pooling 2	**Max**† pooling applied using stride 2 and kernel (2,2)	$5 \times 5 \times 64$
1^{st} Fully Connected	Compute weighted sum of previous layer using 512 nodes	$1 \times 1 \times 512$
2^{nd} Fully Connected	Compute weighted sum of previous layer using 10 nodes	$1 \times 1 \times 10$

$*$ – Several types of activation functions are used here
† – Max, Average, and Scaled mean (sum) Pooling types are used

TABLE 5. ConvNet architecture for training and testing using CIFAR-10.

Layer	Parameters	Output Size
1^{st} Convolution	Input image of size $32 \times 32 \times 3$ is fed here. 32 filters each of size $(3 \times 3 \times 3)$, stride (1,1), with padding	$32 \times 32 \times 32$
Activation 1	**ReLU*** function is applied to previous layer inputs	$32 \times 32 \times 32$
Pooling 1	**Max**† pooling applied using stride 2 and extent 2	$16 \times 16 \times 32$
2^{nd} Convolution	64 filters each of size $(3 \times 3 \times 32)$, stride (1,1) with padding	$16 \times 16 \times 64$
Activation 2	**ReLU*** function is applied to previous layer inputs	$16 \times 16 \times 64$
Pooling 2	**Max**† pooling applied using stride 2 and extent 2	$8 \times 8 \times 64$
3^{rd} Convolution	128 filters each of size $(3 \times 3 \times 64)$, stride (1,1) with padding	$8 \times 8 \times 128$
Activation 3	**ReLU*** function is applied to previous layer inputs	$8 \times 8 \times 128$
Pooling 3	**Max**† pooling applied using stride 2 and extent 2	$4 \times 4 \times 128$
1^{st} Fully Connected	Compute weighted sum of previous layer using 256 nodes	$1 \times 1 \times 256$
2^{nd} Fully Connected	Compute weighted sum of previous layer using 10 nodes	$1 \times 1 \times 10$

$*$ – Several types of activation functions are used here
† – Max, Average, and Scaled mean (sum) Pooling types are used

In our proposed method used to apply polynomial functions in character recognition using MNIST and image classification using CIFAR-10, we employed the models shown in detail in Tables 4 and 5. We utilized convolution layers, pooling layers, fully-connected (or dense) layers, and applied various activation functions alongside the polynomials we derived. The architecture in Table 4 used for the MNIST prediction follows the LeNet-5 structure introduced in [15] and the one in Table 5 used for CIFAR-10 is the default structure adopted by most literature that does CIFAR-10 image classification.

We run our computations on a workstation with a Linux operating system. We implemented our homomorphic inference on a CPU after we had done preliminary training tests on a GPU using MNIST and CIFAR-10 datasets to assess the activation functions we derived. The details of the hardware configuration are as shown in Table 6.

TABLE 6. Details of hardware used for performing our experiments.

Model Feature	CPU	GPU
Brand	Intel Xeon E7-8880 v3	Nvidia Tesla K20Xm
Processor Name	GenuineIntel	GK110
Architecture	x86_64	Kepler
No. of Processing Units	4	1
No. of Cores	64	2688
Processor Frequency	2.30 GHz	732 MHz
Compute Capability	–	3.5
Memory Capacity	1,952 GB	6 GB

A. MNIST PLAINTEXT RESULTS

We first present the results of training and testing using the MNIST dataset and the CNN in Table 4. These results are as illustrated in Figs. 8 and 9. The former plot with its sparse nature allows to easily discriminate individual activation functions as being measured in two important metrics as runtime and accuracy using one pooling method whiles the latter does a side-by-side comparison of pooling methods within one metric of measurement. The accuracy is the value attained at test time and the runtime is the training time. The runtime at test is rather instantaneous in the plaintext domain and also varying degrees of the polynomial are handled differently in the encrypted domain due to factors such as multiplicative depth thus we shifted focus to the training time, even though homomorphic inference will be performed in the aftermath, in order to make some explanations in the plaintext domain.

As was mentioned in Section IV, we utilized both HE-friendly pooling methods as determining the combination of activation function and pooling method that yields the best performance is almost always empirical. Using Tensorflow, we performed training for 100 epochs using a batch size of 128 and utilized the Stochastic Gradient Descent with momentum method and Batchnormalization [21] technique. It is obvious though that the Keras API [40], which provides a much higher level of (language) abstraction can provide much faster training and inference and even cause to reduce the number of epochs needed for training to achieve satisfactory results but we utilized it here nonetheless in order to assess the efficacy of the activation functions and polynomials to differing API abstraction. We limited its use to the MNIST though and for the more complex CIFAR-10 dataset, we used Keras to improve training efficiency.

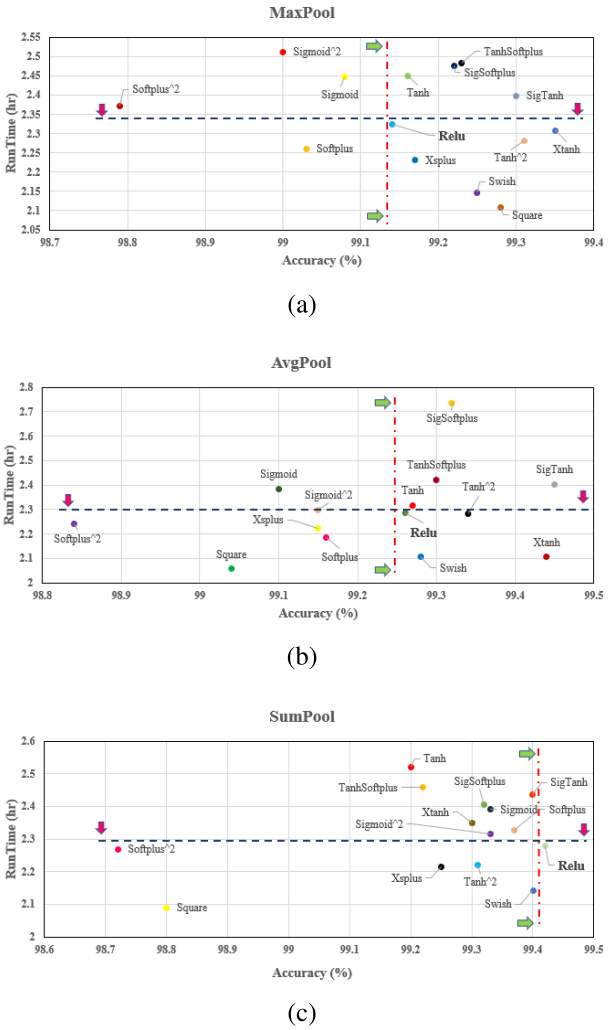


FIGURE 8. Plot of Results in utilizing MaxPooling (a), AveragePooling (b), Scaled-Mean or SumPooling (c), and all the activation functions under the MNIST category.

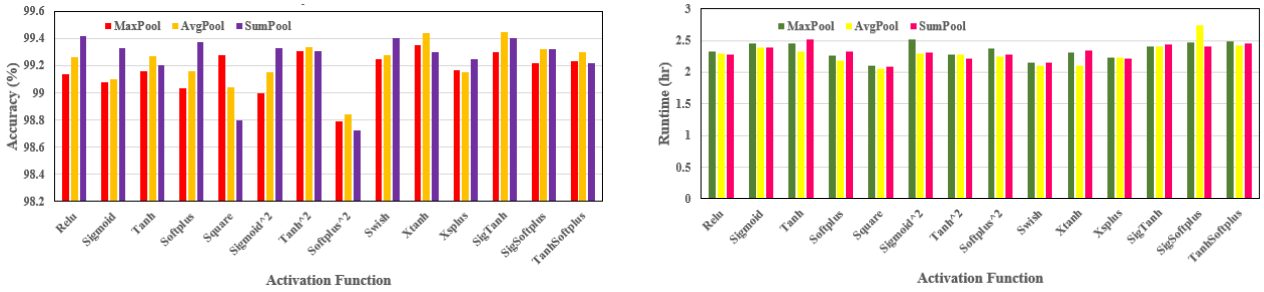
1) ACTIVATION FUNCTIONS

Fig. 8a illustrates the results when the activation functions and max pooling method were used. We included max pooling here in order to compare with the other HE-friendly alternatives being, average pooling (Fig. 8b) and scaled-mean pooling (Fig. 8c). To explain the results more easily, we decided to use the ReLU function as a yardstick to assess the others as it is used more often in neural network classification tasks due to its ability to improve efficiency and increase accuracy. It even serves as the basis upon which polynomials are derived via approximation or otherwise for use in HE training and/or inference. The plot is demarcated into 4 quadrants in both Figs. 8 and 10 with the ReLU in the bottom-right quadrant. The vertical dissecting line separates the activation functions in their accuracy values relative to the ReLU function whiles the horizontal dissecting line separates the activation functions in their training (run) time values relative to ReLU.

In Fig. 8a, if the resultant accuracy value is more of the emphasis, then there will be several other functions to choose from asides ReLU as their results are also very considerable. Assuming there are much accurate ways of dealing with the *max* function and conventional activation functions in the homomorphic domain, then we can outright make more selections from the results. As the HE task nonetheless involves training in the plaintext prior to homomorphic inference, the runtime in training can also be considered especially in cases where faster executions are needed and/or as models scale up and become more complex which augments the overall running time. Looking at the alternative pooling methods in Fig. 8, we realized that average pooling in Fig. 8b has the best returns in overall activation function performance relative to ReLU as compared to scaled-mean pooling in Fig. 8c. Fig. 8c though has a larger quantity of accuracy throughput of activation functions due to how many of them give an increased accuracy relative to ReLU and as compared to Fig. 8b. In the charts in Fig. 9, we can state that the average and scaled-mean pooling methods can provide good alternatives to the max pooling method (Fig. 9a) when paired with several other activation functions (and in comparison to the ReLU nevertheless). In Fig. 9b, it is observed that the runtimes of all the functions taper approximately to a single value thus it can be rendered as a negligible metric for smaller models used for training. In theory, much emphasis (especially in cases such as image classification and inference using HE) is in increasing accuracy or throughput and thus the best out of the lot can be selected. There though can be changes to expect if polynomials are used in place of the activation functions and we discuss the results for the approximated polynomials in the following section.

2) APPROXIMATED POLYNOMIALS

We also present the results of the approximated polynomials derived and used for training and testing using the MNIST dataset. These are as presented in Tables 7, 8, and 9. Due to the results being large in number, we made a summary of it using appropriate statistical parameters as the mean (μ), standard deviation (σ), minimum (min.), and maximum (max.) values of the test accuracy and training time. We discovered through our analysis that $Tanh^2$ had the worst approximation as can be seen in Fig. 5b using least squares thus, we omitted it from the approximated polynomial results; it either generated very bad results or was difficult to use for training in other cases. Also, the square function is not present as it exists uniquely and not considered as a polynomial derived from a conventional activation function. The μ and σ, being parameters of a normal distribution, gives a good measure of the data and we included the min. and max. values as in most privacy-preserving machine learning tasks, the key objective is to achieve the best accuracy and better it almost always. With our polynomials generated from the activation function pairings and including that of the individual functions, we can determine from Table 7, the ones that yielded the best accuracy, the average accuracy to likely expect, and the spread of



(a) (b)

FIGURE 9. Plot of results showing how Average Pooling and Scaled-Mean Pooling measures with MaxPooling on both the accuracy (a) and runtime scales (b) under the MNIST category.

the results of the activation function polynomial. To assess how the results of these polynomials measure up to that of the conventional functions in Table 2, we also presented the various deviations (being the differences) as shown by the 'margin' parameters in the Tables. This shows how good or bad the polynomial is when used for training as compared to its parent activation function. This value can be positive or negative with a number being far from 0 considered weak whiles that towards 0 is strong with the activation function being the reference 'zero' value. Also with the belief being that pairing activation functions might increase cost in training due to the computation of two functions, we included the training time parameters to assess whether such pairings will induce any significant costs as compared to using just one activation function. The training time inclusions clearly depicts that generally the activation function pairs can be used just as the individual functions used to derive them without accruing any substantial costs in training. As training is also done in plaintext, this can be overlooked so unless training is done in an encrypted domain, this cost, whether any, can be considered negligible.

B. CIFAR-10 PLAINTEXT RESULTS

We now present the results of training and testing using the CIFAR-10 dataset and the CNN in Table 5. These results are as illustrated in Figs. 10 and 11. We also utilized both HE-friendly pooling methods here in addition to the activation function combinations. Using Keras and the network architecture in Table 5, we performed training for 400 epochs using a batch size of 64 and utilized the Stochastic Gradient Descent with momentum method and Batchnormalization [21] technique. The latter technique is known to improve classification accuracy significantly and is as stated in [16] and finds popular use in machine learning thus we used it directly and its effect in the network is as described in Section IV-B.

1) ACTIVATION FUNCTIONS

In Figs. 10b and 10c, it can be observed that the average pooling and scaled-mean (or sum) pooling respectively gave much better accuracy results owing to how many of the activation functions cluster close to the ReLU, being the reference function, as compared to that of the max pooling (Fig. 10a) which has the activation functions spread out and far from ReLU. There are cases though were a few improved the accuracy beyond that of the ReLU and overall, most of the activation functions had reduced runtimes relative to the ReLU. In general, it can be said that the Swish function had much consistency in giving a favourable performance across both datasets whiles others do better more specifically when paired with a particular pooling method.

2) APPROXIMATED POLYNOMIALS

We also present the results of the approximated polynomials derived and used for training and testing using CIFAR-10. These results are as summarized in Tables 10, 11, and 12.

From Table 10, using ReLU as a reference, it can be observed that several other approximated polynomials gave better classification accuracy results being in the mean, standard deviation, and maximum values given. With the mean and standard deviation of the classification accuracy, we can determine how well using a particular type of approximated polynomial gives a better chance of achieving a satisfactory accuracy as compared to others. The minimum value gives a measure of how strong the approximated polynomial is even for some worse case parameters which gives a poor performance whiles the maximum value gives the highest achieved classification accuracy from the observations of the approximated polynomials. This is one of the, if not, the most important aspect that researchers look out for in determining which approximated polynomial to use for their homomorphic training and/or inference as they try to get close to the value of that yield by using a conventional activation function. We again stress on this here as using CIFAR-10 and the accompanying network in Table 5 presented a more challenging task in comparison to using the MNIST as previously discussed. Again using ReLU as a reference, we can select the polynomials with the increased accuracy to use in the encrypted domain nonetheless. The corresponding runtime in Table 10 gives us a picture of how

TABLE 7. Summary of Plaintext Results on MNIST Showing Variations in Comparison to the Conventional Activation Functions.

Approx. Poly.	Accuracy (%)				Runtime (hr)				Accuracy Margin(%)				Runtime Margin (hr)			
	mean	std	min	max	mean	std	min	max	mean	std	min	max	mean	std	min	max
ReLU	99.18	0.12	98.80	99.35	2.76	0.14	2.27	2.76	0.16	0.17	-0.09	0.62	-0.23	0.14	-0.48	0.02
Tanh	99.30	0.06	99.20	99.40	2.82	0.16	2.29	2.82	-0.06	0.04	-0.14	0.00	-0.13	0.25	-0.50	0.23
Sigmoid	98.97	0.29	98.17	99.21	2.93	0.25	2.23	2.93	0.25	0.26	-0.10	0.93	-0.19	0.26	-0.54	0.15
Softplus	98.91	0.21	98.47	99.15	2.85	0.20	2.17	2.85	0.15	0.35	-0.34	0.99	-0.16	0.19	-0.52	0.16
Swish	99.27	0.08	99.03	99.36	2.87	0.17	2.27	2.87	0.07	0.12	-0.08	0.37	-0.42	0.16	-0.73	-0.13
XTanh	98.87	0.49	97.63	99.40	2.92	0.22	2.11	2.92	0.50	0.45	0.04	1.67	-0.29	0.31	-0.81	0.24
XSoftplus	99.13	0.14	98.78	99.31	2.64	0.11	2.20	2.64	0.07	0.14	-0.09	0.46	-0.14	0.11	-0.43	0.02
SigTanh	99.26	0.10	98.97	99.43	3.34	0.35	2.22	3.34	0.17	0.09	0.02	0.43	-0.22	0.35	-0.93	0.22
SigSoftplus	99.00	0.34	97.56	99.30	2.84	0.17	2.24	2.84	0.32	0.34	0.02	1.76	0.03	0.21	-0.41	0.38
TanhSoftplus	99.22	0.11	98.86	99.39	3.48	0.38	2.17	3.48	0.04	0.10	-0.11	0.36	-0.37	0.39	-1.06	0.29
Sigmoid^2	99.20	0.09	99.01	99.37	3.05	0.31	2.19	3.05	0.04	0.15	-0.22	0.32	-0.30	0.31	-0.75	0.11
Softplus^2	99.09	0.16	98.79	99.32	2.44	0.09	2.19	2.44	-0.31	0.15	-0.48	0.04	-0.04	0.08	-0.17	0.05

Approx. Poly. – polynomial derived from parent activation function from least squares approximation.
mean, std, min, max – mean, standard deviation, minimum, and maximum values.
Accuracy and Runtime – respectively gives the accuracy at test time and the runtime of training.
Accuracy Margin and Runtime Margin – respectively gives the deviation of the classification accuracy and the deviation of the runtime of training of the polynomial result from that of its respective conventional activation function.

TABLE 8. Summary of Plaintext Results on MNIST Depicting Relationships in Pooling Methods, Ranges, and Degree of Polynomials.

Approx. Poly.	count	Poly. degree	Range	R^2 value		Accuracy (%)		Accu. Mgn. (%)		Runtime (hr)		Rt. Mgn. (hr)	
				mean	std	mean	max	mean	min	mean	min	mean	min
ReLU	27	2,4	[-1, 1]	0.999	0.004	99.06	99.43	0.21	-0.42	2.54	2.19	-0.20	-1.01
Tanh	14	3	[-2, 2]	0.994	0.010	99.19	99.40	0.08	-0.48	2.56	2.19	-0.22	-0.93
Sigmoid	14	3	[-3, 3]	0.988	0.020	99.15	99.41	0.11	-0.38	2.55	2.14	-0.21	-0.98
Softplus	28	2,4	[-4, 4]	0.982	0.027	99.15	99.37	0.11	-0.39	2.54	2.17	-0.20	-0.94
Swish	28	2,4	[-5, 5]	0.977	0.034	99.14	99.39	0.12	-0.35	2.56	2.11	-0.22	-1.06
XTanh	27	2,4	[-6, 6]	0.974	0.039	99.12	99.40	0.14	-0.25	2.56	2.18	-0.22	-0.94
XSoftplus	28	2,3	[-10, 10]	0.963	0.052	99.02	99.33	0.24	-0.28	2.58	2.18	-0.23	-1.05
SigTanh	42	2,3,4											
SigSoftplus	42	2,3,4	**Pool. M.**	**count**		**Accuracy (%)**			**Runtime (hr)**			**Accu. Mgn. (%)**		**Rt. Mgn. (%)**	
TanhSoftplus	42	2,3,4				mean	min	max	mean	min	max	mean	min	mean	min
Sigmoid^2	28	2,3	Average	166		99.17	98.17	99.43	2.61	2.18	3.48	0.09	-0.48	-0.27	-1.06
Softplus^2	14	2	Scaled	168		99.07	97.56	99.37	2.50	2.11	2.97	0.20	-0.46	-0.15	-0.73

Poly. degree and count – respectively gives the degree of polynomials derived from the activation function and the number of observations of the approximated polynomial derived from the activation function.
Range and R^2 value – gives the range used for approximation of polynomials and the R^2 value of fitting of the polynomial respectively.
Pool. M. – gives the type of pooling and the count associated gives the number of observations.
Accu., Rt., and Mgn. – means accuracy, runtime, and margin in order.

TABLE 9. Results on MNIST Deducing Efficacy of Polynomial Degrees.

Poly. degree	count	Accuracy (%)				Runtime (hr)				Accuracy Margin. (%)				Runtime Margin (hr)			
		mean	std	min	max	mean	std	min	max	mean	std	min	max	mean	std	min	max
2	138	99.13	0.20	98.12	99.41	2.37	0.16	2.11	2.84	0.09	0.25	-0.48	1.18	-0.06	0.19	-0.53	0.29
3	98	99.14	0.23	97.86	99.43	2.60	0.24	2.23	3.18	0.13	0.23	-0.14	1.46	-0.21	0.29	-0.76	0.38
4	98	99.09	0.35	97.56	99.41	2.76	0.27	2.43	3.48	0.23	0.35	-0.28	1.76	-0.43	0.28	-1.06	0.14

Poly. degree and count – respectively gives the degree of the polynomial derived from the activation function and the number of observations of this particular polynomial degree used in the experiments.

efficient the approximated polynomial makes the training process and this can help us in our decision and design procedures. This metric was considered to ascertain the fact that different approximated polynomials will have dissimilar impacts on the efficiency of neural network training and in cases whereby a trade-off is to be made between classification accuracy and overall runtime of training. The former gives the throughput or say the amount of correct predictions achieved over some time period whiles the latter is tied to the overheads and computational costs. In cases whereby much complex network architectures are employed and/or several activation functions/polynomials are used, this trade-off becomes much vital in design considerations. By comparing with the results given under Section V-B1, we also present the classification accuracy and training time margins. These deviations in the accuracy helps us visualize the efficacy of the approximated polynomials in their being able to measure up to their respective activation functions and this loosely describes how we can relate the approximated polynomials to the activation functions in their approximation capacity. The accompanying runtime margin is able to show how these approximated polynomials fair in being measured in efficiency in comparison to

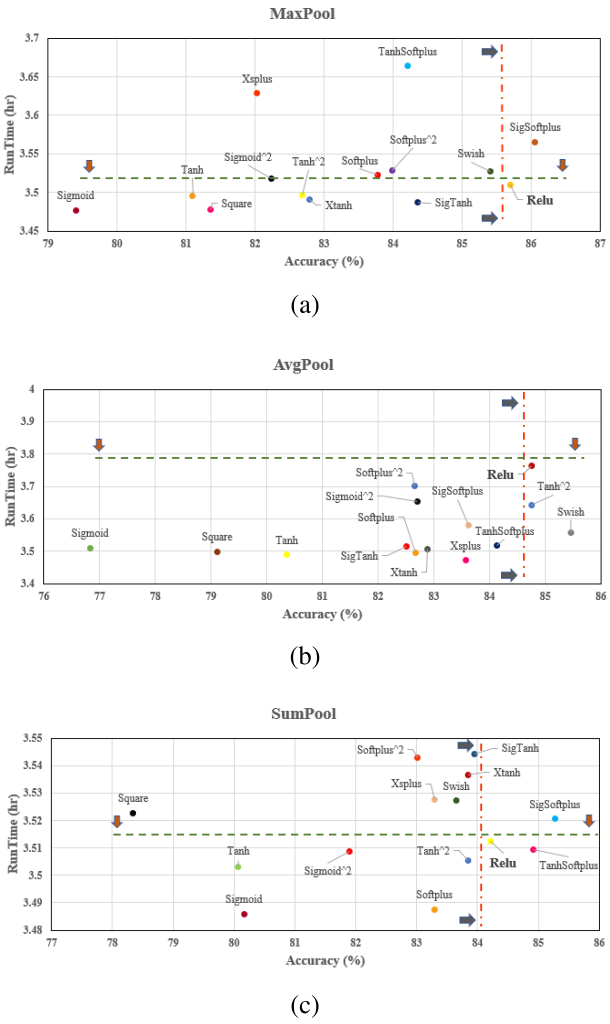


FIGURE 10. Plot of Results in utilizing MaxPooling (a), AveragePooling (b), Scaled-Mean or SumPooling (c), and all the activation functions under the CIFAR-10 category.

the activation functions and from Table 10, we realize that certain polynomials will do better in efficiency as compared to their parent activation function governed by the negative values presented.

Table 11 gives the observations of individual type of approximated polynomial and the number of polynomial degrees used. As we made emphasis on using smaller degree of polynomials (and limiting it to degree 4), the parameters we selected and utilized in our approximation means the achievable degrees of approximation of the approximated polynomial is what is as used and presented in this table (and this is as mentioned in Section IV). In isolating the range in the same table, the strength in approximation is illustrated via the R^2value, where approaching the value 1.0 shows a better approximation and approaching 0.0 otherwise and as is seen, the range $[-1, 1]$ gives a better approximation due to its closeness to the origin of the x-axis (in the $x - y$ plane) and $[-10, 10]$ gave the worst approximation due to being the farthest from 0. Using much constricted ranges (i.e. being more closer to 0 as seen in $[-1, 1]$) is much difficult to utilize in training but as was described in Section IV-B, we can counter against that by using Batchnormalization. From the classification accuracy, we can determine that the ranges of $[-3, 3]$, $[-4, 4]$, and $[-5, 5]$ gives a better chance of yielding a polynomial that can improve performance (via the larger mean values given) though from the maximum values also given, we realize that achieving much higher accuracy is skewed towards the wider ranges from $[-5, 5]$ upwards. Looking at the accuracy margin, we also realize that the ranges $[-3, 3]$, $[-4, 4]$, and $[-5, 5]$ do better generally in performance and with the minimum values, we can find out which ranges had better deviations from the activation functions' accuracy with $[-3, 3]$ being the best owing to its lowest negative value. Touching on the pooling methods, being average and scaled-mean, we found out from the Table 11 that using scaled-mean pooling method generally gave better accuracy whiles average pooling makes training more efficient.

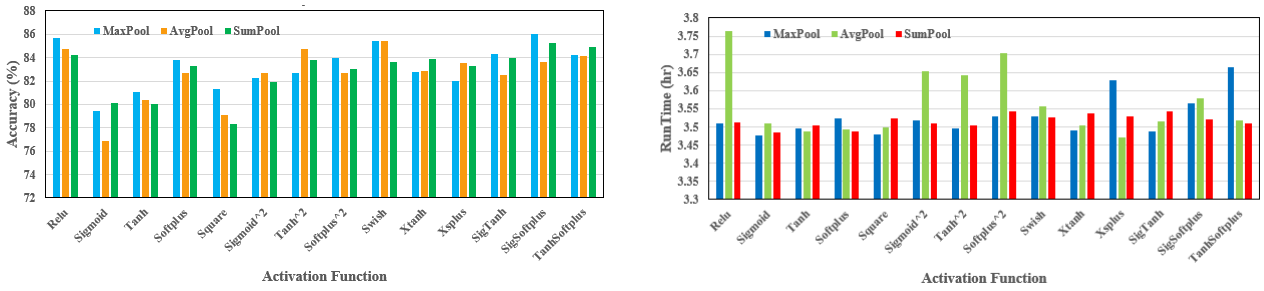
Lastly, in Table 12, the overall performance of the approximated polynomials in their degrees is given. This is very important to note as the selection of the degree of polynomial to use will alter the multiplicative depth of the neural network thus impacting the cost of performing the homomorphic inference. Generally, as the drive is to reduce computational costs via reducing parameters such as the level or multiplicative depth, deriving a much lower degree polynomial which yields the best accuracy is to be striven for. Though we can realize that in Table 12, the degree 2 polynomial has the best yield via its mean and standard deviation, the degree 4 polynomial can produce even better outliers as shown by its maximum value being the highest which thus can be opted for to attain the best yield in the encrypted inference.

VI. HOMOMORPHIC INFERENCE EVALUATION

In this section, we describe the results of our analysis in the homomorphic inference phase. The parameters used for the encrypted MNIST and CIFAR-10 predictions are as shown in Tables 13 and 14. They were derived based on the description given in Section IV and also considering the standard conventions stated by [37]. Both tables satisfy $\lambda = 128$-bit security, which is a very high security level for our inference process.

A. UTILIZATION OF A GRAPH COMPILER FOR ENCRYPTED INFERENCE

Over the past few years, a lot of works have improved upon the first proposed Cryptonets [1] and several improvements have been made. This has been in terms of minimizing the overall execution times in both training and testing phases, reducing memory capacity needed for implementations, and most importantly improving upon the throughput of the network in homomorphic prediction or accuracy and adopting better optimization techniques in the encrypted domain to ameliorate previous proposed works. As more research is being carried out in privacy-preserving machine learning,



(a) (b)

FIGURE 11. Plot of results showing how Average Pooling and Scaled-Mean Pooling measures with MaxPooling on both the accuracy (a) and runtime scales (b) under the CIFAR-10 category.

TABLE 10. Breakdown of Plaintext Results on CIFAR-10 Showing Variations in Comparison to the Conventional Activation Functions.

Approx. Poly.	Accuracy (%)				Runtime (hr)				Accuracy Margin(%)				Runtime Margin (hr)			
	mean	std	min	max	mean	std	min	max	mean	std	min	max	mean	std	min	max
ReLU	78.63	1.92	73.53	81.21	3.80	0.66	3.20	5.27	5.84	1.90	3.08	10.68	0.16	0.62	-0.56	1.51
Tanh	74.11	4.11	64.74	77.35	3.50	0.13	3.23	3.77	6.11	4.11	3.01	15.62	0.004	0.12	-0.27	0.26
Sigmoid	73.51	3.85	65.17	77.07	3.65	0.27	3.26	4.23	4.98	4.29	-0.24	14.96	0.15	0.28	-0.25	0.74
Softplus	76.50	1.97	72.60	79.66	3.62	0.22	3.21	4.13	6.48	1.94	3.63	10.69	0.13	0.22	-0.29	0.64
Swish	79.17	1.85	74.30	82.22	3.58	0.29	3.16	4.16	5.39	2.04	1.63	11.16	0.04	0.30	-0.40	0.63
XTanh	76.06	4.19	65.14	80.31	3.79	0.53	3.08	4.74	7.26	4.03	2.58	17.75	0.27	0.52	-0.42	1.21
XSoftplus	79.59	1.46	74.10	81.56	3.66	0.46	3.13	4.79	3.85	1.50	1.74	9.47	0.16	0.44	-0.34	1.26
SigTanh	76.58	4.50	64.14	81.61	3.63	0.32	3.18	4.64	6.65	4.54	0.90	19.64	0.10	0.31	-0.36	1.10
SigSoftplus	78.73	2.31	71.81	82.35	3.58	0.20	3.35	4.39	5.72	2.44	1.93	11.81	0.03	0.21	-0.23	0.87
TanhSoftplus	78.76	2.36	71.20	82.62	3.40	0.24	3.11	4.03	5.77	2.39	1.91	12.94	-0.12	0.24	-0.41	0.52
Sigmoid^2	74.25	5.30	59.32	79.99	3.48	0.13	3.13	3.80	8.04	5.30	1.91	22.58	-0.10	0.16	-0.53	0.29
Softplus^2	79.39	1.60	74.73	81.48	3.47	0.13	3.11	3.89	3.44	1.64	1.17	8.28	-0.16	0.17	-0.60	0.19

TABLE 11. Plaintext Results on CIFAR-10 Depicting Relationships in Pooling Methods, Ranges, and Degree of Polynomials.

Approx. Poly.	count	Poly. degree	Range	R^2 value		Accuracy (%)		Accu. Mgn. (%)		Runtime (hr)		Rt. Mgn. (hr)	
				mean	std	mean	max	mean	min	mean	min	mean	min
ReLU	25	2,4	[-1, 1]	0.999	0.004	77.56	81.69	5.61	1.49	3.60	3.13	0.06	-0.58
Tanh	14	3	[-2, 2]	0.994	0.010	77.37	80.69	5.87	0.43	3.59	3.11	0.04	-0.60
Sigmoid	14	3	[-3, 3]	0.988	0.020	78.09	80.94	5.17	-0.24	3.62	3.20	0.07	-0.45
Softplus	28	2,4	[-4, 4]	0.982	0.027	78.30	81.30	4.97	1.92	3.45	3.11	-0.09	-0.56
Swish	28	2,4	[-5, 5]	0.977	0.034	78.14	82.62	5.14	1.23	3.65	3.17	0.11	-0.42
XTanh	20	2,4	[-6, 6]	0.974	0.039	77.69	82.23	5.57	0.90	3.50	3.13	-0.05	-0.41
XSoftplus	28	2,3	[-10, 10]	0.963	0.052	74.99	82.22	8.28	1.32	3.64	3.08	0.09	-0.42
SigTanh	40	2,3,4											
SigSoftplus	42	2,3,4											
TanhSoftplus	40	2,3,4											
Sigmoid^2	41	2,3,4											
Softplus^2	42	2,3,4											

Pool. M.	count	Accuracy (%)			Runtime (hr)			Accu. Mgn. (%)		Rt. Mgn. (%)	
		mean	min	max	mean	min	max	mean	min	mean	min
Average	181	77.40	60.97	82.23	3.48	3.08	5.27	5.65	-0.24	-0.09	-0.60
Scaled	181	77.52	59.32	82.62	3.67	3.14	4.79	5.94	1.63	0.15	-0.37

TABLE 12. Results on CIFAR-10 Deducing Efficacy of Polynomial Degrees.

Poly. degree	count	Accuracy (%)				Runtime (hr)				Accuracy Margin. (%)				Runtime Margin (hr)			
		mean	std	min	max	mean	std	min	max	mean	std	min	max	mean	std	min	max
2	137	77.59	3.579	59.32	80.88	3.60	0.31	3.13	5.21	6.02	3.41	1.63	22.58	0.05	0.32	-0.39	1.44
3	112	77.32	3.835	64.74	81.69	3.59	0.35	3.11	4.79	5.11	3.37	-0.24	17.47	0.05	0.36	-0.53	1.26
4	113	77.43	3.769	64.17	82.62	3.54	0.35	3.08	5.27	6.19	3.50	0.90	18.34	-0.01	0.34	-0.60	1.51

we can state emphatically that these systems would become much realised pragmatically by being deployed in real applications. Thus in our work, we utilized the graph compiler proposed by [35], [36] to implement our homomorphic inference as this affords the easy and rapid prototyping of experiments carried out. A compiler such as this, and just like others, abstracts most tedious tasks from the user or programmer thus work can be carried out much faster. Also, the compiler would come shipped with much better features and attributes which greatly improves work and considering one such as this graph compiler, implementations can be done vividly whiles observing standard practices in the field as presented by recognized avenues as [37].

The Intel graph compiler for performing deep learning (DL) tasks on (homomorphically) encrypted data, represented as nGraph-HE, is based on the Intel

nGraph [38], [39] deep learning graph compiler by the same vendor. The latter allows the implementation of models using frameworks such as Tensorflow [23], [24] thus an HE library such as Microsoft SEAL [25], [27], [32] which acts as a backend within nGraph-HE will allow the HE task to be easily implemented and enhance the benchmarking of models in privacy-preserving deep learning domain. As stated in the work of [35], graph compilers such as this one designed for HE-based tasks discriminates high-level based implementations in DL frameworks as Tensorflow from low-level routines in Microsoft SEAL thus executing directly to the HE target. Due to the tight integration between the DL-based end (in the nGraph DL compiler) and the HE-based Microsoft SEAL, one is able to compute tasks homomorphically for a wider range of DL frameworks without much difficulty. nGraph-HE supports most neural network operations - they can be easily mapped to their equivalent in Tensorflow - whiles others such as ReLU and MaxPool are not supported. The existence of instruction representation means we can develop optimized code for operations irrespective of the underlying hardware as compilation will yield a computation graph thus rather making higher-level optimizations the focal point. Most importantly, as the drive over the years is to improve upon the performance of HE-DL models, this HE-based graph compiler helps to induce optimizations in HE via computation depth reduction, parallelizing code as much as possible and so on.

B. ENCRYPTED INFERENCE ANALYSIS

The variation in the degree of polynomials meant we had to manually tweak the parameters before conducting the inference operation as the dissimilar feature present is the degree of polynomial, which affects the multiplicative depth or levels, whiles others such as the network architecture is same in each dataset evaluation. Using the CPU hardware available (Table 6), we performed predictions on 8,192 images, though this can be scaled for increasing amounts of main memory and CPU cores, and we measured the execution time using 128 threads with OpenMP. The CPU hardware given is able to offer up to 2×64 cores of threads for execution thus the 128 threads in total. Considering the numerosity of the plaintext results, we resolved to pick randomly from the lot by considering both the best and worst outcomes. This is to test the efficacy of reproducibility of results in the encrypted domain which necessarily does not merit repeating all observations. The factors we considered were being able to apply the derived polynomial in homomorphic inference, achieving close to or bettering the plaintext classification accuracy, and being able to replicate a plaintext result by picking randomly from the lot whiles not being biased.

In relation to the parameters used for the inference we were able to use batching or the plaintext packing feature of the CKKS scheme [28] to pack as many as $N/2$ complex scalar values into one ciphertext. This thus allowed us to classify all the images in one inference procedure given that in theory, the CKKS encoding mapping implies that $\mathbb{C}^{N/2} \rightarrow \mathcal{R}$ for some plaintext space $\mathcal{R}$ where $N/2$ is the number of slots in the plaintext. We did not utilize *complex packing* [36], an optimization feature that doubles the throughput in inference without ciphertext-ciphertext multiplication. The approximated polynomials require a multiplication operation in some variable say x when raised to some integer power or degree thus as complex packing does not support this, we did not utilize it. We chose a 30-bit prime to be the first and last element in q whiles the intermediate primes were 24-bit. With this, we were able to use at least q and the scale factor of 24 as shown in the Tables 13 and 14 for inference.

TABLE 13. SEAL Parameters for MNIST Homomorphic Inference.

Degree of Polynomial	t or N	q/bits	Scale Factor	Level
2	2^{14} or 16384	228	24	5
3	✓	300	✓	7
4	✓	✓	✓	7

TABLE 14. SEAL Parameters for CIFAR-10 Homomorphic Inference.

Degree of Polynomial	t or N	q/bits	Scale Factor	Level
2	2^{14} or 16384	300	24	8
3	✓	372	✓	11
4	✓	✓	✓	11

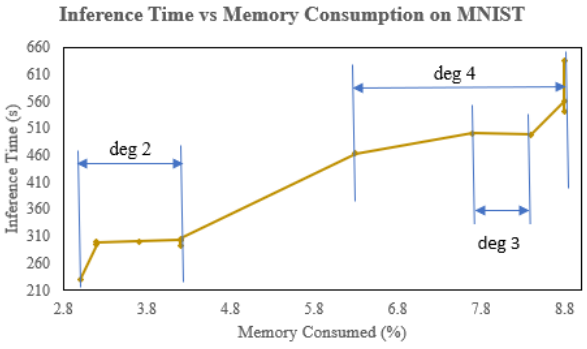
We also stated in Section IV-D that applying *folding* in the activation polynomial function reduces the multiplicative depth and this is as shown in Tables 16 and 18 via the factors of the polynomials. As seen, we have a scalar coefficient which can be evaluated in a convolution and/or dense layer leaving the polynomial with a reduced level in its highest degree to use in computation. Also, it is observed that some polynomials in either a particular degree (or much generally) have varying number of terms as compared to others and this is characteristic of the polynomial being unique to the activation function it was derived from using least squares method. The difference in inference time for the same degree of polynomial used can vary to some extent or be negligible and if it is the former, the inference becomes more significant for much larger network models, complexity of datasets, and the quantity of images to predict.

C. INFERENCE ON MNIST

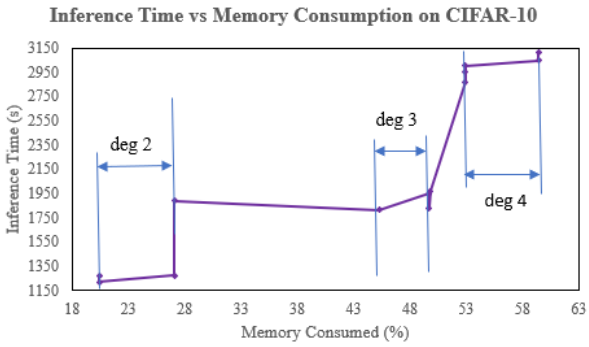
An important baseline we considered was using the square and ReLU functions as references to assess the performance of others. This is as a result of the square being the first proposed function (and mostly used in earlier works) utilized as an approximation to the ReLU whiles the ReLU (considered as one of the best to use) can be assumed that its approximated polynomial equivalent will achieve close to the best performance as compared to others. For logical coherence, we used the same network architecture shown

TABLE 15. Results of Encrypted Inference on MNIST Dataset.

SN	Approx. Polynomial	Poly. degree	Pool. Method	Range	Plaintext Accu. (%)	Memory used in Infer. (%)	Inference Accu. (%)	Inference Time (s)	Amortized Time (s)
–	Square*	–	Average	–	99.04	3.2	99.08	293.564	0.03584
–	Square*	–	Scaled	–	98.80	3.2	98.80	298.602	0.03645
A	ReLU	2	Scaled	[-1, 1]	99.25	4.2	99.25	301.917	0.03686
B	ReLU	4	Scaled	[-1, 1]	99.30	6.3	99.32	463.225	0.05655
C	ReLU	2	Average	[-4, 4]	99.26	4.2	99.30	291.857	0.03563
D	ReLU	4	Average	[-2, 2]	99.35	8.8	99.35	559.033	0.06824
E	Swish	4	Average	[-5, 5]	99.34	8.8	99.31	539.658	0.06588
F	Swish	4	Scaled	[-3, 3]	99.31	8.8	99.30	634.785	0.07749
G	Swish	2	Scaled	[-3, 3]	99.32	4.2	99.31	304.739	0.03720
H	Sigmoid	3	Scaled	[-5, 5]	99.10	8.4	99.10	498.389	0.03686
I	Tanh	3	Average	[-6, 6]	99.40	7.7	99.38	500.193	0.06106
J	XTanh	2	Scaled	[-2, 2]	99.16	3.0	99.16	224.18	0.02737
K	XSoftplus	2	Average	[-2, 2]	99.24	3.7	99.24	300.072	0.03663



(a)



(b)

FIGURE 12. From the sample of results in inference, we illustrate graphically how memory consumed varies with runtime of prediction for encrypted MNIST (a) and encrypted CIFAR-10 (b).

in Table 4 but then this time it was used for inference instead of training as we have our pre-trained weights from the plaintext implementation. Shown in Table 15 is the results of inference on the MNIST dataset with the accompanying polynomials we used in Table 16.

A huge factor in privacy-preserving machine learning is the duplicability of plaintext results in ciphertext domain and thus, as we were able to replicate approximate results, just sampling a few from the pool of results can demonstrate the efficacy of our proposed solution. From Table 15, it can be noted that the inference has a small accuracy tolerance or margin with the amortized time being the time of execution for predicting a single image.

Fig. 2a shows how memory consumption varies with runtime (of prediction) for varying degrees of the approximated polynomial. From the graph, we can note how a polynomial of some degree is expected to consume memory and perform prediction within some boundary. From the degree 2, through the degree 3 to the degree 4 region, we can observe how the curve rises gradually showing how a change in the polynomial degree used affects the strain that is put on resources and the involved costs of computation. The MNIST task, being the less complex one of the two datasets, has much reduced inference time and memory consumption as compared to the more sophisticated CIFAR-10 task as seen in Fig. 12b.

D. INFERENCE ON CIFAR-10

The rather complex dataset meant that obviously the execution time for inference will be much longer and will require more memory in operation but like the MNIST, we also considered the baseline of using the square and ReLU functions as a reference to assess the performance yield from employing the other polynomials derived. In inference, we utilized the network architecture shown in Table 5 and the parameters were set using the values in Table 14. Shown in Table 17 is the results of inference on the CIFAR-10 dataset with the polynomials used shown in Table 18.

In Fig. 12b, the curve which gives the variation of memory consumption with runtime of inference for CIFAR-10 is much steeper with the magnitudes of the parameters being very substantial. The regions of the degree of polynomials where results can be anticipated are clearly delimited and given the nature of the curve, we can clearly tell how much the extent of complexity in the task can have an impact in inference, which herein is protracted with accompanying large amounts of memory being consumed.

TABLE 16. Least Square Polynomial Representation of Functions used on MNIST.

SN	Approximated Polynomial	Factors of Polynomial
A	$0.4286x^2 + 0.5x + 0.0857$	$0.4286(x + 0.957833)(x + 0.208756)$
B	$-0.6667x^4 + 1.1667x^2 + 0.5x$	$-0.6667(x - 1.49998)(x + 0.49998)(x + 1.0)(x)$
C	$0.1115x^2 + 0.5x + 0.3901$	$0.1115(x + 1.00579)(x + 3.47852)$
D	$-0.0466x^4 + 0.4079x^2 + 0.5x + 0.1049$	$-0.0466(x + 1.97331)(x + 1.22776)(x + 0.267848)(x - 3.46891)$
E	$-0.0029x^4 + 0.1655x^2 + 0.5x + 0.0788$	$-0.0029(x + 0.166805)(x - 8.7785)(x + 4.30585 + 0.127589i)(x + 4.30585 - 0.127589i)$
F	$-0.0077x^4 + 0.2177x^2 + 0.5x + 0.013$	$-0.0077(x + 0.0263012)(x - 6.22479)(x + 3.09924 + 0.840788i)(x + 3.09924 - 0.840788i)$
G	$0.1501x^2 + 0.5x + 0.082$	$0.1501(x + 0.172983)(x + 3.15813)$
H	$-0.0041x^3 + 0.1945x + 0.5$	$-0.0041(x + 4.52914)(x + 3.39707)(x - 7.92621)$
I	$-0.008x^3 + 0.4222x$	$-0.008(x - 7.26464)(x + 7.26464)(x)$
J	$0.4629x^2 + 0.1792$	$0.4629(x + 0.622193i)(x - 0.622193i)$
K	$0.5x^2 + 1.019x$	$0.5(x + 2.038)(x)$

TABLE 17. Results of Encrypted Inference on CIFAR10 Dataset.

SN	Approx. Polynomial	Poly. degree	Pool. Method	Range	Plaintext Accu. (%)	Memory used in Infer. (%)	Inference Accu. (%)	Inference Time (s)	Amortized Time (s)
–	Square*	–	Average	–	79.11	20.4	79.11	1272.230	0.15530
–	Square*	–	Scaled	–	78.34	20.4	78.32	1218.376	0.14873
I	ReLU	2	Average	[-5, 5]	79.21	27.1	79.21	1271.185	0.15517
II	ReLU	4	Average	[-5, 5]	81.21	52.9	81.20	2869.135	0.35024
III	Swish	4	Scaled	[-10, 10]	82.02	52.9	82.00	2953.483	0.36053
IV	Swish	2	Average	[-10, 10]	78.31	27.1	78.31	1890.174	0.23073
V	Swish	4	Scaled	[-6, 6]	81.20	52.9	81.20	3002.387	0.36650
VI	Sigmoid^2	3	Average	[-3, 3]	79.60	49.7	79.59	1948.290	0.23783
VII	Tanh	3	Scaled	[-5, 5]	76.73	45.3	76.73	1813.351	0.22136
VIII	TanhSoftplus	4	Average	[-6, 6]	82.23	59.4	82.23	3045.333	0.37174
IX	TanhSoftplus	3	Average	[-4, 4]	80.07	49.8	80.05	1966.535	0.24006
X	Softplus^2	3	Scaled	[-10, 10]	81.21	49.7	81.20	1831.450	0.22357
XI	SigSoftplus	4	Average	[-5, 5]	81.54	59.4	81.52	3107.863	0.37938

TABLE 18. Least Square Polynomial Representation of Functions used on CIFAR-10.

SN	Approximated Polynomial	Factors of Polynomial
I	$0.0899x^2 + 0.5x + 0.4855$	$0.0899(x + 1.25352)(x + 4.30821)$
II	$-0.003x^4 + 0.1592x^2 + 0.5x + 0.297$	$-0.003(x - 8.59115)(x + 0.79074)(x + 3.10111)(x + 4.69929)$
III	$-0.0004x^4 + 0.0873x^2 + 0.5x + 0.3878$	$-0.0004(x - 17.158)(x + 0.924127)(x + 5.93958)(x + 10.2943)$
IV	$0.0481x^2 + 0.5x + 0.7988$	$0.0481(x + 1.97152)(x + 8.42349)$
V	$-0.0018x^4 + 0.1433x^2 + 0.5x + 0.1303$	$-0.0018(x - 10.3501)(x + 0.283633)(x + 4.21174)(x + 5.85474)$
VI	$-0.0091x^3 + 0.0221x^2 + 0.2305x + 0.28$	$-0.0091(x - 6.81088)(x + 1.65872)(x + 2.72359)$
VII	$-0.0129x^3 + 0.4899x$	$-0.0129(x + 6.16253)(x - 6.16253)(x)$
VIII	$-0.0004x^4 - 0.0001x^3 + 0.083x^2 + 0.5083x + 0.5333$	$-0.0004(x - 16.8366)(x + 1.34057)(x + 6.16604)(x + 9.57995)$
IX	$-0.0042x^3 + 0.1183x^2 + 0.5638x + 0.3035$	$-0.0042(x - 32.3811)(x + 0.621018)(x + 3.59346)$
X	$0.0353x^3 + 0.4964x^2 + 1.626x + 0.2432$	$0.0353(x + 0.157012)(x + 4.84069)(x + 9.06462)$
XI	$-0.0021x^4 + 0.0014x^3 + 0.1344x^2 + 0.4659x + 0.397$	$-0.0021(x + 4.7351)(x + 2.9868)(x + 1.3698)(x - 9.75838)$

VII. CONCLUSION AND FUTURE WORK

In our proposed work of implementing image classification using homomorphic encryption, we took the approach of introducing several polynomials into convolutional neural networks whiles combining each instance with other arguments as HE-feasible pooling methods and considered diverse ranges to use for approximation in order to replace the overly used Square and ReLU variants. We also showed that by utilizing the Batch normalization technique and approximating polynomials close to 0, accuracy can be considerably increased. We thus derived a pool of approximated polynomial functions to use for homomorphic inference, with not only being viable to train CNNs in plaintext with a favourable output accuracy, but also capable of replicating approximate results in encrypted domain. It becomes much easier to make design considerations without much limitations especially in terms of selecting the HE-feasible polynomial which impacts the latency, throughput and the hardware resource to utilize.

In comparison to most proposed solutions, our work focused on performing the inference task using the set of procedures stated under Section IV as in some cases performance might drop particularly concerning this area of privacy-preserving machine learning where such instances can be common.

Based on our findings and future research work, we will look into applying approximation methods to other machine learning areas, such as advanced regression and real-world tasks. We will also employ more sophisticated CNN

architectures and use a dataset such as CIFAR-100, which serves as an industry standard and is also a benchmark in the area of machine learning to design more sophisticated models in the application of approximation theory to predictive tasks.

Looking ahead, it would deem fit to apply some robust or state-of-the-art search techniques or algorithms in these predictive tasks to sort of automate the way to generate a much optimal parameter being in the pooling method and the range which yields the best accuracy and makes the task more efficient. If possible, polynomials derived this way should be able to generalize more to several other machine learning tasks implemented like how ReLU does in having a very consistent and desirable effect across most tasks, and not just being tailored to a particular neural architecture being utilized. We will also look at being able to implement this inference task on a GPU as it presents much larger number of cores and thus giving a huge ratio in comparison to the CPU which will therefore greatly improve speedup considerably.

REFERENCES

[1] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 201–210.

[2] C. Gentry, "Computing arbitrary functions of encrypted data," *Commun. ACM*, vol. 53, no. 3, pp. 97–105, Mar. 2010.

[3] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, 2009.

[4] A. Krizhevsky, V. Nair, and G. Hinton. (2009). *The CIFAR-10 Dataset*. Accessed: Oct. 8, 2020. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html

[5] P. Xie, M. Bilenko, T. Finley, R. Gilad-Bachrach, K. Lauter, and M. Naehrig, "Crypto-nets: Neural networks over encrypted data," 2014, *arXiv:1412.6181*. [Online]. Available: http://arxiv.org/abs/1412.6181

[6] E. Hesamifard, H. Takabi, and M. Ghasemi, "CryptoDL: Deep neural networks over encrypted data," 2017, *arXiv:1711.05189*. [Online]. Available: http://arxiv.org/abs/1711.05189

[7] A. Al Badawi, J. Chao, J. Lin, C. F. Mun, J. J. Sim, B. H. M. Tan, X. Nan, K. M. M. Aung, and V. R. Chandrasekhar, "The AlexNet moment for homomorphic encryption: HCNN, theFirst homomorphic CNN on encrypted data with GPUs," 2018, *arXiv:1811.00778*. [Online]. Available: http://arxiv.org/abs/1811.00778

[8] S. Wagh, D. Gupta, and N. Chandran, "SecureNN: 3-party secure computation for neural network training," *Proc. Privacy Enhancing Technol.*, vol. 2019, no. 3, pp. 26–49, 2019.

[9] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "{GAZELLE}: A low latency framework for secure neural network inference," in *Proc. 27th USENIX Secur. Symp. (USENIX Secur.)*, 2018, pp. 1651–1669.

[10] H. Chen, R. Gilad-Bachrach, K. Han, Z. Huang, A. Jalali, K. Laine, and K. Lauter, "Logistic regression over encrypted data from fully homomorphic encryption," *BMC Med. Genomics*, vol. 11, no. S4, p. 81, Oct. 2018.

[11] Z. Zhang, "Revisiting fully homomorphic encryption schemes and their cryptographic primitives," Ph.D. dissertation, School Comput. Sci. Softw. Eng., Univ. Wollongong, Wollongong, NSW, Australia, 2014.

[12] A. Torralba, R. Fergus, and W. T. Freeman, "80 million tiny images: A large data set for nonparametric object and scene recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 11, pp. 1958–1970, Nov. 2008.

[13] E. Y. Remez, "Sur le calcul effectif des polynomes d'approximation de Tschebyscheff," *CR Acad. Sci. Paris*, vol. 199, no. 2, pp. 337–340, 1934.

[14] Y. LeCun, C. Cortes, and C. J. C. Burges. (2019). *The Mnist Database of Handwritten Digits*. Accessed: Oct. 8, 2020. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[15] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[16] T. Ishiyama, T. Suzuki, and H. Yamana, "Highly accurate CNN inference using approximate activation functions over homomorphic encryption," 2020, *arXiv:2009.03727*. [Online]. Available: http://arxiv.org/abs/2009.03727

[17] J. W. Bos, K. Lauter, and M. Naehrig, "Private predictive analysis on encrypted medical data," *J. Biomed. Inform.*, vol. 50, pp. 234–243, Aug. 2014.

[18] J. H. Cheon, J. Jeong, J. Lee, and K. Lee, "Privacy-preserving computations of predictive medical models with minimax approximation and non-adjacent form," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.* Cham, Switzerland: Springer, 2017, pp. 53–74.

[19] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," *ACM Trans. Comput. Theory*, vol. 6, no. 3, pp. 1–36, Jul. 2014.

[20] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *IACR Cryptol. ePrint Arch.*, vol. 2012, p. 144, Mar. 2012.

[21] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015, *arXiv:1502.03167*. [Online]. Available: http://arxiv.org/abs/1502.03167

[22] F. Scheid, *Schaum's Outline of Numerical Analysis*, 2nd ed. New York, NY, USA: McGraw-Hill. 1989, p. 480.

[23] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, and M. Kudlur, "TensorFlow: A system for large-scale machine learning," in *Proc. 12th USENIX Symp. Operating Sys. Design Implem. (OSDI)*, 2016, pp. 265–283.

[24] G. Brain. (2016). *An End-to-End Open Source Machine Learning Platform*. Accessed: Nov. 29, 2020. [Online]. Available: https://www.tensorflow.org/

[25] Microsoft Research. (2017). *Microsoft SEAL*. Accessed: Nov. 29, 2020. [Online]. Available: https://www.microsoft.com/en-us/research/project/microsoft-seal/

[26] S. J. Miller. (2006). The Method of Least Squares. Mathematics Department Brown University. Accessed: Apr. 25, 2020. [Online]. Available: https://web.williams.edu/Mathematics/sjmiller/public_html/BrownClasses/54/handouts/MethodLeastSquares.pdf

[27] H. Chen, K. Laine, and R. Player, "Simple encrypted arithmetic library-SEAL v2. 1," in *Proc. Int. Conf. Financial Crypt. Data Secur.* Cham, Switzerland: Springer, 2017, pp. 3–18.

[28] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Proc. Int. Conf. Theory Appl. Cryptol. Secur.* Cham, Switzerland: Springer, 2017, pp. 409–437.

[29] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Dept. Comput. Sci., Stanford Univ., Stanford, CA, USA, 2009.

[30] C. Gentry and D. Boneh, "Fully homomorphic encryption using ideal lattices," in *Proc. Stoc*, 2009, vol. 9, no. 2009, pp. 169–178.

[31] H. Chabanne, A. de Wargny, J. Milgram, C. Morel, and E. Prouff, "Privacy-preserving classification on deep neural network," *IACR Cryptol. ePrint Arch.*, vol. 2017, p. 35, Mar. 2017.

[32] K. Laine. (2020). *Microsoft SEAL*. Accessed: May 8, 2020. [Online]. Available: https://github.com/microsoft/SEAL

[33] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," 2017, *arXiv:1710.05941*. [Online]. Available: http://arxiv.org/abs/1710.05941

[34] S. Elfwing, E. Uchibe, and K. Doya, "Sigmoid-weighted linear units for neural network function approximation in reinforcement learning," in *Neural Networks*, vol. 107. Amsterdam, The Netherlands: Elsevier, 2018, pp. 3–11.

[35] F. Boemer, Y. Lao, R. Cammarota, and C. Wierzynski, "NGraph-HE: A graph compiler for deep learning on homomorphically encrypted data," in *Proc. 16th ACM Int. Conf. Comput. Frontiers*, Apr. 2019, pp. 3–13.

[36] F. Boemer, A. Costache, R. Cammarota, and C. Wierzynski, "nGraph-HE2: A high-throughput framework for neural network inference on encrypted data," in *Proc. 7th ACM Workshop Encrypted Comput. Appl. Homomorphic Cryptogr.*, 2019, pp. 45–56.

[37] M. Albrecht, M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, S. Halevi, J. Hoffstein, K. Laine, K. Lauter, S. Lokam, D. Micciancio, D. Moody, T. Morrison, A. Sahai, and V. Vaikuntanathan, "Homomorphic encryption security Standard," in *Proc. HomomorphicEncryption*, Toronto, ON, Canada, 2018, p. 33.

[38] S. Cyphers *et al.*, "Intel nGraph: An intermediate representation, compiler, and executor for deep learning," 2018, *arXiv:1801.08058*. [Online]. Available: http://arxiv.org/abs/1801.08058

[39] A. Kulkarni. (2020). *Intel nGraphT Compiler and Runtime for Tensorflow*. Accessed: Dec. 8, 2020. [Online]. Available: https://github.com/tensorflow/ngraph-bridge
[40] F. Chollet. (2020). *Keras: Deep Learning for Python*. Accessed: Dec. 8, 2020. [Online]. Available: https://github.com/keras-team/keras

JONAS T. AGYEPONG received the B.Sc. degree in computer science and engineering from the University of Mines and Technology, Tarkwa, Ghana, in 2017. He is currently pursuing the master's degree with the Department of Computer Science and Engineering, Egypt-Japan University of Science and Technology, Alexandria, Egypt. He was a Research and Teaching Assistant, from 2017 to 2018. Previously, he worked on optimizing renewable power generation systems using machine learning to increase output and cut down costs. His current research interests include privacy-preserving machine learning, with a focus on homomorphic encryption applications in computer vision for which his thesis is based on applications of cryptography in cloud-connected smart devices and high-performance computing on the cloud to improve the performance of such systems incorporating security technologies in machine learning.

MOSTAFA SOLIMAN received the B.Sc. and M.Sc. degrees in computer science and engineering from the University of Assiut, Egypt, in 1994 and 1998, respectively, and the Ph.D. degree in computer science and engineering from The University of Aizu, Japan, in 2004. He is a Full Professor at Aswan University, Egypt. His research interests include computer architecture, parallel processing, vector/matrix processing, performance evaluation, parallel algorithms, FPGA, and SystemC implementations.

YASUTAKA WADA (Member, IEEE) received the B.S. and M.S. degrees in electrical engineering and the Ph.D. degree in computer science and engineering from Waseda University, Tokyo, Japan. He is currently an Associate Professor with the Department of Information Science, Meisei University, Tokyo. Before joining Meisei University in 2015, he was an Assistant Professor at Waseda University, an Assistant Professor at The University of Electro-Communications, Tokyo, and a Junior Researcher and a Research Associate at Waseda University. He was also an Associate Professor at Egypt-Japan University of Science and Technology (E-JUST), Alexandria, Egypt, from 2010 to 2012. He has served as the Program Committee Vice Chair of the IEEE COOL Chips Conference Series, since 2015. His research interests include parallel processing and applications, green computing, heterogeneous computing, automatic parallelizing compilers, and multicore/manycore processor architecture. His research and development activities cover various systems from embedded systems to high-performance computing environments to realize high-performance, energy efficient, and easy-to-use computer systems. He is a member of ACM, IEEE Computer Society, IEICE, and IPSJ.

KEIJI KIMURA (Member, IEEE) received the Ph.D. degrees in electrical engineering from Waseda University, in 2001. He was an Assistant Professor, in 2004; an Associate Professor, in 2005; and a Professor, in 2012, at Waseda University. He has been the Director of the Green Computing System Research Organization, Waseda, since 2019. His research interest includes multicore processor architecture and parallelizing compiler technologies. He is a member of IPSJ and ACM. He has served on the program committee of many conferences. He was a recipient of 2014 Ministry of Education, Culture, Sports, Science and Technology in Japan (MEXT) Award.

AHMED EL-MAHDY (Senior Member, IEEE) received the B.Sc. and M.Sc. degrees from Alexandria University and the Ph.D. degree from the School of Computer Science, The University of Manchester, U.K., where he contributed to one of the early multicore processors (JAMAICA). He is a Full Professor at the Computer Science and Engineering Department, Egypt-Japan University of Science and Technology (E-JUST). He is also on leave from the Computer and Systems Engineering Department, Alexandria University. He has visited the Group of Advanced Processor Technologies contributing to porting the IBM Jikes dynamic compiler for JAMAICA. He has also been a Visiting Scientist at IBM Centre for Advanced Studies, Cairo, where he was the first inventor of many issued patents in high-performance computing and image processing. He is currently the Founding Director of the Parallel Computing Laboratory at E-JUST, with many funded research grants/support from IBM, Amazon, ITIDA, STDF, Academy of Science, and Technology in embedded compilers, high-performance GPU acceleration, and high-performance computation on the cloud. He is a Senior Member of the ACM. He is also a TPC Member of ICCD and ARCS conferences.