LOW LATENCY PRIVACY PRESERVING INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Using machine learning in domains such as medicine and finance requires tools that can preserve privacy and confidentiality. In this work, we focus on private inference with neural networks. Following the work of Dowlin et al. (2016), we use Homomorphic Encryption (HE) to allow neural networks to be applied to encrypted data and therefore make predictions while preserving privacy. We present $90\times$ improvement in latency and $7\times$ improvement in throughput compared to prior attempts. The improved performance is achieved via a modern implementation of the encryption scheme and a collection of methods to better represent the data during the computation. We also apply the method of transfer learning to provide private inference services using deep networks. We demonstrate the efficacy of our methods on several computer vision tasks.

1 INTRODUCTION

Machine learning is used today in domains such as education, health, business and more. In many of these domains, data may be private or confidential. Therefore, machine learning algorithms should preserve privacy as well as make accurate predictions. The privacy requirement pertains to all subtasks of the learning process, such as training and inference.

In this work we focus on private neural-networks inference. In this problem, popularized by the work on CryptoNets (Dowlin et al., 2016), the goal is to build an inference service that can make predictions on private data. To achieve this goal, the data is encrypted before it is sent to the prediction service that should be capable of operating on the encrypted data without having access to the raw content. To allow that, several cryptology technologies have been proposed, including Secure Multi-Party Computation (MPC) (Yao, 1982; Goldreich et al., 1987), hardware enclaves, such as Intel's Software Guard Extensions (SGX) (McKeen et al., 2013), Homomorphic Encryption (Gentry, 2009) and hybrid approaches that combine some of these techniques.

The different approaches present different trade-offs in terms of computation and security. HE presents the most stringent security model. The security assumption relies on the hardness of solving a mathematical problem, which in most approaches is the problem of Ring Learning With Errors (RLWE) or some variant of it (Gentry, 2009; Albrecht et al., 2018). There are no known algorithms for solving this problem efficiently, even in the presence of quantum computers. MPC protocols make a different set of assumptions, they typically assume that the parties involved do not collude in addition to assuming the security of some underlying encryption scheme such as AES (Yao, 1982; Goldreich et al., 1987). Many MPC protocols make additional assumptions about the honesty of the parties, namely, that they will not deviate from the prescribed protocol. Hardware based solutions, such as SGX (McKeen et al., 2013), require trusting the hardware manufacturer to have implemented a secure protocol that does not suffer from weaknesses. Some vulnerabilities of SGX have been discovered recently (Chen et al., 2018; Koruyeh et al., 2018).

In this work we use HE to provide privacy. We show that while providing high level of privacy, it can deliver results that are comparable or even superior to other privacy techniques. Specifically, we use the SEAL[1] library version 2.3.1 which implements the Brakerski/Fan-Vercauteren scheme (BFV) (Fan & Vercauteren, 2012) and select parameters to provide 128 bits of security (Albrecht et al., 2018).

[1] `http://sealcrypto.org/`

Homomorphic Encryption is limited in the type of operations it supports. The BFV scheme operates on messages that are elements of the ring $R = \frac{\mathbb{Z}_p[x]}{x^n+1}$. By using the Chinese Reminder Theorem (CRT) and by choosing the parameters n and p appropriately, we have that R can be thought of as a vector space of dimension n over $\mathbb{Z}_p = \frac{\mathbb{Z}}{p\mathbb{Z}}$ (Brakerski et al., 2014). Hence, we can think of every encrypted message as a vector of dimension n over the integers modulus p. When viewed in this way, the permissible operations over such vectors by the BFV scheme are element-wise additions, element-wise multiplications and rotations. The ideal rotation operation of size k sends the value in m_i to $m_{(i+k) \mod n}$. The BFV scheme allows a slight modified version of the ideal rotation (see Appendix A) but for the use we make of rotations in this work, there is no different between the two versions and therefore, for the rest of the paper, we assume the ideal rotation.

With these tools in hand, the goal is to build a private prediction service. The service should allow users to encrypt their private data and send it to the service that will compute the predictions and send them, still encrypted, to the user which can decrypt them by using the secret key. The security of such a service follows directly from the security of the BFV scheme and therefore, the main challenge is the complexity of this approach.

Following CryptoNets (Dowlin et al., 2016) we use MNIST (LeCun et al., 2010) as a first benchmark. On this data-set, we present 3 different solutions, all of them implement the same neural network that CryptoNets implements, and have the same accuracy of 98.95%. The first solution, dubbed here CryptoNets 2.3, follows the same methods used of Dowlin et al. (2016), but uses the modern SEAL 2.3.1 library and better memory and multi-threading. CryptoNets 2.3 achieves a throughput of almost 1.2M predictions per hour which is $7\times$ more than any other solution we are familiar with.[2] Next, we present two Low-Latency (LoLa) versions. The LoLa solution makes a single prediction in 7.2 seconds and has the benefit that a single image is encrypted using a single HE message (compared to 784 messages in CryptoNets). Finally, LoLa-conv has the lowest latency of any HE based private predictor with similar accuracy, it makes a prediction in 2.2 seconds ($> 90\times$ improvement over CryptoNets).

The low latency of LoLa and LoLa-conv are achieved by utilizing a flexible data representation during the computation. Unlike CryptoNets, that uses a single representation of the data throughout the computation, LoLa and LoLa-conv use diverse set of representations and move between them during the computation. We show that this technique can be extended to more challenging tasks such as making predictions on the Cifar data-set. The neural network used for the Cifar data-set has larger hidden layers. Since CryptoNets use a message (cipher-text) for every unit in the hidden layer, the amount of memory needed to implement this neural-net using the CryptoNet approach requires 100's of GBs of RAM while the LoLa approach uses only 10GB of RAM.

Finally, we push the idea of flexible data representation even further and use the technique of *transfer learning* to apply deep-networks to private data. We demonstrate that by making predictions on the Caltech data-set in 0.18 second with class balanced accuracy of 75.7%. Here, instead of encrypting an image as a pixel array, we use deep representations. A standard network, in this case AlexNet (Krizhevsky et al., 2012), is used to create a representation of the data which allows the inference to use linear classifiers.

2 Related Work

The task of private predictions gained significant attention in recent years. Dowlin et al. (2016) presented CryptoNets which demonstrated the feasibility of private neural networks predictions using Homomorphic Encryption. CryptoNets are capable of making predictions with high throughput but are limited in both the depth of the network they can support and the latency per prediction. Bourse et al. (2017) used a different HE scheme that allows fast bootstrapping which results in only a linear penalty for additional layers in the network. However, it is slower per operation and therefore, the results they presented on the MNIST data-set use small models with significantly lower accuracy (see Table 1). Hesamifard et al. (2017) achieved very high accuracy of 99.52% on MNIST and 91.5% on CIFAR-10, but with high latency of 320 seconds and 11686 respectivly.

[2]Experiments were conducted on Azure standard B8ms virtual machine with 8vCPUs and 32GB of RAM.

Other researchers proposed using different encryption schemes. For example, the Chameleon system (Riazi et al., 2018) uses MPC to demonstrate private predictions on MNIST in 2.24 seconds while achieving accuracy of 99%. Using MPC allows implementing ReLU activations which HE does not support. However, it comes with a price of significantly reduced security since it relies on semi-honest, non-collusion assumptions. Juvekar et al. (2018) uses a hybrid MPC-HE approach that can make a single prediction in as little as 0.03 seconds. Again, this comes with the price of the reduced security level provided by MPC.

Hardware based solutions were also proposed by, for example, Tramer & Boneh (2018). These methods enjoy the speed that the hardware provides. Tramer & Boneh (2018) go even further and show how some of the computation can be offloaded from the secure enclave to even faster hardware such as GPUs. However, it is important to recall that the security of these approaches is weaker and that there are known vulnerabilities to these hardware based systems (Chen et al., 2018; Koruyeh et al., 2018).

3 Data Representation

Feed-forward neural networks are functions that can be computed by an alternating sequence of linear transformations and non-linear transformations. Linear transformations include dense, convolution layers and average pooling layers. Non-linear transformations include activation functions and max pooling layers. In most cases, we can consider this sequence to be alternating between linear transformations and non-linear ones since consecutive linear transformations can be combined into a single linear transformation and sequences of non-linear transformations can be merged as well.

For most of this work, we restrict the non-linear transformations to the square activation function. This follows CryptoNets (Dowlin et al., 2016) that showed that high accuracy can be achieved even with this restriction. We demonstrate this again on the Cifar data-set in Section 5. Recall that HE supports point-wise multiplication of vectors and therefore it is straight-forward to implement the square activation function.

The main linear transformations we consider are dot-products and matrix-vector multiplications. Given two vectors, we can implement a dot product between two vectors whose size is a power of 2 by first applying point-wise multiplication between the two vectors and then a series of $\log n$ rotations of size $1, 2, 4, \ldots, n/2$ and addition between each rotation. The result of such a dot product operation is a vector that holds the results of the dot-product in all its coordinates.[3]

The dot-product operation can induce a change in representations. For example, given a weights matrix and an input vector represented as a single message, we can multiply the matrix by the vector using r dot-product operations where r is the number of rows in the matrix. The result of this operation is a vector of length r that is spread across r messages. Therefore, the result has a different representation than the representation of the input vector. Different representations can induce different computational costs and therefore choosing the right representations throughout the computation is important for computational efficiency. It is possible to change representations but this requires additional computational steps. Instead, we propose using various representations in the network inference. We start our discussion by presenting different possible vector representations.

3.1 Vector representations

Recall that a message in HE can be thought of as a vector of length n of elements in $\mathbb{Z}_p$. For the sake of brevity, we assume that the dimension of the vector $\mathbf{v}$ to be encoded is of length k such that $k \leq n$, for otherwise multiple messages can be combined. For any vector $\mathbf{u}$ we denote by u_i its ith coordinate.

3.1.1 Dense representation: A vector $\mathbf{v}$ is represented as a single message $\mathbf{m}$ by setting $v_i \mapsto m_i$.

[3]For example, consider calculating the dot product of 4-dimensional vectors $(v_1, ..., v_4)$ and $(w_1, ..., w_4)$. Point-wise multiplication, rotation of size 1 and summation results in the vector $(v_1w_1 + v_4w_4, v_2w_2 + v_1w_1, v_3w_3 + v_2w_2, v_4w_4 + v_3w_3)$. Another rotation of size 2 and summation results in the 4 dimensional vector which contains the dot-product of the vectors in all coordinates.

3.1.2 Sparse representation: A vector $\mathbf{v}$ of length k is represented in k messages $\mathbf{m}^1, \ldots \mathbf{m}^k$ such that $\mathbf{m}^i$ is a vector in which every coordinate is set to v_i.[4]

3.1.3 Stacked representation: For a short (low dimension) vector $\mathbf{v}$, the stacked representation holds several copies of the vector $\mathbf{v}$ in a single message $\mathbf{m}$. Typically this will be done by finding $d = \lceil \log(k) \rceil$, the smallest d such that the dimension of $\mathbf{v}$ is at most 2^d and setting $m_i, m_{i+2^d}, m_{i+2 \cdot 2^d}, \ldots = v_i$.

3.1.4 Interleaved representation: The interleaved representation uses a permutation σ of $[1, \ldots, n]$ to set $m_{\sigma(i)} = v_i$. The dense representation can be viewed as a special case of the interleaved representation where σ is the identity permutation.

3.1.5 Convolution representation: This is a special representation that makes convolution operations efficient. A convolution, when flattened to a single dimension, can be viewed as a restricted linear operation where there is a weight vector $\mathbf{w}$ of length r (the window size) and a set of permutations σ_i such that the i'th output of the linear transformation is $\sum_j w_j v_{\sigma_i(j)}$. The convolution representation takes a vector v and represents it as r messages $\mathbf{m}^1, \ldots, \mathbf{m}^r$ such that $m_i^j = v_{\sigma_i(j)}$. [5]

3.1.6 SIMD representation: CryptoNets (Dowlin et al., 2016) represent each data element as a separate message but maps multiple data vectors into the same set of messages. More details about this representation are in Appendix B.

3.2 Matrix-vector multiplications

Matrix-vector multiplication is a core operation in neural networks. The matrix may contain the learned weights of the network and the vector of the data at different stages of processing. Here we present different ways to implement such matrix-vector operations. Each method operates on vectors in different representations and produces output in yet another representation. Furthermore, the weight matrix has to be represented appropriately as a set of vectors, either column-major or row-major to allow the operation. We assume that the matrix W has k columns $\mathbf{c}^1, \ldots, \mathbf{c}^k$ and r rows $\mathbf{r}^1, \ldots, \mathbf{r}^r$.

3.2.1 Dense Vector – Row Major: If the vector is given as a dense vector and each row $\mathbf{r}^j$ of the weight matrix is encoded as a dense vector then the matrix-vector multiplication can be applied using r dot-product operations. As already described above, a dot-product requires a single multiplication and $\log(n)$ additions and rotations. The result is a sparse vector of length r.

3.2.2 Sparse Vector – Column Major: Recall that $W\mathbf{v} = \sum v_i \mathbf{c}^i$. Therefore, when $\mathbf{v}$ is encoded in a sparse format, the message $\mathbf{m}^i$ has all its coordinate set to v_i and $v_i \mathbf{c}^i$ can be computed using a single point-wise multiplication. Therefore, $W\mathbf{v}$ can be computed using k multiplications and additions and the result is a dense vector.

3.2.3 Stacked Vector – Row Major: For the sake of clarity, assume that $k = 2^d$ for some d. In this case n/k copies of $\mathbf{v}$ can be stacked in a single message $\mathbf{m}$ (this operation requires $\log(n/k) - 1$ rotations and additions). By concatenating n/k rows of W into a single message a special version of the dot-product operation can be used to compute n/k elements of $W\mathbf{v}$ at once. First, a point-wise multiplication of the stacked vector and the concatenated rows is applied followed by $d - 1$

[4] Recall that the encrypted messages are in the ring $R = \frac{\mathbb{Z}_p[x]}{x^n+1}$ which, by the choice of parameters, is homomrphic to $(\mathbb{Z}_p)^n$. When a vector has the same value v_i in all its coordinates, then its polynomial representation in $\frac{\mathbb{Z}_p[x]}{x^n+1}$ is the constant polynomial v_i.

[5] For example, consider a matrix $A \in \mathbb{R}^{4 \times 4}$ which corresponds to an input image and a 2×2 convolution filter that slides across the image with stride 2 in each direction. Let $a_{i,j}$ be the entry at row i and column j of the matrix A. Then, in this case $r = 4$ and the following messages are formed $M^1 = (a_{1,1}, a_{1,3}, a_{3,1}, a_{3,3})$, $M^2 = (a_{1,2}, a_{1,4}, a_{3,2}, a_{3,4})$, $M^3 = (a_{2,1}, a_{2,3}, a_{4,1}, a_{4,3})$ and $M^4 = (a_{2,2}, a_{2,4}, a_{4,2}, a_{4,4})$. In some cases it will be more convinient to combine the interleaved representation with the convolution representation by a permutation τ such that $m_{\tau(i)}^j = v_{\sigma_i(j)}$.

rotations and additions where the rotations are of size $1, 2, \ldots, 2^{d-1}$. The result is in the interleaved representation.[6]

The Stacked Vector - Row Major gets its efficiency from two places. First, the number of modified dot product operations is rk/n and each dot product operation requires a single multiplication and second, only d rotations and additions (compared to $\log n$ rotations and additions in the standard dot-product procedure).

3.2.4 Interleaved Vector – Row Major: This setting is very similar to the dense vector – row major matrix multiplication procedure with the only difference being that the columns of the matrix have to be shuffled to match the permutation of the interleaved representation of the vector. The result is in sparse format.

3.2.5 Convolution vector – Row Major: A convolution layer applies the same linear transformation to different locations on the data vector $\mathbf{v}$. For the sake of brevity, assume the transformation is one-dimensional. In neural network language that would mean that the kernel has a single map. Obviously, if more maps exist, then the process described here can be repeated multiple times.

Recall that a convolution, when flattened to a single dimension, is a restricted linear operation where the weight vector $\mathbf{w}$ is of length r, and there exists a set of permutations σ_i such that the i'th output of the linear transformation is $\sum w_j v_{\sigma_i(j)}$. In this case, the convolution representation is made of r messages such that the i'th element in the message $\mathbf{m}^j$ is $v_{\sigma_i(j)}$. By using a sparse representation of the vector $\mathbf{w}$, we get that $\sum w_j \mathbf{m}^j$ computes the set of required outputs using r multiplications and additions. When the weights are not encrypted, the multiplications used here are relatively cheap since the weights are scalar and BFV supports fast implementation of multiplying a message by a scalar. The result of this operation is in a dense format.

4 Secure Networks for MNIST

The neural network used for the MNIST data-set (LeCun et al., 2010) is the same network used by CryptoNets (Dowlin et al., 2016). After suppressing adjacent linear layers it can be presented as a 5×5 convolution layer with a stride of $(2, 2)$ and 5 output maps, which is followed by a square activation function that feeds a fully connected layer with 100 output maps, another square activation and another fully connected layer with 10 outputs (see Figure 2 in the appendix).

The baseline implementation uses the techniques presented in CryptoNets in Table 1. Recall that CryptoNets use the SIMD representation (Section 3.1.6) in which each pixel requires its own message. Therefore, since each image in the MNIST data-set is made of an array of 28×28 pixels, the input to the CryptoNets network is made of 784 messages. On the reference machine used for this work (Azure standard B8ms virtual machine with 8 vCPUs and 32GB of ram) the original CryptoNets implementation runs in 205 seconds. Re-implementing it to use better memory management and multi-threading in SEAL 2.3 reduces the running time to 24.8 seconds. Since this implementation allows batching of 8192 images to be processed simultaneously, it has a potential throughput of 1189161 predictions per hour which is, as far as we know, the highest throughput reported on this task by a large margin.

While CryptoNets provide high throughput, in many cases, it is hard to utilize this high throughput which requires batching together 8192 requests from sources that share the same secret key. If each user has only a single record to be predicted on, the throughput is governed by the latency and therefore, we move towards reducing latency. We do that by replacing the SIMD representation with other representations. As a result, throughput is sacrificed in favor of latency.

The Low-Latency CryptoNets (LoLa) uses the same network layout and has accuracy of 98.95%. However, it is implemented differently: the input to the network is a single dense message where the pixel values are mapped to coordinates in the encoded vector line after line. The first step

[6]For example, consider a 2×2 matrix W flattened to a vector $\mathbf{w} = (w_{1,1}, w_{1,2}, w_{2,1}, w_{2,2})$ and a two-dimensional vector $\mathbf{v} = (v_1, v_2)$. Then, after stacking the vectors, point-wise multiplication, rotation of size 1 and summation, the second entry of the result contains $w_{1,1}v_1 + w_{1,2}v_2$ and the fourth entry contains $w_{2,1}v_1 + w_{2,2}v_2$. Hence, the result is in an interleaved representation.

Method	Accuracy	Security Model	Latency	Throughput	
FHE–DiNN100	96.35%	TGSW	1.65	2182	(Bourse et al., 2017)
LoLa-Small	96.92%	RLWE	**0.29**	12500	
CryptoNets	98.95%	RLWE	205	71930	(Dowlin et al., 2016)
Gazelle	98.95%	Semi-honest	0.03	120000	(Juvekar et al., 2018)
CryptoNets 2.3	98.95%	RLWE	24.8	**1189160**	
LoLa	98.95%	RLWE	7.2	500	
LoLa-Conv	98.95%	RLWE	**2.2**	1636	
Chameleon	99%	Semi-honest	2.24	1607	(Riazi et al., 2018)
CryptoDL	99.52%	RLWE	**320**	164000	(Hesamifard et al., 2017)

Table 1: MNIST performance comparison. Solutions are grouped by accuracy levels.

in processing this message is breaking it into 25 messages corresponding to the 25 pixels in the convolution map to generate a convolution representation. Creating each message requires a single vector multiplication. This is performed by creating 25 masks. The first mask is a vector of zeros and ones that corresponds to a matrix of size 28×28 such that a one is in the (i, j) coordinate if the i, j pixel in the image appears as the upper left corner of the 5×5 window of the convolution layer. Multiplying point-wise the input vector by the mask creates the first message in the convolution representation as described in Section 3.1.5 hybrided with the interleaved representation as described in footnote 5. Similarly the other messages in the convolution representation are created. Note that all masks are shifts of each other which allows using the convolution representation-row major multiplication to implement the convolution layer (see Section 3.2.5). To do that, think of the 25 messages as a matrix and the weights of a map of the convolution layer as a sparse vector. Therefore, the outputs of the entire map can be computed using 25 multiplications (of each weight by the corresponding vector) and 24 additions. Note that there are 169 windows and all of them are computed simultaneously. However, the process repeats 5 times for the 5 maps of the convolution layer.

The result of the convolution layer are 5 messages, each one of them contains 169 results. They are united into a single vector by rotating the messages such that they will not have active values in the same locations and summing the results. At this point, a single message holds all the 845 values (169 windows $\times 5$ maps). This vector is squared, using a single multiplication operation, to implement the activation function that follows the convolution layer. This demonstrates one of the main differences between CryptoNets and LoLa; In CryptoNets, the activation layer requires 845 multiplication operations, whereas in LoLa it is a single multiplication. Even if we add the manipulation of the vector to place all values in a single message, as described above, we add only 4 rotations and 4 additions which are still much fewer operations than in CryptoNets.

Next, we apply a dense layer with 100 maps. LoLa uses messages of size $n = 16384$ where the 845 results of the previous layer, even though they are in interleaving representation, take fewer than 1024 dimensions. Therefore, 16 copies are stacked together which allows the use of the Stacked vector – Row Major multiplication method. This allows computing 16 out of the 100 maps in each operation and therefore, the entire dense layer is computed in 7 iterations resulting in 7 interleaved messages. By shifting the ith message by $i - 1$ positions, the active outputs in each of the messages are no longer in the same position and they are added together to form a single interleaved message that contains the 100 outputs. The following square activation requires a single point-wise-multiplication of this message. The final dense layer is applied using the Interleaved vector – Row Major method to generate 10 messages, each of which contains one of the 10 outputs.[7]

Overall, applying the entire network takes only 7.2 seconds on the same reference hardware which is $28.5\times$ faster than CryptoNets and $3.4\times$ faster than CryptoNets 2.3. This result can be further

[7]It is possible, if required, to combine them into a single message in order to save communication.

improved by changing the input to the network; Instead of taking as an input a dense representation of the image, the LoLa-Conv network takes as its input 25 messages which are the convolution representation of the image. This saves a processing step which saves time but also reduces the amount of noise accumulated during the computation and allows working with messages of size $n = 8192$, which further reduces the computation time.

The LoLa-Conv starts with a convolution vector – row major multiplication for each of the 5 maps of the convolution layer. The 5 dense output messages are joined together with a rotation and addition to form a single dense vector of 845 elements. This vector is squared using a single multiplication and 8 copies of the results are stacked before applying the dense layer as 13 rounds of Stacked vector – Row Major multiplication. The 13 vectors of interleaved results are rotated and added to form a single interleaved vector of results which is squared using a single multiplication. Finally, Interleaved vector – Row Major multiplication is used to obtain the final result. This version computes the entire network in only 2.2 seconds which is $3.3\times$ faster than LoLa, $11\times$ faster than CryptoNets 2.3 and $90\times$ faster than CryptoNets.

Table 1 shows a summary of the performance of different methods and more details can be found in Appendix C. Bourse et al. (2017) showed faster results with similar security level, albiet with lower accuracy. To compare with that, LoLa-Small is similiar to Lola-Conv but has only a convolution layer, square activation and a dense layer. This solution is more accuracte than the networks used by Bourse et al. (2017) and at the same time it is $5.5\times$ faster.

5 Secure Networks for Cifar

The Cifar-10 data-set (Krizhevsky & Hinton, 2009) presents a more challenging task of recognizing one of 10 different types of objects in a small image. The neural network used has the following layout: the input is a $3 \times 32 \times 32$ image (i) 3×3 linear convolution with stride of $(1, 1)$ and 128 output maps, (ii) 2×2 average pooling with $(2, 2)$ stride (iii) 3×3 convolution with $(1, 1)$ stride and 83 maps (iv) Square activation (v) 2×2 average pooling with $(2, 2)$ stride (vi) 3×3 convolution with $(1, 1)$ stride and 163 maps (vii) Square activation (vii) 2×2 average pooling with stride $(2, 2)$ (viii) fully connected layer with 1024 outputs (ix) fully connected layer with 10 outputs (x) softmax. ADAM was used for optimization (Kingma & Ba, 2014) together with dropouts after layers (vii) and (viii). We use zero-padding in layers (i) and (vii). See Figure 3 for an illustration of the network.

For inference, adjacent linear layers were collapsed to form the following structure: (i) $8 \times 8 \times 3$ convolutions with a stride of $(2, 2, 0)$ and 83 maps (ii) square activation (iii) $6 \times 6 \times 83$ convolution with stride $(2, 2, 0)$ and 163 maps (iv) square activation (v) dense layer with 10 output maps. This network is much larger than the network used for MNIST by CryptoNets. The input to the Cifar network has 3072 nodes, the first hidden layer has 16268 nodes and the second hidden layer has 4075 nodes (compared to 784, 845, and 100 nodes respectively for MNIST).[8] The accuracy of this network is 74.1% and it uses plain-text modulus $p = 2148728833 \times 2148794369 \times 2149810177$ (the factors are combined using the Chinese Reminder Theorem) and $n = 16384$. See Figure 4 for an illustration of this network.

Due to the sizes of the hidden layers, implementing this network with SIMD representation requires more memory than available on the reference machine, since the SIMD representation requires a message for each node in each layer. Therefore, we used the LoLa-Conv approach to implement this network. The image is encoded using the convolution representation into $3 \times 8 \times 8 = 192$ messages. The convolution layer is implemented using the convolution vector – row major matrix-vector multiplication technique. The results are combined into a single message using rotations and additions which allows the square activation to be performed with a single point-wise multiplication. The second convolution layer is performed using row major-dense vector multiplication. Although this layer is a convolution layer, each window of the convolution is so large that it is more efficient to implement it as a dense layer. The output is a sparse vector which is converted into a dense vector by point-wise multiplications and additions which allows the second square activation to be performed with a single point-wise multiplication. The last dense layer is implemented with a row major-dense vector technique again resulting in a sparse output.

[8]The map counts of the different layers were selected such that the sizes of the hidden layers will fit inside the encrypted messages.

Executing this network takes 730 seconds out of which the second layer consumes 711 seconds. Therefore, for this task the bottleneck in performance is the sizes of the weight matrices and data vectors as evident by the number of parameters which is $< 90,000$ in the MNIST network and $> 500,000$ in the Cifar network. In the following section we present an approach to mitigate this problem.

6 Applying Deep Nets using Deep Representations

Homomorphic Encryption has two main limitations when used for evaluating deep networks: noise growth and message size growth. Noise growth is a result of the number of operations that has to take place. Each such operation increases the noise in the encrypted message and when this noise becomes too large, it is no longer possible to decrypt the message correctly. This problem can be mitigated using bootstrapping, while taking a performance hit. The message size grows with the size of the network as well. Since, in its core, the HE scheme operates in $\mathbb{Z}_p$, the parameter p has to be selected such that the largest number obtained during computation would be smaller than p. Since every multiplication might double the required size of p, it has to grow exponentially with respect to the number of layers in the network. The recently introduced HEAAN scheme (Cheon et al., 2017) is more tolerant towards message growth but even HEAAN would not be able to operate efficiently on very deep networks.

We propose solving both the message growth and the noise growth problems using deep representations. Instead of encrypting the data in its raw format, it is first converted, by a standard network, to create a deep representation. For example, if the data is an image, then instead of encrypting the image as an array of pixels, a networks, such as AlexNet (Krizhevsky et al., 2012), VGG (Simonyan & Zisserman, 2014), or ResNet (He et al., 2016), first extracts a deep representation of the image, using one of its last layers. The resulting representation is encrypted and sent for evaluation. This approach has several advantages. First, this representation is small even if the original image is large. Moreover, with deep representations it is possible to obtain high accuracies using shallow networks: in most cases a linear predictor is sufficient which translates to a fast evaluation with HE. It is also a very natural thing to do since in many cases of interest, such as in medical imagine, training a very deep network from scratch is almost impossible since the data is insufficient. Hence, it is a common practice to use deep representation and train only the top layer(s) (Yosinski et al., 2014). Nevertheless, offloading some of the computation to be done before encryption may be prohibitive in some case.

To test the deep representation approach we used AlexNet (Krizhevsky et al., 2012) to generate features and trained a linear model to make predictions on the Caltech-101 data-set (Fei-Fei et al., 2006).[9] Since the Caltech 101 data-set is not class balanced, we used only the first 30 images from each class where the first 20 where used for training and the other 10 examples where used for testing. The obtained model demonstrated class-balanced accuracy of 75.7. The inference time, on the encrypted data, takes only 0.178 seconds when using the dense vector – row major multiplication.

7 Conclusions

The problem of privacy in machine learning is gaining importance due to legal requirements and greater awareness to the benefits and risks of machine learning systems. The task of private inference, specifically with neural networks, serves as a benchmark and catalyst to promote further study in this domain. In this work, we showed how data representations can be used to accelerate private predictions using Homomorphic Encryption. We demonstrated both high throughput and low latency (not necessarily simultaneously) which exceed previously known results for the same level of accuracy and security by orders of magnitude. While the methods presented here show great promise, they do have limitations that should be addressed in further studies: these methods protect the privacy of the data, but they do leak some information about the network that is used for predictions, specifically about its layout. The risk associated with that (if any), as well as ways to mitigate this risk, requires further study.

[9] More complex classifiers did not improve accuracy.

REFERENCES

Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Jeffrey Hoffstein, Kristin Lauter, Satya Lokam, Daniele Micciancio, et al. Homomorphic encryption standard. 2018.

Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. Fast homomorphic evaluation of deep discretized neural networks. Technical report, Cryptology ePrint Archive, Report 2017/1114, 2017.

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

Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten H Lai. Sgxpectre attacks: Leaking enclave secrets via speculative execution. *arXiv preprint arXiv:1802.09085*, 2018.

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

Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*, pp. 201–210, 2016.

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

Li Fei-Fei, Rob Fergus, and Pietro Perona. One-shot learning of object categories. *IEEE transactions on pattern analysis and machine intelligence*, 28(4):594–611, 2006.

Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, volume 9, pp. 169–178, 2009.

Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pp. 218–229. ACM, 1987.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. Cryptodl: Deep neural networks over encrypted data. *arXiv preprint arXiv:1711.05189*, 2017.

Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. Gazelle: A low latency framework for secure neural network inference. *arXiv preprint arXiv:1801.05507*, 2018.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Esmaeil Mohammadian Koruyeh, Khaled Khasawneh, Chengyu Song, and Nael Abu-Ghazaleh. Spectre returns! speculation attacks using the return stack buffer. In *12th USENIX Workshop on Offensive Technologies (WOOT 18)*. USENIX Association, 2018.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Yann LeCun, Corinna Cortes, and Christopher JC Burges. Mnist handwritten digit database. at&t labs, 2010.

Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R Savagaonkar. Innovative instructions and software model for isolated execution. *HASP@ ISCA*, 10, 2013.

M Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M Songhori, Thomas Schneider, and Farinaz Koushanfar. Chameleon: A hybrid secure computation framework for machine learning applications. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pp. 707–721. ACM, 2018.

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

Florian Tramer and Dan Boneh. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. *arXiv preprint arXiv:1806.03287*, 2018.

Andrew C Yao. Protocols for secure computations. In *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*, pp. 160–164. IEEE, 1982.

Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pp. 3320–3328, 2014.

A ROTATIONS

For the rotation operation in the BFV encryption scheme it is easier to think of the message as a $2 \times {}^{n}/_{2}$ matrix:

$$\begin{bmatrix} m_1 & m_2 & \cdot & \cdot & m_{n/2} \\ m_{n/2+1} & m_{n/2+2} & \cdot & \cdot & m_n \end{bmatrix}$$

with this representation in mind, there are two rotations allowed, one switches the row, which will turn the above matrix to

$$\begin{bmatrix} m_{n/2+1} & m_{n/2+2} & \cdot & \cdot & m_n \\ m_1 & m_2 & \cdot & \cdot & m_{n/2} \end{bmatrix}$$

and the other rotates the columns. For example, rotating the original matrix by one column to the right will result in

$$\begin{bmatrix} m_{n/2} & m_1 & \cdot & \cdot & m_{n/2-1} \\ m_n & m_{n/2+1} & \cdot & \cdot & m_{n-1} \end{bmatrix} .$$

Since n is a power of two, and the rotations we are interested in are powers of two as well, for the sake of this work, thinking about the rotations as simple rotations of the elements in the message yields similar results. In this view, the row-rotation is a rotation of size ${}^{n}/_{2}$ and smaller rotations are achieved by column rotations.

B THE SIMD REPRESENTATION

The vector structure of messages used by CryptoNets allow parallel execution over multiple data simultaneously. CryptoNets takes n input vectors $\mathbf{v}^1, \ldots, \mathbf{v}^n$ and creates a dense representation in which these n messages of length k are encoded in k messages $\mathbf{m}^1, \ldots, \mathbf{m}^k$ such that $m_i^j = v_j^i$. All operations between vectors and matrices are implemented using additions and multiplications only. For example, a dot product between two vectors of length k is implemented by k multiplications and additions. Therefore, it acts as a sparse representation.

The advantage of this representation, which we call the *SIMD Representation,* is that the cost of applying an operation to a vector is the same cost of applying the same operation to n vectors, hence it supports the Single Instruction Multiple Data (SIMD) framework. However, it is costly in two ways: the computational complexity of multiplying a matrix of size $r \times k$ with a vector of length k is $O(rk)$ HE operations, and the memory consumption is large as well since a vector of length k requires k messages. In this sense it is similar to the sparse representation. However, the ability to perform SIMD operations provides it with high throughput, much like the dense representation.

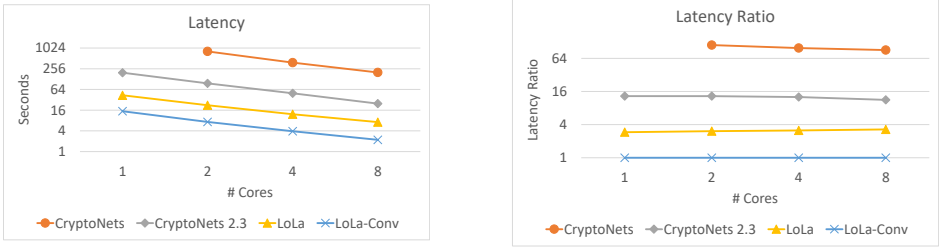

Figure 1: The latency of the different network implementations for the MNIST task with respect to the number of available cores. The right figure shows the ratio between the latency of each solution and the latency of the LoLa-Conv

C PARALLEL SCALING

The performance of the different solutions is affected by the amount of parallelism allowed. The hardware used for experimentation in this work has 8 cores. Therefore, we tested the performance of the different solutions with 1, 2, 4, and 8 cores to see how the performance varies. The results of these experiments are presented in Figure 1. These results show that at least up to 8 cores the performance of all methods scales linearly when tested on the MNIST data-set. This suggests that the latency can be further improved by using machines with higher core count.

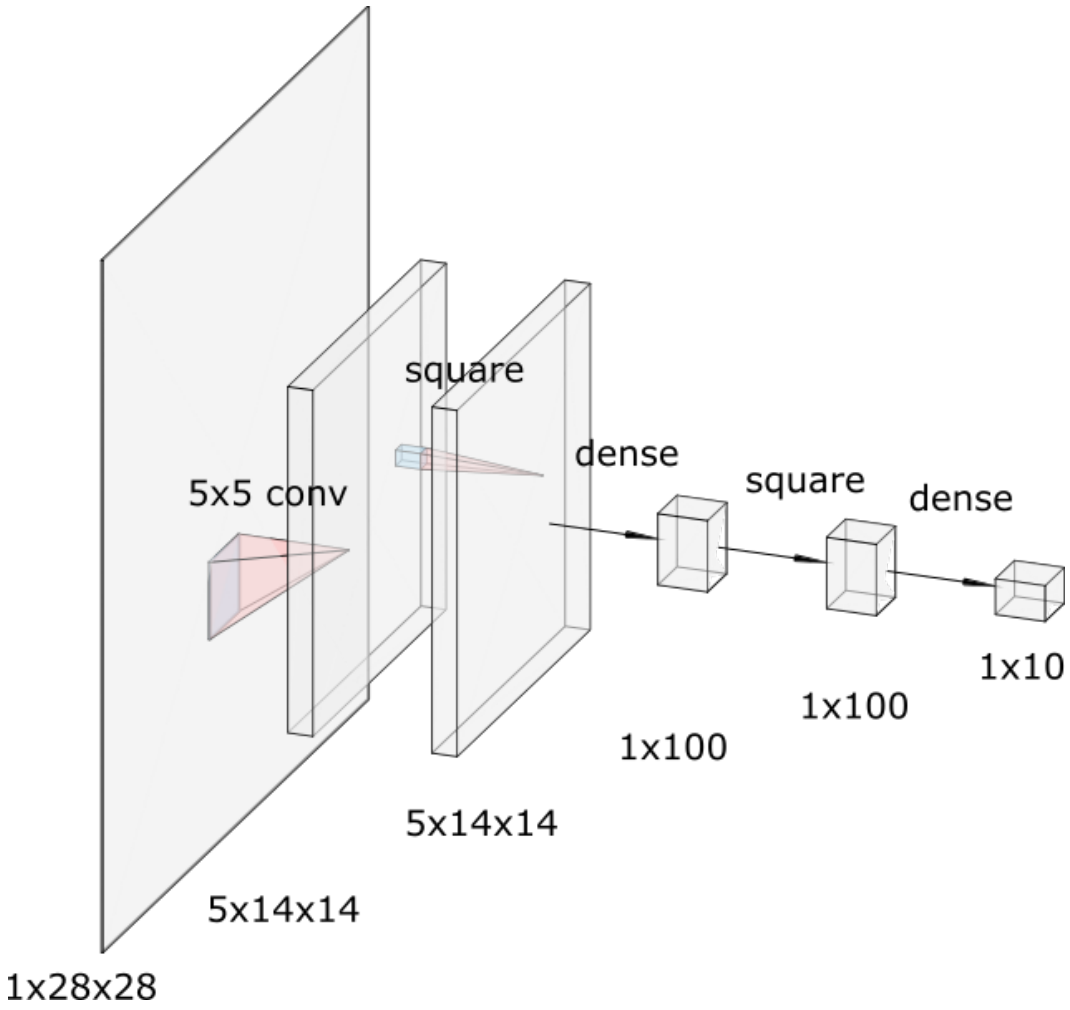

Figure 2: The structure of the network used for MNIST classification

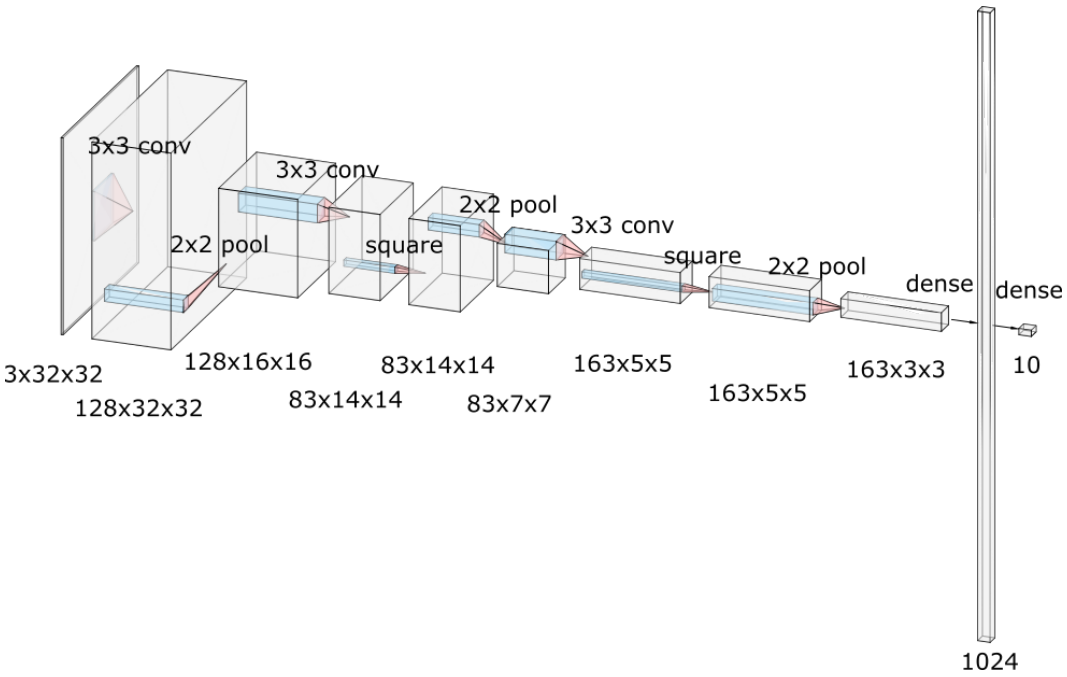

Figure 3: The structure of the network used for CIFAR classification.

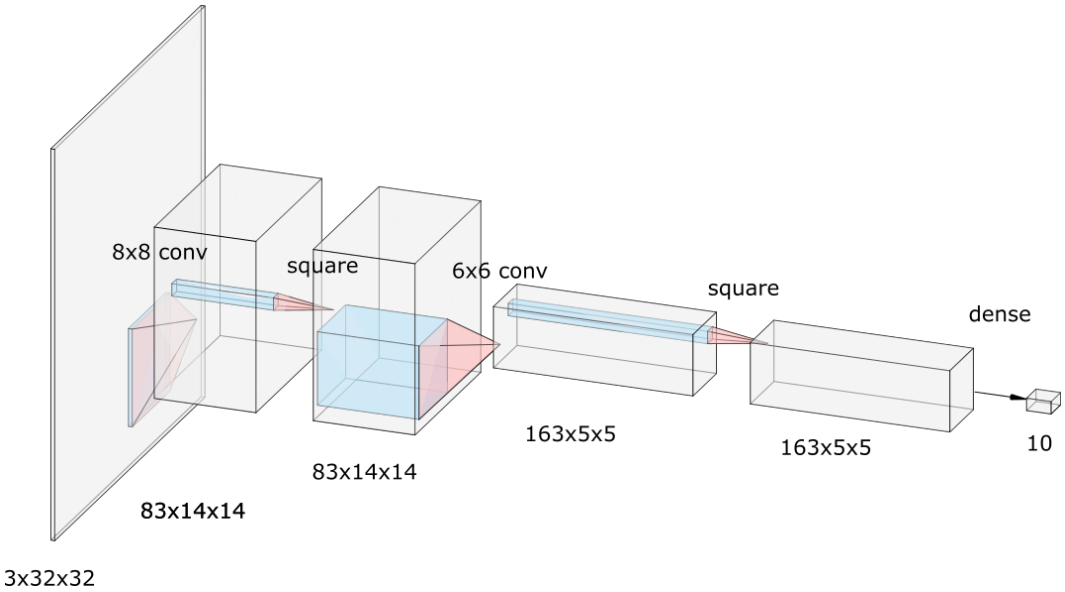

Figure 4: The structure of the network used for CIFAR classification after collapsing adjacent layers.