DOReN: Toward Efficient Deep Convolutional Neural Networks with Fully Homomorphic Encryption

Souhail Meftah, Benjamin Hong Meng Tan, *Member, IEEE*, Chan Fook Mun, Khin Mi Mi Aung, *Senior Member, IEEE*, Bharadwaj Veeravalli, *Senior Member, IEEE*, and Vijay Chandrasekhar, *Senior Member, IEEE*

***Abstract*— Fully homomorphic encryption (FHE) is a powerful cryptographic primitive to secure outsourced computations against an untrusted third-party provider. With the growing demand for AI and the usefulness of machine learning as a service (MLaaS), the need for secure training and inference of artificial neural networks is rising. However, the computational complexity of existing FHE schemes has been a strong deterrent to this. Prior works suffered from accuracy degradation, lack of scalability, and ciphertext expansion issues. In this paper, we take the first step towards the problem of space-efficiency in evaluating deep neural networks through designing DOReN: a low depth, batched neuron that can simultaneously evaluate multiple quantized ReLU-activated neurons on encrypted data without approximations. Our circuit design reduced the complexity of the accumulator circuit depth from $O(\log m \cdot \log n)$ to $O(\log m + \log n)$ for n bit integers. The experimental results show that the amortized processing time of our homomorphic neuron is approximately 1.26 seconds for 300 inputs and less than 0.13 seconds for 10 inputs at 80 bit security, which is a 20 fold improvement upon Lou and Jiang, NeurIPS 2019.**

***Index Terms*— Fully homomorphic encryption, neural networks, depth optimization, ReLu.**

I. Introduction

ARTIFICIAL Neural Networks (ANNs) have been used on a variety of tasks such as computer vision, neural machine translation and speech recognition. They have proved to be especially effective in the fields of healthcare, finance, cybersecurity and advanced manufacturing. However, these domains tend to work with private, integrity-sensitive data which have to be handled with care. In the case of healthcare and finance sectors [1], [2], patients' personal identifiable information, clinical and banking data are highly sensitive and if leaked can lead to serious harm to the patients as well as the institutions. In advanced manufacturing [3], [4], many sensitive trade secrets and intellectual properties are being used in each product. If such data is compromised by adversaries, there could be substantial potential loss in revenue and competitive advantage. Similarly, in the cybersecurity field [5]–[7], confidentiality and integrity of the data can be intrinsic requirements to many processes and programs ranging from government and military applications to enterprise governance and extending to personal data-driven applications.

Cloud computing promises high flexibility, availability, scalability and reliability, without the need to buy, own and maintain hardware. Companies can leverage its capabilities to reduce capital and operating costs, outsourcing storage, and computation of business data to the cloud. With artificial neural networks being computationally intensive, the cloud can be an enabler for its use. However, a big stumbling block to wider adoption of the cloud is the need for privacy and security for sensitive data.

Currently, new modes of collaboration and services that leverage the cloud are being proposed; with *Machine Learning as a service* (MLaaS) being one such innovation to enable the use of computationally expensive artificial neural networks on the cloud. This is a common model for applications of machine learning such as personalized medicine services and recommendation systems. In such scenarios, users would send data that is potentially sensitive to the cloud for processing but this data must be revealed to neither the cloud nor the model owners.

Fully Homomorphic Encryption (FHE) as a technology lends itself as a natural fit to addressing the issue of privacy in these situations as it enables data processing without decryption, thereby preserving the privacy of the sensitive data with encryption. By sending users' encrypted data along with a set of evaluation keys to the cloud, it enables the cloud to compute on that encrypted data and send results back without additional communication or decryption during

Manuscript received January 7, 2021; revised April 11, 2021; accepted June 3, 2021. Date of publication June 21, 2021; date of current version July 28, 2021. The work of Benjamin Hong Meng Tan, Chan Fook Mun, and Khin Mi Mi Aung was supported by the Agency of Science, Technology and Research (A*STAR) through the RIE2020 Advanced Manufacturing and Engineering (AME) Programmatic Programme under Award A19E3b0099. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Daniele Venturi. *(Corresponding author: Souhail Meftah.)*

Souhail Meftah is with the Institute of Infocomm Research, Agency of Science, Technology and Research (A*STAR), Singapore 138632, and also with the Department of Electrical and Computer Engineering, National University of Singapore, Singapore 119077 (e-mail: souhailmeftah@u.nus.edu).

Benjamin Hong Meng Tan, Chan Fook Mun, Khin Mi Mi Aung, and Vijay Chandrasekhar are with the Institute of Infocomm Research, Agency of Science, Technology and Research (A*STAR), Singapore 138632.

Bharadwaj Veeravalli is with the Department of Electrical and Computer Engineering, National University of Singapore, Singapore 119077.

Digital Object Identifier 10.1109/TIFS.2021.3090959

computation. However, the current paradigm for constructing homomorphic encryption schemes presents a few challenges. Current schemes support limited computation expressivity due to efficiency issues and employ an expensive refreshing operation called *Bootstrapping* to evaluate more complex programs and algorithms. As ANNs typically require many layers to present useful insights and achieve good levels of accuracy, bootstrapping appears to be required to evaluate such networks on encrypted data. The current state of the art for evaluating deep networks on encrypted data leaves much to be desired in terms of computational and communication complexity, resource usage, and total running time.

Various attempts have been made to design efficient methods for evaluating neural networks with homomorphic encryption. First, leveraging highly efficient homomorphic operations on numbers for fast neural network evaluations, Dowlin *et al.* [8] introduced CryptoNets for the MNIST dataset, opting to approximate the non-polynomial activation functions such as ReLU which are incompatible with the available homomorphic operations (addition and multiplication). Many have extended their ideas to larger datasets and improve performance but unfortunately this approach has not been shown to achieve accurate inference on the ImageNet dataset.

On the other hand, some have opted to adapt methods from the deep learning community that sought to improve the efficiency of neural networks. Bourse *et al.* [9] considered binarized neural networks [10] which are neural networks where activations and weights are all single bits and therefore much easier to work with. However, their experiments indicated that accuracy was degraded and it was unclear if the methods could scale to the deeper networks needed for accurate inference on the ImageNet dataset. In a recent work, Lou and Jiang proposed to evaluate quantized neural networks with the TFHE [11] scheme. They were able to exactly replicate the plaintext operations with homomorphic circuits with their quantization scheme but did not successfully evaluate full networks with the gate-bootstrapped TFHE scheme, which were projected to take at least several months to complete. They managed to evaluate ShuffleNet [12] with a leveled variant of TFHE [11] in around 5 hours, but SHE suffers from large ciphertext expansion overheads, needing 7.7 GB for an encryption of a single image.

Our Contributions and Strategy. In this work, we propose DOReN, a new method for evaluating a convolutional neural network (CNN) with homomorphic encryption, which has lower ciphertext expansion overhead and improved performance compared to previous approaches involving bootstrapping. To that end, we design a depth-optimized accumulate-then-activate circuit to minimize the number of bootstrapping operations needed to evaluate the neuron on encrypted integers. This is achieved by using 3:2 compressors instead of full adders at the initial stages of accumulations unlike previous proposals. On top of that, we use the Brakerski-Gentry-Vaikuntanathan (BGV) [13] scheme which supports encryption and homomorphic operations on vectors of bits and adopted a novel packing strategy to maximize the amortized performance of neuron evaluation, instead of minimizing the latency of homomorphic circuits in Cheon *et al.* [14] and Liu *et al.* [15]. Our approach requires adjusting the location of data packed in the output ciphertexts of one layer to the input ciphertexts for the next. To that end, we introduce inter-layer transformations to bridge outputs of one layer to inputs of the next. Finally, we analyze the security of our methods and perform some experiments to evaluate the efficiency of our neuron.

For circuit-based approaches, the bulk of the evaluation time is in the accumulation of weighted inputs, especially for schemes where the weights are applied with bit-shifts instead of integer multiplication. Therefore, it is very important to optimize the depth of the accumulator, which is related to the number of bootstrapping operations required to compute it. The full adders proposed Cheon *et al.* [14] and Liu *et al.* [15] for n-bit inputs require $O(\log n)$ depth and halves the number of summands with each use. 3:2 compressors, which are essentially carry-save full adders, transform 3 summands into 2 which is less efficient but only use 1 depth regardless of input bit-widths and are highly parallel. Once the summands have been compressed to two large bit-width inputs, we apply a full adder to compute the final value.

Besides that, DOReN uses SIMD packing differently to previous proposals, which focused on lowering the latency of the homomorphic circuits by packing multiple bits of the same input into a single ciphertext. This may be effective when we want to be able to compute the circuits for arithmetic quickly, but deep CNNs typically have upwards of hundreds of neurons and it would be more important to optimize for amortized performance of these circuits. Better amortized performance allows us to complete entire layers of neural networks more effectively and therefore achieve a lower network evaluation time overall.

In the convolution layers of a neural network, neurons can be divided into channels that share a set of associated weights $w_i{}_{i=0}^{m}$ with w_0 corresponding to the bias, and differ only in the neurons from the previous layer that are connected to them. We pack the inputs to this layer such that we simultaneously evaluate as many neurons from a single channel as possible, i.e. for neurons $j = 0, 1, \ldots, \ell$, with inputs labeled as $x_{i,j}$, we compute $f(w_0 + \sum_{i=1}^{m} w_i x_{i,j})$, for some activation function f, with one set of homomorphic operations. Thus, ciphertexts would encrypt plaintext vectors organized in the style of $(x_{i,0}, x_{i,1}, \ldots, x_{i,\ell})$, assuming there were ℓ neurons in a single channel.

Finally, ciphertext outputs in our system are not correctly formatted inputs for subsequent convolution or pooling layers. Some neurons in the same ciphertext would be inputs to the same neuron and so need to be moved to the same slot but in different ciphertexts. Therefore, we propose to use a set of inter-layer transformations, which consists of a collection of Benes permutation networks to transform the outputs of one layer into a set of input ciphertexts for the next layer.

Related Work. Integer arithmetic and comparison operations on encrypted data for privacy-preserving applications have been explored in several papers. Cheon *et al.* [14] proposed the use of SIMD packing by Smart and Vercauteren [16] to improve the performance of computing equality and order comparisons as well as fixed bit-width integer addition. They

showed that the latency of such operations could be significantly reduced with SIMD. Liu *et al.* [15] further considered signed integers and integer addition that accommodated for overflows. However, these two work are not optimized for accumulations of large number of integers as they exclusively use full adders for accumulation. Full adders require $O(\log n)$ depth in its SIMD optimized form per use and therefore require $O(\log n \cdot \log m)$ depth to accumulate m n-bit inputs, necessitating the use of many bootstrapping operations. Cheon *et al.* [14] proposed an alternative to use the integer ring $\mathbb{Z}_{2^{15}}$ for accumulations, but how to recover a low bit-width result in this setting is not clear.

Currently, there are two general trends for neural network evaluation with homomorphic encryption. The first targets efficient evaluation of inference, beginning with CryptoNets [8]. This approach uses polynomials to approximate ReLU or sigmoid activations which is incompatible with the fast integer addition and multiplication that word-size homomorphic encryption can provide. This has been extended to support batching techniques by others [17]–[19] for faster evaluation times. Unfortunately, this line of work has not been shown to be capable of scaling to deep networks such as those for ImageNet.

Besides that, there have been attempts to achieve encrypted inference that can scale to arbitrary-depth neural networks by adapting techniques from the deep learning community aimed at reducing the computational demands of traditional neural networks. Bourse *et al.* [9] adapted the idea of binarized neural networks [10] and proposed discretized neural networks which they evaluated using the TFHE scheme [11]. They showed that a basic network of 100 neurons can compute inference for encrypted MNIST images with an accuracy loss of around 2-3%. However, they did not consider deeper neural networks that could perform inference for larger datasets, which will likely suffer greater accuracy loss.

Recently, Lou and Jiang [20] proposed SHE, building circuits for ReLU neuron evaluation with the TFHE [11] scheme and exploiting quantization [21] to reduce the complexity of neuron evaluation. Although their proposed architecture could be evaluated with the TFHE scheme, they did not manage to evaluate the complete network and only gave projected timings. However, they successfully evaluated ShuffleNet [12] using a leveled variant of TFHE [11]. As TFHE has large ciphertext expansion overheads, around 16 KB to encrypt a single bit, 7.7 GB is required to encrypt a single image for inference. Furthermore, leveled FHE schemes have limited depth, although the parameters specified by TFHE support up to 32K. Deeper networks with more capacity can necessitate even larger parameter sets which mean even bigger overheads and compounds the size of encrypted data for inference.

Batching with Leveled TFHE. Although batched homomorphic operations were proposed for TFHE in Chillotti *et al.* [22], the model of computation is very different from the circuit model used with the BGV, BFV, and gate-bootstrapped TFHE schemes. They proposed accelerated look-up table evaluations for random functions on one variable with horizontal and vertical packing techniques. The former allows the same look-up table to be evaluated on multiple data simultaneously while the latter enables faster evaluations of a single look-up table on one piece of data. The operations used to evaluate neural networks are generally not univariate functions and thus the benefits of packing with leveled TFHE in this setting are not clear.

Other methods of exploiting the available plaintext space in leveled TFHE were proposed. Deterministic weighted finite automata (det-WFA) takes advantage of the available plaintext space to evaluate functions representable with det-WFA more efficiently but does not permit evaluating the automata on more than one set of inputs at once. Bit sequence representation was introduced to evaluate multi-addition and other arithmetic operations on encrypted more efficiently than det-WFAs but again does not permit evaluation of the homomorphic operations on multiple inputs simultaneously.

Outline of the Paper. In Section II, we introduce some background that will be used throughout the paper: fully homomorphic encryption, depth-optimized Boolean circuits, and quantized convolutional neural networks. Next, in Section III, we present optimized architectures for the various neural network layers used in convolutional neural networks. After that, in Section IV, we introduce how batching and the optimized neural network layers are put together into a complete neural network for encrypted data. Then, in Section V, we report the experiments that we conducted to evaluate the performance of the proposed architecture, how it compares to the SHE neuron architecture, and how it scales to full neural networks. Finally, we provide some concluding remarks in the last section.

TABLE I
NOTATION USED IN THE PAPER

Notation	Meaning
$x \in \{0,1\}, x \in [0, 2^n)$	Plaintext x
$\overline{x}, x \in \{0,1\}$	Encryption of x
$\overline{x_i}, x \in [0, 2^n)$	Encryption of bits of x_i, $(\overline{x_{i,0}}, ..., \overline{x_{i,n-1}})$
$\overline{(x^{(1)}, ..., x^{(\ell)})}$ $x^{(i)} \in \{0,1\}$	Encryption of the vector $(x^{(1)}, ..., x^{(\ell)})$
$\oplus, \wedge$	XOR, AND operations
$b \cdot x$, $b \in \{0,1\}, x \in [0, 2^n)$	b AND each bit of x, i.e. $(b \wedge x_i)_{i=0}^{n-1}$
$+, \times$	Add, Mul operations
$\log$	Logarithm with Base 2
$\log_q$	Logarithm with Base q

II. BACKGROUND

System Model. The system model for MLaaS consists of a client and a server. The client has data on which they wish to apply the machine learning model. The server, which might be on the cloud, runs the model on the client's data. In this situation, the client does not want to reveal any information

about the data that they want to apply the model on to the server and potential adversaries.

To address this data privacy and security concern, we consider the scenario where the client encrypts their data with fully homomorphic encryption and provides the encrypted data along with the evaluation keys necessary for the server to compute on it without decryption. This model has been considered in several previous works [8], [17]–[20], [23].

A. Fully Homomorphic Encryption

A fully homomorphic encryption (FHE) scheme consists of four probabilistic, polynomial-time algorithms.

- $\mathsf{KeyGen}(1^\lambda, \alpha)$: Generates the keys needed for encryption (pk), decryption (sk) and homomorphic evaluation (evk), using a security parameter λ and an auxiliary input α for specifying the maximum circuit depth that can be evaluated;
- $\mathsf{Encrypt}(pk, m)$: Encrypts a message m into a ciphertext $\overline{m}$ with pk;
- $\mathsf{Decrypt}(sk, \overline{m})$: Decrypts a ciphertext $\overline{m}$ into a message m with sk;
- $\mathsf{Eval}(evk, f, \overline{m_1}, \ldots, \overline{m_n})$: Computes a ciphertext $\overline{f(m_1, \ldots, m_n)}$ with evk, given a function f and n input ciphertexts $\overline{m_1}, \ldots, \overline{m_n}$, such that $\mathsf{Decrypt}(sk, \overline{f(m_1, \ldots, m_n)}) = f(m_1, \ldots, m_n)$.

For FHE schemes such as BGV [13] and BFV [24], [25], Eval is done by expressing functions as Boolean circuits and using homomorphic additions and multiplications to evaluate XOR and AND gates on encrypted bits.

Single Instruction Multiple Data (SIMD) or Batching Plaintexts with FHE. Some FHE schemes, such as BGV [13] and BFV [24], [25], can encrypt a vector of bits unlike TFHE [11]. This is because the plaintext spaces of these schemes has additional structure to it. For plaintext modulus t, the plaintext space is $R_t = \mathbb{Z}_t[X]/\Phi_m(X)$, where $\Phi_m(X)$ is the m-th cyclotomic polynomial.

By Chinese Remainder Theorem, $\Phi_m(X) = \prod_{i=1}^{\ell} f_i(X)$ modulo t. The ℓ factors $f_i(X)$ are each degree d polynomials with $\ell \cdot d = \phi(m)$ and $\phi(\cdot)$ being Euler's totient function. This means that $R_t \cong \prod_{i=1}^{\ell} \mathbb{Z}_t[X]/f_i(X) \cong \mathbb{F}_{t^d}$ and we can encrypt ℓ elements of the finite extension field $\mathbb{F}_{t^d}$ in a single ciphertext and have component-wise addition and multiplication. Besides these operations, entries of the encrypted vectors can be manipulated through shifts and rotations following Gentry *et al.* [26].

To efficiently handle arbitrary permutations on the slots of encrypted vectors with the operations available with FHE, Halevi and Shoup [27] proposed the use of Benes networks. These are realized with shifts/rotations, multiplicative masking and additions, costing a minimal amount of noise and maximum computational complexity of $O(\log n)$ shifts/rotations and masking.

Bootstrapping. Current FHE schemes are based on noisy encryptions of plaintexts, and; homomorphic operations cause the noise embedded in ciphertexts to grow. After some number of operations, especially multiplications, noise levels in ciphertexts become very large and the decryption algorithm will no longer be able to recover the correct plaintext. Bootstrapping, introduced by Gentry [28], keeps noise growth from homomorphic operations in check to ensure correct decryption even after many rounds of homomorphic operations. The main idea is to homomorphically evaluate the decryption of ciphertexts with an encrypted secret key. This removes the original noise in the ciphertexts and introduces a moderate amount of noise from the homomorphic decryption process but the new noise levels are still low enough for further use.

Unfortunately, this is expensive computationally for HE schemes that support batching but recently Chen and Han [29] described a thin bootstrapping method that supports much faster bootstrapping of vectors of base field elements, i.e. plaintext vectors of the form $(\mathbb{F}_t)^\ell$ rather than the complete plaintext space $(\mathbb{F}_{t^d})^\ell$.

B. Quantized Convolutional Neural Networks

A convolutional neural network consists of several neurons that are organized into layers, according to what functions the neurons serve. Some of the most commonly used layers in this kind of neural networks include convolution, max-pooling, and fully connected layers.

- Neurons in the max-pooling layer take the outputs of a subset of neurons $\{x_i\}_{i=1}^m$ from the previous layer and returns the largest among them,

$$output = \max(x_1, \ldots, x_m).$$

- Neurons in the convolution (fully-connected respectively) layer takes the outputs of a subset (all respectively) of the neurons $\{x_i\}_{i=1}^m$ from the previous layer and performs a weighted-sum on them followed an activation function f on the sum,

$$output = f\left(\sum_{i=1}^{m} w_i x_i\right).$$

One of the most prevalent activation functions for convolutional neural networks is the rectified linear unit (ReLU). In this work, our focus is on neurons in the convolution or fully-connected layers with the ReLU activation function, which we call ReLU-neurons.

$$output = \mathsf{ReLU}\left(\sum_{i=1}^{m} w_i x_i\right), \ \mathsf{ReLU}(z) = \begin{cases} z, & \text{if } z \geq 0; \\ 0, & \text{otherwise.} \end{cases}$$

Conventionally, neural networks are designed to use floating-point numbers for inputs, activations, and weights. However, this is a big contributor to the high computational requirements and memory use of large networks. For small, computationally limited devices, many have been working to reduce the computational and memory requirements and one of the main techniques is quantization. In this approach, inputs, activations, and weights use low precision number representations such as 8-bit integers and logarithmic representation as these are much easier to work with and smaller than 32-bit/64-bit floating-point numbers.

Logarithmic Representation for Quantized Weights. In improving the performance of evaluating ReLU-neurons without sacrificing accuracy, Zhou *et al.* [21] showed how to incrementally quantize traditional convolutional neural networks to have weights that are either zero or some power of two, i.e. $w_i \in \{0, \pm 2^{n_1}, \ldots, \pm 2^{n_2}\}$ for integers $n_1 < n_2$. Besides reducing the bit-width required to store the weights, the weighted sum operation uses much simpler bit-shifts instead of integer multiplication, reducing the computational complexity significantly.

Handling Quantized Inputs and Activations. With inputs being expressed as fixed bit-width integers, we require intermediate accumulations of the weighted inputs of ReLU-neurons to grow so that the accuracy of the network is not severely compromised. Then, once the weighted sum is complete, we evaluate ReLU on it and apply a scale factor, a power of two $2^{-\sigma}$, where $\sigma \geq 0$, to recover a low bit-width integer.

C. Boolean Circuits on Encrypted Bits

The main operations in ReLU-activated neural networks are weighted sums and order comparisons, "$\geq$" in particular. With logarithmic quantization, applying the weights are performed with bit-shifts so we focus on Boolean circuits for comparisons and accumulation. Let $x = \sum_{i=0}^{n-1} x_i 2^i$ be an n-bit integer and an encryption of x, which we denote with $\overline{x}$ when there is no confusion, would be the vector of ciphertexts, $(\overline{x_0}, \ldots, \overline{x_{n-1}})$.

Depth-Optimized Comparisons and Max Poolings. Comparisons are very important in convolutional neural networks. Besides the ReLU activation function, there are pooling layers where multiple neurons are combined by some function. For ReLU, checking if a number was greater than 0 only requires checking if its two's complement representation has the most significant bit set to 0. However, one of the most popular functions for pooling is the maximum function, where only the largest value in the set of input neurons is propagated to the next layer. Thus, depth-optimized comparison Boolean circuits for encrypted integers are necessary to achieve this efficiently with SIMD-capable FHE schemes.

For completeness, we describe the less than circuit ($x < y$) proposed by Cheon *et al.* [14] but without the use of SIMD to minimize the latency of its computation. Let $\mathsf{LT}(\overline{x}, \overline{y}) = \overline{1}$ if $x < y$ and $\overline{0}$ otherwise, and $\overline{x} = (\overline{x_0}, \ldots, \overline{x_{n-1}})$ and $\overline{y} = (\overline{y_0}, \ldots, \overline{y_{n-1}})$. We compute it as

$$\mathsf{LT}(\overline{x}, \overline{y}) = \overline{\ell_{n-1}} \oplus \bigoplus_{i=0}^{n-2} \left(\overline{\ell_i} \bigwedge_{j=i+1}^{n-1} \overline{d_j} \right),$$

where $\overline{\ell_i} = (1 \oplus \overline{x_i}) \cdot \overline{y_i}$ and $\overline{d_i} = 1 \oplus \overline{x_i} \oplus \overline{y_i}$ for $0 \leq i \leq n-1$. The above can be evaluated in $\log(n-2)+1$ depth by evaluating each of the $\bigwedge_{j=i+1}^{n-1} \overline{d_j}$ terms by multiplying the individual components in a binary tree.

From this, the maximum of two inputs, a key component of max pooling layers, can be computed as

$$\mathsf{max}(\overline{x}, \overline{y}) = \mathsf{LT}(\overline{x}, \overline{y}) \cdot \overline{y} + \left(\overline{1} \oplus \mathsf{LT}(\overline{x}, \overline{y})\right) \cdot \overline{x}.$$

Depth-Optimized Full Adders. The simplest full adder and one that uses the least operations is the ripple carry. It applies the 1-bit full adder iteratively from the least significant bit to the most significant, propagating the carry to the next level as we go along. However, this also means that it requires the most depth, linear in the bit-width of the inputs.

For a low depth full adder circuit for adding two encrypted integers, $\overline{x} = (\overline{x_0}, \ldots, \overline{x_{n-1}})$ and $\overline{y} = (\overline{y_0}, \ldots, \overline{y_{n-1}})$, we follow the methods of Cheon *et al.* [14]. The idea is to "unroll" the iterative application of the 1-bit full adder. First, we compute the modulo 2 sums $\overline{x_i} \oplus \overline{y_i}$ and carries $\overline{x_i} \cdot \overline{y_i}$ of each position. Then, we use the modulo 2 sums $\overline{x_i} \oplus \overline{y_i}$ to determine carry propagation,

$$\overline{p_{j,k}} = \bigwedge_{i=j+1}^{k-1} (\overline{x_i \oplus y_i}) = \begin{cases} 1, & \text{if carry at } j \text{ propogates to } k; \\ 0, & \text{otherwise.} \end{cases}$$

These can be computed in $\log(k-j)$ depth by multiplying the components pair-wise in a tree. Finally, we combine all the components we computed, $\{\overline{x_i \oplus y_i}\}_{i=0}^{n-1}$, $\{\overline{x_i \wedge y_i}\}_{i=0}^{n-1}$ and the carry propagations for all necessary pair-wise combinations $\{\overline{p_{j,k}} \mid 1 \leq j \leq n-1 \text{ and } j < j \leq n-1\}$, obtaining an encryption of

$$z = x + y = \sum_{i=0}^{n} z_i 2^i, \overline{z} = (\overline{z_0}, \ldots, \overline{z_n}), \text{ where}$$

$$\overline{z_0} = \overline{x_0 \oplus y_0}, \quad \overline{z_i} = \overline{(x_i \oplus y_i)} \oplus \bigoplus_{j=0}^{i-1} \left(\overline{(x_j \cdot y_j)} \wedge \overline{p_{j,i}}\right);$$

for $1 \leq i \leq n$, with $x_n = y_n = 0$.

Per Cheon *et al.* [14], this circuit has $\log(n-2)+1$ depth.

3:2 Compressor. The 3:2 compressor is a low-depth method of reducing the number of summands in an accumulator by 2/3 by expressing the (integer) sum of three numbers as the sum of two. We first describe the case for 1-bit inputs. Let $x, y, z \in \mathbb{Z}_2$ and $\oplus$ denote the XOR operation, then

$$\overline{x} + \overline{y} + \overline{z} = \overline{a_1} \times 2 + \overline{a},$$

where $\overline{a_1} = \overline{xy} \oplus \overline{z}(\overline{x} \oplus \overline{y})$ and $\overline{a_0} = \overline{x} \oplus \overline{y} \oplus \overline{z}$. With monomials that only involve products of two bits, between $\overline{x}$ and $\overline{y}$ and $\overline{z}$ and $\overline{x} \oplus \overline{y}$, the 3:2 compressor only requires 1 depth.

Crucially, we can extend the 3:2 compressor to work for n-bit inputs without increasing the multiplicative depth. The main idea is to apply the 1-bit compressor bit-sliced, taking the three bits that correspond to the same power of two. For $x = \sum_{i=0}^{n-1} x_i 2^i$, $y = \sum_{i=0}^{n-1} y_i 2^i$, $z = \sum_{i=0}^{n-1} z_i 2^i$, we get

$$\overline{x} + \overline{y} + \overline{z} = \overline{a_1} \times 2 + \overline{a_0},$$

where $\overline{a_{1,i}} = \overline{x_i y_i} \oplus \overline{z_i}(\overline{x_i} \oplus \overline{y_i})$ and $\overline{a_{0,i}} = \overline{x_i} \oplus \overline{y_i} \oplus \overline{z_i}$, for $i \in \{0, \ldots, n-1\}$. We can also express $\overline{a'_1} = \overline{a_1} \times 2 = \sum_{i=0}^{n} a'_{1,i} 2^i$, with $a'_{1,0} = 0$ and $a'_{1,i} = a_{1,i-1}$ for $i \in \{1, \ldots, n\}$.

III. Depth-Optimized Neural Network Layers

To control the noise growth of ciphertexts during homomorphic computation, we apply bootstrapping at various points in the various neural network layers. With FHE schemes that support batching, for efficiency, circuits of limited depth are typically evaluated between consecutive bootstrappings instead of single gates with a tradeoff between requires parameters

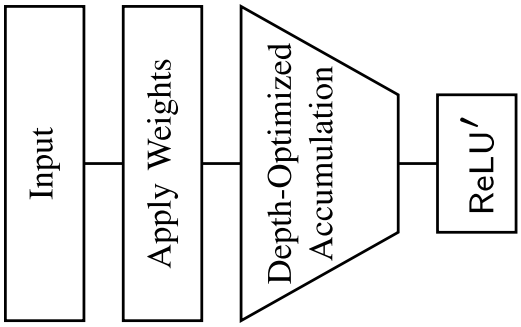

Fig. 1. Depth-Optimized ReLU-Activated Neuron.

sizes and depth supported. Larger parameter sizes increases both the ciphertext expansion factor and computational complexity of bootstrapping, which ranges from several seconds to minutes, from experiments by Halevi and Shoup [30].

Due to this difference in approach to using bootstrapping, the method by Lou and Jiang [20] of using ripple-carry full adders, which requires depth linear in bit-width, for accumulation is not optimal for schemes with batching capabilities. TFHE also has a leveled variant that uses a controlled multiplex gate (CMux), equipped with circuit bootstrapping [22] to enable unlimited CMux operations, but applying this paradigm for neural network evaluation is out of scope for this work.

In this section, we develop a highly-parallelizable, depth-optimized architecture for neural network layers tuned for the FHE schemes that support batching. To simplify the exposition for this section, we use $\overline{x}$ to denote the set of ciphertexts encrypting x in a bit-sliced manner, i.e. $(\overline{x_0}, \overline{x_2}, \ldots, \overline{x_{b-1}})$ for $x = \sum_{i=0}^{b-1} x_i 2^i \in [0, 2^b)$.

A. ReLU-Activated Convolution & Fully Connected Layers

The convolution and fully connected layers are concerned with computing weighted sums of outputs of neurons from the previous layer, with the main difference being the number of inputs. Neurons in the convolution layer take a subset of the neurons the previous layer, called filters or kernels, while those in the fully-connected layer take all neurons of the previous layer. This weighted sum would then be passed through the ReLU activation function to yield the final output.

The general equation for neurons in these layers, for m inputs $\{x_i\}_{i=1}^m$, associated weights $\{w_i\}_{i=1}^m$ and bias b is

$$\overline{output} = \mathsf{ReLU}\left(\sum_{i=1}^{m} w_i \times \overline{x_i} + b\right).$$

For quantized networks, we need to recover a low bit-width integer from the activation, which requires applying a scale factor and then truncation to a specified bit-width. Therefore, we consider the evaluation of the above equation and quantization in four parts, applying weights, accumulation, evaluating ReLU, and lastly extracting a low bit-width result. This flow is summarized in Fig. 4.

Applying Weights to Neuron Inputs. With weights in logarithmic representation, applying them to encrypted inputs is straightforward for positive-weighted inputs, being a simple bit-shift operation. For negative-weighted inputs, however, there are two possibilities. We can use two's complement representation to encode them or use two accumulators, shifting all inputs by the appropriate amount and then separately summing positive and negative-weighted inputs before subtracting the latter from the former. In this work, we adopt the two's complement representation to reduce the computational complexity of the convolution operation.

Applying positive weights to inputs only requires setting $w_i\overline{x_i} = (0, \ldots, 0, \overline{x_{i,\lfloor \log w_i \rfloor}}, \ldots, \overline{x_{i,\lfloor \log w_i \rfloor + b - 1}})$, so there is no depth required. However, for negative weights, we need to compute a 2's complement negation, which is done by computing $2^b - |w_i| \cdot \overline{x_i} \bmod 2^b$ using a full adder and $O(b + \log |w_i|)$ depth.

Depth-Optimized Accumulation. There are two main approaches to compute the sum of m b-bit integers, one of which is to iteratively apply full adders to pairs of integers until only one is left, which was proposed by Cheon *et al.* [14] and Liu *et al.* [15]. With full adders requiring at least $O(\log b)$ circuit depth for b-bit integers, accumulators of this kind have high circuit depth, $O(\log m \cdot \log b)$. This is the approach used in SHE by Lou and Jiang [20], but for SIMD-capable schemes, this approach is prohibitively expensive due to the high depth which necessitates many bootstrapping operations.

Instead, we exploit compressors to design an accumulator that is optimized for much lower depth, recursively reducing the number of summands to 2 before applying a full adder in an accumulation tree as shown in Fig. 2. The summands are reduced by $2/3$ with each iteration, reaching 2 inputs after around $\log_{3/2} m$ iterations. This meant that the compression to two integers require $O(\log m)$ depth and we then need one $O(b + \log m)$-bit full adder to complete the accumulation. Altogether, this means that the compressor-based accumulator only uses $O(\log m + \log(b + \log m))$ depth, compared to $O(\log m \cdot \log b)$ of the full adder-based accumulator.

Lou and Jiang proposed to use mixed bit-width accumulators, where the output of each layer were allowed to grow in bit-width rather than use the maximally expected output bit-width. We can use the same idea with our accumulator, but that would actually over-estimate the maximally expected output bit-width because the summands were reduced by $2/3$ instead of $1/2$. For our depth-optimized accumulator, we note that the maximally expected output bit-width, n_{end}, depends only on the number of inputs. In general, we may need one more bit to hold the output of the sum of two numbers. Therefore, we can upper bound the bit-width of all values in our accumulator given m n-bit inputs with $n_{\mathsf{end}} = b + \lceil \log m \rceil$.

For an illustration of this, suppose we have 6 inputs of 1-bit each as in Fig. 2. In this case, $n_{\mathsf{end}} = 1 + \lceil \log 6 \rceil = 4$, which is achieved with a full adder-based accumulator. However, if we simply allowed the intermediate values to grow after each compressor or full adder, the encrypted output would have 5 bits instead.

Evaluating the ReLU Function. The ReLU as defined earlier, can be done using an encrypted comparison with 0. With n_{end}-bit outputs from the accumulator, the comparison circuit would require $\mathcal{O}(\log n_{\mathsf{end}})$ depth. However, we design a variation with only one depth, exploiting the fact that the most significant bit (MSB) of a two's complement encoded

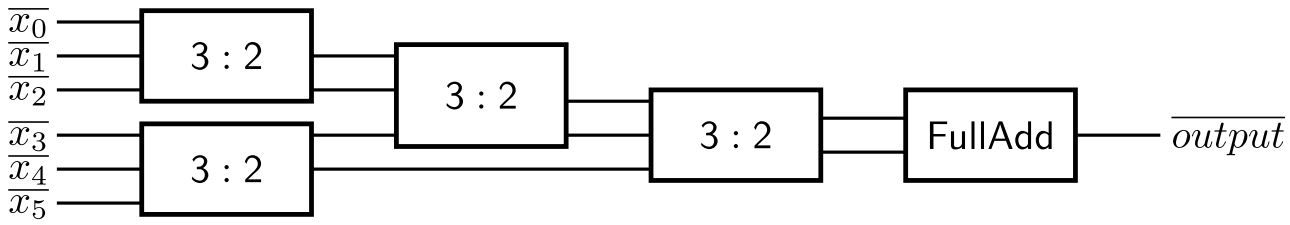

Fig. 2. Example of a 6 input accumulator tree with 3:2 compressors.

integer represents its sign, to express $\mathsf{ReLU}(\overline{s})$ for some $s = \sum_{i=0}^{n_{\mathsf{end}}-1} s_i 2^i$ as

$$\mathsf{ReLU}(\overline{s}) = (1 \oplus \overline{s_{n_{\mathsf{end}}-1}}) \cdot \overline{s}, \text{ where } \overline{s_{n_{\mathsf{end}}-1}} = \begin{cases} \overline{1}, & \text{if } s < 0; \\ \overline{0}, & \text{otherwise.} \end{cases}$$

$1 \oplus \overline{s_{n_{\mathsf{end}}-1}}$ negates the encrypted sign bit of s and we multiply it to $\overline{s}$ by multiplying it to each of $\overline{s_0}, \ldots, \overline{s_{n_{\mathsf{end}}-1}}$.

Recovering a Low Bit-Width Output. For quantized convolutional neural networks, we apply a scale factor, $2^{-\sigma}$ which is equivalent to keeping the bits at positions b, for $\sigma \leq b \leq n_{\mathsf{end}} - 2$. Besides that, we have to clamp outputs whose absolute values are greater than $2^B - 1$ to $2^B - 1$. To achieve that, we test if the output has at least one "1" in its higher-order bits, ignoring position $n-1$ because it is the sign bit for two's complement representation.

$$\mathsf{Clamp}(\overline{x}) = \left(1 \oplus \prod_{i=B}^{n_{\mathsf{end}}-2} (1 \oplus \overline{x_i})\right) \cdot (2^B - 1) + \left(\prod_{i=B}^{n_{\mathsf{end}}-2} (1 \oplus \overline{x_i})\right) \cdot \mathsf{Truncate}(\overline{x}),$$

where $x = \sum_{i=0}^{n_{\mathsf{end}}-1} x_i 2^i$ and $\mathsf{Truncate}(\overline{x}) = (\overline{x_0}, \ldots, \overline{x_{B-1}})$ for $n_{\mathsf{end}} \geq B$. To optimize depth, we fold Clamp into ReLU,

$$\mathsf{ReLU}'(\overline{s}) = (1 \oplus \overline{s_{n-1}}) \cdot \mathsf{Clamp}(\overline{s}) = \begin{cases} \overline{2^B - 1}, & \text{if } s \geq 2^B; \\ \overline{s}, & \text{if } 0 \leq s < 2^B; \\ \overline{0}, & \text{otherwise.} \end{cases}$$

B. Max Pooling Layers

Max pooling layers are straightforward, given a set of inputs $\{\overline{x_1}, \ldots, \overline{x_m}\}$, output the largest value in the set. Therefore, we simply apply the max circuit described in Section II-C.

With inputs being low bit-width and pooling "filters" being small, we minimize the depth of the comparison circuit but optimize the number of homomorphic comparisons together with depth. To that end, we put the inputs at the leaves of a binary tree and eliminate the lower value as we move up the tree, with the maximum value surviving to the root. For example, with 2×2 filters and input set $\{\overline{x_1}, \overline{x_2}, \overline{x_3}, \overline{x_4}\}$, we compute the max pooling of the set with $\mathsf{Max}(\mathsf{Max}(\overline{x_1}, \overline{x_2}), \mathsf{Max}(\overline{x_3}, \overline{x_4}))$.

C. Bootstrapping in the Neural Network Layers

To simplify implementation and optimization considerations, we apply bootstrapping to the outputs of the neurons in any layer, so inputs to all neural network layers in our system are always at similar noise levels.

Balancing Circuit Depth and Scheme Parameters. On top of that, we note that most of the components of neural network layers in our architecture are either constant depth, such as the 3:2 compressor or the sign function, or logarithmic depth in the input bit-width, like the full adder, ReLU' and maximum circuits. This gives us a natural circuit depth to aim for between consecutive bootstrappings; the depth needed to evaluate a full adder on n_{end} bits. In practice, we can expect that the outputs of a quantized neural network to be at most 32-bits, as typical inputs are 8-bit integers with weights of between 5–8 bits, allowing for up to several hundred thousand inputs to neurons in the convolution or fully connected layers.

When to use Bootstrapping. With the intermediate depth being able to accomodate any logarithic depth circuit used in our architecture, we enforce that inputs to full adders, ReLU' and maximum circuits are freshly bootstrapped so that bootstrapping is not needed within these functions. Note that we do not apply bootstrapping to the inputs of the 2's complement negation circuit during weight application as it is already assumed to be freshly bootstrapped.

Now, the only parts of the various networks that has not been addressed are the inter-layer transforms and the layers of 3:2 compressors that are used to compress the weighted inputs to just two values. In these parts, we use bootstrapping dynamically when needed instead of manually fixing bootstrapping stages within the accumulator. This allows us to simplify the overall design of the network as the number of inputs to the accumulator depends on its layer's hyperparameters and be difficult to manually design bootstrapping placements for each possible configuration.

We note that there are heuristics [31] to estimate worst-case noise levels in BGV or BFV ciphertexts. Therefore, we propose to track these heuristics during homomorphic computation within the accumulator and apply bootstrapping to ciphertexts when noise estimates suggest that further homomorphic operations are not possible. This way, we can accomodate any filter size without significant changes to the design of the accumulator by adding a conditional bootstrapping to the a_0 output of the compressor in Section II-C; the only output that requires multiplications to compute.

IV. Efficient Neural Networks with SIMD

Within each layer of a neural network, there are typically upwards of hundreds and thousands of neurons. Thus, it is more important to consider the amortized performance of neuron evaluations rather than the latency, so that the

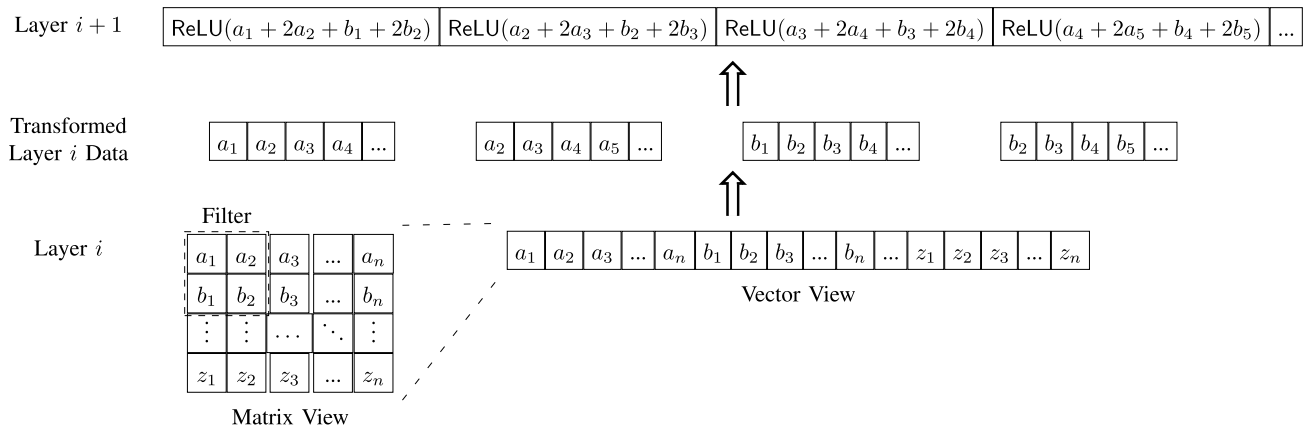

Fig. 3. Example of a Transformation between Layers.

entire layer is completed as soon as possible. Tan *et al.* [32] found that it was more effective for amortized performance to evaluate more of the same circuit rather than use batching to get the lowest latency possible. To simplify the exposition for this section, we use $\overline{x}$ as in Section III and additionally use $\overline{(x^{(1)}, \ldots, x^{(\ell)})}$ to mean the set of ciphertexts encrypting a batch of bit-sliced integers $x^{(i)} \in [0, 2^b)$ for $1 \leq i \leq \ell$.

A. Batching Neurons in Max Pooling Layers

There are two main types of layers as discussed in Section III, convolution and fully connected layers, and max pooling layers, where neurons evaluate $\mathsf{ReLU}'(\sum_{i=1}^{m} w_i \overline{x_i})$ and $\mathsf{max}(\overline{x_1}, \ldots, \overline{x_m})$ respectively. Batching works best when the same computation is applied to every entry of the plaintext vector packed in a single ciphertext.

For max pooling layers, we can achieve this very easily. Using "2×2" filters to illustrate the method, we evaluate $\mathsf{max}(\overline{x_1}, \overline{x_2}, \overline{x_3}, \overline{x_4})$ in the max pooling layer. To simultaneously evaluate ℓ max pooling neurons, we assign each to a slot index and pack the x_j's of each neuron in a ciphertext at that neuron's slot index, i.e. we compute $\mathsf{max}(c_1, c_2, c_3, c_4)$, where "ciphertexts" $c_j = \overline{(x_j^{(1)}, \ldots, x_j^{(\ell)})}$ for $j = 1, \ldots, 4$. The result of $\mathsf{max}(c_1, c_2, c_3, c_4)$ is $\overline{(o^{(1)}, o^{(2)}, \ldots, o^{(\ell)})}$, where $o^{(i)} = \mathsf{max}(x_1^{(i)}, x_2^{(i)}, x_3^{(i)}, x_4^{(i)})$.

B. Batching Neurons in ReLU-Activated Layers

The considerations for batching in the convolution and fully connected layers are similar to the max pooling layer. The operations within neurons in these layers are exactly the same, applying weights, accumulating the weighted inputs and then evaluating ReLU'. However, there are some subtle differences due to the fact that not all neurons in these layers share the same weights. This complicates batching in these layers since we would need to perform slightly different operations, shifting by different amounts or evaluating the two's complement negation circuit, for differently weighted neurons.

Structure of Inputs to Neurons in a Convolution Layer. For image classification with neural networks, the neurons in each layer are arranged in a three dimensional structure of height H, width W and channel C. For convolution layers, neurons in these layers take a subset of neurons in the previous layer as inputs called a filter which is a cuboid of height h, width w and channel C. All neurons in the same channel share the same set of weights and only differ in the location of the filter in the two dimensional grid of size $H \times W$. Sometimes, the filter goes beyond the bounds of the grid. In those cases, we simply pad the grid around the edges with 0.

Packing. Following the principle of having the same computation occur across all the slots of every ciphertext, we assign a slot index to each convolution neuron in the same channel and compute $\overline{(o^{(1)}, o^{(2)}, \ldots, o^{(\ell)})} = \mathsf{ReLU}'(c_1, \ldots, c_m)$, where $c_j = \overline{(x_j^{(1)}, x_j^{(2)}, \ldots, x_j^{(\ell)})}$, for $1 \leq j \leq m$. Each output slot would contain $o^{(i)} = \mathsf{ReLU}'(\sum_{j=1}^{m} w_j x_j^{(i)})$.

In the event there are more slots than number of neurons in a single channel, it is possible to pack neurons from different channels in the same ciphertexts. We would apply masks to separate out slots belonging to different channels, if they have different weights associated with the ciphertext. Note that it would be best to have each ciphertext correspond to the same weight for optimal performance but failing that, having weights with the same sign grouped together would be the next best outcome. This would reduce the number of ciphertexts that require the two's complement negation step which requires logarithmic depth in the bit-width of the weighted inputs.

C. Patching the Format of Ciphertexts for Different Layers

By packing inputs so that we maximize the number of neurons being evaluated simultaneously, we now have to cater for the the ciphertext inputs having a different packing structure from the outputs of batched neurons. For example, in the convolution layers, we pack neurons from the same channel in the same ciphertext but this is not the structure expected by subsequent convolution layers; each neuron in a filter is expected to be in a different ciphertext. We propose a new transformation layer to rectify this situation, see Fig. 3 for an illustration. These are a series of permutations on cyphertext vectors to move the data into the format expected by the neurons in the next layer.

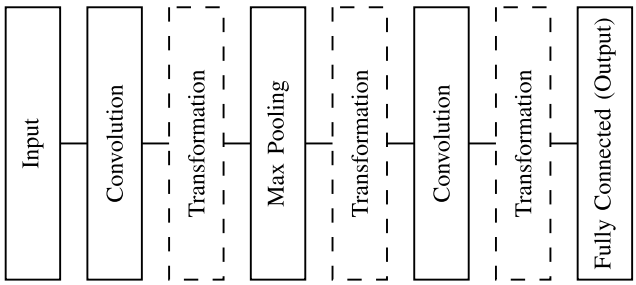

Fig. 4. Example Convolutional Neural Network for Encrypted Data.

To achieve that, we implement general Benes networks that enable us to arbitrarily permute the slots in a cyphertext usuing only rotations, multiplicative masking and additions. For $n = 2^r$, a Benes network for a pernutation π on $[n]$ is a shift network of depth $2r - 1 = O(logn)$ and costs at most $4r - 2 = O(logn)$ [27]. The recursive nature of the network and unique path binding each input-output pair allows for the implementation of SIMD operations.

In our design, we first define a permutation π_i describing the input-output correlation between two given layers L_i and L_{i+1}. Then, we generate a Benes network for π_i, a shift networks in which each level λ_i, is a shift column holding values δ_j, $j \in \{1, \ldots, \ell\}$ by which elements should respectively be shifted. We then generate a mask $M \in \{0, 1\}^\ell$ to isolate the input vector values for which the shift values δ_j are identical and apply the shift operation on the masked vector. Finally, we calculate the sum of all the shifted masked vectors, feed the resulting vector to the next level of the general Benes Network, and repeat the above steps for all λ_i's.

D. Putting Everything Together

DOReN's quantized convolutional neural networks for encrypted data are built from four types of layers: convolution, fully connected, max/average pooling and inter-layer transformation layers. The first three, described in Section III, are optimized for depth and computation complexity and evaluate batches of neurons simultaneously with the use of batching. The last type of layer, also optimized for depth, bridges the difference in batching expected of inputs in the next layer and outputs of the typical neural network layers.

Packing the Inputs to a Neural Network. For inputs to the neural network, we can prepare them so that there is no transformation needed to process the inputs to the first layer. Assuming that there are sufficient slots to evaluate all the neurons in one channel simultaneously, we only need $h \times w \times c \times b$ ciphertexts, where h, w, c are the height, width and channel parameters of the first layer's filter and b is the bit-width of the input integers.

These ciphertexts would be sufficient to compute all neurons in the first convolution layer. Since the only difference between the channels of the first layer are the weights that are applied to the inputs, we simply use the same set of inputs and vary the weights.

Security Analysis. With an honest but curious server, we assume that they would strictly follow the algorithms for evaluating the neural network on encrypted data. However, the server is also free to try to infer information about the encrypted data based on the ciphertexts sent from the client and any intermediate or final results from evaluating the neuron.

In our neural network architecture, the inputs and all intermediate values is protected by encryption with a fully homomorphic encryption scheme (BGV in our implementation). For the server to be able to extract any information about the data being evaluated without access to the client's private key, they would have to first break the IND-CPA property of the scheme and solve the Ring Learning with Errors problem underlying the scheme's instantiation. More information on the security of the BGV scheme and the RLWE problem can be found in Brakerski *et al.* [13] and in Brakerski [24] and Fan and Vercauteren [25] for the BFV scheme. For any other honest but curious adversaries, the situation is the same, since they have the exact capabilities of the server, except they may not have access to the client's public and evaluation keys. The IND-CPA property of the FHE scheme prevents them from decrypting the ciphertexts sent between the client and servers to obtain their underlying plaintexts without access to the private key of the client.

V. Experiments

The experiment platform is an Intel® Xeon® Platinum 8170 with maximum turbo frequency of 3.7 GHz and 192 GB RAM. The prototype implementation uses up to 104 threads with OpenMP.

We evaluate a neuron with inputs of $b = 8$ bits wide and weights in the range of $\{\pm 2^4, \pm 2^3, \ldots, \pm 2^1, 0\}$. Inputs are bit-sliced and packed into 8 ciphertexts, one for each bit of their binary representation, with up to ℓ many inputs packed in a single ciphertext. Crucially, this unlocks the use of the slim bootstrapping mode from Chen and Han [29] since plaintext elements lie in $(\mathbb{F}_2)^\ell \subset (\mathbb{F}_{2^d})^\ell$.

We target scheme parameters so that the deepest circuit, which is the final full adder in the depth-optimized accumulator, between bootstrappings. Recall that depth-optimized variants of this circuits require $O(\log n_{\mathsf{end}})$ depth where n_{end} is the maximum bit-width of intermediate values in the accumulator.

We used $p = 2$, $m = 21845$ and capacity of 400 for HElib, giving us the ciphertext space consisting of polynomials with degree less than 16384. These parameters yield a ciphertext modulus that is less than 412 bits and allows us to pack 1024 bits in a single ciphertext and gives us a ciphertext expansion factor of around 6600. Therefore, we can simultaneously evaluate 1024 neurons with a security level of around 80 bits according to `lwe-estimator` by Albrecht *et al.* [33], Albrecht [34].

With multi-threading, we note that it takes about 10-15 seconds on average to bootstrap the set of ciphertexts associated with a single integer in the network.

Reducing Ciphertext Expansion. As these parameters were chosen to support limited-depth circuits between bootstrapping, we get around 125 bits of usable capacity between consecutive bootstrappings. To reduce the amount of

communication required to send encrypted data, the inputs to the network can be instead encrypted at this reduced capacity. With this optimization, the ciphertext expansion factor is reduced to around 2000.

Note that the techniques of Tan *et al.* [32] can be applied to pack the entire 8-bit input integers into a single ciphertext instead of relying on separately encrypting them. This comes with a slight cost, before the inputs can be used with our neural network architecture, it has to be pre-processed. We need to run the extract algorithm on the packed ciphertext to recover the set of 8 ciphertexts each encrypting single bit of the input integers. Doing this can further decrease the ciphertext expansion factor to $2000/8 = 125$. We do not consider this optimization in the rest of the section as our focus is on the performance of the neural network.

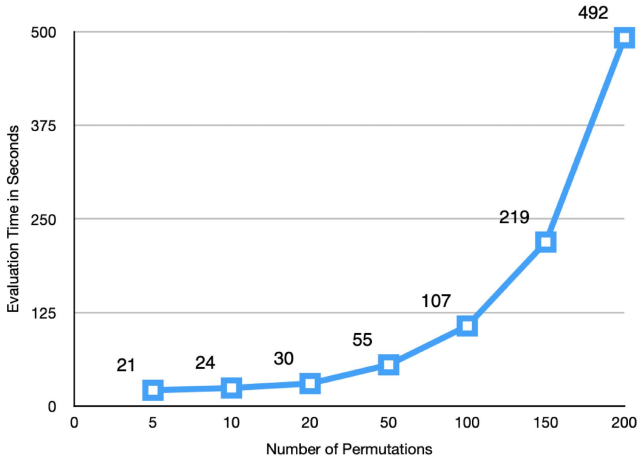

Fig. 5. Performance Evaluation of Benes Permutations.

A. Neuron Performance

Implementation Details.

DOReN's architecture can be broken down into three stages: (1) applying weights to the inputs; (2) accumulating the weighted sums; (3) deriving the activation output from the intermediate results; (4) transforming the inter-layer cyphertext vectors.

Within the depth-optimized accumulator, we enforce that inputs to the final full adder and ReLU' circuits are freshly bootstrapped ciphertexts. This is to allow us to implement these circuits without the need to add additional bootstrapping operations within. However, that is not the only depth-intensive part of the neural network.

Thus, we chose to apply bootstrapping dynamically at the internal nodes, as and when ciphertexts can no longer support additional homomorphic operations. To estimate when this happens, we use the capacity parameter that all ciphertexts have in HElib.

In HElib, which implements the BGV scheme, there is a modulus switching step to ensure that the noise in the ciphertexts do not grow too large. From HElib 1.0.0, the strict levelled nature of the BGV implementation was changed to one where ciphertexts have a capacity instead. Modulus switching is more dynamic now and HElib will reduce a ciphertext's capacity after multiplication based on an estimate on its current noise variance. Therefore, in our neuron implementation, we check that capacity of the a_1 output is above a certain limit (roughly 20) and apply bootstrapping if it falls below that before continuing with accumulation.

Performance Analysis. Table II shows the performance of our neuron architecture for different input sizes. We take the time taken to complete various stages of the neuron to analyze the bottlenecks in the DOReN architecture. In our current implementation, the accumulator assignment is not multi-threaded and takes less than 30ms to process each input. At larger input sizes, this step begins to take an increasing amount of time. Yet, it remains negligible in proportion to the accumulation time (<1%).

The second stage is the most time-consuming stage, as this is where most of the multiplication gates are and several rounds of bootstrapping are required during the accumulation. Of the reported times, roughly 50 seconds is spent on the mandatory bootstrapping of the inputs to the root node and outputs of the accumulator. With a small number of inputs and multi-threading to parallelize full adder operations, the number of inputs does not significantly impact this portion of the accumulation. Furthermore, this is effectively a fixed cost and gradually becomes a minor part of the accumulation time, despite it representing around 20–50% of accumulation time for smaller input sizes of 10 to 40.

Based on the results in Table II, although the input size is doubled with each row, the accumulation time does not increase proportionally. This is because accumulation comprised mainly of layers of independent 3:2 compressor operations, which greatly benefits from the use of SIMD operations. This effect begins to taper when we go from 80 inputs to 160 and 300, where the amount of computation begins to overwhelm the number of threads available on the experiment platform.

The final stage of DOReN, corresponding to *Bootstrapping Time* and *ReLu Activation Time* is very fast and in fact, appears to be independent of input size. This is due to the fact that the bit-width of the inputs to ReLU depends logarithmically on the number of inputs and so the number of gates necessary is fairly constant. Furthermore, there are sufficient threads in the system to parallelize most of the operations in this stage and so the small increase in gate count did not affect the run time. Overall, accumulation is the most computationally intensive part of the architecture (taking between 84–97.5% of the neuron evaluation time) and there is room to improve the implementation with better input handling.

Inter-layer Transforms. Our experimentation shows that the time required to complete bootstrapping after a single permutation using Benes networks ranges between 9–11s. Generally, permutations run independently across neurons and are highly parallelizable. As shown in Figure 5, the amortized evaluation time of a random permutation of size 1024 approaches 1 second the more permutations are being executed in parallel. Once the number of threads available (104 in our experimental setup) is exceeded, we notice an increasing gap in processing time. Practically, the number of permutations to be executed per-layer in most modern real-world DNN architectures can reasonably be upper-bounded to 100 permutations/layer.

TABLE II
NEURON PERFORMANCE FOR VARIOUS INPUT SIZES

Input Size	Apply Weights	Accumulation Time	Bootstrapping Time	ReLU Time	Total Time	Amortized Time
10	0.28	109.83	10.9	9.08	130.13	0.13
20	0.54	174.9	12.03	9.56	197.05	0.19
40	1.12	238.75	11.72	9.47	261.08	0.25
80	2.44	372.098	13.33	10.72	398.70	0.38
160	4.58	687.78	13.87	10.52	716.815	0.70
300	7.97	1259.42	12.93	10.31	1290.73	1.26

Timings are given in seconds.

TABLE III
CIFAR10 RESULTS (E. INPUT SIZE: SIZE OF ENCRYPTED INPUTS IN BYTES; #ADDITIONS: NUMBER OF HOMOMORPHIC ADDITIONS; #P/C MUL: NUMBER OF MULTIPLICATIONS BETWEEN A PLAINTEXT AND A CIPHERTEXT; #C/C MUL: NUMBER OF MULTIPLICATIONS BETWEEN TWO CIPHERTEXTS; #SHIFTS: NUMBER OF SHIFTS BETWEEN A CIPHERTEXT AND A PLAINTEXT; #HOP: TOTAL NUMBER OF HOMOMORPHIC OPERATIONS; MUL DEPTH: MULTIPLICATIVE DEPTH

Network	E. Input size	#Additions	#P/C Mul	#C/C Mul	#Shifts	Total #HOP	Mul Depth	Latency(s)	Accuracy(%)
VGG7	1.32G	3.41M	298k	2.02M	13.17M	5.73M	1305	329k	84.76
ResNet20	1.32G	2.66M	159k	1.59M	2.80M	4.42M	5339	200k	87.84
Lola	9.42G	1.13M	213k	501k	1.05M	1.84M	2577	80.5k	81.60
GHE	3.68G	438k	40.9k	248k	885k	727k	990	35.1k	73.85
SHE	1.32G	1.56M	115k	930k	3.82M	2.60M	3244	132k	92.32

B. Comparison with SHE

Besides evaluating the performance of our proposed neuron, we compare it with an implementation of the neuron architecture introduced by Lou and Jiang [20]. Their use of the TFHE scheme precludes them from applying the batching techniques that are available to the BGV/BFV schemes. However, they make up for that with fast gate-bootstrapping and can be easily used to run Boolean circuits on encrypted data. The main difference between our architecture and that of Lou and Jiang [20] is that they use deep circuits to compute their neuron. Full adders are used to convert negative-weighted inputs to their two's complement form and so only a single accumulator is necessary.

We implemented the gate-bootstrapped SHE neuron using the TFHE [35] and OpenMP libraries and obtained the results contrasted in Figure 6, using the same parameters as [20]. Although the paper [20] wrote that their parameters had at least 128-bit security, they have been downgraded to 80-bit security due to new attack models, according to the function `default_80bit_gate_bootstrapping_parameters` in `tfhe_gate_bootstrapping.cpp` of the TFHE library [35].

Overall, batching significantly improves the performance of our neuron, yielding amortized performance of $10-20\times$ faster compared to SHE's neuron performance as illustrated in Figure 6. Just like our architecture, the bulk of the computation in TFHE lies in the accumulation of the weighted inputs. The main difference lies in the use of consecutive full adders in the accumulation tree, which was optimized for low gate counts to minimize the number of gate bootstrappings

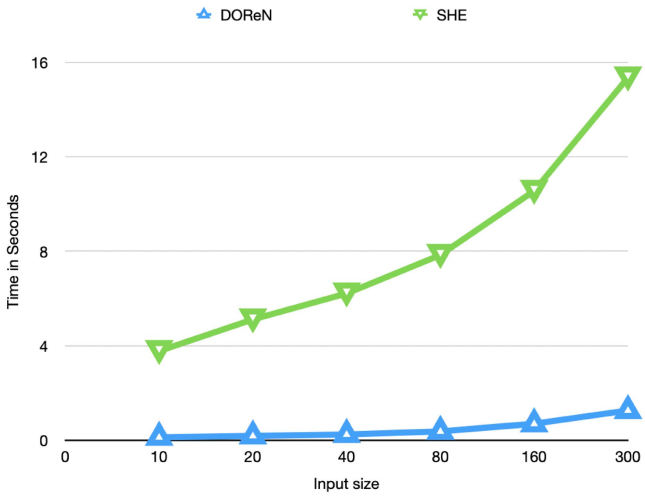

Fig. 6. Neuron Performance Comparison between DOReN and SHE for various input sizes.

used. Although there is no need for additional inter-layer transforms, due to the lack of batching, the performance of our architecture can still be expected to outperform SHE for most inter-layer transforms.

C. Network Performance

Network Architecture. In order to test the performance of our neuron design deployed on a full scale network, we consider five CIFAR10 architectures: VGG-7 [36], Resnet-20 [37], LoLa [18], GHE [38], and SHE [20], [39]. We adopt a 5-bit power-of-2 quantization for weights and biases, and

summarize our findings in Table III. The tailored optimization of our ground-up design is reflected in the low multiplicative depth required to run all networks, which demonstrates potential for highly efficient parallelized computing. The space efficiency of the approach is also shown in the Encrypted Input size which is 6× smaller than TFHE. This propagates throughout the network and renders the saved space rather considerable. The latency captured in our results is for single-threaded runs and does not fully draw from the benefits of these optimizations as they are designed for a massively parallel processing environment such as the cloud, which we are planning to test in our future works.

VI. Conclusion and Future Work

In this work, we lay the foundations for space-efficient, highly optimized privacy-preserving deep neural networks leveraging a plethora of techniques such as batching, bootstrapping, quantization, and depth-optimized compressor-based accumulators. With an improvement of between 2.4–8× in ciphertext expansion overhead compared to TFHE, it brings us one step closer to MLaaS for smaller client devices. Furthermore, the performance of our FHE-based neuron architecture outperforms SHE by an average of 20 folds.

Our experiments suggest that the accumulation of weighted inputs is the bottleneck for circuit-based evaluations of quantized neural networks and more work can be done to optimize accumulators. Our ground-up approach in building the circuits for this FHE-derived innovation creates room for big improvements. Further efforts are being deployed to explore the potential of this technology in building cloud-based, highly-parallelized, efficient, large-scale, complex ANNs.

Acknowledgment

The authors thank Lin Jie and Lu Yuxiao for sharing with them trained quantized variants of the five CIFAR-10 neural network architectures used in their experiments.

References

[1] O. Kocabas, T. Soyata, J. Couderc, M. Aktas, J. Xia, and M. Huang, "Assessment of cloud-based health monitoring using homomorphic encryption," in *Proc. IEEE 31st Int. Conf. Comput. Design (ICCD)*, Oct. 2013, pp. 443–446.

[2] A. Page, O. Kocabas, S. Ames, M. Venkitasubramaniam, and T. Soyata, "Cloud-based secure health monitoring: Optimizing fully-homomorphic encryption for streaming algorithms," in *Proc. IEEE Globecom Workshops (GC Wkshps)*, Dec. 2014, pp. 48–52.

[3] Q. Li, Y. Yue, and Z. Wang, "Deep robust Cramer Shoup delay optimized fully homomorphic for IIOT secured transmission in cloud computing," *Comput. Commun.*, vol. 161, pp. 10–18, Sep. 2020.

[4] M. Chraibi, S. Meftah, A. Maach, and H. Harroud, "Policy based context aware service level agreement (SLA) management in the cloud," in *Proc. CLOUD Comput. 8th Int. Conf. Cloud Comput., GRIDs, Virtualization*, 2017, pp. 122–128.

[5] M. T. I. Ziad, A. Alanwar, Y. Alkabani, M. W. El-Kharashi, and H. Bedour, "Homomorphic data isolation for hardware trojan protection," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, Jul. 2015, pp. 131–136.

[6] Y. Wang and A. Kogan, "Designing confidentiality-preserving blockchain-based transaction processing systems," *Int. J. Accounting Inf. Syst.*, vol. 30, pp. 1–18, Sep. 2018.

[7] M. Souhail et. al., "Network based intrusion detection using the UNSW-NB15 dataset," *Int. J. Comput. Digit. Syst.*, vol. 8, no. 5, pp. 477–487, Jan. 2019.

[8] 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. 33rd Int. Conf. Mach. Learn.*, vol. 48. Proceedings of Machine Learning Research, 2016, pp. 201–210. [Online]. Available: http://proceedings.mlr.press/v48/gilad-bachrach16.html

[9] F. Bourse, M. Minelli, M. Minihold, and P. Paillier, "Fast homomorphic evaluation of deep discretized neural networks," in *Advances in Cryptology—CRYPTO*. Cham, Switzerland: Springer, 2017, pp. 483–512.

[10] M. Courbariaux and Y. Bengio, "BinaryNet: Training deep neural networks with weights and activations constrained to +1 or −1," 2016, *arXiv:1602.02830*. [Online]. Available: https://arxiv.org/abs/1602.02830

[11] I. G. N. Chillotti *et al.*, "Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds," in *Advances in Cryptology—ASIACRYPT 2016* (Lecture Notes in Computer Science), vol. 10031. Berlin, Germany: Springer, 2016, pp. 3–33.

[12] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Salt Lake City, UT, USA, Jun. 2018, pp. 6848–6856.

[13] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," in *Proc. ITCS*, 2012, pp. 309–325.

[14] J. H. Cheon, M. Kim, and M. Kim, "Search-and-compute on encrypted data," in *Financial Cryptography and Data Security* (Lecture Notes in Computer Science), vol. 8976. Heidelberg, Germany: Springer, 2015, pp. 142–159.

[15] X. Liu, R. H. Deng, K.-K.-R. Choo, Y. Yang, and H. Pang, "Privacy-preserving outsourced calculation toolkit in the cloud," *IEEE Trans. Dependable Secure Comput.*, vol. 17, no. 5, pp. 898–911, Sep. 2020.

[16] N. P. Smart and F. Vercauteren, "Fully homomorphic SIMD operations," *Designs, Codes Cryptography*, vol. 71, no. 1, pp. 57–81, Apr. 2014.

[17] X. Jiang, M. Kim, K. Lauter, and Y. Song, "Secure outsourced matrix computation and application to neural networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2018, pp. 1209–1222.

[18] A. Brutzkus, R. Gilad-Bachrach, and O. Elisha, "Low latency privacy preserving inference," in *Proc. 36th Int. Conf. Mach. Learn. (ICML)* in Proceedings of Machine Learning Research, vol. 97, K. Chaudhuri and R. Salakhutdinov, Eds. Long Beach, CA, USA: PMLR. Jun. 2019, pp. 812–821. [Online]. Available: http://proceedings.mlr.press/v97/brutzkus19a.html

[19] J. Chao *et al.*, "CaRENets: Compact and resource-efficient CNN for homomorphic inference on encrypted medical images," 2019, *arXiv:1901.10074*. [Online]. Available: https://arxiv.org/abs/1901.10074

[20] Q. Lou and L. Jiang, "SHE: A fast and accurate deep neural network for encrypted data," in *Proc. Annu. Conf. Neural Inf. Process. Syst. NeurIPS*, Dec. 2019, Vancouver, BC, Canada, pp. 10035–10043.

[21] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless CNNs with low-precision weights," 2017, *arXiv:1702.03044*. [Online]. Available: https://arxiv.org/abs/1702.03044

[22] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE," in *Advances in Cryptology—ASIACRYPT* (Lecture Notes in Computer Science), vol. 10624, Heidelberg, Germany: Springer, 2017, pp. 377–408.

[23] A. QaisarAhmadAlBadawi *et al.*, "Towards the AlexNet moment for homomorphic encryption: HCNN, the first homomorphic CNN on encrypted data with GPUs," *IEEE Trans. Emerg. Topics Comput.*, early access, Aug. 6, 2021, doi: 10.1109/TETC.2020.3014636.

[24] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical GapSVP," in *Advances in Cryptology—CRYPTO* (Lecture Notes in Computer Science), vol. 7417. Berlin, Germany: Springer, 2012, pp. 868–886.

[25] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," Cryptology ePrint Archive, Tech. Rep. 2012/144, 2012.

[26] C. Gentry, S. Halevi, and N. P. Smart, "Fully homomorphic encryption with polylog overhead," in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science), vol. 7237. Berlin, Germany: Springer, 2012, pp. 465–482.

[27] S. Halevi and V. Shoup, "Algorithms in Helib," in *Proc. 34th Annu. Cryptol. Conf.*, Santa Barbara, CA, USA, Aug. 2014, pp. 554–571.

[28] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. 41st Annu. ACM Symp. Symp. Theory Comput. (STOC)*. New York, NY, USA: Association for Computing Machinery, 2009, pp. 169–178.
[29] H. Chen and K. Han, "Homomorphic lower digits removal and improved FHE bootstrapping," in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science), vol. 10820. Heidelberg, Germany: Springer, 2018, pp. 315–337.
[30] S. Halevi and V. Shoup, "Bootstrapping for Helib," in *Proc. 34th Annu. Int. Conf. Theory Appl. Cryptograph. Techn.*, Sofia, Bulgaria, Apr. 2015, pp. 641–670.
[31] A. Costache, K. Laine, and R. Player, "Evaluating the effectiveness of heuristic worst-case noise analysis in FHE," in *Computer Security—ESORICS*. Cham, Switzerland: Springer, 2020, pp. 546–565.
[32] B. H. M. Tan, H. T. Lee, H. Wang, S. Q. Ren, and A. M. M. Khin, "Efficient private comparison queries over encrypted databases using fully homomorphic encryption with finite fields," *IEEE Trans. Dependable Secure Comput.*, early access, Jan. 17, 2020, doi: 10.1109/TDSC.2020.2967740.
[33] M. R. Albrecht, R. Player, and S. Scott, "On the concrete hardness of learning with errors," *J. Math. Cryptol.*, vol. 9, no. 3, pp. 169–203, Jan. 2015.
[34] M. R. Albrecht, "On dual lattice attacks against small-secret LWE and parameter choices in HElib and SEAL," in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science), vol. 10211. Heidelberg, Germany: Springer, 2017, pp. 103–129.
[35] T. L. Authors. *Tfhe Library*. Accessed: Dec. 2020. [Online]. Available: https://github.com/tfhe/tfhe
[36] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015, *arXiv:1409.1556*. [Online]. Available: https://arxiv.org/abs/1409.1556
[37] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2015, pp. 770–778.
[38] 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*. New York, NY, USA: Association for Computing Machinery, Apr. 2019, pp. 3–13, doi: 10.1145/3310273.3323047.
[39] H. Chabanne, A. de Wargny, J. Milgram, C. Morel, and E. Prouff, "Privacy-preserving classification on deep neural network," Cryptology ePrint Archive, Tech. Rep. 2017/035, 2017. [Online]. Available: https://eprint.iacr.org/2017/035

Souhail Meftah received the B.Sc. (Hons.) degree in computer science and the M.Sc. degree in information systems security from Al Akhawayn University, Ifrane, Morocco, in 2016 and 2018, respectively. He is currently pursuing the Ph.D. degree in electrical and computer engineering with the National University of Singapore. He is also affiliated as a Research Scholar with the Institute of Infocomm Research, A*Star, Singapore. His research interests include deep learning, homomorphic encryption, and federated learning.

Benjamin Hong Meng Tan (Member, IEEE) received the B.Sc. and Ph.D. degrees in mathematical sciences from Nanyang Technological University, Singapore, in 2013 and 2019, respectively. He is currently a Researcher with the Institute of Infocomm Research, A*STAR, Singapore. His research interests include cryptography, multiparty computation, and secure cloud computing.

Chan Fook Mun received the B.Eng. (Hons.) degree in computer engineering from Nanyang Technological University, Singapore, in 2006. His main research interests include the areas of data and information security.

Khin Mi Mi Aung (Senior Member, IEEE) received the Ph.D. degree of computer engineering from Korea Aerospace University, Seoul, South Korea, in 2006. She is currently a Senior Scientist with the Institute of Infocomm Research, A*STAR, Singapore. Her current research interests include privacy-preserving and data security technologies.

Bharadwaj Veeravalli (Senior Member, IEEE) received the B.Sc. degree in physics from Madurai Kamaraj University, India, in 1987, the master's degree in electrical communication engineering from the Indian Institute of Science (IISc), Bengaluru, India, in 1991, and the Ph.D. degree from the Department of Aerospace Engineering, Indian Institute of Science, in 1994. He is currently with the Communications and Information Engineering (CIE) Division, Department of Electrical and Computer Engineering, National University of Singapore, Singapore, as a Tenured Associate Professor. He is also one of the earliest researchers in the field of divisible load theory (DLT). His main stream research interests include cloud/grid/cluster computing, such as big data processing, analytics, and resource allocation, scheduling in parallel and distributed systems, cybersecurity, and multimedia computing. He is a Senior Member of the IEEE CS. He received Gold Medals for his bachelor degree overall performance and for an outstanding Ph.D. thesis (IISc) in 1987 and 1994, respectively. He is serving on the Editorial Board for IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS as an Associate Editor.

Vijay Chandrasekhar (Senior Member, IEEE) received the B.S. and M.S. degrees from Carnegie Mellon University, in 2005 and the Ph.D. degree in electrical engineering from Stanford University, in 2013. He has published more than 100 articles/MPEG contributions in a wide range of top-tier journals/conferences. His research interests include deep learning and machine learning algorithms, computer vision, large-scale image and audio search, and augmented reality and deep learning hardware.